Throughput Oriented FPGA Overlays Using DSP Blocks

Abhishek Kumar Jain, Douglas L. Maskell
School of Computer Engineering
Nanyang Technological University, Singapore
Email: {abhishek013,asdouglas}@ntu.edu.sg

Suhaib A. Fahmy
School of Engineering
University of Warwick, Coventry, UK
Email: s.fahmy@warwick.ac.uk

***Abstract*—Design productivity is a major concern preventing the mainstream adoption of FPGAs. Overlay architectures have emerged as one possible solution to this challenge, offering fast compilation and software-like programmability. However, overlays typically suffer from area and performance overheads due to limited consideration for the underlying FPGA architecture. These overlays have often been of limited size, supporting only relatively small compute kernels. This paper examines the possibility of developing larger, more efficient, overlays using multiple DSP blocks and then maximising utilisation by mapping multiple instances of kernels simultaneously onto the overlay to exploit kernel level parallelism. We show a significant improvement in achievable overlay size and overlay utilisation, with a reduction of almost 70% in the overlay tile requirement compared to existing overlay architectures, an operating frequency in excess of 300 MHz, and kernel throughputs of almost 60 GOPS.**

I. Introduction and Related Work

FPGAs have been shown to be suitable for computational acceleration in a wide range of applications. However, they are typically restricted to niche applications with the difficulty of hardware design, long compilation times, and design productivity being major issues preventing the mainstream adoption of FPGA based accelerators in general purpose computing. High-level synthesis (HLS) is a promising techniques for addressing the design productivity problem. 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 design efficiency, 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 precompiled hard macros [2], partial reconfiguration, and overlay architectures.

Overlay architectures are an attractive solution for hardware acceleration of compute kernels on FPGAs because of their improved design productivity, by virtue of fast compilation, software-like programmability and run-time management, and high-level design abstraction [3], [4], [5], [6], [7], [8]. Other advantages include application portability across devices, better design reuse, and rapid reconfiguration that is orders of magnitude faster than partial reconfiguration on fine-grained FPGAs. However, when implemented on top of a fine grained FPGA, there is often a significant cost in terms of area and performance due to limited consideration for the underlying FPGA architecture. These overheads currently limit the use of overlays to relatively small compute kernels, preventing their realistic use in practical FPGA-based systems. Current research is attempting to reduce area overheads while improving performance, as we summarize next.

QUKU is a reconfigurable coarse grained overlay architecture [6] with a fixed configuration array of processing elements (PEs) interconnected via an application specific customized interconnect. QUKU achieved a frequency of 175 MHz on a Xilinx Virtex-4 LX25, with a 1000× reduction in configuration data sizes and configuration times compared to standard FPGA implementations.

The VDR overlay [7] is an array of heterogeneous PEs interconnected by programmable switches. A newer architecture optimized for high frequency and throughput [8] consisted of a 24×16 array with a nearest-neighbour-connected mesh of 214 routing cells and 170 heterogeneous functional units (FUs). The overlay consumes 75% of a Stratix IV 210K ALMs, with the routing network using 90% of resources. A frequency of 355 MHz and a peak throughput of 60 Giga-operations/s (GOPS) was reported.

The Intermediate Fabric (IF) [3], [9] overlay was proposed to support near-instantaneous placement and routing. It consists of 192 heterogeneous functional units with a 16-bit datapath. The IF in [9] exhibited a 700× improvement in compilation time compared to vendor tools at the cost of approximately 40% extra resources, but with a frequency of 125 MHz suffered from relatively low throughput. IF was also mapped to a Xilinx XC5VLX330 FPGA, along with a low overhead version of the interconnect [10]. The original 14×14 IF used 44% of the LUTs with a frequency of 131 MHz while the optimized IF used 24% of the LUTs with a frequency of 148 MHz.

The DySER [5] coarse grained overlay architecture is a heterogeneous array of 64 functional units interconnected with a circuit-switched mesh network. It was implemented on a Xilinx XC5VLX110T FPGA along with an OpenSPARC T1 processor [5]. However it was only possible to fit a 2×2 32-bit, a 4×4 8-bit, or an 8×8 2-bit DySER overlay on the device. An adapted version of a 6×6 16-bit DySER overlay was implemented on a Xilinx Zynq-7020 [11] by optimising the functional units around the Xilinx DSP blocks.

An overlay architecture with FUs based on Xilinx DSP blocks was proposed in [12]. The overlay combines multiple kernel operations in a single tile mapped to DSP blocks resulting in a significant reduction in the number of FUs required, and hence the routing overhead. A frequency of 370 MHz with throughputs better than a direct implementation using Xilinx Vivado HLS were reported.

In reviewing the literature, we find that many overlays are developed with little consideration for the underlying FPGA architecture. Previous work has demonstrated that the DSP-rich FPGA fabrics in modern devices can support general purpose processing at near to DSP block theoretical lim-

its [13], [14]. This has resulted in the development of overlays with improved frequency and throughput [8], [12], however, these overlays still suffer from significant resource overheads, specifically in the routing network. While techniques such as runtime LUT content manipulation [15] and interconnect multiplexing [16] can reduce routing network overheads, they become unsuitable as the overlay frequency approaches the theoretical limit of the FPGA fabric.

Another problem is that different applications require different sized overlays, with an overlay large enough to satisfy the requirements of larger kernels being heavily underutilized when a small kernel is executing. To avoid underutilization, other researchers have proposed using multiple instances of small overlays for smaller kernels and a large overlay for larger kernels, reconfiguring the FPGA fabric at runtime based on kernel requirements [17]. However, this approach negates a key advantage of overlay architectures, specifically rapid configuration to support fast switching of kernels. Reconfiguring the FPGA fabric takes orders of magnitude more time than a kernel context switch.

In this paper, we take the approach of building a single large overlay and mapping multiple instances of kernels, possibly replicated, to the overlay to achieve effective utilization of resources. We also improve the compute-to-interconnect resource usage ratio by using multiple DSP blocks inside the FU. Efficient resource utilisation, both in terms of the optimum number of DSP blocks that can be used in an FU and the mapping of multiple applications to the new architecture, requires investigation. As such, we firstly analyse the characteristics of a number of compute kernels from the literature to ascertain the suitability of mapping multiple versions to the new DSP-rich overlay. We then use a vendor-independent mapping to our overlay architecture and show how it significantly improves the scalability and performance of coarse-grained overlays while achieving better utilization of available resources.

II. Analysis of Compute Kernels

Most benchmarks used to analyse the performance of overlays are relatively small, limited by small overlay sizes. As FPGAs have increased in size, these benchmarks are no longer sufficient to fully test newer more efficient overlays. We have compiled a benchmark set (shown in Table I) containing a number of compute kernels from the literature [12], [18], [19], [20].

Table I shows the characteristics of the kernels after extracting the data flow graphs (DFGs), including the number of I/O nodes, graph edges, operation nodes, average parallelism, graph depth, and graph width. The graph depth is the critical path length of the graph, while the graph width is the maximum number of nodes that can execute concurrently, 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.

Mapping these kernels to DSP blocks allows us to reduce the number of functional units required by combining simple arithmetic operations into the more complex compound instructions supported by the DSP block. We perform this transformation on all of the kernels and re-analyse the benchmark characteristics for DSP-aware DFGs (in brackets in Table I). It is clear from the *op nodes* column that an overlay with at least 25 DSP blocks is needed to accommodate all benchmarks, down from 44 for single operation nodes ignoring DSP block capabilities.

While DSP aware mapping does reduce the number of FUs required, I/O requirements remain unchanged. Additionally, using a large overlay and mapping multiple instances of the smaller kernels to it impacts the availability of both compute and I/O resources. As the size of an island-style overlay increases, the number of I/O interfaces grows linearly while the number of compute tiles grow quadratically. Thus, an $N \times N$ overlay supports N^2 FUs, but as the I/O is determined by the overlay perimeter it is proportional to N (e.g. $4N$, $8N$, $12N$ depending on the number of I/O nodes per tile). The scalability curves for three different architectures with different numbers of I/O nodes per tile, assuming single DSP block based FU, are shown in Fig. 1. Here, an 8×8 $4N$ overlay is limited to 64 DSP nodes and 32 I/O nodes, while an 8×8 $8N$ overlay has 64 DSP nodes and 64 I/O nodes. Fig. 1 also shows plots of I/O vs DSP nodes required for multiple replicated instances of the compute kernels from Table I. It can be seen that the replicated kernels towards the top left are I/O bound and require more I/O nodes, as provided by the $8N$ and $12N$ architectures. However the kernels with points below the architecture curves are compute bound and can make use of the available FUs. For example, the replicated compute kernels towards the bottom right of Fig. 1 have a limited I/O requirement and can consume the majority of DSP blocks in a $4N$ architecture.

TABLE I: The Characteristics of the Benchmarks

No.	Benchmark Name	I/O nodes	DFG Characteristics (DSP-aware Characteristics) graph edges	op nodes	graph depth	average parallelism	graph width
1.	chebyshev	1/1	12(10)	7(5)	7(5)	1.00(1.00)	1(1)
2.	sgfilter	2/1	27(19)	18(10)	9(5)	2.00(2.00)	4(3)
3.	mibench	3/1	22(14)	13(6)	6(4)	2.16(1.50)	3(3)
4.	qspline	7/1	50(46)	26(22)	8(7)	3.25(3.14)	7(7)
5.	poly1	2/1	15(12)	9(6)	4(3)	2.25(2.00)	4(4)
6.	poly2	2/1	14(10)	9(6)	5(3)	1.80(2.00)	3(3)
7.	poly3	6/1	17(13)	11(7)	5(3)	2.20(2.30)	4(4)
8.	poly4	5/1	13(9)	6(3)	4(2)	1.50(1.50)	2(2)
9.	poly5	3/1	43(28)	27(14)	9(6)	3.00(2.30)	6(6)
10.	poly6	3/1	72(51)	44(25)	11(9)	4.00(2.77)	11(10)
11.	poly7	3/1	62(44)	39(21)	13(8)	3.00(2.62)	10(7)
12.	poly8	3/1	51(35)	32(17)	11(5)	2.90(3.40)	8(8)
13.	fft	6/4	24(22)	10(8)	3(3)	3.33(2.66)	4(4)
14.	kmeans	16/1	39(36)	23(20)	9(7)	2.55(2.85)	8(8)
15.	mm	16/1	31(24)	15(8)	8(8)	1.88(1.00)	8(1)
16.	mri	11/2	24(20)	11(7)	6(5)	1.83(1.40)	4(2)
17.	spmv	16/2	30(24)	14(8)	4(4)	3.50(2.00)	8(2)
18.	stencil	15/2	30(24)	14(8)	5(3)	2.80(2.66)	6(4)
19.	conv	24/8	40(32)	16(8)	2(1)	8.00(8.00)	8(8)
20.	radar	10/2	18(16)	8(6)	3(3)	2.66(2.00)	4(2)
21.	arf	26/2	58(50)	28(20)	8(8)	3.50(2.50)	8(4)
22.	fir2	17/1	47(32)	23(8)	9(8)	2.55(1.00)	8(1)
23.	hornerbezier	12/4	32(22)	14(8)	4(3)	3.50(2.66)	5(4)
24.	motionvector	25/4	52(40)	24(12)	4(3)	6.00(4.00)	12(4)
25.	atax	12/3	123(99)	60(36)	6(6)	12.00(7.20)	27(9)
26.	bicg	15/6	66(54)	30(18)	3(3)	10.00(6.00)	18(6)
27.	trmm	18/9	108(90)	54(36)	4(4)	13.50(9.00)	27(9)
28.	syrk	18/9	126(99)	72(45)	5(4)	14.40(11.25)	36(18)

To determine the impact of adding additional compute nodes into the FU we re-examine the scalability curves for the $4N$ architecture with an FU consisting of one, two and four DSP blocks, referred to as *4N-1D*, *4N-2D* and *4N-4D*, respectively. The resulting scalability curves, along with the I/O and DSP node requirements for replicated instances of the compute kernels are shown in Fig. 2. It can be seen that the *4N-4D* architecture is only suitable for a very small number of

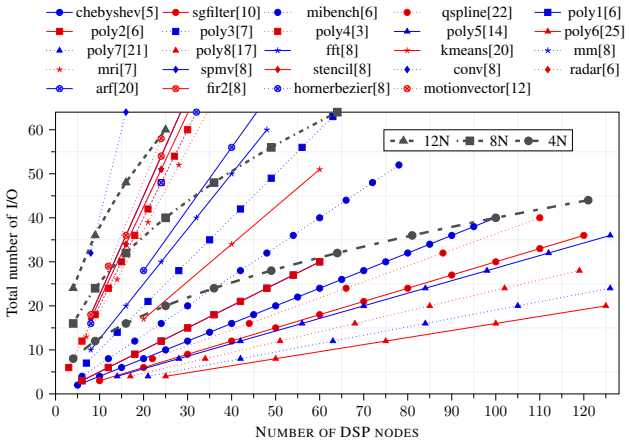


Fig. 1: I/O Scalability Analysis

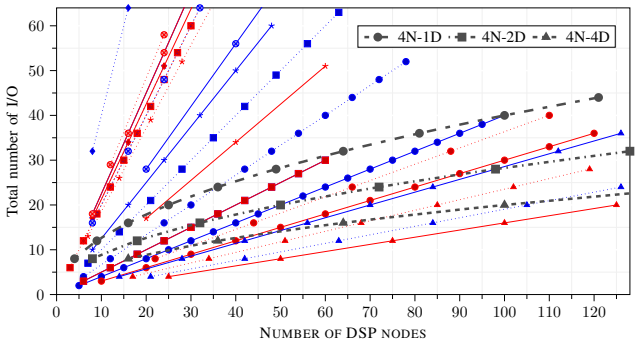


Fig. 2: DSP Scalability Analysis (4N Architecture)

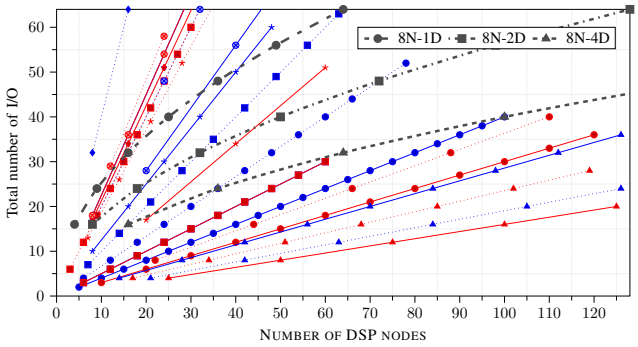


Fig. 3: DSP Scalability Analysis (8N Architecture)

kernels (those below the *4N-4D* curve), with a significant underutilization of DSP blocks for the other kernels, and is not considered further. Similarly, the scalability curves for the $8N$ architecture with an FU consisting of one, two, and four DSP blocks was also considered, referred to as *8N-1D*, *8N-2D* and *8N-4D* in Fig. 3, respectively.

The benefit of using a *4N-2D* over a *4N-1D* architecture, or an *8N-4D* over a *4N-1D* architecture, is the reduced cost of the routing network per DSP processing element. The FPGA resource cost of the *4N-2D* architecture is 100 Slices per DSP Block, compared to 160 Slices per DSP block for the *4N-1D* architecture. Thus, a 128 DSP block *4N-2D* overlay would consume 12.8K Slices while a 128 DSP block *4N-1D* overlay would consume 20.5K Slices. Due to the low cost of the *4N-2D* architecture, an overlay with 128 DSP blocks can easily fit onto a Xilinx Zynq device having 13K Slices. In the next section, we describe the detailed architecture of the dual-DSP block based overlay and its associated mapping tool flow.

III. Overlay Architecture and Mapping Tool

We now examine the use of a cluster of DSP48E1 primitives as a programmable FU in an efficient overlay architecture targeting data-parallel compute kernels. We use a conventional tile-based island-style overlay architecture, similar to those in [5] and [9], where a tile consists of an FU and programmable routing resource, consisting of one switch box (SB), two connection boxes (CB)) and horizontal and vertical routing tracks, all 16-bits wide to support a 16-bit datapath. The number of tracks in a channel is referred to as the channel width (CW), and as this increases, application routing becomes easier but with a higher area overhead. Multiplexer-based connection boxes and switch boxes connect tracks to the FU and other tracks in intersecting channels, respectively.

A. Functional Unit

The functional unit provides the compute resource for the overlay, and its design is critical to high performance. Using fully pipelined DSP blocks as the functional units allows us to achieve very high throughput. To improve the compute to routing ratio and hence reduce the number of slices per DSP block in the implementation of the overlay, we improve the FU in [12] by using two DSP48E1 blocks.

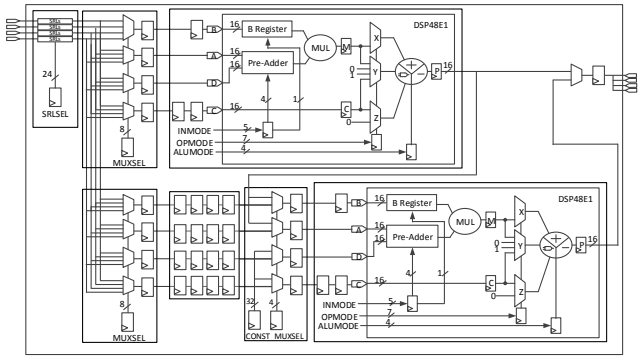


Fig. 4: Architecture of Functional Unit.

The FU, shown in Fig. 4, has a 4-input, 4-output structure, allowing it to connect to any of the four adjacent channels. To ensure signal timing across the array is correctly matched, pipeline latencies at the different FUs must be balanced by introducing a delay into each path. This is achieved by adding variable-length shift registers, implemented using LUT-based SRL32 primitives at each FU input. The maximum depth of the variable shift registers can be set to 16, 32, or 64 cycles, depending on the flexibility required. As long as the inputs at any node are not misaligned by more than the depth of the variable shift registers, the VPR [21] place and route algorithms can be used for placement and routing on the overlay. Multiplexer-based reordering logic is used to connect the delayed inputs of the FU to the DSP blocks. This is required as any of the four inputs of the FU can connect to any of the four inputs of a DSP48E1 primitive which unlike LUTs are not logically equivalent.

The two DSP blocks are connected in series, with four additional registers added to each input of the second DSP

block for pipeline balancing. Lastly, the output from either DSP block can be selected as the FU output. To maintain high F_{max}, all three pipeline stages in the DSP48E1 primitives are enabled. Additional registers are added at the output of each reordering multiplexer, at the input selector of the second DSP block and at the FU output. These registers, along with the registered outputs of the SRLs result in a total FU latency of 8 clock cycles when using only one DSP block and 13 clock cycles when using both DSP blocks.

B. Mapping tool

The main advantage of using an overlay is that application kernels can be mapped directly to it with software-like speed and programmability and do not rely on vendor tools to generate a device specific bitstream. We use an automated mapping flow to provide a rapid, vendor independent, mapping to our proposed overlay. The mapping process comprises DFG extraction from HLL descriptions of compute kernels, mapping and clustering the DFG nodes onto the dual-DSP FU, architecture-aware kernel replication, VPR compatible FU netlist generation, placement and routing of the FU netlist onto the overlay, latency balancing and finally, configuration generation. Like the typical software compilation process, this mapping process is done offline, and does not impact performance. The three steps distinct from [12], DFG to dual-DSP FU based clustering, architecture aware kernel replication and latency balancing, and configuration generation, are described below, with more detail in [12].

1) DFG to Dual-DSP FU based Clustering: Firstly, the DFG description is parsed and translated into a DSP-aware DFG. In order to reduce the number of compute nodes, we merge multiple nodes based on the capabilities of the DSP48E1 primitive. Next, in order to support the dual-DSP FU, we apply an additional clustering step. We cluster two consecutive nodes in the DSP-aware DFG if the fan-in of the resulting node, excluding constants which are instantiated inside the FU, is ≤ 4. Dual-DSP based clustering results in a significant reduction in the number of FUs required compared to an FU with just a single DSP block, also meaning less global routing resources are needed.

2) Architecture Aware Kernel Replication: If architecture aware kernel replication is enabled, the mapping tool instantiates multiple parallel instances of the compute kernels. This can be either as separate (e.g. different) kernel instances or as identical (e.g. replicated) instances. This optional step attempts to make better use of a large overlay based on the availability of I/O and compute resources.

3) Latency Balancing and Configuration Generation: The mapped kernel will only function correctly if all FU inputs arrive at the same execution cycle. This latency balancing is achieved by adding additional cycles to the SRL chains incorporated into the FUs to ensure alignment. To determine the latency imbalance at each node, we developed a tool to parse the VPR PAR output files and generate a routing resource graph. The routing resource graph is used to generate the overlay configuration (including the depth of each SRL) for the compute kernel. The tool also generates various kernel statistics including input to output latency, latency imbalance at each node, and maximum latency imbalance in the graph. The overlay configuration data is then used to set the programmable settings of the FU and the routing network.

IV. Experimental Evaluation

We synthesize and map the CW2-4N-2D overlay architecture (an overlay with CW=2, 1 I/O per row/column and an FU with 2 DSP blocks) using Vivado 2014.2 onto a Xilinx Zynq XC7Z020 and evaluate its performance and that of our custom mapping tool using a benchmark set of compute kernels.

A. Mapping to FPGA Fabric and Resource Usage

An $N \times N$ overlay would require N^2 FUs, $(N+1)^2$ SBs and $2 * N * (N+1)$ CBs. Table II shows the FPGA resources required for the FU, FU configuration registers (FUCR), SB, SB configuration registers (SBCR), CB and CB configuration registers (CBCR) for both the CW2-4N-2D and the CW4-8N-2D (in brackets) overlays. Note that there is no difference between the FU and FUCR for both overlays, the difference being restricted to just the routing tile. Next, we describe the individual overlay component mapping onto the physical FPGA fabric and their micro-architectural resource usage.

1) Resource Usage for the Functional Unit: As mentioned in Section III, the FU consists of programmable PEs, latency balancing logic and reordering logic. We use four 16-bit wide variable length shift registers, implemented as SLICEM shift register LUTs (SRLs) as the latency balancing logic, one on each of the four PE inputs. As we require a maximum delay of 64 clock cycles for our benchmark set we use two cascaded SRLs to form a chain and use 16 chains at each input of the PE to achieve a 16-bit variable delay of between 1 and 64 cycles. Thus each input consumes 32 LUTs and 16 FFs, resulting in 128 LUTs and 64 FFs for the complete latency balancing logic. The reordering logic requires 4 multiplexers with registered outputs, at the input of each PE, consuming 128 LUTs and 128 FFs in total. Additionally, we require three 16-bit registers at the DSP input ports (as shown in Fig. 4) for each PE, consuming 96 FFs. Delay lines at the four inputs of the second PE require 64 LUTs (SRLs) and 64 FFs and the second PE input selection logic requires 32 LUTs and 64 FFs. Finally at the output of the FU, we require a multiplexer with a registered output, consuming 8 LUTs and 16 FFs. Thus the total FU resource usage is 360 LUTs, 432 FFs and 2 DSP blocks. The FU configuration register includes 16 bits for each DSP block configuration, 16 bits for the two immediate operands, 8 bits for each reordering logic, 4 bits for the second PE input selection logic, 1 bit for the FU output selection logic and 24 bits for depth selection of the latency balancing logic. Hence, the FUCR consumes 109 FFs. The FU and FUCR resource utilization are given in Table II.

TABLE II: FPGA resource usage for overlay components

Resource	FU	FUCR	SB	SBCR	CB	CBCR
LUTs	360	0	64 (128)	0	48 (96)	0
FFs	432	109	0	8 (16)	64 (128)	6 (12)
DSPs	2	0	0	0	0	0

2) Resource Usage for Routing Resources: For CW=2, a SB requires four 16-bit 4×1 muxes, each consisting of 16 LUTs. The SB configuration register needs 8 bits, 2 for each of the 4 muxes. Hence the SBCR consumes 8 FFs. The total SB and SBCR resource usage, for CW=2, is shown in Table II. A CB consists of two 2×1 and two 4×1 muxes (each 16-bits wide). As each mux is registered, the total resource usage is 48 LUTs and 64 FFs. The CB configuration register requires 6 bits for the selection inputs of the 4 muxes and hence the SBCR consumes 6 FFs. Total CB and CBCR resource usage, for CW=2, is shown in Table II.

3) Total Resource Usage and Performance Analysis: The CW2-4N-2D overlay tile contains 1 FU, 1 SB, 2CBs and their configuration registers, while a border tile contains 1 SB, 1 CB and the configuration registers. Thus, an overlay tile consumes

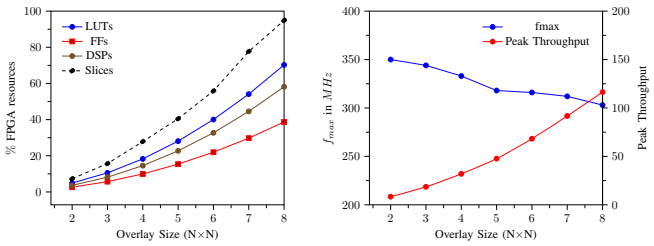
520 LUTs, 625 FFs and 2 DSP blocks while a border tile consumes 112 LUTs and 76 FFs. The post-place and route resource consumption on Zynq, as a percentage of total FPGA resources, is shown in Fig. 5(a).

The overlay operating frequency approaches the DSP theoretical limit of 400 MHz on Zynq for small overlays, but as the overlay grows in size the frequency decreases slightly, as shown in Fig. 5(b). Since the DSP48E1 can support three operations, an $N \times N$ overlay can support a maximum of $6 \times N^2$ operations, and hence the peak throughput is $6 * N^2 \times F_{max}$ operations per second, as shown in Fig. 5(b) for different overlay sizes.

The CW2-4N-2D overlay requires 109 configuration bits for the dual-DSP FU and 20 configuration bits for programming the routing network tile. Thus, an 8×8 overlay has a configuration size of 9100 bits (1137 Bytes), and can be configured entirely in 45.5 us, compared to 31.6 ms for the entire Zynq programmable fabric using the PCAP port, a 1000× improvement in reconfiguration time.

We also mapped the overlay to a mid-sized Virtex-7 (XC7VX690T-2) device where we were able to implement a 20×20 CW=2 overlay, resulting in a frequency of 380 MHz and a peak throughput of 912 GOPS. A quantitative comparison of the proposed overlay architecture with some existing overlays from the research literature is given in Table III.

For the different FPGA devices and overlay sizes we compare the resource usage in terms of the LUTs used and the percentage of total LUTs, frequency, maximum number of simultaneous arithmetic operations (Max OPs), peak throughput of the arithmetic operations in GOPS and the programmability cost in terms of LUTs/OP. These results show that we are competitive on almost all metrics. Programmability cost can be reduced further using alternative interconnect architectures such as hierarchical and nearest-neighbor interconnect.



(a) % resource usage (b) F_{max} and Peak Throughput

Fig. 5: Zynq-7020 CW2-4N-2D overlay scalability results

TABLE III: Quantitative Comparison of Overlays

Resource	IF [10]	IF (opt) [10]	[11]	[12]	Proposed	Proposed
Device	XC5VLX330	XC5VLX330	XC7Z020	XC7Z020	XC7Z020	XC7VX690T
Slices|LUTs	51.8K|207K	51.8K|207K	13.3K|53K	13.3K|53K	13.3K|53K	108.3K|433.2K
Overlay	14×14	14×14	6×6	8×8	8×8	20×20
LUTs used	91K(44%)	50K(24%)	48K(90%)	28K(52%)	37K(70%)	228K(52%)
Fmax (MHz)	131	148	175	338	300	380
Max OPs	196	196	36	192	384	2400
GOPS	25.6	29	6.3	65	115	912
LUTs/OP	465	255	1333	146	96	95

B. Mapping of Kernels onto the Overlay

Our mapping tool takes a C description of the compute kernel and maps it onto the overlay using the steps described in Section III-B. The number of overlay tiles needed for each of the benchmarks, for DFGs, DSP-aware DFGs and clustered DFGs, is shown in Fig. 6. (The numbers on x-axis relate to the benchmark number from Table I.) We observe a reduction in the number of tiles required for DSP-aware DFGs and clustered DFGs of up to 50% and 69%, respectively. The advantage of DSP-aware clustered DFGs becomes apparent by examining specific benchmarks, such as *poly7* (benchmark 11), where only a single DFG instance can fit onto an 8×8 overlay (as a minimum it requires a 7×7 overlay). However, with DSP-aware clustered DFGs, 4 separate instances of the *poly7* benchmark are able to fit onto an 8×8 overlay, utilising 56 of the 64 tiles.

Next we replicate multiple instances of the benchmarks from Table I and map them to our proposed CW2-4N-2D overlay. The x-axis of Fig. 7 indicates the benchmark number, followed by the number of replicated instances in brackets and shows that many applications are able to map multiple instances to the overlay. There are 4 benchmarks (benchmarks 25-28) which are unable to map to the CW2-4N-2D overlay due to I/O and internal routing requirements, instead requiring a CW4-8N-2D overlay. However, due to space constraints, we will not discuss that architecture here though we have successfully created it and mapped to it. Additionally, there are a number of benchmarks (benchmark 14-15, 17-19, 21-22, and 24) that are unable to map more than 1 instance due to the I/O limitations of the CW2-4N-2D overlay, as indicated in Fig. 2.

The actual throughput, in GOPS, for the replicated benchmark instances is shown by the left bar in Fig. 7, calculated as the product of the DFG compute nodes and the implementation operating frequency. For example, an overlay throughput of 57.6 GOPS is achieved by instantiating 6 instances of the *poly8 (12)* benchmark. This is 50% of the absolute peak performance, of 115 GOPS, which could be hypothetically achieved by a synthetic kernel having 384 operations (128 Add/sub, 128 MUL, 128 ALU ops) which would fully utilise the DSP block resources of the 64 FUs in our 8×8 overlay. It is clear that the benchmarks with modest I/O requirements benefit from replication, while those with larger I/O requirements would benefit from the CW4-8N-2D overlay.

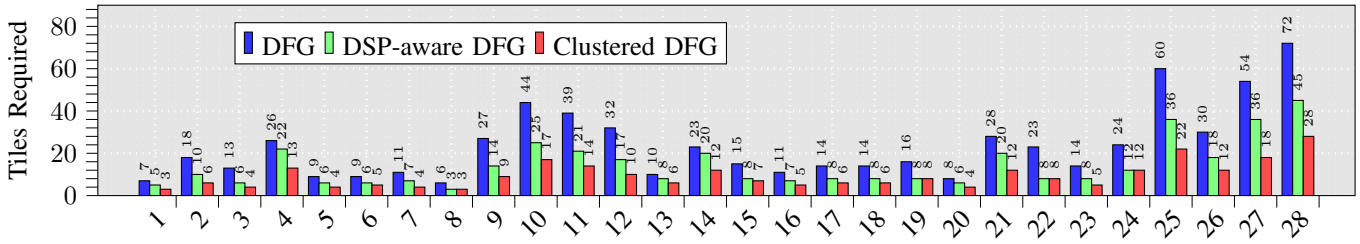


Fig. 6: The number of tiles required for the kernels in table I

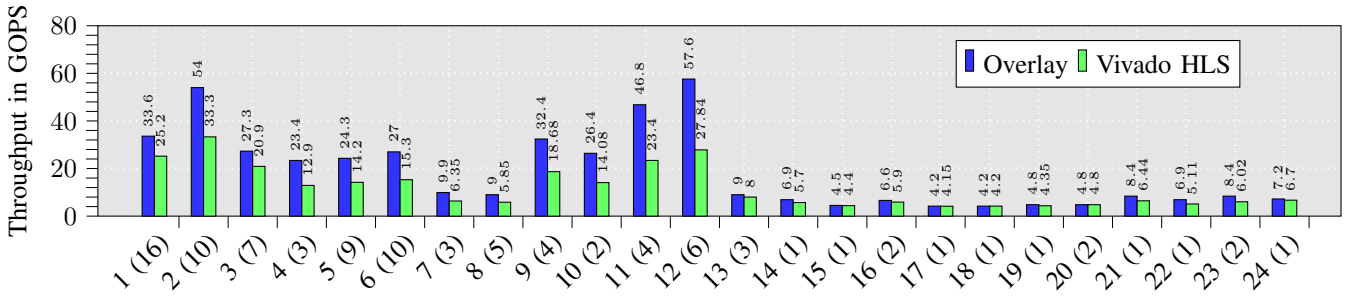


Fig. 7: The performance comparisons of the CW2-4N-2D overlay and Vivado HLS implementations

To further demonstrate the performance of our proposed overlay, we generate RTL implementations of the same kernel instances replicated an identical number of times using Vivado HLS 2014.2. In this way, we are able to perform a quantitative comparison of performance between the two implementations. Fig. 7 shows the performance comparison of the overlay implementations (left bar) and Vivado HLS implementations (right bar) in terms of throughput. Our overlay is able to achieve an average throughput improvement of 40% due to the highly pipelined architecture, something which Vivado HLS is currently unable to exploit. The Vivado HLS implementations of the replicated benchmarks require significantly less hardware resource (on average, our overlay requires 30× and 70× more slices, for benchmarks 1-12 and 13-24, respectively). However, this hardware penalty is the result of a general overlay architecture that can be effortlessly integrated into a virtualised hardware/software environment on the Zynq FPGA like the one in [4] that would incorporate both static and PR accelerators as well as overlays for generality and performance. The key advantage of an overlay is the fast compilation, software-like programmability and run-time management, with a relatively small configuration data size and fast non-preemptive hardware context switching, all of which are missing in a static Vivado HLS accelerator design. As indicated in Section IV-A, our proposed overlay is able to perform a hardware context switch in just 45.5 us (1000× faster than reconfiguring the Zynq FPGA) using just 1137 Bytes of configuration data.

V. Conclusion

We have presented an FPGA overlay architecture that uses multiple instances of the Xilinx DSP48E1 primitive as a programmable FU, resulting in an efficient overlay architecture for pipelined execution of compute kernels, with better resource utilization and significantly improved performance metrics. We demonstrate the efficiency of our overlay architecture by mapping a benchmark set of compute kernels using our automated mapping tool flow. We show that we are able to map multiple instances of the benchmark kernels to the overlay automatically, resulting in more efficient utilization of overlay resources, without resorting to reconfiguring the FPGA fabric at runtime. Our experimental evaluation shows that the overlay delivers a throughput of up to 57.6 GOPS (50% of the peak theoretical throughput of the overlay) and provides an average throughput improvement of 40% over Vivado HLS for the same implementations of our benchmark set. The overlay allows for fast non-preemptive hardware context switching 1000× faster than reconfiguring the FPGA, using just 1137 Bytes of configuration data. In future, we plan to explore alternative architectures for the routing network in order to further reduce the programmability cost.

References

[1] G. Stitt, "Are field-programmable gate arrays ready for the mainstream?" *IEEE Micro*, vol. 31(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 *FCCM*, 2011.

[3] J. Coole and G. Stitt, "Intermediate fabrics: Virtual architectures for circuit portability and fast placement and routing," in *CODES+ISSS*, 2010, pp. 13–22.

[4] 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.

[5] 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 *HPCA*, 2012.

[6] N. W. Bergmann, S. K. Shukla, and J. Becker, "QUKU: a dual-layer reconfigurable architecture," *ACM TECS*, vol. 12(1s), 2013.

[7] D. Capalija and T. Abdelrahman, "Towards synthesis-free JIT compilation to commodity FPGAs," in *FCCM*, 2011.

[8] D. Capalija and T. S. Abdelrahman, "A high-performance overlay architecture for pipelined execution of data flow graphs," in *FPL*, 2013.

[9] G. Stitt and J. Coole, "Intermediate fabrics: Virtual architectures for near-instant FPGA compilation," *IEEE ESL*, vol. 3(3), pp. 81–84, 2011.

[10] A. Landy and G. Stitt, "A low-overhead interconnect architecture for virtual reconfigurable fabrics," in *CASES*, 2012.

[11] 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 *HEART*, 2015.

[12] A. K. Jain, S. A. Fahmy, and D. L. Maskell, "Efficient Overlay architecture based on DSP blocks," in *FCCM*, 2015.

[13] H. Y. Cheah, S. A. Fahmy, and D. L. Maskell, "iDEA: A DSP block based FPGA soft processor," in *FPT*, 2012.

[14] B. Ronak and S. A. Fahmy, "Efficient mapping of mathematical expressions into DSP blocks," in *FPL*, 2014.

[15] C. H. Hoo and A. Kumar, "An area-efficient partially reconfigurable crossbar switch with low reconfiguration delay," in *FPL*, 2012.

[16] N. Kapre, N. Mehta, M. deLorimier, R. Rubin, H. Barnor, M. Wilson, M. Wrighton, and A. DeHon, "Packet switched vs. time multiplexed FPGA overlay networks," in *FCCM*, 2006.

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

[18] S. Gopalakrishnan, P. Kalla, M. B. Meredith, and F. Enescu, "Finding linear building-blocks for RTL synthesis of polynomial datapaths with fixed-size bit-vectors," in *ICCAD*, 2007.

[19] D. Bini and B. Mourrain, "Polynomial test suite, 1996," *See http://www-sop. inria. fr/saga/POL*.

[20] C.-H. Ho, V. Govindaraju, T. Nowatzki, Z. Marzec, P. Agarwal, C. Frericks, R. Cofell, J. Benson, and K. Sankaralingam, "Performance evaluation of a DySER FPGA prototype system spanning the compiler, microarchitecture, and hardware implementation," in *ISPASS*, 2015.

[21] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," in *FPL*, 1997, pp. 213–222.