*The Fifteenth International Conference on Advances in Circuits, Electronics and Micro-electronics (**CENICS 2022**)*
October 16, 2022 to October 20, 2022 - Lisbon, Portugal

High-Level Synthesis of Hardware Accelerators for Deconvolution Engines

Cristian Sestito[1], Robert Stewart[3] , and Stefania Perri[2]
[1]*Department of Informatics, Modeling, Electronics and System Engineering, University of Calabria, Rende, Italy*
[2]*Department of Mechanical, Energy and Management Engineering, University of Calabria, Rende, Italy*
[3]*Department of Computer Science, Heriot-Watt University, Edinburgh, United Kingdom*

Presenter: **Cristian Sestito**
cristian.sestito@unical.it

Presenter: Cristian Sestito

Academic Experience

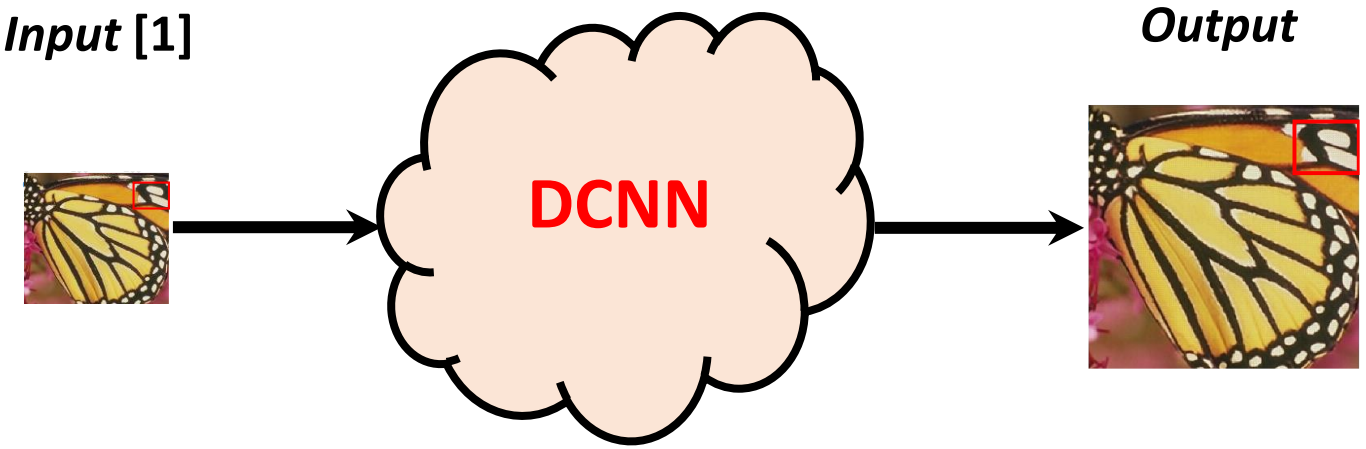
- Student at University of Calabria, Italy:
  - ✓ **PhD student** in Information and Communication Technologies (since November 2019)
  - ✓ **MSc Electronic Engineering** (April 2019)
  - ✓ **BSc Electronic Engineering** (December 2016)
- **Visiting Scholar** at Heriot-Watt University, Edinburgh (2021-2022)

Research interests

- Hardware acceleration of image processing using FPGAs**:**
  - ✓ Deep Learning for image up-sampling
  - ✓ Neural Network compression

Deconvolutions and Applications

Deconvolutions (also known as **Transposed Convolutions**) are mainly used in deep learning



Deconvolutional Neural Networks (**DCNNs**) provide high-resolution output images

Main applications:

- **Semantic Segmentation**: U-Net [2]
- **Object Generation**: DCGAN [3]
- **Super-Resolution Imaging**: FSRCNN [4]

→ **High computational complexity. Need of hardware acceleration**

[1] Set-5 dataset: http://mmlab.ie.cuhk.edu.hk/projects/FSRCNN.html

[2] O. Ronneberger, P. Fischer, T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation". Proc. 18th Int. Conf. Medical Image Computing and Computer-Assisted Intervention (MICCAI 2015), Munich, Germany, Oct. 2015, pp. 234-241.

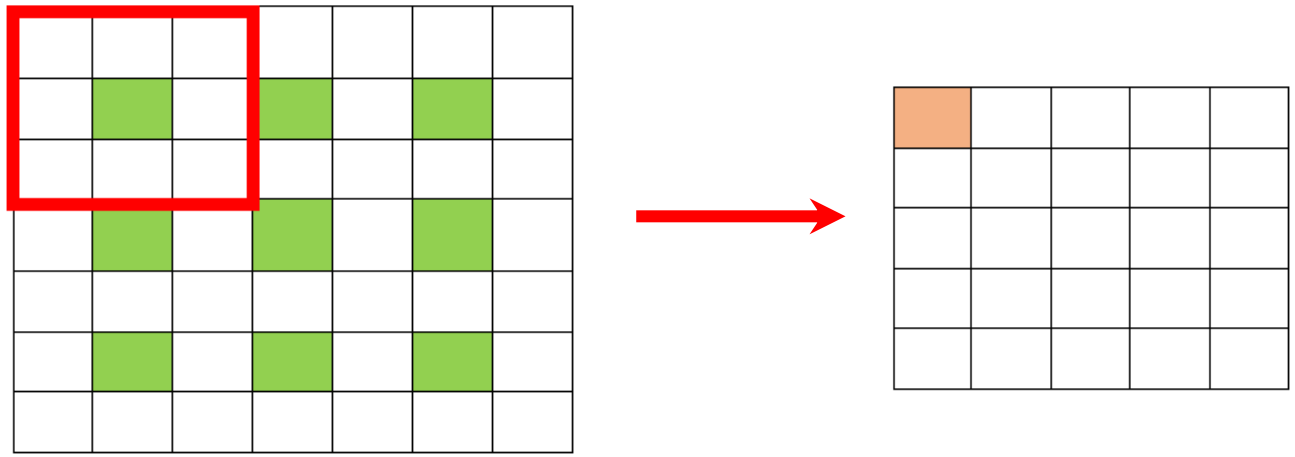
[3] A. Radford. L. Metz and S. Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", in Proc. 4th Int. Conf. on Learning Representations (ICLR 2016), San Juan, Puerto Rico, May 2016.

[4] C. Dong, C.C. Loy, X. Tang, "Accelerating the super-resolution convolutional neural network," in Proc. European Conference on Computer Vision, Amsterdam, The Netherlands, 8–16 October 2016.

Deconvolutions and Algorithmic Strategies

Conventional method [5]

- Input image dilation by inserting zeros
- Then, conventional convolutions through Multiply-Accumulations (MACs)



Pros

- Simplicity

Cons

- Redundant computations. Waste of resources
- High latency

[5] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," 12 January 2018. [Online]. Available: https://arxiv.org/pdf/1603.07285.pdf.

Deconvolutions and Algorithmic Strategies

Input-Oriented Mapping [6]

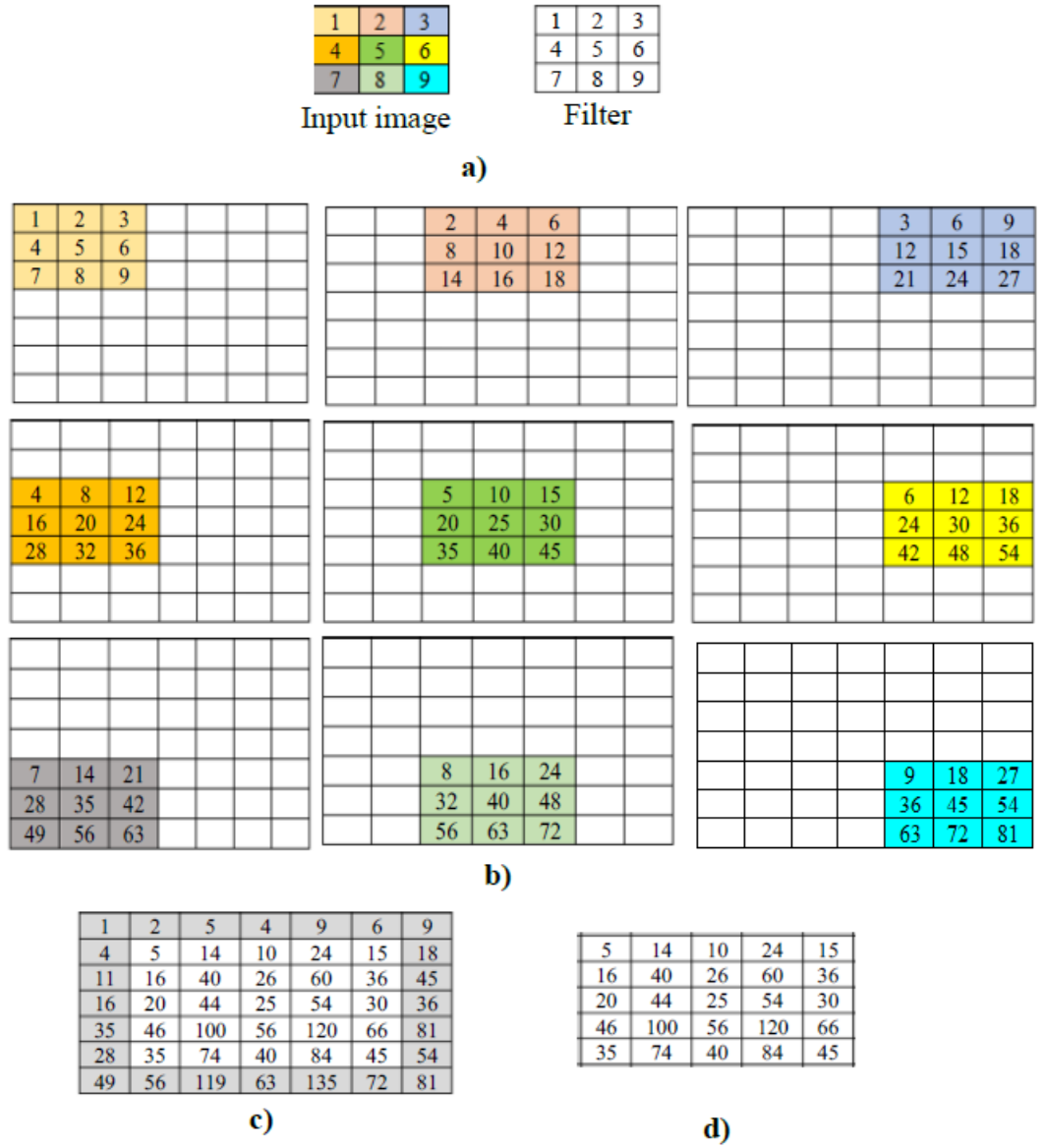
- Multiply each pixel by the filter
- Sum up overlapping regions
- Crop borders

Pros

- No redundant computations
- Multiple output pixels provided in parallel

Cons

- More complex algorithm to be carefully managed in hardware



[6] D. Wang, J. Shen, M. Wen, and C. Zhang, “Efficient Implementation of 2D and 3D Sparse Deconvolutional Neural Networks with a Uniform Architecture on FPGAs,” Electronics, vol. 8, no. 7, pp. 1–13, 2019.

Deconvolutions and FPGAs

Field-Programmable Gate Arrays (FPGAs)

- Flexible
- High power-efficiency
- Rapid prototyping

High-Level Synthesis (HLS)

- C/C++ routines into RTL abstraction
- Limited realization time
- Good for high-level designers
- Platform-independent architectures

The Synthesizable C++ Deconvolution Engine

Deconvolution design

- C++ routine
- Xilinx VIVADO HLS 2019.2

Steps

- Load the current filter
- Multiply each pixel by the filter
- Sum up overlapping regions
- Save pixels in memory

```
1: for (unsigned int i = 0; i < k; i++) {
2:     for (unsigned int j = 0; j < k; j++) {
3:         #pragma HLS PIPELINE II=1
4:         filt[i][j]=filter.read();
5: for (unsigned int r = 0; r < H; r++) {
6:     for (unsigned int c = 0; c < W; c++) {
7:         #pragma HLS PIPELINE II=1
8:         Pix=InIm.read();
9:         for (unsigned int i = 0; i < k; i++) {
10:            for (unsigned int j = 0; j < k; j++) {
11:                // Multiply the generic pixel by the filter
12:              Prods[i][j]=Pix*filt[i][j];
13:              // Store the products to be reused for the column overlap
14:              if (j >= S)
15:                CBuff[i][j-S]=Prods[i][j];
16               // Sum up overlapped columns
17:              if (j<k-S)
18:                 if (c==0)
19:                     SumCol [i][j]=Prods [i][j];
20:                 else SumCol[i][j]=Prods [i][j]+CBuff[i][j];
21:              else SumCol[i][j]=Prods[i][j];
22:              // Store the results to be reused for the row overlap
23:              if (i>=S) {
24:                 if (j<S)
25:                   RBuff[i-S][j][c]=SumCol[i][j];
26:              }
27:              //Sum up overlapping rows
28:              if (i < k-S) {
29:                 if (j < S) {
30:                    if (r == 0)
31:                        SumRow[i][j]=SumCol [i][j];
32:                    else SumRow[i][j]=SumCol[i][j]+RBuff[i][j][c];
33:                 }
34:              }
35:              else SumRow [i][j]=SumCol [i][j];
36:              // Map the results to the output space
37:              for (unsigned int i = 0; i < S; i++) {
38:                  for (unsigned int j = 0; j < S; j++) {
39:                    OBuff[c+i*W+r*S*H].range(16*j+15,16*j)=SumRow[i][j];
40:                  }
41:              }
42:          }
43:      }
44:  }
45: }
```

The Synthesizable C++ Deconvolution Engine

Deconvolution design

- C++ routine
- Xilinx VIVADO HLS 2019.2

Steps

- **Load the current filter**
- **Multiply each pixel by the filter**
- Sum up overlapping regions
- Save pixels in memory

```
for (unsigned int i = 0; i < k; i++) {
    for (unsigned int j = 0; j < k; j++) {
        #pragma HLS PIPELINE II=1
        filt[i][j]=filter.read();
for (unsigned int r = 0; r < H; r++) {
    for (unsigned int c = 0; c < W; c++) {
        #pragma HLS PIPELINE II=1
        Pix=InIm.read();
        for (unsigned int i = 0; i < k; i++) {
            for (unsigned int j = 0; j < k; j++) {
                // Multiply the generic pixel by the filter
                Prods[i][j]=Pix*filt[i][j];
```

#pragma HLS PIPELINE ensures a new input is read each *II* clock cycles, with *II* being the Initiation Interval

The Synthesizable C++ Deconvolution Engine

Deconvolution design

- C++ routine
- Xilinx VIVADO HLS 2019.2

Steps

- Load the current filter
- Multiply each pixel by the filter
- **Sum up overlapping regions**
- Save pixels in memory

```
// Store the products to be reused for the column overlap
if (j >= S)
  CBuff[i][j–S]=Prods[i][j];
 // Sum up overlapped columns
if (j<k–S)
   if (c==0)
       SumCol [i][j]=Prods [i][j];
   else SumCol[i][j]=Prods [i][j]+CBuff[i][j];
else SumCol[i][j]=Prods[i][j];
// Store the results to be reused for the row overlap
if (i>=S) {
   if (j<S)
     RBuff[i-S][j][c]=SumCol[i][j];
}
//Sum up overlapping rows
if (i < k-S) {
   if (j < S) {
      if (r == 0)
          SumRow[i][j]=SumCol [i][j];
      else SumRow[i][j]=SumCol[i][j]+RBuff[i][j][c];
   }
}
else SumRow [i][j]=SumCol [i][j];
```

The Synthesizable C++ Deconvolution Engine

Deconvolution design

- C++ routine
- Xilinx VIVADO HLS 2019.2

Steps

- Load the current filter
- Multiply each pixel by the filter
- Sum up overlapping regions
- **Save pixels in memory**

```
// Map the results to the output space
for (unsigned int i = 0; i < S; i++) {
    for (unsigned int j = 0; j < S; j++) {
      OBuff[c+i*W+r*S*H].range(16*j+15,16*j)=SumRow[i][j];
```

Proper re-organization of pixels considering that multiple outputs, belonging to different positions, are computed in parallel

Post HLS Synthesis Results

Set-up

- Inputs:
  - ✓ 8-bit unsigned images
  - ✓ 8-bit signed filters

Resource Occupation

- Look-Up Tables (LUTs)
- Flip-Flops (FFs)
- Block RAMs (BRAMs)

Speed performance

- Minimum clock period (T*clk*)
- Frames-per-Second (*fps*)

Chip		XC7Z020-clg484-1					
k	*S*	*H*×*W*, *Ho*×*Wo*	*Tclk* [ns]	*fps*	#BRAMs	#LUTs	#FFs
3	2	64×64, 128×128	7.81	4878	18	1648	741
		128×128, 256×256	7.81	1219	66	1674	756
		256×256, 512×512	7.81	304	258	1729	771
5	2	64×64, 128×128	7.81	4878	20	2256	1105
		128×128, 256×256	7.81	1219	68	2282	1122
		256×256, 512×512	7.81	304	260	2307	1139
5	4	64×64, 256×256	7.85	840	68	2862	795
		128×128, 512×512	7.85	210	260	2887	817
Chip		XC7VX980tffg1930-1					
5	4	256×256, 1024×1024	8.24	53	1028	2917	641
7	4	256×256, 1024×1024	8.01	37	1036	5132	1230

Parametric C++ template

- *k* is the filter size
- *S* is the up-sampling factor (or stride)
- *H*×*W* (H_O×W_O) are the sizes of input (output) images

Post HLS Synthesis Results

Resource Occupation

- Look-Up Tables (LUTs)
- Flip-Flops (FFs)
- Block RAMs (BRAMs)

BRAMs usage depends on *S* and *H×W*

Chip		XC7Z020-clg484-1					
k	***S***	***H×W, H_O×W_O***	***Tclk* [ns]**	***fps***	**#BRAMs**	**#LUTs**	**#FFs**
3	2	64×64, 128×128	7.81	4878	18	1648	741
		128×128, 256×256	7.81	1219	66	1674	756
		256×256, 512×512	7.81	304	258	1729	771
5	2	64×64, 128×128	7.81	4878	20	2256	1105
		128×128, 256×256	7.81	1219	68	2282	1122
		256×256, 512×512	7.81	304	260	2307	1139
5	4	64×64, 256×256	7.85	840	68	2862	795
		128×128, 512×512	7.85	210	260	2887	817
Chip		**XC7VX980tffg1930-1**					
5	4	256×256, 1024×1024	8.24	53	1028	2917	641
7	4	256×256, 1024×1024	8.01	37	1036	5132	1230

Parametric C++ template

- *k* is the filter size
- *S* is the up-sampling factor (or stride)
- *H×W* (H_O×W_O) are the sizes of input (output) images

Post HLS Synthesis Results

Resource Occupation

- Look-Up Tables (LUTs)
- Flip-Flops (FFs)
- Block RAMs (BRAMs)

BRAMs usage depends on *S* and *H*×*W*

LUTs/FFs usage depends on *k* mainly

Chip		XC7Z020-clg484-1					
k	***S***	***H*×*W*, *H*$_O$×*W*$_O$**	***Tclk* [ns]**	***fps***	**#BRAMs**	**#LUTs**	**#FFs**
3	2	64×64, 128×128	7.81	4878	18	1648	741
		128×128, 256×256	7.81	1219	66	1674	756
		256×256, 512×512	7.81	304	258	1729	771
5	2	64×64, 128×128	7.81	4878	20	2256	1105
		128×128, 256×256	7.81	1219	68	2282	1122
		256×256, 512×512	7.81	304	260	2307	1139
5	4	64×64, 256×256	7.85	840	68	2862	795
		128×128, 512×512	7.85	210	260	2887	817
Chip		XC7VX980tffg1930-1					
5	4	256×256, 1024×1024	8.24	53	1028	2917	641
7	4	256×256, 1024×1024	8.01	37	1036	5132	1230

Parametric C++ template

- *k* is the filter size
- *S* is the up-sampling factor (or stride)
- *H*×*W* (*H*$_O$×*W*$_O$) are the sizes of input (output) images

Post HLS Synthesis Results

Speed performance

- Frames-per-Second (*fps*)
- It mainly depends on the image size

Chip		XC7Z020-clg484-1					
k	*S*	$H\times W, H_O\times W_O$	*Tclk* [ns]	*fps*	#BRAMs	#LUTs	#FFs
3	2	64×64, 128×128	7.81	4878	18	1648	741
		128×128, 256×256	7.81	1219	66	1674	756
		256×256, 512×512	7.81	304	258	1729	771
5	2	64×64, 128×128	7.81	4878	20	2256	1105
		128×128, 256×256	7.81	1219	68	2282	1122
		256×256, 512×512	7.81	304	260	2307	1139
5	4	64×64, 256×256	7.85	840	68	2862	795
		128×128, 512×512	7.85	210	260	2887	817
Chip		XC7VX980tffg1930-1					
5	4	256×256, 1024×1024	8.24	53	1028	2917	641
7	4	256×256, 1024×1024	8.01	37	1036	5132	1230

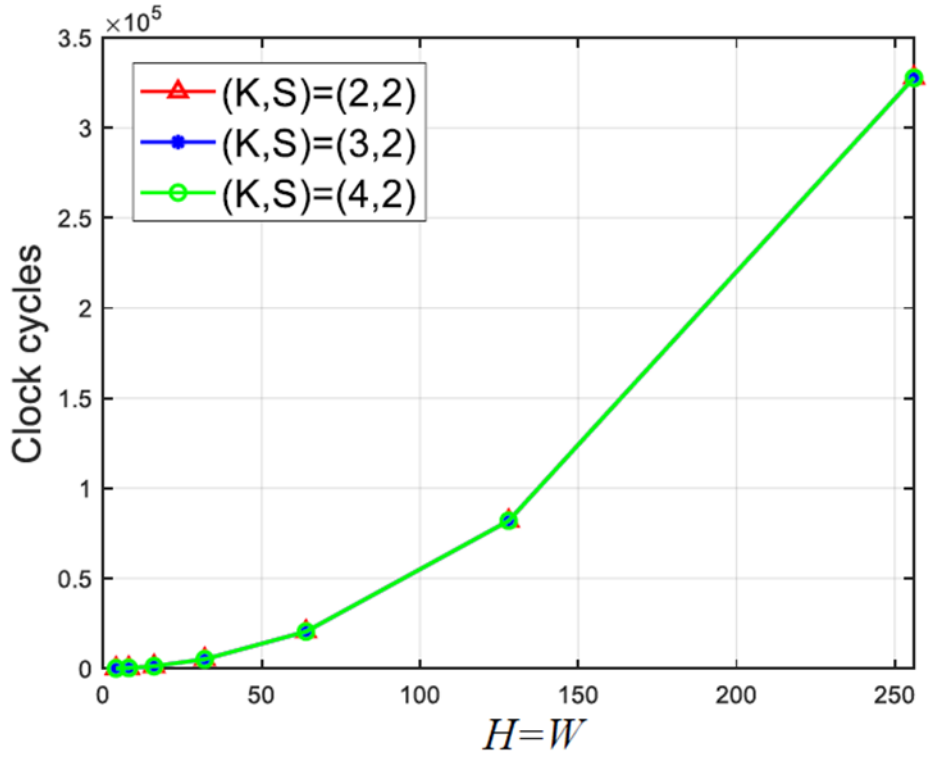
Parametric C++ template

- *k* is the filter size
- *S* is the up-sampling factor (or stride)
- $H\times W$ ($H_O\times W_O$) are the sizes of input (output) images

Post HLS Synthesis Results

Speed performance

- Clock cycles



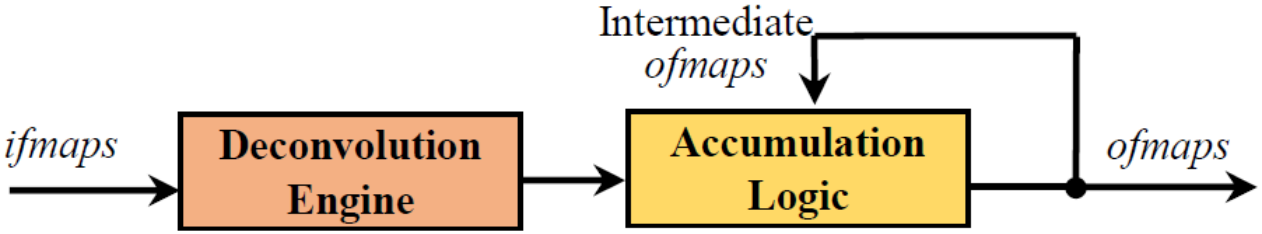
Parametric C++ template

- k is the filter size
- S is the up-sampling factor (or stride)
- $H \times W$ ($H_O \times W_O$) are the sizes of input (output) images

Work in Progress

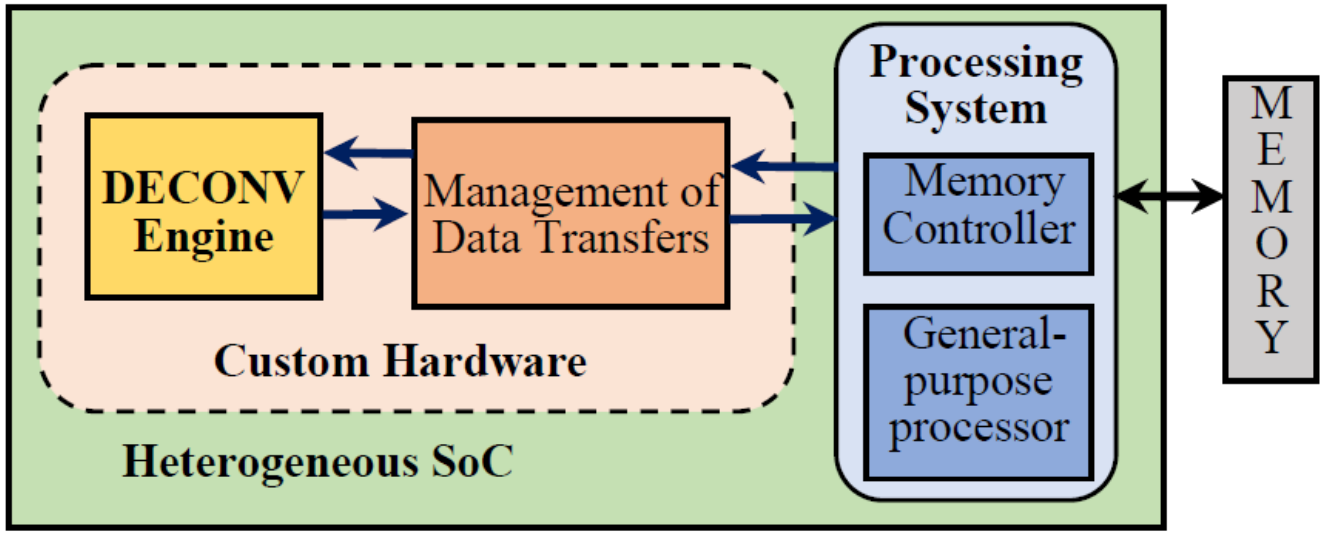
Deconvolution Engine Extension

- Multiple input and output images in parallel
- Streaming interface



FPGA-based embedded system

- Unit to manage data transfer from/to an external memory
- Use of a GP Processor to control and configure all the hardware units
- Real-time scenarios emulation



*The Fifteenth International Conference on Advances in Circuits, Electronics and Micro-electronics (**CENICS 2022**)*

October 16, 2022 to October 20, 2022 - Lisbon, Portugal

High-Level Synthesis of Hardware Accelerators for Deconvolution Engines

Cristian Sestito[1], Robert Stewart[3] , and Stefania Perri[2]

[1]*Department of Informatics, Modeling, Electronics and System Engineering, University of Calabria, Rende, Italy*

[2]*Department of Mechanical, Energy and Management Engineering, University of Calabria, Rende, Italy*

[3]*Department of Computer Science, Heriot-Watt University, Edinburgh, United Kingdom*