POLITECNICO DI MILANO

Introduction to Bambu

Tutorial @ FPL Conference 2017 – Ghent – Belgium

Fabrizio Ferrandi

Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria
fabrizio.ferrandi@polimi.it

Outline

- Introduction
- Bambu HLS flow
- Synthesis C subset supported by Bambu
- Interfacing Bambu with GCC
- Component generation
- Synthesis of function pointers
- Integration of hand-written components
- Extended language support
  - C++
  - Fortran
  - OpenMP simd

Bambu: an example of modern HLS tools

- HLS tool developed at Politecnico di Milano (Italy) within the PandA framework
  - Available under GPL v3 at
    - http://panda.dei.polimi.it/
    - https://github.com/ferrandi/PandA-bambu
- Example features
  - Front-end Input: interfacing with GCC for parsing C code
    - Complete support for ANSI C (except for recursion)
      - Support for pointers, user-defined data types, built-in C functions, etc..
    - Source code optimizations
      - may alias analysis, dead-code elimination, hoisting, loop optimizations, etc...
  - Target-aware synthesis
    - Characterization of the technology library based on target device
  - Verification
    - Integrated testbench generation and simulation
      - automated interaction with Iverilog, Verilator, Xilinx Isim, Xilinx Xsim, Mentor Modelsim
  - Back-end: Automated interaction with commercial synthesis tools
    - FPGA: Xilinx ISE, Xilinx Vivado, Altera Quartus, Lattice Diamond
    - ASIC: Synopsis Design Compiler

A matter of complexity

- Complexity of embedded systems is constantly growing
  - Designing for FPGAs is mostly an engineering discipline carried out by highly-trained specialists
  - Difficult for a hardware designer when working at RTL (or lower) level(s)
  - Moving to higher levels of abstractions is becoming more and more important
- Example: 1M gates design (study from NEC)
  - RTL description: ~300k lines of code (avg)
  - Behavioral description: ~30k lines of code (avg)
- Writing behavioral code is much "easier"
  - e.g. C/C++ compared to Verilog/VHDL

Design gap (ITRS Roadmap)

- HLS raises the abstraction level and is going to increase the designers productivity
- It is starting to be an option for software engineers to accelerate key application kernels
- Quality of design produced by HLS?
  - Worse than hand-made RTL design Not true anymore!
  - In several cases better than pure software

Why reinventing the wheel?

- From LegUp 1.0:

> Well, it turns out that none of the existing high-level synthesis tools have source code available for researchers. GAUT claims to be open-source but the code is not available for download. xPilot from UCLA is an advanced research tool but only the binary is available and it hasn't been updated since 2007. ROCCC provides an open source eclipse plugin based on SUIF and LLVM but only supports small C programs. Standard C code must be rewritten to work with ROCCC because all function parameters must be structs. Trident uses a very old version of LLVM to interface with an extensive amount of Java code, but unfortunately no longer compiles with the latest version of LLVM.

A bit of history

- PandA framework development started on 2004 as a support research infrastructure for PoliMi in the context of ICODES – FP6-IST EU-funded project
  - Parsing and analysis of TLM 2.0 SystemC descriptions (gcc v.3.5)
- In the hArtes EU-funded project (2006-2010), it was used to
  - Analyzing generic C-based application annotated with pragmas (OpenMP)
  - Extracting parallel tasks
  - Estimating performance of embedded app
  - C-to-c rewriting
- Later, in Synaptic (2009-2013) and in Faster (2011-2014) EU-funded projects, logic- and high-level synthesis has been extended
  - Bambu (HLS tool) was first released in March 2012.
- ESA funded many research on code predictability analysis, performance analysis, and integration of HLS in model-based design flows.

Why free SW or free HW

1. 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's systems?
     - HW vs SW
2. Funding of PandA comes mainly from public institutions

3. … because it's fun

Some References Inspiring Us

- Leon Stok, "Data path synthesis", Integration VLSI Journal, 18(1): 1-71 (1994).
- Jason Cong, Zhiru Zhang, "An efficient and versatile scheduling algorithm based on SDC formulation", DAC 2006, 433-438.
- Florent de Dinechin, Bogdan Pasca: "Designing Custom Arithmetic Data Paths with FloPoCo". IEEE Design & Test of Computers 28(4), 18-27 (2011).
- Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, F. Kenneth Zadeck, "Efficiently Computing Static Single Assignment Form and the Control Dependence Graph", ACM Trans. Program. Lang. Syst. 13(4), 451-490 (1991).
- Preston Briggs, Keith D. Cooper, Timothy J. Harvey, L. Taylor Simpson, "Practical Improvements to the Construction and Destruction of Static Single Assignment Form", Softw., Pract. Exper. 28(8), 859-881 (1998).
- Benoit Boissinot, Florian Brandner, Alain Darte, Benoît Dupont de Dinechin, Fabrice Rastello, "A Non-iterative Data-Flow Algorithm for Computing Liveness Sets in Strict SSA Programs". APLAS 2011, 137-154.

Synthesis per function

- one component per function
  - function interface
  - start and done
  - parameter passing
    - wires
    - memory interaction
- hierarchy based on call graph
  - no-recursion
  - proxy
- option
  - `-fwhole-program`

Example: Multiply & Accumulate


```
Fmin=100Mhz
Device=Xilinx Zynq
```
Constraints

```
int mac(int a, int b, int c)
{
   return a+(b*c);
}
```
C code

Verilog

```
`timescale 1ns / 1ps
module mac(clock, reset, start_port, done_port, a, b, c, return_port);
  // IN
  input clock;
  input reset;
  input start_port;
  input signed [31:0] a;
  input signed [31:0] b;
  input signed [31:0] c;
  // OUT
  output done_port;
  output signed [31:0] return_port;
  // Component and signal declarations
  wire wrenable_reg_0;

  controller_mac Controller_i (.done_port(done_port), .wrenable_reg_0(wrenable_reg_0),
                               .clock(clock), .reset(reset), .start_port(start_port));
  datapath_mac Datapath_i (.return_port(return_port), .clock(clock), .reset(reset),
        .in_port_a(a), .in_port_b(b), .in_port_c(c), .wrenable_reg_0(wrenable_reg_0));

endmodule
```

Example: (FSM) Controller


```
Fmin=100Mhz
Device=Xilinx Zynq
```
Constraints

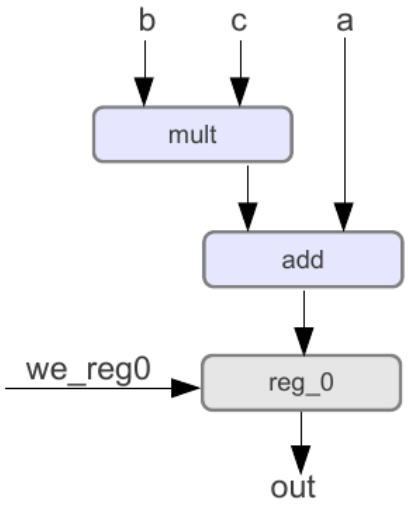
```
int mac(int a, int b, int c)
{
   return a+(b*c);
}
```
C code

```
module controller_mac(done_port, wrenable_reg_0,
                      clock, reset, start_port);
  . . .
  // IN,OUT signals declaration;
  // regs/parameters declaration and init
  . . .

  always @(_present_state or start_port)
  begin
    _next_state = S_0;
    done_port = 1'b0;
    wrenable_reg_0 = 1'b0;
    case (_present_state)
      S_0 :
        if(~start_port)
        begin
          _next_state = S_0;
        end
        else
        begin
          wrenable_reg_0 = 1'b1;
          _next_state = S_1;
        end
      S_1 :
        begin
          _next_state = S_0;
          done_port = 1'b1;
        end
      default :
        begin
          done_port = 1'b0;
          wrenable_reg_0 = 1'b0;
        end
    endcase
  end
endmodule
```
controller_mac

Example: Datapath


```
Fmin=100Mhz
Device=Xilinx Zynq
```
Constraints

```
int mac(int a, int b, int c)
{
   return a+(b*c);
}
```
C code

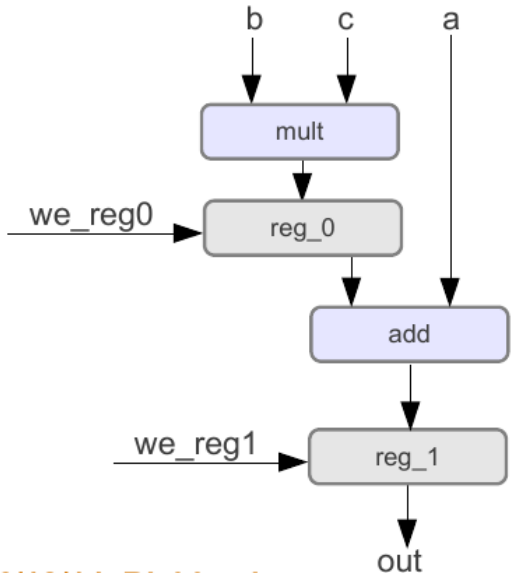



```
`timescale 1ns / 1ps
module datapath_mac(clock, reset, in_port_a,
in_port_b, in_port_c, return_port, wrenable_reg_0);
  // IN
  input clock;
  input reset;
  input signed [31:0] in_port_a;
  input signed [31:0] in_port_b;
  input signed [31:0] in_port_c;
  input wrenable_reg_0;
  // OUT
  output signed [31:0] return_port;
  // Component and signal declarations
  wire signed [31:0] out_mult_expr_FU_fu_mac_1448_1472;
  wire signed [31:0] out_plus_expr_FU_fu_mac_1448_1473;
  wire [31:0] out_reg_0_reg_0;

  mult_expr_FU #(.BITSIZE(32), .PIPE_PARAMETER(0)) fu_mac_1448_1472
    (.out1(out_mult_expr_fu_mac_1448_1472),
     .clock(clock), .in1(in_port_c), .in2(in_port_b));
  plus_expr_FU #(.BITSIZE(32)) fu_mac_1448_1473
    (.out1(out_plus_expr_fu_mac_1448_1473),
    .in1(out_mult_expr_fu_mac_1448_1472),
    .in2(in_port_a));

  register_SARSE #(.BITSIZE_in1(32), .BITSIZE_out1(32))
   reg_0 (.out1(out_reg_0_reg_0), .clock(clock), .reset(reset),
   .in1(out_plus_expr_fu_mac_1448_1473), .wenable(wrenable_reg_0));
  // io-signal post fix
  assign return_port = out_reg_0_reg_0;

endmodule
```
datapath_mac

Example: Datapath, 200 Mhz


```
Fmin=200Mhz
Device=Xilinx Zynq
```
Constraints

```
int mac(int a, int b, int c)
{
   return a+(b*c);
}
```
C code

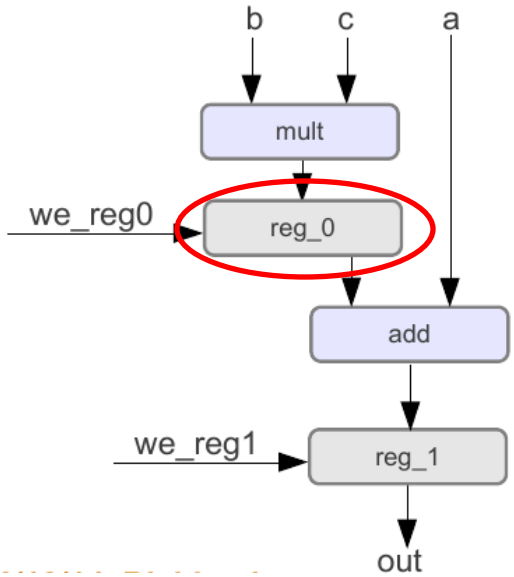



```
`timescale 1ns / 1ps
module datapath_mac(clock, reset, in_port_a, in_port_b,
  in_port_c, return_port, wrenable_reg_1, wrenable_reg_0);

  // IN
  input clock, reset;
  input signed [31:0] in_port_a, in_port_b, in_port_c;
  input wrenable_reg_1, wrenable_reg_0;
  // OUT
  output signed [31:0] return_port;
  // Component and signal declarations
  wire signed [31:0] out_mult_expr_fu_mac_1448_1472;
  wire signed [31:0] out_plus_expr_fu_mac_1448_1473;
  wire [31:0] out_reg_0;
  wire [31:0] out_reg_1;

  mult_expr_FU #(.BITSIZE(32), .PIPE_PARAMETER(4)) fu_mac_1448_1472
   (.out1(out_mult_expr_fu_mac_1448_1472),
   .clock(clock), .in1(in_port_c), .in2(in_port_b));
  plus_expr_FU #(.BITSIZE(32)) fu_mac_1448_1473
   (.out1(out_plus_expr_fu_mac_1448_1473),
   .in1(out_reg_0), .in2(in_port_a));
  register_SARSE #(.BITSIZE_in1(32), .BITSIZE_out1(32)) reg_0
   (.out1(out_reg_0), .clock(clock), .reset(reset),
   .in1(out_mult_expr_FU_fu_mac_1448_1472), .wenable(wrenable_reg_0));
  register_SARSE #(.BITSIZE_in1(32), .BITSIZE_out1(32)) reg_1
   (.out1(out_reg_1_reg_1), .clock(clock), .reset(reset),
   .in1(out_plus_expr_fu_mac_1448_1473), .wenable(wrenable_reg_1));
  // io-signal post fix
  assign return_port = out_reg_1_reg_1;

endmodule
```
datapath_mac

Example: Datapath, 200 Mhz


```
Fmin=200Mhz
Device=Xilinx Zynq
```
Constraints

```
int mac(int a, int b, int c)
{
   return a+(b*c);
}
```
C code

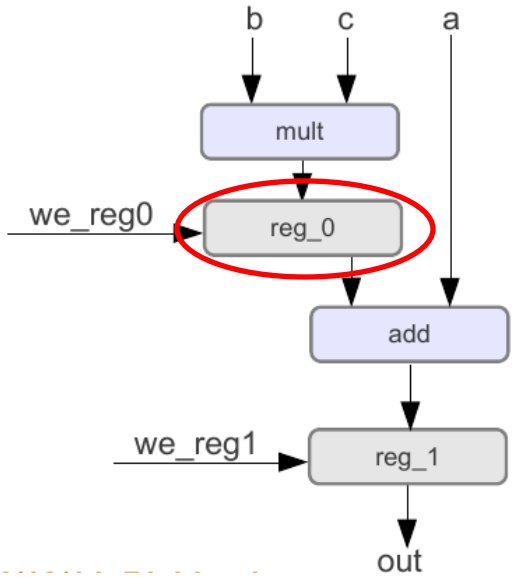



```
`timescale 1ns / 1ps
module datapath_mac(clock, reset, in_port_a, in_port_b,
  in_port_c, return_port, wrenable_reg_1, wrenable_reg_0);

  // IN
  input clock, reset;
  input signed [31:0] in_port_a, in_port_b, in_port_c;
  input wrenable_reg_1, wrenable_reg_0;
  // OUT
  output signed [31:0] return_port;
  // Component and signal declarations
  wire signed [31:0] out_mult_expr_fu_mac_1448_1472;
  wire signed [31:0] out_plus_expr_fu_mac_1448_1473;
  wire [31:0] out_reg_0;
  wire [31:0] out_reg_1;

  mult_expr_FU #(.BITSIZE(32), .PIPE_PARAMETER(4)) fu_mac_1448_1472
   (.out1(out_mult_expr_fu_mac_1448_1472),
   .clock(clock), .in1(in_port_c), .in2(in_port_b));
  plus_expr_FU #(.BITSIZE(32)) fu_mac_1448_1473
   (.out1(out_plus_expr_fu_mac_1448_1473),
   .in1(out_reg_0), .in2(in_port_a));
  register_SARSE #(.BITSIZE_in1(32), .BITSIZE_out1(32)) reg_0
   (.out1(out_reg_0), .clock(clock), .reset(reset),
   .in1(out_mult_expr_FU_fu_mac_1448_1472), .wenable(wrenable_reg_0));
  register_SARSE #(.BITSIZE_in1(32), .BITSIZE_out1(32)) reg_1
   (.out1(out_reg_1_reg_1), .clock(clock), .reset(reset),
   .in1(out_plus_expr_fu_mac_1448_1473), .wenable(wrenable_reg_1));
  // io-signal post fix
  assign return_port = out_reg_1_reg_1;

endmodule
```
datapath_mac

Example: Datapath, 200 Mhz


```
Fmin=200Mhz
Device=Xilinx Zynq
```
Constraints

```
int mac(int a, int b, int c)
{
   return a+(b*c);
}
```
C code

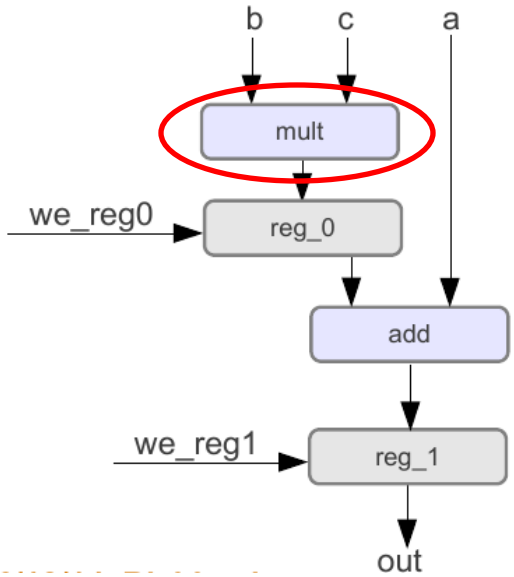



```
`timescale 1ns / 1ps
module datapath_mac(clock, reset, in_port_a, in_port_b,
  in_port_c, return_port, wrenable_reg_1, wrenable_reg_0);

  // IN
  input clock, reset;
  input signed [31:0] in_port_a, in_port_b, in_port_c;
  input wrenable_reg_1, wrenable_reg_0;
  // OUT
  output signed [31:0] return_port;
  // Component and signal declarations
  wire signed [31:0] out_mult_expr_fu_mac_1448_1472;
  wire signed [31:0] out_plus_expr_fu_mac_1448_1473;
  wire [31:0] out_reg_0;
  wire [31:0] out_reg_1;

  mult_expr_FU #(.BITSIZE(32), .PIPE_PARAMETER(4)) fu_mac_1448_1472
   (.out1(out_mult_expr_fu_mac_1448_1472),
   .clock(clock), .in1(in_port_c), .in2(in_port_b));
  plus_expr_FU #(.BITSIZE(32)) fu_mac_1448_1473
   (.out1(out_plus_expr_fu_mac_1448_1473),
   .in1(out_reg_0), .in2(in_port_a));
  register_SARSE #(.BITSIZE_in1(32), .BITSIZE_out1(32)) reg_0
   (.out1(out_reg_0), .clock(clock), .reset(reset),
   .in1(out_mult_expr_FU_fu_mac_1448_1472), .wenable(wrenable_reg_0));
  register_SARSE #(.BITSIZE_in1(32), .BITSIZE_out1(32)) reg_1
   (.out1(out_reg_1_reg_1), .clock(clock), .reset(reset),
   .in1(out_plus_expr_fu_mac_1448_1473), .wenable(wrenable_reg_1));
  // io-signal post fix
  assign return_port = out_reg_1_reg_1;

endmodule
```
datapath_mac

Example: Datapath, 200 Mhz


```
Fmin=200Mhz
Device=Xilinx Zynq
```
Constraints

```
int mac(int a, int b, int c)
{
   return a+(b*c);
}
```
C code




```
`timescale 1ns / 1ps
module datapath_mac(clock, reset, in_port_a, in_port_b,
  in_port_c, return_port, wrenable_reg_1, wrenable_reg_0);

  // IN
  input clock, reset;
  input signed [31:0] in_port_a, in_port_b, in_port_c;
  input wrenable_reg_1, wrenable_reg_0;
  // OUT
  output signed [31:0] return_port;
  // Component and signal declarations
  wire signed [31:0] out_mult_expr_fu_mac_1448_1472;
  wire signed [31:0] out_plus_expr_fu_mac_1448_1473;
  wire [31:0] out_reg_0;
  wire [31:0] out_reg_1;

  mult_expr_FU #(.BITSIZE(32), .PIPE_PARAMETER(4)) fu_mac_1448_1472
   (.out1(out_mult_expr_fu_mac_1448_1472),
   .clock(clock), .in1(in_port_c), .in2(in_port_b));
  plus_expr_FU #(.BITSIZE(32)) fu_mac_1448_1473
   (.out1(out_plus_expr_fu_mac_1448_1473),
   .in1(out_reg_0), .in2(in_port_a));
  register_SARSE #(.BITSIZE_in1(32), .BITSIZE_out1(32)) reg_0
   (.out1(out_reg_0), .clock(clock), .reset(reset),
   .in1(out_mult_expr_FU_fu_mac_1448_1472), .wenable(wrenable_reg_0));
  register_SARSE #(.BITSIZE_in1(32), .BITSIZE_out1(32)) reg_1
   (.out1(out_reg_1_reg_1), .clock(clock), .reset(reset),
   .in1(out_plus_expr_fu_mac_1448_1473), .wenable(wrenable_reg_1));
  // io-signal post fix
  assign return_port = out_reg_1_reg_1;

endmodule
```
datapath_mac

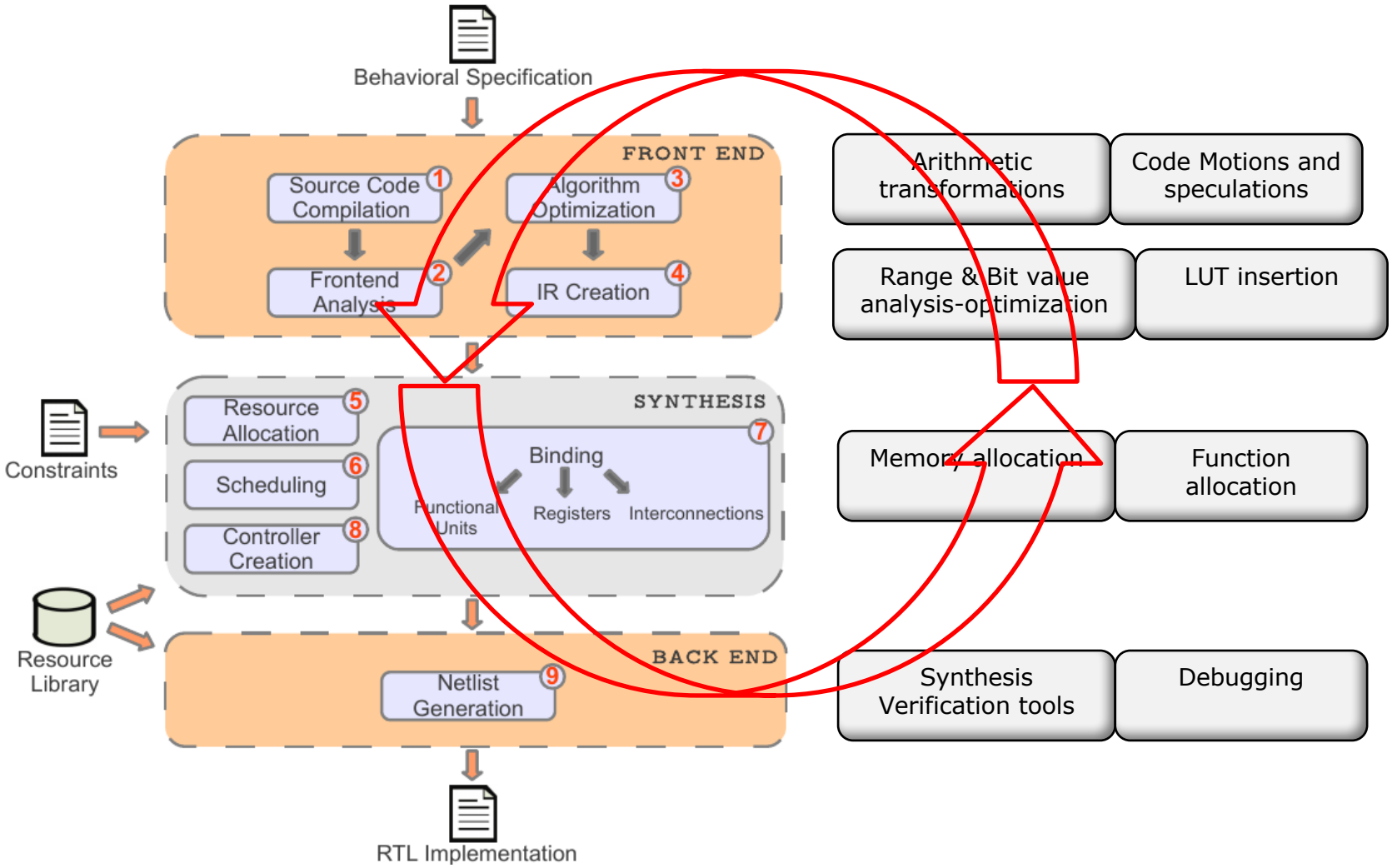
Example: (FSM) Controller, 200 Mhz


```
Fmin=200Mhz
Device=Xilinx Zynq
```
Constraints

```
int mac(int a, int b, int c)
{
   return a+(b*c);
}
```
C code

```
module controller_mac(done_port, wrenable_reg_0,
      wrenable_reg_1, clock, reset, start_port);
  . . .
  // IN,OUT signals declaration;
  // regs/parameters declaration and init
  . . .
  always @(_present_state or start_port)
  begin
    _next_state = S_0;
    done_port = 1'b0;
    wrenable_reg_1 = 1'b0;
    wrenable_reg_0 = 1'b0;
    case (_present_state)
      S_0 :
        if(~start_port)
          _next_state = S_0;
        else
          _next_state = S_1;
      S_1 :
          _next_state = S_2;
      S_2 :
          _next_state = S_3;
      S_3 :
          _next_state = S_4;
      S_4 :
          wrenable_reg_0 = 1'b1;
          _next_state = S_5;
      S_5 :
          wrenable_reg_1 = 1'b1;
          _next_state = S_6;
      S_6 :
          _next_state = S_0;
          done_port = 1'b1;
        end
    endcase
  end
endmodule
```
controller_mac

Bambu Overview



Modular Framework based on the specialization of **HLS_step**

Bambu: command-line interface

- Minimal command
  - `$ bambu filename.c`
- Controlling the clock period (100Mhz)
  - `$ bambu filename.c --clock-period=10`
- Select the device
  - `$ bambu filename.c –device-name=xc7z020,-1,clg484,VVD`

Subset of synthesizable C (1)

- We support the standard GCC support
- Supported features:
  - Expressions of any kind: arithmetic, logical, bitwise, relational, conditional, comma-based expressions.
  - Types: integers, single- and double-precision floating point, _Bool and Complex, struct-or-union, bitfields, enum, typedef, pointers and arrays, type qualifiers.
  - Variable declarations, initialization, storage-specifiers
  - Functions definition and declaration, extern or static, pointer to functions, parameters passed by copy or reference, tail recursive functions.
  - Statements and blocks: labeled (case), compound, expression, selection (if,switch), iteration(while,do,for), jump (goto,continue,break,return)
  - All preprocessor directives
  - Unaligned memory accesses and dynamic pointers resolution
  - GCC vectorization

Subset not supported

- struct returned by copy
- Non-tailing recursive functions

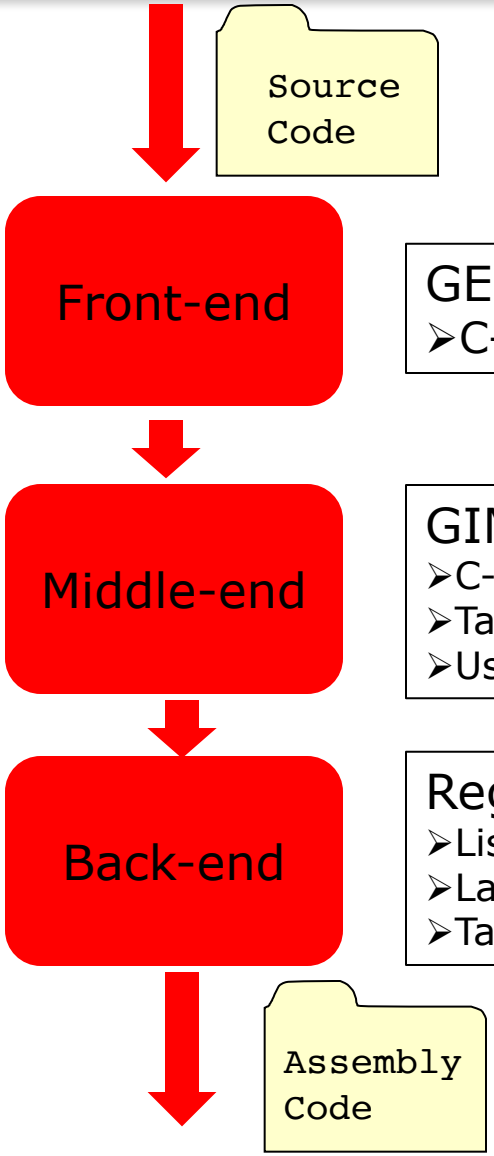
libbambu

- assert, puts, putchar, read, open, close, write, printf, exit, abort
- libc functions: bswap32,memcmp, memcpy, memmove, memset, malloc, free, memalign, alloca_with_align, calloc, bcopy, bzero, memchr, mempcpy, memrchr, rawmemchr, stpcpy, stpncpy, strcasecmp, strcasestr, strcat, strchr, strchrnul, strcmp, strcpy, strcspn, strdup, strlen, strncasecmp, strncat, strncmp, strncpy, strndup, strnlen, strpbrk, strrchr, strsep, strspn, strstr, strtok
- libm functions: acos, acosh, asin, asinh, atan, atan2, atanh, cbrt, ceil, cexpi, copysign, cos, cosh, drem, erf, exp, exp10, expm1, fabs, fdim, finite, floor, lfloor, fma, fmax, fmin, fmod, fpclassify, frexp, gamma, lgamma, tgamma, hypot, ilogb, infinity, isinf, isnan, j0, j1, jn, ldexp, log, log2, log10, log1p, modf, nan, nearbyint, nextafter, pow, pow10, remainder, remquo, rint, lrint, llrint, round, lround, llround, scalb, scalbln, scalbn, signbit, significand, sin, sincos, sinh, sqrt, tan, tanh, trunc.

Why GCC?

- Efficient, extensible well maintained open source project
- Provide a front-end for a lot of programming language: C,C++,fortran, etc
- From GCC version 3.5 it has an unified intermediate representation
- From version 4.1 it supports some parallelizing techniques: auto-vectorization
- From version 4.5 it supports custom plugins to integrate external code transformation/analysis
- Not easy to deal with anyway…

GNU GCC Internal Representation



GENERIC
- C-Like representation

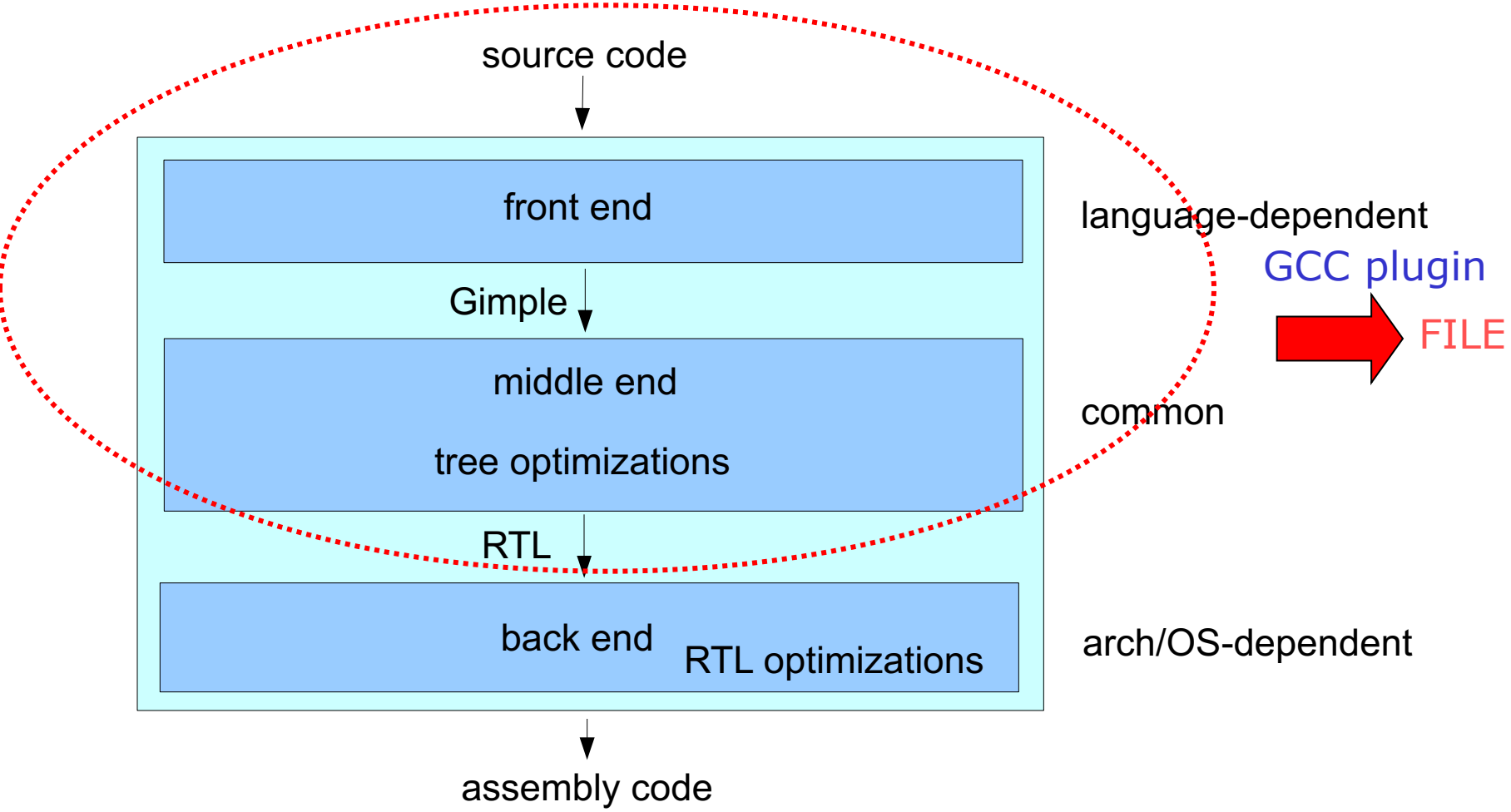
GIMPLE
- C-Like representation (Max 3 operands)
- Target independent
- Used during target-independent optimization

Register Transfer Language (RTL)
- Lisp-Like representation
- Language is target independent, BUT
- Target dependent 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 GCC



Text-based IR

- We start from the standard output GCC creates with the `-fdump-tree-all-raw`
  - We add information we needed
  - We make it unambiguous
  - We add a linker starting from this format
  - We add a new command wrapping GCC:
    - `tree-panda-gcc`

Details in these directories:
- `src/tree`
- `src/parser/treegcc`
- `src/wrapper/treegcc`

```
GCC_VERSION: "4.9.3"
PLUGIN_VERSION: "0.11"
@1 function_decl name: @2 type: @3 scpe: @4 srcp: "/panda/etc/libbambu/assert.c":6:6 uid: 1392
arg: @5 arg: @6 arg: @7 body: @8
@2 identifier_node strg: "__assert" lngt: 8
@3 function_type size: @9 algn: 8 retn: @10 prms: @11
@4 translation_unit_decl srcp: "<built-in>":0:0 uid: 1404
@5 parm_decl name: @12 type: @13 scpe: @1 srcp: "/panda/etc/libbambu/assert.c":6:27 uid: 1389
argt: @13 size: @14 algn: 32 used: 1
@6 parm_decl name: @15 type: @13 scpe: @1 srcp: "/panda/etc/libbambu/assert.c":6:52 uid: 1390
argt: @13 size: @14 algn: 32 used: 1
@7 parm_decl name: @16 type: @17 scpe: @1 srcp: "/panda/etc/libbambu/assert.c":6:64 uid: 1391
argt: @17 size: @14 algn: 32 used: 1
@8 statement_list bloc: 2 loop_id: 0 pred: ENTRY stmt: @18 stmt: @19
```

Creating a custom library

- `tree-panda-gcc` could be used to create a custom library that could be deployed by `bambu` by passing `–l` and `–L` options.
- Example:

```
$ tree-panda-gcc –O3 -o librho_a-rho.o rho.c
$ ar cru librho.a librho_a-rho.o
…
$ tree-panda-gcc –O3 -o keccak keccak-keccak_coproc.o
chi/libchi.a iota/libiota.a pi/libpi.a rho/librho.a
theta/libtheta.a
$ bambu --use-raw keccak
```

Autotools based project described in directory `examples/crypto_designs/multi-keccak`

synthesis of multiple files

- `bambu` uses the text-based IR and the builtin linker to build a single in-memory representation and perform HLS
  - `$ bambu file1.c file2.c --top-fname=fname`
- In case the option `--top-fname` is not used, `bambu` automatically identifies which function can act as the top one.

Synthesis per function: interface

- Based on C description
  - Global variables
  - Top function parameters
- No restriction on top function signature
- C semantic
  - Scalar parameters: input
  - struct/union parameters: input
  - array/pointer: input/output
  - return value: output
  - Global variables: input/output

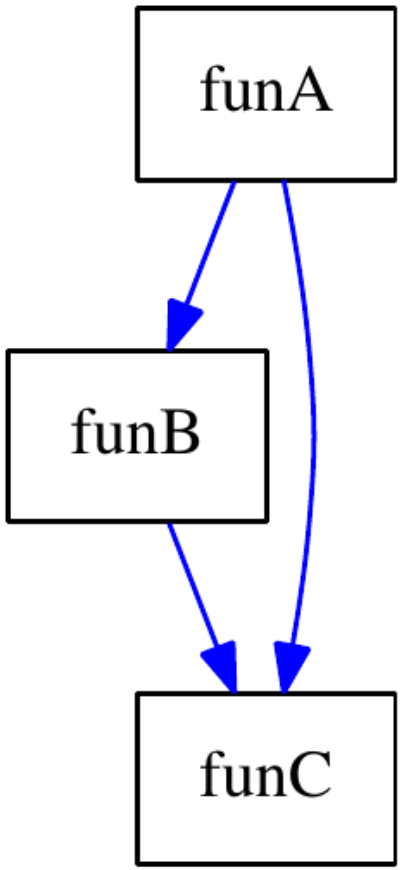
Example


```
int global;

float my_top_function(int * first, struct s second, int
third[10], float * fourth)
{…}
```

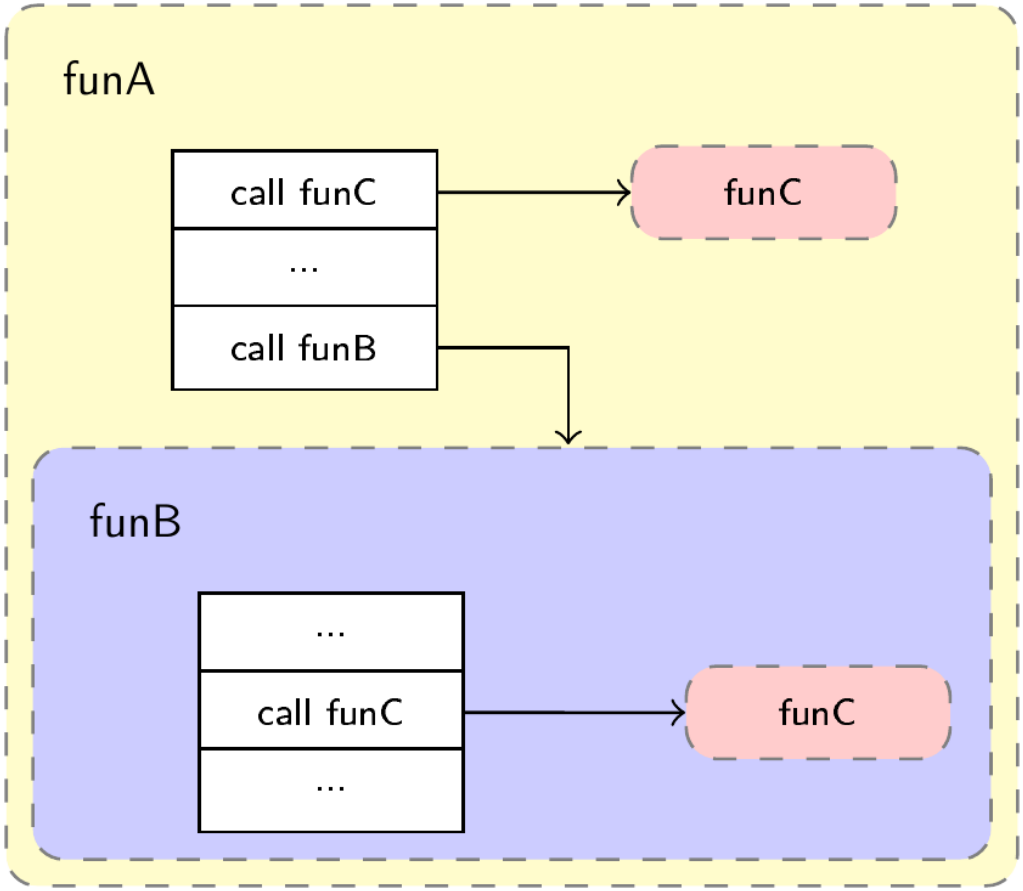
- Input parameters:
  - Second
- Input/ouptut parameters:
  - First, third, fourth
- Ouptut parameters:
  - Return value
- Shared memory:
  - global

Synthesis per function: hierarchy

- One component per function

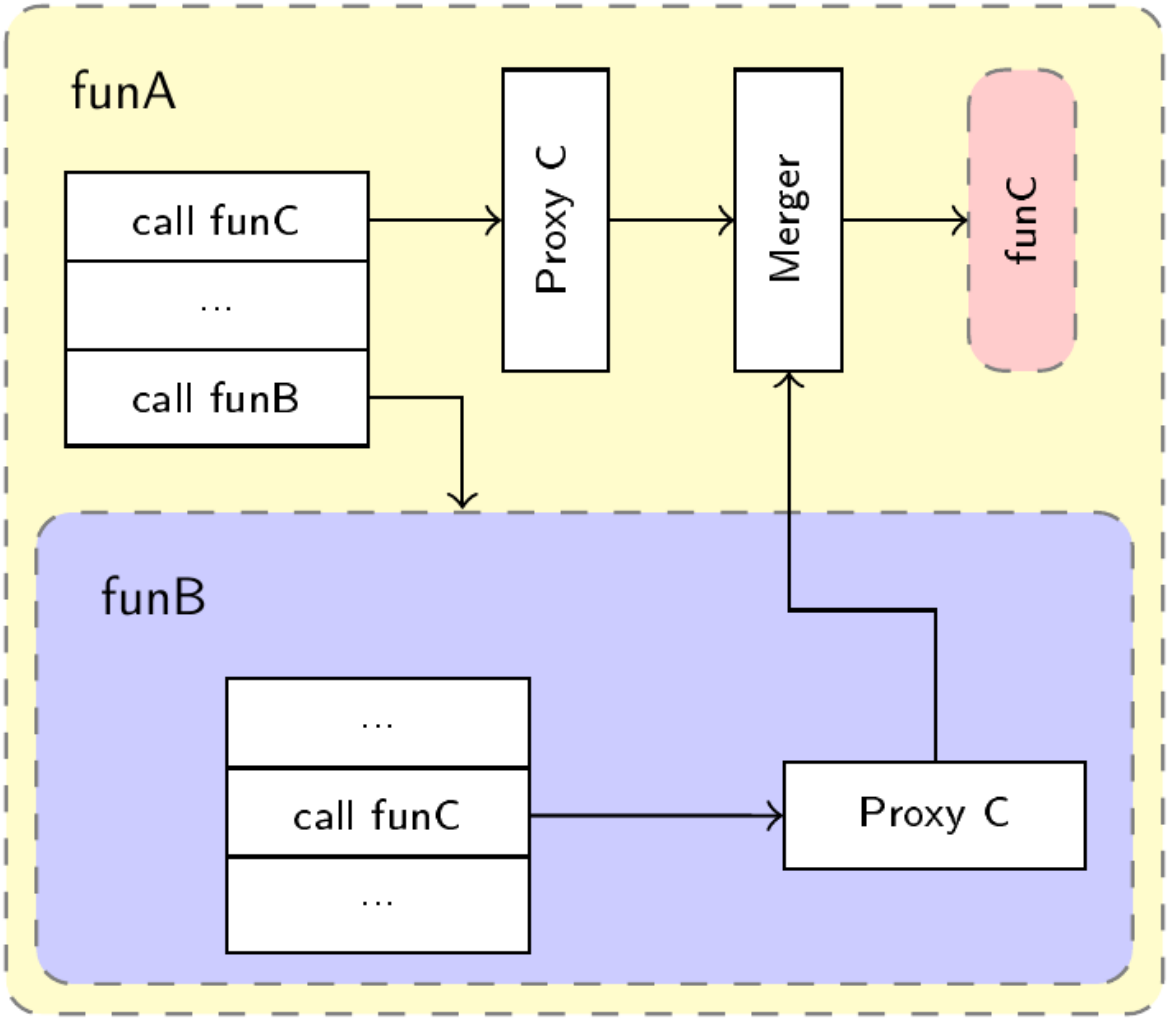


call graph



RTL hierarchy

Synthesis per function: proxy



Option: `--do-not-expose-globals`

- bambu assumes that all global variables could be accessed by external CPU/accelerators
- It is possible to change this default with option:
  - `--do-not-expose-globals`
  - All global variables are considered local to the compilation units as it they are declared `static`.

Option `--fwhole-program`

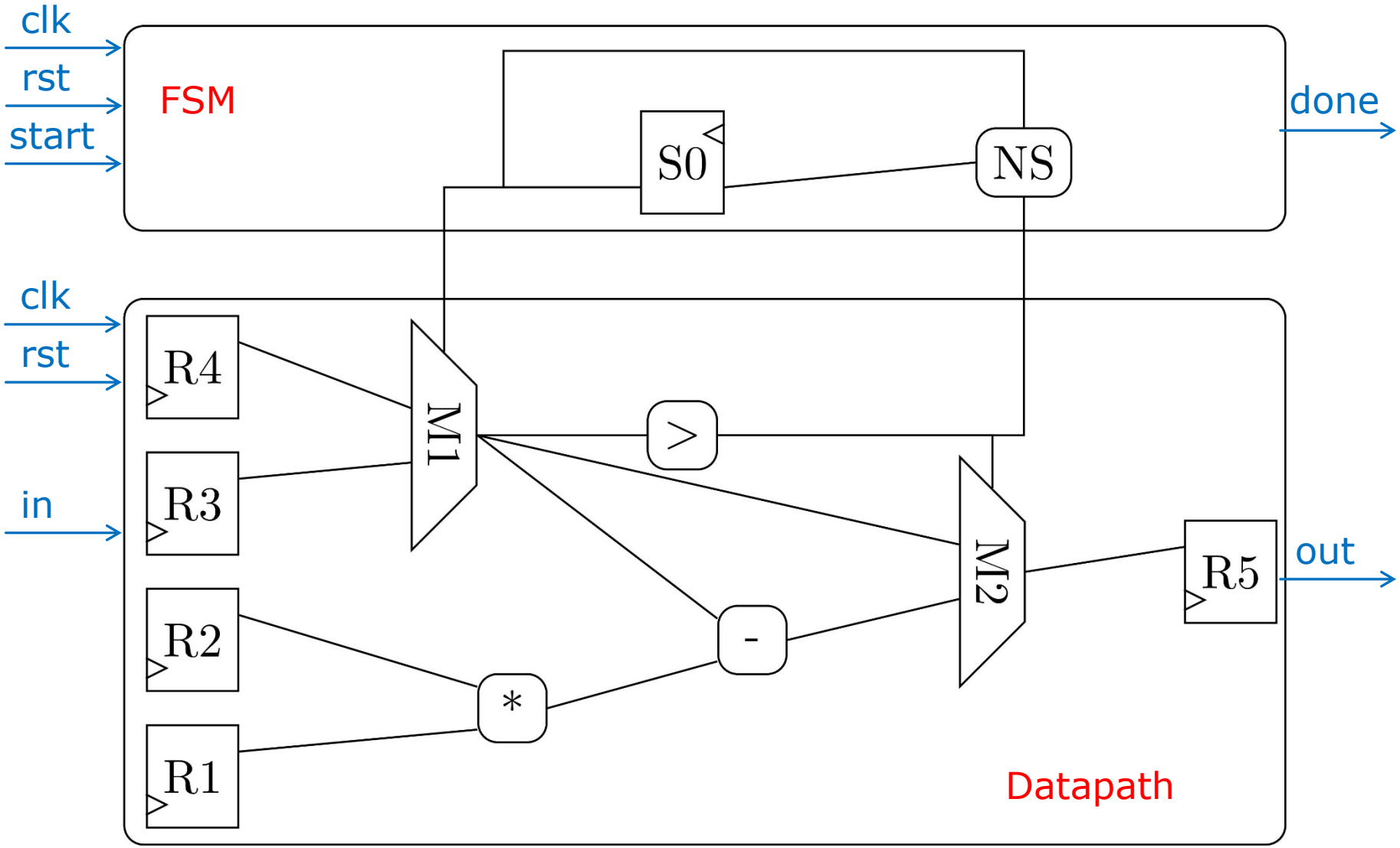
- From GCC man:
  - Assume that the current compilation unit represents the whole program being compiled. All public functions and variables with the exception of "main" and those merged by attribute "externally_visible" become static functions and in effect are optimized more aggressively by interprocedural optimizers.

- You can apply this optimization when the top function is **`main`** or when you specify an alternative top function (`--top-fname`) provided that the option `--do-not-expose-globals` is also given (currently supported only with GCC 4.9)

Synthesis per function: FSMD model

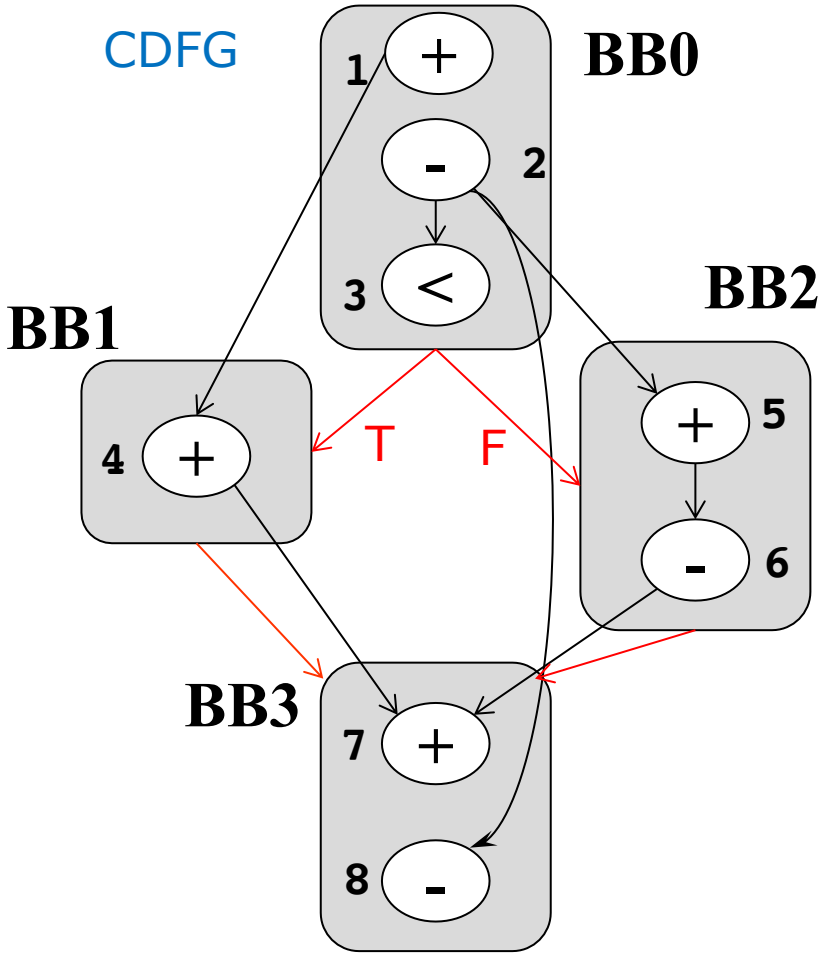
- FSM with datapath (FSMD)
  - set of storage variables *VAR*
  - set of expressions $EXP=\{\Delta(x,y,z,...) | x,y,z... \in VAR\}$
  - set of assignments $ASG=\{X=e \mid X \in VAR,\ e \in EXP\}$
  - Set of status variables $STAT=\{(a,b) |\ a,b \in EXP\}$
- An FSMD is defined by the following tuple
  - $<S, I \cup B, O \cup A, \Delta, \Lambda>$
- where S, Δ, Λ are the standard set of states, the output function and the next state function:
  - B = set of some status variables ($B \subseteq STAT$)
  - A = set of storage variable assignments ($A \subseteq ASG$)

Synthesis per function: Architectural template



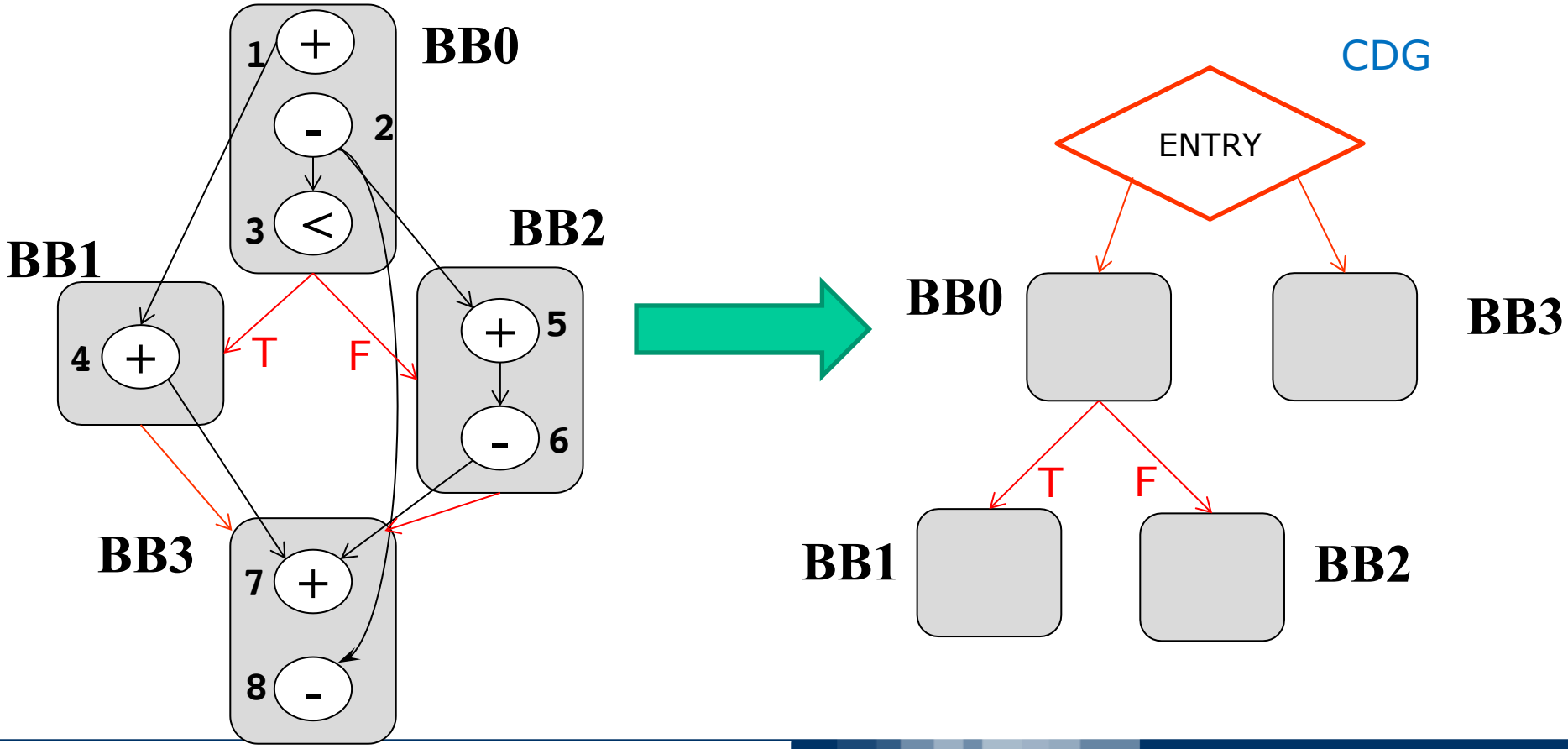
Component generation: graphs


```
1: t = a + b;          BB0
2: u = a – b;
3: if (a < b)
4:     v = t + c;      BB1
   else
   {
5:     w = u + c;      BB2
6:     v = w – d;
   }
7: x = v + e;          BB3
8: y = u – e;
```

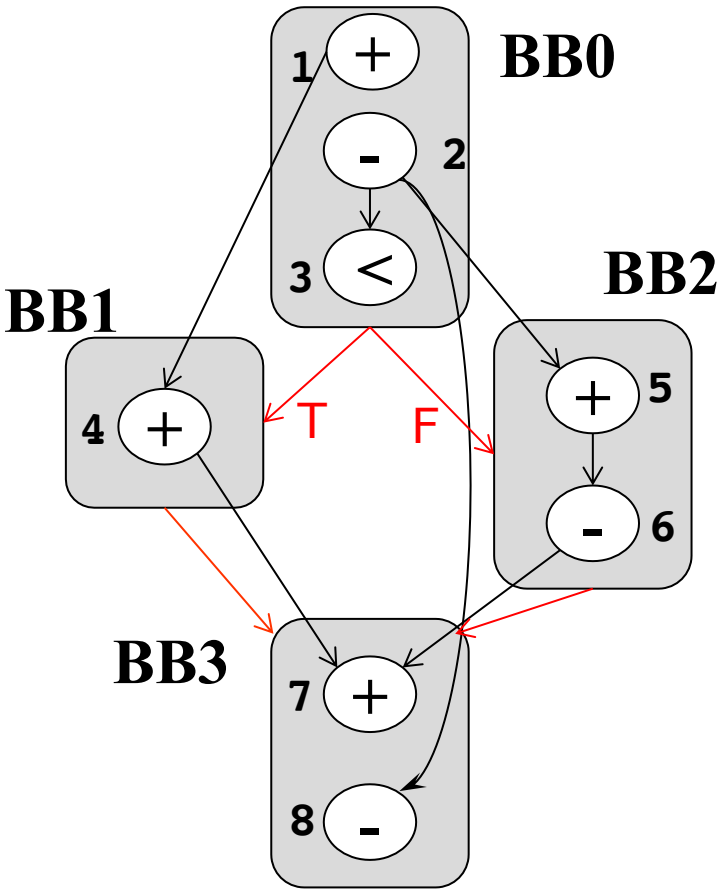


Component generation : graphs

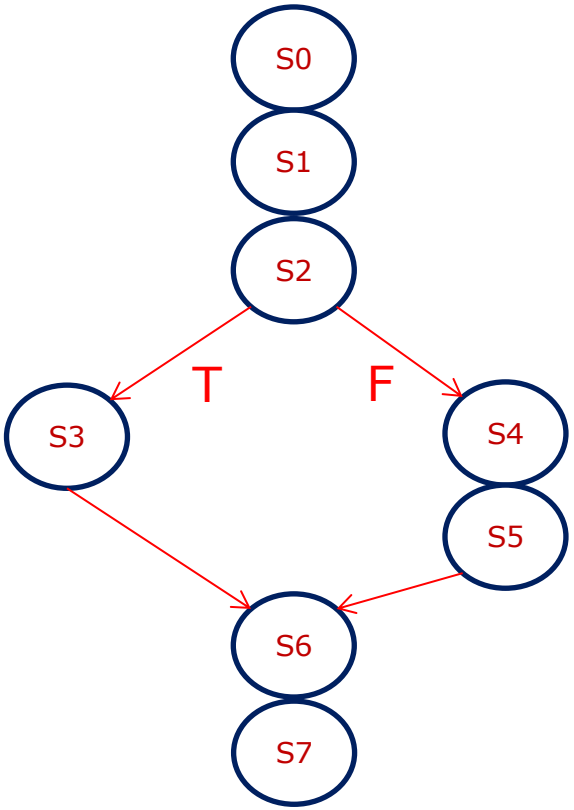
- Program Dependency Graphs – PDG
  - Data Flow Graph – DFG
  - Control Dependence Graph - CDG



Component generation : FSM

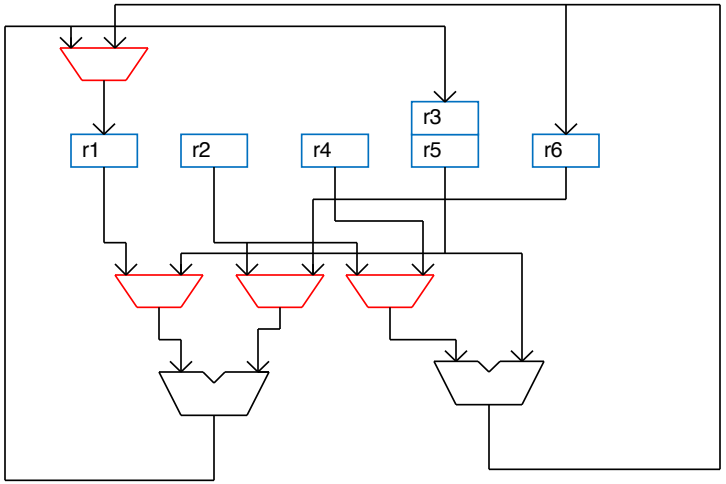


scheduling



Component generation : datapath

Module binding
Register binding
Interconnection definition



Operation modeling

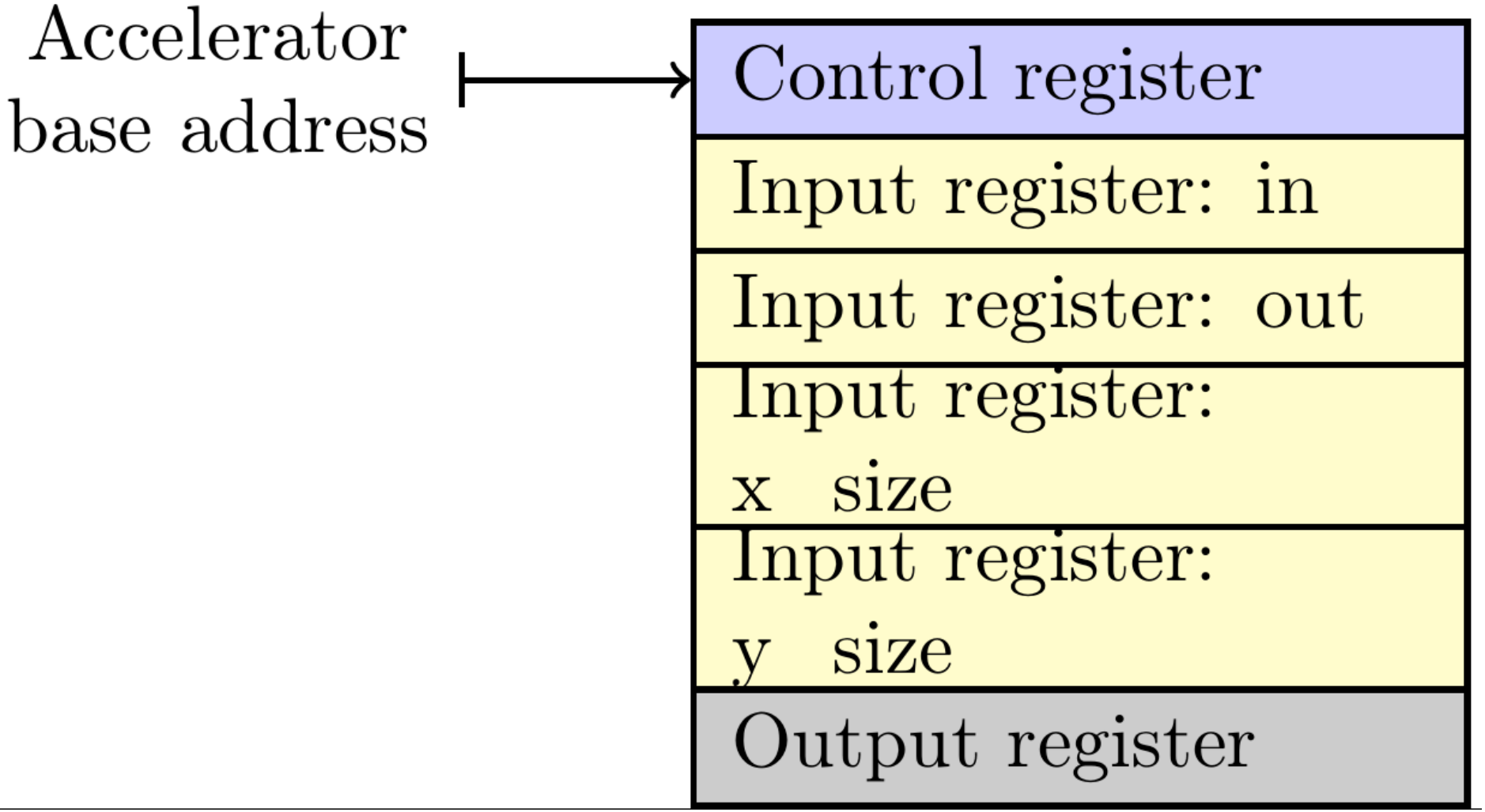
- Single cycle
- Multi-cycle
- Pipelined operator
- Multi-function
- Variable latency

- XML files to describe operations and resources

Synthesis of function pointers


```
int laplacian(char *, char *, int, int);
int make_inverse_image(char *, char *, int, int);
int sharpen(char *, char *, int, int);
int sobel(char *, char *, int, int);

int (*pipeline[MAX_DEPTH])(char *, char *, int, int);

void UserApp(char *in, char *out, int x_size, int y_size) {
  // ...
  // Pipeline configuration using function pointers
  add_filter(0, make_inverse_image);
  add_filter(1, sharpen);
  // ...
  // execute is synthesized in hardware
  execute(in, out, x_size, y_size);
}
void execute(char *in, char *out, int x_size, int y_size) {
  int i = 0;
  for (i = 0; i < MAX_PIPELINE_DEPTH; i++) {
    if (pipeline[i] == 0) break;
    // here other hw accelerator are called
    // using function pointers
    int res = pipeline[i](in, out, x_size, y_size);
    if (res != 0) return;
    swap(in, out);
  }
  move_if_odd(i, in, out);
}
```

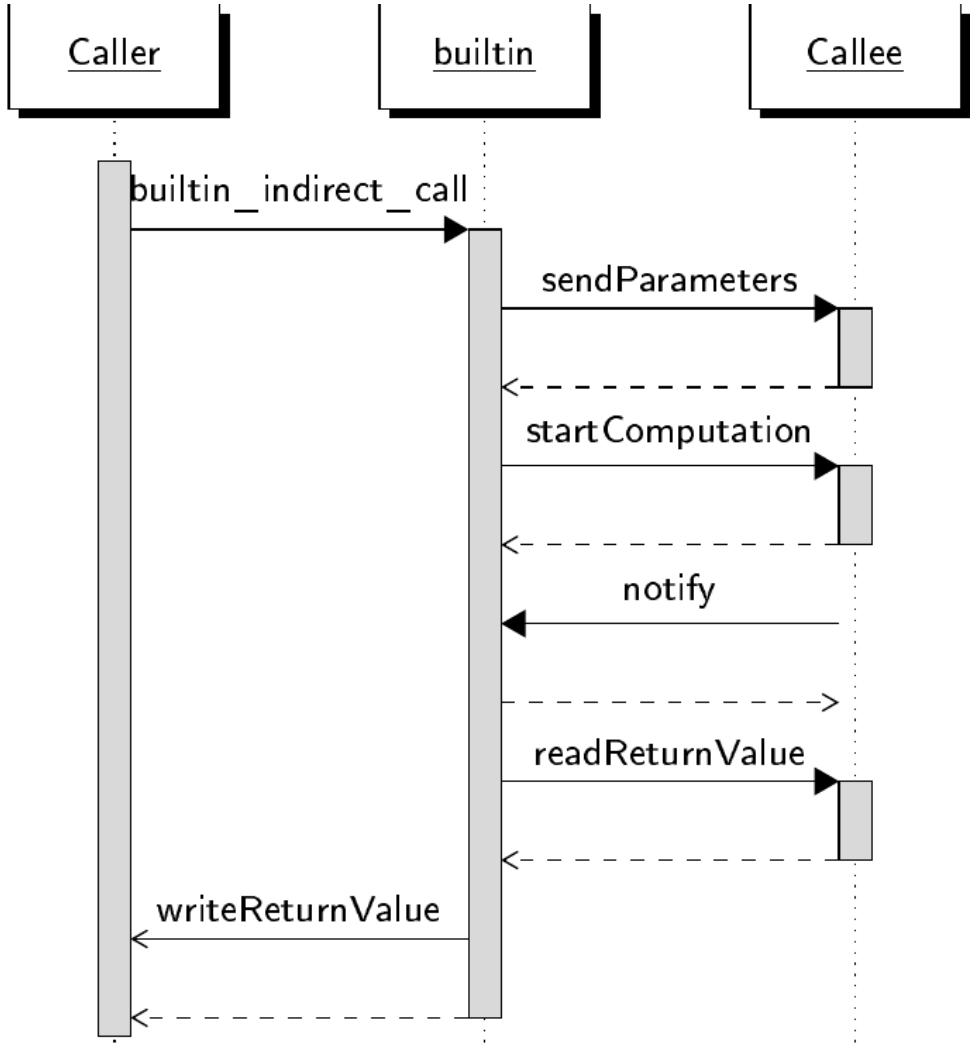
Adding a memory mapped interface to filters



Code transformation performed


```
void execute(char *in, char *out, int x_size, int y_size) {
  int i = 0;
  for (i = 0; i < MAX_PIPELINE_DEPTH; i++) {
    if (pipeline[i] == 0) break;
    // here other hw accelerator are called
    // using function pointers
    __builtin_indirect_call(
        pipeline[i], 1, in, out, x_size, y_size, &res);
    if (res != 0) return;
    swap(in, out);
  }
  move_if_odd(i, in, out);
}
```

Sequence diagram for function indirect call



Call mechanism complexity: #cycles=Wl(Np+1)+lhs(Wl+Rl)

Integration of hand-written components in the HLS flow


```
#include "module_lib.h"

void my_ip(uint8_t command, uint32_t param1, uint32_t param2) {
    static module1_output_t module1_output;
    static module2_output_t module2_output;
    switch(command) {
        case 0:
            module1(param1, param2 >> 16, &module1_output);
            break;
        case 1:
            module2(param1, &module2_output);
            break;
        case 2:
            printer1(module1_output.output1, module1_output.output2, module1_output.output3,
module1_output.output4);
            break;
        case 3:
            printer2(module2_output.output1, module2_output.output2, module2_output.output3);
            break;
        default:
            break;
    }
}
```

Integration of hand- written components in the HLS flow

- Function mapped on IPs has to be declared as extern:
  - `extern void module1(uint32_t input1, uint16_t input2, module1_output_t *outputs);`
- C code has to be passed with the following option
  - `--C-no-parse=module1.c,…`
- Binding between function **module1** and component **module1** has to be specified with a XML file and passed as an option to bambu
  - `$ bambu … module_lib.xml`
- Check these examples:
  - `examples/IP_integration`
  - `examples/breakout`
  - `examples/pong`
  - `examples/led_example`

C++ support

- Support of c++ is ongoing:
  - templates
  - C++11 and beyond
  - ac_types from Mentor Graphics could be used
    - ac_int and ac_fixed currently cannot be used in the function interfaces
    - not c++11 friendly

```
#include <algorithm>
int gcd(int x, int y )
 {
        if( x < y )
           std::swap( x, y );

        while( y > 0 )
        {
           int f = x % y;
           x = y;
           y = f;
        }
        return x;
 }
```

Fortran support (1)


```
*     euclid.f (FORTRAN 77)
*     Find greatest common divisor using the Euclidean algorithm

      FUNCTION NGCD(NA, NB)
        IA = NA
        IB = NB
    1   IF (IB.NE.0) THEN
          ITEMP = IA
          IA = IB
          IB = MOD(ITEMP, IB)
          GOTO 1
        END IF
        NGCD = IA
        RETURN
      END
```

By default parameters are passed by reference

Fortran support (2)


```
*     euclid.f (FORTRAN 77)
*     Find greatest common divisor using the Euclidean algorithm

      FUNCTION NGCD(NA, NB)
        integer, value :: NA ! input
        integer, value :: NB ! input
        IA = NA
        IB = NB
    1   IF (IB.NE.0) THEN
          ITEMP = IA
          IA = IB
          IB = MOD(ITEMP, IB)
          GOTO 1
        END IF
        NGCD = IA
        RETURN
      END
```

Here parameters are passed by copy

- Fortran support in progress…

OpenMP support

- #pragma omp simd is supported

```
#pragma omp declare simd
void add (int accelnum, int startidx, int endidx)
{
   int sum = 0;
   int i;
   for (i = 0; i < OPS_PER_ACCEL; i++) {
      sum += array[i+startidx];
   }
   output[accelnum] = sum;
}

main() {
...
#pragma omp simd
   for (i = 0; i < NUM_ACCELS; i++)
   {
      add(i, i * OPS_PER_ACCEL, (i + 1)*OPS_PER_ACCEL);
   }
```

- OmpMP support in progress…