FloPoCo
an arithmetic core generator for FPGAs

http://www.ens-lyon.fr/LIP/Arenaire/Ware/FloPoCo/

Florent de Dinechin, Bogdan Pasca, *projet Arénaire*

FPGAs as floating-point accelerators?

IEEE-754-compatible $+, -, \times, \div, \sqrt{}$

⊕ Massive parallelism

⊖ Each operator 10X slower than a processor's

⟶ no match to GPGPU, ClearSpeed, ...

Use FPGA flexibility!

Flexibility in precision

- There is life between single and double precision
- Never compute more accurately than needed
  - mix-and-match fixed-point and floating-point [2]
  - FP interfaces, internal fixed-point computations [4]
  - Automated error analysis [5, 7, 8, 6, 4]

Flexibility in algorithms

- Operators may be specialized to a context
  - Multipliers by a constant [1]
  - Squarers [3]
- Specific hardware may be designed for coarser operators
  - elementary functions (10× the throughput of a Pentium) [6, 7, 8]
  - application-specific accumulator [4] – see below

Floating-Point Cores, but not only...

Not a library, a generator

- An infinite virtual library
- Greater parametrization and flexibility
  - Optimize for different hardware targets (timings, LUT and DSP structure)
- Better design-space exploration
  - Example: evaluation of arbitrary functions using HOTBM [5]
- Generate simple and tidy VHDL code
- Written in C++, using GMP, MPFR and Sollya
- Freely available under the LGPL

Goal: Not your neighbour's FPU

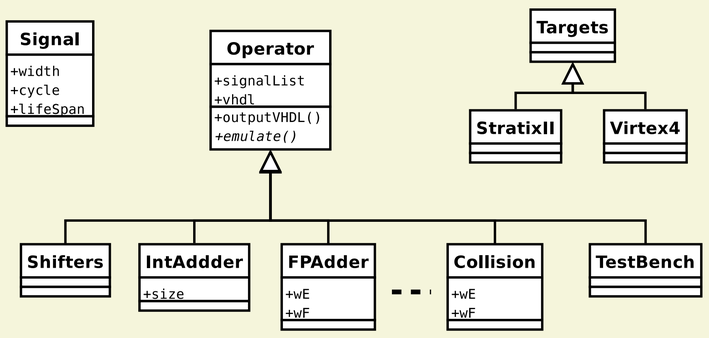
- Basic operations, with a bit of pepper
- Elementary functions (sine, exponential, logarithm...)
- Algebraic functions ($1/x$, $\sqrt{x}$, polynomials, ...)
- Compound functions ($\log_2(1 \pm 2^x)$, e^{-Kt^2}, ...)
- Sums, dot products, sums of squares, norms...
- Interval arithmetic
- ...

Current FloPoCo operators

- Integer Adder / Multiplier / Squarer [3]
- FP Adder / Multiplier / Divider
- Integer / FP constant multiplier [1]
- Long accumulator / LongAcc2FP
- FP Exponential (not pipelined yet) and FP Logarithm [8]
- fixed-point functions by HOTBM (not pipelined yet) [5]
- Automatic test-bench generation for all these operators

FloPoCo - the framework

Class hierarchy overview



Automatic pipeline management

- Designers think in terms of pipeline levels
- Attributes for: *cycle* of a signal, *currentCycle, lifeSpan*
- Automatic register insertions – Ask for a demo!

```
(...)
// at some cycle
vhdl << declare("finalFraction", wF+g) << " <= " ...
(...)
// at some other cycle
vhdl << declare("finalExp", wE+1)  << " <= " ...
(...)
// enter next cycle
nextCycle();
vhdl << declare("finalSoP", wE+wF+g)  << " <= "
 << use("finalExp") << range(wE-1,0)
 << " & " << use("finalFraction") << "; ";
```

Targets

- Abstract target FPGA features through methods:
  - Architecture related: `lutInputs()`
  - Delay related: `suggestAdderSize(double delay)`
- Single-code for target-optimized operators

TestBench Generation

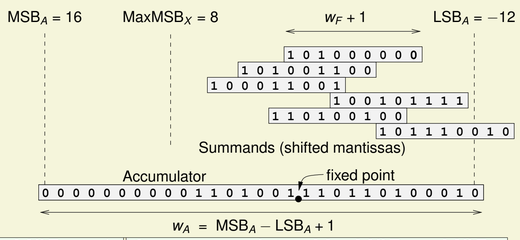
- Exploit mathematical nature of arithmetic operators
  - FP operator output = Rounding ∘ f(inputs)
  - Simpler and less error prone than mimicking the architecture
- Generic or operator-specific test-case generation
  - double-precision exp in (−1024,1023) only
  - addition of FP numbers with close exponents

Example 1 - Accumulation of FP numbers

An application-tailored fixed-point accumulator for floating-point inputs

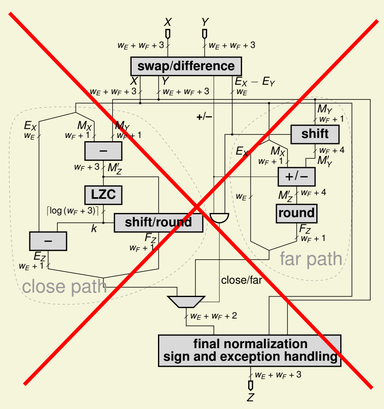
- designed to never overflow
- designed as accurate as the application requires

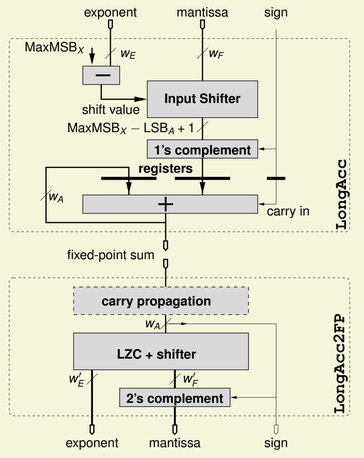
Accumulator significand never needs to be shifted



summand (w_E, w_F)	CoreGen FP adder (w_E, w_F)	$2w_F$ accumulator, MaxMSB$_X$ = 1	$2w_F$ accumulator, MaxMSB$_X$ = MSB$_A$
(7,16)	304 slices + 1 DSP, 12 cycles @ 359 MHz	129 slices, 8 cycles @ 472 MHz	176 slices, 9 cycles @ 484 MHz
(8,23) SP	317 slices + 4 DSP, 16 cycles @ 450 MHz	165 slices, 8 cycles @ 434 MHz	229 slices, 9 cycles @ 434 MHz
(10,37)	631 slices + 1 DSP, 14 cycles @ 457 MHz	295 slices, 10 cycles @ 428 MHz	399 slices, 11 cycles @ 428 MHz
(11,52) DP	771 slices + 3 DSP, 15 cycles @ 366 MHz	375 slices, 11 cycles @ 414 MHz	516 slices, 12 cycles @ 416 MHz

- **faster and arbitrarily more accurate than a naive floating-point approach**
- **cost: designer-provided bounds on the accumulated values**





Example 2 - Collision detection predicate

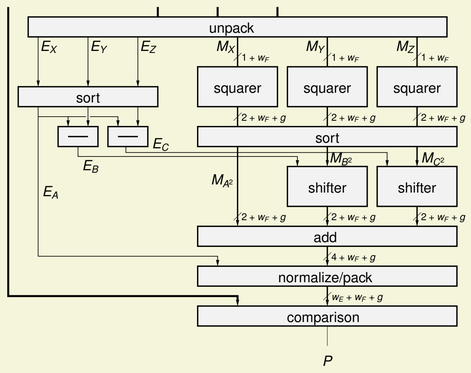
Need to compute: $x^2 + y^2 + z^2 < r_2$

Naive approach

- 3 FP multipliers + 2 FP adders + comparison

Specific operator

- Squarers instead of multipliers
- Addition of positive numbers is simpler
- Mantissa alignment in parallel
- Less rounding logic



	Slow version (`freq=200`)		Fast version (`freq=400`)	
Precision	area	perf	area	perf
(8,23) FP	940 sl, 12 DSP	20 cycles @ 210 MHz	1188 sl, 12 DSP	29 cycles @ 289 MHz
(8,23) custom	456 sl, 9 DSP	10 cycles @ 319 MHz	453 sl, 9 DSP	11 cycles @ 368 MHz
(9, 32) FP	1268 sl, 12 DSP	20 cycles @ 171 MHz	1874 sl, 12 DSP	37 cycles @ 302 MHz
(9, 32) custom	629 sl, 9 DSP	10 cycles @ 368 MHz	640 sl, 9 DSP	13 cycles @ 368 MHz
(11, 52) FP	2868 sl, 27 DSP	20 cycles @ 106 MHz	4480 sl, 27 DSP	46 cycles @ 276 MHz
(11, 52) custom	1532 sl, 18 DSP	10 cycles @ 237 MHz	1845 sl, 18 DSP	16 cycles @ 362 MHz

Custom slow version: 48% smaller, 29% less DSPs, 50% lower latency, 89% faster
Custom fast version: 61% smaller, 29% less DSPs, 64% lower latency, 27% faster
Custom version is also more accurate

References

[1] N. Brisebarre, F. de Dinechin, and J.-M. Muller. Integer and floating-point constant multipliers for FPGAs. In *Application-specific Systems, Architectures and Processors*, pages 239–244. IEEE, 2008.

[2] F. de Dinechin, J. Detrey, I. Trestian, O. Creţ, and R. Tudoran. When FPGAs are better at floating-point than microprocessors. Technical Report ensl-00174627, École Normale Supérieure de Lyon, 2007.

[3] F. de Dinechin and B. Pasca. Large multipliers with less DSP blocks. Technical Report 2009-03, LIP, École Normale Supérieure de Lyon, 2009.

[4] F. de Dinechin, B. Pasca, O. Creţ, and R. Tudoran. An FPGA-specific approach to floating-point accumulation and sum-of-products. In *Field-Programmable Technologies*, pages 33–40. IEEE, 2008.

[5] J. Detrey and F. de Dinechin. Table-based polynomials for fast hardware function evaluation. In *Application-specific Systems, Architectures and Processors*, pages 328–333. IEEE, 2005.

[6] J. Detrey and F. de Dinechin. Floating-point trigonometric functions for FPGAs. In *Field-Programmable Logic and Applications*, pages 29–34. IEEE, 2007.

[7] J. Detrey and F. de Dinechin. Parameterized floating-point logarithm and exponential functions for FPGAs. *Microprocessors and Microsystems, Special Issue on FPGA-based Reconfigurable Computing*, 31(8):537–545, 2007.

[8] J. Detrey, F. de Dinechin, and X. Pujol. Return of the hardware floating-point elementary function. In *18th Symposium on Computer Arithmetic*, pages 161–168. IEEE, 2007.

This work was partly supported by the Egide Brancusi programme of the French government, and the XtremeData university programme.