Tri de Données générique et paramétrable en C

Laboratoire Leprince Ringuet
LLR Polytechnique IN2P3/CNRS

L. Pacheco, T. Romanteau, J. B. Sauvan

LLR Ecole Polytechnique
F - 91128 PALAISEAU Cedex

12/10/2021

Introduction et Contexte

Ce qui est décrit ici illustre une réalisation concrète dans le cadre de l'expérience CMS basée au CERN et plus précisément pour le système de trigger du project « High Granulometry CALorimeter »

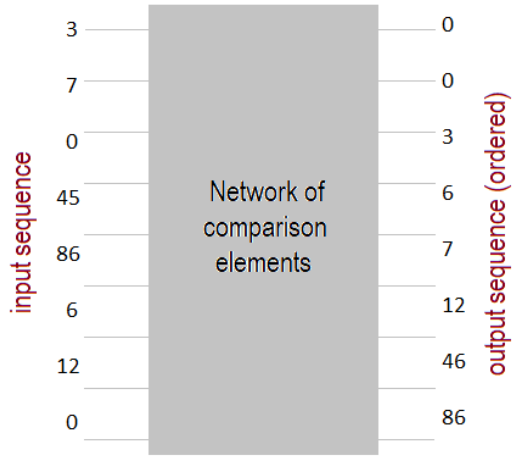
- ***Objectif:*** implémentation, en électronique numérique, d'un algorithme pour sélectionner les cellules trigger élémentaires dans la partie Front End du calorimètre HGCAL
  - Algorithme développé au LLR dit « Best Choice » et basé sur du tri, qui cohabite avec deux autres algorithmes nommés « Threshold » et « Super-TC »
  - L'ensemble est Implanté physiquement dans d'un ASIC Front End « ECON-T » développé au FERMILAB (USA)
  - Sélectionner et trier par valeur d'énergie les N cellules de trigger (ou régions d'intérêt) les plus énergétiques et ceci à chaque collision LHC

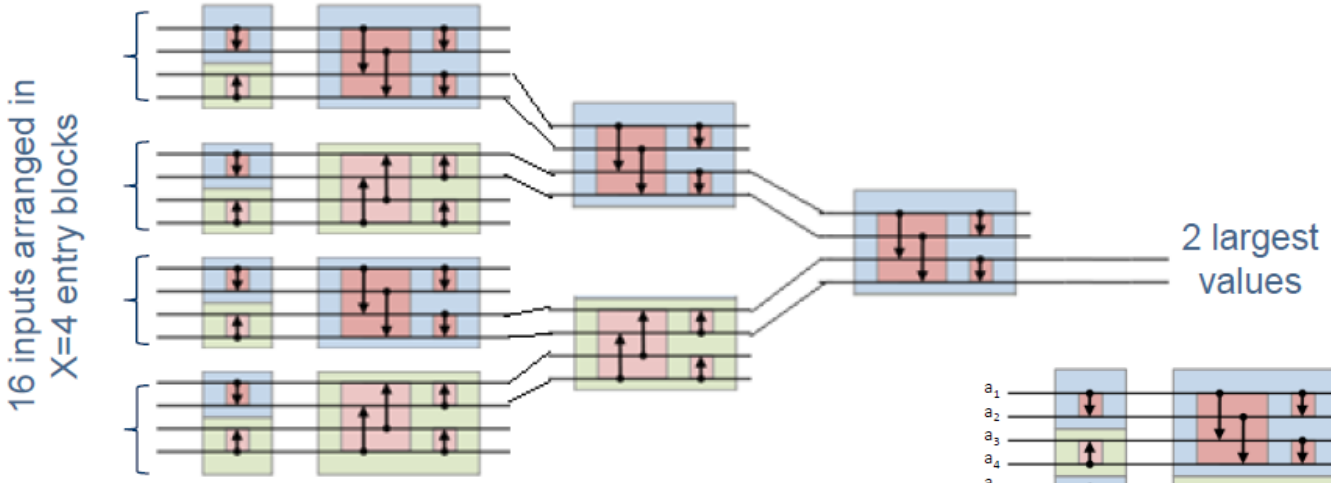
- ***Contraintes électroniques:*** *48 « Trigger Cell » en entrée, choix d'horloge : multiple de Clk-LHC, latence, interface de réception et de sortie des données, surface occupée et puissance dissipée..*

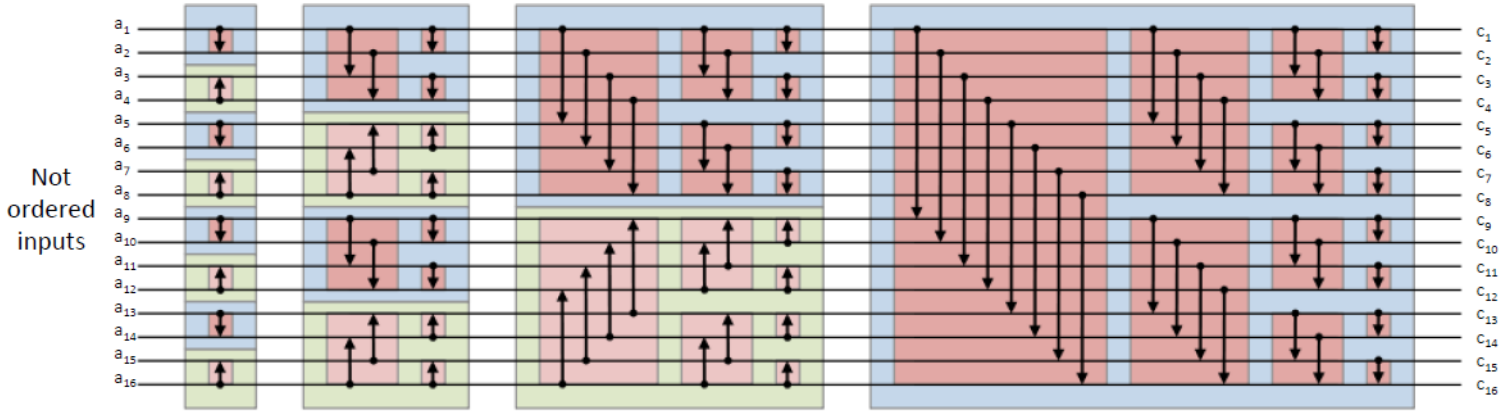
Pré-étude, évaluations

- Recherche de solutions algorithmiques connues et besoins plus spécifiques
  - ***Algorithme de tri :*** Bitonic sorter, Odd-even merge sort, etc..
  - ***Optimisation matérielle :*** Nombre de comparateurs → *ressources*, nombre d'étage → *latence*
  - ***Adaptabilité au besoin :*** Nombre d'entrées paramétrable ($N \neq 2^n$)

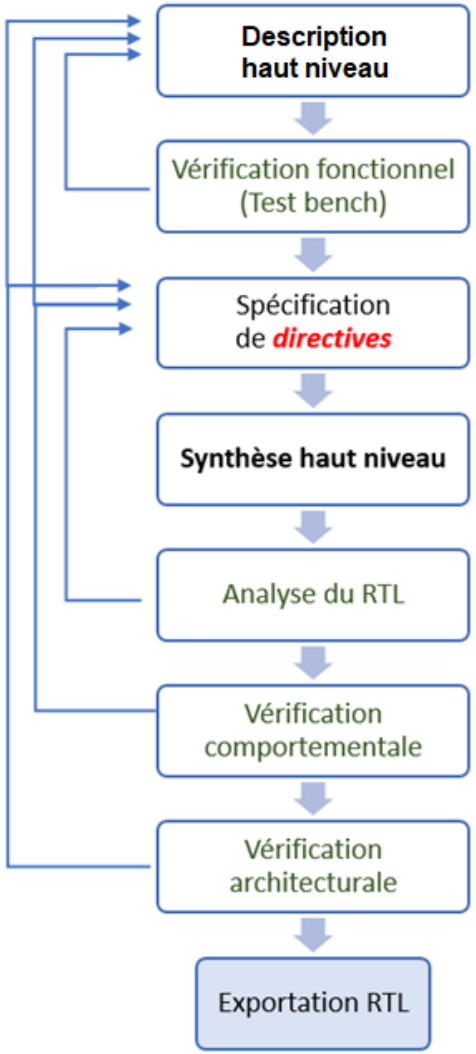
Sélection des M valeurs les plus grandes



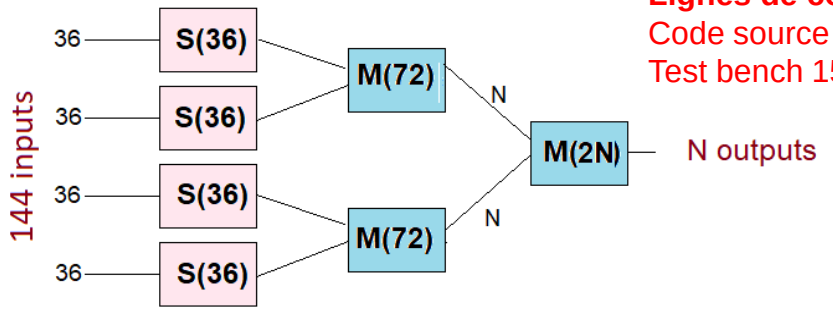




Vivado_HLS Flux de Conception



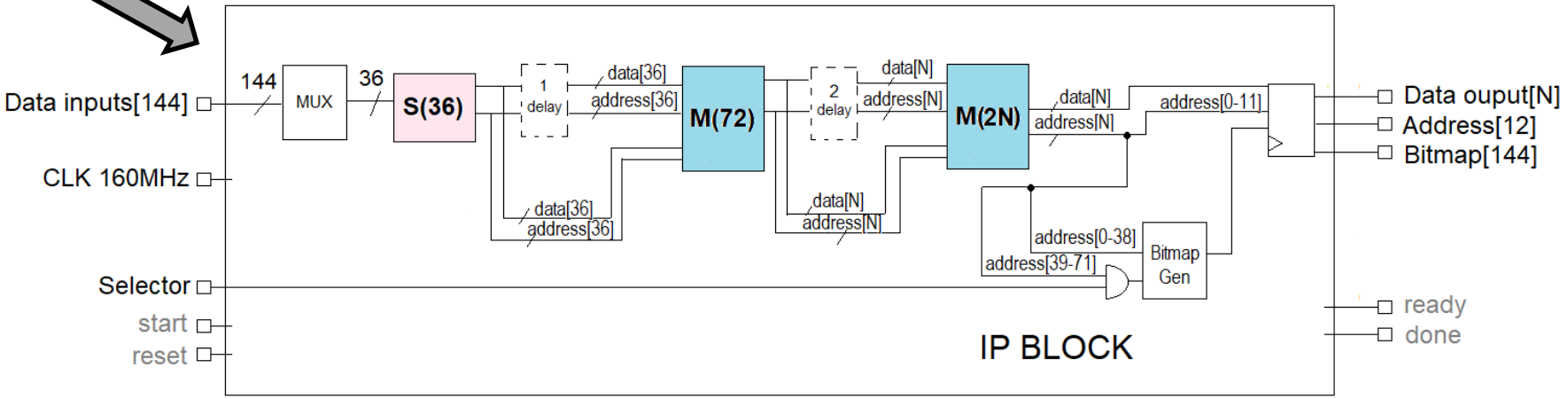
Représentation Algorithmique



Lignes de code C:
Code source 439
Test bench 151

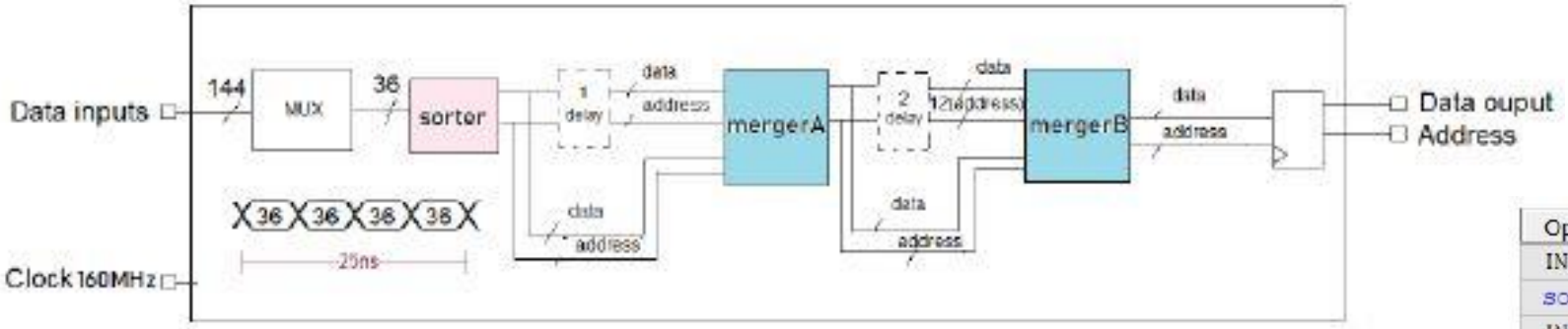
Représentation RTL

Lignes de code VHDL générées
18 296



Solution Retenue

- *Une implémentation efficace est basée sur les algorithmes « bitonic » et « odd/even »*
  - Ils sont par nature récursifs et sont transposés en code par les instructions « while, for, do while loop »
  - Nous voulions coder en langage C et prototyper avec Vivado_HLS (Xilinx), mais la récursivité n'est pas supportée par Vivado_HLS
  - Un moyen efficace de contournement a été d'utiliser l'algorithme de « Kenneth Batcher » : Batcher odd-even mergesort, celui-ci n'est pas récursif et générique (support d'un nombre d'entrée quelconque)

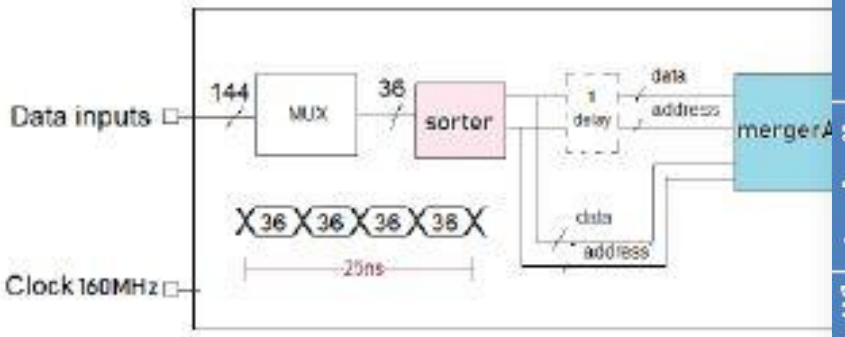


Operation\Control Step	C0	C1	C2	C3
IN_energies_0_35 (read)				
sorter(function)				
IN_energies_36_71 (read)				
sorter(function)				
mergerA(function)				
IN_energies_72_107 (read)				
sorter(function)				
IN_energies_108_143(read)				
sorter(function)				
mergerA(function)				
mergerB(function)				
OUT_energies_0_143(write)				

- *La solution dédiée Asic a été produite avec Stratus HLS (Cadence)*
  - Langage SystemC, code réécrit (J.B. Sauvan)
  - Code compatible Stratus / Vivado_HLS avec préprocesseurs / pragmas directives
  - 48 entrées, 48 sorties, horloge à 40 Mhz (Clock LHC)
  - Techno digitale 0,65 nm, partiellement tripliquée, librairie Rad-hard RD53 CERN
  - Solution produite et simulée en collaboration avec l'Université de SPLIT - FESB

Solution Retenue

- *Une implémentation efficace est basée su*
  - Ils sont par nature récursifs et sont transpos
  - Nous voulions coder en langage C et protot
  - Un moyen efficace de contournement a été pas récursif et générique (support d'un nom



- *La solution dédiée Asic a été produite ave*
  - Langage SystemC, code réécrit (J.B. Sauva
  - Code compatible Stratus / Vivado_HLS ave
  - 48 entrées, 48 sorties, horloge à 40 Mhz (C
  - Techno digitale 0,65 nm, partiellement tripli
  - Solution produite et simulée en collaboratio

RD53 libs		100 Mrad TID		200 Mrad TID		500 Mrad TID	
		Initial	Post-synth	Initial	Post-synth	Initial	Post-synth
Gates	Sequential	2,307	2,233	2,307	2,233	2,307	2,233
	Inverter	13,310	5,565	13,310	5,607	13,310	8,131
	Buffer	0	429	0	490	0	1,176
	Logic	48,251	31,387	48,251	33,267	48,251	41,141
	Total	**63,868**	**39,614**	**63,868**	**41,597**	**63,868**	**52,681**
Area [µm²]	Net	1,118	69,099	1,118	72,559	1,118	87,468
	Cell	261,151	128,675	261,151	138,247	261,151	202,951
	Total	**262,269**	**197,774**	**262,269**	**210,806**	**262,269**	**290,419**
Power [mW]	Dynamic	N/A	24.219	N/A	25.78	N/A	38.123
	Leakage	N/A	0.001	N/A	0.001	N/A	0.004
	Total	**N/A**	**24.22**	**N/A**	**25.78**	**N/A**	**38.127**
Critical path [ps]	Input to Reg	13,199.2	0.0	10,390.2	0.0	5,958.7	0.0
	Reg to Output	24,574.4	24,629.4	24,486.1	24,543.5	24,317.8	24,330.5
	Reg to Reg	11,616.5	0.0	8,427.6	0.0	3,394.7	0.0
	Input to Output	N/A	N/A	N/A	N/A	N/A	N/A
Synthesis time		28:00 (1,680 seconds)		28:00 (1,680 seconds)		35:20 (2,120 seconds)	

Prototypage avec Vivado_HLS

- *Avoir un design Paramétrable est vite apparu d'un grand intérêt*
  - Ecrit en code C / SystemC, puis synthétisé à l'aide de l'outil Vivado_HLS
  - Le nombre d'entrées, de sorties, la taille des vecteurs sont paramétrables
  - Niveau de pipeline, instances des blocks internes sont paramétrables
  - RTL ou IP niveau porte peuvent être produit via Vivado en mode batch depuis Vivado_HLS
  - Le même test bench en C est utilisable à tous les niveaux (C, RTL, niveau porte) grâce à la génération automatique d'un wrapper
  - Un repository dédié contenant tous les IP de tri de taille différentes peut être créé et importez dans Vivado

```
/*****************************************************************************/
// Types, Constants
//*****************************************************************************/

#define N 32 //48 //32 //48      //number of total elements to sort
#define NS 8 //12 //8 //12  //number of elements to sort sequentially (block sorter input size)
#define NMA 16 //24 //16 //24   //number of elements to merge in block merger a (block merger a input size)
#define NMB 16 //24 //16 //48   //number of elements to merge in block merger b (block merger b input size)
#define M 16 //24 //16 //48     //number of elements to select

//7 bits per TC
//6 bits for address
typedef uint7 datain_t[N];          //Array type for containing total elements to sort
typedef uint7 datasorter_t[NS];     //Array type for containing elements to process in sorter
typedef uint7 datamergera_t[NMA];   //Array type for containing elements to process in merger a
typedef uint7 datamergerb_t[NMB];   //Array type for containing elements to process in merger b
typedef uint7 dataout_t[M];         //Array type for containing elements selected
```

Prototypage avec Vivado_HLS

- *Avoir un design Paramétrable est vite apparu d'un grand intérêt*
  - Ecrit en code C / SystemC, puis synthétisé à l'aide de l'outil Vivado_HLS
  - Le nombre d'entrées, de sorties, la taille des vecteurs sont paramétrables
  - Niveau de pipeline, instances des blocks internes sont paramétrables
  - RTL ou IP niveau porte peuvent être produit via Vivado en mode batch depuis Vivado_HLS
  - Le même test bench en C est utilisable à tous les niveaux (C, RTL, niveau porte) grâce à la génération de automatique d'un wrapper
  - Un repository dédié contenant tous les IP de tri de taille différentes peut être créé et importez dans Vivado

```
//-----------------------------------------------------------------------------
void sorting_network(datain_t arr_input, dataout_t arr_output, adressout_t arr_adresses){
#pragma HLS INTERFACE ap_ctrl_none port=return
#pragma HLS ALLOCATION instances=sorter limit=1 function
#pragma HLS ALLOCATION instances=mergerA limit=1 function
#pragma HLS ALLOCATION instances=mergerB limit=1 function
#pragma HLS PIPELINE II=4
#pragma HLS ARRAY_PARTITION variable=arr_adresses complete dim=1
#pragma HLS ARRAY_PARTITION variable=arr_output complete dim=1
#pragma HLS ARRAY_PARTITION variable=arr_input complete dim=1

    //Variables declaration
    datasorter_t arr_tmp;
    datamergera_t arr_tmp_so, arr_tmp_so1, arr_tmp_so2;
    datamergerb_t arr_tmp_mo;
    adresssorter_t arr_tmp_adresses;
    adressmergera_t arr_tmp_so_adresses, arr_tmp_so1_adresses, arr_tmp_so2_adresses;
    adressmergerb_t arr_tmp_mo_adresses;

    #pragma HLS ARRAY_PARTITION variable=arr_tmp complete dim=1
    #pragma HLS ARRAY_PARTITION variable=arr_tmp_so2 complete dim=1
    #pragma HLS ARRAY_PARTITION variable=arr_tmp_so1 complete dim=1
    #pragma HLS ARRAY_PARTITION variable=arr_tmp_so complete dim=1
    #pragma HLS ARRAY_PARTITION variable=arr_tmp_mo complete dim=1
```

Prototypage avec Vivado_HLS

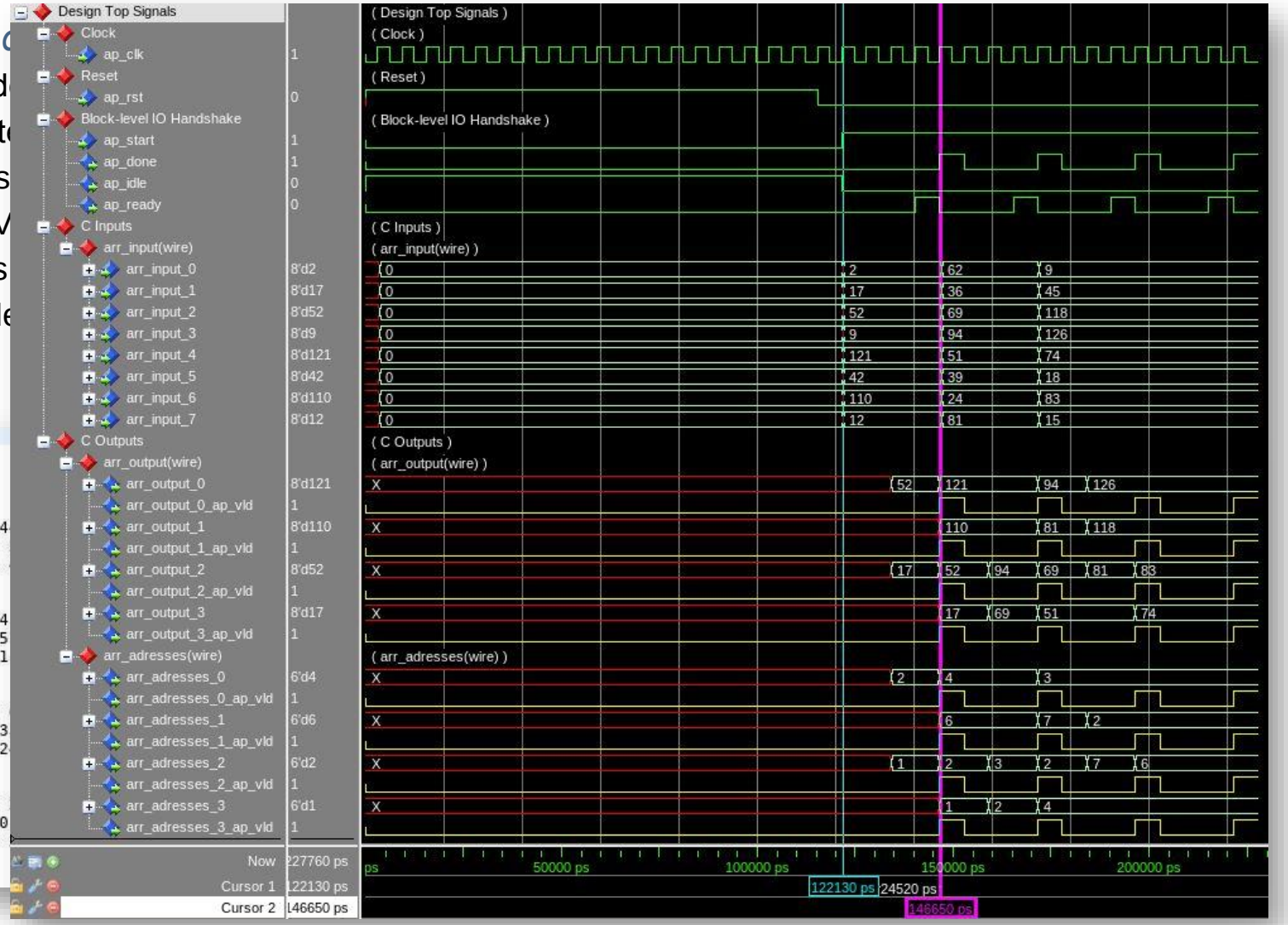
- *Avoir un design Paramétrable est vite apparu d'un grand intérêt*
  - Ecrit en code C / SystemC, puis synthétisé à l'aide de l'outil Vivado_HLS
  - Le nombre d'entrées, de sorties, la taille des vecteurs sont paramétrables
  - Niveau de pipeline, instances des blocks internes sont paramétrables
  - RTL ou IP niveau porte peuvent être produit via Vivado en mode batch depuis Vivado_HLS
  - Le même test bench en C est utilisable à tous les niveaux (C, RTL, niveau porte) grâce à la génération de automatique d'un wrapper
  - Un repository dédié contenant tous les IP de tri de taille différentes peut être créé et importez dans Vivado

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
make: `csim.exe' is up to date.

BX=0
Unsorted list: 525 252 900 54 420 180 46 158 525 1080 1332 680 2800 360 330 1344 0 0 44 81 68 18 44 116 180 0 0 748 222 187 612 377 490 279 310 0 1280 1612 518 32 1440 1628 480 116 140 374 248 225
Sorted list:   2800 1628 1612 1440 1344 1332 1280 1080 900 748 680 612 525 525 518 490 480 420 377 374 360 330 310 279 252 248 225 222 187 180 180 158 140 116 116 81 68 54 46 44 44 32 18 0 0 0 0 0
Adresses of sorted list:   12 41 37 40 15 10 36 9 2 27 11 30 0 8 38 32 42 4 31 45 13 14 34 33 1 46 47 28 29 5 24 7 44 23 43 19 20 3 6 18 22 39 21 26 16 25 17 35

BX=1
Unsorted list: 245 0 210 360 1260 165 434 910 1260 165 166 952 910 1080 248 224 88 60 420 0 600 360 44 0 150 310 1224 42 74 128 85 29 560 93 232 962 144 496 555 512 1728 185 320 290 202 340 1116 4725
Sorted list:   4725 1728 1260 1260 1224 1116 1080 962 952 910 910 600 560 555 512 496 434 420 360 360 340 320 310 290 248 245 232 224 210 202 185 166 165 165 150 144 128 93 88 85 74 60 44 42 29 0 0 0
Adresses of sorted list:   47 40 8 4 26 46 13 35 11 12 7 20 32 38 39 37 6 18 21 3 45 42 25 43 14 0 34 15 2 44 41 10 9 5 24 36 29 33 16 30 28 17 22 27 31 1 19 23

BX=2
Unsorted list: 525 364 38 234 192 720 85 350 1960 195 74 374 61 504 107 364 48 60 158 72 60 0 525 170 195 248 340 612 166 0 612 102 770 116 310 102 120 93 333 176 162 130 160 1508 0 0 116 1495
Sorted list:   1960 1508 1495 770 720 612 612 525 525 504 374 364 364 350 340 333 310 248 234 195 195 192 176 170 166 162 160 158 130 120 116 116 107 102 102 93 85 74 72 61 60 60 48 38 0 0 0 0
Adresses of sorted list:   8 43 47 32 5 30 27 0 22 13 11 1 15 7 26 38 34 25 3 24 9 4 39 23 28 40 42 18 41 36 33 46 14 35 31 37 6 10 19 12 17 20 16 2 44 29 21 45

BX=3
Unsorted list: 560 98 300 198 455 540 1984 192 1120 165 1776 1088 490 1440 726 28 64 38 1540 1440 135 198 385 372 75 434 60 76 102 408 128 44 280 496 279 83 320 70 240 208 396 56 0 130 78 1632 341 1835
Sorted list:   1984 1835 1776 1632 1540 1440 1440 1120 1088 726 560 540 496 490 455 434 408 396 385 372 341 320 300 280 279 240 208 198 198 192 165 135 130 128 102 98 83 78 76 75 70 64 60 56 44 38 28 0
Adresses of sorted list:   6 47 10 45 18 19 13 8 11 14 0 5 33 12 4 25 29 40 22 23 46 36 2 32 34 38 39 21 3 7 9 20 43 30 28 1 35 44 27 24 37 16 26 41 31 17 15 42
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

Prototypage avec Vivado_HLS

- *Avoir un design Paramétrable est vite apparu c*
  - Ecrit en code C / SystemC, puis synthétisé à l'aid
  - Le nombre d'entrées, de sorties, la taille des vecte
  - Niveau de pipeline, instances des blocks internes
  - RTL ou IP niveau porte peuvent être produit via V
  - Le même test bench en C est utilisable à tous les
  - Un repository dédié contenant tous les IP de tri de

```
1 INFO: [SIM 2] *************** CSIM start ***************
2 INFO: [SIM 4] CSIM will launch GCC as the compiler.
3 make: `csim.exe' is up to date.
4
5 BX=0
6 Unsorted list: 525 252 900 54 420 180 46 158 525 1080 1332 680 2800 360 330 134
7 Sorted list:   2800 1628 1612 1440 1344 1332 1280 1080 900 748 680 612 525 525
8 Adresses of sorted list:   12 41 37 40 15 10 36 9 2 27 11 30 0 8 38 32 42 4 31
9
10 BX=1
11 Unsorted list: 245 0 210 360 1260 165 434 910 1260 165 166 952 910 1080 248 224
12 Sorted list:   4725 1728 1260 1260 1224 1116 1080 962 952 910 910 600 560 555 5
13 Adresses of sorted list:   47 40 8 4 26 46 13 35 11 12 7 20 32 38 39 37 6 18 21
14
15 BX=2
16 Unsorted list: 525 364 38 234 192 720 85 350 1960 195 74 374 61 504 107 364 48
17 Sorted list:   1960 1508 1495 770 720 612 612 525 525 504 374 364 364 350 340 3
18 Adresses of sorted list:   8 43 47 32 5 30 27 0 22 13 11 1 15 7 26 38 34 25 3 2
19
20 BX=3
21 Unsorted list: 560 98 300 198 455 540 1984 192 1120 165 1776 1088 490 1440 726
22 Sorted list:   1984 1835 1776 1632 1540 1440 1440 1120 1088 726 560 540 496 490
23 Adresses of sorted list:   6 47 10 45 18 19 13 8 11 14 0 5 33 12 4 25 29 40 22
24 INFO: [SIM 1] CSim done with 0 errors.
25 INFO: [SIM 3] *************** CSIM finish ***************
```



Prototypage avec Vivado_HLS

- *Avoir un design Paramétrable est vite apparu d'un grand intérêt*
  - Ecrit en code C / SystemC, puis synthétisé à l'aide de l'outil Vivado_HLS
  - Le nombre d'entrées, de sorties, la taille des vecteurs sont paramétrables
  - Niveau de pipeline, instances des blocks internes sont paramétrables
  - RTL ou IP niveau porte peuvent être produit via Vivado en mode batch de
  - Le même test bench en C est utilisable à tous les niveaux (C, RTL, niveau
  - Un repository dédié contenant tous les IP de tri de taille différentes peut êt

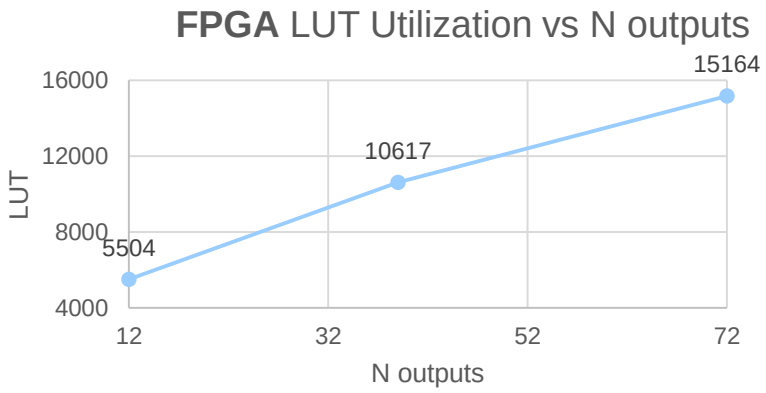
```
11 //
12 #ifndef __SORTER_H__
13 #define __SORTER_H__
14
15 #include <cmath>
16 #include <systemc.h>
17 #include "types.h"
18
19 #ifdef STRATUS
20 #include <stratus_hls.h>
21 #endif
22
23
24 SC_MODULE(Sorter) {
25   public:
26     sc_in<bool> clk;
27     sc_in<bool> reset;
28
29     sc_in<data_t> p_in[input_size];
30     sc_out<data_t> p_out[output_size];
31     sc_out<address_t> p_out_address[output_size];
32
33     SC_CTOR(Sorter) {
34       SC_CTHREAD(process, clk.pos());
35       reset_signal_is(reset, true);
36     }
```

```
12 #include "sorter.h"
13
14 void Sorter::process() {
15 #ifdef STRATUS
16   HLS_FLATTEN_ARRAY(p_in);
17   HLS_FLATTEN_ARRAY(p_out_address);
18   HLS_FLATTEN_ARRAY(p_out);
19 #endif
20 #ifdef VIVADO
21 #pragma HLS ARRAY_PARTITION variable=p_in complete dim=1
22 #pragma HLS ARRAY_PARTITION variable=p_out_address complete dim=1
23 #pragma HLS ARRAY_PARTITION variable=p_out complete dim=1
24 #endif
25   // reset behaviour
26   {
27 #ifdef STRATUS
28     HLS_DEFINE_PROTOCOL("Reset");
29     HLS_UNROLL_LOOP();
30 #endif
31 #ifdef VIVADO
32 #pragma HLS UNROLL
33 #endif
34     RESET_LOOP:for(int i=0; i<output_size; i++) {
35       p_out[i].write(0);
36       p_out_address[i].write(0);
37     }
38     wait();
39   }
40   // main loop
41   while(1) {
42 #ifdef STRATUS
43     HLS_PIPELINE_LOOP("pipeline");
44 #endif
45     input_data_t in;
46     output_data_t out;
47     output_address_t out_address;
48     {
49 #ifdef STRATUS
50       HLS_DEFINE_PROTOCOL("Input_Label");
51 #endif
52       INPUT_LOOP:for(int i=0; i<input_size; i++) {
53         in[i] = p_in[i].read();
```

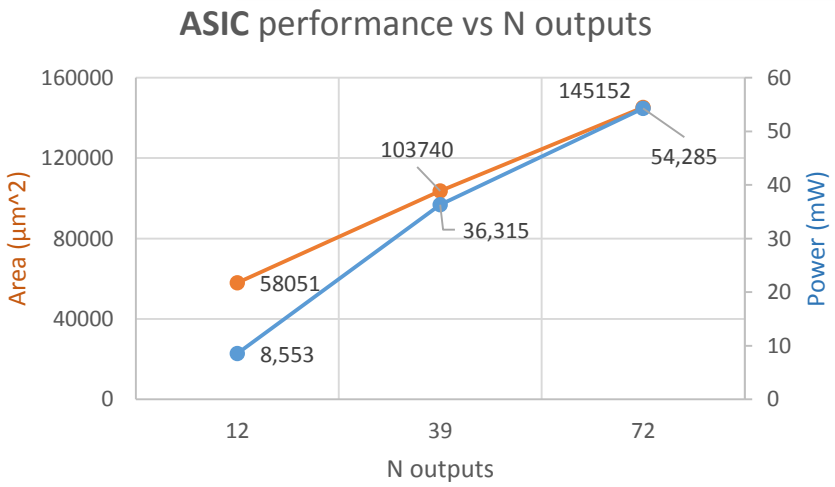
Vivado_HLS, Analyse et Bilan

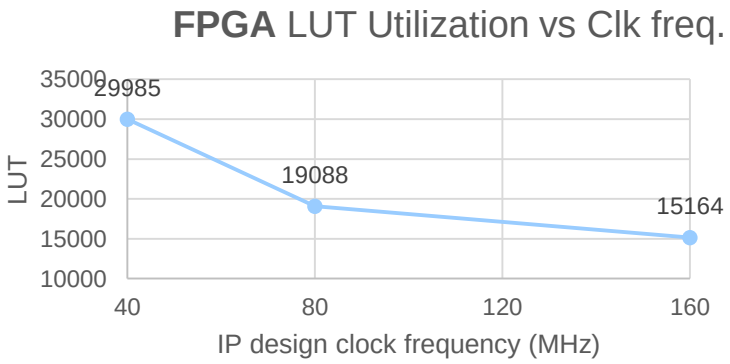
- Synthèse logique FPGA et ASIC du RTL issue de la synthèse de haut niveau

- Performances vs:
  - # entrées
  - # sorties
  - # bits codage
  - Étages de pipeline (latence)
  - Fréquence de d'horloge



20 Solutions produites et testées en 3 mois





Conclusions

- L'évolution des techniques de conception via des langages de haut niveau C, C++, SystemC, Phyton, a atteint un niveau de maturité effectif :
  - Ces langages sont applicable à la conception de FPGA, d'ASIC, de développement IA et d'accélération matériel, etc..
  - La synthèse de haut niveau produits des résultats aussi qualitatif voir meilleurs qu'en RTL
  - Les temps de développement sont largement plus courts et autorisent le tests de solutions variées
  - Les familles de composant FPGA récents (Versal ACAP) obligent à l'utilisation de ces langages
  - Les langages de haut niveau et le niveau d'abstraction associé, permettent d'élargir le nombre d'acteur potentiel (physicien)

- La généricité et le paramétrage d'une conception présente de réels avantages
  - Avec le même code il est possible de générer toute une variété d'une même fonction (Off Detector Stage1 HGCAL Trigger)
  - Un « test bench » est automatiquement adaptable à un design avec des dimensions variées (Vectorisation 3D)

- Les outils existants d'origine différentes sont compatibles entre eux, avec un même langage
  - Le prototypage sur FPGA d'un code SytemC est possible tout en destinant le design à être ciblé sur ASIC

Les langages de conception RTL tendront sûrement à disparaître, il apparait raisonnable de s'y préparer, d'autant que les outils sont disponibles et efficaces!

Les algorithmes sont applicables tant par software qu'en hardware, une même conception débouchera plus souvent sur un mix des deux, ce qu'autorise aisément la conception de haut niveau.