Efficient Overlay Architecture Based on DSP Blocks

Abhishek Kumar Jain, Suhaib A. Fahmy, Douglas L. Maskell
School of Computer Engineering
Nanyang Technological University, Singapore
Email: abhishek013@ntu.edu.sg

***Abstract*—Design productivity and long compilation times are major issues preventing the mainstream adoption of FPGAs in general purpose computing. Several overlay architectures have emerged to tackle these challenges, but at the cost of increased area and performance overheads. This paper examines a coarse grained overlay architecture designed using the flexible DSP48E1 primitive on Xilinx FPGAs. This allows pipelined execution at significantly higher throughput without adding significant area overheads to the PE. We map several benchmarks, using our custom mapping tool, and show that the proposed overlay architecture delivers a throughput of up to 21.6 GOPS and provides an 11–52% improvement in throughput compared to Vivado HLS implementations.**

I. Introduction and Related work

The effectiveness of FPGAs for application acceleration has been demonstrated across a wide range of applications. However, apart from a few specialized domains, this has not resulted in mainstream use of FPGAs. One reason is that design productivity remains a major challenge, restricting the effective use of FPGA accelerators to niche disciplines involving highly skilled hardware engineers. High-level synthesis (HLS) has been proposed as a way of addressing the limited design productivity and manpower capabilities associated with hardware design by focusing on high level functionality instead of low-level implementation details. However, prohibitive compilation times (specifically the place and route times in the backend flow) still limit productivity and mainstream adoption [1]. Hence, there is a growing need to make FPGAs more accessible to application developers who are accustomed to software API abstractions and fast development cycles. Researchers have approached this problem from many angles, including through the use of pre-compiled hard macros in the tool flow [2], partial reconfiguration and overlay architectures.

In the area of overlay architectures, both fine-grained and coarse-grained architectures that map to the fine grained FPGA fabric have been proposed. Coarse-grained overlay architectures have a number of advantages, including reduced compilation times, smaller configuration data sizes, application portability across devices, better design reuse, improved programmability and run-time management [3], [4], [5], [6]. However, these advantages come at a cost in terms of area and performance overheads which limit overlay architectures to relatively small to moderate applications. The primary metrics than need to be improved include: frequency and peak throughput of the overlay [5], configuration data size and configuration time [4], and the throughput and area-time product of the compute kernels on the overlay [3].

QUKU is a coarse grained overlay [3] designed to bridge the gap between soft processors and customized circuit implementations. It achieves an frequency of 175 MHz on a Xilinx Virtex 4, with speedups of up to 140× and an area-time product improvement of up to 40× compared to soft-core processor implementations. The VDR overlay [7] is an array of coarse-grained heterogeneous processing elements (PEs) interconnected by a set of programmable switches. It achieves a frequency of 172 MHz on an Altera Stratix III with a 9× improvement in performance over a NIOS II soft processor. No configuration or performance metrics comparing to customized circuit implementations were provided.

Intermediate Fabric (IF) [4] was proposed to reduce the compilation time associated with the conventional mapping process targeting FPGAs. It comprises a 16-bit datapath supporting fully parallel and pipelined implementation of compute kernels. A 700× speedup over vendor place and route tools is achieved at the cost of a 34–44% area overhead. It achieves an frequency of $\approx$ 125 MHz on a Altera Stratix III. An overlay architecture optimized for high frequency and throughput was proposed in [5]. It consists of a 24×16 overlay with a nearest-neighbor-connected mesh of 214 routing cells and 170 heterogeneous functional units (FU). A frequency of 355 MHz was reported when implemented on an Altera Stratix IV. No performance metrics comparing with customized circuit implementations were reported.

Many of the overlay architectures from the literature were developed with little consideration for the underlying FPGA architecture. Embedded hard blocks, such as DSP blocks provide area, performance, and power advantages over the equivalent function implemented directly in the logic fabric. Existing overlay architectures do not specifically use these blocks, other than possible inference by the synthesis tools which does not always lead to maximum throughput. Xilinx DSP blocks have been demonstrated as enabling high speed soft processors through their dynamic programmability [8].

In this paper, we show how the Xilinx DSP48E1 primitive can be used, at near to its theoretical limits, as a programmable PE in an efficient overlay architecture for

978-1-4799-9969-9/15 $31.00 © 2015 IEEE
DOI 10.1109/FCCM.2015.15

pipelined execution of compute kernels, improving both performance and area. The main contributions can be summarized as:

- An RTL implementation of a pipelined overlay architecture for Xilinx FPGAs using the DSP48E1 primitive, achieving near maximum frequency.
- A mapping flow that takes a high level description of a compute kernel, bypasses the conventional FPGA compilation process, and maps to the overlay.

II. Overlay Architecture and Mapping Tool

The architecture of the overlay consists of a traditional, island-style topology, arranged as a virtual homogeneous two-dimensional array of tiles. Each tile instantiates virtual routing resources and a functional unit (FU) and each border instantiates routing to form the boundary at the top and right of the array.

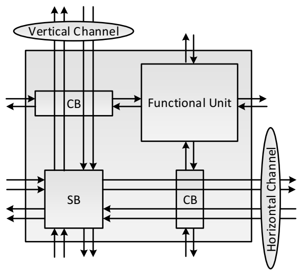


Figure 1: Overlay tile architecture.

A. *Overlay Tile Architecture*

Each tile contains a functional unit (FU) and virtual routing resources, as shown in Fig. 1. Virtual routing resources are switch boxes (SBs), connection boxes (CBs), and horizontal and vertical channels consisting of 16-bit routing tracks to support a 16-bit datapath. Multiple tracks can exist in both the horizontal and vertical directions, forming channels. The number of tracks in a channel is the channel width (CW), and as this increases, application routing becomes easier but at a higher area cost.

The functional units (FUs) provide the resources for the mathematical or logical operations of the application and consist of a programmable processing element (PE), MUX based reordering logic and shift register LUT (SRL) based synchronization logic for balancing pipeline latencies, as shown in Fig. 2. The FU has 4 input and 4 output ports logically organised at the 4 cardinal points. We use the DSP48E1 primitive flexibly to implement computation functions.

B. *Architectural Optimization and Design Issues*

This section describes some of the architectural optimizations to improve the performance of our overlay architecture.

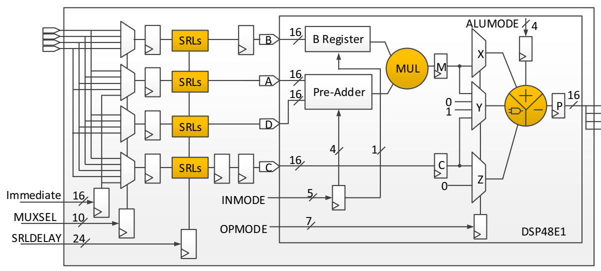


Figure 2: Functional unit architecture.

1) Frequency and Throughput Optimization: To achieve a high frequency, thus maximizing application throughput, we enable all three pipeline stages of the DSP48E1 primitive in the FU, add a register at the output of each reordering multiplexer, and register the outputs of the SRLs. As a result, the total latency of the FU is 7 clock cycles. Additionally, to further increase frequency and eliminate the possibility of combinational loops in the resulting HDL we use a 16-bit register at the output of each MUX in the CB.

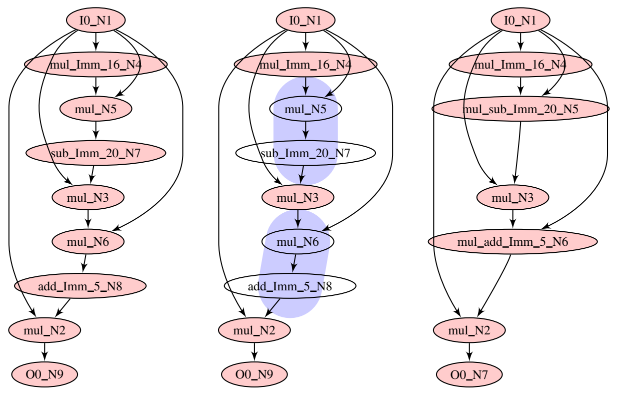
2) Distinguishing PE Inputs: Any of the four inputs of the FU can connect to any of the four inputs of the PE. However, as the input pins of the DSP48E1 block are not logically equivalent, we must implement reordering logic for each input pin using a multiplexer as shown in Fig. 2. The four outputs of the FU are wired to the single output of the DSP block.

3) Latency Imbalance at FU Inputs: With a large pipeline latency in each node, and the need for signal timing to be correctly matched, balancing pipeline latencies at the different FUs is necessary. Delays must be added to paths to ensure that inputs to each FU are correctly aligned. We use variable-length shift registers, implemented using the LUT-based SRL32 primitives. The depth of the variable shift registers is set to introduce the right amount of delay for each path, and the maximum can be set to 16, 32, or 64 cycles, depending on the flexibility desired. We experimentally determine the optimal depth of the variable shift registers for our benchmark set.

C. *Mapping Tool*

Implementation of the overlay architecture relies on the conventional hardware design flow using vendor tools. This process is performed offline and so does not impact the rapid compilation of an application to the overlay. We use an in-house automated mapping flow to provide a rapid, vendor-independent, mapping to the overlay as described below.

1) C to DFG transformation: From a C description of the compute kernel, the tool generates a dataflow graph (DFG). A node in the DFG executes in a pipelined manner and produces an output only when all of its inputs are ready. Fig. 4(a) shows the nodes and edges in an example DFG.



(a) Input DFG (b) Node-merging (c) Mapped DFG

Figure 4: DSP48E1 aware mapping.

2) DSP48E1 Aware Mapping: To reduce the number of compute nodes, we merge multiple nodes based on the capabilities of the DSP48E1 primitive. For example, we can use multiply-subtract and multiply-add to collapse N5-N7 and N6-N8 in Fig. 4(a) into N5 and N6 of Fig. 4(c), respectively. As a result, DSP48E1 aware mapping requires only 5 DSP blocks instead of the 7 required if each node were mapped to a single FU, as in many other overlays. This results in the mapped DFG shown in Fig. 4(c).

3) Placement and Routing of FU Netlist: We make use of VPR to map nodes in the graph to FUs, and edges onto the 16-bit tracks, effectively using VPR at a higher level of abstraction than its intended purpose.

4) Latency Balancing: To determine the latency imbalance at each node, we developed a tool to parse the VPR output files and generate a routing resource graph, which is then used to generate the configuration of the overlay (including the depth of each SRL) for the compute kernel and also for generating experimental results including latency imbalance at each node, and maximum latency imbalance in the graph.

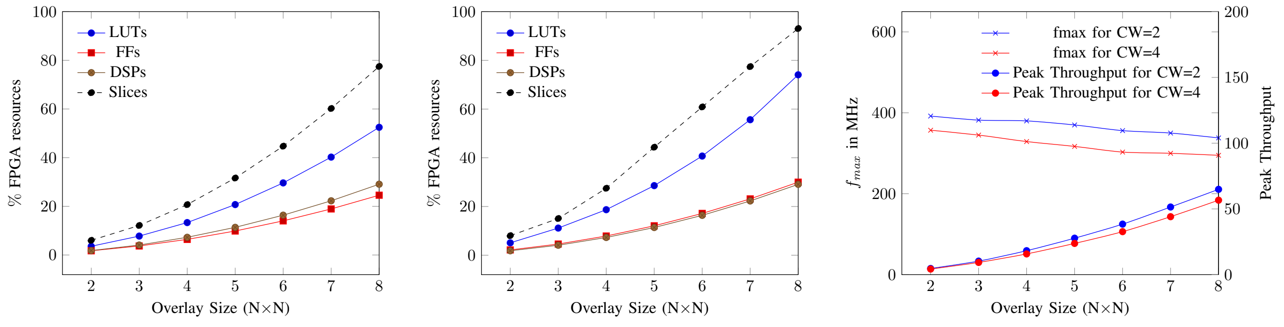
III. Experimental Evaluation

We synthesize and map the overlay using Xilinx ISE 14.6 onto a Xilinx Zynq XC7Z020 and evaluate its performance using 14 compute kernels. Table I shows the resources required for the FU, FU configuration registers (FUCR), SB, SB configuration registers (SBCR), CB and CB configuration registers (CBCR), for CW=2 and CW=4. The overlay tile contains 1 FU, 1 SB, 2CBs and their configuration registers, while a border contains 1 SB, 1 CB and their configuration registers. Hence for CW=2, an overlay tile consumes 416 LUTs, 390 FFs and 1 DSP block and a border tile consumes 112 LUTs and 76 FFs. For CW=4, an overlay tile consumes 544 LUTs, 474 FFs and 1 DSP block while a border consumes 192 LUTs and 120 FFs.

Table I: FPGA resource usage for overlay components having CW=2 and CW=4

Resource			CW=2				CW=4			
	FU	FUCR	SB	SBCR	CB	CBCR	SB	SBCR	CB	CBCR
LUTs	224	0	64	0	64	0	128	0	96	0
FFs	176	66	0	8	64	6	0	16	96	12
DSPs	1	0	0	0	0	0	0	0	0	0

Peak Throughput, Resource Usage and Frequency with Varying Size: Since the DSP48E1 can support three operations, an overlay of size $N \times N$ can support up to $3 * N^2$ operations. Hence the peak throughput of an overlay of size $N \times N$ is equal to $3 * N^2 * F_{max}$. Resource consumption on the Zynq, as a percentage of total resources, for CW=2 and CW=4 is shown in Figs. 3(a) and 3(b), respectively. All these numbers are post-place and route. Fig. 3(c) shows the decrease in frequency and increase in peak throughput as the size of the overlay increases. A modest drop in frequency is observed, but even for large overlays, a frequency of 300 MHZ is achieved, which is very respectable for the Zynq. We were able to map a 15×15 overlay with CW=2 onto a Virtex-6 (XC6VLX240T) device at 315 MHz.



(a) Resources usage for CW=2. (b) Resources usage for CW=4. (c) Frequency trend.

Figure 3: Resource usage and frequency on the Zynq.

Table II: Experimental results for the implementations of the benchmark set

Benchmark	Benchmark Characteristics				Routability		Overlay Results			HLS Implementation Results				
	i/o nodes	op nodes	merged nodes	savings	CW=2	CW=4	Latency	MLI	GOPS	Latency	Fmax	GOPS	Slices	DSPs
chebyshev	1/1	7	5	28%	3×3	3×3	49	36	2.59	11	278	5.0	40	8
sgfilter	2/1	18	10	44%	4×4	4×4	54	31	6.66	11	278	5.0	40	8
mibench	3/1	13	6	53%	3×3	3×3	47	35	4.81	9	295	3.8	81	3
qspline	7/1	26	22	15%	5×5	5×5	76	64	9.62	21	244	6.3	126	14
poly1	2/1	9	6	33%	3×3	3×3	34	22	3.33	12	285	2.56	62	4
poly2	2/1	9	6	33%	3×3	3×3	29	7	3.33	11	295	2.65	45	4
poly3	6/1	11	7	36%	3×3	3×3	31	11	4.07	12	250	2.75	52	6
poly4	5/1	6	3	50%	2×2	2×2	24	12	2.22	7	312	1.87	36	3
atax	12/3	60	36	40%	—	6×6	72	58	18.0	13	263	15.8	78	18
bicg	15/6	30	18	40%	—	6×6	46	32	9.0	7	270	8.1	91	18
trmm	18/9	54	36	33%	—	7×7	58	30	16.2	8	222	11.9	105	36
syrk	18/9	72	45	37%	—	7×7	41	19	21.6	10	250	18	237	24

Benchmarks, Routability Test and Channel Width Determination: Table II shows the characteristics of the DFGs extracted from the benchmarks. The first 8 benchmarks are polynomials taken from [9]. Next 4 benchmarks are linear algebra kernels from the PolyBench/C 3.2 benchmark suite [10]. We use the size of 3 for single dimensional arrays and 3×3 for two-dimensional arrays to obtain moderate sized DFGs. We determined the minimum overlay size required for each benchmark that achieved routability. Table II shows that linear algebra kernels are not routeable for an architecture with CW=2. Hence, we divide the benchmarks into two sets, set-I (first 8 benchmarks) and set-II (next 4 benchmarks). We prototype 2 overlay architectures: overlay-I (size=5×5) with CW=2 for benchmark set-I and overlay-II (size=7×7) with CW=4 for benchmark set-II.

Performance Evaluation: Overlay-I and Overlay-II have a configuration size of 287 Bytes and 700 Bytes, and can be configured entirely in 11.5 us and 28 us, respectively compared to 31.6 ms for the entire PL using PCAP, providing a 1000× improvement in reconfiguration time. Maximum latency imbalance is the largest difference between any two inputs for a node, labelled MLI in Table II. We calculate the throughput for each benchmark as shown in Table II. We also generate the RTL of the compute kernels using Vivado HLS 2013.2 in order to compare throughput, using fully unrolled compute kernels. Table II shows the results for the Vivado HLS implementations of the benchmark set.

IV. Conclusion

We have presented an FPGA overlay architecture that uses the Xilinx DSP48E1 primitive as a programmable PE, resulting in an efficient overlay architecture for pipelined execution of compute kernels, with improved area and performance metrics. We map several benchmarks, using our custom mapping tool, and show that the proposed overlay can deliver a throughput of up to 21.6 GOPS and provides an 11–52% improvement in throughput compared to Vivado HLS implementations. We aim to reduce the area of the overlay further through careful optimizations of the routing architecture.

References

[1] G. Stitt, “Are field-programmable gate arrays ready for the mainstream?” *IEEE Micro*, vol. 31, no. 6, pp. 58–63, 2011.

[2] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and B. Hutchings, “HMFlow: accelerating FPGA compilation with hard macros for rapid prototyping,” in *IEEE Int. Symp. Field-Programmable Custom Computing Machines*, 2011.

[3] N. W. Bergmann, S. K. Shukla, and J. Becker, “QUKU: a dual-layer reconfigurable architecture,” *ACM Transactions on Embedded Computing Systems*, vol. 12, p. 63, Mar. 2013.

[4] G. Stitt and J. Coole, “Intermediate fabrics: Virtual architectures for near-instant FPGA compilation,” *IEEE Embedded Systems Letters*, vol. 3, no. 3, pp. 81–84, Sep. 2011.

[5] D. Capalija and T. S. Abdelrahman, “A high-performance overlay architecture for pipelined execution of data flow graphs,” in *Int. Conf. Field Prog. Logic and Appl.*, 2013.

[6] A. K. Jain, K. D. Pham, J. Cui, S. A. Fahmy, and D. L. Maskell, “Virtualized execution and management of hardware tasks on a hybrid ARM-FPGA platform,” *Journal of Signal Processing Systems*, vol. 77, no. 1–2, pp. 61–76, Oct. 2014.

[7] D. Capalija and T. Abdelrahman, “Towards synthesis-free JIT compilation to commodity FPGAs,” in *IEEE Int. Symp. Field Prog. Custom Computing Machines*, 2011, pp. 202–205.

[8] H. Y. Cheah, F. Brosser, S. A. Fahmy, and D. L. Maskell, “The iDEA DSP block based soft processor for FPGAs,” *ACM Transactions on Reconfigurable Technology and Systems*, vol. 7, no. 3, p. 19, 2014.

[9] B. Ronak and S. A. Fahmy, “Evaluating the efficiency of DSP block synthesis inference from flow graphs,” in *Int. Conf. on Field Prog. Logic and Appl.*, 2012, pp. 727–730.

[10] L. Pouchet, “Polybench: The polyhedral benchmark suite, v3.2,” 2011.