POLITECNICO DI MILANO

Il framework PandA: software libero per la programmazione di FPGA

Fabrizio Ferrandi

Politecnico di Milano
Dipartimento di Elettronica ed Informazione
fabrizio.ferrandi@elet.polimi.it

Outline

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

PandA project objectives

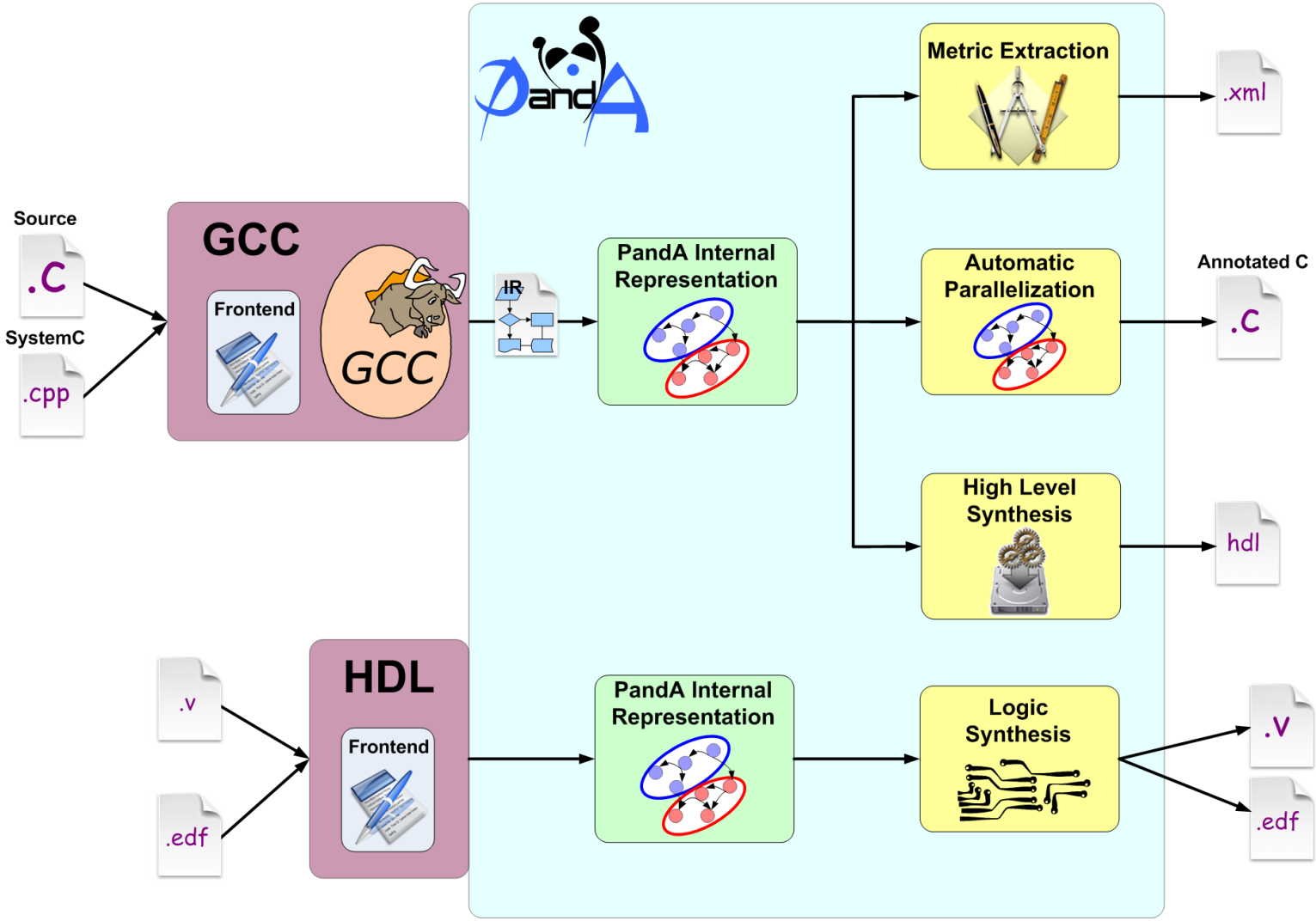
- Definition of design methodologies and tools;
- Design of system architectures;

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

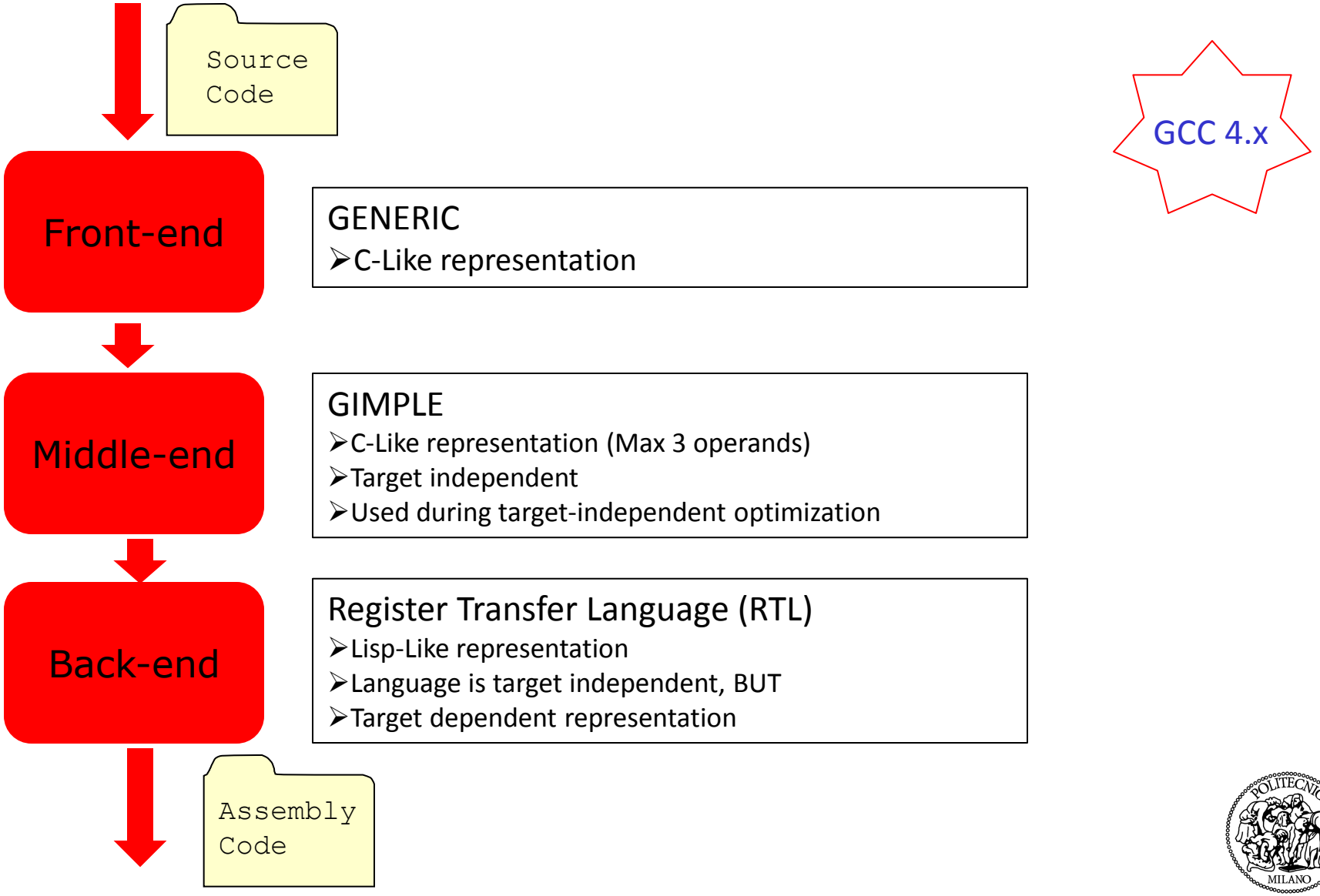
Why free HW

- Are we classical computer hackers, skilled in SW development and with 100% control of our PCs and workstations?
  - Not too surprisingly, that is the perspective from which RMS wrote the GPL.
  - RMS defined freedom as the ability of people like himself to get read-write access to code.
  - How does this definition map to today's world?
  - How much transparent are today systems?
    - HW vs SW

Framework Overview



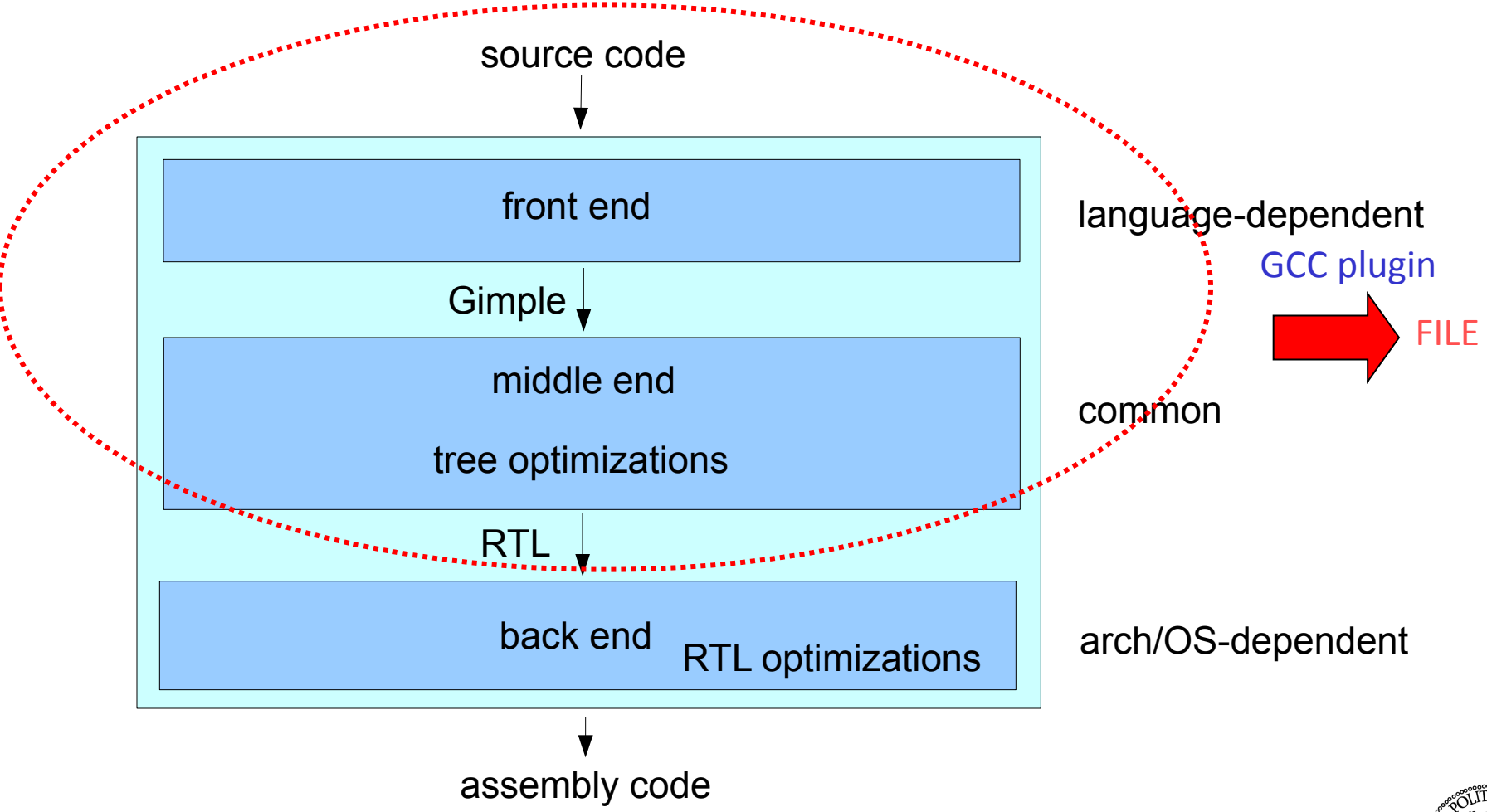
GNU GCC Internal Representation



GCC middle-end optimizations

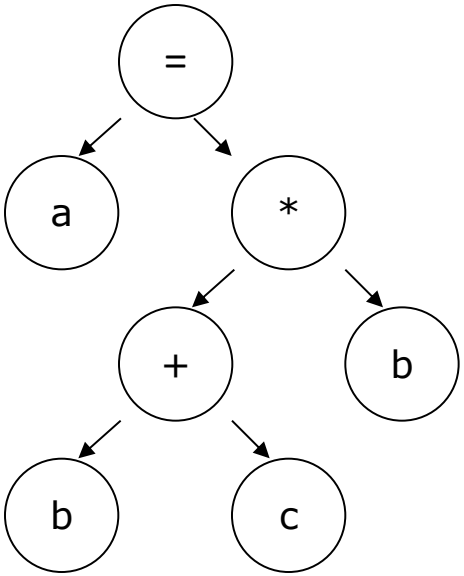
- building the SSA representation
- expanding vector to scalar
- may-alias information
- dead code elimination
- partial redundancy elimination
- full redundancy elimination
- forward propagation of expressions
- SSA copy renaming
- global value numbering
- sparse conditional constant propagation
- scalar 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

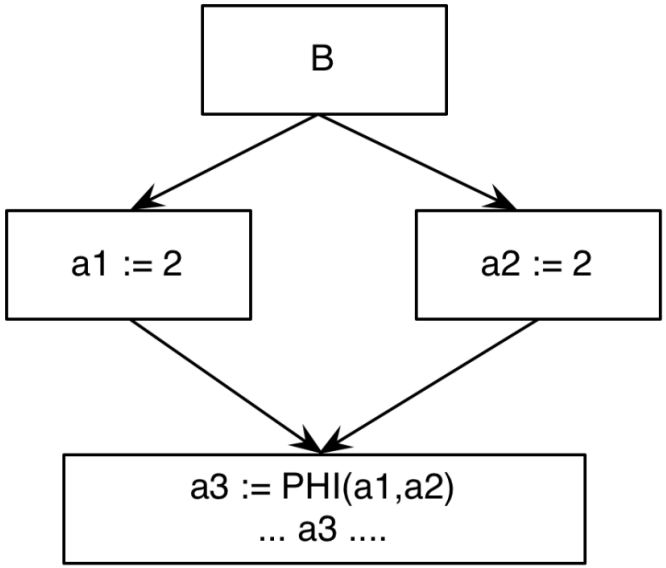
@10 gimple_modify_stmt	**op0: @29**	**op1: @20**	
@18 plus_expr	**type: @7**	**op: @30**	**op: @31**
@20 mult_expr	**type: @7**	**op: @18**	**op: @30**

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

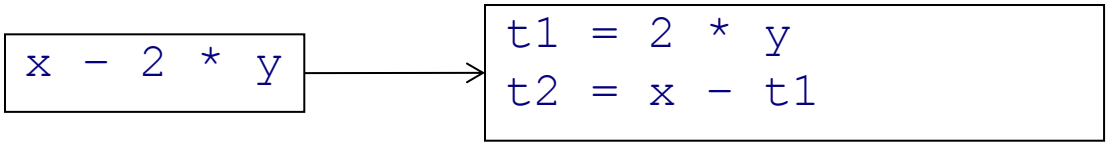
```
if B then
   a1 := 1
else
   a2 := 2
End
a3 := PHI(a1,a2)
   … a3 …
```



- 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

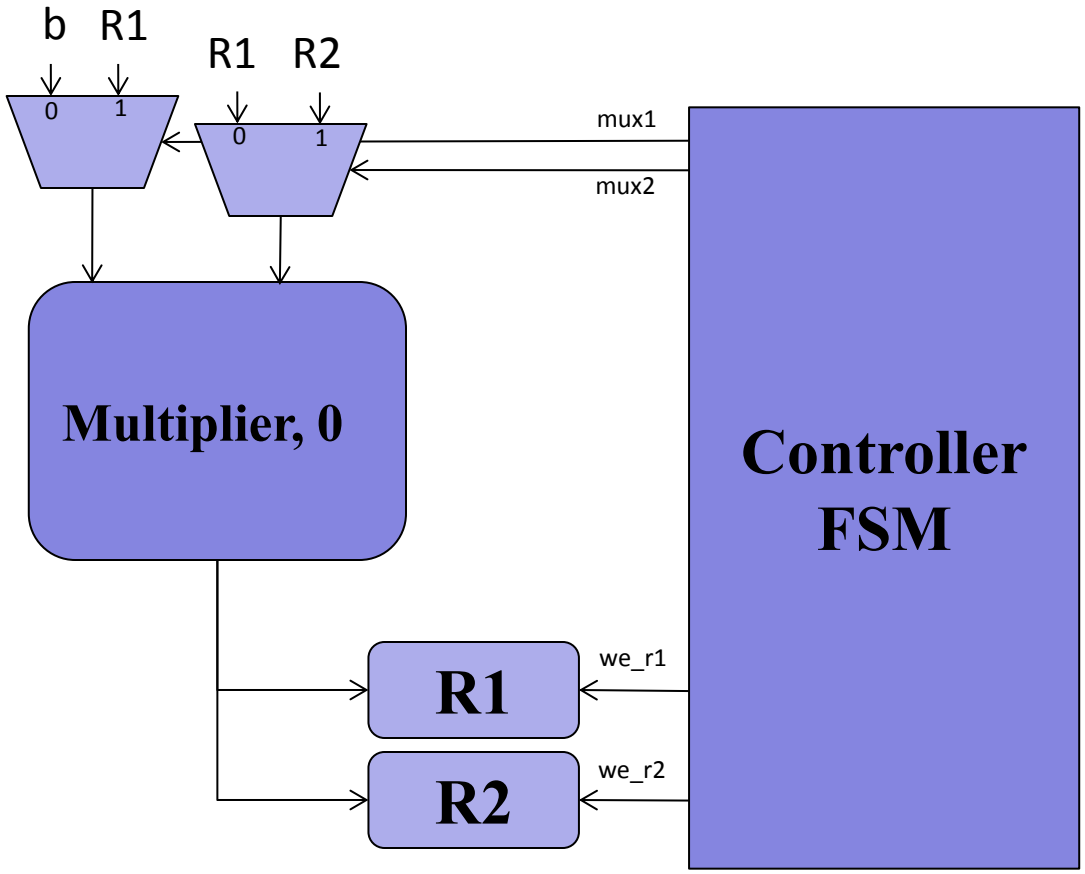
assignments	x = y op z
	x = op y
	x = y[i]
	x = y
branches	goto L
conditional branches	if x rel op y goto L
procedure calls	param x param y call p
address and pointer assignments	x = &y *y = z

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
  - Function allocation
  - 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)

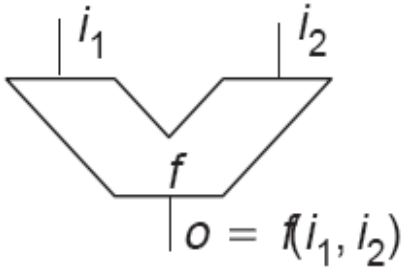
Target Architecture

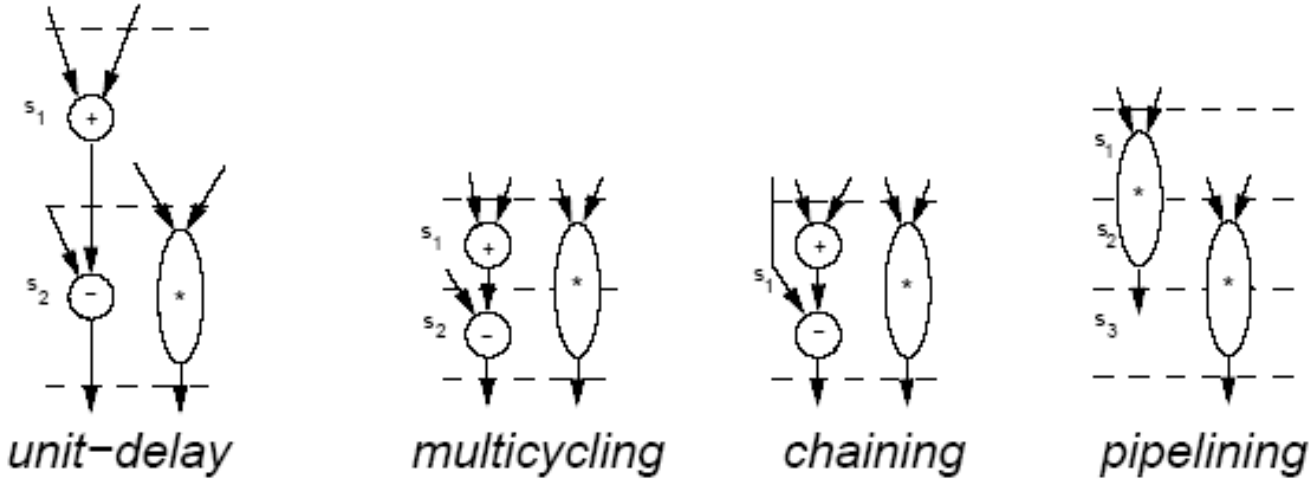
Portion of the Datapath



Functional resources

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





Functional resources

- Resource library composed of about 320 components
- 155 components are actually templates:
  - Parametrizable with respect to the operation data sizes (8, 16, 32, 64)
- Several builtins are available:
  - *exit*, *abort*, *printf*, *puts*, *putchar* for debugging purposes
  - *abs*, *memcpy*, *memset*, *memcmp, malloc, free*
  - *libm functions: sinf, cosf, cbrt, acosh, …*
- *Libm functions, malloc* and *free* supported through a C based library
- Floating point modules generated by exploiting FloPoCo library or the SoftFloat library
- Supported pipelined, multicycling and unbounded functional resources
- Different memory models
- Resource library described in XML or in C
  - 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

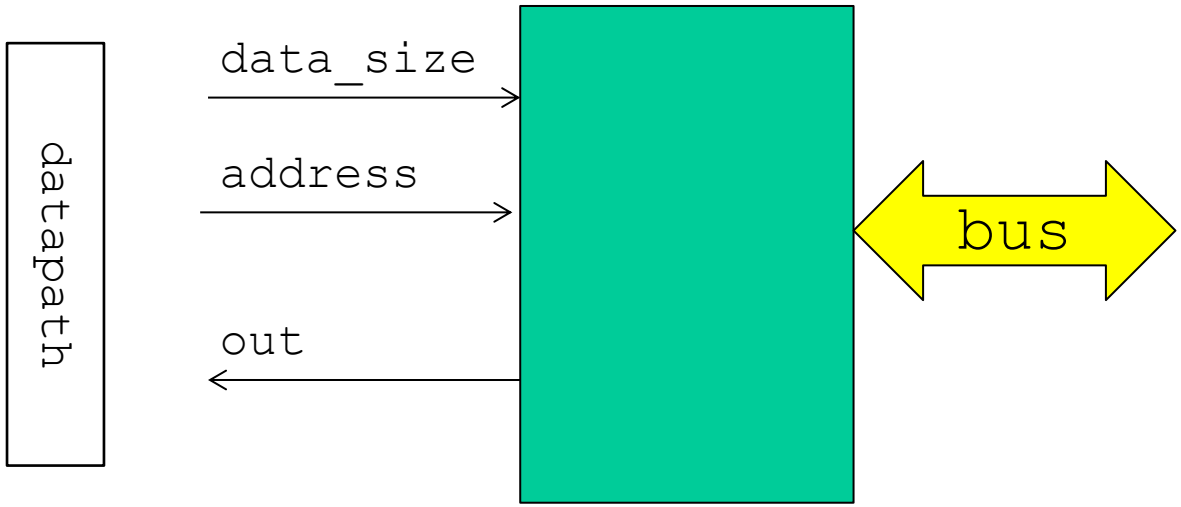
```
<template>
  <name>plus_expr_FU</name>
  <operation operation_name="plus_expr" supported_types="INT:*|UINT:*" />
  <circuit>
    <component_o id="plus_expr_FU">
      <description>This component is part of the BAMBU/PANDA IP LIBRARY</description>
      <copyright>Copyright (C) 2004-2011 Politecnico 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>
```

Memory architecture: support dynamic pointer resolution

- Previous approaches do not support dynamic pointer resolution
- Proposed approach:
  - Define an address for any
    - global scalar/aggregate variable,
    - local aggregate variable
    - local scalar variable used as argument of operator &
- Now pointers may be stored in standard registers
- Loads and stores can be implemented as standard resources

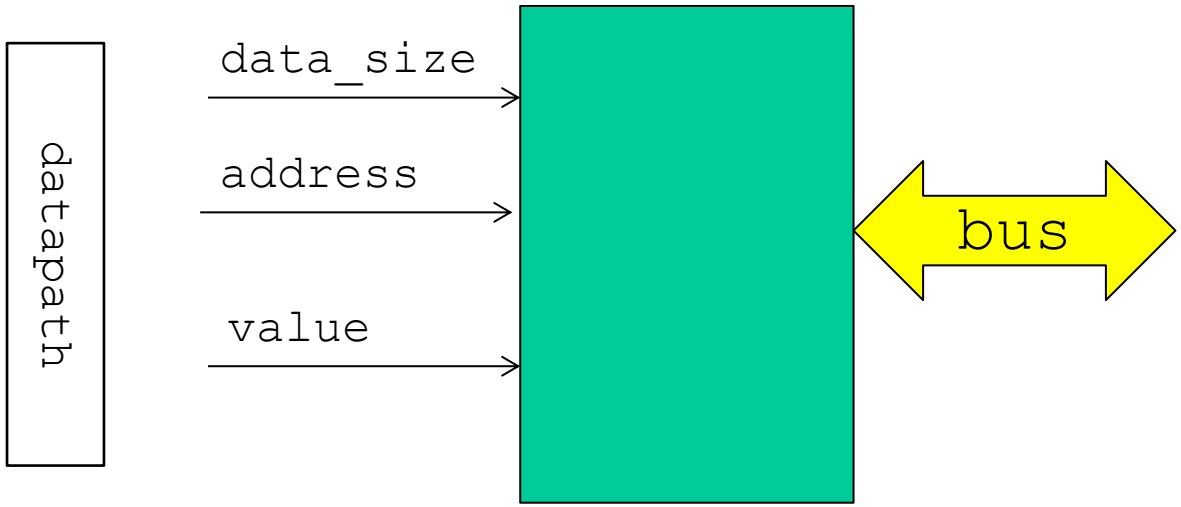
Support of loads

- `temp = *p;`



Support of stores

- `t[n] = temp;`



Memory controller example in Verilog

```
// IN
  input start_port;
  input [BITSIZE_in1-1:0] in1;
  input [BITSIZE_in2-1:0] in2;
  input [BITSIZE_in3-1:0] in3;
  input sel_LOAD;
  input sel_STORE;
```

datapath side

```
  input Min_oe_ram;
  input Min_we_ram;
  input [BITSIZE_Min_addr_ram-1:0] Min_addr_ram;
  input [BITSIZE_M_Rdata_ram-1:0] M_Rdata_ram;
  input [BITSIZE_Min_Wdata_ram-1:0] Min_Wdata_ram;
  input [BITSIZE_Min_data_ram_size-1:0] Min_data_ram_size;
  input M_DataRdy;
```

bus side

```
// OUT
  output done_port;
  output [BITSIZE_out1-1:0] out1;
```

datapath side

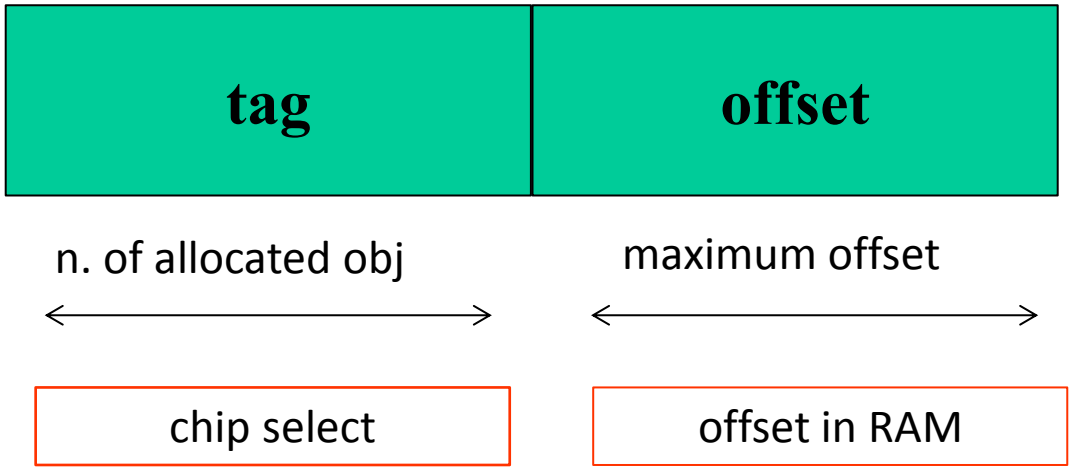
```
  output Mout_oe_ram;
  output Mout_we_ram;
  output [BITSIZE_Mout_addr_ram-1:0] Mout_addr_ram;
  output [BITSIZE_Mout_Wdata_ram-1:0] Mout_Wdata_ram;
  output [BITSIZE_Mout_data_ram_size-1:0] Mout_data_ram_size;
```

bus side

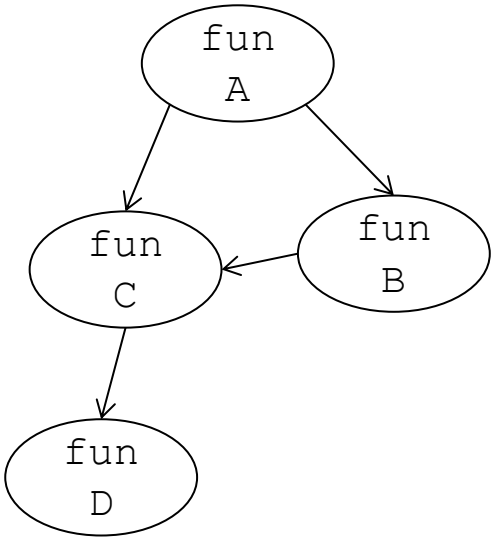
```
  assign Mout_addr_ram = (sel_LOAD || sel_STORE) ? in2 : Min_addr_ram;
  assign Mout_oe_ram = sel_LOAD ? 1'b1 : Min_oe_ram;
  assign Mout_we_ram = sel_STORE ? 1'b1 : Min_we_ram;
  assign out1 = M_Rdata_ram[BITSIZE_out1-1:0];
  assign Mout_Wdata_ram = sel_STORE ? in1 : Min_Wdata_ram;
  assign Mout_data_ram_size = sel_STORE || sel_LOAD ? in3[BITSIZE_in3-1:0] : Min_data_ram_size;
  assign done_port = M_DataRdy && (sel_STORE || sel_LOAD);
```

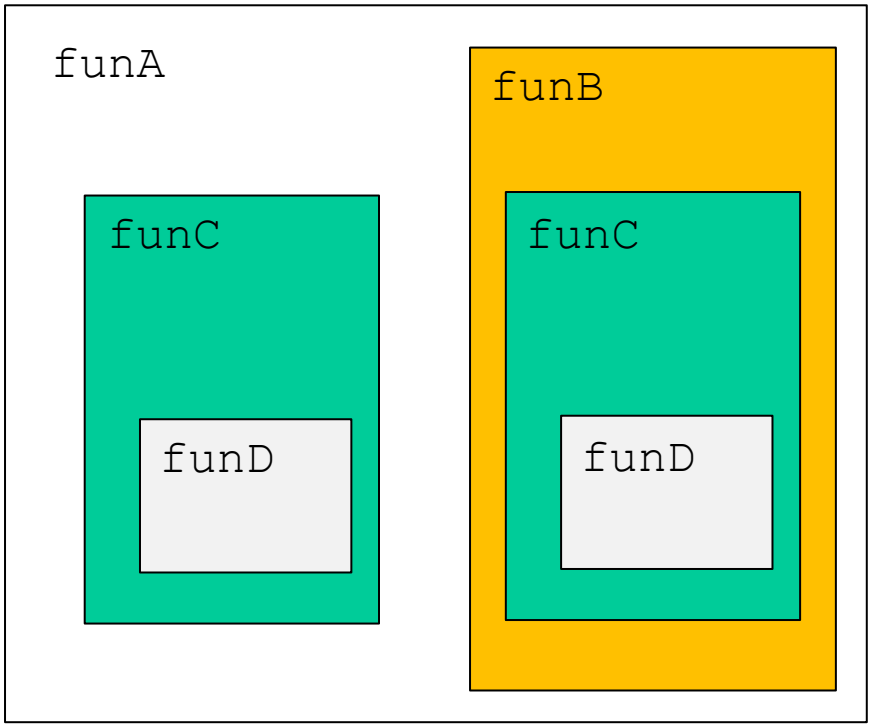
Address encoding

- Contiguous address allocation or tag based
- Tag based can simplify the control logic



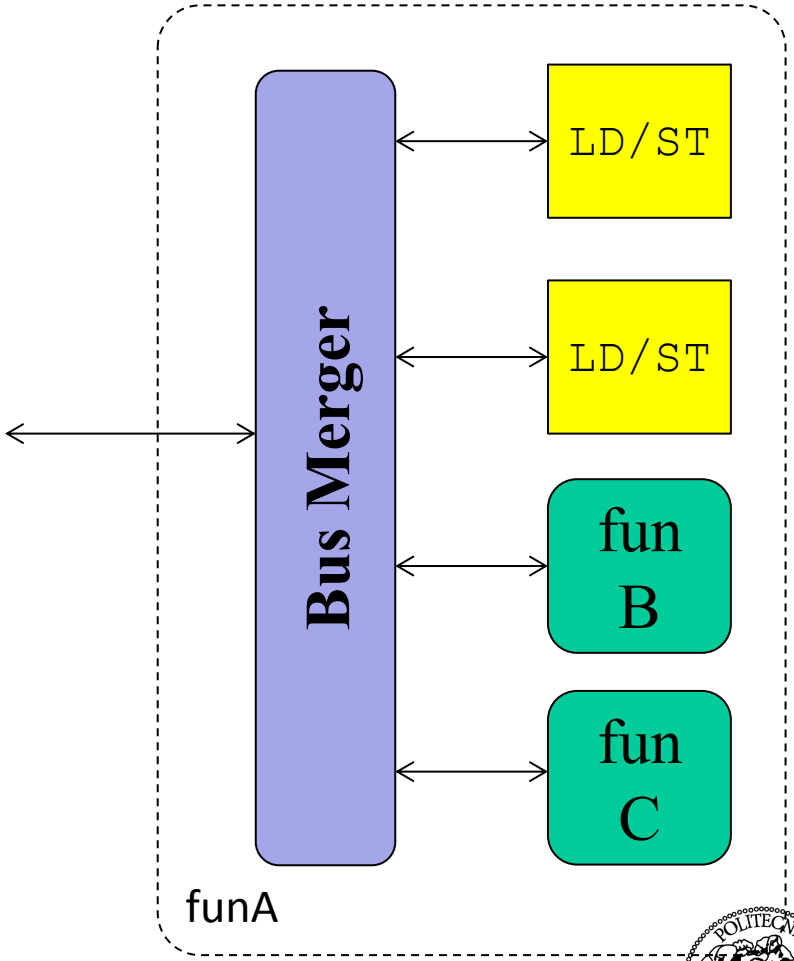
Bus induced by the call graph





Loads/Stores vs bus

- Bus merger implemented with ORs instead of MUXes when functions are executed in a mutual exclusion way



Example 2

```
int arr[2] = {1,2};
void bar(int* a, int b, int *c)
{
  int d;
  *c = 0;
  for (d = 0; d < b; d++)
    if (*(a+d) > *c) *c = a[d];
}
void foo(int a, int* e)
{
  int max = 0;
  bar(arr, 2, &max);
  *e = a + max;
}
```

Example 2: compiler restructuring

```
int arr[2] = {1,2};
void bar(int* a, int b, int *c) {
  int d;
  *c = 0;
  for (d = 0; d < b; d++) {
    int * ptr_4 = a + d;
    int tmp_1 = *(ptr_4);
    int tmp_2 = *c;
    if (tmp_1 > tmp_2) {
      *c = tmp_1;
    }
  }
}
```

```
void foo(int a, int* e)
{
  int max, max_1, tmp_6;
  max = 0;
  bar(arr, 2, &max);
  max_1 = max;
  tmp_6 = a + max_1;
  *e = tmp_6;
}
```

arr & max allocated on memories

in green loads and in blu stores

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/ (free sw)
  - VERILATOR: http://www.veripool.org/wiki/verilator (free sw)
  - ISIM: Xilinx Simulator
  - XSIM: 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: 800

Accelerators synthesis

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

General features

- ANSI C support
  - Complete with few exceptions: recursive functions
- Support of single and double precision floating point computation
- Unaligned memory accesses and dynamic pointers resolution
- GCC compilers supported: v4.5, v4.6, v4.7, v4.8 and v4.9
  - All gcc options can be passed to bambu
- Support of GCC vectorization
- Linux distributions supported:
  - Ubuntu 12.04LTS, 13.04, 13.10, 14.04LTS
  - Debian 7.3 (Wheezy), unstable (Sid)
  - CentOS/Scientific Linux
  - ArchLinux

Demo

Future works

- Support of OpenMP, pthread
- Function sharing
- Support of existing development FPGA boards
- Bitwidth analysis
- Cooperation with CERN hardware group (open hardware repository www.ohwr.org)
- Yosys integration: http://www.clifford.at/yosys/

THANK YOU!

GPL v3 source code available at

http://panda.dei.polimi.it