FPL 2012 Oslo, Norway August 29-31 2012

# On the Automatic Integration of Hardware Accelerators Into FPGA-Based Embedded Systems

Christian Pilato, Andrea Cazzaniga, Gianluca Durelli,
Andres Otero[+], Donatella Sciuto, and Marco D. Santambrogio

Politecnico di Milano - Dipartimento di Elettronica ed Informazione Milano (Italy)
{acazzaniga, pilato, sciuto, santambr}@elet.polimi.it, gianluca.durelli@mail.polimi.it
[+]Universidad Politecnica de Madrid - Centro de Electrònica Industrial - Madrid (Spain)
joseandres.otero@upm.es

## Rationale

- Heterogeneous embedded systems are becoming very popular to perform one or a few dedicated functions
- They allow to accelerate different parts of an applications with general purpose, digital signal and hardware accelerators (FPGA, ASIC) interconnected through various communication mechanisms
- Optimization of such complex embedded systems is a difficult problem and many error prone steps of the traditional design flow are performed manually by the designer
  - Integration of the hardware cores
  - Rewriting of the software code including synchronization/ communication directives and execution coordination

## Goals

- Realization of an automatic framework for the seamless integration of hardware accelerators in an embedded system, starting from an OpenMP-based application and an XML file describing the HW/SW partitioning.
- The framework will assist the designer during all the steps of the traditional flow allowing him/her to automatically:
  - Generate HDL description of HW cores
  - Wrap HW cores with proper interfaces for system level integration (bus and memory)
  - Generate SW code for GPP processors
  - Generate drivers for HW/SW communication and synchronization directives
  - Generate a scheduler which coordinates the execution of the resulting application
- The output of the proposed framework is both HW and SW specifications of the system ready to be synthesized for the target device (i.e. Xilinx EDK project along with the software code)

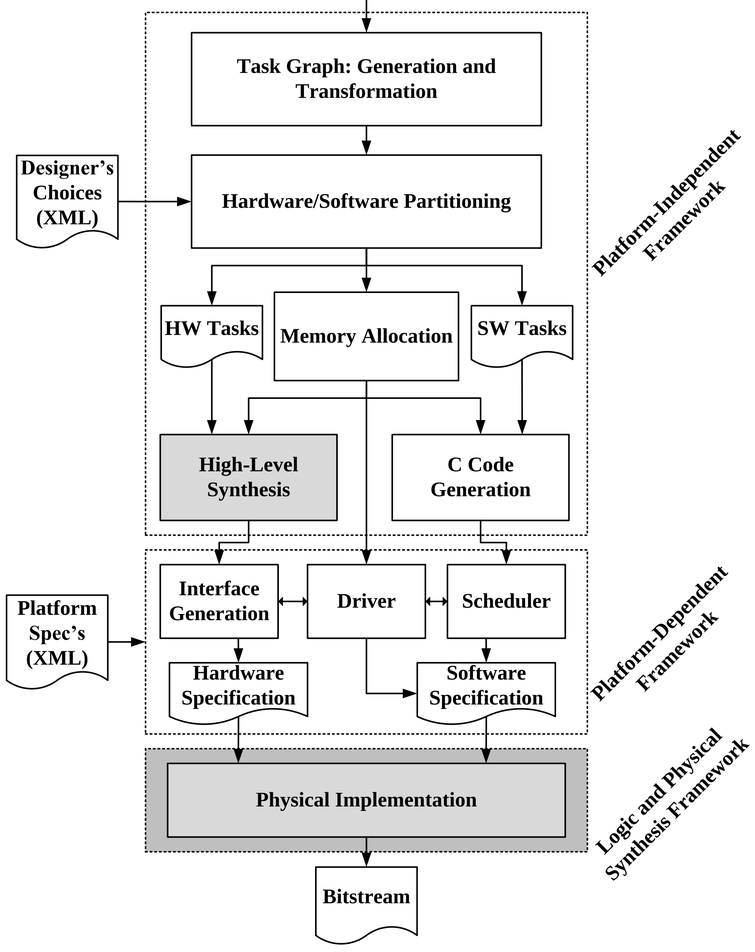
## Use Cases

The resulting framework can be used by:

- Architecture designer
  - Analysis and comparison of different High Level Synthesis tools
  - Validation of core interfaces
- Application designer:
  - Rapid evaluation of HW/SW partitioning solutions or algorithms through actual execution on the target platform
- OS level designer
  - Comparison and validation of different scheduling policies
- Ongoing work:
  - Support for multiple and concurrent applications
  - Integration of dynamic partial reconfiguration
  - Support for different architecture topologies
  - Integration of Graphical User Interface

## Proposed Flow

- Input:
  - Application description annotated with OpenMP pragmas (C code)
  - XML description specifying:
    - Processing elements of the target architecture
    - HW/SW partitioning
- Platform-Independent Framework:
  - Generation of application Task Graph based on OpenMP annotated code
  - Classification and generation of SW and HW tasks
  - Memory allocation
  - High Level Synthesis
  - Software tasks generation
- Platform-Dependent Framework:
  - Generation of core interfaces, drivers and scheduler directives based on target architecture description
  - Generation of HW and SW specifications (i.e. Xilinx EDK project)
- Logic and Physical Synthesis Framework:
  - Synthesis and bitstream generation using commercial tools (Xilinx EDK)
- Output:
  - Bitstream ready to be configured on the target architecture including the scheduler code which coordinates the application execution



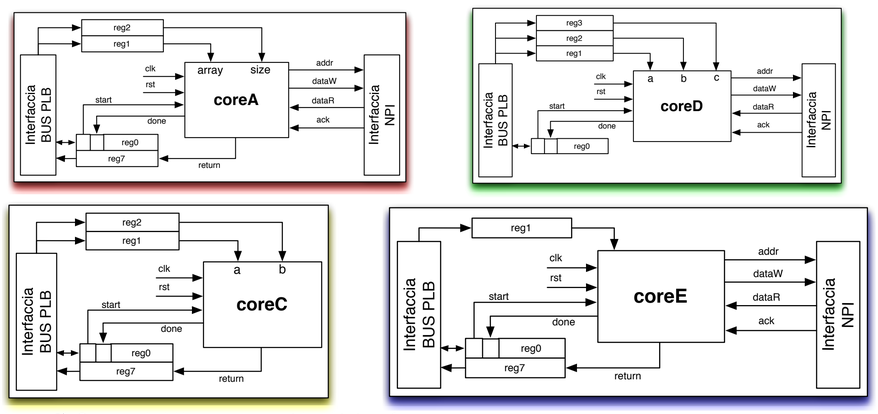

## Example

```
int coreA(int *array, int size);
void coreB(int *array, int size, int* max);
int coreC(int a, int b);
void coreD(int a, int b, int *c);
int coreE(int *a);

int main(int argc, char** argv){
   int array[ARRAY_SIZE] = {4, 3, 6, 5, 4};
   int a, b, c, d, e;
#pragma omp parallel sections num_threads(2)
   {
#pragma omp section
      {
         a = coreA(array, ARRAY_SIZE);
      }
#pragma omp section
      {
         coreB(array, ARRAY_SIZE, &b);
      }
   }
   c=coreC(a, b);
   coreD(a, b, &d);
   e=coreE(&d);
   return e;
}
```

1) Application source code

```
<architecture>
  <system id="fpga_0" type="FPGA">
    <device id="xc5vlx110t-ff1136"/>
    <processing_elements>
      <processing_element id="microblaze_0" type="GPP"/>
      <processing_element id="microblaze_1" type="GPP"/>
      <processing_element id="IpCore0" type="HWCORE"/>
      <processing_element id="IpCore1" type="HWCORE"/>
      <processing_element id="IpCore2" type="HWCORE"/>
      <processing_element id="IpCore3" type="HWCORE"/>
    </processing_elements>
  </system>
</architecture>

<application>
  <mapping task="coreA" component="IpCore0"/>
  <mapping task="coreB" component="microblaze_1"/>
  <mapping task="coreC" component="IpCore1"/>
  <mapping task="coreD" component="IpCore2"/>
  <mapping task="coreE" component="IpCore3"/>
</application>
```

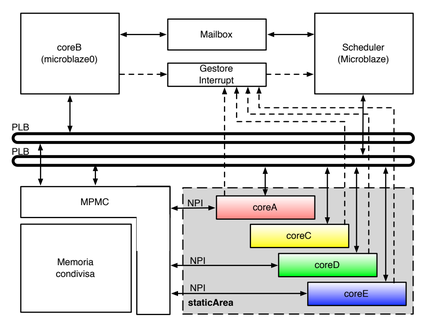
2) XML file



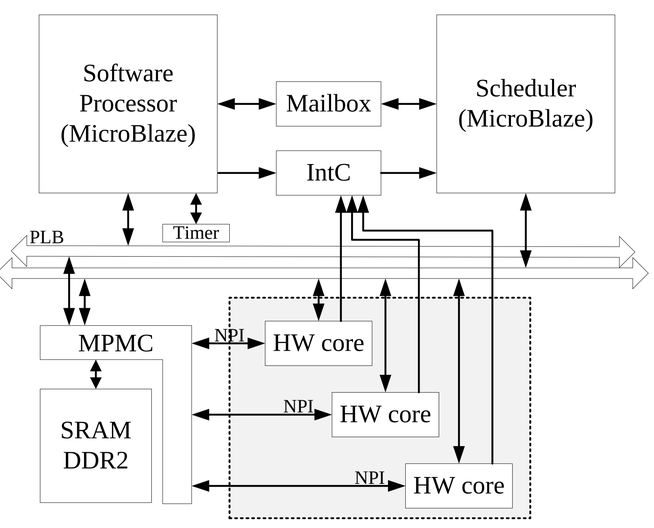

3) Generated HW cores along with their interfaces

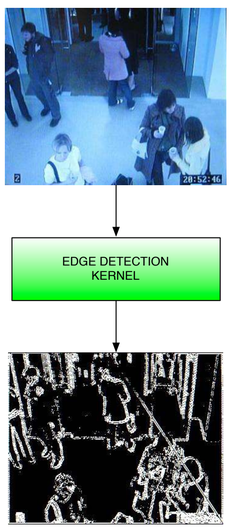
4) System Integration

- Starting from OpenMP application description (1) and the mapping solution (2), the hardware cores are automatically generated (3)
- In this example coreB is set to be executed in SW, while the other functions (*coreA*, *coreC*, *coreD* and *coreE*) will be executed as a HW core
- The system so specified (represented in the figure) is generated automatically by the proposed flow and the output is both HW and SW specification of the system




## Experimental Evaluation: Edge Detection




The table shows the different architectures generated by the proposed framework.
In particular it shows:

- the area occupation and execution time of the kernel computation for the different solutions
- the speed-up with respect to the sequential software execution (which is also represented in the graph).

The automatic generation different solutions permitted to:

- easily evaluate the performance of the analyzed application
- identify the degree of parallelism which is exploitable

The results in fact show that having more that 4 parallel HW tasks results in a degrade of performance due to memory access contention.

| # Cores | | | Area | | Time | |
|---|---|---|---|---|---|---|
| | SW | HW | (# slices) | (%) | (# cycles) | $S_p$ |
| 1 | 1 | 0 | 4,102 | 23.74 | 6,593,760 | 1.00 |
| | 0 | 1 | 5,088 | 29.44 | 2,663,764 | 2.48 |
| 2 | 1 | 1 | 4,452 | 25.76 | 3,348,185 | 1.97 |
| | 0 | 2 | 5,709 | 33.04 | 1,362,660 | 4.84 |
| 4 | 1 | 3 | 6,977 | 40,38 | 1,777,853 | 3.71 |
| | 0 | 4 | 7,908 | 45.76 | 1,244,516 | 5.30 |
| 6 | 1 | 5 | 8,408 | 48.66 | 1,396,466 | 4.72 |
| | 0 | 6 | 9,432 | 54.58 | 1,331,082 | 4.95 |