Design, Automation & Test in Europe
12-16 March, 2012 - Dresden, Germany
The European Event for Electronic System Design & Test

Bambu: A Free Framework for the High-Level Synthesis of Complex Applications

Christian Pilato and Fabrizio Ferrandi

Politecnico di Milano – Dipartimento di Elettronica ed Informazione – Milano (Italy)

Introduction and Main Objectives

Bambu is a free framework to assist the designer during the high-level synthesis of complex applications, aiming at supporting most of the C constructs (e.g., function calls and sharing of the modules, pointer arithmetic and dynamic resolution of memory accesses, accesses to array and structs, parameter passing either by reference or copy, …). *Bambu* is written in C++ and it can be freely downloaded under GPL license.

1. Compiling the Source Code with the GCC Compiler

Bambu has a compiler-based interface to interface with the GNU C Compiler (GCC) ver. 4.5 and build the internal representation in SSA form of the initial C code.

```
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;
}
```

Restructuring of the code to explicit memory accesses.

```
@1:  int  arr[2]  =  {1,2};

void  bar(int*  a,  int  b,  int  *c){    //@2  (@3,  @4,  @5)
@6:  int  d;
@7:  *c  =  0;
@8:  for  (d  =  0;  d  <  b;  d++){
@10:     int  tmp_1  =  *(a+d);
@11:     int  tmp_2  =  *c;
@12:     if  (tmp_1  >  tmp_2){
@14:         int  tmp_3  =  *(a+d);
@15:         *c  =  tmp_3;
@16:     }
@17:  }
}

void  foo(int  a,  int*  e)  //@18  (@19,  @20){
@21:  int  max,  max_2,  max_1  =  0;
@22:  max  =  max_1;
@23:  bar(arr,  2,  &max);
@24:  max_2  =  max;
@25:  *e  =  a  +  max_2;
}
```

2. Frontend Analysis

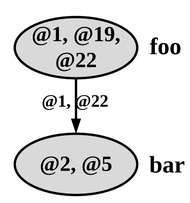
The analysis of the intermediate representation allows to:

- create the call graph of the entire application;
- create the graph-based representation of each function, after GCC optimizations;
- identify variables, function parameters, memory accesses, data types, ...

Function	Operation	Variables
bar	@7	@5
	@10	@2
	@11	@5
	@14	@2
	@15	@5
foo	@22	@21
	@23	@1, @21
	@24	@21
	@25	@20

Arrays and pointer variables refer to memory by definition.

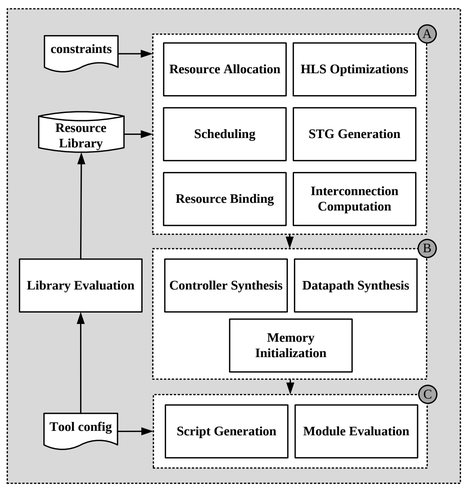
Call Graph with memory variables



Variables given by reference to the sub-functions require a memory location.

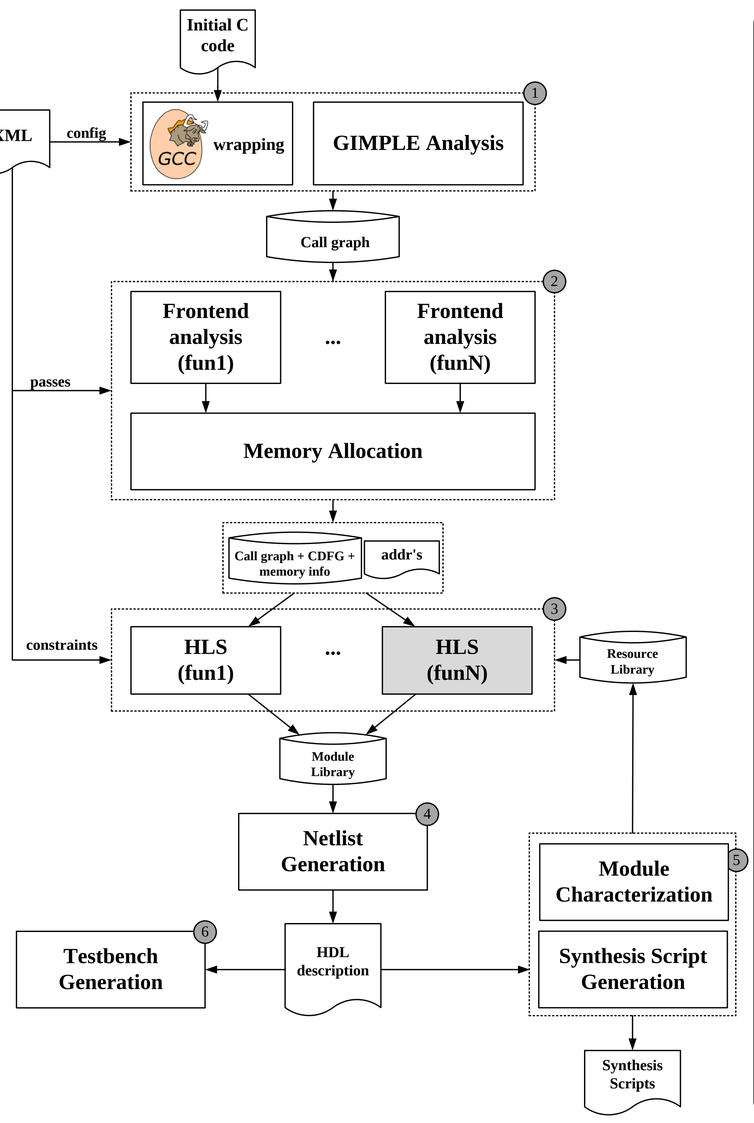
3. High-Level Synthesis of the Functions

We perform the High-Level Synthesis of each function separately



- Custom mux-based architectures based on the dimension of the data types, aiming at reducing the number of flip-flops and bit-level multiplexers.
- Modular structure, easy to be extended with new algorithms and methodologies.
- Possibility to customize any of the algorithms.

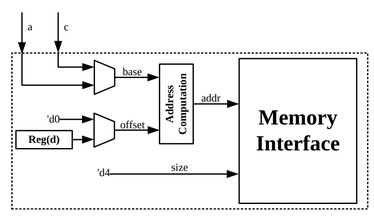
The technology-aware characterization of the library allows to perform operation chaining and correct enabling of register writing for multi-cycling units.



Memory Allocation and Architecture

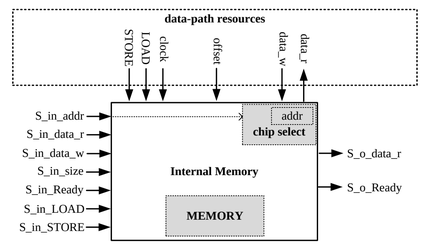
The memory allocation determines where to allocate the different variables with respect to each function:

- internal variables are allocated to dedicated heterogeneous memories, internal to the datapath and directly accessed by the functional units.
- external variables (either external memory or outermost functions) are accessed by a memory interface.

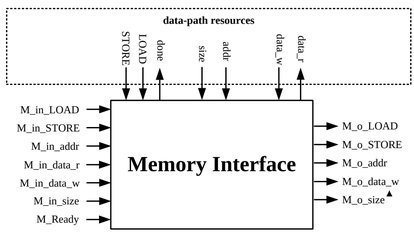


The resulting memory interface has to dynamically resolve the base address and the proper offset, if any.

Access to internal memories: we adopt a chip-select logic to determine if the input address is addressing the memory or not.



Access to external memories: it is connected also to the other interfaces in order to compose a chain and allow the dynamic resolution of the addresses.



4. Netlist Generation

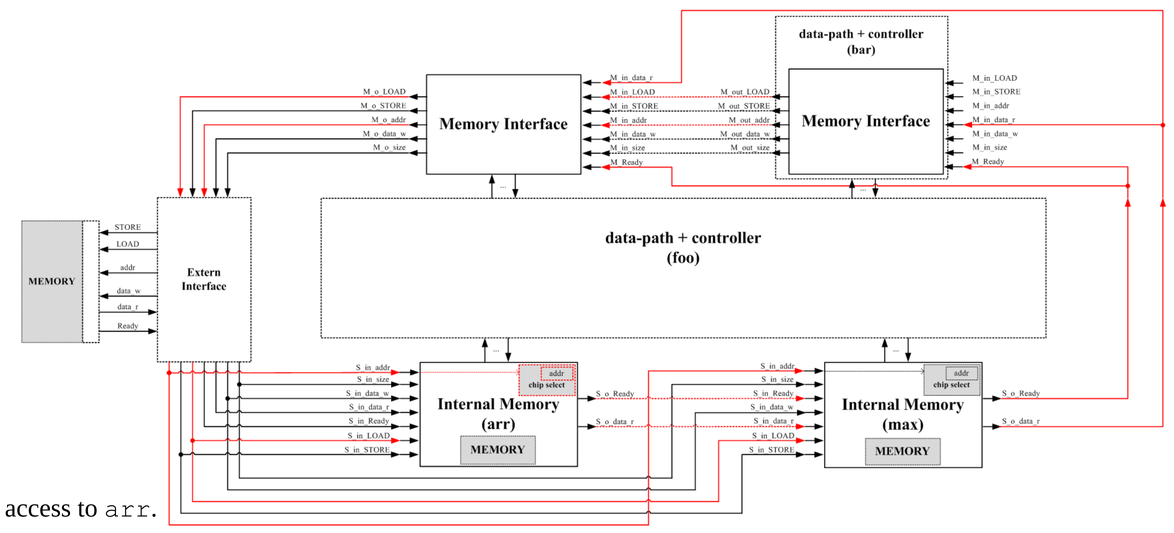
Finally, the global RTL structural description is created by connecting all the modules and the memory interfaces in two master/slave chains:

- the master one produces and propagates the requests;
- the slave one provides the data to the module that performed the request.
- the global external interface closes the chains by determining if the address refers to an memory internal or external to the core.

Only one memory location is active at the same time and thus only one interface will write the data on the corresponding bus, while the others will simply forward them.

- no three-states are required to manage the accesses to the bus.

Bambu interfaces with the FloPoCo library for the generation of floating-point units.



Example of active signals when function `bar` requires an access to `arr`.

5. Generation of Synthesis and Simulation Scripts

Automatic generation of synthesis scripts based on XML configuration for different toolflows:

- FPGA: Xilinx ISE, Altera Quartus
- ASIC: Synopsys Design Compiler

and simulation tools:

- Mentor Modelsim, Xilinx ISIM and Verilog Icarus

This can be also adopted for the characterization of the resource library to have technology-aware details during the High-Level Synthesis.

6. Test-Bench Generation

Generation of test-benches starting from an XML description of the data-set.

```
<?xml version="1.0"?>
<function name="bar">
     <testbench a="1,3,2,5" b="4"/>      #possibility to test the function bar with different arrays; e is not initialized to any value;
</function>
```

Bambu performs:

- the generation of expected values based on the software execution
- the generation of HDL test-bench taking the memory allocation into account
- comparison of the simulation results to verify the execution correctness

Contacts

Fabrizio Ferrandi, Associate Professor, Politecnico di Milano, DEI, ferrandi@elet.polimi.it
Christian Pilato, Post-doc Research Assistant, Politecnico di Milano, DEI, pilato@elet.polimi.it

Download Information and Documentation

Website: http://panda.dei.polimi.it
Mailing List: panda-info@elet.polimi.it