A general-purpose method for faithfully rounded floating-point function approximation in FPGAs

David B. Thomas
Imperial College London
FloPoCo: Parameterised primitives
FloPoCo : Parameterised primitives

```
Not pipelined
---Entity IntAdder_34_f400_uid24
   Pipeline depth = 1
---Entity FixRealKCM_0_7_M26_log_2_unsigned
   Pipeline depth = 2
---Entity IntAdder_26_f484_uid32
   Pipeline depth = 1
---Entity MagicSPExpTable
   Not pipelined
---Entity IntAdder_18_f400_uid41
   Pipeline depth = 1
---Entity IntAdder_18_f400_uid48
   Pipeline depth = 1
---Entity IntMultiplier_UsingDSP_17_18_19_unsigned_uid55
   Pipeline depth = 1
---Entity IntAdder_27_f400_uid60
   Not pipelined
---Entity IntAdder_33_f400_uid67
   Pipeline depth = 1
Entity FPExp_8_23_400
   Pipeline depth = 13
Output file: flopoco.vhdl
vagrant@vagrant-ubuntu-trusty-64:~$
```
FloPoCo : Parameterised primitives

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
library std;
use std.textio.all;
library work;

entity FPExp_8_23_400 is
  port ( clk, rst : in std_logic;
         X : in std_logic_vector(8+23+2 downto 0);
         R : out std_logic_vector(8+23+2 downto 0) );
end entity;

architecture arch of FPExp_8_23_400 is
  component FixRealKCM_0_7_M26_log_2_unsigned is
    port ( clk, rst : in std_logic;
           X : in std_logic_vector(7 downto 0);
           R : out std_logic_vector(33 downto 0) );
  end component;

  component FixRealKCM_M3_6_0_1_log_2_unsigned is
```
FloatApprox : Parameterised anything

- **Input format**
- **Approximation interval**
- **Output format**

```
/vcygdrive/e/_dt10_/VMs/dev
vagrant@vagrant-ubuntu-trusty-64:$ ./flopoco FloatApprox
5 10
0 16
5 10
"sin(x^0.95+0.1)/x+0.06*x"
3
```
FloatApprox : Parameterised anything
FloatApprox

• Architecture for FPGA function approximation
  – Deeply pipelined
  – Floating-point in and out
  – Faithfully rounded

• Method and tool for approximating functions
  – Handles any most twice-differentiable functions
  – Completely automated: expression to VHDL
  – Designed for reliability rather than optimality

David Thomas, Imperial College, dt10@ic.ac.uk
1. Motivation
2. The FloatApprox approach
   1. Range reduction and approximation method
   2. Evaluation architecture
3. Evaluation in hardware
Context: FPGA accelerators

• Mathematical or algorithmic specification
• Convert to HLS or VHDL implementation
  – Rely on optimised IP for floating-point
  – Integrated at link-time into the final design
Context: FPGA accelerators

• Mathematical or algorithmic specification
• Convert to HLS or VHDL implementation
  – Rely on optimised IP for floating-point
  – Integrated at link-time into the final design
• Intellectual challenges for accelerator design
  – Managing memory accesses and bandwidth
  – Rewriting to tolerate latency of operators
  – Keeping pipelines occupied
  – Not: designing low-level IP cores
Floating-point IP: Requirements

• Faithfully rounded
  – Make every bit count
  – Tractable error analysis

• Pipelined for 150MHz+ clock rate
  – Must be pipelined: RAM and DSPs are multi-cycle
  – Synthesis tools have limited retiming capability

• Working RTL (circuit) implementation
  – A paper can’t be synthesised
Floating-point IP: Requirements

• Faithfully rounded
  – Make every bit count
  – Tractable error analysis

• Pipelined for 150MHz+ clock rate
  – Must be pipelined: RAM and DSPs are multi-cycle
  – Synthesis tools have limited retiming capability

• Working RTL (circuit) implementation
  – A paper can’t be synthesised
Subject: Floating-point log1p?
To:    dt10@ic.ac.uk
From:   phd-slash-industry-bod@somewhere.com
Body:

I’m converting some code for an accelerator, and it uses log1p. Can I use your core from that PoC you did a while back?
A fable...

Subject: Re: Floating-point log1p?
To:      phd-slash-industry-bod@somewhere.com
From:    dt10@ic.ac.uk
Body:

Afraid that was written in Handel-C, I don’t have any VHDL. You could recreate it using the attached maple script, plus write a code gen.

> I’m converting some code for an accelerator, and
> it uses log1p. Can I use your core from that
> PoC you did a while back?
A fable...

Subject: Re: Floating-point log1p?
To: phd-slash-industry-bod@somewhere.com
From: dt10@ic.ac.uk
Body:

Any luck?

> Afraid that was written in Handel-C, I don’t
> have any VHDL. You could recreate it using
> the attached maple script, plus write a code gen.

>> I’m converting some code for an accelerator, and
>> it uses log1p. Can I use your core from that
>> PoC you did a while back?
... becomes a nightmare

Subject: Re: Floating-point log1p?
To: phd-slash-industry-bod@somewhere.com
From: dt10@ic.ic.ac.uk
Body:

Oh, we don’t have maple.
It’s ok, I found out log1p(x)=log(1+x), and just did that. Works fine.

ck?

I did that was written in Handel-C, I don’t any VHDL. You could recreate it using attached maple script, plus write a code gen.
## What IP is available?

<table>
<thead>
<tr>
<th>Function</th>
<th>Source</th>
<th>Pipelined</th>
<th>Faithful</th>
<th>RTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>add, mul, div</td>
<td>FloPoCo</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>log, exp</td>
<td>FloPoCo</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
## What IP is available?

<table>
<thead>
<tr>
<th>Operation</th>
<th>Source</th>
<th>Pipelined</th>
<th>Faithful</th>
<th>RTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>add, mul, div</td>
<td>FloPoCo</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>log, exp</td>
<td>FloPoCo</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>sin, cos</td>
<td>FPLibrary</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Altera</td>
<td>Yes</td>
<td>Yes</td>
<td>Altera flow only</td>
</tr>
<tr>
<td></td>
<td>Xilinx</td>
<td>Yes</td>
<td>?</td>
<td>Vivado HLS only</td>
</tr>
</tbody>
</table>
## What IP is available?

<table>
<thead>
<tr>
<th>Function</th>
<th>Source</th>
<th>Pipelined</th>
<th>Faithful</th>
<th>RTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>add, mul, div</td>
<td>FloPoCo</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>log, exp</td>
<td>FloPoCo</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>sin, cos</td>
<td>FPILibrary</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Altera</td>
<td>Yes</td>
<td>Yes</td>
<td>Altera flow only</td>
</tr>
<tr>
<td></td>
<td>Xilinx</td>
<td>Yes</td>
<td>?</td>
<td>Vivado HLS only</td>
</tr>
<tr>
<td>log1p</td>
<td>Altera</td>
<td>Yes</td>
<td>Yes</td>
<td>Altera flow only</td>
</tr>
<tr>
<td>expm1</td>
<td>Altera</td>
<td>Yes</td>
<td>No</td>
<td>OpenCL only</td>
</tr>
<tr>
<td>erf</td>
<td>Altera</td>
<td>Yes</td>
<td>No</td>
<td>OpenCL only</td>
</tr>
</tbody>
</table>
Motivation for FloatApprox

• We currently have: +, -, *, /, log, exp
  – Use existing IP: FloPoCo, Xilinx, Altera, ...
Motivation for FloatApprox

• We currently have: +, -, *, /, log, exp
  – Use existing IP: FloPoCo, Xilinx, Altera, ...

• We should have: log1p, expm1, erf, sin, acos, ...
  – What FloatApprox does badly...
    … but better than anything else available
Motivation for FloatApprox

• We currently have:  +, -, *, /, log, exp
  – Use existing IP: FloPoCo, Xilinx, Altera, ...

• We should have: log1p, expm1, erf, sin, acos, ...
  – What FloatApprox does badly...
    ... but better than anything else available

• What I want:  \( \sqrt{-2 \log(x)} \),  \( \frac{1}{1 + \exp(-x)} \)
  – What FloatApprox does well
Goals of FloatApprox

• As a tool
  – Convert any function \( f(x) \) to RTL
  – Able to handle most smooth functions
    • \( \text{Smooth} = \) twice differentiable for our purposes
  – Suitable for automated use
    • Input: data-types, range, function
    • Output: faithfully rounded circuit
Goals of FloatApprox

• As a tool
  – Convert any function f(x) to RTL
  – Able to handle most smooth functions
    • Smooth = twice differentiable for our purposes
  – Suitable for automated use
    • Input : data-types, range, function
    • Output : faithfully rounded circuit

• As generated IP
  – Pipelined
  – Faithfully rounded
  – Working RTL
FloatApprox: requirements

- User can specify any specified target function
- Parameterised floating-point representation
  - Input and output formats can be distinct
- Portable between platforms
- Usable from many languages
- Open-source

- Low latency
- Minimal resource
Architecture and Approximation

• Architecture:
  – General template for creating any approximator

• Approximation
  – Configuring the template for a given function
FloatApprox : Approximation

• Given a function $f_t$ how do we create $f_a$?
FloatApprox : Approximation

• Given a function $f_t$ how do we create $f_a$?

• Segment the function so that segments are:
FloatApprox : Approximation

• Given a function $f_t$ how do we create $f_a$?

• Segment the function so that segments are:
  1. Contained in one input binade
FloatApprox : Approximation

• Given a function $f_t$ how do we create $f_a$?

• Segment the function so that segments are:
  1. Contained in one input binade

  1. Contained in one output binade
FloatApprox : Approximation

• Given a function $f_t$ how do we create $f_a$?

• Segment the function so that segments are:
  1. Contained in one input binade

  1. Contained in one output binade

  1. FaithfulFixed: can faithfully approximate with fixed-point polynomial of degree $d$
FloatApprox: Approximation

• Given a function $f_t$ how do we create $f_a$?

• Segment the function so that segments are:
  1. Contained in one input binade
  2. Monotonically increasing or decreasing in range
  3. Contained in one output binade
  4. FaithfulReal: can approx. with real degree $d$ poly
  5. FaithfulFixed: can faithfully approximate with fixed-point polynomial of degree $d$
Example: Input function over reals

\[ y = \frac{\sin( x^{0.95} + 0.1)}{x} + 0.06 \ x \]

\[ 0 \leq x < 16 \]
Move to float representation
1 : Segment using input binades
2 : Make segments monotonic
3 : Segment using output binades
3: Segment using output binades
3 : Segment using output binades
3 : Segment using output binades
4 : Split to degree $d$ polynomials
4 : Split to degree $d$ polynomials
4 : Split to degree $d$ polynomials
• Segments form a partition on input domain
• Segment domains and ranges cover one binade
• All segments can be faithfully calculated as degree $d$ fixed-point polynomial
Real-world issues

- Lots of corner cases to worry about
  - Crossing from negative to positive to NaN is fun
  - Method *should* be faithful by construction
Real-world issues

• Lots of corner cases to worry about
  – Crossing from negative to positive to NaN is fun
  – Method *should* be faithful by construction

• Calculations performed using *mpfr* and *sollya*
  – Mostly interval arithmetic via sollya
  – Occasionally bisection search in mpfr
Real-world issues

• Lots of corner cases to worry about
  – Crossing from negative to positive to NaN is fun
  – Method *should* be faithful by construction

• Calculations performed using mpfr and sollya
  – Mostly interval arithmetic via sollya
  – Occasionally bisection search in mpfr

• Speed of approximation is an issue
  – Single precision takes minutes
  – Double precision takes hours
FloatApprox : Architecture

\[ y = \sum_{i=0}^{d} c_i x^i \]

Segmentation
\[ \lfloor S_i \rfloor \leq i \leq \lceil S_i \rceil \]

Table-Lookup

Fixed-Point Polynomial
Compile-time configuration

\textbf{Segmentation Fixed-Point Polynomial Table-Lookup}

\begin{align*}
\begin{array}{|c|c|c|c|}
\hline
\text{flags} & s & \text{expnt} & \text{significand} \\
\hline
\end{array}
\end{align*}

\begin{align*}
\text{Segmentation } \lfloor S_i \rfloor \leq i \leq \lceil S_i \rceil \\
\text{Table-Lookup}
\end{align*}

\begin{align*}
\begin{array}{|c|c|c|c|}
\hline
\text{flags} & s & \text{expnt} & c_0 \ c_1 \ldots \ c_d \\
\hline
\end{array}
\end{align*}

\begin{align*}
\text{Fixed-Point Polynomial } y = \sum c_i x^i
\end{align*}

\begin{align*}
\begin{array}{|c|c|c|c|}
\hline
\text{flags} & s & \text{expnt} & \text{significand} \\
\hline
\end{array}
\end{align*}

David Thomas, Imperial College, dt10@ic.ac.uk
Evaluation: Input

Segmentation
Fixed-Point Polynomial
Table-Lookup

\( y = \sum c_i x^i \)
Evaluation: Segmentation

Segmentation

Fixed-Point Polynomial

Table-Lookup

\[
y = \sum c_i x^i
\]

flags s expnt significand

flags s expnt c_0 c_1 ... c_d

flags s expnt significand

David Thomas, Imperial College, dt10@ic.ac.uk
Evaluation: Segmentation

Segmentation

Fixed-Point Polynomial
Table-Lookup

$s, e, s, d$

$y = \sum c_i x^i$

Table

$s, e, s, d$

$\frac{1}{2^{254}}$ $\frac{1}{2^{253}}$ $\frac{1}{2^{252}}$ $\frac{1}{2^{251}}$ $\frac{1}{2^{250}}$ $\frac{1}{2^{249}}$ $\frac{1}{2^{248}}$

David Thomas, Imperial College, dt10@ic.ac.uk
Evaluation: Segmentation

Segmentation

Fixed-Point Polynomial

Table-Lookup

\[ y = \sum_{i} c_i x^i \]

flags | s | expnt | significand

flags | s | expnt | c_0 | c_1 | ... | c_d

flags | s | expnt | significand

David Thomas, Imperial College, dt10@ic.ac.uk
Evaluation: Segmentation

Segmentation

Fixed-Point Polynomial

Table-Lookup

```
x
Segmentation
\lfloor S_i \rfloor \leq i \leq \lceil S_i \rceil
```

```
Table-Lookup
```

```
Fixed-Point Polynomial
\quad y = \sum c_i x^i
```

David Thomas, Imperial College, dt10@ic.ac.uk
Evaluation: Segmentation

Segmentation

Fixed-Point Polynomial

Table-Lookup

\[
y = \sum c_i x^i
\]
Evaluation: Segmentation

<table>
<thead>
<tr>
<th>flags</th>
<th>s</th>
<th>expnt</th>
<th>significand</th>
</tr>
</thead>
</table>

Segmentation

Table-Lookup

Fixed-Point Polynomial

\[ y = \sum c_i x^i \]

flags | s | expnt  | c_0 | c_1 | ... | c_d |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ x = \prod 2^{-k} \leq x \leq \prod 2^{-k+1} \]

flags | s | expnt | significand |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

David Thomas, Imperial College, dt10@ic.ac.uk
Evaluation: Table Lookup

Segmentation

Fixed-Point Polynomial

Table-Lookup

\[
y = \sum c_i x^i
\]

flags | s | expnt | significand
---|---|---|---
| | | | |

2^0 2^-1 2^-2 2^-3 2^-4

2^-4 2^-3 2^-2 2^-1 2^0 2^1 2^2 2^3

flags | s | expnt | c_0 | c_1 | ... | c_d
---|---|---|---|---|---|---
| | | | | | | |

1.0 2.0 1.0 1.5 1.5

David Thomas, Imperial College, dt10@ic.ac.uk
Evaluation: Fraction

\[
y = \sum_{i=0}^{d} c_i x^i
\]

flags | s | expnt | significand
---|---|---|---

Segmentation 
\[\lfloor S_i \rfloor \leq i \leq \lceil S_i \rceil\]

Table-Lookup

c0 cd ...

Fixed-Point Polynomial

y = \sum_{i=0}^{d} c_i x^i
Evaluation: Flags and Exponent

\[ y = \sum c_i x^i \]

- Segmentation: \( \lfloor S_i \rfloor \leq i \leq \lceil S_i \rceil \)
- Table-Lookup
- Fixed-Point Polynomial

- Flags
- Significand
- Expnt
- C0 C1 ... Cd

David Thomas, Imperial College, dt10@ic.ac.uk
FloatApprox : Architecture

Segmentation
Fixed-Point Polynomial
Table-Lookup

flags s expnt significand

<table>
<thead>
<tr>
<th>flags</th>
<th>s</th>
<th>expnt</th>
<th>significand</th>
</tr>
</thead>
</table>

\[
y = \sum_{i} c_i x^i
\]
Architectural pros and cons

• Key strengths of the architecture:
  – **Simplicity**: building and verifying is very simple
  – **Generality**: it can handle any function
  – **Speed**: very easy to make it fast
Architectural pros and cons

• Key strengths of the architecture:
  – **Simplicity**: building and verifying is very simple
  – **Generality**: it can handle any function
  – **Speed**: very easy to make it fast

• Weaknesses of the architecture:
  – **Table size**: exponential in exponent width
  – **Table blow-up**: periodic functions are impractical
Evaluation: Test Method

• Three classes of function
  – Primitives: faithfully rounded IP available
  – Composite: can express in terms of available IP
  – Approximate: no direct method for evaluation
Evaluation: Test Method

• Three classes of function
  – Primitives: faithfully rounded IP available
  – Composite: can express in terms of available IP
  – Approximate: no direct method for evaluation

• Source of reference IP is FloPoCo
  – OpenSource; good performance; portable
Evaluation: Test Method

• Three classes of function
  – Primitives: faithfully rounded IP available
  – Composite: can express in terms of available IP
  – Approximate: no direct method for evaluation

• Source of reference IP is FloPoCo
  – OpenSource; good performance; portable

• Approximations are “found on the internet”
  – i.e. Abramowitz and Stegun
Evaluation: Test Method

• Three classes of function
  – Primitives: faithfully rounded IP available
  – Composite: can express in terms of available IP
  – Approximate: no direct method for evaluation
• Source of reference IP is FloPoCo
  – OpenSource; good performance; portable
• Approximations are “found on the internet”
  – i.e. Abramowitz and Stegun
• All results are post place-and-route in Virtex-6
<table>
<thead>
<tr>
<th>Name</th>
<th>Method</th>
<th>Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primitive</td>
<td></td>
<td></td>
</tr>
<tr>
<td>log</td>
<td>IP core</td>
<td>[0, ∞]</td>
</tr>
<tr>
<td>exp</td>
<td>IP core</td>
<td>(-∞, ∞)</td>
</tr>
<tr>
<td>Composite</td>
<td>normpdf</td>
<td>[-16, 16]</td>
</tr>
<tr>
<td></td>
<td>exp(-x^2)/sqrt(2pi)</td>
<td></td>
</tr>
<tr>
<td>sigmoid</td>
<td>1/(1+exp(-x))</td>
<td>(-∞, +∞)</td>
</tr>
<tr>
<td>log1p</td>
<td>log(x+1)</td>
<td>[-1, +∞]</td>
</tr>
<tr>
<td>expm1</td>
<td>exp(x)-1</td>
<td>(-∞, +∞)</td>
</tr>
<tr>
<td>Approximate</td>
<td>sin</td>
<td>[-π, +π]</td>
</tr>
<tr>
<td></td>
<td>mul:7, add:5</td>
<td></td>
</tr>
<tr>
<td>cos</td>
<td>mul:6, add:5</td>
<td>[-π, +π]</td>
</tr>
<tr>
<td>erf</td>
<td>mul:7, add:7, inv:1, exp:1</td>
<td>[-32, +32]</td>
</tr>
<tr>
<td>Name</td>
<td>LUTs</td>
<td>BRAM</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>log</td>
<td>1376 (1.7x)</td>
<td>12 (12x)</td>
</tr>
<tr>
<td>exp</td>
<td>1372 (3.4x)</td>
<td>24 (24x)</td>
</tr>
<tr>
<td>normpdf</td>
<td>1498 (1.8x)</td>
<td>25 (25x)</td>
</tr>
<tr>
<td>sigmoid</td>
<td>1259 (0.8x)</td>
<td>5 (5x)</td>
</tr>
<tr>
<td>log1p</td>
<td>2203 (1.9x)</td>
<td>10 (10x)</td>
</tr>
<tr>
<td>expm1</td>
<td>1304 (1.8x)</td>
<td>12 (12x)</td>
</tr>
<tr>
<td>sin</td>
<td>1366 (0.5x)</td>
<td>5 (∞x)</td>
</tr>
<tr>
<td>cos</td>
<td>1220 (0.5x)</td>
<td>5 (∞x)</td>
</tr>
<tr>
<td>erf</td>
<td>881 (0.2x)</td>
<td>4 (4x)</td>
</tr>
<tr>
<td>Name</td>
<td>LUTs</td>
<td>BRAM</td>
</tr>
<tr>
<td>--------</td>
<td>------------</td>
<td>--------</td>
</tr>
<tr>
<td>log</td>
<td>1376 (1.7x)</td>
<td>12 (12x)</td>
</tr>
<tr>
<td>exp</td>
<td>1372 (3.4x)</td>
<td>24 (24x)</td>
</tr>
<tr>
<td>normpdf</td>
<td>1498 (1.8x)</td>
<td>25 (25x)</td>
</tr>
<tr>
<td>sigmoid</td>
<td>1259 (0.8x)</td>
<td>5 (5x)</td>
</tr>
<tr>
<td>log1p</td>
<td>2203 (1.9x)</td>
<td>10 (10x)</td>
</tr>
<tr>
<td>expm1</td>
<td>1304 (1.8x)</td>
<td>12 (12x)</td>
</tr>
<tr>
<td>sin</td>
<td>1366 (0.5x)</td>
<td>5 (∞x)</td>
</tr>
<tr>
<td>cos</td>
<td>1220 (0.5x)</td>
<td>5 (∞x)</td>
</tr>
<tr>
<td>erf</td>
<td>881 (0.2x)</td>
<td>4 (4x)</td>
</tr>
<tr>
<td>Name</td>
<td>LUTs</td>
<td>BRAM</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>log</td>
<td>1376 (1.7x)</td>
<td>12 (12x)</td>
</tr>
<tr>
<td>exp</td>
<td>1372 (3.4x)</td>
<td>24 (24x)</td>
</tr>
<tr>
<td>normpdf</td>
<td>1498 (1.8x)</td>
<td>25 (25x)</td>
</tr>
<tr>
<td>sigmoid</td>
<td>1259 (0.8x)</td>
<td>5 (5x)</td>
</tr>
<tr>
<td>log1p</td>
<td>2203 (1.9x)</td>
<td>10 (10x)</td>
</tr>
<tr>
<td>expm1</td>
<td>1304 (1.8x)</td>
<td>12 (12x)</td>
</tr>
<tr>
<td>sin</td>
<td>1366 (0.5x)</td>
<td>5 (∞x)</td>
</tr>
<tr>
<td>cos</td>
<td>1220 (0.5x)</td>
<td>5 (∞x)</td>
</tr>
<tr>
<td>erf</td>
<td>881 (0.2x)</td>
<td>4 (4x)</td>
</tr>
<tr>
<td>Name</td>
<td>LUTs</td>
<td>BRAM</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>log</td>
<td>2.5x</td>
<td>18x</td>
</tr>
<tr>
<td>exp</td>
<td>2x</td>
<td>10x</td>
</tr>
<tr>
<td>normpdf</td>
<td>2x</td>
<td>10x</td>
</tr>
<tr>
<td>sigmoid</td>
<td>2x</td>
<td>10x</td>
</tr>
<tr>
<td>log1p</td>
<td>2x</td>
<td>10x</td>
</tr>
<tr>
<td>expm1</td>
<td>2x</td>
<td>10x</td>
</tr>
<tr>
<td>sin</td>
<td>0.4x</td>
<td>~5x</td>
</tr>
<tr>
<td>cos</td>
<td>0.4x</td>
<td>~5x</td>
</tr>
<tr>
<td>erf</td>
<td>0.4x</td>
<td>~5x</td>
</tr>
<tr>
<td>Name</td>
<td>LUTs</td>
<td>BRAM</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>log</td>
<td></td>
<td></td>
</tr>
<tr>
<td>exp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>normpdf</td>
<td>2x</td>
<td>10x</td>
</tr>
<tr>
<td>sigmoid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>log1p</td>
<td>0.4x</td>
<td></td>
</tr>
<tr>
<td>expm1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cos</td>
<td></td>
<td></td>
</tr>
<tr>
<td>erf</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>LUTs</td>
<td>BRAM</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>log</td>
<td></td>
<td></td>
</tr>
<tr>
<td>exp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>normpdf</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sigmoid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>log1p</td>
<td></td>
<td></td>
</tr>
<tr>
<td>expm1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sin</td>
<td>0.4x</td>
<td>~5x</td>
</tr>
<tr>
<td>cos</td>
<td></td>
<td></td>
</tr>
<tr>
<td>erf</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Worse in every way

Poor resource utilisation
Much better accuracy
<table>
<thead>
<tr>
<th>Name</th>
<th>LUTs</th>
<th>BRAM</th>
<th>DSP</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>log</td>
<td></td>
<td></td>
<td></td>
<td>Worse in every way</td>
</tr>
<tr>
<td>exp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>normpdf</td>
<td></td>
<td></td>
<td></td>
<td>Poor resource utilisation</td>
</tr>
<tr>
<td>sigmoid</td>
<td></td>
<td></td>
<td></td>
<td>Much better accuracy</td>
</tr>
<tr>
<td>log1p</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>expm1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sin</td>
<td></td>
<td></td>
<td></td>
<td>More accurate and smaller</td>
</tr>
<tr>
<td>cos</td>
<td></td>
<td></td>
<td></td>
<td>except for RAMs</td>
</tr>
<tr>
<td>erf</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Current and Future Work

• Optimisation of final segments
  – Many segments contain 0 or +Inf
  – Segments could be merged, coefficients shared
Current and Future Work

• Optimisation of final segments
  – Many segments contain 0 or +\text{Inf}
  – Segments could be merged, coefficients shared

• Improve fixed-point polynomial evaluation
  – Range of intermediate polynomial stages is large
  – Can apply cheap pre-scaling to normalise
Current and Future Work

• Optimisation of final segments
  – Many segments contain 0 or +\( \text{Inf} \)
  – Segments could be merged, coefficients shared

• Improve fixed-point polynomial evaluation
  – Range of intermediate polynomial stages is large
  – Can apply cheap pre-scaling to normalise

• Export as packaged IP cores
  – Add AXI and Avalon stream interfaces
Current and Future Work

• Optimisation of final segments
  – Many segments contain 0 or +Inf
  – Segments could be merged, coefficients shared
• Improve fixed-point polynomial evaluation
  – Range of intermediate polynomial stages is large
  – Can apply cheap pre-scaling to normalise
• Export as packaged IP cores
  – Add AXI and Avalon stream interfaces
• Support generation of HLS compatible C code
Conclusion

• General method for function approximation
  – Parameterisable template architecture
  – Method for generating parameters from function
Conclusion

• General method for function approximation
  – Parameterisable template architecture
  – Method for generating parameters from function

• Faithfully rounded by construction
  – Ok method for creating primitives that don’t exist
  – Good method for creating complex function
Conclusion

• General method for function approximation
  – Parameterisable template architecture
  – Method for generating parameters from function
• Faithfully rounded by construction
  – Ok method for creating primitives that don’t exist
  – Good method for creating complex function
• Currently in FloPoCo on an obscure branch
  – Hopefully going to roll into trunk some time soon