



POLITECNICO
MILANO 1863

DEPARTMENT OF ELECTRONICS
INFORMATION AND BIOENGINEERING

September 1st, 2025

Fast Machine Learning for Science Conference 2025

Bambu: An Open-Source Research Framework for the High-Level Synthesis of Complex Applications

09.1.2025 | **Fabrizio Ferrandi**, Serena Curzel, Michele Fiorito, Giovanni Gozzi

Outline

1. **Bambu HLS**

Bambu compilation flow

FPGA integration

2. **Bambu features**

Interface generation

Dataflow

Functional and loop pipelining

3. **Research opportunities**

AI/ML design flow integration

MLIR-based design flows

Verification and debug

Floating point customization

Function proxies

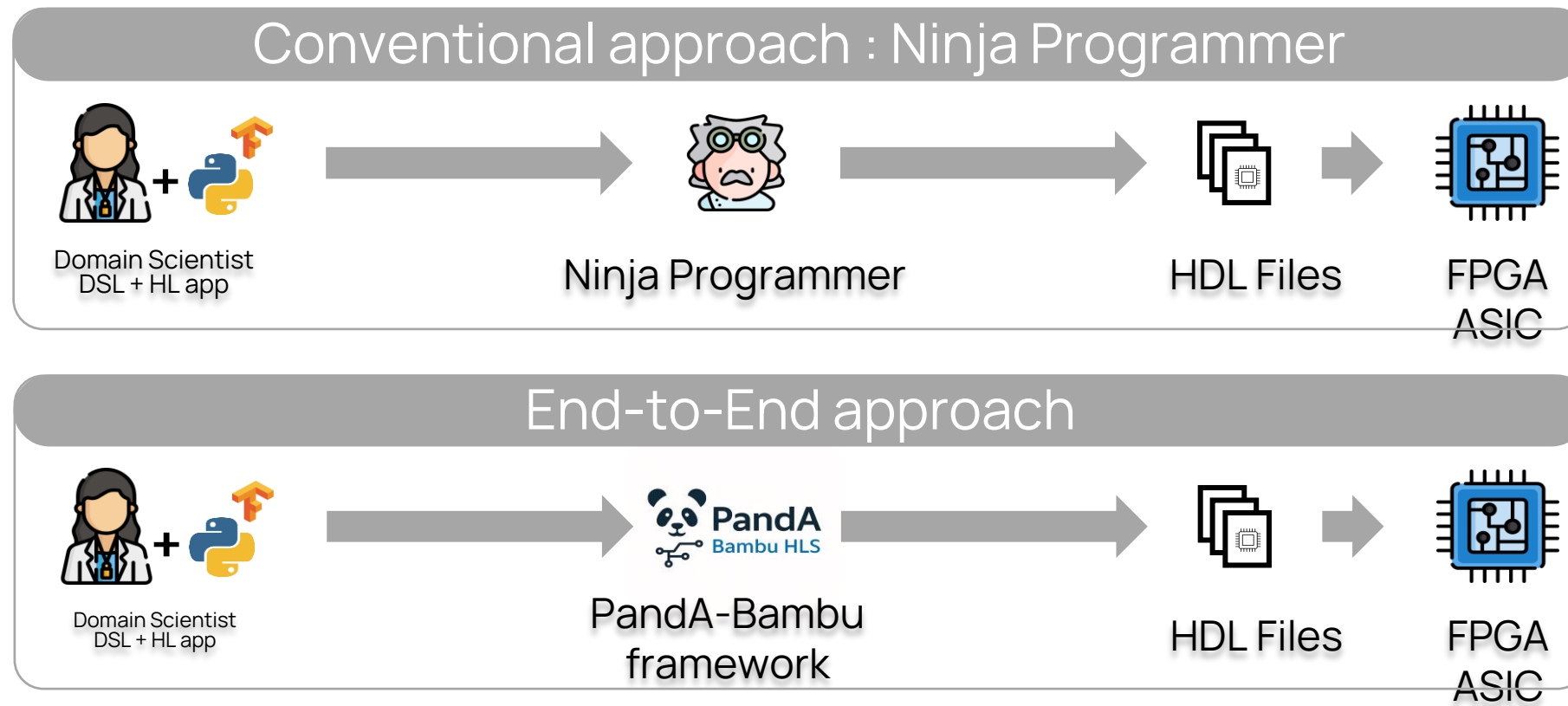
Parallel architectural template

4. **Conclusions**

Bambu HLS

01

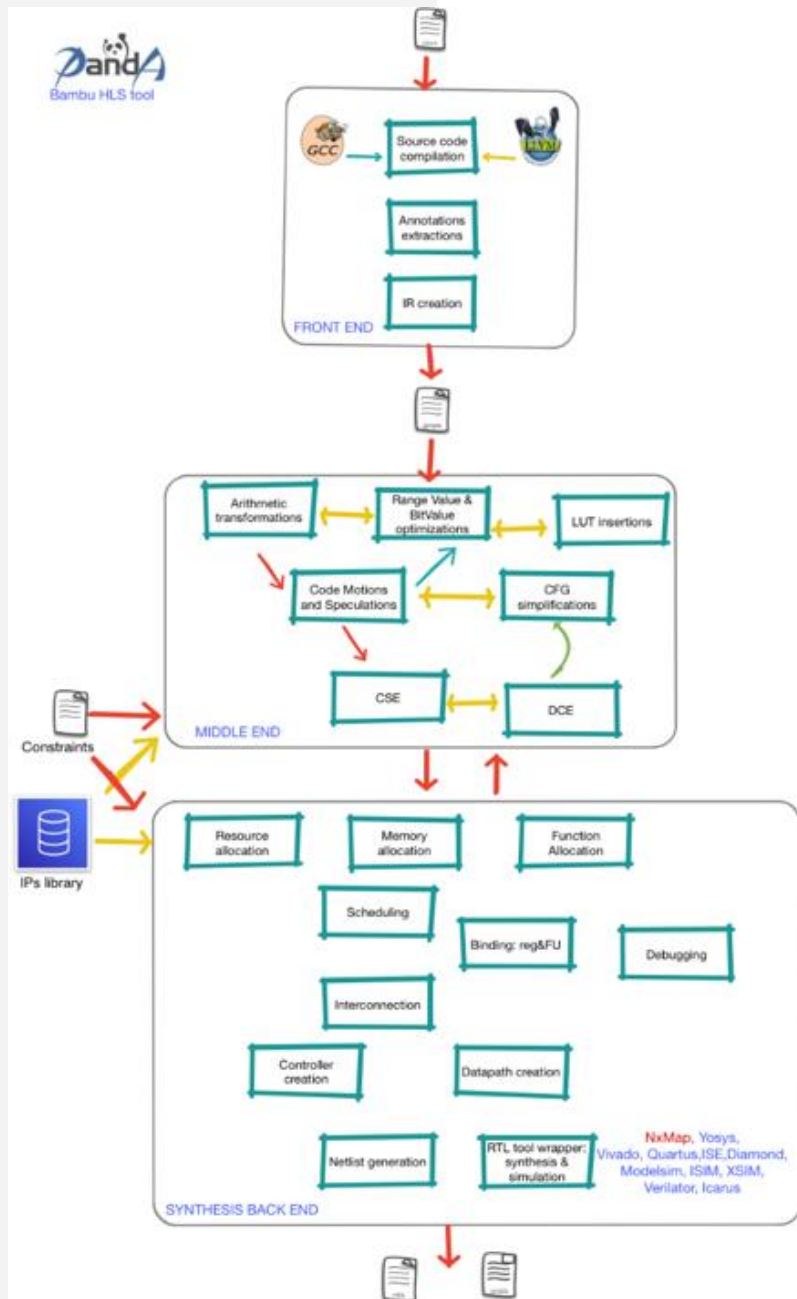
High-level synthesis of complex applications



Bambu: an example of modern HLS tools

Open-source HLS tool developed at Politecnico di Milano (Italy)

- 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
 - 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 Verilator, Xilinx Xsim, Mentor Modelsim
- Back-end: Automated interaction with commercial synthesis tools
 - FPGA: Xilinx ISE, Xilinx Vivado, Altera Quartus, Lattice Diamond, NanoXplore
 - ASIC: OpenRoad (Nangate 45, ASAP7)



Compilation Flow

HLS tools simplify the implementation of accelerators on FPGA

HLS starts from high-level languages (C/C++)

- Optimizes the intermediate representations
- Allocates resources
- Schedules operations
- Binds them to the resources
- And generates RTL descriptions for synthesis tools

The increased performance offered by FPGAs is made available also to software developers that do not have hardware design expertise

Front-end

```

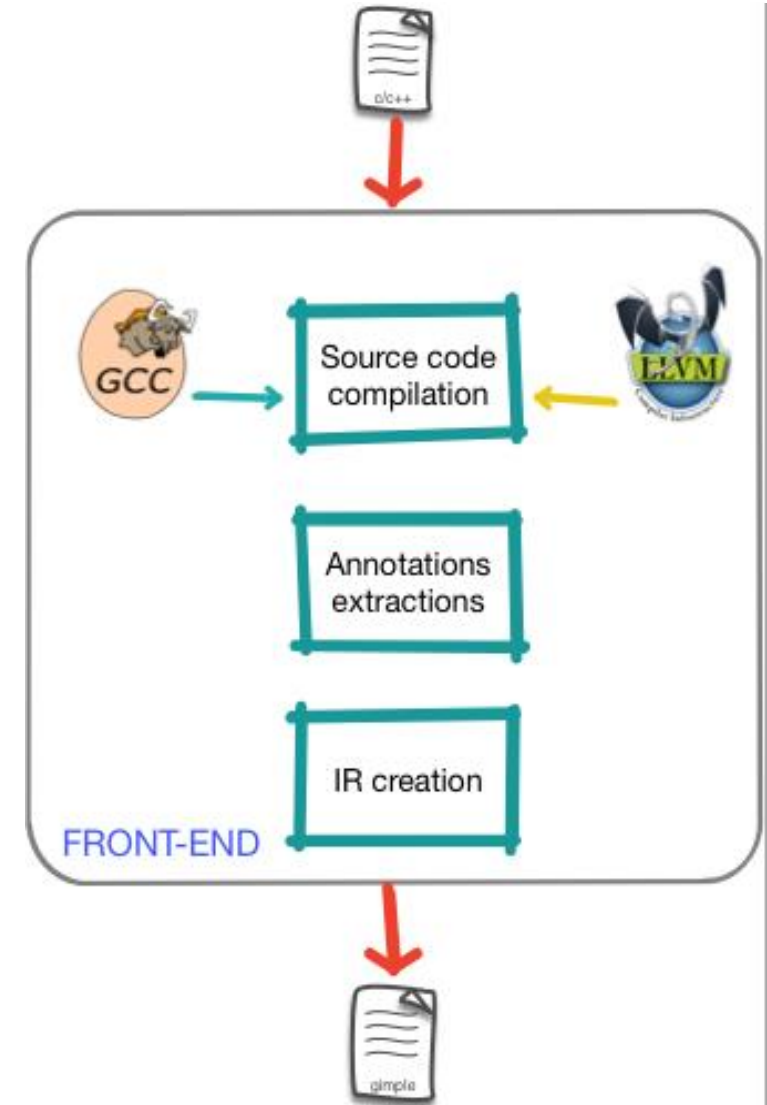
1 #pragma HLS interface port = a mode = fifo
2 #pragma HLS interface port = b mode = fifo
3 #pragma HLS interface port = c mode = fifo
4 #pragma HLS interface port = d mode = fifo
5 void sum3numbers(short* a, short* b, short* c, short* d)
6 {
7     int i;
8     #pragma nounroll
9     for(i = 0; i < 8; ++i)
10        d[i] = a[i] + b[i] + c[i];
11 }

```

```

1 #define A_ROWS 16
2 #define A_COLS 16
3 #define B_ROWS 16
4 #define B_COLS 16
5 // matrix multiplication of a A*B matrix
6 void mm (int in_a[A_ROWS][A_COLS], int in_b[A_COLS][B_COLS], int out_c[A_ROWS][B_COLS])
7 {
8     int i,j,k;
9     for (i = 0; i < A_ROWS; i++)
10    {
11        for (j = 0; j < B_COLS; j++)
12        {
13            int sum_mult = 0;
14            #pragma HLS unroll
15            for (k = 0; k < A_COLS; k++)
16            {
17                sum_mult += in_a[i][k] * in_b[k][j];
18            }
19            out_c[i][j] = sum_mult;
20        }
21    }
22 }

```



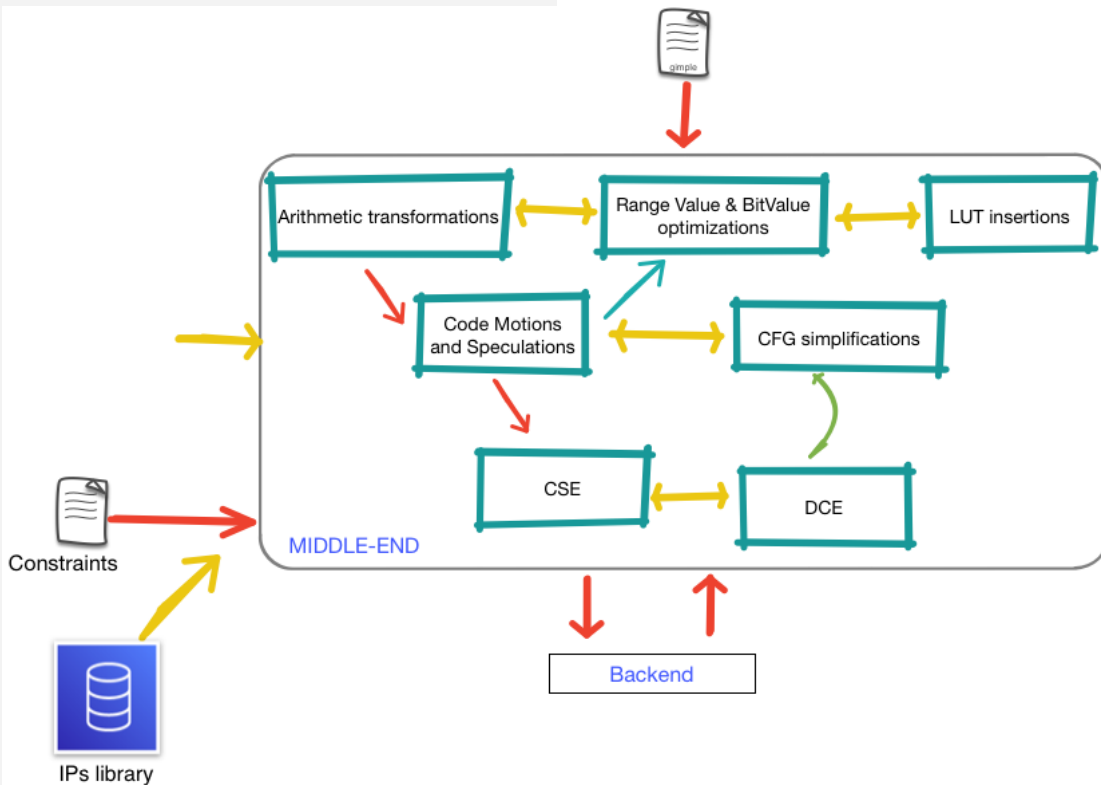
Compilation flow of the Bambu HLS front-end

Middle-end

At this stage, several transformations are applied to simplify the intermediate representations.

The transformations are focused on adapting the code to the HLS process by:

- Simplifying constant multiplication and division
- Propagating constants
- Performing “if conversions” where is needed
- Removing dead code
- Restricting the range of the values



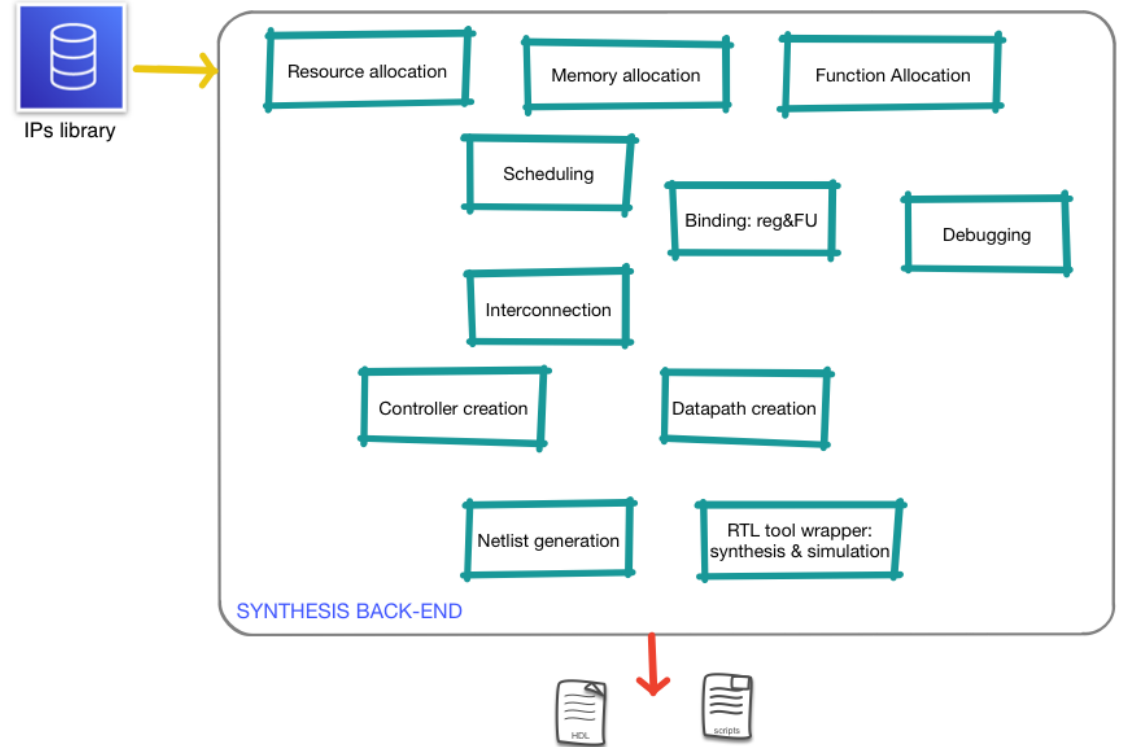
Compilation flow of the Bambu HLS middle-end

Back-end

Once the IR has been simplified, the HLS steps start:

- Resource and memory allocation
- Scheduling
- Module and register binding
- FSM&Datapath creation

Followed by the simulation and synthesis scripts generation.



Compilation flow of the Bambu HLS back-end

Target-aware HLS

Seamless integration between Bambu and existing FPGA vendors

- automatic generation of backend synthesis scripts

Bambu IP resource library characterized with respect to a specific FPGAs

- Resource occupation and latency under different design constraints
- Automatically performed with PandA Eucalyptus tool

```

LOBE[Rx2]: 2/20
LOBE[Rx3]: 1/20

Lobe contents:
Lobe: LOBE [Lx2]:
Net: clock from instance clock
Analyzing graph succeeded in 369 milli-seconds

Options
Analysis conditions
MaximumClock
searchPathsLimit

Directions
createClock(falling = 10000, name = "clock", period = 10000, rising = 5000, target = "getClock")

Reporting domains
Domain Target Frequency Hold/Removal Summary
Source Target Required Maximum Slack Minimum Data Minimum Arrival Time Relat
clock (Rising) clock (Rising) 100.000 MHz | 101.005 MHz | 707ps | 707ps |
Total
  
```

Category	Power (mW)	%
Overall	807.28	100.0%
RF	26.32	3.3%
Clocks	0.00	0.0%
Logic	460.00	57.3%
Ram	33.80	4.2%
DSP	47.04	5.8%
IO	25.40	3.1%
CLK	0.31	0.0%
HSSL	2.96	0.4%
SOC	33.65	3.9%

```

<?xml version="1.0"?>
<document>
  <application>
    <section stringID="NANOXPLORE_SYNTHESIS_SUMMARY">
      <item stringID="NANOXPLORE_FE" value="696"/>
      <item stringID="NANOXPLORE_LUTS" value="360"/>
      <item stringID="NANOXPLORE_REGISTERS" value="404"/>
      <item stringID="NANOXPLORE_MEM" value="2"/>
      <item stringID="NANOXPLORE_IOPIN" value="292"/>
      <item stringID="NANOXPLORE_DSPS" value="0"/>
      <item stringID="NANOXPLORE_POWER" value="0.833"/>
      <item stringID="NANOXPLORE_SLACK" value="2.929"/>
    </section>
  </application>
</document>
  
```

Integration of Bambu HLS with NanoXplore

Bambu features

02

Interface generation

Bambu uses pragmas to generate specific interfaces for each parameter passed to the accelerator.

These pragmas are very similar to those used by Vitis HLS and exhibit the same behavior: if no pragma is provided, the default interface for that parameter type is used; otherwise, the interface specified by the pragma is applied.

Supported protocols are:

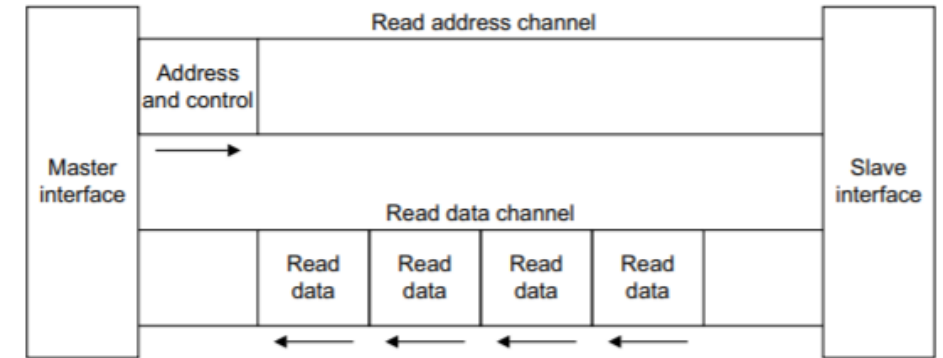
- FIFO, AxiStream
- Handshake
- Array
- Axi4-Master

In many cases it is possible to use the same VitisHLS syntax. Not all the options are supported.

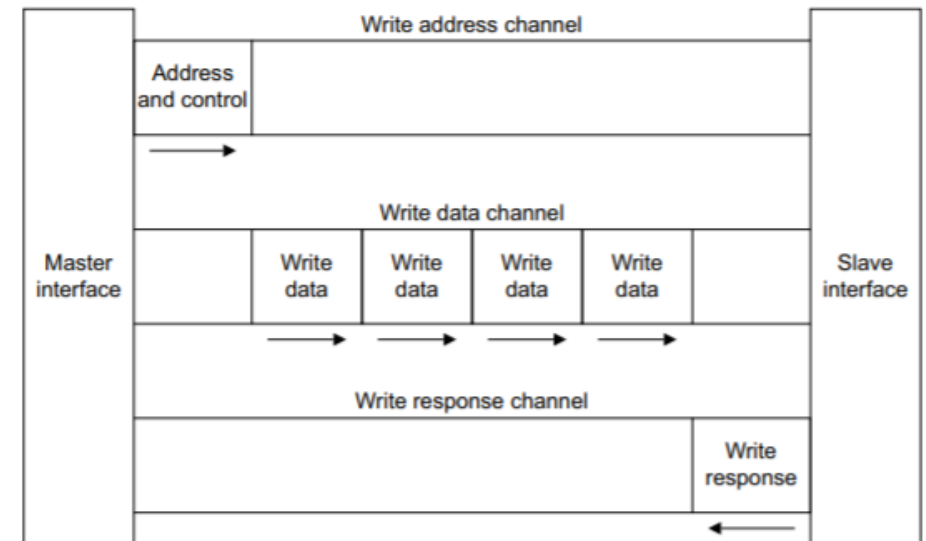
Interface generation - Axi4/AxiS

Support to Axi4 master and AxiS interfaces

- Easier integration with ARM processor on the NG-ULTRA board
- Easy way to integrate existing IPs
- Automatic generation of AXI testbench supported
- Memory latency can be configured
- Unaligned accesses are supported



Channel architecture for read transaction with Axi4 (AMBA® AXI™ and ACE™ Protocol Specification)



Channel architecture for write transaction with Axi4 (AMBA® AXI™ and ACE™ Protocol Specification)

Interface generation - Cache

Support for caches on AXI4 interfaces:

- Customizable cache Parameters: line size, cache size...
- Support for associative caches
- Support for different replacement policies (least recently used...)
- Support for different write policies (write back or write through)
- Support for larger AXI xDATA signal size (up to 512 bits)
- Support for pipelined write transactions
- Support for Axi4 burst transactions
- Automatic cache flush at the end of the computation
- Includes simulation-only hit/miss counters

Interface generation - FIFO

C++ FIFO interface support:

- `ac_channel<T>`
- `hls::stream<T>`

```
void sum3numbers(ac_channel<ap_uint<64>>& a,
                ac_channel<ap_uint<64>>& b,
                ac_channel<ap_uint<64>>& c,
                ac_channel<ap_uint<64>>& d)
{
    int i;
    for(i = 0; i < 8; ++i)
        d.write(a.read() + b.read() + c.read());
}
```

Example of c++ code with ac_channels

Dataflow

```

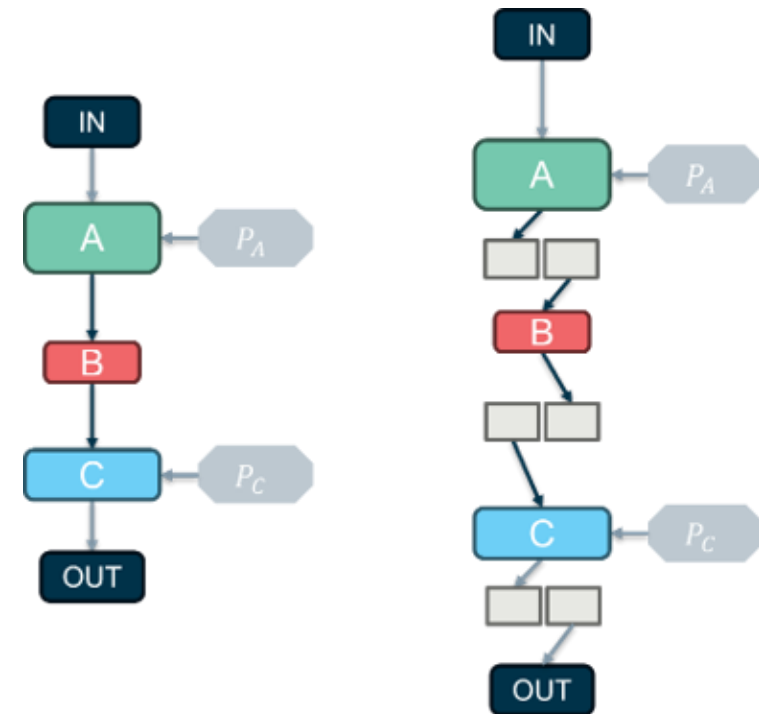
class SimpleSystem {
    AddBlock<5> A;
    MulBlock B;
    SubSystem C;
    ac_channel<int> x, y;

public:
    void top(ac_channel<int>& in1, ac_channel<int>& in2,
            ac_channel<int>& in3, ac_channel<int>& out) {
        #pragma HLS dataflow
        A.compute(in1, x);
        B.compute(x, in2, y);
        C.compute(y, in3, out);
    }
};

void dataflow_top(ac_channel<int>& in1, ac_channel<int>& in2,
                 ac_channel<int>& in3, ac_channel<int>& out)
{
    static SimpleSystem sys;
    sys.top(in1, in2, in3, out);
}

```

Source code in C++ with pragma dataflow

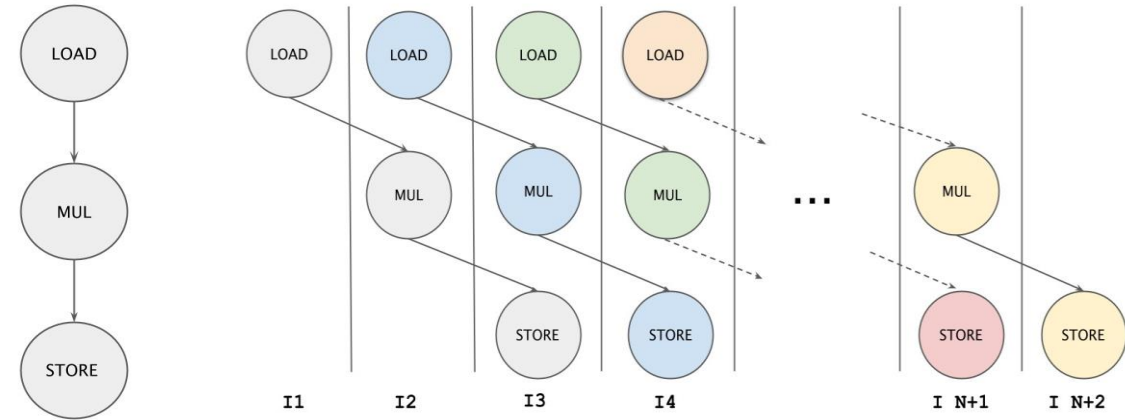


Comparison of the resulting task graph with and without dataflow

Functional and Loop pipelining

Extended function pipelining support

- User-defined initiation interval (II)
- Optimized pipelining algorithm



Loop pipelining overlaps execution of different iterations

- Latency reduction, less area overhead than loop unrolling
- Requires information about operation dependencies and available resources
- Goal: initiation interval (II) = 1

Research opportunities

03

Complete and modular HLS flow

Bambu is open-source, available on GitHub, and can be used to advance research in High-Level Synthesis (HLS).

Over the years, numerous algorithms and synthesis techniques have been added to Bambu to improve performance, area, and flexibility.

For instance, Bambu includes multiple allocation and binding algorithms, such as:

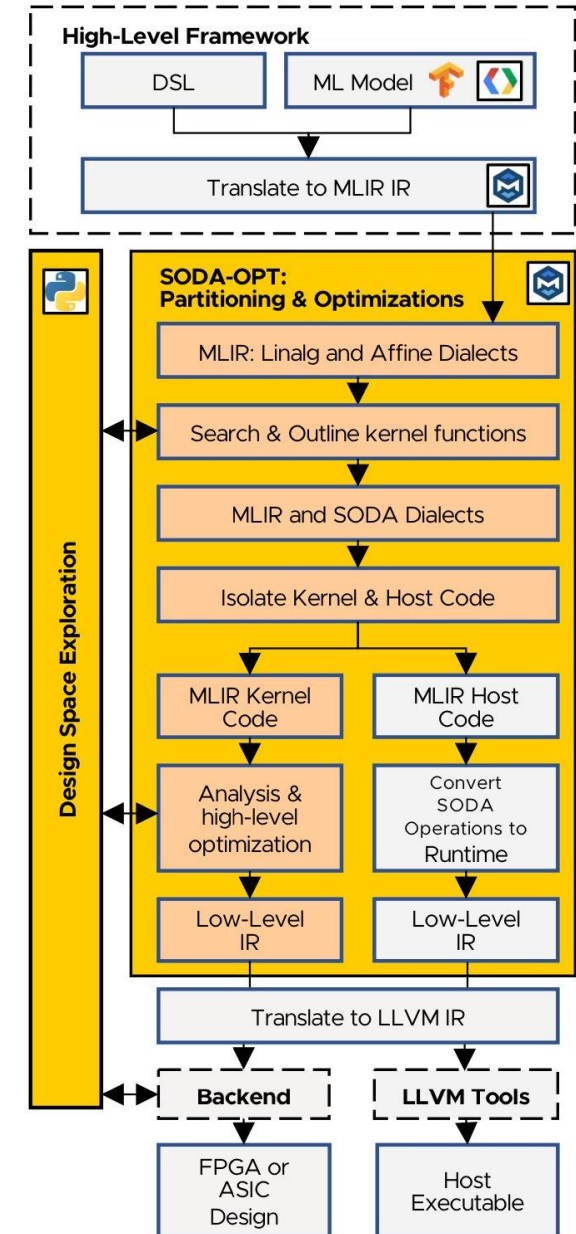
```
--register-allocation= < type >
    WEIGHTED TS      - solve the weighted clique covering problem by exploiting the
                      Tseng&Siewiorek heuristics (default)
    WEIGHTED COLORING - use weighted coloring algorithm
    COLORING         - use simple coloring algorithm
    CHORDAL COLORING - use chordal coloring algorithm
    ...

--module-binding= < type >
Set the algorithm used for module binding. Possible values for the
< type > argument are one the following:
    WEIGHTED TS      - solve the weighted clique covering problem by exploiting the
                      Tseng&Siewiorek heuristics (default)
    TTT FAST        - use Tomita, A. Tanaka, H. Takahashi maxima weighted cliques heuristic
    BIPARTITE MATCHING - bipartite matching approach
    UNIQUE          - use a 1-to-1 binding algorithm
    ...
```

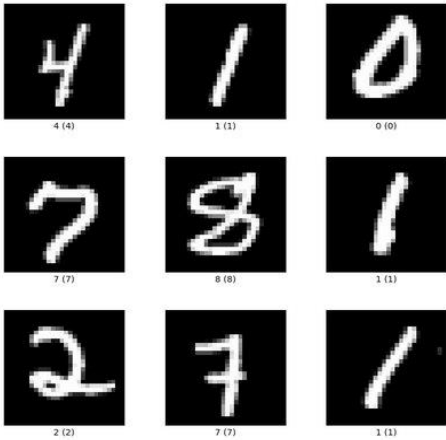
Integrating ML-design flow in Bambu

- Research ongoing on Domain Specific Languages to synthesize accelerators for machine learning-based inference.
- We did some experiments with SODA-OPT framework aiming at optimizing one of the TAS use-cases.
- SODA-OPT jointly developed by PNNL, Northwestern University and Politecnico di Milano

Serena Curzel, Nicolas Bohm Agostini, Vito Giovanni Castellana, Marco Minutoli, Ankur Limaye, Joseph B. Manzano, Jeff Zhang, David Brooks, Gu-Yeon Wei, Fabrizio Ferrandi, Antonino Tumeo, "**End-to-End Synthesis of Dynamically Controlled Machine Learning Accelerators**". IEEE Trans. Computers 71(12): 3074-3087 (2022)

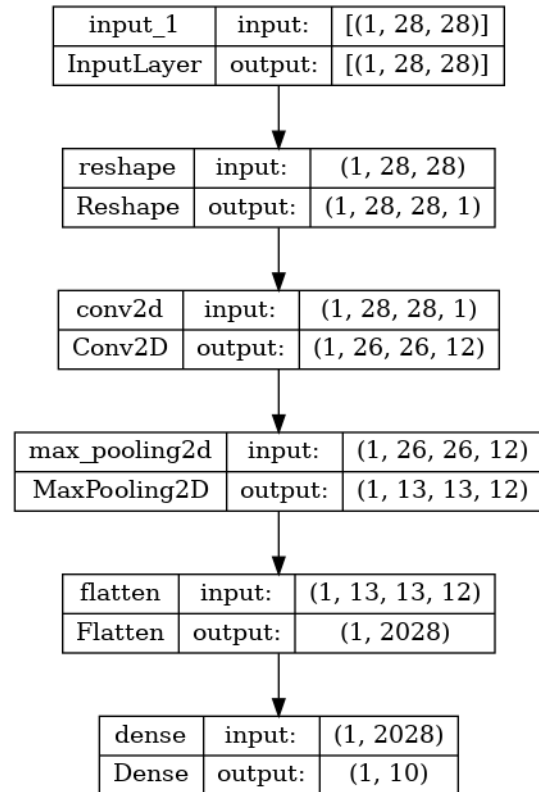


Integrating ML-design flow - MNIST



Example of input to the MNIST kernel

	NG-Ultra embedded
LUTS	4627
Registers	5714
Frequency	45.7 MHz
DSP	54
MEM	34
cycles	169,649

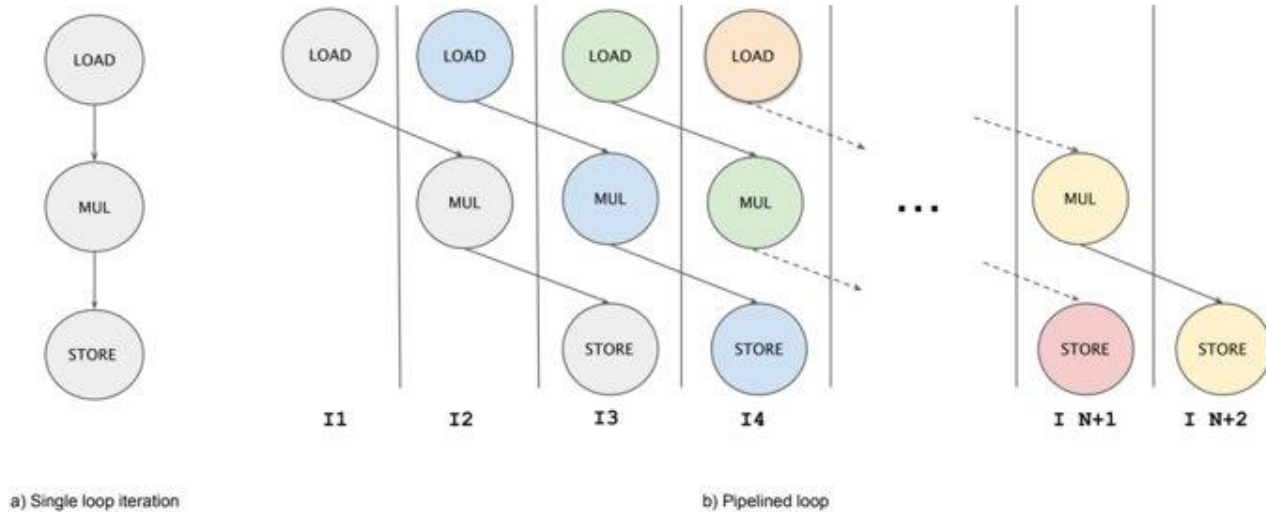


MNIST Network architecture

Synthesis result of the MNIST network in a NG-Ultra FPGA

- The neural network was trained and quantized to 8-bit to reduce area and improve hardware performance
- TensorFlow 2.15 was used for model training and post-training quantization.
- MLIR was used as the input to Clang, enabling flexible optimizations and modular code transformations.
- Clang version 18 was used as the front-end compiler of Bambu

MLIR + Bambu loop pipelining



```

#map = affine_map<(d0) -> (d0 - 2)>
func @example(%arg0: memref<1000xi32>,
              %arg1: memref<1000xi32>) {
  %c0 = arith.constant 0 : index
  %0 = affine.load %arg0[%c0] : memref<1000xi32>
  %c1 = arith.constant 1 : index
  %1 = affine.load %arg0[%c1] : memref<1000xi32>
  %2 = arith.muli %0, %0 : i32
  %3:2 = affine.for %arg2 = 2 to 1000
    iter_args(%arg3 = %1, %arg4 = %2) -> (i32, i32) {
      %5 = affine.load %arg0[%arg2] : memref<1000xi32>
      %6 = arith.muli %arg3, %arg3 : i32
      %7 = affine.apply #map(%arg2)
      affine.store %arg4, %arg1[%7] : memref<1000xi32>
      affine.yield %5, %6 : i32, i32
    }
  %4 = arith.muli %3#0, %3#0 : i32
  %c998 = arith.constant 998 : index
  affine.store %3#1, %arg1[%c998] : memref<1000xi32>
  %c999 = arith.constant 999 : index
  affine.store %4, %arg1[%c999] : memref<1000xi32>
  return
}

```

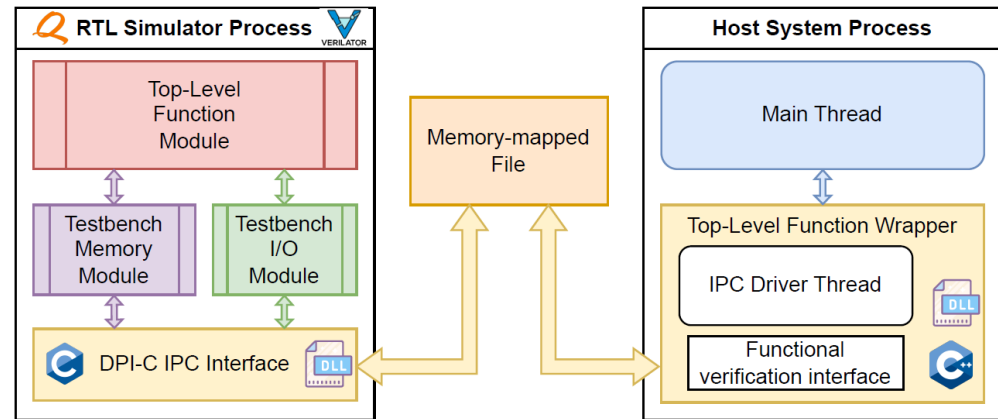
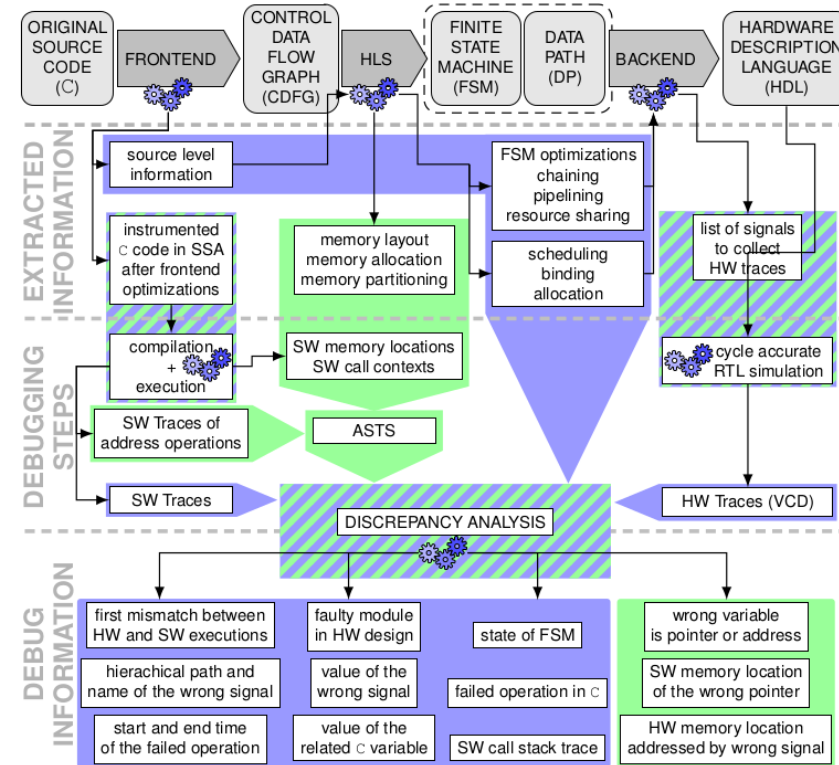
S.Curzel, S.Jovic, M.Fiorito, A.Tumeo and F.Ferrandi, “Higher-level Synthesis: experimenting with MLIR polyhedral representations for accelerator design” IMPACT workshop 2022.

S.Curzel, S.Jovic, M.Fiorito, A.Tumeo and F.Ferrandi. “MLIR Loop Optimizations for High-Level Synthesis: A Case Study” PACT 2022

S.Curzel, S.Jovic, M.Fiorito, A.Tumeo and F.Ferrandi, “Pre-Scheduling of Affine Loops for HLS Pipelining” to be presented at the WHPC Special Session of Euro-Par 2024.

Verification and Debug

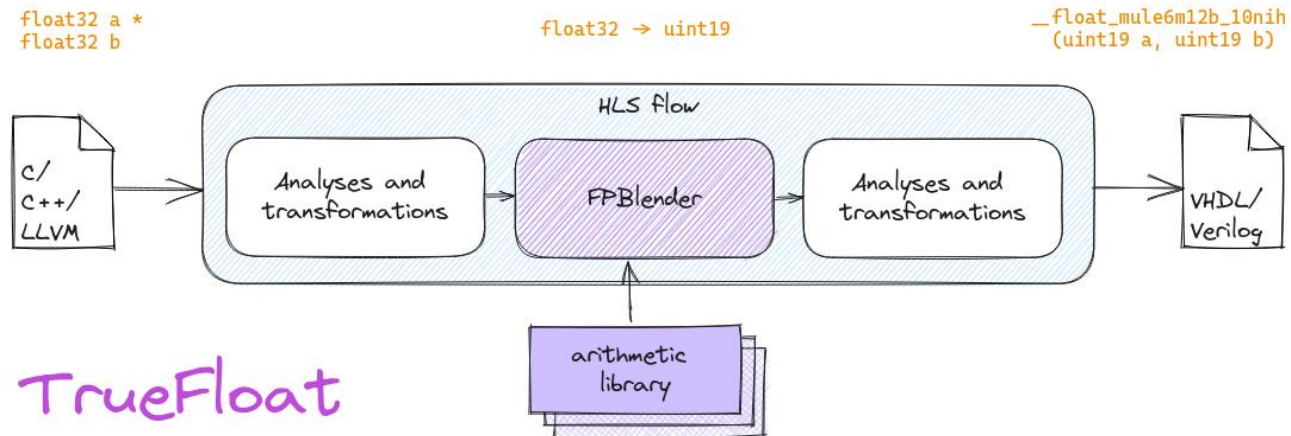
- Automated technique for bug identification in HLS
 - Comparing HLS-generated hardware traces with the software traces
- Automatic Error Detection by C-HDL co-simulation
 - Comparing results from hardware and software simulations to identify inconsistencies automatically



Pietro Fezzardi, Fabrizio Ferrandi, “Automated Bug Detection for High-level Synthesis of Multi-threaded Irregular”, Applications. ACM Trans. Parallel Comput. 7(4): 27:1-27:26 (2020).

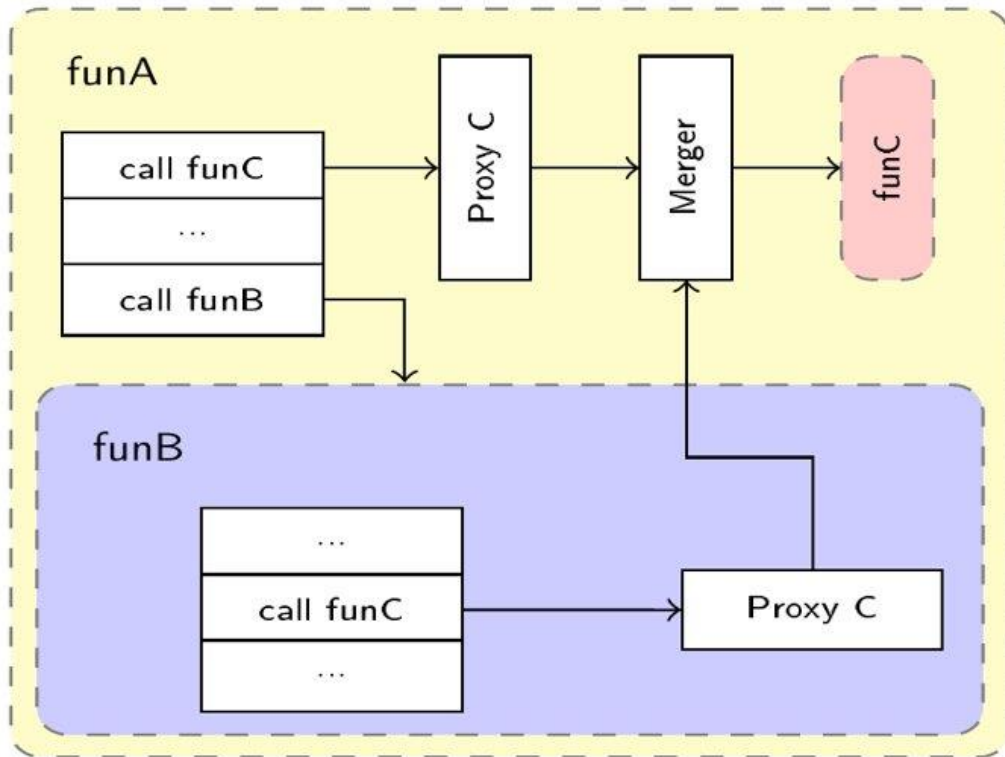
Customization of floating-point formats with TrueFloat

HLS tools usually support custom fixed-point formats, Bambu also has TrueFloat



- Users specify the required format for each function, Bambu generates optimized arithmetic units
- Integrated within the HLS flow -> QoR higher than inserting «black box» IPs
- Effortless translation between encodings through command-line options

Function proxies



- Function proxy allows hardware sharing, enabling multiple functions to reuse the same hardware resources.
- When the functions do not execute concurrently, this sharing has no impact on execution time.

M. Minutoli, V. G. Castellana, A. Tumeo, and F. Ferrandi, “**Inter-procedural resource sharing in High Level Synthesis through function proxies**,” in Proceedings of the 25th International Conference on Field Programmable Logic and Applications, FPL, 2015, pp. 1-8.

Parallel Architecture template

Parallel Architectural template that exploits both instruction level and task-level parallelism:

- Dynamic hardware scheduler
- Single-cycle hardware context switching
- Parallel support with OpenMP: `omp for`, `omp simd`, `omp task`

- Giovanni Gozzi, Michele Fiorito, Serena Curzel, Claudio Barone, Vito Giovanni Castellana, Marco Minutoli, Antonino Tumeo, and Fabrizio Ferrandi. 2024. **SPARTA: High-Level Synthesis of Parallel Multi-Threaded Accelerators**. ACM Trans. Reconfigurable Technol. Syst. 18, 1, Article 9 (March 2025), 30 pages. <https://doi.org/10.1145/3677035>
- M. Minutoli, V. G. Castellana, N. Saporetti, S. Devecchi, M. Lattuada, P. Fezzardi, A. Tumeo, and F. Ferrandi, “**Svelto: High-Level Synthesis of Multi-Threaded Accelerators for Graph Analytics**,” *IEEE Transactions on Computers*, vol. 71, iss. 3, pp. 520-533, 2022.
- M. Lattuada and F. Ferrandi, “**Exploiting Vectorization in High Level Synthesis of Nested Irregular Loops**,” *Journal of Systems Architecture*, vol. 75, pp. 1-14, 2017.
- V. G. Castellana and F. Ferrandi, “**An automated flow for the High-Level Synthesis of coarse-grained parallel applications**,” in Proceedings of the International Conference on Field-Programmable Technology (FPT), 2013, pp. 294-301.

Conclusions

04

Conclusions

- FPGAs are very versatile and suitable for many markets
- Integrating HLS will improve productivity
- Bambu has all the features of a state-of-the-art HLS tool

<https://panda.dei.polimi.it>
<https://github.com/ferrandi/Panda-bambu>

This work has been partially supported by the Spoke 1 "FutureHPC \ & BigData" of the Italian Research Center on High-Performance Computing, Big Data and Quantum Computing (ICSC) funded by MUR Missione 4 - Next Generation EU (NGEU).

