Platform-Agnostic Dataflow-to-Hardware Design Flow for Reconfigurable Systems

Francesca Palumbo[1], Claudio Rubattu[1,2], **Carlo Sau[3]**, Luigi Raffo[3] and Maxime Pelcat[2]

[1]University of Sassari, IDEA Lab

[2]University of Rennes, INSA

[3]University of Cagliari, DIEE, EOLAB

Naples, 20-22 June 2018

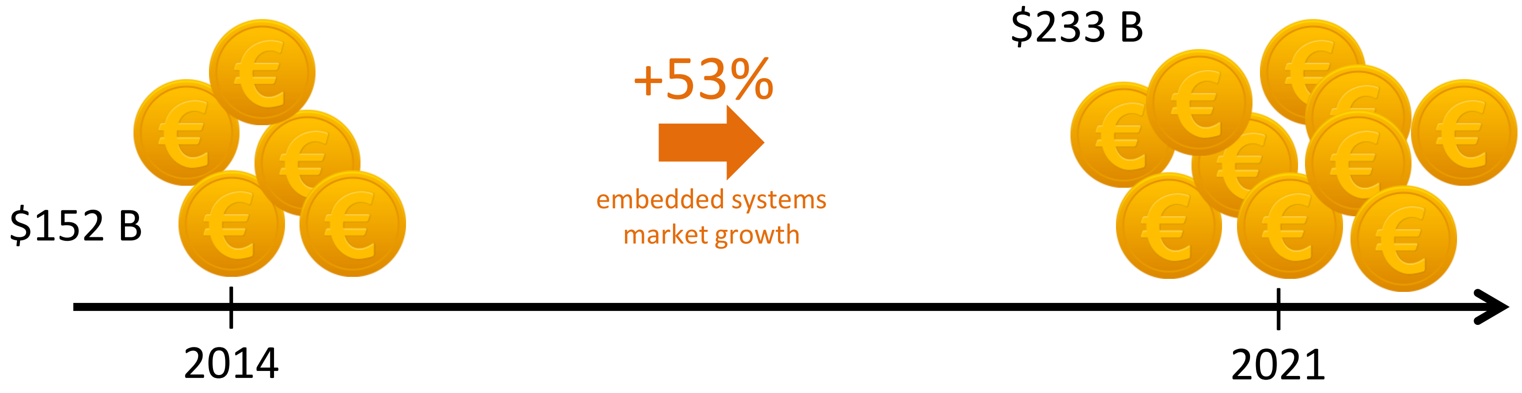
Outline

Outline

Introduction

Embedded Moving Toward IoT and CPS

Embedded Systems (*real-time* computing systems with a dedicated functionality), often working in *constrained* and *portable* contexts, are pervading the market.



source: Transparency Market Research

Introduction

Embedded Moving Toward IoT and CPS

Embedded Systems (*real-time* computing systems with a dedicated functionality), often working in *constrained* and *portable* contexts, are pervading the market.

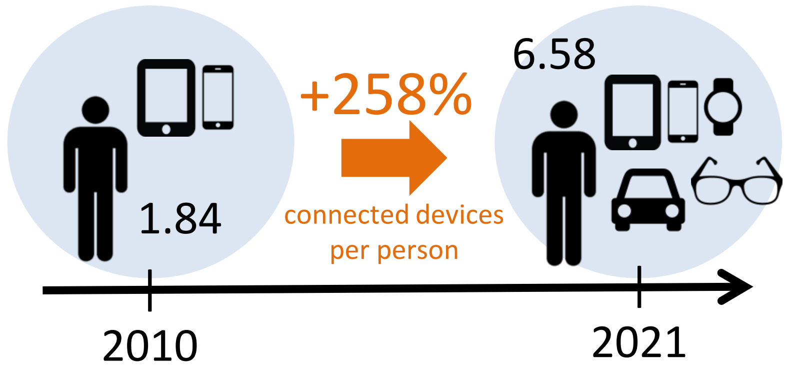
HIGH PERFORMANCE required to achieve real-time behaviour. Due to the constrained environment and to the portability an **EXTREME EXECUTION EFFICIENCY (RESOURCES, POWER CONSUMPTION)** is mandatory.

source: Transparency Market Research

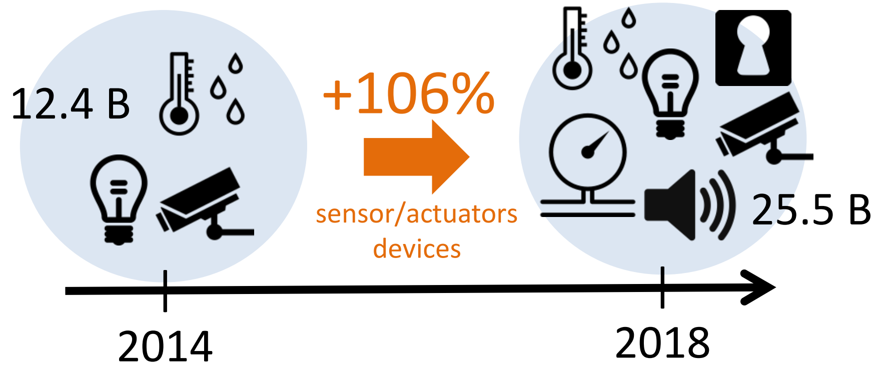
Introduction

Embedded Moving Toward IoT and CPS

Embedded Systems are becoming *connected and capable of interacting* with *Environment*, the *User* and the *System* itself, to *adapt* their behaviour according to changing requirements.



source: CISCO ISBG



source: IC Insights

Introduction

Embedded Moving Toward IoT and CPS

Embedded Systems are becoming *connected and capable of interacting* with *Environment*, the *User* and the *System* itself, to *adapt* their behaviour according to changing requirements.

Adaptivity pushes farther **FLEXIBILITY** need of the systems that have to provide several working points while, at the same time, exhibiting a **SHORT RESPONSE TIME**.

source: CISCO ISBG

source: IC Insights

Introduction

Computing Architectures for Embedded Systems

Emerging needs for modern embedded systems:

- high performance
- execution efficiency
- flexibility
- short response time

Introduction

Computing Architectures for Embedded Systems

Emerging needs for modern embedded systems:

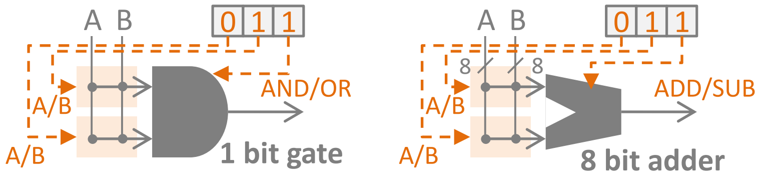
- high performance
- execution efficiency
- flexibility
- short response time
- minimum time to market

Introduction

Computing Architectures for Embedded Systems

Emerging needs for modern embedded systems:

- high performance
- execution efficiency
- flexibility
- short response time
- minimum time to market



	FGRA	CGRA
reconf granularity	bit-level	word-level
flexibility	☺	😐
reconf speed	😐	☺

CPU

Central Processing Unit

GPU

Graphics Processing Unit

DSP

Digital Signal Processor

RA

Reconfigurable Architectures

ASIC

Application Specific Integrated Circuit

flexibility ⟷ **performance/efficiency**

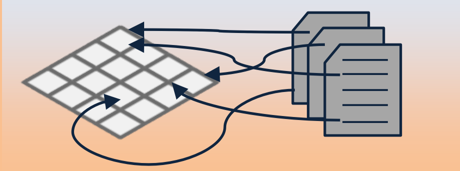
Introduction

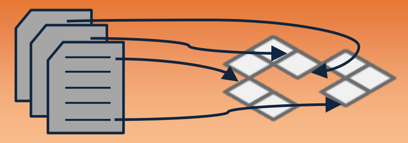
Computing Architectures for Embedded Systems

Emerging needs for modern embedded systems:

- high performance
- execution efficiency
- flexibility
- short response time
- minimum time to market

CGRA design approach





	CGRA	
	word-level	**reconf granularity**
	😐	**flexibility**
	☺	**reconf speed**

CPU

Central Processing Unit

GPU

Graphics Processing Unit

DSP

Digital Signal Processor

RA

Reconfigurable Architectures

ASIC

Application Specific Integrated Circuit

flexibility ⟵⟶ **performance/efficiency**

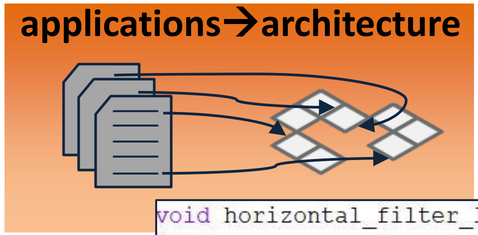
Outline

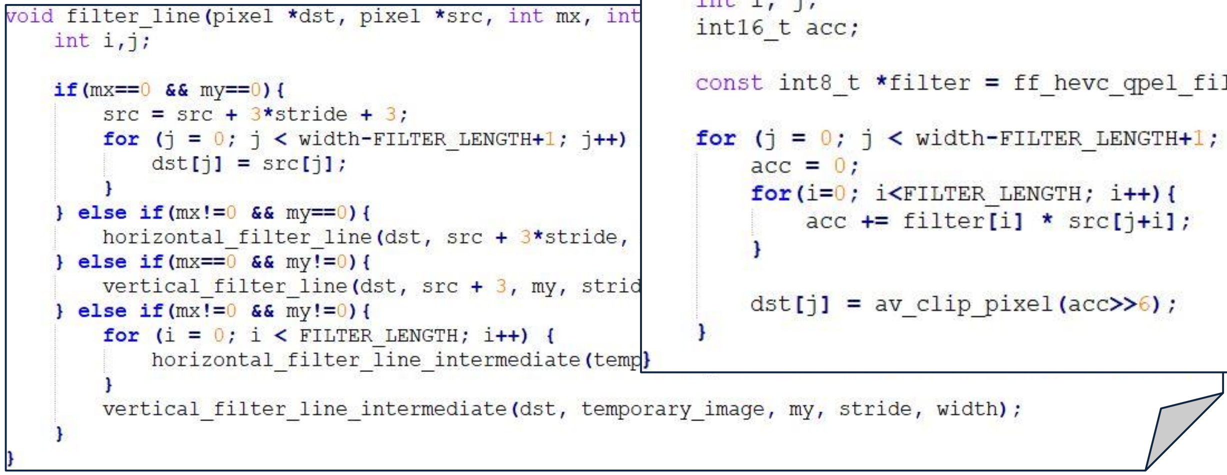
Background

CGRA Design From Applications to Architecture

Imperative

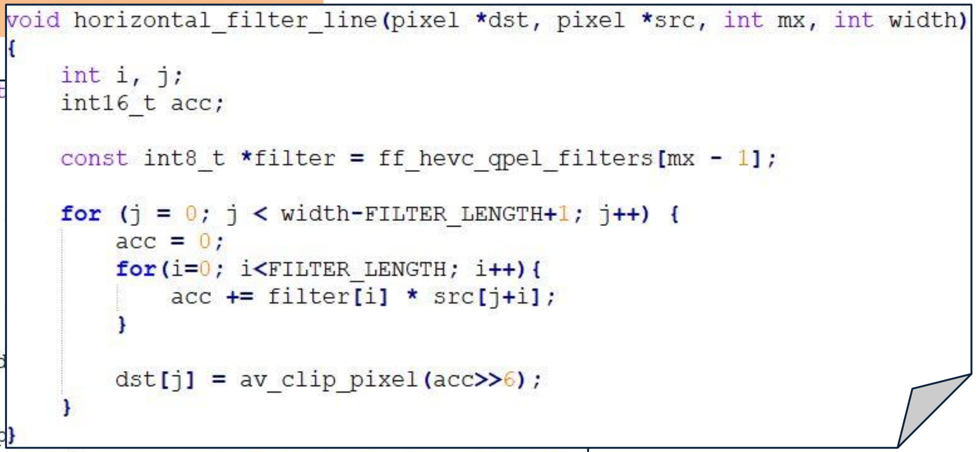
High-Level Specifications





```
void filter_line(pixel *dst, pixel *src, int mx, int
    int i,j;

    if(mx==0 && my==0){
        src = src + 3*stride + 3;
        for (j = 0; j < width-FILTER_LENGTH+1; j++)
            dst[j] = src[j];
        }
    } else if(mx!=0 && my==0){
        horizontal_filter_line(dst, src + 3*stride,
    } else if(mx==0 && my!=0){
        vertical_filter_line(dst, src + 3, my, strid
    } else if(mx!=0 && my!=0){
        for (i = 0; i < FILTER_LENGTH; i++) {
            horizontal_filter_line_intermediate(temp
        }
        vertical_filter_line_intermediate(dst, temporary_image, my, stride, width);
    }
}
```

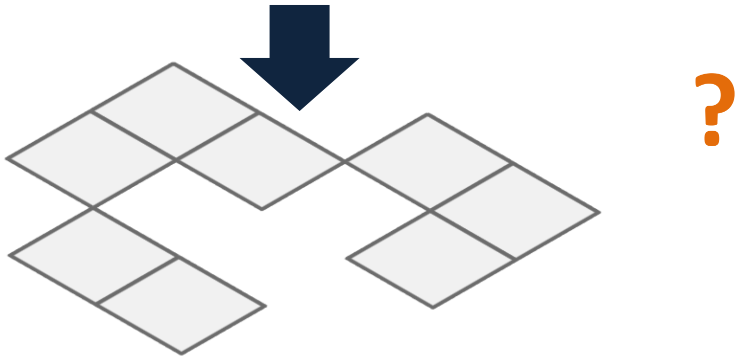


```
void horizontal_filter_line(pixel *dst, pixel *src, int mx, int width)
{
    int i, j;
    int16_t acc;

    const int8_t *filter = ff_hevc_qpel_filters[mx - 1];

    for (j = 0; j < width-FILTER_LENGTH+1; j++) {
        acc = 0;
        for(i=0; i<FILTER_LENGTH; i++){
            acc += filter[i] * src[j+i];
        }

        dst[j] = av_clip_pixel(acc>>6);
    }
```

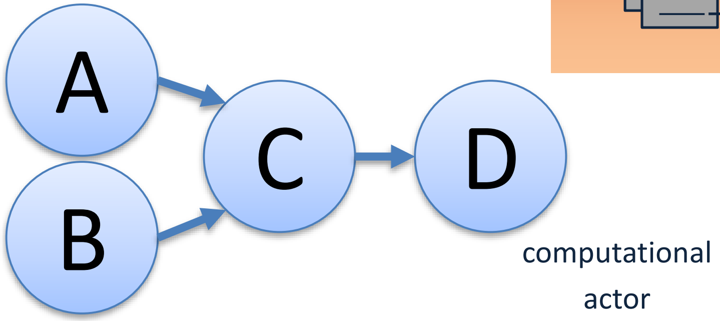


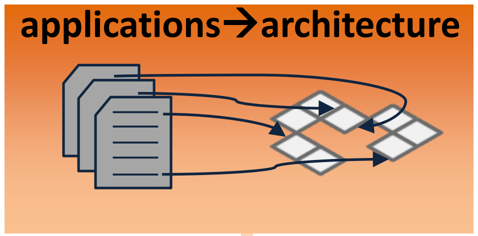
Background

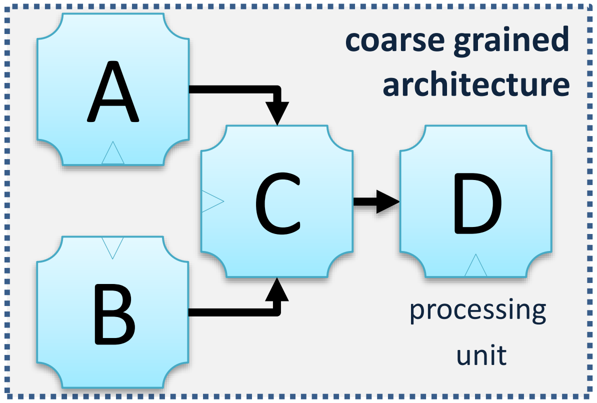
CGRA Design From Applications to Architecture

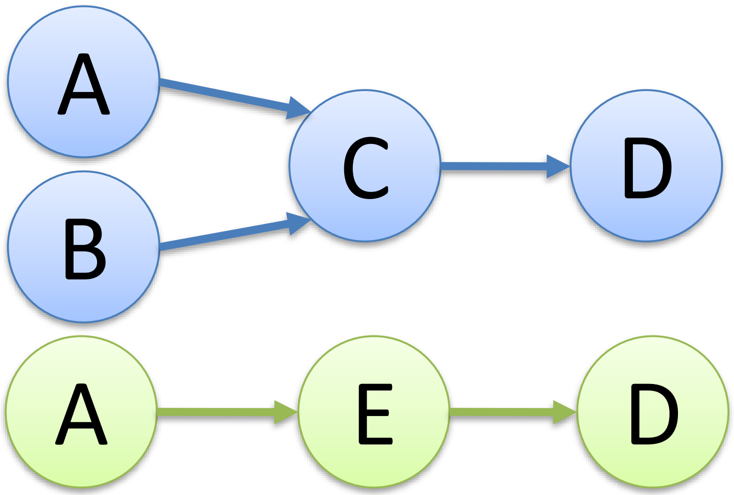
Functional

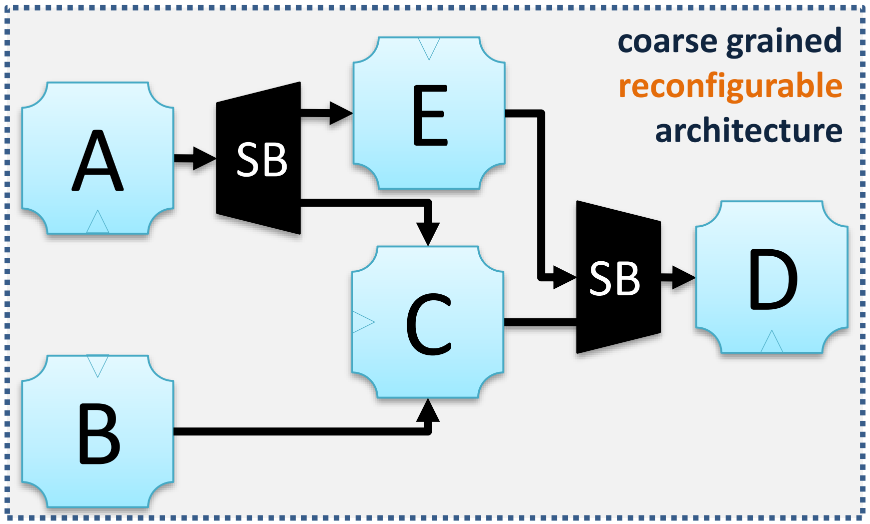
High-Level Specifications





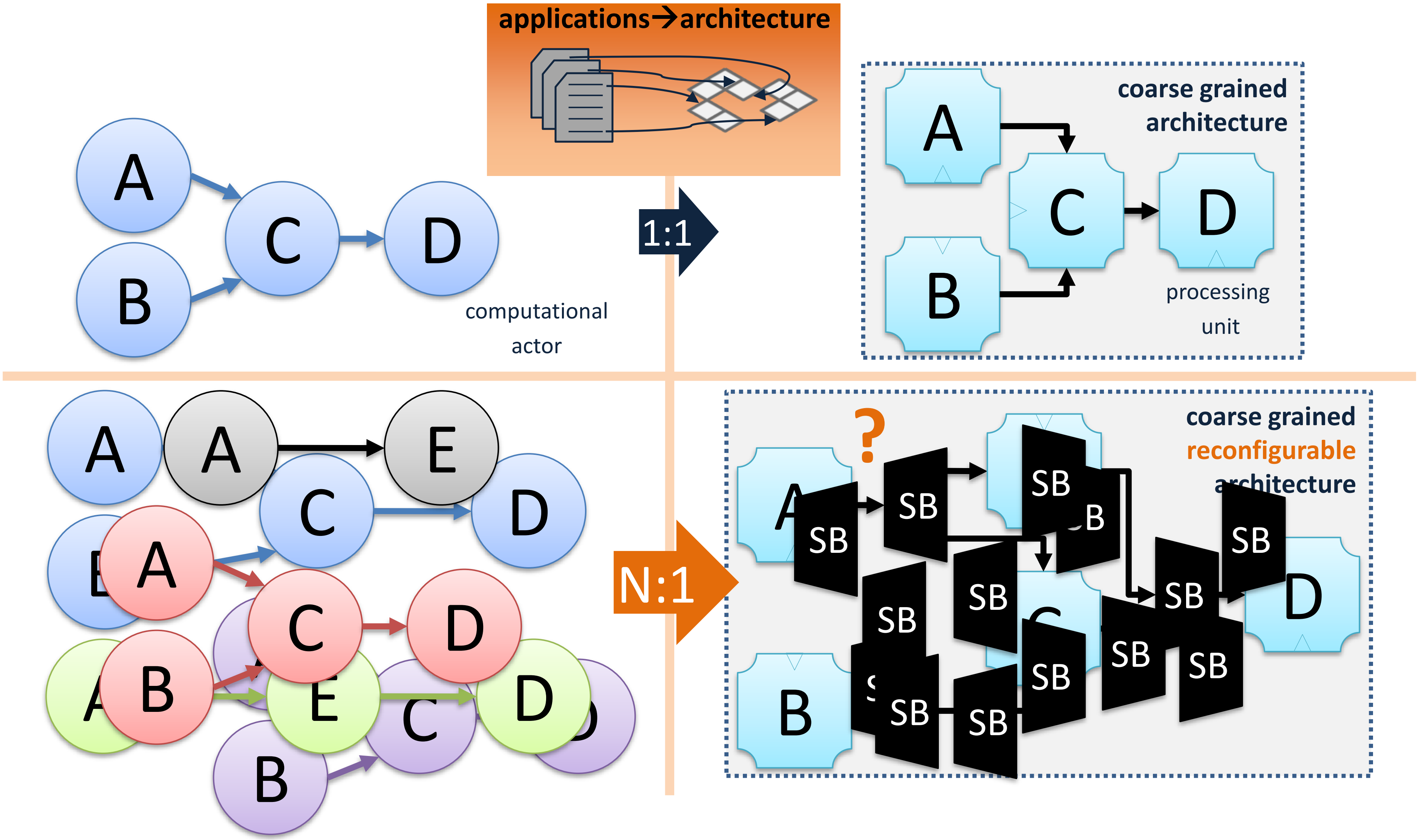






Background

CGRA Design Open Issues

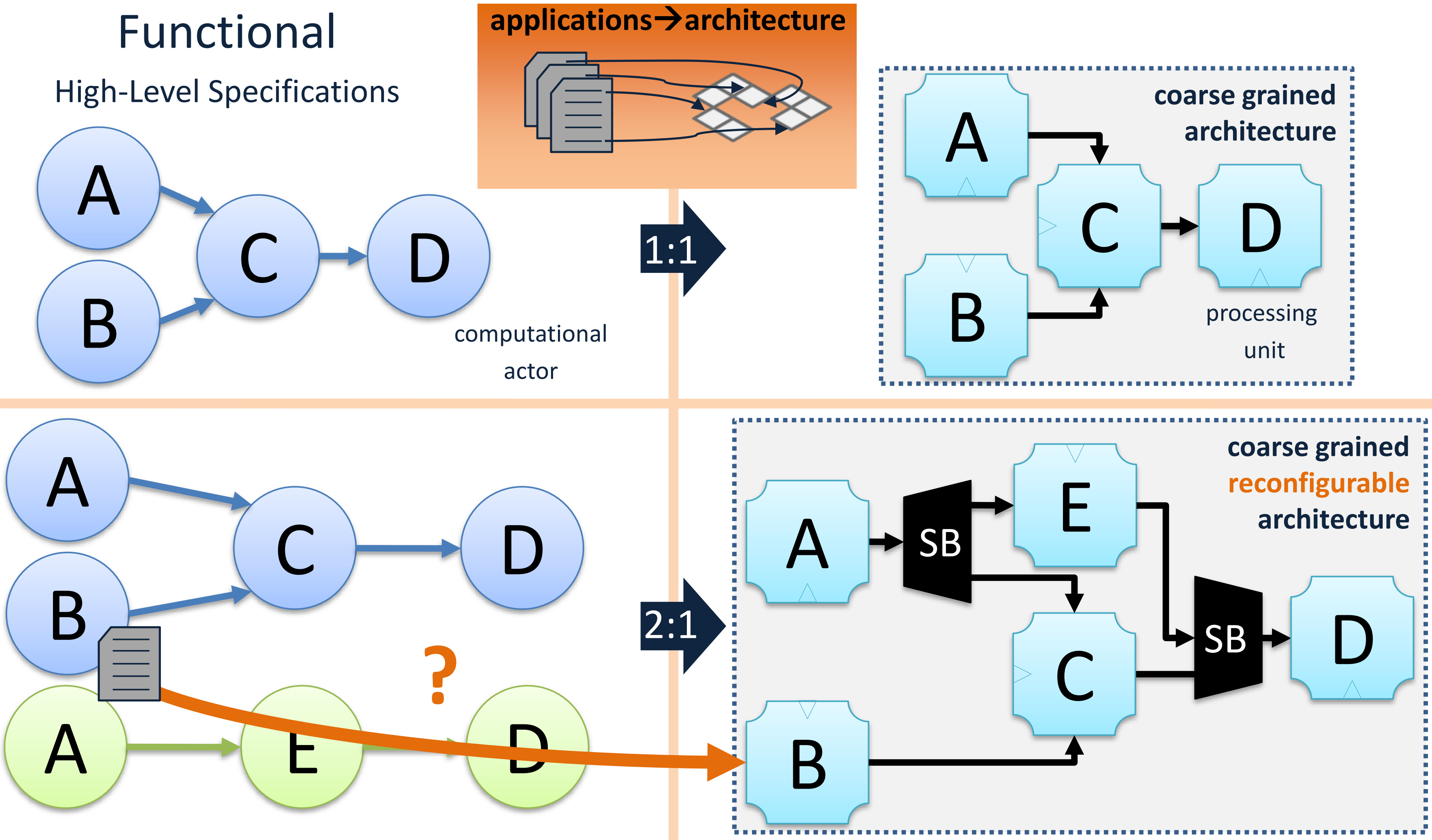

applications→architecture
A
B
C
D
computational actor
1:1
coarse grained architecture
A
C
D
B
processing unit
A
A
E
C
D
A
C
D
B
E
C
D
B
N:1
?
coarse grained reconfigurable architecture
SB
A
B
C
D

Background

CGRA Design Open Issues

Functional

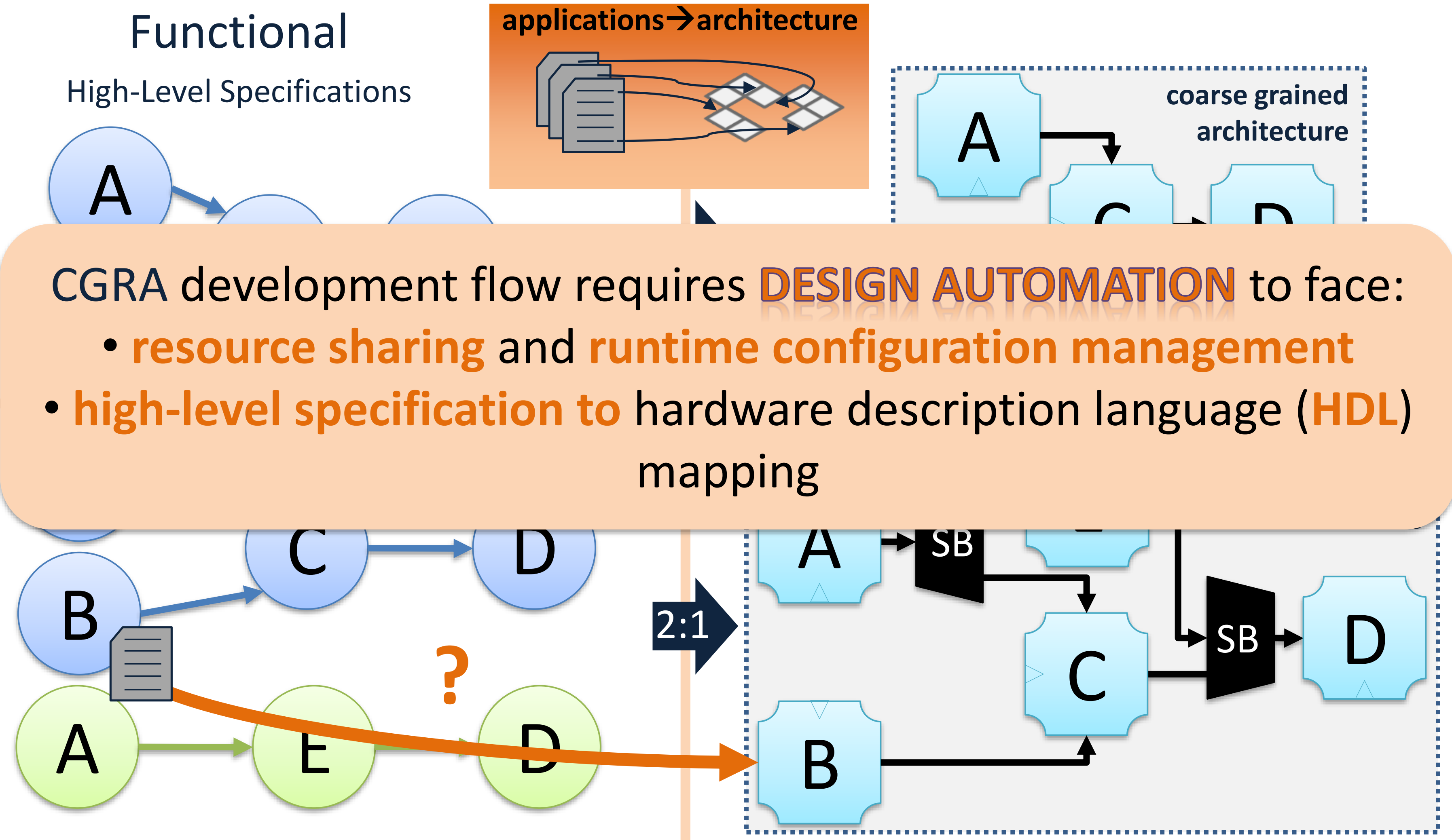
High-Level Specifications



applications→architecture
A
B
C
D
computational actor
1:1
coarse grained architecture
A
B
C
D
processing unit
A
B
C
D
A
E
D
?
2:1
coarse grained reconfigurable architecture
A
SB
E
C
SB
D
B

Background

CGRA Design Open Issues

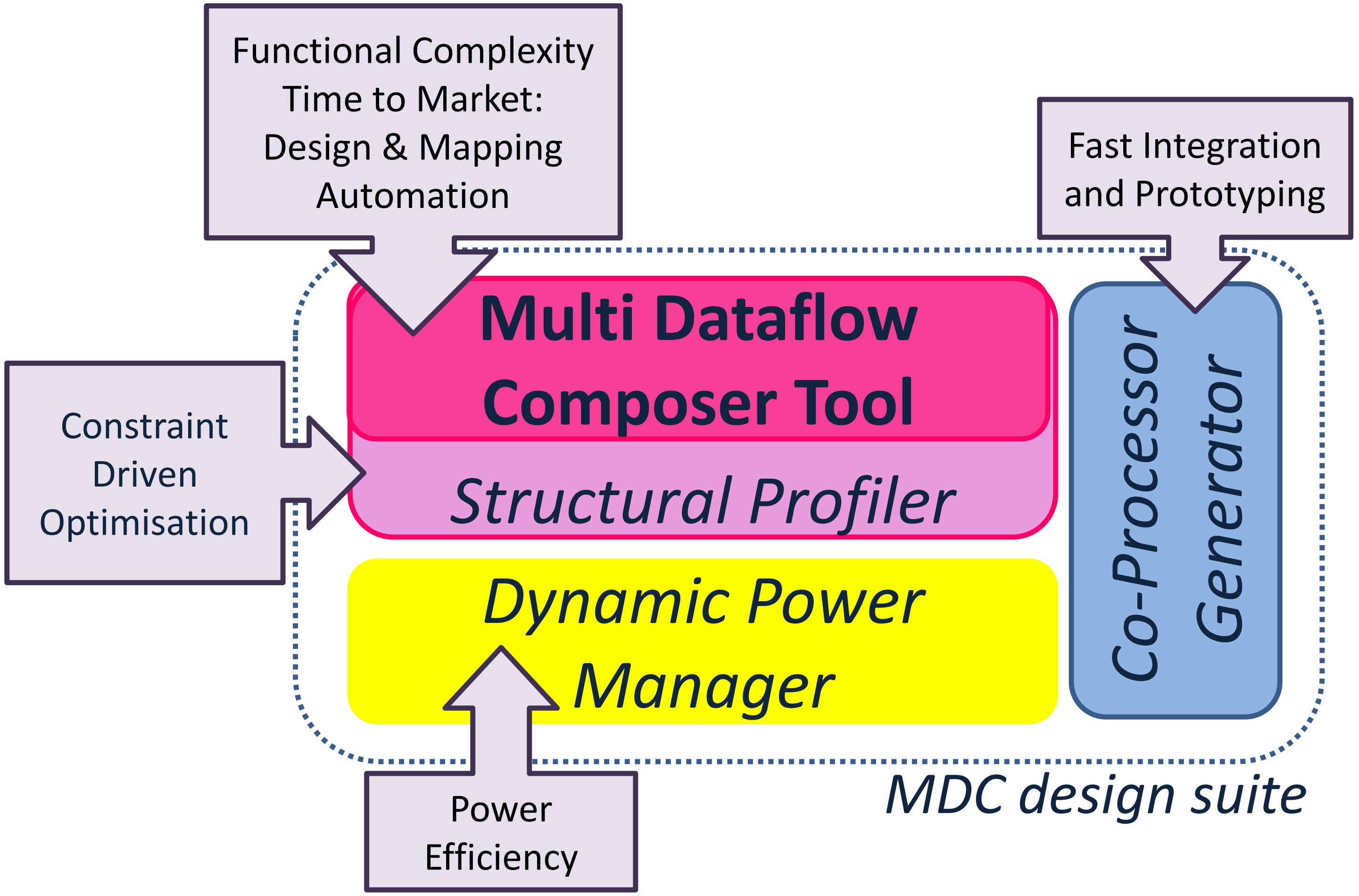

Functional
High-Level Specifications
applications→architecture
coarse grained architecture
A
C
D
B
A
E
D
?
2:1
A
SB
C
SB
D
B

CGRA development flow requires **DESIGN AUTOMATION** to face:

- **resource sharing** and **runtime configuration management**
- **high-level specification to** hardware description language (**HDL**) mapping

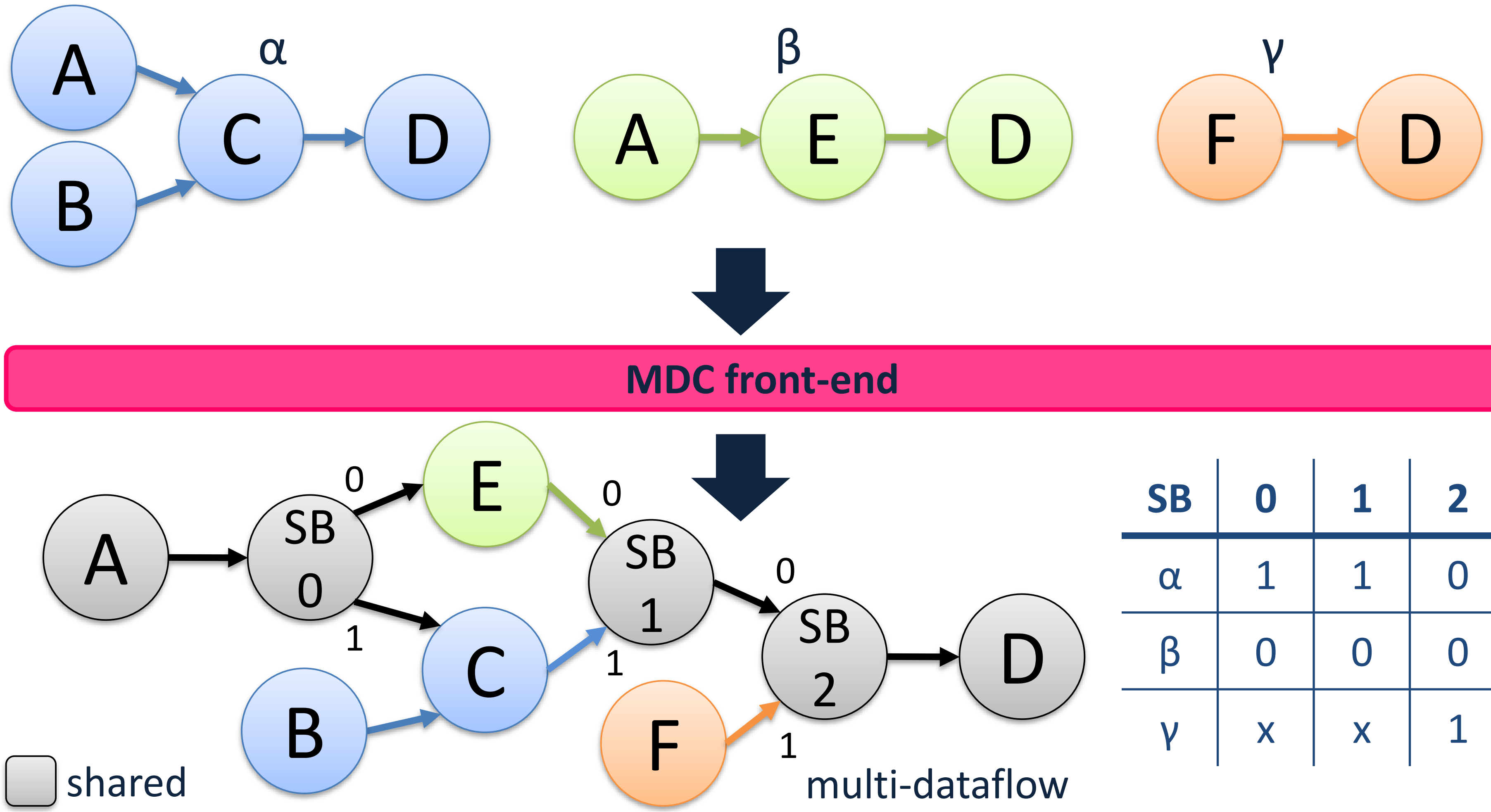
Outline

CGRA Composition and Management



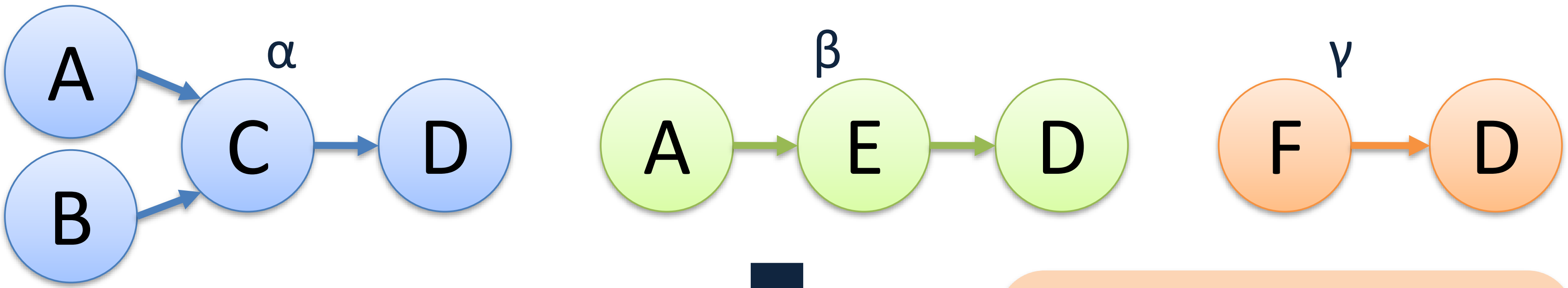
http://sites.unica.it/rpct/

CGRA Composition and Management



SB	0	1	2
α	1	1	0
β	0	0	0
γ	x	x	1

CGRA Composition and Management

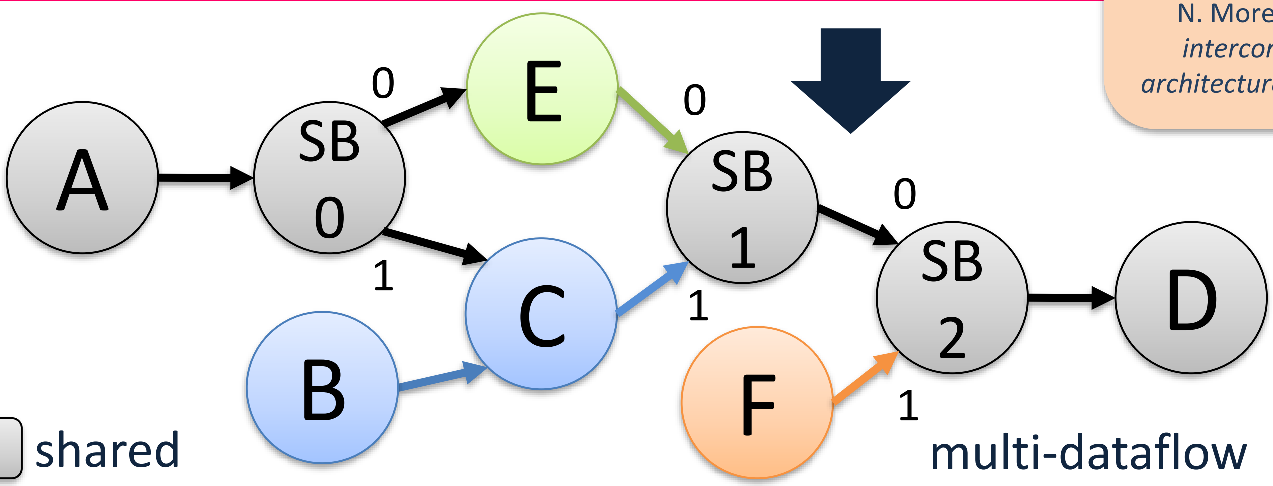


MDC front-end

DATAPATH MERGING PROBLEM

NP-complete solved with Moreano's heuristic method:

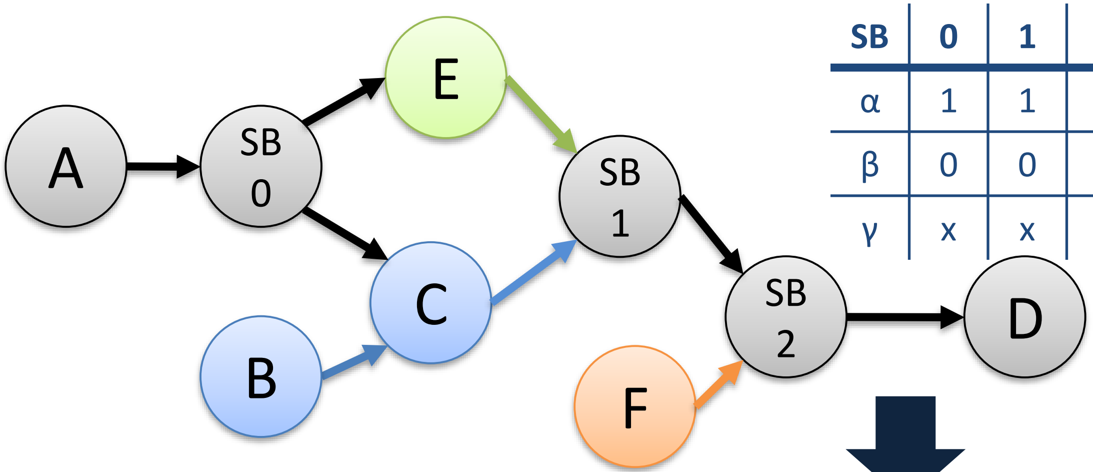
N. Moreano, et al., "*Datapath merging and interconnection sharing for reconfigurable architectures*", Symp. On *System Synthesis, 2002.*



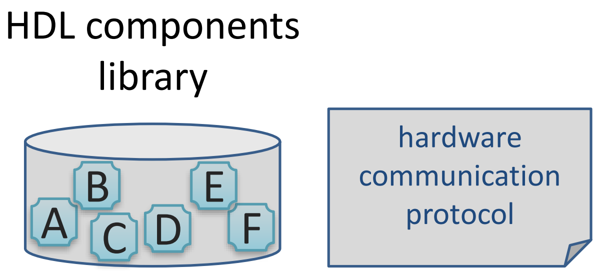
⬜ shared

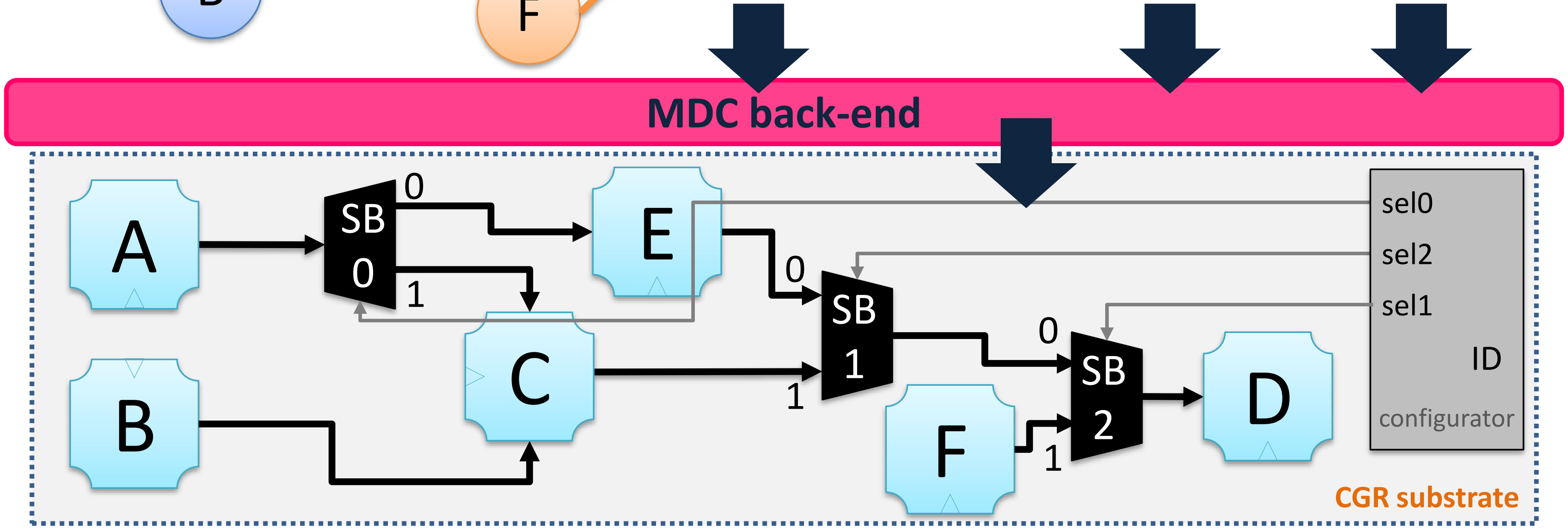
α	1	1	0
β	0	0	0
γ	x	x	1

CGRA Composition and Management

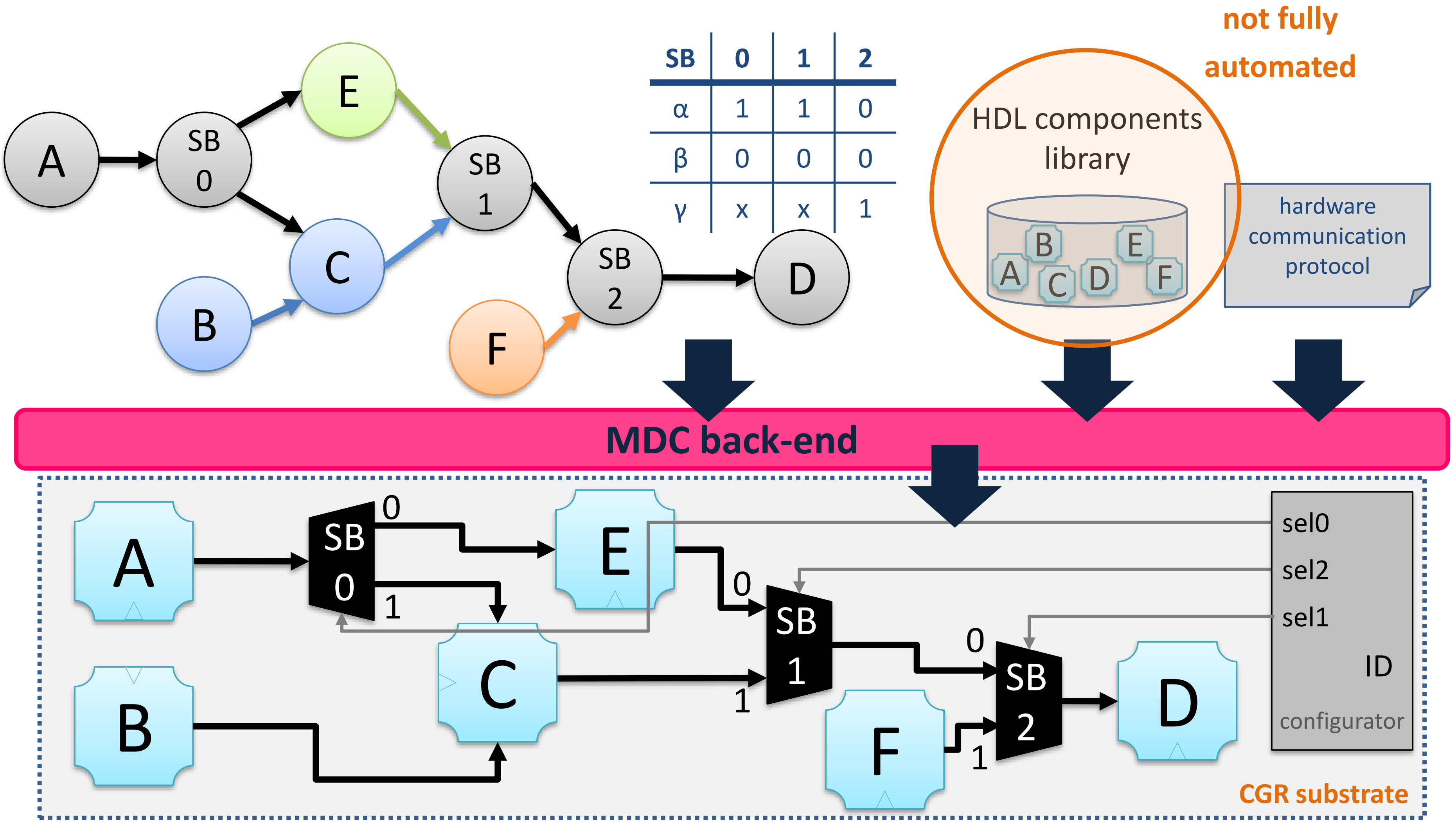


SB	0	1	2
α	1	1	0
β	0	0	0
γ	x	x	1





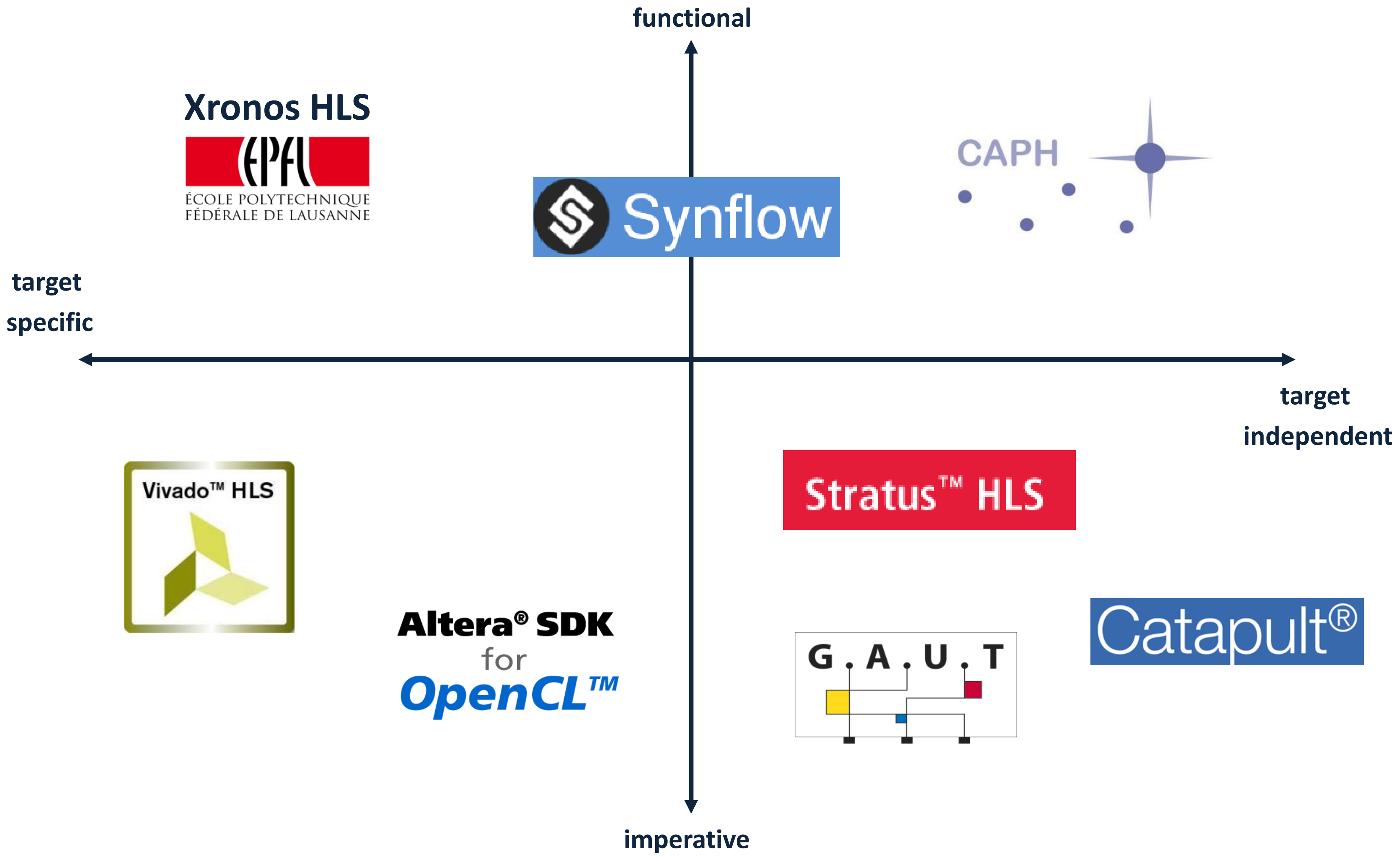
CGRA Composition and Management



SB	0	1	2
α	1	1	0
β	0	0	0
γ	x	x	1

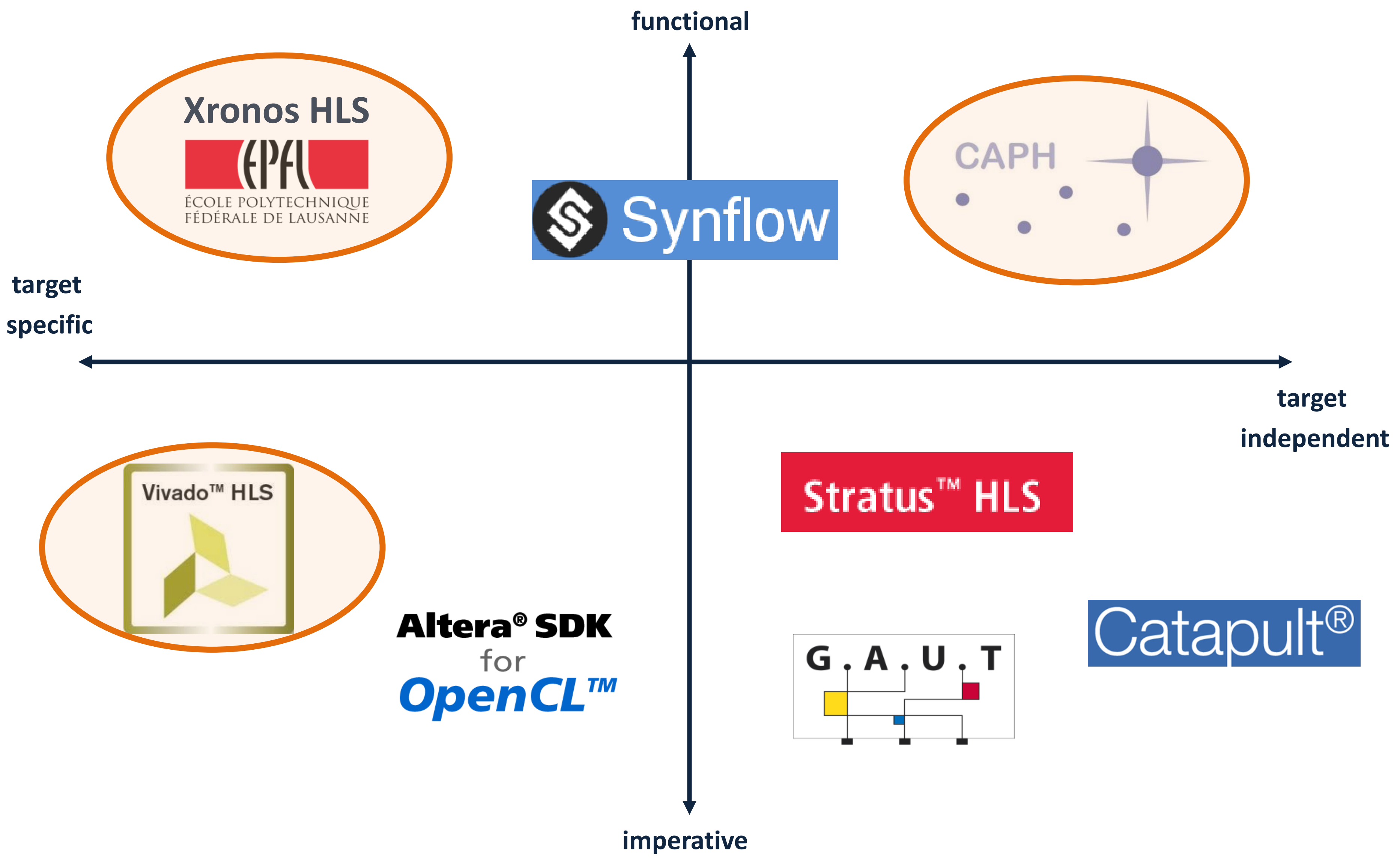
CGRA Composition and Management

High Level Synthesis



CGRA Composition and Management

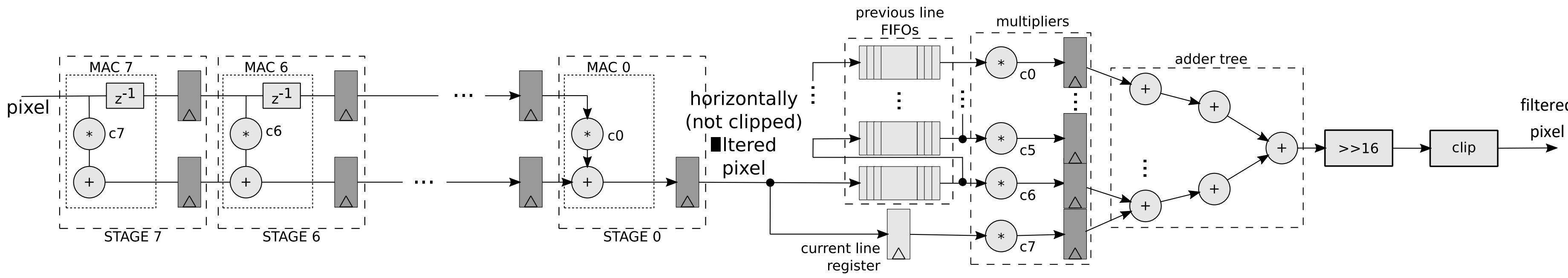
High Level Synthesis



Outline

Results: Functional vs. Imperative HLS for CGRA

Test Case: HEVC Fractional Pixel Interpolators

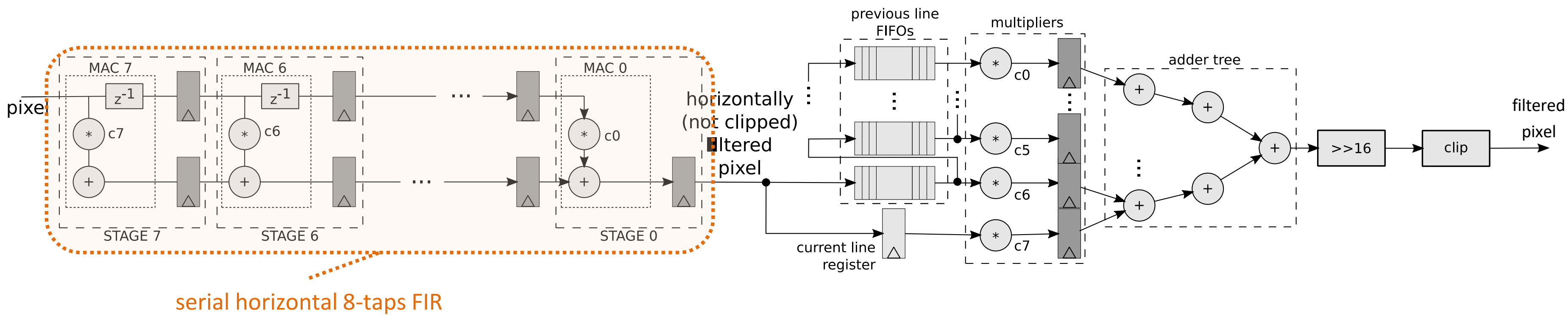


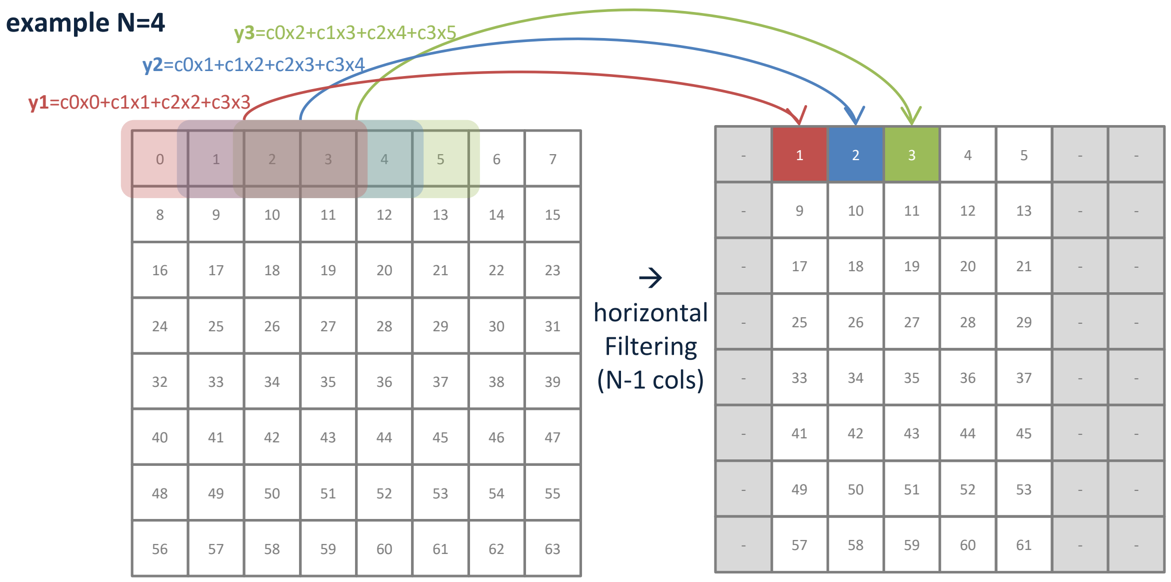
example N=4

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

Results: Functional vs. Imperative HLS for CGRA

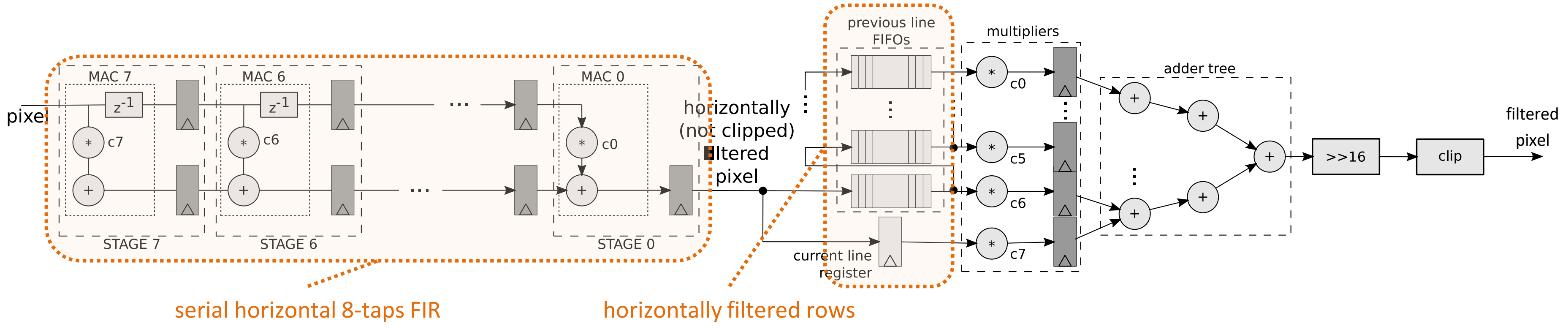
Test Case: HEVC Fractional Pixel Interpolators



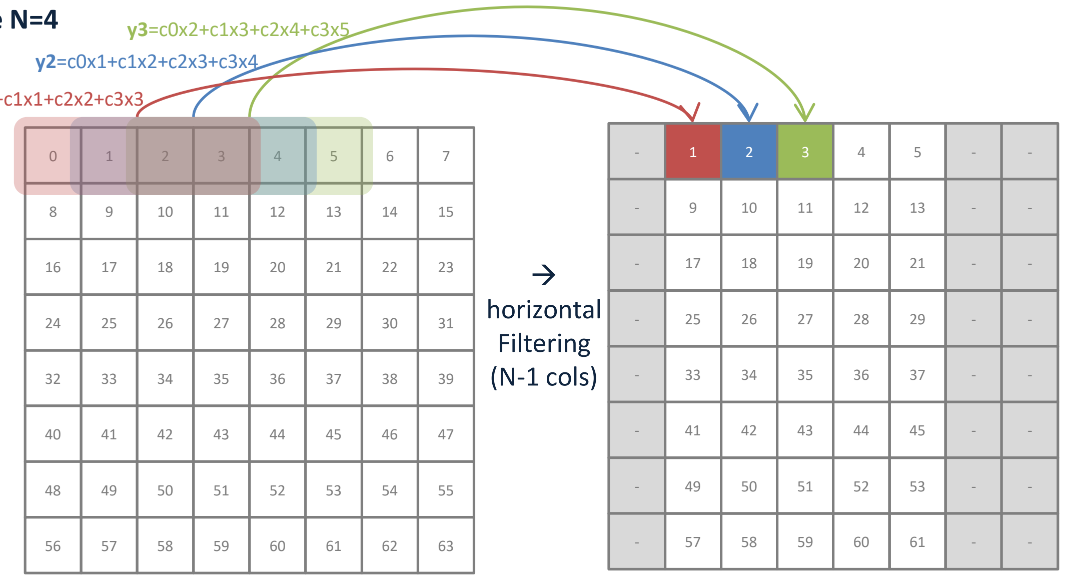


Results: Functional vs. Imperative HLS for CGRA

Test Case: HEVC Fractional Pixel Interpolators

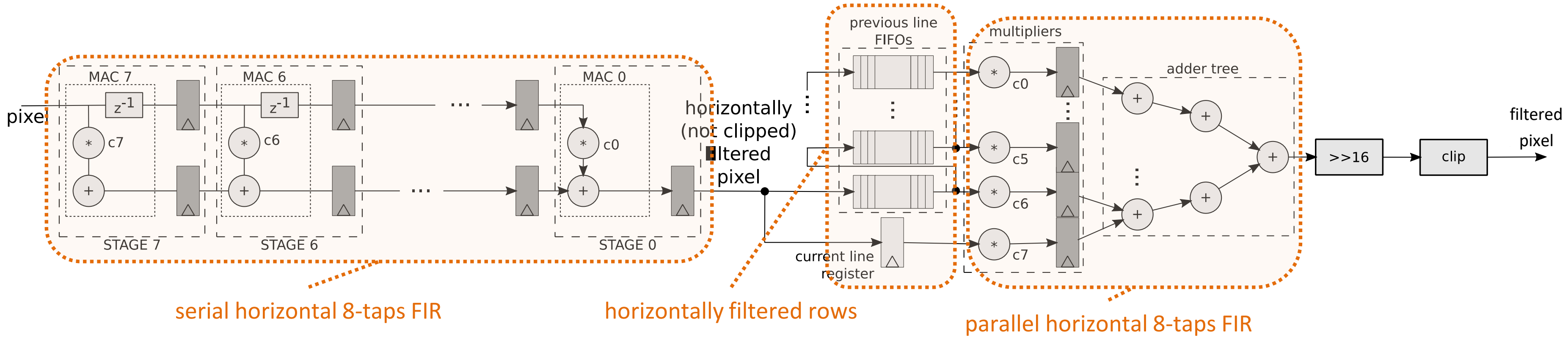


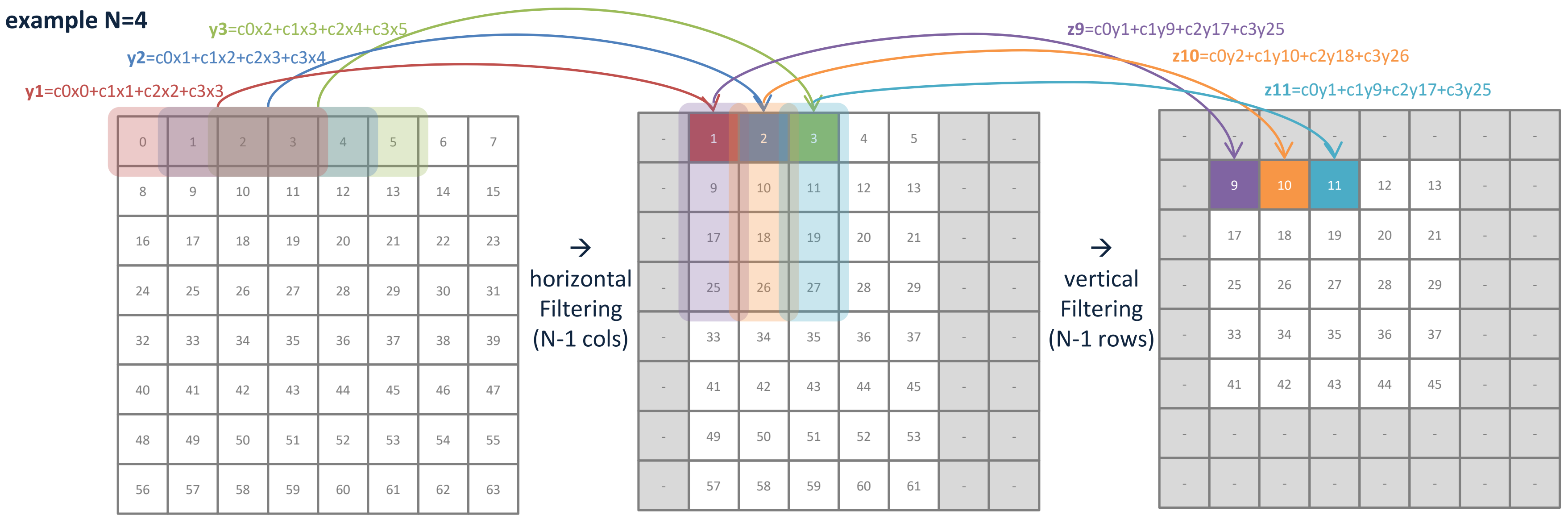
example N=4



Results: Functional vs. Imperative HLS for CGRA

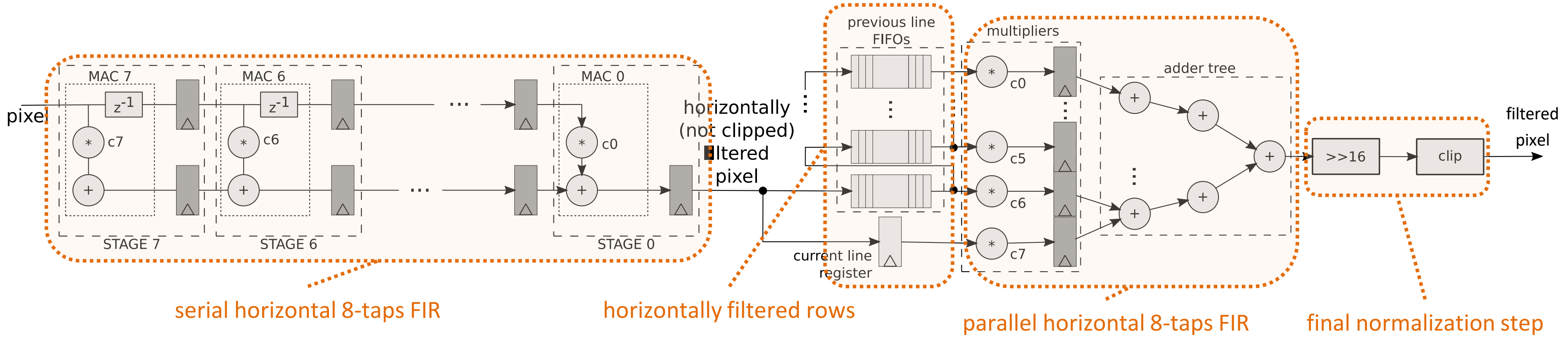
Test Case: HEVC Fractional Pixel Interpolators

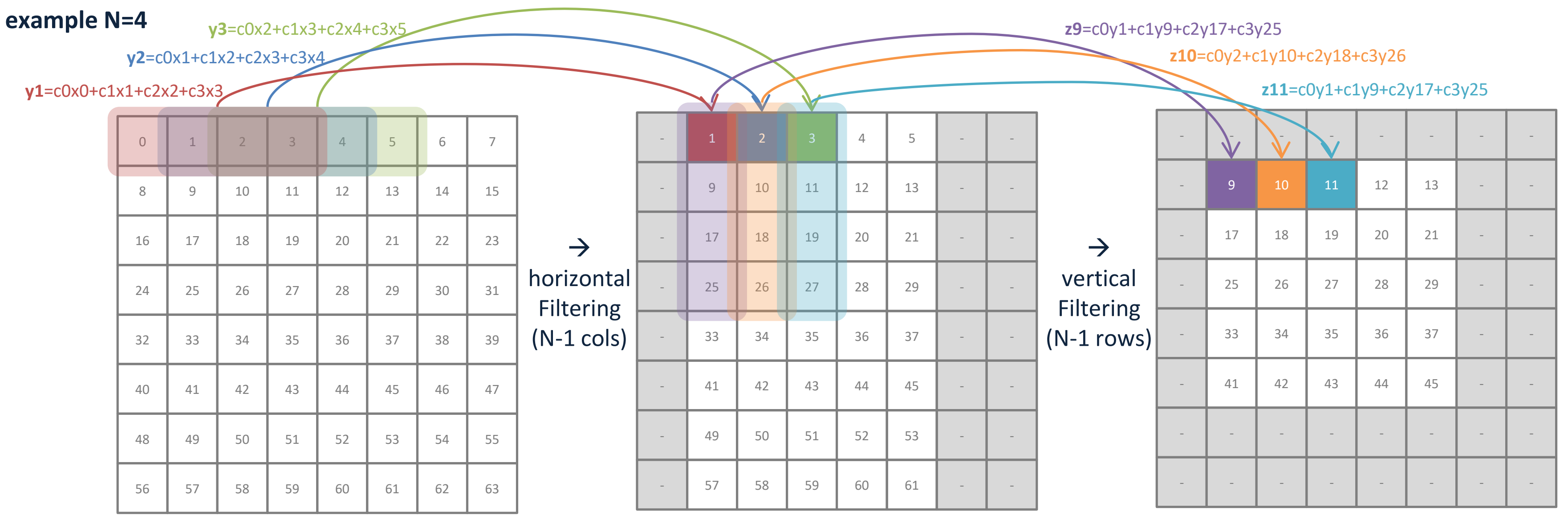



example N=4
y1=c0x0+c1x1+c2x2+c3x3
y2=c0x1+c1x2+c2x3+c3x4
y3=c0x2+c1x3+c2x4+c3x5
z9=c0y1+c1y9+c2y17+c3y25
z10=c0y2+c1y10+c2y18+c3y26
z11=c0y1+c1y9+c2y17+c3y25
→ horizontal Filtering (N-1 cols)
→ vertical Filtering (N-1 rows)

Results: Functional vs. Imperative HLS for CGRA

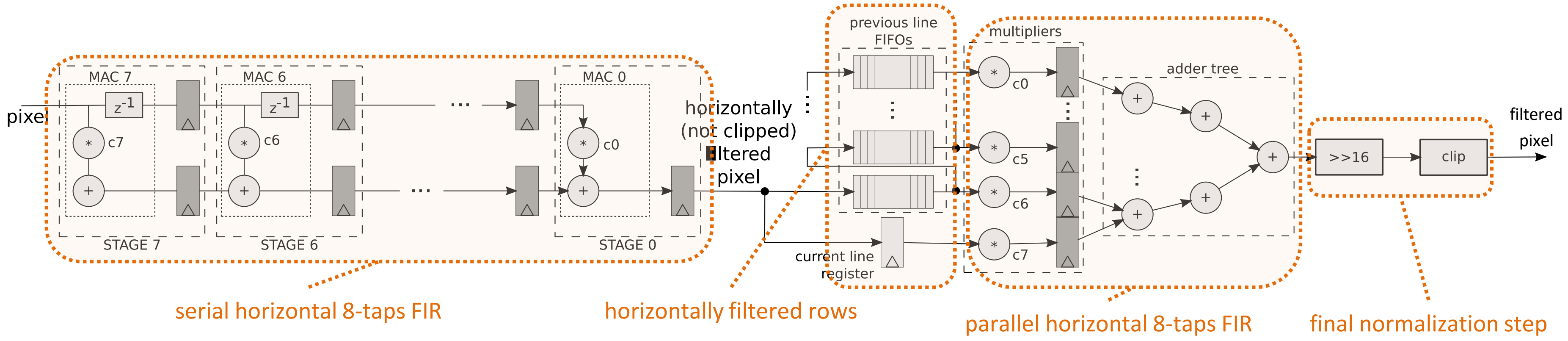
Test Case: HEVC Fractional Pixel Interpolators

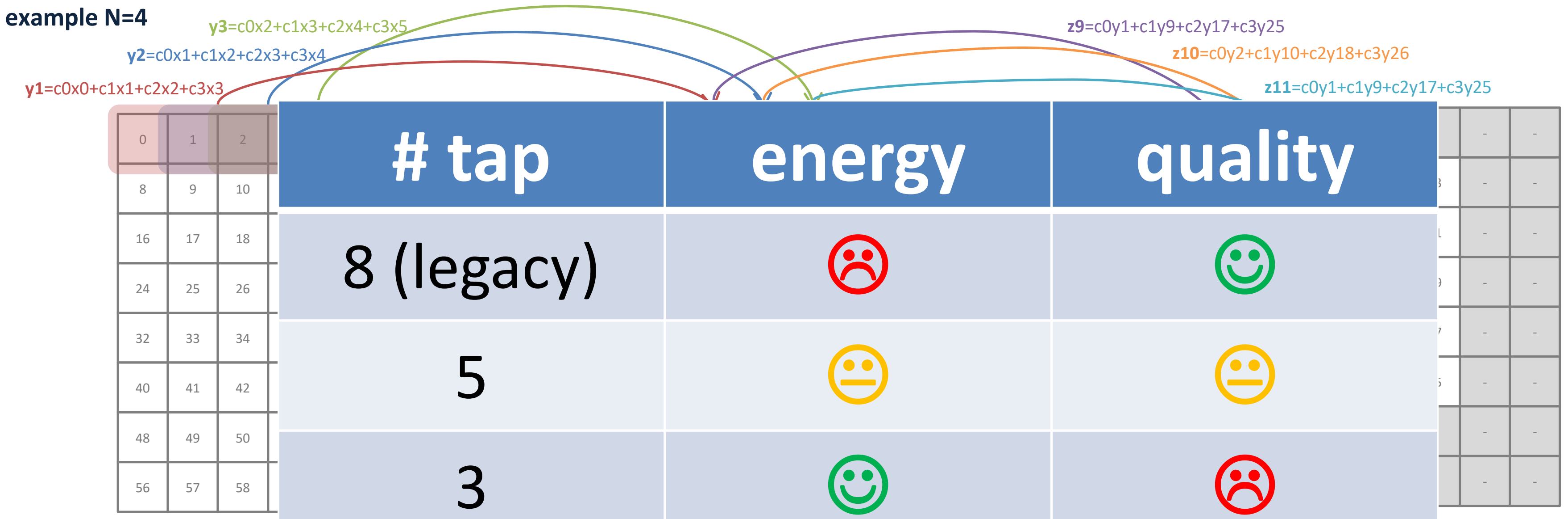



example N=4
y3=c0x2+c1x3+c2x4+c3x5
y2=c0x1+c1x2+c2x3+c3x4
y1=c0x0+c1x1+c2x2+c3x3
z9=c0y1+c1y9+c2y17+c3y25
z10=c0y2+c1y10+c2y18+c3y26
z11=c0y1+c1y9+c2y17+c3y25
→ horizontal Filtering (N-1 cols)
→ vertical Filtering (N-1 rows)

Results: Functional vs. Imperative HLS for CGRA

Test Case: HEVC Fractional Pixel Interpolators



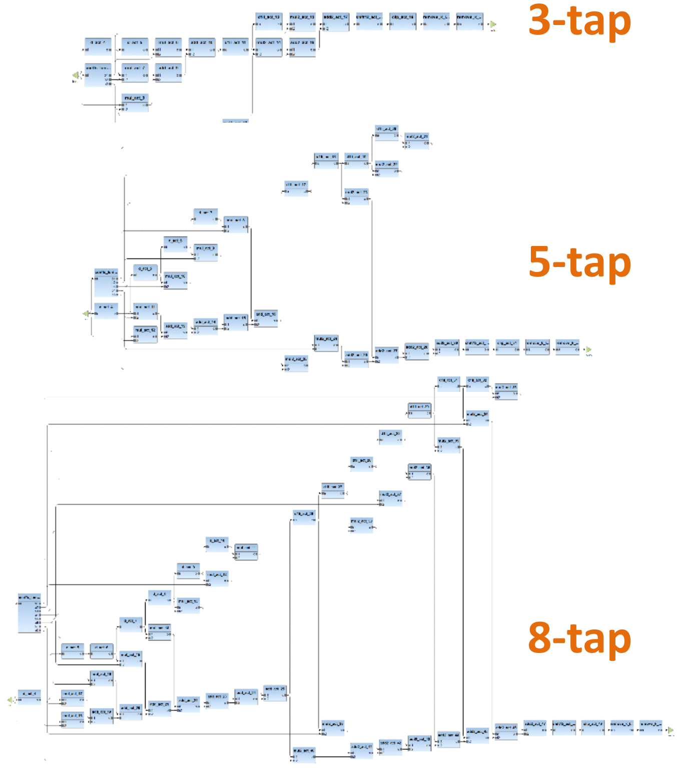


# tap	energy	quality
8 (legacy)	☹	☺
5	😐	😐
3	☺	☹

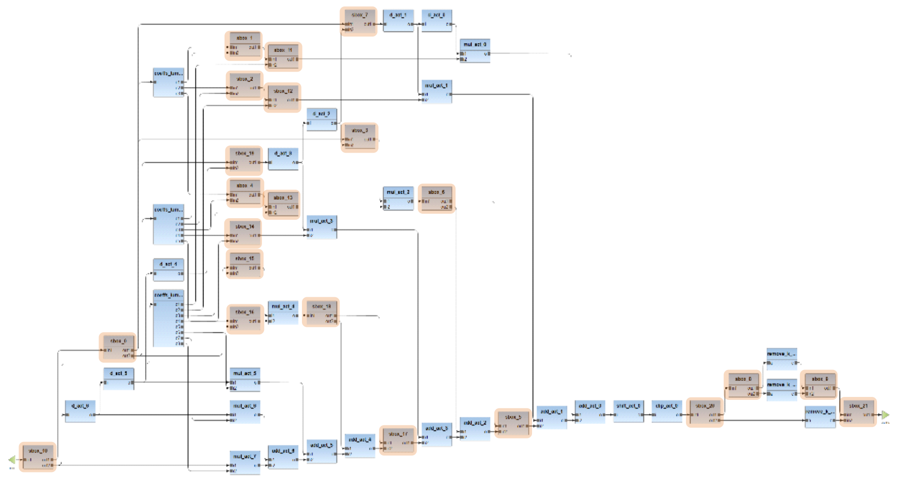
Results: Functional vs. Imperative HLS for CGRA

Test Case: HEVC Fractional Pixel Interpolators

INPUT FUNCTIONAL SPECIFICATIONS (CAPH dataflows)



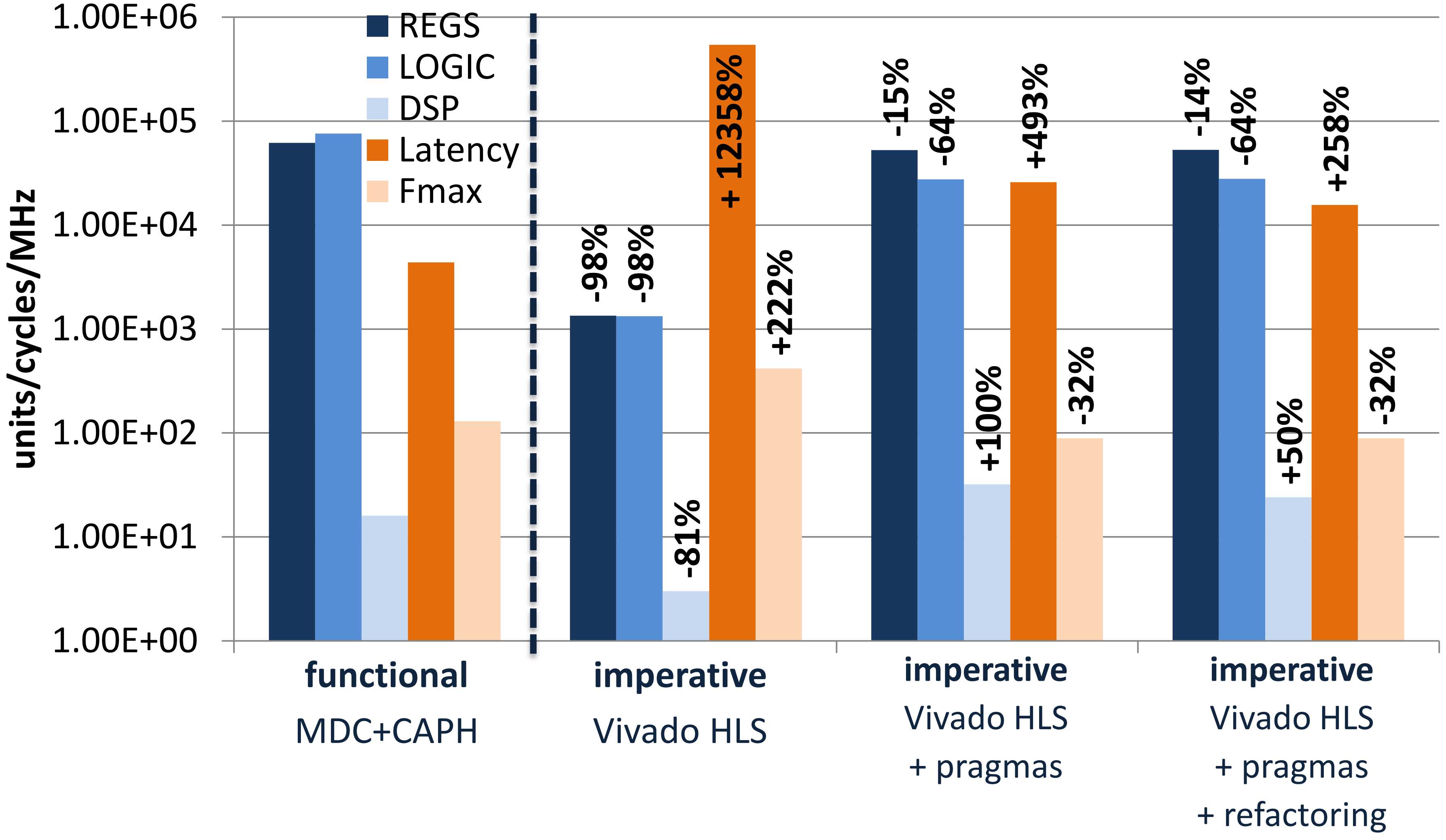
MULTI-FUNCTIONAL DATAFLOW SPECIFICATION (composed by MDC)



🟧 switching module

Results: Functional vs. Imperative HLS for CGRA

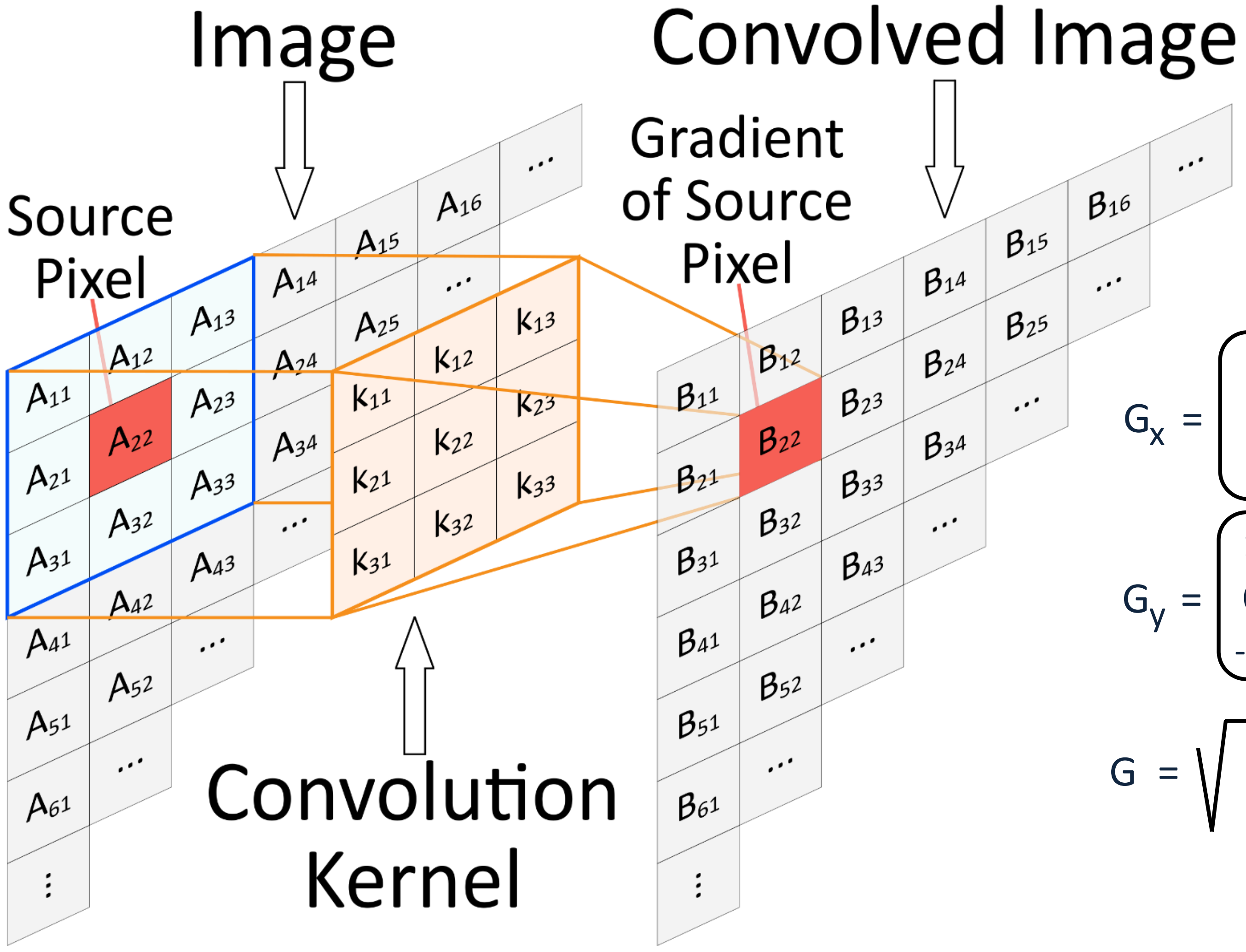
Test Case: HEVC Fractional Pixel Interpolators



target: Xilinx XC7VX485T *Virtex-7FPGA*

Results: Target Specific vs. Independent HLS for CGRA

Test Case: Sobel and Prewitt edge detectors



$$G_x = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} * A$$

$$G_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} * A$$

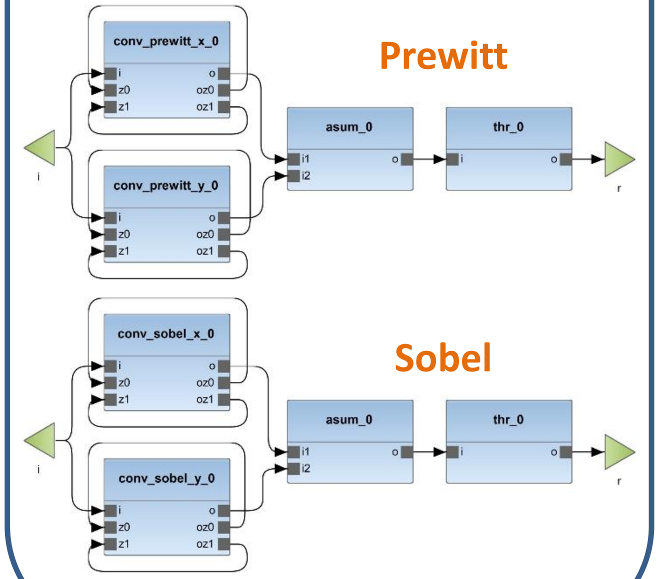
$$G = \sqrt{G_x^2 + G_y^2}$$

$$G \geq 80$$

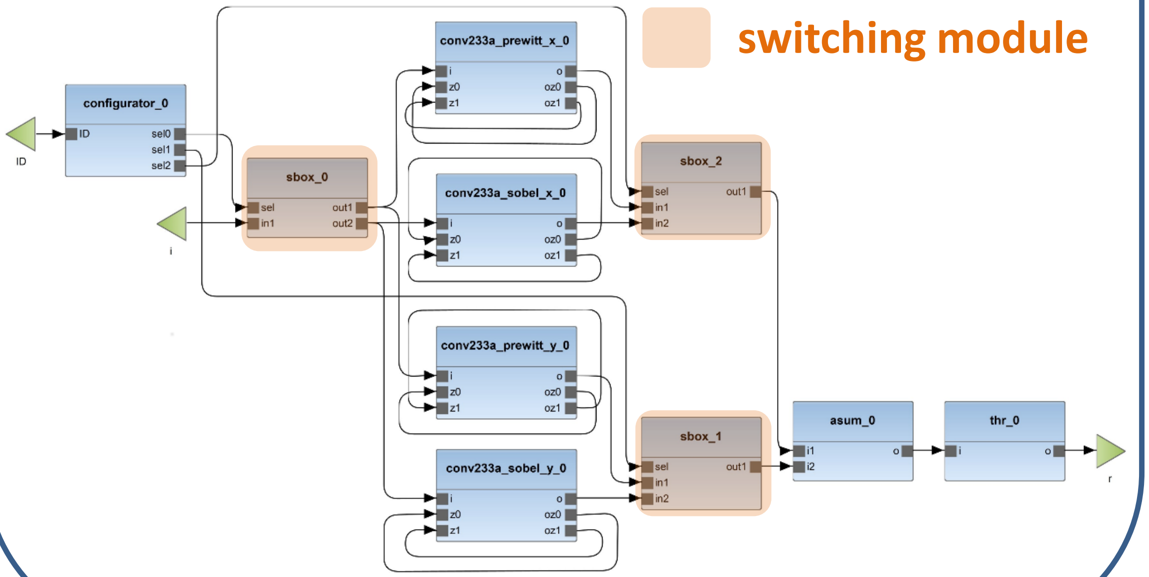
Results: Target Specific vs. Independent HLS for CGRA

Test Case: Sobel and Prewitt edge detectors

INPUT FUNCTIONAL SPECIFICATIONS
(CAPH dataflows)



MULTI-FUNCTIONAL DATAFLOW SPECIFICATION
(composed by MDC)



Results: Target Specific vs. Independent HLS for CGRA

Test Case: Sobel and Prewitt edge detectors

FPGA*	MDC+CAPH		MDC+XRONOS		XRONOS vs CAPH	
	Altera	Xilinx	Altera	Xilinx	Altera	Xilinx
REG	1484	780	-	632	-	-18,97%
LOGIC	1047	2347	-	1533	-	-34,68%
RAM	15	0	-	6.5	-	+100%
DSP	36	36	-	0	-	-100%
MAX FREQ [MHz]	105,80	93,69	-	142,86	-	+58,50%
EXEC TIME [cck]	15340	15340	-	15348	-	+0,05%

ASIC**	MDC+CAPH	MDC+XRONOS
AREA [kGE]	466,90	-
Max Freq [MHz]	399.04	-

**target: Xilinx XXX Zynq-7 FPGA*

***target: TMSC 45 nm CMOS technology*

Results: Target Specific vs. Independent HLS for CGRA

Test Case: Sobel and Prewitt edge detectors

CAPH VHDL

```
entity thr is
  port (
   i_empty: in std_logic;
   i: in std_logic_vector(13 downto 0);
   i_rd: out std_logic;
   o_full: in std_logic;
   o: out std_logic_vector(9 downto 0);
   o_wr: out std_logic;
   clock: in std_logic;
   reset: in std_logic
   );
end thr;
architecture FSM of thr is
begin
 comb: process(i, i_empty, o_full)
  variable p_p : signed(13 downto 0);
 begin
  -- i.rdy, i=4096, o.rdy / rd(i), wr(o,256)
  if i_empty='0' and from_std_logic_vector(i,14)=to_signed(4096,14) and o_full='0'
   then
   i_rd <= '1';
   o <= std_logic_vector(to_unsigned(256,10));
   o_wr <= '1';
  -- i.rdy, i=8192, o.rdy / rd(i), wr(o,512)
  elsif i_empty='0' and from_std_logic_vector(i,14)=to_signed(8192,14) and o_full='0'
   then
   ...
  end if;
 end process;
 seq: process(clock, reset)
 begin
  if (reset='0') then
  elsif rising_edge(clock) then
  end if;
 end process;
end FSM;
```

XRONOS Verilog

```
module thr_0(o_SEND, i_DATA, i_ACK,
o_RDY, i_COUNT, o_ACK, CLK, RESET,
i_SEND, o_COUNT, o_DATA);
wire                thrSof_go;
output              o_SEND;
input    [13:0]     i_DATA;
output              i_ACK;
wire                thr_done;
wire                thrSof_done;
input               o_RDY;
input    [15:0]     i_COUNT;
...
assign thrSof_go=scheduler_u214;
assign o_SEND=or_024e0490_u0;
assign i_ACK=or_173c13d7_u0;
assign thr_done=bus_50b9a749_;
...
thr_0_scheduler
 thr_0_scheduler_instance(.CLK(CLK),
 .RESET(bus_3fb49b2f_), .GO(bus_72d0efa6_),
  .port_431807df_(o_RDY),
  .port_74145b7a_(i_DATA),
  .port_64e9829f_(thrEoF_done),
  .port_3adf3763_(thr_done),
  .port_1779a109_(i_SEND),
  .port_0bb7799c_(thrSof_done),
  .DONE(thr_0_scheduler_instance_DONE),
  .RESULT(scheduler),
  .RESULT_u981(scheduler_u214),
  .RESULT_u982(scheduler_u215));
thr_0_thrSof
 thr_0_thrSof_instance(.CLK(CLK),
 .GO(thrSof_go), .port_269cbf08_(i_DATA),
 .DONE(thr_0_thrSof_instance_DONE),
 .RESULT(thrSof), .RESULT_u983(thrSof_u3),
 .RESULT_u984(thrSof_u4),
 .RESULT_u985(thrSof_u5));
endmodule
```

Outline

Final Remarks

- Modern **embedded systems**, pushed into the IoT and CPS era, have **lots of colliding requirements**.
- **Coarse-Grained Reconfigurable Architectures** (CGRAs) could be a **suitable solution** to meet these requirements.
- The development of CGRAs require **HLS support** to complete **design automation**
  - **Functional HLS outperforms imperative HLS** for this specific kind of architectures
  - **Target specific HLS** is more efficient on its target, but it **lacks in code compactness and readability**

Thanks To …

EU Commission for funding the ***CERBERO*** (*Cross-layer modEl-based fRamework for multi-oBjective dEsign of Reconfigurable systems in unceRtain hybRid envirOnments*) project as part of the H2020 Programme under grant agreement No 732105.

Coordinator:

Michal Masin (IBM), michaelm@il.ibm.com

Scientific Coordinator:

Francesca Palumbo (UniSS), fpalumbo@uniss.it

Innovation Manager:

Katiuscia Zedda (Abinsula),
katiuscia.zedda@abinsula.com

Dissemination-Communication Manager:

Francesco Regazzoni (USI),
francesco.regazzoni@usi.ch

www.cerbero-h2020.eu

info@cerbero-h2020.eu

@CERBERO_h2020

CERBERO

Platform-Agnostic Dataflow-to-Hardware Design Flow for Reconfigurable Systems

Francesca Palumbo, Claudio Rubattu, **Carlo Sau**, Luigi Raffo and Maxime Pelcat

Alghero (Italy) – September 25-30, 2017

Designing Cyber-Physical Systems – From concepts to implementation

Distinguished lecturers: Alberto Sangiovanni-Vincentelli, XXX

Application deadline: **July XXth, 2018** http://www.cpsschool.eu/

Naples, 20-22 June 2018