High-Level Synthesis for Model-Based Design with Automatic Folding including Combined Common Subcircuits

Patrick Sittel
Universität Kassel
sittel@uni-kassel.de

Martin Kumm
Universität Kassel
kumm@uni-kassel.de

Konrad Möller
Universität Kassel
konrad.moeller@uni-kassel.de

Martin Hardieck
Universität Kassel
hardieck@uni-kassel.de

Peter Zipf
Universität Kassel
zipf@uni-kassel.de

Abstract. We present an automatic tool-flow for combined folding with common subcircuits that is embedded into the open-source high-level synthesis (HLS) tool Origami HLS. Origami HLS is able to parse designs from Matlab/Simulink to apply the folding transformation and to generate optimized hardware implementations for different target architectures that meet specified design constraints. Current commercial tools, like the Matlab HDL Coder, often struggle to provide efficient implementations under tight area/throughput constraints. As a result, it is very challenging to find good implementations of a design. We address this problem and provide a number of solutions for transforming the circuit to allow a system designer to develop correct hardware implementations meeting his or her constraints. This is done by using common subcircuits and combinations of different kernels within the folding transformation for digital circuits. For this task, we adopt and combine subgraph identification, maximum independent set (MIS) and different selection algorithms from literature. We show that the Origami HLS folding transformation flow is able to reduce the FPGA resources used by up to 69% compared to folded circuits provided by the Matlab HDL Coder. The experiments also show that relevant area/throughput tradeoffs can be provided to the user automatically. At last, a design-space exploration for a practical example model is provided to support our claims.

1. Introduction

The complexity of digital circuits is growing at an enormous rate and the required time to market for electronic products is a permanent pressure [7]. Therefore, advanced design methods have to be developed in order to further reduce time to market [20]. HLS tools that generate hardware implementations by transforming an unscheduled system into a hardware model have proven to be very effective for this task [8]. Within this context, designers typically want their hardware to use a mimimum amount of silicon area while meeting tight latency, throughput and power contraints [19, 22]. The steps that are traditionally used to achieve these goals are scheduling, allocation and binding. Since the occupied area is critical for the overall costs, the reduction of silicon area by resource sharing has always been a central research topic [11].

Due to their flexibility and computational power, FPGAs are becoming more and more popular in several application domains. Therefore, it is a critical task for HLS tools to provide cross plattform implementations that can be used on various targets like SoCs, FPGAs, DSPs and ASICs of different technology generations [21]. For this task we adopt the concept of model-based design (MBD), since it is an industry standard in many domains like automotive, image/video or digital signal processing. Matlab/Simulink is a powerful and often-used MBD tool to describe embedded processor code or digital signal processing circuits like FIR and IIR filters, FFT and controllers [9].

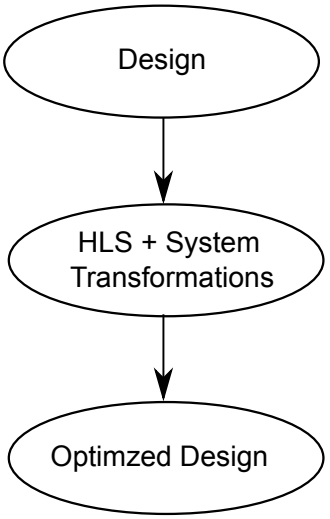


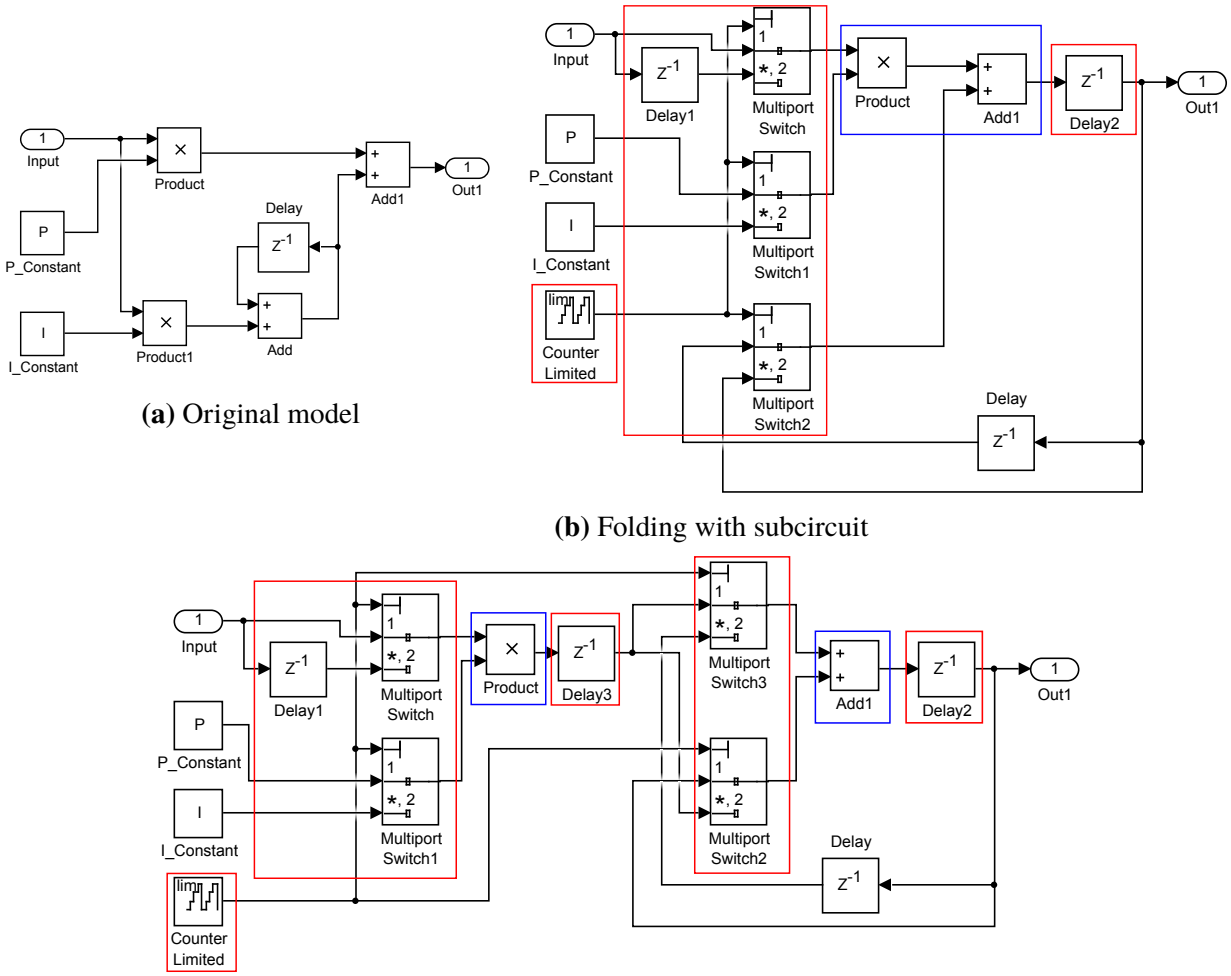
Figure 1: Model-based HLS

Figure 1 shows the flow of Origami HLS [15], the open source tool developed by our group. The general concept of Origami HLS is to use abstract design models, like Matlab/Simulink models, and apply transformations within this description domain. This results in refined and optimized models, thus supporting as many target devices as possible, as the binding and register-transfer level generation is done later. With Origami HLS being a *push-button flow*, a system designer is able to generate a constraint fit and optimized hardware model. In that way, Origami HLS bridges the gap between software-oriented system design and the need to implement very efficient hardware. Additionally, it is also possible to continue with the optimized model for other optimizations or verification. The goals of this work are to introduce this concept and to investigate the benefits of the folding transformation using combined common subcircuits for a design that is parsed from Matlab/Simulink. For this task, we propose graph-based algorithms to identify subgraphs for folding in Section 2. Finally, efficient FPGA implementations are generated as VHDL code and analyzed.

2. Background

It is the main goal of HLS to enable a very fast and automated generation of efficient hardware implementations by exploiting the benefits of higher abstraction levels [7]. For this task, a large number of tools have been presented. LegUp [3] is an open-source HLS tool that accepts C programs as input and produces a hardware description for a FPGA-based MIPS soft processor. Zhang et al. [24] took a similiar approach with their CMOST compilation flow. Cong and Jiang [6] presented with xPilot a tool that uses pattern-based behavior synthesis to optimize FPGA implementations. Other HLS research efforts for automating system generation are CoRAM [5] and LEAP [1]. Huthmann et al. [13] proposed the Nymble compiler system that accepts C code and generates hardware implementations for FPGA-based computers. While these tools accept C as input to provide FPGA implementations, Origami HLS uses model-based descriptions as input.

In signal processing applications where the sample rate is significantly lower than the clock rate, it is possible to reduce the number of functional units needed in the target implementation by time-multiplexing them. In general, the corresponding circuit transformations are done during the



(a) Original model

(b) Folding with subcircuit

(c) Folding with single operations

Figure 2: Simulink description of a PI controller

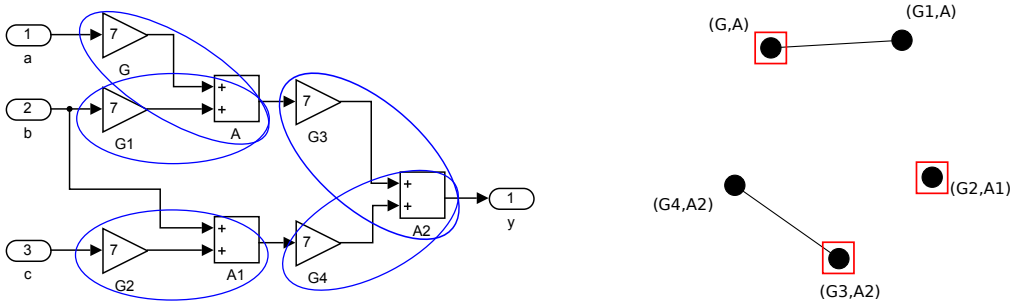
scheduling and binding process. One systematic method to reduce the required functional units in DSP applications by serializing the input design is called *folding* [17]. Traditionally, the folding transformation maps identical operations onto one functional unit in the target design. The selected operations are called *folding core*. The functional unit is implemented and shared by using registers and multiplexers, which is an overhead produced by the folding transformation. This overhead has to be smaller than the resources saved to achieve any resource reduction. The extension of this concept by using common subcircuits for folding was presented recently [16]. Two subcircuits are called *common* if and only if they are isomorphic to each other. These subcircuits are assumed to be connected. In the following, we will use common subcircuits as folding core.

To give an example, Figure 2a shows the Matlab/Simulink description of a time-discrete PI controller. Figure 2b and 2c show two different ways of using the folding transformation. While Figure 2c shows the classical approach where the single operators $\{Add\}$ and $\{Product\}$ were used as folding cores, the circuit in Figure 2b results from using the common subcircuit $\{Product, Add\}$. The folding cores are highlighted in blue and the overhead is highlighted in red.

In this example, it is beneficial to chose the common subcircuit for folding, because one multi-

plexer and one register can be saved. This results from the non-optimal binding of the adder. It is profitable to use a common subcircuit in this case, because the optimal binding is induced by the subcircuit. Recent results show that the use of carefully selected subcircuits as folding cores is very beneficial in widely used DSP applications [16]. In these experiments, the selection of folding cores was done by hand. The main contribution of this work is an automated flow that is able to identify all common subcircuits as folding core candidates. A first approach to combine them is presented that provides alternatives in the design space and finds solutions with resource reduction close to the hand-selected folding cores.

From a graph-theoretic point of view, valid folding cores are isomorphic subgraphs of the data-flow graph (DFG). Their automatic identification has been an active research topic for more than a decade [23, 4]. Algorithms like *HSiGraM* and *VSiGraM* [14] that search for all frequent subgraphs in a single graph have been used in a large number of other domains and this work introduces their application in the context of automatic common subcircuit folding. Unfortunately, the problem of finding all common subgraphs is known to be NP-hard [6] and heuristic approaches have to be used in the future. In the following, we will call the embeddings of a subgraph g into the input graph G the *occurrence*s of g in G. Elements of the power set of the occurrences are called *occurrence set*. Different combinations of occurrences of the same subgraph form different occurrence sets. Thus, the embeddings of one subgraph induce a large number of occurrence sets.



(a) Occurrences of a subgraph **(b)** Conflict graph of occurrences

Figure 3: Example: Overlapping of occurrences

For folding, it is only viable to use a selection of occurrences in an occurrence set that do not overlap and share operations [16]. Occurrences are called *compatible*, if they do not share any operators and the compatibility of occurrence sets is defined in the same way. Therefore, some occurrences might be incompatible and a selection of occurrences from every occurrence set has to be made. This topic was discussed in [2] and [10] and it could be shown that a frequency definition based on an MIS is suitable. To be more precise, the frequency of an occurrence set is defined as the size of the MIS of the corresponding conflict graph. It follows that the frequency of an occurrence set that is used for folding can be different from the total number of occurrences in the occurrence set. Any independent set of occurences with at least two elements is a valid folding core selection. As the number of independent sets can become very large, we decided to select one MIS to maximize the number of occurences. Note that this MIS selection is not always unique. Therefore, this heuristic selection of occurrences for folding means pruning other valid combinations to apply the folding transformation. Its performance is evaluated in this paper and comparisons to more advanced methods are planed within future work.

Figure 3a shows an example where all five occurrences of the subgraph $g = \{Gain, Adder\}$ are highlighted. Figure 3b shows a conflict graph that describes these overlaps. In this example, it is possible to find four different compatible occurrence sets g_i, containing only compatible occurrences, of g with the maximum frequency of three: $g_1 = \{(G, A), (G3, A2), (G2, A1)\}$, $g_2 = \{(G, A), (G4, A2), (G2, A1)\}$, $g_3 = \{(G1, A), (G3, A2), (G2, A1)\}$ and $g_4 = \{(G1, A), (G4, A2), (G2, A1)\}$. It is also possible to find these four maximum independet sets with three vertices in the graph of Figure 3b. As an example, the selection g_1 is highlighted by the red squares.

While the benefits of using isomorphic subcircuits for resource sharing were presented by Cong and Jiang [6], the topic of selecting the best isomorphic subcircuits for folding is still an open question. The expected benefits are very promising, because the reduction of size and number of multiplexers is very beneficial as they are very costly on FPGAs or SoCs. In the following, we will show the advantages of our tool flow for automatic subgraph enumeration, combination, folding and HDL circuit generation that is realized in Origami HLS. Based on this, a design-space exploration for applying the folding transformation with isomorphic subcircuits is presented.

3. Origami HLS and the Folding Tool-Flow

The basic idea is to provide a tool for system designers to generate efficient hardware implementations. Origami HLS is written in C++ and provides an interface to read from and write to Matlab/Simulink models. The left side of Figure 4 shows a detailed view on the Origami HLS flow, where a Matlab/Simulink model is used as input. A Simulink parser generates a graph model. All system transformations use this graph model, where vertices with ports and edges are used to represent the DFG. After the system transformations are done, an HDL description of the resulting Simulink model is generated using the Matlab HDL coder.

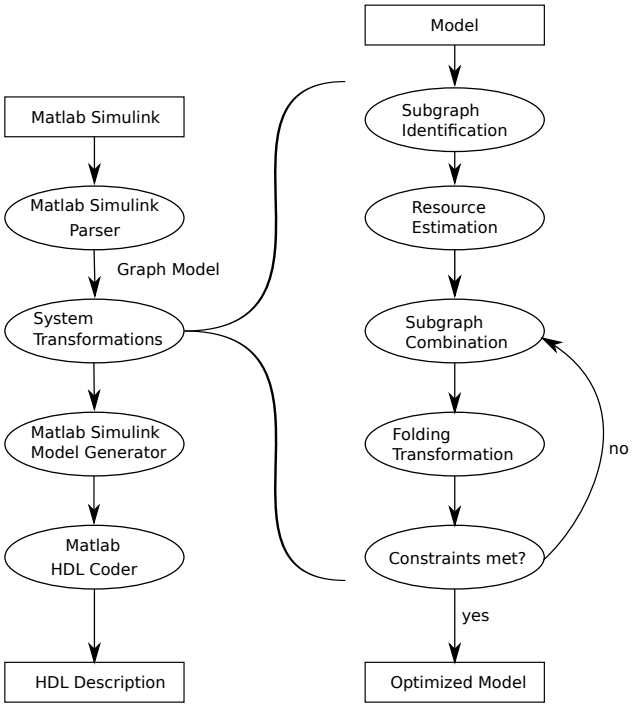


Figure 4: Origami HLS: Overall and Folding Flow

The main system transformation of Origami HLS is the automatic circuit optimization by applying the folding transformation. The right side of Figure 4 shows a detailed view on the folding flow of Origami HLS. Before the folding transformation with common subcircuits can be applied, the identification and combination has to take place. For the subgraph identification, we used *HSiGraM* [14] and added support for multiple inputs and outputs to the graph model. During the subgraph identification process, the MIS of every occurrence set is calculated with a heuristic as proposed in [12] and stored to be used during the folding-core selection. In that way, we want to apply the folding transformation with the most promising selections first. After the enumeration is done, non-overlapping occurrence sets can be used for combined folding. After that,

an estimation function or the user is able to decide whether Origami HLS will continue with the next selection to generate another design. The folding flow of Origami HLS combines the described algorithms to provide an automatic HLS folding tool-flow for flexible design and efficient hardware generation. Currently, the estimation function is simply based on the number of operations and occurrence frequency. We plan to implement more accurate resource estimation metrics in the future.

4. Experiments

Using Origami HLS, an experimental evaluation of the subgraph enumeration and identification using a benchmark set of 32-bit fixed-point precision Matlab/Simulink models is described in this section. The set contains widely used circuits in digital signal processing like the Park-Clarke transformation (PCT), a model with three PID controllers and two FIR filters. All benchmark models can be accessed online [18]. Additionally, a design-space exploration of all automatically generated and folded PCT models is provided.

Table 1: Subgraph identification analysis

circuit	edges	min. frequency	frequent subgraphs	occurrences	max. edges	avg. edges
FIR-8 tab	32	2	1830	3753	14	8.77
		3	37	167	6	3.24
		4	31	149	6	2.97
		5	6	49	2	0.67
FIR-12 tab	48	2	64316	129526	22	13.45
		3	785	2553	14	7.89
		4	170	786	10	5.45
		5	37	253	6	3.24
PCT	32	2	85	24	4	1.67
		3	9	55	3	1.11
		4	9	55	3	1.11
		5	6	45	2	0.67
Tripple-PID	48	2	11489	33095	16	9.45
		3	11489	33095	16	9.45
		4	7	51	2	0.57
		5	7	51	2	0.57

4.1. Subgraph Identification Statistics

Table 1 shows the results of the subgraph identification. The number of edges in each circuit are counted without the ones that are connected to the ports. The frequent-subgraph identification algorithm accepts subgraphs that occur at least with a specified minimum frequency. This input constraint can be used to reduce the search space. The number of identified frequent subgraphs and occurrences depends on the minimum frequency and the input model. The number of frequent subgraphs in both FIR filters and the PCT transformation drops significantly when the minimum frequency is changed from two to three. Two is the minimum frequency as a circuit must at least appear twice for sharing. The Tripple-PID controller model shows this behavior when the minimum frequency is larger than three. This is reasonable, because the model contains three times the same

structure. One can also observe from Table 1 that the number of frequent subgraphs found is not only based on the number of edges in the input model, but on the maximum number of edges in the largest identified frequent subgraph. The PCT and the eight tab FIR model have the same number of edges, but the number of identified frequent subgraphs of the latter is larger by an order of magnitude. We can also observe this behavior by comparing the 8 tab and the 12 tab FIR filters. While the number of edges in the input model rises by 50% from 32 up to 48, the number of identified frequent subgraphs grows by a factor of 35.

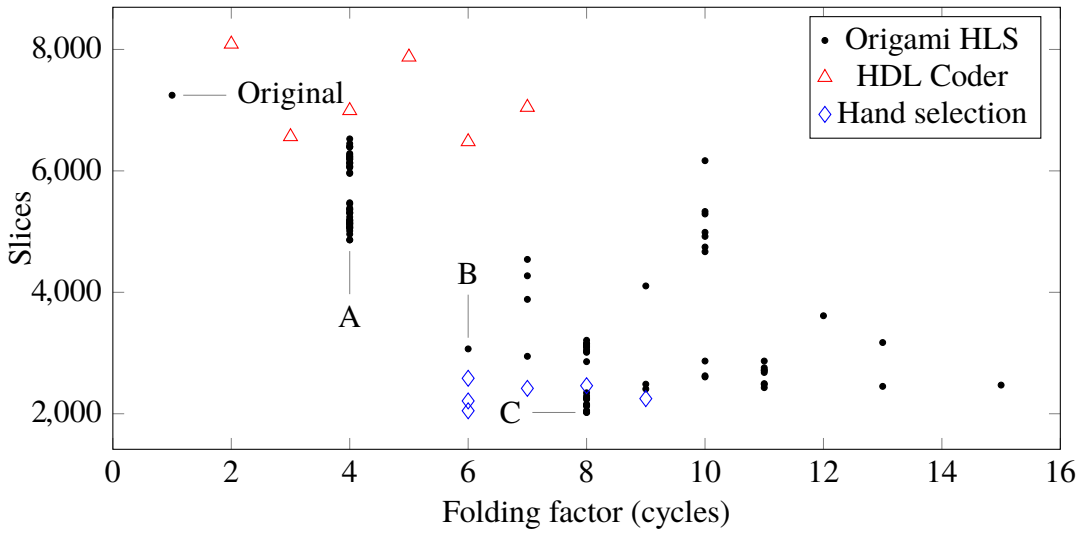
4.2. Synthesis Experiments for the PCT

In this section, we investigate the synthesis results of the folded PCT models. The verification of the folded models was done with Matlab 8.5(2015a), the VHDL code was generated with HDL coder (v3.6) and the target FPGA was a Xilinx Virtex 6 (xc6vlx75t-2ff484). The synthesis results were created with Xilinx ISE 13.4. After the frequent-subgraph identification with a minimum frequency of two, an MIS of every frequent subgraph was calculated to remove overlapping of occurrences. These maximum independent sets were stored as occurence sets to be used for folding-core combination. Finally, all combinations of compatible occurrence sets were constructed and used for a combined folding transformation of the original model, where every occurrence set represents a different folding core. Due to the restriction to MISes, other existing solutions will be omitted. Still, 98 Matlab/Simulink models where different combinations of folding cores are used could be generated and synthesized using HDL coder and ISE. As a comparison, we generated solutions with the resource-sharing function of Matlab HDL Coder as well as hand-selected ones that were published in an earlier evaluation [16] of the same model.

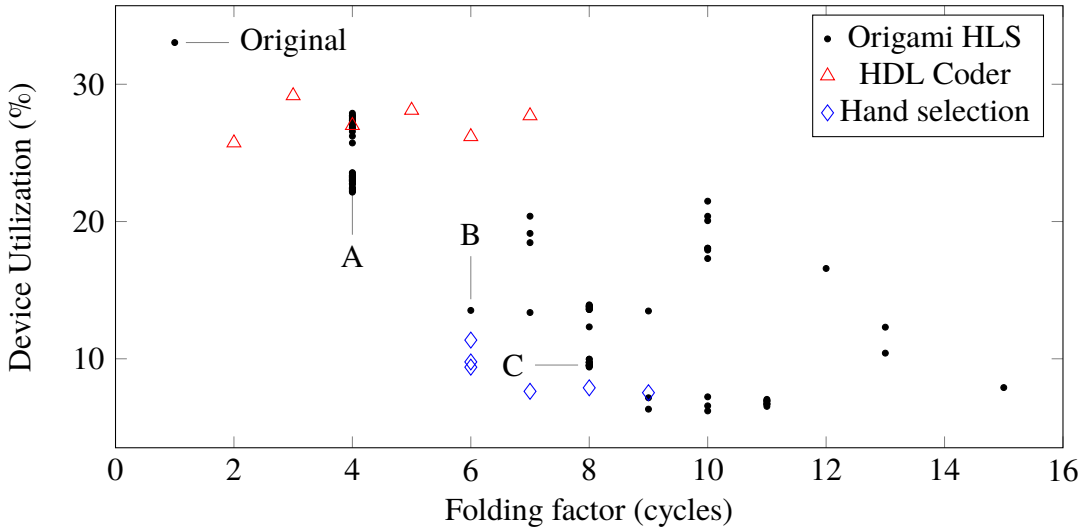
Figure 5 shows the synthesis results of the folded PCT circuits for a Virtex 6 FPGA. Every point describes the *folding factor* in clock cycles and the occupied FPGA resources of a folded model. The folding factor is identical to the number of clock cycles required to compute one sample, which is the length of the schedule. For this experiment we used an ASAP scheduler with hardware constraints. The operations for this experiment are not pipelined.

Slice results where the usage of DSP blocks was prohibited are shown in Figure 5a. One can observe that the original model consumes the most slices (7246) and has the smallest folding factor. All of our folded models consume fewer slices and show higher folding factor than the original one. The pareto-optimal solutions ($A - C$) are marked. The solution that is marked as C, for example, is able to reduce the number of occupied slices by 72.1% down to 2021 at the cost of eight cycles computation. Clearly, there are lot of worse points in the design space and metrics for pruning bad folding core combinations have to be implemented in the future. The solutions provided by the Matlab HDL Coder with different sharing factors could not perform well in comparison to the Origami HLS folding flow. In two cases the folded circuit is even larger than the original one and no case is an improvement to our solutions. In terms of slice reduction, Origami HLS performs almost as well as the hand-optimized solutions. The hand selection was able to reduce the number of slices used down to 2049 with a folding factor of six. The resource reduction with the presented automatic tool-flow is able to reduce the used slices to an equal number, but lacks in folding factor efficiency. The reason for that is that there is currently no support for splitting an occurrence set into several folding cores. Analysis shows that this was done in the best hand-optimized solution.

Figure 5b shows the synthesis results for the same experiment with the additional use of DSP blocks. To compare circuits that use a combination of Slices and DSP blocks is difficult. We use the



(a) Without DSPs



(b) With DSPs

Figure 5: PCT synthesis results (Virtex 6)

percentage device utilization of the target FPGA, which is given as $DU = \frac{1}{2} \cdot \frac{usedSlices}{totalSlices} + \frac{1}{2} \cdot \frac{usedDSPs}{totalDSPs}$. Using the scaling of $\frac{1}{2}$, a $DU = 1$ describes a 100% utilization of the device arithmetic. This metric can be used to consider different relevance (situation-dependent) of the resources, e.g., measured as absolute silicon area consumption, by simply adjusting the two scaling factors accordingly. Using this metric, the Matlab HDL Coder results compared to the original model perform better, but are still not good overall compared to our solutions. The circuits that were transformed with Origami HLS clearly result in a better device utilization at the cost of a increased folding factor compared to the hand-selected ones, providing interesting area/clock-cycle tradeoffs.

We can conclude that our automatic combined common subcircuit folding flow is able to generate good solutions and performs close to the solutions using hand-optimized folding cores. The smallest device utilization was automatically generated by Origami HLS at the cost of throughput.

Nevertheless, it is the goal to close the gap to the hand-optimized solutions within future work.

5. Conclusion

In this paper, we presented our high-level synthesis tool-flow for automatic folding with a combination of common subcircuits that is embedded into Origami HLS. Additionally, a theoretical discussion about using subgraphs for folding was provided. The identification and combination of common subcircuits is done automatically by Origami HLS, followed by the folding transformation. In that way, a system designer is able to generate optimized and validated hardware from an unscheduled design. The results in Section 4 show that the combination of occurrence sets for folding on the MBD level has a significant impact on the synthesized circuits properties. While some folded circuits occupy almost the same area as the input model while operating with a much lower throughput, others are using up to 69% less resources than solutions provided by the Matlab HDL Coder resource sharing. The results also show that it is possible to effectively trade off resources and clock cycles by using common subcircuits and combining occurrence sets for folding. One can observe that the hand-optimized solutions in some cases perform better than Origami HLS. This encourages an improvement of the common subcircuit combination. The further investigation of the combination of different folding cores is the consequential step. The presented results indicate that the number of all occurrences and occurrence sets might become intractable for larger designs. Therefore, subgraph mining heuristics suited for applying the folding transformation and accurate selection metrics have to be developed or adapted in the future.

References

[1] Adler, M., K. E Fleming, A. Parashar, M. Pellauer, and J. Emer: *Leap Scratchpads: Automatic Memory and Cache Management for Reconfigurable Logic*. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 25–28. ACM, 2011.

[2] Bringmann, B. and S. Nijssen: *What is Frequent in a Single Graph?* In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 858–863. Springer, 2008.

[3] Canis, A., J. Choi, M. Aldham, V. Zhang, A. Kammoona, J. H Anderson, S. Brown, and T. Czajkowski: *LegUp: High-Level Synthesis for FPGA-based Processor/Accelerator Systems*. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 33–36. ACM, 2011.

[4] Cheng, H., X. Yan, and J. Han: *Mining Graph Patterns*. In *Frequent Pattern Mining*, pages 307–338. Springer, 2014.

[5] Chung, E. S, J. C Hoe, and K. Mai: *CoRAM: An In-fabric Memory Architecture for FPGA-based Computing*. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 97–106. ACM, 2011.

[6] Cong, J. and W. Jiang: *Pattern-based Behavior Synthesis for FPGA Resource Reduction*. In *Proceedings of the 16th International ACM/SIGDA Symposium on Field Programmable Gate Arrays*, pages 107–116. ACM, 2008.

[7] Cong, J., B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang: *High-Level Synthesis for FPGAs: From Prototyping to Deployment*. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 30, pages 473–491. IEEE, 2011.

[8] Coussy, P., D. D. Gajski, M. Meredith, and A. Takach: *An Introduction to High-Level Synthesis*. In *IEEE Design & Test of Computers*, volume 26, pages 8–17, 2009.

[9] Deissenboeck, F., B. Hummel, E. Jürgens, B. Schätz, S. Wagner, J. F. Girard, and S. Teuchert: *Clone Detection in Automotive Model-based Development*. In *Proceedings of the 30th International Conference on Software Engineering*, pages 603–612. ACM, 2008.

[10] Fiedler, M. and C. Borgelt: *Support Computation for Mining Frequent Subgraphs in a Single Graph*. In *5th International Workshop on Mining and Learning with Graphs*, 2007.

[11] Hadjis, S., A. Canis, J. H Anderson, K. Nam J. Choi, S. Brown, and T. Czajkowski: *Impact of FPGA Architecture on Resource Sharing in High-Level Synthesis*. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 111–114. ACM, 2012.

[12] Hochbaum, D.: *Approximation Algorithms for NP-hard Problems*. In *ACM SIGACT News*, volume 28, pages 40–52. ACM, 1997.

[13] Huthmann, J., B. Liebig, J. Oppermann, and A. Koch: *Hardware/Software Co-Compilation with the Nymble System*. In *Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), 2013 8th International Workshop on*, pages 1–8. IEEE, 2013.

[14] Kuramochi, M. and G. Karypis: *Finding Frequent Patterns in a Large Sparse Graph*. In *Data Mining and Knowledge Discovery*, volume 11, pages 243–271. Springer, 2005.

[15] M. Kumm and Sittel, P. and K. Möller and M. Hardieck and P. Zipf: *Origami HLS*, 2016. `http://www.uni-kassel.de/go/origami`.

[16] Möller, K., M. Kumm, C. F. Müller, and P. Zipf: *Model-based Hardware Design for FPGAs using Folding Transformations based on Subcircuits*. Second International Workshop on FPGAs for Software Programmers (FSP 2015), 2015.

[17] Parhi, K. K: *VLSI Digital Signal Processing Systems: Design and Implementation*. John Wiley & Sons, 2007.

[18] Sittel, P. and M. Kumm and K. Möller and M. Hardieck and P. Zipf: *Simulink Benchmarks*, 2017. `http://www.uni-kassel.de/go/simulink_benchmarks`.

[19] Sun, W., M. J Wirthlin, and S. Neuendorffer: *FPGA Pipeline Synthesis Design Exploration Using Module Selection and Resource Sharing*. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 26, pages 254–265. IEEE, 2007.

[20] Van Den Hurk, J. and J. AG Jess: *System Level Hardware/Software Co-Design: An Industrial Approach*. Springer Science & Business Media, 2013.

[21] Venkataramani, G. and K. Kintali and S. Prakash and S. van Beek: *Model-Based Hardware Design*. In *Proceedings of the International Conference on Computer-Aided Design*, pages 69–73. IEEE, 2013.

[22] Wolf, W.: *Computers As Components: Principles of Embedded Computing System Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001, ISBN 1-55860-541-X.

[23] Yan, X. and J. Han: *gSpan: Graph-Based Substructure Pattern Mining*. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 721–724. IEEE, 2002.

[24] Zhang, P., M. Huang, B. Xiao, H. Huang, and J. Cong: *CMOST: A System-Level FPGA Compilation Framework*. In *Proceedings of the 52nd Annual Design Automation Conference*, pages 158:1–158:6. ACM, 2015.