Efficient Overlay Architecture Based on DSP Blocks

NANYANG TECHNOLOGICAL UNIVERSITY
School of Computer Engineering

Abhishek Kumar Jain, Suhaib A. Fahmy, Douglas L. Maskell
abhishek013@ntu.edu.sg, sfahmy@ntu.edu.sg, asdouglas@ntu.edu.sg

Background and Motivation

Major issues in mainstream adoption of FPGAs:

- Difficulty of accelerator design at low level
- Long compilation times (Place and route)
- Poor design productivity

One possible solution is to use FPGA Overlay:

- Accelerator design in a high level language
- Fast compilation and development cycles
- Cost: Area and performance overheads
- Little consideration for the underlying FPGA architecture in existing overlays
- Possible inference of hard blocks by synthesis tools for compute logic
- Does not exploit full cabability of the block

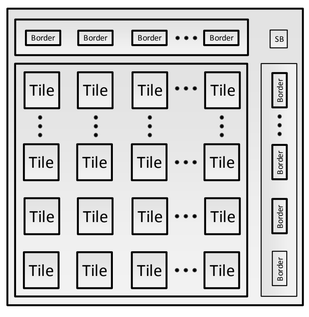
Exploit fully pipelined DSP Blocks:

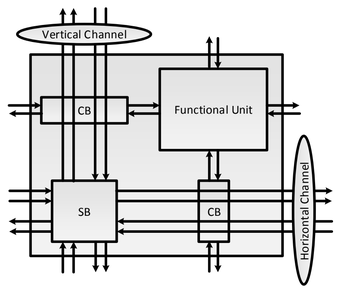
- As programmable processing elements
- To develop high throughput overlays

Efficient Overlay Architecture

2D array of tiles:

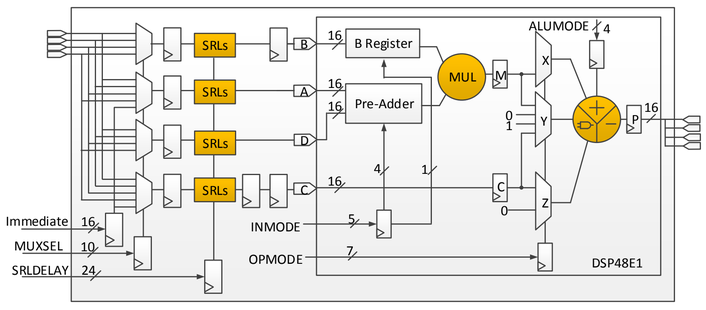
- Programmable functional unit (FU) and routing resources in each tile
- Functional units interconnected via an island-style routing network
- Coarse grained switch boxes, connection boxes and routing channels as programmable routing resources
- Customizable channel width (CW), number of tracks in a routing channel





DSP Block Based Functional Unit

- Fully pipelined DSP48E1 as a programmable processing element (PE)
- Achievable frequency near theoretical limits for providing high throughput
- A pre-adder, a multiplier and an ALU inside the functional unit
- Can support upto 3 operations
- MUX based reordering logic to handle logical inequivalence at the PE inputs
- SRL based variable-length shift registers for balancing pipeline latencies



Contributions

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

Observations

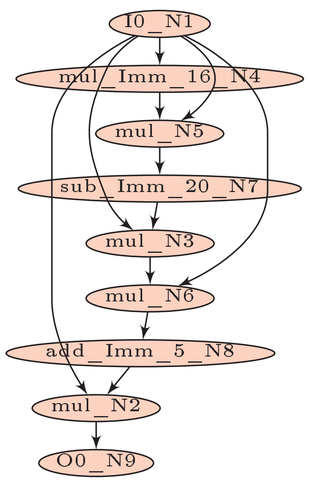
- Resource usage tracks our expectations
- Slice usage becomes a limiting factor
- A modest drop in frequency
- A frequency of 300 MHz for an 8×8 overlay with a peak throughput of 56 GOPS
- Upto 53% savings in the number of tiles required to map the benchmark set (Using DSP48E1 aware mapping)
- A throughput of up to 21.6 GOPS for the benchmarks using the proposed overlay
- Reconfiguration time of 11.5 us and 28 us for Overlay-I and Overlay-II, respectively, compared to 31.6 ms for the entire PL using PCAP

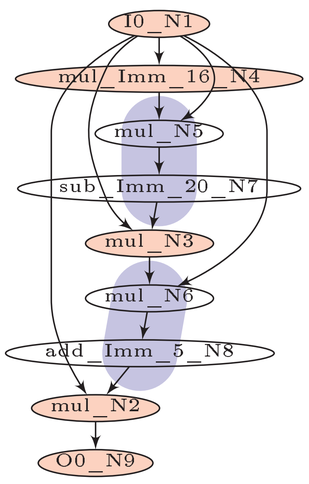
Conclusions and Future Work

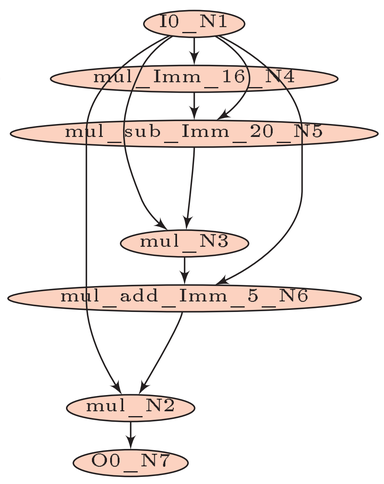
- Pipelined execution of compute kernels using DSP block based efficient overlay
- An improvement of 11–52% in throughput compared to Vivado HLS implementations
- Area reduction of the overlay further through careful optimizations of the routing architecture and synchronization logic.
- Balancing DSP/CLB resource usage across FUs and overlay routing
- Alternative interconnect architectures for a low overhead routing network

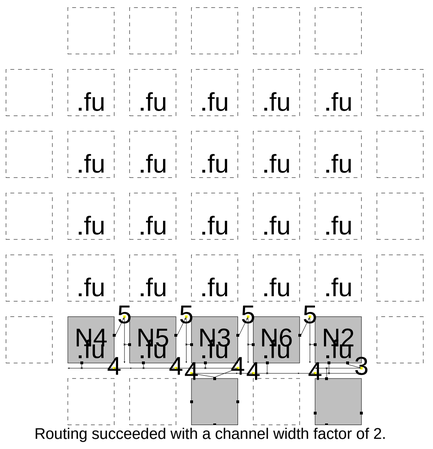
Rapid, Vendor-Independent, Automated Mapping of Compute Kernels

- C to DFG Transformation: DFG generation from a C description of the compute kernel
- DSP48E1 Aware Mapping: Compute node merging based on the capability of the DSP block
- Placement and Routing of FU Netlist: Using VPR for mapping nodes in the graph to the DSP blocks, and edges onto the coarse grained tracks
- Latency Balancing: Parsing VPR output files and generating a routing resource graph to determine the latency imbalance at each node and hence the required delays at the FU inputs





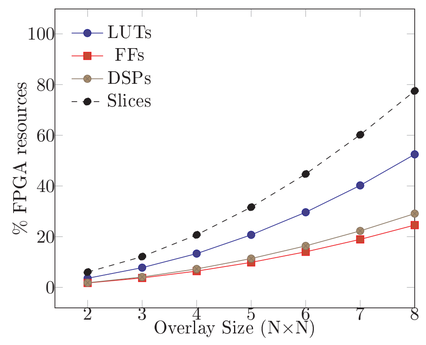


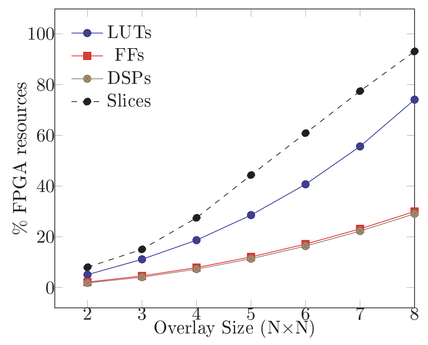


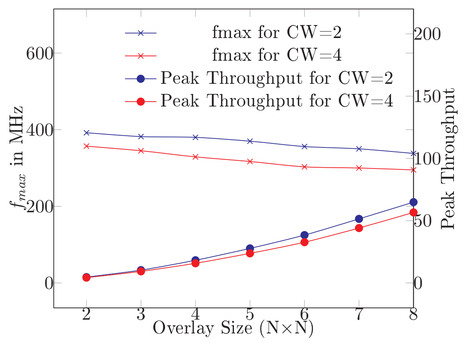
Routing succeeded with a channel width factor of 2.

Experimental Evaluation

- Two example overlays on Zynq device to execute the benchmark set: a 5×5 Overlay-I with CW=2 operating at 370 MHz and a 7×7 Overlay-II with CW=4 operating at 300 MHz
- RTL generation of benchmarks using Vivado HLS for performance (throughput) comparison







Benchmark	Benchmark Characteristics i/o nodes	op nodes	merged nodes	savings	Routability CW=2	CW=4	Overlay Results Latency	MLI	**GOPS**	HLS Implementation Results Latency	Fmax	**GOPS**	Slices	DSPs
chebyshev	1/1	7	5	28%	3×3	3×3	49	36	**2.59**	13	333	**2.3**	24	3
sgfilter	2/1	18	10	44%	4×4	4×4	54	31	**6.66**	11	278	**5.0**	40	8
mibench	3/1	13	6	53%	3×3	3×3	47	35	**4.81**	9	295	**3.8**	81	3
qspline	7/1	26	22	15%	5×5	5×5	76	64	**9.62**	21	244	**6.3**	126	14
poly1	2/1	9	6	33%	3×3	3×3	34	22	**3.33**	12	285	**2.56**	62	4
poly2	2/1	9	6	33%	3×3	3×3	29	7	**3.33**	11	295	**2.65**	45	4
poly3	6/1	11	7	36%	3×3	3×3	31	11	**4.07**	12	250	**2.75**	52	6
poly4	5/1	6	3	50%	2×2	2×2	24	12	**2.22**	7	312	**1.87**	36	3
atax	12/3	60	36	40%	—	6×6	72	58	**18.0**	13	263	**15.8**	78	18
bicg	15/6	30	18	40%	—	6×6	46	32	**9.0**	7	270	**8.1**	91	18
trmm	18/9	54	36	33%	—	7×7	58	30	**16.2**	8	222	**11.9**	105	36
syrk	18/9	72	45	37%	—	7×7	41	19	**21.6**	10	250	**18**	237	24