FPGA Latency Optimization Using System-level Transformations and DFG Restructuring

Daniel Gomez-Prado, Maciej Ciesielski, and Russell Tessier
Department of Electrical and Computer Engineering
University of Massachusetts, Amherst
{*dgomezpr, ciesiel, tessier*}*@ecs.umass.edu*

***Abstract*—This paper describes a system-level approach to improve the latency of FPGA designs by performing optimization of the design specification on a functional level prior to high-level synthesis. The approach uses Taylor Expansion Diagrams (TEDs), a functional graph-based design representation, as a vehicle to optimize the dataflow graph (DFG) used as input to the subsequent synthesis. The optimization focuses on critical path compaction in the functional representation before translating it into a structural DFG representation. Our approach engages several passes of a traditional high-level synthesis (HLS) process in a simulated annealing-based loop to make efficient cost trade-offs. The algorithm is time efficient and can be used for fast design space exploration. The results indicate a latency performance improvement of 22% on average versus HLS with the initial DFG for a series of designs mapped to Altera Stratix II devices.**

I. Introduction

As field-programmable gate arrays (FPGAs) mature, the level of design representation used to target the devices continues to move upward from the register transfer level (RTL) to algorithmic and behavioral level specifications. Much FPGA design exploration work has focused on optimizations which are performed during traditional high-level synthesis (HLS), such as operation scheduling, register binding, and functional unit allocation. These algorithms are typically applied to a dataflow graph (DFG) which represents the desired computation. In many HLS systems, these graphs are treated as a fixed starting point, limiting the exploration of design implementations. Flexible FPGA architectures provide an opportunity for algorithmic approaches which restructure a DFG at a level above traditional HLS.

The use of FPGAs for design implementation necessitates specific DFG restructuring that accommodates the lookup tables, memory blocks and multipliers commonly found in the devices. For example, whenever two or more functional units write into a register, a multiplexer is required at the input to the register. In the case of FPGAs, a multiplexer made from lookup tables may require as much logic as a second functional unit, altering tradeoffs [1][2]. Critical path analysis during FPGA design mapping must carefully consider both FPGA functional block and interconnect delay at multiple hierarchical levels. These resources are configurable on a per-application level, allowing for a variety of area and performance tradeoffs. For HLS, operations require multiple clock cycles, so minimizing design latency requires effective functional unit use in addition to achieving a high design clock frequency.

In this work, a high-level design representation, the Taylor expansion diagram (TED), is used to transform dataflow graphs prior to the final application of traditional high level synthesis optimizations in an effort to improve the latency of FPGA designs. This analysis considers FPGA-specific optimization such as critical path compaction including expected routing when evaluating expected FPGA design latency. TED optimizations involve a series of design restructurings based on desired design operations. In this paper we show that it is possible to rapidly explore and modify dataflow graph representations at the behavioral level to improve FPGA design latency with limited impact on design area. An interesting aspect of our approach is the use of some HLS operations during TED optimization to evaluate the benefit of specific optimization steps. Effectively, a minimal set of HLS operations is provided ”in the loop” in assessing the impact of higher-level transforms. A full set of standard HLS operations is performed once the TED-based restructuring is complete. Our new design flow consists of the following phases:

1) High-level algorithmic transformation using TEDs.
2) High-level synthesis using GAUT [3]. This tool is used to evaluate intermediate behavioral design representations and convert the final behavioral design to RTL.
3) RTL design synthesis and physical design (place and route) using Altera Quartus II for commercial Stratix II FPGA devices.

We demonstrate that TED transformations can directly lead to an average of 22.6% post-mapped improvement in design latency for a set of previously-used HLS benchmarks mapped to Stratix II FPGAs versus mapping using only the initial, static DFG.

II. FPGA Design Latency Optimization during High Level Synthesis

Most high-level synthesis optimizations for FPGAs consider critical path reduction and multiplexer minimization. Although these optimizations are performed during HLS, rather than at a higher design level, they provide insight into the types of design explorations that can be considered by TEDs.

HLS algorithms have been widely explored in the context of latency optimization. The problem of register allocation and binding and its impact on the final architecture of synthesized

978-3-9815370-0-0/DATE13/©2013 EDAA

designs have been extensively studied. In Chen and Cong [1], a register binding algorithm with multiplexer optimization is modeled as a minimum cost flow in a network, and then a greedy algorithm is used to optimize performance. The problem of register binding for clock period minimization without register overhead is formulated in Huang and Chen [4]. In Cong *et al.* [5], the overall resource usage of functional units, registers and multiplexers is simultaneously optimized. The synthesis process is not broken into a sequence of optimization steps, as is traditionally practiced. Instead, the scheduler transmits global optimization information between each step of the algorithm. In Kim and Liu [6], a simultaneous register and functional unit binding algorithm targeting multiplexer input reduction to shorten the total interconnect length is developed. The improvements are based on the observation that not all functional units operate at the same time. When a functional unit is idle, it can be reconfigured as pass-through logic for data transfer, reducing interconnect requirements. Although these approaches provide significant steps forward in latency optimization, they do not attempt to *restructure* the dataflow graph targeted to FPGAs in an effort to enhance register allocation and binding.

Perhaps the work most similar to ours is the low-power architectural synthesis system (LOPASS) [7]. This approach performs a simulated-annealing optimization over the entire synthesis process of scheduling, resource selection and allocation, functional unit binding, register binding, and interconnection estimation to effectively reduce power. The main objective of this approach is to reduce the overall connectivity between functional units and registers (including multiplexers) and to reduce the total design interconnections, which ultimately leads to a reduction in power consumption. In their work, the DFG is fixed, and an annealing algorithm is applied to swap the type of functional units being used (e.g., Wallace multiplier versus other multiplier types), merge functional units, and swap functional unit operands.

Our method is complementary to the previous approaches as it restructures a dataflow graph *prior* to high level synthesis in an attempt to create a DFG which minimizes design latency by increasing design clock frequency, reducing the number of clock cycles per high-level operation, or both. The modified DFG is then used as input to a high level synthesis flow where the approaches mentioned above can still be performed as part of the synthesis process.

III. Review of Taylor Expansion Diagrams

Taylor expansion diagrams were originally developed in the context of formal verification for data-intensive computations and arithmetic datapaths. A TED is a canonical, graph-based data structure that can efficiently represent designs expressed as multivariate polynomial expressions, often encountered in signal processing and computer graphics applications. A multivariate polynomial expression, $F(x_1, \cdots, x_n)$, is decomposed recursively using Taylor series expansion, one variable at a time. The resulting decomposition is stored as a directed acyclic graph, the TED.

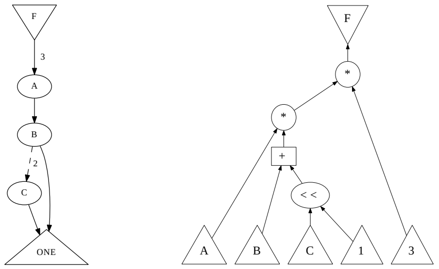


Fig. 1. a) TED of $F = 3A(B + 2C)$; b) DFG of $F = 3A(B + 2C)$.

To illustrate the concept of functional-level transformations supported by TED, consider a simple computation $F = 3A(B+2C)$, whose TED is shown in Fig. 1(a). Solid edges in the graph correspond to multiplications, dotted edges represent additions, weights on edges represent constant multiplications, and node ONE is a multiplication by 1. Because of the canonical nature of a TED (for a given variable order, in this case A, B, C) the function encoded in the TED can be interpreted as $F = 3A(B + 2C)$, which requires two multiplications, one addition and one shift. Fig. 1(b) shows a DFG obtained from this TED by performing TED decomposition under this variable order.

DFG Generation: In general, an ordered, minimized TED can be converted to a DFG with a minimum number of arithmetic operations by means of functional decomposition of the TED [8] involving factorization and common subexpression elimination (CSE). The arithmetic expression obtained by such a decomposition, called *Normal Factored Form* (NFF), is unique and minimal (in the number of arithmetic operators) for a TED with a fixed variable order. The normal factored form for the TED in Fig. 1 is $F = 3A(B + 2C)$. A structural DFG representation can then be readily constructed by replacing the arithmetic operations (additions and multiplications) with hardware operators, ADD and MULT, as shown in Fig. 1(b).

It should be noted that, unlike NFF, the DFG representation is not unique. While the number of operations remains fixed, a DFG can be further restructured and/or balanced to minimize latency. This offers an additional potential for optimization in which the best DFG can be constructed, depending on the required cost function, be it resource optimization, latency optimization, or an optimization under resource or latency constraints. Several methods employed by logic and HLS can be used for this purpose [9]. In addition, constant multiplications can be replaced by shifters and adders [8].

DFG Selection and Optimization: While TED decomposition minimizes the total number of arithmetic operations in the resulting normal factored form (and in the corresponding DFG), it does not address the issue of optimizing the final hardware implementation. Specifically, it does not minimize the number of hardware resources (adders and multipliers) actually used in the final, scheduled implementation; neither does it minimize the latency of the design. Such an optimization is only possible by performing *scheduling* and *resource allocation* on the structural level for the selected DFG. That is, the usage of the operators and the resulting design latency depends on the scheduling and allocation steps of the subsequent high level synthesis that has not been considered during the TED decomposition.

For the TED-based decomposition to be practical, it must gain insight into the construction of an actual DFG from the resulting NFF. It can then choose the structure which requires fewer resources (to minimize area) or provides more hardware parallelism (to minimize latency). The enhancements applied to the TED decomposition to address this issue include: 1) hashing all the extracted terms to avoid resource replication; 2) modifying the factorization and extraction methods to extract the minimum clique partitioning; and 3) forcing all irreducible TEDs to become reducible through a guided variable ordering according to the selected optimization goal.

A. TED-based Decomposition System (TDS)

A guided TED-based decomposition provides a means for effective design space exploration. This idea is illustrated in Fig. 2. In a traditional synthesis flow, a *single* DFG is extracted from an initial (functional) specification and used as input to high-level synthesis to generate the final hardware implementation. Optimization of the resulting implementation is therefore limited to local modifications between the solutions that are obtainable from only that DFG.

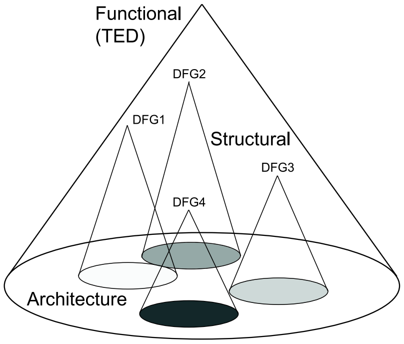


Fig. 2. Space exploration with TDS.

Fig. 2 shows a set of such solutions, each represented as a cone associated with a DFG. To improve the solution, an attempt can be made to transform the DFG into another DFG. However, such a transformation, if at all possible, is limited to a set of structural modifications of the graph, such as tree height reduction and balancing, which are limited in scope. In contrast, transformation of the functional specification, obtained by means of TED decomposition, can produce a family of functionally equivalent DFGs. One such DFG can be selected on the basis of a given objective and design constraints to generate the final hardware implementation.

This approach has been implemented in a system called TDS [8], which transforms the function extracted from a design specification into a TED and uses a host of TED-based decomposition and DFG optimization techniques to obtain an optimized DFG; it then passes the modified DFG to a high level synthesis tool, in this case GAUT. The TDS system has been used as a framework for design latency optimization for FPGA designs, which is described in the remainder of the paper.

IV. System Level Exploration

Although the TED data structure encodes a functional representation of the design, it provides limited structural information necessary for final implementation. Therefore, it is difficult to directly use a TED graph to characterize important structural design properties, such as the complexity of the steering logic, the number of multiplexers, or the amount of resources required by the control unit.

In this work, a *pseudo critical path discovery* algorithm is implemented and used to partition the TED into two clusters. One cluster targets latency minimization and the other targets area minimization. Then we use the resource usage in each cluster as a cost function to guide the TED space exploration. The overall objective of this approach is to reduce the number of iterations between the TED decomposition and high level synthesis by pruning those solutions that are not likely to improve the design latency obtained after FPGA place and route.

Estimating resource usage within the TED: The *as soon as possible* (ASAP) and *as late as possible* (ALAP) schedules derived as a bi-product of TED decomposition, which involves factorization and common subexpression elimination, give an early estimate of the resource requirements of a decomposed TED. For example, the TED shown in Fig. 3(a) is decomposed into a set of product terms (PT) and sum terms (ST), as shown in Fig. 3(b). The resulting structure is then balanced hiearchically to minimize the delay of the DFG generated from this TED. Specifically, each term can be implemented as a balanced tree to minimize its contribution to the overall latency, and the higher level structure composed of nodes $PT1$ and $ST1$ can then be implemented as a balanced tree, while considering the internal path delay imposed by each of its terms. Furthermore, when subtrees have different path delays, the slack (obtained from the ASAP and ALAP schedules) is used to reduce the resource requirements of the tree with the smallest path delay.

For example, the sum term depicted at the left in Fig. 3(c) by a balanced tree seems to imply that two adders are required, but when the delay of the operators are considered, the slack between both trees can be used to reduce the resource requirement. Assuming that the MULT operator has a delay of 2 and the ADD operator has a delay of 1, the delays of the sum and product term are 3 and 4, respectively. Therefore, it is possible to accommodate one more addition in the adder tree or delay one of its operations by one cycle, thus reducing the total number of adders in the scheduled design to 1.

Computing the critical path delay from a TED: Since a TED is an acyclic graph, its critical path delay can be found in $O(n + e)$, where n and e denote the number of nodes and edges in the TED. The critical path delay algorithm traverses all nodes of the TED and computes the height of the

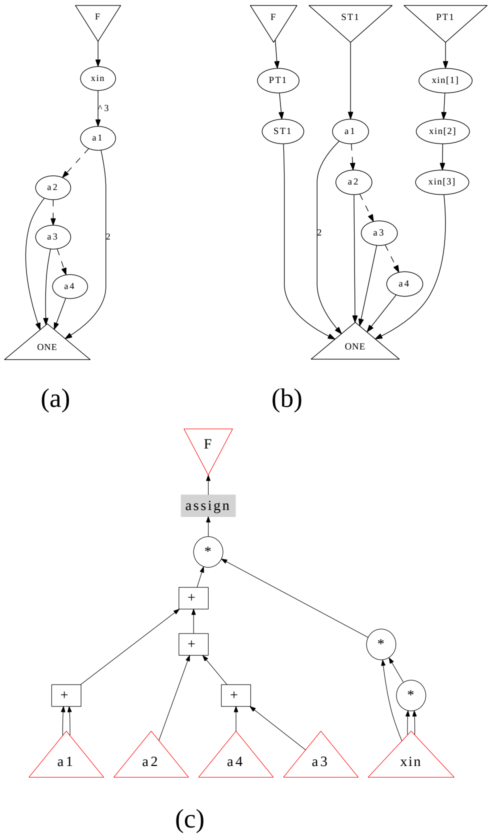


Fig. 3. (a) Initial TED for the function $F = x_{in}^3(2a_1 + a_2 + a_3 + a_4)$. (b) Decomposed TED with a product and a sum term, PT1 and ST1. (c) Resulting netlist (DFG) with latency equal to 3 MULT and a resource requirement of 1 MULT and 1 ADD.

adder tree formed by the incoming edges of a node. When an edge connecting a node to its children contains a register, the height contributed by that child is assigned 0 as if it were a primary input. Another special case arises when a node belongs to a multiplicative or an additive chain. In this case, the height is not computed immediately but postponed until the entire chain is discovered, at which point the local height is adjusted accordingly. The critical path computation obtained from the TED helps generate an initial clustering for target optimizations. Critical paths are considered for delay minimization and paths with slack are considered for area minimization. The clustering is dynamically updated because the discovered critical path delay assumes unlimited resources and, hence, some misclassification is bound to occur.

Iterative high level synthesis: It is fair to say that, when targeting FPGAs, the area and delay results obtained during high level synthesis can be far from accurate. Even though these metrics could be somewhat improved by logic synthesis tools, interconnect might ultimately be the principal cause of delay in an FPGA design. We use multiple iterations of simplified high-level synthesis operations to explore architectures that improve the design latency after mapping and routing the design into an FPGA. We use the metrics derived from the TED not to discover the best solution, but rather to prune the search.

The following design written in C is used as an example to demonstrate the core procedure of the iterative synthesis.

```
main(int a, int b, int c, int * out) {
  const static int mem[5];
  int pathr, path4, path5;
  int path1 = a+b+mem[0];
  int path2 = b*mem[1]*c*a;
  if (path1>path2) path4 = path1;
  else path4 = path2;
  int path3 = path2*mem[2];
  if (path2<path3) path5 = pathr;
  else path5 = path3;
  pathr = path2;
  *out = path4*(mem[2]+b) + mem[3]*path5;
}
```

The high level synthesis tool, GAUT, first compiles the C code with a modified gcc compiler into a control data flow graph (CDFG). The obtained CDFG, shown in Fig. 4(a), expresses all the operations specified in the original code. The CDFG is then transformed through scheduling, allocation and binding into an architecture as shown in Fig. 4(b).

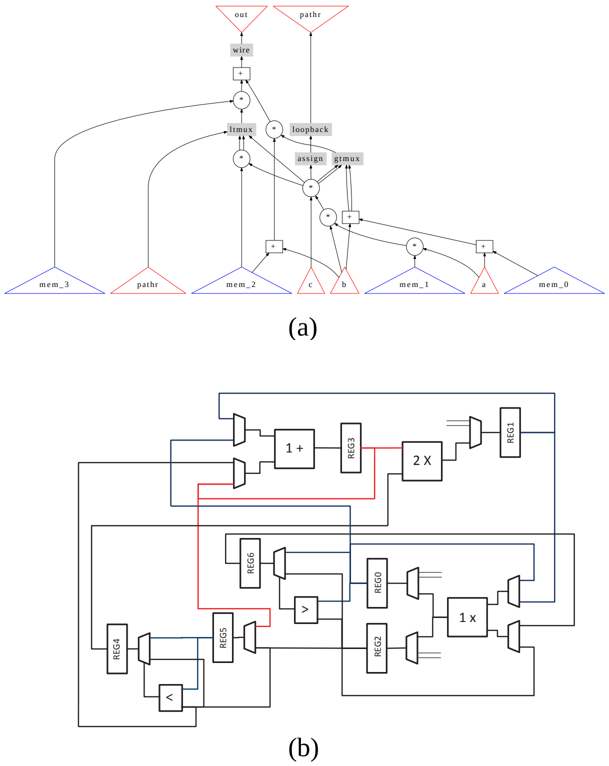


Fig. 4. (a) The CDFG netlist generated by GAUT from the C design. (b) The original datapath architecture generated from the CDFG.

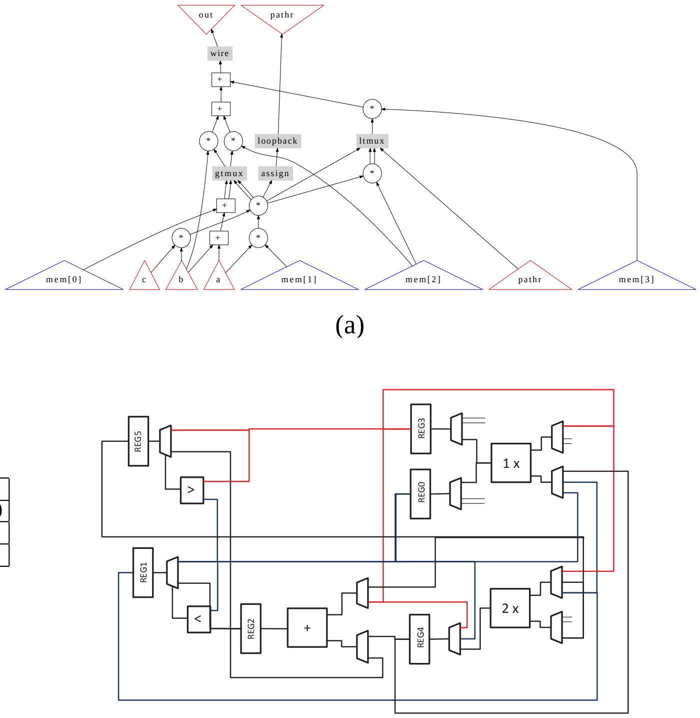
The list scheduler of GAUT prioritizes the set of operations to be executed next according to the arrival and required completion times. When more than one operation of the same type needs to be completed at a certain time, the scheduler instantiates another resource of that type. The resource allocation and binding then establishes which operations are performed in which hardware resource, which in turn determines how many registers will be needed to store the partial computations throughout the data flow. The number of multiplexers generated when connecting the registers to and from the operator ports is optimized [7]. These steps are performed in a loop which includes TED restructuring to determine the lowest latency design. In the loop, GAUT uses the target design latency (clock period × number of clock cycles) and clock period of the design to determine the number of cycles that the scheduler can use. If the scheduler cannot reach the target latency, regardless of the number of instantiated resources, the synthesis process fails to successfully complete the schedule for that TED and further TED restructuring can be considered.

Design	GAUT						Quartus II			
	Cycles	Mux	Reg	Area	+	×	LUTs	FFs	Freq (MHz)	Des. Lat. (ms)
Orig. design	14	96	7	208	1	2	533	126	159.5	87.8
Restr. design	11	128	6	208	1	2	528	107	160.6	68.5

TABLE I
GAUT AND QUARTUS II REPORT FOR THE ORIGINAL DFG (FIG. 4(B)) AND THE RESTRUCTURED DESIGN (FIG. 5(B))

The results reported by GAUT (high-level synthesis) and Quartus II (logic synthesis, place and route) for the original design in Fig. 4 are shown in Table I. The design latency corresponds to the clock period obtained by Quartus multiplied by the number of cycles scheduled by GAUT. It is important to note in Table I that two multipliers are instantiated and that for the given original CDFG, GAUT cannot generate any other architecture which results in fewer clock cycles. Additionally, note that the second multiplier instantiated by GAUT, in the datapath architecture shown in Fig. 4(b), has no multiplexer at its input ports, because the second multiplier was instantiated by the scheduler after other multiplications have already been scheduled on the other multiplier. The key idea, therefore, is to restructure segments of the CDFG that could benefit from this second instance to reduce the number of clock cycles and the overall design latency.

The CDFG shown in Fig. 4(a) is imported into the TDS system, and the different algebraic paths in the CDFG are translated into multiple TEDs. Each extracted CDFG path becomes a TED output, which can be classified by its critical path delay. Each TED variable has an associated initial arrival time that is used to compose a more suitable CDFG to reduce design latency. After performing TED optimizations in an attempt to reduce resource requirements, the cost of the new CDFG is evaluated using GAUT. The annealing algorithm accepts and rejects the new costs which calibrates the window size into which a TED performs reordering and decomposition. For example, Fig. 5(a) shows an improved CDFG and the corresponding architecture is shown in Fig. 5(b). This new design requires fewer cycles to complete the computation and improves the design latency of the overall design, as shown in Table I.



(a)

(b)

Fig. 5. (a) Reduced latency CDFG obtained after TED manipulations. (b) Datapath for the optimized CDFG.

V. Experimental Results

We have successfully integrated the TDS, GAUT, and Quartus system and applied it to a series of DSP algorithms [10]. As mentioned in the previous section, TDS and GAUT are used in a simulated annealing loop where GAUT provides design latency estimates. Each iteration of the annealing algorithm includes a TED reordering operation and TED-based decomposition based on paths with critical delay. Non-critical paths are transformed to minimize area. Then, the GAUT tool is used to generate a datapath and get an estimated hardware cost. As the temperature in the annealing algorithm decreases, the TED-based decomposition and TED-to-DFG transformation options are gradually reduced to those that provide the best results for the current design. The final, best result is output by GAUT in VHDL and run through Altera Quartus synthesis, place and route to validate the improvements reported by GAUT. For each benchmark, we used the smallest Stratix II device that would fit the design. The Quartus II tool was set to maximum effort in all cases with an unreachable target clock frequency of 1 GHz. DSP block extraction and automated shift register chain insertion was turned off.

Table II illustrates the results obtained using our approach. Execution times include both the time required by TED exploration and the time consumed by GAUT. The GAUT tool has been used in a number of academic projects and its

HLS algorithms for binding, allocation, and scheduling are well documented [3]. The column named *original* corresponds to results obtained by processing the original design CDFGs with GAUT and Quartus without TED manipulation. The column named *TDS* corresponds to the results obtained from optimizing the TEDs. The results in the table indicate that our goal of reducing design latency (labelled *latency* in the table) has been achieved. Overall, the average design latency has been reduced by about 22% versus GAUT synthesis without TED manipulation. This latency decrease caused an average lookup table (LUT) increase of 10.1%. In all cases, design frequency improved. The worst case TDS/GAUT execution time across all designs was 154 seconds, a small value in comparison to the half hour Quartus compile times.

VI. Conclusions

In this paper a new FPGA design latency optimization tool has been demonstrated. Our system uses TEDs to restructure dataflow graphs and a high-level synthesis tool to quickly evaluate the datapath design space at a behavioral level. HLS passes are used in a loop with TED restructuring to fully explore the design space. Once a final DFG is selected, GAUT performs a final HLS pass and the resulting design is sent to FPGA physical design tools for final implementation. For a collection of benchmark designs our approach shows a 22% reduction in design latency versus the direct use of HLS binding, allocation, and scheduling on an unmodified graph. In the future, further operations at the HLS level could be used to guide TED manipulation.

VII. Acknowledgement

This work has been supported in part by a grant from the National Science Foundation, award CCF-0702506.

References

[1] D. Chen and J. Cong, "Register binding and port assignment for multiplexer optimization," in *Proc., Asia and South Pacific Design Automation Conference*, Jan. 2004, pp. 68–73.

[2] J. Cong and J. Xu, "Simultaneous FU and register binding based on network flow method," in *Proc., IEEE/ACM Design and Test in Europe Conference*, Mar. 2008, pp. 1057–1062.

[3] P. Coussy, et al., "GAUT: A High-Level Synthesis Tool for DSP Applications," *High-Level Synthesis: From Algorithm to Digital Circuits*, Springer, Berlin, Germany, 2008.

[4] S.-H. Huang and C.-H. Cheng, "Minimum-period register binding," *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, vol. 28, no. 8, pp. 1265–1269, Aug. 2009.

[5] J. Cong, B. Liu, and J. Xu, "Coordinated resource optimization in behavioral synthesis," in *Proc., IEEE/ACM Design and Test in Europe Conference*, Mar. 2010, pp. 1267–1272.

[6] T. Kim and X. Liu, "A functional unit and register binding algorithm for interconnect reduction," *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, vol. 29, no. 4, pp. 641–646, Apr. 2010.

[7] D. Chen, et al., "LOPASS: a low-power architectural synthesis system for FPGAs with interconnect estimation and optimization," *IEEE Trans. on VLSI Systems*, vol. 18, no. 4, pp. 564–577, Apr. 2010.

[8] M. Ciesielski, D. Gomez-Prado, Q. Ren, J. Guillot, and E. Boutillon, "Optimization of data-flow computation using canonical TED representation," *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, vol. 28, no. 9, pp. 1321–1333, Sept. 2009.

[9] G. DeMicheli, *Synthesis and Optimization of Digital Circuits*. New York, N.Y.: McGraw-Hill, Inc., 1994.

[10] K. Parhi, *VLSI Digital Signal Processing Systems*. New York, N.Y.: John Wiley & Sons, Inc., 1999.

Design			Original	TDS	%Δ
conv3x3	GAUT	Cycles	12	9	-25.0
		Registers	145	260	
		Muxes	240	160	
		Hw +,-,×,≪	48,-,64,-	80,-108,-	
	Quartus	Freq (MHz)	37.1	38.3	
		LUTs	15139	13292	-12.2
		FFs	2332	4033	
		Latency (ms)	324	235	-27.3
		Time (s)		154	
Cooktom	GAUT	Cycles	9	8	-11.0
		Register	12	18	
		Muxes	256	256	
		Hw +,-,×,≪	-,3,1,2	-,3,1,2	
	Quartus	Freq (MHz)	163.4	181.7	
		LUTs	247	218	-11.7
		FFs	197	303	
		Latency (ms)	55	44	-20.0
		Time (s)		7	
Lattice	GAUT	Cycles	15	13	-13.9
		Register	13	12	
		Muxes	208	272	
		Hw +,-,×,≪	2,-,4,-	2,-,4,-	
	Quartus	Freq (MHz)	150.0	151.0	
		LUTs	592	359	-39.4
		FFs	223	325	
		Latency (ms)	100	86	-13.9
		Time (s)		11	
Prodmat	GAUT	Cycles	8	6	-25.0
		Register	180	288	
		Muxes	128	64	
		Hw +,-,×,≪	28,-,54,-	72,-,216,-	
	Quartus	Freq (MHz)	122.3	224.0	
		LUTs	2693	7349	173.0
		FFs	1448	2310	
		Latency (ms)	65	27	-59.1
		Time (s)		109	
Sobel	GAUT	Cycles	13	13	0.0
		Register	203	204	
		Muxes	544	448	
		Hw +,-,×,≪	17,30,-,15	28,30,-,-	
	Quartus	Freq (MHz)	135.5	144.2	
		LUTs	7386	7708	4.4
		Registers	3231	3277	
		Latency (ms)	96	90	-6.1
		Time (s)		89	
Volterra	GAUT	Cycles	19	17	-10.5
		Register	13	18	
		Muxes	288	400	
		Hw +,-,×,≪	1,-,4,-	1,-,2,3	
	Quartus	Freq (MHz)	138.0	138.8	
		LUTs	463	502	8.4
		Registers	227	289	
		Latency (ms)	138	123	-11.0
		Time (s)		21	
Winograd	GAUT	Cycles	9	9	0.0
		Register	11	11	
		Muxes	208	192	
		Hw +,-,×,≪	2,1,3,1	3,1,4,1	
	Quartus	Freq (MHz)	270.0	278.6	
		LUTs	183	248	35.0
		FFs	119	138	
		Latency (ms)	33	32	-3.1
		Time (s)		9	
Geomean Freq			127.9	144.4	12.9
Geomean LUTs			1207	1329	10.1
Geomean FFs			537	739	37.4
Geomean Design Latency			90	70	-22.6

TABLE II
TABLE OF RESULTS FOR OPTIMIZED AND ORIGINAL DFGS. TIME (S) INDICATES THE RUN TIME OF THE TDS/GAUT ANNEALING LOOP. LATENCY (MS) INDICATED THE DESIGN LATENCY OF THE FINAL DESIGN.