Reflections on 10 years of FloPoCo

Florent de Dinechin

citi lab

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON
Inria informatiques mathématiques
UNIVERSITÉ DE LYON
The FloPoCo project

A generator of **application-specific** hardware arithmetic operators
- open-ended list (division by 3, exp, log, trigs, ... function approximators, FIIR, IIR, ...)
- each operator heavily parameterized

![Diagram of FloPoCo with functional and performance specifications](image)

---

F. de Dinechin

Reflections on 10 years of FloPoCo
The FloPoCo project

- A generator of **application-specific** hardware arithmetic operators
- open-ended list (division by 3, exp, log, trigs, ... function approximators, FIIR, IIR, ...)
- each operator heavily parameterized

![Diagram of FloPoCo process]

- A philosophy of **computing just right**
  - Interface: never output bits that are not numerically meaningful
    (output format $\Rightarrow$ accuracy specification)
  - Inside: never compute bits that are not useful to the final result

---

F. de Dinechin
Reflections on 10 years of FloPoCo
A candidate for the Worst Logo Ever contest

Right: a floating-point exponential (with bits of M. Joldes and B. Pasca)

e^\sqrt{x^2+y^2+z^2}

π

sin

e^x + y

n

∑
i=0

x^i

log

each wire, each component tailored to its context

F. de Dinechin

Reflections on 10 years of FloPoCo
Genesis

Focus on two features

The future
All my life, I have been afflicted with very good students

Very good students tend to write kilolines of (very good?) code

- **FPLibrary** (Jérémie Detrey’s PhD, 2004-2007):
  - open-source VHDL for floating-point +, −, ×, ÷, \(\sqrt{}\),
  - then \(\sin\), \(\cos\), \(\exp\), \(\log\), ...
  - then LNS (logarithm number system) arithmetic
  - plus two generic HW function approximation techniques
All my life, I have been afflicted with very good students

Very good students tend to write kilolines of (very good?) code

- FPLibrary (Jérémie Detrey’s PhD, 2004-2007):
  - open-source VHDL for floating-point +, −, ×, /, \(\sqrt{-}\),
  - then \(\sin, \cos, \exp, \log, \ldots\)
  - then LNS (logarithm number system) arithmetic
  - plus two generic HW function approximation techniques

- ... plus bits of Java/Python/C++ to generate some of the VHDL
  - from SRT tables for division and square root
  - ... to Remez + error analysis + design-space exploration
All my life, I have been afflicted with very good students

Very good students tend to write kilolines of (very good?) code

- FPLibrary (Jérémie Detrey’s PhD, 2004-2007):
  - open-source VHDL for floating-point +, −, ×, /, √, 
  - then sin, cos, exp, log, ...
  - then LNS (logarithm number system) arithmetic
  - plus two generic HW function approximation techniques
- ... plus bits of Java/Python/C++ to generate some of the VHDL
  - from SRT tables for division and square root
  - ... to Remez + error analysis + design-space exploration

A solid and well-tested agile development methodology

*one paper, one bit of quick-and-dirty code*
All my life, I have been afflicted with very good students

Very good students tend to write kilolines of (very good?) code

- **FPLibrary** (Jérémie Detrey’s PhD, 2004-2007):
  - open-source VHDL for floating-point $+,-,\times,/,\sqrt{}$
  - then $\sin,\cos,\exp,\log,...$
  - then LNS (logarithm number system) arithmetic
  - plus two generic HW function approximation techniques
- ... plus bits of Java/Python/C++ to generate some of the VHDL
  - from SRT tables for division and square root
  - ... to Remez + error analysis + design-space exploration

A solid and well-tested agile development methodology

one paper, one bit of quick-and-dirty code

That’s a lot of work doomed to oblivion when the student leaves
(this particular traitor defected to finite-field arithmetic)
And then a scientific Grand Plan

When FPGAs are better at floating-point than microprocessors
And then a scientific Grand Plan

When FPGAs are better at floating-point than microprocessors

- Submitted to ISFPGA
- In my humble opinion, a visionary paper
  “We can do this, we should do that”
- Tepid reviews (“prove it”, “lack of results”)...
- \( \rightarrow \) poster

Then, overwhelming response to the poster...
Evolution of the Grand Plan

Initial brand

When FPGAs are better at floating-point than microprocessors
Initial brand

When FPGAs are better at floating-point than microprocessors
Not your neighbour’s FPU
Evolution of the Grand Plan

Initial brand

When FPGAs are better at floating-point than microprocessors
Not your neighbour’s FPU

First rebranding

FPGA-specific arithmetic (floating-point, but not only)
Evolution of the Grand Plan

Initial brand
When FPGAs are better at floating-point than microprocessors
Not your neighbour's FPU

First rebranding
FPGA-specific arithmetic (floating-point, but not only)
All the operators you will never see in a processor (and how to build them)
(Arith 2011 panel)
Evolution of the Grand Plan

Initial brand

When FPGAs are better at floating-point than microprocessors
Not your neighbour’s FPU

First rebranding

FPGA-specific arithmetic \hspace{1cm} (floating-point, but not only)
All the operators you will never see in a processor \hspace{1cm} (and how to build them)
(Arith 2011 panel)

Current rebranding

Application-specific arithmetic \hspace{1cm} (FPGA, but not only)
Evolution of the Grand Plan

Initial brand

When FPGAs are better at floating-point than microprocessors
Not your neighbour’s FPU

First rebranding

FPGA-specific arithmetic  (floating-point, but not only)
All the operators you will never see in a processor  (and how to build them)

(Arith 2011 panel)

Current rebranding

Application-specific arithmetic  (FPGA, but not only)
Circuits computing just right

Save routing! Save power! Don’t move around useless bits!

F. de Dinechin  Reflections on 10 years of FloPoCo
Other technical motivations (piling up with the code)

- VHDL doesn’t scale well with number of parameters
  (especially with Jeremie insisting in writing recursive hardware)
Other technical motivations (piling up with the code)

- VHDL doesn’t scale well with number of parameters
  (especially with Jeremie insisting in writing recursive hardware)

---

<table>
<thead>
<tr>
<th>Research code</th>
<th>design-space exploration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- I/O sizes
- ... but also design choices (e.g. SRT radix etc)
- ... and open-ended parameters (e.g. the constant in a constant multiplier)
- ... and we want to parameterize with the target FPGA!
Other technical motivations (piling up with the code)
- VHDL doesn’t scale well with number of parameters (especially with Jeremie insisting in writing recursive hardware)

Research code $\iff$ design-space exploration $\iff$ many many parameters
- I/O sizes
- ... but also design choices (e.g. SRT radix etc)
- ... and open-ended parameters (e.g. the constant in a constant multiplier)
- ... and we want to parameterize with the target FPGA!

A recurrent silly promise, each time we submit a paper: “the design will be pipelined in the final version”
- a perfect waste of good student’s time
- exponential complexity WRT number of parameters
Other technical motivations (piling up with the code)
- VHDL doesn’t scale well with number of parameters
  (especially with Jeremie insisting in writing recursive hardware)

Research code $\iff$ design-space exploration
$\iff$ many many many parameters
- I/O sizes
- ... but also design choices (e.g. SRT radix etc)
- ... and open-ended parameters (e.g. the constant in a constant multiplier)
- ... and we want to parameterize with the target FPGA!

- A recurrent silly promise, each time we submit a paper:
  “the design will be pipelined in the final version”
  - a perfect waste of good student’s time
  - exponential complexity WRT number of parameters
- Heroic experiments with Xilinx JBits
Disputable Technical Choices (erreurs de jeunesse?)

- C++ because Jérémie had written HOTBM in C++
- Generating VHDL because FPLibrary was written in VHDL
- A very modest approach to VHDL generation
  (print out the VHDL code of FPLibrary)

Still, immediate benefits

- single code base
- scaling with parameterization
- and very soon: automatic pipelining
Focus on two features

Genesis

Focus on two features

The future
So much VHDL to write, so few slaves to write it

Adder  Multiplier  Complex product
Multi-adder  Constant multiplier

Polynomial  Multipartite

I know how to optimize by hand each operator on each target
... But I don’t want to do it.
One data-structure to rule them all

Adder | Multiplier | Complex product
---|---|---
Multi-adder | Constant multiplier | Polynomial
Multipartite

Algorithmic description

\[ \sum b_i 2^{w_i} \]

Architecture generation

Spartan 5 | Spartan 6 | Virtex-4 | Virtex-5 | Virtex-6 | Kintex-7 | Zynq 7000 | Stratix V | Stratix 10

The sum of weighted bits as a first-class arithmetic object

- A very wide class of operators: multi-valued polynomials, and more
  - the \( b_i \) can come from look-up tables (e.g. multipartite method)
- Bit-level parallelism, bit-level optimization opportunities
- Generating an architecture is well known:
  
  bit array compressor trees can be optimized for each target
When you have a good hammer, you see nails everywhere

A sine/cosine architecture (Ištoan, HEART 2013):

\[
\begin{array}{c|c|c|c}
 s & q & o & A \\
\hline
 \text{Sin/Cos table} & \times \pi & Z^2/2 & Z^3/6 \\
\hline
 \sin\pi A & \cos\pi A & \sin A \cos Z & \cos A \cos Z \sin A \sin Z \\
\hline
 \times & \times & \times & \times \\
\hline
 \sin A \cos Z & \cos A \cos Z & \sin A \sin Z & \cos A \sin Z \\
\hline
 \text{Swap/negate} \\
\end{array}
\]
When you have a good hammer, you see nails everywhere

A sine/cosine architecture (Istoan, HEART 2013): 5 bit heaps
A bit heap for $Z - Z^3/6$ in the previous architecture

Full bit heap

Faithfully rounded bit heap (computing just right)

Why are some people still insisting I should call this “bit arrays”?
Bit heaps for other operators and filters
It sounds like another Grand Plan

Arithmetic core generation using bit heaps

- Submitted to Arith 2013
- In my humble opinion, a visionary paper

Tepid reviews ("bit arrays are old stuff", "lack of results", "many papers already with merged operators")...
⇒ reject

Conclusion: I'm not very good at writing visionary papers...

New interest in bit heap array compression
and I think Martin Kumm more or less solved it
It sounds like another Grand Plan

Arithmetic core generation using bit heaps

- Submitted to Arith 2013
- In my humble opinion, a visionary paper
- Tepid reviews ("bit arrays are old stuff", "lack of results", "many papers already with merged operators")...

⇒ reject

Conclusion: I'm not very good at writing visionary papers...

New interest in bit-heap array compression

and I think Martin Kumm more or less solved it

F. de Dinechin
Reflections on 10 years of FloPoCo
It sounds like another Grand Plan

Arithmetic core generation using bit heaps

- Submitted to Arith 2013
- In my humble opinion, a visionary paper
- Tepid reviews ("bit arrays are old stuff", "lack of results", "many papers already with merged operators")...
- ⇒ reject

Conclusion: I’m not very good at writing visionary papers...

New interest in bit-heap array compression

- and I think Martin Kumm more or less solved it
Second focus: optimization techniques

I used to write ad-hoc heuristics to optimize my architectures.

I’m now facing an invasion of generic optimization libraries!

- Euclidean lattices for function approximation
- Integer linear programming for
  - function approximation
  - bit heap compression (several algos)
  - constant multiplication design (several algos)

*When you have a good hammer, you see nails everywhere.*

- What, no SAT solving yet?
The future

Genesis

Focus on two features

The future
HLS killed the FloPoCo star?

(HLS means High-Level Synthesis, also known as C-to-hardware)

- Several successful experiments exploiting C descriptions of floating-point operators
- HLS does better what FloPoCo did 10 years ago
  - Optimize a floating-point operation for its context because the compiler knows the context
  - automatic pipelining for the whole application! (out of reach of FloPoCo)
- Some design-space explorations cannot be done in HLS
  - constant multipliers, function approximators

HDL generators \iff HLS source-to-source tools?

(meanswhile, FloPoCo is being used as a back-end for open-source HLS projects such as Bambu or Origami)
Coarser and coarser operators: Where do we stop?

- As long as I can compute it just right, it is in scope of FloPoCo
- (for instance, I’m not sur AI accelerators are in scope...)

Better code separation between arithmetic optimization and VHDL generation so that the arithmetic optimization can be shared with HLS.

Generating parametric hardware was the easy part!

The difficult part of the problem is: What precision is needed at this point of this application?
Coarser and coarser operators: Where do we stop?
- As long as I can compute it just right, it is in scope of FloPoCo
- (for instance, I’m not sur AI accelerators are in scope…)

Find the proper balance between maintaining a tool and advancing research?
- very good students tend to leave a lot of mess behind…
Coarser and coarser operators: Where do we stop?
  - As long as I can compute it just right, it is in scope of FloPoCo
  - (for instance, I’m not sur AI accelerators are in scope…)

Find the proper balance between maintaining a tool and advancing research?
  - very good students tend to leave a lot of mess behind…

Better code separation between arithmetic optimization and VHDL generation
  - so that the arithmetic optimization can be shared with HLS
The next ten years

- Coarser and coarser operators: Where do we stop?
  - As long as I can compute it just right, it is in scope of FloPoCo
  - (for instance, I’m not sure AI accelerators are in scope...)

- Find the proper balance between maintaining a tool and advancing research?
  - very good students tend to leave a lot of mess behind...

- Better code separation between arithmetic optimization and VHDL generation
  - so that the arithmetic optimization can be shared with HLS

Generating parametric hardware was the easy part!
The difficult part of the problem is: What precision is needed at this point of this application?
Thanks for your attention

Thanks to all contributors
S. Banescu, L. Besème, N. Bonfante,
M. Christ, N. Brunie, S. Collange, J. Detrey,
P. Echeverría, F. Ferrandi, L. Forget, M. Grad,
K. Illyes, M. Istoan, M. Joldes, J. Kappauf, C. Klein,
M. Kleinlein, M. Kumm, D. Mastrandrea, K. Moeller,
B. Pasca, B. Popa, X. Pujol, G. Sergent, D. Thomas,
R. Tudoran, A. Vasquez.

and the authors of NVC, Sollya, FPLLL, ScaLP, ...

http://flopoco.gforge.inria.fr/