International Conference
on Supercomputing 2021
June 14 - 18, 2021. Worldwide online event

POLITECNICO DI MILANO

ICS21 Tutorial

Bambu: High-level synthesis for parallel programming

Fabrizio Ferrandi, Serena Curzel, Michele Fiorito (Politecnico di Milano),
Vito Giovanni Castellana, Marco Minutoli, Antonino Tumeo (PNNL)

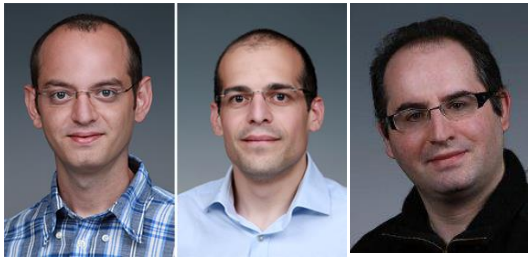
Outline

- 9:00 Presentation of Bambu
- 10:00 break
- 10:30 Compiler Based Optimizations, Tuning and Customization of Generated Accelerators
- 11:30 break
- 12:30 Target Customization and Tool Integration
- 13:30 break
- 14:00 Exploiting Vectorization in High Level Synthesis of Nested Irregular Loops
- 15:00 break
- 15:30 Enabling the High-Level Synthesis of Data Analytics Accelerators
- 16:30 Future research on Bambu

Material prepared by

- Politecnico di Milano
  - Fabrizio Ferrandi
  - Marco Lattuada
  - Christian Pilato
  - Pietro Fezzardi
  - Serena Curzel
  - Michele Fiorito

- PNNL
  - Vito Giovanni Castellana
  - Marco Minutoli
  - Antonino Tumeo

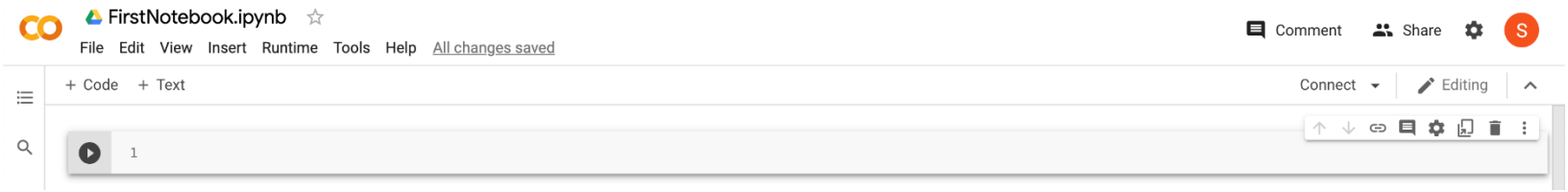


Tutorial Material

- Slides and material can be downloaded from:

ICS 2021 tutorial | panda.dei.polimi.it

- Project Jupyter is a project and community whose goal is to "develop open-source software, open-standards, and services for interactive computing across dozens of programming languages".



PandA-bambu/documentation/tutorial_ics_2021 at tutorial_2021 · ferrandi/PandA-bambu · GitHub

Bambu binary distribution

- The code is available as AppImage on https://panda.dei.polimi.it
  - bambu-x86_64.AppImage
  - AppImage is a format for distributing portable software on Linux without needing superuser permissions to install the application.
  - Once downloaded just add the execution rights with this command:

    ```
    chmod +x bambu-x86_64.AppImage
    ```
  - Source code is available on GitHub:
    - https://github.com/ferrandi/PandA-bambu/

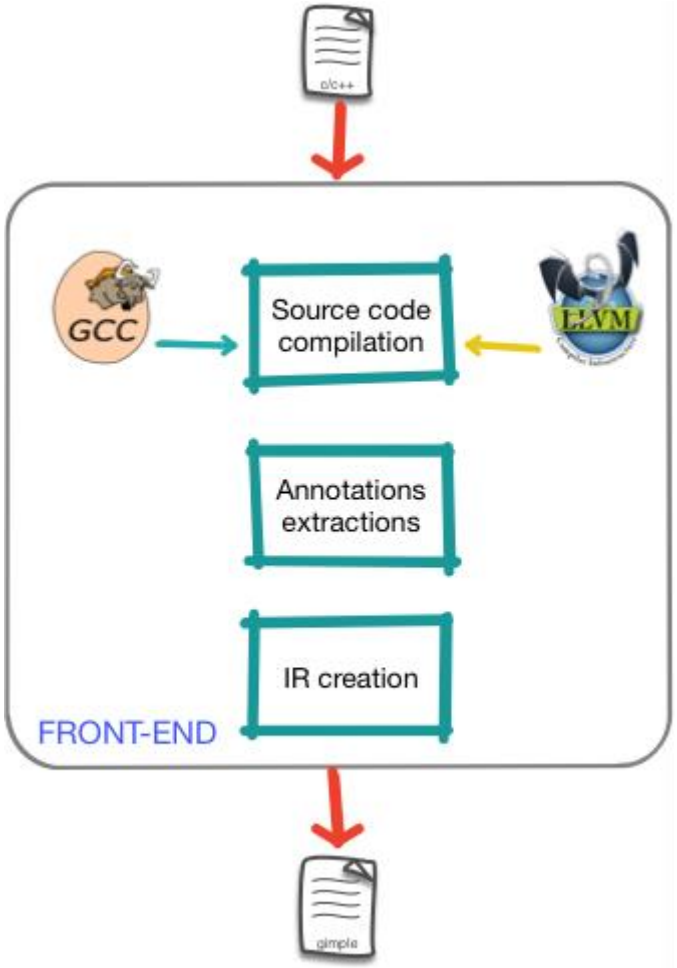
A bit of history about PandA

- 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.
- Current research funded by two EU H2020 projects: EVEREST and HERMES.

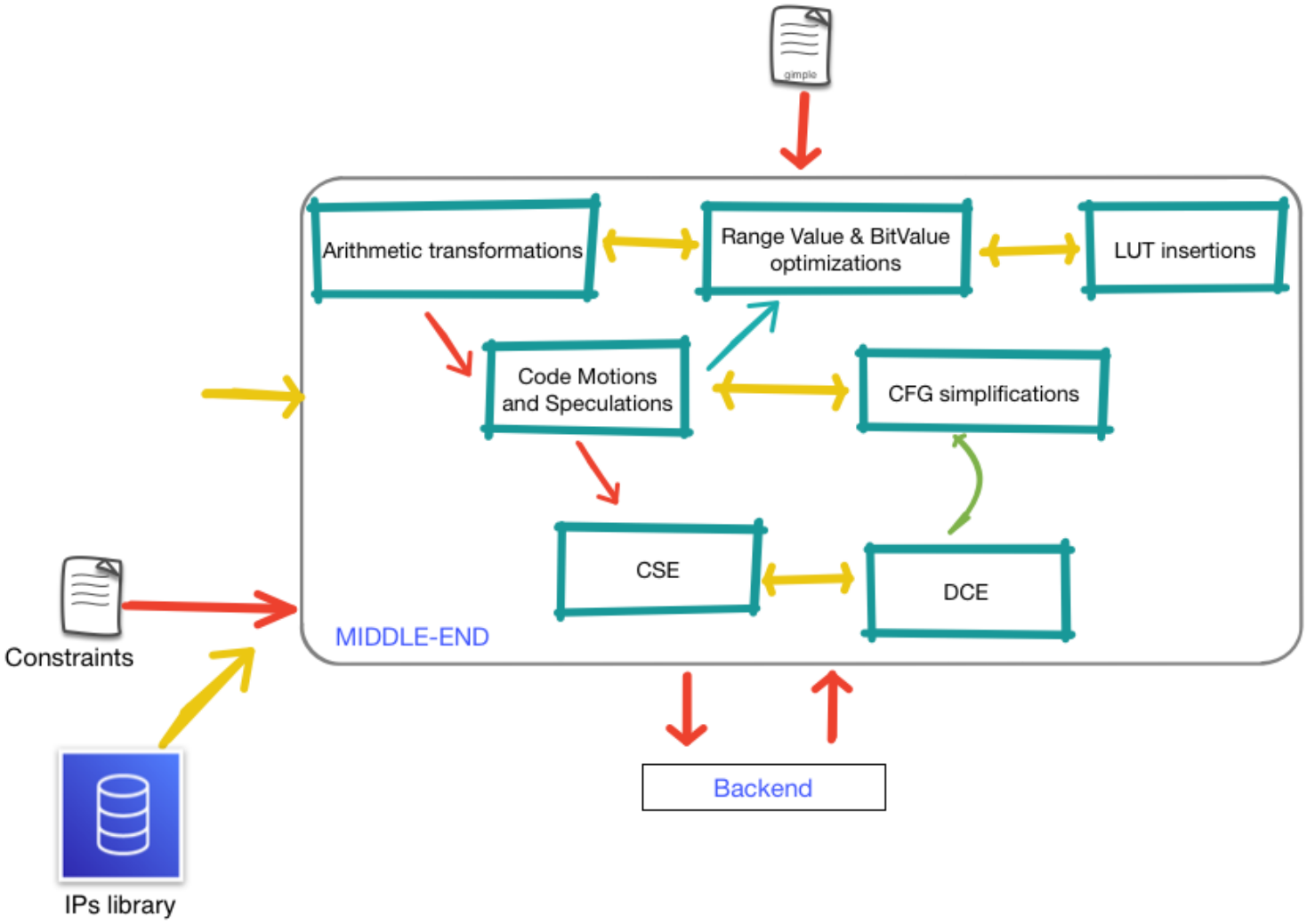
Bambu: an example of modern HLS tools

- 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/CLANG-LLVM 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: OpenRoand

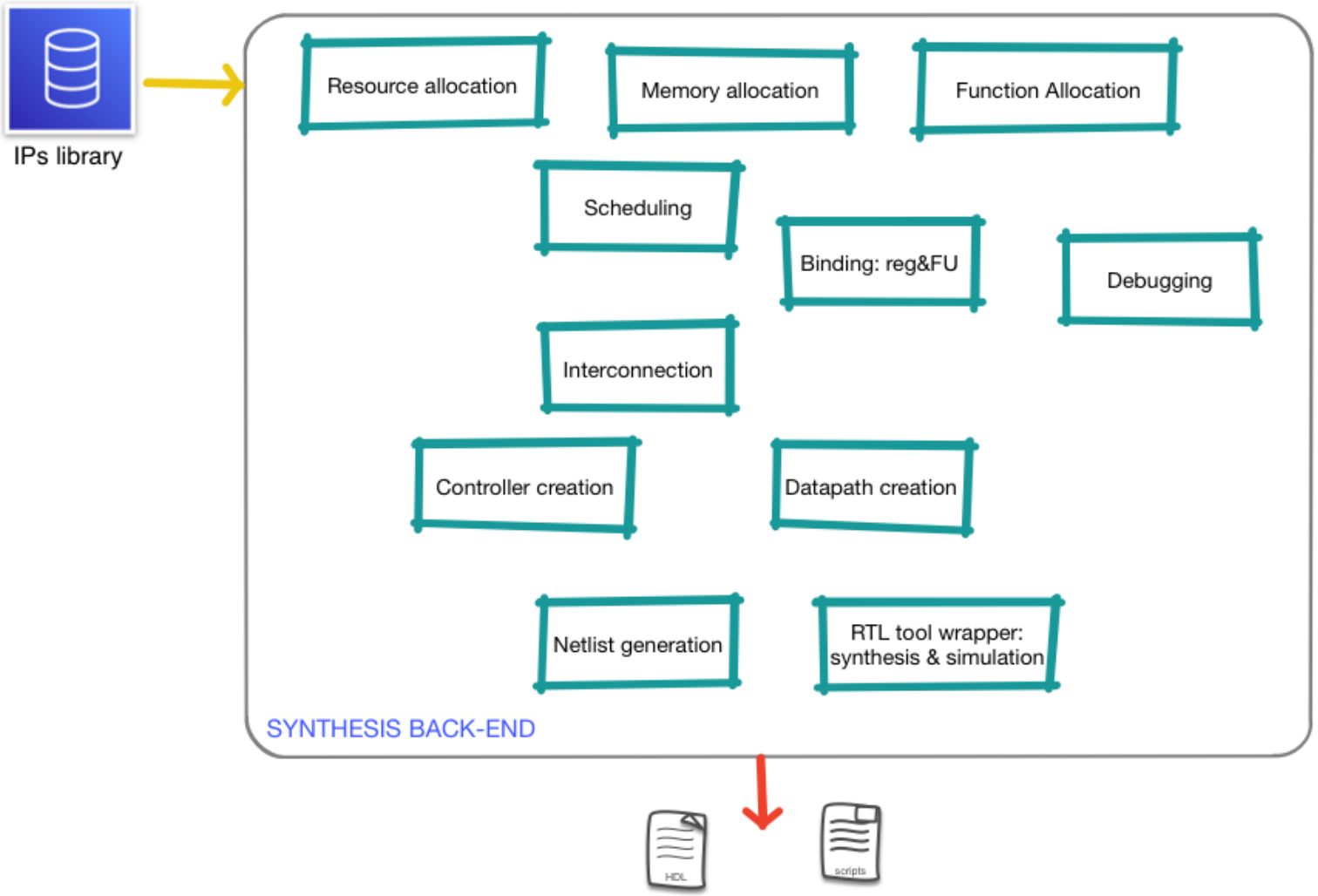
Bambu: front-end



Bambu: middle-end



Bambu: back-end



Bambu: command-line interface

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

First synthesis

- Go To Colab

Subset of synthesizable C (1)

- We support what standard compilers accept as input (CLANG/LLVM and GCC)
- 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.

Second example

- Search and insertion in a binary tree
  - Two data structures: stack and binary tree
  - Static memory allocators
  - Tail recursive functions
  - Use of pointer to pointers (some HLSs have problems)
- Goto Colab

Third example

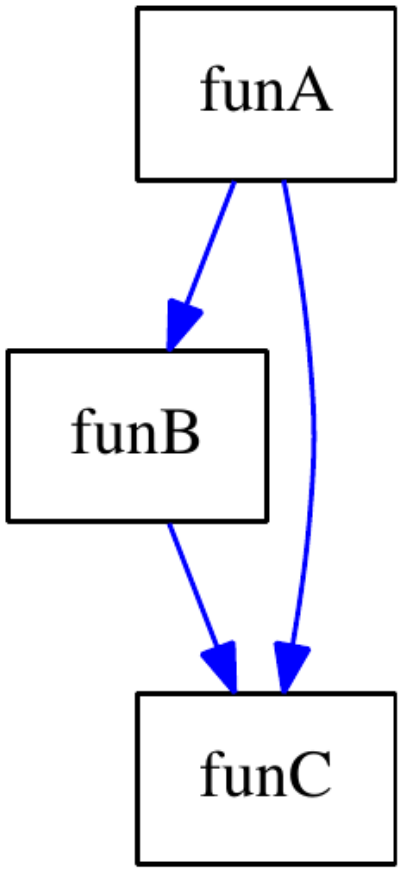
- Crypto core
  - 1) compiled with a standard clang-11
  - 2) bambu synthesis starting from the .ll file

Synthesis per function

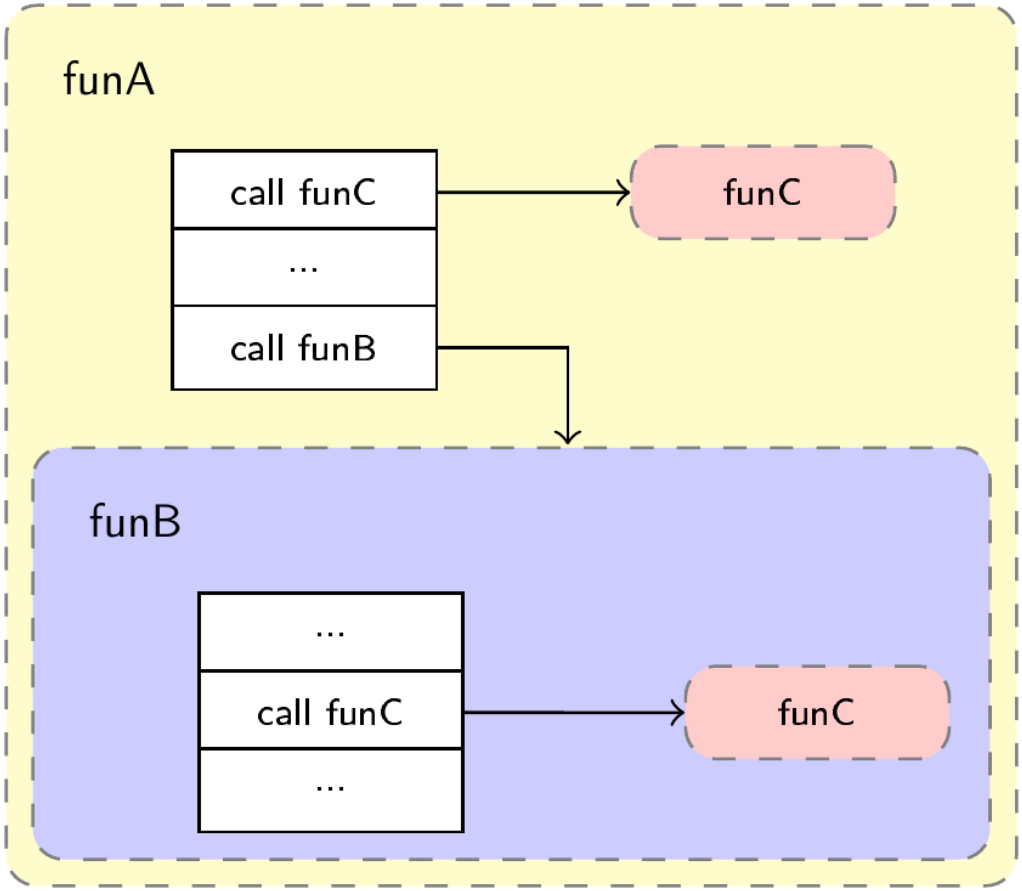
- one component per function
  - function interface
  - start and done
  - parameter passing
    - wires
    - memory interaction
    - none (ap_none), acknowledge (ap_ack), valid (ap_vld), ovalid (ap_ovld), handshake (ap_hs), fifo (ap_fifo) and array (ap_memory)
- hierarchy based on call graph
  - no-recursion
  - proxy

Synthesis per function: hierarchy

- One component per function

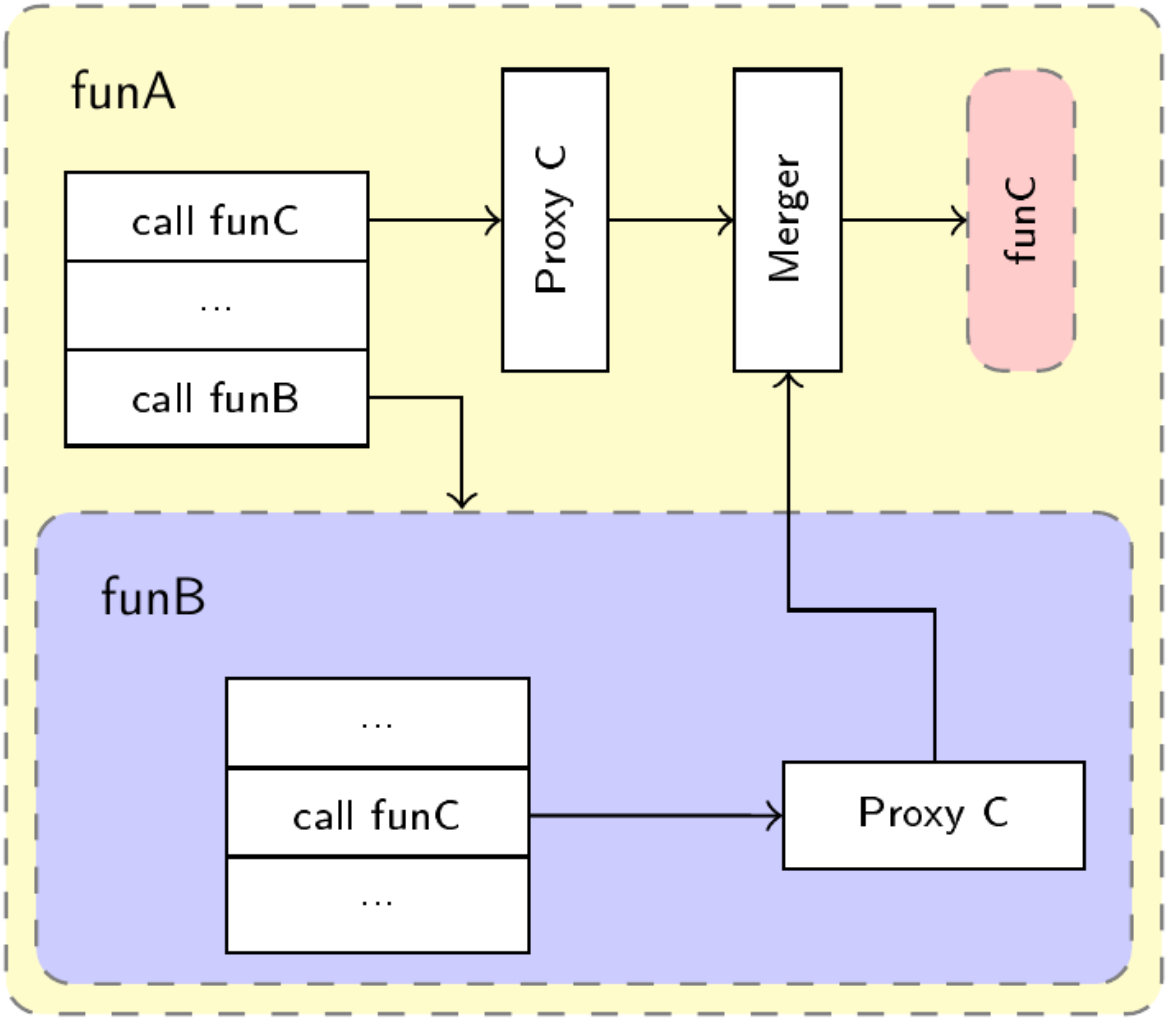


call graph



RTL hierarchy

Synthesis per function: proxy



Fourth example

- Lu-decomposition with single precision floating point arithmetic
- Goto Colab

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`

Fifth example

- Integration of existing IPs written in Verilog that receives structs passed by pointers
- Goto Colab

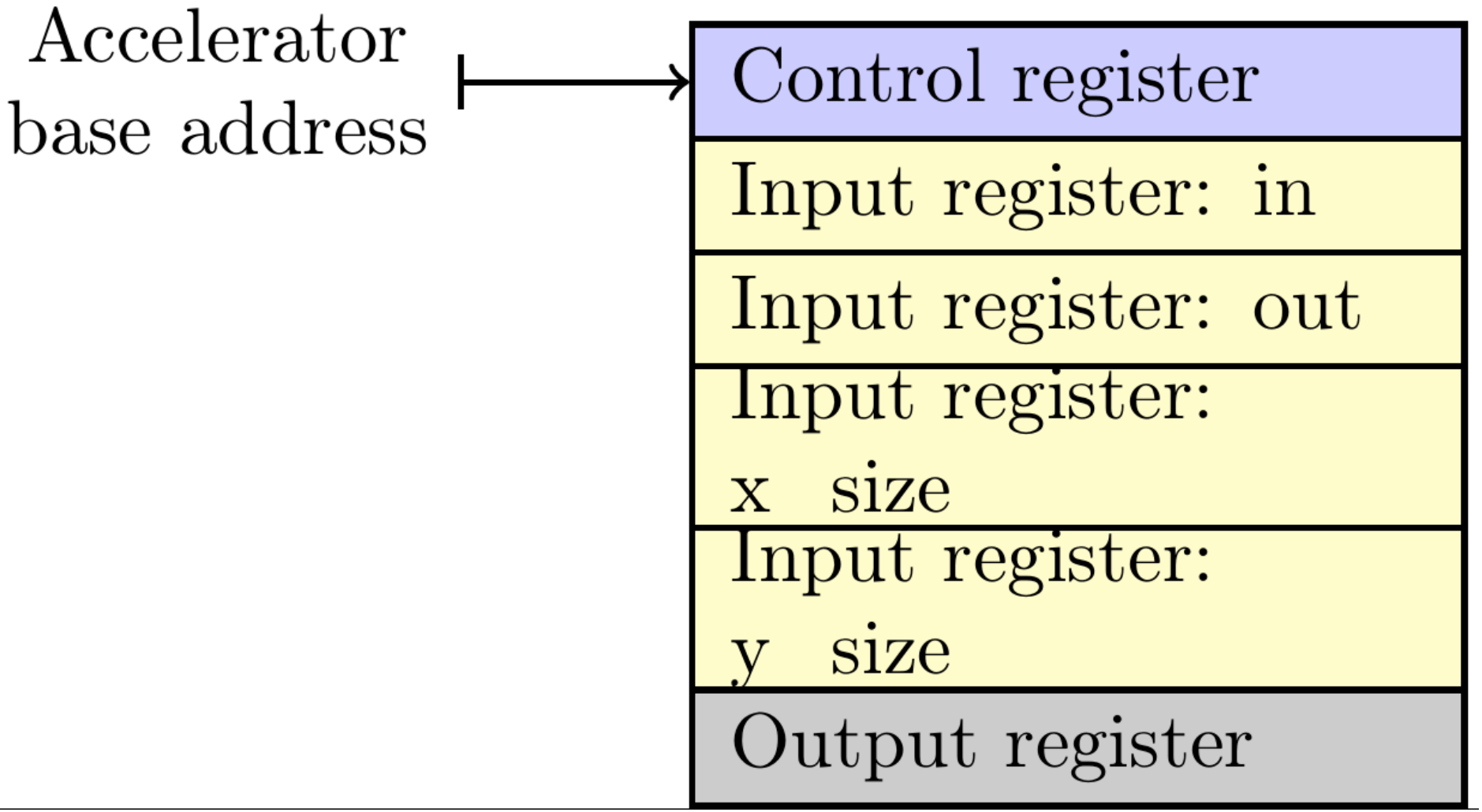
Synthesis of function pointers


```
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);
}
```

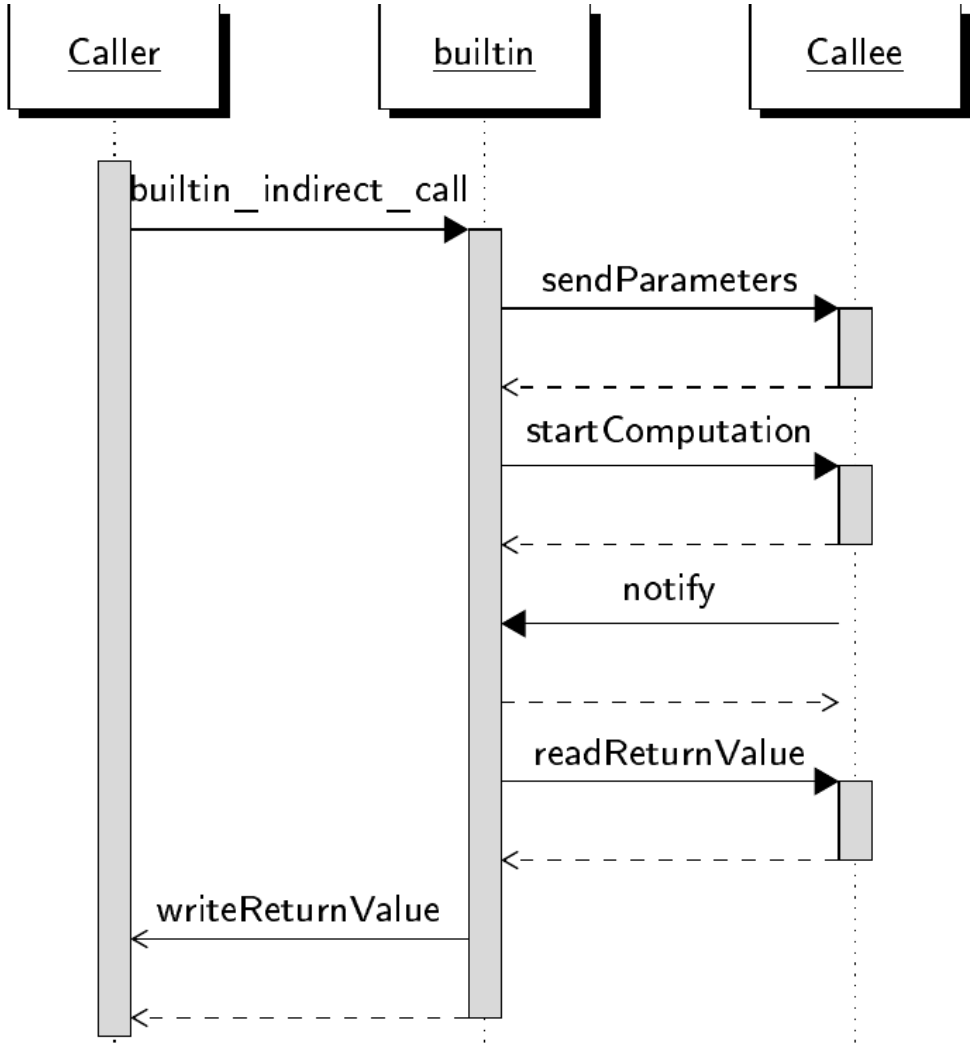
Adding a memory mapped interface to filters



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)

Sixth example

- Parametric quick sort
- Quick sort parametric with respect to the comparison function
- Goto Colab

C++ support

- Support of C++ is ongoing:
  - templates
  - C++11 and beyond
  - ac_types from Mentor Graphics could be used
  - ap_types from Xilinx support by wrapping ac_types

```
#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;
 }
```

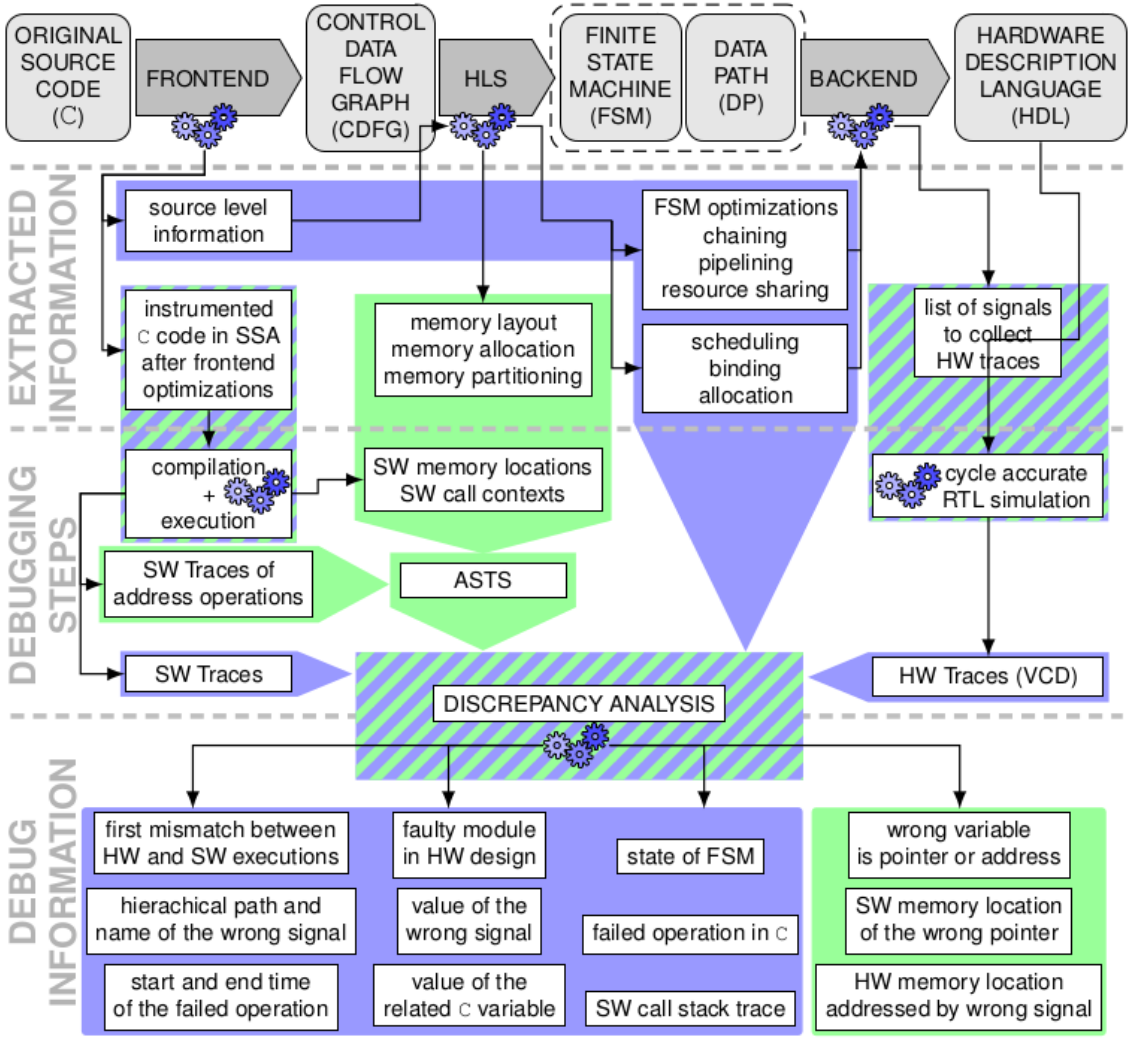
Fortran support (1)


```
*     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

Discrepancy Analysis Debug Flow



http://panda.dei.polimi.it