QuickDough: A Rapid FPGA Loop Accelerator Design Framework Using Soft Coarse-Grained Reconfigurable Array Overlays

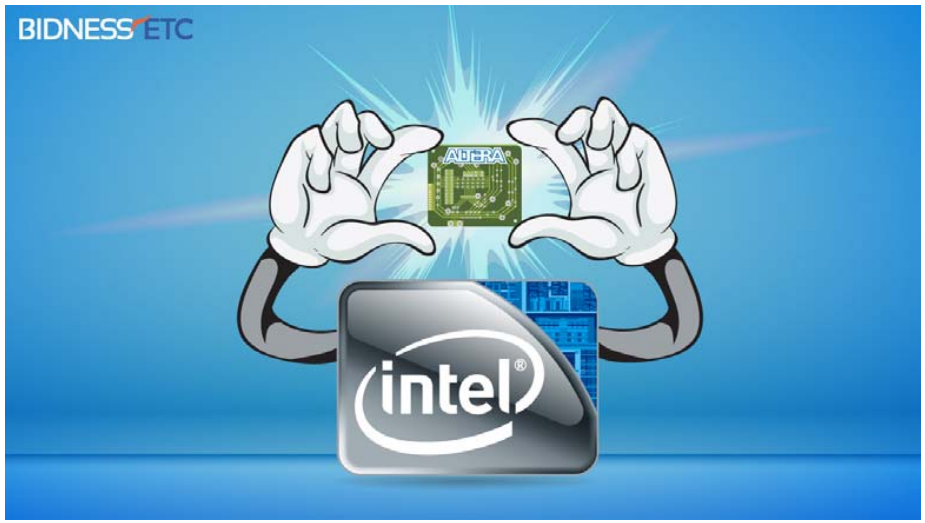
Cheng Liu

Department of Electrical and Electronic Engineering
The University of Hong Kong

Outline

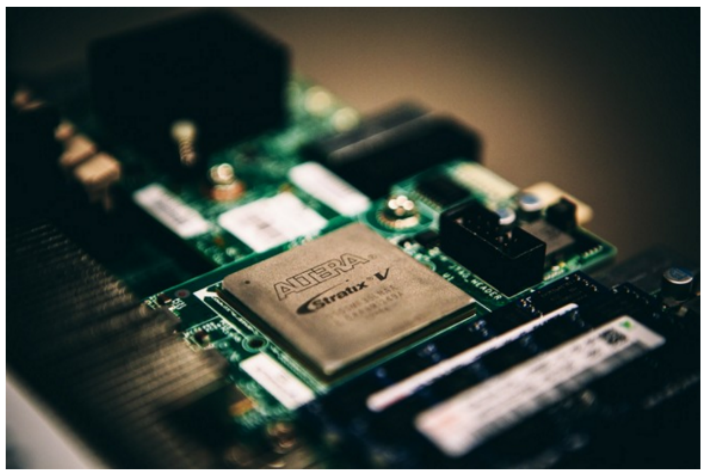
- Previous Research
  - QuickDough
  - Network-on-Chip
- Future Plans
  - Machine-Learning Based Domain Specific Computing Architecture
  - Imaging Algorithms Acceleration

FPGA News



[1]

MICROSOFT SUPERCHARGES BING SEARCH WITH PROGRAMMABLE CHIPS

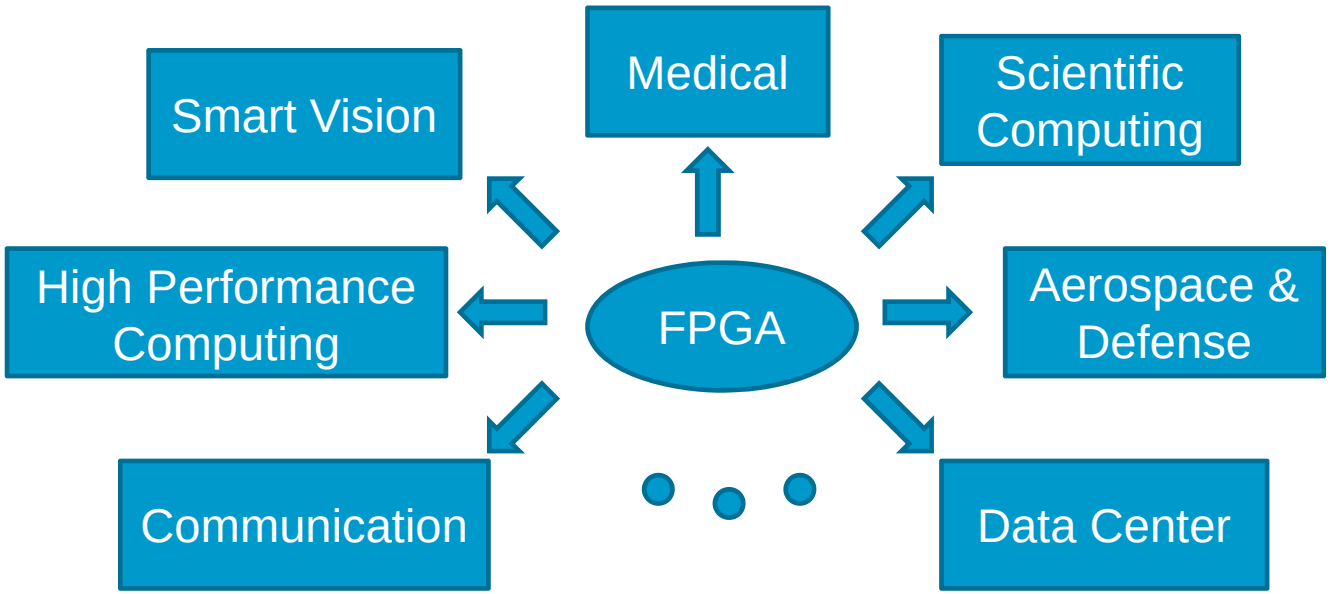


[2]

[1] http://www.bidnessetc.com/38354-jpmorgan-outlines-impact-of-intel-corporation-intc-acquisition-of-altera-co/
[2] http://www.wired.com/2014/06/microsoft-fpga/

Advantages of Using FPGAs

Successful demonstrations of using FPGA as hardware accelerators can be found in multitude of disciplines.

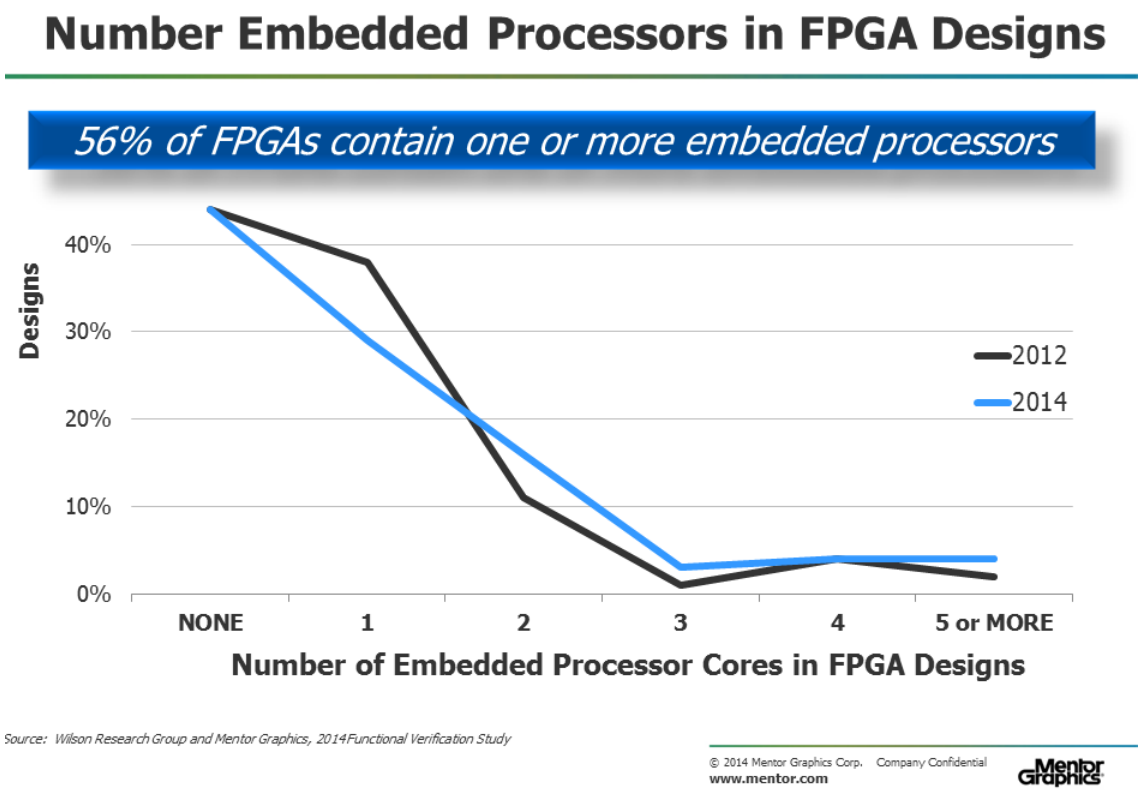


General advantages of using FPGAs:

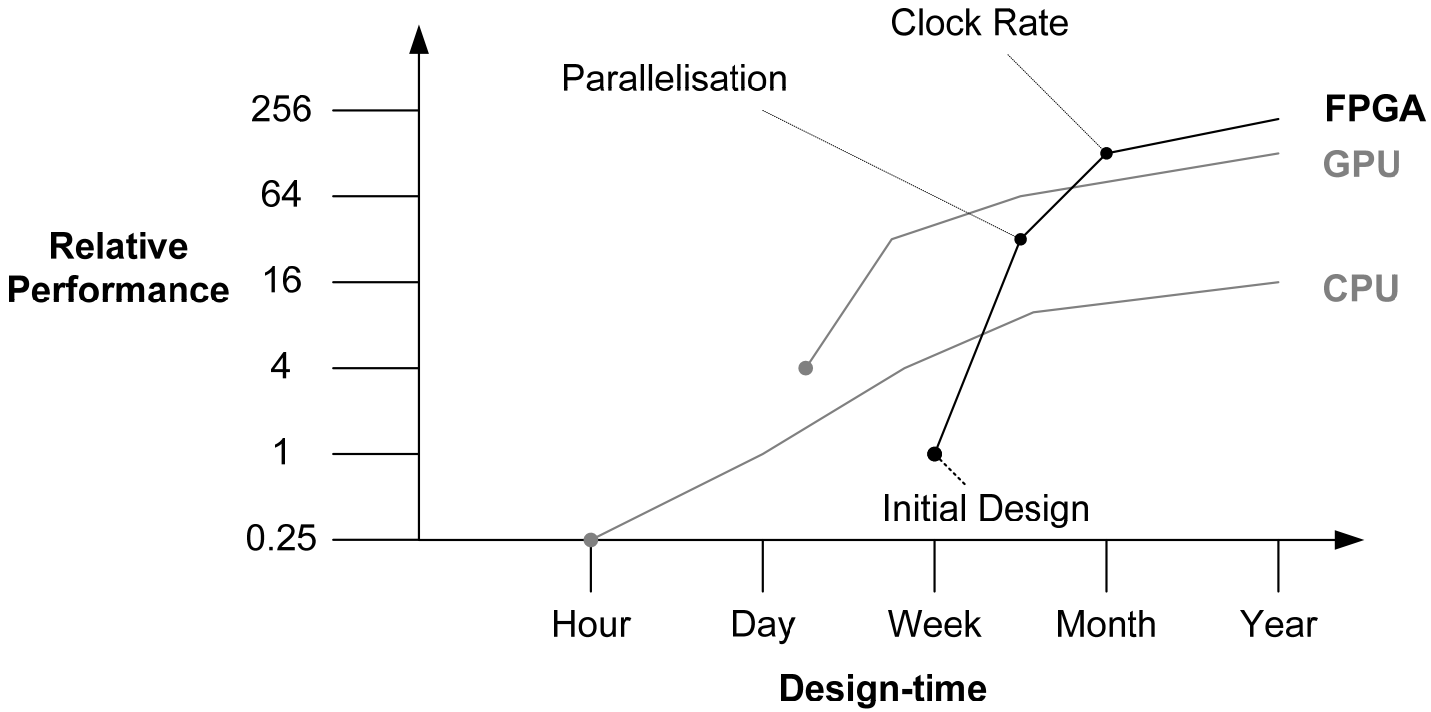
- **High energy efficiency**
- **Low latency, real-time processing**
- **Competitive performance**
- **...**

Hybrid CPU-FPGA Computing System

- FPGA: deep pipelining, massive fine-grained parallelism, fixed data flow graph, weird bit manipulation, low-latency real-time processing, ...
- CPU: complex software environment such as OS, GUI, ...

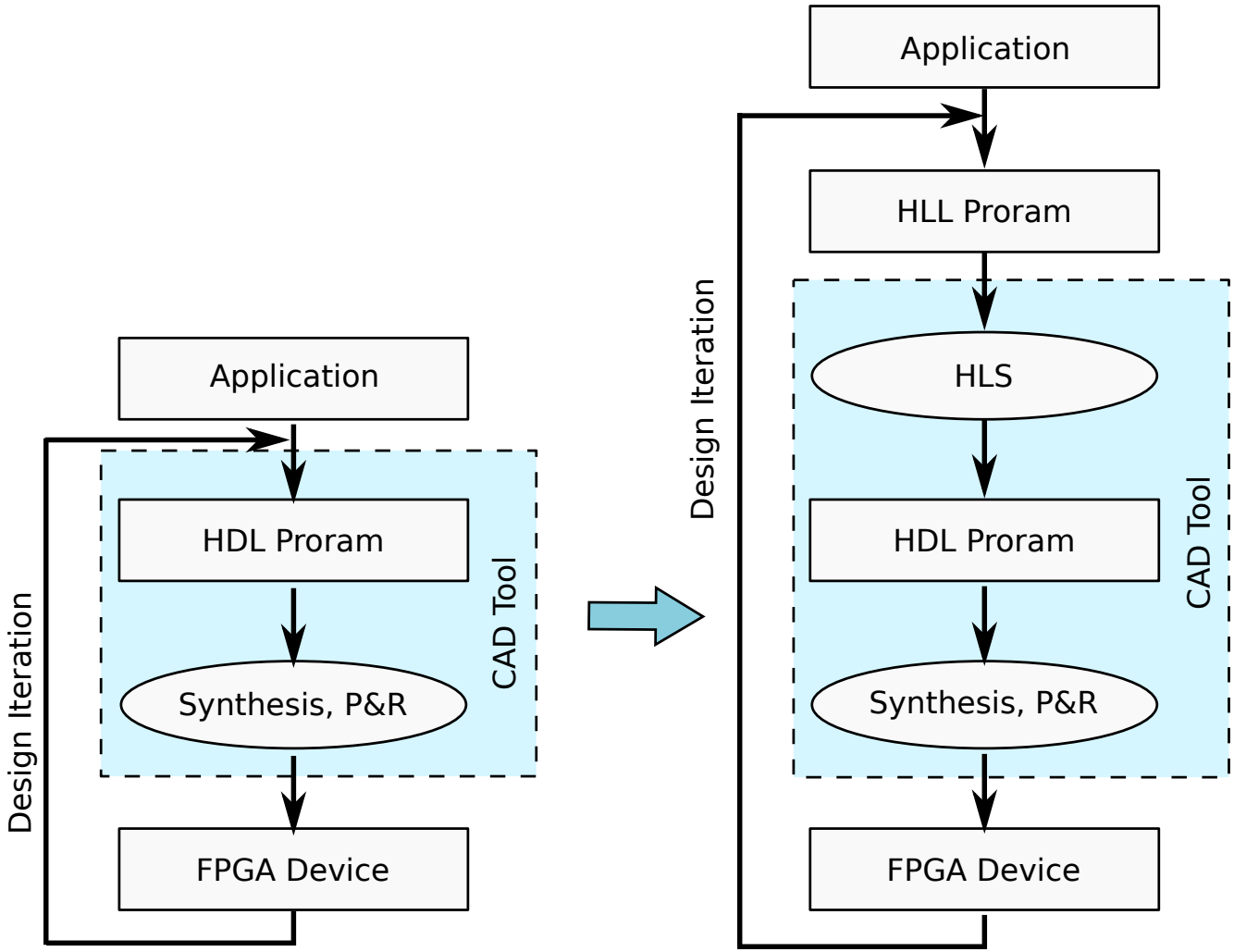


Design Productivity Challenge

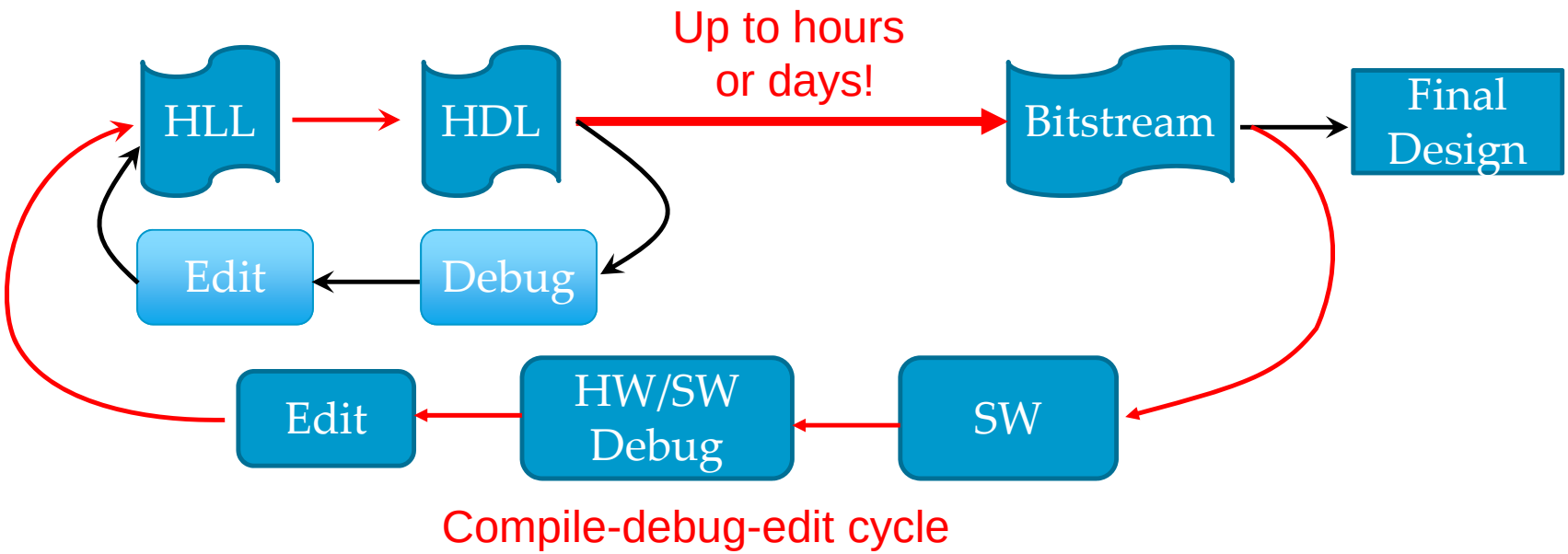


- FPGAs provide large speed-up and power savings – at a price!
  - Days or weeks to get an initial version working
  - Multiple optimisation and verification cycles to get high performance
- Too risky and too specialised for most users
  - Months of retraining for an uncertain speed-up

High Level Synthesis



FPGA Accelerator Design Using HLS Tools



Multiple levels of compile-debug-edit iterations are needed to develop a satisfying FPGA accelerator. The implementation process remains critical to the overall productivity.

Approaches to Improve the Design Productivity

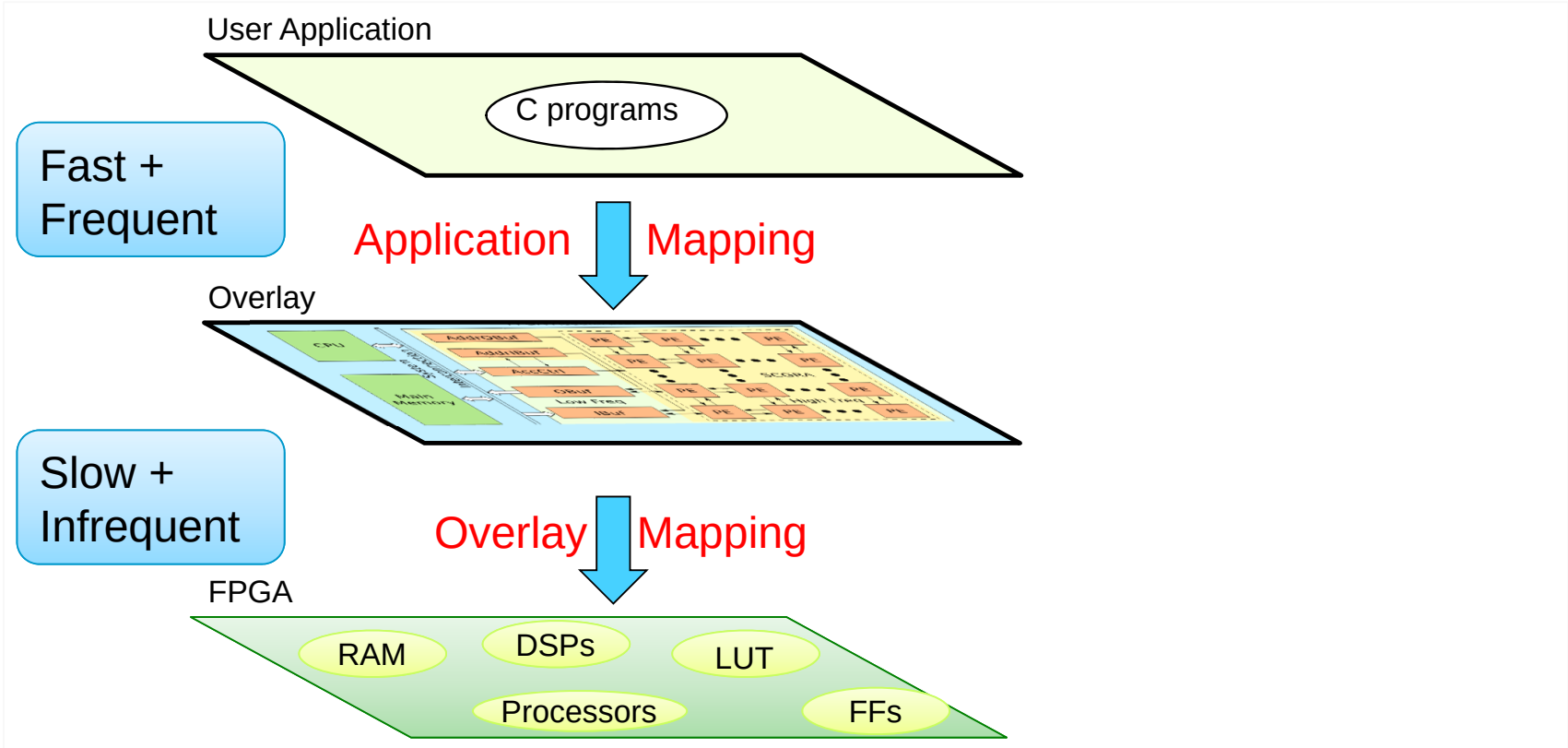
The community have tried to approach this problem from various angles:

- **Raise the abstraction level (HLS tools, LegUp, Vivado HLS, ...)**
- Provide HW/SW run-time support (BORPH, ReconOS, ...)
- **Reduce the FPGA implementation time (partial reconfiguration, hard macro, implementation algorithms, ...)**
- Provide FPGA debugging facilities
- **Automatic design space exploration**
- **...**

FPGA design productivity is greatly improved, but remains much lower compared to software development.

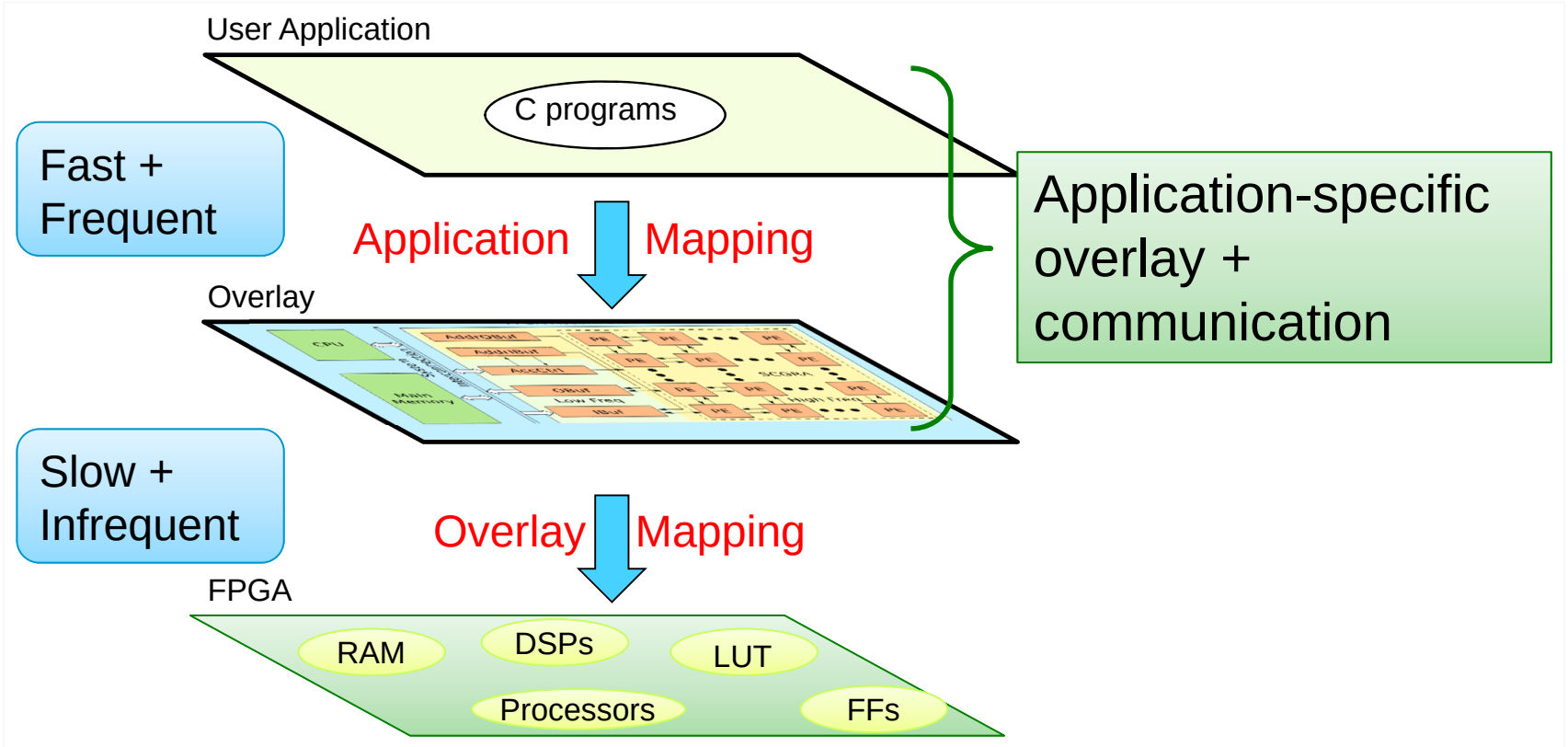
Overlay Architecture

A two-step HW/SW compilation approach:



Overlay Architecture

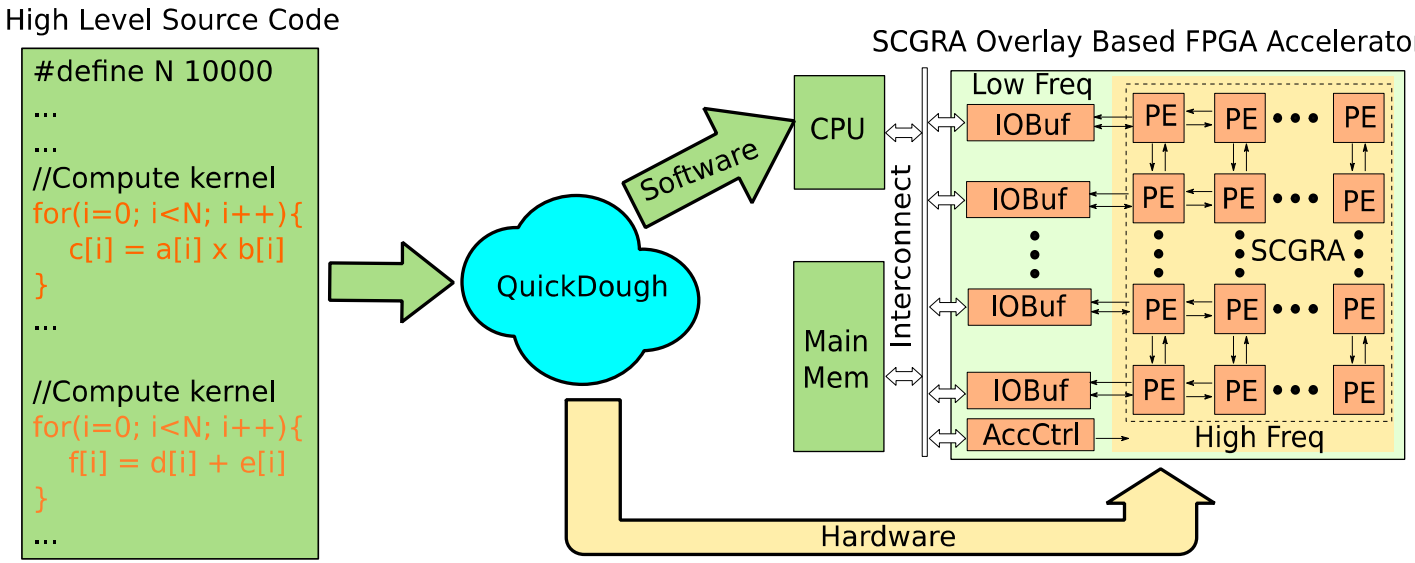
A two-step HW/SW compilation approach:



QuickDough Overview

A high productivity compilation framework for high-level applications on CPU-FPGA computers.

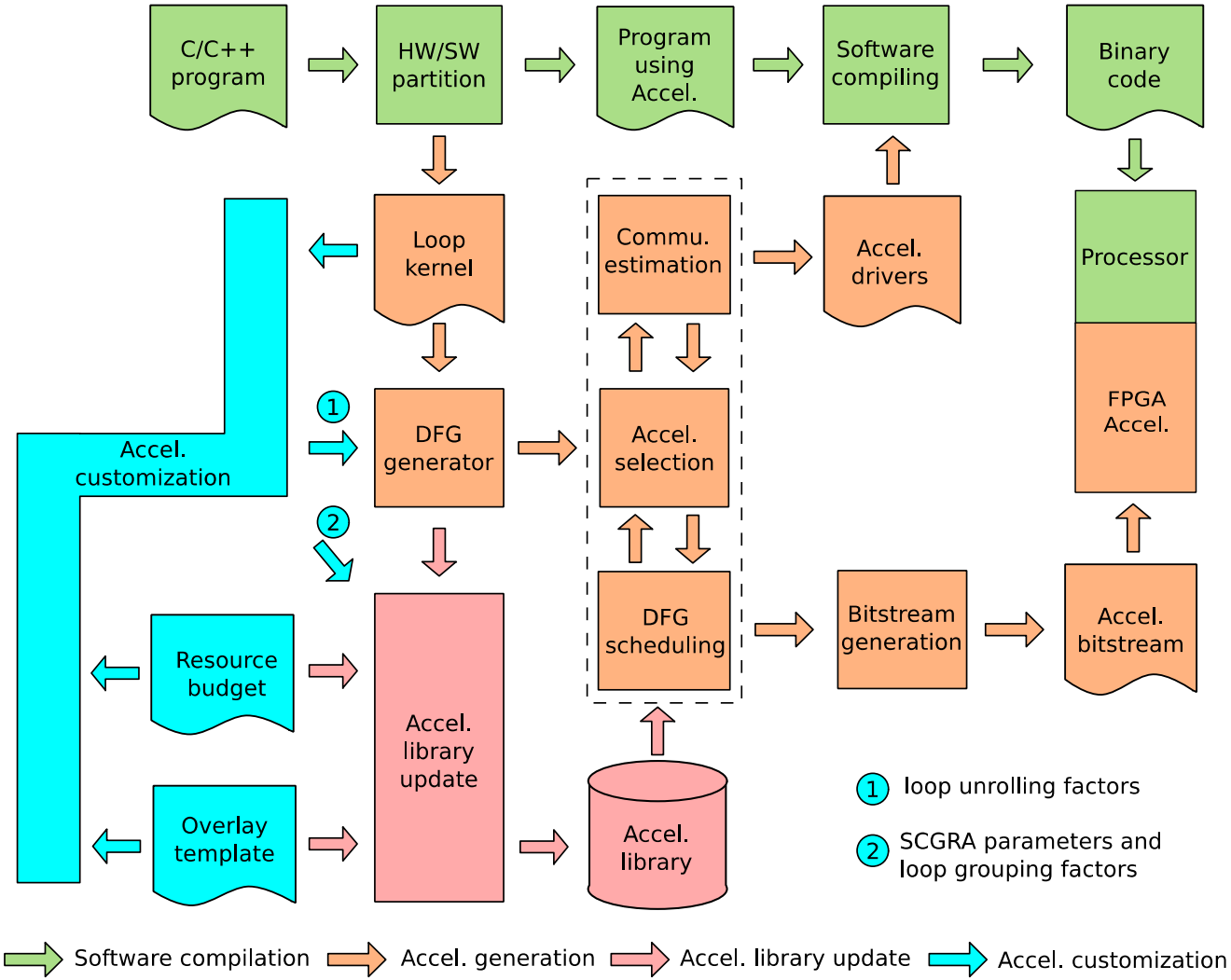
- Compiles software and FPGA accelerator design within the same framework
- Overall SW-like compile speed



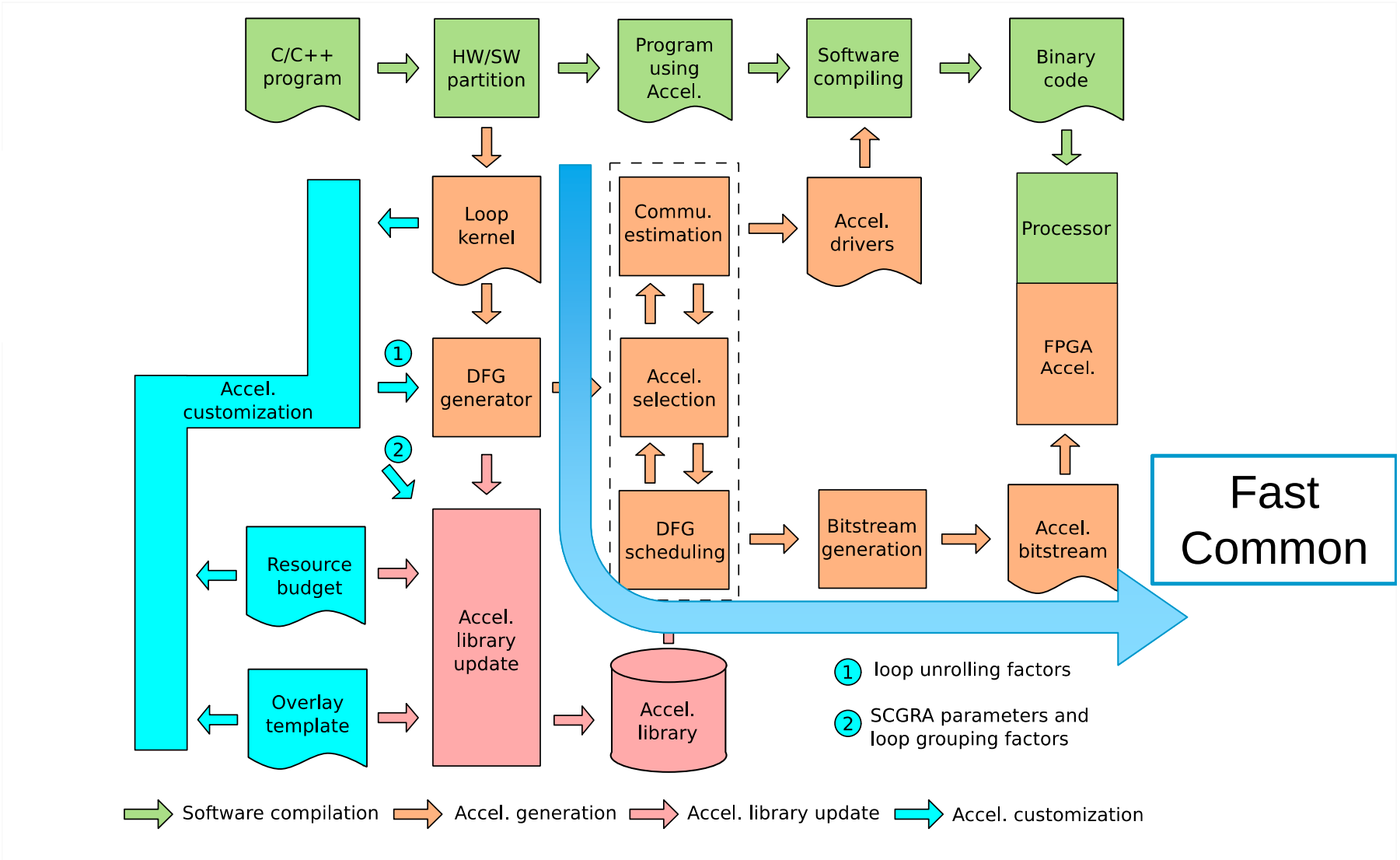
Goals:

- Fast design iterations
- Good performance

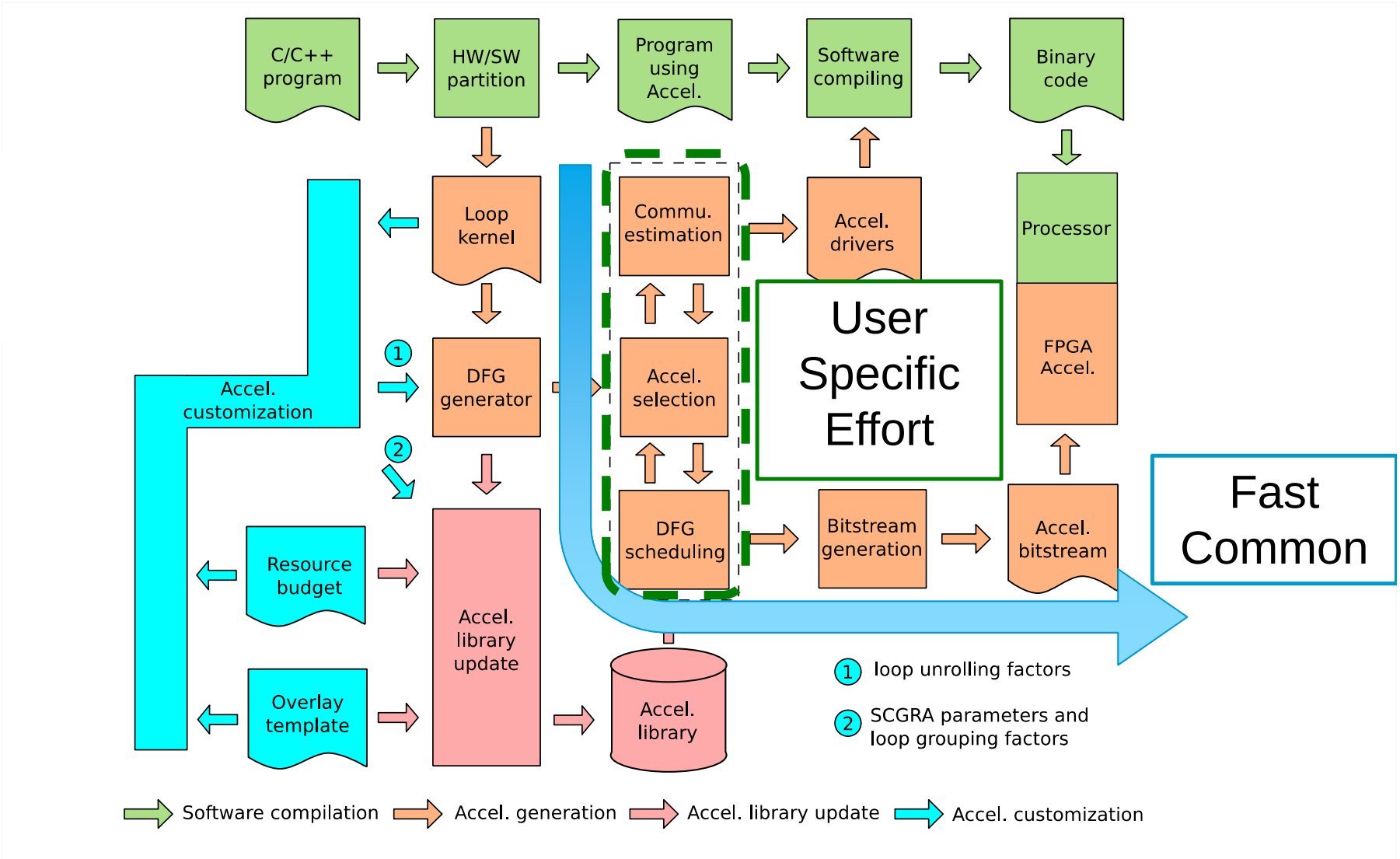
QuickDough Design Flow



QuickDough Design Flow

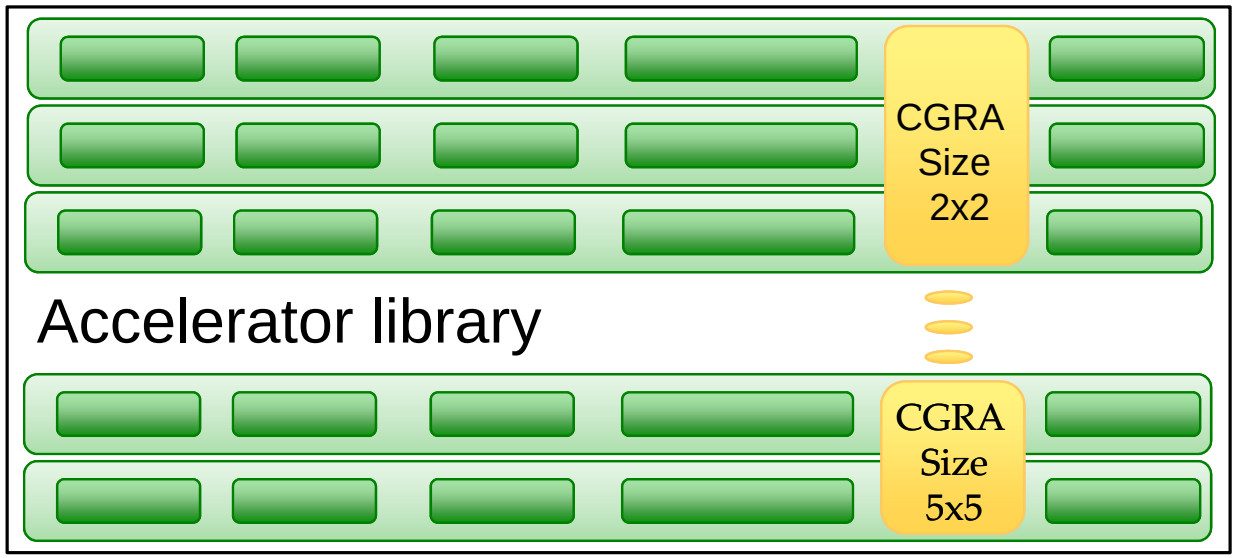


QuickDough Design Flow



Accelerator Selection

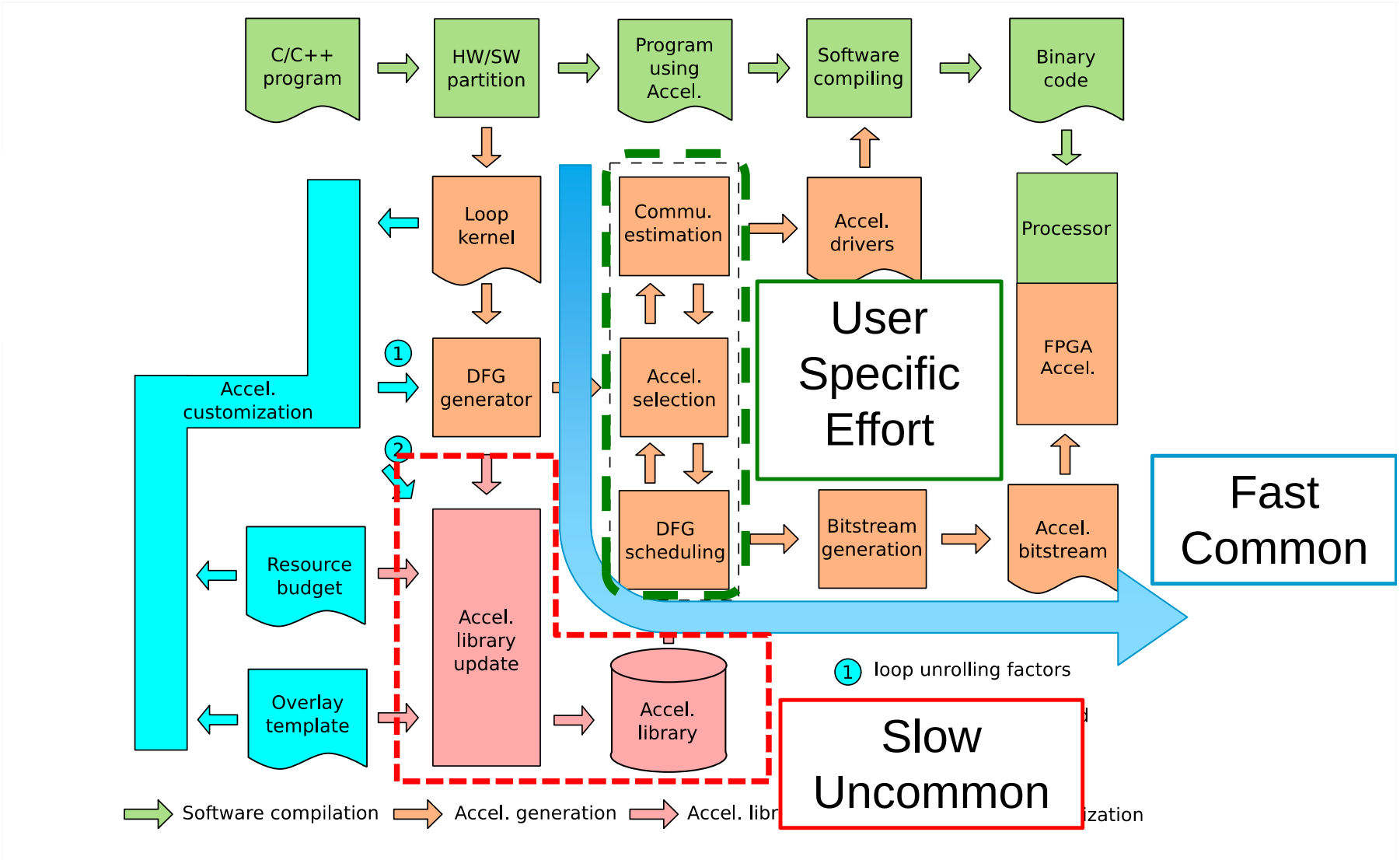
The accelerator library can be organized, so that the time consuming operation scheduling process can be reused to evaluate a group of configurations in the library.



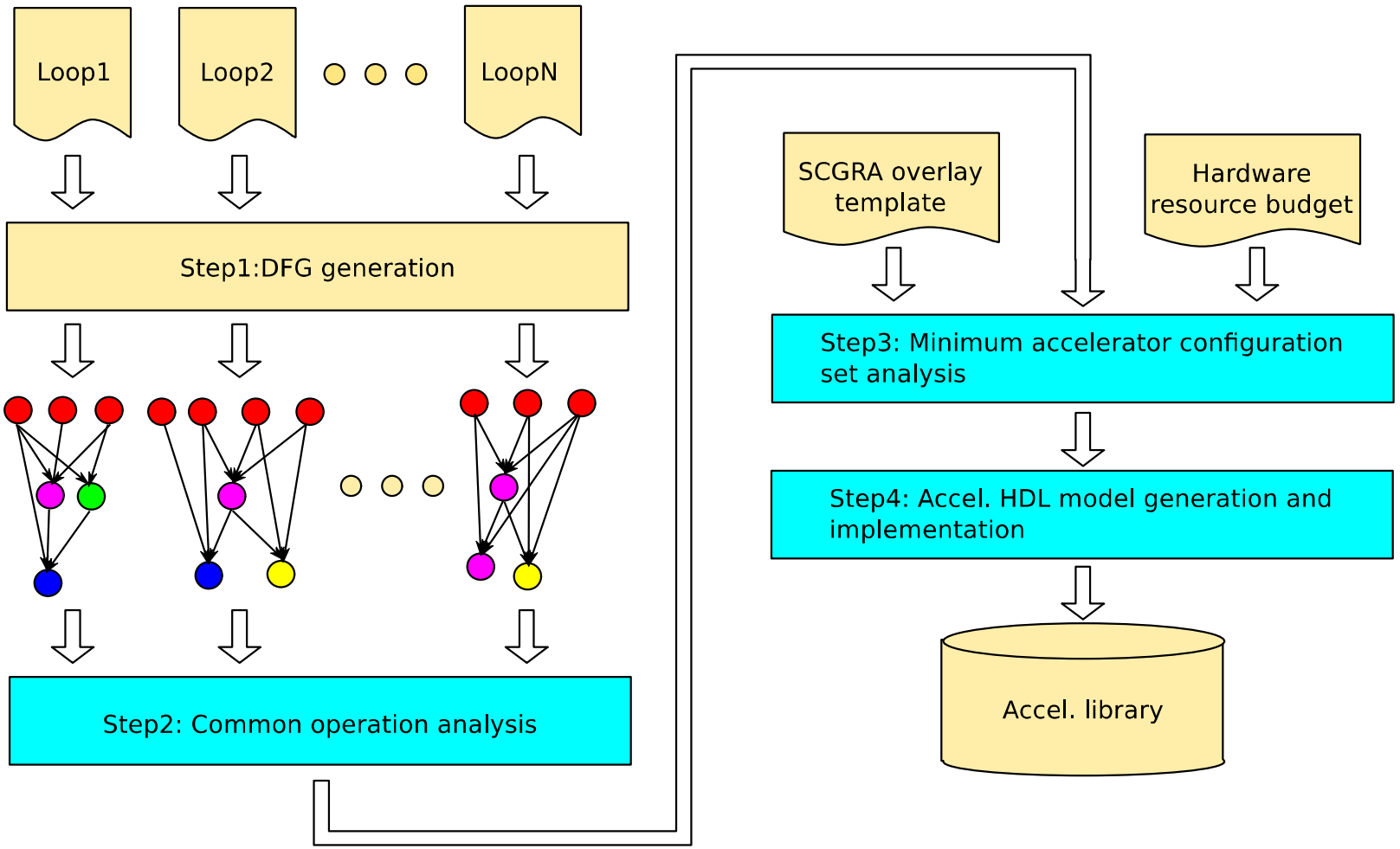
User specified accelerator selection strategy:

- -O1 select the accelerator with smallest SCGRA size
- -O2 target three groups of accelerators with different SCGRA size (small, medium, large)
- -O3 full search

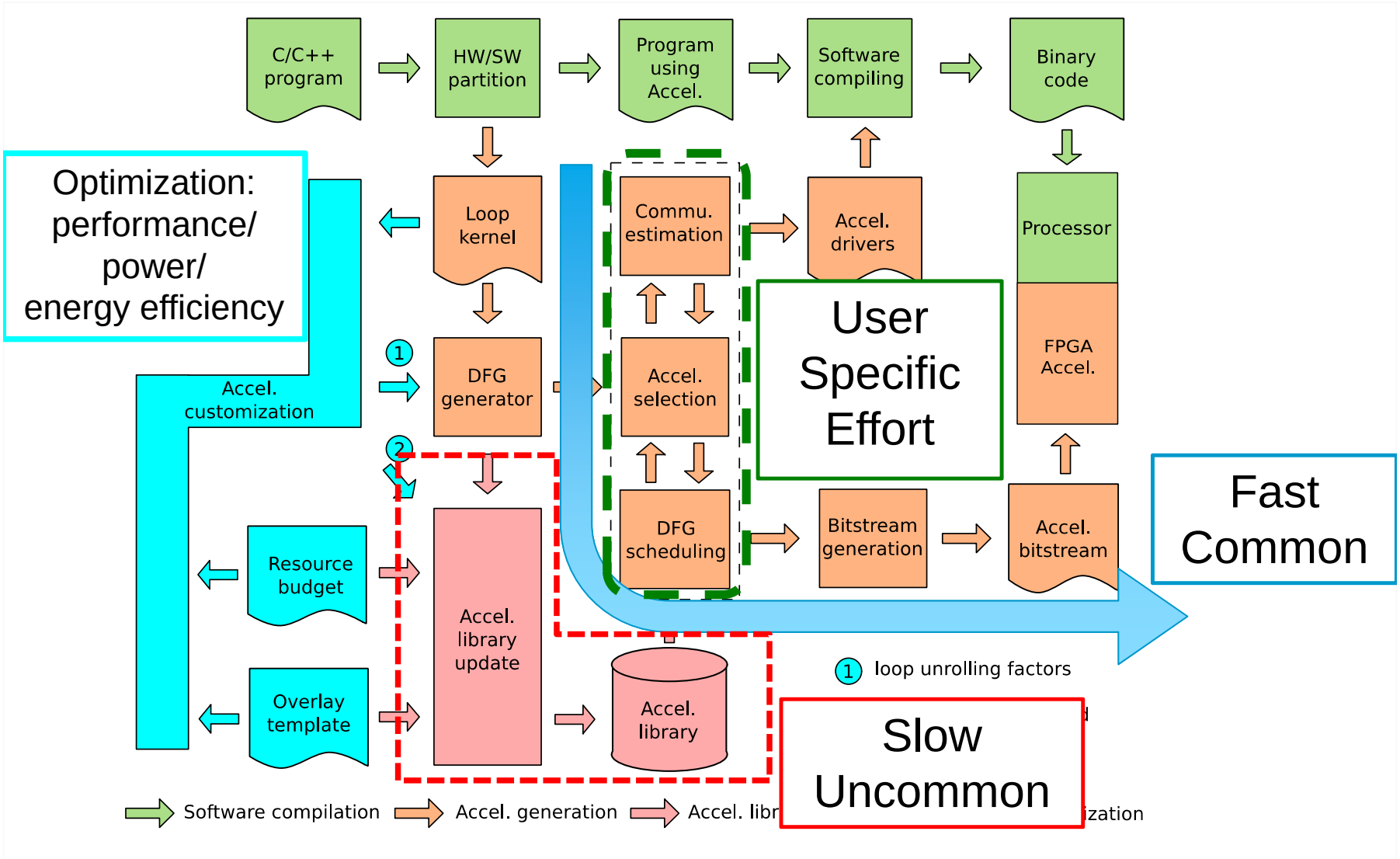
QuickDough Design Flow



Accelerator Library Pre-build & Update



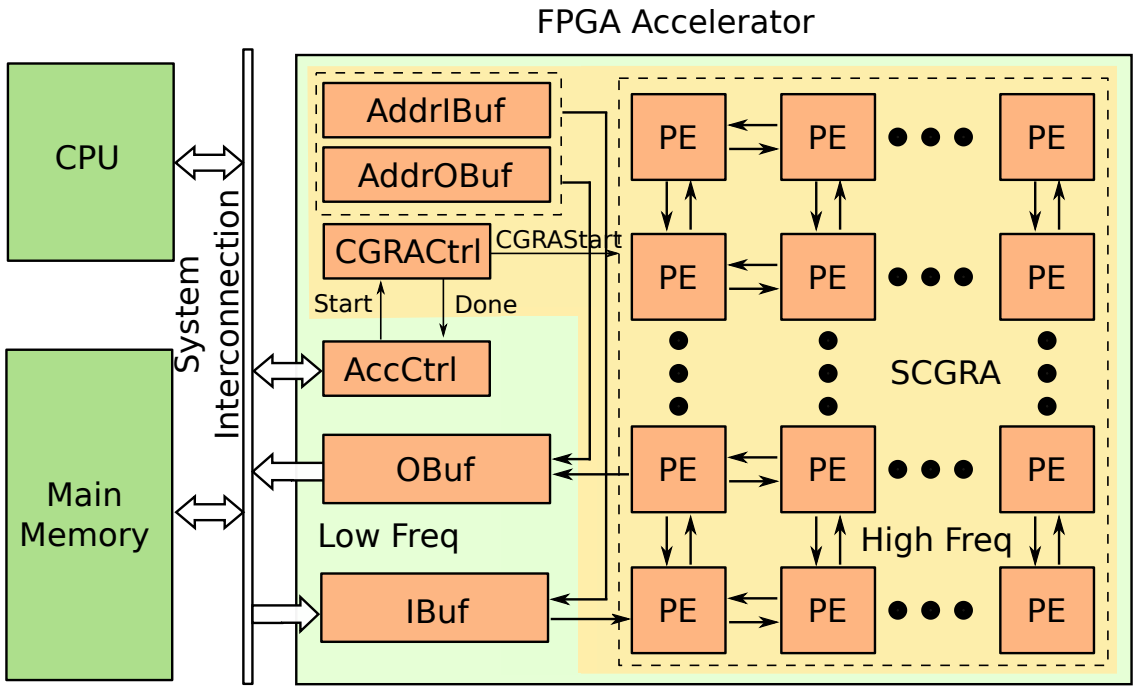
QuickDough Design Flow



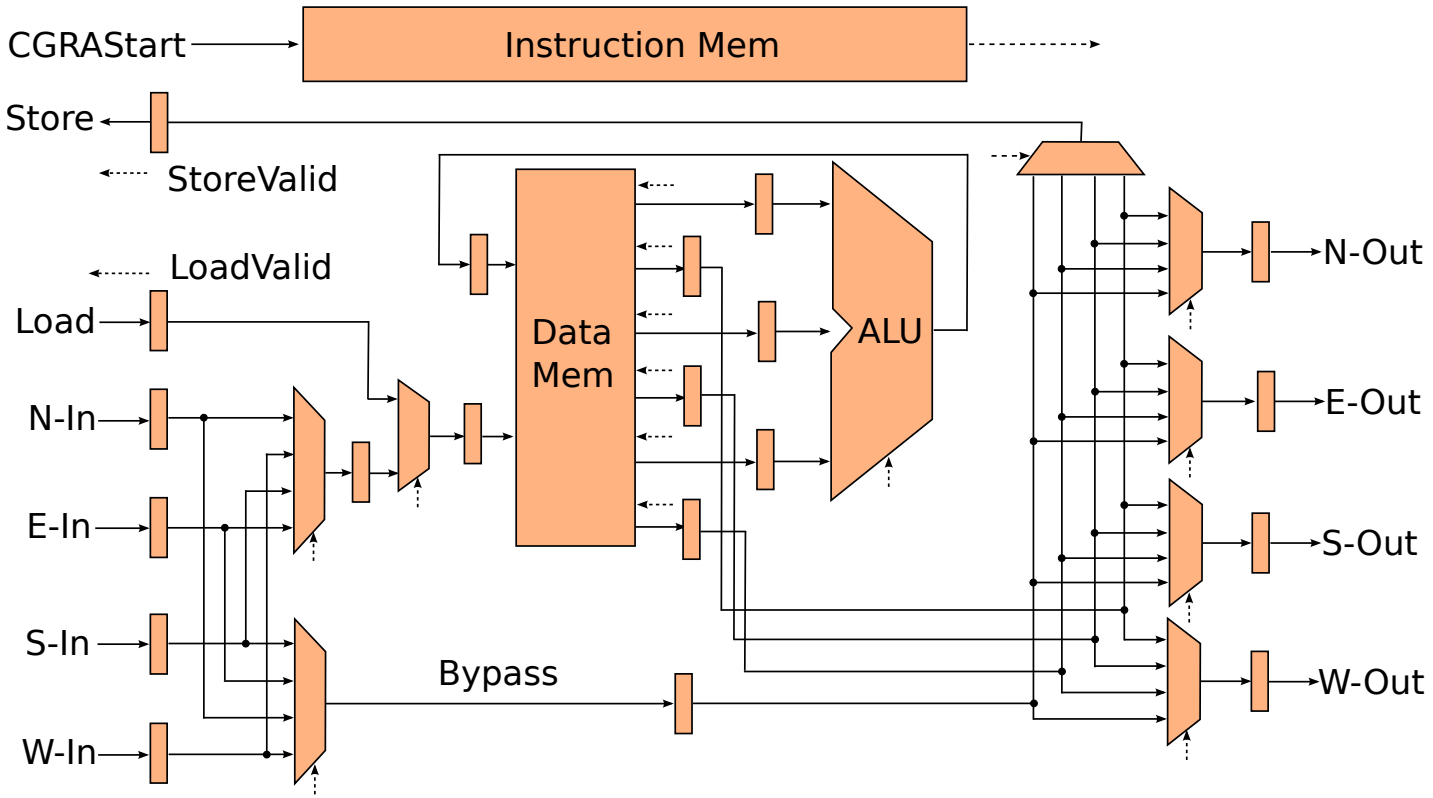
Proposed Soft CGRA Overlay

Design principles:

- Highly pipelined –> performance acceleration
- Reconfigurable –> Application-specific optimization
- Simple and easy to extend
- Make best use of the underlying FPGA resources

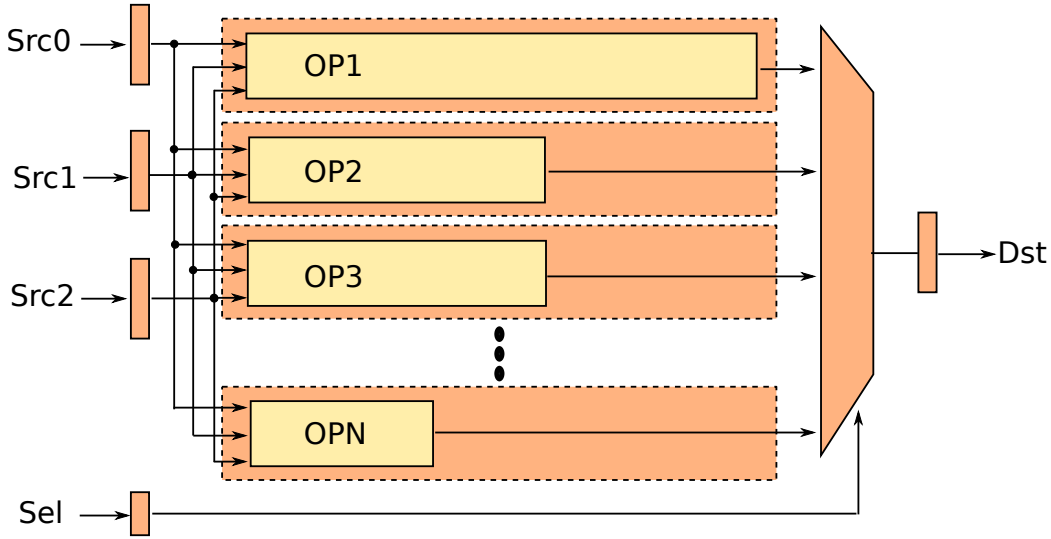


Processing Element (PE)



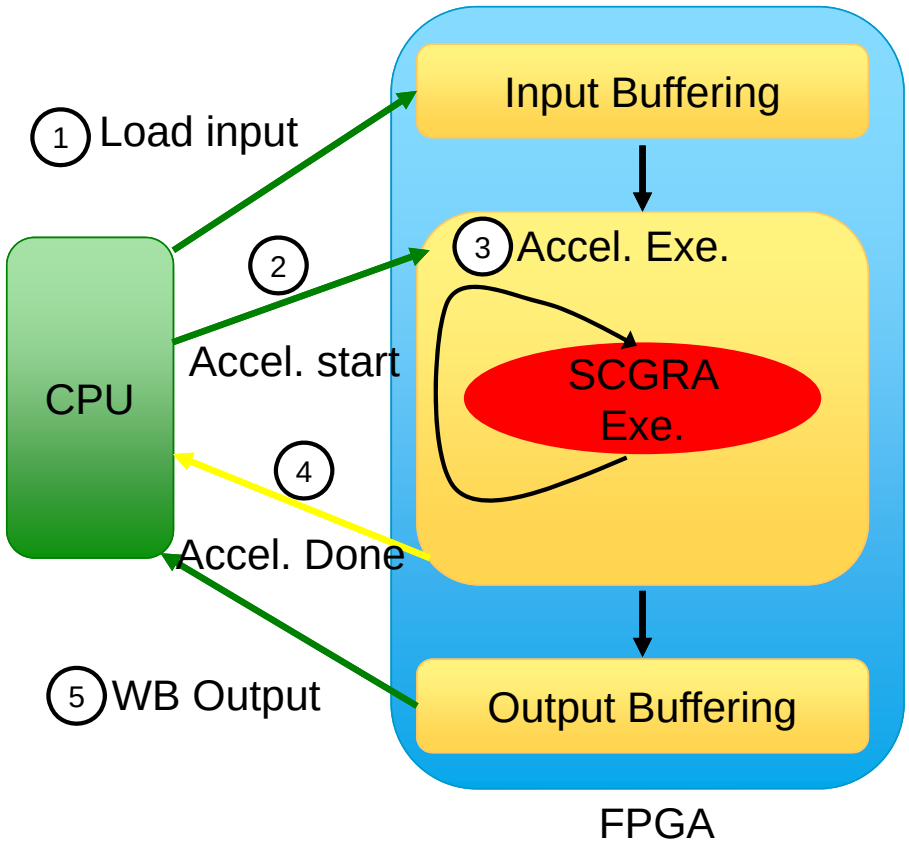
When there are sufficient parallel operations in the data memory, the PE can perform 1 operation per cycle ideally while bypassing one data to the neighboring PE at the same time.

ALU



Type	Opcode	Expression
MULADD	0001	Dst = Src0 x Src1 + Src2
MULSUB	0010	Dst = Src0 x Src1 - Src2
ADDADD	0011	Dst = Src0 + Src1 + Src2
ADDSUB	0100	Dst = Src0 + Src1 - Src2
SUBSUB	0101	Dst = Src0 - Src1 - Src2
PHI	0110	Dst = Src0 ? Src1 : Src2
RSFAND	0111	Dst = (Src0 >> Src1) & Src2
LSFADD	1000	Dst = (Src0 << Src1) + Src2
ABS	1001	Dst = ABS(Src0)
GT	1010	Dst = (Src0 > Src1) ? 1 : 0
LET	1011	Dst = (Src0 <= Src1)? 1 : 0
ANDAND	1100	Dst = (Src0 & Src1) & Src2

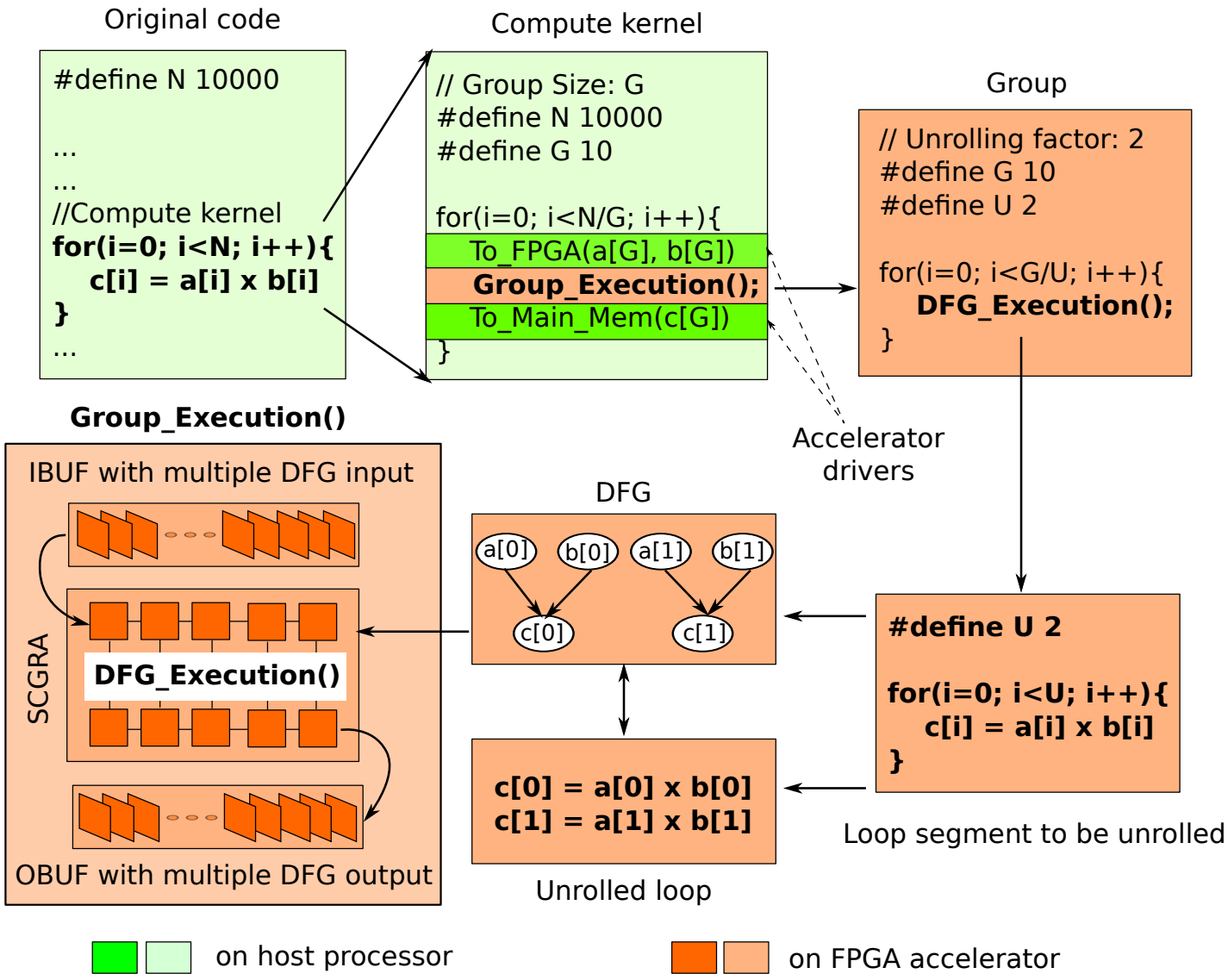
Accelerator Controller and CGRA Controller



Accelerator Controller exhibits a standard interface to high level software.

SCGRA Controller generates

Loop Execution on the Accelerator



Outline

- Background & Motivation
- Related Work
- QuickDough Design Framework
- SCGRA Overlay Design & Implementation
- **FPGA Loop Accelerator Customization**
- Experiments
  - SCGRA Overlay Architecture
  - Loop Accelerator Generation
  - Loop Accelerator Customization
- Conclusion

Automatic Loop Accelerator Customization

Why customization?

- difficult for high-level designers to decide the parameters
- critical to achieve good **performance & energy efficiency**
- ...

What to customize?

	Parameters	**Supported?**
Compiling options	Loop grouping	Y
	Loop unrolling	Y
Overlay architecture parameters	SCGRA Size (Row * Col)	Y
	SCGRA topology	Torus
	Input buffer depth	Y
	Output buffer depth	Y
	Input address buffer depth	Y
	Output address buffer depth	Y
	Data memory depth	Y
	Instruction memory depth	Y
	Pipeline depth	Fixed
	Operation set	As needed

Loop Accelerator Modeling

$$Energy = LoopRuntime \times Power(AccelConfig)$$
$$EnergyDelayProduct = LoopRuntime \times Energy$$

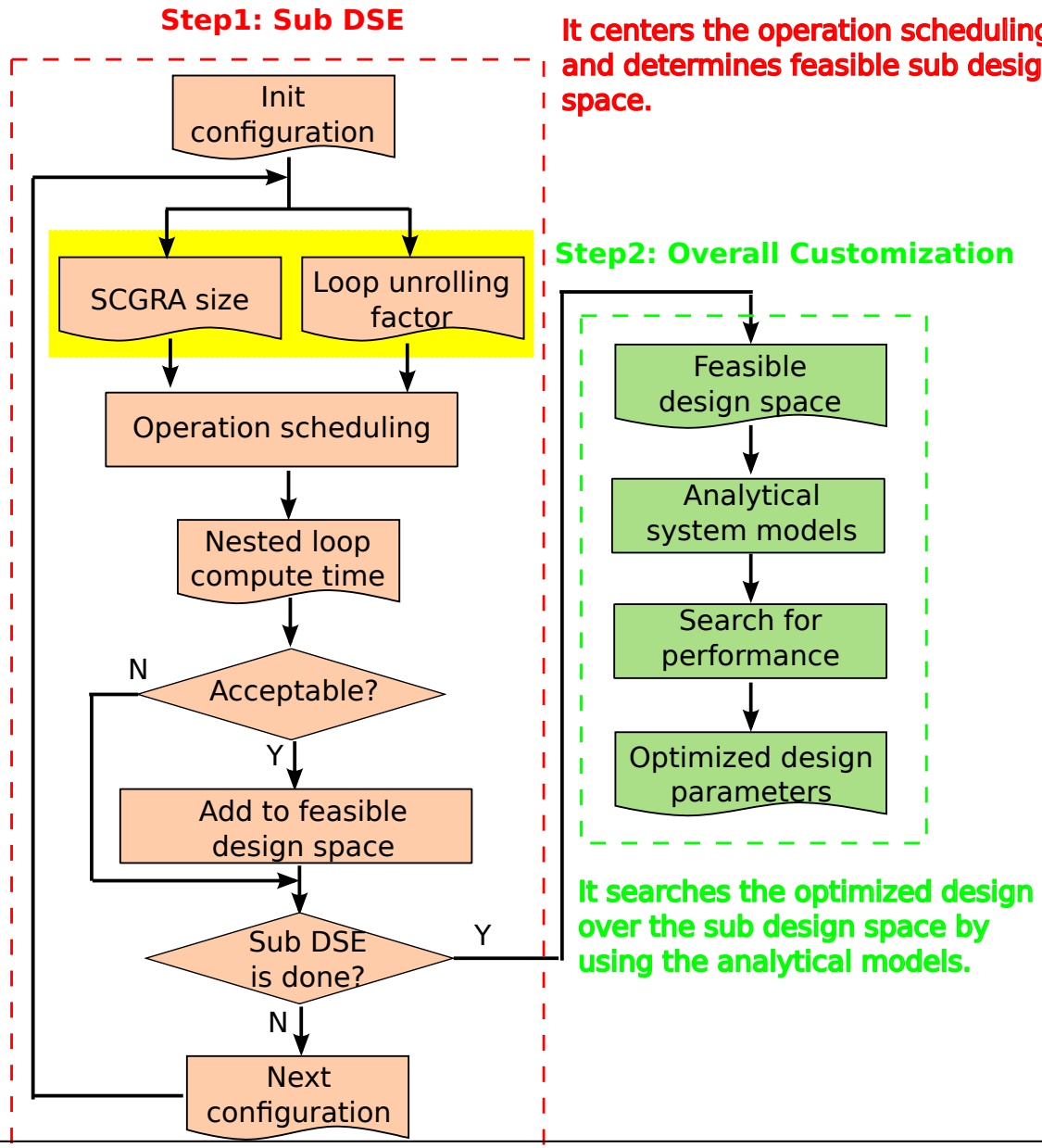
$$LoopRuntime = CommuTime + CompuTime$$
$$CommuTime = \big(DMALat(GroupIn) + DMALat(GroupOut)\big) \times GroupPerLoop$$
$$CompuTime = GroupPerLoop \times DFGPerGroup \times DFGLat \times ImplFreq$$

$$[\boldsymbol{DFGLat, DataMem, InstMem}] = \boldsymbol{Scheduling(Row, Col, Unrolling)}$$

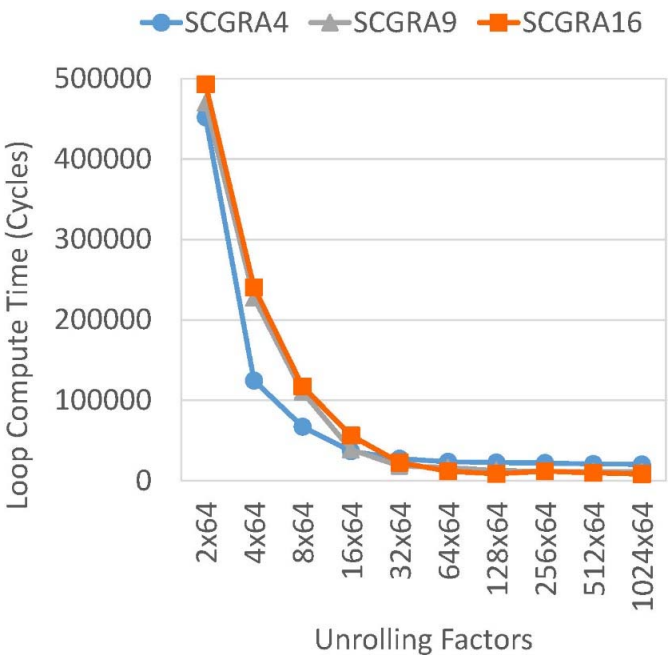
$$InBuffer \geq GroupIn, OutBuffer \geq GroupOut$$
$$InAddrBuffer \geq DFGIn \times DFGPerGroup$$
$$OutAddrBuffer \geq DFGIn \times DFGPerGroup$$
$$Resource(DataMem, InstMem, InBuffer, OutBuffer, InAddrBuffer, OutAddrBuffer, Row, Col) \leq ResourceBudget$$

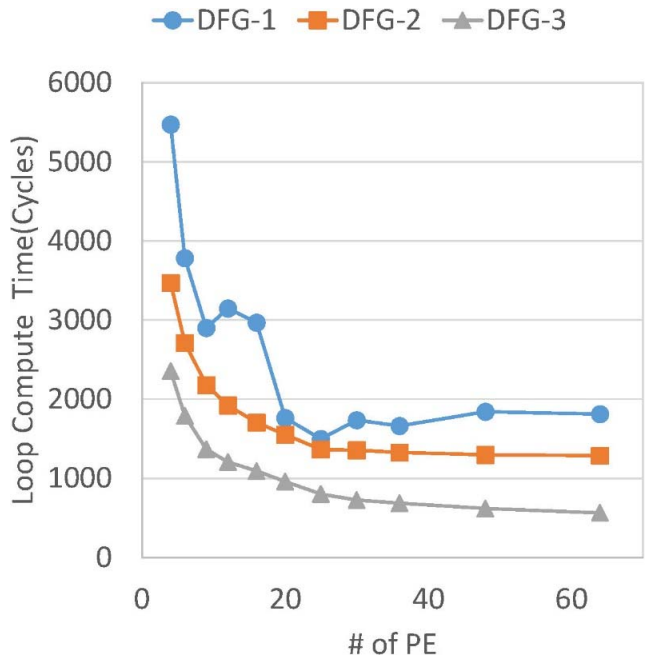
Most parameters can be modeled easily.

Two-step Customization Strategy



Sub Design Space Exploration





Both unrolling and SCGRA size have clear monotonic influence on the performance. Thus

$$\frac{Runtime(Unrolling1) - Runtime(Unrolling2)}{Runtime(Unrolling1)} < \varepsilon$$

can be used as Sub DSE unrolling search metric. SCGRA size can be analyzed similarly.

Outline

- Background & Motivation
- Related Work
- QuickDough Design Framework
- SCGRA Overlay Architecture
- FPGA Loop Accelerator Customization
- **Experiments**
  - Loop Accelerator Generation
  - Loop Accelerator Customization
- Conclusion

Benchmark & Experiment Setup

Benchmark

Benchmark	Kernel Parameters	Loop kernel
MM (Matrix multiplication)	matrix size: 100×100	**100** × **100** ×100
FIR	# of input: 10000 # of Taps +1: 50	**10000** × 50
SE (Sobel edge detector)	# of vertical pixels:128 # of horizontal pixels: 128	**128** × **128** × 3 × 3
KM (Kmean)	# of nodes: 5000 # of centroids: 4 # of node dimension: 2	**5000** × 4 × 2

Basic SCGRA Configurations

SCGRA Topology	2D Torus
Instruction Memory Data Width	72 bits
Data Memory	256 x 32 bits
I/O Buffer Data Width	32 bits
I/O Addr Buffer Data Width	18 bits

DMA Latency

DMA Size	**≥512**	**256**	**128**	**64**	**32**	**16**	**≤8**
Latency/Word (ns)	**10.08**	**11.28**	**13.32**	**15.18**	**21.45**	**36.24**	**63.00**

Outline

- Background & Motivation
- Related Work
- QuickDough Design Framework
- SCGRA Overlay Design & Implementation
- FPGA Loop Accelerator Customization
- **Experiments**
  - **Loop Accelerator Generation**
  - Loop Accelerator Customization
- Conclusion

Experiment Setup

Specified Loop Unrolling

Benchmark	Unrolling	DFG Size
MM	1× 5 ×100	750
FIR	50×50	2500
SE	16× 16 × 3 ×3	9720
KM	125 × 4 ×2	5768

Resource Constraints

RAMB36	140
DSP48	220
LUT	53200
FF	53200

Supported Operation Set

Type	Opcode	Expression
MULADD	0001	Dst = Src0 x Src1 + Src2
MULSUB	0010	Dst = Src0 x Src1 - Src2
ADDADD	0011	Dst = Src0 + Src1 + Src2
ADDSUB	0100	Dst = Src0 + Src1 - Src2
SUBSUB	0101	Dst = Src0 - Src1 - Src2
PHI	0110	Dst = Src0 ? Src1 : Src2
RSFAND	0111	Dst = (Src0 >> Src1) & Src2
LSFADD	1000	Dst = (Src0 << Src1) + Src2
ABS	1001	Dst = ABS(Src0)
GT	1010	Dst = (Src0 > Src1) ? 1 : 0
LET	1011	Dst = (Src0 <= Src1)? 1 : 0
ANDAND	1100	Dst = (Src0 & Src1) & Src2

Resulting Accelerator Configurations

Opt. Option	Resulting Config	MM	FIR	SE	KM
O0	SCGRA size	2x2	2x2	2x2	2x2
	Inst. Mem Depth	4K	4K	4K	4K
	I/O Buffer Depth	4K	4K	4K	4K
	Grouping Factor	50x5x100	2500x50	128x64x3x3	1250x4x2
O1	SCGRA Size	3x3	3x3	3x3	5x5
	Inst. Mem Depth	2K	2K	4K	1K
	I/O Buffer Depth	2K	2K	1K	2K
	Grouping Factor	25x5x100	1250x50	64x32x3x3	1000x4x2
O2	SCGRA size	3x3	4x4	4x4	5x5
	Inst. Mem Depth	2K	1K	2K	1K
	I/O Buffer Depth	2K	8K	1K	2K
	Grouping Factor	25x5x100	5000x50	64x32x3x3	1000x4x2

Performance of the Resulting Accelerators

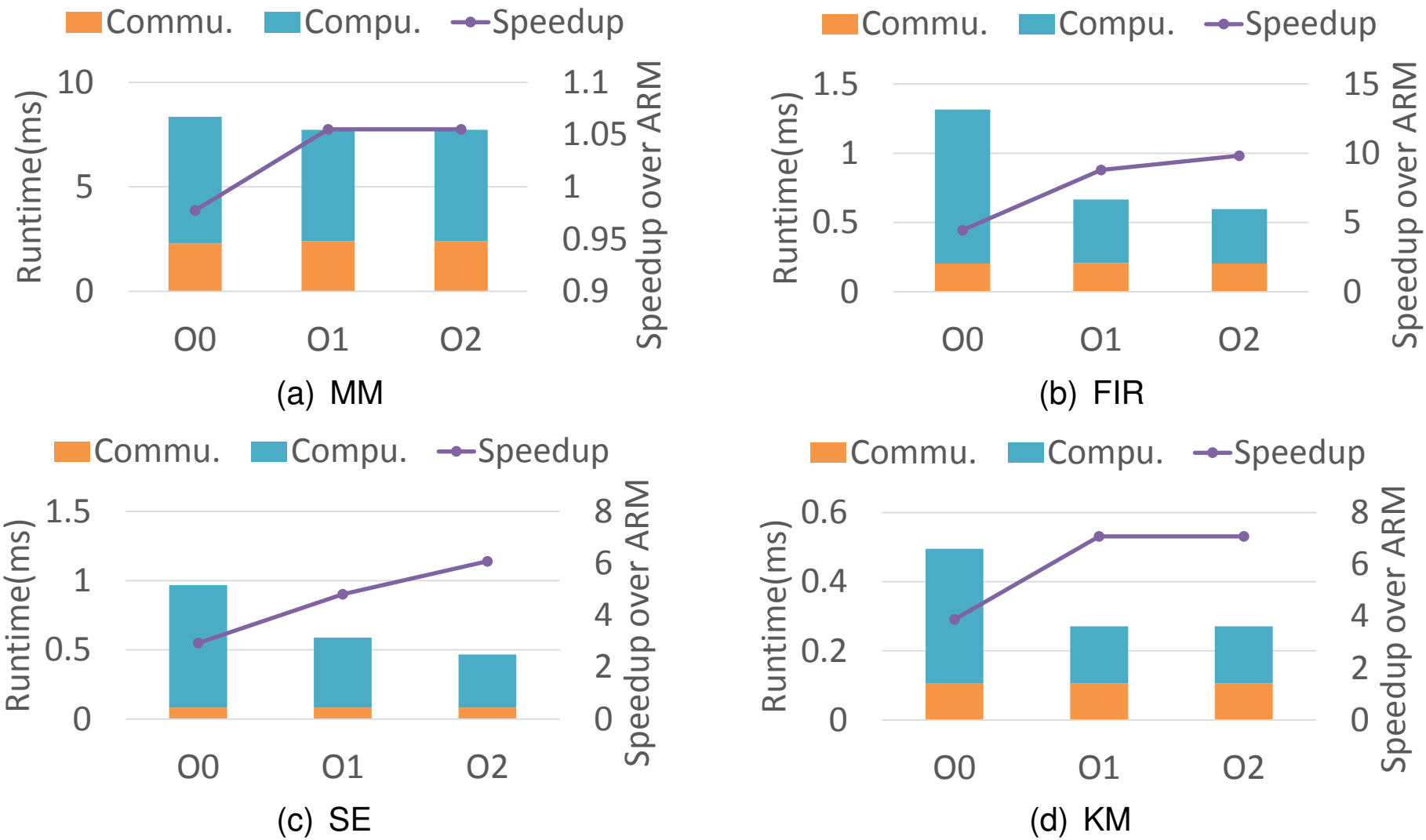
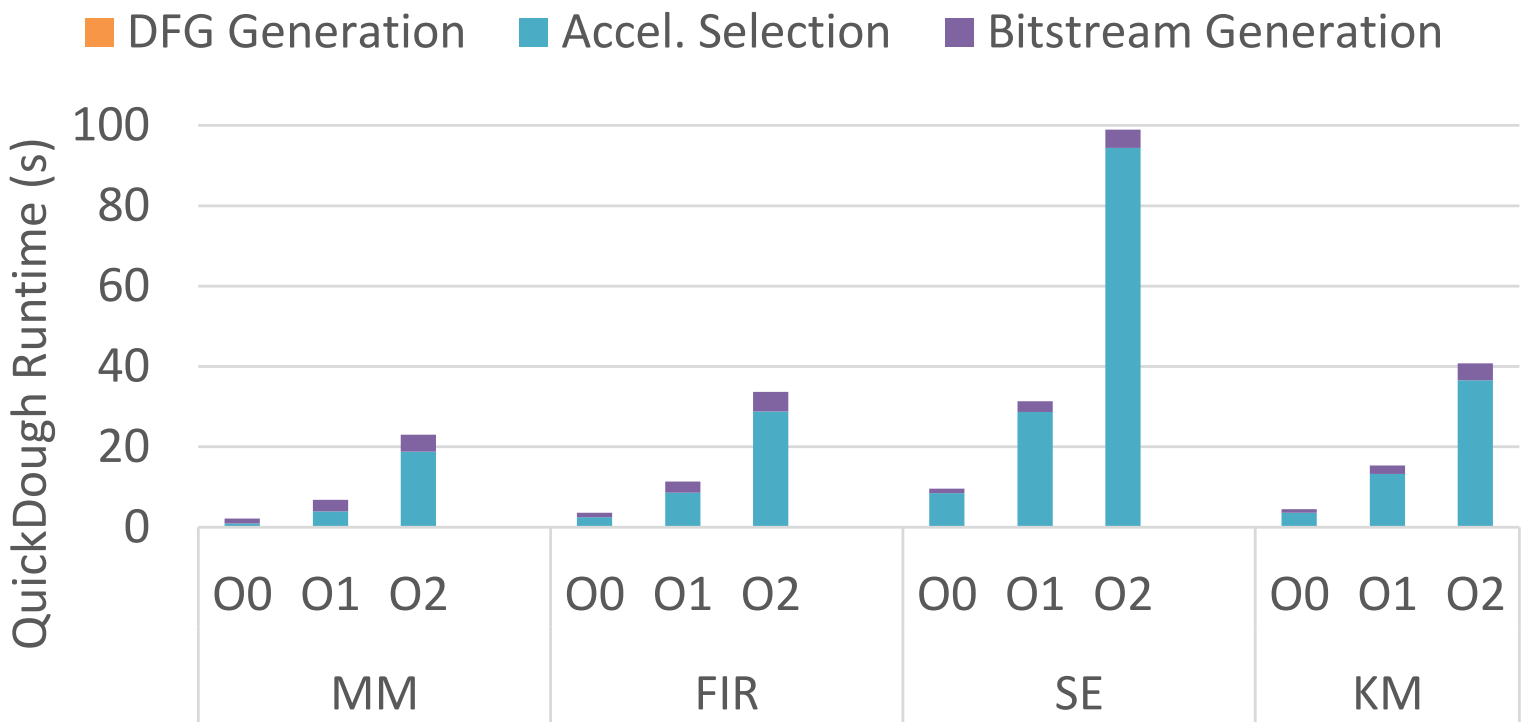


Figure: Benchmark performance speedup over software executed on ARM processor and execution time decomposition of loop accelerators generated using QuickDough.

Time Consumption of the Loop Accelerator Generation

With the pre-built library, each design iteration in QuickDough involves 3 steps:

- DFG generation
- Accelerator selection and DFG scheduling
- Bitstream generation



Outline

- Background & Motivation
- Related Work
- QuickDough Design Framework
- SCGRA Overlay Architecture
- FPGA Loop Accelerator Customization
- **Experiments**
  - Loop Accelerator Generation
  - **Loop Accelerator Customization**
- Conclusion

Experiment Setup

Basic setup:

- The resources on Zedboard are taken as the constraints.
- $\epsilon = 0.05$
- FPGA accelerator performance are calculated using the models.
- ARM processor performance are obtained from Zedboard.
- Power consumption is extracted from XPower.
- Exhaustive search (ES) and the two-step (TS) customization methods are used.

Assumptions used for the customization:

- All the overlay based accelerators run at 250 MHz on Zedboard.

Exhaustive search (ES) vs. two-step customization (TS)

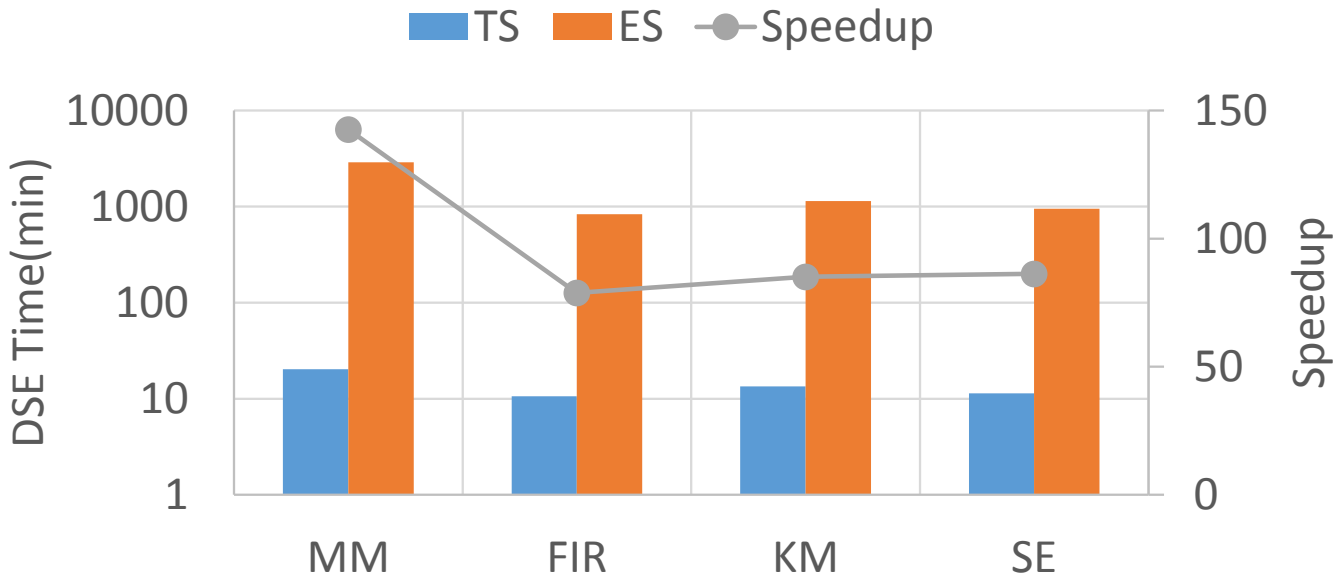


Figure: Customization Time Using Both TS and ES

Exhaustive search (ES) vs. two-step customization (TS)

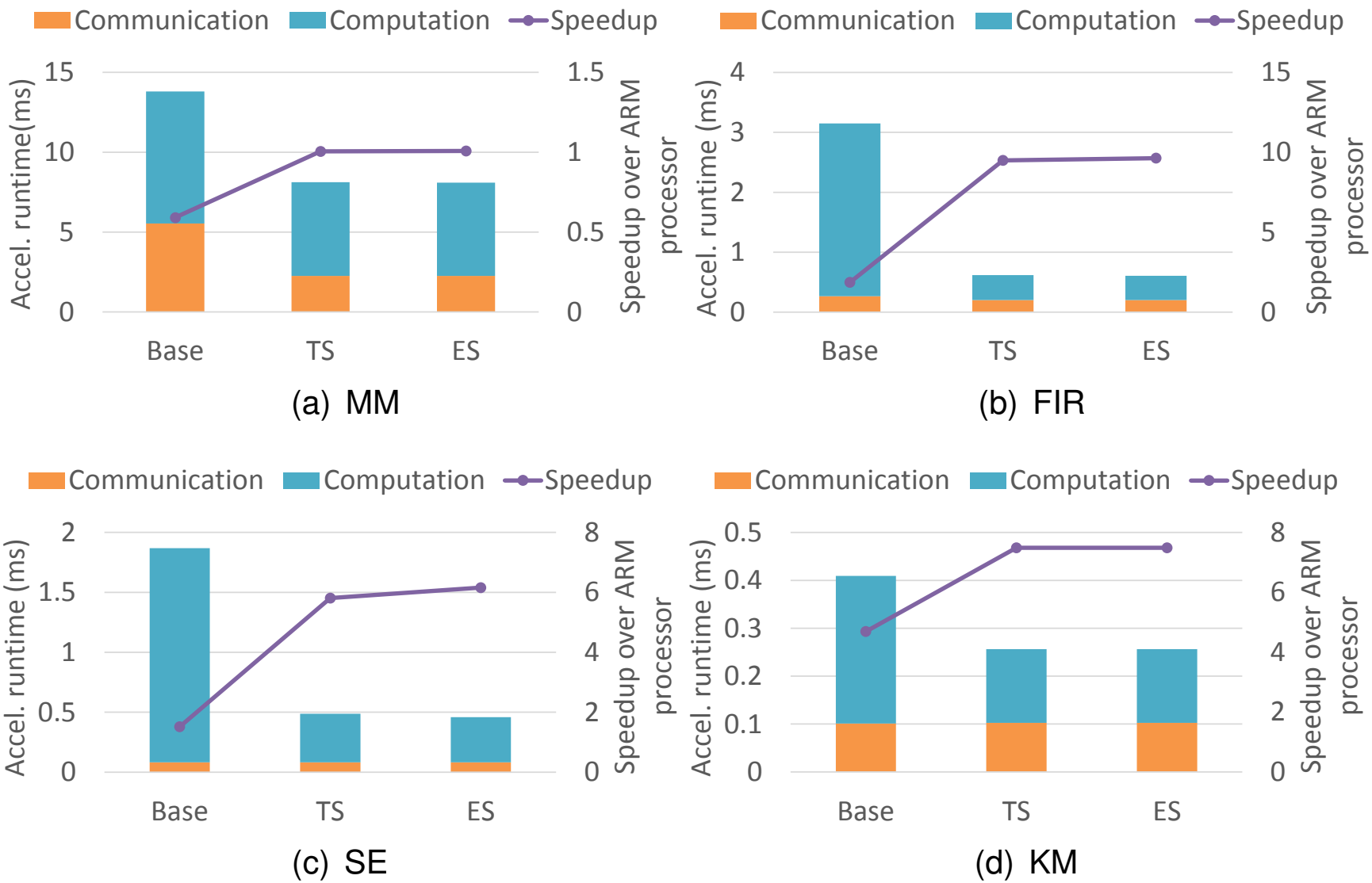


Figure: Customized FPGA Loop Accelerator Performance Comparison

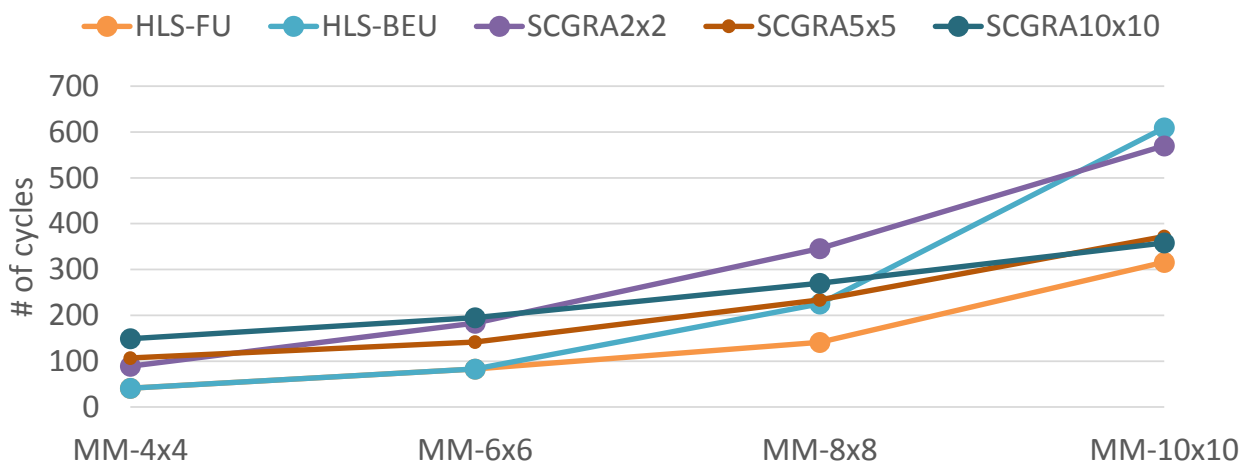
Outline

- Background & Motivation
- Related Work
- QuickDough Design Framework
- SCGRA Overlay Design & Implementation
- FPGA Loop Accelerator Customization
- Experiments
  - SCGRA Overlay Architecture
  - Loop Accelerator Generation
  - Loop Accelerator Customization
- **Conclusion**

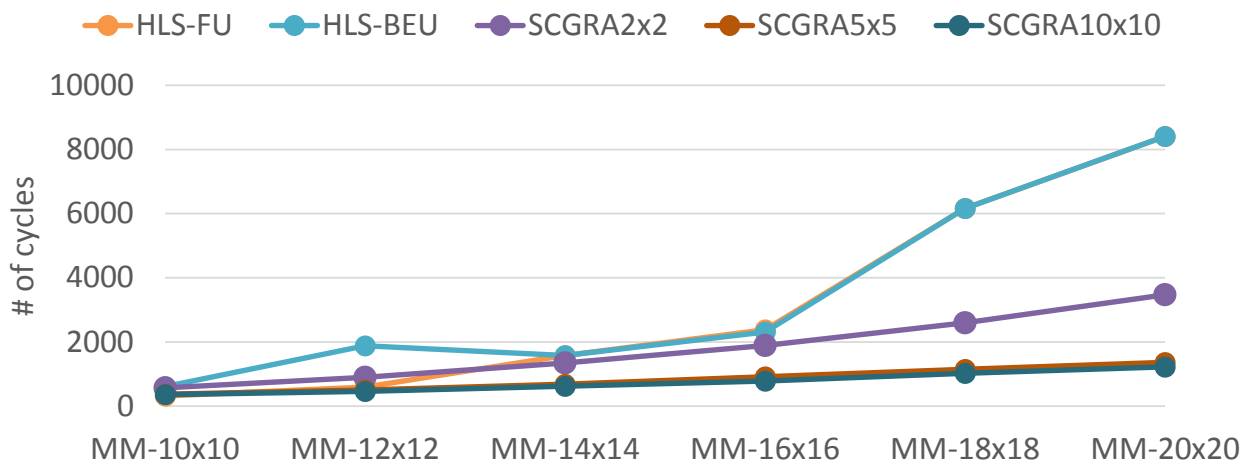
Conclusion

- Presented the QuickDough compilation framework for high productivity FPGA loop accelerator design on a hybrid CPU-FPGA computing system.
- Implemented a highly pipelined, flexible and scalable SCGRA overlay on top of FPGAs which contributes to the good performance of the resulting accelerators as well as the capability of application specific customization of QuickDough.
- Taking advantage of the regularity of the SCGRA overlay, an automatic customization framework was developed to customize the design parameters of the resulting accelerators.

SCGRA Overlay Scalability (1)



(a)



(b)

Figure: # of cycles of MM Execution on Both HLS Based Design and SCGRA Overlay

SCGRA Overlay Scalability(2)

Table: SCGRA Based FPGA Accelerator Configuration

SCGRA Size	Inst. Rom	Data Mem	IBuf /OBuf	Addr Buf
2x2, 3x3, ..., 10x10	1k, 4k, ..., 8k	256x32	2kx32, 4kx32, ..., 8kx32	4kx16, 8kx16, ..., 16kx16

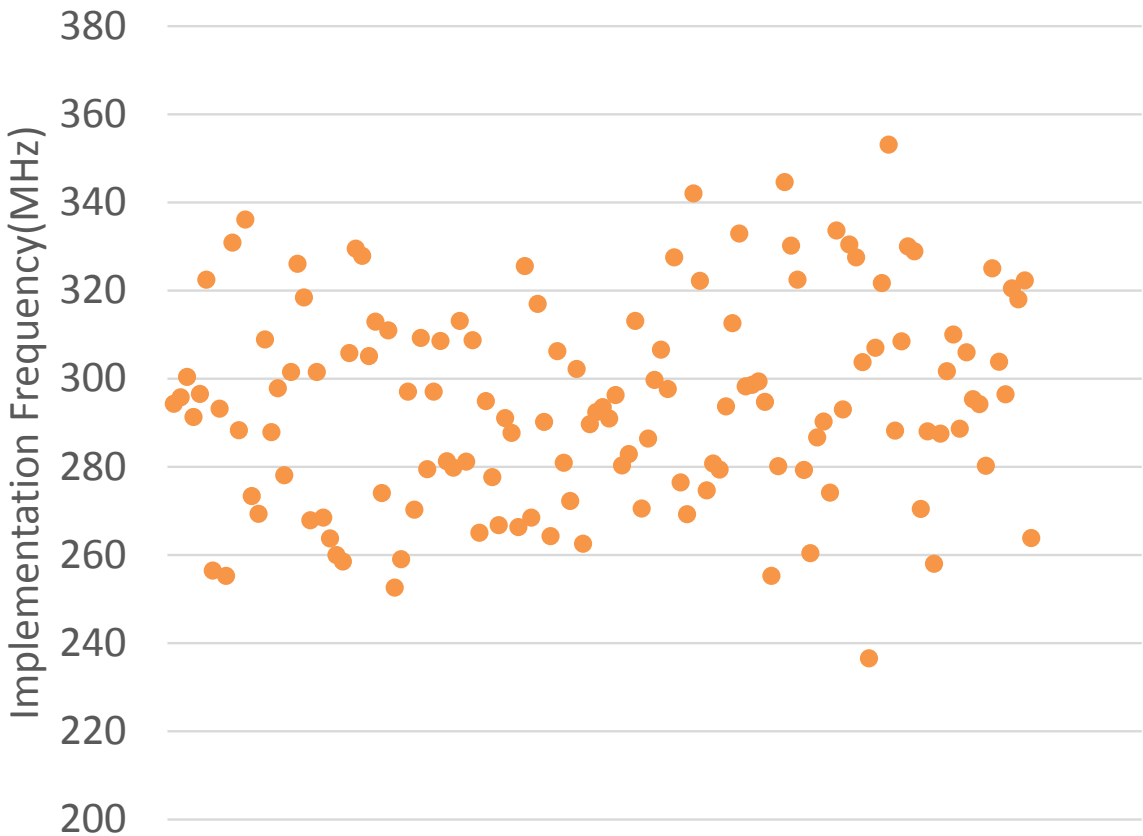
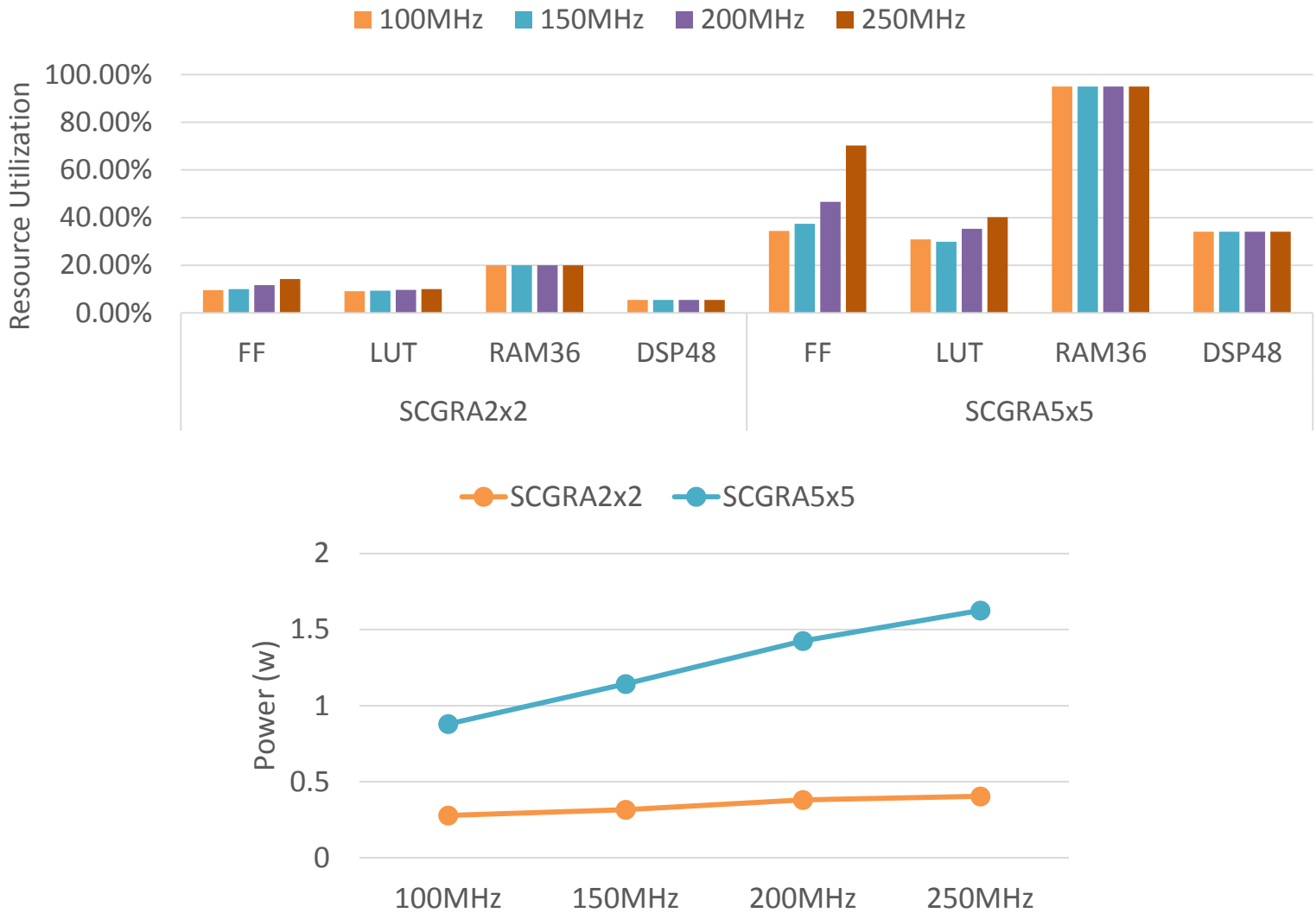
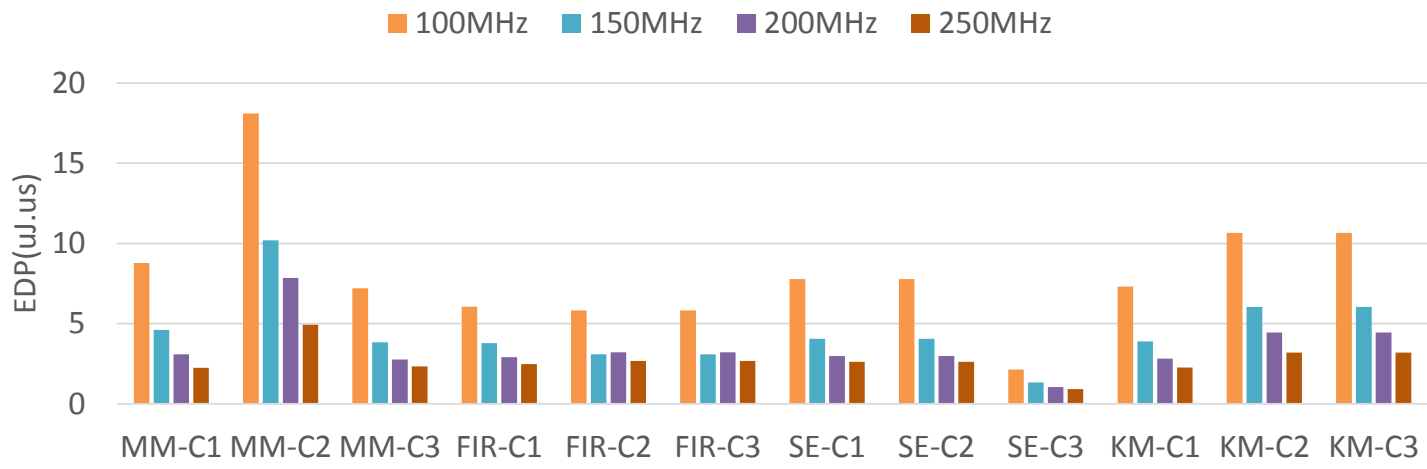


Figure: fmax of SCGRA Overlay with Various Configurations

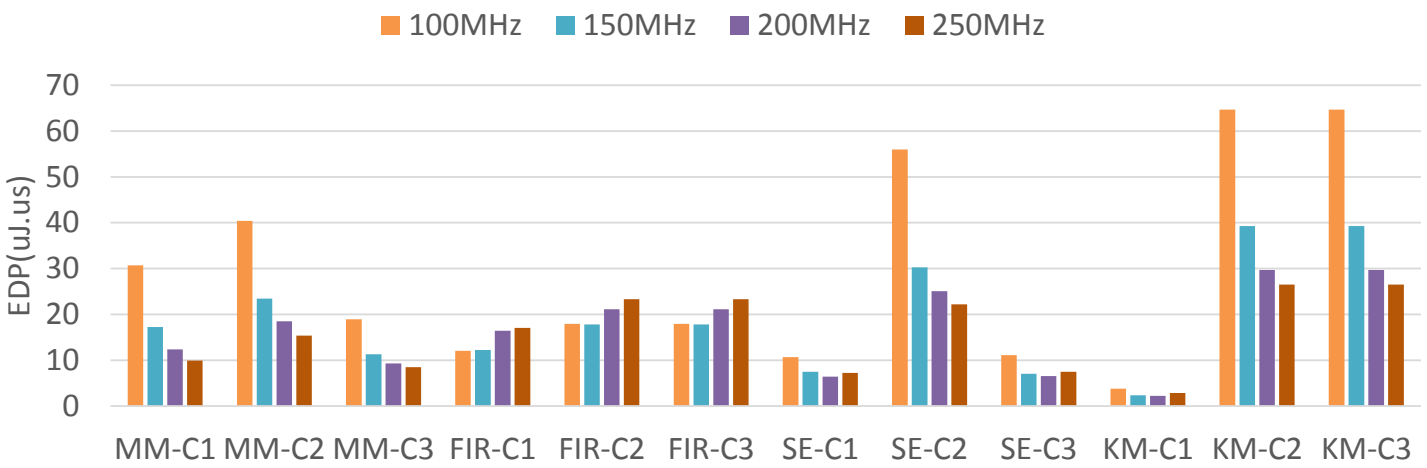
Pipelining Influence on HW Resource Consumption



Pipelining Influence on Energy Efficiency



(a) SCGRA2 $\times$ 2



(b) SCGRA 5 $\times$ 5

Figure: Energy Delay Product of SCGRA Overlays with Different Pipelining

Implementation of the Resulting Accelerators

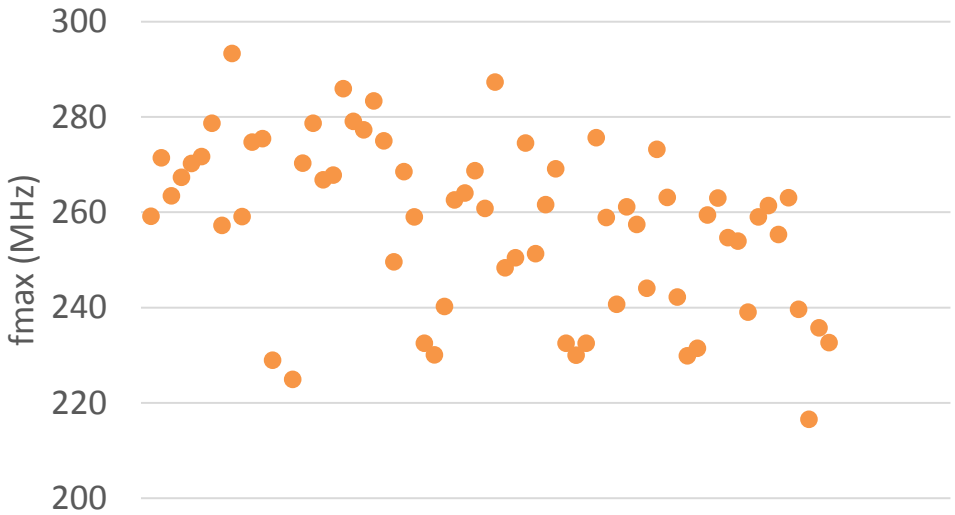


Figure: fmax of the Accelerators in the Library

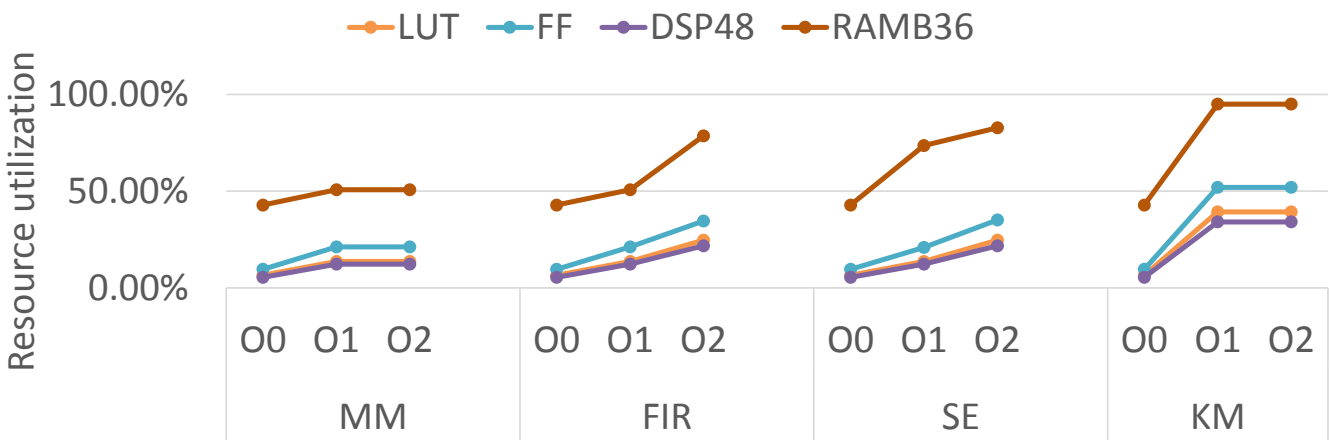


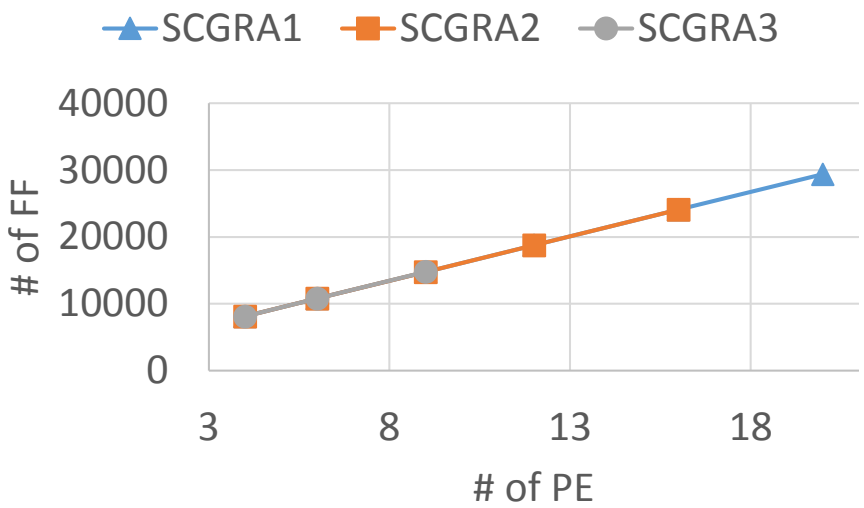
Figure: FPGA Accelerator Recource Utilization

Experiment Setup

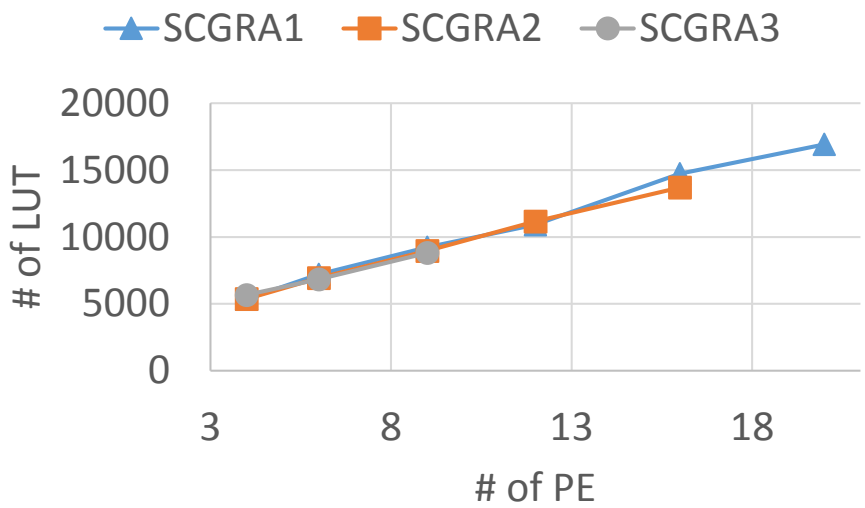
Table: SCGRA Based FPGA Accelerator Configuration

Group	Size	Inst. Rom	Data Mem	IBuf /OBuf	Addr Buf
SCGRA1	2x2, 3x2, 3x3, 4x3, 4x4, 5x4	1kx72	256x32	2kx32	4kx16
SCGRA2	2x2, 3x2, 3x3, 4x3, 4x4	2kx72	256x32	2kx32	4kx16
SCGRA3	2x2, 3x2, 3x3	4kx72	256x32	2kx32	4kx16

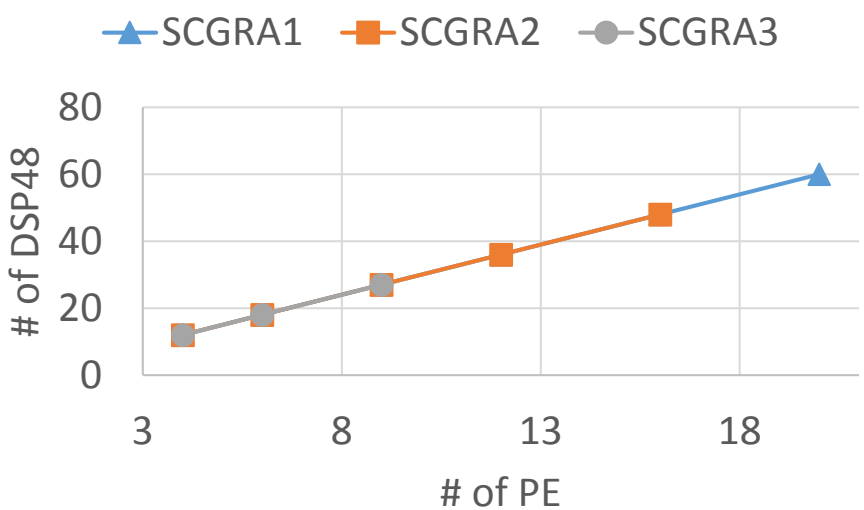
Overlay Based FPGA Accelerator Implementations



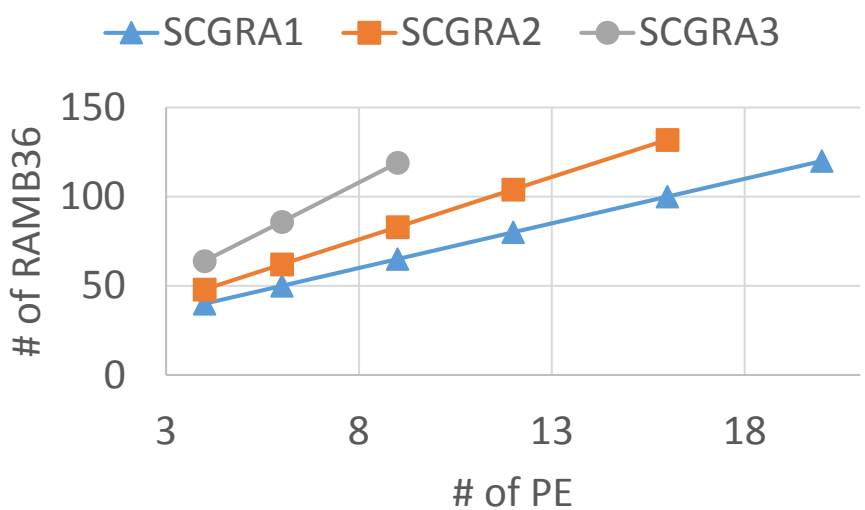
(a)



(b)



(c)



(d)

Overlay Based FPGA Accelerator Implementations

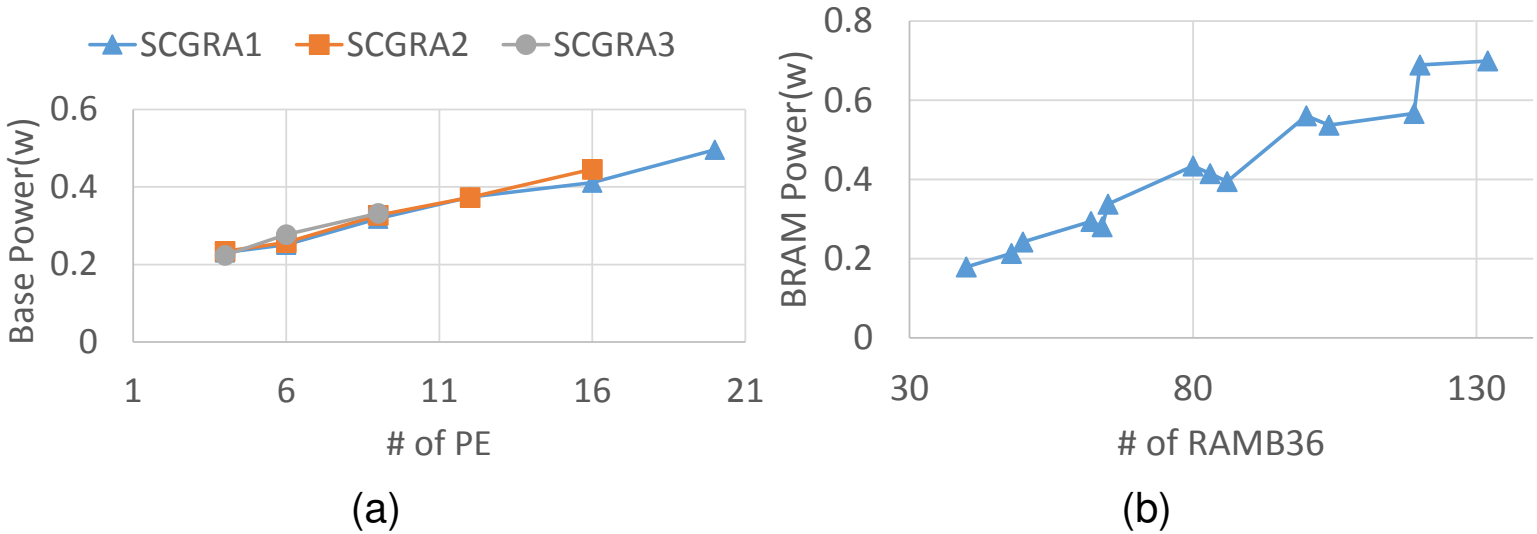


Figure: Power Consumption of the SCGRA Overlay Based FPGA Accelerators, (a) Base System Power Including DSP Power, Clock power, Signal Power, etc., (b) BRAM Power

DMA Latency on Zedboard

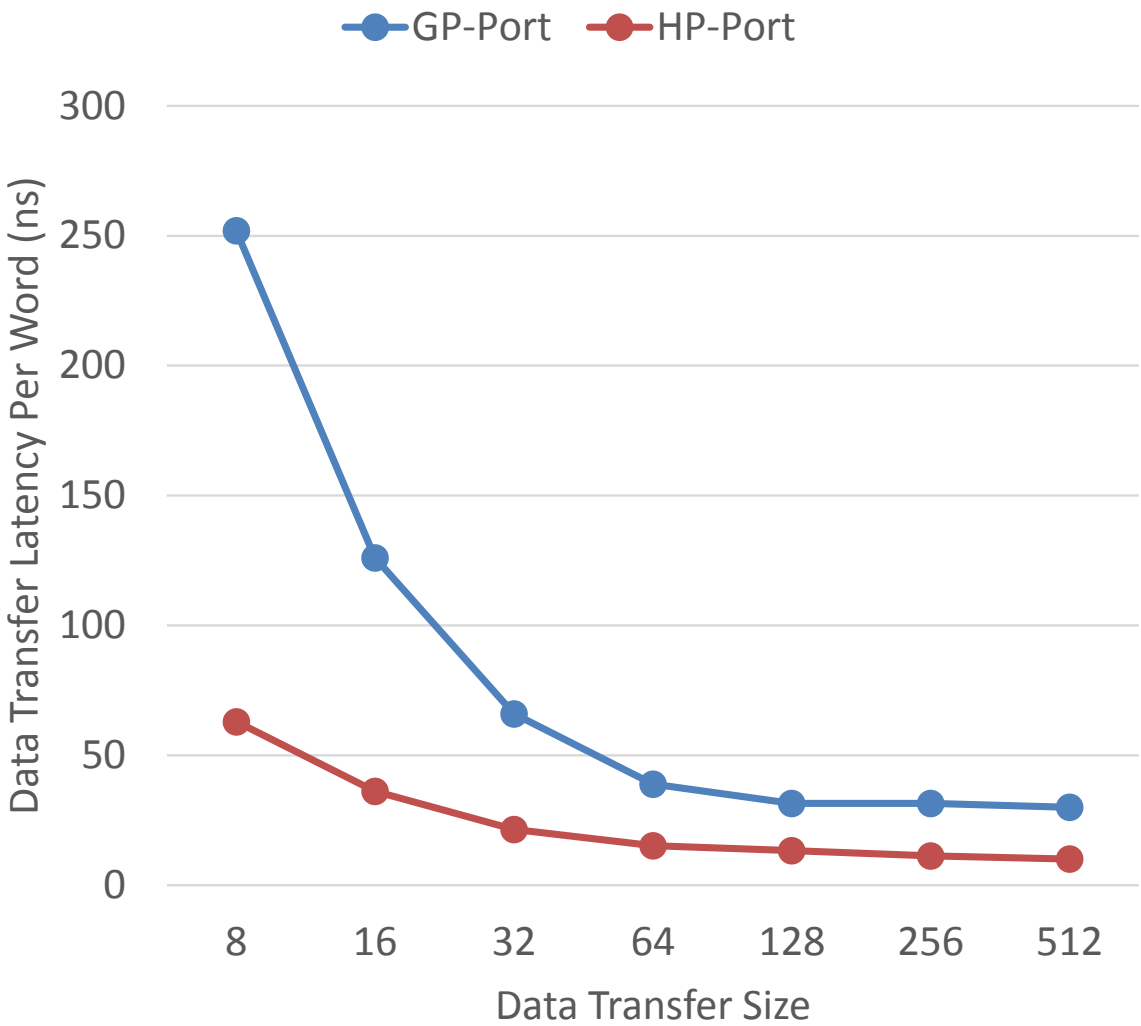


Figure: Zedboard DMA Transfer Latency Per Word

Epsilon Sensitivity

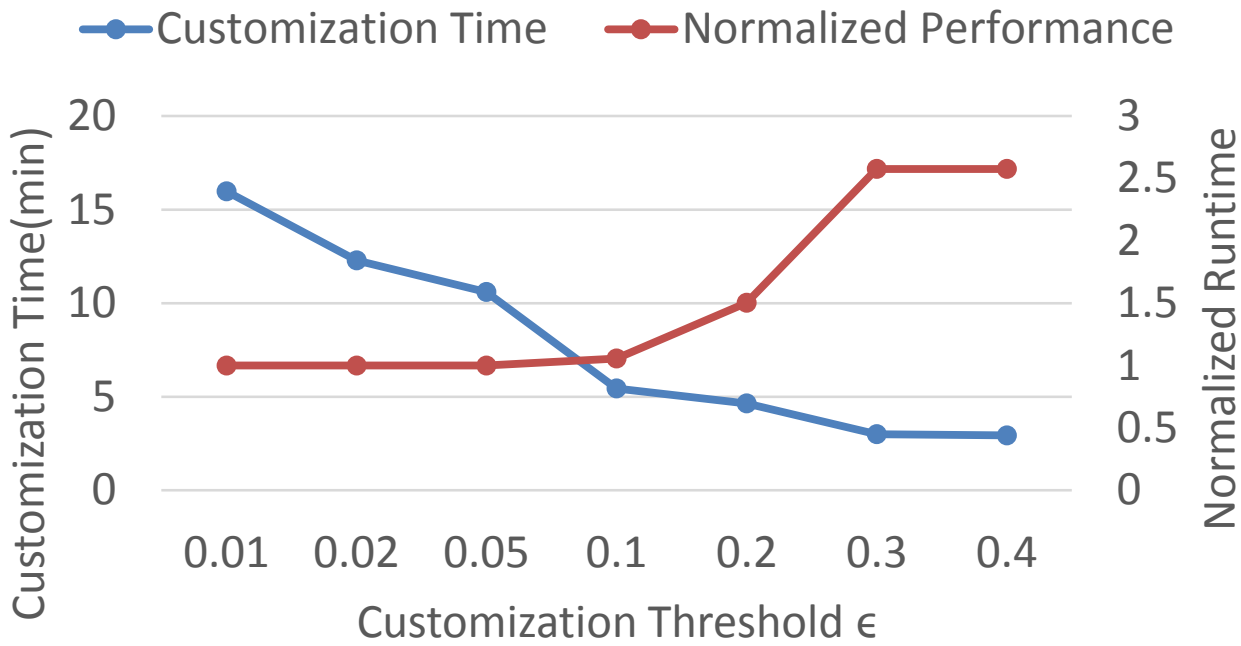
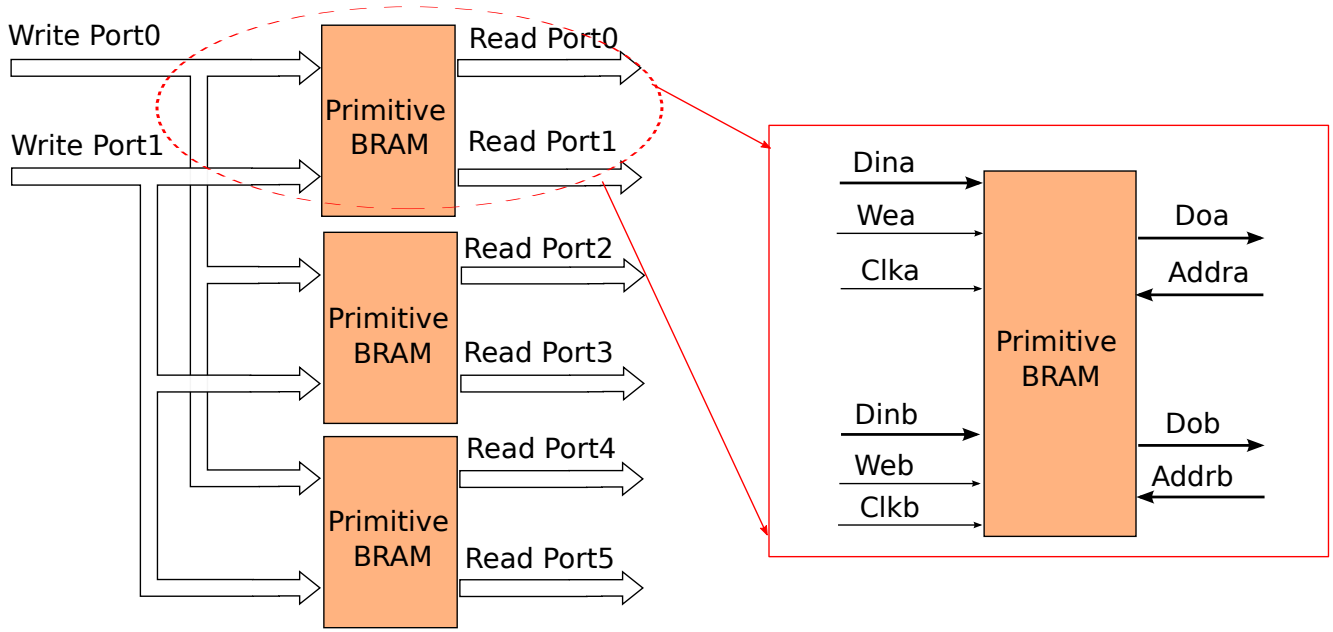
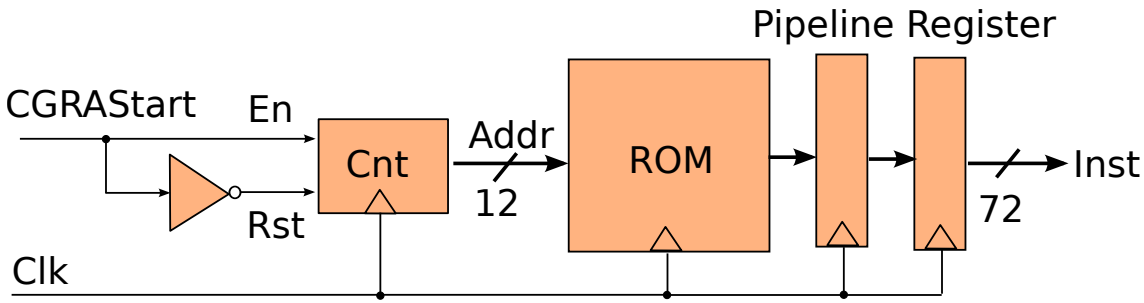


Figure: ϵ Influence on FIR Customization Time and Resulting Accelerator Run-time

Data Memory & Inst Memory

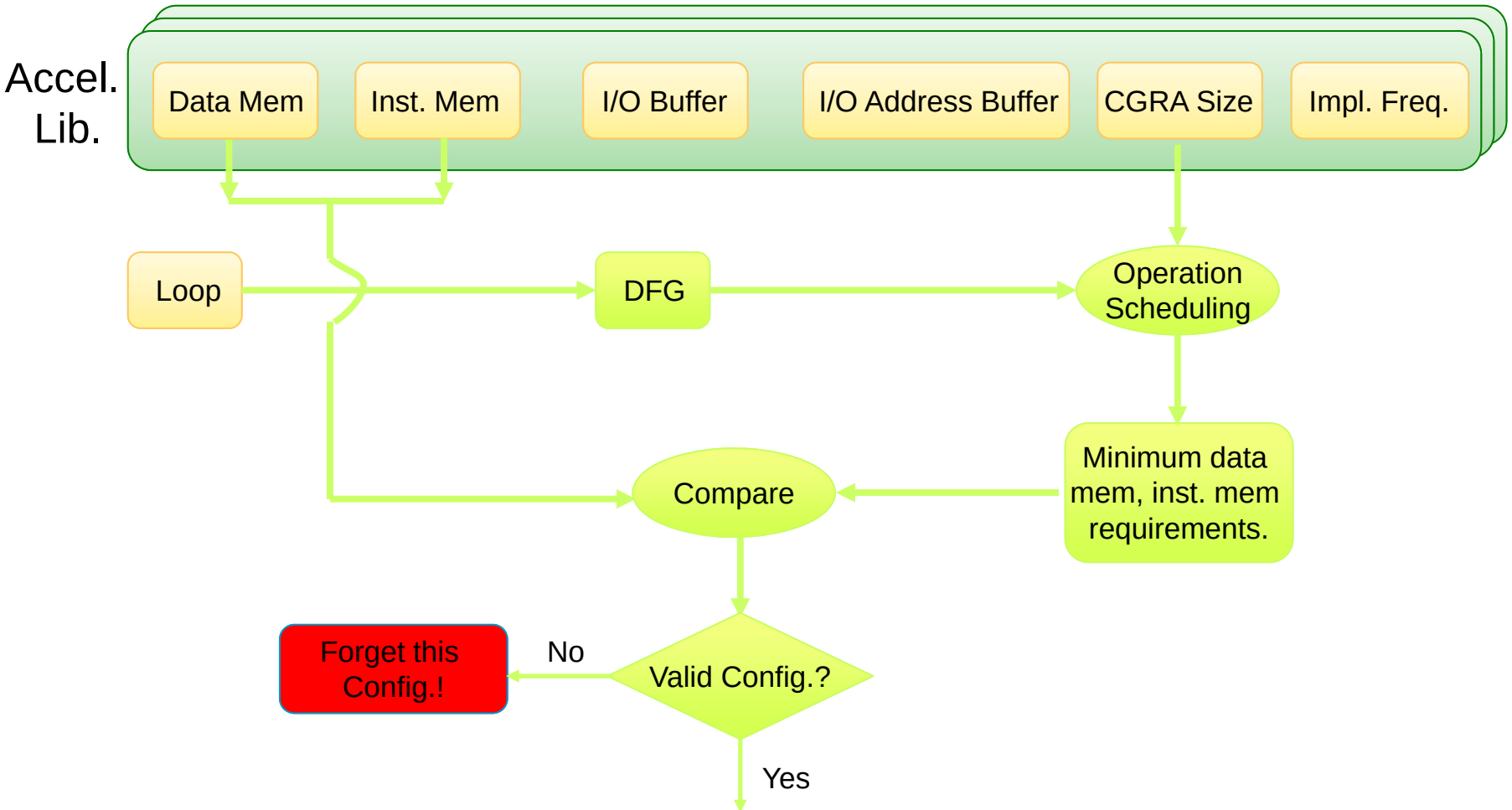


- The memory has 2 write ports and 6 read ports, but it is **NOT** 2W6R memory.
- Write port 0 and read port 0, 2, 4 will not function at the same time.
- Write port 1 and read port 1, 3, 5 will not function at the same time.

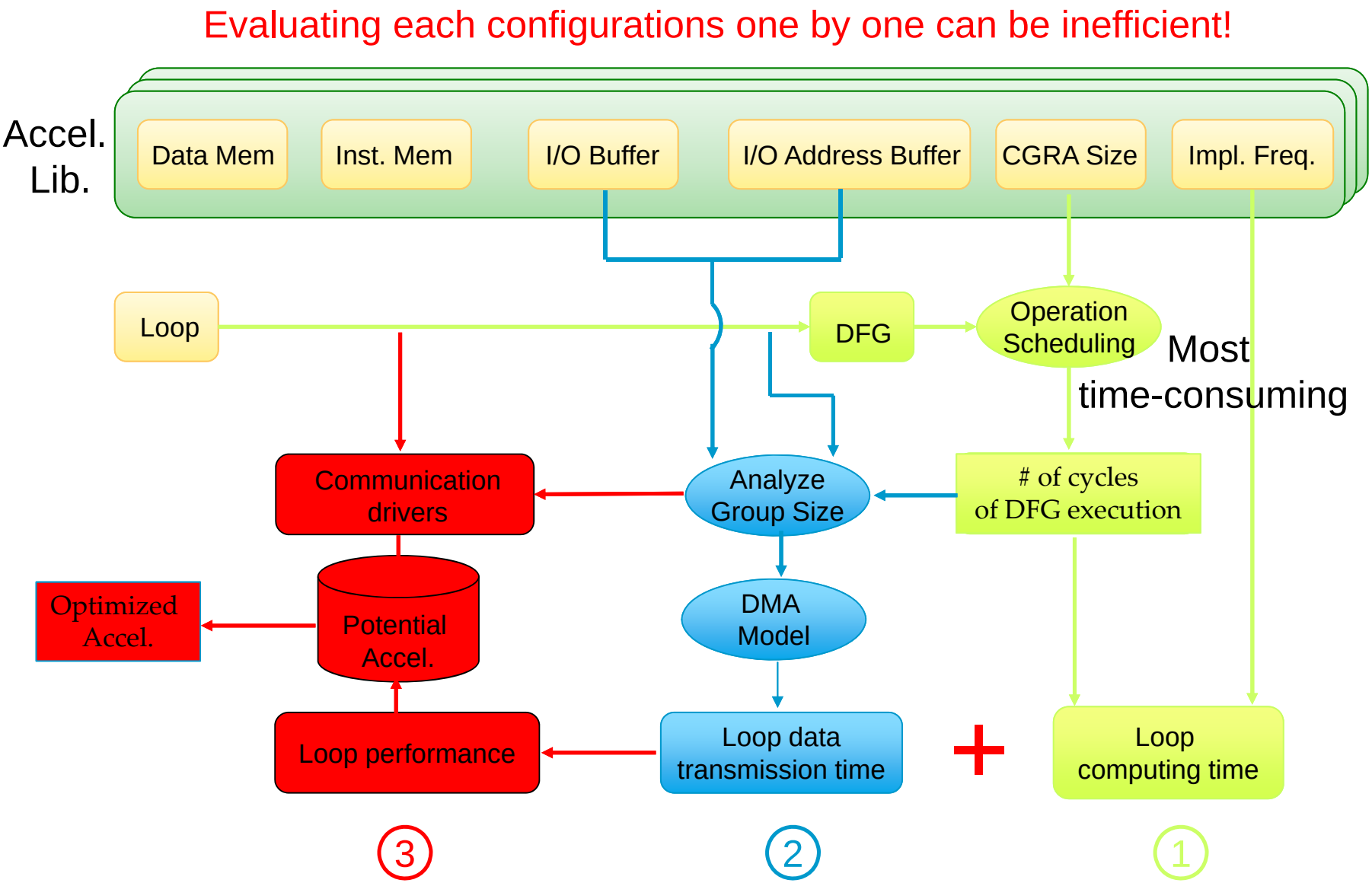


Accelerator Selection (1)

Select the optimized SCGRA overlay based accelerator configurations from the pre-built library.



Accelerator Selection (2)



Accelerator Library Pre-building Time

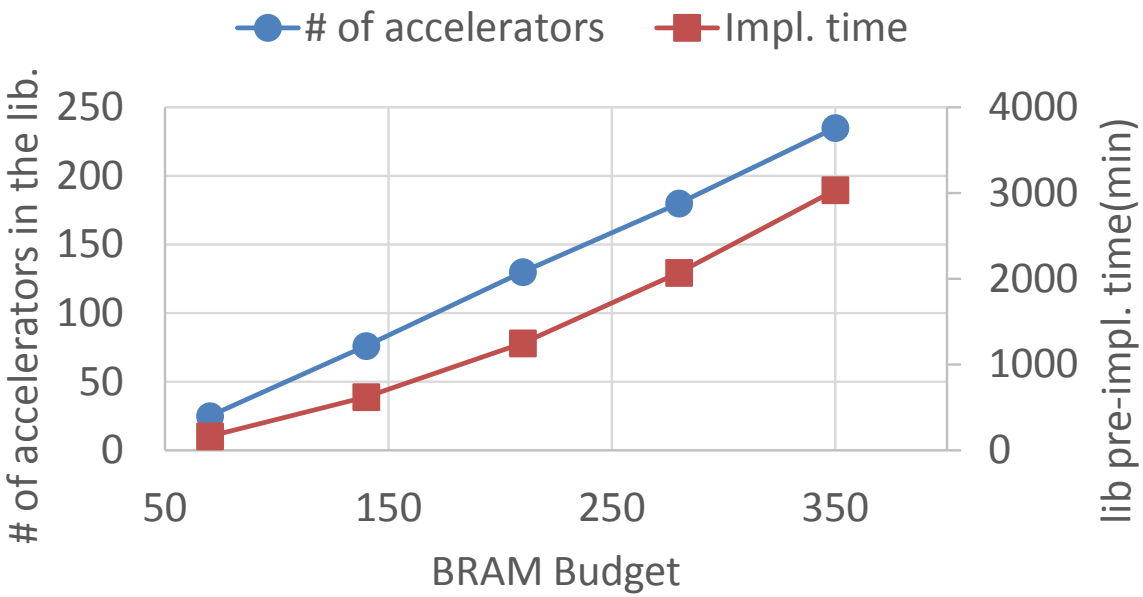
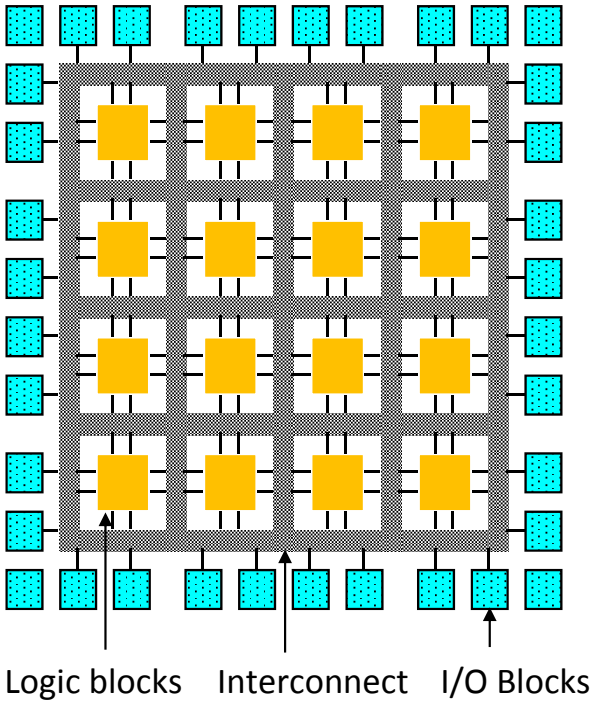


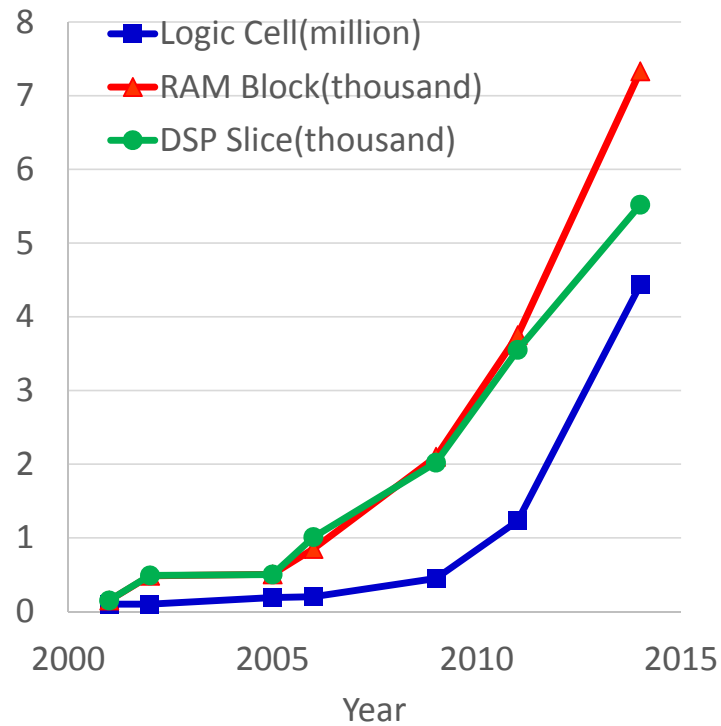
Figure: Accelerator library size and implementation time given different BRAM budgets.

- With simple empirical constraints, the accelerator library pre-building time will not increase dramatically with the BRAM budgets.
- The implementations are completely independent and can be done on a distributed computing system with much shorter time.

FPGA Introduction



Typical FPGA Architecture



Maximum FPGA Resources of Xilinx FPGA