Time Sharing of Runtime Coarse-Grain Reconfigurable Architectures Processing Elements in Multi-Process Systems

Benjamin Carrion Schafer
The Hong Kong Polytechnic University
Department of Electronic and Information Engineering
b.carrionschafer@polyu.edu.hk

Design Automation and Reconfigurable Computing LAB
DARC

@DARC_lab

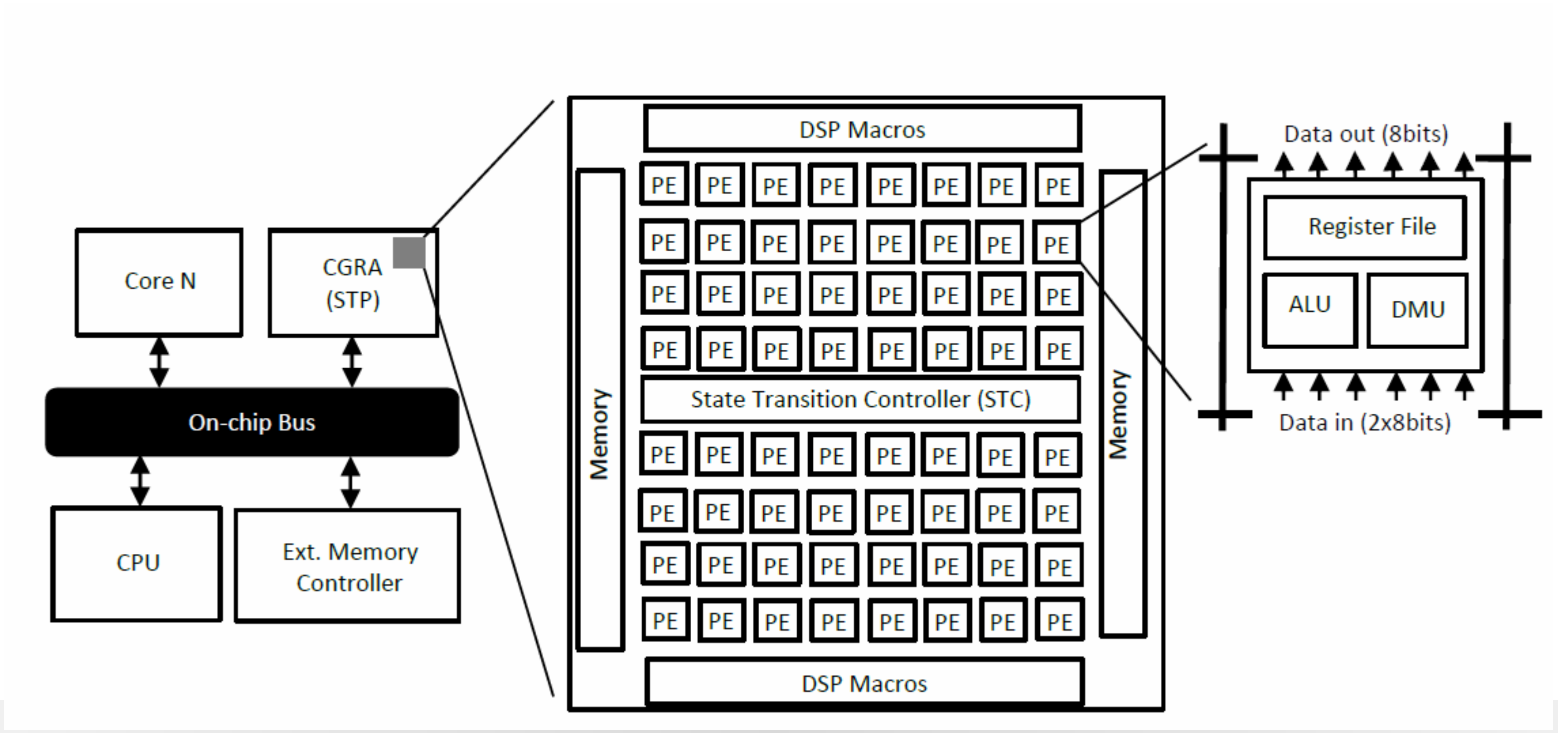
The Hong Kong Polytechnic University
香港理工大學

Outline

- Target architecture – Stream Transpose Processor
- STP configuration flow (High-Level Synthesis)
- Motivational Example for mapping multiple processes onto the same STP core
- Proposed Sharing Method (2 phases)
  - Single Process characterization
    - 3 steps
  - PE sharing method
    - 4 steps
- Experimental Setup and Results
- Summary and Conclusions

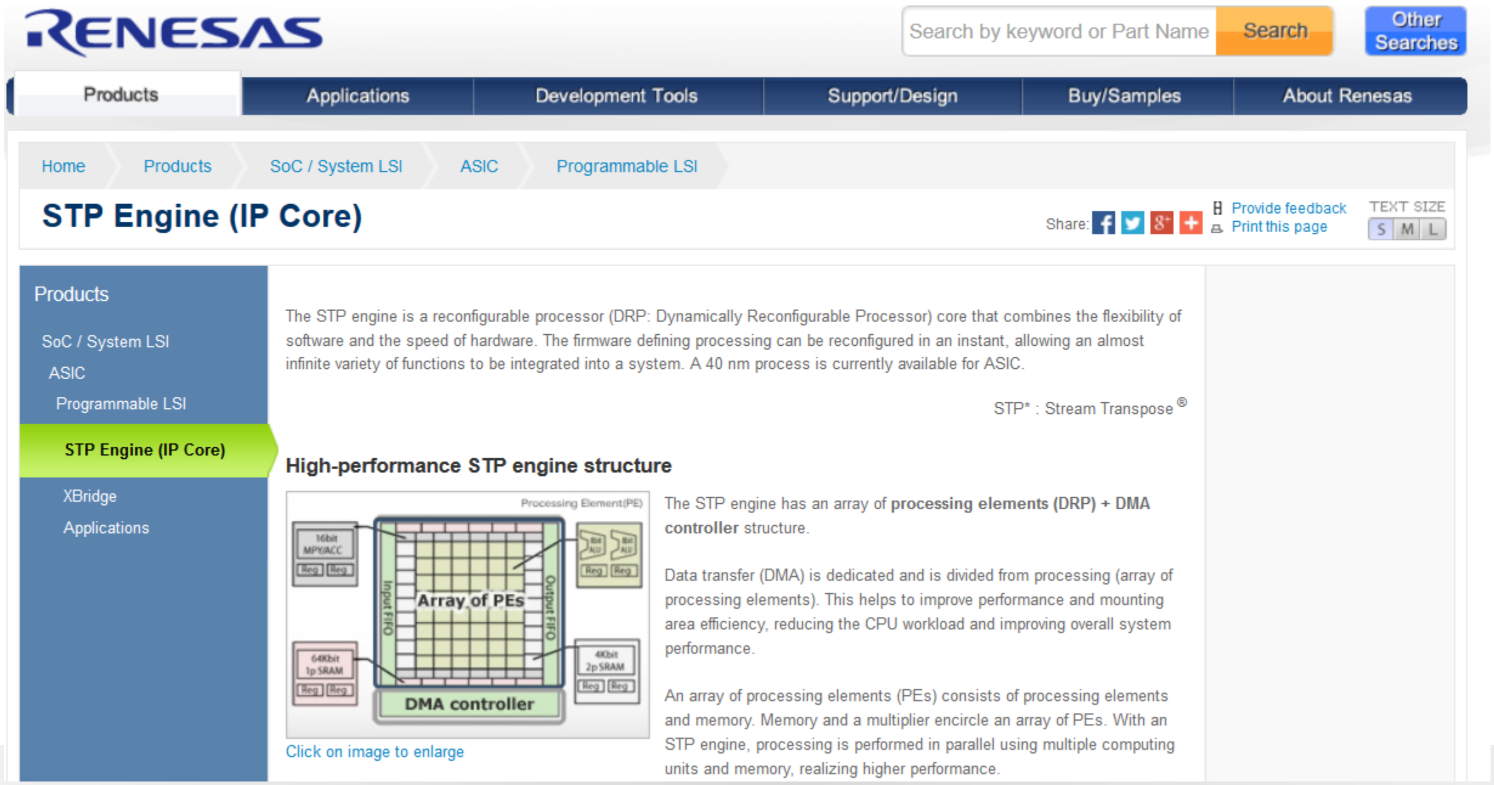
Target Architecture

- Reconfigurable (Programmable) SoC (RSoCs)
  - Include reconfigurable IPs
  - Used to accelerate computational intensive function with high parallelism (e.g image processing or DSP applications)



Stream Transpose Processor (STP)

1. Runtime Coarse Grain Reconfigurable Architecture (1ns to reconfigure PE functionality and routing)
2. Programmed using High-Level Synthesis (HLS)



Search by keyword or Part Name | Search | Other Searches

Products | Applications | Development Tools | Support/Design | Buy/Samples | About Renesas

Home > Products > SoC / System LSI > ASIC > Programmable LSI

STP Engine (IP Core)

Share: | Provide feedback | Print this page | TEXT SIZE S M L

Products
- SoC / System LSI
- ASIC
- Programmable LSI
- STP Engine (IP Core)
- XBridge
- Applications

The STP engine is a reconfigurable processor (DRP: Dynamically Reconfigurable Processor) core that combines the flexibility of software and the speed of hardware. The firmware defining processing can be reconfigured in an instant, allowing an almost infinite variety of functions to be integrated into a system. A 40 nm process is currently available for ASIC.

STP* : Stream Transpose®

High-performance STP engine structure

Click on image to enlarge

The STP engine has an array of **processing elements (DRP) + DMA controller** structure.

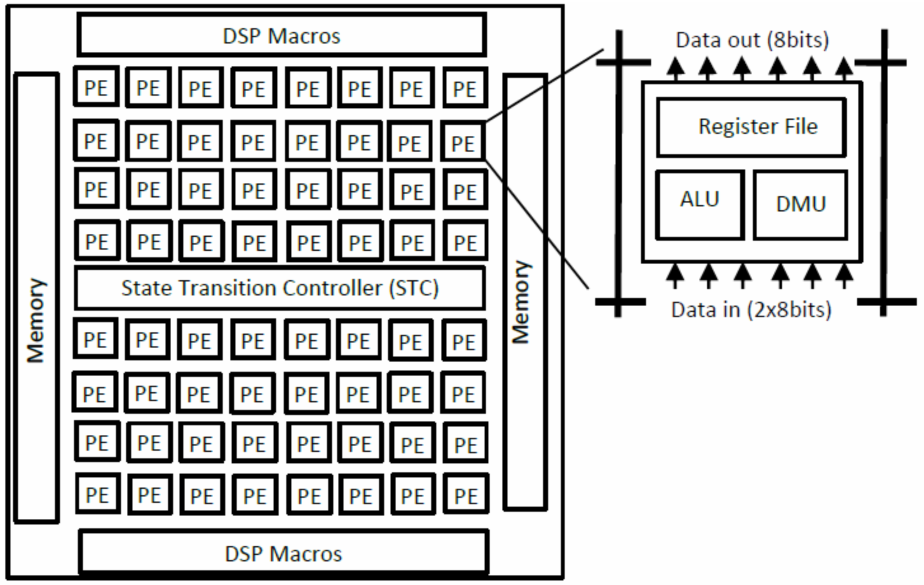
Data transfer (DMA) is dedicated and is divided from processing (array of processing elements). This helps to improve performance and mounting area efficiency, reducing the CPU workload and improving overall system performance.

An array of processing elements (PEs) consists of processing elements and memory. Memory and a multiplier encircle an array of PEs. With an STP engine, processing is performed in parallel using multiple computing units and memory, realizing higher performance.

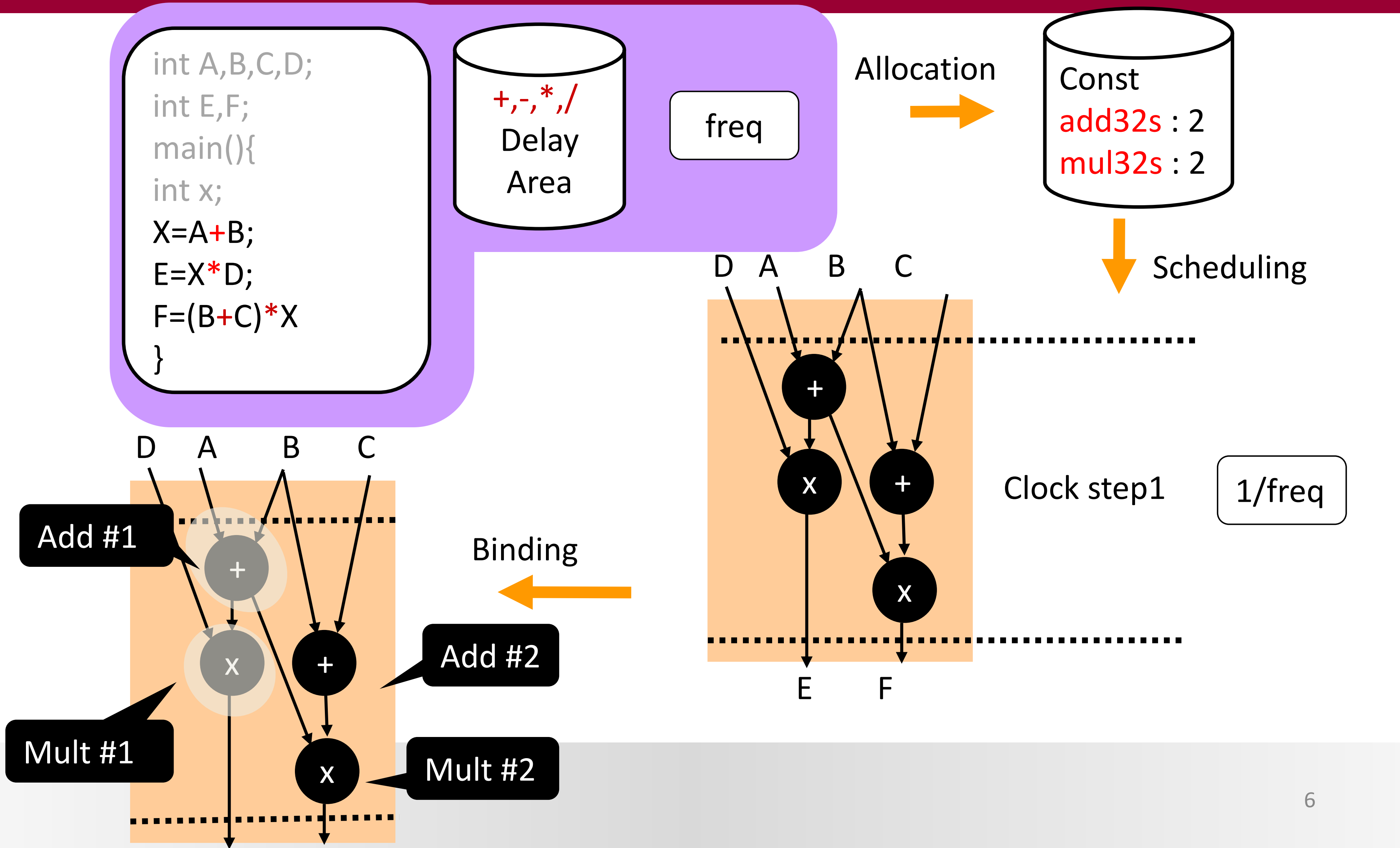
http://am.renesas.com/products/soc/asic/programmable/stp/index.jsp

Stream Transpose Processor cont.

- The main building blocks are called tiles
- Each tile consists of an array of 8x8 PEs. Each PE contains:
  - 8-bit arithmetic logic unit (ALU), an 8-bit data manipulation unit (DMU) for 1-bit logic operations and 8-bit shifting and masking and an 8-bit flip-flop unit (FFU).
- Surrounded by embedded memory and embedded multipliers
- The STP can hold up to 64 contexts in its State Transition Controller located in the middle of the PE array



High Level Synthesis 101

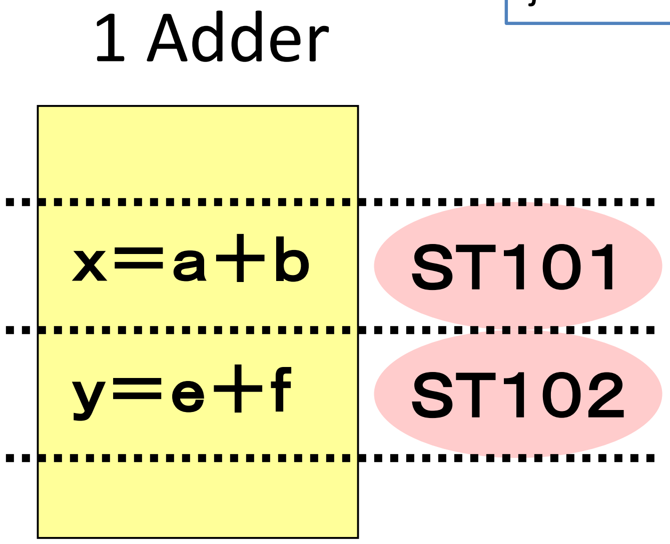


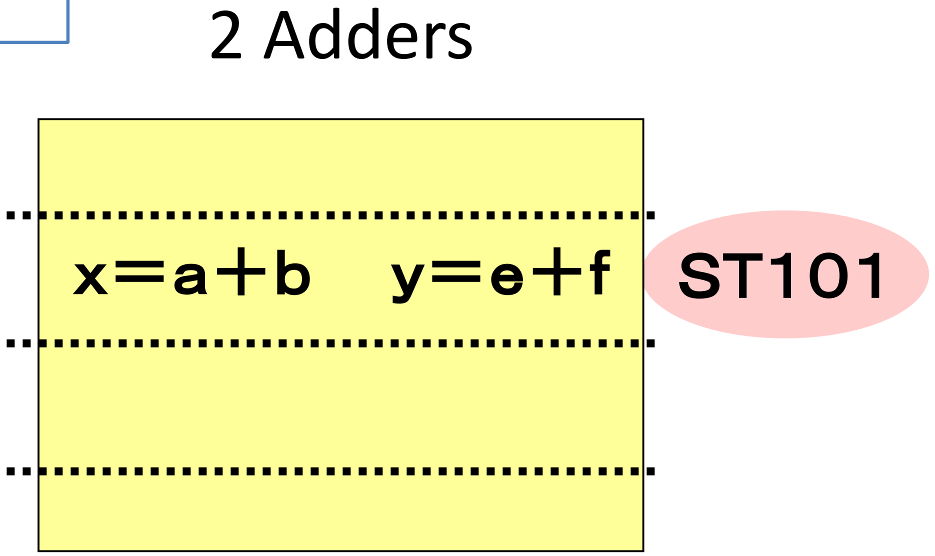
```
int A,B,C,D;
int E,F;
main(){
int x;
X=A+B;
E=X*D;
F=(B+C)*X
}
```

HLS Resources Constraints

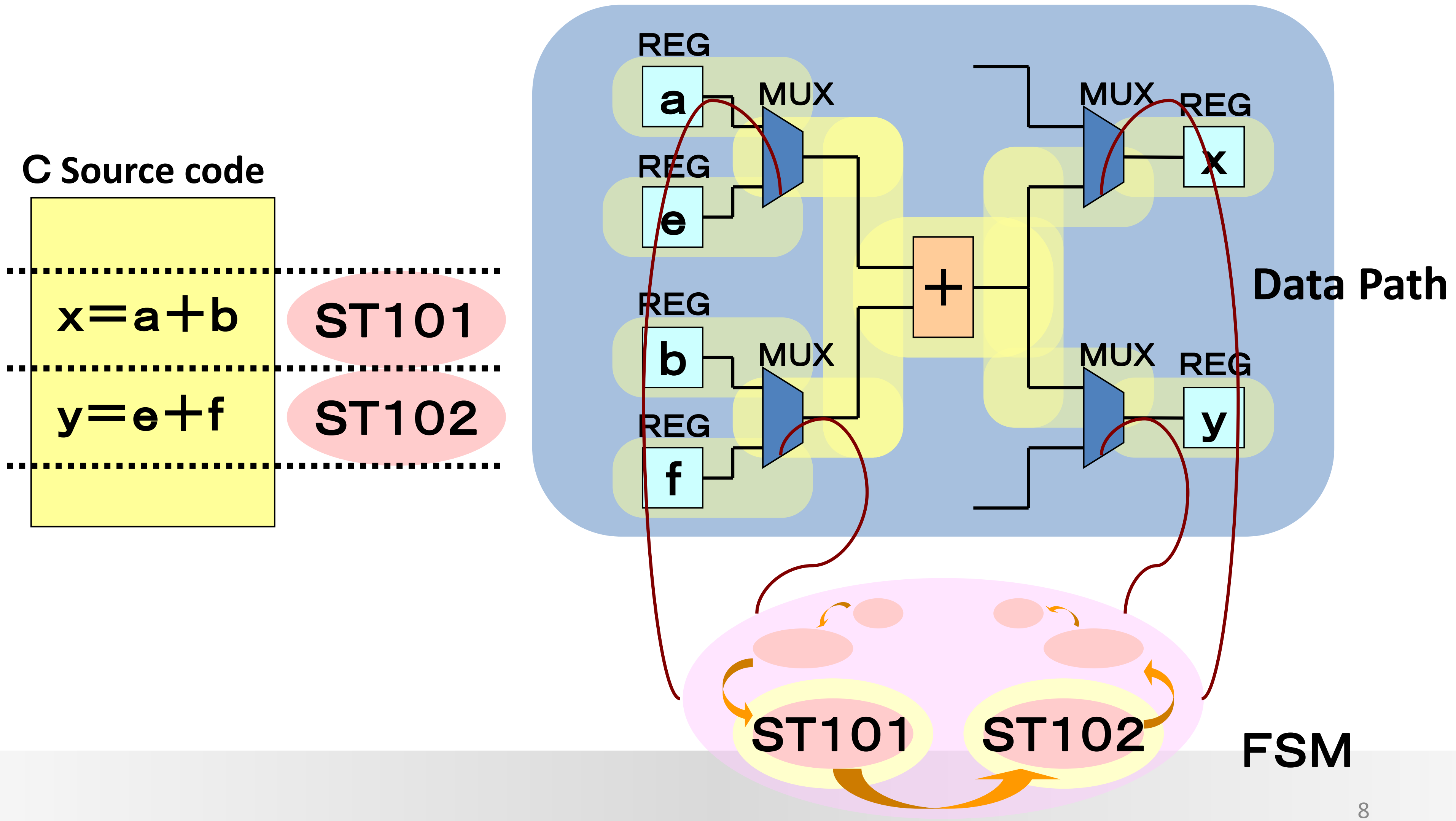
- Functional Unit Constraint file specifies how many FUs can be instantiated → Impacts the synthesized architecture

```
int main(){
  x = a+b;
  y = e+f;
}
```



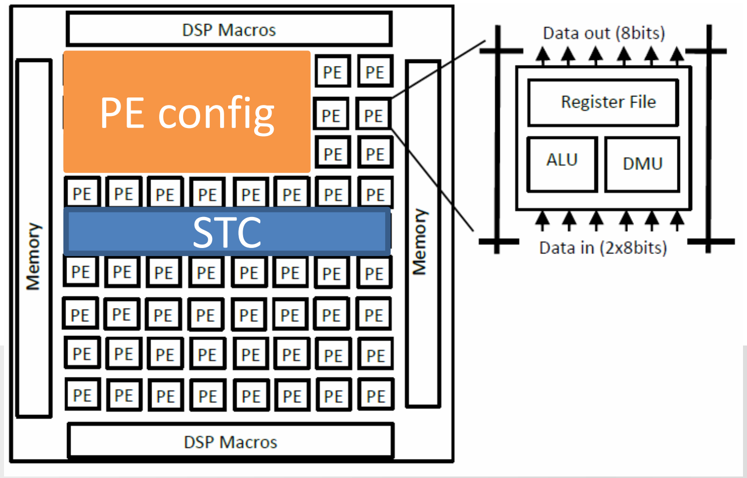


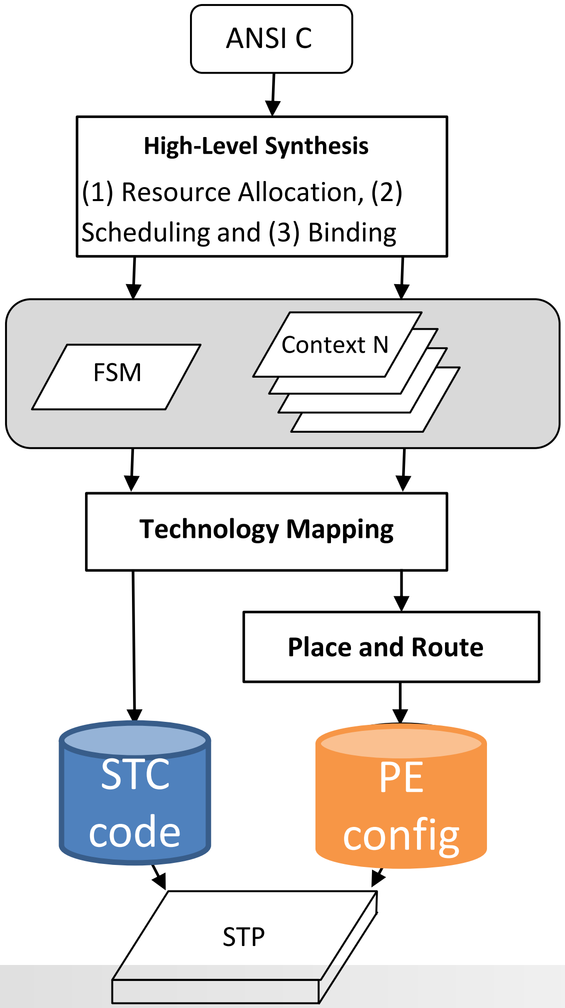
High Level Synthesis–Min FUs (Resource Sharing)



STP Configuration Flow

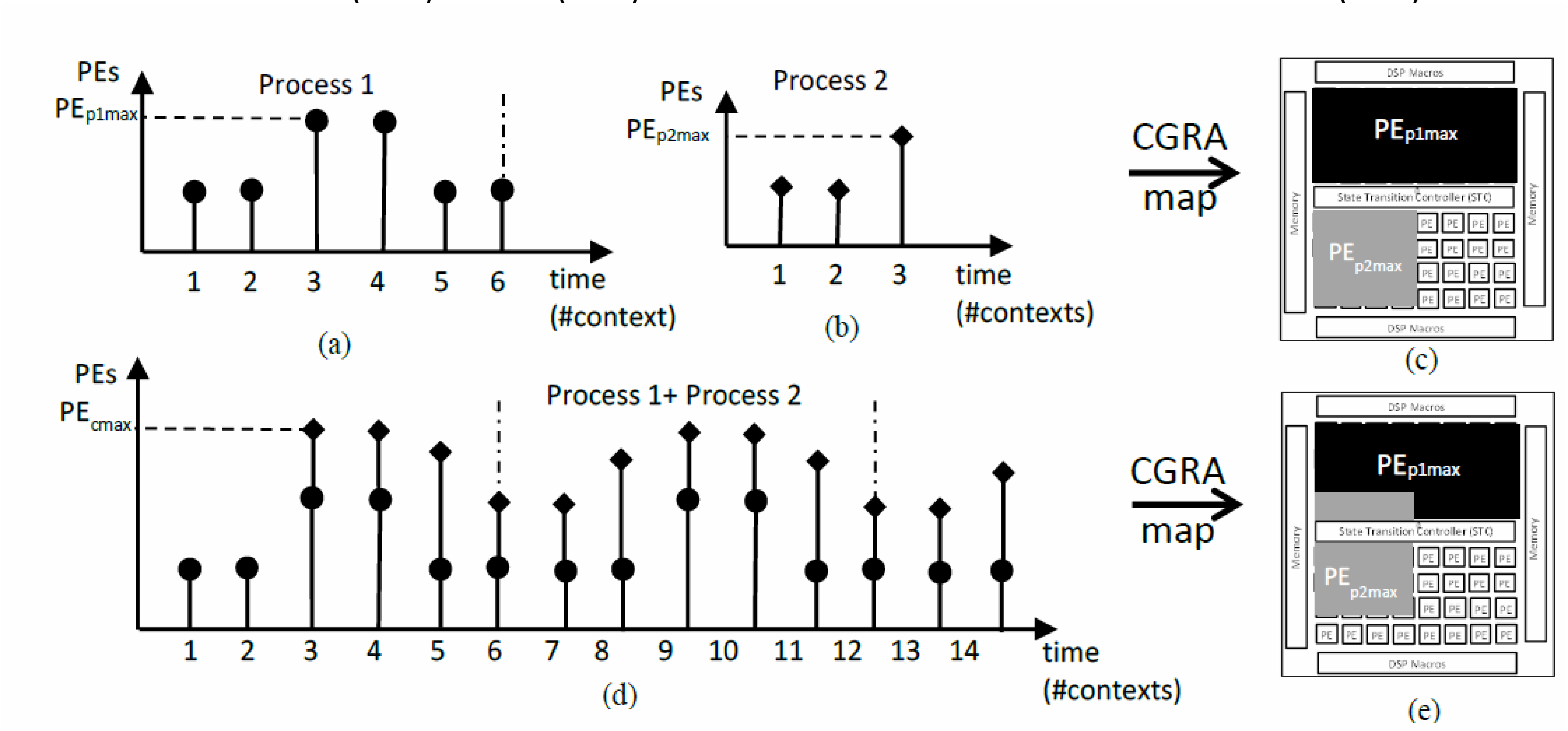
- Start with sequential description in C
- Perform HLS:
  - FSM mapped to State Transition Controller (STC)
  - Data Path mapped to PEs





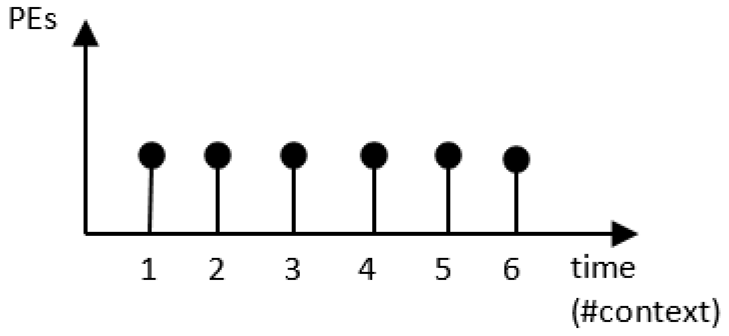
Motivational Example

- Mapping multiple processes onto the same STP
- Each process synthesized (HLS) individually
- PE usage = $PE_{P1(max)}+PE_{P2(max)}$ ➔ HLS GOAL is to minimize $PE_{(max)}$

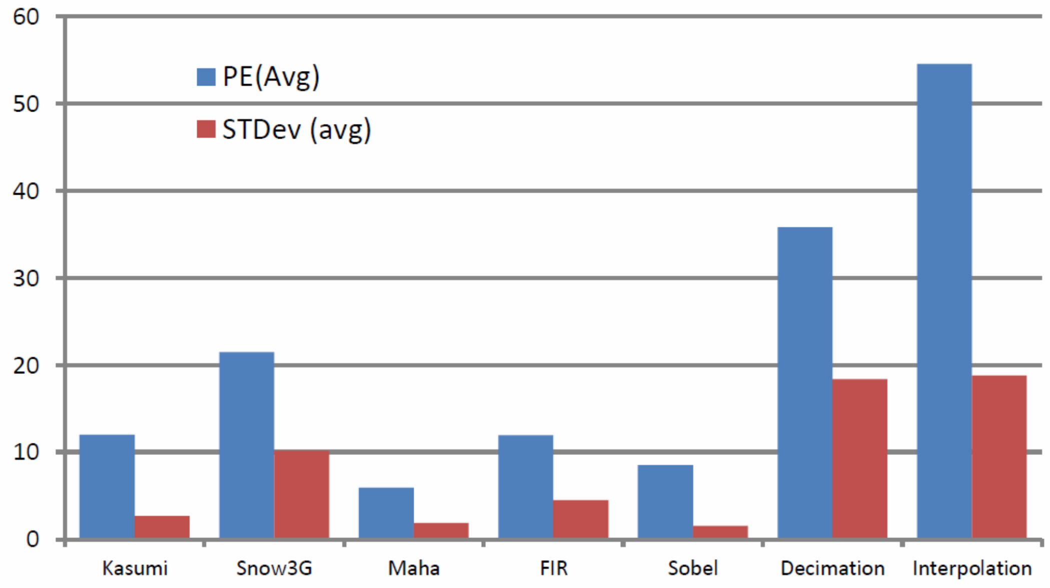


Main Idea Behind this Work

- Exploit the imbalance in PE usage across contexts to share PEs across processes
- No optimizations possible of PE usage optimized across contexts



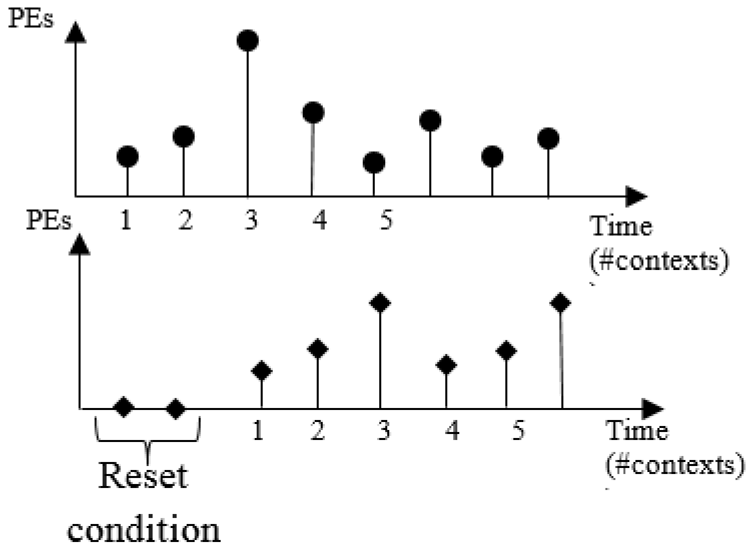
Ideal synthesis result



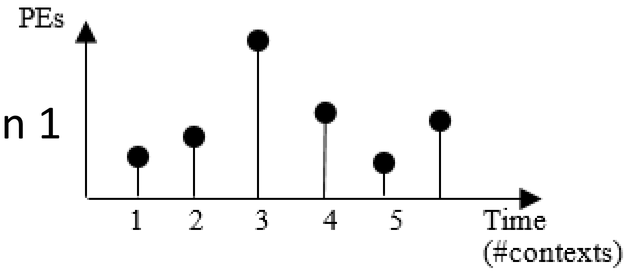
Actual Results

Proposed Flow Multi-Processes PE Sharing (MPPE Share)

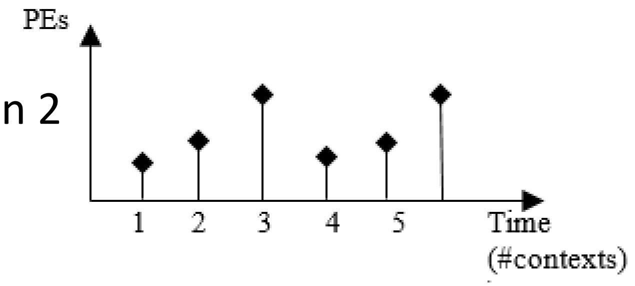
- Align contexts by setting different reset conditions
- 2 Phases:
  1. Single Process characterization
  2. PE alignment

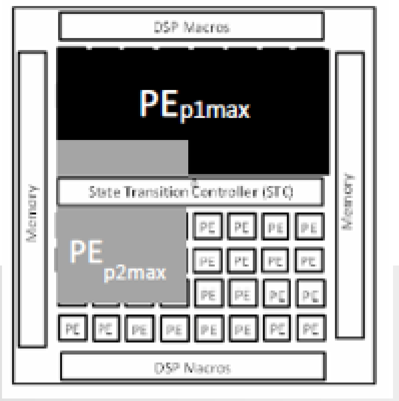


Application 1



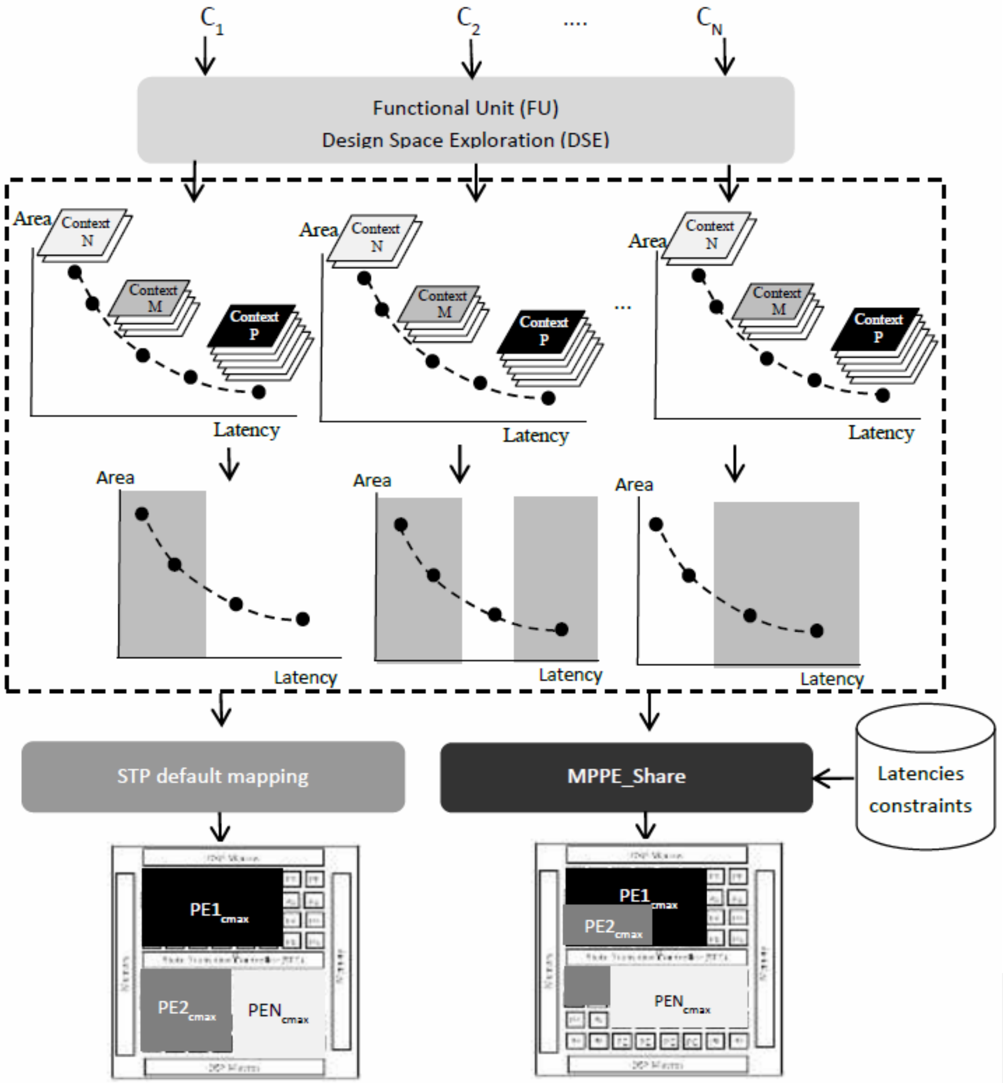
Application 2





Phase 1 – Single Process Characterization

- Perform HLS for each process in the system
- Annotate the PEs used in each context (location)
- FU design space exploration (optional):
  - Reduce the number of FUs by 20% to increase resource sharing and thus reduce the area

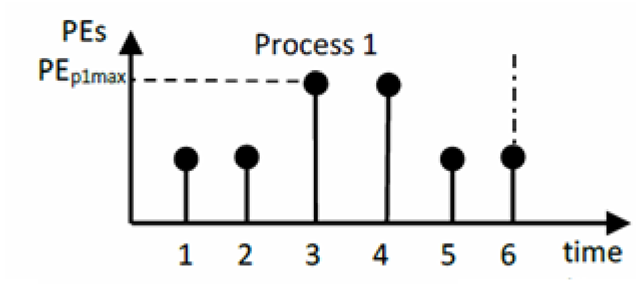


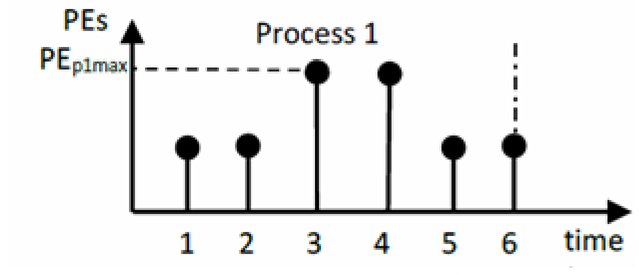
Phase 2: PE Allocation Method – Pruned Search

- 4 Steps
  - Step 1 : Latency adjustment
  - Step 2: PE Sharing Slack Computation (PESS)
  - Step 3:Extract contexts with Max and Min PE
  - Step 4: Context alignments
    - Find contexts with large PE usage and context of process with low PE usage
    - Check if alignment is valid → Manhattan distance computation

PE Allocation – Step 1 Latency Adjustment

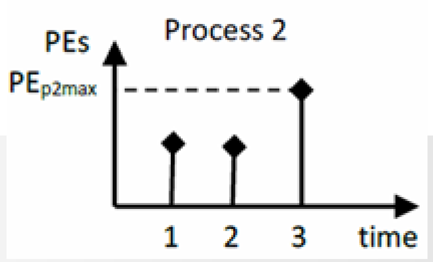
- In order to fully evaluate the effects of PEs sharing ➔make all processes of equal latency
- Computing the least common multiple (lcm)
- lcm smallest positive integer that is divisible by both *Latency A* of process 1 and Latency B of process 2
- Processes' contexts are extended N times :
  - Number copies process N = lcm/#contexts.

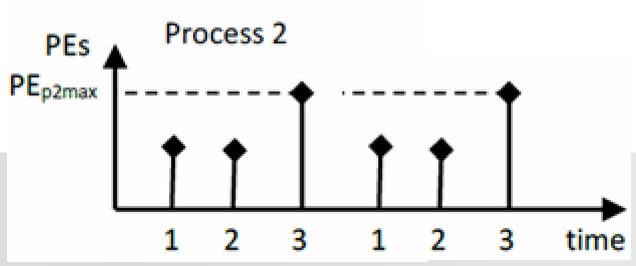




lcm =6

N=6/6=1



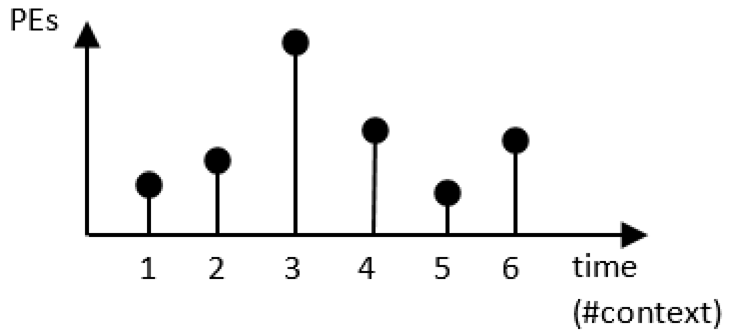


lcm =6

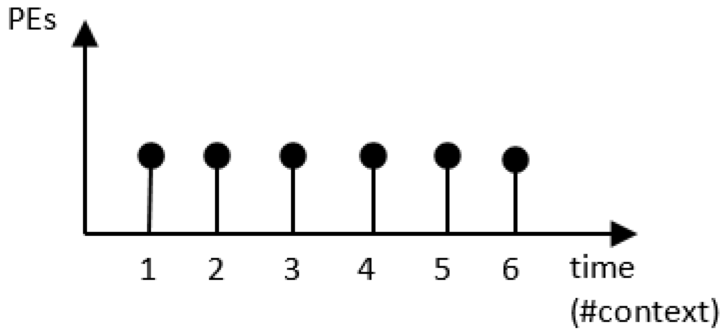
N=6/3=2

PE Allocation – Step 2 PE Sharing Slack

- PE sharing slack (PESS) = Variance in PE usage in each process
- Calculated for each process and sort processes based on PESS
- Used as pairing criteria for PE sharing
- The higher the PESS is, the
  - High PESS -> higher potential for possible sharing is
  - PESS = 0, then all the contexts require the same amount of PEs and there is no possibility for PE sharing



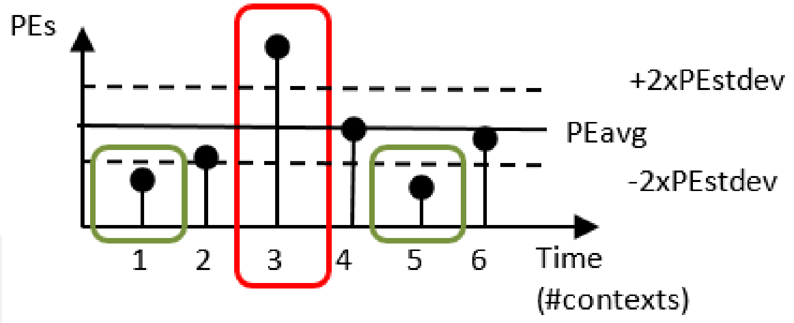
PESS = high

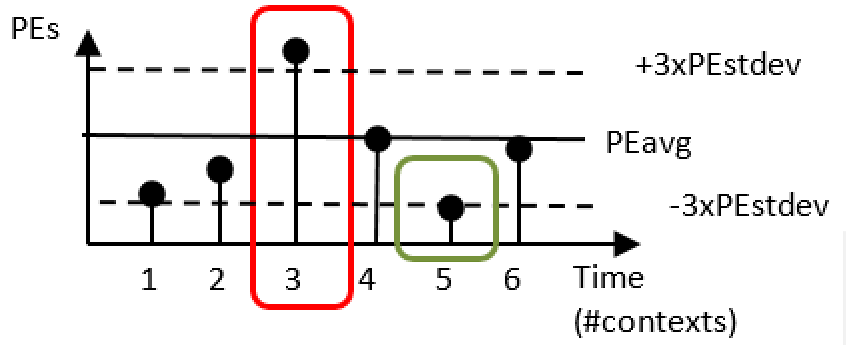


PESS = 0

PE Allocation – Step 3 Max/Min PE usage Contexts

- Extract from each process the *N* Contexts with highest PE usage and *M* contexts with lowest PE usage
- Two cases for N,M :
  - $PE_{avg} \pm 2 \times PE_{stdev}$ → More contexts in the list
  - $PE_{avg} \pm 3 \times PE_{stdev}$ → Less contexts in the list



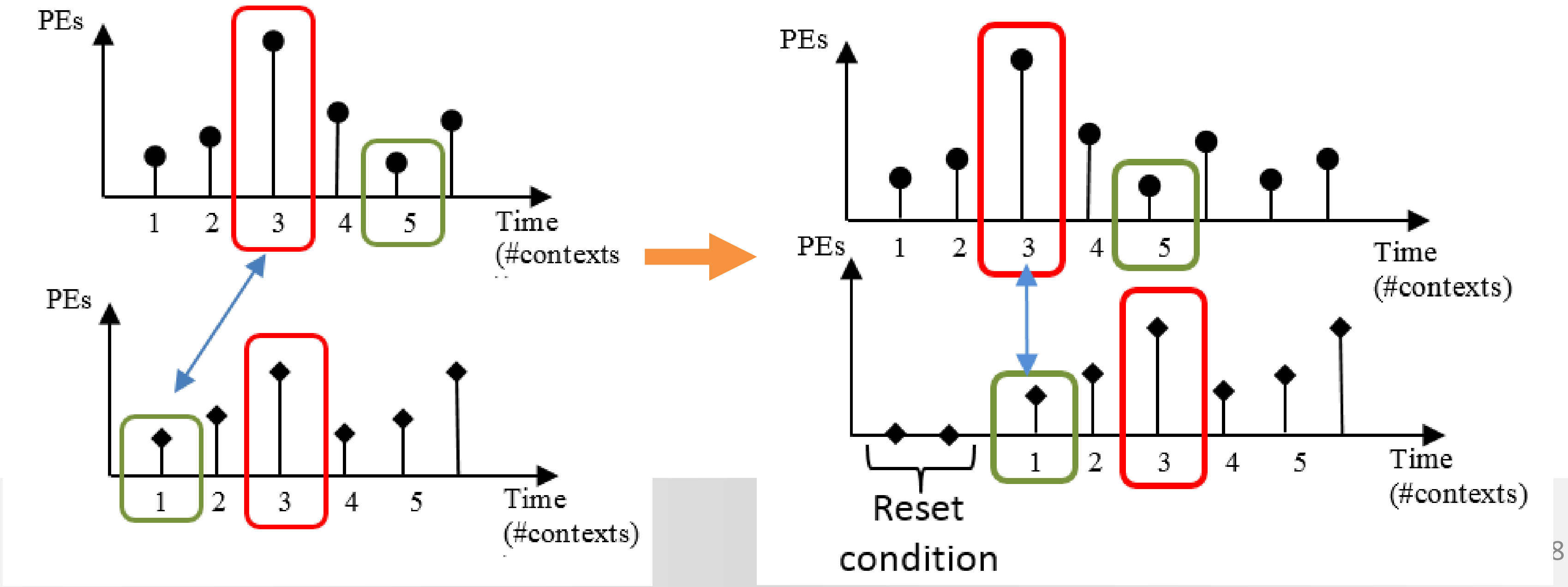


PE Allocation – Step 4 Context Alignments

- Find Context Alignment point by considering only the contexts selected in Step 3
- Pairs of processes with high slack are considered first
- Once the alignment of the contexts is done:

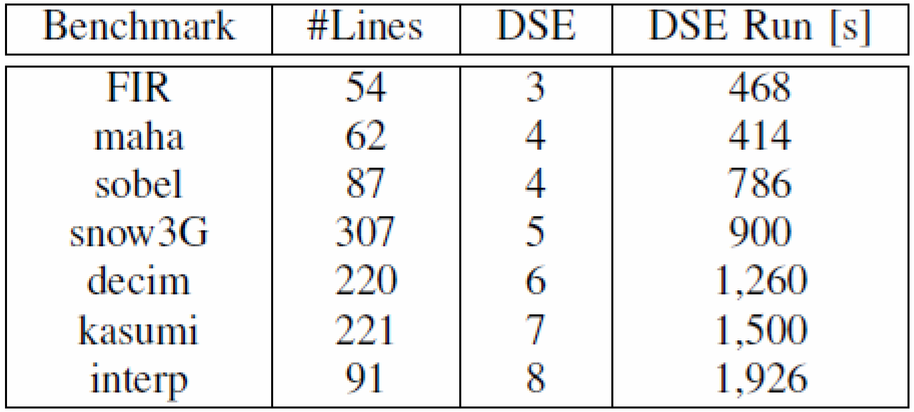
  → check if the PE assignment is valid or not

  → critical path is estimated based on the Manhattan distance of the longest path (due to the regularity of the CGRA it is easy to estimate this delay)

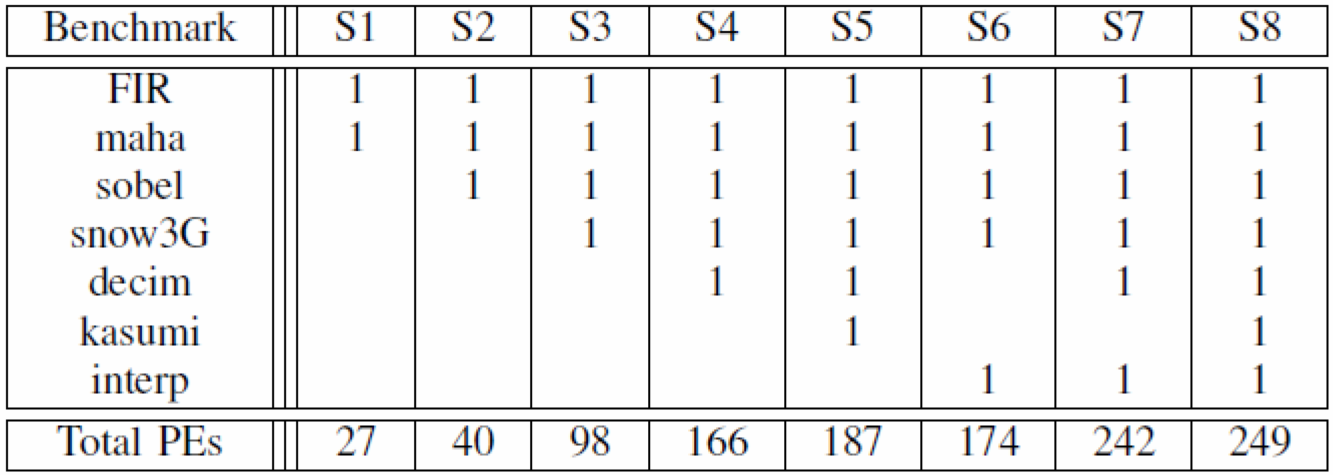


Experimental Results Setup

- 7 Synthesizable SystemC Benchmarks (www.S2CBench.org) rewritten in ANSI-C
- HLS using 50 MHz as target frequency (20ns)
- Renesas Electronics Musketeer 1.23



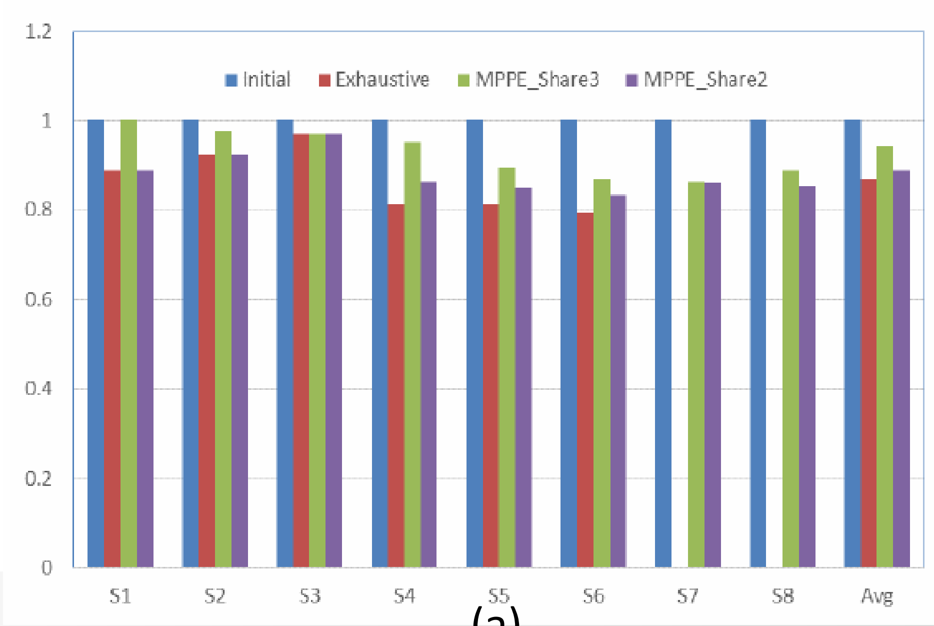
Benchmark	#Lines	DSE	DSE Run [s]
FIR	54	3	468
maha	62	4	414
sobel	87	4	786
snow3G	307	5	900
decim	220	6	1,260
kasumi	221	7	1,500
interp	91	8	1,926



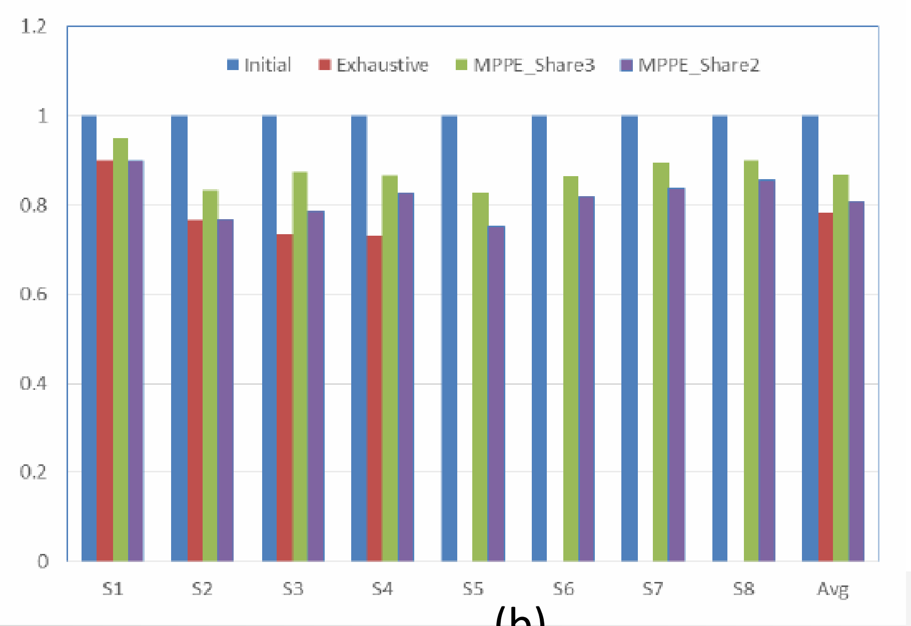
Benchmark	S1	S2	S3	S4	S5	S6	S7	S8
FIR	1	1	1	1	1	1	1	1
maha	1	1	1	1	1	1	1	1
sobel		1	1	1	1	1	1	1
snow3G			1	1	1	1	1	1
decim				1	1		1	1
kasumi					1			1
interp						1	1	1
Total PEs	27	40	98	166	187	174	242	249

Experimental Results

- Two set of experiments conducted
  - a) With latency constraint (the fastest design reported by the DSE is Used)
  - b) Without latency constraint (results of DSE used and smallest design reported)
- MPPE_share3 = =±3STDev, MPPE2=±2STDev
- Compared against
  - – initial configuration (initial) – No PE sharing
  - – Exhaustive search – Optimal solution (Executed for 5 days-didn't finish for s7 and s8)



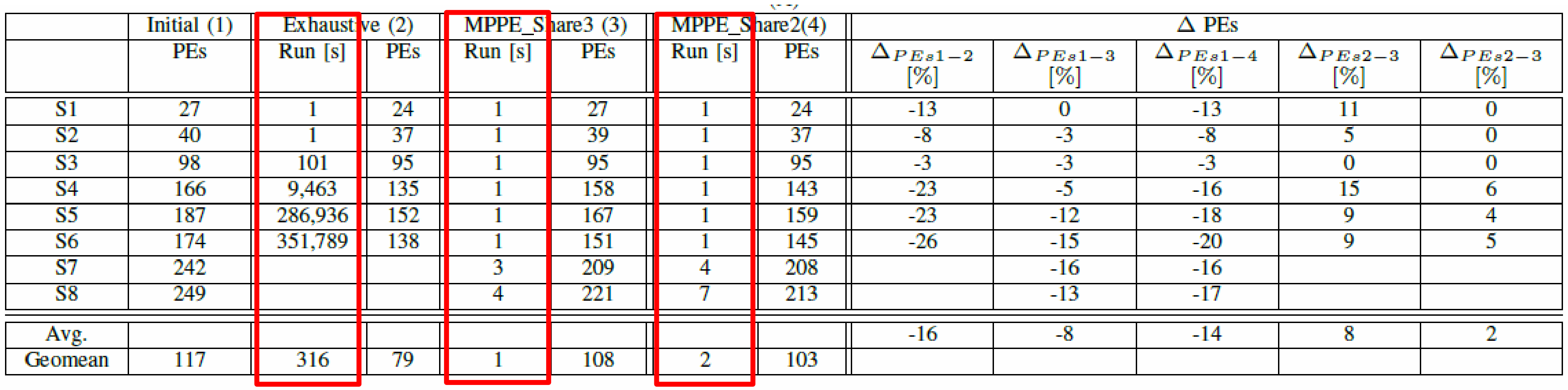
(a)
Single latency



(b)
DSE results

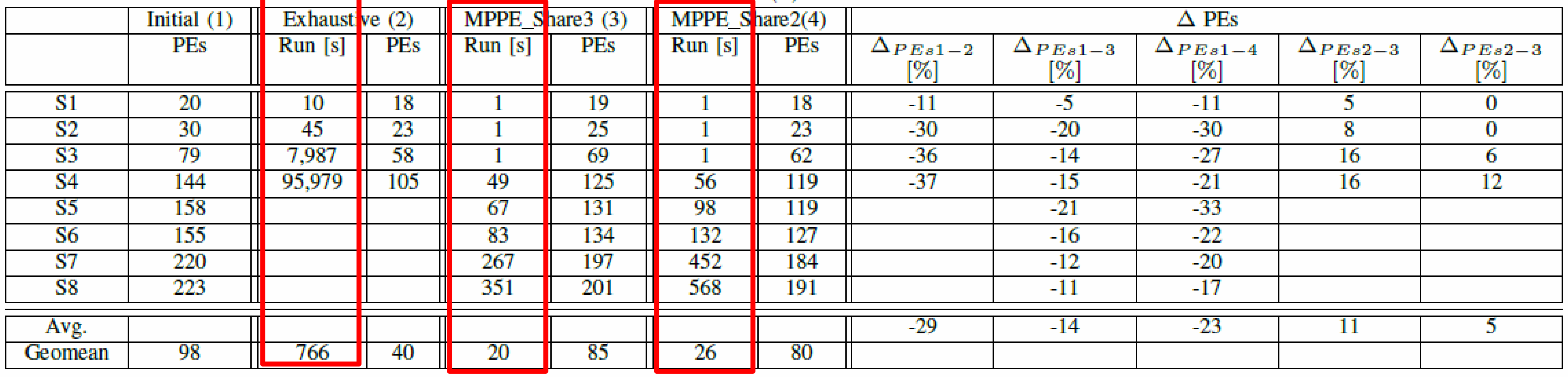
Experimental Results cont.

- Detailed experimental results



	Initial (1)	Exhaustive (2)		MPPE_Share3 (3)		MPPE_Share2(4)		Δ PEs				
	PEs	Run [s]	PEs	Run [s]	PEs	Run [s]	PEs	Δ_{PEs1-2} [%]	Δ_{PEs1-3} [%]	Δ_{PEs1-4} [%]	Δ_{PEs2-3} [%]	Δ_{PEs2-3} [%]
S1	27	1	24	1	27	1	24	-13	0	-13	11	0
S2	40	1	37	1	39	1	37	-8	-3	-8	5	0
S3	98	101	95	1	95	1	95	-3	-3	-3	0	0
S4	166	9,463	135	1	158	1	143	-23	-5	-16	15	6
S5	187	286,936	152	1	167	1	159	-23	-12	-18	9	4
S6	174	351,789	138	1	151	1	145	-26	-15	-20	9	5
S7	242			3	209	4	208		-16	-16		
S8	249			4	221	7	213		-13	-17		
Avg.								-16	-8	-14	8	2
Geomean	117	316	79	1	108	2	103					

(B)



	Initial (1)	Exhaustive (2)		MPPE_Share3 (3)		MPPE_Share2(4)		Δ PEs				
	PEs	Run [s]	PEs	Run [s]	PEs	Run [s]	PEs	Δ_{PEs1-2} [%]	Δ_{PEs1-3} [%]	Δ_{PEs1-4} [%]	Δ_{PEs2-3} [%]	Δ_{PEs2-3} [%]
S1	20	10	18	1	19	1	18	-11	-5	-11	5	0
S2	30	45	23	1	25	1	23	-30	-20	-30	8	0
S3	79	7,987	58	1	69	1	62	-36	-14	-27	16	6
S4	144	95,979	105	49	125	56	119	-37	-15	-21	16	12
S5	158			67	131	98	119		-21	-33		
S6	155			83	134	132	127		-16	-22		
S7	220			267	197	452	184		-12	-20		
S8	223			351	201	568	191		-11	-17		
Avg.								-29	-14	-23	11	5
Geomean	98	766	40	20	85	26	80					

Summary and Conclusions

- Presented a method for sharing PE in multi-processes systems synthesized individually in runtime reconfigurable CGRA
- Input each synthesized design or DSE trade-off curve
- Average PE savings of 14% and only 2% worse than the optimal solution for fixed latency.