DATE 18

MARCH 19—23, 2018

DRESDEN, GERMANY

INTERNATIONAL CONGRESS CENTER

DESIGN, AUTOMATION AND TEST IN EUROPE THE EUROPEAN EVENT FOR ELECTRONIC SYSTEM DESIGN & TEST

A Time-Multiplexed FPGA Overlay with Linear Interconnect

Xiangwei Li, Douglas L. Maskell
School of Computer
Science and Engineering
Nanyang Technological University

Abhishek K. Jain,
Lawrence Livermore
National Laboratory

Suhaib A. Fahmy,
School of Engineering
University of Warwick

NANYANG TECHNOLOGICAL UNIVERSITY

Lawrence Livermore National Laboratory

THE UNIVERSITY OF WARWICK

Design Productivity of Modern FPGAs

Problems

- **Low level of abstraction**
  - **Register-transfer level (RTL) design**
- **Complexity of SoC design**
  - **CPU, GPU, hardware, OS support, interfacing…**
- **Lengthy hardware compilation time**
  - **Fine-grained level placement and route**

Design Productivity of Modern FPGAs

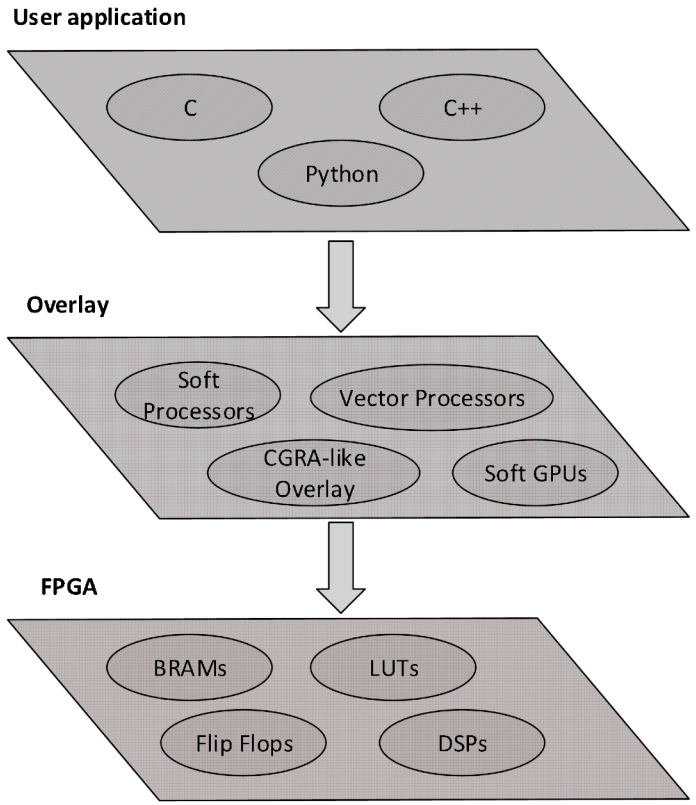
Solutions

- **High-level Synthesis (HLS)**
  - **SoC design is still difficult**
  - **Long compilation time**
- **SoC EDA Tools**
  - **Long compilation time**
- **Coarse-grained FPGA Overlays**
  - **Could be included as a processing technology into the SoC EDA tools**

Coarse-grained FPGA Overlays

- A programmable coarse-grained hardware abstraction layer, implemented on top of an FPGA.
- Advantages
  - A higher level of abstraction
  - Software-like programmability
  - Fast compilation
- Typical overlays
  - Soft processors
  - Soft GPUs
  - Vector processors
  - CGRA-like overlays

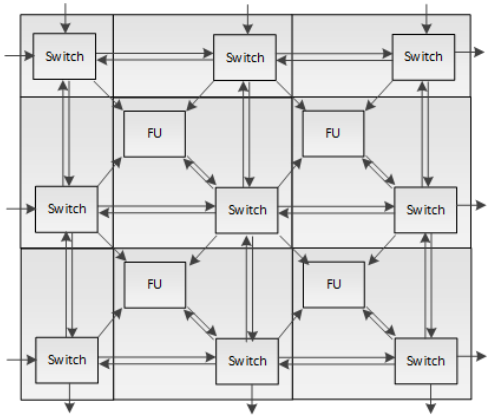
Processor-based



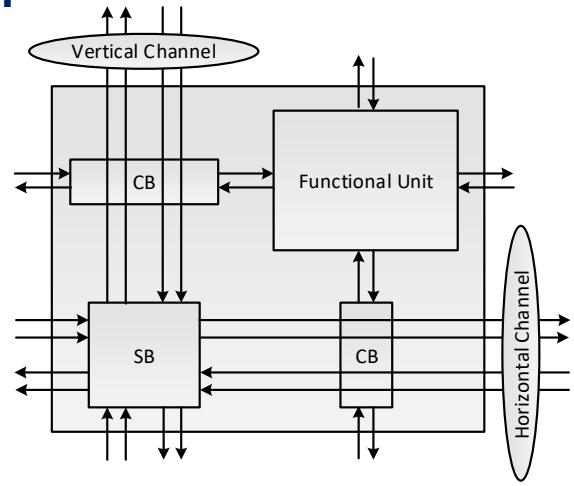
CGRA-like: Spatially Configured Overlays

Consist of an array of processing elements connected by a routing network (such as NN, IS)

- **They are throughput oriented with an II of 1**
- **No sharing of FUs among multiple operations**
  - **to achieve high throughput**
- **Resource hungry due to FU requirement for each operation and the connection network**
  - **Examples: IF [1], DySER [2], DSP based Overlay [3], DeCO [4]**
- **Can we share FUs to reduce area requirements**
  - **Possibly at the cost of reduced throughput?**



DySER Overlay

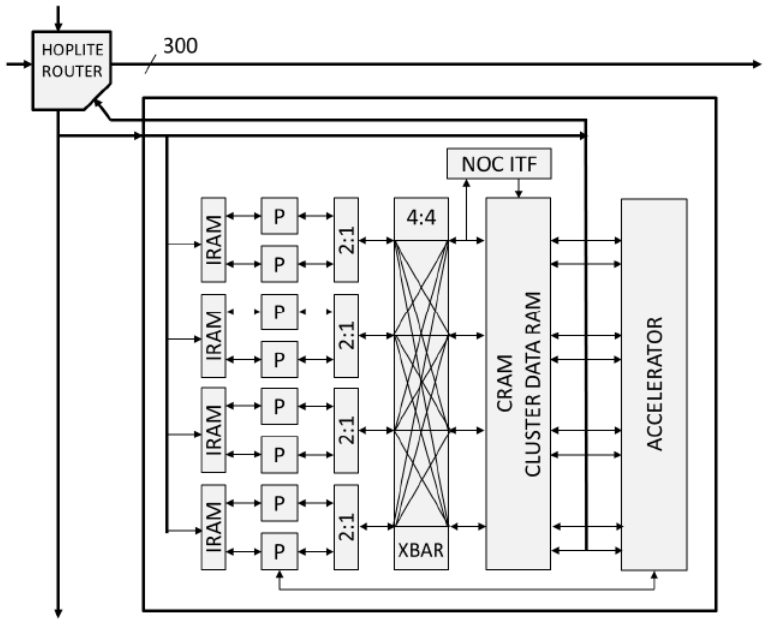


Island-style DSP based Overlay

CGRA-like: Time-Multiplexed Overlays

Many different configurations

- **Processor arrays**
  - **NoC based**
  - **High performance**
  - **Significant area overhead**
  - **Examples: GRVI Phalanx [5], 120-core MIPS Overlay [6]**
- **Medium-grained overlays**
  - **NN or Island-style**
  - **Moderate performance**
  - **Lower area consumption**
  - **Examples: SCGRA Overlay [7], reMORPH [8]**



GRVI Phalanx

CGRA-like Medium-grained Overlays

Reduced FU requirements, but at the expense of II, and hence throughput

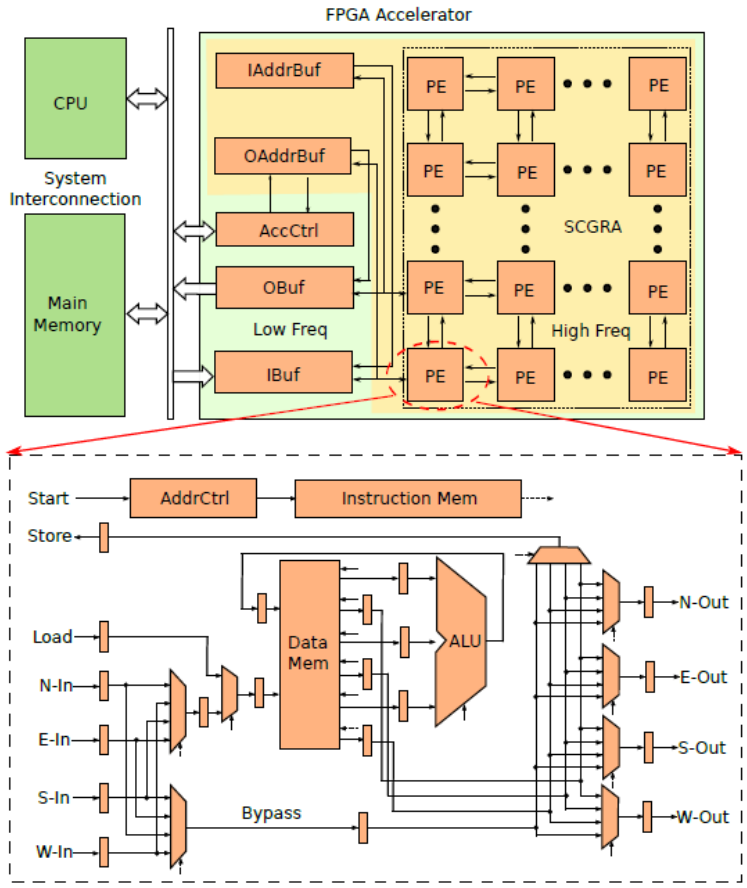
- **Still use considerable FPGA resource**
  - **Interconnect**
  - **BRAMs**

Some examples

- **5x5 SCGRA can fit on Zynq-7020**
  - **Limited scalability due to instruction storage requirement**
  - **Need to store completely unrolled instruction stream in BRAMs**

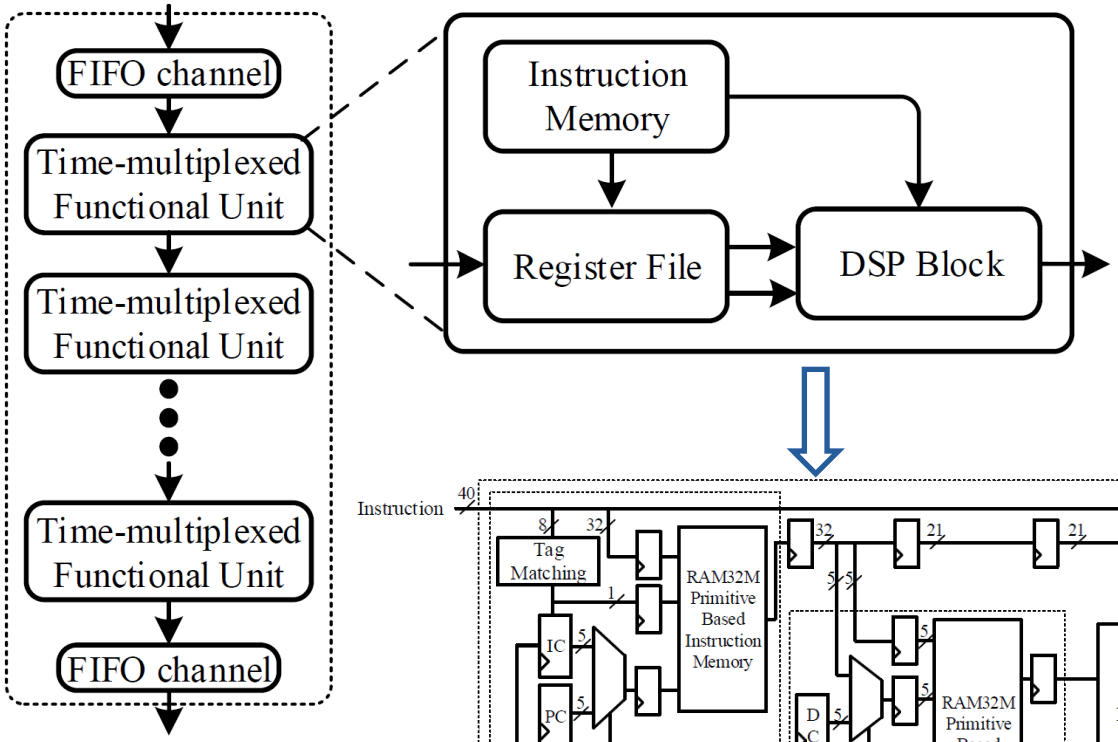
- **reMORPH: Another similar overlay**
  - **Same problem of instruction storage**
  - **FU not really FPGA architecture friendly**

- **So, can we reduce the FPGA hardware requirements further?**



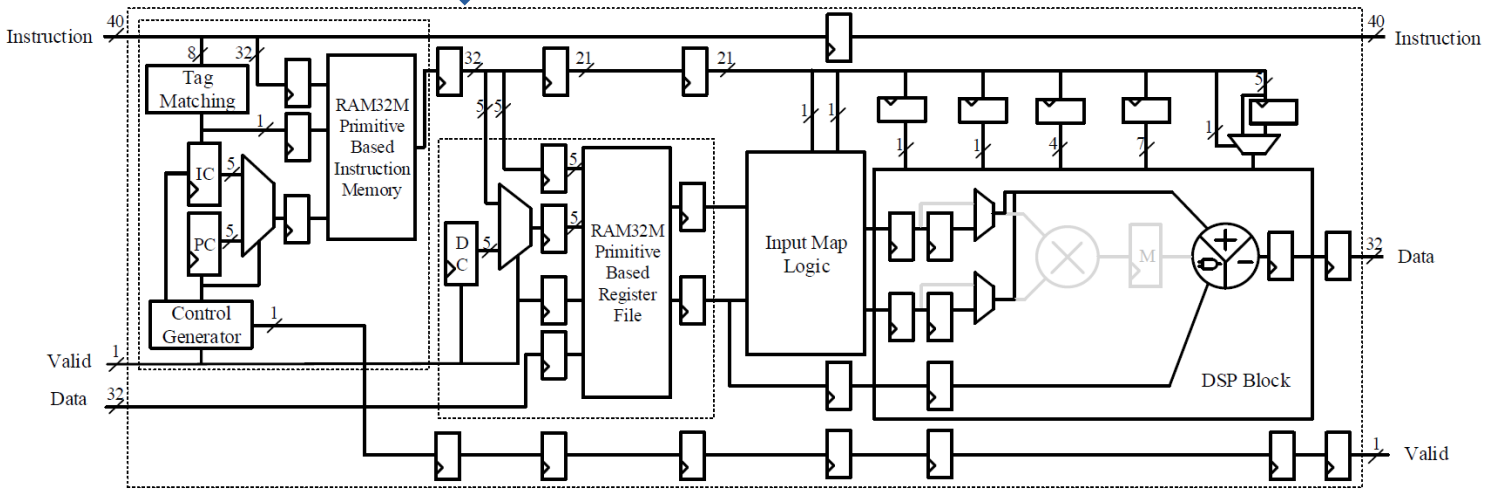
SCGRA overlay

A Linear TM Overlay [9]



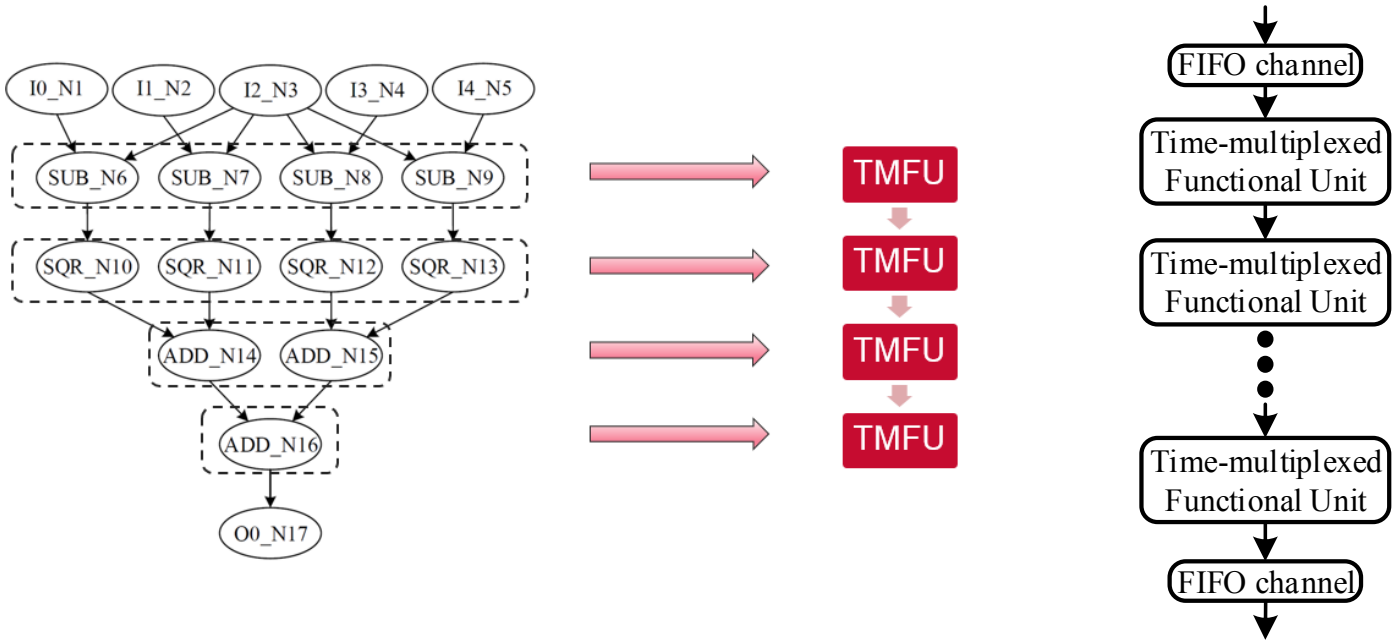
No need for switch box and connection box

- Compared to a conventional array-based overlay.



Uses RAM32M primitives for the instruction memory and register file instead of BRAMs.
FU = 1 DSP + 160 LUTs + 293 FFs, and achieves up to 325 MHz on Zynq and 600 MHz on V7.

Mapping to the Linear TM Overlay

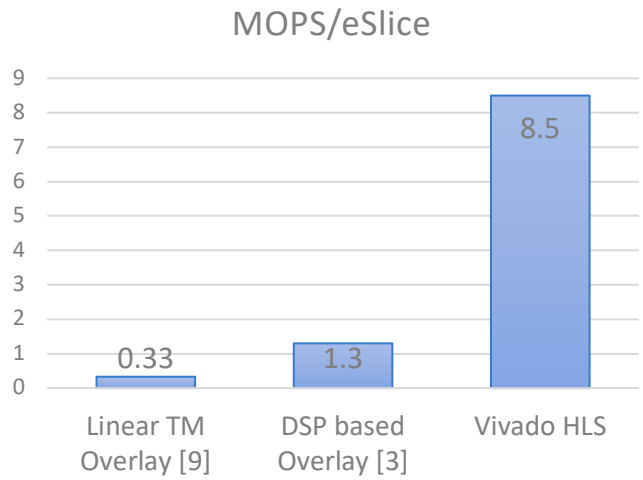


ASAP scheduling was used where each stage is mapped to a FU in the overlay.

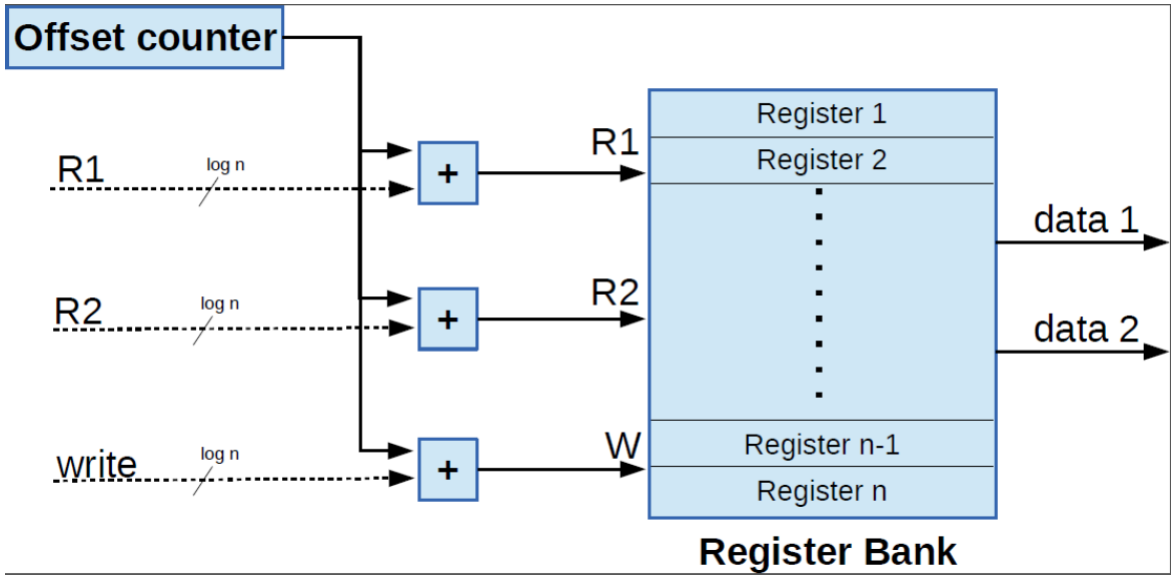
Limitations of the Linear TM Overlay

The compute efficiency is relatively low

- Initiation interval is large: Low throughput (~10% of Vivado HLS)
  - Due to the non-overlap of data load and execution
    - ➢ Add a rotating register file
    - ➢ Replicate the streaming datapath (Reuse the IM)
- And it can only handle feed-forward DFGs. Also, the size (depth) of overlay varies with application
  - Change the FU mapping by adding write-back support



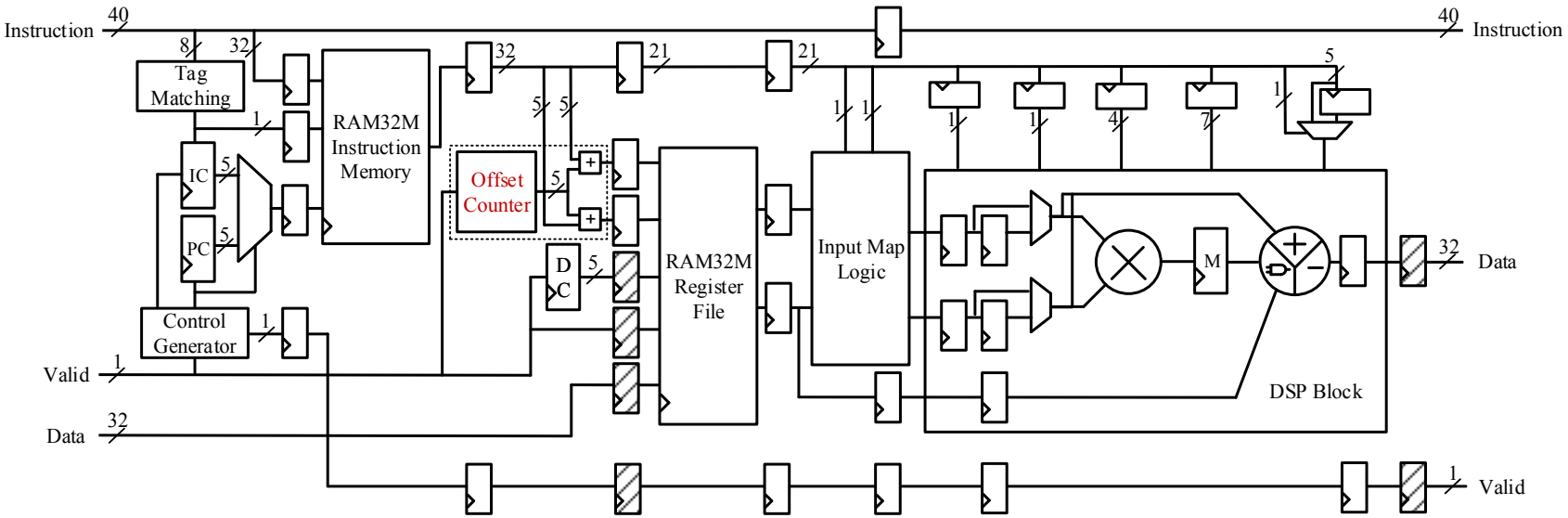
Rotating Register File



With rotating register files, it is possible to execute the arithmetic operations and load/store new set of input data simultaneously when there is no conflict.

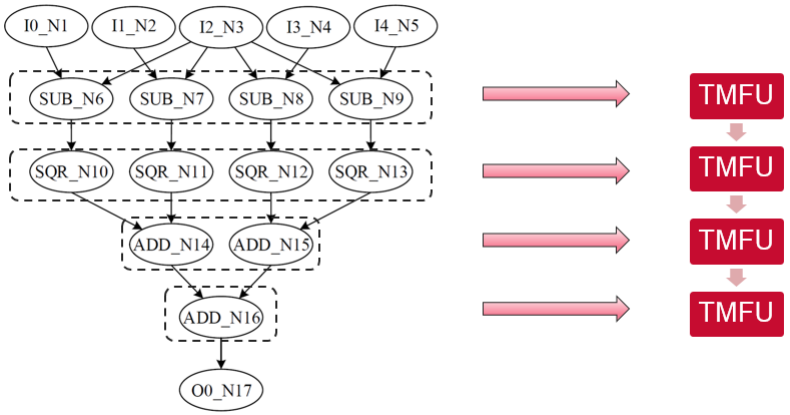
Architecture Enhancement (V1)

- **Rotating Register File**



V1 implementation: 1 FU = 1 DSP + 196 LUTs + 237 FFs
(22.5% more LUTS and 19.1% less FFs than [9])
Running at 334 MHz on Zynq (2.8% higher than [9])

Original Instruction Scheduling [9]

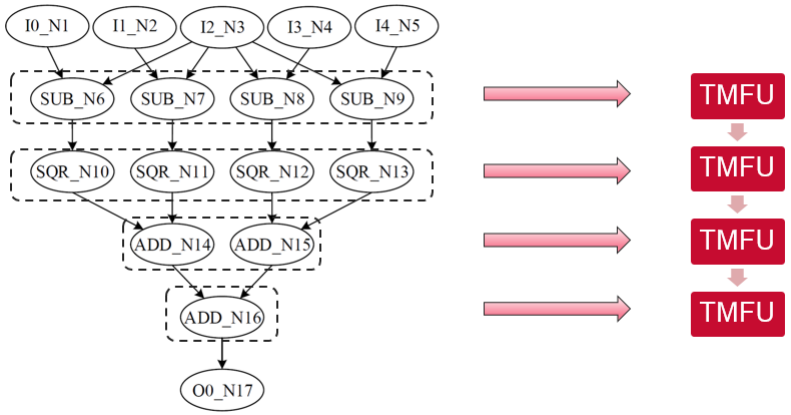


cycle	FU0	FU1	FU2	FU3
1	Load R0			
2	Load R1			
3	Load R2			
4	Load R3			
5	Load R4			
6	SUB (R0 R2)			
7	SUB (R1 R2)			
8	SUB (R2 R3)			
9	SUB (R2 R4)			
10		Load R0		
11		Load R1		
12	Load R0	Load R2		
13	Load R1	Load R3		
14	Load R2	SQR (R0 R0)		
15	Load R3	SQR (R1 R1)		
16	Load R4	SQR (R2 R2)		
17	SUB (R0 R2)	SQR (R3 R3)		
18	SUB (R1 R2)		Load R0	
19	SUB (R2 R3)		Load R1	
20	SUB (R2 R4)		Load R2	
21		Load R0	Load R3	
22		Load R1	ADD (R0 R1)	
23	Load R0	Load R2	ADD (R2 R3)	
24	Load R1	Load R3		
25	Load R2	SQR (R0 R0)		
26	Load R3	SQR (R1 R1)		Load R0
27	Load R4	SQR (R2 R2)		Load R1
28	SUB (R0 R2)	SQR (R3 R3)		ADD (R0 R1)
29	SUB (R1 R2)		Load R0	
30	SUB (R2 R3)		Load R1	
31	SUB (R2 R4)		Load R2	
32		Load R0	Load R3	

Initiation interval (II) = 11. Latency =32.

Instruction Scheduling

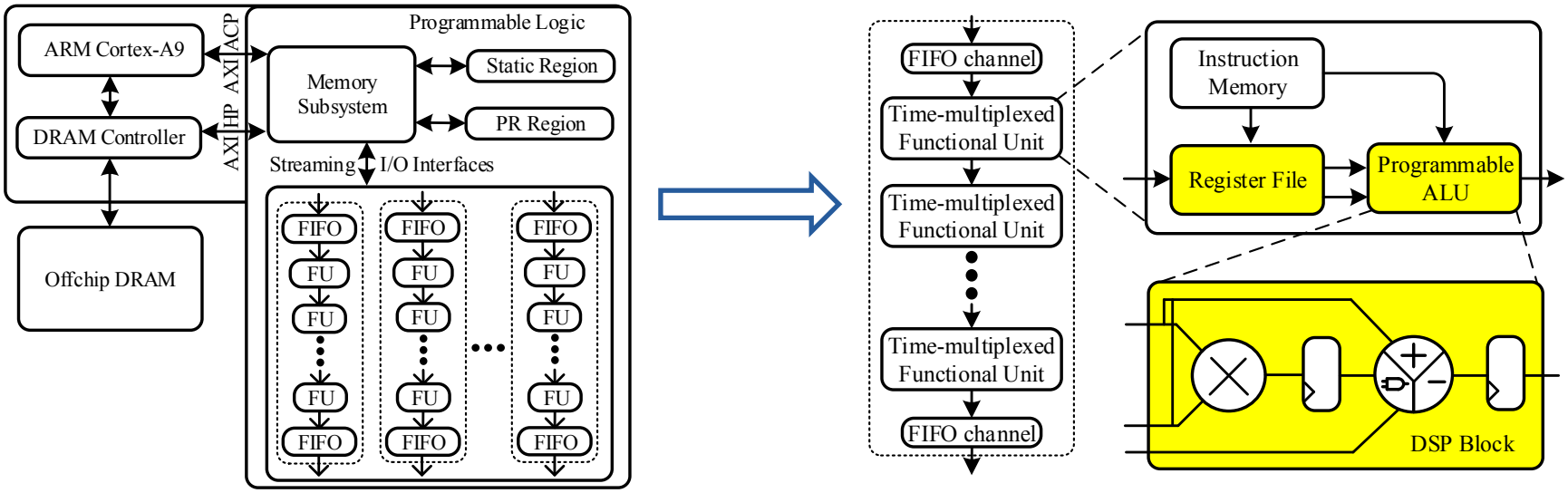
V1 Implementation: Rotating Register File



cyc	FU0		FU1		FU2		FU4	
1	Load R0							
2	Load R1							
3	Load R2							
4	Load R3							
5	Load R4							
6		SUB (R0 R2)						
7	Load R0	SUB (R1 R2)						
8	Load R1	SUB (R2 R3)						
9	Load R2	SUB (R2 R4)	Load R0					
10	Load R3		Load R1					
11	Load R4		Load R2					
12		SUB (R0 R2)	Load R3					
13	Load R0	SUB (R1 R2)		SQR (R0 R0)				
14	Load R1	SUB (R2 R3)		SQR (R1 R1)				
15	Load R2	SUB (R2 R4)	Load R0	SQR (R2 R2)				
16	Load R3		Load R1	SQR (R3 R3)	Load R0			
17	Load R4		Load R2		Load R1			
18		SUB (R0 R2)	Load R3		Load R2			
19	Load R0	SUB (R1 R2)		SQR (R0 R0)	Load R3			
20	Load R1	SUB (R2 R3)		SQR (R1 R1)		ADD (R0 R1)		
21	Load R2	SUB (R2 R4)	Load R0	SQR (R2 R2)		ADD (R2 R3)		
22	Load R3		Load R1	SQR (R3 R3)	Load R0			
23	Load R4		Load R2		Load R1		Load R0	
24		SUB (R0 R2)	Load R3		Load R2		Load R1	
25	Load R0	SUB (R1 R2)		SQR (R0 R0)	Load R3			ADD (R0 R1)
26	Load R1	SUB (R2 R3)		SQR (R1 R1)		ADD (R0 R1)		
27	Load R2	SUB (R2 R4)	Load R0	SQR (R2 R2)		ADD (R2 R3)		
28	Load R3		Load R1	SQR (R3 R3)	Load R0			
29	Load R4		Load R2		Load R1		Load R0	
30		SUB (R0 R2)	Load R3		Load R2		Load R1	
31	Load R0	SUB (R1 R2)		SQR (R0 R0)	Load R3			ADD (R0 R1)
32	Load R1	SUB (R2 R3)		SQR (R1 R1)		ADD (R0 R1)		

Initiation interval (II) reduces from 11 to 6.
Latency drops from 32 to 28.

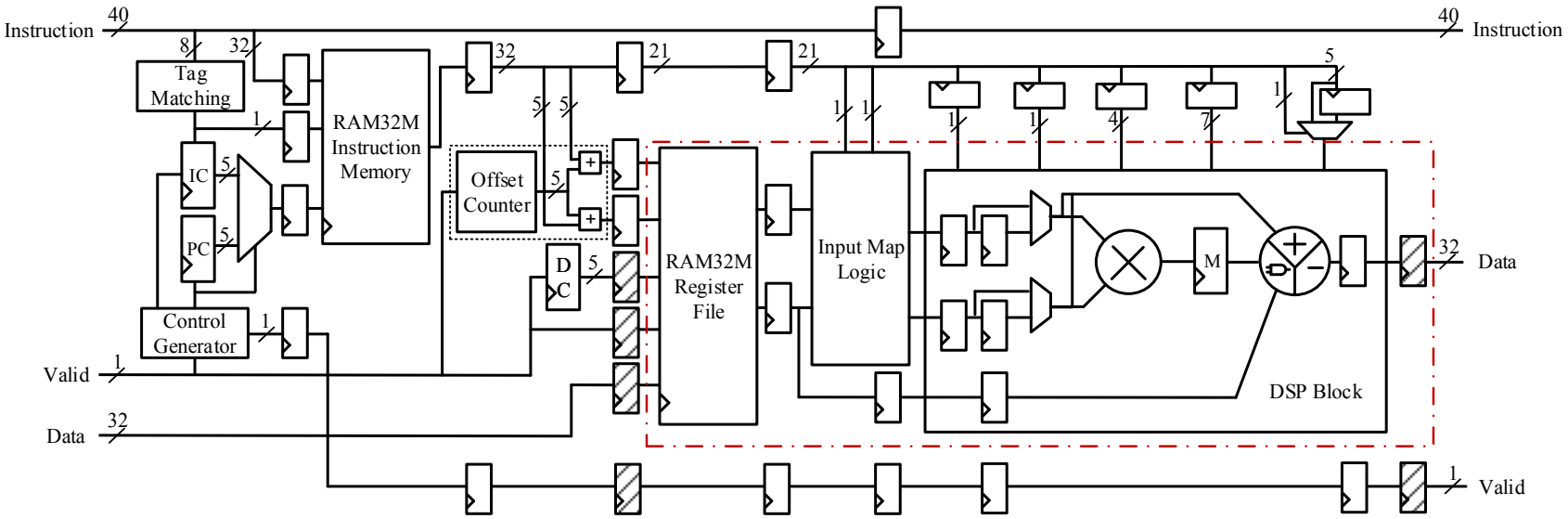
Replicating the Stream Datapath



Replicating the data processing part of the FU and increasing the data I/O to 64-bit can further reduce the II into half, while the IM and other control circuitry are reused at runtime.

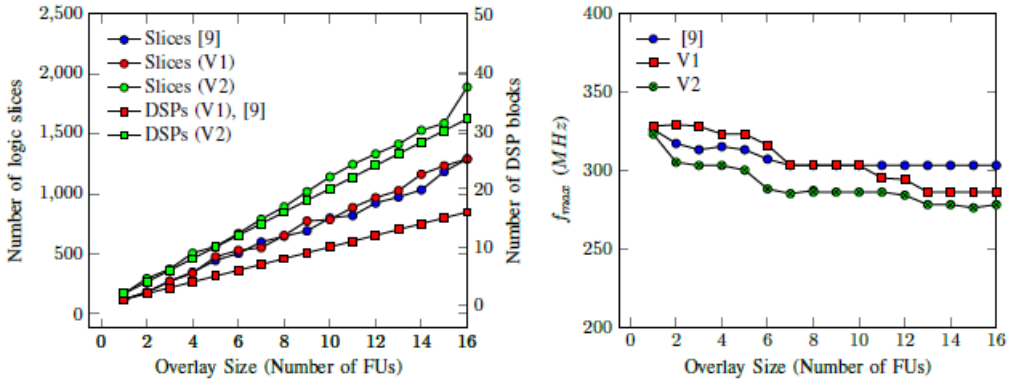
Architecture Enhancement (V2)

- **Replicating the Stream Datapath**



V2 Implementation: 1 FU = 2 DSPs + 292 LUTs + 333 FFs
(49.0% more LUTS and 40.5% more FFs than V1)
Running at 335 MHz on Zynq (almost same as V1)

Overlay Scalability



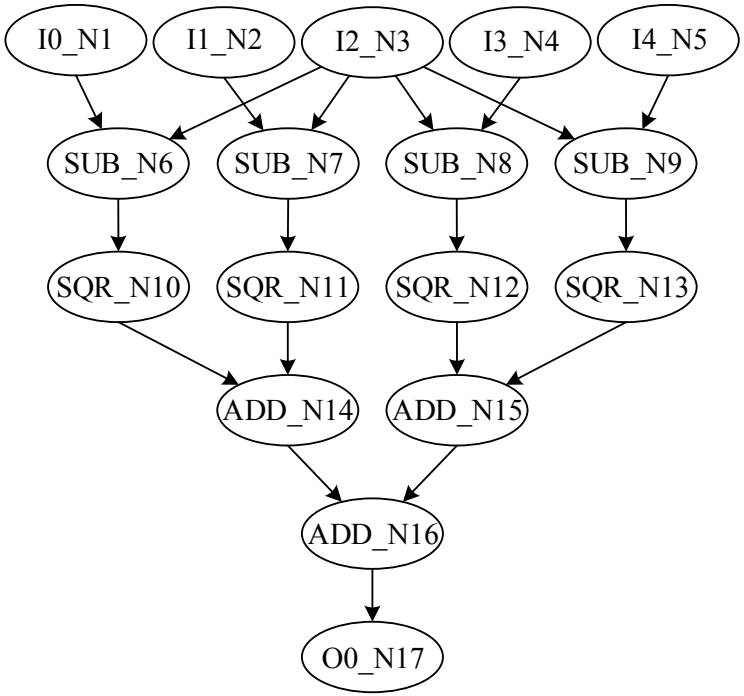
(a) Resource usage. (b) f_{max} Drop.

Fig. 5: V1 and V2 Overlay scalability on Zynq XC7Z020

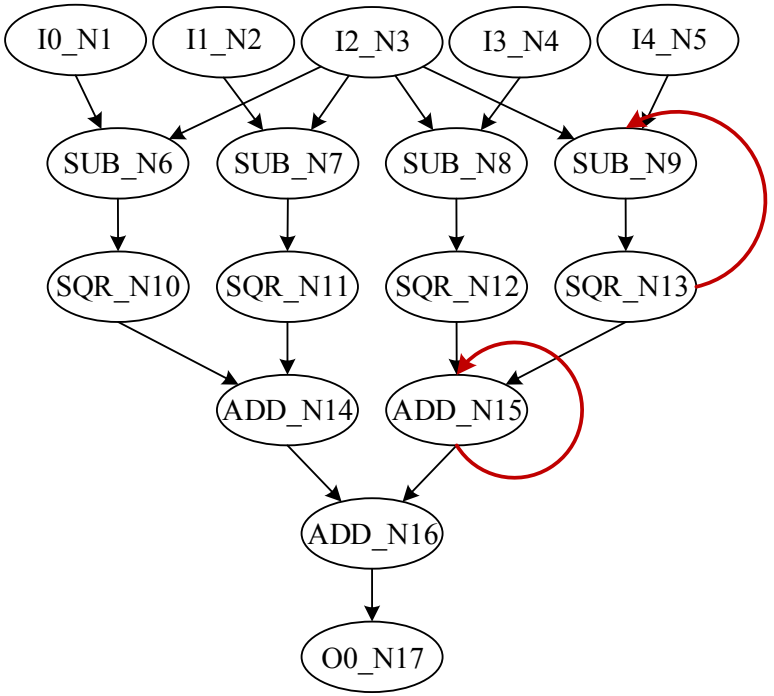
V1 overlay (depth=8) consumes less than 5% of the Zynq resources. Fmax =303 MHz
V2 overlay (depth=8) consumes less than 8% of the Zynq resources. Fmax = 287 MHz

DFG Characteristics

Feed-forward DFG

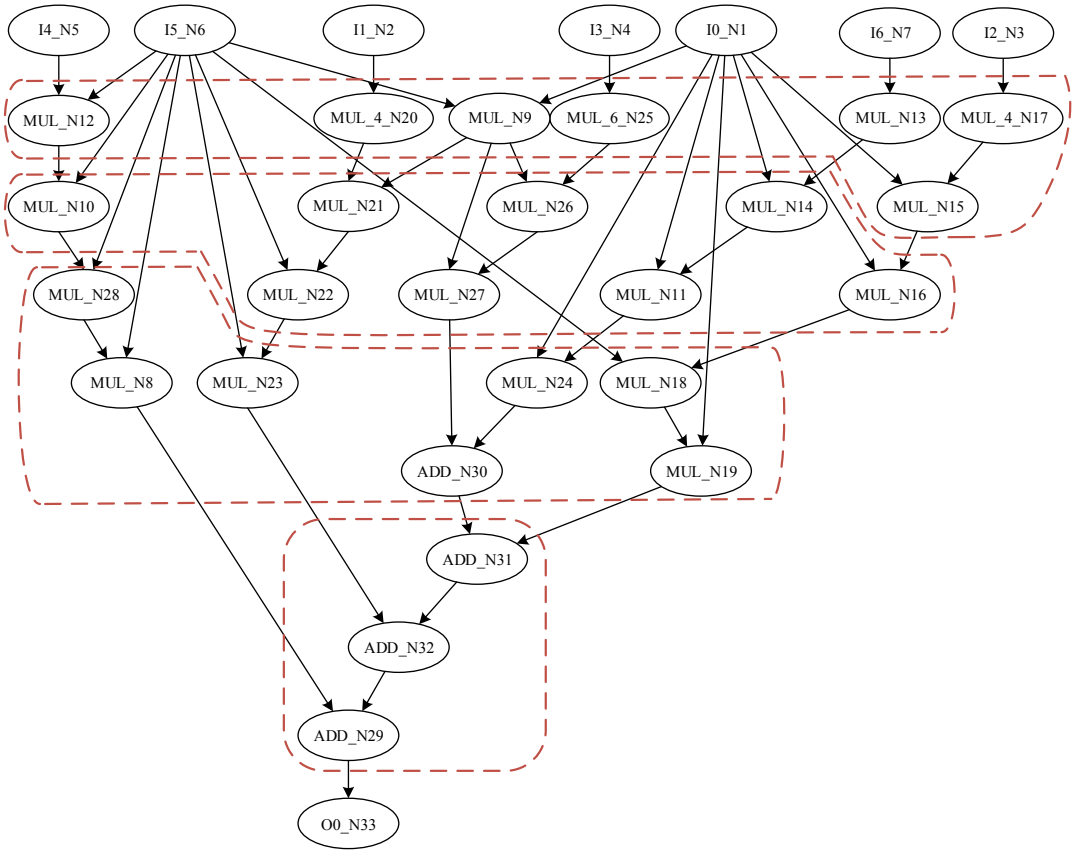


Feedback DFG



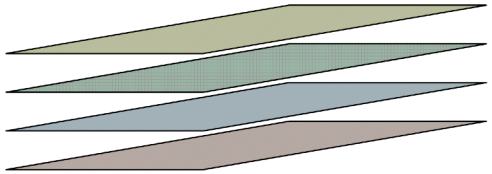
Similar to [9], V1 and V2 can only handle feedforward DFGs.
When the DFG has inter dependences, **FU write-back support** is necessary.

Overlay Reconfiguration



Overlay Depth: 8 → 4
II: 11 → 15

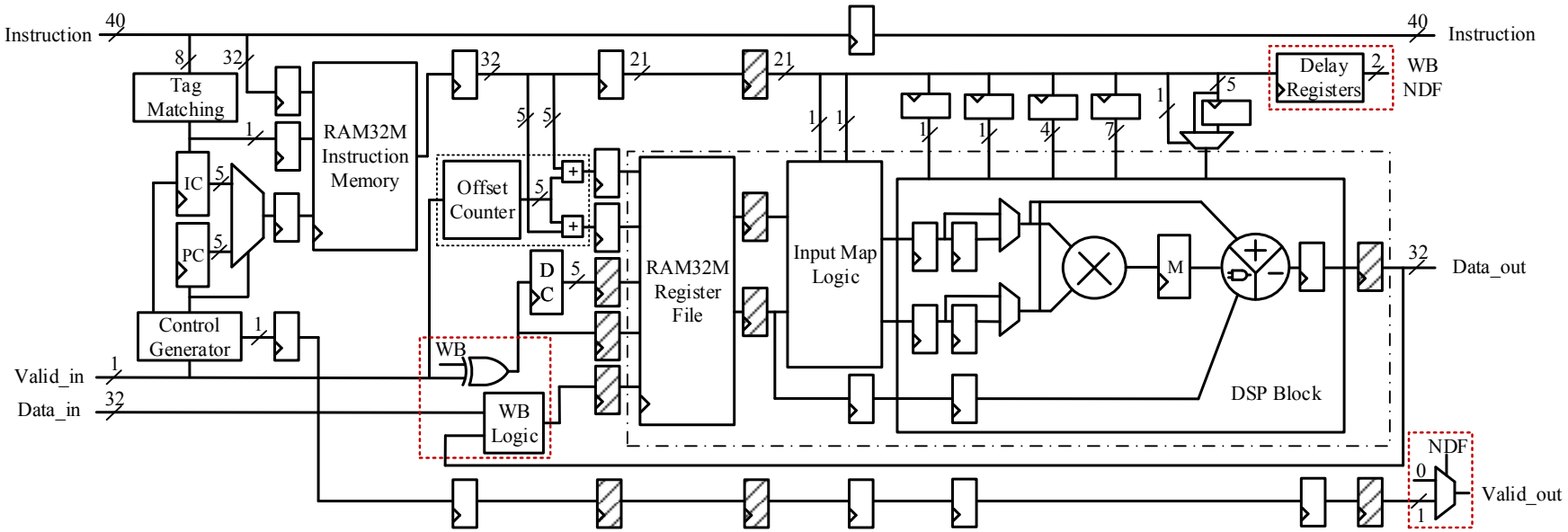
Pre-synthesized overlay library



The overlay has to be reconfigured when the depth (critical path) of the DFG is changed.
To avoid frequent overlay reconfiguration, **FU write-back** should be introduced.

Architecture Enhancement (V3-V5)

- **FU Write-back Support**



V3 implementation: 1 FU = 1 DSP + 212 LUTs + 228 FFs
(8.2% more LUTS and 4.0% less FFs than V1)
Running at 323 MHz on Zynq (3.3% lower than V1)

Summary of Area and Frequency

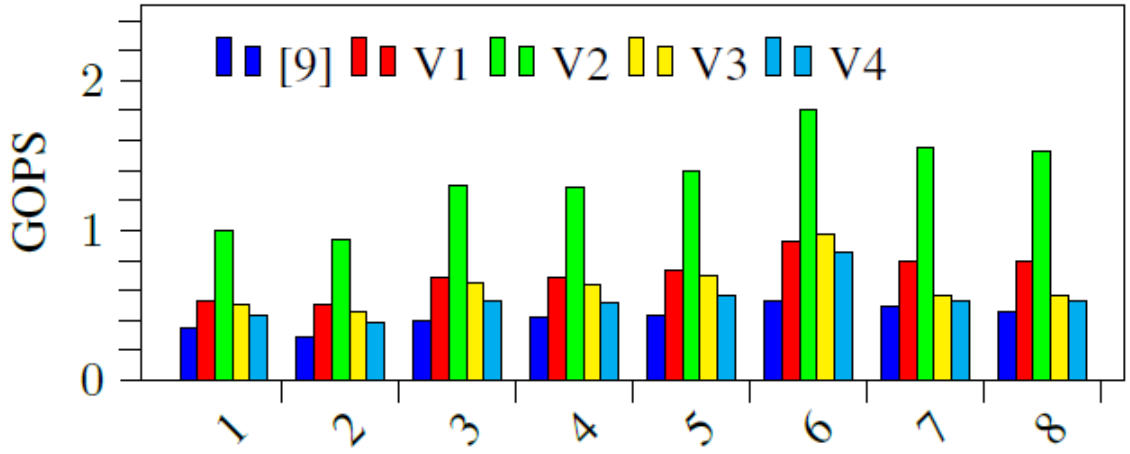
	FU [9]	FU (V1)	FU (V2)	FU (V3)	FU (V4)	FU (V5)
DSP	1	1	2	1	1	1
LUTs	160	196	292	212	207	248
FFs	293	237	333	228	163	126
Slices	81	57	104	107	84	107
Fmax on Zynq	325 MHz	334 MHz	335 MHz	323 MHz	254 MHz	182 MHz
IWP	--	--	--	5	4	3
Write-back support	No	No	No	Yes	Yes	Yes
Rotating register file	No	Yes	Yes	Yes	Yes	Yes

Although V4 and V5 are able to further reduce the internal write-back path, the clock frequencies drop significantly, especially for V5.

Benchmark Evaluation (Throughput)

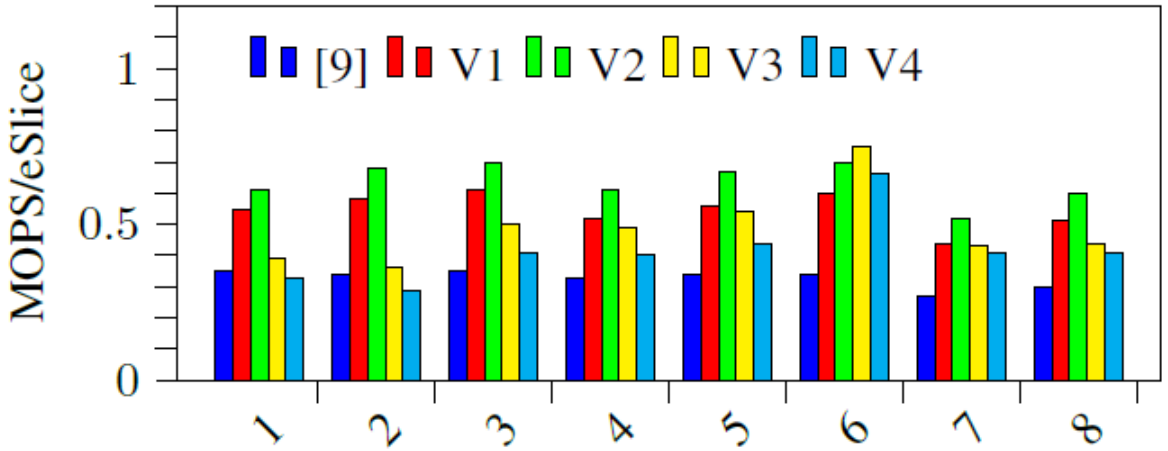
No.	Benchmark	I/O	#Ops	Depth	$II_{[1]}$	II_{V1}	II_{V2}	II_{V3}	II_{V4}
1.	chebyshev	1/1	7	7	6	4	2	4	4
2.	mibench	3/1	13	6	14	8	4	8	8
3.	qspline	7/1	25	8	19	11	5.5	11	11
4.	sgfilter	2/1	18	9	13	8	4	8	8
5.	poly5	3/1	27	9	19	11	5.5	11	11
6.	poly6	3/1	44	11	25	14	7	13	12
7.	poly7	3/1	39	13	24	14	7	20	17
8.	poly8	3/1	32	11	21	12	6	16	14

As expected, the V1 II is around 60% of the original II. The V2 II is exactly half of the V1 II. The V3 and V4 II are close to the V1 II.



Benchmark Evaluation (Efficiency)

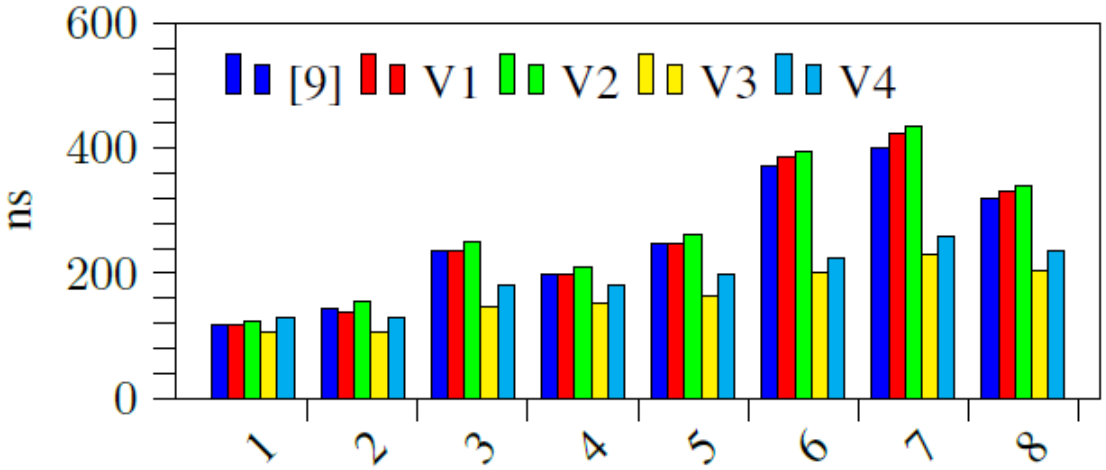
No.	Benchmark	I/O	#Ops	Depth	$II_{[1]}$	II_{V1}	II_{V2}	II_{V3}	II_{V4}
1.	chebyshev	1/1	7	7	6	4	2	4	4
2.	mibench	3/1	13	6	14	8	4	8	8
3.	qspline	7/1	25	8	19	11	5.5	11	11
4.	sgfilter	2/1	18	9	13	8	4	8	8
5.	poly5	3/1	27	9	19	11	5.5	11	11
6.	poly6	3/1	44	11	25	14	7	13	12
7.	poly7	3/1	39	13	24	14	7	20	17
8.	poly8	3/1	32	11	21	12	6	16	14



V1, V2, V3, and V4 achieve 66.7%, 93.7%, 48.5%, 27.3% better compute efficiency compared to that of [9] on average, respectively.

Benchmark Evaluation (Latency)

No.	Benchmark	I/O	#Ops	Depth	$II_{[1]}$	II_{V1}	II_{V2}	II_{V3}	II_{V4}
1.	chebyshev	1/1	7	7	6	4	2	4	4
2.	mibench	3/1	13	6	14	8	4	8	8
3.	qspline	7/1	25	8	19	11	5.5	11	11
4.	sgfilter	2/1	18	9	13	8	4	8	8
5.	poly5	3/1	27	9	19	11	5.5	11	11
6.	poly6	3/1	44	11	25	14	7	13	12
7.	poly7	3/1	39	13	24	14	7	20	17
8.	poly8	3/1	32	11	21	12	6	16	14



Adding write-back and fixing the overlay depth along with a better scheduling strategy significantly reduces the latency.

Conclusion

- Presented an area efficient Overlay with linear interconnect
- Built using fully pipelined DSP blocks
- Architectural enhancement on the overlay
  - Rotating register file
  - Replicating the stream datapath
  - FU write-back support
- Along with a better instruction scheduling strategy
- Improvement (V3) compared to the Linear TM overlay [9]
  - 50.0% higher throughput in GOPS
  - 48.5% higher compute efficiency in MOPS/eSlice
  - 32.0% lower latency in ns

References

1. J. Coole and G. Stitt, “Intermediate fabrics: Virtual architectures for circuit portability and fast placement and routing,” in Proc. Int. Conf. Hardware/Software Codesign and Syst. Synthesis (CODES+ISSS), 2010, pp. 13–22.
2. 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 Proc. 18th Int. Symp. High Performance Comput. Archit. (HPCA), 2012, pp. 1–12.
3. A. K. Jain, S. A. Fahmy, and D. L. Maskell, “Efficient overlay architecture based on DSP blocks,” in Proc. 23rd Int. Symp. Field- Programmable Custom Comput. Mach. (FCCM), 2015, pp. 25–28.
4. A. K. Jain, X. Li, P. Singhai, D. L. Maskell, and S. A. Fahmy, “DeCO: a DSP block based FPGA accelerator overlay with low overhead interconnect,” in Proc. 24th Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM), 2016, pp. 1–8.
5. J. Gray, “GRVI-Phalanx: A massively parallel RISC-V FPGA accelerator,” in Proc. 24th Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM), 2016, pp. 17–20.
6. C. Kumar HB, P. Ravi, G. Modi, and N. Kapre, “120-core microAptiv MIPS Overlay for the Terasic DE5-NET FPGA board,” in Proc. 25th Int. Symp. Field Program. Gate Arrays (FPGA), 2017, pp. 141– 146.
7. C. Liu, H.-C. Ng, and H. K.-H. So, “QuickDough: a rapid FPGA loop accelerator design framework using soft CGRA overlay,” in Proc. Int. Conf. Field-Programmable Technol. (FPT), 2015, pp. 56–63.
8. K. Paul, C. Dash, and M. S. Moghaddam, “remorph: a runtime reconfigurable architecture,” in Proc. 15th Euromicro Conf. Digit. Syst. Design (DSD), 2012, pp. 26–33.
9. X. Li, A. Jain, D. Maskell, and S. A. Fahmy, “An area-efficient FPGA overlay using DSP block based time-multiplexed functional units,” in Proc. 2nd Int. Workshop on Overlay Archit. for FPGAs (OLAF), 2016.

Thank you!