PROSSIMO
Progettazione, sviluppo e ottimizzazione di sistemi intelligenti multi-oggetto

Architetture eterogenee on-chip riconfigurabili: analisi, sviluppo e caratterizzazione del loro comportamento con sistemi di monitoring

Tiziana Fanni[1], Carlo Sau[2], Giacomo Valente[3]

[1]University of Sassari, Intelligent system DEsign and Application (IDEA) Group

[2]University of Cagliari, Diee – Microelectronics and Bioengineering (EOLAB) Group

[3]University of L'Aquila, Disim – Embedded Systems Group

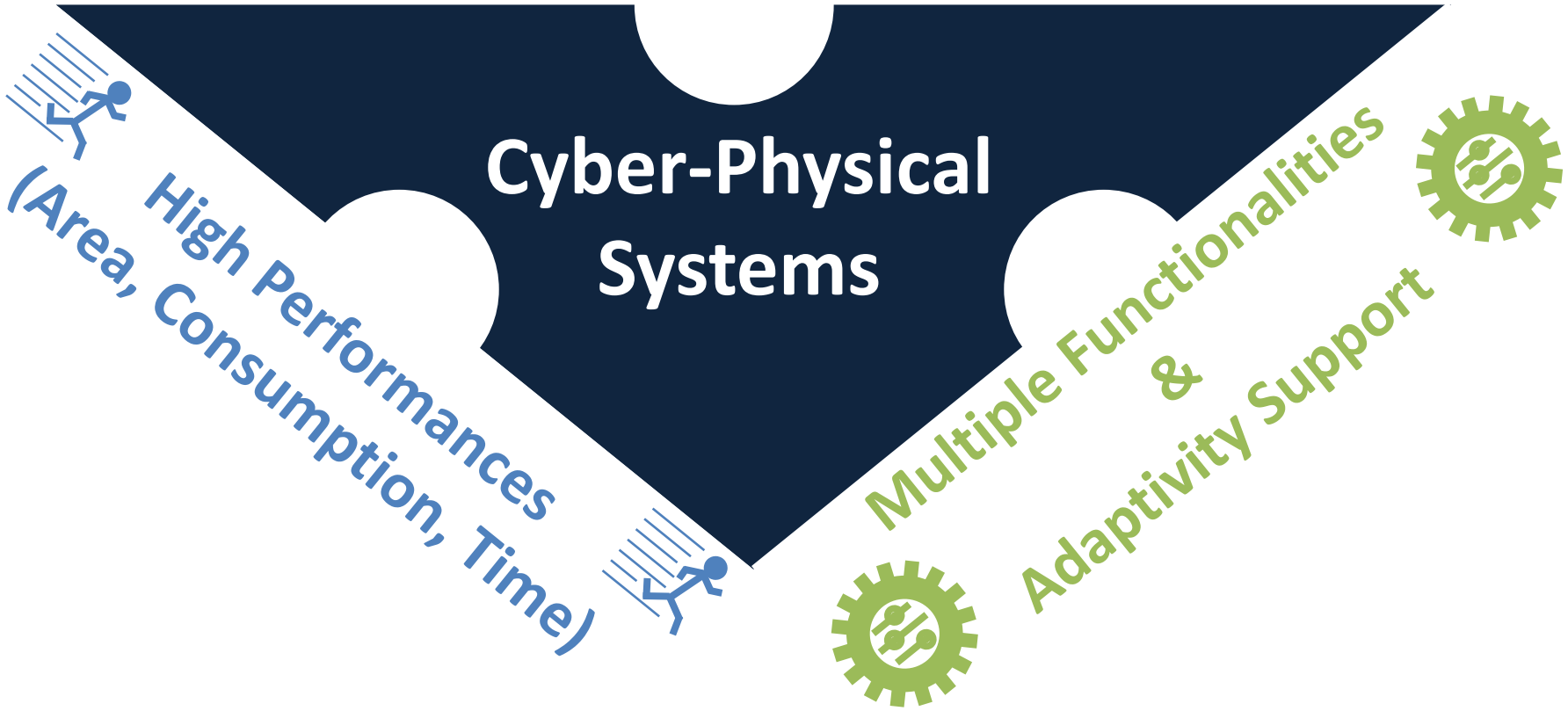
tfanni@uniss.it, carlo.sau@unica.it, giacomo.valente@univaq.it

PROSSIMO
Progettazione, sviluppo e ottimizzazione di sistemi intelligenti multi-oggetto

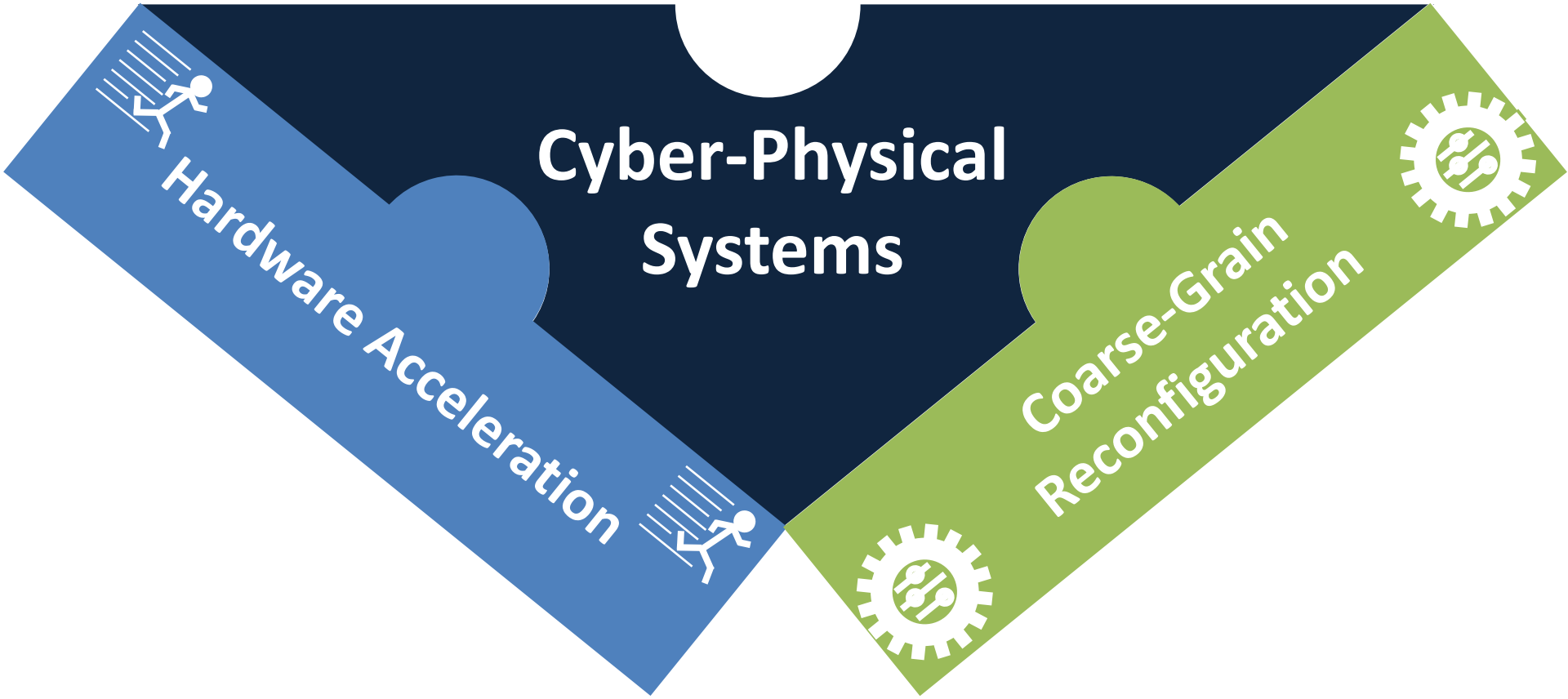
Architetture eterogenee on-chip riconfigurabili: analisi, sviluppo e caratterizzazione del loro comportamento con sistemi di monitoring

Introduction and Motivation

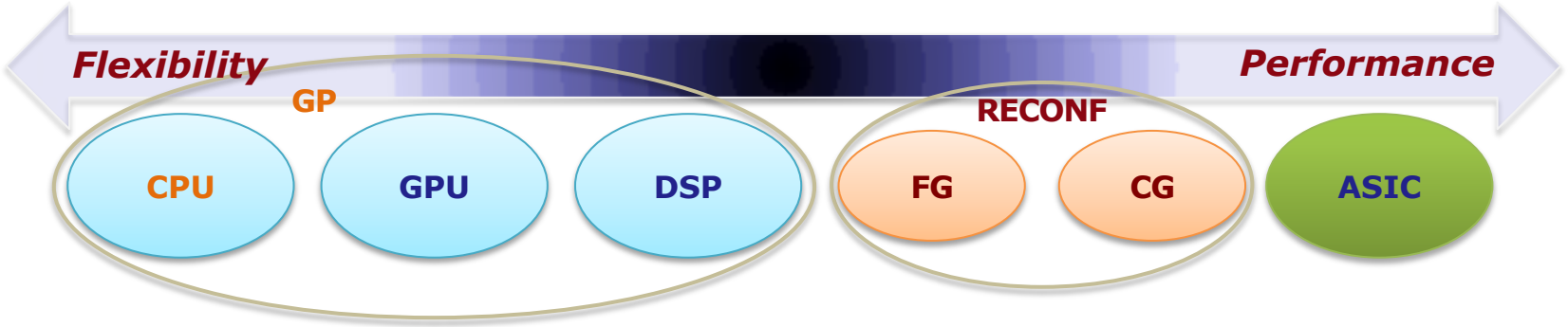
Cyber-Physical Systems Issues



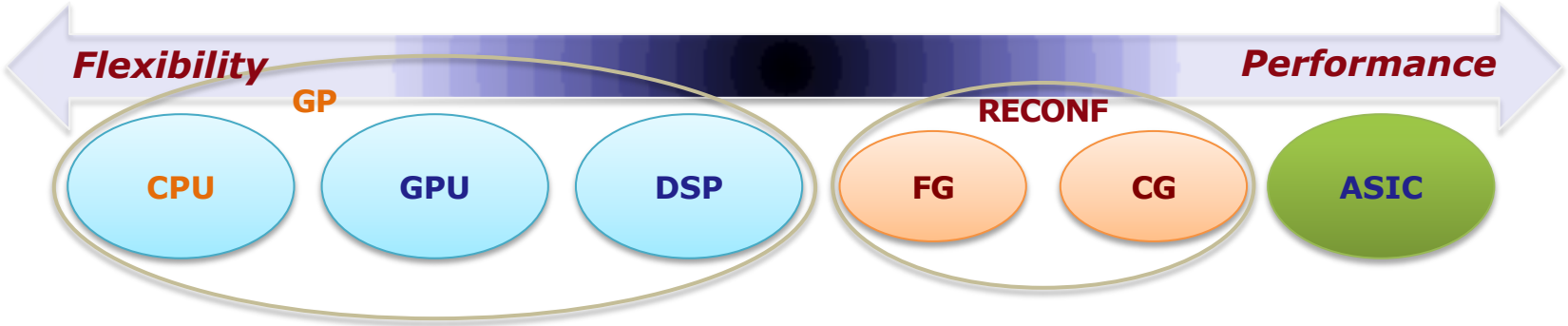
Cyber-Physical Systems Issues



Reconfigurable Hardware



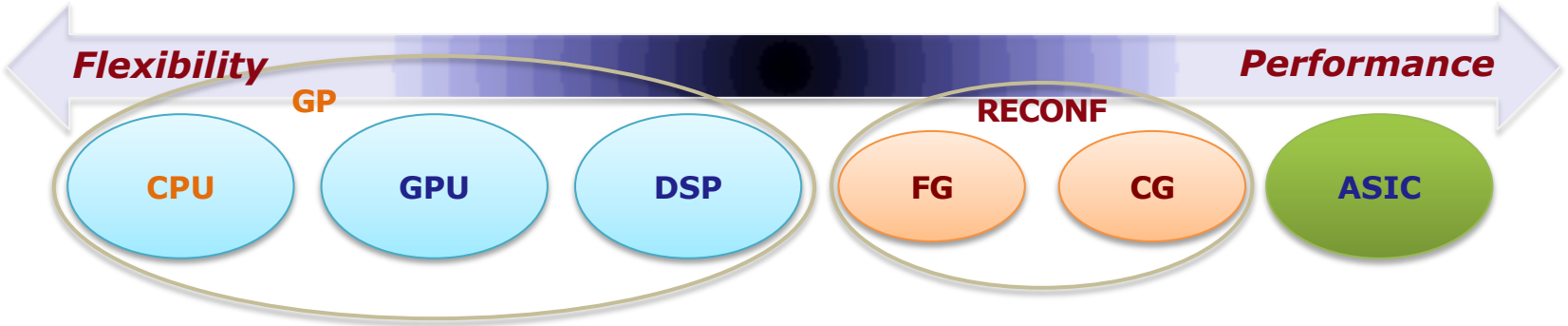
Reconfigurable Hardware



	Fine Grained	Coarse Grained
	bit-level	word-level
Flexibility	☺	😐
Speed	😐	☺
Memory	☹	😐

- **Coarse Grained (CG):**

Reconfigurable Hardware

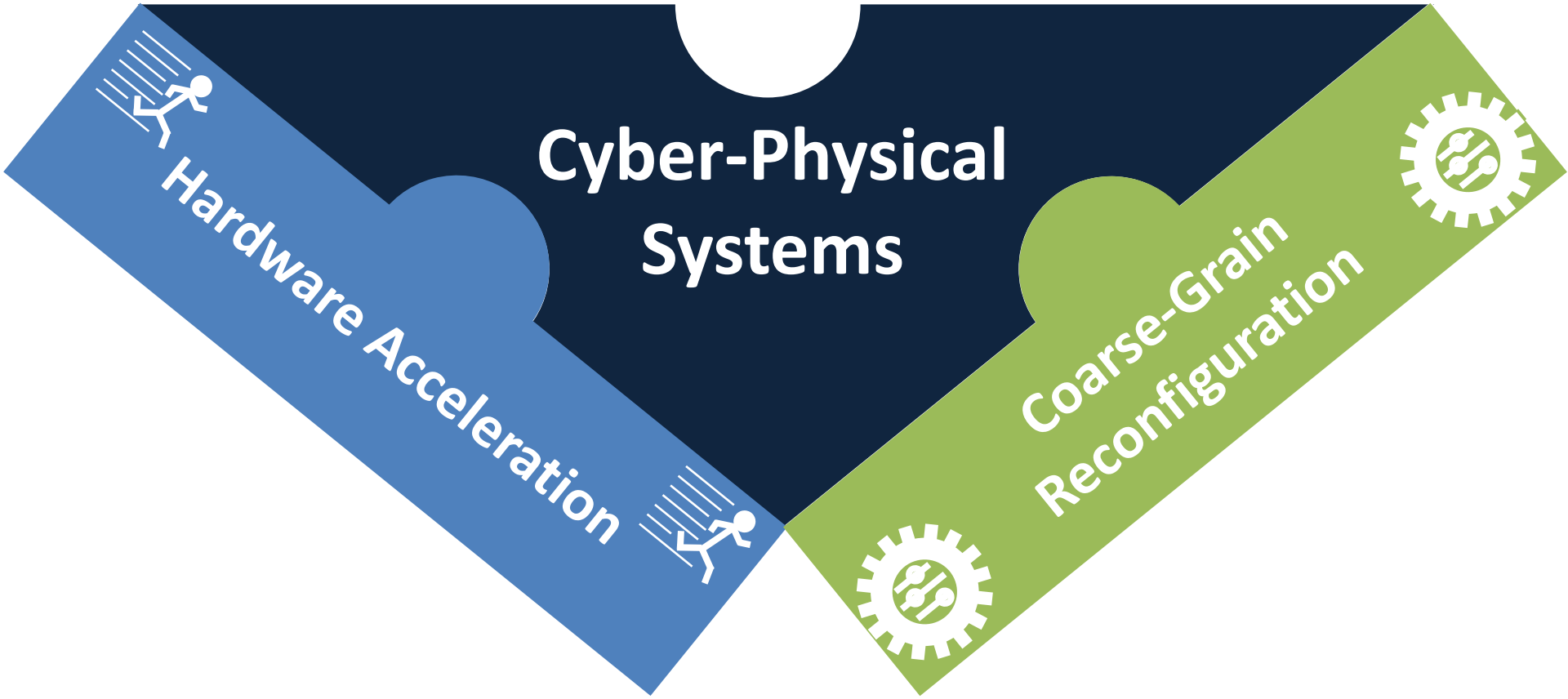


	Fine Grained	Coarse Grained
	bit-level	word-level
Flexibility	☺	😐
Speed	😐	☺
Memory	☹	😐

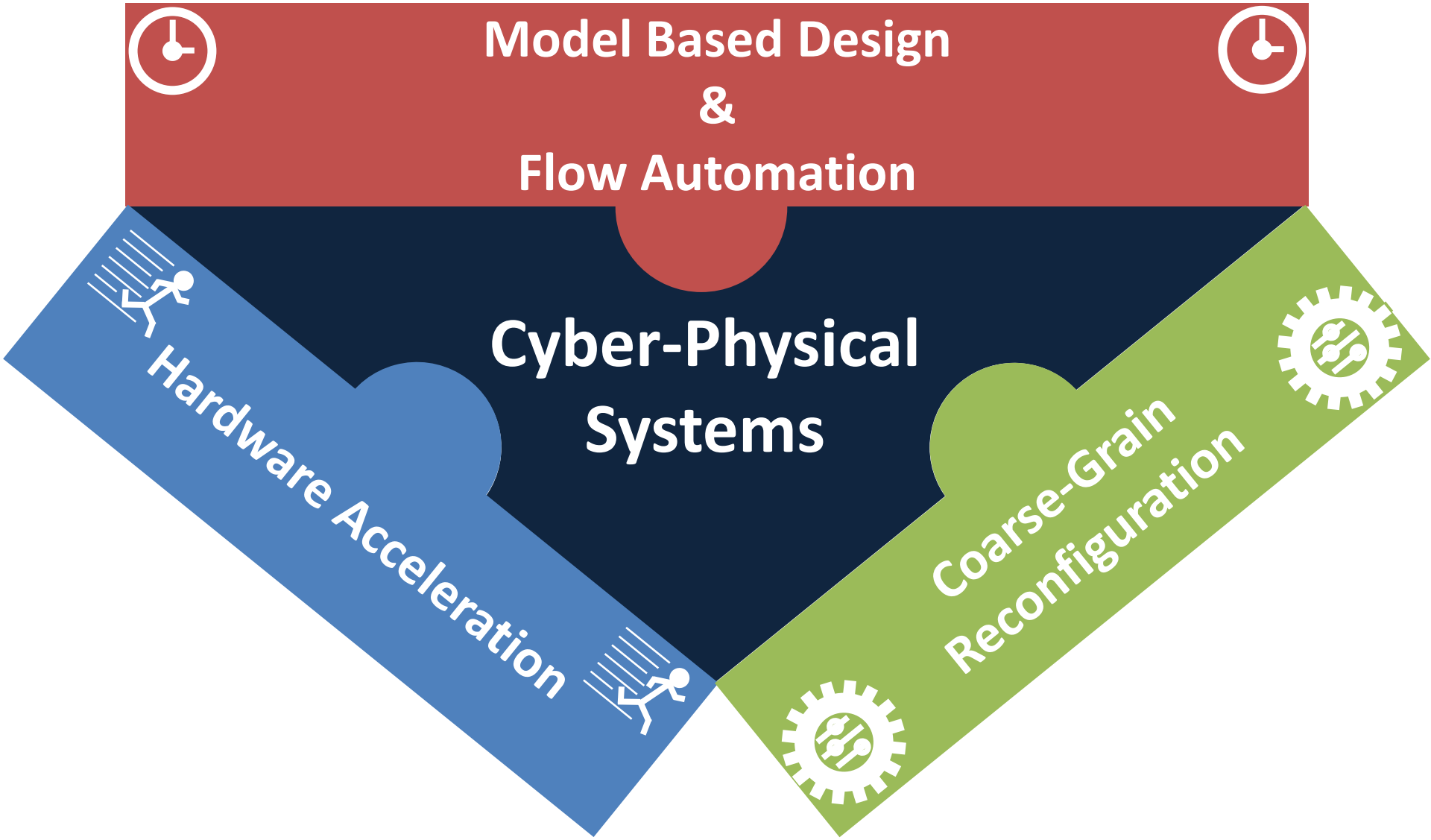
- **Coarse Grained (CG):**
  - both in ASIC and FPGA
  - 1 clock cycle switching, with dedicated switching blocks.
- **Fine Grained (FG):**
  - FPGA only
  - switching requires a new bit-stream

Cyber-Physical Systems Issues

Short Time to Market
(Development, Optimization, Integration)

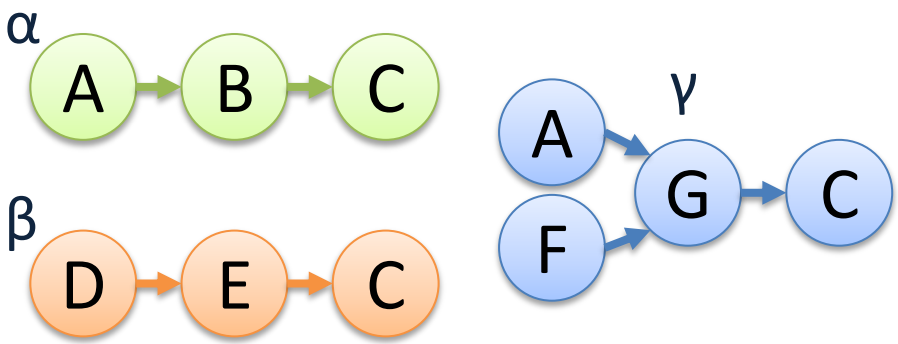


Cyber-Physical Systems Issues



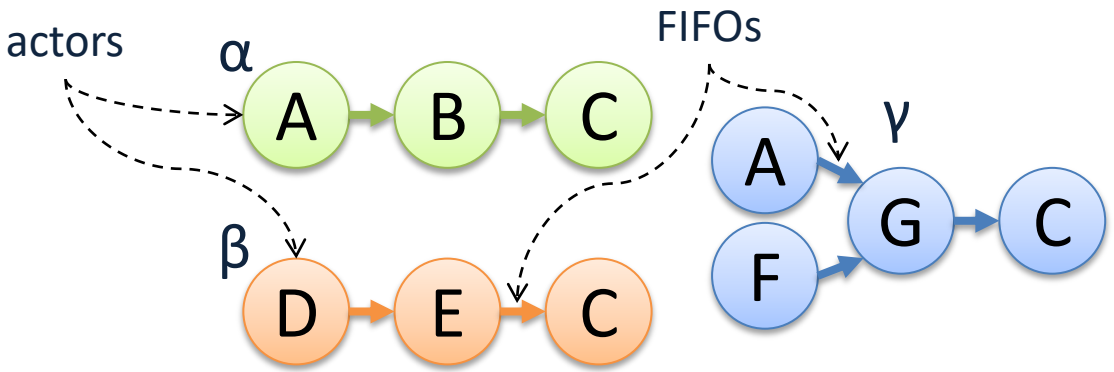
Model-Based Design Automation

Dataflow Models of Computation



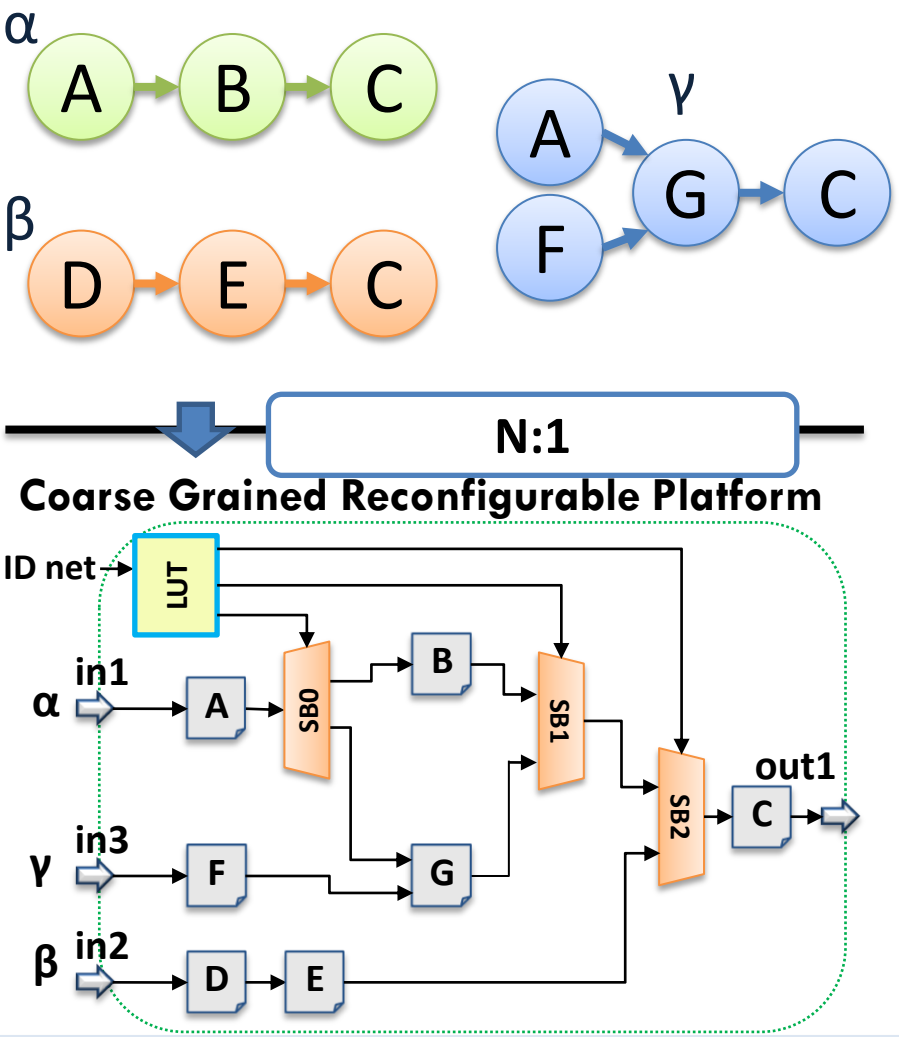
Model-Based Design Automation

Dataflow Models of Computation



Model-Based Design Automation

Dataflow Models of Computation



Model-Based Design Automation

Dataflow Models of Computation

Multi Dataflow Composer Tool

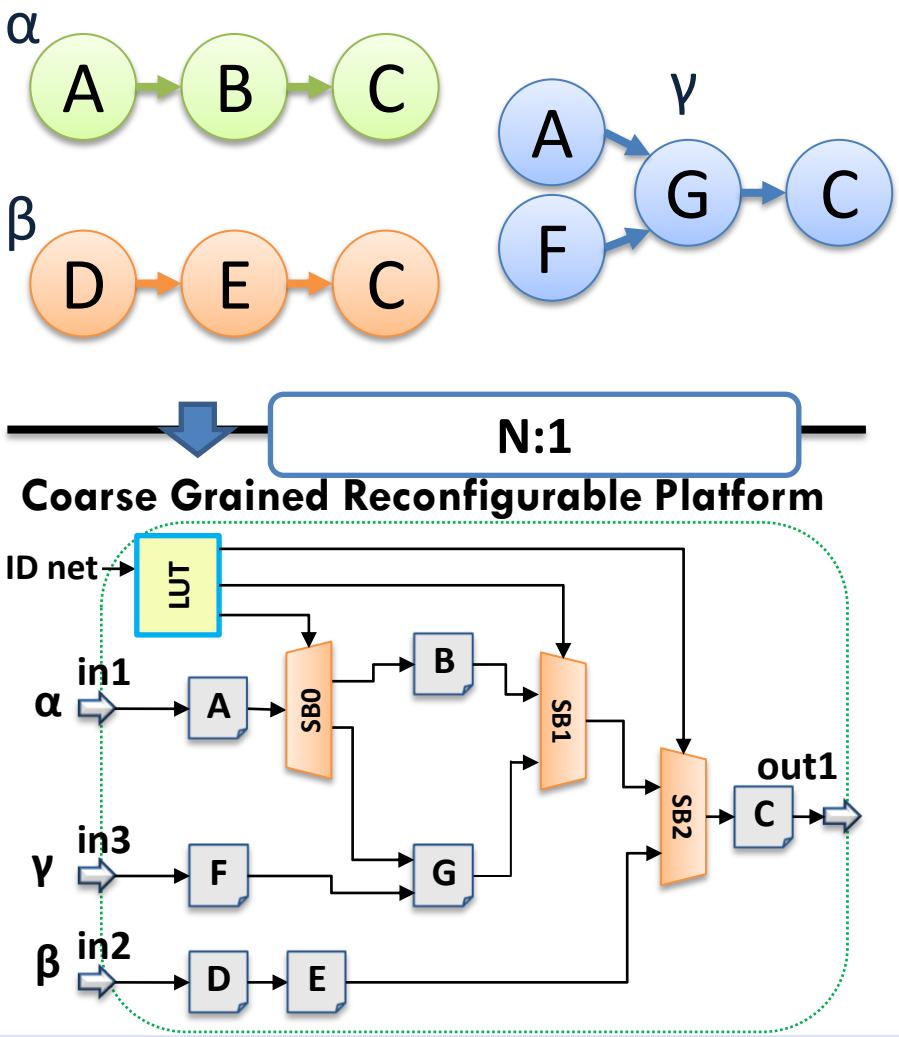
Structural Profiler

Power Manager

Co-Processor Generator

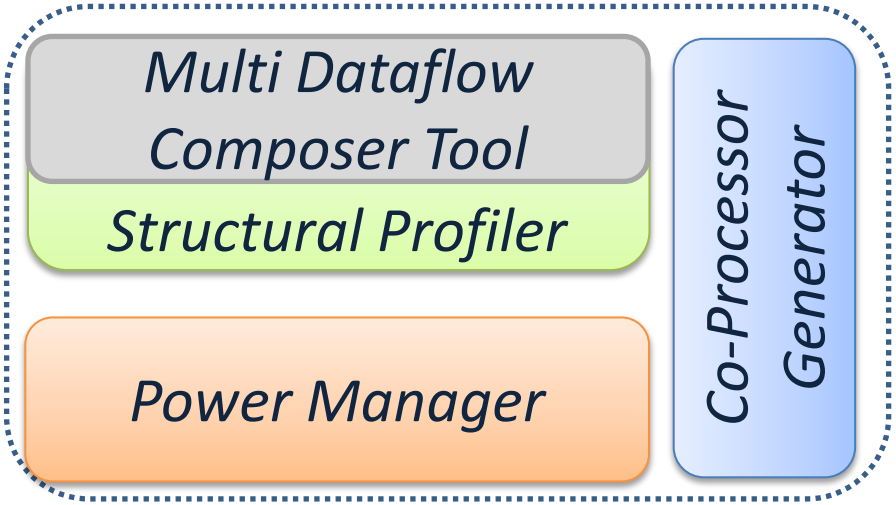
MDC design suite

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



Multi-Dataflow Composer

Additional Features



MDC design suite

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

Structural Profiler:

low-level feedback (from synthesis) and DSE for topology optimization.

- (ASIC + FPGA)

Co-Processor Generator:

generation of ready-to-use Xilinx IPs

- (FPGA)

Power Manager:

automatic application of clock-gating and/or power-gating.

- CG (ASIC + FPGA)
- PG(ASIC)

PROSSIMO
Progettazione, sviluppo e ottimizzazione di sistemi intelligenti multi-oggetto

Architetture eterogenee on-chip riconfigurabili: analisi, sviluppo e caratterizzazione del loro comportamento con sistemi di monitoring

MDC Contexts of Application

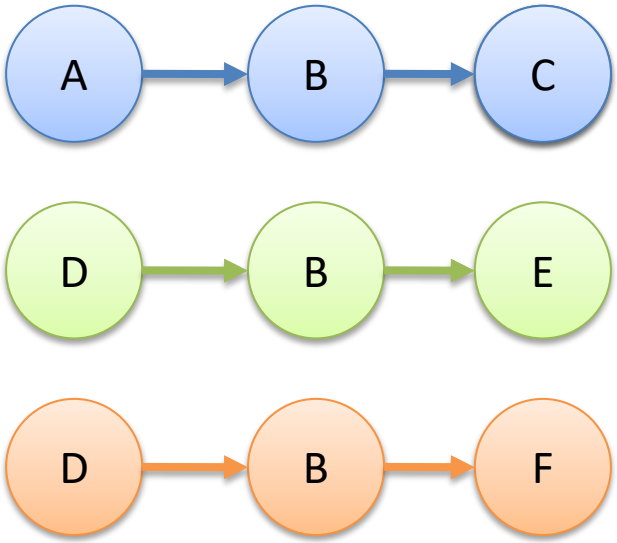
MDC Contexts of application

What kinds of applications can be combined with MDC?

MDC Contexts of application

What kinds of applications can be combined with MDC?

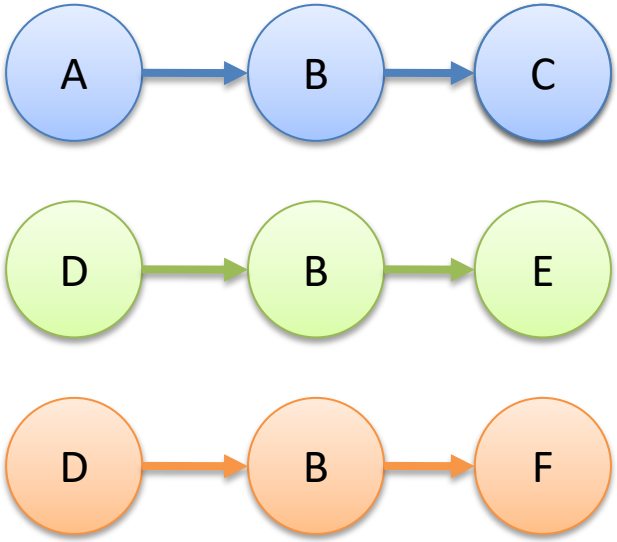
1. **Different applications with common computational operations**: it is achieved by considering applications from the **same application field** or **small actor granularities**.



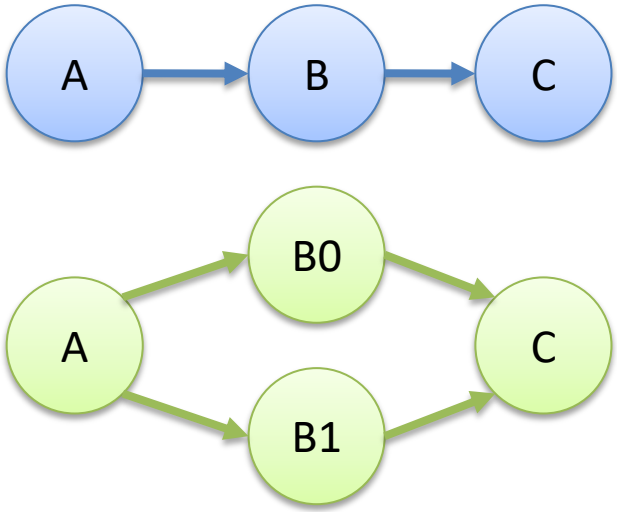
MDC Contexts of application

What kinds of applications can be combined with MDC?

1. **Different applications with common computational operations**: it is achieved by considering applications from the **same application field** or **small actor granularities**.



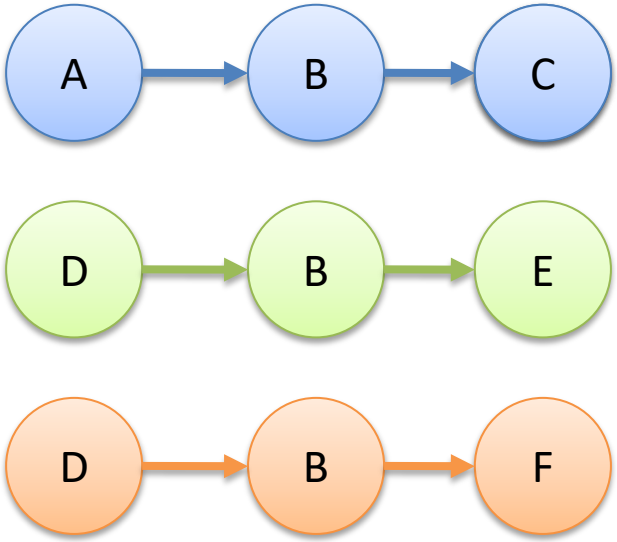
2. **Different working points of the same applications** obtained through several strategies (e.g. **actor parallelization**, actor variants, granularity modification, **approximate computing**, ...)



MDC Contexts of application

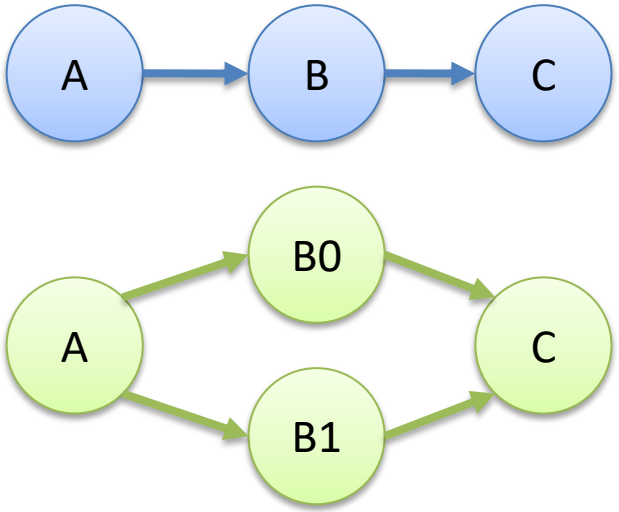
What kinds of applications can be combined with MDC?

1. **Different applications with common computational operations**: it is achieved by considering applications from the **same application field** or **small actor granularities**.



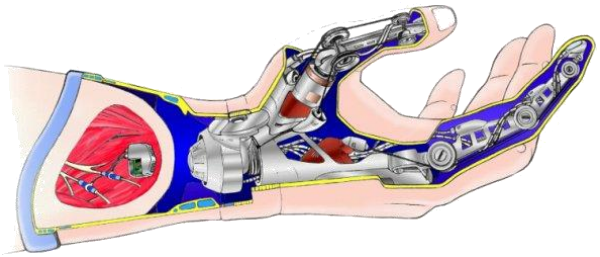
EXAMPLE: Neural Signal Decoding

2. **Different working points of the same applications** obtained through several strategies (e.g. **actor parallelization**, actor variants, granularity modification, **approximate computing**, ...)



EXAMPLE: HEVC interpolation filters

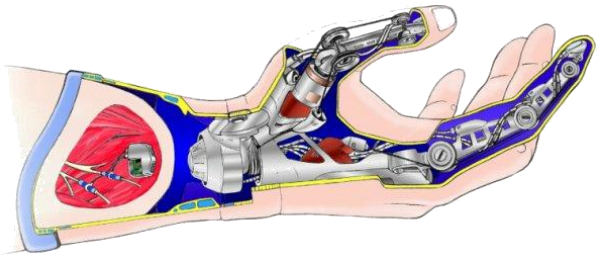
Neural Signal Decoding



Resource Optimization

Implantable Devices: strict **area** & **power** requirements

Neural Signal Decoding

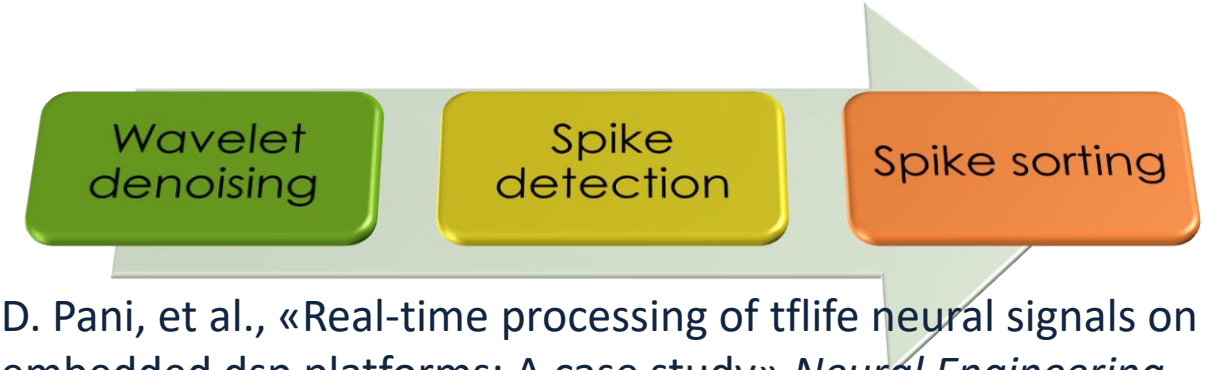


Resource Optimization

Implantable Devices: strict **area** & **power** requirements

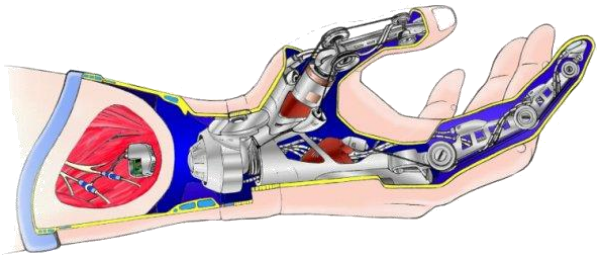
Neural Signal Decoding:

- Fast
- Low Area
- Low Power



D. Pani, et al., «Real-time processing of tflife neural signals on embedded dsp platforms: A case study» *Neural Engineering*, 2011.

Neural Signal Decoding

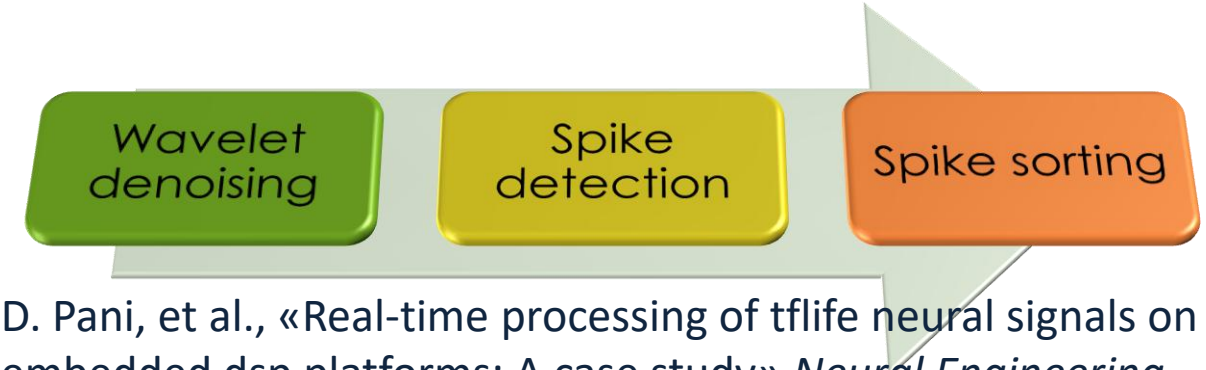


Resource Optimization

Implantable Devices: strict **area** & **power** requirements

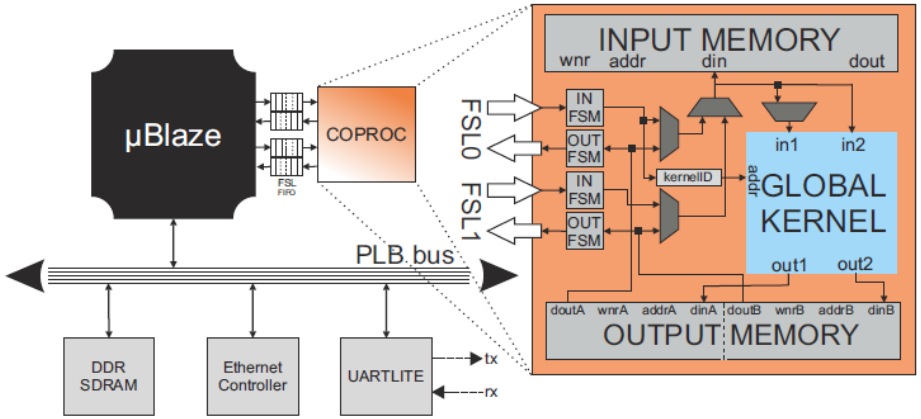
Neural Signal Decoding:

- Fast
- Low Area
- Low Power



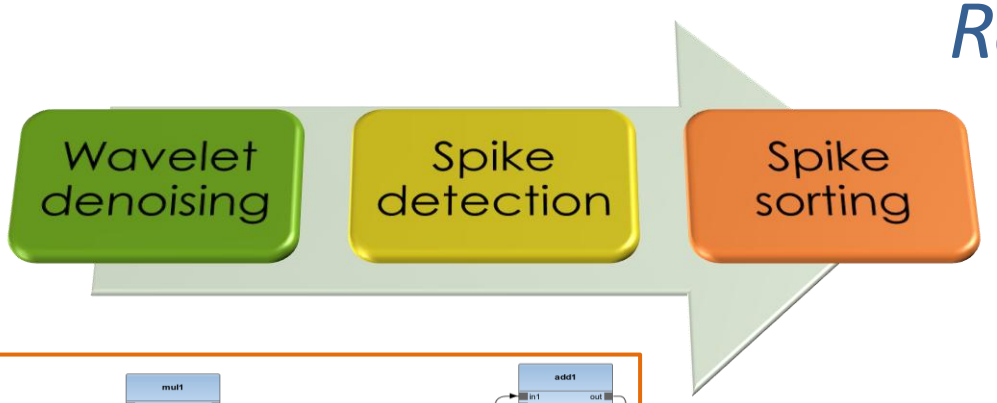
D. Pani, et al., «Real-time processing of tflife neural signals on embedded dsp platforms: A case study» *Neural Engineering*, 2011.

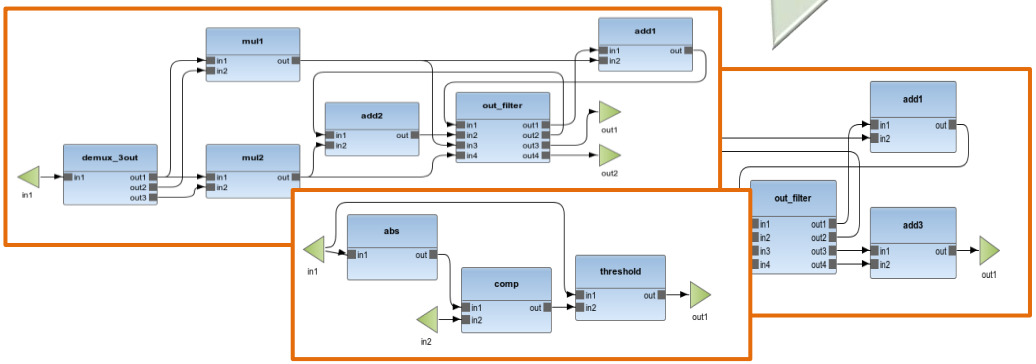
MDC can be used to build the accelerators compliant to those constraints.

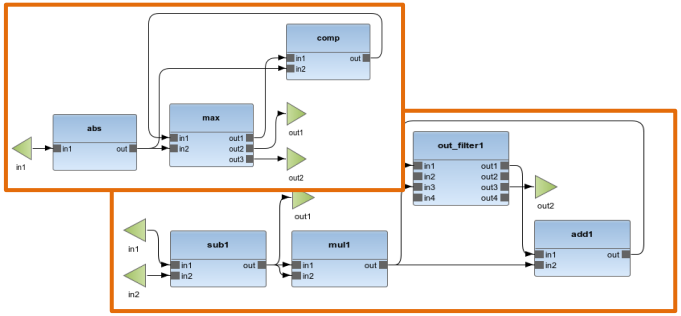


Neural Signal Decoding

Resource Optimization



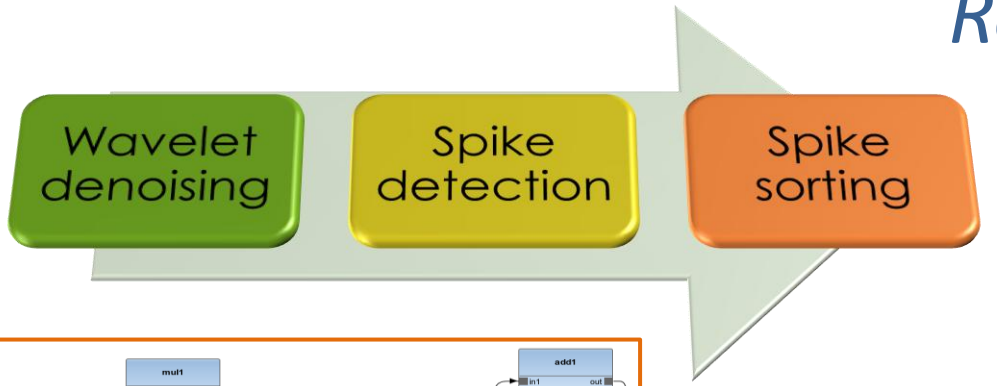


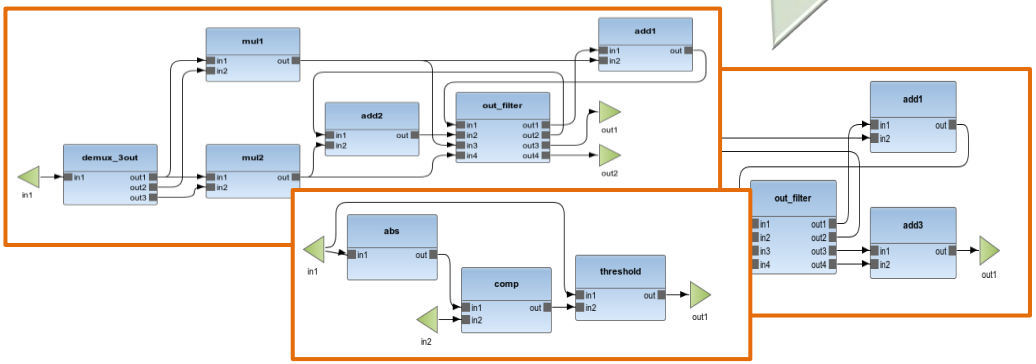


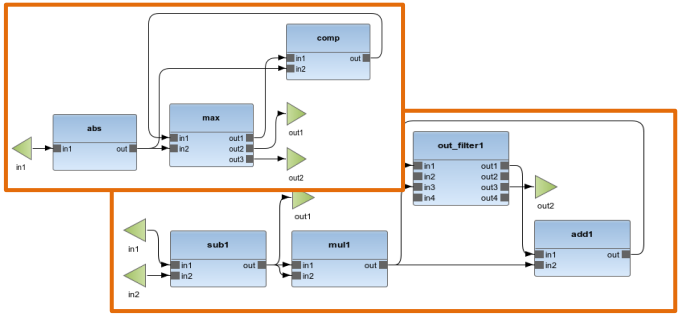
	# actors	#sbox
12 networks (dec_filter, Thr, rec_filter, NEO, idx_max_abs, Avg, sqr_sum, weight_mul, dot_prod, idx_max, sync_avg, sync_wavg)	46	0
MDC network	14	86

Neural Signal Decoding

Resource Optimization



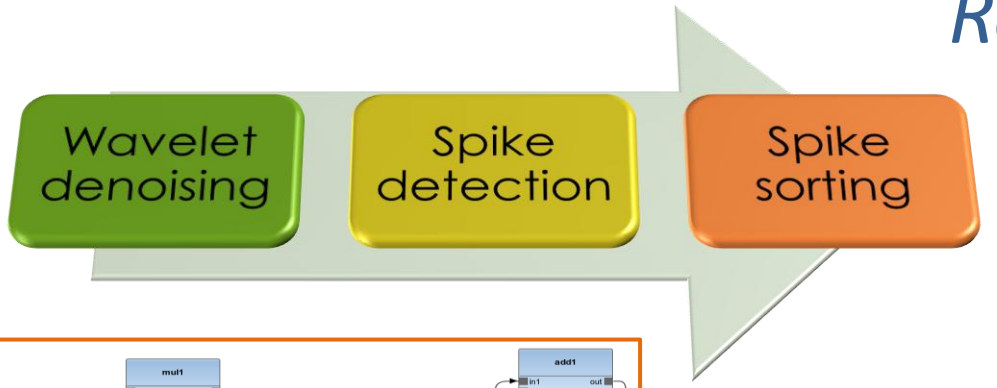


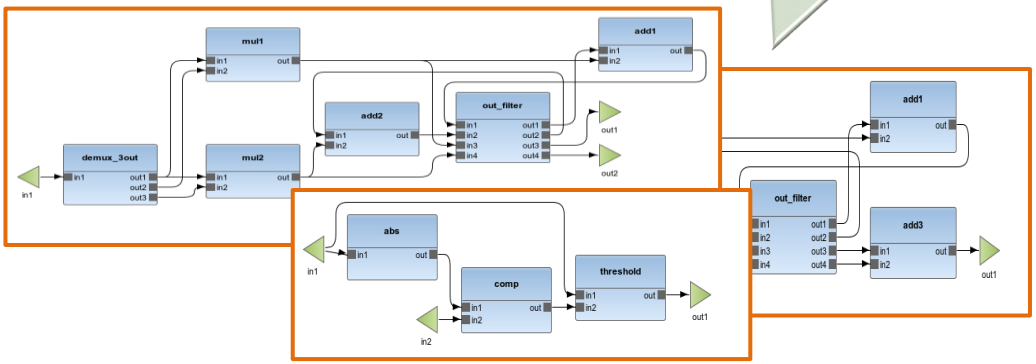


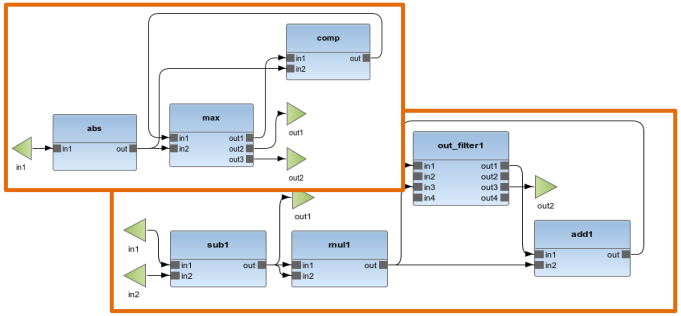
	# actors	#sbox
12 networks (dec_filter, Thr, rec_filter, NEO, idx_max_abs, Avg, sqr_sum, weight_mul, dot_prod, idx_max, sync_avg, sync_wavg)	46	0
MDC network	14	86

Neural Signal Decoding

Resource Optimization



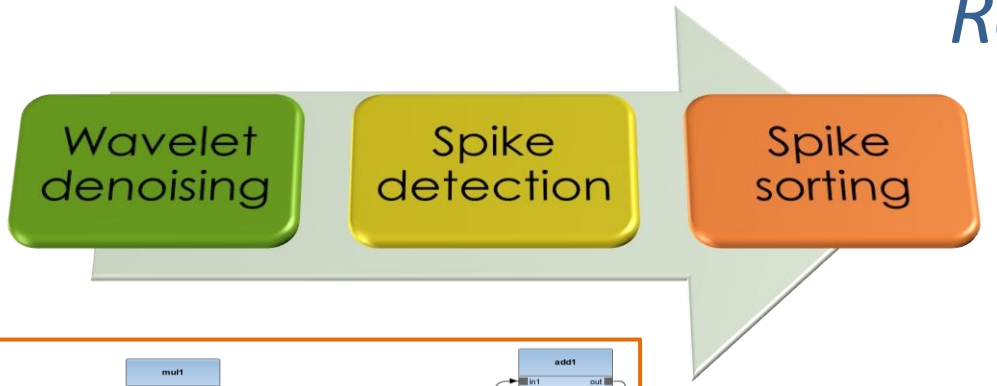


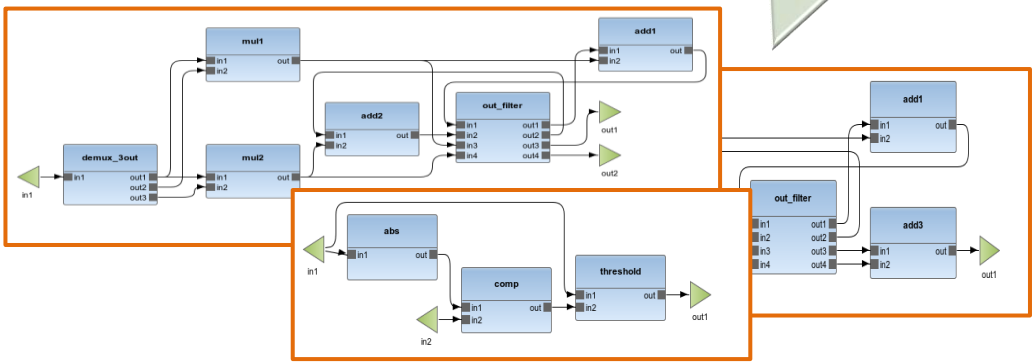


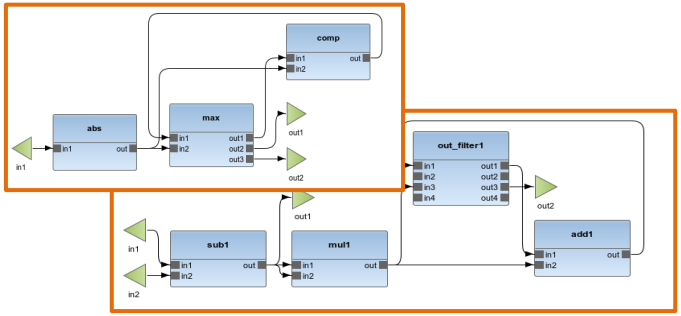
	# actors	#sbox
12 networks (dec_filter, Thr, rec_filter, NEO, idx_max_abs, Avg, sqr_sum, weight_mul, dot_prod, idx_max, sync_avg, sync_wavg)	46	0
MDC network	14	86

Neural Signal Decoding

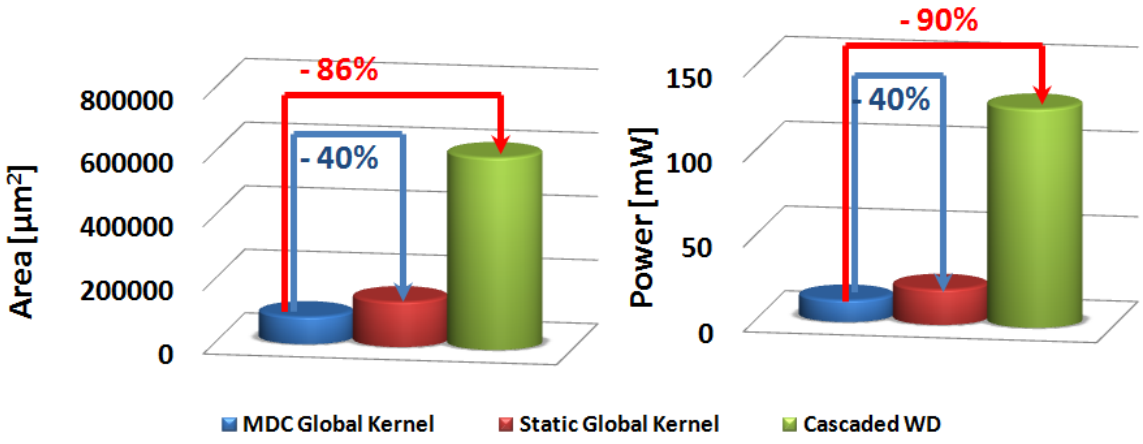
Resource Optimization







	# actors	#sbox
12 networks (dec_filter, Thr, rec_filter, NEO, idx_max_abs, Avg, sqr_sum, weight_mul, dot_prod, idx_max, sync_avg, sync_wavg)	46	0
MDC network	14	86



HEVC Interpolation Filters

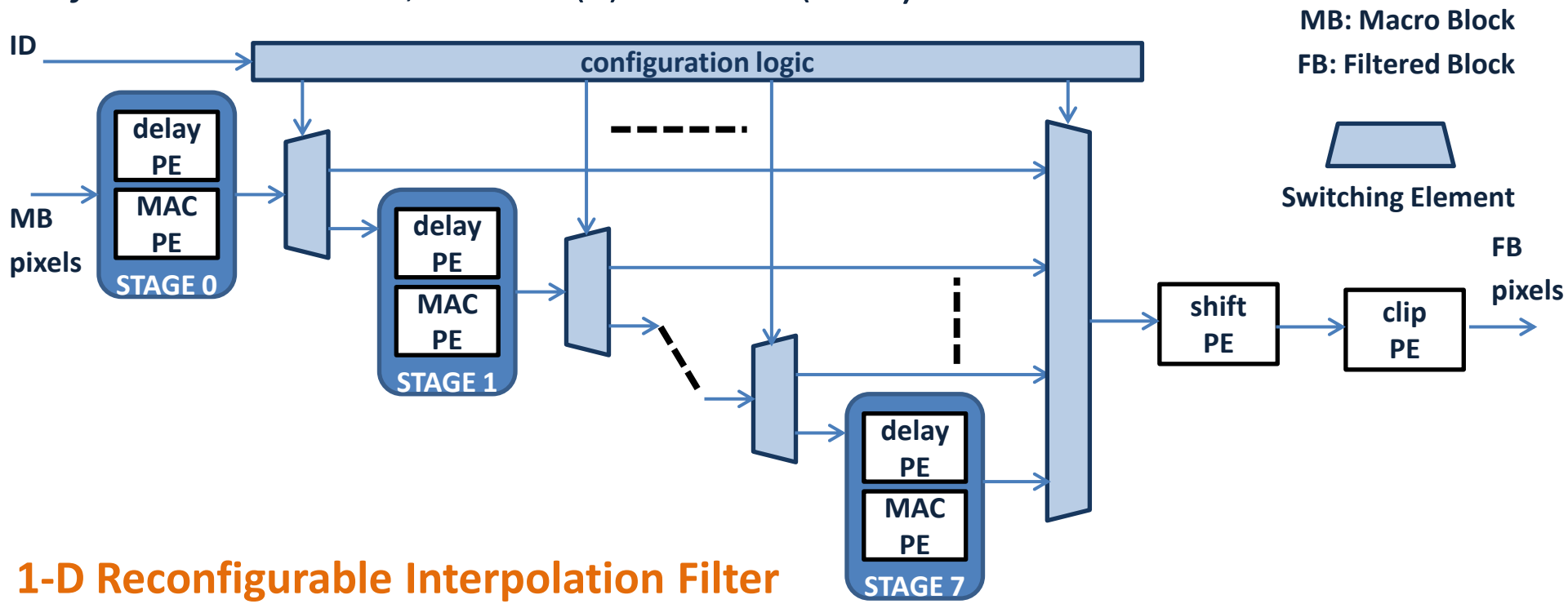
Multiple Working Points

- **Approximate Computing:** trading a controlled quality degradation (# taps) for an increased energy efficiency
- **Software Implementation**: Erwan Raffin, et al., "*Low power HEVC software decoder for mobile devices*", JRTIP 12(2): 495-507 (2016)

HEVC Interpolation Filters

Multiple Working Points

- **Approximate Computing:** trading a controlled quality degradation (# taps) for an increased energy efficiency
- **Software Implementation**: Erwan Raffin, et al., "*Low power HEVC software decoder for mobile devices*", JRTIP 12(2): 495-507 (2016)

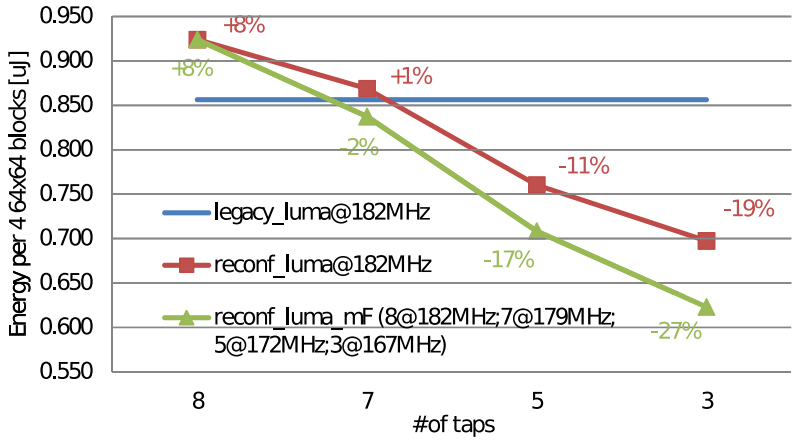


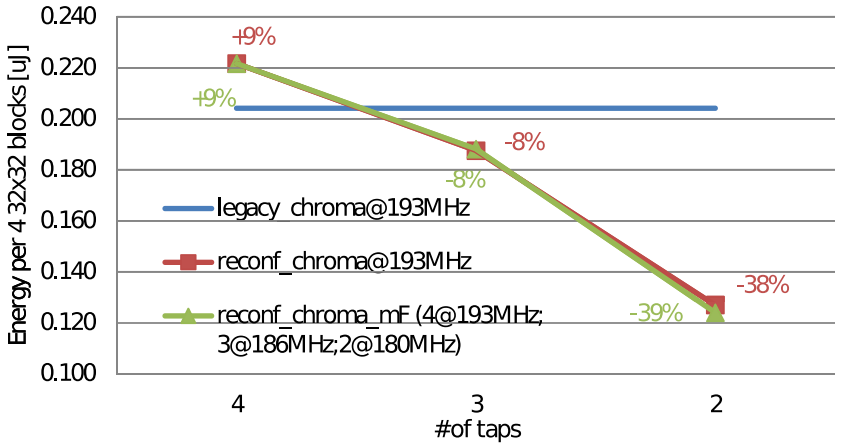
1-D Reconfigurable Interpolation Filter

HEVC Interpolation Filters

Multiple Working Points

design @200 MHz Xilinx XC7Z020	LUT	FF	BRAM	DSP	Fmax [MHz]	tap	dP (Vivado) [mW]	dE [µJ]	time per block [cycles]	# interpolated pixels in a fixed time
legacy_luma	212	37	4	16	213	8	11	0.248	460	57957
reconf_luma (vs legacy %)	582 **(+175%)**	85 (+130%)	4 (+0%)	16 (+0%)	200 **(-6%)**	8	12 (+9%)	0.270 (+9%)	460 (+0%)	57957 (+0%)
						7	11 (+0%)	0.245 (-1%)	395 (-14%)	59033 (+2%)
						5	10 (-9%)	0.217 (-12%)	265 (-42%)	61191 (+6%)
						3	10 **(-9%)**	0.211 **(-15%)**	135 **(-71%)**	63357 **(+9%)**
legacy_chroma	163	33	2	8	217	4	9	0.053	107	14753
reconf_chroma (vs legacy %)	383 **(+135%)**	65 (+97%)	2 (+0%)	8 (+0%)	200 **(-12%)**	4	9 (+0%)	0.053 (+0%)	107 (+0%)	14753 (+0%)
						3	8 (-11%)	0.045 (-13%)	73 (-32%)	15293 (+4%)
						2	6 **(-33%)**	0.033 **(-37%)**	39 **(-64%)**	15835 **(+7%)**



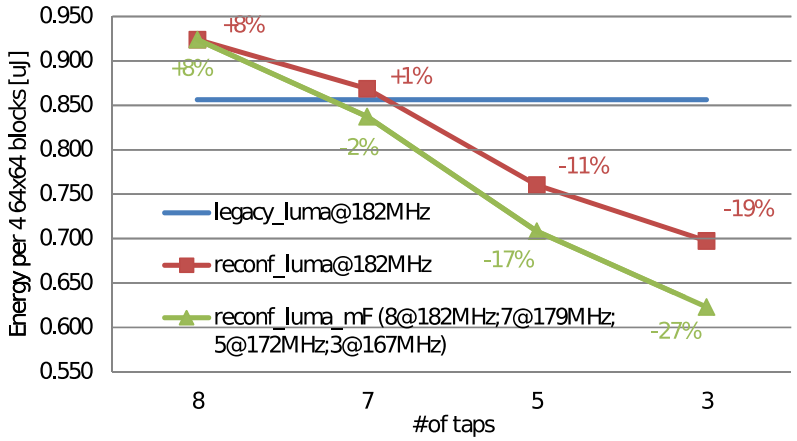


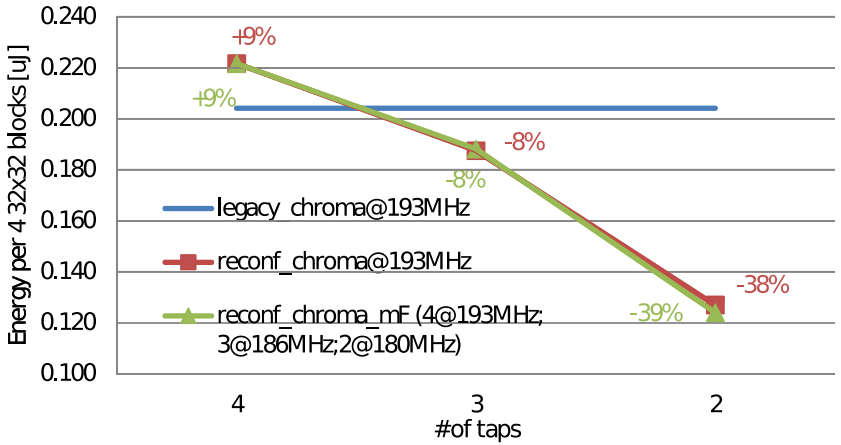
C. Sau et al. <<*Challenging the Best HEVC Fractional Pixel FPGA Interpolators with Reconfigurable and Multi-frequency Approximate Computing.*>> IEEE Embedded Systems Letters, 9 (3), pp. 65-68, 2017, ISSN: 1943-0663.

HEVC Interpolation Filters

Multiple Working Points

design @200 MHz Xilinx XC7Z020	LUT	FF	BRAM	DSP	Fmax [MHz]	tap	dP (Vivado) [mW]	dE [µJ]	time per block [cycles]	# interpolated pixels in a fixed time
legacy_luma	212	37	4	16	213	8	11	0.248	460	57957
						8	12 (+9%)	0.270 (+9%)	460 (+0%)	57957 (+0%)
reconf_luma (vs legacy %)	582 **(+175%)**	85 (+130%)	4 (+0%)	16 (+0%)	200 **(-6%)**	7	11 (+0%)	0.245 (-1%)	395 (-14%)	59033 (+2%)
						5	10 (-9%)	0.217 (-12%)	265 (-42%)	61191 (+6%)
						3	10 **(-9%)**	0.211 **(-15%)**	135 **(-71%)**	63357 **(+9%)**
legacy_chroma	163	33	2	8	217	4	9	0.053	107	14753
						4	9 (+0%)	0.053 (+0%)	107 (+0%)	14753 (+0%)
reconf_chroma (vs legacy %)	383 **(+135%)**	65 (+97%)	2 (+0%)	8 (+0%)	200 **(-12%)**	3	8 (-11%)	0.045 (-13%)	73 (-32%)	15293 (+4%)
						2	6 **(-33%)**	0.033 **(-37%)**	39 **(-64%)**	15835 **(+7%)**



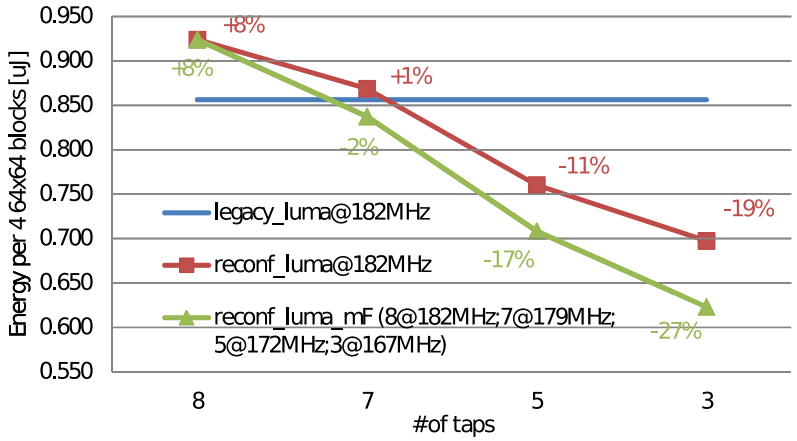


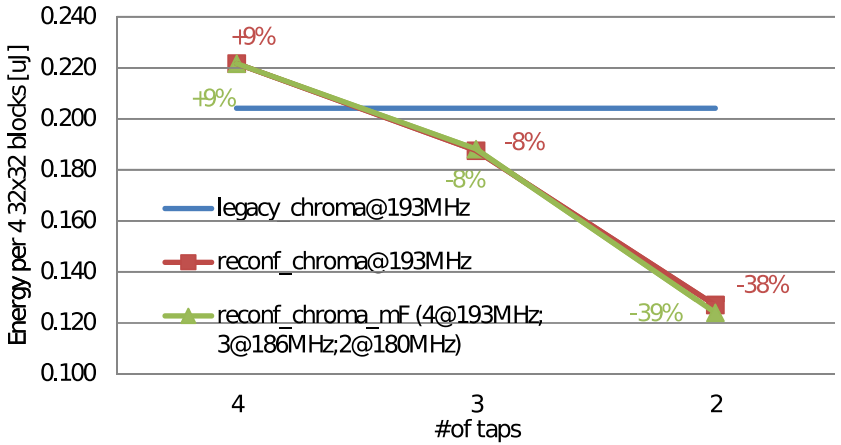
C. Sau et al. <<*Challenging the Best HEVC Fractional Pixel FPGA Interpolators with Reconfigurable and Multi-frequency Approximate Computing.*>> IEEE Embedded Systems Letters, 9 (3), pp. 65-68, 2017, ISSN: 1943-0663.

HEVC Interpolation Filters

Multiple Working Points

design @200 MHz Xilinx XC7Z020	LUT	FF	BRAM	DSP	Fmax [MHz]	tap	dP (Vivado) [mW]	dE [µJ]	time per block [cycles]	# interpolated pixels in a fixed time
legacy_luma	212	37	4	16	213	8	11	0.248	460	57957
						8	12 (+9%)	0.270 (+9%)	460 (+0%)	57957 (+0%)
reconf_luma (vs legacy %)	582 **(+175%)**	85 (+130%)	4 (+0%)	16 (+0%)	200 **(-6%)**	7	11 (+0%)	0.245 (-1%)	395 (-14%)	59033 (+2%)
						5	10 (-9%)	0.217 (-12%)	265 (-42%)	61191 (+6%)
						3	10 **(-9%)**	0.211 **(-15%)**	135 **(-71%)**	63357 **(+9%)**
legacy_chroma	163	33	2	8	217	4	9	0.053	107	14753
						4	9 (+0%)	0.053 (+0%)	107 (+0%)	14753 (+0%)
reconf_chroma (vs legacy %)	383 **(+135%)**	65 (+97%)	2 (+0%)	8 (+0%)	200 **(-12%)**	3	8 (-11%)	0.045 (-13%)	73 (-32%)	15293 (+4%)
						2	6 **(-33%)**	0.033 **(-37%)**	39 **(-64%)**	15835 **(+7%)**



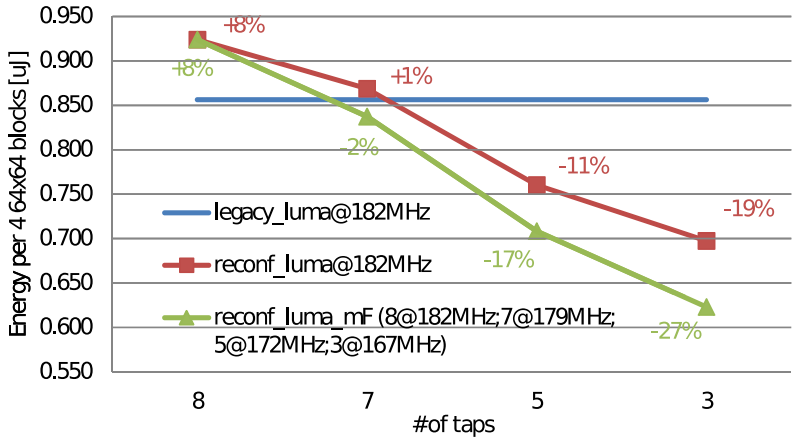


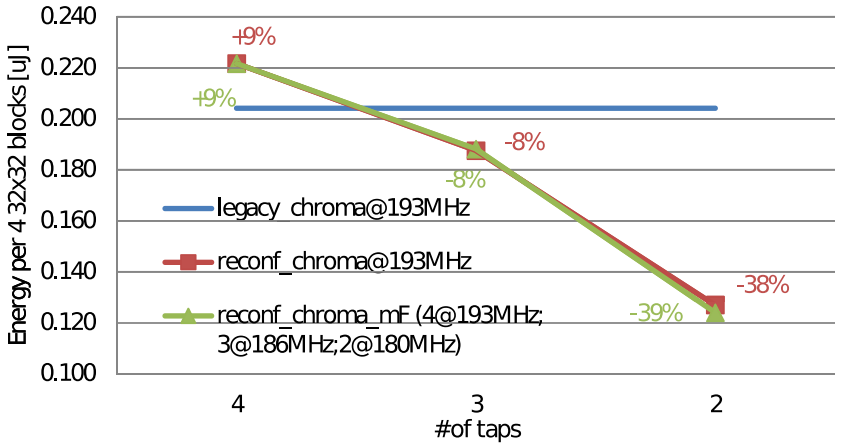
C. Sau et al. <<*Challenging the Best HEVC Fractional Pixel FPGA Interpolators with Reconfigurable and Multi-frequency Approximate Computing.*>> IEEE Embedded Systems Letters, 9 (3), pp. 65-68, 2017, ISSN: 1943-0663.

HEVC Interpolation Filters

Multiple Working Points

design @200 MHz Xilinx XC7Z020	LUT	FF	BRAM	DSP	Fmax [MHz]	tap	dP (Vivado) [mW]	dE [µJ]	time per block [cycles]	# interpolated pixels in a fixed time
legacy_luma	212	37	4	16	213	8	11	0.248	460	57957
						8	12 (+9%)	0.270 (+9%)	460 (+0%)	57957 (+0%)
reconf_luma (vs legacy %)	582 **(+175%)**	85 (+130%)	4 (+0%)	16 (+0%)	200 **(-6%)**	7	11 (+0%)	0.245 (-1%)	395 (-14%)	59033 (+2%)
						5	10 (-9%)	0.217 (-12%)	265 (-42%)	61191 (+6%)
						3	10 **(-9%)**	0.211 **(-15%)**	135 **(-71%)**	63357 **(+9%)**
legacy_chroma	163	33	2	8	217	4	9	0.053	107	14753
						4	9 (+0%)	0.053 (+0%)	107 (+0%)	14753 (+0%)
reconf_chroma (vs legacy %)	383 **(+135%)**	65 (+97%)	2 (+0%)	8 (+0%)	200 **(-12%)**	3	8 (-11%)	0.045 (-13%)	73 (-32%)	15293 (+4%)
						2	6 **(-33%)**	0.033 **(-37%)**	39 **(-64%)**	15835 **(+7%)**



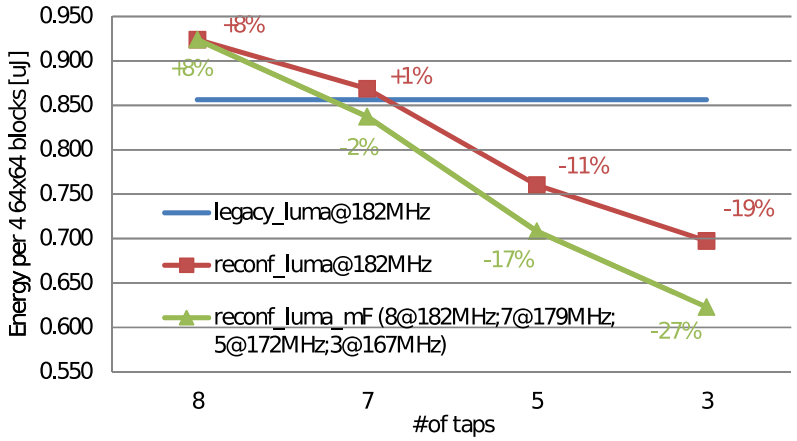


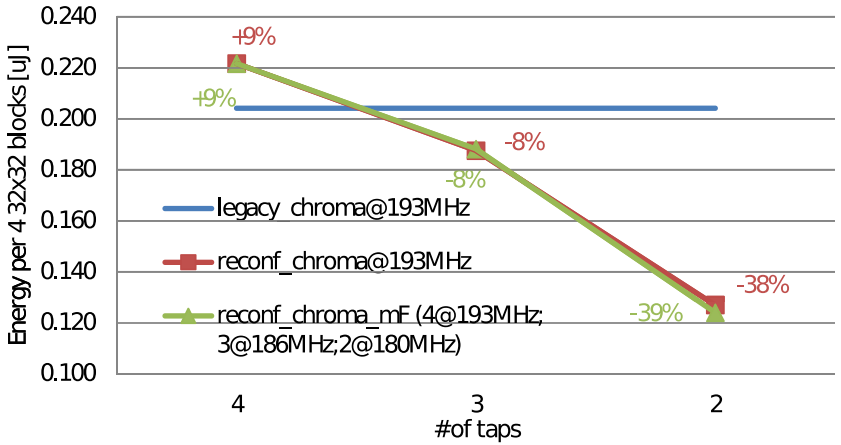
C. Sau et al. <<*Challenging the Best HEVC Fractional Pixel FPGA Interpolators with Reconfigurable and Multi-frequency Approximate Computing.*>> IEEE Embedded Systems Letters, 9 (3), pp. 65-68, 2017, ISSN: 1943-0663.

HEVC Interpolation Filters

Multiple Working Points

design @200 MHz Xilinx XC7Z020	LUT	FF	BRAM	DSP	Fmax [MHz]	tap	dP (Vivado) [mW]	dE [µJ]	time per block [cycles]	# interpolated pixels in a fixed time
legacy_luma	212	37	4	16	213	8	11	0.248	460	57957
						8	12 (+9%)	0.270 (+9%)	460 (+0%)	57957 (+0%)
reconf_luma (vs legacy %)	582 **(+175%)**	85 (+130%)	4 (+0%)	16 (+0%)	200 **(-6%)**	7	11 (+0%)	0.245 (-1%)	395 (-14%)	59033 (+2%)
						5	10 (-9%)	0.217 (-12%)	265 (-42%)	61191 (+6%)
						3	10 **(-9%)**	0.211 **(-15%)**	135 **(-71%)**	63357 **(+9%)**
legacy_chroma	163	33	2	8	217	4	9	0.053	107	14753
						4	9 (+0%)	0.053 (+0%)	107 (+0%)	14753 (+0%)
reconf_chroma (vs legacy %)	383 **(+135%)**	65 (+97%)	2 (+0%)	8 (+0%)	200 **(-12%)**	3	8 (-11%)	0.045 (-13%)	73 (-32%)	15293 (+4%)
						2	6 **(-33%)**	0.033 **(-37%)**	39 **(-64%)**	15835 **(+7%)**





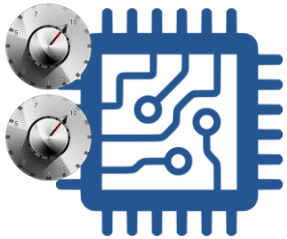
C. Sau et al. <<*Challenging the Best HEVC Fractional Pixel FPGA Interpolators with Reconfigurable and Multi-frequency Approximate Computing.*>> IEEE Embedded Systems Letters, 9 (3), pp. 65-68, 2017, ISSN: 1943-0663.

PROSSIMO
Progettazione, sviluppo e ottimizzazione di sistemi intelligenti multi-oggetto

Architetture eterogenee on-chip riconfigurabili: analisi, sviluppo e caratterizzazione del loro comportamento con sistemi di monitoring

Triggers for Adaptation

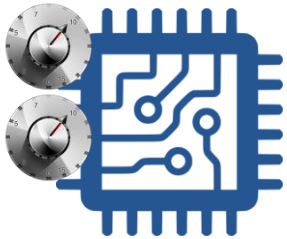
Triggers for Adaptation



Adaptable Hardware Accelerator

How to Decide When and How to Adapt?

Triggers for Adaptation

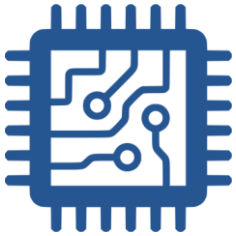


Adaptable Hardware Accelerator

How to Decide When and How to Adapt?

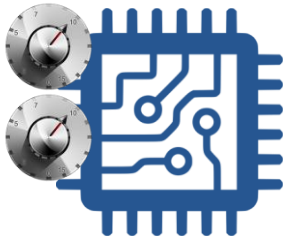
ENVIRONMENTAL AWARENESS: Influence of the environment on the system, i.e. daylight vs. nocturnal, radiation level changes, etc.
Sensors are needed to interact with the environment and capture conditions variations.

USER/EXTERNALLY-COMMANDED: System-User interaction, i.e. user preferences, commands from SoS managers (the boss), etc.
Proper human-machine interfaces are needed to enable interaction and capture commands.



SELF-AWARENESS: The internal status of the system varies while operating and may lead to reconfiguration needs, i.e. chip temperature variation, low battery.
Status monitors are needed to capture the status of the system.

Triggers for Adaptation

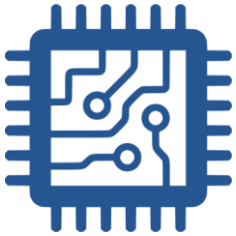


Adaptable Hardware Accelerator

How to Decide When and How to Adapt?

ENVIRONMENTAL AWARENESS: Influence of the environment on the system, i.e. daylight vs. nocturnal, radiation level changes, etc.
Sensors are needed to interact with the environment and capture conditions variations.

USER/EXTERNALLY-COMMANDED: System-User interaction, i.e. user preferences, commands from SoS managers (the boss), etc.
Proper human-machine interfaces are needed to enable interaction and capture commands.



SELF-AWARENESS: The internal status of the system varies while operating and may lead to reconfiguration needs, i.e. chip temperature variation, low battery.
Status monitors are needed to capture the status of the system.

Need for Monitoring

- Cyber-Physical Systems are **adaptive** to changing requirements, among which **metrics related to the system itself**
  - **understanding** such **metrics** becomes increasingly **difficult** if systems are complex
  - e.g. to obtain information on the run-time behaviour of threads, **visibility** into the processor **architecture** is **needed** to monitor workload interactions

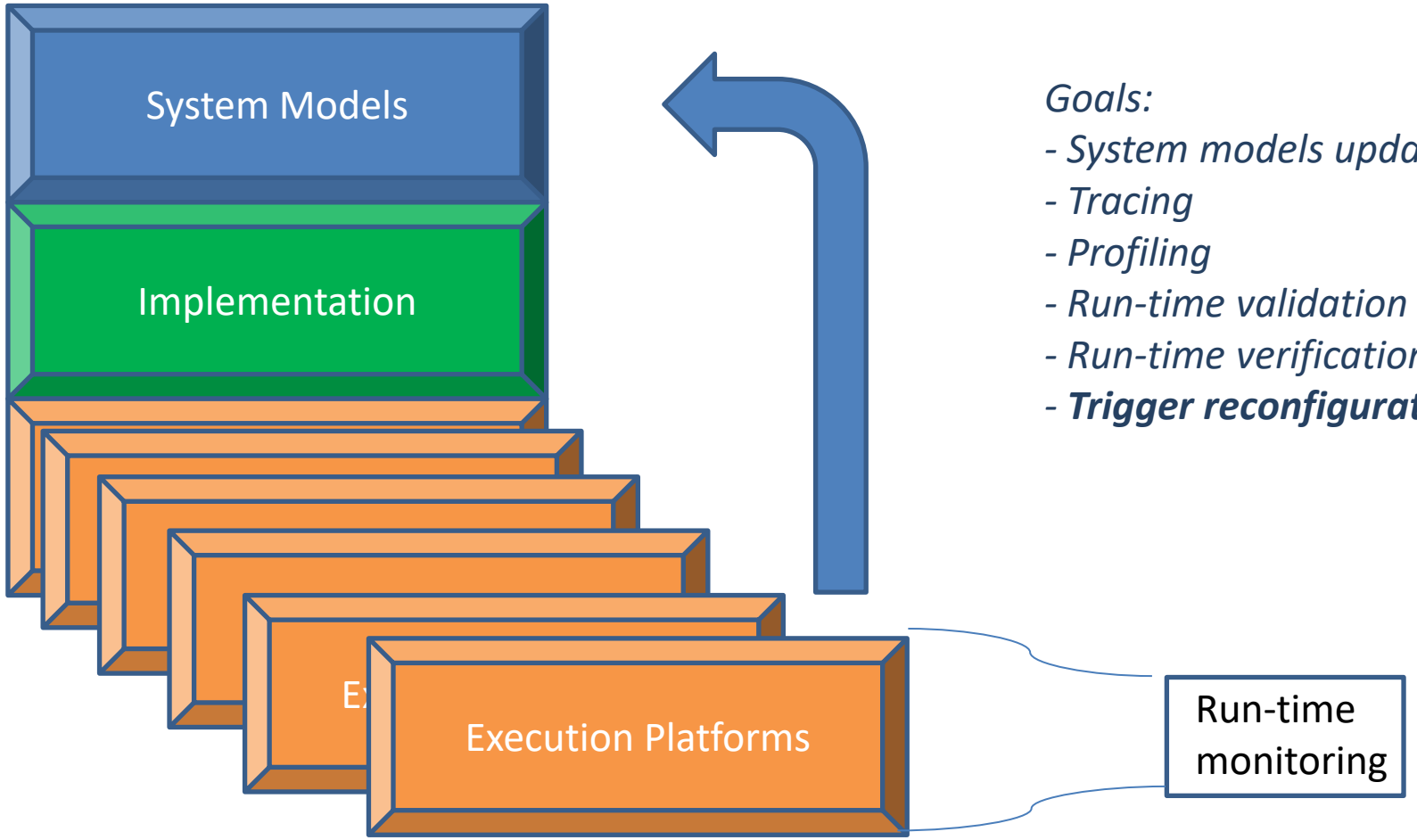
Need for Monitoring

- Cyber-Physical Systems are **adaptive** to changing requirements, among which **metrics related to the system itself**
  - **understanding** such **metrics** becomes increasingly **difficult** if systems are complex
  - e.g. to obtain information on the run-time behaviour of threads, **visibility** into the processor **architecture** is **needed** to monitor workload interactions

Need for Monitoring

- Cyber-Physical Systems are **adaptive** to changing requirements, among which **metrics related to the system itself**
  - **understanding** such **metrics** becomes increasingly **difficult** if systems are complex
  - e.g. to obtain information on the run-time behaviour of threads, **visibility** into the processor **architecture** is **needed** to monitor workload interactions
- **Simulators** represent a first answer but
  - often **focus** on a **particular level** in the **system hierarchy** (due to performance and complexity issues
  - **slow down execution** when implemented in software and provide such a combined level of visibility
- **Monitoring** is a valid alternative

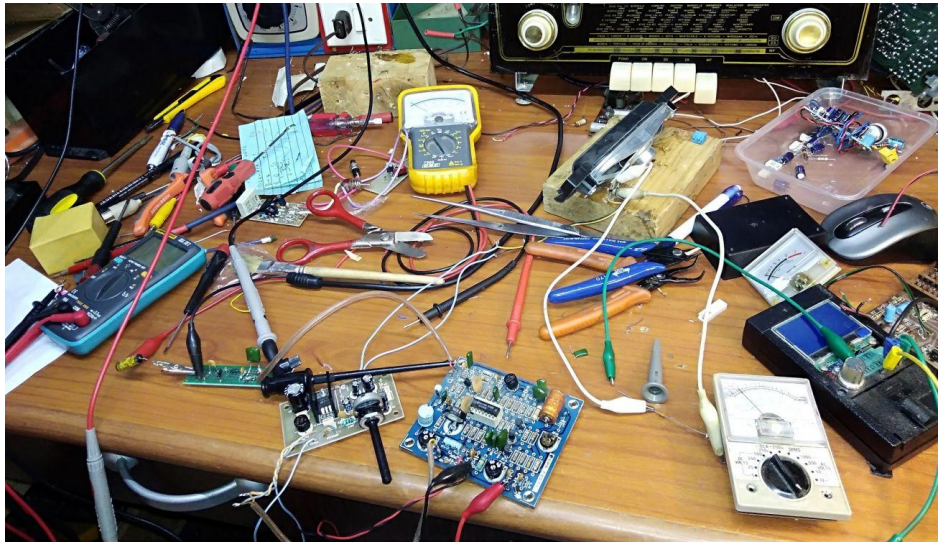
Monitoring goals



Goals:
- *System models update*
- *Tracing*
- *Profiling*
- *Run-time validation*
- *Run-time verification*
- ***Trigger reconfiguration***

Issues of Monitoring

- **What** is the object of monitoring?
- **When** it has to be monitored?
- **How** it is possible to **monitor** it?
- **How** monitored data should be **interpreted**?
- CPS challenges
  - Complexity
  - Adaptivity
  - Heterogeneity



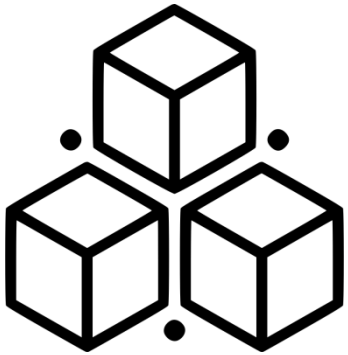
PROSSIMO
Progettazione, sviluppo e ottimizzazione di sistemi intelligenti multi-oggetto

Architetture eterogenee on-chip riconfigurabili: analisi, sviluppo e caratterizzazione del loro comportamento con sistemi di monitoring

Structure of the Course

Structure of the Course

The course is organized in three steps:

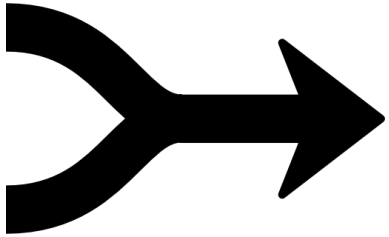


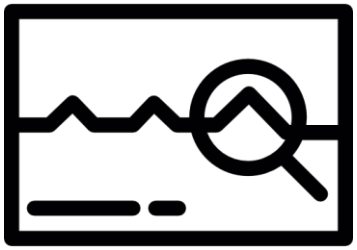
Step 1: practical exercises on

Getting Familiar with the Orcc Environment

Step 2: theory and practical exercises on

Multi-Dataflow Composer tool





Step 3: theory and practical exercises on

Monitoring

Step 1: Getting Familiar with the Orcc Environment

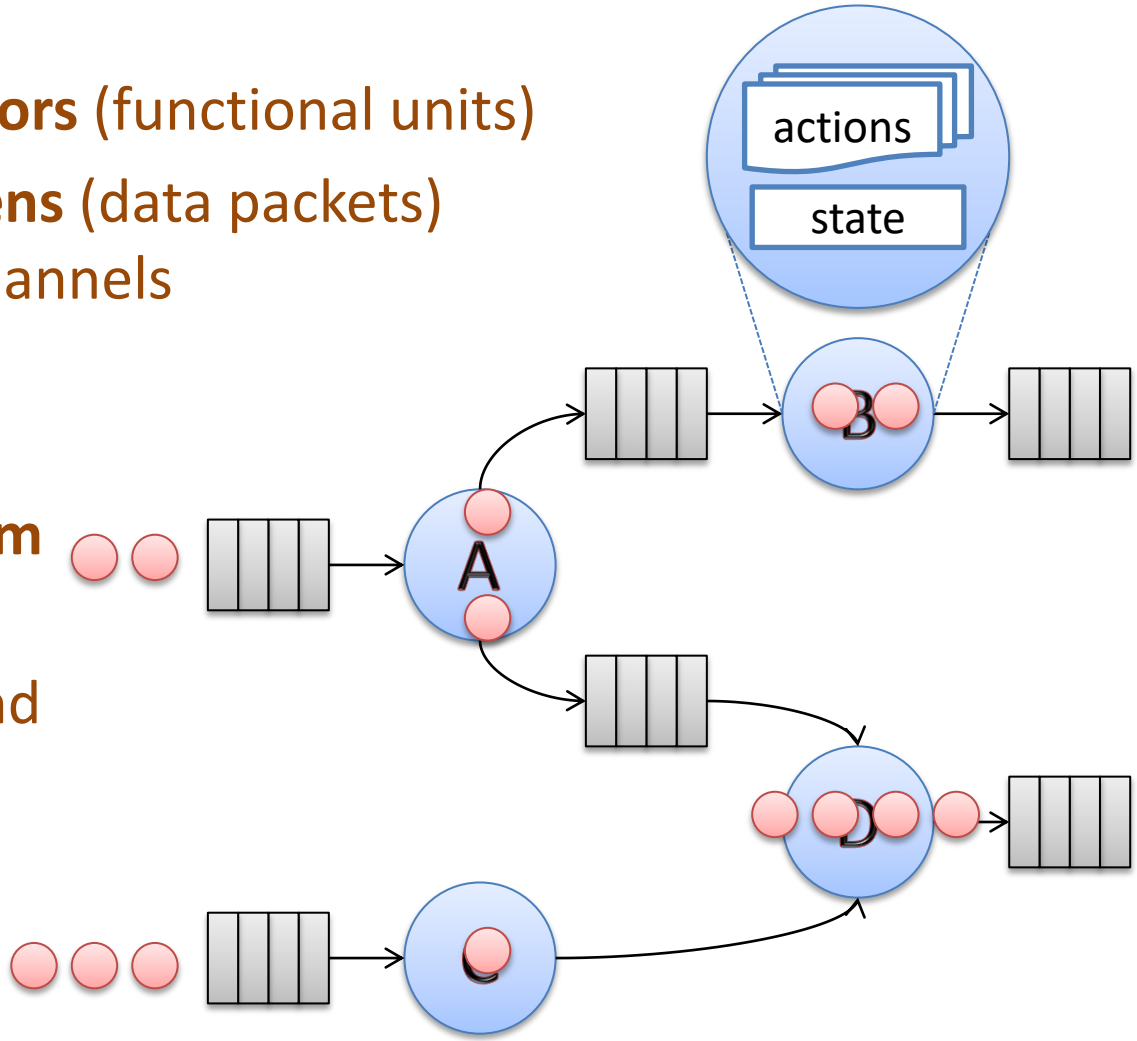
Tutorial

Model Driven Design

Dataflow Models

- Directed graph of **actors** (functional units)
- Actors exchange **tokens** (data packets) through dedicated channels

- Explicit intrinsic application **parallelism**
- **Modularity** favours model **re-usability** and **adaptivity**

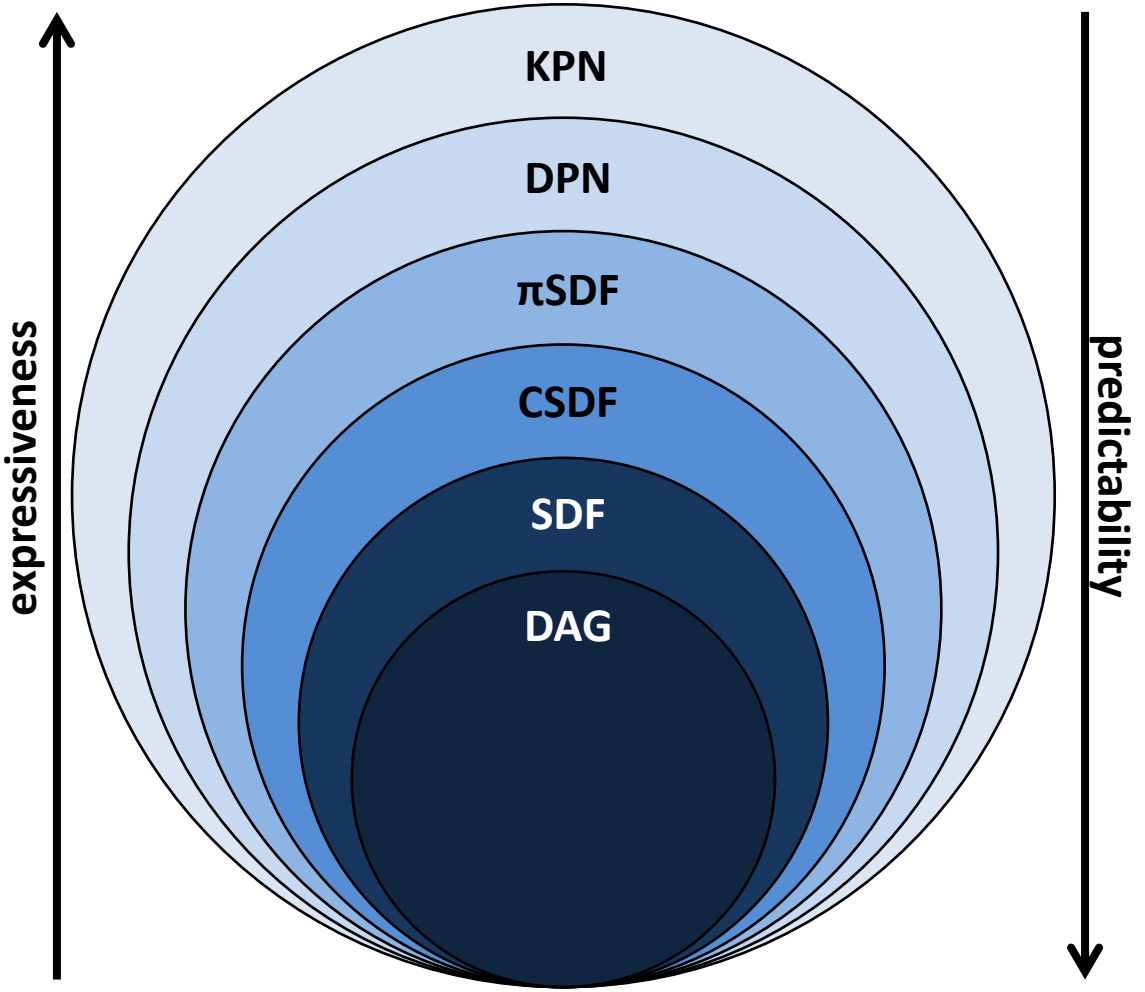


Model Driven Design

Dataflow Models

Several Models
- depending on how actors process tokens

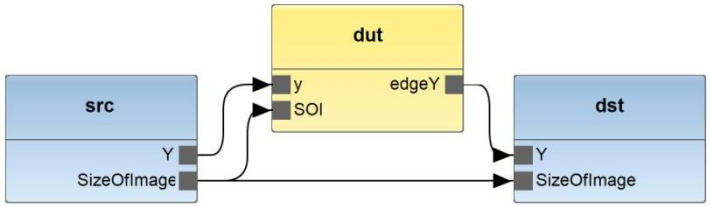
e.g. SDF has
- fixed token rates for reading and writing



Model Driven Design

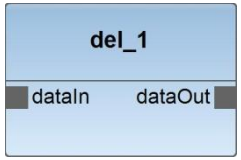
RVC-CAL Dataflow Formalism

XDF Networks




```
<?xml version="1.0" encoding="UTF-8"?>
  <XDF name="Testbench">
    <Instance id="src">
      <Class name="common.SourceImage"/>
    </Instance>
    <Instance id="dst">
      <Class name="common.ShowImage"/>
    </Instance>
    <Instance id="dut">
      <Class name="baseline.Sobel"/>
    </Instance>
    <Connection dst="dut" dst-port="y" src="src" src-port="Y"/>
    <Connection dst="dst" dst-port="SizeOfImage" src="src"
src-port="SizeOfImage"/>
    <Connection dst="dut" dst-port="SOI" src="src" src-port="SizeOfImage"/>
    <Connection dst="dst" dst-port="Y" src="dut" src-port="edgeY"/>
  </XDF>
```

CAL Actors

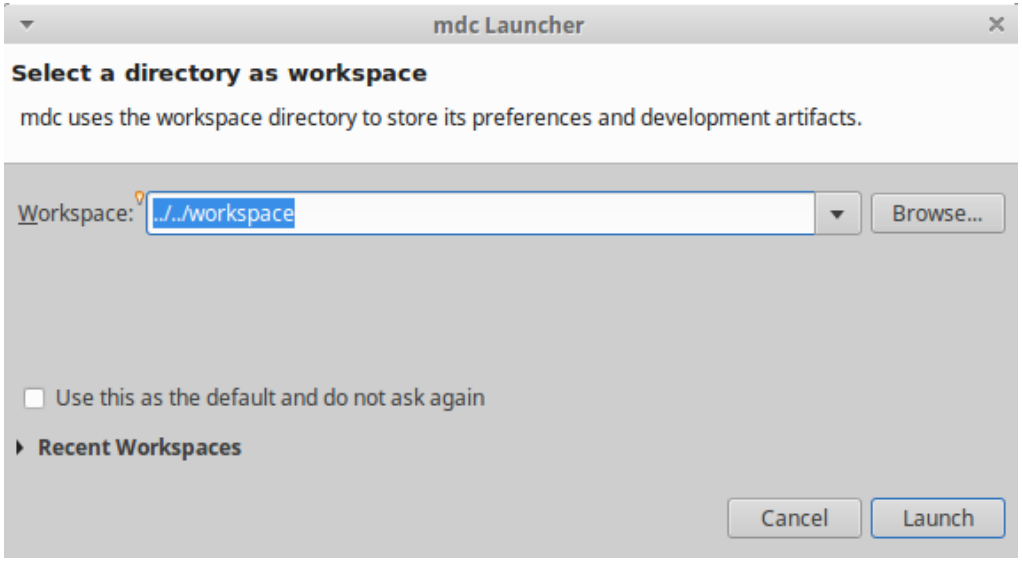



```
package common;

actor Delay()
 uint(size=8) dataIn ==>
 uint(size=8) dataOut :

 uint(size=8) dataReg := 0;

 action dataIn:[dataNew] ==> dataOut:[data]
 var uint(size=8) data
 do
 data := dataReg;
 dataReg := dataNew;
 end

end
```

Try It Yourself

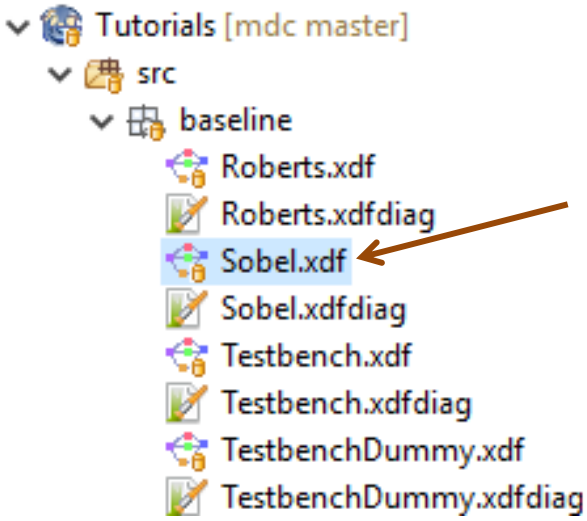
Launch the MDC tool

1. Double click on *mdc* in /home/monitoring_cgr/Tutorial/cgr/mdc/eclipse/mdc
2. Leave as workspace the default one (../../workspace)

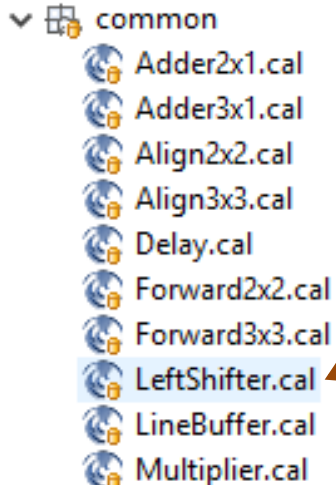


Try It Yourself

Open XDF and CAL files



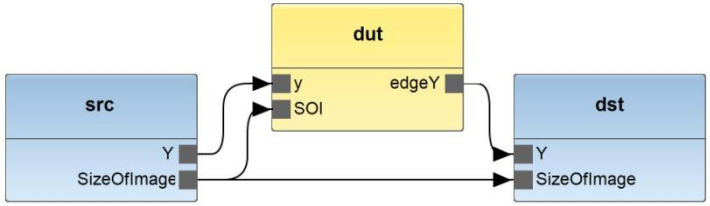
XDF file with default graphical editor on double click



CAL file with default graphical editor on double click

right click → Open With

- **Network/CAL Editor (graphical)**
- Text editor
- System Editor



Double click on an instance (sub-network in yellow, actor in blue) on the network editor to open it

Test Case

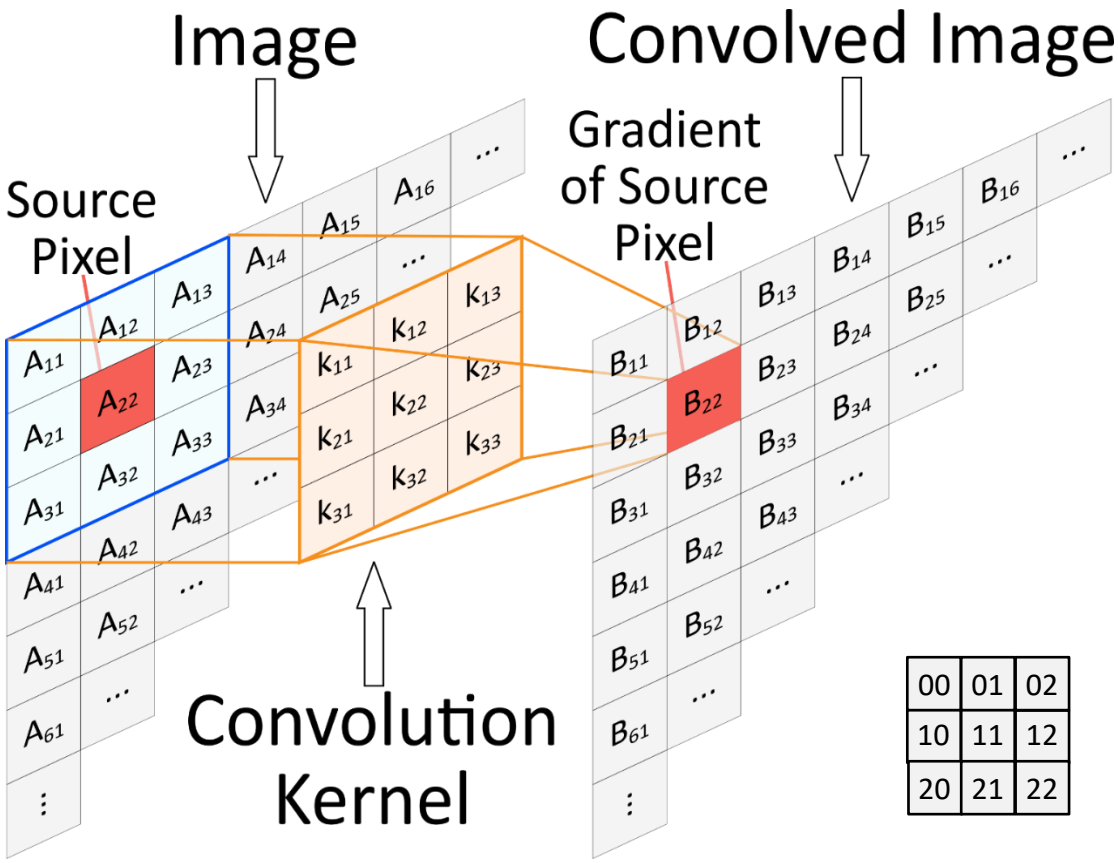
Edge Detection

Sobel Operator

$$\mathbf{G} = \sqrt{\mathbf{G}_x{}^2 + \mathbf{G}_y{}^2}$$

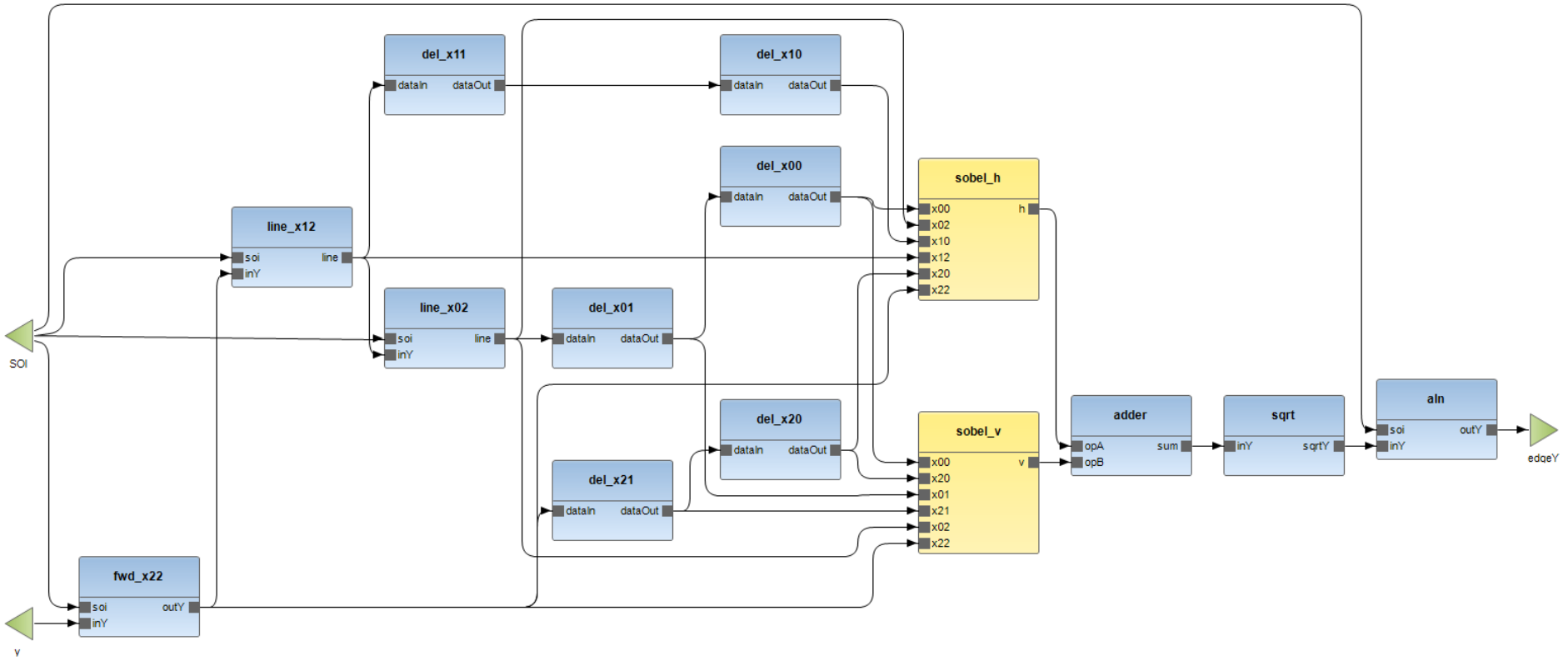
$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$$

$$\mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$



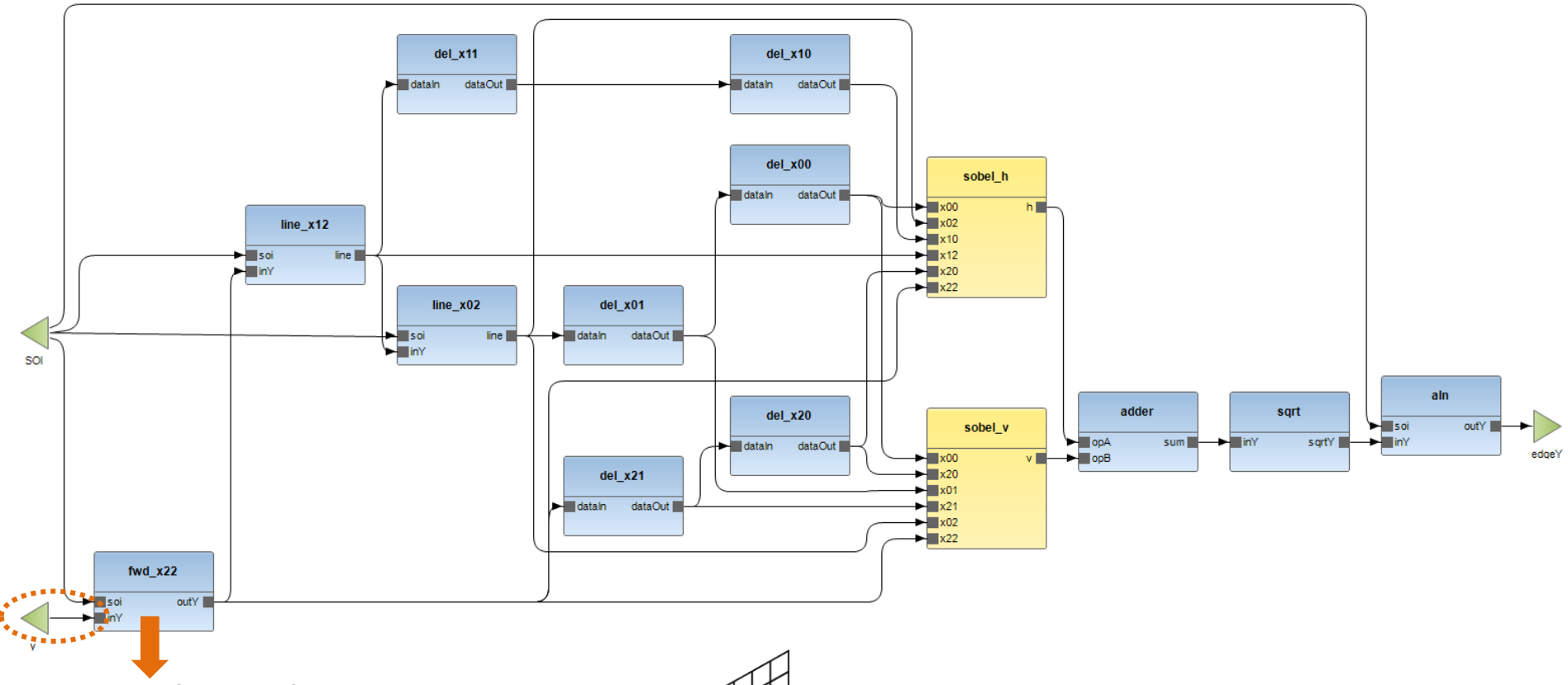
Test Case

Sobel XDF



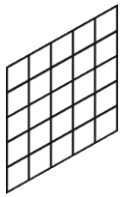
Test Case

Explore Sobel XDF



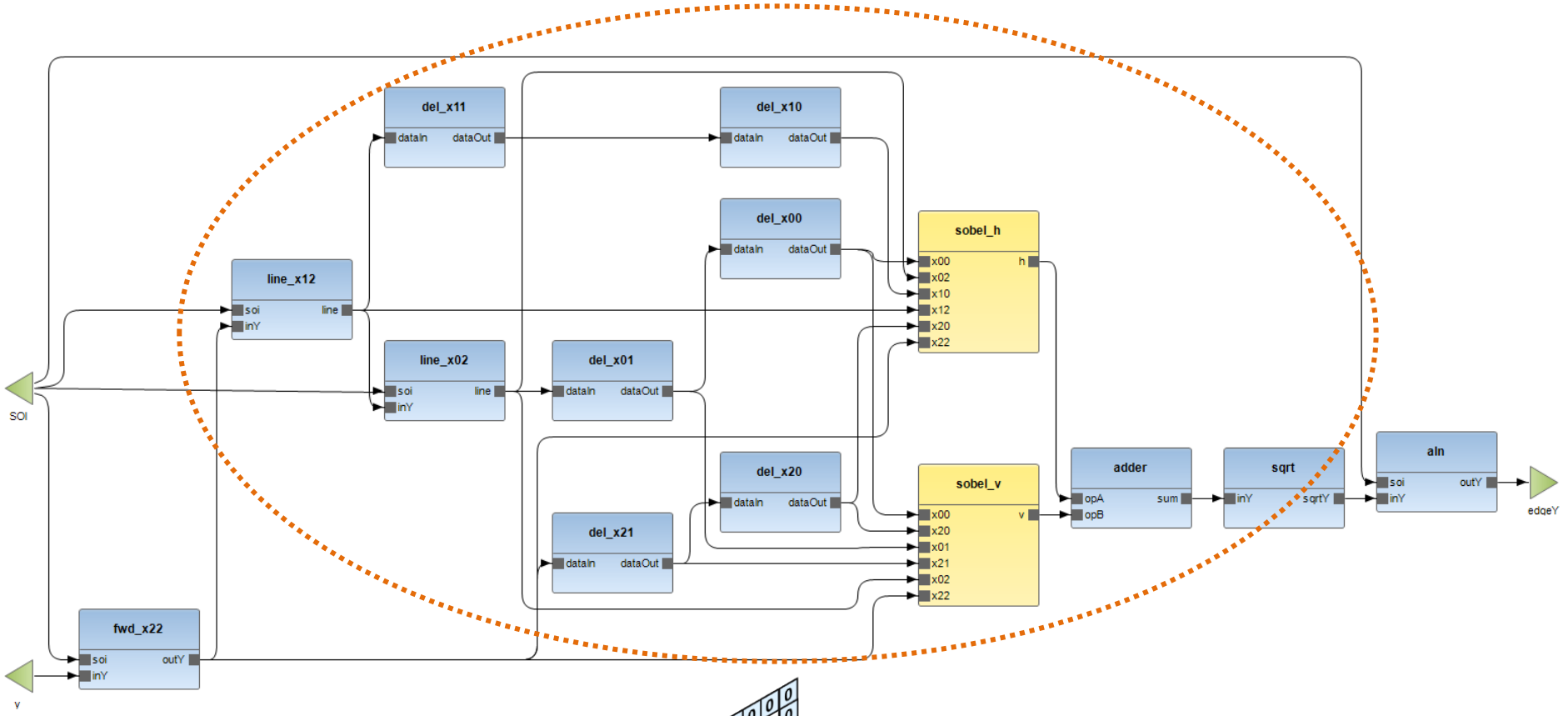
Forward3x3.cal actor

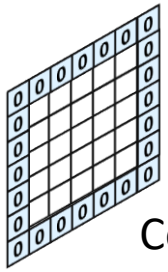
Add 2 rows and 2 columns frame all around the input image



Test Case

Explore Sobel XDF

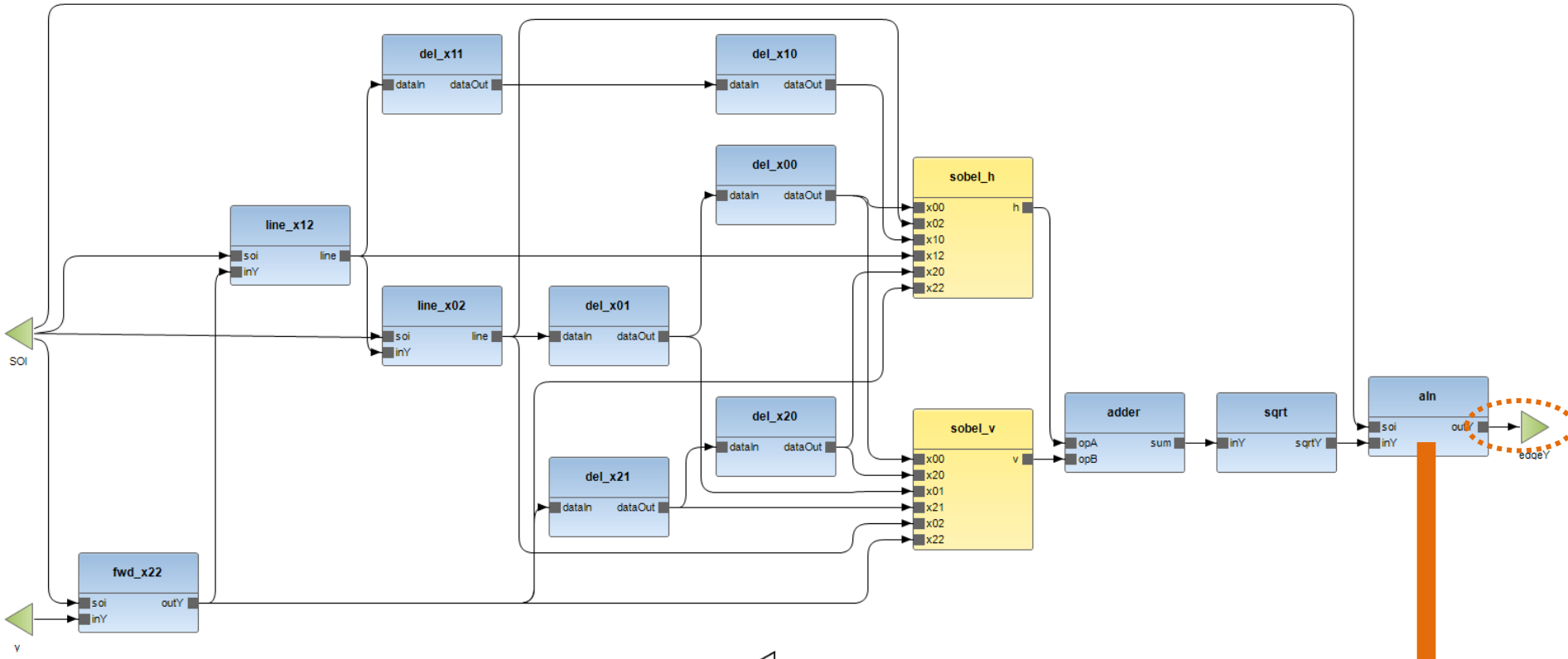


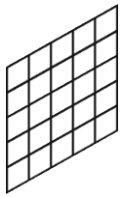


Computation on the extended image

Test Case

Explore Sobel XDF



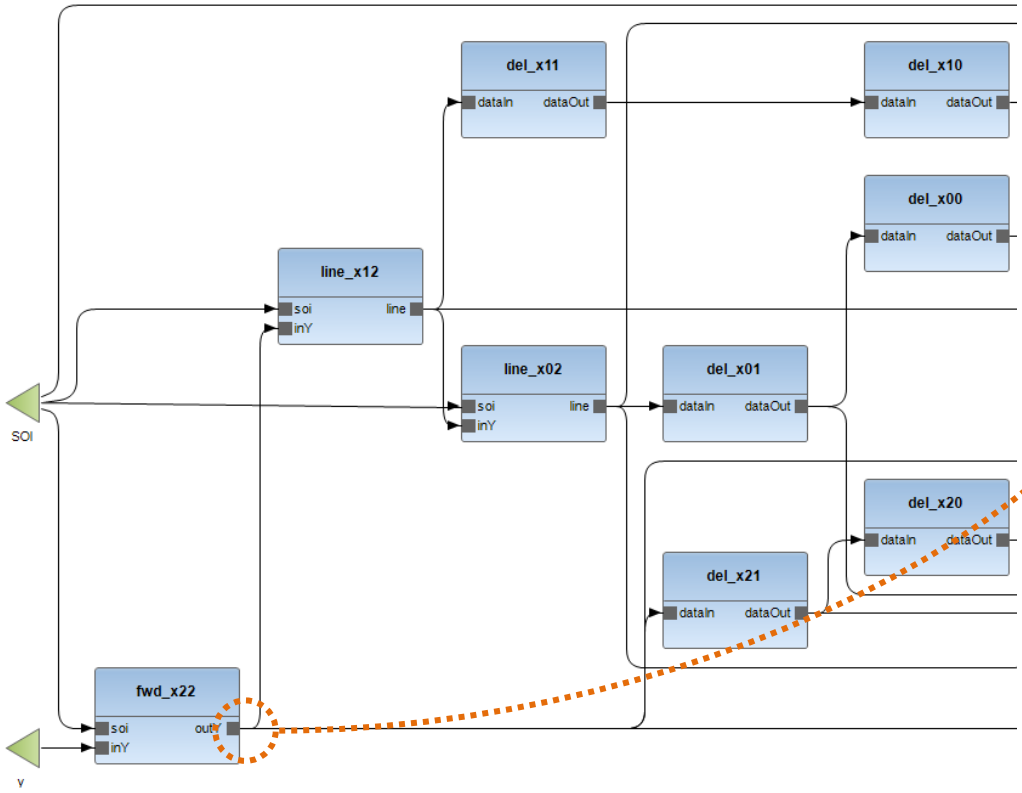


Align3x3.cal actor

Remove 2 rows and 2 columns frame from the input image

Try It Yourself

Explore Sobel XDF

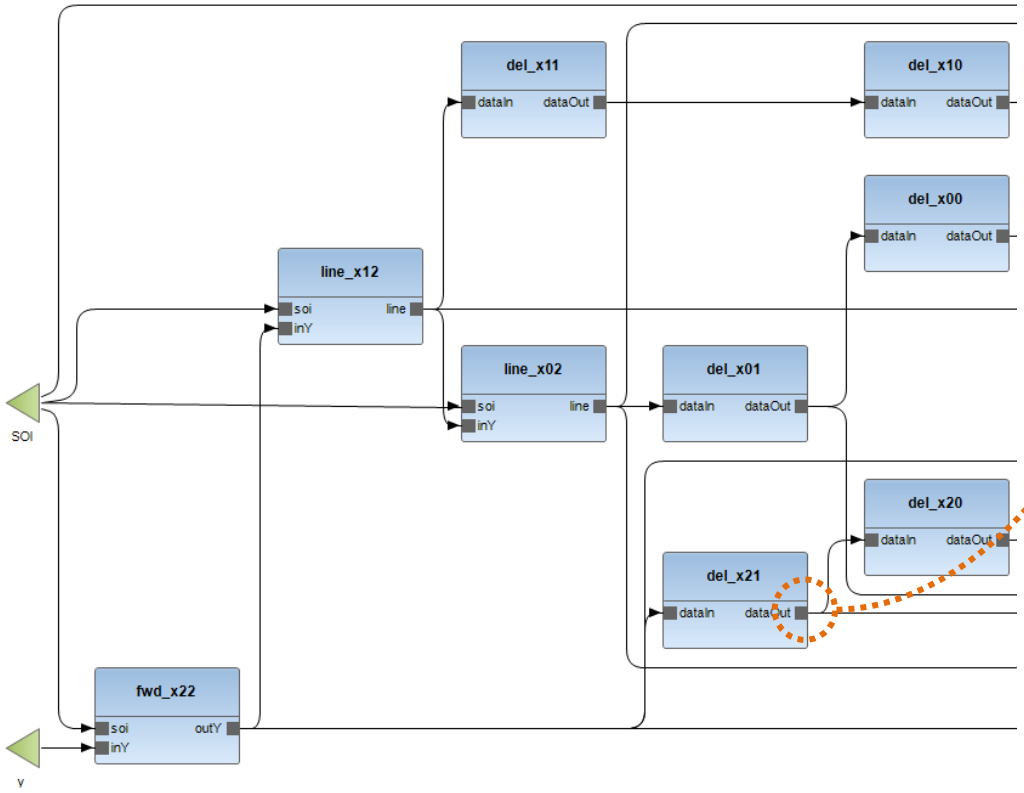


00	01	02			…
10	11	12			…
20	21	22			…
					…
					…
					…
					…
					…
					…
					…

Since the image comes **pixel by pixel**, it is necessary to **build 3x3 sub-images** on which the **convolution kernel** has to be applied

Try It Yourself

Explore Sobel XDF



00	01	02			…
10	11	12			…
20	21	22			…
					…
					…
					…
					…
					…
					…
					…

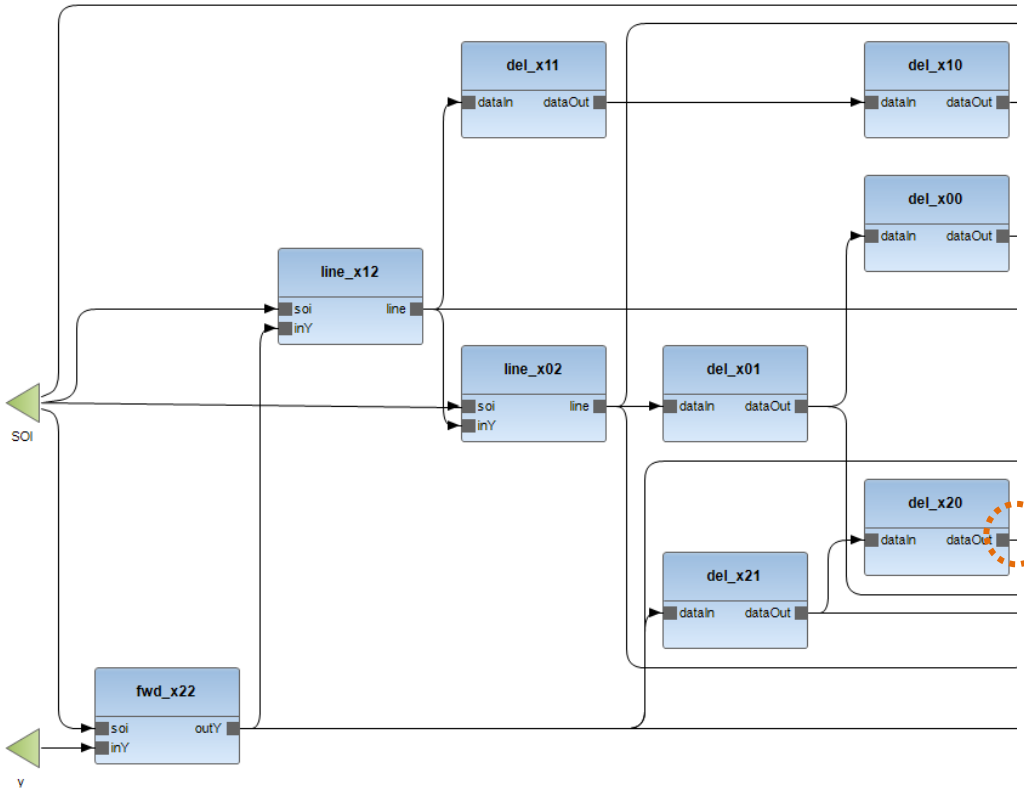
Delay.cal actor

```
action dataIn:[dataNew] ==> dataOut:[data]
   var uint(size=8) data
   do
      data := dataReg;
      dataReg := dataNew;
   end
```

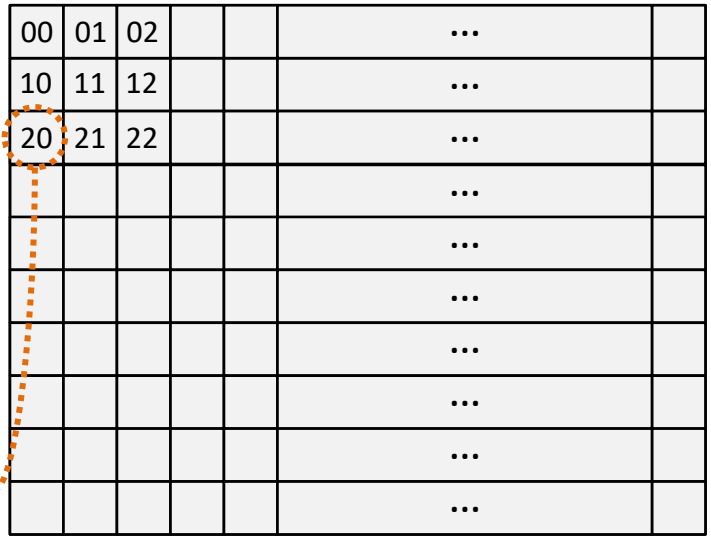
Delay actor **stores one pixel**

Try It Yourself

Explore Sobel XDF



Delay actor **stores one pixel**

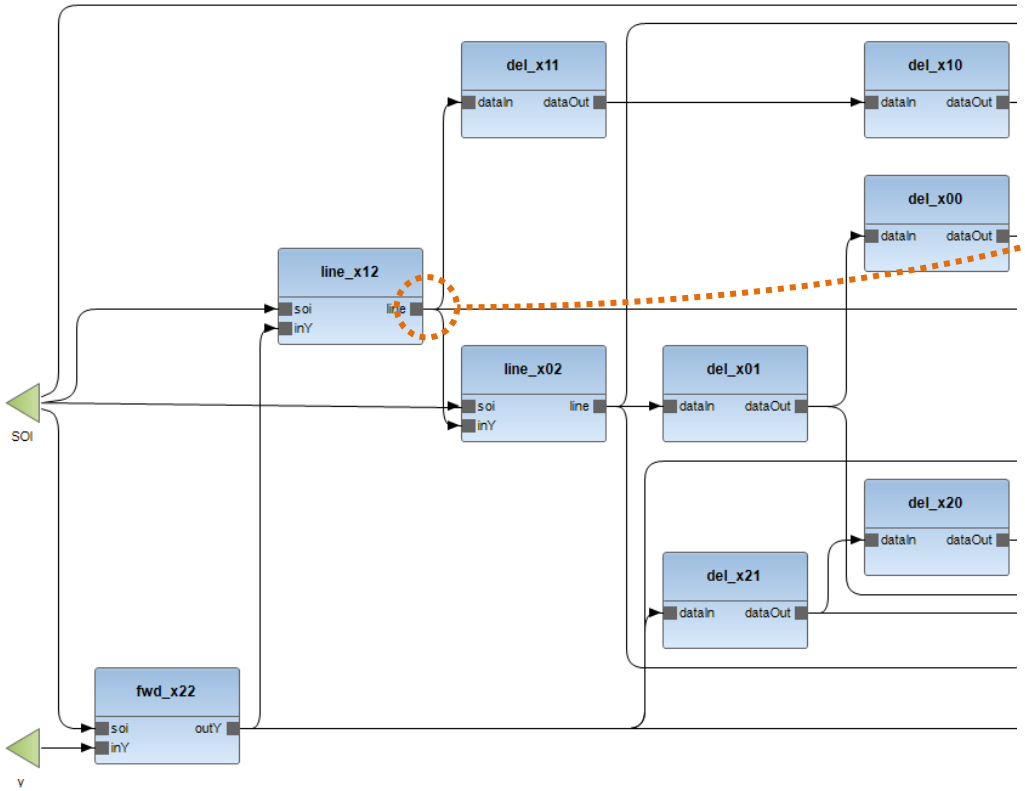


Delay.cal actor

```
action dataIn:[dataNew] ==> dataOut:[data]
   var uint(size=8) data
   do
      data := dataReg;
      dataReg := dataNew;
   end
```

Try It Yourself

Explore Sobel XDF



00	01	02			…	
10	11	12			…	
20	21	22			…	
					…	
					…	
					…	
					…	
					…	
					…	
					…	

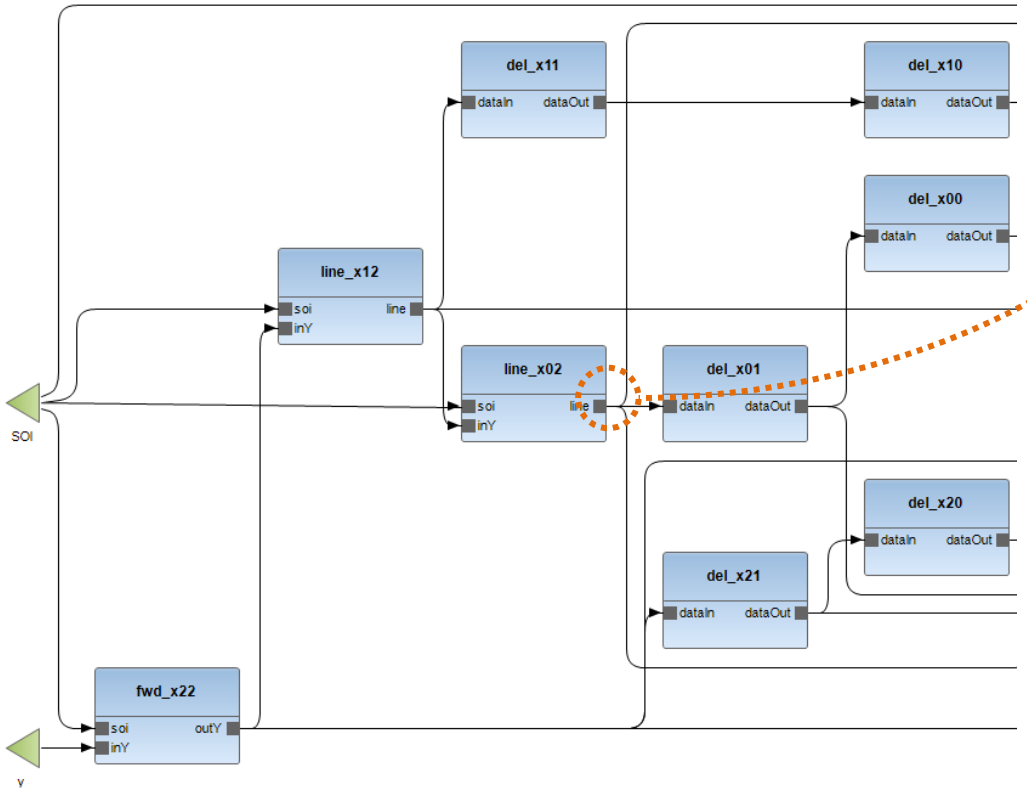
LineBuffer.cal actor

```
action Y:[inY] ==> Line:[outY]
  var uint(size=8) outY
  do
     outY := lineBuffer[x];
     lineBuffer[x]:= inY;
     if x = width then
        x := 0;
        if y = height then
           y := 0;                    …
```

LineBuffer actor **stores one row of pixels**

Try It Yourself

Explore Sobel XDF



00	01	02			…
10	11	12			…
20	21	22			…
					…
					…
					…
					…
					…
					…
					…

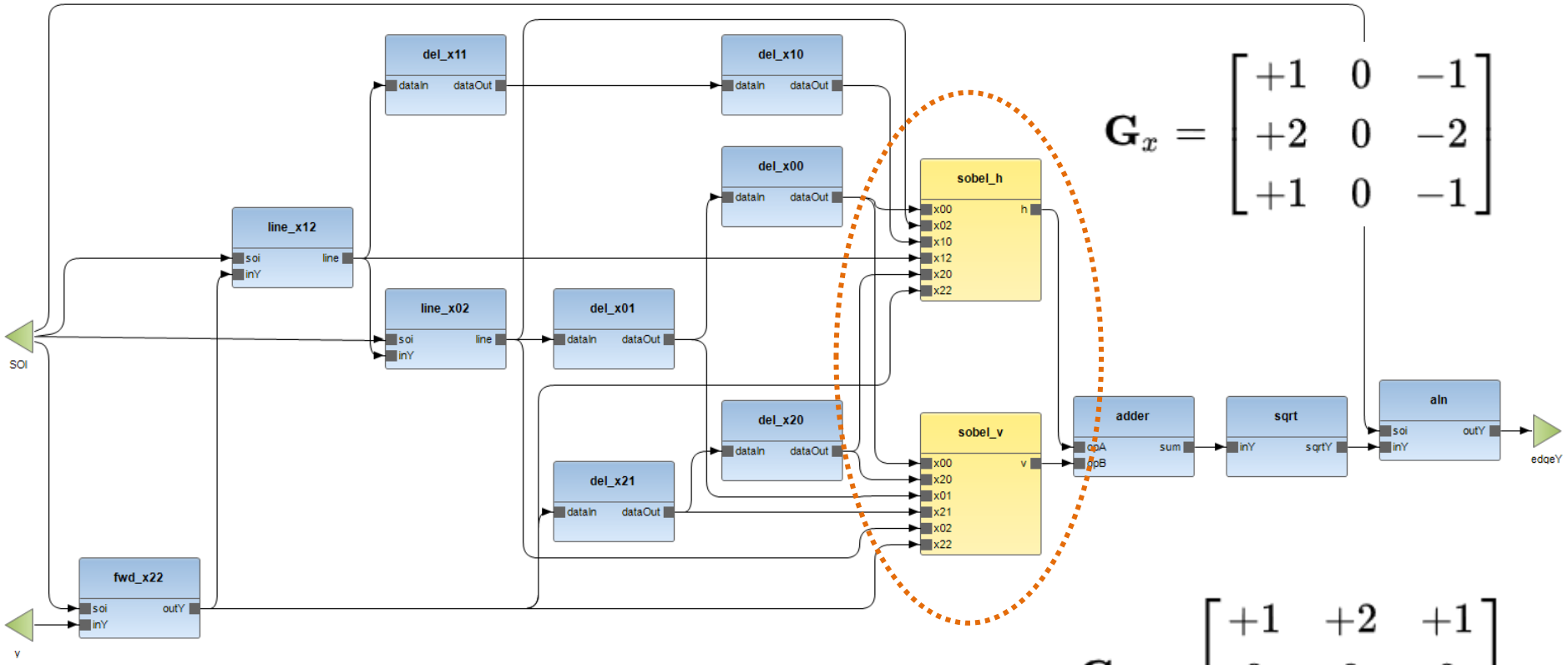
LineBuffer actor **stores one row of pixels**

LineBuffer.cal actor

```
action Y:[inY] ==> Line:[outY]
  var uint(size=8) outY
  do
     outY := lineBuffer[x];
     lineBuffer[x]:= inY;
     if x = width then
        x := 0;
        if y = height then
           y := 0;                    …
```

Test Case

Explore Sobel XDF



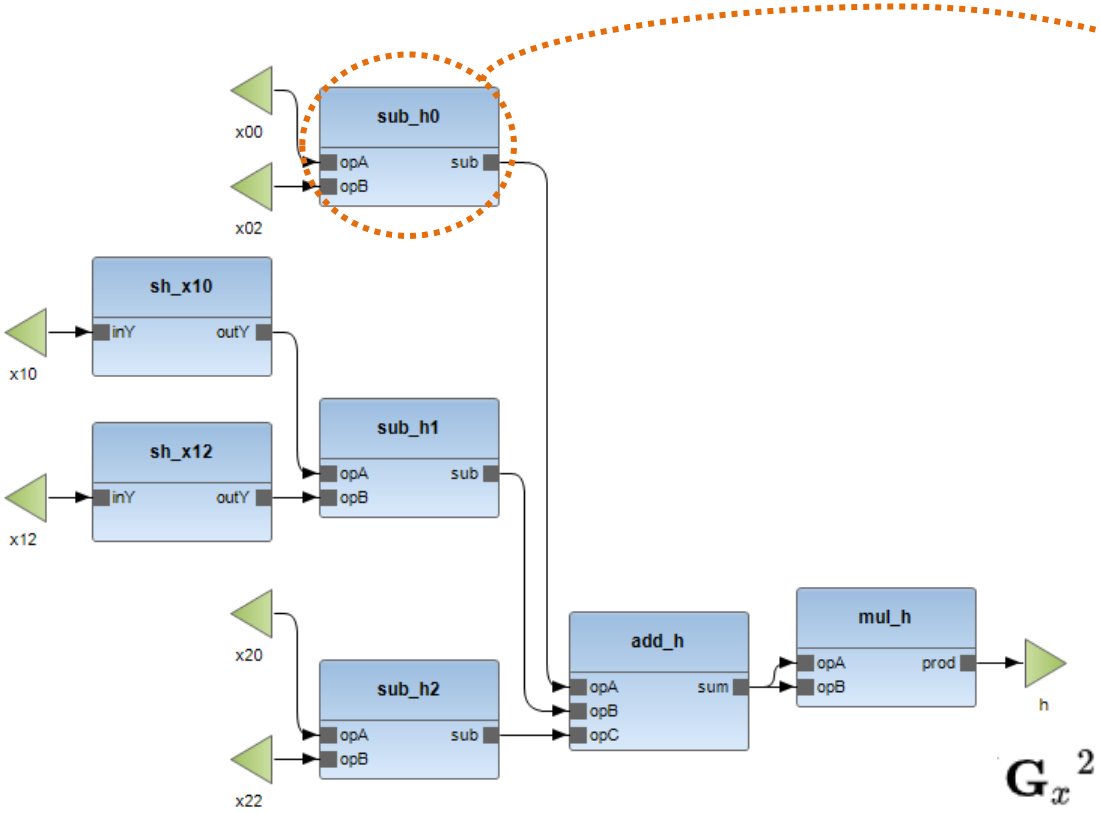
$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$$

$$\mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

double click on sobel_h and sobel_v **sub-networks** to **explore** them

Try It Yourself

Explore Sobel_Kernel_h XDF



$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$$

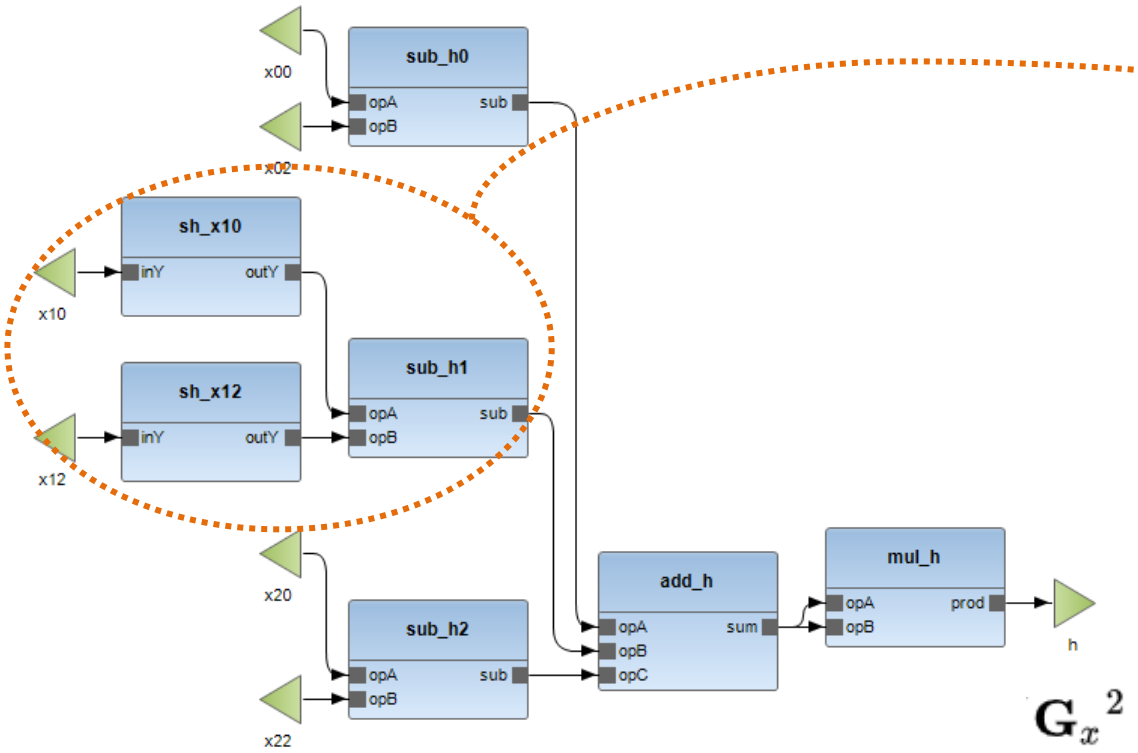
+1	0	-1			
+2	0	-2			
+1	0	-1			

00	01	02			…	
10	11	12			…	
20	21	22			…	
					…	
					…	
					…	
					…	
					…	
					…	
					…	

$\mathbf{G}_x{}^2$

Try It Yourself

Explore Sobel_Kernel_h XDF



$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$$

+1	0	-1			
+2	0	-2			
+1	0	-1			

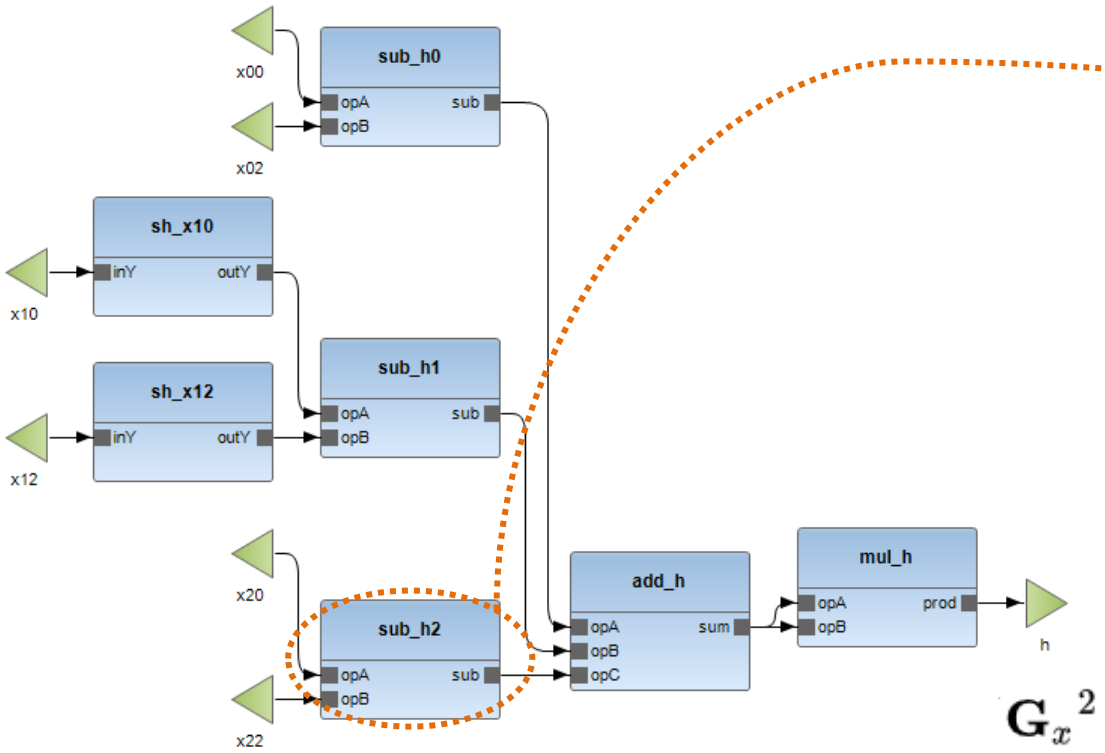
00	01	02			…	
10	11	12			…	
20	21	22			…	
					…	
					…	
					…	
					…	
					…	
					…	
					…	

LeftShifter actors apply **left shift** by **one position** of the input data, corresponding to **multiplying** it **by 2**

$$x<<1 = x*(2\hat{}1) = x*2$$

Try It Yourself

Explore Sobel_Kernel_h XDF



$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$$

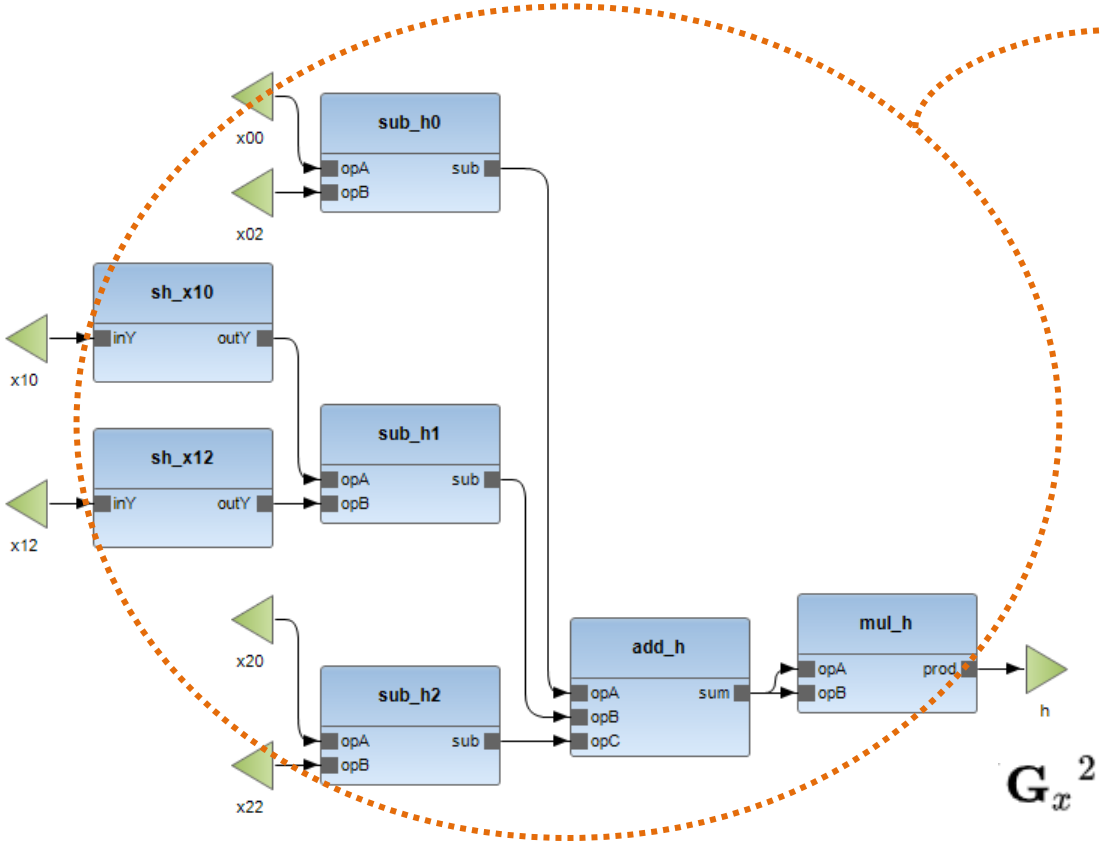
+1	0	-1			
+2	0	-2			
+1	0	-1			

00	01	02			…	
10	11	12			…	
20	21	22			…	
					…	
					…	
					…	
					…	
					…	
					…	
					…	

$\mathbf{G}_x{}^2$

Try It Yourself

Explore Sobel_Kernel_h XDF



$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$$

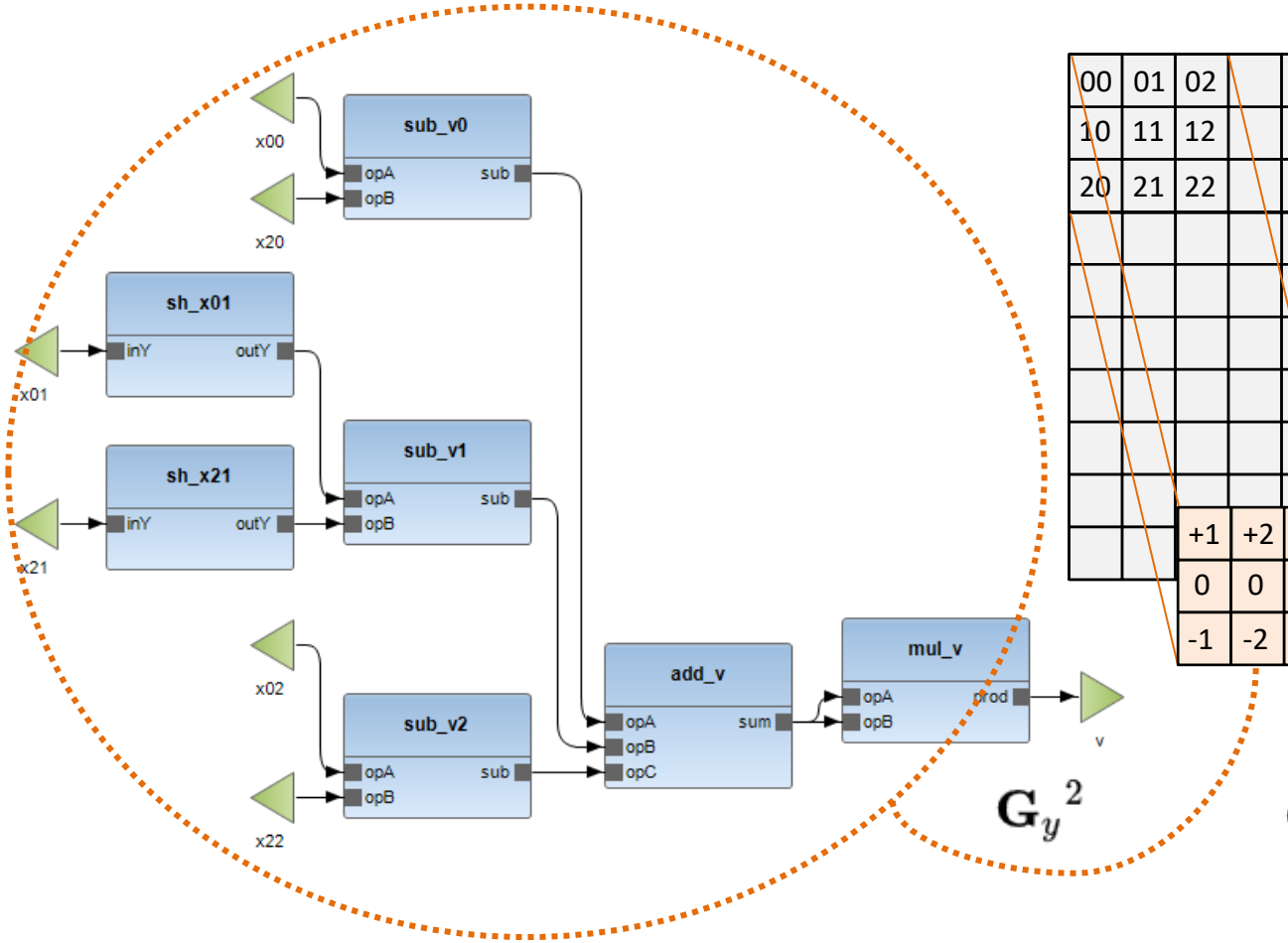
+1	0	-1			
+2	0	-2			
+1	0	-1			

00	01	02			…	
10	11	12			…	
20	21	22			…	
					…	
					…	
					…	
					…	
					…	
					…	
					…	

Adder3x1 and **Multiplier** actors complete the **squared gradient computation**

Try It Yourself

Explore Sobel_Kernel_v XDF



00	01	02			…
10	11	12			…
20	21	22			…
					…
					…
					…
					…
					…
					…
					…

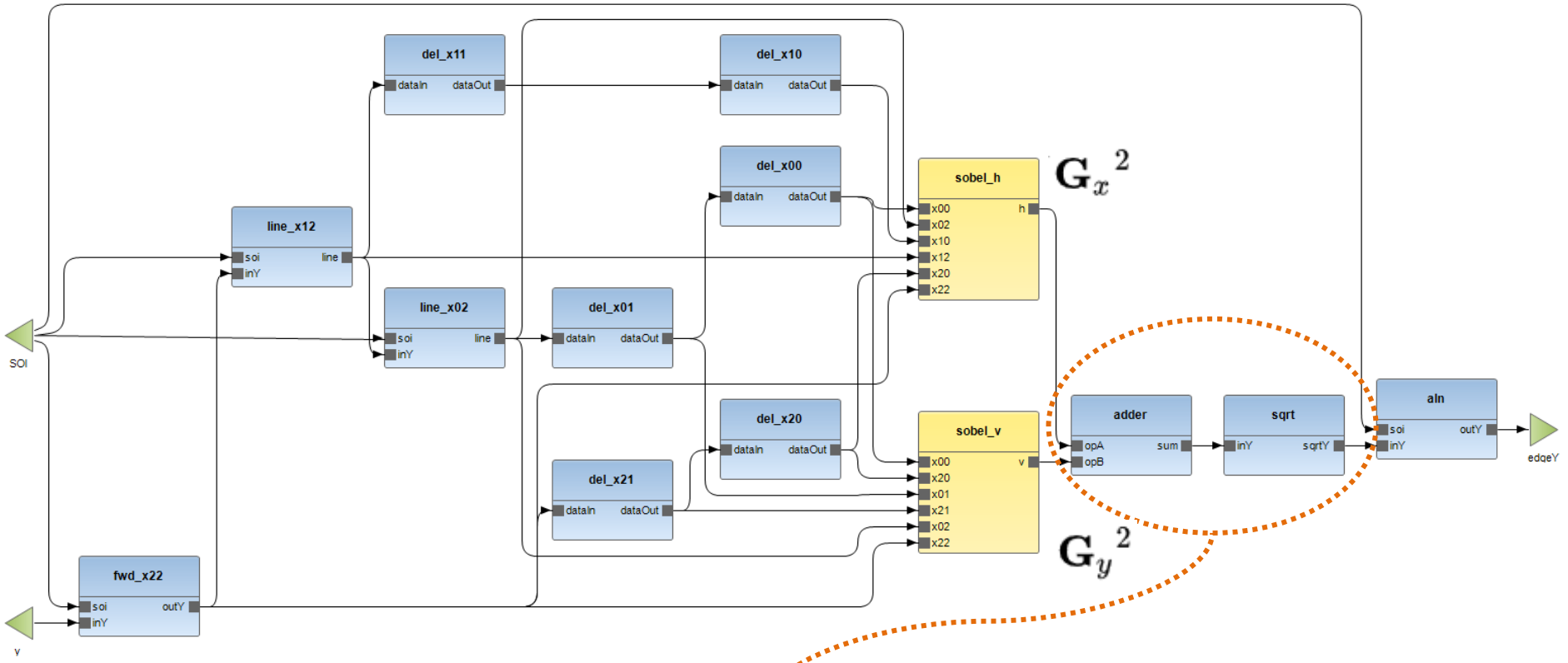
+1	+2	+1
0	0	0
-1	-2	-1

$$\mathbf{G}_y{}^2$$

$$\mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Try It Yourself

Explore Sobel XDF



$$\mathbf{G} = \sqrt{\mathbf{G}_x^{\,2} + \mathbf{G}_y^{\,2}}$$

Test Case

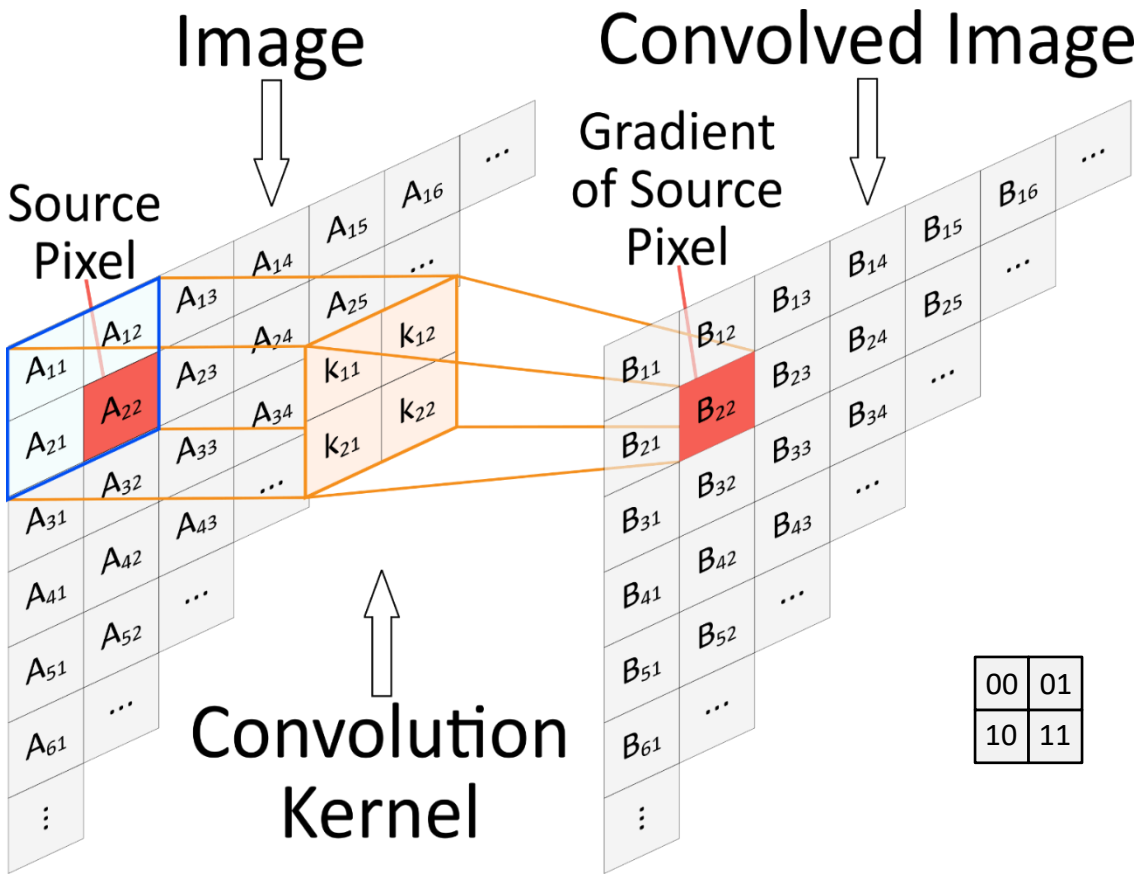
Edge Detection

Roberts Operator

$$\mathbf{G} = \sqrt{\mathbf{G}_x{}^2 + \mathbf{G}_y{}^2}$$

$$G_x = \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix}$$

$$G_y = \begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix}$$



Try It Yourself

Derive Roberts from Sobel

1. Duplicate Sobel.xdf, Sobel_Kernel_h.xdf and Sobel_Kernel_v.xdf dataflows
2. Remove all unnecessary actor instances
3. Rename actor instances to respect name conventions (x00, x01, x10 and x11 image pixels are needed)
4. Disconnect wrong connections
5. Connect unconnected actor instances
6. Change entity of actor instances (fwd and aln from 3x3 to 2x2 kernel actor)

Try It Yourself

Derive Roberts from Sobel

1. File Duplication
    1. right click on Sobel.xdf → Copy
    2. right click on package baseline → Paste
    3. do the same for Sobel_Kernel_h.xdf and Sobel_Kernel_v.xdf
2. Remove an actor instance
    1. click on actor instance on the xdf graphical editor
    2. move the cursor on the top right corner
    3. click on the trash bin

    (or right click on actor instance → Delete)

Try It Yourself

Derive Roberts from Sobel

3. Rename an actor/network instance
   1. slow double click on actor/network instance
   2. type a new name

   (or right click on actor/network instance → Rename)
4. Disconnect wrong connections
   1. right click on connection → Delete

Try It Yourself

Derive Roberts from Sobel

5. Connect unconnected actor instances
    1. click on Connections under the Connection folder on the Palette tab (right side of the screen)
    2. click on the desired output port of the source actor instance
    3. drag a line to the desired input port of the destination actor instance

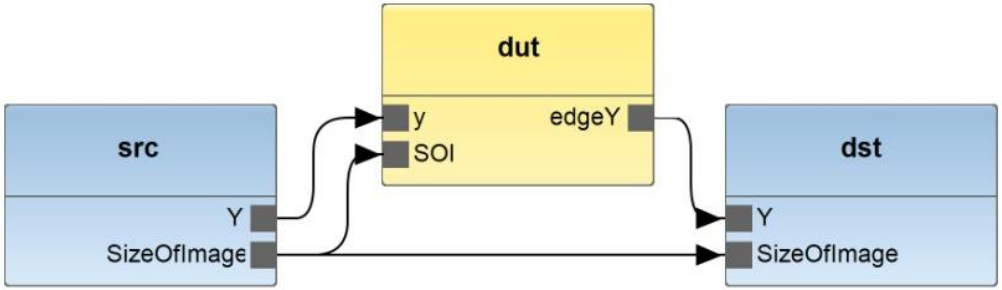
Try It Yourself

Derive Roberts from Sobel

6. Change entity of actor/network instances
   1. right click on actor instance → Set/Update refinement…
   2. select the new entity (cal file for actors, xdf file for sub-networks) from the list of available ones

Dataflow Simulation

Testbench



- Possibility of simulating dataflow models before going ahead with their processing
- Testbench network should not present inputs or outputs, but source and destination actors emulate the surrounding environment

Dataflow Simulation

SourceImage Actor

```
sendY: action ==> Y:[val]
guard open
var uint(size=8) val
do
   val := source_readByte();
   if x < WIDTH-1 then
      x := x+1;
   else
      if y < HEIGHT-1 then
         y := y+1;
      else
         y := 0;
         open := false;
      end
      x := 0;
   end
end
```

- Read Y component of source image from file byte by byte
- Fixed image size to 300 pixel widht and 255 pixel height
- Infinitely read the image: after one complete read it starts again reading the image from the beginning

Dataflow Simulation

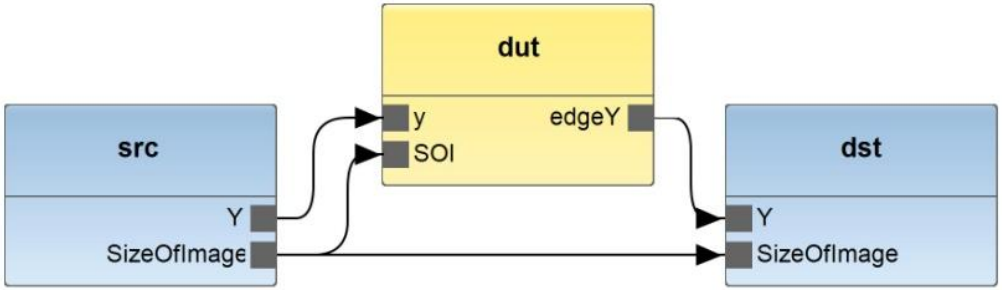
ShowImage Actor

```
recvY: action Y:[val] ==>
guard init and open
do
   pictBuffY[x+y*width] := val;
   if x < width-1 then
      x := x+1;
   else
      if y < height-1 then
         y := y+1;
      else
         y := 0;
         open := false;
         displayYUV_displayPicture(pictBuffY, pictBuffU, pictBuffV,width, height);
         fpsPrintNewPicDecoded();
      end
      x := 0;
   end
end
```

- Display image on a window
- Print the time required for computation on the system console

Dataflow Simulation

Device Under Test (DUT)



- DUT is a network instance to be simulated
- The Testbench.xdf file provides a ready-to-use testbench with the Sobel.xdf network as DUT
- It is possible to change the DUT in the same way we changed instance entity from one actor to another

Try It Yourself

Simulate Sobel

1. Click on Run → Run Configurations… on the main menu
2. Double click on Orcc Simulation to create a new run configuration
3. Choose a Name for the new run configuration (e.g. Tutorial Simulation 1)
4. Select Tutorials as project referenced by the new run configuration

Try It Yourself

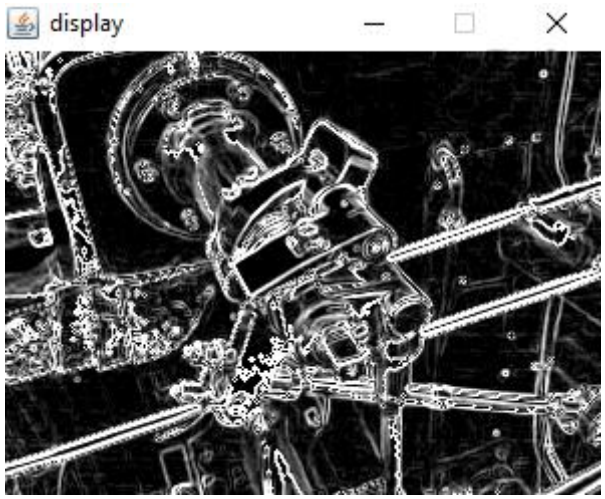
Simulate Sobel

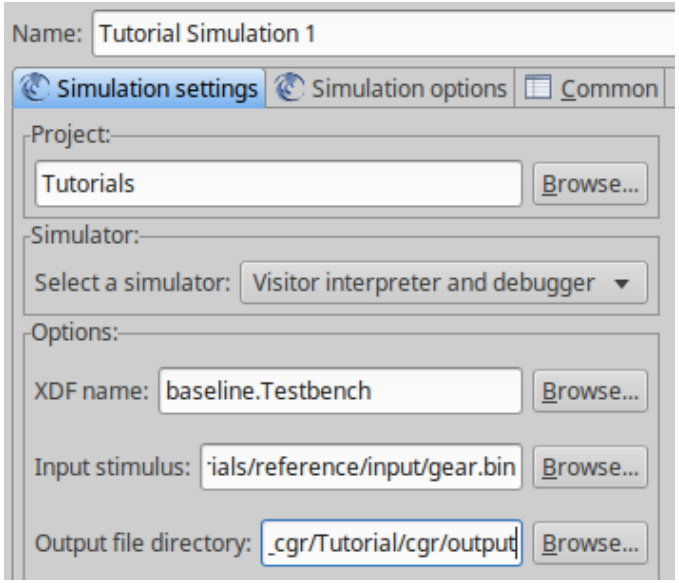
5. Choose Visitor interpreter and debugger as simulator
6. Choose Testbench as xdf network to be simulated
7. Choose gear.bin as input stimuli for the simulation (this is the file subjected to edge detection)
   /home/monitoring_cgr/Tutorial/cgr/workspace/Tutorials/reference/input/gear.bin

Try It Yourself

Simulate Sobel

8. Select an output file directory (no output will be generated in this case)
    8. /home/monitoring_cgr/Tutorial/cgr/output
9. Click Run to launch the simulation





Try It Yourself

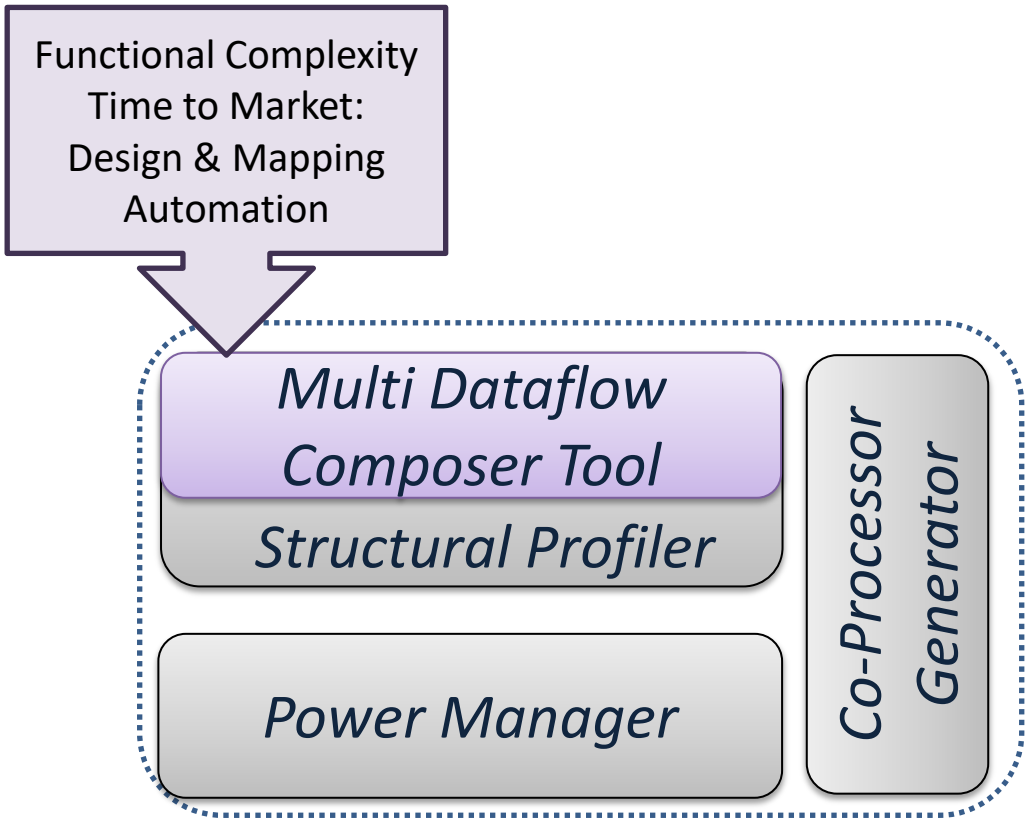
Simulate Roberts

1. Change the DUT from Sobel to Roberts and Save the network
2. Run again the simulation and see the difference with respect to the Sobel simulation in terms of:
    1. Detected edges
    2. Frame rate

Step 2: the Multi-Dataflow Composer tool

Baseline MDC Datapath Merging

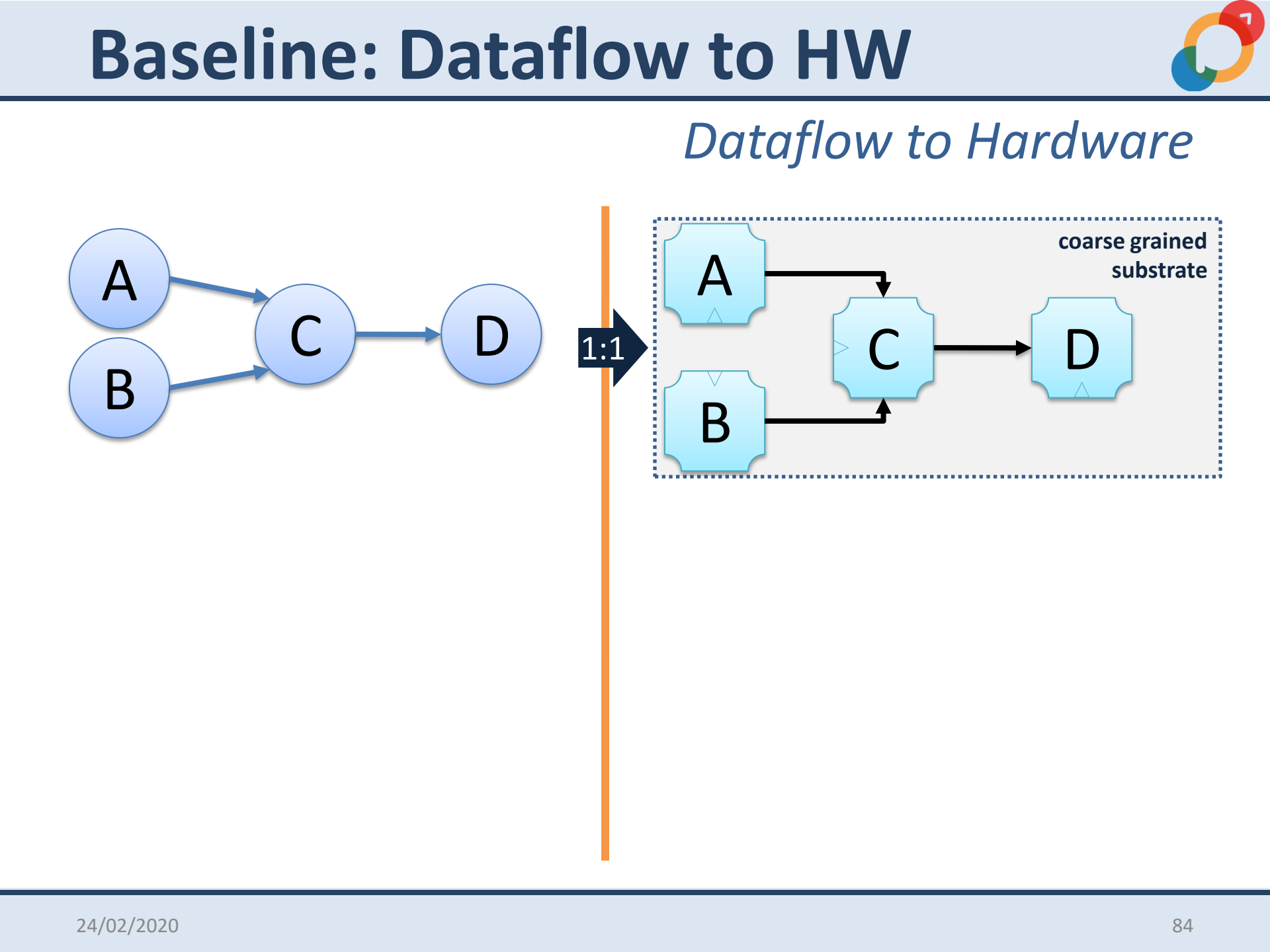
Baseline: Dataflow to HW



MDC design suite

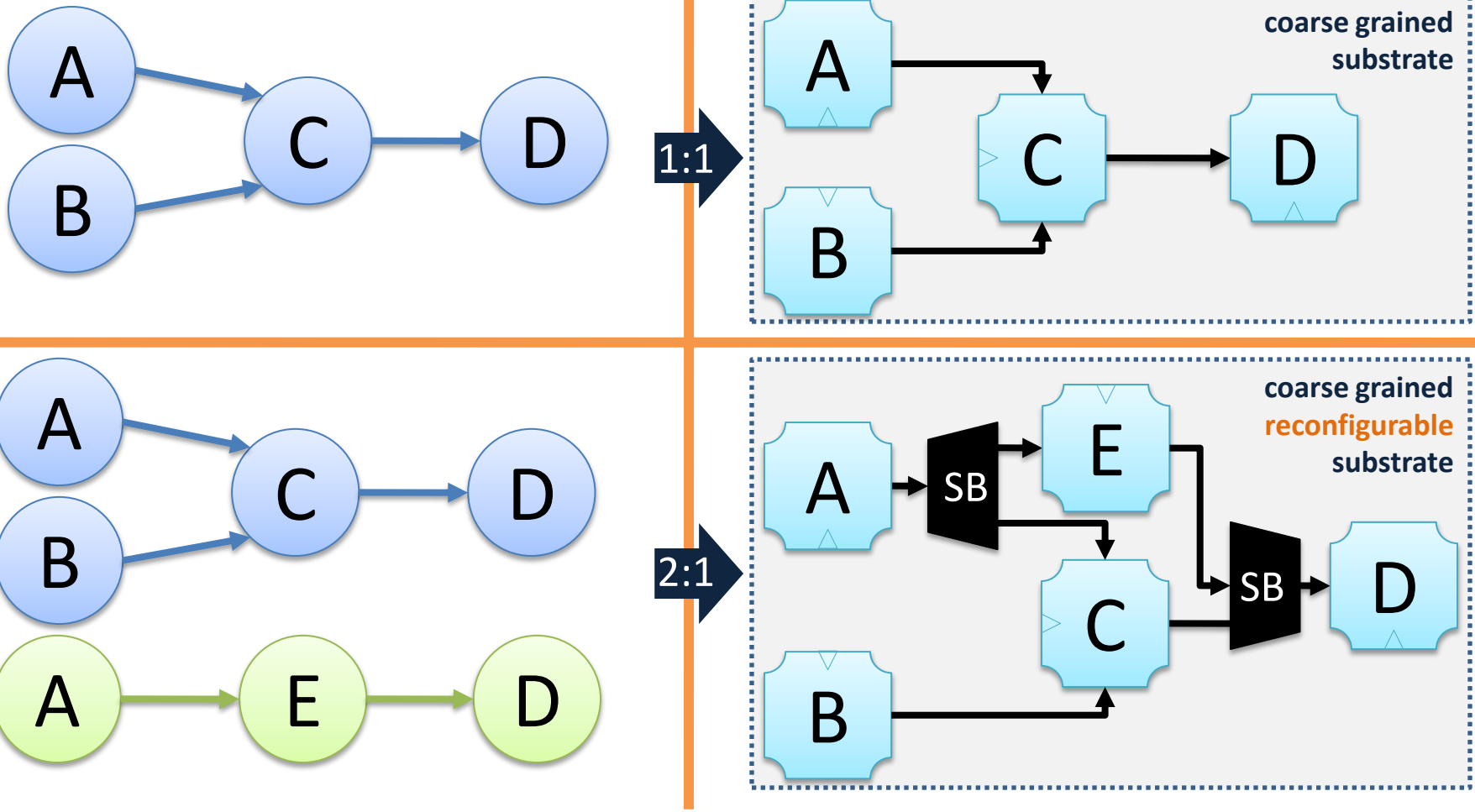
http://sites.unica.it/rpct/

Baseline: Dataflow to HW



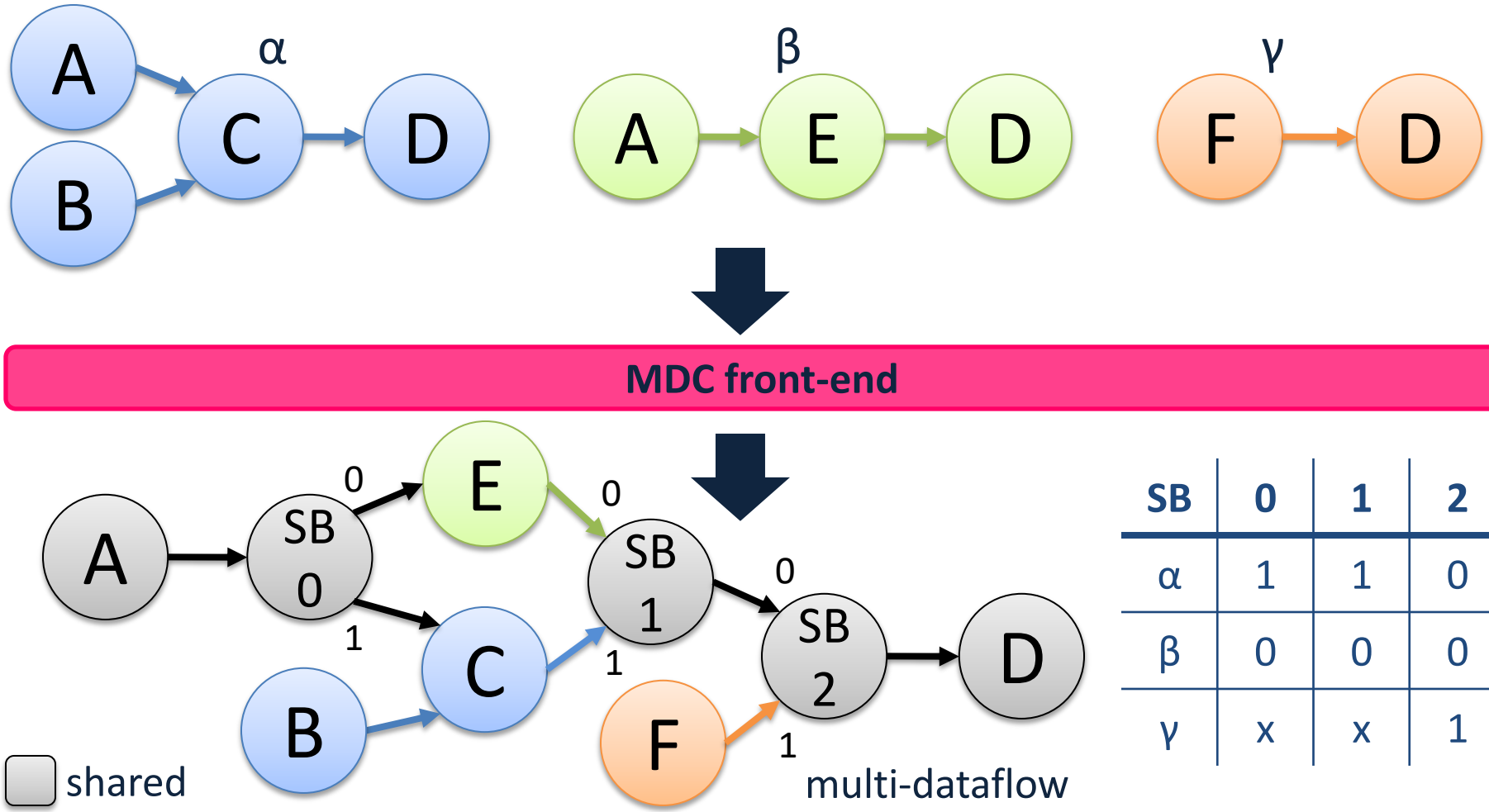
Baseline: Dataflow to HW

Dataflow to Hardware



MDC Front-End

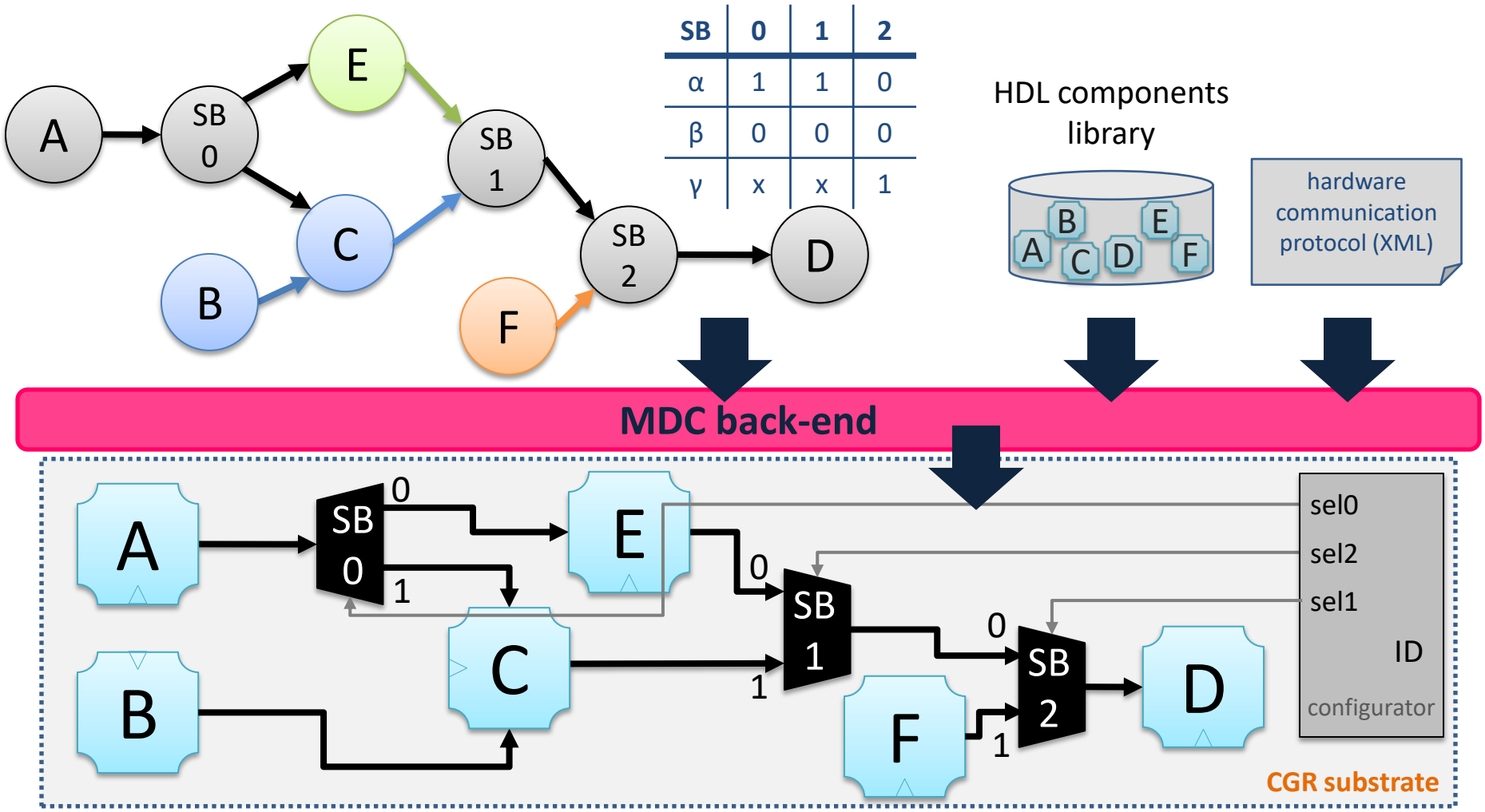
Multi-Dataflow Generation



SB	0	1	2
α	1	1	0
β	0	0	0
γ	x	x	1

MDC Back-End

Platform Composer



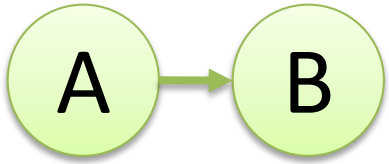
Step 2: the Multi-Dataflow Composer tool

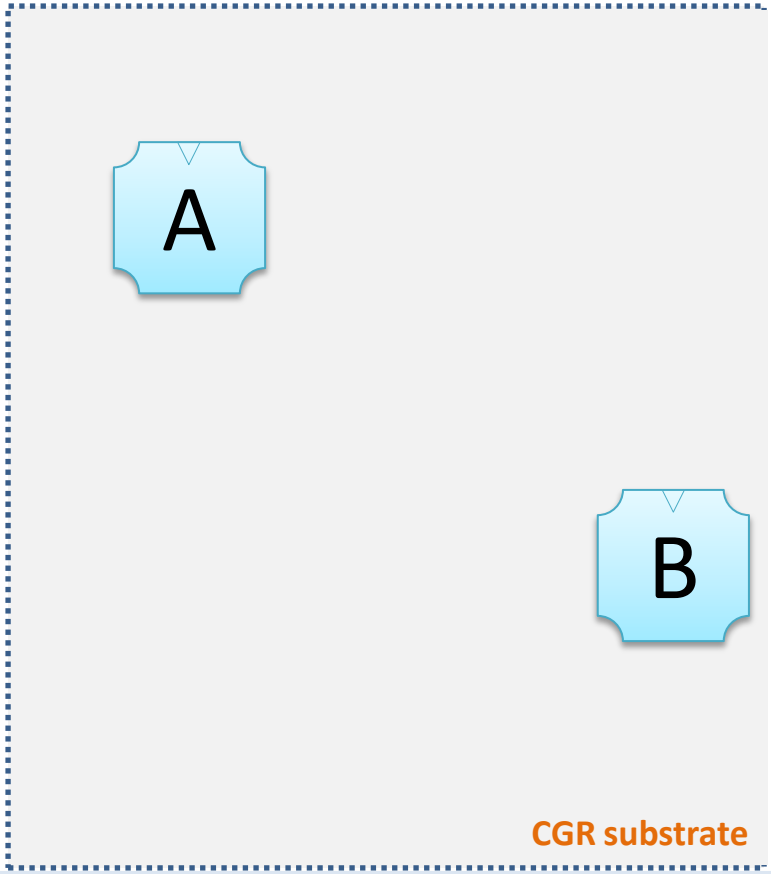
High Level Synthesis (HLS) support

HLS Support

```
<protocol>
  <sys_signals>
    <signal id="0" net_port="clock"  is_clock=""…></signal>
    …
  </sys_signals>
  <actor>
    <sys_signals>
      <signal id="0" port="clk" net_port="clock" …></signal>
      …
    </sys_signals>
    <comm_signals>
      <signal id="0" port="din" channel="data"…></signal>
      <signal id="1" port="dout" channel="data"…></signal>
      <signal id="2" port="wr" channel="en"…></signal>
      …
    <comm_signals>
  </actor>
  <predecessor>
    <sys_signals>…</sys_signals>
    <comm_signals>…<comm_signals>
  </predecessor>
  <successor>
    <sys_signals>…</sys_signals>
    <comm_signals>…<comm_signals>
  </successor>
</protocol>
```

Communication Protocol

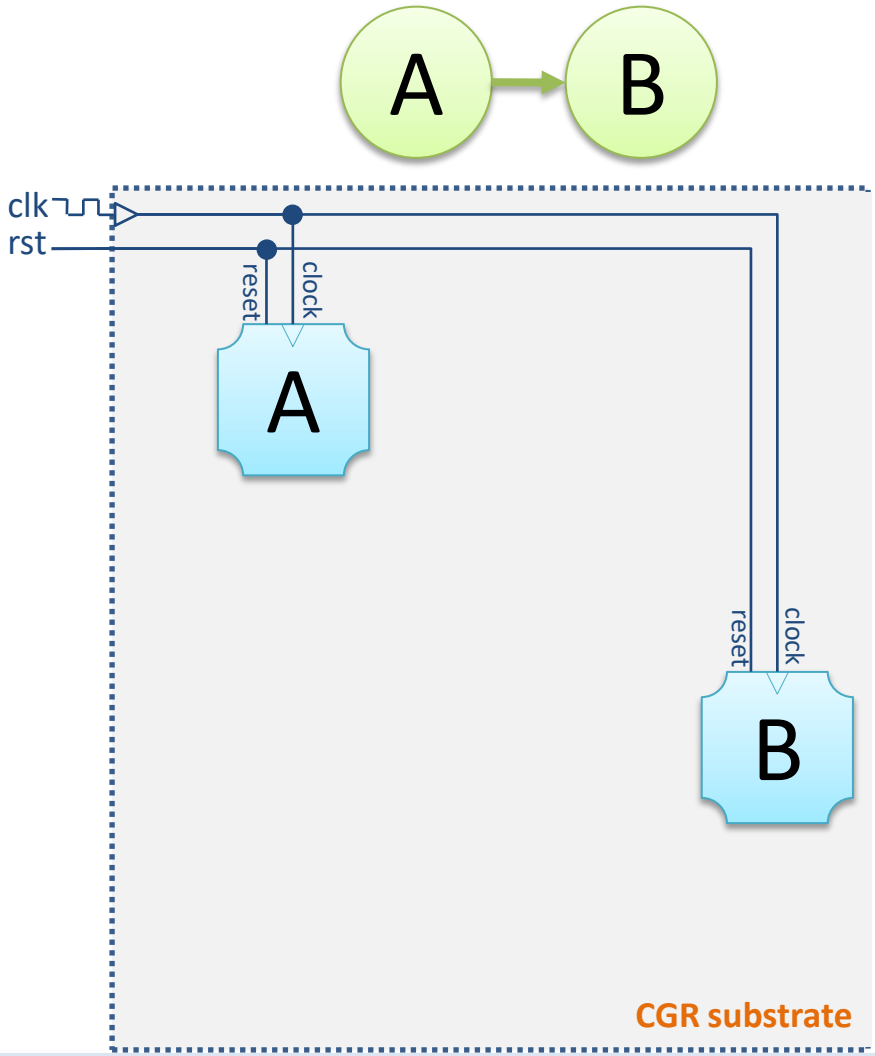




HLS Support

```
<protocol>
  <sys_signals>
    <signal id="0" net_port="clock"  is_clock="" ...></signal>
    ...
  </sys_signals>
  <actor>
    <sys_signals>
      <signal id="0" port="clk" net_port="clock" ...></signal>
      ...
    </sys_signals>
    <comm_signals>
      <signal id="0" port="din" channel="data"...></signal>
      <signal id="1" port="dout" channel="data"...></signal>
      <signal id="2" port="wr" channel="en"...></signal>
      ...
    <comm_signals>
  </actor>
  <predecessor>
    <sys_signals>...</sys_signals>
    <comm_signals>...<comm_signals>
  </predecessor>
  <successor>
    <sys_signals>...</sys_signals>
    <comm_signals>...<comm_signals>
  </successor>
</protocol>
```

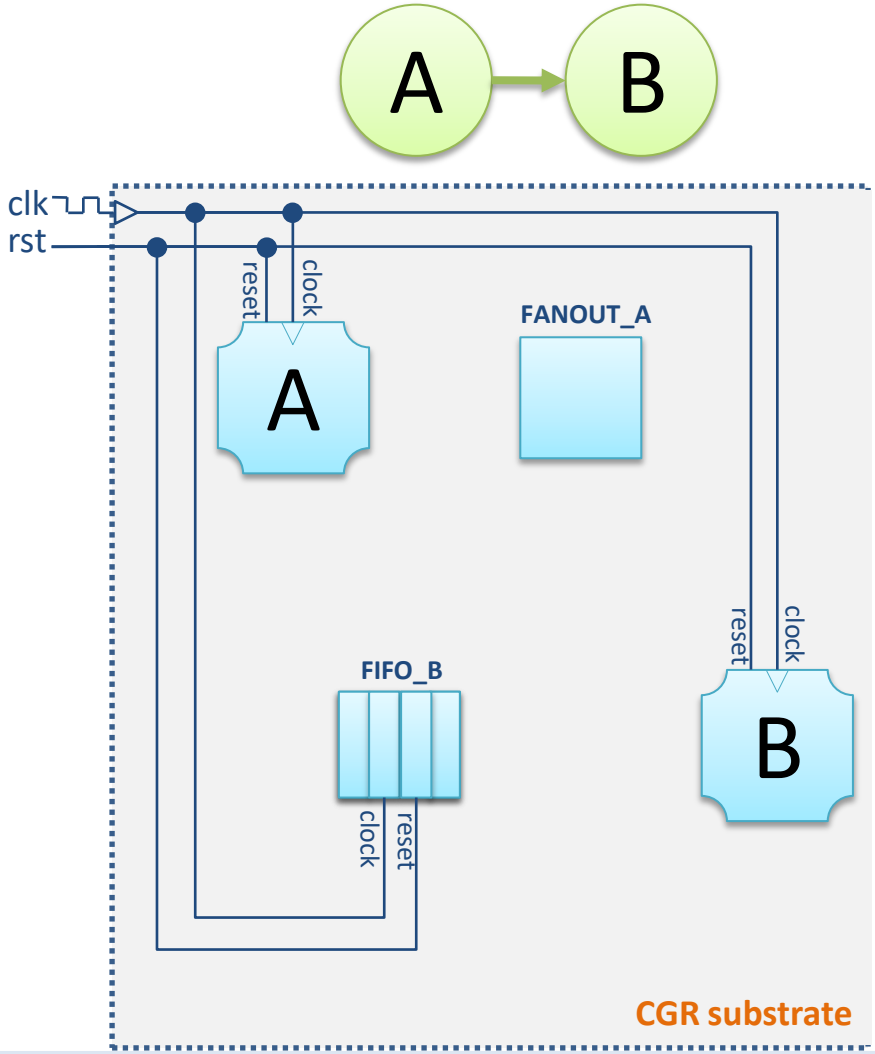
Communication Protocol



HLS Support

```
<protocol>
  <sys_signals>
    <signal id="0" net_port="clock"  is_clock=""…></signal>
    …
  </sys_signals>
  <actor>
    <sys_signals>
      <signal id="0" port="clk" net_port="clock" …></signal>
      …
    </sys_signals>
    <comm_signals>
      <signal id="0" port="din" channel="data"…></signal>
      <signal id="1" port="dout" channel="data"…></signal>
      <signal id="2" port="wr" channel="en"…></signal>
      …
    <comm_signals>
  </actor>
  <predecessor>
    <sys_signals>…</sys_signals>
    <comm_signals>…<comm_signals>
  </predecessor>
  <successor>
    <sys_signals>…</sys_signals>
    <comm_signals>…<comm_signals>
  </successor>
</protocol>
```

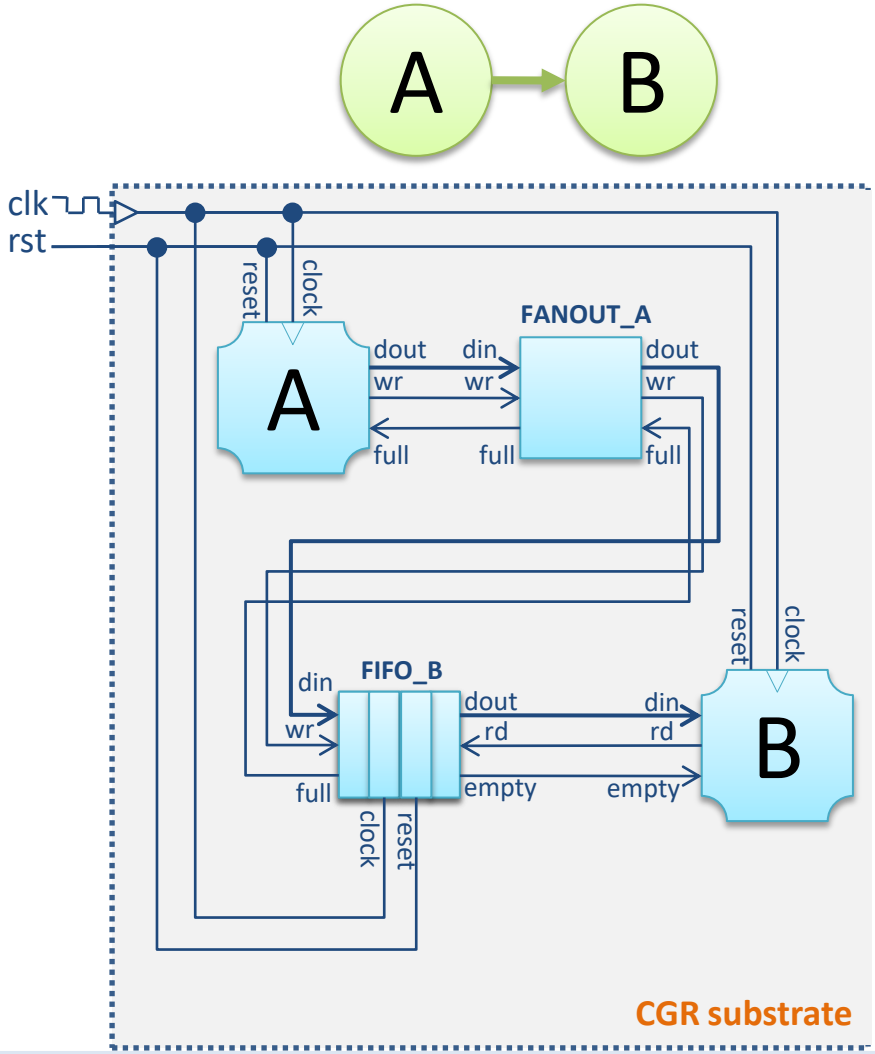
Communication Protocol



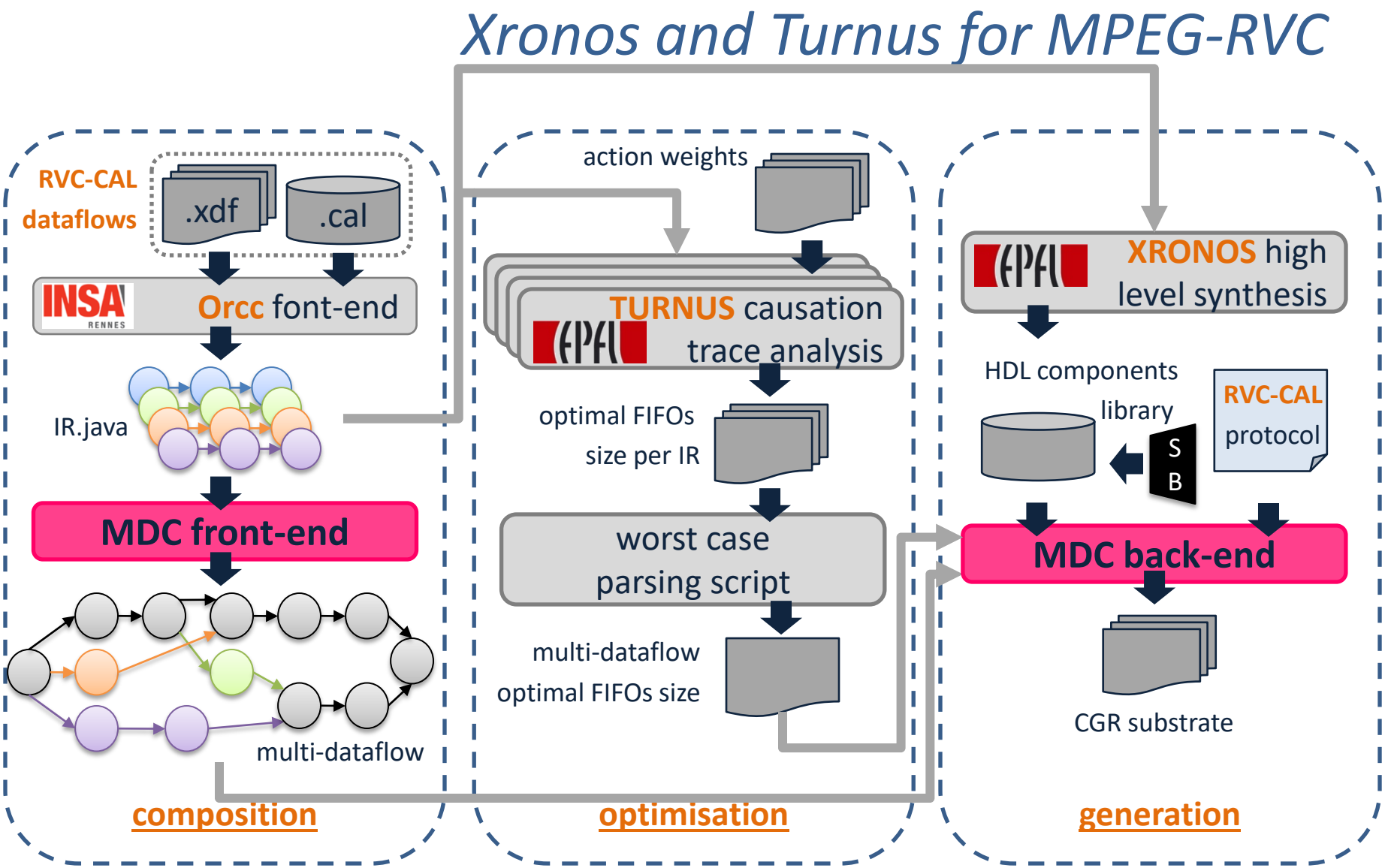
HLS Support

```
<protocol>
  <sys_signals>
    <signal id="0" net_port="clock"  is_clock=""…></signal>
    …
  </sys_signals>
  <actor>
    <sys_signals>
      <signal id="0" port="clk" net_port="clock" …></signal>
      …
    </sys_signals>
    <comm_signals>
      <signal id="0" port="din" channel="data"…></signal>
      <signal id="1" port="dout" channel="data"…></signal>
      <signal id="2" port="wr" channel="en"…></signal>
      …
    <comm_signals>
  </actor>
  <predecessor>
    <sys_signals>…</sys_signals>
    <comm_signals>…<comm_signals>
  </predecessor>
  <successor>
    <sys_signals>…</sys_signals>
    <comm_signals>…<comm_signals>
  </successor>
</protocol>
```

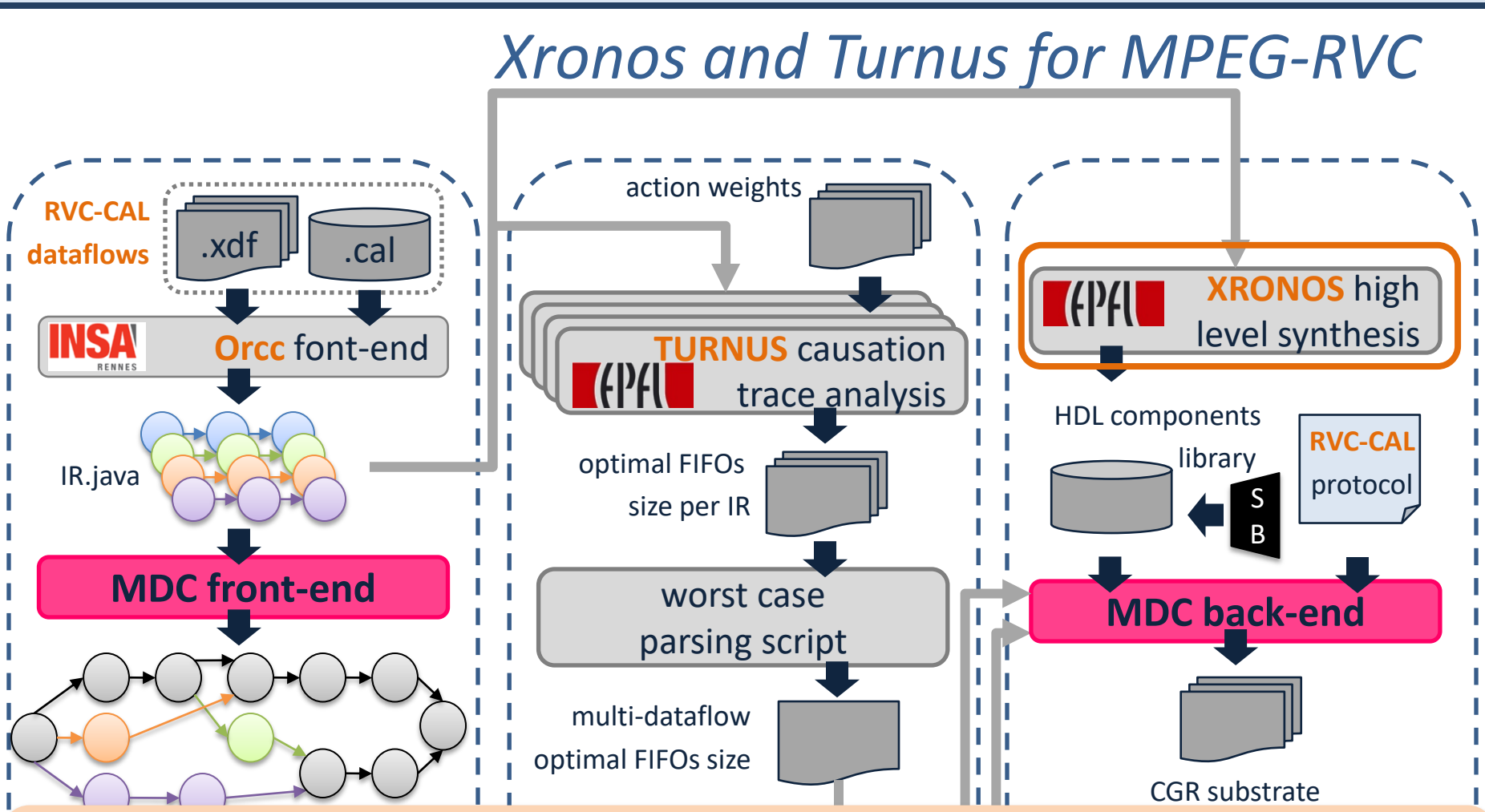
Communication Protocol



HLS Support

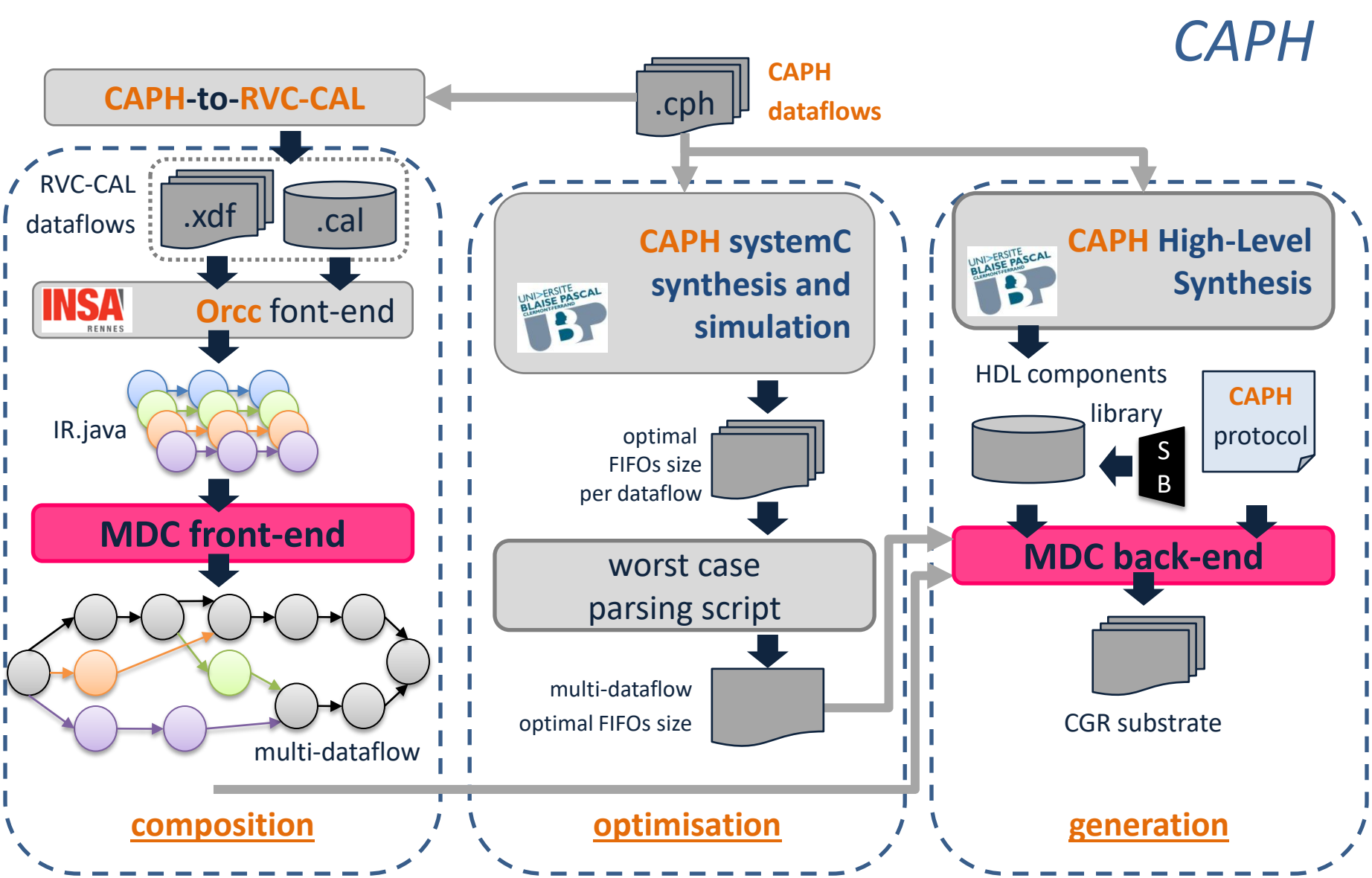

Xronos and Turnus for MPEG-RVC
RVC-CAL dataflows
.xdf
.cal
INSA RENNES
Orcc font-end
IR.java
MDC front-end
multi-dataflow
composition
action weights
TURNUS causation trace analysis
optimal FIFOs size per IR
worst case parsing script
multi-dataflow optimal FIFOs size
optimisation
XRONOS high level synthesis
HDL components library
S B
RVC-CAL protocol
MDC back-end
CGR substrate
generation

HLS Support

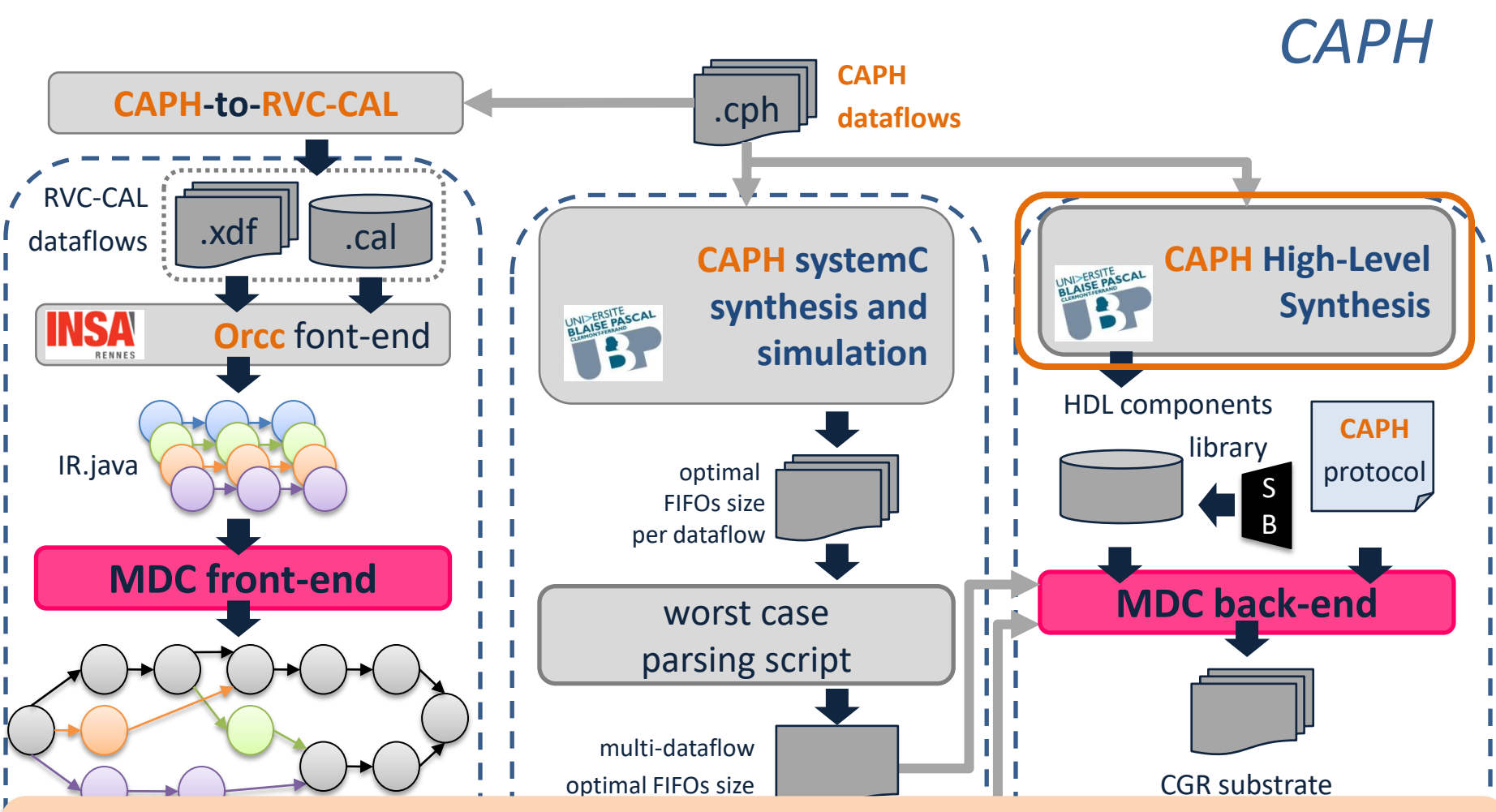


- High-Level Synthesis supports **only FPGAs from one specific FPGA vendor (Xilinx)**

HLS Support


CAPH
CAPH-to-RVC-CAL
.cph
CAPH dataflows
RVC-CAL dataflows
.xdf
.cal
INSA RENNES
Orcc font-end
IR.java
MDC front-end
multi-dataflow
CAPH systemC synthesis and simulation
optimal FIFOs size per dataflow
worst case parsing script
multi-dataflow optimal FIFOs size
CAPH High-Level Synthesis
HDL components library
S
B
CAPH protocol
MDC back-end
CGR substrate
composition
optimisation
generation

HLS Support



- **Platform Agnostic** High-Level Synthesis: it supports any kind of **FPGA** from **any vendor,** as well as **ASIC design flows**

PROSSIMO

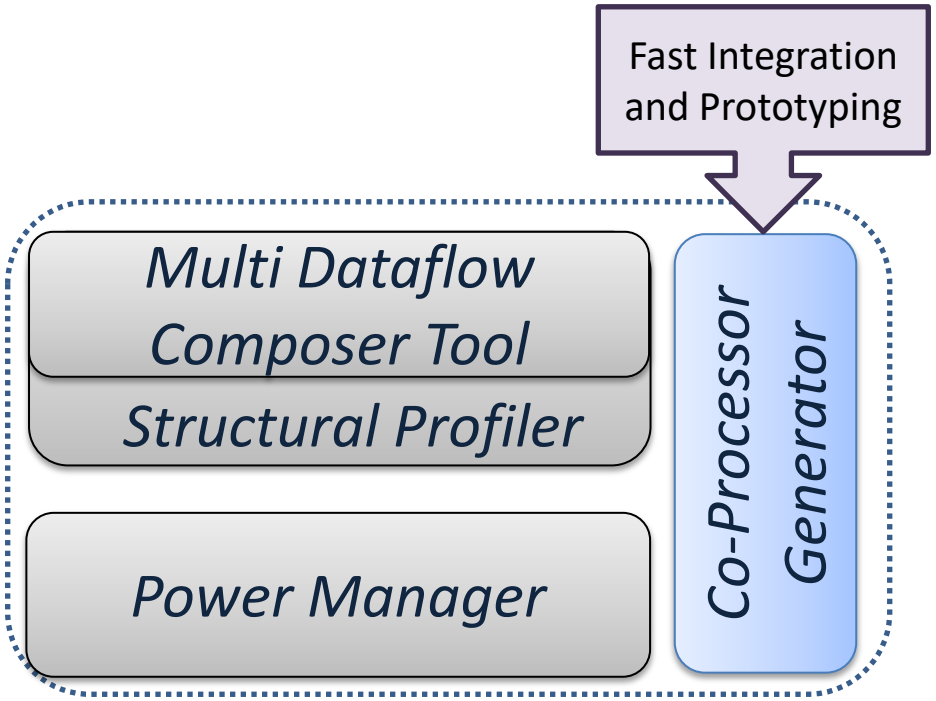
Progettazione, sviluppo e ottimizzazione di sistemi intelligenti multi-oggetto

Step 2: the Multi-Dataflow Composer tool

Coprocessor Generator

Co-processor Generator

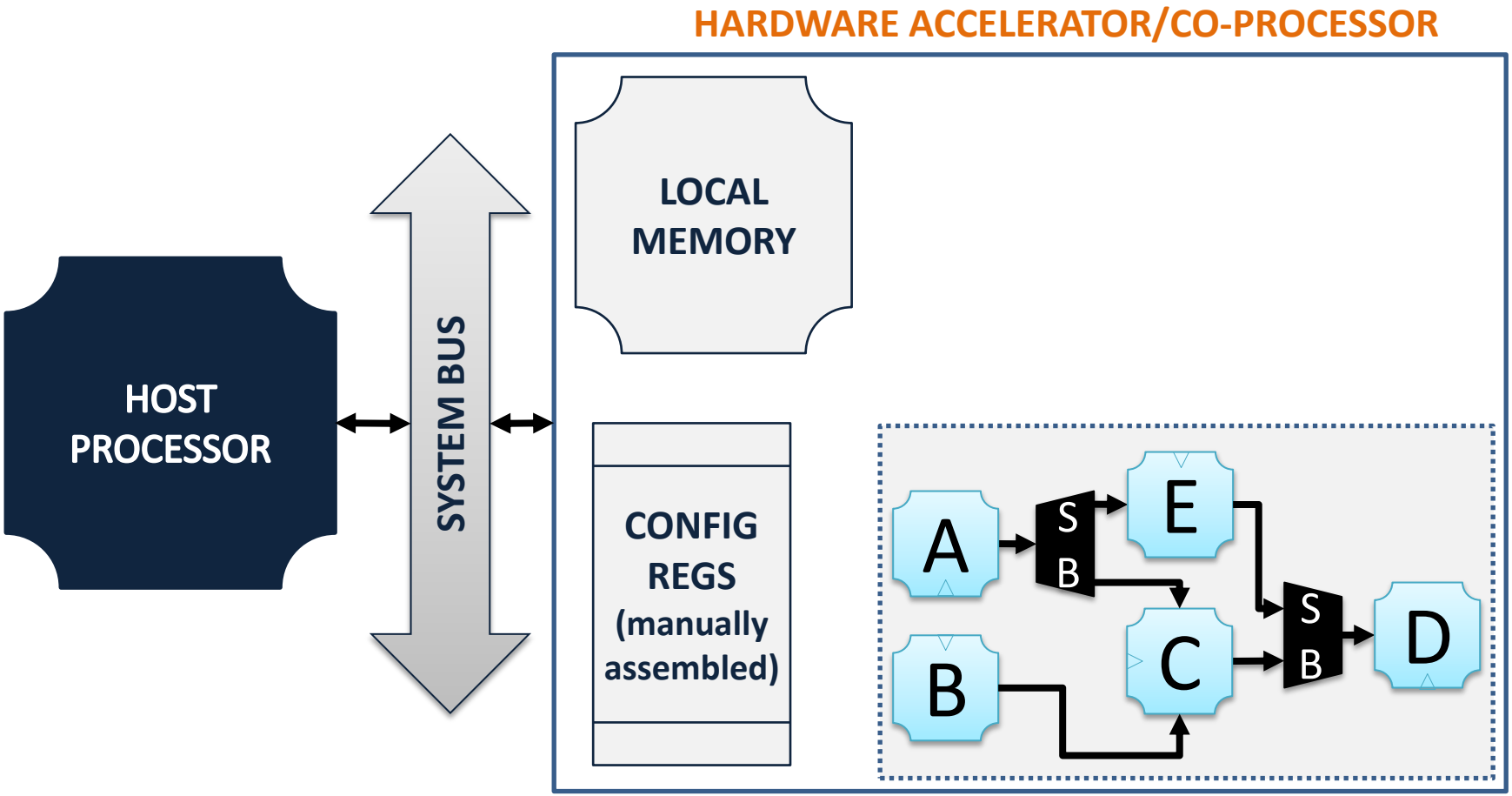
Ready to use Xilinx IPs



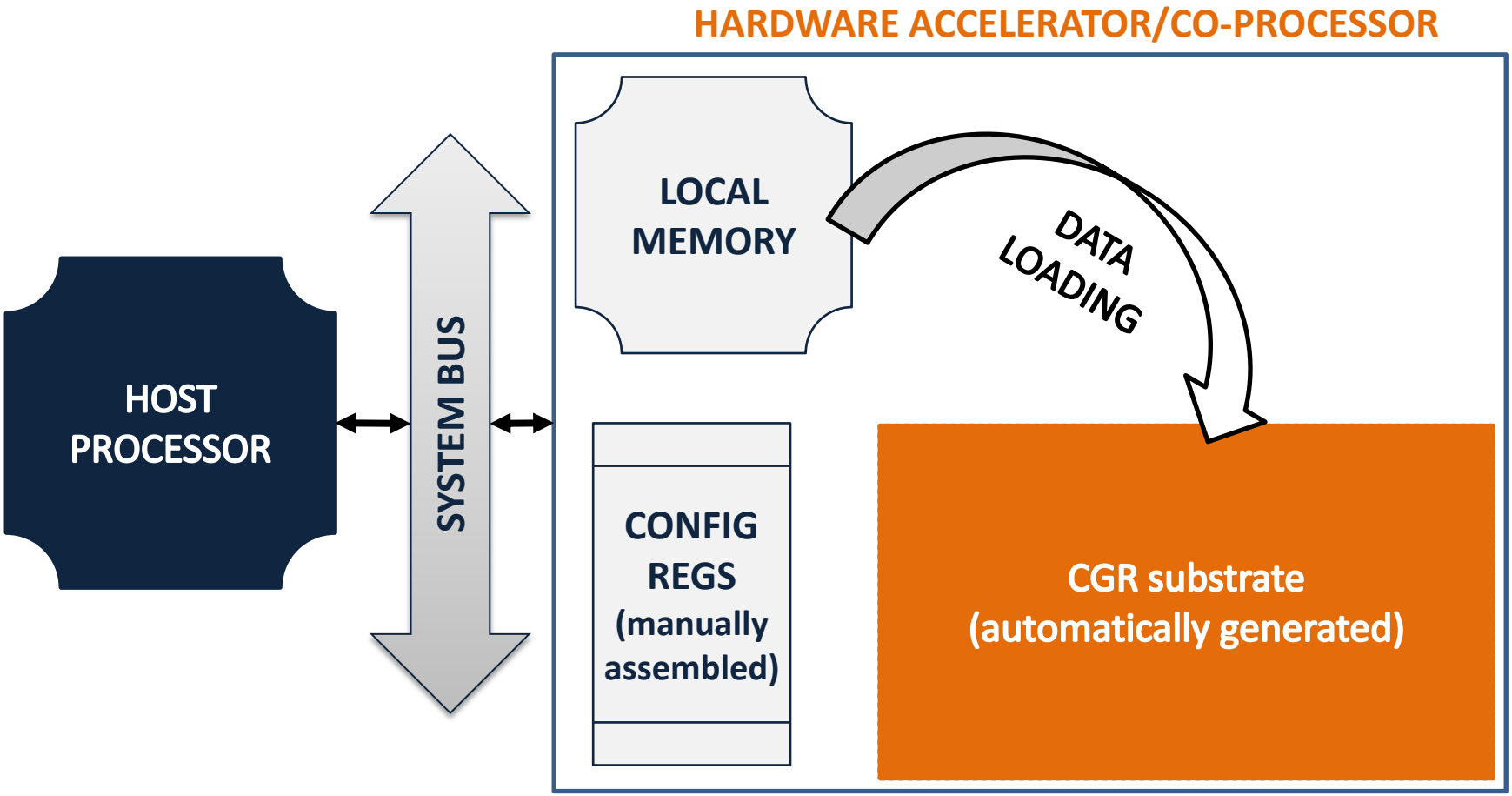
MDC design suite

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

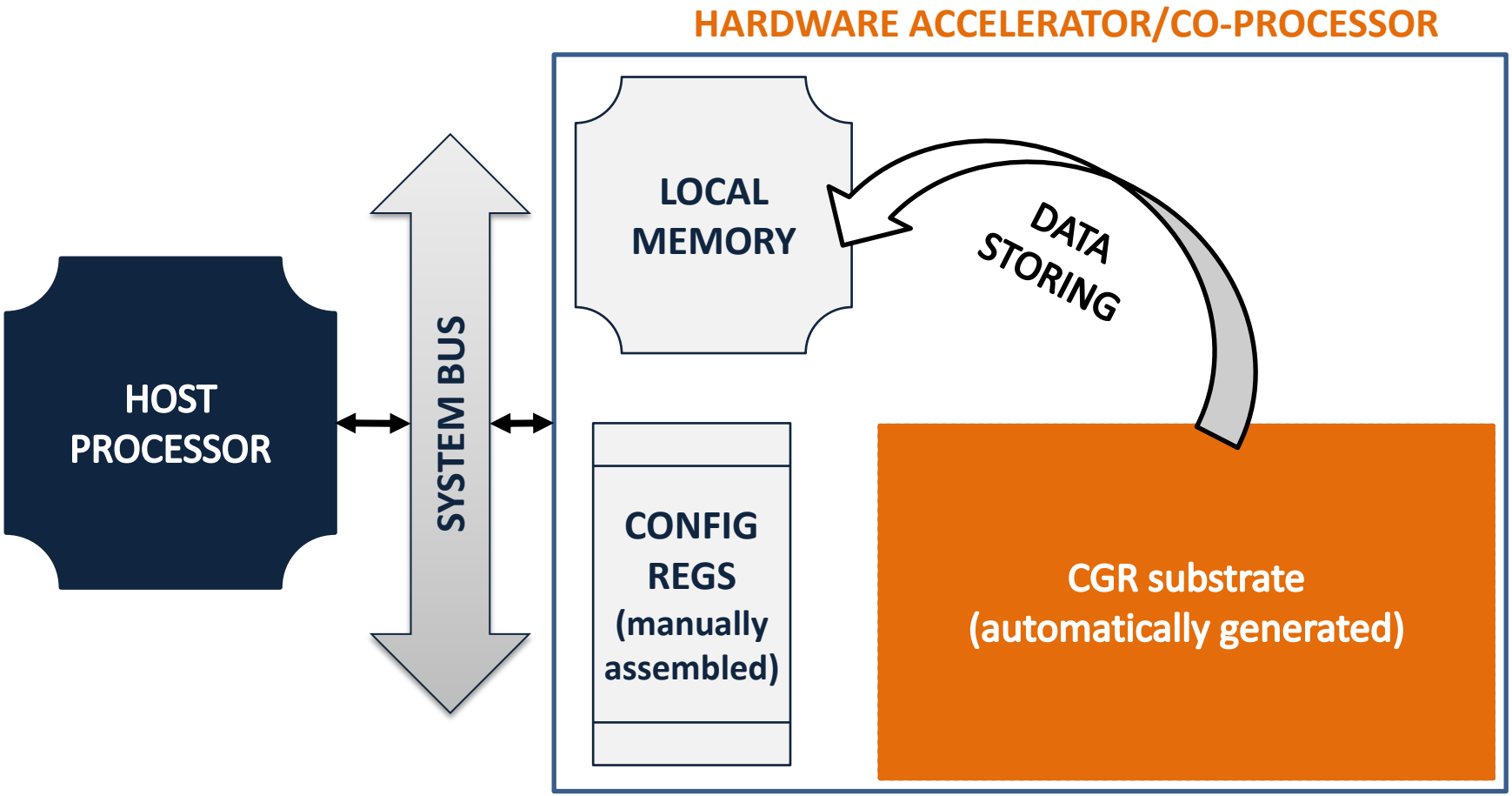
Co-Processor Generator



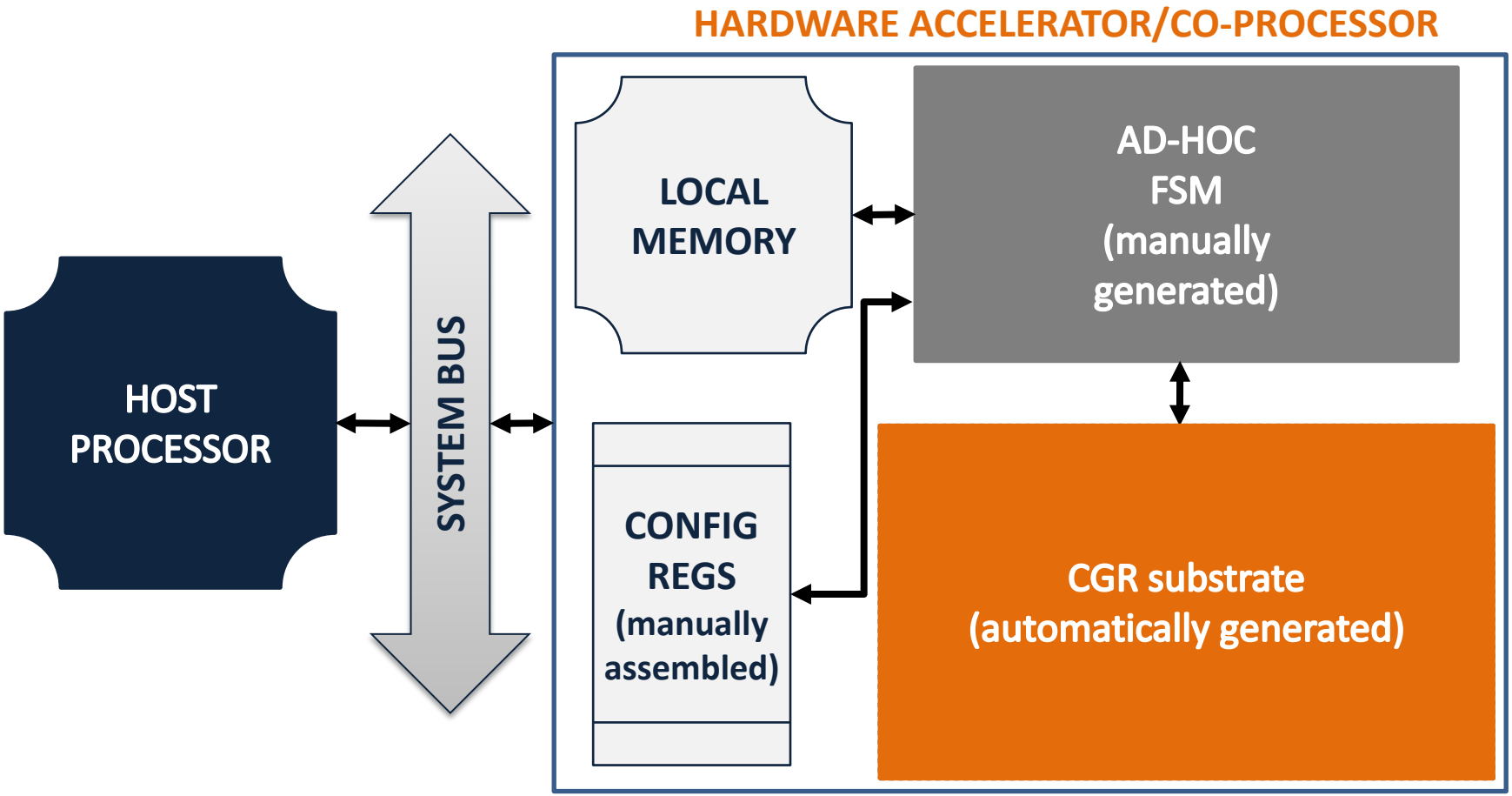
Co-Processor Generator



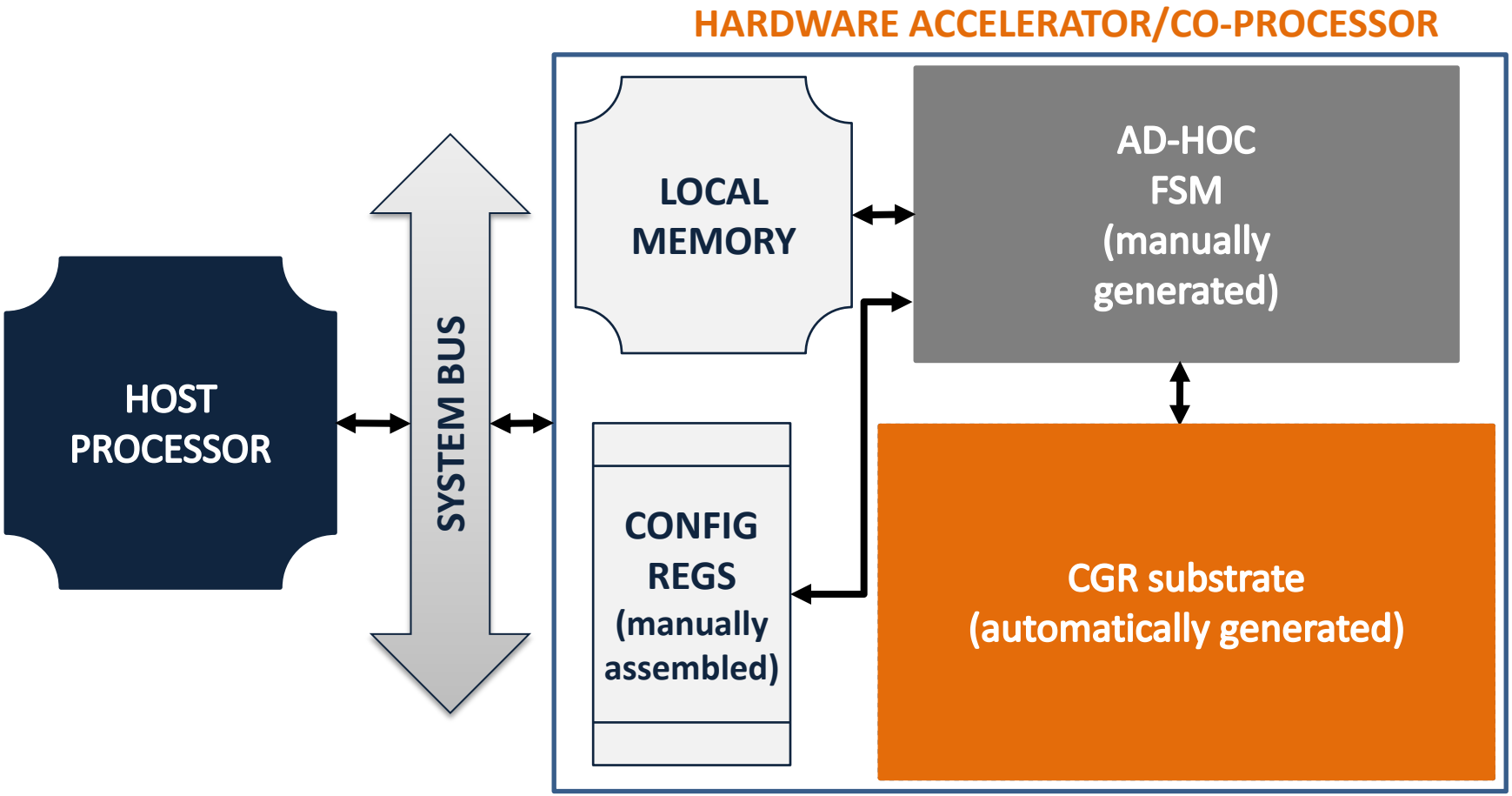
Co-Processor Generator



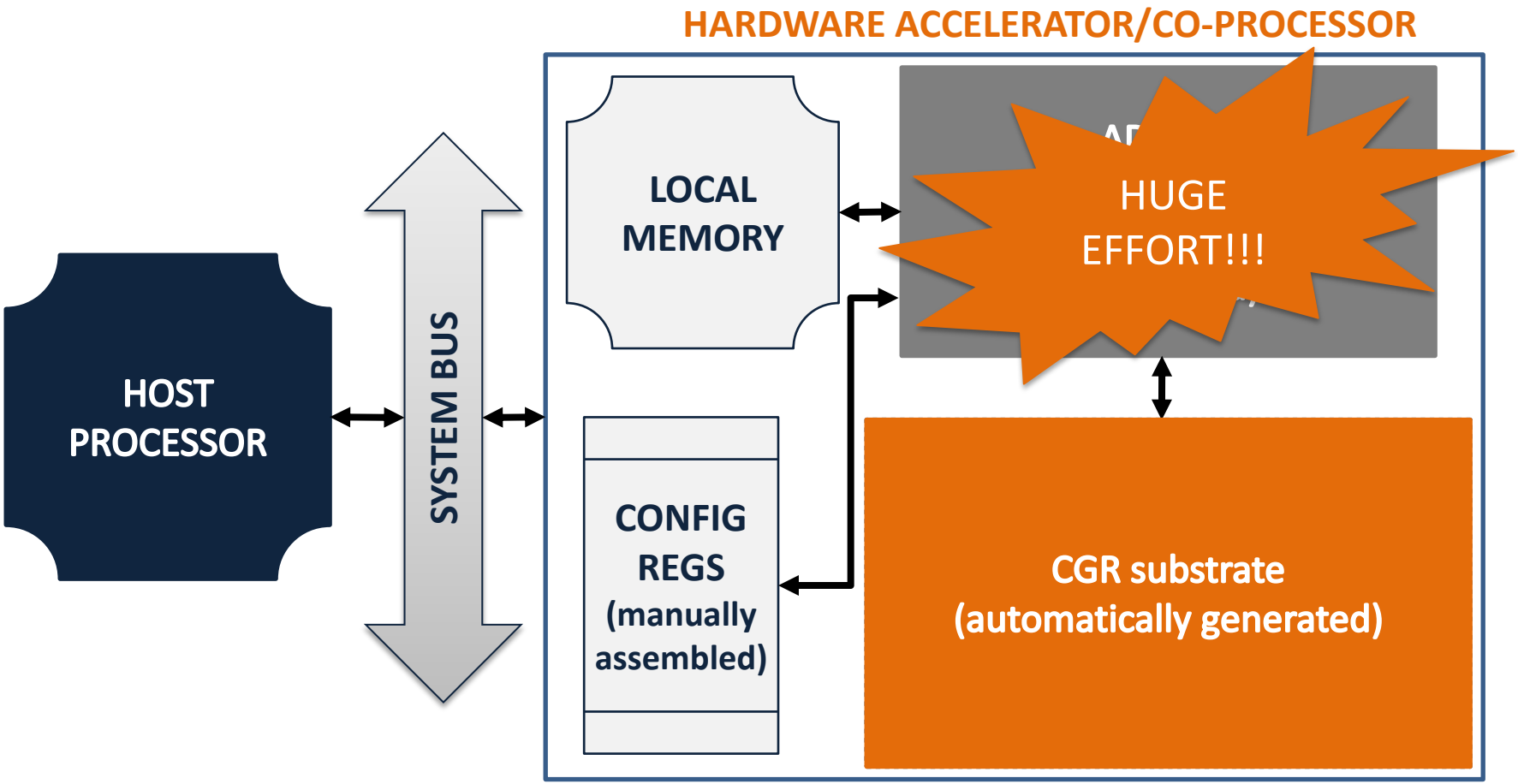
Co-Processor Generator



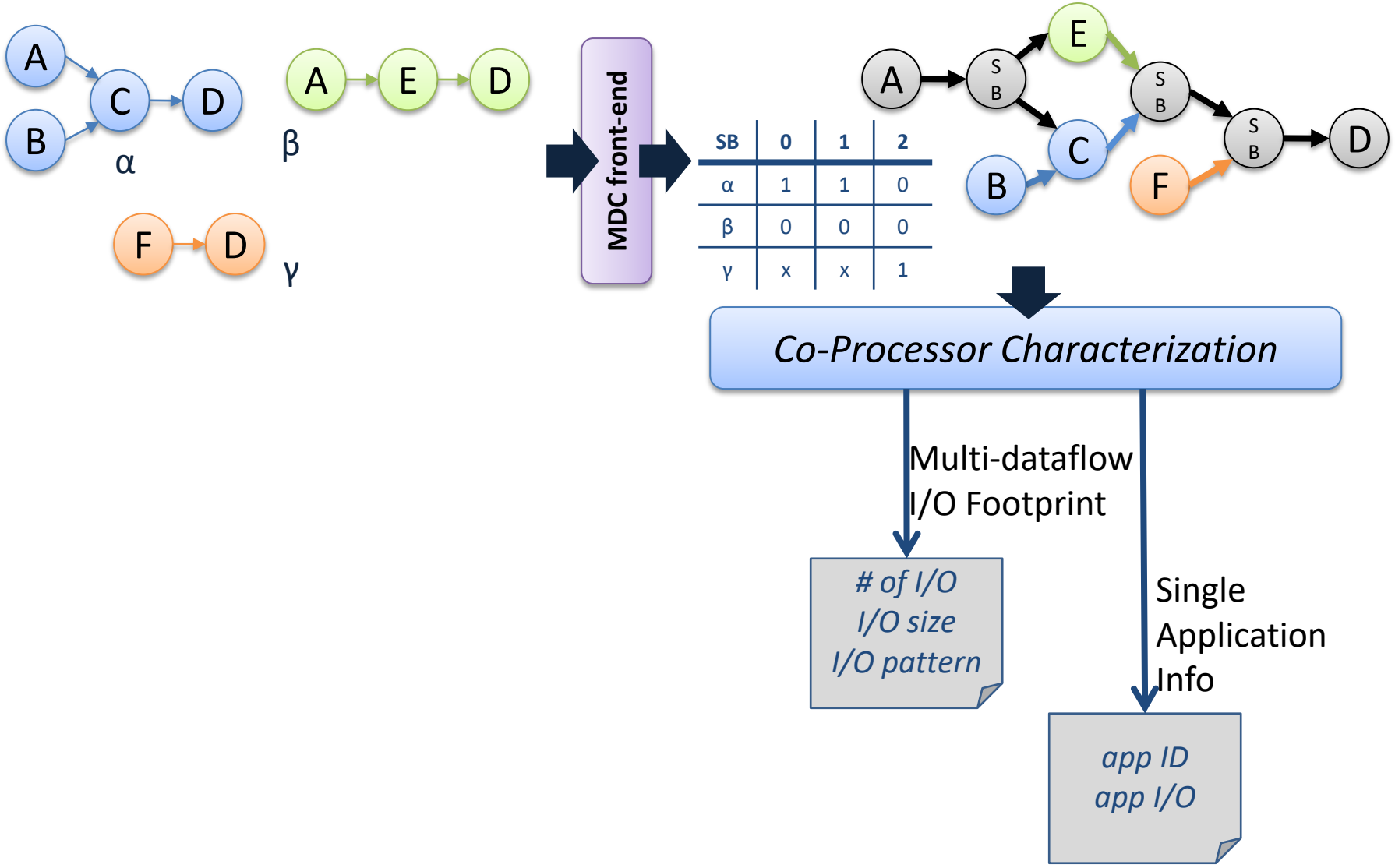
Co-Processor Generator



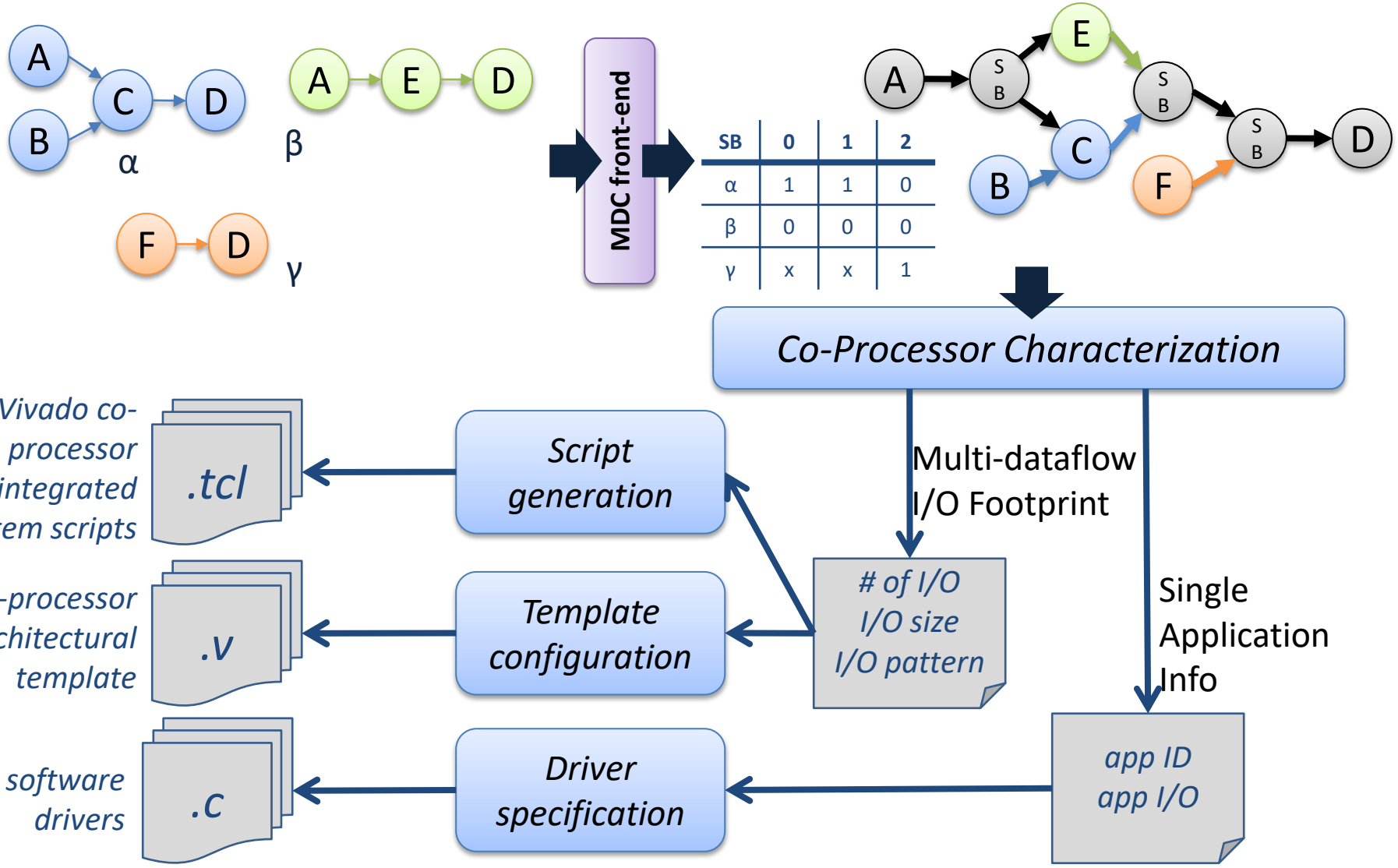
Co-Processor Generator



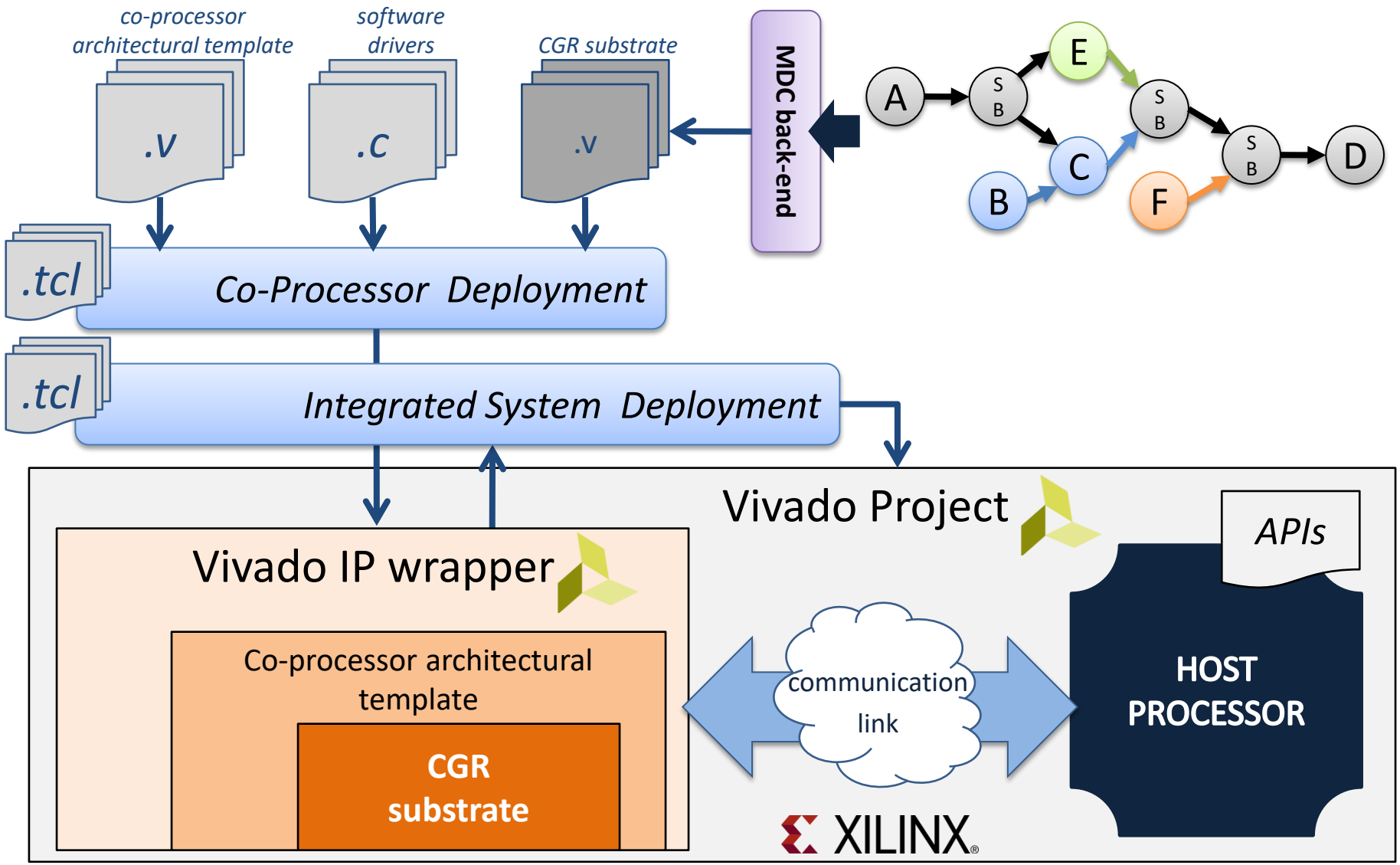
Co-Processor Generator



Co-Processor Generator



Co-Processor Generator



PROSSIMO
Progettazione, sviluppo e ottimizzazione di sistemi intelligenti multi-oggetto

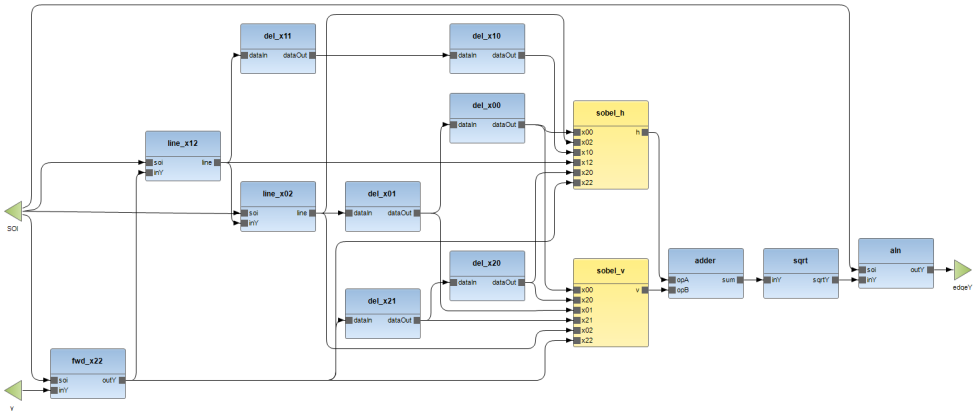
Tutorial

Step 2: Baseline MDC Datapath Merging

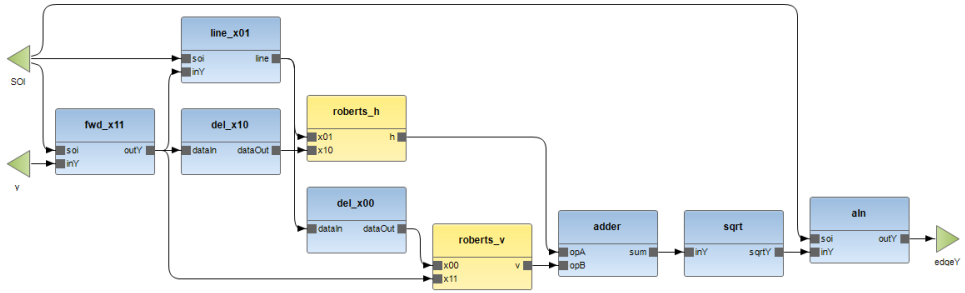
MDC Datapath Merging

Merging Expectations

Sobel dataflow



Roberts dataflow



actor	Sobel	Roberts	NS	S
Forward2x2	0	1	1	0
Forward3x3	1	0	1	0
Delay	6	2	4	2
LineBuffer	2	1	1	1
LeftShifter	4	0	4	0
Subtractor	6	2	4	2
Adder3x1	2	0	2	0
Multiplier	2	2	0	2
Adder2x1	1	1	0	1
Sqrt	1	1	0	1
Align2x2	0	1	1	0
Align3x3	1	0	1	0
Total	26	11	19	9

NS = Non Shareable, **S** = Shareable

Try It Yourself

Merge Sobel and Roberts

1. Click on Run → Run Configurations… on the main menu
2. Double click on Orcc Compilation to create a new run configuration
3. Choose a Name for the new run configuration (e.g. Tutorial Merging 1)
4. Select Tutorials as project referenced by the new run configuration

Try It Yourself

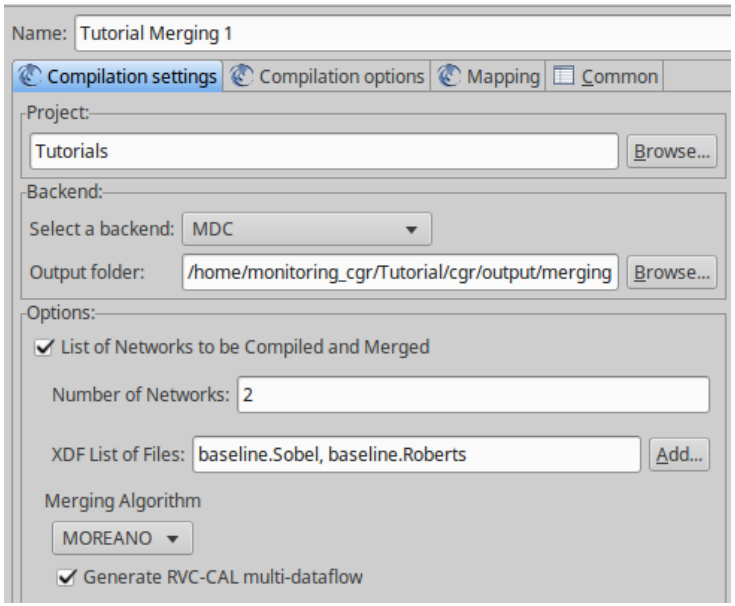
Merge Sobel and Roberts

5. Choose MDC as backend for the run configuration
6. Select an output file directory where generated outputs will be stored
   - /home/monitoring_cgr/Tutorial/cgr/output/merging
7. Check the List of Networks to be Compiled and Merged box
8. Make sure that the Number of Networks is 2 (we are going to merge Sobel and Roberts)

Try It Yourself

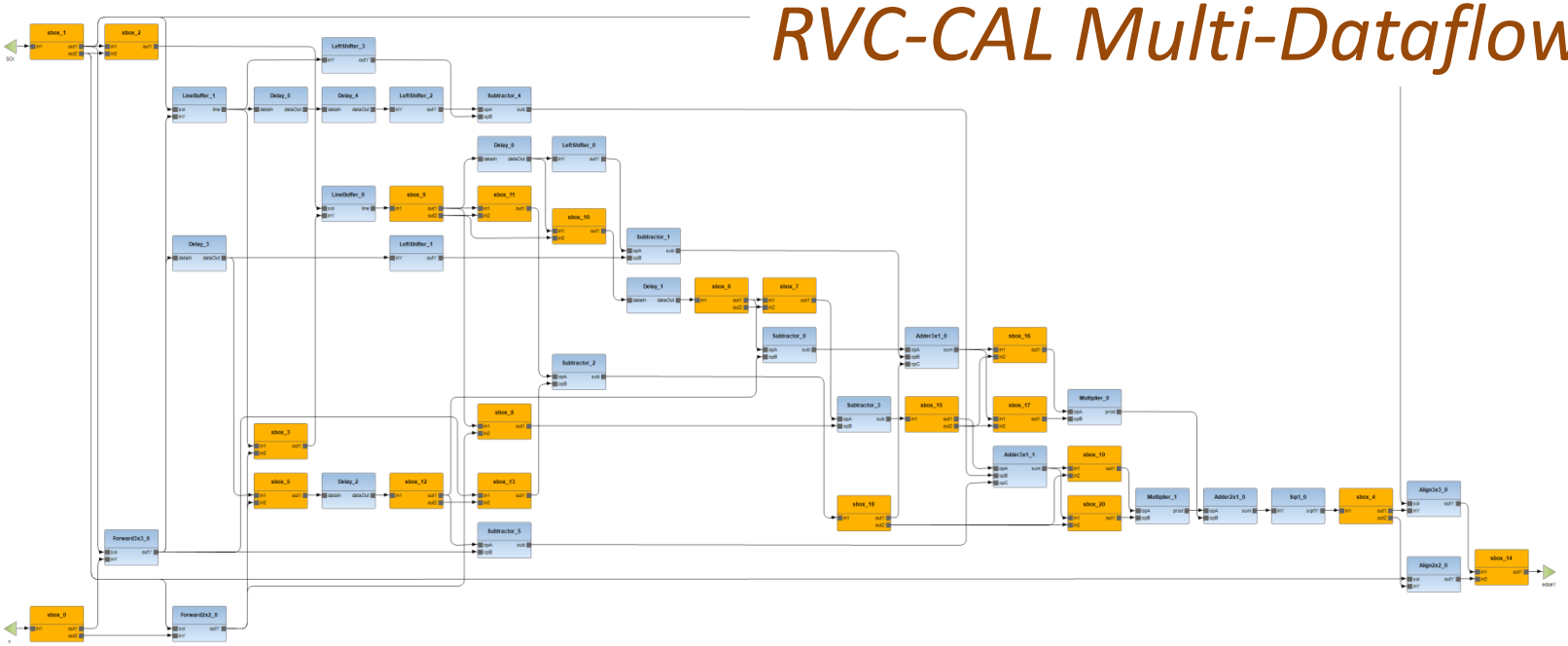
Merge Sobel and Roberts

9. Add Sobel and Roberts to the XDF List of Files
10. Select MOREANO as Merging Algorithm
11. Check the Generate RVC-CAL multi-dataflow box



Try It Yourself

RVC-CAL Multi-Dataflow



Multi-Dataflow Composer report

input networks composition

- network Sobel number of actors: 26
- network Roberts number of actors: 11
  --> input networks total number of actors: 37

multi-dataflow network composition: 00_Sobel1Roberts1

- number of merged networks: 2,0
- number of actors: 49
  -- original actors: 28 (57.14286%)*
  -- sbox actors: 21 (42.857143%)*
  -- shared actors 9 (24.324326%)**

* with respect to the multi-dataflow network number of actors
** with respect to the input networks total number of actors

Try It Yourself

Sbox Actor

```
@sbox actor Sbox1x2int16 (int ID=0, int SIZE=16)
   int(size=SIZE) in1
      ==>
   int(size=SIZE) out1,
   int(size=SIZE) out2
:

   forward0: action in1: [a] ==> out1: [a]
   guard not SEL[ID]
   end

   forward1: action in1: [a] ==> out2: [a]
   guard SEL[ID]
   end
end
```

- Sbox CAL special actors are created
- One different Sbox cal actor per data size and kind of Sbox (1x2 or 2x1)

Try It Yourself

Configurator Unit

```
unit Configurator:

   bool SEL[21] = SEL2;

   // ID = 1 Sobel
   bool SEL1[21] = [
      false, false, false, false, false,
      false, false, false, false, false,
      false, false, false, false, false,
      false, false, false, false, false,
      false ];

   // ID = 2 Roberts
   bool SEL2[21] = [
      true, true, true, true, true,
      true, true, true, true, true,
      true, true, true, true, true,
      true, true, true, true, true,
      true ];

end
```

- Special CAL file capable of configuring Sboxes and then the multi-dataflow
- By changing the SEL (SEL1 or SEL2) it is possible to execute Sobel or Roberts

Try It Yourself

Multi-Dataflow Simulation

1. Change the Testbench network to set the Multi-Dataflow as DUT and Save
2. Run the simulation with the new DUT
3. Change the SEL variable in the Configurator unit from SEL1 (Sobel) to SEL2 (Roberts)
4. Check results in terms of detected edges (on the display) and processing time (on the console)

Try It Yourself

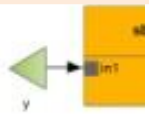
Generate Platform

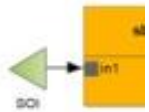
1. Open Run Configurations
2. Select the merging run configuration previously created (Tutorial Merging 1)
3. Uncheck Generate RVC-CAL multi-dataflow
4. Check Generate HDL multi-dataflow
5. Put input protocol.xml as Protocol File
   - /home/monitoring_cgr/Tutorial/cgr/input/protocol.xml
6. Put input/lib folder as HDL component library
   - /home/monitoring_cgr/Tutorial/cgr/input/lib

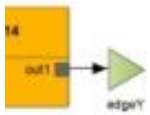
HW Reconfigurable Datapath

Check Platform Interface

```
// ----------------------------------------------------
// Multi-Dataflow Network module
// Date: 2019/05/08 16:03:48
// ----------------------------------------------------

module multi_dataflow (

    input [7 : 0] y_data,
    input y_wr,
    output y_full,

    input [15 : 0] SOI_data,
    input SOI_wr,
    output SOI_full,

    output  [7 : 0] edgeY_data,
    output  edgeY_wr,
    input  edgeY_full,

    input [7:0] ID,

    input clock,
    input reset
);
```








```
<protocol>
  <predecessor>
    <name>fifo_small</name>
    <sys_signals>
      <signal id="0" port="clk" size="1" net_port="clock"></signal>
      <signal id="1" port="rst" size="1" net_port="reset"></signal>
    </sys_signals>
  <comm_parameters>
    <parameter id="0" name="depth" value="bufferSize"></parameter>
    <parameter id="1" name="size" value="variable"></parameter>
  </comm_parameters>
  <comm_signals>
      <signal id="0" port="datain" channel="data" size="variable" kind="input" dir="direct"></signal>
      <signal id="1" port="dataout" channel="data" size="variable" kind="output" dir="direct"></signal>
      <signal id="2" port="enr" channel="rd" size="1" kind="input" dir="reverse"></signal>
      <signal id="3" port="enw" channel="wr" size="1" kind="input" dir="direct"></signal>
      <signal id="4" port="empty" channel="empty" size="1" kind="output" dir="direct"></signal>
      <signal id="5" port="full" channel="full" size="1" kind="output" dir="reverse"></signal>
    </comm_signals>
  </predecessor>
  <actor>
    <sys_signals>
      <signal id="0" port="clock" size="1" net_port="clock"></signal>
      <signal id="1" port="reset" size="1" net_port="reset"></signal>
    </sys_signals>
    <comm_signals>
      <signal id="0" port="" channel="data" size="variable" kind="input" dir="direct"></signal>
      <signal id="1" port="" channel="data" size="variable" kind="output" dir="direct"></signal>
      <signal id="2" port="rd" channel="rd" size="1" kind="output" dir="reverse"></signal>
      <signal id="3" port="wr" channel="wr" size="1" kind="output" dir="direct"></signal>
      <signal id="4" port="empty" channel="empty" size="1" kind="input" dir="direct"></signal>
      <signal id="5" port="full" channel="full" size="1" kind="input" dir="reverse"></signal>
    </comm_signals>
  </actor>
  <sys_signals>
    <signal id="0" net_port="clock" size="1" kind="input" is_clock=""></signal>
    <signal id="1" net_port="reset" size="1" kind="input" is_resetn=""></signal>
  </sys_signals>
</protocol>
```

HW Reconfigurable Datapath

Check Platform Actors

```
fifo_small #(
        .depth(64),
        .size(9)
) fifo_small_Delay_0_dataIn(
        .datain(fifo_small_Delay_0_dataIn_data),
        .dataout(Delay_0_dataIn_data),
        .enr(Delay_0_dataIn_rd),
        .enw(fifo_small_Delay_0_dataIn_wr),
        .empty(Delay_0_dataIn_empty),
        .full(fifo_small_Delay_0_dataIn_full),
        .clk(clock),
        .rst(reset)
);
```

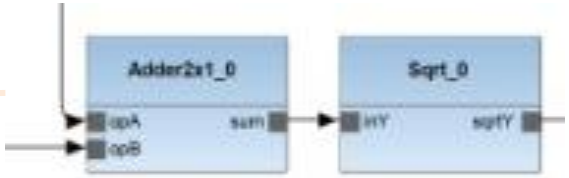
```
Delay actor_Delay_0 (
        .dataIn(Delay_0_dataIn_data),
        .dataIn_rd(Delay_0_dataIn_rd),
        .dataIn_empty(Delay_0_dataIn_empty),
        dataOut(Delay_0_dataOut_data),
        .dataOut_wr(Delay_0_dataOut_wr),
        .dataOut_full(Delay_0_dataOut_full),
        .clock(clock),
        .reset(reset)
);
```


```
<protocol>
  <predecessor>
    <name>fifo_small</name>
    <sys_signals>
      <signal id="0" port="clk" size="1" net_port="clock"></signal>
      <signal id="1" port="rst" size="1" net_port="reset"></signal>
    </sys_signals>
  <comm_parameters>
    <parameter id="0" name="depth" value="bufferSize"></parameter>
    <parameter id="1" name="size" value="variable"></parameter>
  </comm_parameters>
  <comm_signals>
      <signal id="0" port="datain" channel="data" size="variable" kind="input" dir="direct"></signal>
      <signal id="1" port="dataout" channel="data" size="variable" kind="output" dir="direct"></signal>
      <signal id="2" port="enr" channel="rd" size="1" kind="input" dir="reverse"></signal>
      <signal id="3" port="enw" channel="wr" size="1" kind="input" dir="direct"></signal>
      <signal id="4" port="empty" channel="empty" size="1" kind="output" dir="direct"></signal>
      <signal id="5" port="full" channel="full" size="1" kind="output" dir="reverse"></signal>
    </comm_signals>
  </predecessor>
  <actor>
    <sys_signals>
      <signal id="0" port="clock" size="1" net_port="clock"></signal>
      <signal id="1" port="reset" size="1" net_port="reset"></signal>
    </sys_signals>
    <comm_signals>
      <signal id="0" port="" channel="data" size="variable" kind="input" dir="direct"></signal>
      <signal id="1" port="" channel="data" size="variable" kind="output" dir="direct"></signal>
      <signal id="2" port="rd" channel="rd" size="1" kind="output" dir="reverse"></signal>
      <signal id="3" port="wr" channel="wr" size="1" kind="output" dir="direct"></signal>
      <signal id="4" port="empty" channel="empty" size="1" kind="input" dir="direct"></signal>
      <signal id="5" port="full" channel="full" size="1" kind="input" dir="reverse"></signal>
    </comm_signals>
  </actor>
  <sys_signals>
    <signal id="0" net_port="clock" size="1" kind="input" is_clock=""></signal>
    <signal id="1" net_port="reset" size="1" kind="input" is_resetn=""></signal>
  </sys_signals>
</protocol>
```

HW Reconfigurable Datapath

Check Platform Connections

```
assign fifo_small_Adder2x1_0_opA_data = Multiplier_0_prod_data;
assign fifo_small_Adder2x1_0_opA_wr = Multiplier_0_prod_wr;
assign Multiplier_0_prod_full = fifo_small_Adder2x1_0_opA_full;

assign fifo_small_Adder2x1_0_opB_data = Multiplier_1_prod_data;
assign fifo_small_Adder2x1_0_opB_wr = Multiplier_1_prod_wr;
assign Multiplier_1_prod_full = fifo_small_Adder2x1_0_opB_full;

assign fifo_small_Sqrt_0_inY_data = Adder2x1_0_sum_data;
assign fifo_small_Sqrt_0_inY_wr = Adder2x1_0_sum_wr;
assign Adder2x1_0_sum_full = fifo_small_Sqrt_0_inY_full;

assign sbox_0_in1_data = y_data;
assign sbox_0_in1_wr = y_wr;
assign y_full = sbox_0_in1_full;

assign fifo_small_Forward2x2_0_inY_data = sbox_0_out2_data;
assign fifo_small_Forward2x2_0_inY_wr = sbox_0_out2_wr;
assign sbox_0_out2_full = fifo_small_Forward2x2_0_inY_full;
```



HW Reconfigurable Datapath

Check Platform Configurator

```
// ----------------------------------------------------
// Configurator module
// Date: 2019/05/08 16:03:48
// ----------------------------------------------------

module configurator(
                input [7:0] ID,
                output reg [20:0] sel
);

always@(ID)
                case(ID)
                8'd1:           begin           // Sobel
                sel[0]=1'b0;
                …
                sel[20]=1'b0; end

                8'd2:           begin           // Roberts
                sel[0]=1'b1;
                …
                sel[20]=1'b1; end

                default:        sel=21'bx;
endcase
```

Cal Configurator

```
unit Configurator:

  bool SEL[21] = SEL2;

  // ID = 1 Sobel
  bool SEL1[21] = [
    false, false, false, false, false,
    false, false, false, false, false,
    false, false, false, false, false,
    false, false, false, false, false,
    false ];

  // ID = 2 Roberts
  bool SEL2[21] = [
    true, true, true, true, true,
    true, true, true, true, true,
    true, true, true, true, true,
    true, true, true, true, true,
    true ];

end
```

Try It Yourself

Run Co-Processor Generator

1. Click on Run → Run Configurations… on the main menu
2. Right clock on the Tutorial Merging 1 run configuration → Duplicate
3. Select a new name for the duplicated run configuration (e.g. Tutorial Coprocessor 1)
4. Select a new output folder
   - /home/monitoring_cgr/Tutorial/cgr/output/coprocessor
5. Under the Generate HDL multi-dataflow section check Generate Accelerator IP box

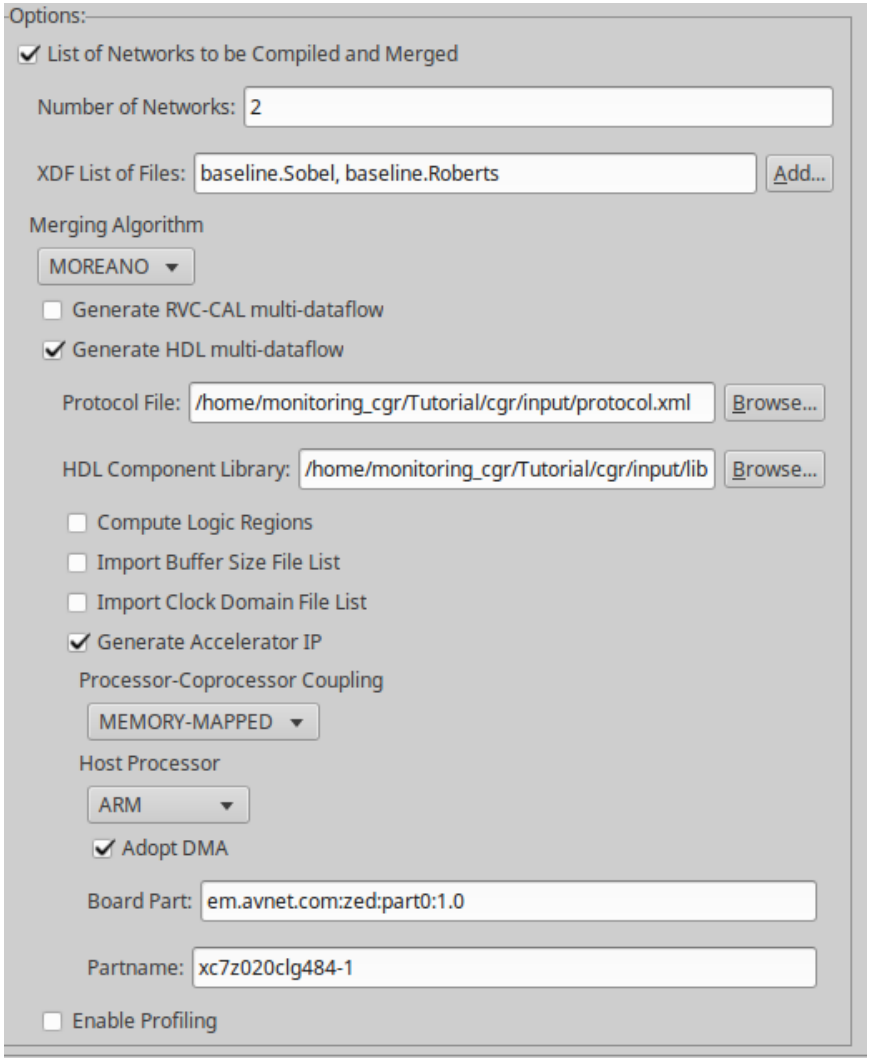
Try It Yourself

Run Co-Processor Generator

5. Select MEMORY-MAPPED as Processor Coprocessor Coupling
6. Select ARM as Host Processor
7. Check the Adopt DMA box
8. Leave the default Board Part and Partname (em.avnet.com:zed:part0:1.0 and xc7z020clg484-1) that are referred to the Avnet Zedboard Development board

Try It Yourself

Merge Sobel and Roberts

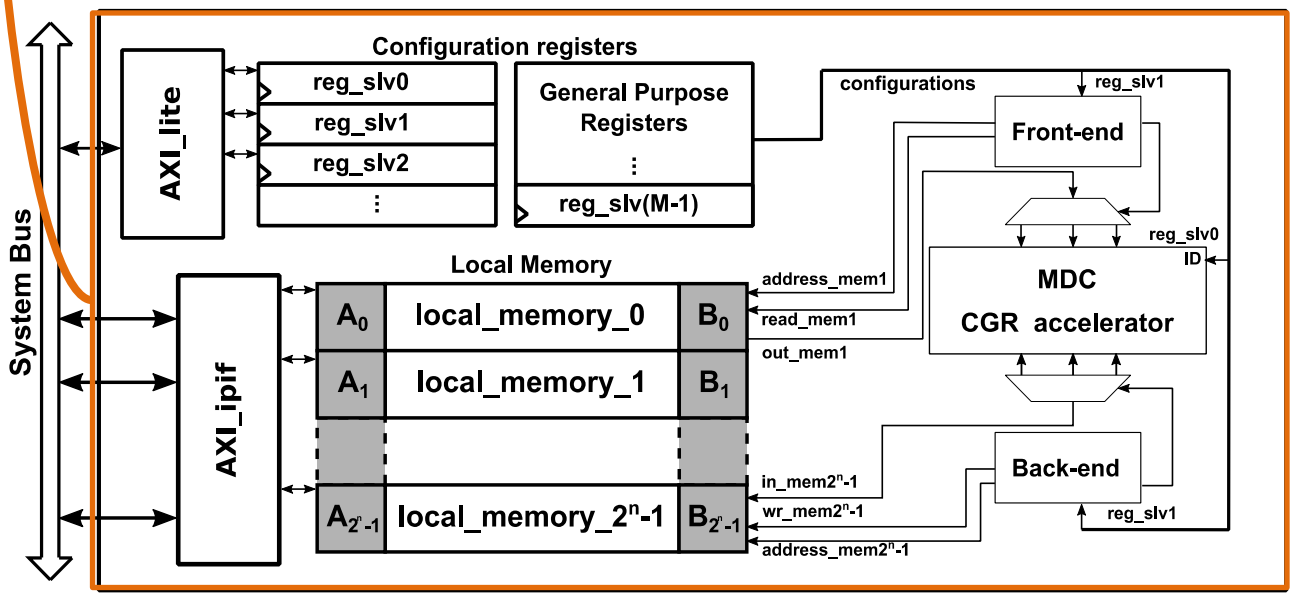


Try It Yourself

Check Generated HW Code

mm_accelerator.v

Top module of the accelerator instantiating all the other modules

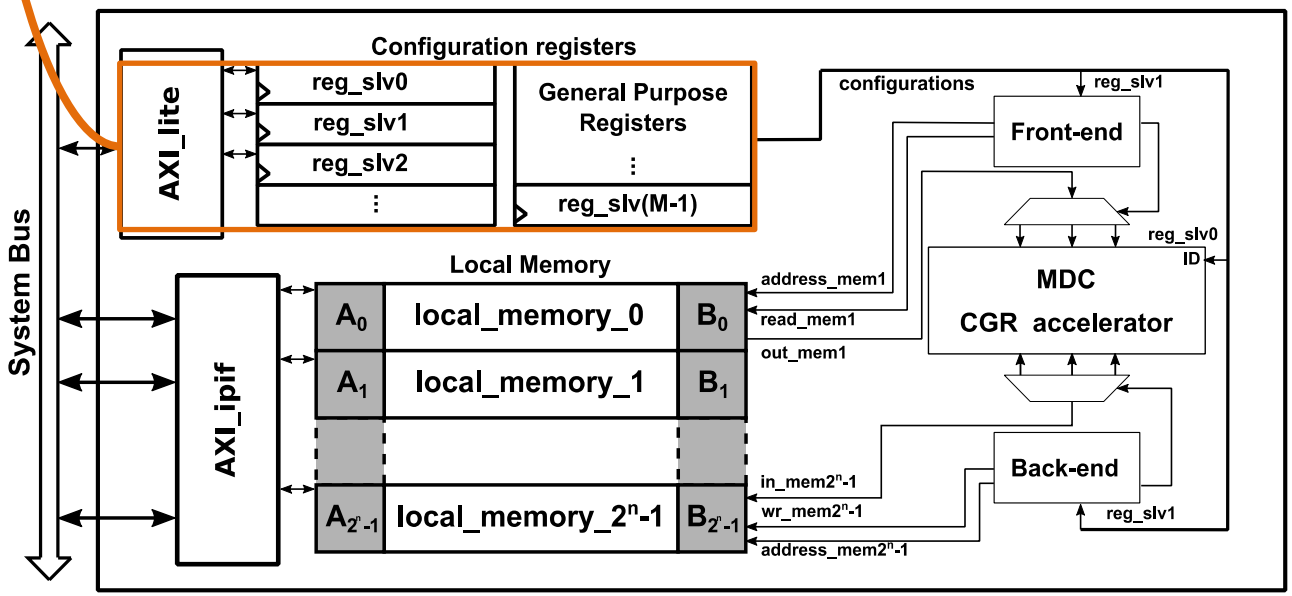


Try It Yourself

Check Generated HW Code

mm_accelerator.v
config_regs.v

AXI lite slave bus interface and configuration registers

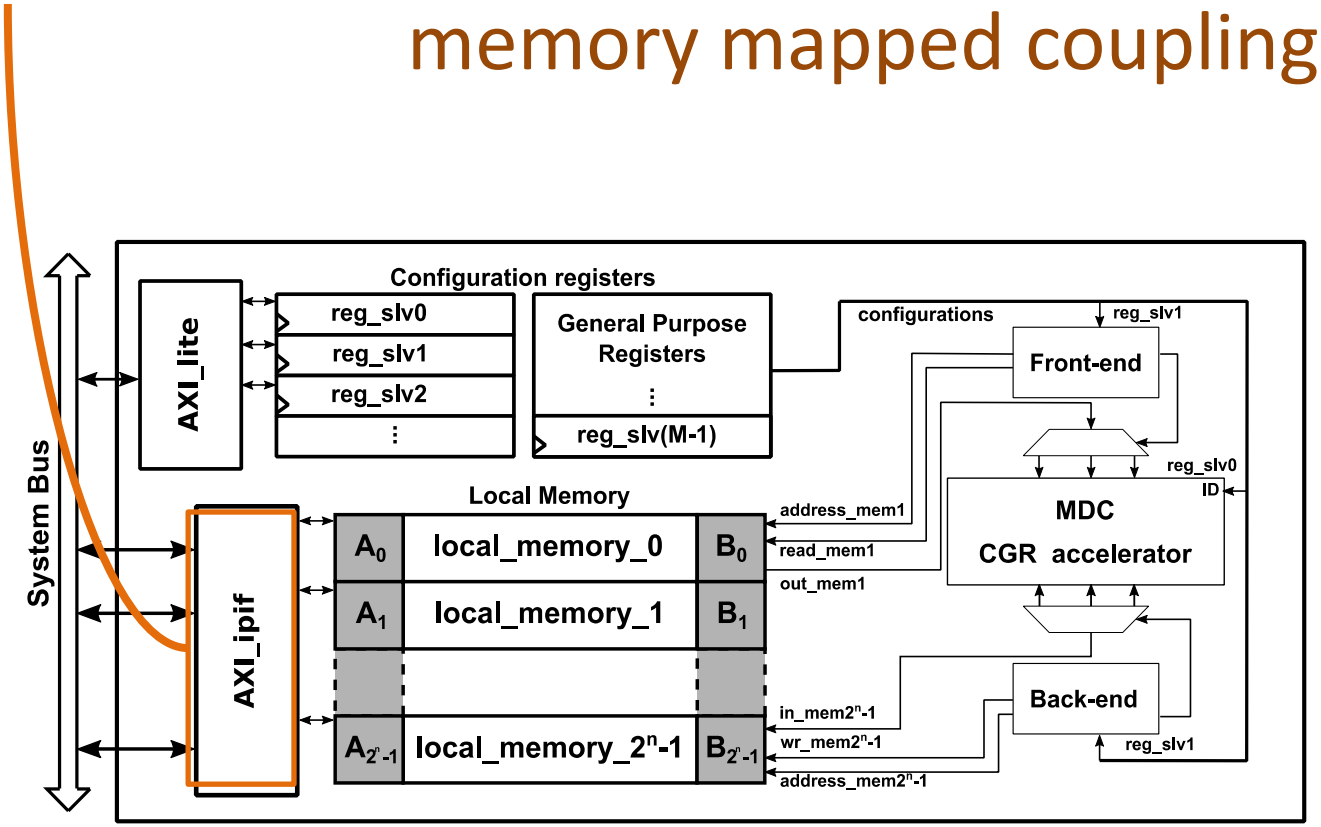


Try It Yourself

Check Generated HW Code

mm_accelerator.v
config_regs.v
axi_full_ipif.v

AXI full slave bus interface (only for memory mapped coupling)

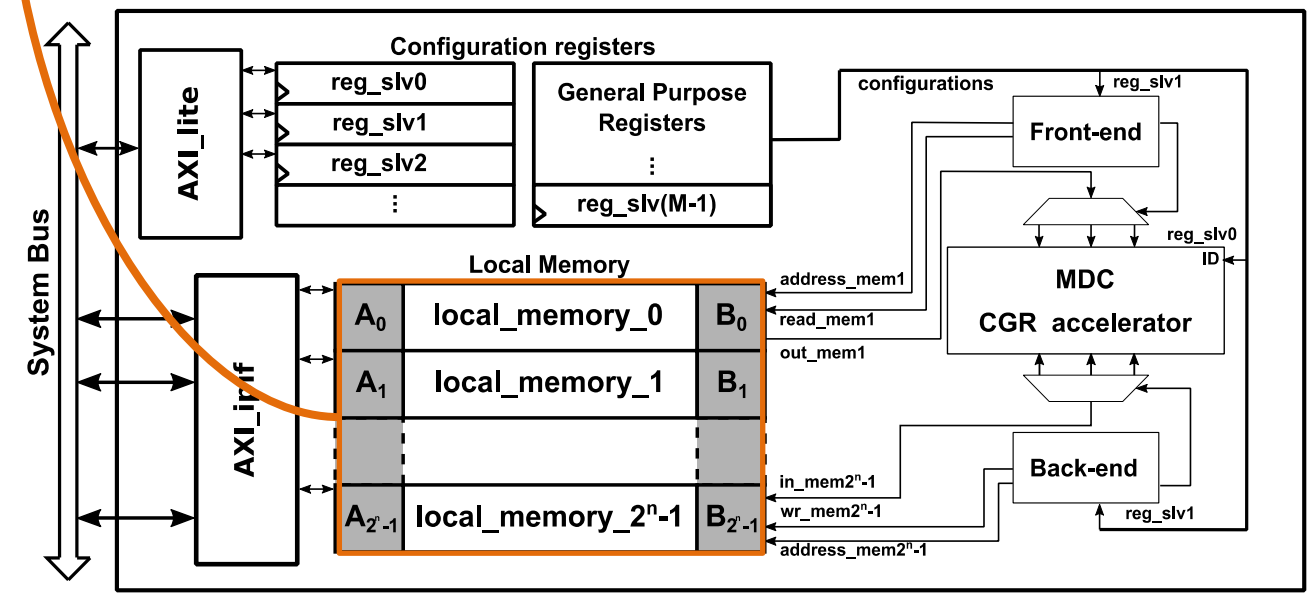


Try It Yourself

Check Generated HW Code

mm_accelerator.v
config_regs.v
axi_full_ipif.v
local_memory.v

Local memory banks to locally store input and output data (only for memory mapped coupling)

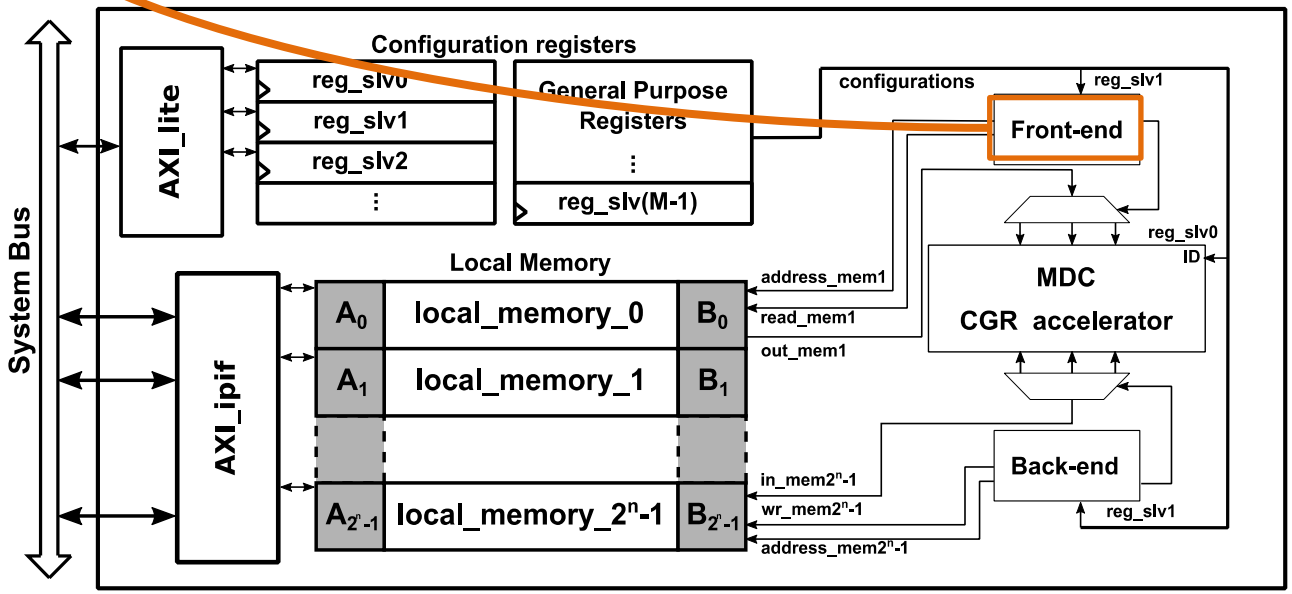


Try It Yourself

Check Generated HW Code

mm_accelerator.v
config_regs.v
axi_full_ipif.v
local_memory.v
front_end.v

Front-end to feed the CGR datapath inputs (only for memory mapped coupling)

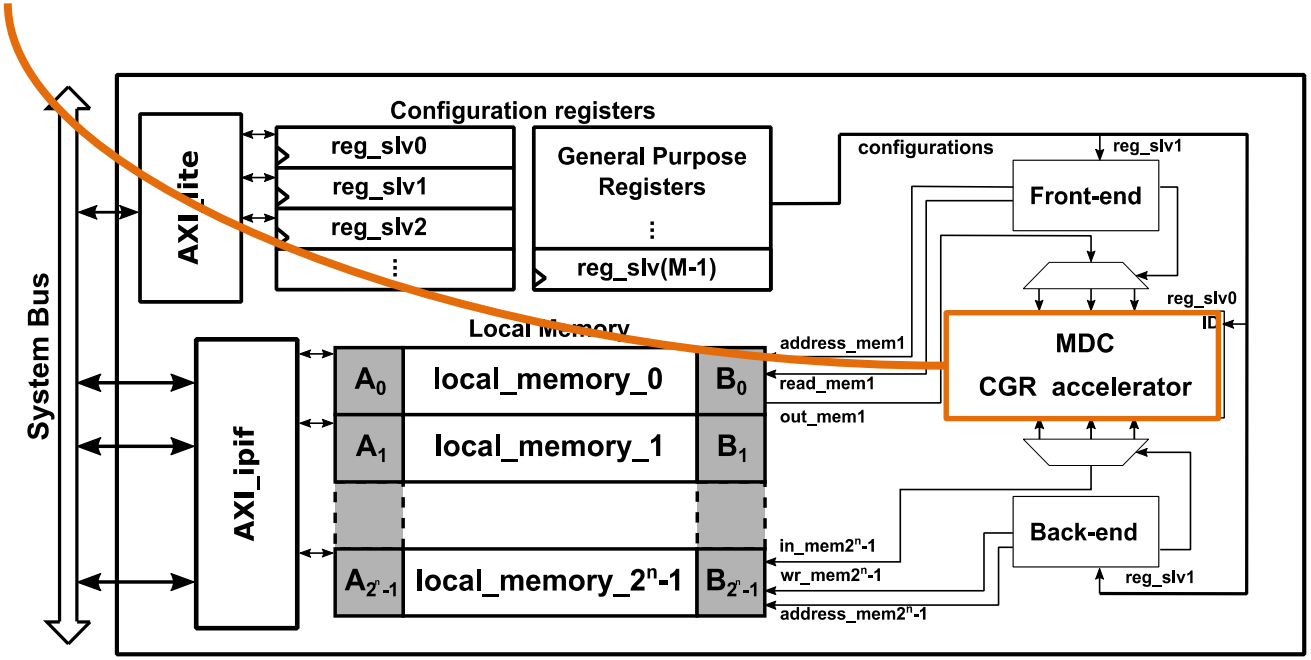


Try It Yourself

Check Generated HW Code

mm_accelerator.v
config_regs.v
axi_full_ipif.v
local_memory.v
front_end.v
multi_dataflow.v

The CGR datapath generated by the baseline MDC feature

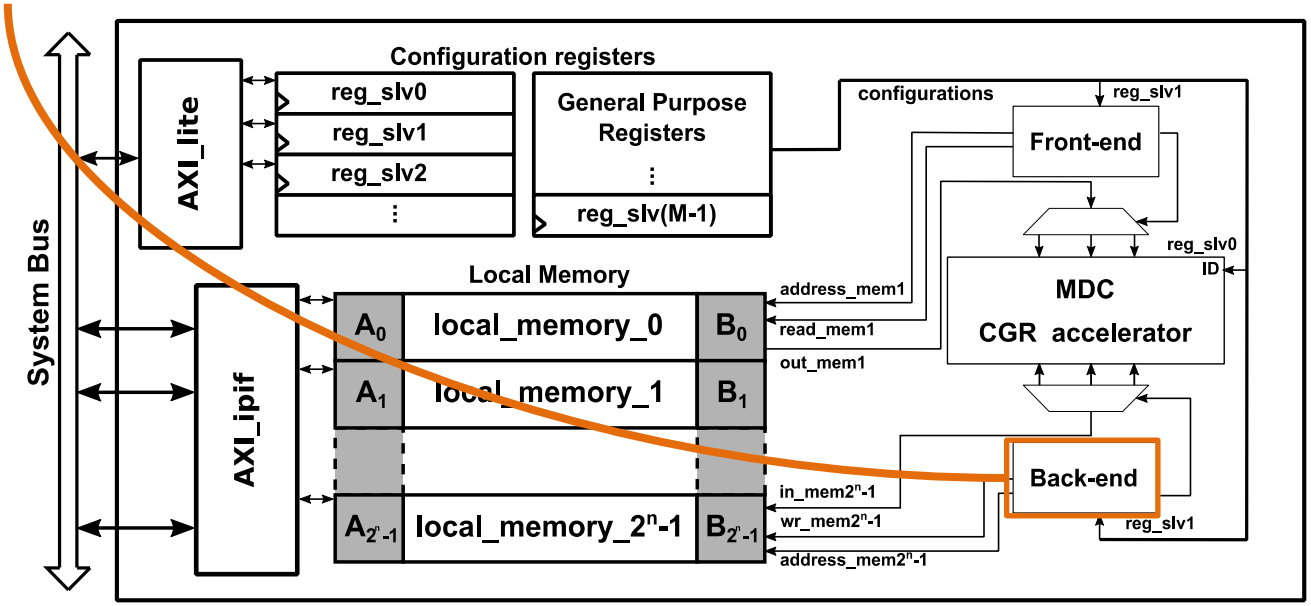


Try It Yourself

Check Generated HW Code

mm_accelerator.v
config_regs.v
axi_full_ipif.v
local_memory.v
front_end.v
multi_dataflow.v
back_end.v

Back-end to retrieve results from the CGR datapath outputs (only for memory mapped coupling)



Try It Yourself

Check Generated Drivers

```
#include "mm_accelerator.h"

int mm_accelerator_Sobel(
        // port edgeY
        int size_edgeY, int* data_edgeY,
        // port SOI
        int size_SOI, int* data_SOI,
        // port y
        int size_y, int* data_y
) {

volatile int* config = (int*) XPAR_MM_ACCELERATOR_0_CFG_BASEADDR;

// configure I/O
*(config + 1) = size_y - 1;
*(config + 3) = size_edgeY - 1;
*(config + 2) = size_SOI - 1;
```

- two parameters for each I/O port
  - number of data
  - data pointer
- configure registers with number of data for each I/O port

Try It Yourself

Check Generated Drivers

```
// send data port y
*((volatile int*) XPAR_AXI_CDMA_0_BASEADDR + (0x04>>2)) = 0x00000002; // verify idle
//*((volatile int*) XPAR_AXI_CDMA_0_BASEADDR + (0x00>>2)) = 0x00001000;  // irq en (optional)
*((volatile int*) XPAR_AXI_CDMA_0_BASEADDR + (0x18>>2)) = (int) data_y; // src
*((volatile int*) XPAR_AXI_CDMA_0_BASEADDR + (0x20>>2)) =
            XPAR_MM_ACCELERATOR_0_MEM_BASEADDR + MM_ACCELERATOR_MEM_1_OFFSET;
*((volatile int*) XPAR_AXI_CDMA_0_BASEADDR + (0x28>>2)) = size_y*4; // size [B]
while((*((volatile int*) XPAR_AXI_CDMA_0_BASEADDR + (0x04>>2)) & 0x2) != 0x2);

// send data port SOI
*((volatile int*) XPAR_AXI_CDMA_0_BASEADDR + (0x04>>2)) = 0x00000002; // verify idle
//*((volatile int*) XPAR_AXI_CDMA_0_BASEADDR + (0x00>>2)) = 0x00001000;  // irq en (optional)
*((volatile int*) XPAR_AXI_CDMA_0_BASEADDR + (0x18>>2)) = (int) data_SOI; // src
*((volatile int*) XPAR_AXI_CDMA_0_BASEADDR + (0x20>>2)) =
            XPAR_MM_ACCELERATOR_0_MEM_BASEADDR + MM_ACCELERATOR_MEM_2_OFFSET;
*((volatile int*) XPAR_AXI_CDMA_0_BASEADDR + (0x28>>2)) = size_SOI*4; // size [B]
while((*((volatile int*) XPAR_AXI_CDMA_0_BASEADDR + (0x04>>2)) & 0x2) != 0x2);
```

- send input data by means of DMA

Try It Yourself

Check Generated Drivers

```
// start execution (check matching ID)
*(config) = 0x1000001;

// wait for completion
while( ((*(config)) & 0x4) != 0x4 );

// receive data port edgeY
*((volatile int*) XPAR_AXI_CDMA_0_BASEADDR + (0x04>>2)) = 0x00000002; // verify idle
//*((volatile int*) XPAR_AXI_CDMA_0_BASEADDR + (0x00>>2)) = 0x00001000;  // irq en (optional)
*((volatile int*) XPAR_AXI_CDMA_0_BASEADDR + (0x18>>2)) =
            XPAR_MM_ACCELERATOR_0_MEM_BASEADDR + MM_ACCELERATOR_MEM_3_OFFSET;
*((volatile int*) XPAR_AXI_CDMA_0_BASEADDR + (0x20>>2)) = (int) data_edgeY; // dst
*((volatile int*) XPAR_AXI_CDMA_0_BASEADDR + (0x28>>2)) = size_edgeY*4; // size [B]
while((*((volatile int*) XPAR_AXI_CDMA_0_BASEADDR + (0x04>>2)) & 0x2) != 0x2);

return 0;

}
```

- launch execution and poll ready register

- receive output data by means of DMA

Step 3: Monitoring

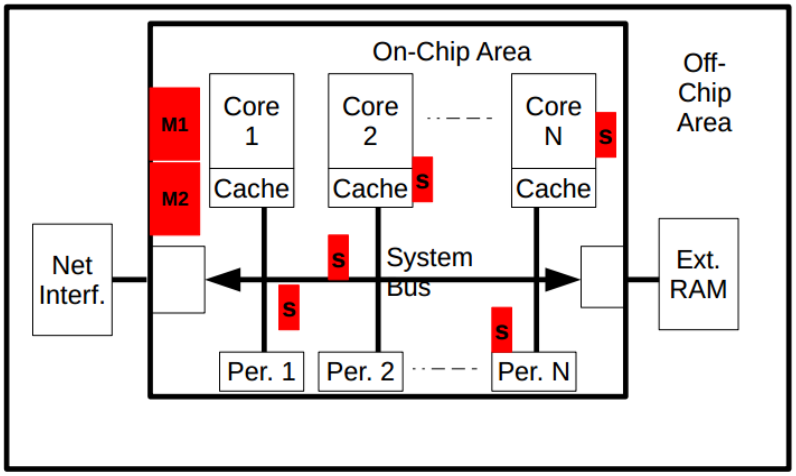
A Framework to Build Monitors

Monitoring Possibilities

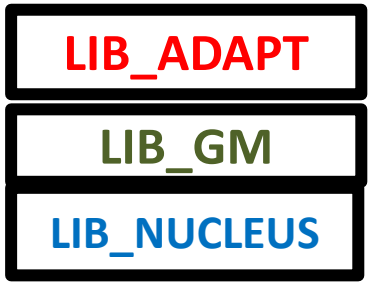
- A **better way** to gain the desired **visibility** into CPSs behaviours is to create **additional elements** to “watch” the processor for these types of events
- **Monitoring systems** can be of two types, each with different features:

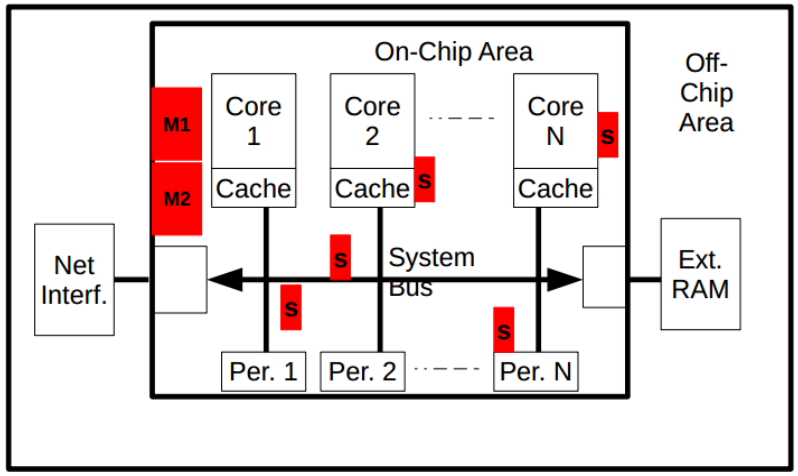
	HW	SW
Modification of the behaviour	☺	☹
SW overhead	☺	☹
Physical area	☹	☺
Power	☹	☺
Memory footprint	☺	☹
Flexibility	☹	☺
Re-usability	☹	☺
Micro-architectural events	☺	☹

Monitoring Framework

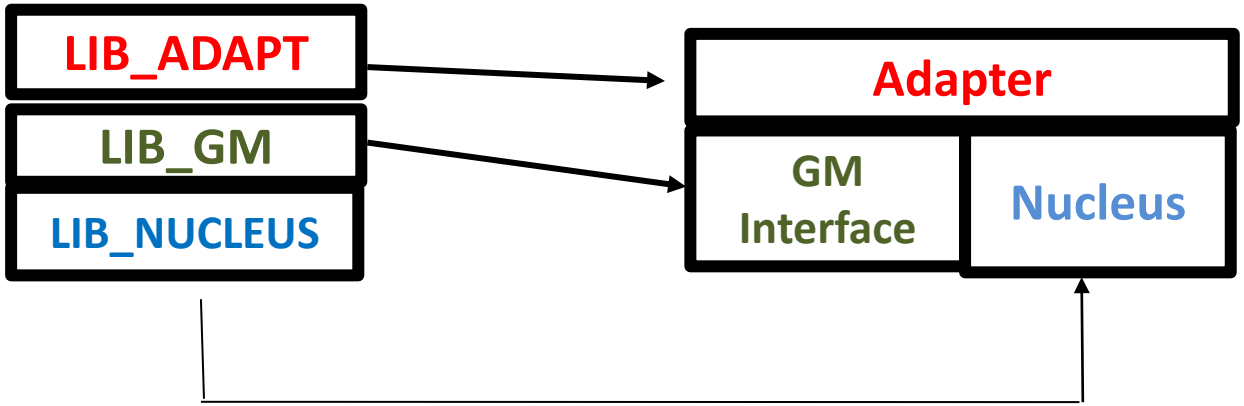


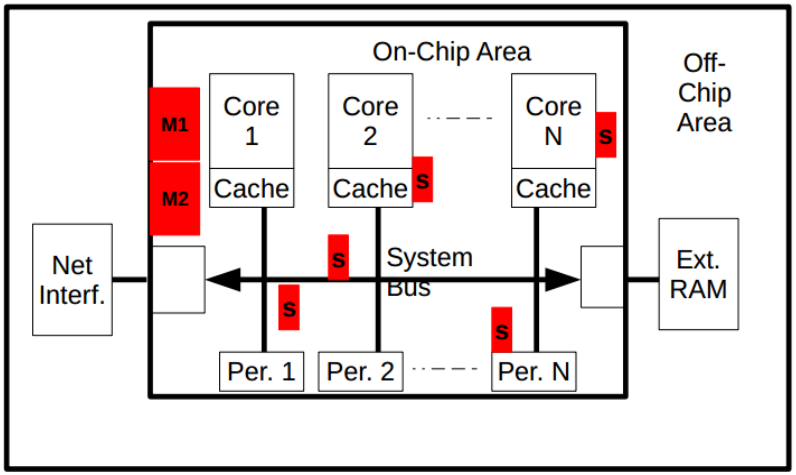
Monitoring Framework



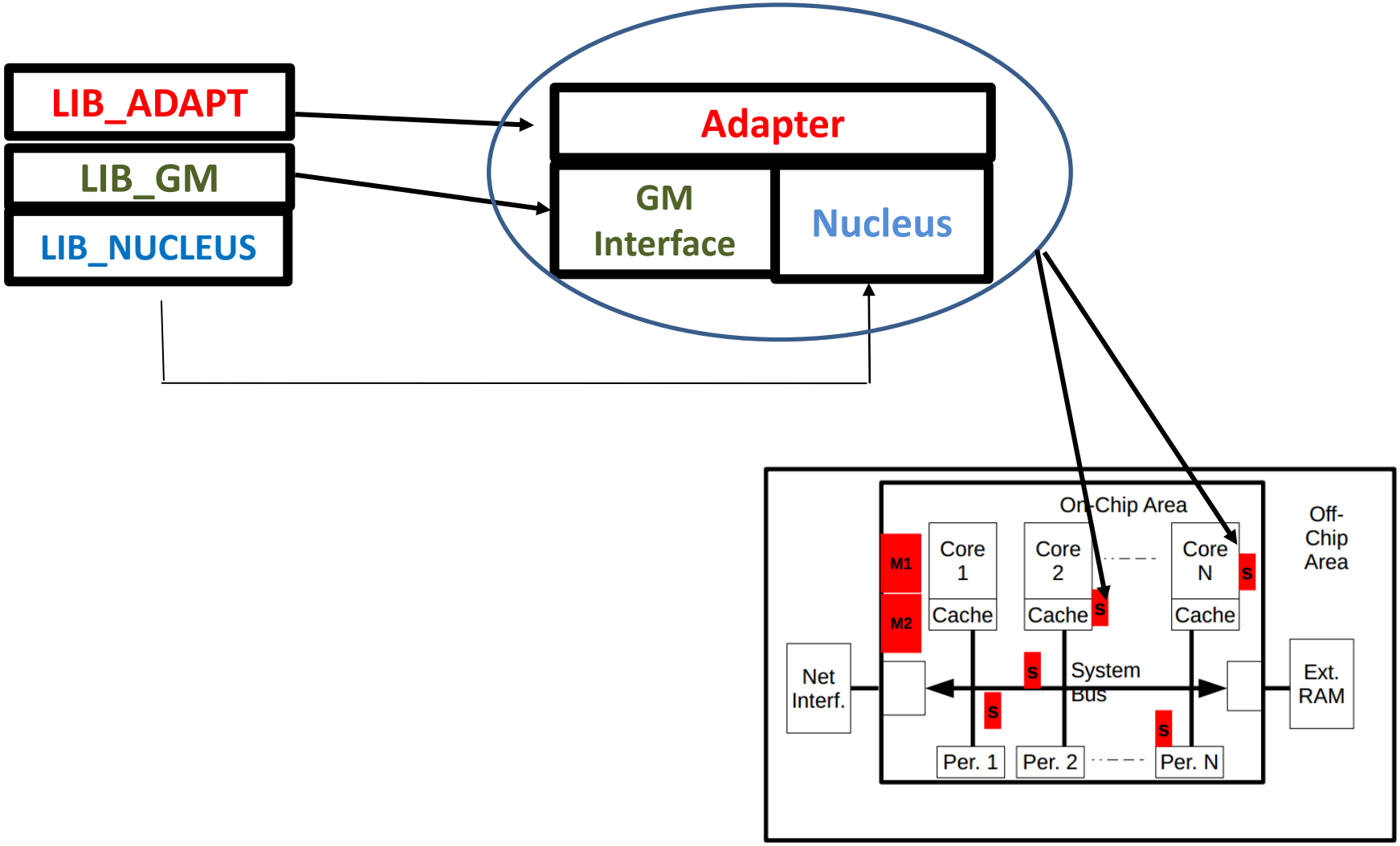


Monitoring Framework

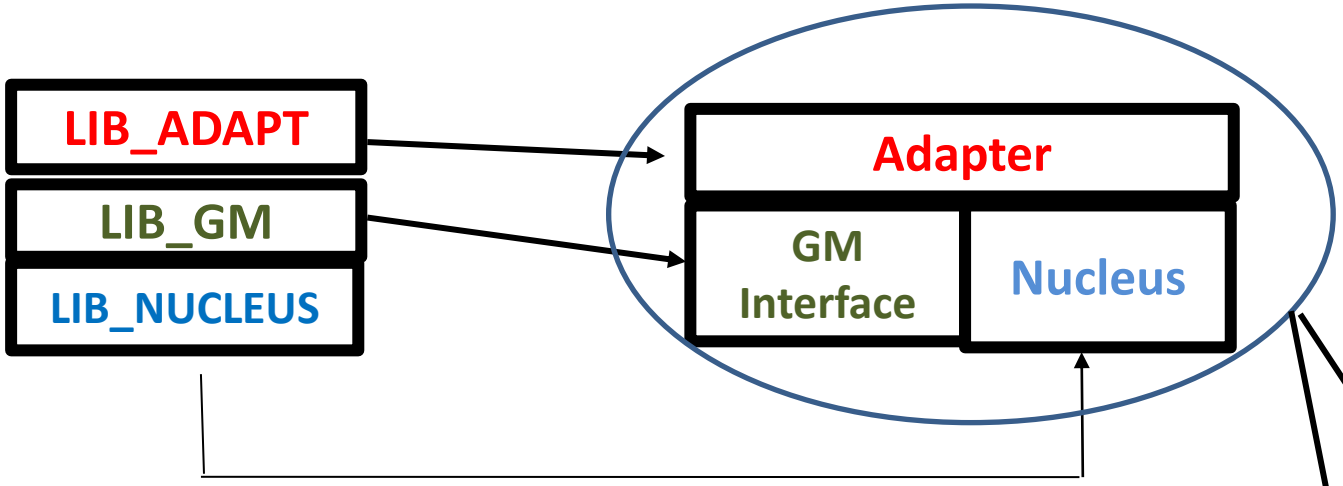




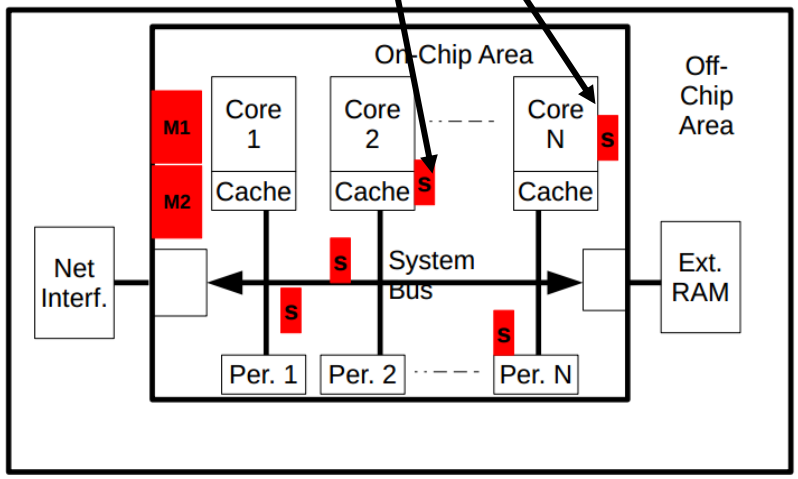
Monitoring Framework



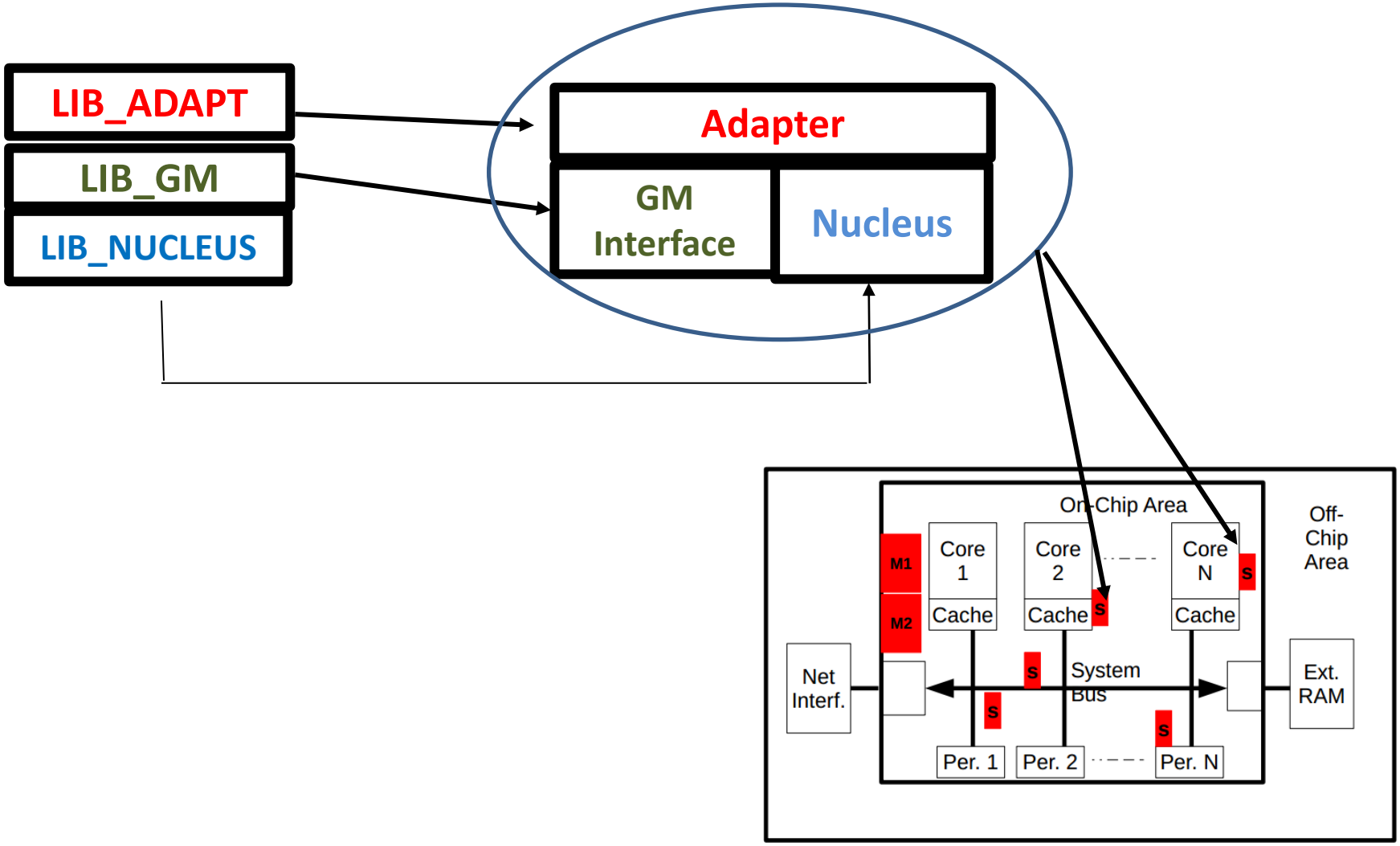
Monitoring Framework



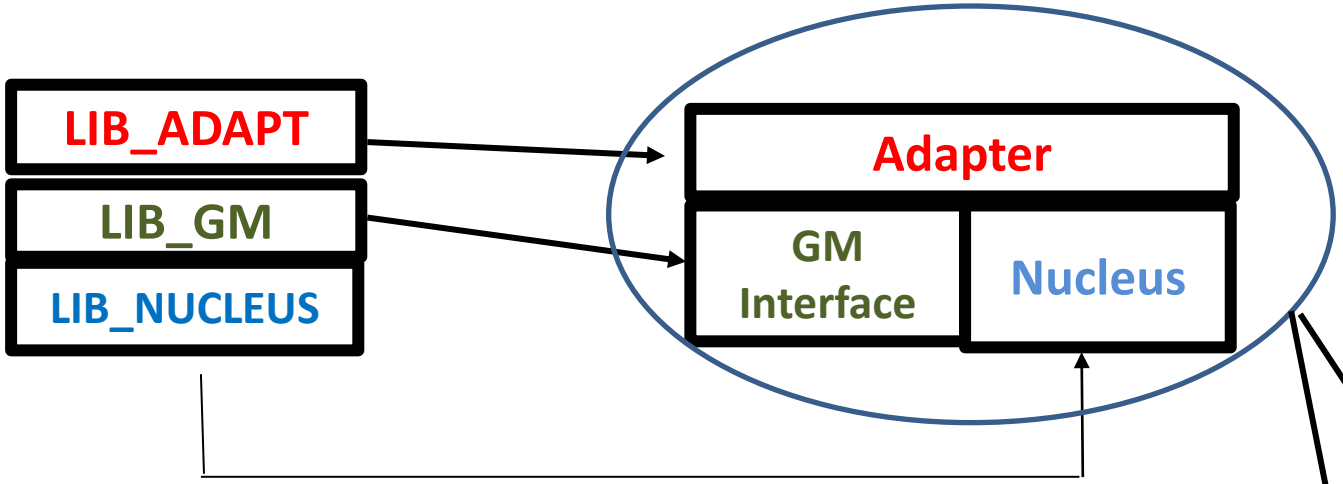
- **Adapter** is the part to be changed to monitor different interconnections



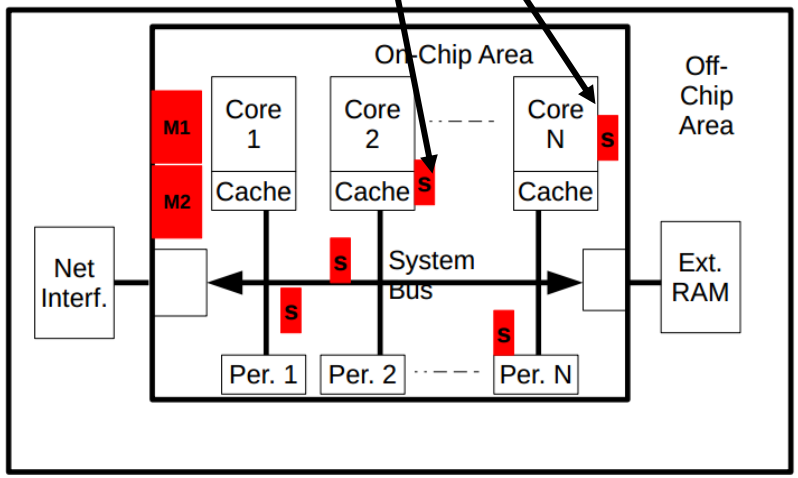
Monitoring Framework



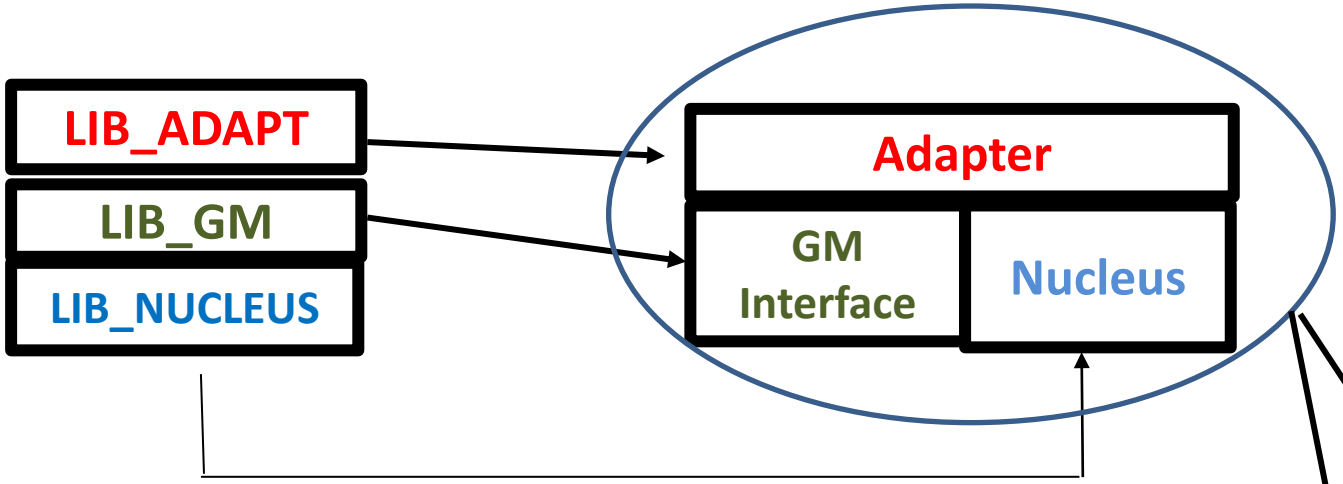
Monitoring Framework



- **Adapter** is the part to be changed to monitor different interconnections
- **Nucleus** is the part that contains the logic to measure metrics
- **GM Interface** is the part that initializes and sends data to a central monitor

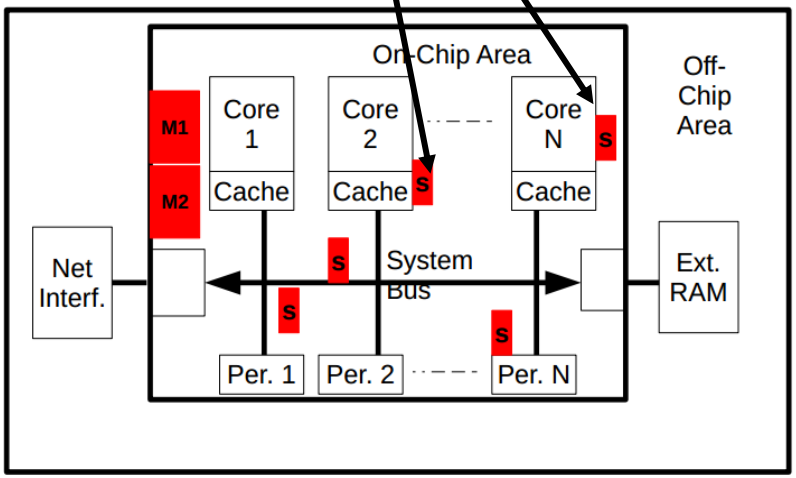


Monitoring Framework

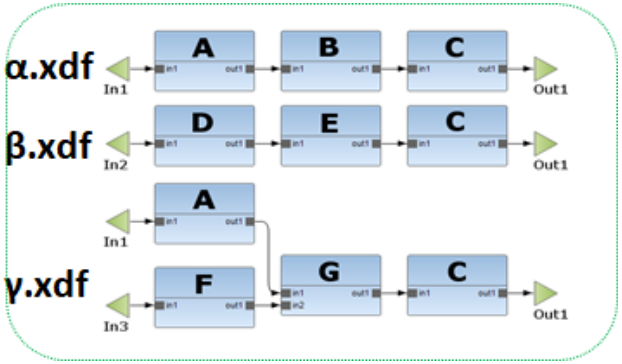


- **Targets**
  - **Microblaze**
  - **Leon3**
  - **Nios II**

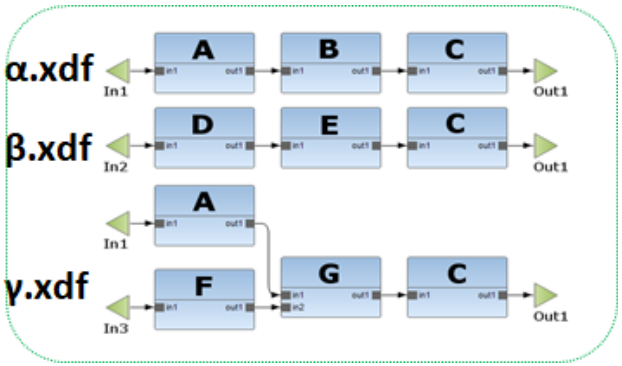
- **Adapter** is the part to be changed to monitor different interconnections
- **Nucleus** is the part that contains the logic to measure metrics
- **GM Interface** is the part that initializes and sends data to a central monitor

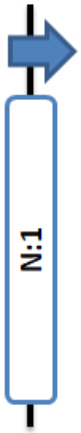


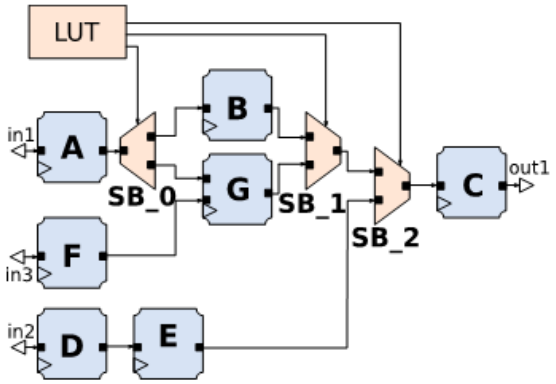
Dataflow to Hardware Mapping



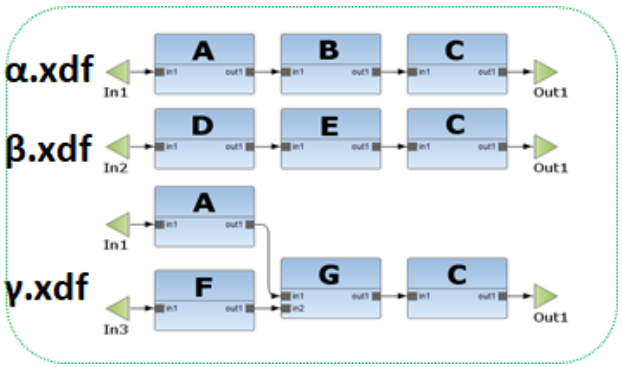
Dataflow to Hardware Mapping

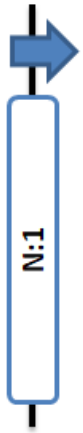


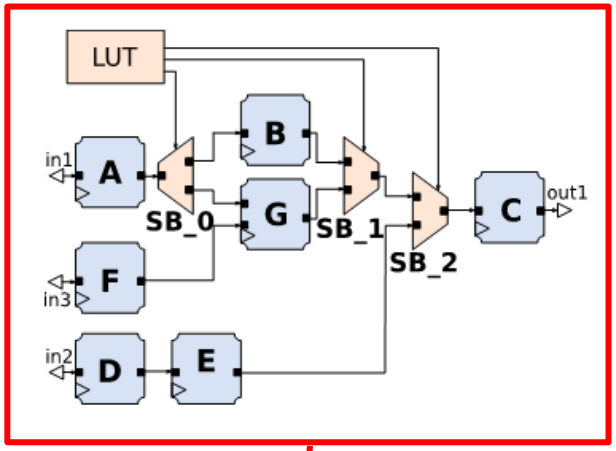


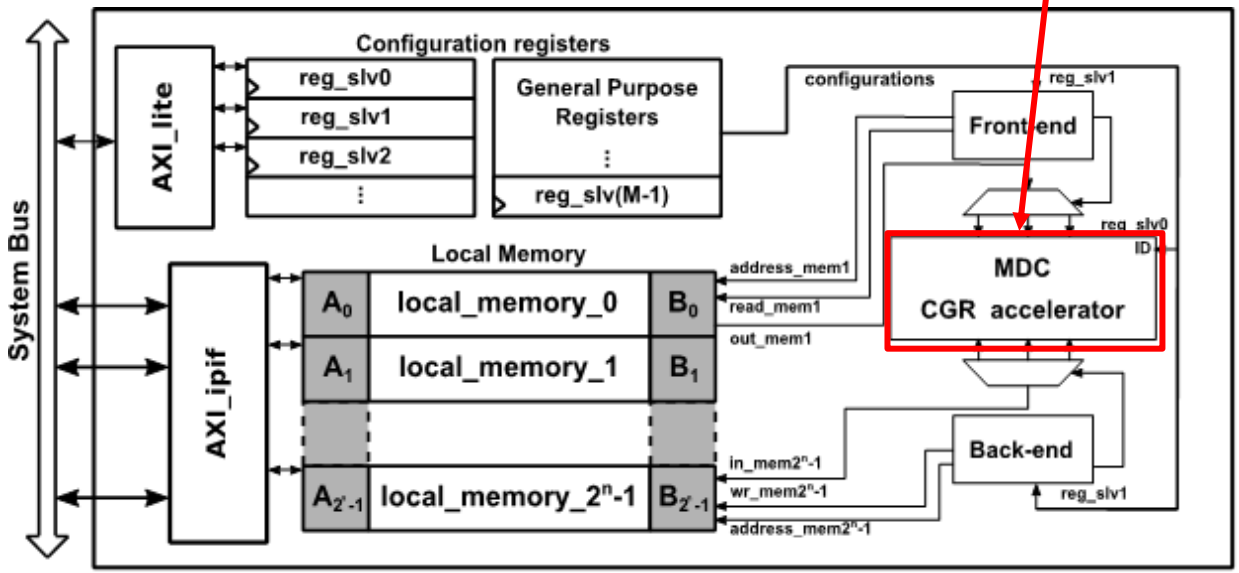


Dataflow to Hardware Mapping

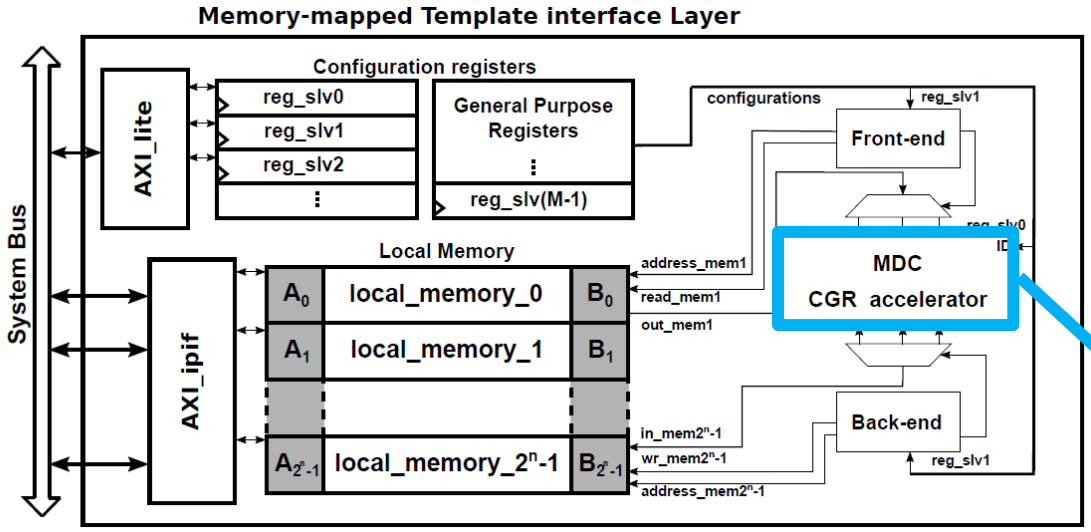


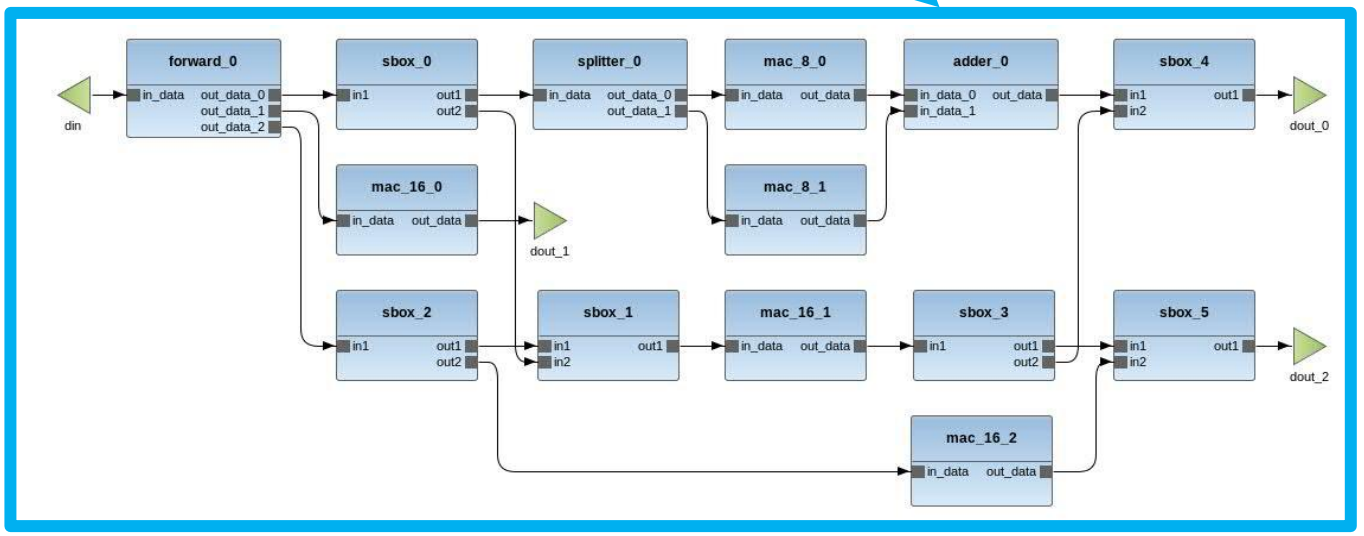




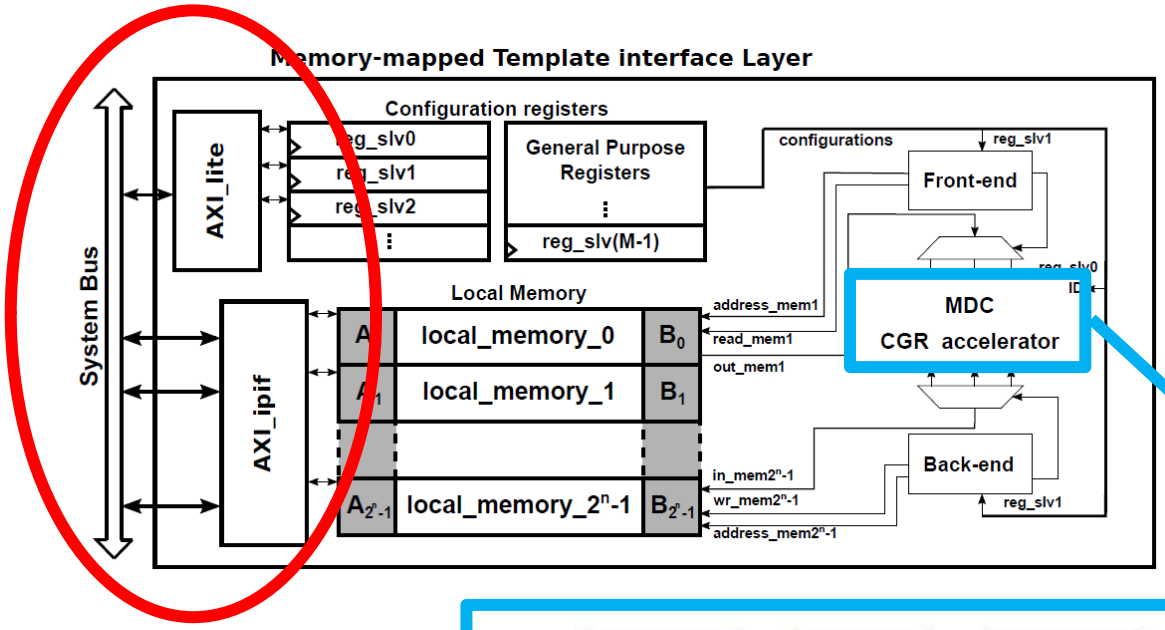


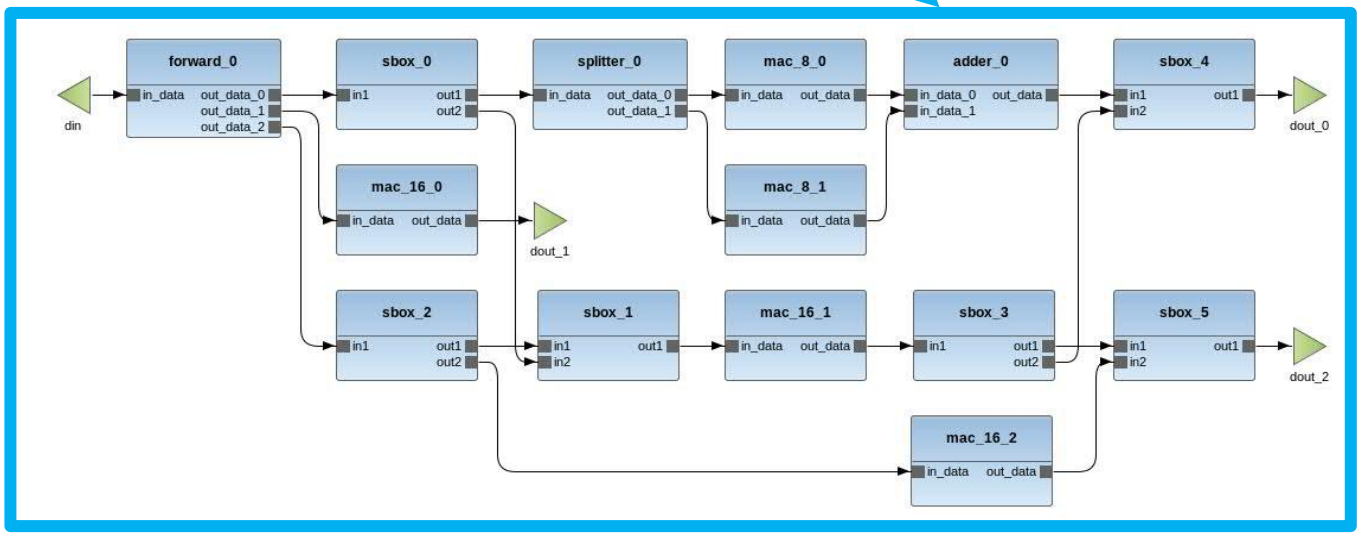
Monitoring Accelerators



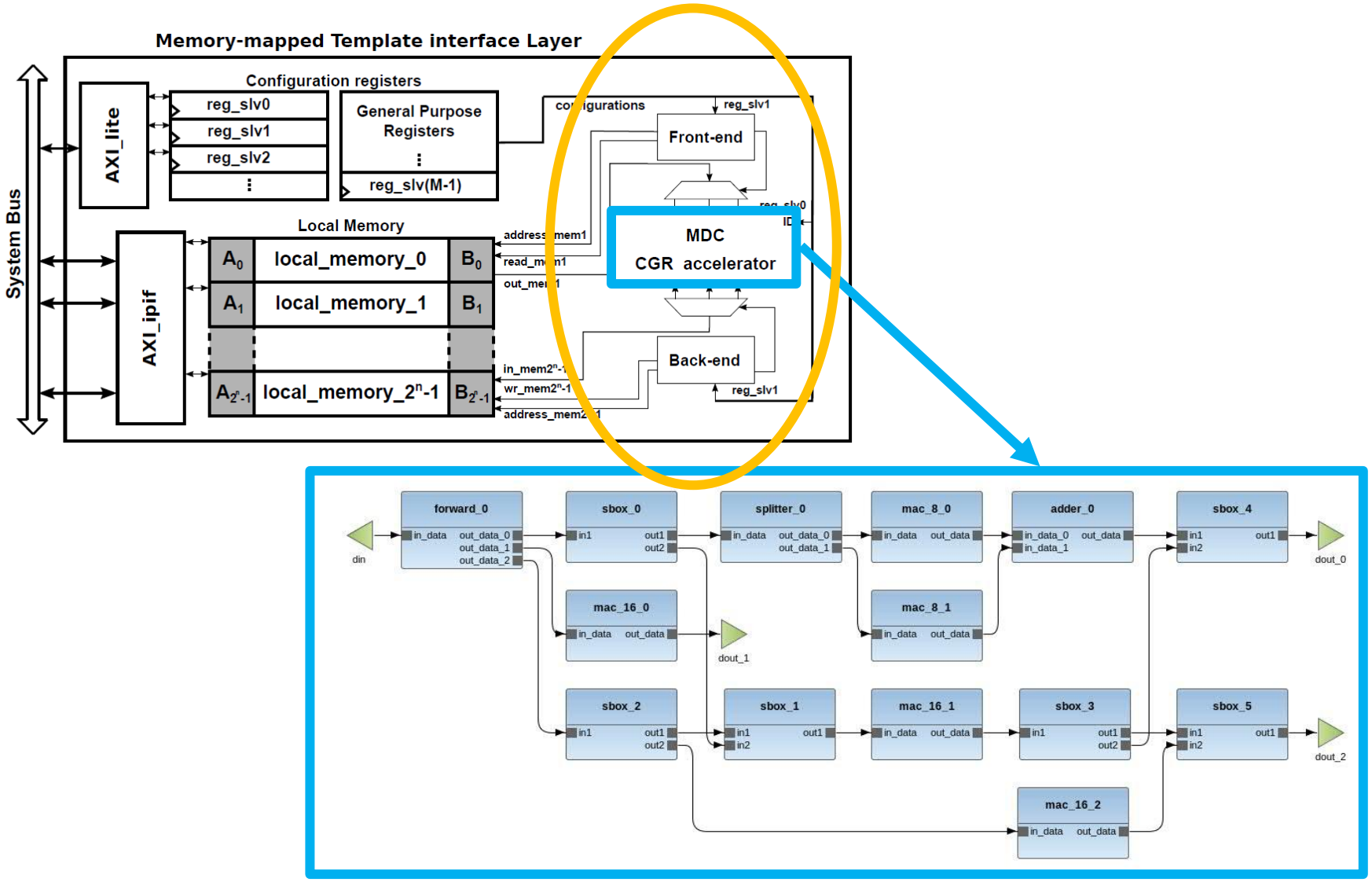


Monitoring Accelerators

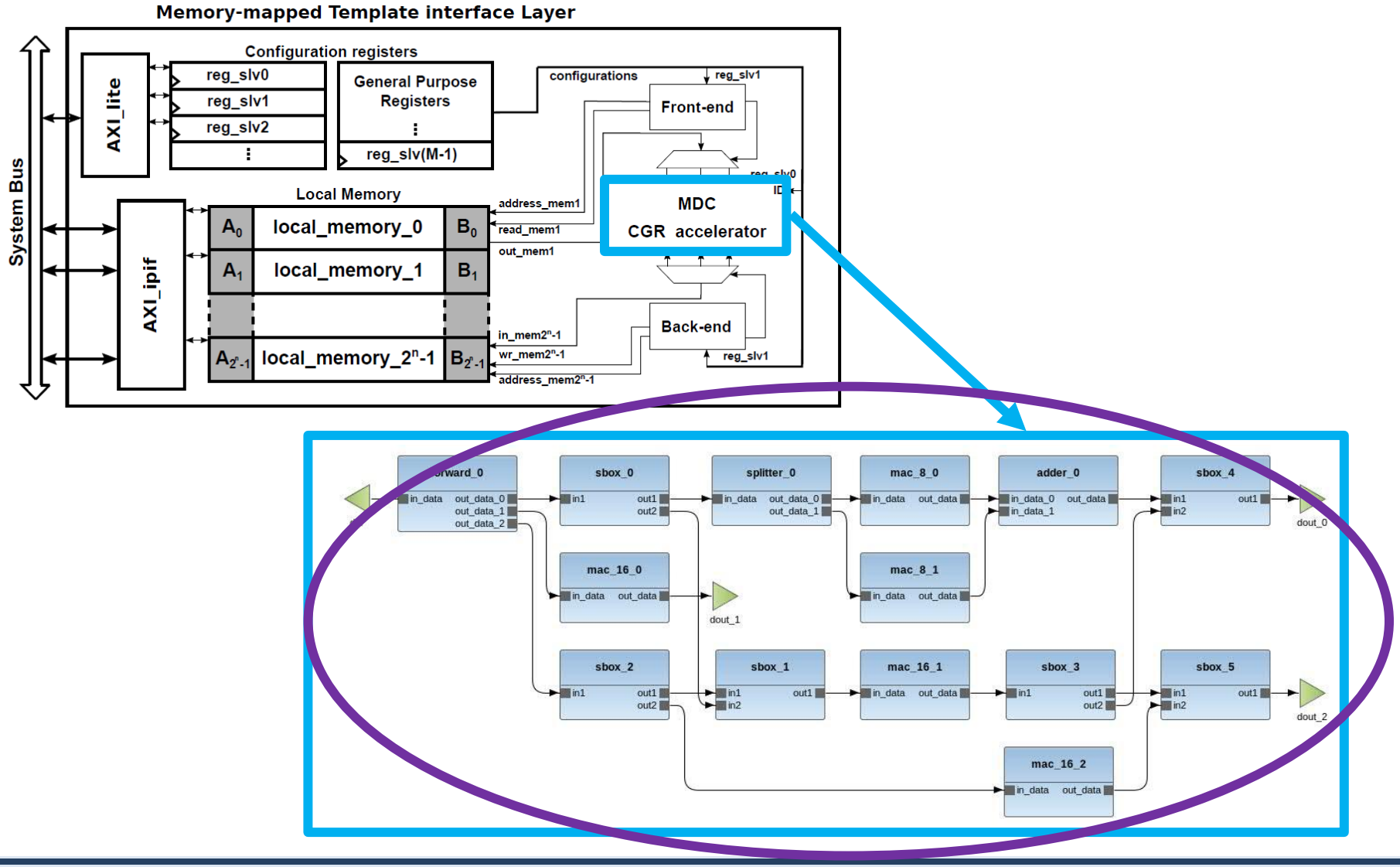




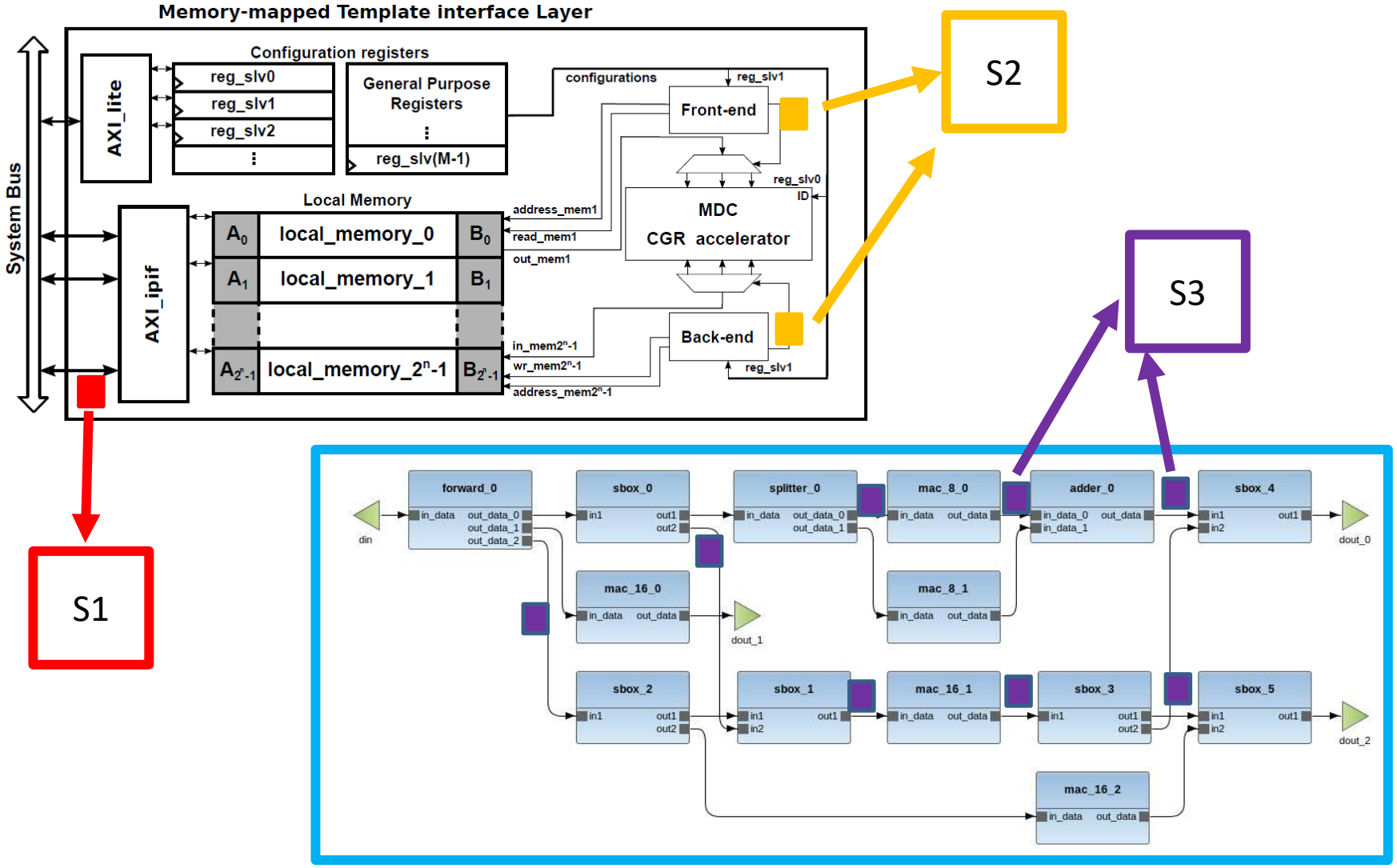
Monitoring Accelerators



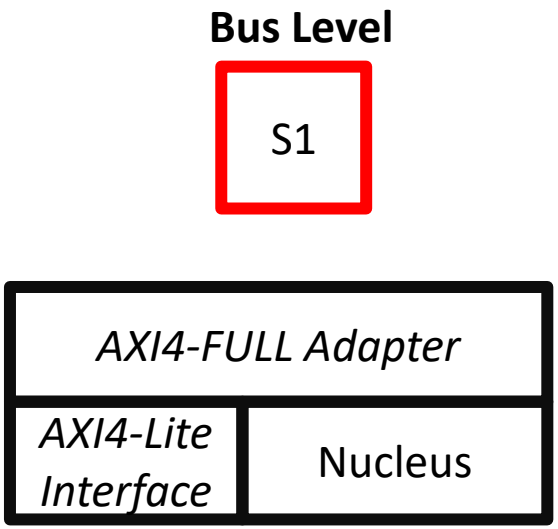
Monitoring Accelerators



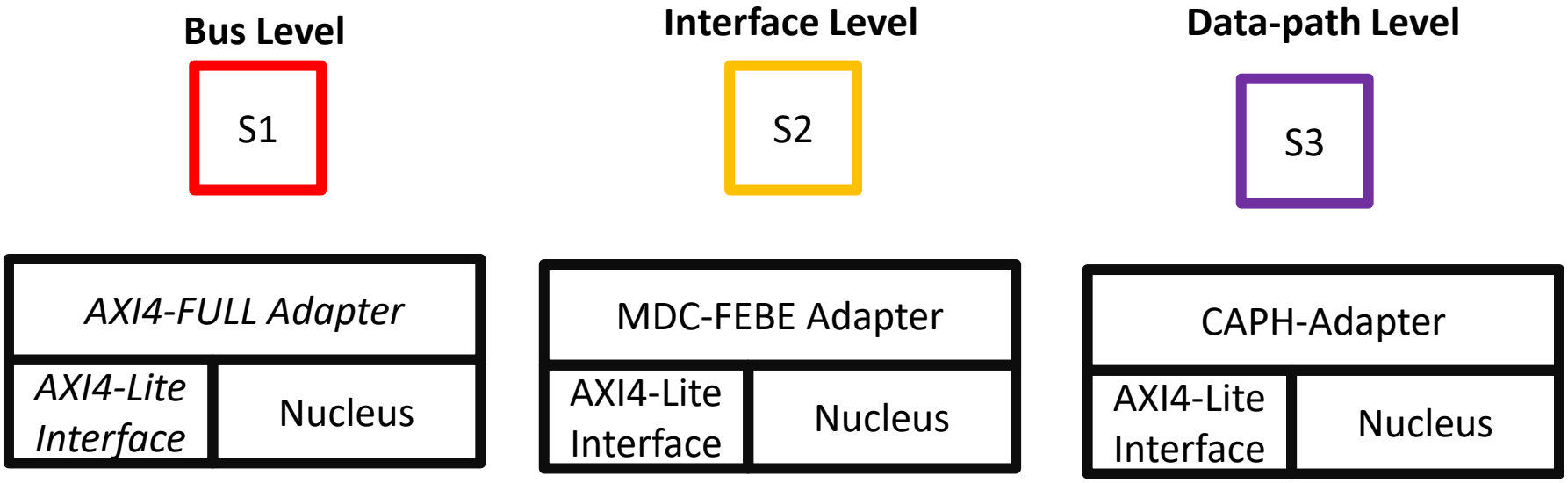
Monitoring Accelerators



Monitoring Accelerators



Monitoring Accelerators



Step 3: Monitoring

Tutorial

Before to Start

Prerequisites

1. *Xilinx Vivado 2017.4* (Webpack license)
    1. Xilinx environment variable setting
2. *pypng* python3 library
3. *gtkterm* terminal emulator

Note 1: in the provided Virtual Machine there are only Vivado Lab version and XSDK. This means that you can program FPGAs with existing bitstreams, as well as compile, execute and debug applications, but you cannot build hardware (new bitstreams) for FPGA.

Note 2: CheckK Points (CKP) are created along the proposed steps in order to allow you to start from different points, depending on you interest.

Try It Yourself

Prepare the Environment

1. From MDC output to Xilinx Vivado project
    1. Open Vivado
    2. Go in the folder *starting/MDC_to_Vivado*
    3. *source* the TCL script *generate_ip.tcl*
    4. *source* the TCL script *generate_top.tcl*
        1. the Vivado project of the integrated system (processor-coprocessor) will be automatically created
    5. generate the bitstream
        1. CKP1: the project together with the bitstream can be found in ready/CKP1_proj

Try It Yourself

Prepare the Environment

2. Export the hardware to XSDK
    1. Open Vivado
    2. Go in the folder *ready/CKP1_proj*
    3. Export the hardware to XSDK by clicking File->Export->Export Hardware
    4. Launch XSDK by clicking File -> Launch SDK in Vivado, or by typing *xsdk* in a Xubuntu shell
        1. CKP2: the XSDK workspace with the hardware already exported can be found in *ready/CKP2_ws*

Try It Yourself

Develop SW Applications

3. Create the SW Application (within XSDK)
    1. Create a new Application by using the content of *app_src* in */home/monitoring_cgr/Tutorial/Cluster_prossimo_monitor/starting/app_src*
        1. Click on File>New>Application Project
        2. Insert a valid project name (e.g. sobel_mon), then click on *Finish*
        3. Copy source files from *home/monitoring_cgr/Tutorial/Cluster_prossimo_monitor/starting/app_src*
        4. When asked, click on *Yes overwrite existing files*

Try It Yourself

Develop SW Applications

3. Create the SW Application (within XSDK)
    2. Open the *main.c* file and remove the comment from *#define EXPORT_RESULT*
        1. In this way, the input image, the SW filtered image and the HW filtered image will be printed by the UART
    3. Set the linker size to 0x20000000 by clicking to lscript.ld among source files
    4. Click on *Compile* (CTRL + B)
        1. CKP3: the workspace with the compiled application can be found in *ready/CKP3_ws* (from this workspace it is also possible to load files using the SD card)

Try it Yourself

Program the FPGA

4. FPGA programming
    1. Program the FPGA via JTAG
        1. Download the bitstream to the Zedboard by means of the JTAG connection (Xilinx>Program FPGA>Program)
    2. Set up serial communication
        1. Open gtkterm from the shell by typing *sudo gtkterm*
        2. Configure the communication
            1. Configuration>Port>Port ttyACM0
            2. Baud Rate 115200
            3. Leave the rest as default
        3. Log the received data to file (Log>To File>sobel.log)

Try it Yourself

Program the FPGA

4. FPGA programming
    3. Copy the log file sobel.log to the folder of the check image script *starting/script/script_py.sh*
    4. Execute the script to show the output image *script_py.sh*
        - CKP4: the script and the log file can be found in *ready/CKP4_sc*

Try It Yourself

Monitor the Accelerator

5. Second Level Monitors Insertion
    1. Open a Xubuntu shell and go into the folder */home/monitoring_cgr/Tutorial/Cluster_prossimo_monitor/starting/mon_generation*
    2. Execute *script.sh* (*source script.sh*)
        1. the previously created Vivado project will be extracted and it will be asked to the user which levels of monitor have to be connected to the accelerator (either second or both second and third)
            1. Select only the second level monitoring
            2. The project will be populated with the selected monitors

Try It Yourself

Monitor the Accelerator

5. Second Level Monitors Insertion
    3. The project will be opended and it is necessary to generate the bitstream with the monitors
        1. Since Vivado is not installed in the Virtual Machine this step will fail, but if Vivado is installed in the adopted machine (e.g. offline), it can be performed
        2. CKP5: the project with the second level monitors can be found in *ready/CKP5_proj*
    4. Repeat the steps of point 2 to set up the XSDK workspace

Try It Yourself

Monitor the Accelerator

5. Second Level Monitors Insertion
    5. Repeat the steps of point 3 to import the *app_src* source files within the workspace
        5. CKP6: the workspace with the created application can be found in *ready/CKP6_ws*
    6. Repeat the steps of point 4 to program the FPGA

Try It Yourself

Monitor the Accelerator

6. Second and Third Level Monitors Insertion
    1. Repeat the steps of point 5, now choosing the monitor at both second and third level (5.2.1.1)
        1. CKP7: the project with the second and third level monitors can be found in *ready/CKP7_proj*
    2. Repeat the steps of point 2 to set up the XSDK workspace

Try It Yourself

Monitor the Accelerator

6. Second and Third Level Monitors Insertion
    3. Repeat the steps of point 3 to import the *app_src* source files within the workspace
        1. CKP8: the workspace with the created application can be found in *ready/CKP8_ws*
    4. Repeat the steps of point 4 to program the FPGA

Acknowledgments

CERBERO H2020 Project

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 (IBM) :

Evgeny Shindin (from 16.08.2019)
EVGENSH@il.ibm.com

Michal Masin (1.01.2017-16.08.2019)

Scientific Coordinator (UniSS):

Francesca Palumbo, fpalumbo@uniss.it

Innovation Manager (Abinsula):

Katiuscia Zedda, katiuscia.zedda@abinsula.com

Dissemination-Communication Manager (USI):

Francesco Regazzoni,
francesco.regazzoni@usi.ch

www.cerbero-h2020.eu

info@cerbero-h2020.eu

@CERBERO_h2020

Acknowledgments

MegaM@RT2 ECSEL Project

EU Commission for funding the ***MegaM@RT2*** (MegaModelling at Runtime) project under grant agreement No 737494.

Coordinator (MDH) :

Gunnar Widforss

gunnar.widforss@mdh.se

Scientific Coordinator (Innopolis University) :

Andrey Sadovykh

a.sadovykh@innopolis.ru

megamart2-ecsel.eu

gunnar.widforss@mdh.se

@megamart2_ecsel

MegaM@Rt²

Acknowledgments

FitOptiVis ECSEL Project

EU Commission for funding the ***FitOptiVis*** (***From the cloud to the edge: smart IntegraTion and OPtimization Technologies for highly efficient Image and VIdeo processing Systems***) project as part of the H2020 Programme under grant agreement No 783162.

Coordinator (PHILIPS) :

Frank Van der Linden
frank.van.der.linden@philips.com

Scientific Coordinator:

Francesca Palumbo **(UniSS)**, fpalumbo@uniss.it

Twan Basten **(TUE)**, A.A.Basten@tue.nl

Zaid Al-Ars **(TUD)**, z.al-ars@tudelft.nl

Innovation Manager (Abinsula):

Katiuscia Zedda, katiuscia.zedda@abinsula.com

https://fitoptivis.eu/

#FitOptiVis

@FitOptiVis

FIT PTI VIS

ECSEL Joint Undertaking

PROSSIMO
Progettazione, sviluppo e ottimizzazione di sistemi intelligenti multi-oggetto

Architetture eterogenee on-chip riconfigurabili: analisi, sviluppo e caratterizzazione del loro comportamento con sistemi di monitoring

Tiziana Fanni[1], Carlo Sau[2], Giacomo Valente[3]

[1]University of Sassari, Intelligent system DEsign and Application (IDEA) Group

[2]University of Cagliari, Diee – Microelectronics and Bioengineering (EOLAB) Group

[3]University of L'Aquila, Disim – Embedded Systems Group