DATE 26
DESIGN, AUTOMATION AND TEST IN EUROPE
THE EUROPEAN EVENT FOR ELECTRONIC SYSTEM DESIGN & TEST
20 – 22 APRIL 2026
VERONA, ITALY
PALAZZO DELLA GRAN GUARDIA

Synthesis and Optimization of Custom Accelerators with Bambu

Serena Curzel

serena.curzel@polimi.it

POLITECNICO MILANO 1863

DEPARTMENT OF ELECTRONICS INFORMATION AND BIOENGINEERING

DandA

Documentation/training material

Branch: dev/panda

PandA Bambu HLS Tutorials License GPL v3

A group of little tutorials to introduce each aspect of the PandA Bambu High-Level Synthesis tool individually. Learn only what you need and take the best from the tool.

High-Level Synthesis 101

- Introduction to High-Level Synthesis Open in Colab
- Kernel Offloading Quick Start Open in Colab

High-Level Synthesis 102

- Integrate external IPs with HLS generated designs Open in Colab
- Exploit TrueFloat custom floating-point cores Open in Colab

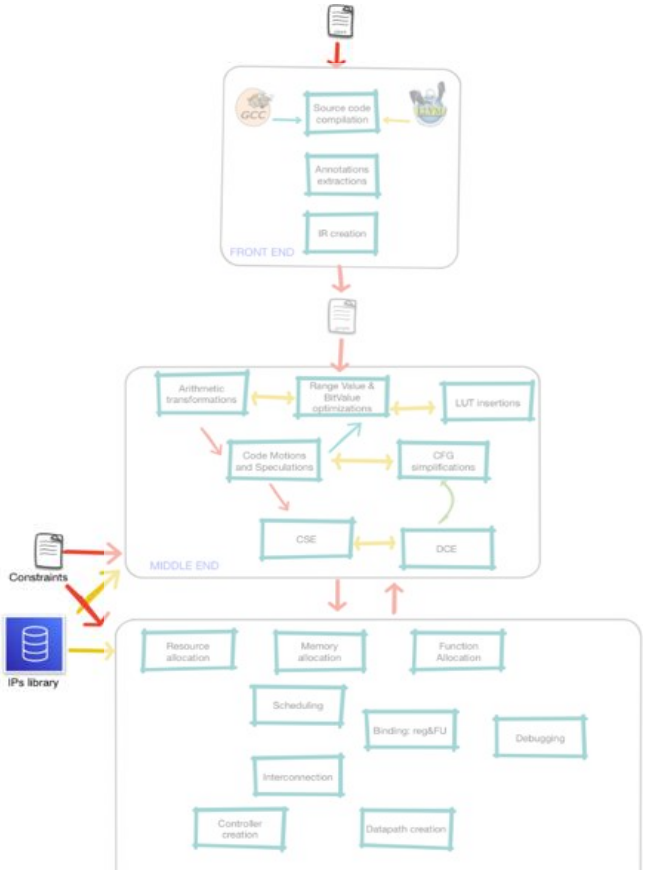
Conference Tutorials

- Open in Colab

Bambu: a modern HLS tool

Inputs:

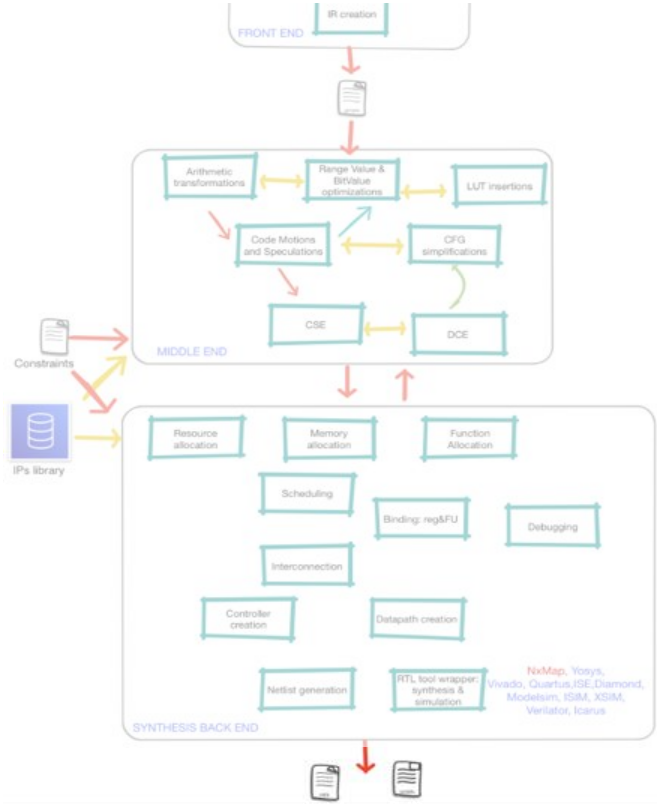
- C/C++/LLVM IR
  - Frontend compilation through GCC or Clang
  - Support for STL-like C++ structures
  - Support for fixed-point HLS types
- Command-line optimization directives and constraints, pragmas
- Library of functional units characterized for each target
  - Performance estimation essential for scheduling and optimization



Bambu: a modern HLS tool

Outputs:

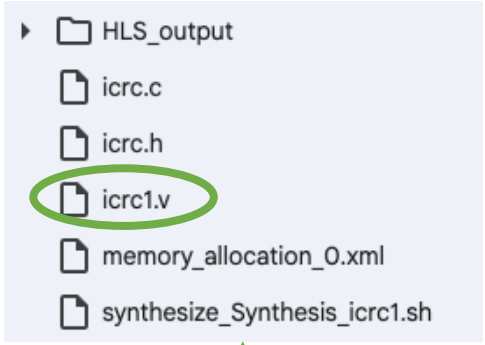
- Synthesizable Verilog/VHDL
  - Target-specific
  - FPGA targets from AMD/Xilinx, Intel/Altera, Lattice, NanoXplore
  - ASIC targets thorugh OpenROAD (Nangate45, ASAP7)
- Automatically generated testbench
  - RTL simulation with Verilator/Modelsim/XSIM
- Scripts for logic synthesis/implementation
  - Vivado/Quartus/Diamond…



Inputs and outputs

icrc.c

```
unsigned short icrc1(unsigned short crc,
unsigned char onech)
{
    int i;
    unsigned short ans=(crc^onech << 8);

    for (i=0;i<8;i++) {
        if (ans & 0x8000)
            ans = (ans <<= 1) ^ 4129;
        else
            ans <<= 1;
    }
    return ans;
}
```




```
bambu icrc.c --top-fname=icrc1
```

Inputs and outputs

icrc.c

```
unsigned short icrc1(unsigned short crc,
unsigned char onech)
{
    int i;
    unsigned short ans=(crc^onech << 8);

    for (i=0;i<8;i++) {
        if (ans & 0x8000)
            ans = (ans <<= 1) ^ 4129;
        else
            ans <<= 1;
    }
    return ans;
}
```

icrc1.v

```
module icrc1(clock,
             reset,
             start_port,
             crc,
             onech,
             done_port,
             return_port);
// IN
  input clock;
  input reset;
  input start_port;
  input [15:0] crc;
  input [7:0] onech;
  …
```

No interface generation option requested – simplest control + input/output wires

Commonly used options

```
bambu icrc.c --top-fname=icrc1
```

Mandatory

```
--clock-period=5
```

ns, default: 10

```
-v4
```

from 0 to 6, very useful

```
--device-name=xcu280-2Lfsvh2892-VVD
```

default: Zynq7000

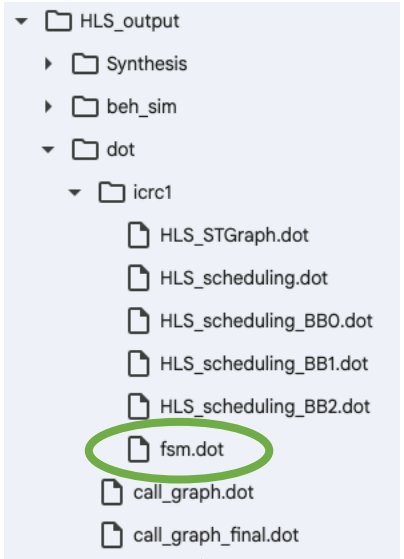
VENDOR	MODEL	BAMBU ID
Intel	Cyclone V SE	5CSEMA5F31C6
Intel	Stratix V	5SGXEA7N2F45C1
Intel	Cyclone II	EP2C70F896C6 EP2C70F896C6-R (with retiming)
Intel	Stratix IV GX	EP4SGX530KH40C2
Lattice	ECP3 LFE3-35	LFE335EA8FN484C
Lattice	ECP5 LFE5U-85	LFE5U85F8BG756C
Lattice	ECP5 LFE5U-M85	LFE5UM85F8BG756C
NanoXplore	NG-Large	nx1h140tsp

… and more:
https://panda.deib.polimi.it/?page_id=1061

Graphical FSM representation

icrc.c

```
unsigned short icrc1(unsigned short crc,
unsigned char onech)
{
    int i;
    unsigned short ans=(crc^onech << 8);

    for (i=0;i<8;i++) {
        if (ans & 0x8000)
            ans = (ans <<= 1) ^ 4129;
        else
            ans <<= 1;
    }
    return ans;
}
```

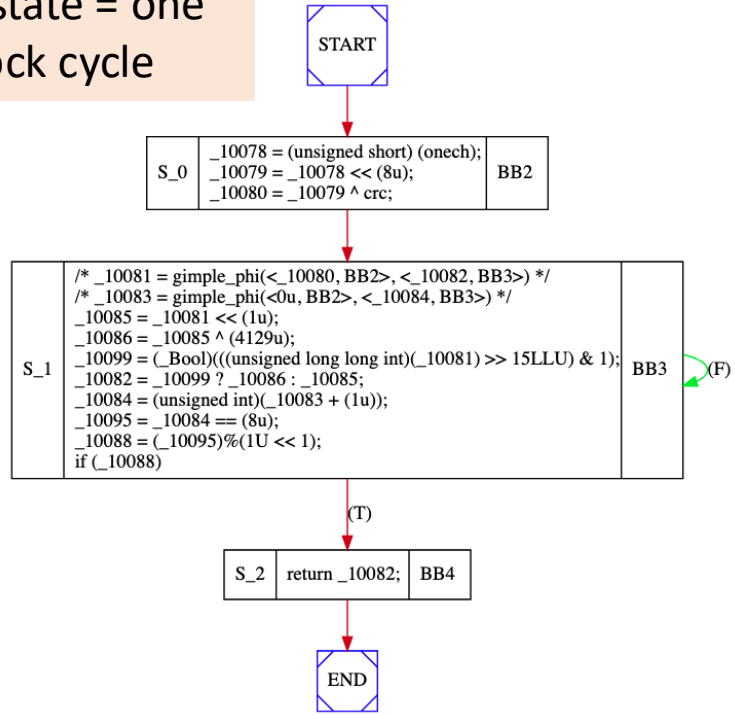



```
bambu icrc.c --top-fname=icrc1
--print-dot
```

Graphical FSM representation

```
unsigned short icrc1(unsigned short crc,
unsigned char onech)
{
    int i;
    unsigned short ans=(crc^onech << 8);

    for (i=0;i<8;i++) {
        if (ans & 0x8000)
            ans = (ans <<= 1) ^ 4129;
        else
            ans <<= 1;
    }
    return ans;
}
```

One state = one clock cycle

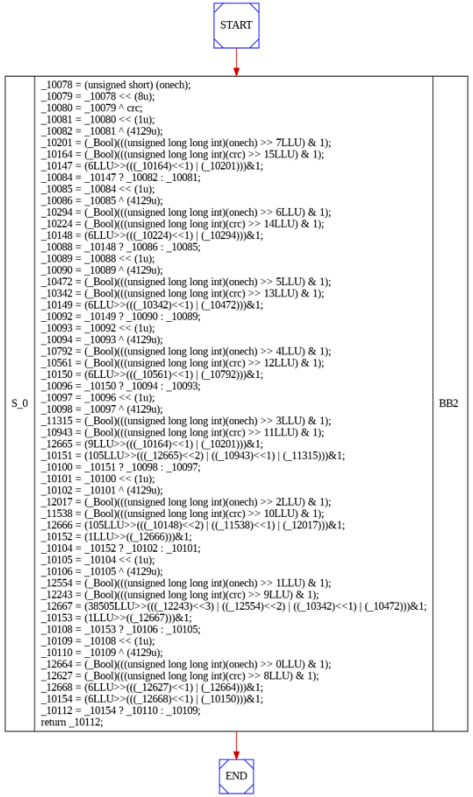


No loop unrolling

Graphical FSM representation

```
unsigned short icrc1(unsigned short crc,
unsigned char onech)
{
    int i;
    unsigned short ans=(crc^onech << 8);

    for (i=0;i<8;i++) {
        if (ans & 0x8000)
            ans = (ans <<= 1) ^ 4129;
        else
            ans <<= 1;
    }
    return ans;
}
```

One state = one clock cycle



Full loop unrolling

RTL simulation

icrc.c

```
unsigned short icrc1(unsigned short crc,
unsigned char onech)
{
    …
}
```

testbench.c

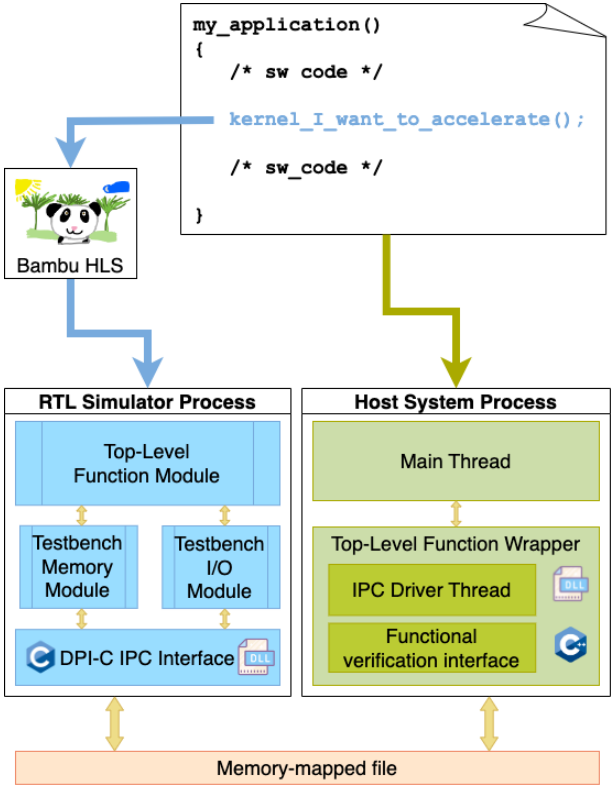
```
unsigned short main()
{
   unsigned short crc = 245;
   unsigned char onech = 5;

   unsigned short res = icrc1(crc, onech);

   crc = 134;
   onech = 3;

   res += icrc1(crc, onech);

   printf("res = %d\n", res);

   return res;
}
```

```
bambu icrc.c -top-fname=icrc1
--generate-tb=testbench.c
--simulate
```

```
Total cycles            : 2 cycles
  Number of executions  : 2
  Average execution     : 1 cycles
```

RTL simulation

```
bambu icrc.c –
top-fname=icrc1
--generate-
tb=testbench.c
--simulate
```

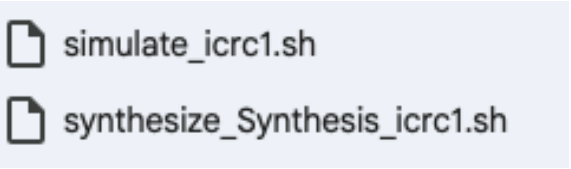



```
Total cycles : 2
cycles
  Number of
executions : 2
  Average execution
: 1 cycles
```

RTL simulation & synthesis

```
bambu icrc.c -top-fname=icrc1 --generate-tb=testbench.c
--evaluation
```

```
Slices                  : 10
Luts                    : 27
Registers               : 23
Frequency               : 321.95750160978753
Total cycles            : 20 cycles
Number of executions    : 2
Average execution       : 10 cycles
```



Interfaces

Inferred interfaces

When the `--generate-interface=INFER` option is specified, Bambu evaluates the following factors, in order of precedence, to set the correct interface protocol for each argument of the top function:

1. Presence of a user-specified XML file (`--interface-xml-filename=<filename>`)
2. Presence of `HLS_interface` pragmas in the code
3. Default assignment according to the argument type

In cases 1. and 2. the user selects the desired interfaces for one or more function arguments through the options described here. For all arguments without a specification, and in case 3., the following table applies:

C argument type	Direction	Interface protocol
Scalar (pass by value)	in	`none`
Array	in/out	`array`
Pointer or reference	in	`none`
Pointer or reference	out	`valid`
Pointer or reference	in/out	`o_valid`

Interfaces

Example: AXI4

```
#pragma HLS interface port = v mode = m_axi offset = direct
#pragma HLS interface port = n mode = m_axi offset = direct
int sum(int* v, unsigned* n)
{
   int sum = 0;

   for(unsigned i = 0; i < *(n); i++)
   {
      sum += v[i];
   }

   return sum;
}
```

```
module sum(clock,
  reset,
  start_port,
  v,
  n,
  m_axi_n_awready,
  m_axi_n_wready,
  …
  m_axi_v_awready,
  m_axi_v_wready,
  …
```

Interfaces

Example: AXI4 + caches

```
/* AXI pragmas */
#pragma HLS interface port = a mode = m_axi offset = direct bundle = gmem0

/* Cache pragmas */
#pragma HLS cache bundle = gmem0 line_count = 16 line_size = 16 bus_size = 32 ways = 1
     num_write_outstanding = 2 rep_policy = lru write_policy = wt
```

```
module IOB_cache_axi(clock,
  reset,
  flush,
  dirty,
  valid,
  addr,
  wdata,
  …
```

107.8x speedup on matrix multiplication!

Frontend options

Very common

`--compiler=I386_GCC/CLANG -O<level>`

Possibly useful

`-f/-fno`

Any other GCC/Clang option

Also: `--experimental-setup=<setup>` as aggregation of multiple options:

`BAMBU-AREA(-MP)`

`BAMBU-BALANCED(-MP)`

`BAMBU-PERFORMANCE(-MP)`

HLS options

Very common

```
--pipelining,-p=<func_name>[=<init_interval>]
```

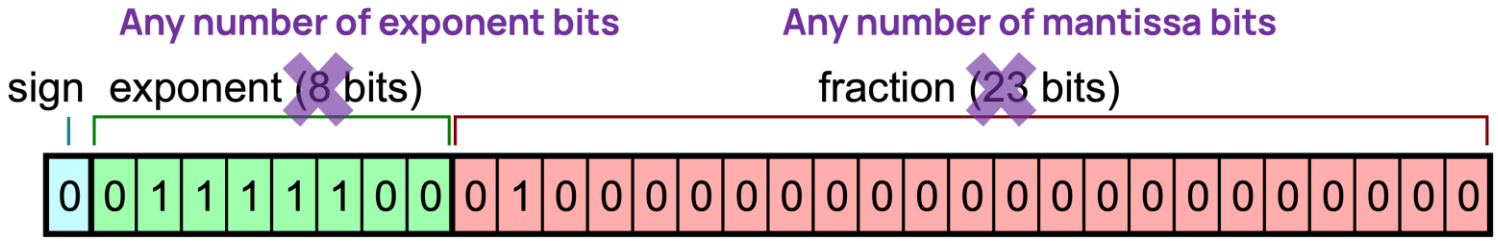
Possibly useful

```
--memory-allocation-policy
```

Expert use only

```
--register-allocation, --module-binding,
--parametric-list-based, --speculative-sdc-scheduling
```

Arithmetic options



Possibly useful

```
--fp-format=<func_name>
*e<exp_bits>m<frac_bits>b<exp_bias>
<rnd_mode><exc_mode><?spec><?sign>
```

Expert use only

```
--hls-div, --hls-fpdiv
```

Backend options

Very common

```
--simulator, --generate-vcd
```

Possibly useful

```
--mem-delay-read, --mem-delay-write
```

Thank you!

https://github.com/ferrandi/PandA-bambu

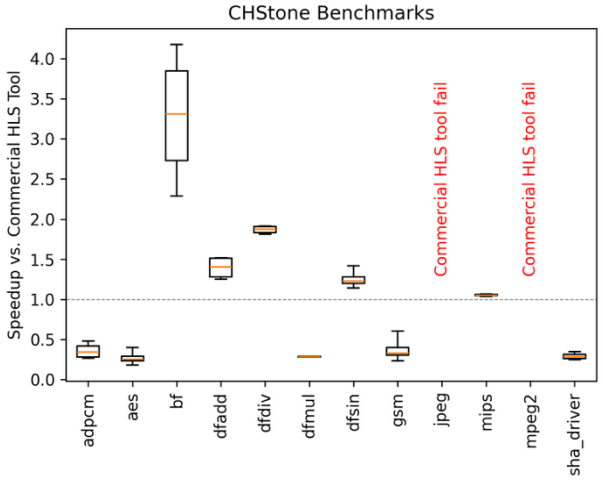
Backup slides

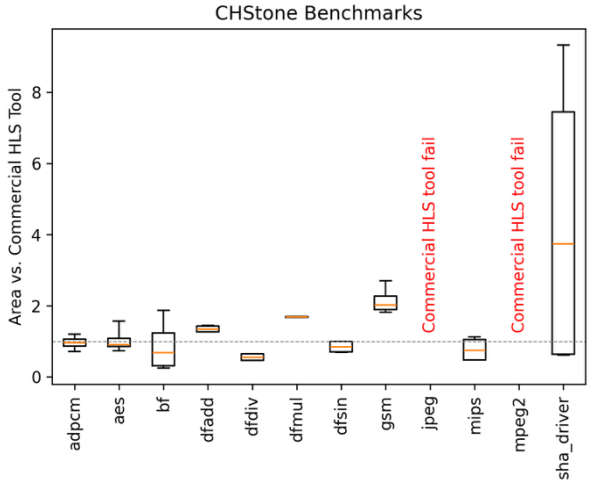
Bambu quality of results

CHStone: speedup and area consumption over commercial HLS tool across different Bambu configurations.

*Latency is measured in ns (clock cycles * achieved period post-implementation). > 1 is better.*

*Area is measured in Equivalent LUTs (BRAMs * 40 + DRAMs * 40 + DSPs * 40 + Registers * 0.5 + LUTs). < 1 is better.*

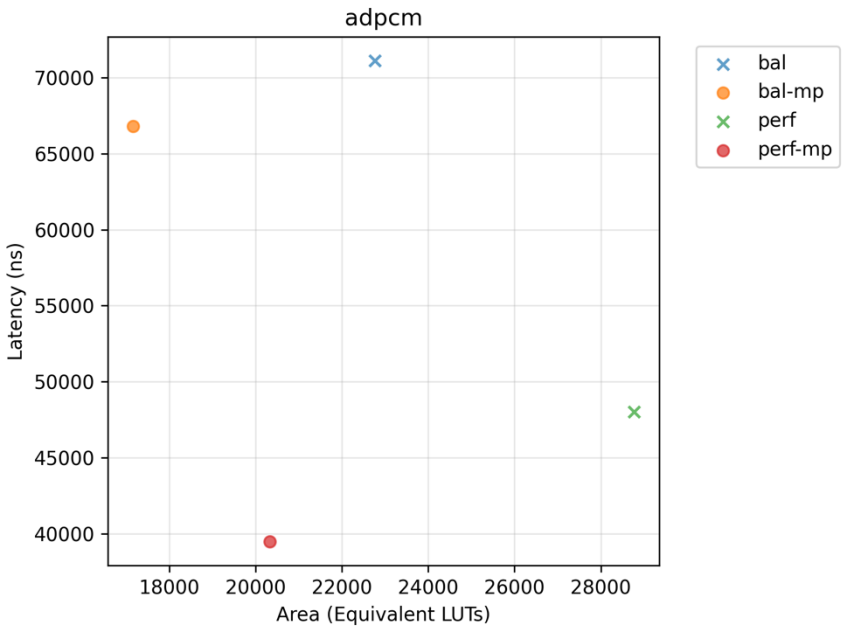


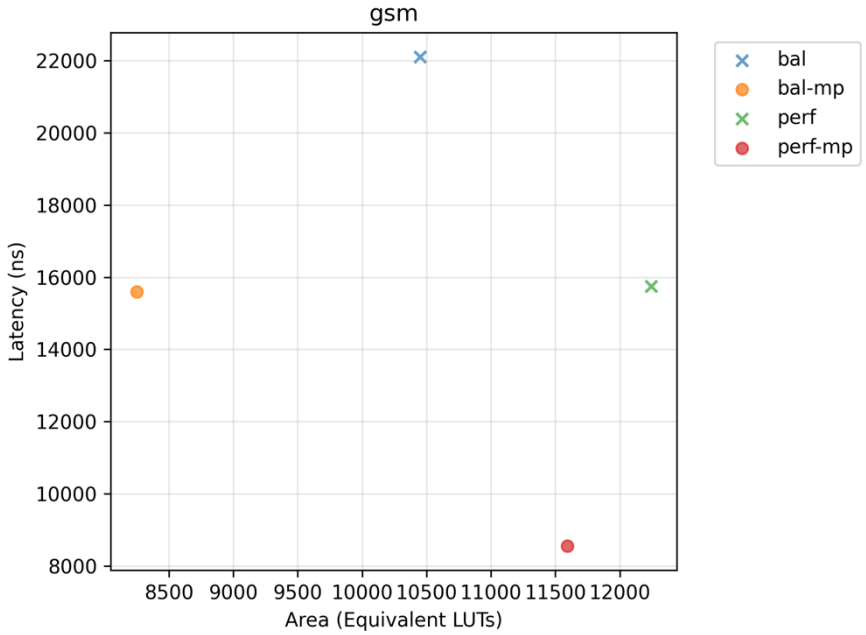


Bambu quality of results

Pareto plots (Latency vs. Area) for selected benchmarks.

Points marked with x are dominated.

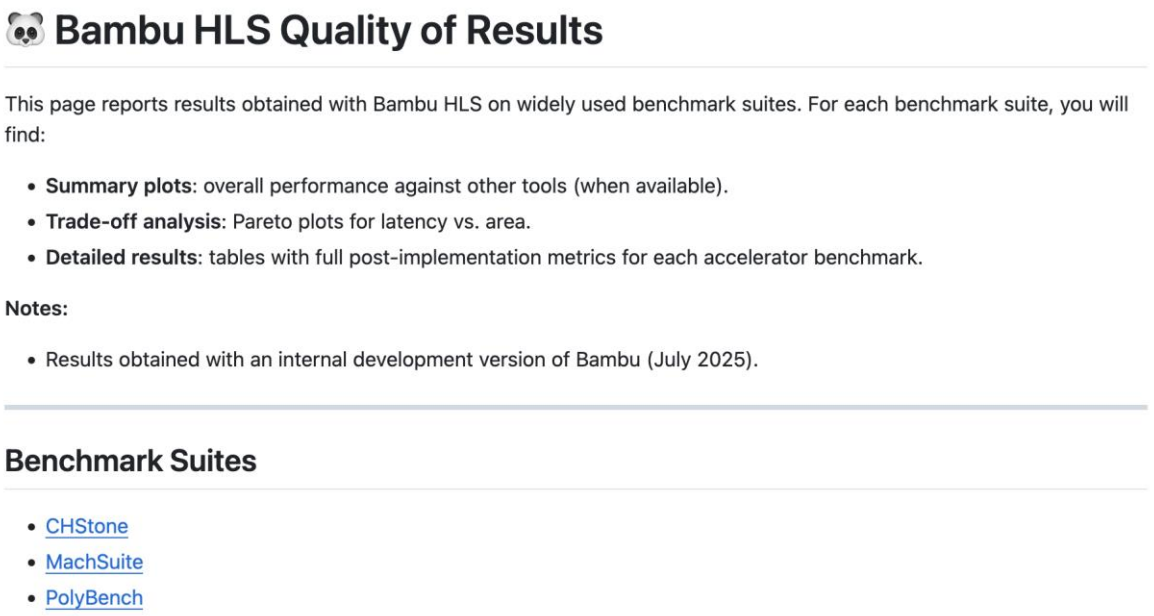




Bambu quality of results

Other benchmarks + raw data at:

https://github.com/ferrandi/PandA-bambu/wiki/Quality-of-Results



Bambu HLS Quality of Results

This page reports results obtained with Bambu HLS on widely used benchmark suites. For each benchmark suite, you will find:

- **Summary plots**: overall performance against other tools (when available).
- **Trade-off analysis**: Pareto plots for latency vs. area.
- **Detailed results**: tables with full post-implementation metrics for each accelerator benchmark.

Notes:

- Results obtained with an internal development version of Bambu (July 2025).

Benchmark Suites

- CHStone
- MachSuite
- PolyBench