

# Adapting the DySER Architecture with DSP Blocks as an Overlay for the Xilinx Zynq

Abhishek Kumar Jain, Xiangwei Li, Suhaib A. Fahmy, Douglas L. Maskell  
School of Computer Engineering, Nanyang Technological University, Singapore  
{abhishek013, xli045, sfahmy, asdouglas}@ntu.edu.sg

## ABSTRACT

Coarse-grained overlay architectures have been shown to be effective when paired with general purpose processors, offering software-like programmability, fast compilation, and improved design productivity. These architectures enable general purpose hardware accelerators, allowing hardware design at a higher level of abstraction, but at the cost of area and performance overheads. This paper examines the DySER overlay architecture as a hardware accelerator paired with a general purpose processor in a hybrid FPGA such as the Xilinx Zynq. We evaluate the DySER architecture mapped on the Xilinx Zynq and show that it suffers from a significant area and performance overhead. We then propose an improved functional unit architecture using the flexibility of the DSP48E1 primitive which results in a 2.5 times frequency improvement and 25% area reduction compared to the original functional unit architecture. We demonstrate that this improvement results in the routing architecture becoming the bottleneck in performance.

## 1. INTRODUCTION

Emerging reconfigurable platforms tightly couple capable processors with high performance reconfigurable fabrics [1]. This promises to move the focus of reconfigurable computing systems from static accelerators to a more software oriented view, where reconfiguration is a key enabler for exploiting available hardware resources. This requires a revised look at how to use reconfigurable hardware within a software-centric processor-based system. Recently, coarse grained overlay architectures have been shown to be effective when paired with general purpose processors [2, 3] as this allows the hardware fabric to be viewed as a software-managed hardware task, enabling more shared use. Overlay architectures consist of a regular arrangement of coarse grained routing and compute resources. The key attraction of overlay architectures is software-like programmability through mapping from high-level descriptions, application portability across devices, design reuse, fast compilation by avoiding the complex FPGA implementation flow, and hence, improved design productivity. Another main advantage is rapid reconfiguration since the overlay architectures have smaller configuration data size due to the coarse granularity. Reconfiguration time in a dynamically reconfigurable system has an impact on overall application performance as shown in [4]. Although research

This work was presented in part at the international symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2015) Boston, MA, USA, June 1-2, 2015.

in the area of overlay architectures has increased over the last decade, the field is still in its infancy with only relatively few overlay architectures demonstrated in prototype form [3, 5, 6]. One such example is the DySER architecture targeted to the Xilinx Virtex-5 FPGA [7]. Area and performance overheads have, however, prevented the realistic use of DySER in practical FPGA-based systems. One of the reasons for this poor performance is that overlays are typically designed without serious consideration of the underlying FPGA architecture.

Embedded hard macros, such as DSP blocks, have been added to FPGAs in recent years. By building often used functions into optimised compact primitives, area, performance, and power advantages are achieved over equivalent “soft” implementations in the logic fabric. Many existing overlay architectures [3, 5, 6, 7] do not specifically use these macros, except insofar as they are inferred by the synthesis tools. However, it is well known that inference of hard macros by synthesis tools does not result in optimal throughput [8]. Xilinx DSP blocks have recently been shown to enable high speed soft processors by taking advantage of flexible control signals to implement alternative functions on the same primitive at different times [9].

The novel contribution of this paper is the efficient and practical implementation of the DySER architecture on the Xilinx Zynq. This DySER overlay can then be used to host accelerators to offload data-parallel compute kernels from compute-intensive applications running on the ARM processor. We demonstrate how adopting the Xilinx DSP48E1 primitive in the functional unit (FU) of the DySER architecture improves both performance and area. The main contributions can be summarized as follows:

- RTL implementation of a functional unit (compatible with the DySER architecture) using the DSP48E1 primitive, which can operate at near theoretical maximum frequency.
- A quantitative analysis of area overheads of the modified DySER architecture by mapping a set of benchmarks to DySER and to the FPGA fabric using Vivado HLS.

The remainder of this paper is organized as follows: Section 2 presents the state of the art in high performance overlay architectures as hardware accelerators. In Section 3, we describe the DySER architecture, functional unit and implementation on the Xilinx Zynq. Section 4 presents the modified functional unit architecture using the flexible DSP48E1 primitive. We then present a quantitative analysis of area overheads in Section 5 and conclude the paper in Section 6.

## 2. RELATED WORK

Overlay architectures have been proposed as a technique for reducing the prohibitive compilation time required to map an application to the conventional fine-grained FPGA fabric. They can be broadly classified into two categories. In the first, the virtual logic and routing of the overlay are unchanged while a compute kernel is executing [3, 5, 6], and in the second, the virtual logic and routing of the overlay change on a cycle by cycle basis while a compute kernel is executing [10, 11]. In this work we only consider overlay architectures from the first category.

QUKU [3] 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. A  $4 \times 4$  reconfigurable homogeneous array of PEs required 40% extra resources compared to point-to-point connectivity with four immediate neighbouring PEs. Another overlay architecture, referred to as an intermediate fabric (IF) [5], was proposed to support near-instantaneous placement and routing. The IF in [5] was implemented on an Altera Stratix III FPGA in order to evaluate area and performance. It enabled a  $700 \times$  improvement in compilation time compared to vendor tools at the cost of approximately 40% extra resources on the FPGA. It consists of 192 heterogeneous functional units comprising 64 multipliers, 64 subtractors, 63 adders, one square root unit, and five delay elements with a 16-bit datapath and supports fully parallel, pipelined implementation of compute kernels. A high throughput (60 GOPS) overlay [6] was implemented on Altera Stratix IV FPGA at the cost of 67% area overhead. This  $24 \times 16$  overlay is a nearest-neighbour-connected mesh of 214 routing cells and 170 heterogeneous functional units (FUs) comprising 51 multipliers, 103 adders and 16 shift units. The major problem with all of these architectures is that they consume significant resources to offer this programmability and ease of compilation.

DySER [12, 13] was proposed to improve the performance of general purpose processors by integrating dynamically specialized execution resources into the processor pipeline. The concept of DySER is very similar to the coarse grained overlay architectures of [3, 5, 6]. It also exhibits similarities with conventional tiled architectures such as RAW [14], WaveScalar [15] and TRIPS [16]. DySER was originally designed as a heterogeneous array of 64 functional units interconnected with a circuit-switched mesh network and implemented as an ASIC.

The DySER architecture was improved and then prototyped, along with the OpenSPARC T1 RTL, on a Xilinx XC5VLX110T FPGA [7]. DySER had a critical timing path of 12.7 ns, compared to the 10.1 ns critical timing path of OpenSPARC. However, due to excessive LUT consumption, it was only possible to fit a  $2 \times 2$  32-bit DySER, a  $4 \times 4$  8-bit DySER or an  $8 \times 8$  2-bit DySER on the FPGA. The  $2 \times 2$  32-bit DySER (supporting just 4 operations) is of limited value in performance evaluation, and instead a  $4 \times 4$  DySER (supporting up-to 16 operations) or an  $8 \times 8$  DySER (supporting up-to 64 operations) is required to provide meaningful performance comparisons.

The DySER architecture, although relatively efficient from an application mapping perspective, suffered because it was implemented without much consideration for the underlying FPGA architecture. Considering the presence of hard macro blocks, and previous work that has demonstrated how these

can be used for general processing at near to their theoretical limits [9, 17], we propose enhancing DySER by using the DSP48E1 found in all modern Xilinx FPGAs to take on most functions of the FU.

## 3. THE DYSER ARCHITECTURE

The DySER architecture consists of two blocks, the tile fabric and the edge fabric, where each tile in the tile fabric instantiates a switch and a functional unit (FU), while the edge fabric only instantiates a switch, forming the boundary at the top and left of the tile fabric. The resulting architecture contains I/O ports around the periphery of the fabric, which are connected to FIFOs. A simple  $2 \times 2$  DySER overlay, consists of four tile instances and five switch instances along the North and West boundaries, resulting in 4 FUs and 9 switches, as shown in Fig. 1. Extrapolating this to an  $N \times N$  DySER architecture results in  $N^2$  FUs and  $(N + 1)^2$  switches.

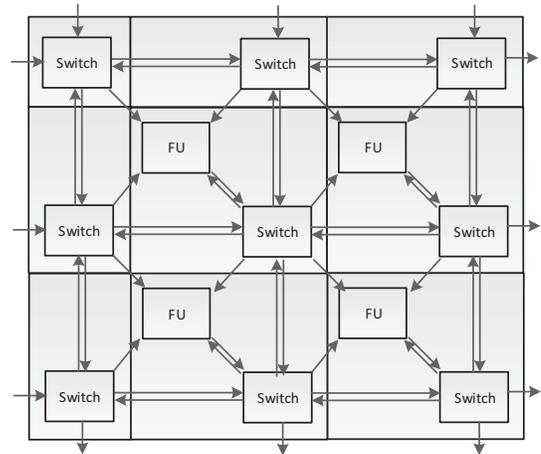


Figure 1: Architecture of a  $2 \times 2$  DySER.

### 3.1 DySER Switch

The switches allow datapaths to be dynamically specialized. They form a circuit-switched network that creates paths from inputs to the functional units, between functional units, and from functional units to outputs. Switches in DySER have 5 inputs (4 from neighbour switches and 1 from the functional unit at the North-West direction) and 8 outputs (to all 8 directions). Hence, switches require a 5:1 multiplexer and a state machine for synchronization at each output.

### 3.2 DySER Functional Unit

The functional unit (FU) provides resources for the mathematical and logical operations, and synchronization logic. It receives its input values from the four neighbouring switches and outputs its result to the switch in the south-east direction. The FU consists of programmable computation logic and a state machine as synchronization logic at each input and output of the computation logic. The state machine implements a credit-based flow-control protocol to enable receiving of inputs asynchronously at arbitrary times from the FIFO interfaces.

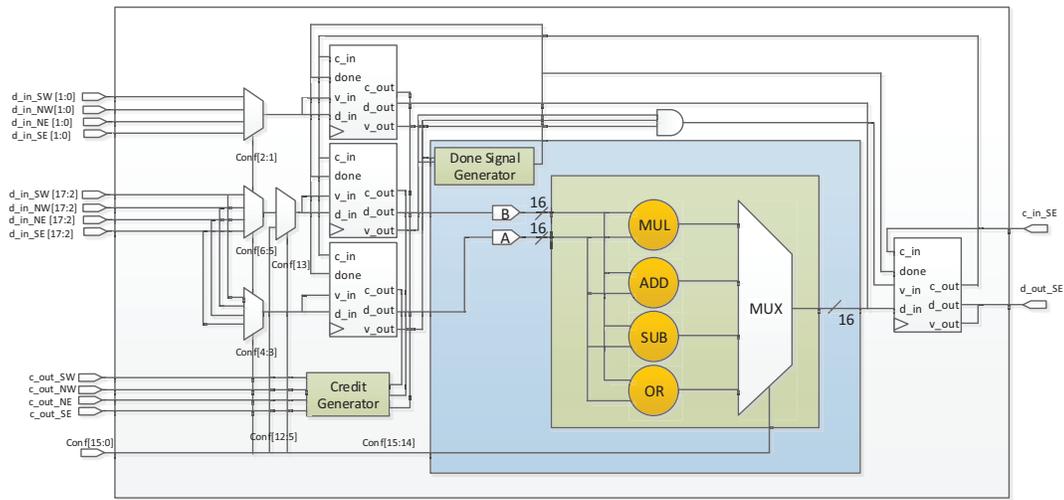


Figure 2: Functional unit architecture.

The operators in the FU can be selected according to application requirements. We choose four operators: Add, Sub, Mul and OR in the FU, as shown in Fig. 2, to map the benchmarks from [18]. The benchmark characteristics are shown in Table 1. Benchmarks, where a small code region dominates the runtime, and where computation can easily be scheduled, are taken from [18]. These benchmarks mimic the workloads of the PARBOIL suite.

Table 1: Benchmark Characteristics

No.	Benchmark	Add	Sub	Mul	OR	Total
1.	fft	3	3	4		10
2.	kmeans	7	8	8		23
3.	mm	7		8		15
4.	mri-q	3		6	1	10
5.	spmv	6		8		14
6.	stencil	10	2	2		14
7.	conv	8		8		16
8.	radar	6		2		8

The original DySER FU was implemented using Xilinx ISE 14.6 targeting a Xilinx Zynq XC7Z020. The FU consumes 49 Slices (148 LUTs, 66 FFs) and 1 DSP48E1 block, with a critical path of 6.7 ns. Hence the maximum operating frequency of the FU is 150 MHz. Fig. 3 shows the physical mapping of the FU to the FPGA fabric. While synthesizing, the tool infers a DSP block for multiplication. The remainder of the operations and the multiplexer in the compute logic are mapped to 17 Slices (57 LUTs). State machines and input selection multiplexers are mapped to 32 Slices (91 LUTs and 66 FFs). After integrating the FU into the DySER tile and implementing it on the FPGA fabric, we found that the critical path in the DySER Tile is the same as the critical path of the FU (6.7 ns), and hence the FU limits the performance of the DySER tile.

#### 4. DSP BLOCK BASED DYSER

Building on the advantages of hard DSP macros for implementing high speed processing elements, we examine the use of the Xilinx DSP48E1 primitive as a programmable FU

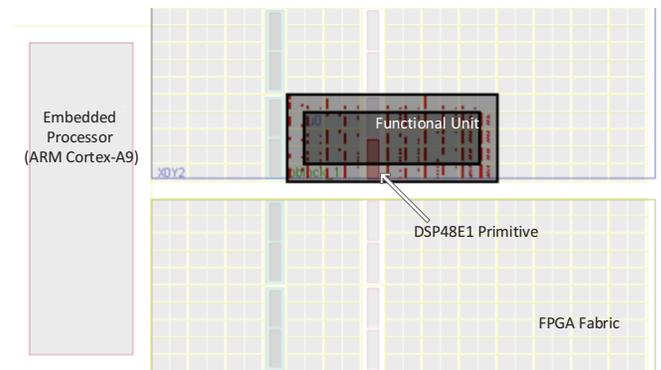


Figure 3: Physical mapping of functional unit on FPGA.

in DySER targeting data-parallel compute kernels. Despite the fact that the original FU uses a DSP block for multiplication, it does not fully exploit the performance advantage of the DSP block. Since the DSP48E1 can be dynamically configured and used for operations required by the FU, we show that an area and performance efficient FU can be built by making use of DSP block as an ALU, instead of just as a multiplier, and enabling the internal pipeline registers of the DSP block.

#### 4.1 DSP48E1 Based Functional Unit

We use the DSP48E1 primitive, as shown in Fig. 4, to implement computation logic in the modified functional unit. The DSP48E1 primitive has a pre-adder, a multiplier, an ALU, four input ports for data, and one output port P, as shown in Fig. 4, and can be configured to support various operations such as multiply, add, sub, bitwise OR, etc. These functions are determined by a set of dynamic control inputs that are wired to configuration registers. The DSP48E1 primitive is directly instantiated providing total control of the configuration of the primitive. This allows us to maximize the compute kernel throughput and achieve a high FU frequency by operating the DSP48E1 at its maximum frequency.

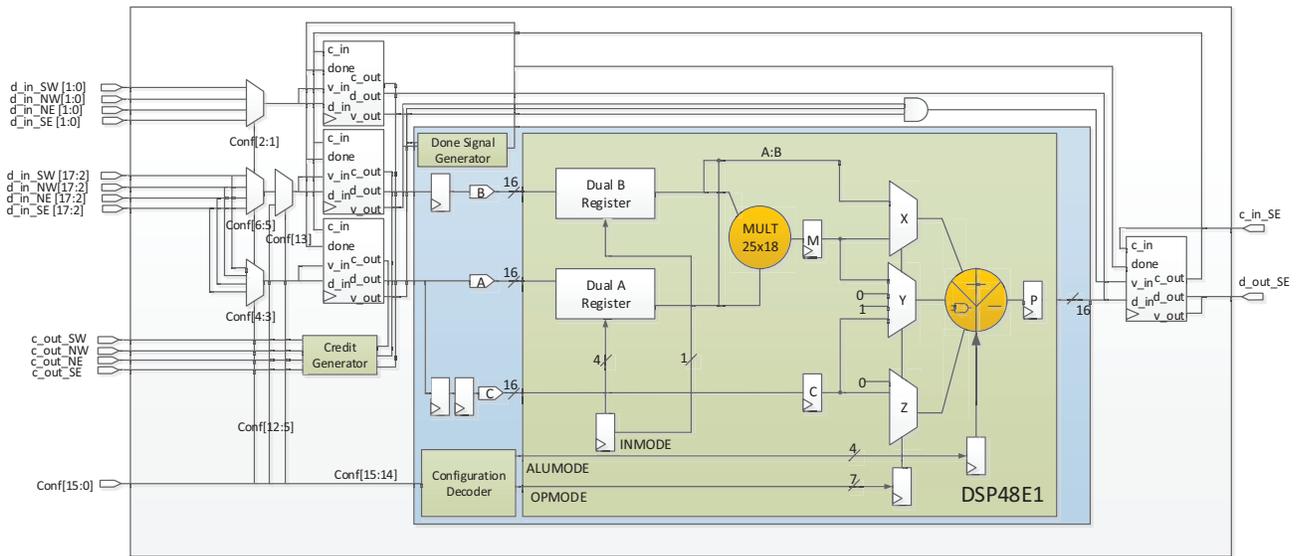


Figure 4: DSP48E1 based functional unit architecture.

We enable all of the pipeline stages of the DSP48E1 primitive. The redesign of the DySER functional unit replaces the original compute unit (CU), shown in Fig. 2, with the fully pipelined DSP48E1 primitive, along with modifications to the done signal generation logic and configuration decoding logic, as shown in Fig. 4. The two inputs from the FU (to the CU) are connected to the three ports of the DSP48E1 primitive, as shown in Fig. 4. The FU configuration register includes 2 bits for operation selection with the other 14 bits for constant and input multiplexers. Additionally, we require three 16-bit registers at the DSP input ports (as shown in Fig. 2), consuming 48 FFs to balance the internal pipeline stages of the DSP block. Table 2 shows the DSP48E1 configuration settings required for each operation. Inmode remains same for all of the operations and hence we hard-code it to 00000.

Table 2: DSP48E1 configuration for each operation

Operation	ALUMODE	OPMODE	INMODE
ADD	0000	011 0011	00000
SUB	0011	011 0011	00000
MUL	0000	000 0101	00000
OR	1100	011 1011	00000

## 4.2 Analysis of Performance Improvement

We analyze the performance improvement of the FU in terms of frequency and resource usage. The DSP48E1 based FU consumes 37 Slices (116 LUTs, 117 FFs) (25% less than the original FU) and 1 DSP block. Apart from obvious area savings, the strategy of using a fully pipelined DSP block as the computational part of the FU also improves overall timing performance. The FU has a critical path of just 2.7 ns, resulting in a maximum frequency of 370 MHz, which is 2.5× that of the original FU. Fig. 5 shows the physical mapping of functional unit onto the FPGA fabric.

Since a hard primitive is used for the implementation of CU operations, only minimal additional circuitry is implemented in the logic fabric which consists of configuration de-

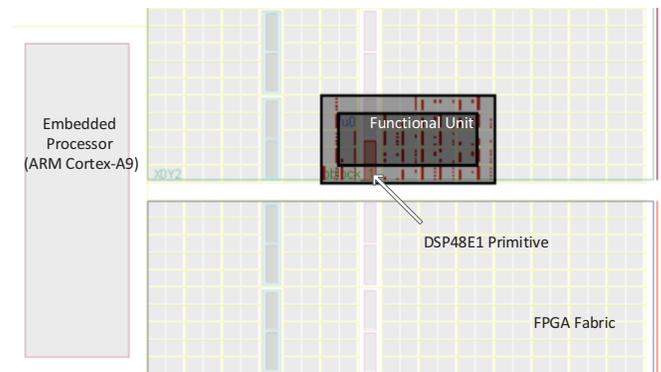


Figure 5: Physical mapping of enhanced functional unit.

coding logic, three 16-bit balancing registers and done signal generation logic. All of this additional circuitry is mapped to 10 Slices (25 LUTs and 51 FFs). State machines and input selection multiplexers are mapped to 27 Slices (91 LUTs and 66 FFs).

By integrating the enhanced FU into the DySER tile and implementing it on the FPGA fabric, we found that the critical path of the switch, which is 5.3 ns, now limits the performance of the DySER tile. Fig. 6 shows the physical mapping of the DySER Tile to the FPGA fabric. It is clear that the major area overhead in DySER is due to significant resources consumed in the switch implementation. The switch consumes 251 Slices (995 LUTs and 325 FFs) and hence the whole tile consumes 288 Slices (1118 LUTs and 447 FFs). The largest source of area overhead comes from the multiplexing logic in the switch which can be minimized by using techniques mentioned in [19, 20, 21].

We have shown that a more architecture-oriented approach to designing the FU enables it to be small and fast. As a result the routing for the coarse grained array becomes the limiting factor which must be addressed.

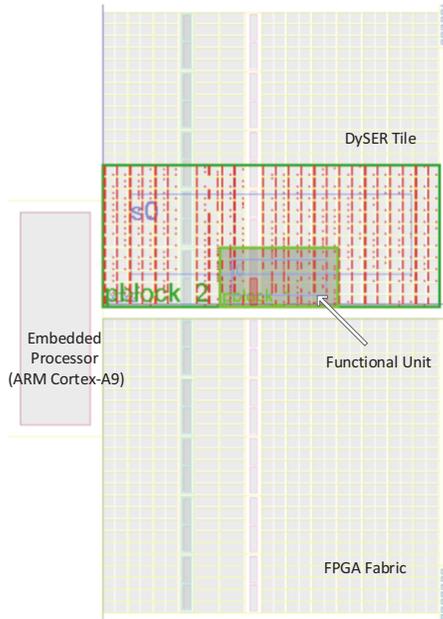


Figure 6: Physical mapping of the DySER Tile on FPGA.

## 5. AREA OVERHEAD QUANTIFICATION

The overlay fabric is implemented by replicating tiles and switches on the FPGA fabric. One tile consumes 2.16% of Slices and one switch consumes 1.88% of the Slices present in the fabric. As discussed previously, an  $N \times N$  DySER overlay incorporates  $N^2$  Tiles in the tile fabric and  $2N+1$  switches in the edge fabric. Hence, theoretically a  $6 \times 6$  DySER overlay is the largest that can fit on the Zynq-7020. Table 4 shows the resource usage for different DySER overlay sizes while Fig. 7 shows the FPGA resource utilization.

As a comparison, albeit an unfair one as we are comparing static implementations requiring a relatively long compile time with rapidly compiled dynamic implementations, we generate RTL of the compute kernels using Vivado HLS 2013.2 in order to perform a quantitative analysis of area overheads. Table 3 shows the results for the Vivado HLS implementations of the benchmarks. The compute kernels ranged from using 0.3-1.1% (on average 0.6%) of the total

Table 4: Resource usage for 16-bit DySER on Zynq-7020

Resource type	2x2	3x3	4x4	5x5	6x6
LUTs	5330	12785	22306	33875	48171
FFs	2781	5493	8950	13390	18728
Slices	2458	6538	9700	12284	13244
DSPs	4	9	16	25	36

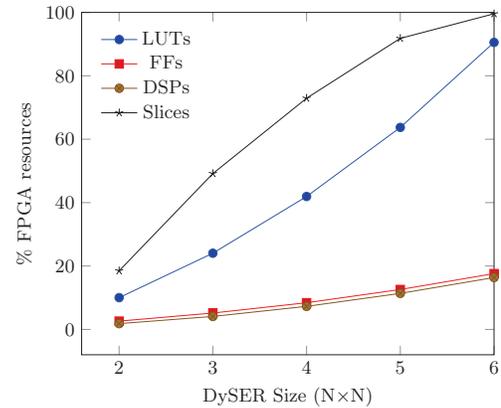


Figure 7: % Resource usage of Zynq-7020 for 16-bit DySER

LUTs in the FPGA, 0.4-1.2% (on average 0.7%) of the total FFs in the FPGA, 0.7-1.6% (on average 1.3%) of the total Slices in the FPGA and 0.9-3.6% (on average 2.8%) of the total DSP blocks in the FPGA.

A fixed configuration  $5 \times 5$  FU array can be used to implement all of the compute kernels without flexible routing. This consumes 5.5% LUTs, 2.7% FFs, 6.9% Slices and 11.4% DSP blocks, while a fully functional  $5 \times 5$  DySER overlay consumes 63.7% LUTs, 12.6% FFs, 92.4% Slices and 11.4% DSP blocks. We assess the overhead of the programmability in a similar way to [6]. The programmability overhead is the ratio of the DySER overlay resources to those of the fixed configuration array of FUs that comprise it. Hence, a  $5 \times 5$  DySER overlay can be used to implement all of the compute kernels with a programmability overhead of  $11 \times$  more LUTs,  $5 \times$  more FFs, and  $13 \times$  more Slices.

Table 3: Experimental results for the Vivado-HLS implementations of the benchmark set

Benchmark	LUTs	FFs	Slices	DSPs	Frequency (MHz)
fft	218 (0.4%)	485 (0.4%)	117 (0.9%)	4 (1.8%)	324
kmeans	613 (1.1%)	1252(1.2%)	215 (1.6%)	8 (3.6%)	249
mm	315 (0.6%)	920 (0.8%)	205 (1.5%)	8 (3.6%)	295
mri-q	243 (0.4%)	588 (0.5%)	147 (1.1%)	6 (2.7%)	268
spmv	292 (0.5%)	842 (0.8%)	180 (1.3%)	8 (3.6%)	297
stencil	460 (0.8%)	870 (0.8%)	200 (1.5%)	2 (0.9%)	303
conv	353 (0.6%)	918 (0.8%)	222 (1.6%)	8 (3.6%)	272
radar	163 (0.3%)	457 (0.4%)	92 (0.7%)	6 (2.7%)	304
$5 \times 5$ FU array	2900 (5.5%)	2925 (2.7%)	925 (6.9%)	25 (11.4%)	370
$5 \times 5$ DySER	33875 (63.7%)	13390 (12.6%)	12284 (92.4%)	25 (11.4%)	175

## 6. CONCLUSION

We have presented an enhancement to the DySER coarse-grained overlay that uses the Xilinx DSP48E1 primitive to implement most of the functional unit, improving area and performance. We show an improvement of  $2.5\times$  in frequency and a reduction of 25% in area compared to the original functional unit design. We quantify the area overheads by mapping a set of benchmarks to the DySER overlay and to the FPGA fabric using Vivado HLS. The experimental results show that a  $5\times 5$  DySER overlay can be used to implement all of the compute kernels with a programmability overhead of  $11\times$  LUTs,  $5\times$  FFs,  $13\times$  Slices.

We have demonstrated that an architecture-focused FU design exposes the significant overhead of the flexible routing. Hence we believe optimizing the switch network to reduce this overhead is a key priority. We are exploring alternative approaches to communication in the context of such overlays. Integrating DySER control with the ARM processor and cycle by cycle reconfiguration of the DSP block in the functional unit would enable us to explore resource sharing for larger applications.

## 7. REFERENCES

- [1] (2013) Zynq-7000 technical reference manual. Xilinx Ltd. [Online]. Available: [http://www.xilinx.com/support/documentation/user\\_guides/ug585-Zynq-7000-TRM.pdf](http://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf)
- [2] 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, October 2014.
- [3] N. W. Bergmann, S. K. Shukla, and J. Becker, "QUKU: a dual-layer reconfigurable architecture," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 12, pp. 63:1–63:26, Mar. 2013.
- [4] K. Vipin and S. A. Fahmy, "ZyCAP: Efficient partial reconfiguration management on the Xilinx Zynq," *IEEE Embedded Systems Letters*, vol. 6, no. 3, pp. 41–44, September 2014.
- [5] 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.
- [6] 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, pp. 1–8.
- [7] 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, pp. 1–12.
- [8] B. Ronak and S. A. Fahmy, "Efficient mapping of mathematical expressions into DSP blocks," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, 2014.
- [9] H. Y. Cheah, F. Brossier, 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.
- [10] C. Liu, C. Yu, and H. So, "A soft coarse-grained reconfigurable array based high-level synthesis methodology: Promoting design productivity and exploring extreme FPGA frequency," in *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2013.
- [11] K. Paul, C. Dash, and M. Moghaddam, "reMORPH: a runtime reconfigurable architecture," in *Euromicro Conference on Digital System Design*, 2012.
- [12] V. Govindaraju, C.-H. Ho, T. Nowatzki, J. Chhugani, N. Satish, K. Sankaralingam, and C. Kim, "DySER: Unifying functionality and parallelism specialization for energy-efficient computing," *IEEE Micro*, vol. 32, no. 5, pp. 38–51, 2012.
- [13] 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, pp. 503–514.
- [14] M. B. Taylor, J. Kim, J. Miller, D. Wentzloff, F. Ghodrati, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee, W. Lee *et al.*, "The Raw microprocessor: A computational fabric for software circuits and general-purpose programs," *Micro, IEEE*, vol. 22, no. 2, pp. 25–35, 2002.
- [15] S. Swanson, A. Schwerin, M. Mercaldi, A. Petersen, A. Putnam, K. Michelson, M. Oskin, and S. J. Eggers, "The wavescalar architecture," *ACM Transactions on Computer Systems (TOCS)*, vol. 25, no. 2, p. 4, 2007.
- [16] D. Burger, S. W. Keckler, K. e. McKinley, M. Dahlin, L. K. John, C. Lin, C. R. Moore, J. Burrill, R. G. McDonald, and W. Yoder, "Scaling to the end of silicon with EDGE architectures," *Computer*, vol. 37, no. 7, pp. 44–55, 2004.
- [17] A. K. Jain, S. A. Fahmy, and D. L. Maskell, "Efficient overlay architecture based on DSP blocks," in *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2015.
- [18] 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," *Energy (mJ)*, vol. 5, no. 10, p. 15.
- [19] 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*, 2012, pp. 111–120.
- [20] C. H. Hoo and A. Kumar, "An area-efficient partially reconfigurable crossbar switch with low reconfiguration delay," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, 2012, pp. 400–406.
- [21] K. Heyse, T. Davidson, E. Vansteenkiste, K. Bruneel, and D. Stroobandt, "Efficient implementation of virtual coarse grained reconfigurable arrays on FPGAs," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, 2013, pp. 1–8.