POLITECNICO DI MILANO

PandA-Bambu – A free software framework for the High-Level Synthesis of Complex Applications

Fabrizio Ferrandi
Christian Pilato

Politecnico di Milano
Dipartimento di Elettronica ed Informazione
{ferrandi,pilato}@elet.polimi.it

Outline

- PandA framework
- GNU GCC integration
- High-level synthesis with bambu
- Demo
- Future Works

PandA project objectives

- Definition of design methodologies and tools;
- Design of system architectures;

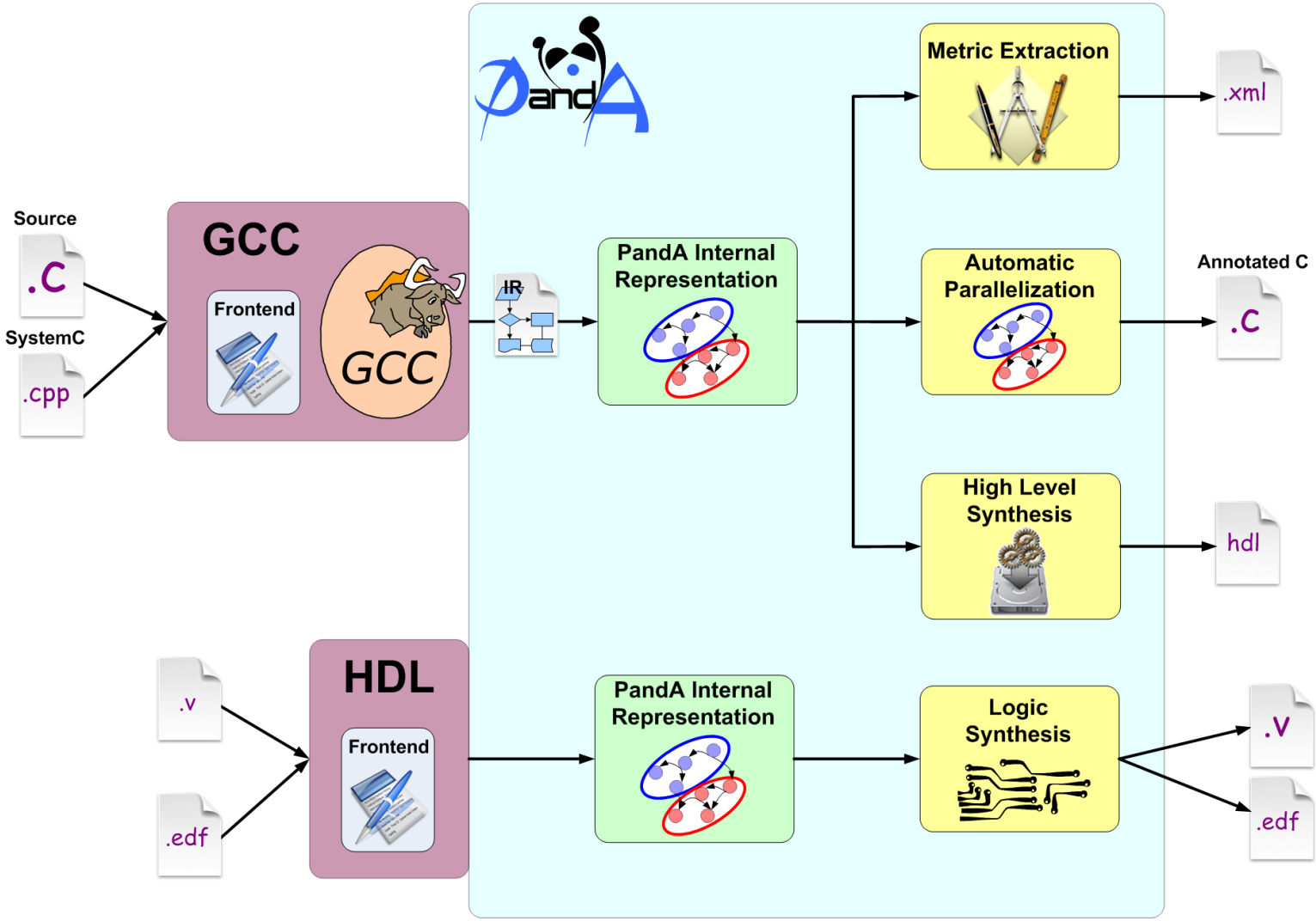
- Support the development of hardware/software embedded systems:
  - telecommunication
  - automotive
  - industry field
  - consumer field

Goals of the framework

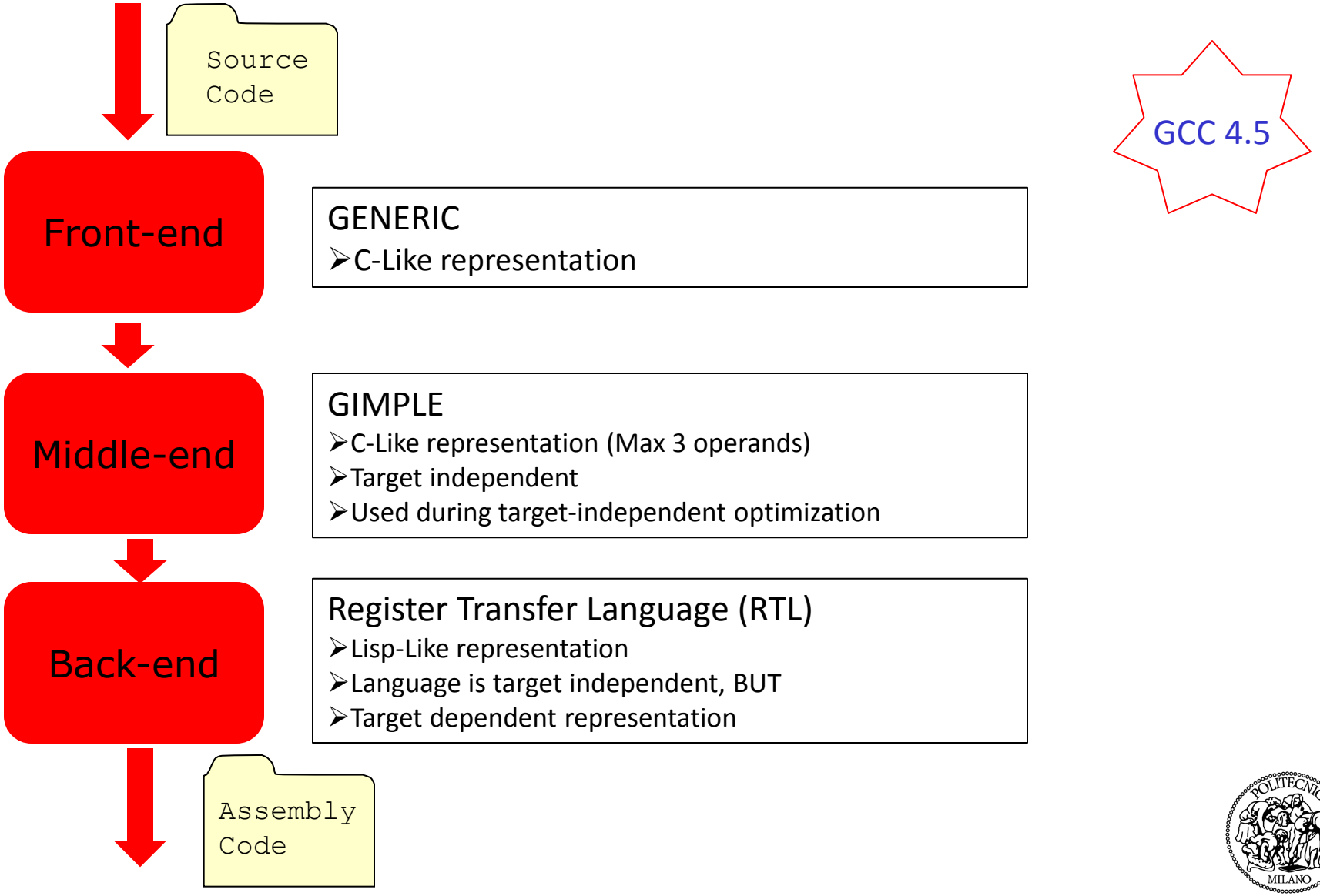
- Allow users to concentrate on the specific algorithms and not in the parsing of system-level or software specifications.

- Provide tools supporting the research on:
  - high-level synthesis of hardware systems;
  - parallelism extraction for software;
  - hardware/software partitioning;
  - definition of metrics for the analysis and mapping onto multiprocessor architectures;
  - design of dynamically reconfigurable systems.

Framework Overview



GNU GCC Internal Representation

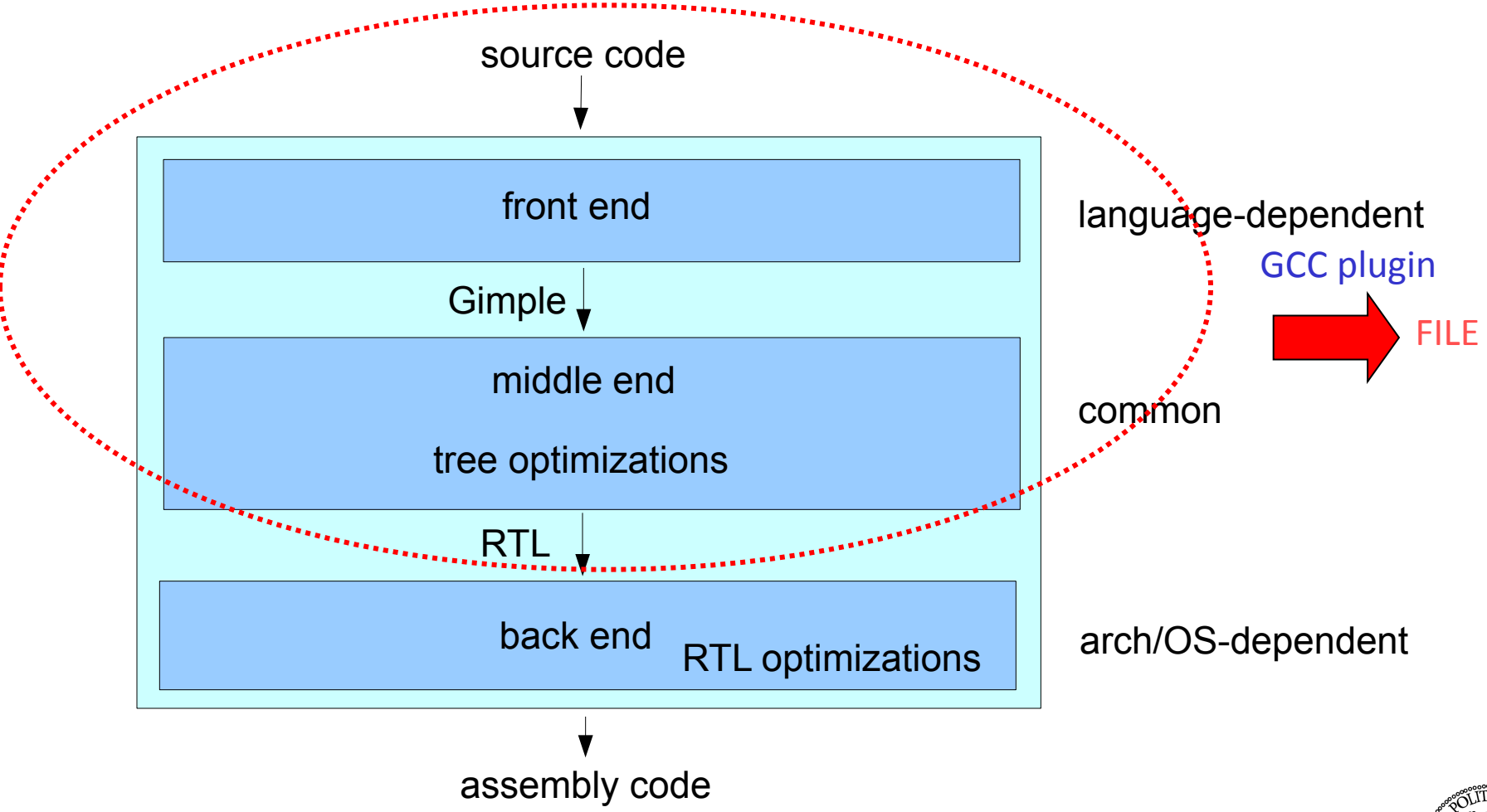


GCC 4.5

GCC middle-end optimizations

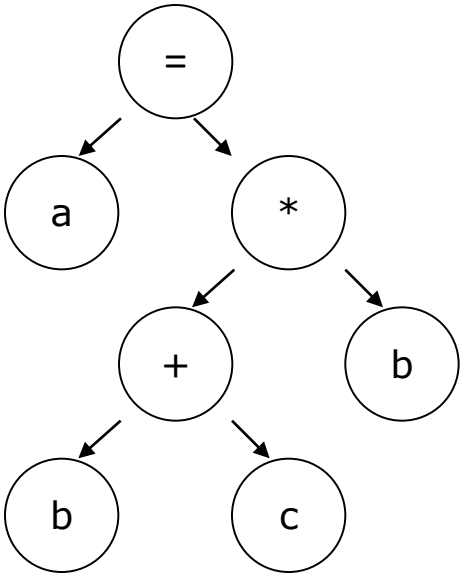
- building the SSA representation
- expanding vector to scalar
- may-alias information
- dead code elimination
- partial redundancy elimination
- full redundancy elimination
- forward propagation of expressions
- SSA copy renaming
- global value numbering
- sparse conditional constant propagation
- scalar replacement of aggregates
- value range propagation
- loop optimization
- common subexpression elimination
- reassociation pass
- dead store elimination

Interfacing to GCC



GCC IR: example

```
int function
{
    int a, b, c;
    a= (b+c)*b;
    return a;
}
```



Gimple Representation:

@4 statement_list

bloc: 2 pred: ENTRY succ: EXIT stmt: @9 stmt: @10 stmt: @11

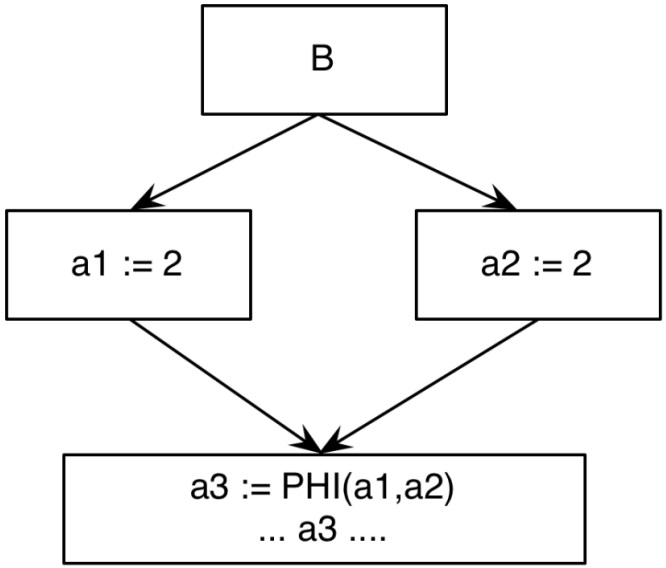
@10 gimple_modify_stmt	**op0: @29**	**op1: @20**	
@18 plus_expr	**type: @7**	**op: @30**	**op: @31**
@20 mult_expr	**type: @7**	**op: @18**	**op: @30**

Graph creation

- Panda transforms the GCC tree into a Control Flow Graph - CFG
- Starting from CFG several other graphs are created:
  - Data Flow Graph (DFG): data dependences between instructions
  - Control Dependence Graph (CDG): control dependences between instructions
  - Anti-Dependence Graph (ADG): anti-dependences between instructions
  - Flow edges Graph (FG): precedences between instructions inside the loop and instructions outside the loop
- Basic graphs could be composed to be used by the next steps of the development flow

SSA: a Simple Example

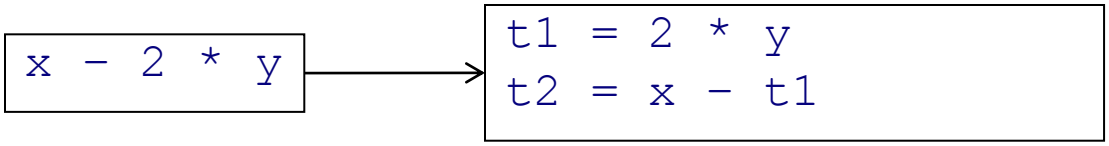
```
if B then
   a1 := 1
else
   a2 := 2
End
a3 := PHI(a1,a2)
   … a3 …
```



- It allows several optimizations at intermediate representation and at synthesis level

3-address code

- Statements take the form: x = y op z
  - single operator and at most three names



- Advantages:
  - compact form
  - names for intermediate values

Typical 3-address codes

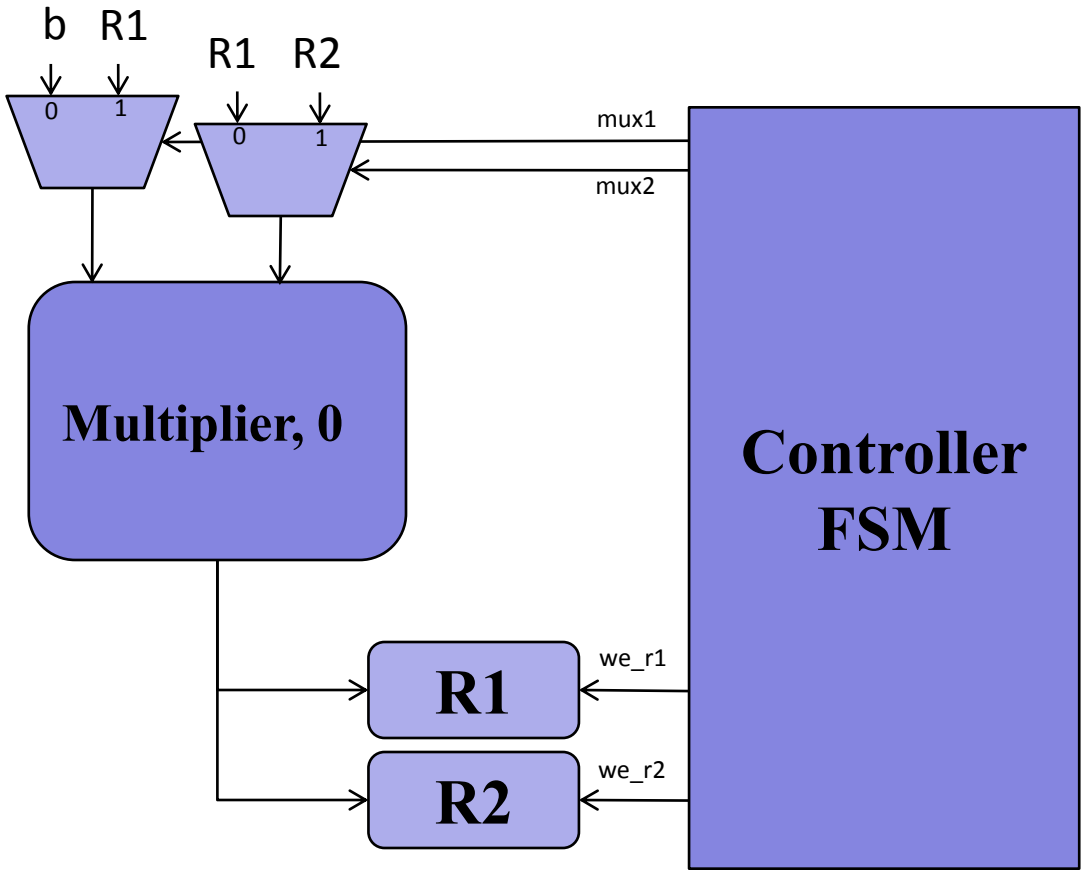
assignments	x = y op z
	x = op y
	x = y[i]
	x = y
branches	goto L
conditional branches	if x rel op y goto L
procedure calls	param x param y call p
address and pointer assignments	x = &y *y = z

High-Level Synthesis

- Design and implementation of a digital circuit starting from a behavioral representation
  - We aim at a full support of ANSI C
- Different state-of-art algorithms have been implemented
  - Memory allocation
  - Module allocation and binding
  - Register allocation and binding
  - Interconnection allocation and binding
  - Controller and Datapath generation
- The output is a synthesizable code in a common hardware description language (e.g., Verilog, VHDL)

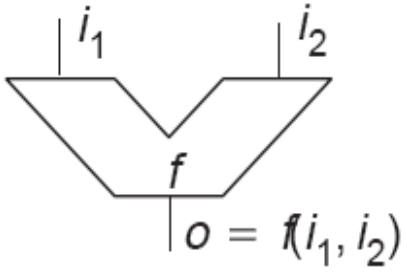
Target Architecture

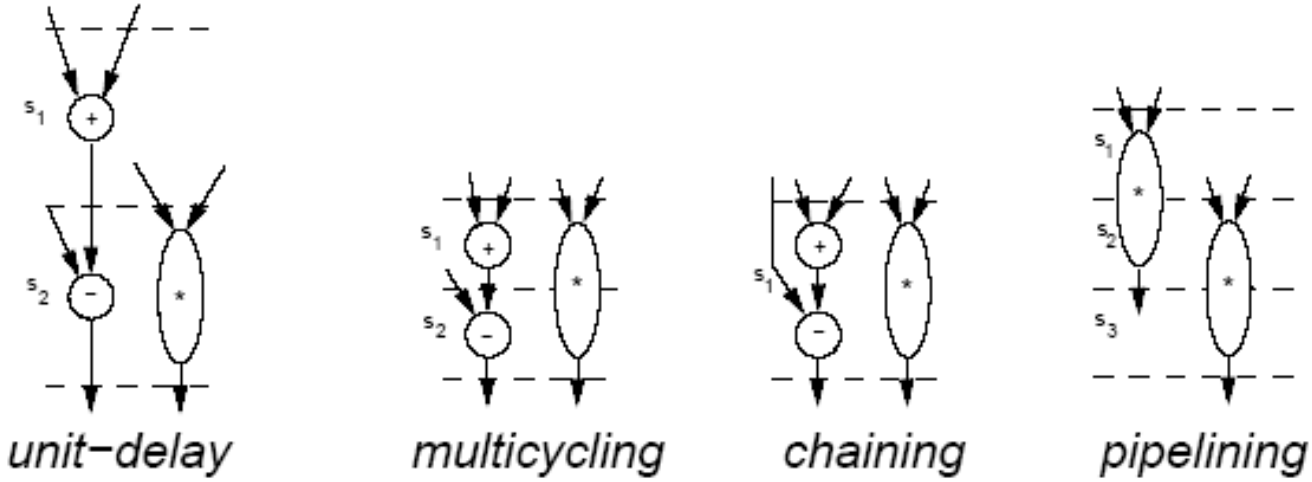
Portion of the Datapath



Functional resources

- Standard resources:
  - Existing macro-cells.
  - Well characterized (area/delay).
  - Example: adders, multipliers, ...





Functional resources

- Resource library composed of about 100 components
- 56 components are actually templates:
  - Parametrizable with respect to the operation data sizes (8, 16, 32, 64)
- Some builtins are available:
  - *exit*, *abort*, *printf*, *puts*, *putchar* for debugging purposes
  - *abs*, *memcpy*, *memset*, *memcmp*
- *malloc* and *free* supported through a C based library
- Floating point modules generated by exploiting FloPoCo library
- Supported pipelined, multicycling and unbounded functional resources
- Resource library described in XML
  - Easily extendible

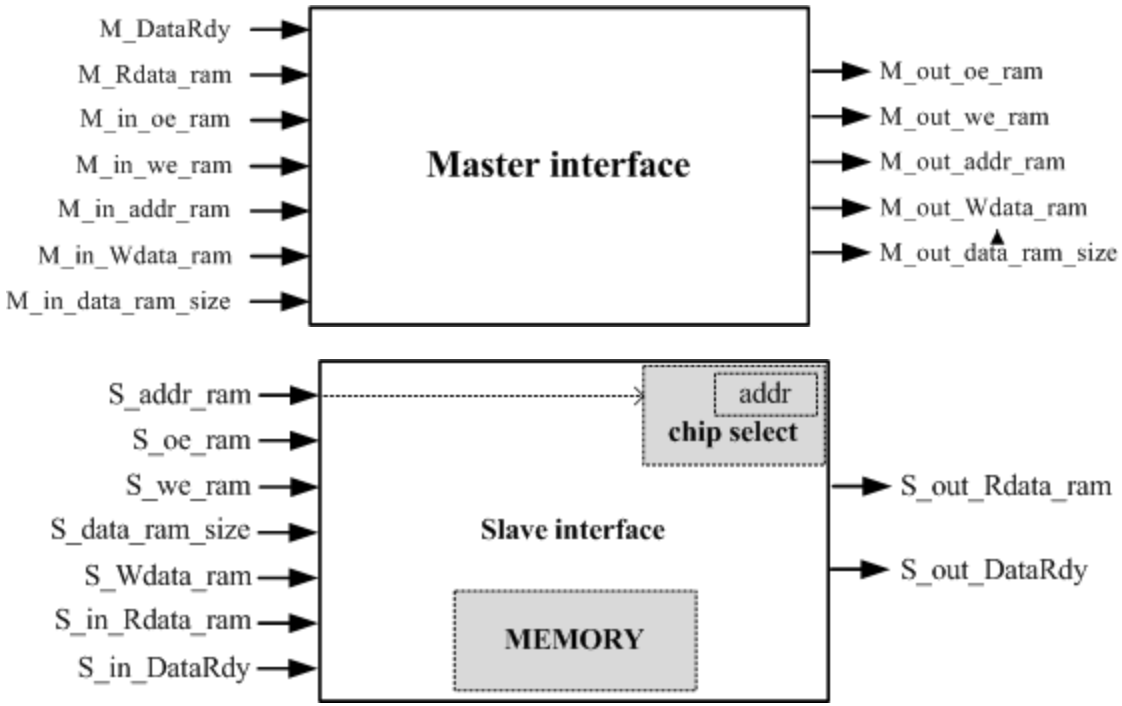
Simple component description

```
<cell>
      <name>__builtin_fabs</name>
      <operation operation_name="fabs" supported_types="REAL:64"/>
      <operation operation_name="abs_expr" supported_types="REAL:64"/>
      <circuit>
        <component_o id="__builtin_fabs">
          <description>This component is part of the BAMBU/PANDA IP LIBRARY</description>
          <copyright>Copyright (C) 2004-2011 Politecnico di Milano</copyright>
          <authors>Fabrizio Ferrandi &lt;ferrandi@elet.polimi.it&gt;</authors>
          <license>PANDA_GPLv3</license>
          <structural_type_descriptor id_type="__builtin_fabs"/>
          <port_o id="clock" dir="IN" is_clock="1">
            <structural_type_descriptor type="BOOL" size="1"/>
          </port_o>
          <port_o id="reset" dir="IN">
            <structural_type_descriptor type="BOOL" size="1"/>
          </port_o>
          <port_o id="start_port" dir="IN">
            <structural_type_descriptor type="BOOL" size="1"/>
          </port_o>
          <port_o id="X" dir="IN">
            <structural_type_descriptor type="REAL" size="1" />
          </port_o>
          <port_o id="sel_fabs" dir="IN">
            <structural_type_descriptor type="BOOL" size="1"/>
          </port_o>
          <port_o id="sel_abs_expr" dir="IN">
            <structural_type_descriptor type="BOOL" size="1"/>
          </port_o>
          <port_o id="done_port" dir="OUT">
            <structural_type_descriptor type="BOOL" size="1"/>
          </port_o>
          <port_o id="R" dir="OUT">
            <structural_type_descriptor type="REAL" size="1"/>
          </port_o>
          <NP_functionality LIBRARY="__builtin_fabs X R" VERILOG_PROVIDED="assign R = {1&apos;b0,X[BITSIZE_X-2:0]};"/>
        </component_o>
      </circuit>
</cell>
```

Template component description

```
    <template>
      <name>plus_expr_FU</name>
      <operation operation_name="plus_expr" supported_types="INT:*|UINT:*" />
      <circuit>
        <component_o id="plus_expr_FU">
          <description>This component is part of the BAMBU/PANDA IP LIBRARY</description>
          <copyright>Copyright (C) 2004-2011 Politecnico di Milano</copyright>
          <authors>Fabrizio Ferrandi &lt;ferrandi@elet.polimi.it&gt;</authors>
          <license>PANDA_GPLv3</license>
          <structural_type_descriptor id_type="plus_expr_FU"/>
          <port_o id="in1" dir="IN">
            <structural_type_descriptor type="VECTOR_BOOL" size="1" vector_size="1"/>
            <connected_objects/>
          </port_o>
          <port_o id="in2" dir="IN">
            <structural_type_descriptor type="VECTOR_BOOL" size="1" vector_size="1"/>
            <connected_objects/>
          </port_o>
          <port_o id="out1" dir="OUT">
            <structural_type_descriptor type="VECTOR_BOOL" size="1" vector_size="1"/>
            <connected_objects/>
          </port_o>
          <NP_functionality LIBRARY="plus_expr_FU in1 in2 out1" VERILOG_PROVIDED="assign out1 = in1 +
in2;" VHDL_PROVIDED="begin\nout1 &lt;= std_logic_vector(resize(signed(in1) + signed(in2),
BITSIZE_out1));"/>
        </component_o>
      </circuit>
    </template>
```

Bus based integration of accelerators



- Very similar to Wishbone Bus
- Synthetized accelerators may have a master, a slave interface or both

Accelerator verification

- bambu HLS tool is able to pretty-print the IR as C code
- Direct execution of a program based on this code could be used to produce expected functional values
- Simulation of the HDL description of the accelerator is done comparing the expected results produced by the C code and the values obtained by the hardware simulation
  - Testbench generation done automatically
  - Testbench generation could be customized through XML files
- Hardware simulators currently supported
  - ICARUS Verilog: http://iverilog.icarus.com/
  - ISIM: Xilinx Simulator
  - Modelsim from Mentor

Regression tests currently used

- CHStone http://www.ertl.jp/chstone/
  - DFADD, DFMUL, DFDIV, DFSIN: Double-precision floating-point
  - MIPS: Simplified MIPS processor
  - ADPCM: Adaptive differential pulse code modulation decoder and encoder
  - GSM: Linear predictive coding analysis of global system for mobile communications
  - JPEG: JPEG image decompression
  - MOTION: Motion vector decoding of the MPEG-2
  - AES: Advanced encryption standard
  - BLOWFISH: Data encryption standard
  - SHA: Secure hash algorithm
- Subset of GCC regression suite: 777 over a total of 1100

Accelerators synthesis

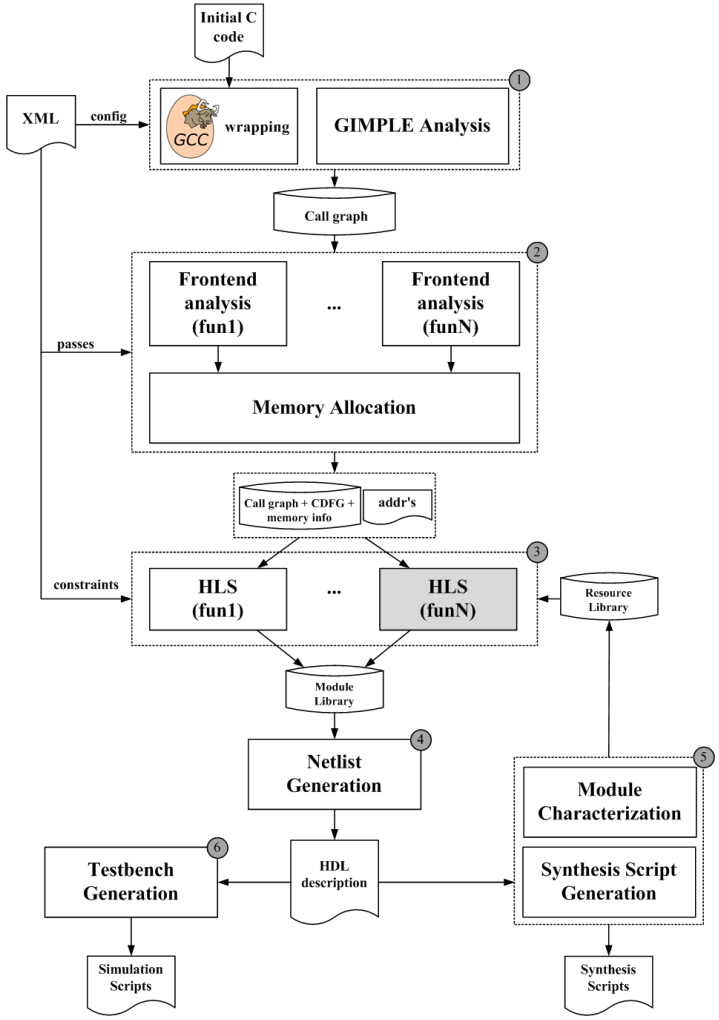
- Automatic generation of synthesis scripts based on XML configuration for different tool flows:
- FPGA:
  - Xilinx ISE
  - Altera Quartus
- ASIC
  - Synopsys Design Compiler

bambu at a glance

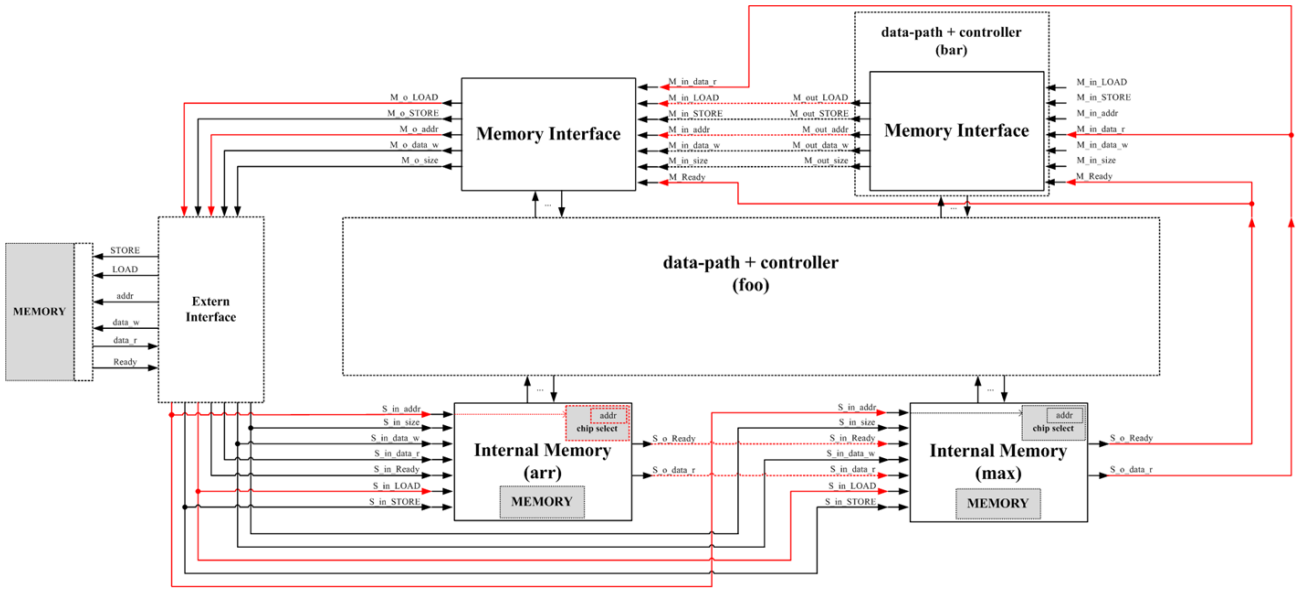
Framework Usage and Demonstrations

Overall Structure

- Configuration possible via:
  - Command-line
  - XML file
- Compiler-like interface
- Modular structure
  - Easy to be extended
- Function-based synthesis
  - External interface “similar” to SW functions
  - Easy to be integrated in MPSoC



Hardware core architecture



- Dynamic resolution of addresses
  - Master/slave chains to avoid tri-states
  - Parallel accesses to local (heterogeneous) memories
  - Interface with external memory
- Synthesis can be performed at function level
  - Global netlist is generated by composing function modules

Example of *bambu* configuration

- Possibility to specify the target device and the synthesis options

```
/opt/panda/bin/bambu --device-name=xc5vlx50,-3,ff1153 module.c

/opt/panda/bin/bambu --target-device-type=ASIC module.c

/opt/panda/bin/bambu --clock-period=7.5 module.c
```

- Target device represented in XML, with proper synthesis options and library characterization

```
<?xml version="1.0"?>
<target>
  <device>
    <vendor value="Xilinx"/>
    <family value="Virtex-5"/>
    <model value="xc5vlx50"/>
    <package value="ff1153"/>
    <speed_grade value="-3"/>
  </device>
  <technology>
    <library>
      <cell>
        <name>mult_expr_FU_64_64_64</name>
        <attribute name="area" value_type="float64">215</attribute>
        <template name="mult_expr_FU" parameter="64 64 64"/>
        <operation operation_name="mult_expr" supported_types="INT:64" execution_time="10.127"/>
      </cell>
      ...
```

Integration with GNU GCC

- Compiler-like interface
  - Possibility to specify compiler directives/flags to be passed to GCC
  - Synthesis performed *after* compiler optimizations

```
-O<num>                  Enable a particular optimization level (default=0).
-f<step>                 Enable or disable an optimization step.
-I<path>                 Specify a path where headers are stored.
-W=<warning>             Specify a warning passed to the gcc.
-E                       Enable preprocessing mode of gcc.
--std=<standard>         Assume that the input sources are for <standard> (default=gnu89).
-D<name>                 Predefine name as a macro, with definition 1.
-U<name>                 Remove existing definition for macro <name>.
--param <name>=<value>   Set the amount <value> for the parameter <name> that could be used
                           for some optimizations.
-l<library>              Search the library named <library> when linking.
-L<dir>                  Add directory <dir> to the list of directories to be searched for -l.
```

Examples

```
/opt/panda/bin/bambu -O1 gsm.c

/opt/panda/bin/bambu -O4 -fwhole-program gsm.c
```

Synthesis

- Scripts are automatically generated based on the target device type
  - FPGA (Xilinx ISE, Altera Quartus)
  - ASIC (Synopsys Design Compiler, …)

- Synthesis flow can be easily customized via XML file

```
<?xml version="1.0"?>
<synthesis>
   <flow name="Synthesis">
      <step id="xst" config="Synthesis"/>
      <step id="ngdbuild" config="Synthesis"/>
      <step id="map" config="Synthesis"/>
      <step id="trce" config="pre_layout"/>
      <step id="par" config="Synthesis"/>
      <step id="trce" config="post_layout"/>
   </flow>
   <flow name="Characterization">
      <step id="xst" config="Synthesis"/>
   </flow>
   <xst config="Synthesis">
      <param name="intstyle" value="${__ise_style__}"/>
      <param name="ofn" value="${__xst_log_file__}"/>
      <set name="tmpdir" value="${__xst_tmpdir__}"/>
      <set name="xsthdpdir" value="${__xst_hdpdir__}"/>
      <cmd name="run">
      ...
```

Script Generation

- One script is then generated at the end to run the synthesis
  - Internal wrapping to start the synthesis and analyze the results
  - Possibility to re-run the synthesis in case of external/manual modifications

```
#!/bin/bash
############################################################
#     Automatically generated by the PandA framework       #
############################################################

# COMPONENT: main
. /opt/Xilinx/13.2/ISE_DS/settings64.sh >& /dev/null

# STEP: xst
xst -ifn HLS_output/Synthesis/xst/xst.tcl -intstyle silent -ofn HLS_output/Synthesis/xst/main.log

# STEP: ngdbuild
ngdbuild -quiet -intstyle silent -dd HLS_output/Synthesis/ngdbuild -nt timestamp -p xc5vlx50ff1153-3
    HLS_output/Synthesis/xst/main.ngc HLS_output/Synthesis/ngdbuild/main.ngd

# STEP: map
map -w -detail -logic_opt on -ol std -global_opt area -equivalent_register_removal on -mt off -cm area -ir off
    -pr off -lc auto -p xc5vlx50ff1153-3 -o HLS_output/Synthesis/map/main.ncd
    HLS_output/Synthesis/ngdbuild/main.ngd HLS_output/Synthesis/map/main.pcf

...
```

Simulation

- A dedicated testbench is generated for each synthesis
- Possibility to execute different simulations
  - Computing the number of cycles varying the memory latency
  - Functional validations, including initialization of external memory

Application code

```
void gcd (int xi, int yi, int *ou);
```

Testbench specification

```
<?xml version="1.0"?>
<function>
   <testbench xi="8" yi="6"/>
   <testbench xi="5" yi="1"/>
   <testbench xi="1" yi="5"/>
   <testbench xi="3" yi="5"/>
   <testbench xi="6" yi="10"/>
</function>
```

Simulation Scripts

- Unique file generated for the selected simulator
  - Xilinx ISIM, Mentor ModelSim, Verilog Icarus

```
#!/bin/bash
############################################################
#     Automatically generated by the PandA framework       #
############################################################

# COMPONENT: gcd
. /opt/Xilinx/13.2/ISE_DS/settings64.sh >& /dev/null

#configuration
#setting up the simulation
fuse -intstyle silent -lib simprims_ver -lib unisims_ver -lib unimacro_ver -lib
    xilinxcorelib_ver -lib secureip -nodebug -o HLS_output/isim_beh/gcd_isim.exe -prj
    HLS_output/isim_beh/gcd.prj work.gcd_tb >& HLS_output/isim_beh/fuse.log

#performing the simulation
HLS_output/isim_beh/gcd_isim.exe -tclbatch HLS_output/isim_beh/isim.command >&
    HLS_output/isim_beh/gcd_isim.log
```

Future works

- Integrate design space exploration approaches: explore different architectural solutions
  - simultaneously reduce area/latency
  - obtain multiple implementations (i.e., trade-offs)
  - take into account all architectural contributions to the final design (e.g., cost of interconnection elements)
- Better component characterization: pipelining, memory customization
- Integrate accelerators with soft processors to build a complete SOC
  - WISHBONE bus support
  - XILINX supported bus

THANK YOU!

GPL v3 source code available at

http://panda.dei.polimi.it