2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines

DeCO: A DSP Block Based FPGA Accelerator Overlay With Low Overhead Interconnect

Abhishek Kumar Jain*, Xiangwei Li*, Pranjul Singhai*, Douglas L. Maskell* and Suhaib A. Fahmy†
*School of Computer Engineering, Nanyang Technological University, Singapore
†School of Engineering, University of Warwick, United Kingdom
Email: {abhishek013, asdouglas}@ntu.edu.sg, s.fahmy@warwick.ac.uk

***Abstract*—Coarse-grained FPGA overlay architectures paired with general purpose processors offer a number of advantages for general purpose hardware acceleration because of software-like programmability, fast compilation, application portability, and improved design productivity. However, the area overheads of these overlays, and in particular architectures with island-style interconnect, negate many of these advantages, preventing their use in practical FPGA-based systems. Crucially, the interconnect flexibility provided by these overlay architectures is normally over-provisioned for accelerators based on feed-forward pipelined datapaths, which in many cases have the general shape of inverted cones. We propose DeCO, a cone shaped cluster of FUs utilizing a simple linear interconnect between them. This reduces the area overheads for implementing compute kernels extracted from compute-intensive applications represented as directed acyclic dataflow graphs, while still allowing high data throughput. We perform design space exploration by modeling programmability overhead as a function of overlay design parameters, and compare to the programmability overhead of island-style overlays. We observe 87% savings in LUT requirements using the proposed approach compared to DSP block based island-style overlays. Our experimental evaluation shows that the proposed overlay exhibits an achievable frequency of 395 MHz, close to the DSP theoretical limit on the Xilinx Zynq. We also present an automated tool flow that provides a rapid and vendor-independent mapping of the high level compute kernel code to the proposed overlay.**

I. Introduction and Related Work

Research efforts have shown the advantages of FPGA accelerators in a wide range of application domains [1]. However, while the rapidly increasing logic density and more capable hard resources like DSP blocks in modern FPGAs should make them applicable to a wider range of domains, the difficulty of hardware design, long compilation times, and poor design productivity are major issues preventing the mainstream adoption of FPGA based accelerators in general purpose computing [2]. Design productivity remains a major challenge, restricting the effective use of FPGA based accelerators to niche disciplines requiring highly skilled hardware design engineers.

One technique for addressing design productivity is to use high-level synthesis (HLS) [3]. Numerous design languages and environments have been developed to allow designers to focus on high level functionality instead of low-level implementation details. However, achieving the desired performance often still requires detailed low-level design engineering effort that is difficult for non-experts. Even as HLS tools improve in efficiency, prohibitive compilation times (specifically the place and route times in the backend flow) still limit productivity and mainstream adoption. Thus, there is a growing need to make FPGAs more accessible to application developers who are accustomed to software API abstractions and fast development cycles, hence the need to relook at how to exploit the key advantages of the FPGA hardware resource, beyond that which is achievable using HLS Tools.

Coarse-grained FPGA overlay architectures have been shown to be effective when paired with general purpose processors, offering software-like programmability, fast compilation, application portability and improved design productivity [4], [5], [6]. Application kernels can be rapidly compiled to target them [7] and they can be tightly coupled with general purpose processors [5] or as a part of the processor pipeline [8]. Run-time management of these accelerator-processor systems, including overlay configuration and data communication, using both an operating system (Linux) [6] and a hypervisor [5] have been demonstrated. Instead of requiring a full cycle through the vendor tools, overlay architectures present a simpler problem, that of programming an interconnected array of functional units (FUs). However, overlays are not intended to replace traditional hardware design tool flows and are instead intended to support FPGA usage models where programmability, resource sharing, context switch time, fast compilation, and design productivity are critical issues.

Although research in the area of overlay architectures has increased recently, the field is still in its infancy with only a few FPGA overlay architectures demonstrated in prototype form [5], [6], [7], [8]. These architectures enable general purpose hardware accelerators, allowing hardware design at a higher level of abstraction, but at a cost in terms of area and performance overheads due to limited consideration for the underlying FPGA architecture. These overheads prevent the realistic use of overlays in practical FPGA-based systems [8]. One of the main reasons for this poor performance is that many of the early overlay architectures are designed without serious consideration of the underlying FPGA architecture [9]. Previous work has shown that DSP block based overlays can greatly improve performance in terms of throughput by using the full capability of highly pipelined DSP blocks along with a flexible island-style interconnect architecture [10], [11]. However, supporting the full flexibility of island-style interconnect contributes to a significant area overhead. Furthermore, this level of interconnect flexibility is normally not required for implementing accelerators based on feed-forward pipelined datapaths, which in many cases have the general shape of inverted cones [12], [13].

978-1-5090-2356-1/16 $31.00 © 2016 IEEE
DOI 10.1109/FCCM.2016.10

A. Related Work

Extracting maximum performance from DSP blocks is rarely possible using generic RTL design. However, even large datapaths can reach close to the theoretical frequency limit if designed carefully around the structure and pipelining of these blocks [14].

Coarse grained overlay architectures have gained prominence for the hardware acceleration of compute kernels, as they offer fast compilation, hardware design at a higher level of abstraction, improved programmability and run-time management [4], [5], [6], [7], [8], [9], [10], [11], [15], [16], [17]. However, when implemented on top of a fine grained FPGA, these features come at the cost of additional area and performance overheads. Hence, significant recent research effort has aimed to reduce area overheads while improving performance.

An overlay architecture based on an island-style interconnect architecture with a channel width (CW) of two, referred to as an intermediate fabric (IF), was mapped to a Xilinx XC5VLX330 FPGA, along with a low overhead interconnect [18]. A baseline and the optimized overlay used 196 FUs (DSP blocks) resulting in a 14×14 processing array. The baseline overlay used 91K LUTs (44% of the available LUTs) with an F_{max} of 131 MHz while the optimized overlay used 50K LUTs (24% of the LUTs) with an F_{max} of 148 MHz. This resulted in a LUT/FU count of 465 and 255, respectively. However, the overlay did not fully utilize the DSP block capabilities as the FU only utilized the DSP multiplier.

A fully pipelined DSP block based overlay architecture [10] mapped to a Xilinx Zynq XC7Z020 used node merging to combine multiple compute kernel operations to better target the DSP block, resulting in an average reduction of 35% in the number of processing nodes required. An 8×8 overlay used 28K LUTs (52% of the LUTs) and achieved an F_{max} of 338 MHz resulting in a LUT/FU count of 437. A throughput better than the direct implementation of the benchmarks using Xilinx Vivado HLS was reported due to the deep pipelining used in the FU.

DySER [19] was proposed as a coarse grained overlay architecture for improving the performance of general purpose processors. Originally designed as a heterogeneous array of 64 functional units interconnected with a circuit-switched mesh network and implemented in an ASIC, the DySER architecture was improved and implemented along with the OpenSPARC T1 on to a Xilinx XC5VLX110T FPGA [8]. However, due to excessive LUT consumption, it was only possible to fit a 2×2 32-bit, a 4×4 8-bit, or an 8×8 2-bit overlay on the FPGA. An adapted version of a 6×6 16-bit DySER overlay was implemented on a Xilinx Zynq XC7Z020 [9] by using a DSP block as the PE, thus better targeting the architecture to the FPGA. A benchmark set of 8 compute kernels having up to 23 operations required a 5×5 overlay, consuming 34K LUTs (64% of the LUTs) on the Zynq, resulting in a LUT/FU count of 1360.

While some specialized overlays have also been proposed, such as QUKU [4], these require manual design and description based on an architectural template. QUKU was proposed as a rapidly reconfigurable coarse grained overlay architecture [4] designed to bridge the gap between soft processors and customized circuit implementations. It was implemented on a Xilinx Virtex-4 LX25 device as a fixed configuration array of processing elements (PEs) interconnected via an application specific customized interconnect, and achieved an F_{max} of 175 MHz. A similar approach was presented in [20] by merging the datapaths of a kernel set to generate a general overlay referred to as a supernet. Supernets were compared to island-style overlays and demonstrated an area overhead reduction of up to 9×.

B. Motivation

In reviewing the literature, we find that many proposed overlays are developed with little consideration for the underlying FPGA architecture. The presence of DSP block rich FPGA fabrics in modern devices, and previous work that has demonstrated how DSP blocks can be used for general processing at near to their theoretical limits [21], has resulted in the development of overlays with improved F_{max} and throughput [10]. Those overlays that do achieve good F_{max} and throughput [16], [10] still suffer from significant resource overheads associated with the routing network. Thus the resource overhead associated with overlay architectures, particularly the routing network, is still a concern. Many existing overlay architectures use general-purpose island-style interconnect which allows all FUs to communicate with each other at the expense of a significant area overhead. However, this interconnect flexibility is a significant over-provision.

A number of authors have instead proposed a linear array of interconnected FUs [15], [20] to improve resource utilization. A domain specific reconfigurable array, with a linear feed forward interconnect structure and with array dimensions determined by merging the datapaths of all DFGs, was proposed in [13]. However this approach resulted in heavy underutilization of FUs, with not more than 40% utilization when mapping DFGs. Instead, it was observed that better utilization could be achieved if the FU architecture was customized to better match the shape of the DFGs [13], which in many cases have the general shape of an inverted cone.

We propose performing an analysis of linear interconnect overlays from the perspectives of both DFG structure and interconnect flexibility, which we term the *programmability overhead*, aiming to customize the typical rectangular array of FUs to produce a cone shaped cluster of FUs utilizing a simple linear interconnect. This would reduce the area overheads when implementing compute kernels extracted from compute-intensive applications represented as directed acyclic dataflow graphs by better targeting the set of FUs, while still allowing high data throughput. To achieve this, we perform design space exploration of possible linear data flow overlay architectures by modeling the programmability overhead as a function of the design parameters. We also present an automated tool flow that provides a rapid and vendor-independent mapping of the high level compute kernel code to the proposed overlay. The main contributions in this paper can be summarized as:

- Design space exploration of linear overlay architectures by modeling programmability overhead
- A parameterized and area efficient RTL implementation of DeCO, a DSP-based efficient Compute Overlay, which

can operate at near to the DSP block theoretical maximum frequency
- An automated tool flow that takes a C description of a compute kernel and maps it to the proposed overlay by performing graph optimizations such as rebalancing, node merging and architecture aware rescheduling
- A quantitative performance comparison of DeCO against island-style overlays

The remainder of this paper is organized as follows: Section II presents interconnect architecture analysis including programmability overhead modeling and benchmark-set specific overlay design. In Section III, we present the architecture of the DeCO overlay. Section IV presents our automated mapping flow. In Section V, we present experiments that evaluate the overlay architecture and mapping tool. We conclude in Section VI and discuss our future work.

II. Interconnect Architecture Analysis

A linear interconnect architecture, consisting of a one-dimensional array of programmable homogeneous tiles where each tile contains a programmable routing network and a cluster of DSP block based FUs and data forwarding (DF) links is shown in Fig. 1. The DF links are required in each cluster to allow bypassing of the current tile, and thus they have a latency equivalent to that of the DSP block based FUs. The resource requirement should be significantly reduced from that of an island-style or nearest-neighbor architecture as the routing network is only required in one direction between FU stages. Additionally, the FU array is a simple, strictly balanced, pipelined structure with no feedback, and hence does not require any delay balancing or reordering logic which are required in the case of an island-style architecture. For example, the overlay in [10] requires reordering logic, synchronization logic, and routing network logic that consumes 6, 8 and 10 LUTs/bit/tile, respectively, to support connectivity between the DSP block based FUs. This represents a programmability overhead of 384 LUTs/FU for a 16-bit FU. However, in modern Xilinx fabrics there are 4 DSP blocks for every 10×16 slices, equivalent to 640 LUTs, resulting in a LUT/DSP ratio of 160.

An overlay architecture should ideally target an interconnect network with a programmability overhead of less than 160 LUTs/FU, so as to get the most out of the available FPGA resources, particularly the DSP blocks. To achieve this, we propose further customizing the number of FUs, DF links and the complexity of the routing network in each tile according to the set of compute kernels. This is similar to datapath merging [13] except that we only merge computation blocks of sequenced DFGs in a stage-wise manner, leaving a fully flexible interconnect between stages, hence retaining significant flexibility. This allows us to handle unknown DFGs, so long as the DFG can be scheduled on the merged datapath.

A. Programmability Overhead Modeling

For each tile, the number of DSP blocks, delay lines, and routing network complexity can be decided based on a set of compute kernels. The complexity of the routing network in the n^{th} tile depends on the resources in the n^{th} tile and $(n+1)^{th}$ tile, that is the number of DSP blocks and delay lines. The routing network can then be designed using an $(X_n + Y_n) \times (I * X_{n+1} + Y_{n+1})$ crossbar switch, where I is the number of FU inputs. The 16-bit version has 4 inputs as it is able to utilise the pre-adder in the DSP block, while the 32-bit version has 3 inputs as it cannot. X_n and Y_n are the number of DSP blocks and delay lines in n^{th} tile and X_{n+1} and Y_{n+1} are the number of DSP blocks and delay lines in $(n+1)^{th}$ tile.

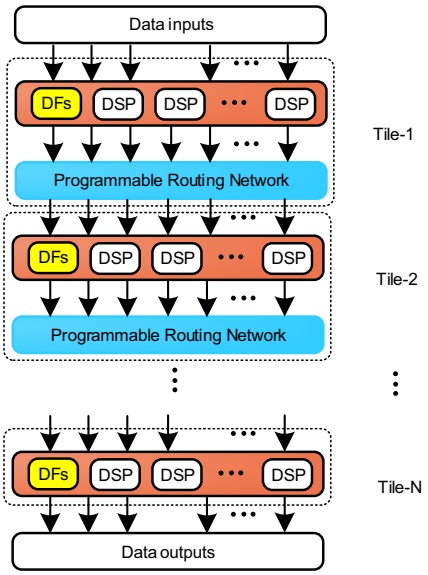

Fig. 1: Block diagram of linear dataflow overlay.

If L_n is the number of LUTs/bit required to build an $(X_n + Y_n)$:1 multiplexer, the programmability overhead (PO) of the overlay network, in LUTs/bit, is:

$$PO = \sum_{n=1}^{N-1} (L_n * (I * X_{n+1} + Y_{n+1})), \quad (1)$$

where N is the number of tiles in the overlay.

Given a set G of M sequenced DFGs, we refer to the number of operation nodes in the n^{th} stage as $g_m x_n$, the total number of stages as Ng_m, and the crossing edges as $g_m y_n$ in the m^{th} DFG, G_m. We can then find X_n, Y_n and N using:

$$X_n = \max(g_1 x_n, g_2 x_n, ...g_M x_n) \quad (2)$$

$$Y_n = \max(g_1 x_n + g_1 y_n, g_2 x_n + g_2 y_n, ...g_M x_n + g_M y_n) - \max(g_1 x_n, g_2 x_n, ...g_M x_n) \quad (3)$$

$$N = \max(Ng_1, Ng_2, Ng_3, ...Ng_M) \quad (4)$$

B. Set-Specific Overlay Design

We calculate the overlay design parameters and programmability overhead using a subset of DFGs from [8], given in Table I. The graph depth is the critical path length of the graph, while the graph width is the maximum number of nodes in one schedule time, both of which impact the ability to efficiently map a kernel to the overlay. The average parallelism

is the ratio of the total number of operations and the graph depth. We note that many of these DFGs are poorly balanced, so we firstly apply tree-balancing to all DFGs, which both reduces graph depth and better shapes the DFG. Next we apply DSP aware node merging [10], which better targets the DSP block based FU, and lastly we apply rescheduling techniques, possibly increasing the graph depth, which results in different FU requirements at each stage. This results in a number of different DFGs for the same benchmark, with different values for the overlay design parameters (N, X_n, and Y_n), resulting in different POs.

For example, the original *kmeans* benchmark [8] is shown in Fig. 2(a), and has a depth of 9 requiring 9 stages in a linear overlay. The rebalanced graph is shown in Fig. 2(b), and requires just 5 stages, with a significantly reduced latency. One example of node merging and rescheduling using ASAP scheduling is shown in Fig. 2(c) resulting in a reduced FU count. It should be noted that we observe a higher PO for the ALAP scheduled version, due to the additional DF resource needed to forward input data to the 2nd stage. The 3 implementations shown in Fig. 2 result in a PO of 6.25, 4.75 and 4.3 LUTs/bit per FU, respectively. Thus, we would choose DFG 3, the one with the smallest PO, as the candidate DFG for the design process.

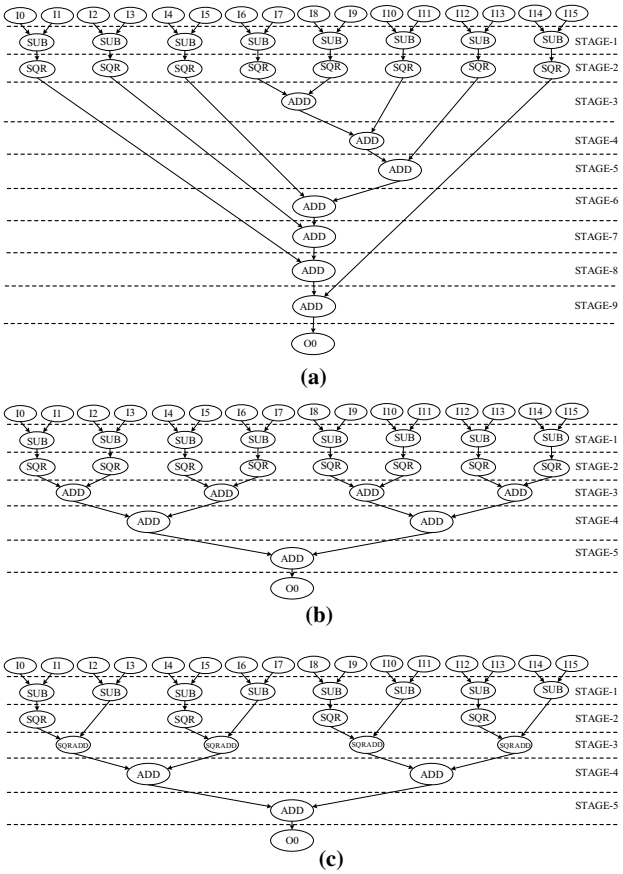

Fig. 2: Applied transformations including graph balancing and DSP aware node merging.

TABLE I: DFG characteristics of Benchmark set

No.	Benchmark Name	Characteristics (After transformation characteristics)				
		I/O nodes	graph edges	op nodes	graph depth	graph width
1.	fft	6/4	24(22)	10(8)	3(3)	4
2.	kmeans	16/1	39(35)	23(19)	9(5)	8
3.	mm	16/1	31(31)	15(15)	8(4)	8
4.	spmv	16/2	30(30)	14(14)	4(3)	8
5.	mri	11/2	24(21)	11(9)	6(5)	4
6.	stencil	15/2	30(23)	14(8)	5(3)	6

We repeat this process for the remaining DFGs in the benchmark set in Table I), to determine the best FU and DF configuration to support all compute kernels in the benchmark set. This results in a structure with different resource requirements in each scheduling stage, which we refer to as a cone [12]. This cone consists of 20 DSP block based FUs and four layers of connections networks, as shown in Fig. 3.

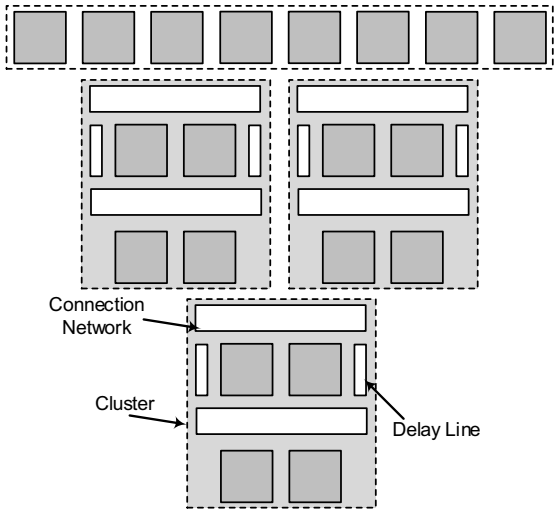

Fig. 3: Design of the optimized cone.

III. DECO OVERLAY ARCHITECTURE

Building on the ideas presented in the previous sections and the advantages of hard DSP macros for implementing high speed processing elements, we use the Xilinx DSP48E1 primitive as a programmable FU in the proposed overlay architecture. Unlike in other overlays from the literature [9], [10], [18] the interconnect network in the proposed overlay is very lightweight, enabling us to efficiently implement both 16-bit and 32-bit implementations. The overlay architecture of the two designs is slightly different, due to the characteristics of the DSP block. We also detail the physical mapping to the FPGA fabric and the resource usage. To achieve this we use Xilinx ISE 14.6 and a Verilog HDL description of the overlay targeting the Xilinx Zynq XC7Z020.

A. The 32-Bit Architecture

The Xilinx DSP48E1 primitive does not provide a full 32-bit implementation, even though some internal signals are much larger than 32-bits. The problem arises, because the pre-adder is just 25 bits wide, while the multiplier is 25×18-bit. This can be problematic for $long$ integer multiplication, but is suitable if all variables are restricted to the C language 32-bit int data type. This is because the default conversion rules for the int data type will truncate the result of an integer multiplication, discarding the most significant part. To avoid possible overflow with the 25-bit pre-adder, we choose not to use it, and instead bypass the pre-adder. The resulting architecture, showing the 32 bit connection network and the 32-bit FU, which is able to support the C language 32-bit int data type, is shown in Fig. 4. In this case, the FU uses the multiplier, the ALU, the three separate A, B and C ports for input data, and one output port P, as shown in Fig. 4, and can be configured to support various operations such as multiply-add, multiply-subtract, etc. The actual DSP48E1 function is determined by a set of control inputs that are wired to the $ALUMODE$, $INMODE$ and $OPMODE$ configuration registers. The DSP48E1 primitive is directly instantiated to provide total configuration control and allowing us to achieve a high frequency.

The topmost routing layer of the 20 FU overlay, shown in Fig. 3, requires connections to four FUs (with three inputs each) and four DF delay lines (with a single input). Thus, for full routing flexibility, the topmost routing network layer requires 16 8:1 multiplexers. However, during the design process we observed that there is no requirement for connectability between the left and right regions in the top part of the cone, and thus they do not need to be fully connected. Hence in the top connection layer, we use two separate smaller connection networks, each having 8 4:1 multiplexers. Similarly, in the second layer, we again isolate the left and right regions, allowing the use of 4:1 multiplexers. In the third connection layer, we combine the signals from the left and right regions, which now also only require 4:1 multiplexers. Thus, the resulting 32-bit cone, shown in Fig. 3, consists of a top layer of 8 FUs followed by three identical clusters of 4 FUs and 2 DFs. It has a total latency of 24 clock cycles with a theoretical PO of 2.1 LUTs/bit per FU, which is 40% less than the equivalent cone with full interconnect flexibility. Thus the theoretical LUT/FU ratio for the proposed 32-bit overlay is 67.2, which is well below the original target of 160 LUTs/FU.

B. Resource Usage for the 32-Bit Architecture

Each cluster consists of four FUs preceded by three individual 4:1 muxes at the input of each FU, and two DF delay lines preceded by a 4:1 mux, as shown for the 16-bit version in Fig. 6. The 32-bit FU requires one DSP block and four additional registers at the DSP input ports (18-bit register for B input, 25-bit register for A input, two 32-bit registers for C input) for pipeline balancing (as shown in Fig. 4), consuming 107 FFs. The 25-bit 2:1 multiplexer in front of the A input port consumes 13 LUTs and allows us to choose between 25 bits (for multiplication) and the 14 extra bits that need to be concatenated (for addition). Each FU connection network requires three 4:1 multiplexers with

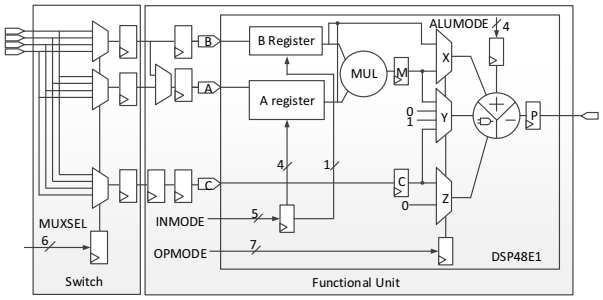

Fig. 4: The 32-bit functional unit and interconnect switch.

registered outputs, consuming 96 LUTs and 89 FFs. The delay line requires 32 LUTs (SRLs) and 32 FFs and the delay line input selection logic requires 32 LUTs and 32 FFs. This results in a total cluster resource consumption of 564 LUTs, 912 FFs and 4 DSP blocks. The cluster configuration register includes 16 bits for configuring each DSP block, 2 bits for each mux and 2 bits for delay line input selection logic. Hence, the cluster configuration register consumes 92 FFs.

The post-place and route resource consumption of the 32-bit cone is 2076 LUTs, 3984 FFs and 20 DSP blocks, and it achieves a frequency of 395 MHz, which approaches the theoretical limit for DSP blocks of 400 MHz on Zynq. For the DSP block based FU, programming the FU settings requires 16 configuration bits while programming the routing network requires 2 configuration bits per 4:1 multiplexer. Thus, the entire routing network requires 84 bits for the 32-bit cone and hence the entire cone can be reconfigured using just 404 bits (50.5 Bytes) of configuration data.

C. The 16-Bit Architecture

The 16-bit FU is similar to the 32-bit FU, except that it can now make use of the DSP block A input and the pre-adder, allowing additional instructions such as add-multiply and subtract-multiply-add. This results in the four input, one output FU, shown in Fig. 5. As with the 32-bit version, the DSP48E1 function is determined by a set of control inputs that are wired to the $ALUMODE$, $INMODE$ and $OPMODE$ configuration registers.

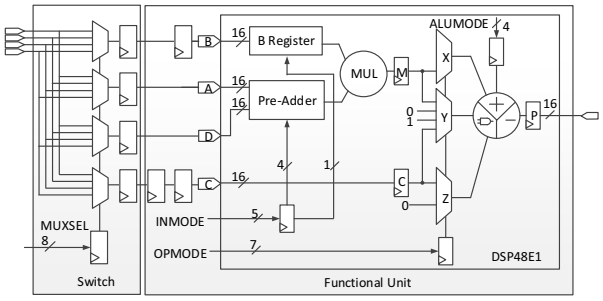

Fig. 5: The 16-bit functional unit and interconnect switch.

Because of the extra DSP block input (used by the pre-adder), the topmost routing layer requires connections to four FUs (with four inputs each) and four DF delay lines (with a single input). Again, because we do not need full routing flexibility, the topmost connection layer requires two sets of 10 4:1 multiplexers. Similar to the 32-bit version, we use 4:1 multiplexers in the other connection layers. The 16-bit cone also has a total latency of 24 clock cycles with a theoretical PO of 2.7 LUTs/bit per FU, which is again 40% less than the equivalent cone with full interconnect flexibility. The theoretical LUT/FU ratio for the proposed 16-bit overlay is 43.2, which is again significantly below the original target of 160 LUTs/FU.

D. Resource Usage for the 16-bit Architecture

In the 16-bit version of the cone, which also consists of three tiles with eight FUs at the top layer, each cluster consists of four FUs preceded by four individual 4:1 muxes at the input of each FU, and two DF delay lines preceded by a 4:1 mux, as shown in Fig. 6. The 16-bit FU requires one DSP block and three additional 16-bit registers at the DSP input ports for pipeline balancing (as shown in Fig. 5), consuming 48 FFs. Each FU connection network requires four 4:1 multiplexers with registered outputs, consuming 64 LUTs and 64 FFs, while the delay line requires 16 LUTs (SRLs) and 16 FFs and the delay line input selection logic requires 16 LUTs and 16 FFs. This results in a total 16-bit cluster resource consumption of 320 LUTs, 512 FFs and 4 DSP blocks. The cluster configuration register includes 16 bits for each DSP block configuration, 2 bits for each mux and 2 bits for delay line input selection logic, resulting in a cluster configuration register size of 100 FFs.

The post-place and route resource consumption of the 16-bit cone is 1368 LUTs, 2348 FFs and 20 DSP blocks, and it achieves a frequency of 395 MHz, which approaches the DSP theoretical limit of 400 MHz on Zynq. The overlay was subsequently mapped to a Xilinx Virtex-7 XC7VX690T-2 and achieved a frequency of 645MHz.

For the DSP block based FU, programming the FU settings requires 16 configuration bits while programming the routing network requires 2 configuration bits per 4:1 multiplexer. Thus, the entire routing network requires 108 bits for the 16-bit cone and hence the entire cone can be reconfigured using just 428 bits (53.5 Bytes) of configuration data.

IV. Mapping Tool

The main advantage of using an overlay is that application kernels can be mapped to it with software-like speed and programmability. The design and implementation of the overlay itself still relies on the conventional hardware design flow using vendor tools, but because it is done offline once, it does not impact subsequent application kernel implementation, which has its own toolchain. Typically, high level application kernels are mapped to device primitives, either directly in the case of HLS or through an intermediary HDL, and then vendor tools are used to generate a device specific bitstream. For DeCO, we use an in-house automated mapping flow to provide a rapid, vendor independent, mapping to the overlay. The mapping process comprises DFG extraction from HLL descriptions of compute kernels, graph balancing, DSP48E1 primitive aware node merging, architecture aware scheduling of nodes onto the overlay, and finally, configuration generation. Like software compilation, this mapping process is typically done offline, with the resulting configurations loaded onto the overlay in minimal time at runtime. The automated overlay mapping flow is described in detail below.

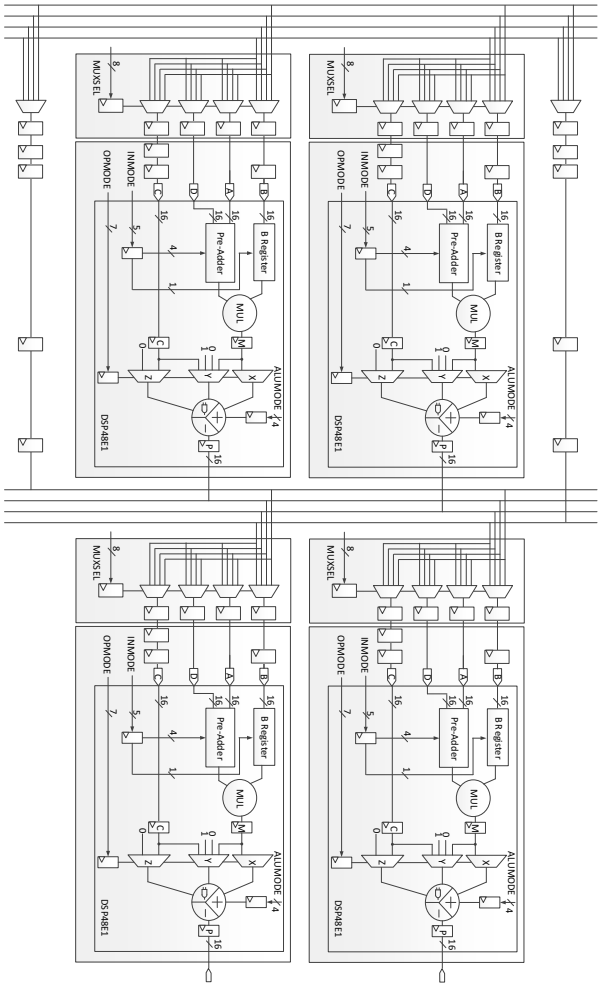

Fig. 6: Architectural design of the 16-bit cluster consisting of four functional units and two delay lines.

A. C to DFG Conversion

The HLL C description of the compute kernel is transformed to a DFG description. The DFG consists of nodes that represent operations and edges that represent the flow of data between operations. A node executes in a pipelined manner, and only produces an output when all of its inputs are ready, as per the dataflow model of computation.

B. DFG Transformations

First, the DFG description is parsed and translated into a balanced DFG, which both reduces the graph depth and better

shapes the DFG to match the cone. Then, in order to reduce the number of compute nodes, we merge multiple nodes based on the capabilities of the DSP48E1 primitive, as discussed previously. Lastly, architecture aware scheduling adjusts the schedule in a resource aware manner to better utilise the available resource in the overlay cone. This transformation process is shown in Fig. 2(a) to Fig. 2(c).

C. Configuration Generation

The FU compute nodes and the graph edges are used to determine the configuration data for the FUs and the connection network, respectively. This configuration data can then be used to set the programmable settings of the FU and the routing network, implementing the kernel.

V. Experimental Evaluation

The DeCO overlay is mapped to the Xilinx Zynq XC7Z020 using Xilinx ISE 14.6. We evaluate the performance of the DeCO architecture and mapping tool, using a benchmark set of compute kernels, which we then compare to other overlay implementations mapped to the Zynq device.

A. 16-Bit Overlay Comparison and Analysis

For comparison purposes, we map the benchmark set onto the proposed 16-bit overlay, and onto on a 5×5 modified DySER overlay [9] (Overlay-I) and a 5×5 DSP block based island-style overlay [10] (Overlay-II), both of which are 16-bit architectures. While all the overlays have comparable DSP usage (20, 25 and 25, respectively), the LUT and FF requirement at (1368, 33875 and 11023) and (2348, 13390 and 10486), respectively, show a significant reduction for DeCO, as shown in Fig. 7. This represents savings in LUT requirements of 96% and 87%, compared to [9] and [10], respectively.

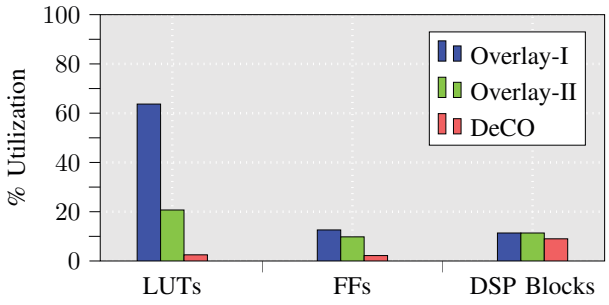

Fig. 7: Comparison of overlay resources required for implementing the benchmark set

As a second experiment, we compare the physical footprint in terms of configuration tiles, operating frequency and configuration time, of the proposed overlay with the two previous overlays and the minimum partial reconfiguration (PR) region which can house the RTL implementation (generated using Vivado HLS 2013.2) of the benchmark set. The Zynq fabric consists of 22 DSP tiles (each containing 10 DSP blocks) and 133 CLB tiles (each containing 50 CLBs), and reconfiguration using PR must be done in a multiple of these tiles to be fast [22]. The proposed overlay requires 2 DSP tiles and 6 CLB tiles. [9] requires 3 DSP tiles and 126 CLB tiles while [10] requires 3 DSP tiles and 42 CLB tiles. We also include a PR region which is big enough to accommodate the largest benchmark in the set (*kmeans*), which requires 1 DSP tile and 3 CLB tiles. These results are presented in Table II and show that the proposed overlay requires 2× CLB tiles compared to a PR-based direct FPGA implementation of the compute kernels, with [9] and [10] having a significantly higher requirement at 21× and 7×, respectively.

TABLE II: Experimental results for the comparison of different implementations

	Area (DSP/CLB)	Tiles (DSP/CLB)	Freq. (MHz)	Config. data (Bytes)	Config. time (us)	Peak GOPS
Overlay-I	25/6142	3/126	175	194.0	7.2	4.37
Overlay-II	25/2095	3/42	370	287.0	11.5	27.75
Proposed	20/258	2/6	395	53.5	2.0	23.70
PR region	10/150	1/3	249	49000.0	382.0	-

Perhaps more notable is the time required to change the kernel context for the various implementations. The PR region can be reconfigured entirely using 49 KBytes of configuration data in 382 us using the Zynq PCAP controller, while DeCO can be reconfigured entirely using just 53.5 Bytes of configuration data in 2 us, representing a 190× improvement in configuration time. The other overlays, while requiring a longer configuration time than DeCO, are also significantly better than PR. We calculate peak throughput of the overlays in GOPS, as the product of overlay frequency and the maximum number of arithmetic operations supported by the DSP blocks, as show in Table II.

B. Mapping Compute Kernels onto DeCO

Fig. 8 shows the mapping of the largest benchmark (*kmeans*) on to the proposed 16-bit overlay and onto [10]. The benchmark implementation on [10] results in an initial latency of 52 cycles while mapping the same benchmark on the proposed cone-shaped overlay results in an initial latency of just 24 cycles.

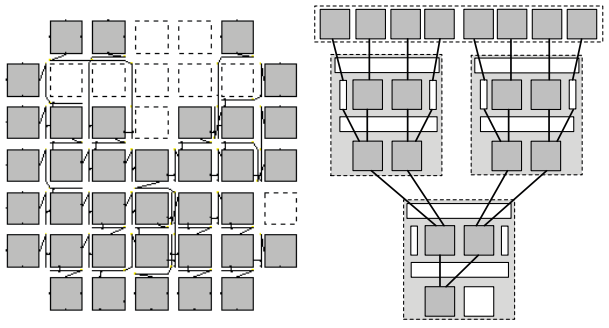

Fig. 8: Mapping of *kmeans* on Overlay-II vs. DeCO.

To analyze the mapping flexibility and utilization of the proposed overlay, we map additional kernels, taken from [10] and [6]. Recall that the overlay was originally designed for the kernels shown previously in Table I. Table III shows the required number of cones, the percentage utilization of the

FUs in a cone, and the achievable GOPS for each benchmark. This shows that the DeCO architecture is able to efficiently map kernels which are unknown at overlay design time.

TABLE III: Experimental results for mapping benchmarks

Benchmark	Required Cones	% Utilization	GOPS
fft	1	40%	3.95
kmeans	1	95%	9.08
mm	1	75%	5.92
spmv	1	70%	5.53
mri	1	75%	4.34
stencil	1	80%	5.53
gradient [6]	0.5	90%	4.34
chebyshev	0.5	40%	2.76
sgfilter	1	50%	7.11
mibench	1	40%	5.13
bicg	3	50%	11.85
trmm	4.5	60%	21.33
syrk	4.5	80%	28.44

Table III shows FU utilizations of up to 95%. For small kernels, with a utilization of less than 50%, such as *gradient* and *Chebyshev*, we are able to improve utilization by mapping replicated versions of the kernels. For example, the FU utilization for *gradient* on a single full cone is 45%, but by mapping two instances of this kernel onto the proposed cone, we can achieve an effective FU utilization of 90%. For some of the compute kernels, such as *FFT* and *Chebyshev*, the FU utilization is less because these kernels are not cone shaped DFGs. FFT is wide while Chebyshev is narrow. Using a single cone, we are able to achieve a throughput of up to 9.08 GOPS (38% of the peak throughput).

To map larger kernels, multiple instances of the overlay cone are used. The last three rows of Table III show the mapping of larger benchmarks to replicated instances of the cone, allowing us to achieve high GOPS. Cones are replicated in a multi-lane pattern to reflect the shape of larger graphs.

VI. Conclusion and Future Work

We have presented DeCO, a DSP block based cone-shaped FPGA accelerator overlay architecture that uses Xilinx DSP48E1 primitives as a programmable FU. DeCO is area-efficient, with low overhead interconnect and supports pipelined execution of compute kernels at high throughputs. We observe 96% and 87% savings in LUT requirements compared to other overlays from the literature. When implemented on a Xilinx Zynq, DeCO achieves a frequency of 395 MHz and when implemented on a Xilinx Virtex-7 device it achieved a frequency of 645 MHz, both of which are close to the DSP block theoretical limits on those respective architectures. In the future, we plan to explore embedding the proposed overlay within a heterogeneous platform where it can be used as a rapidly reconfigurable general purpose accelerator.

References

[1] K. Compton and S. Hauck, "Reconfigurable computing: a survey of systems and software," *ACM Computing Survey*, vol. 34, pp. 171–210, Jun. 2002.

[2] G. Stitt, "Are field-programmable gate arrays ready for the mainstream?" *IEEE Micro*, vol. 31(6), pp. 58–63, 2011.

[3] A. Canis, 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 International Symposium on Field Programmable Gate Arrays (FPGA)*, 2011.

[4] N. W. Bergmann, S. K. Shukla, and J. Becker, "QUKU: a dual-layer reconfigurable architecture," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 12, no. 1s, pp. 63:1–63:26, Mar. 2013.

[5] 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," *J. Signal Process. Syst.*, vol. 77(1–2), 2014.

[6] J. Cong, H. Huang, C. Ma, B. Xiao, and P. Zhou, "A fully pipelined and dynamically composable architecture of CGRA," in *IEEE Symposium on FPGAs for Custom Computing Machines (FCCM)*, 2014.

[7] J. Coole and G. Stitt, "Intermediate fabrics: Virtual architectures for circuit portability and fast placement and routing," in *Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, 2010, pp. 13–22.

[8] J. Benson, R. Cofell, C. Frericks, C.-H. Ho, V. Govindaraju, T. Nowatzki, and K. Sankaralingam, "Design, integration and implementation of the DySER hardware accelerator into OpenSPARC," in *International Symposium on High Performance Computer Architecture (HPCA)*, 2012.

[9] A. K. Jain, X. Li, S. A. Fahmy, and D. L. Maskell, "Adapting the DySER architecture with DSP blocks as an Overlay for the Xilinx Zynq," in *International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies (HEART)*, 2015.

[10] A. K. Jain, S. A. Fahmy, and D. L. Maskell, "Efficient Overlay architecture based on DSP blocks," in *IEEE Symposium on FPGAs for Custom Computing Machines (FCCM)*, 2015.

[11] A. K. Jain, D. L. Maskell, and S. A. Fahmy, "Throughput oriented FPGA overlays using DSP blocks," in *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*, 2016, pp. 1628–1633.

[12] S. Govindarajan and R. Vemuri, "Cone based clustering for list scheduling algorithms," in *Proceedings of the European Design and Test Conference*, 1997, pp. 456–462.

[13] M. Stojilović, D. Novo, L. Saranovac, P. Brisk, and P. Ienne, "Selective flexibility: Breaking the rigidity of datapath merging," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2012.

[14] B. Ronak and S. A. Fahmy, "Mapping for maximum performance on FPGA DSP blocks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 4, pp. 573–585, April 2016.

[15] D. Capalija and T. Abdelrahman, "Towards synthesis-free JIT compilation to commodity FPGAs," in *IEEE Symposium on FPGAs for Custom Computing Machines (FCCM)*, 2011.

[16] D. Capalija and T. S. Abdelrahman, "A high-performance overlay architecture for pipelined execution of data flow graphs," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, 2013.

[17] J. Coole and G. Stitt, "Fast, flexible high-level synthesis from OpenCL using reconfiguration contexts," *IEEE Micro*, vol. 34(1), 2014.

[18] A. Landy and G. Stitt, "A low-overhead interconnect architecture for virtual reconfigurable fabrics," in *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*, 2012, pp. 111–120.

[19] V. Govindaraju, C.-H. Ho, and K. Sankaralingam, "Dynamically specialized datapaths for energy efficient computing," in *International Symposium on High Performance Computer Architecture (HPCA)*, 2011.

[20] J. Coole and G. Stitt, "Adjustable-cost overlays for runtime compilation," in *IEEE Symposium on Field Programmable Custom Computing Machines (FCCM)*, 2015, pp. 21–24.

[21] 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 (TRETS)*, vol. 7, no. 3, pp. 19:1–19:23, 2014.

[22] K. Vipin and S. A. Fahmy, "Mapping adaptive hardware systems with partial reconfiguration using CoPR for Zynq," in *Proceedings of NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2015.