

YLICHRON

Technologies for Computing Systems Design

The HCE: Accessing the FPGA World through ANSI-C Programs

```
OneDDCT(Img[i][0], Img[i][1], Img[i][2], Img[i][3], Img[i][4], Img[i][5], Img[i][6], Img[i][7],  
        IImg[i][0], IImg[i][1], IImg[i][2], IImg[i][3], IImg[i][4], IImg[i][5], IImg[i][6], IImg[i][7]);  
  
TransposeMatrix(IImg);  
  
/**HWST split*/  
for(i=0; i<8; i++)  
    OneDDCT(IImg  
            TImg
```

Paolo Palazzari
Ylichron Srl

Outline of presentation

- Why using the ANSI C
- What is HCE
- How HCE works
- Example of HCE use
- The DRC co-processing boards
- HCE performances
- Conclusions

Outline of presentation

- Why using the ANSI C
- What is HCE
- How HCE works
- Example of HCE use
- The DRC co-processing boards
- HCE performances
- Conclusions

Why an ANSI-C compiler for FPGA?

- Don't need HW-engineering knowledge
- Don't request specific expression of parallelism
- Don't need to learn a new language/dialect
- May use any C compiling environment to test and debug the application
- May port your codes under any new computing technology supporting ANSI-C (for sure a proprietary language will not be supported)

Is ANSI-C adequate to express algorithms?

- ANSI-C can represent any untimed algorithm and is widespread used to express scientific algorithms.
- To efficiently support not standard data types it can be useful to extend the ANSI-C with data types defined by the SystemC standard: HCE supports the **sc_fixed** and the **sc_bv** data types.

Outline of presentation

- Why using the ANSI C
- **What is HCE**
- How HCE works
- Example of HCE use
- The DRC co-processing boards
- HCE performances
- Conclusions

What is HCE

- The HARWEST Compiling Environment (HCE) is a C to VHDL optimizing and parallelizing compiler
- It is the first outcome of the HARWEST research project, funded by the Italian Ministry for University and Research, aimed at creating a fully automated HW/SW co-design environment.

What is HCE

- The HARWEST Compiling Environment (HCE) is a C to VHDL optimizing and parallelizing compiler
- It is the first outcome of the HARWEST research project, funded by the Italian Ministry for University and Research, aimed at creating a fully automated HW/SW co-design environment.
- Let us see the **HCE Design Flow**

Input specs
(ANSI C)

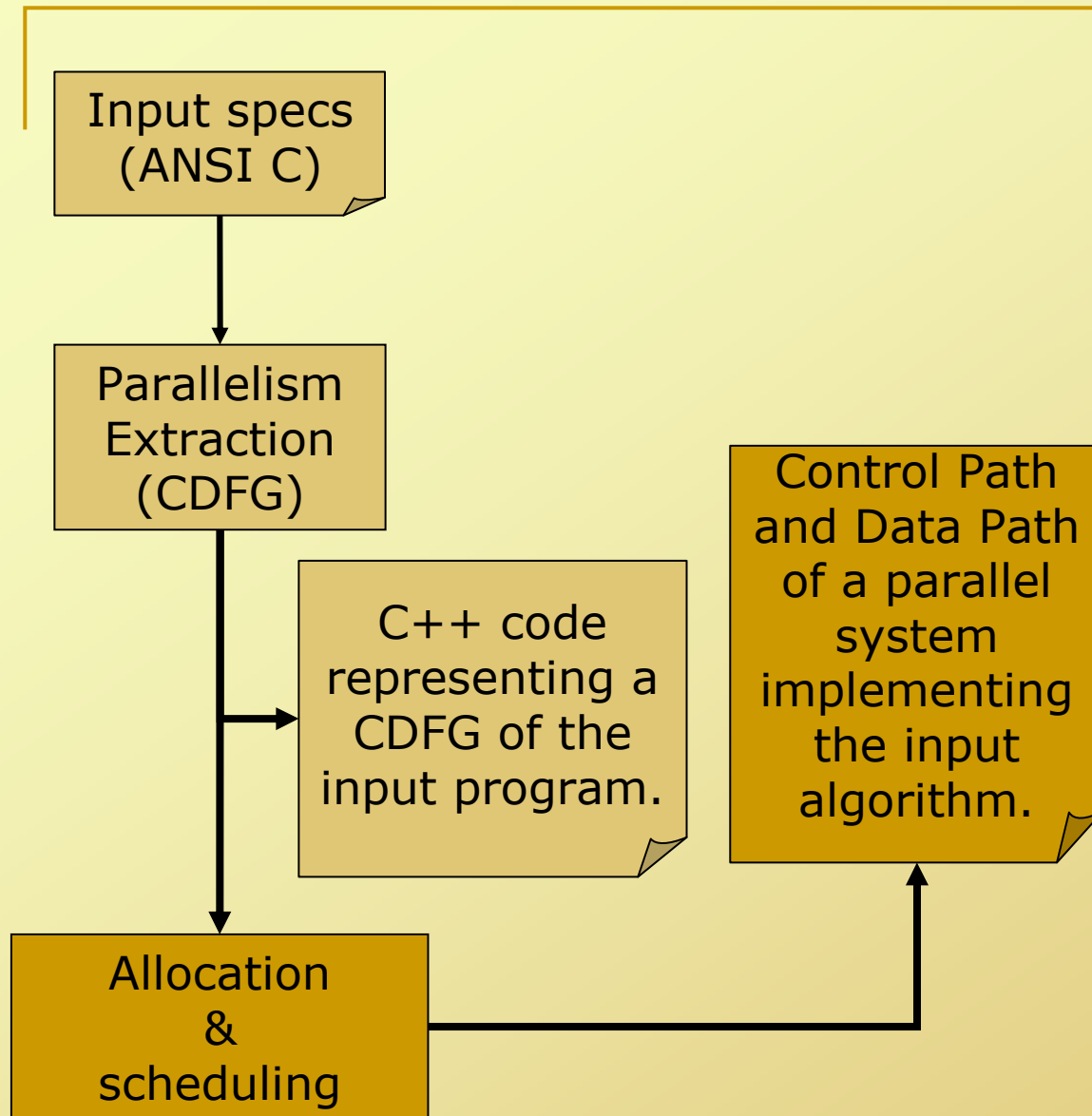
Input specs
(ANSI C)

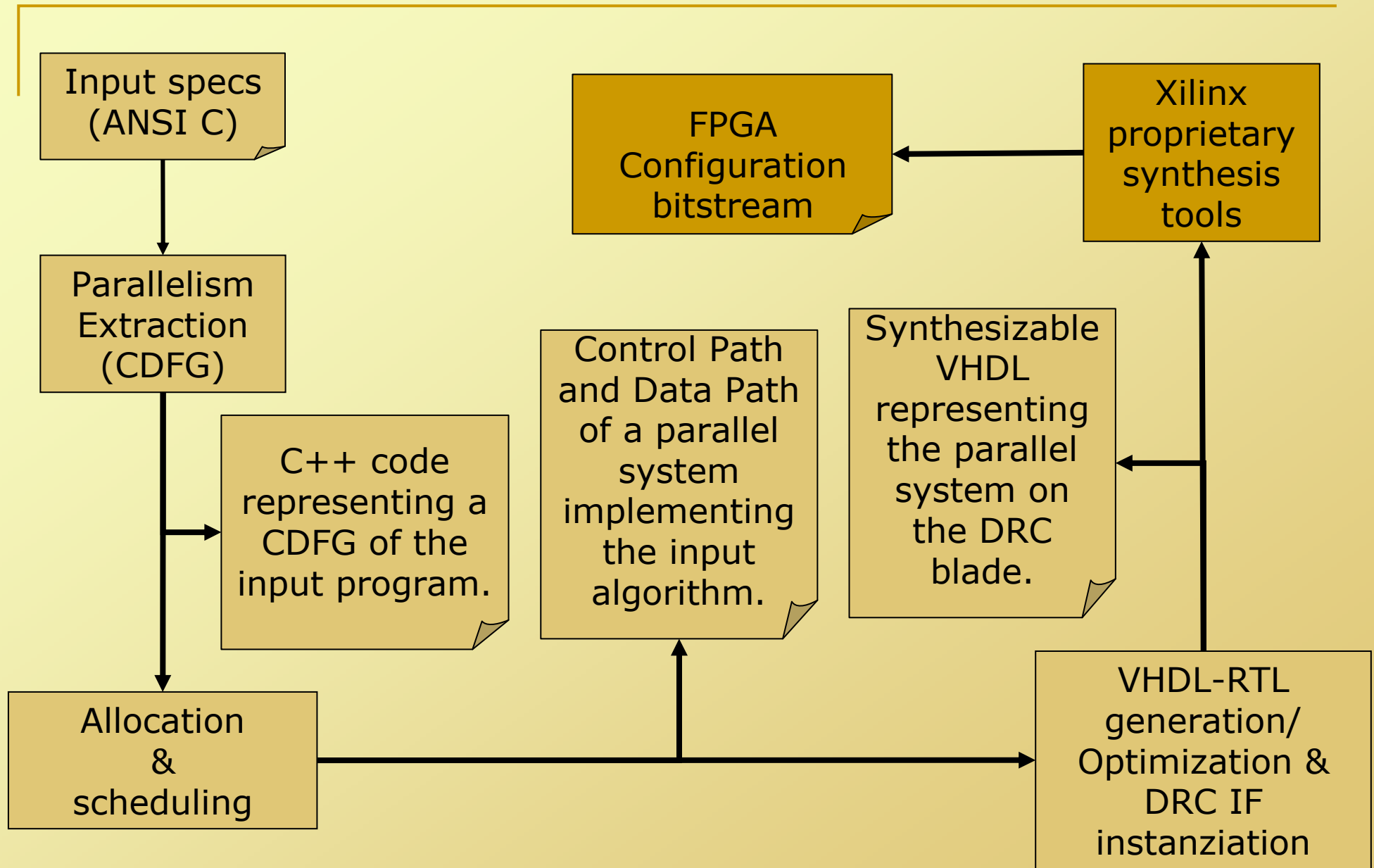


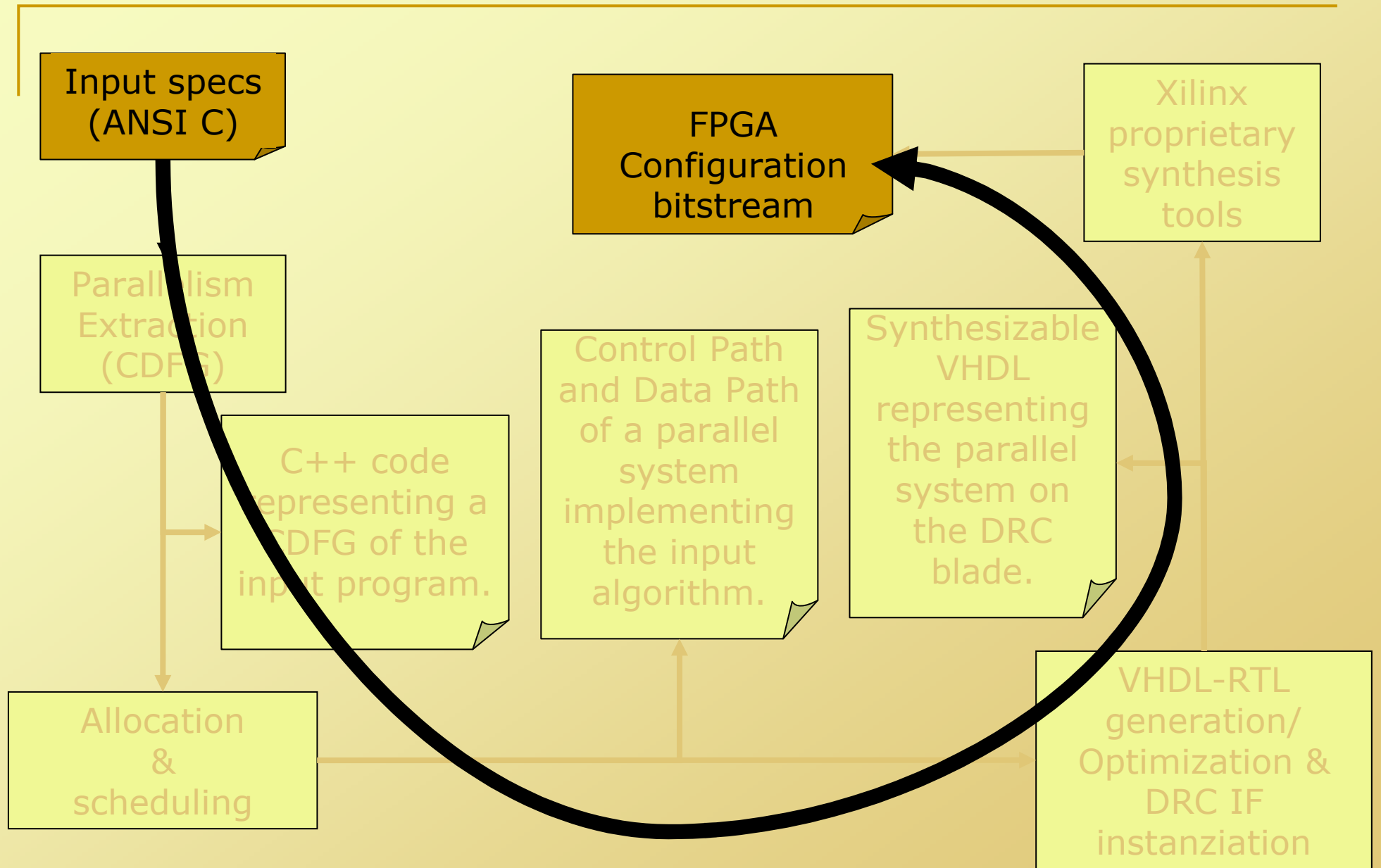
Parallelism
Extraction
(CDFG)



C++ code
representing a
CDFG of the
input program.







How does HCE work?

- The C code is translated into a parallel representation, i.e. the Control Data Flow Graph model (CDFG): roughly each operator corresponds to a CDFG node;
- At the CDFG level some basics optimization, aimed at reducing the complexity of the graph, are performed: constant folding/propagation, common sub-expression elimination, invariant code motion, dead code elimination

How does HCE work?

- Within the HCE we created a library of computing modules which are the building blocks used to set-up the final parallel architecture.
- The modules can be combinational, pipelined, multicycle, asynchronous.
- The CDFG is allocated and scheduled onto the available HW modules, which nearly map 1-1 with the nodes: every node has at least one module which implements it;

Which HW modules does HCE contain?

- modules in the library:

- Pipelined (*, +) and multi-cycle (/) floating point operators
- All the family of operators for char, int, fixed point
- The basic math functions (sinf, cosf, tanf, sqrtf, logf, expf)
- The rand() function
- Modules to manage pipelined memory banks with any (≤ 64) address and (2^k) word length

- The library can be extended with user defined modules

How does HCE work?

- The set of modules to support the C program can both manually and automatically be fixed.
- In the manual case it is possible to specify, for each C function, the multiplicity of the modules that will be used to implement that function: in such a way the user may control at which level of granularity the parallelism is exploited

How does HCE work?

- An efficient scheduling algorithm creates a Finite State Machine which executes, in the (~) shortest time, the original algorithm onto the available resources
- Optimized VHDL is generated by analyzing FSM
- Once the original C program has been translated into a synthesizable parallel architecture, the proper interface with the DRC environment is added

May algorithm hierarchy be exploited?

- Each C function may use another function

```
void f1(param_list_1) {...}
```

```
void f2(param_list_2) {...; f1(actual_param_list); ...}
```

```
void main_f(param_list_main_f) {...,f1();...;f2(); ...}
```

- Each function, starting from the innermost (i.e. f1), is synthesized and constitutes a new (asynchronous) module (m_f1) on which the node f1() can be allocated whenever invoked (both in f2() and in main_f()).

```

void BlockMatrixMAC(float BA[N][BS],float BB[N][BS],float BC[N][BS],int step)
{
    int i,j,k, base_k, BlockNumber;
    for (BlockNumber = 0; BlockNumber < NbProc; BlockNumber++){
        base_k = ((BlockNumber+step)*BS)%N;
        for (i=BlockNumber*BS;i<(BlockNumber+1)*BS;i++){
            for (j=0;j<BS;j++){
                for (k=0;k<BS;k++){
                    BC[i][j] += BA[i][k] * BB[base_k + k][j];
                }
            }
        }
    }
}

```

Implemented with
1 FP Add
1 FP Mul

```

void rotateMatrixBA(float BA[NbProc][N][BS] /*#HWST split 1 NbProc */)
{}

```

```

void CannonMM(float BA[NbProc][N][BS] /*#HWST split 1 NbProc */,
               float BB[NbProc][N][BS] /*#HWST split 1 NbProc */,
               float BC[NbProc][N][BS] /*#HWST split 1 NbProc */)
{
    for (int step = 0; step < NbProc; step++){
        int i;
        /*#HWST split */
        for (i=0; i<NbProc; i++){
            BlockMatrixMAC(BA[i], BB[i], BC[i], step);
            rotateMatrixBA(BA);
        }
    }
}

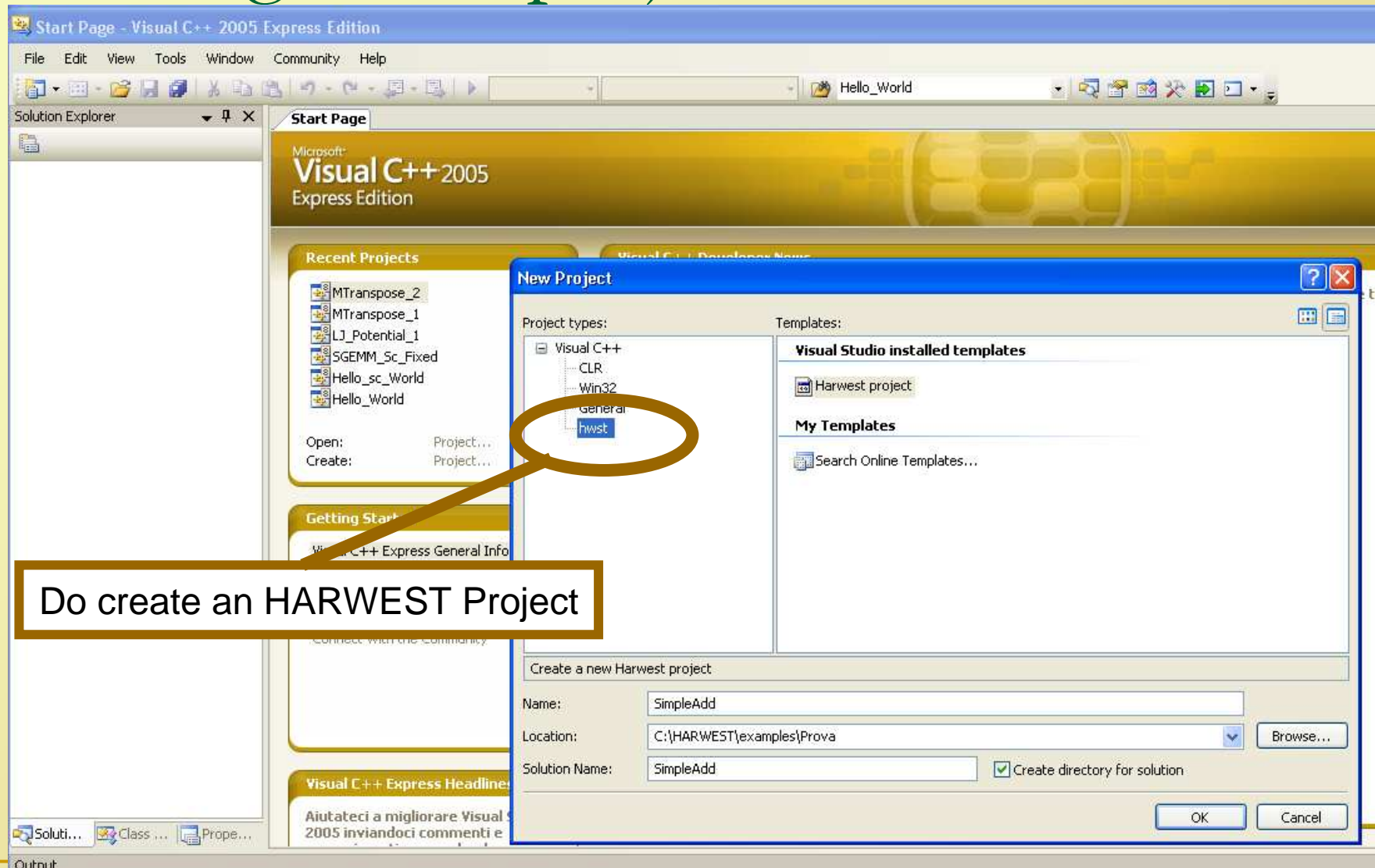
```

Implemented with
NbProc
BlockMatrixMAC

Outline of presentation

- Why using the ANSI C
- What is HCE
- How HCE works
- **Example of HCE use**
- The DRC co-processing boards
- HCE performances
- Conclusions


Creating a new project ...



Do create an HARWEST Project

... and configuring it

Harwest Project Wizard - SimpleAdd



Top Module Settings

Overview
Top Module Settings
Synthesis Settings
Project Settings

Return type: Top module name:

Parameter list (example: int a, int b):

Module signature: int SimpleAdd(int a, int b)
Implementation file: SimpleAdd.cpp

< Previous Next > Finish Cancel

Define the function I/O Behavior

... and configuring it

Harwest Project Wizard - SimpleAdd

 **Synthesis Settings**

Overview
Top Module Settings
Synthesis Settings
Project Settings

Synthesis target:
HWST::System::VHDL::FPGA::Xilinx::Virtex4::SystemVirtex4

Clock frequency (MHz):
0.0f
Insert here the clock frequency.

as soon as possible

Additional options:
 Enable data chaining

< Previous Next > Finish Cancel

Select the target (FPGA or board family)

... and some synthesis options

... and configuring it



SimpleAdd - Visual C++ 2005 Express Edition

File Edit View Project Build Debug Data Tools Window Community Help

Debug Win32 Hello_World

Solution Explorer - Solution ...

Solution 'SimpleAdd' (2 projects)

- SimpleAdd_debug
 - Header Files
 - SimpleAdd.h
 - SimpleAdd_testbench.h
 - Source Files
 - Debugger.cpp
 - SimpleAdd.cpp
 - SimpleAdd_testbench.c
 - readme.txt
- SimpleAdd_synthesizer
 - Header Files
 - PreprocessorSettings.h
 - SimpleAdd.h
 - SynthesisParameters.h
 - Source Files
 - SimpleAdd.cpp
 - Synthesizer.cpp
 - readme.txt

SimpleAdd.cpp* Start Page

(Global Scope)

```
/*
 * HARWEST compiling environment
 * Version 0.2.2.0
 * File created by the Harwest Visual Studio wizard
 *
 * Ylichron S.r.l.
 * visit "http://www.ylichron.it" for more info.
 *
 * File details:
 * - Source implementation file for SimpleAdd module;
 * - Project: SimpleAdd;
 */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "SimpleAdd.h"

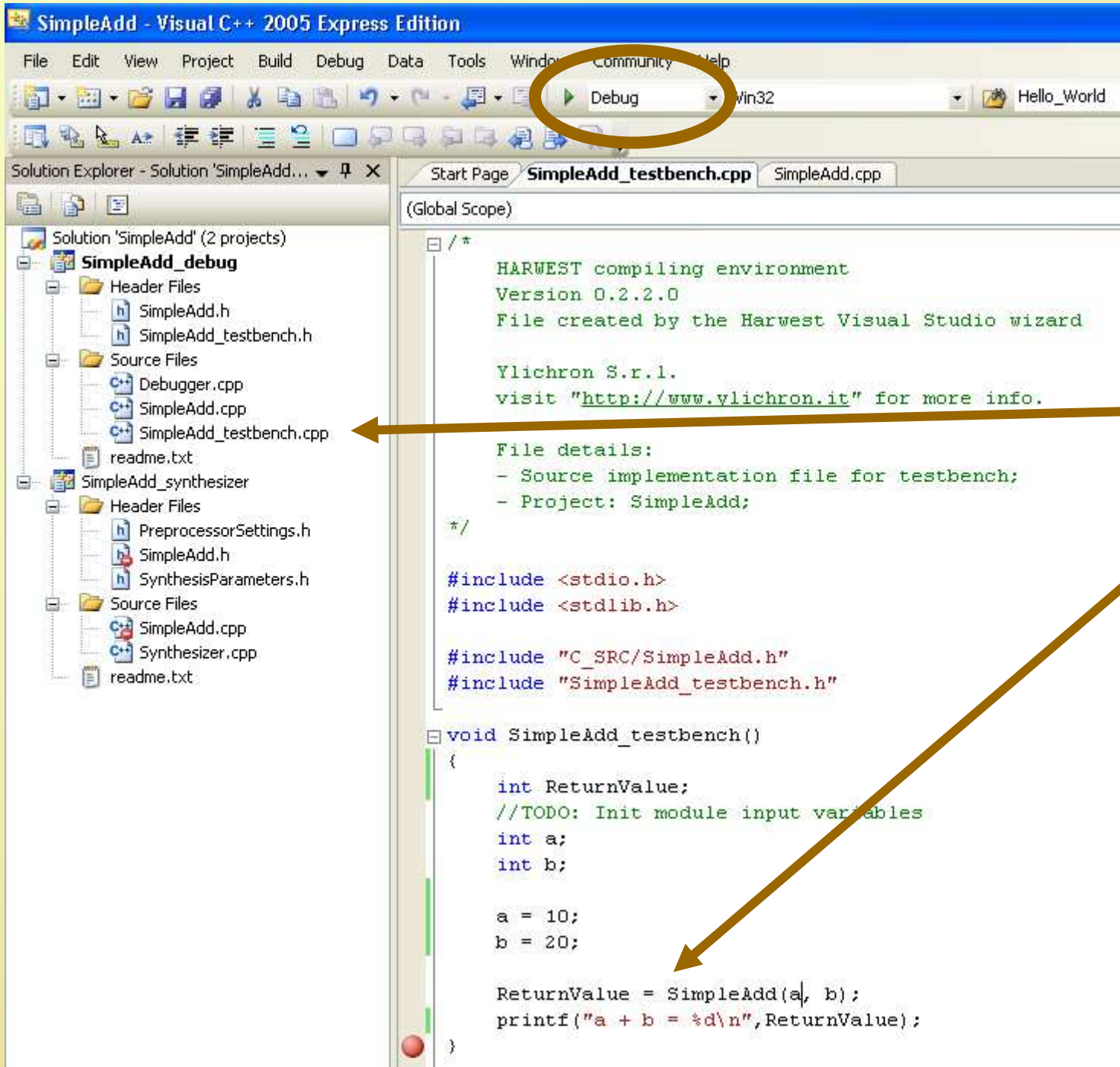
int SimpleAdd(int a, int b)
{
    int ReturnValue;

    //HARWEST TODO: insert here the C code describing .
    //                Check HARWEST documentation to learn about the C/C++ subset accepted
    //                by the HARWEST synthesizing environment.

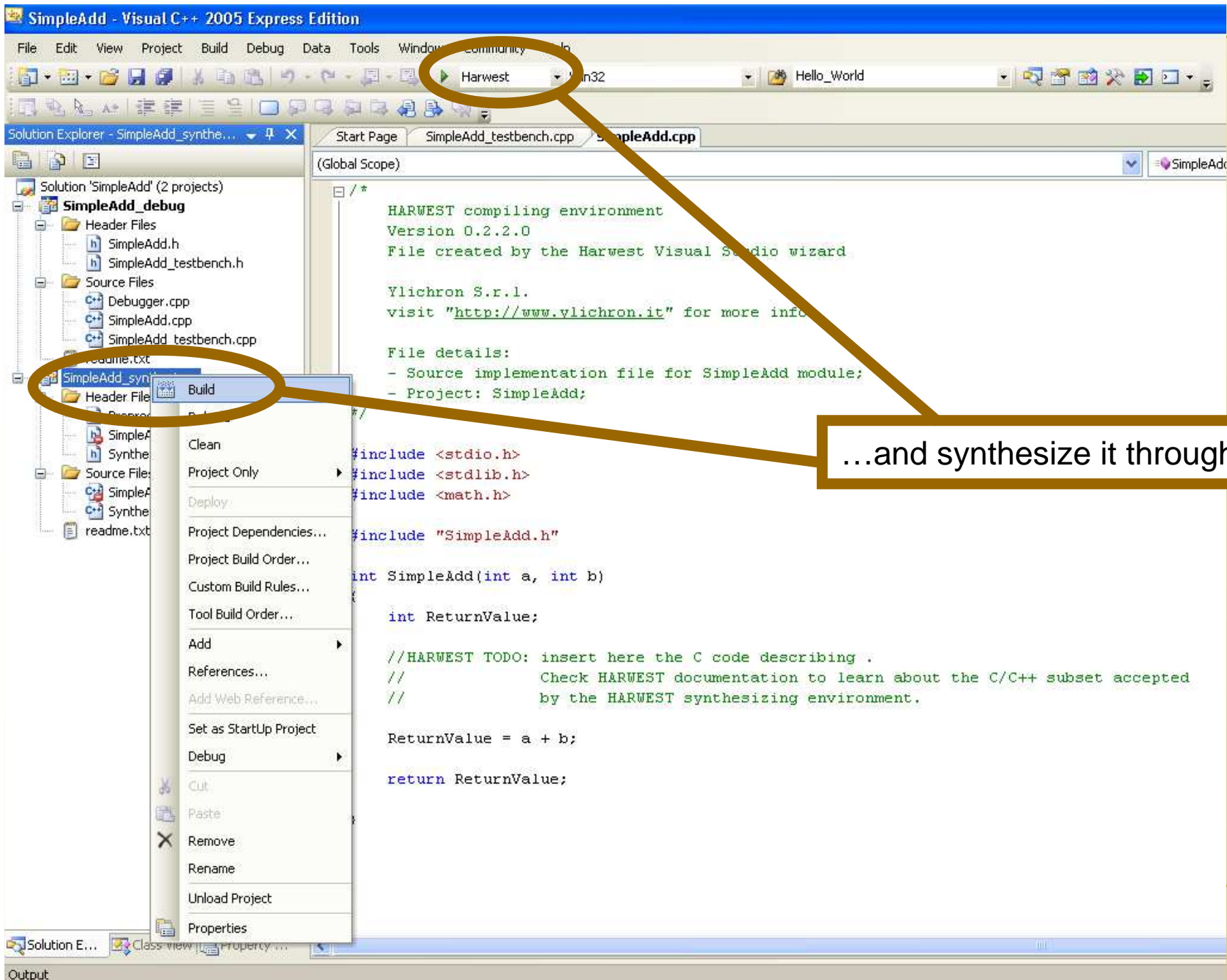
    ReturnValue = a + b;

    return ReturnValue;
}
```

Write your code....



Debug your code using the VisualStudio C Compiler ...



...and synthesize it through HCE Flow

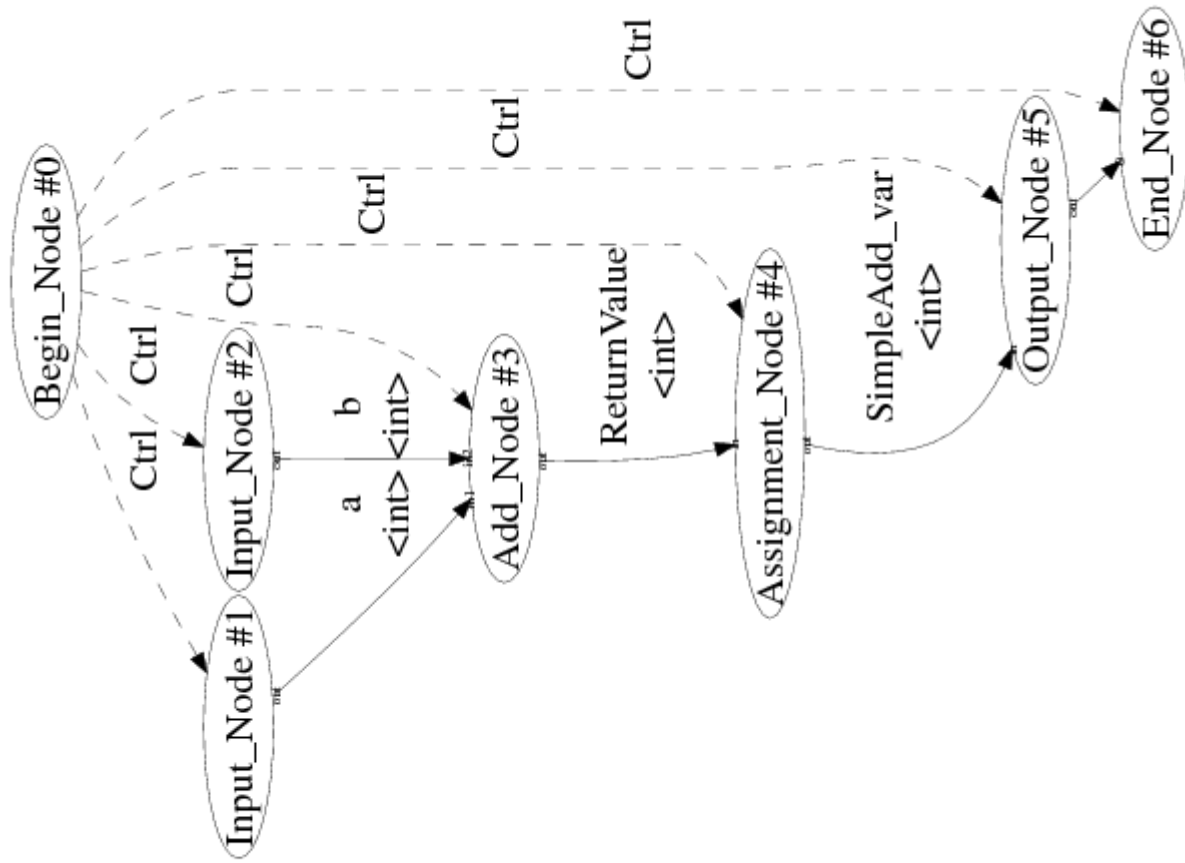
HCE Output Files

The screenshot shows a Windows Explorer window titled "SRC" with the address bar set to "C:\HAR\WEST\examples\Prova\SimpleAdd\SimpleAdd\VHDL\SRC". The left pane shows a tree view of folders, including "SimpleAdd", "debug", "SimpleAdd", "C_SRC", "CDFG", "Debug", "FSM", "HWSTC_SRC", "System", "VHDL", "SIM", and "SRC". The right pane displays a list of files with columns for "Nome", "Dimensione", "Tipo", and "Data ultima modifica".

Nome	Dimensione	Tipo	Data ultima modifica
AddModule_int_int_int	2 KB	File VHD	02/05/2008 11.41
OutputModule_int	2 KB	File VHD	02/05/2008 11.41
SimpleAdd_Ctrlpath	3 KB	File VHD	02/05/2008 11.41
SimpleAdd_Datapath	4 KB	File VHD	02/05/2008 11.41
WireModule_int	1 KB	File VHD	02/05/2008 11.41

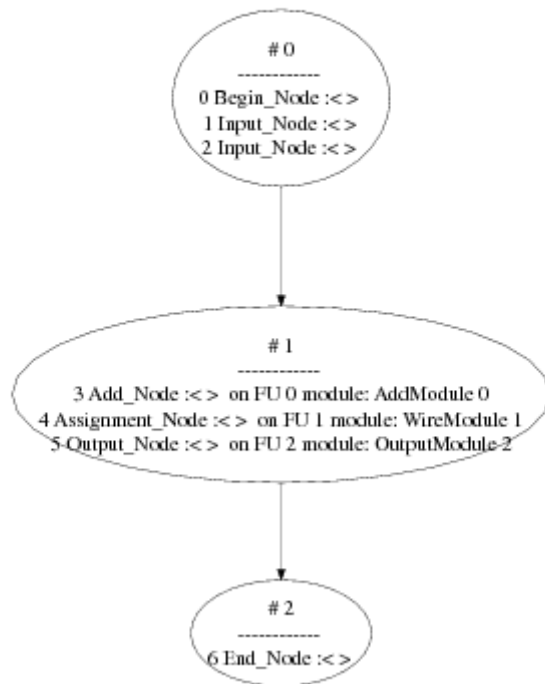
The CDFG

Control Data Flow Graph of SimpleAdd
Nodes Nb: 7



The FSM

Finite State Machine of SimpleAdd_FSM
States Nb: 3



Outline of presentation

- Why using the ANSI C
- What is HCE
- How HCE works
- Example of HCE use
- **The DRC co-processing boards**
- HCE performances
- Conclusions

DRC co-processing boards

- DRC provides a co-processor system which fits on a free Opteron socket.
- Due to the tight interconnection to the host buses it provides high communication bandwidth between the host and the co-processor system.
- At the moment of writing the system is provided in two versions: RPU110-L60 and DRC RPU110-L200, both based on the Xilinx Virtex4 FPGAs (LX60 and LX200)

DRC co-processing boards

DRC co-processor system is equipped with:

- one Xilinx Virtex4 FPGA
- HyperTransport (HT) interconnection;
- Up to 2GB of RPU DRAM with two independent physical buses each with 3.2GB/s peak bandwidth;

DRC co-processing boards

- 128 MB of RPU Low Latency RAM (LLRAM) with two independent physical buses each with 800Mb/s peak bandwidth;
- 256 Mbits of not volatile Flash RAM

Outline of presentation

- Why using the ANSI C
- What is HCE
- How HCE works
- Example of HCE use
- The DRC co-processing boards
- **HCE performances**
- Conclusions

Performances

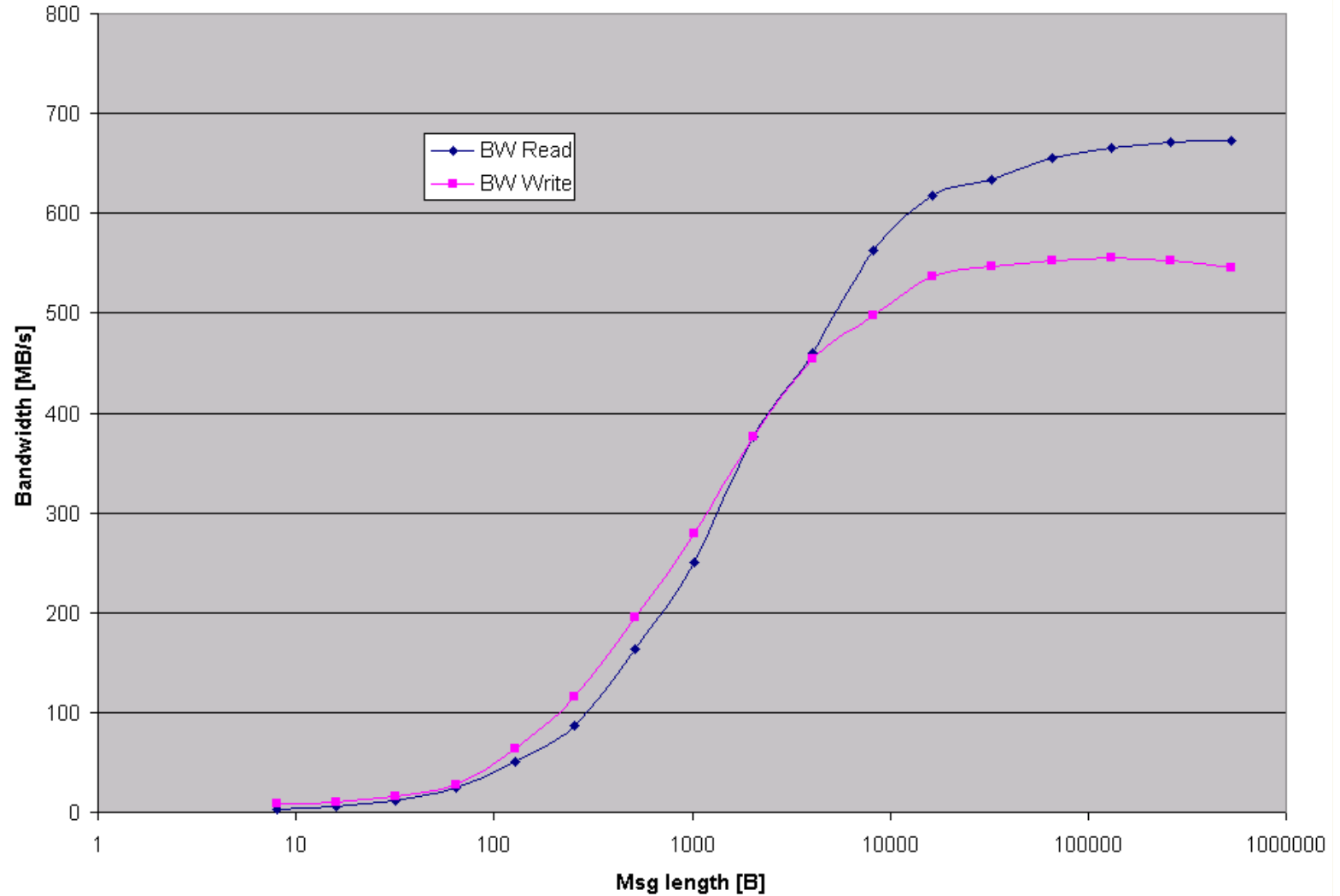
- test 1: measure of Read/Write bandwidth
- test 2: measure of internal memory bandwidth
- test 3: Boolean computation (graph transitive closure)
- test 4: DNA sequences (Smith Watermann)
- test 5: fixed point computations (FIR filter)
- test 6: floating point computations (Cannon algorithm for the matrix product)

test 1: measure of Read/Write bandwidth

Write: host \Rightarrow DRC

Read: DRC \Rightarrow host

The time necessary to write (read) the shortest message (8 bytes) is equal to 0.9 μ s (2.5 μ s).



test 2: measure of internal memory bandwidth

- The test performs the transposition of a 2D square matrix, copying the original matrix $m[M][M]$ into the transposed matrix $mT[M][M]$, ($M=128$)
- 136 clock cycles to transpose the matrix.
- $f_{ck}=100$ MHz,
- $M \times M \times 4$ bytes are read and written in $1.36 \mu s$
- $BW_{Read}=BW_{Write}=45$ GB/s.
- slices used: 6601 (7% of the total for a Virtex4 LX200 FPGA).
- Block RAM modules used: 256.

test 3: graph transitive closure

- $G(N)$ graph represented through its boolean incidence matrix $a[N][N]$ ($N = 2048$)
- $T_{EXE} = 250$ ms, corresponding to $2*N*N*N/T_{EXE} = 68 \times 10^9$ op/s.
- slices used: 3695 (4% of the total for V4LX200)
- Block RAM modules used: 256

test 4: Smith Watermann

- SW uses a dynamic programming approach to find the best alignment (with insertions, deletions and mismatches) between a DNA sequence of size m ($=255$) and another sequence of size n (1024).
- Implemented the computation of the scoring matrix
- HCE run time: $53 \mu\text{s}$, corresponding to 5 GScoreUpdate/sec
- fck = 100 MHz.
- slices used: 20897 (23% of the total for V4LX200).

test 5: FIR filter

- Filter: 128 taps
- Input signal $x[N]$ ($N = 1024$)
- Both the filter taps and the samples used the `sc_fixed<16,8>` SystemC data type
- $f_{ck} = 60$ MHz, $T_{EXE} = 17$ μ s
- sustained computation rate of 14 Gop/s,
- slices used: 30489 (34% of the total for V4LX200)
- all the 96 available DSP blocks have been used

test 6: GEMM Cannon algorithm

- Parallel formulation of the MM algorithm, often used for systolic implementation
- MatSize $M = 128$
- $f_{ck} = 80$ MHz, $T_{EXE} = 3.7$ ms
- sustained computation rate of 1.1 GFlop/s,
- slices used: 40799 (45 % of the total for V4LX200)
- all the 96 available DSP blocks have been used
- Block RAM modules used: 98

Outline of presentation

- Why using the ANSI C
- What is HCE
- How HCE works
- Example of HCE use
- The DRC co-processing boards
- HCE performances
- **Conclusions**

Conclusions

- The HCE, a C to VHDL automatic design suite has been presented;
- Its theoretical bases have been revised;
- Its use (and usability) has been discussed;
- Some performance figure have been reported;

Contact information

- Commercial: info@ylichron.it
- Technical: support@ylichron.it
- Web site: <http://www.ylichron.it>