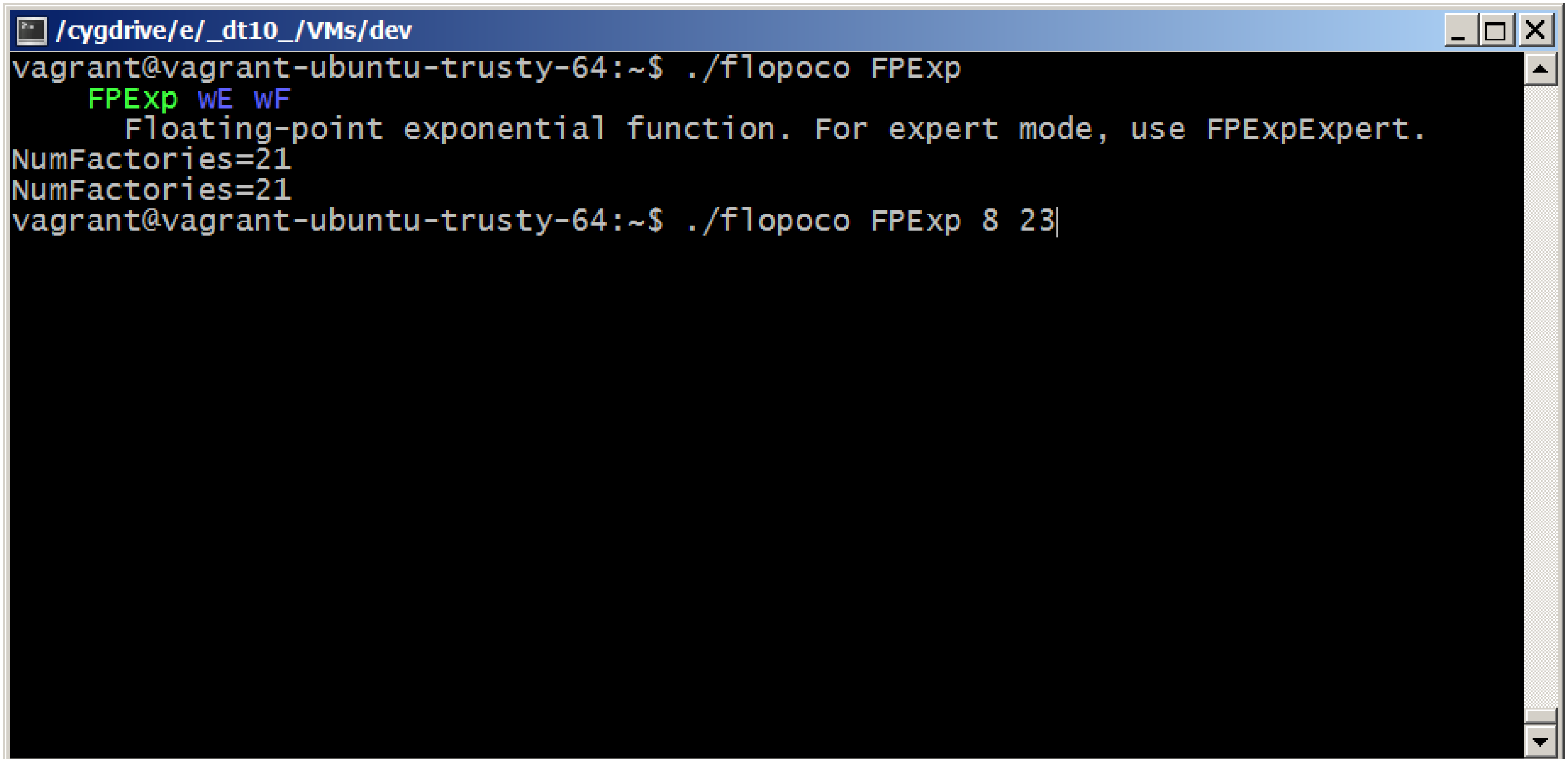
FloatApprox: faithfully rounded floating-point function approximators in FPGAs

David B. Thomas
Imperial College London

FloPoCo : Parameterised primitives



FloPoCo : Parameterised primitives

/cygdrive/e/_dt10_/VMs/dev

```
|     |       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

/cygdrive/e/_dt10_/VMs/dev

```
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

```
/cygdrive/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

/cygdrive/e/_dt10_/VMs/dev

```
Final report:
|---Entity ComparableFloatEncoded_wE5_wF10_uid4
|       Not pipelined
|    |---Entity StaticQuantiser_Table_uid8
|    |       Not pipelined
|    |---Entity StaticQuantiser_Table_uid11
|    |       Not pipelined
|    |---Entity StaticQuantiser_Table_uid14
|    |       Not pipelined
|    |---Entity StaticQuantiser_Table_uid17
|    |       Not pipelined
|    |---Entity StaticQuantiser_Table_uid20
|    |       Not pipelined
|    |---Entity StaticQuantiser_Table_uid23
|    |       Not pipelined
|---Entity StaticQuantiser_w17_uid6
|       Pipeline depth = 6
|---Entity FloatApprox_uid2_table
|       Pipeline depth = 1
|    |---Entity IntMultiplier_UsingDSP_15_11_0_signed_uid29
|    |       Pipeline depth = 1
|    |---Entity IntMultiplier_UsingDSP_22_11_0_signed_uid34
|    |       Pipeline depth = 1
|    |---Entity IntMultiplier_UsingDSP_22_11_0_signed_uid39
|    |       Pipeline depth = 1
|---Entity RoundingPolynomialEvaluator_d3_uid27
|       Pipeline depth = 16
Entity FloatApprox_uid2
   Pipeline depth = 26
Output file: flopoco.vhdl
vagrant@vagrant-ubuntu-trusty-64:~$
```

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 -> VHDL
  - Designed for reliability rather than optimality

1. Motivation
2. The FloatApprox approach
    1. Range reduction and approximation method
    2. Evaluation architecture
3. Evaluation in hardware

Floating-point IP: Requirements

- Faithfully rounded
  - Make every bit count
  - Tractable error analysis
- Pipelined for 250MHz+ clock rate
  - Must be pipelined: RAM and DSPs are multi-cycle
  - HLS tools have retiming built-in
- Working RTL (circuit) implementation
  - A paper can't be synthesised

What floating-point IP is available?

	Source	Pipelined	Faithful	RTL
add, mul, div	FloPoCo	Yes	Yes	Yes
log, exp	FloPoCo	Yes	Yes	Yes

What floating-point IP is available?

	Source	Pipelined	Faithful	RTL
add, mul, div	FloPoCo	Yes	Yes	Yes
log, exp	FloPoCo	Yes	Yes	Yes
sin, cos	FPLibrary	No	Yes	Yes
	Altera	Yes	Yes	Altera flow only
	Xilinx	Yes	?	Vivado HLS only

What floating-point IP is available?

	Source	Pipelined	Faithful	RTL
add, mul, div	FloPoCo	Yes	Yes	Yes
log, exp	FloPoCo	Yes	Yes	Yes
sin, cos	FPLibrary	No	Yes	Yes
	Altera	Yes	Yes	Altera flow only
	Xilinx	Yes	?	Vivado HLS only
log1p	Altera	Yes	Yes	Altera flow only
expm1	Altera	Yes	No	OpenCL only
erf	Altera	Yes	No	OpenCL only

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)), 1/(1+exp(-x))
  - *What FloatApprox does well*

Goals of FloatApprox

- ***Approximation***: FloatApprox as a tool
  - Convert any function f(x) to RTL
  - Able to handle most smooth functions
  - Suitable for automated use
    - Input : data-types, range, function
    - Output : faithfully rounded circuit
- ***Architecture***: FloatApprox as generated IP
  - Pipelined
  - Faithfully rounded
  - Working RTL

FloatApprox : Approximation

- Given a function f_t how do we create f_a?

- Segment the function so that each segment is:
  1. Contained in one input binade

  3. Contained in one output binade

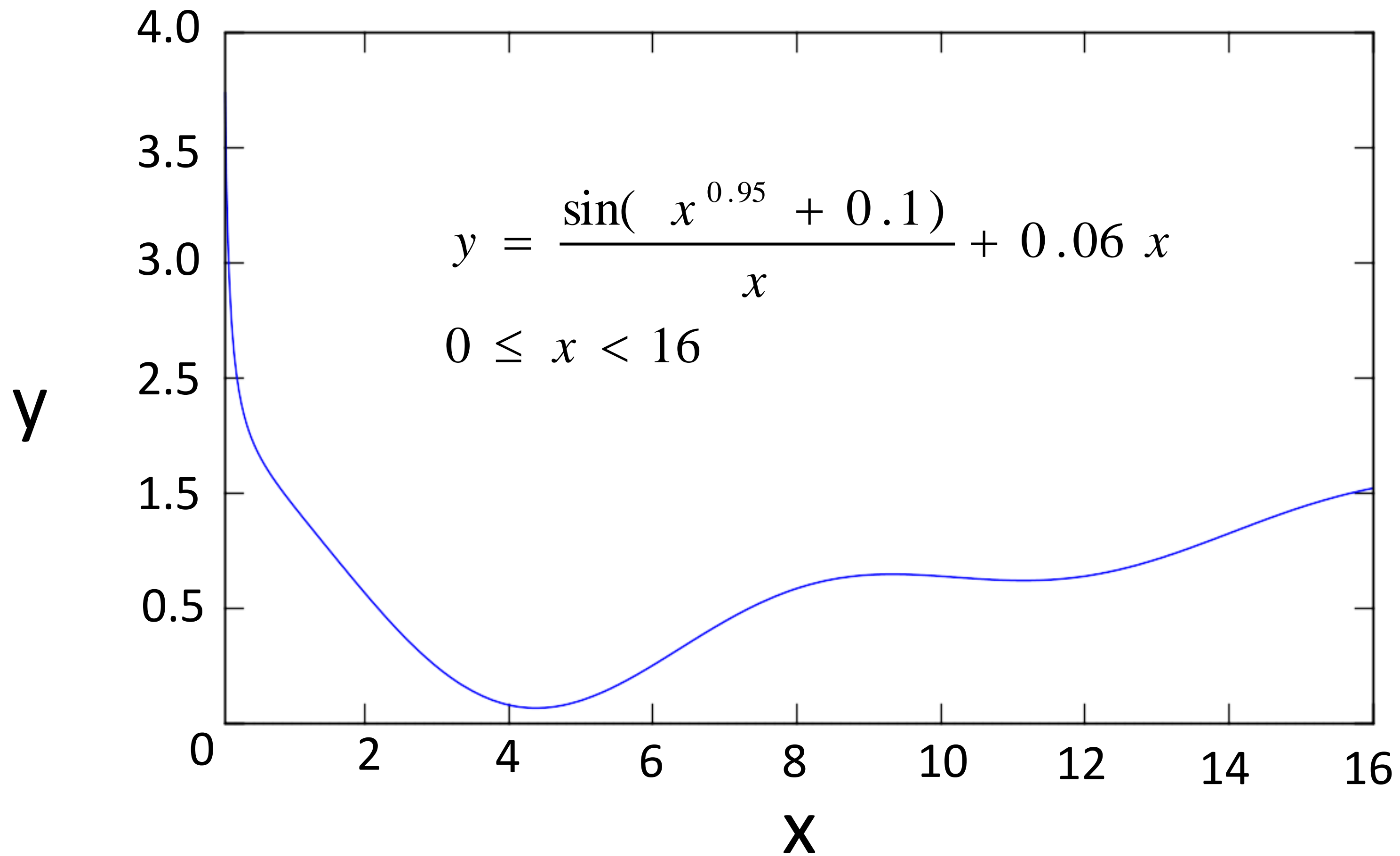
  5. FaithfulFixed: can faithfully approximate with fixed-point polynomial of degree *d*

FloatApprox : Approximation

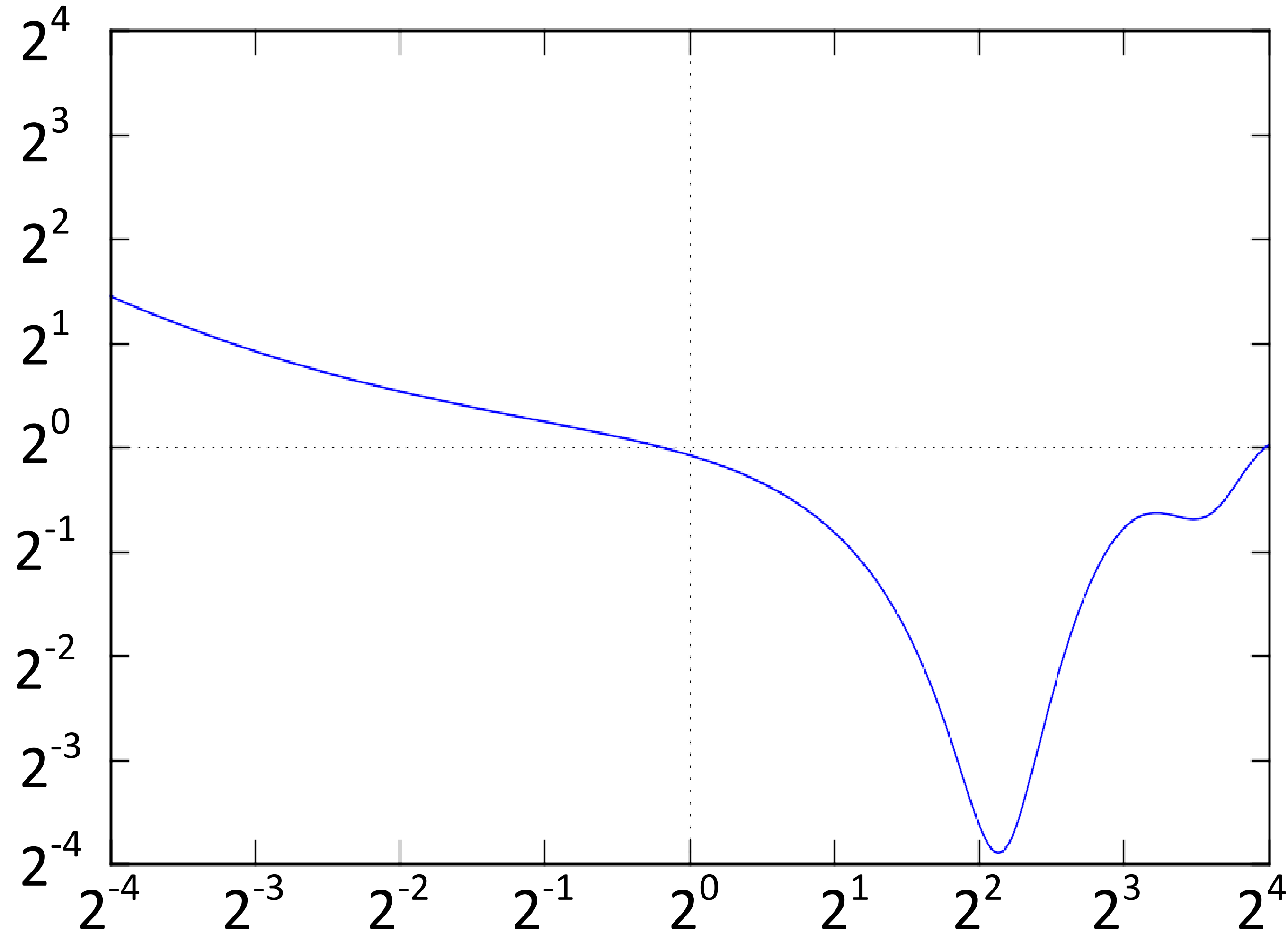
- Given a function f_t how do we create f_a?

- Segment the function so that each segment is:
  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*

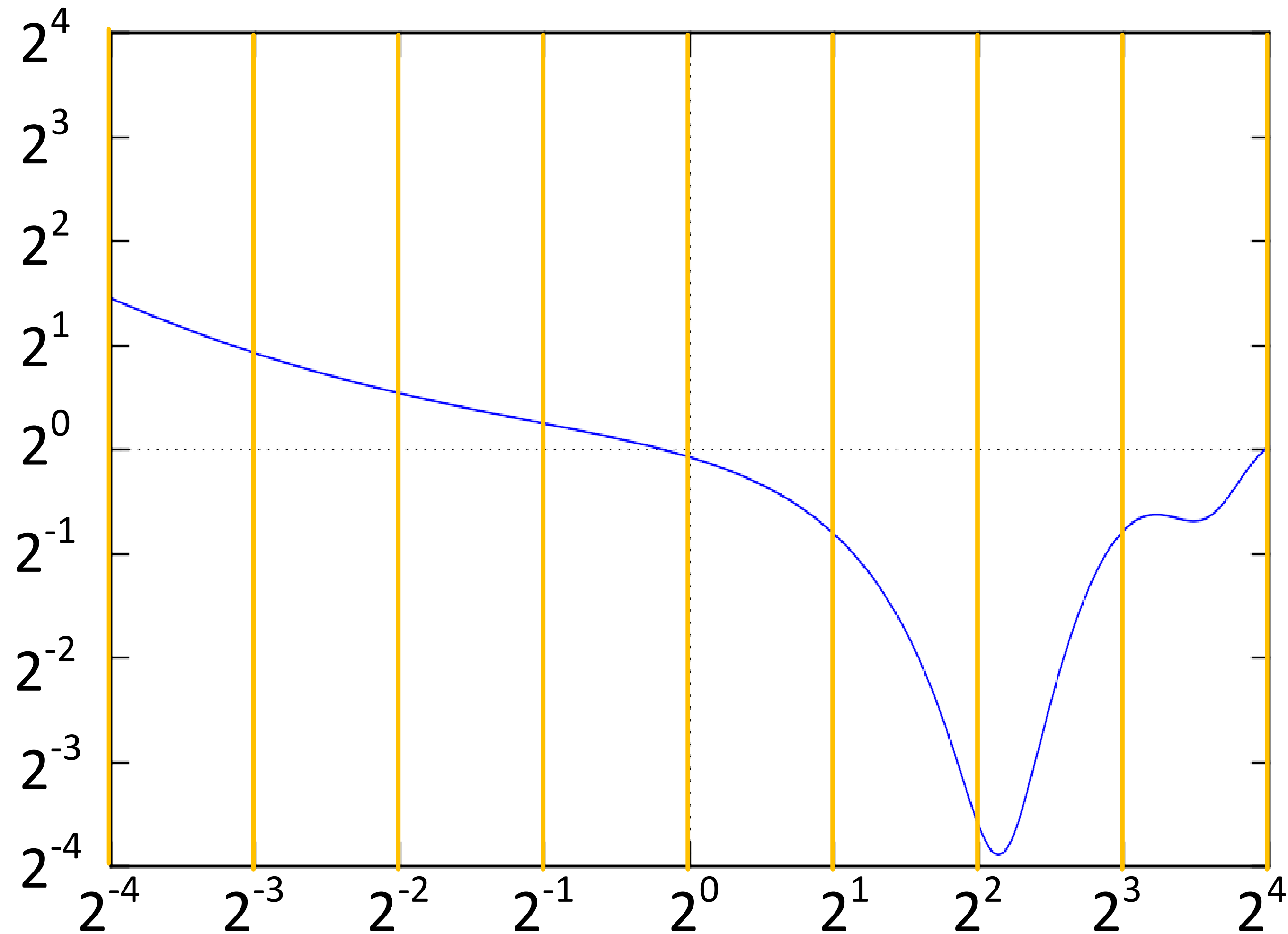
Example: Input function over reals



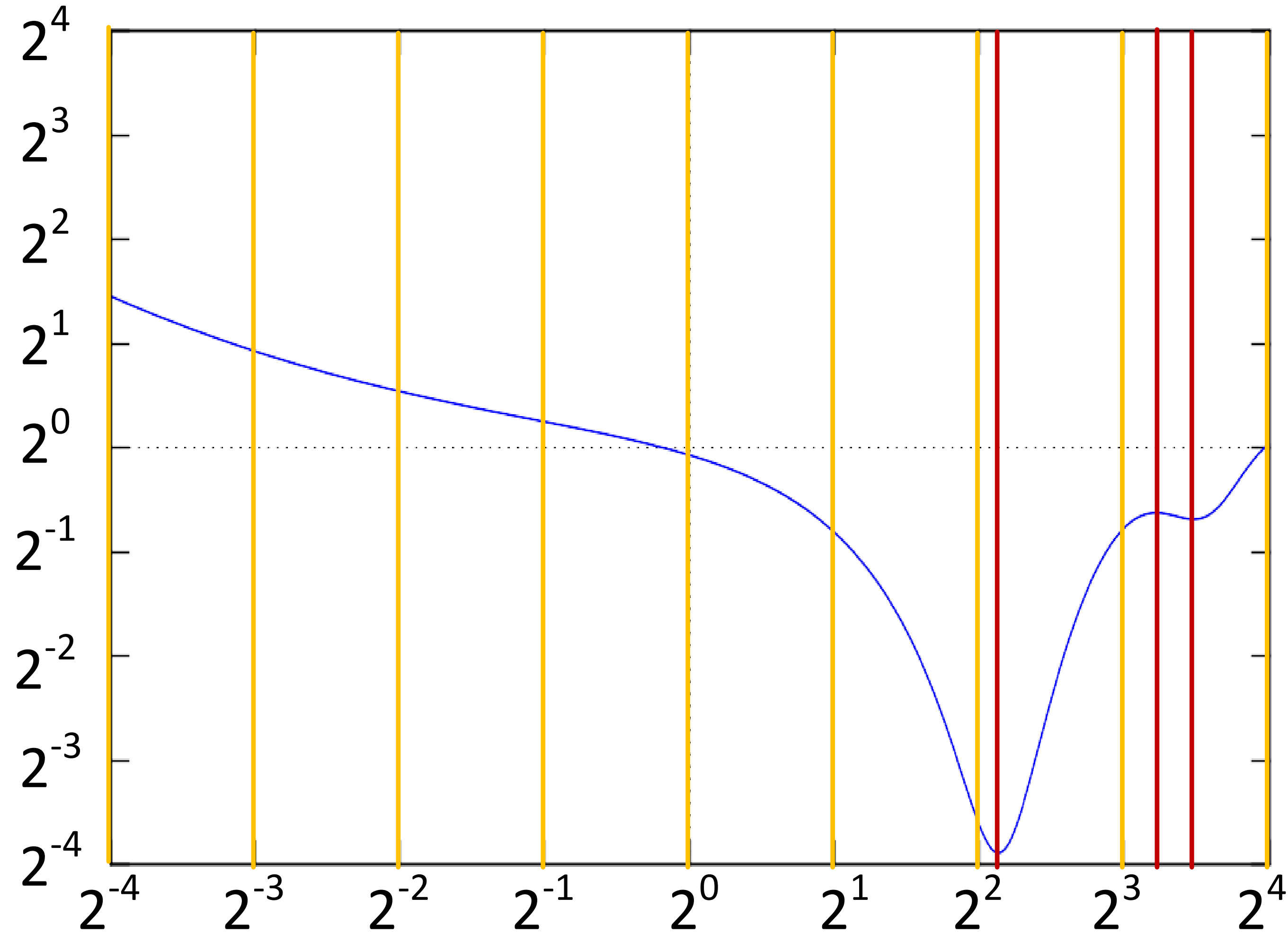
Move to float representation



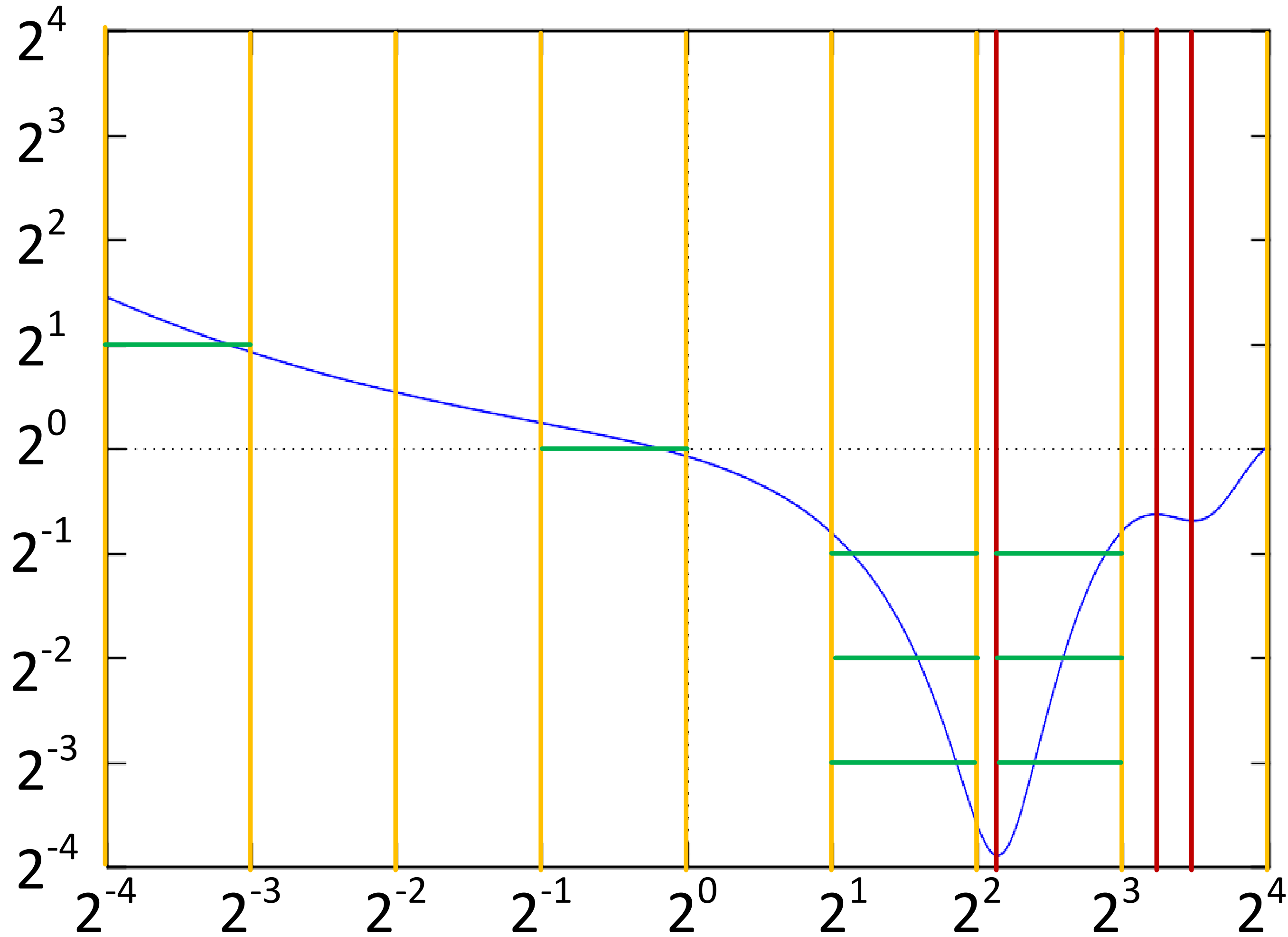
1 : Segment using input binades



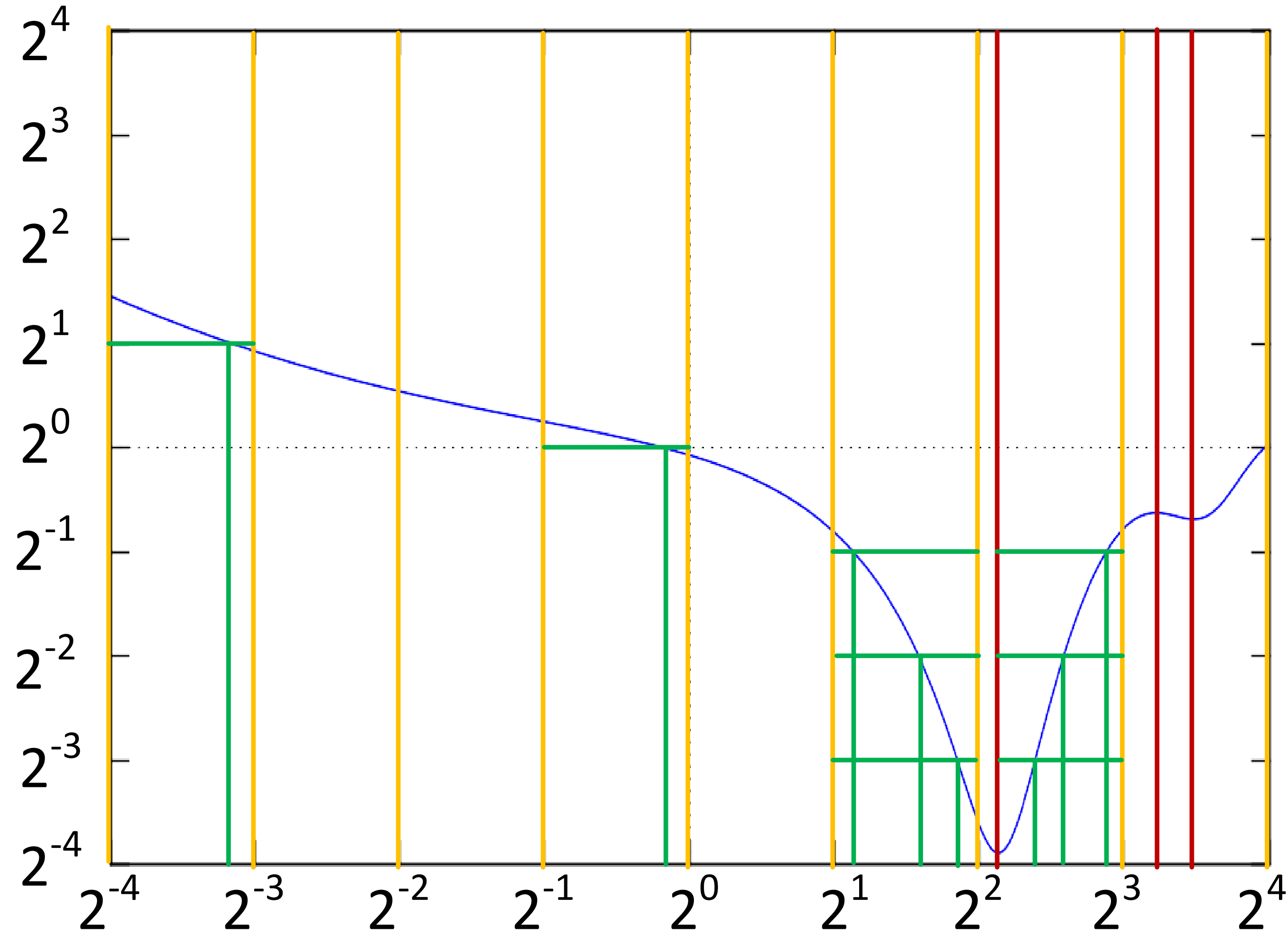
2 : Make segments monotonic



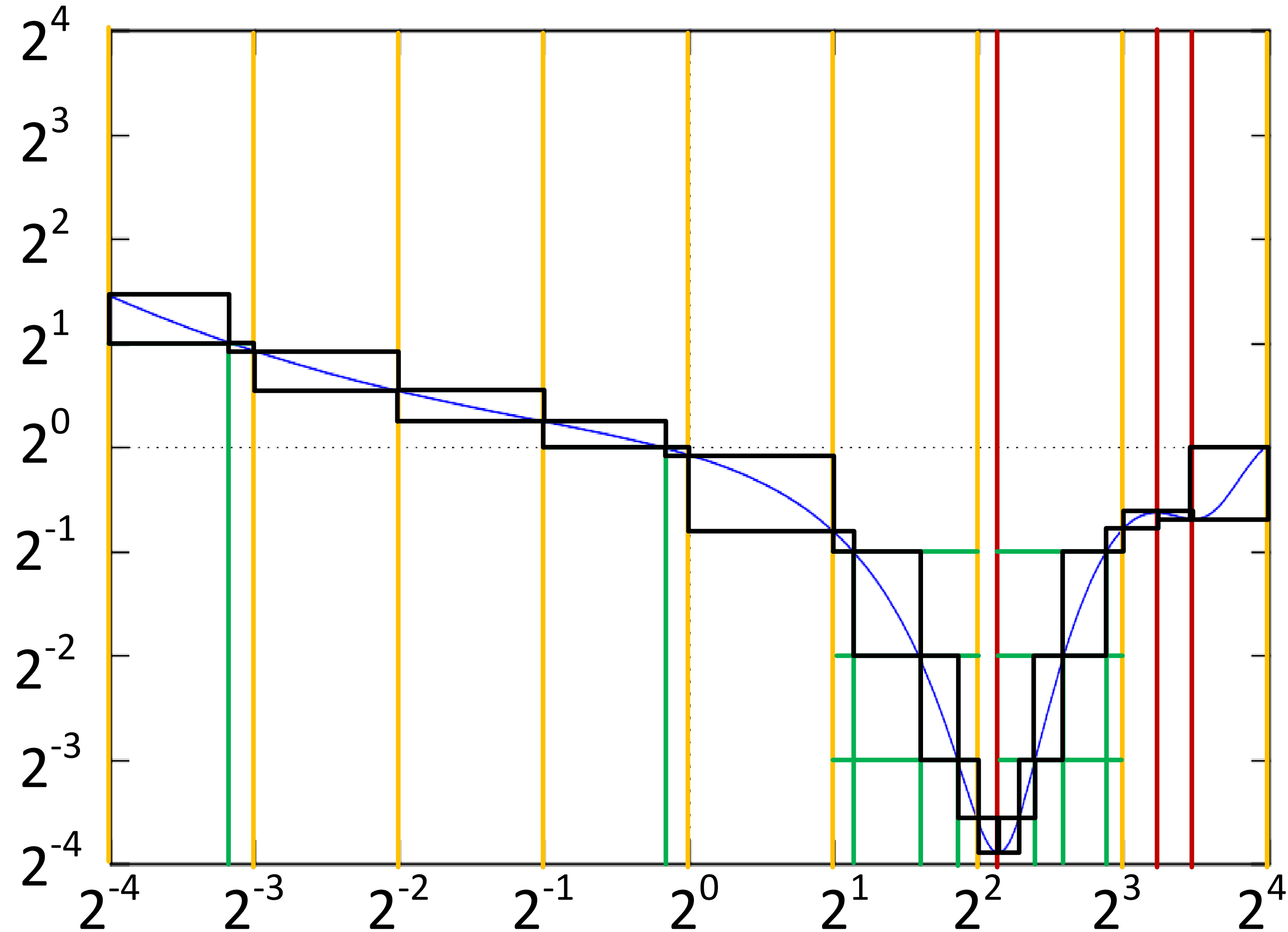
3 : Segment using output binades



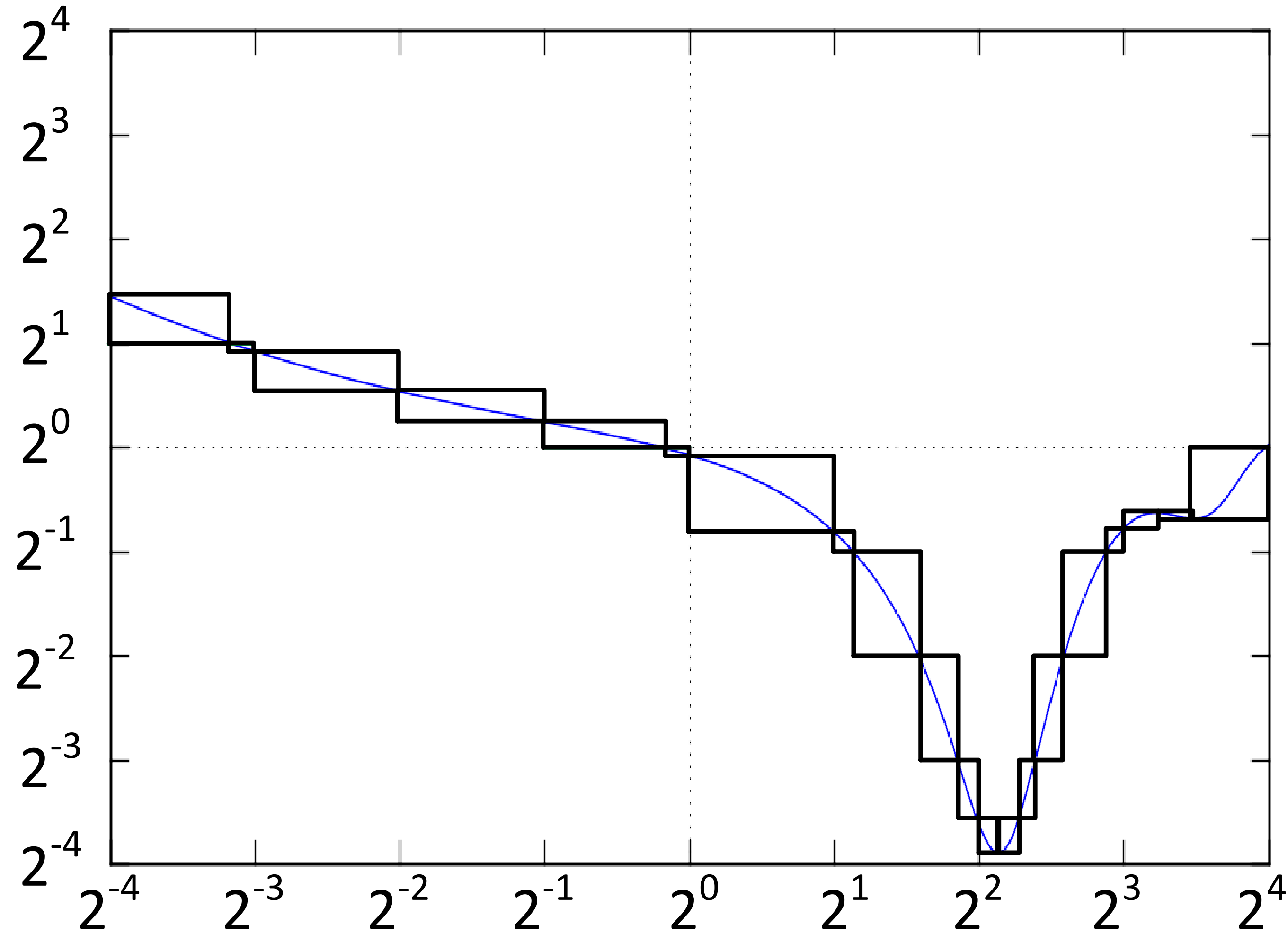
3 : Segment using output binades



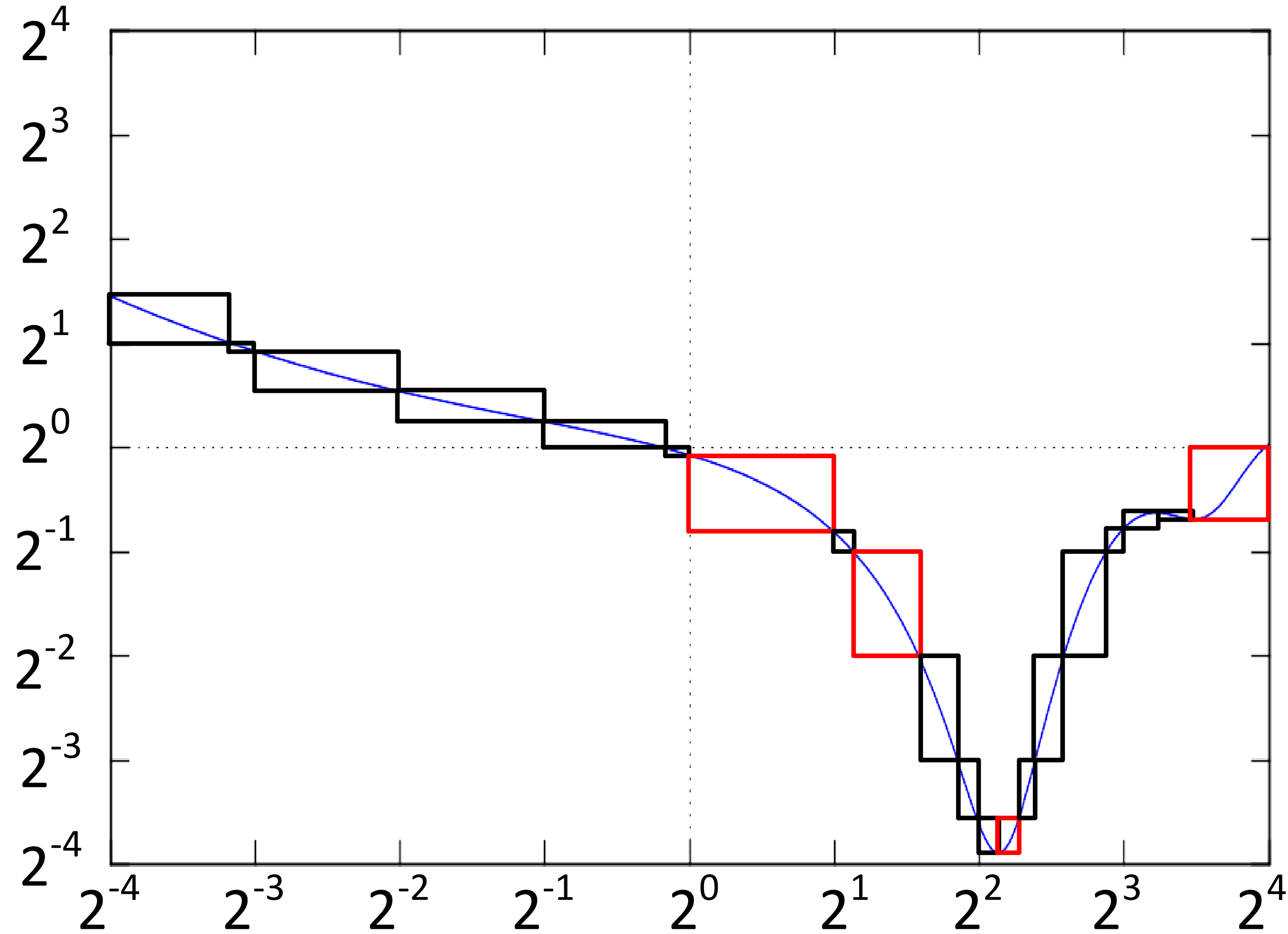
3 : Segment using output binades



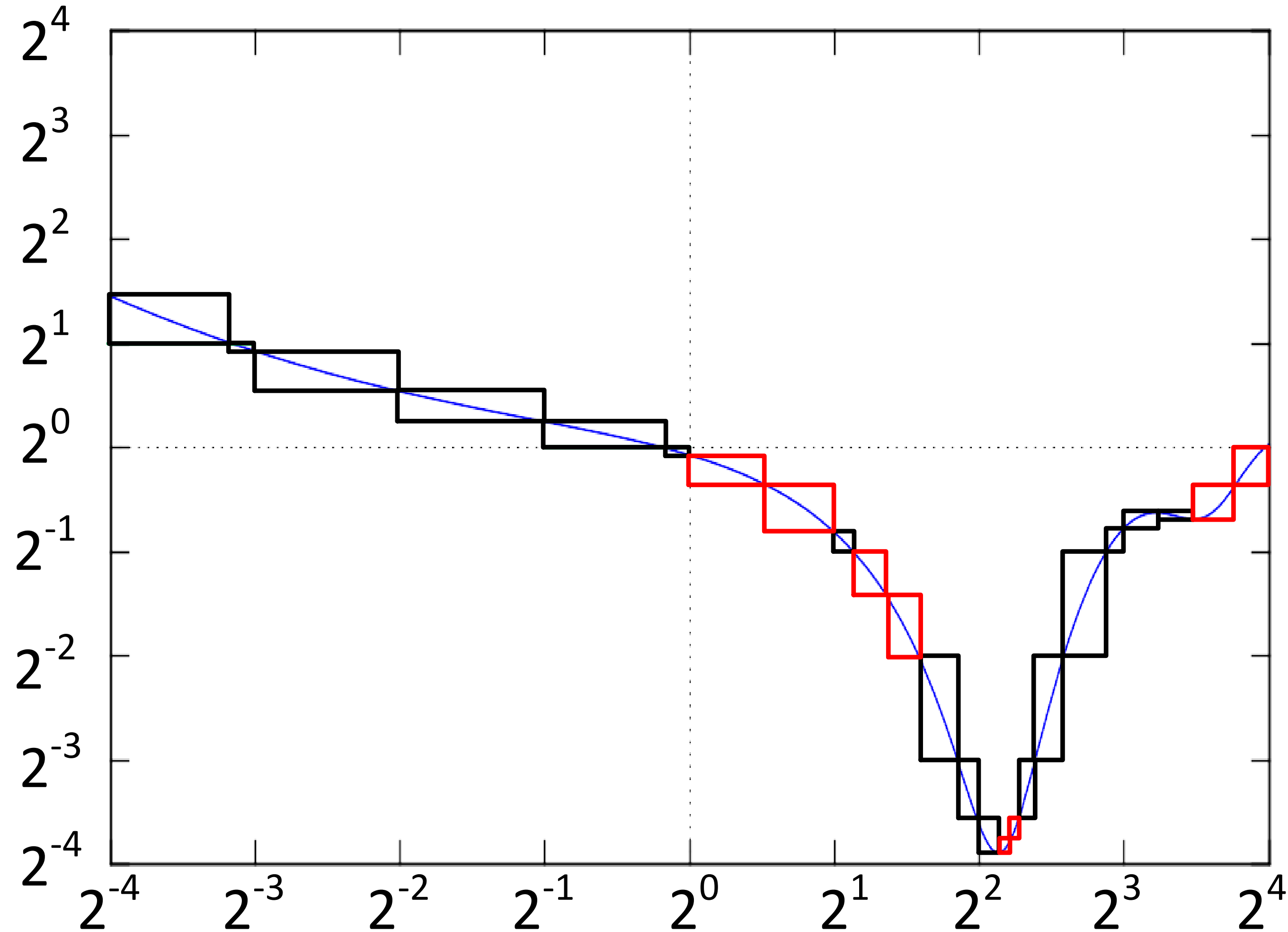
3 : Segment using output binades



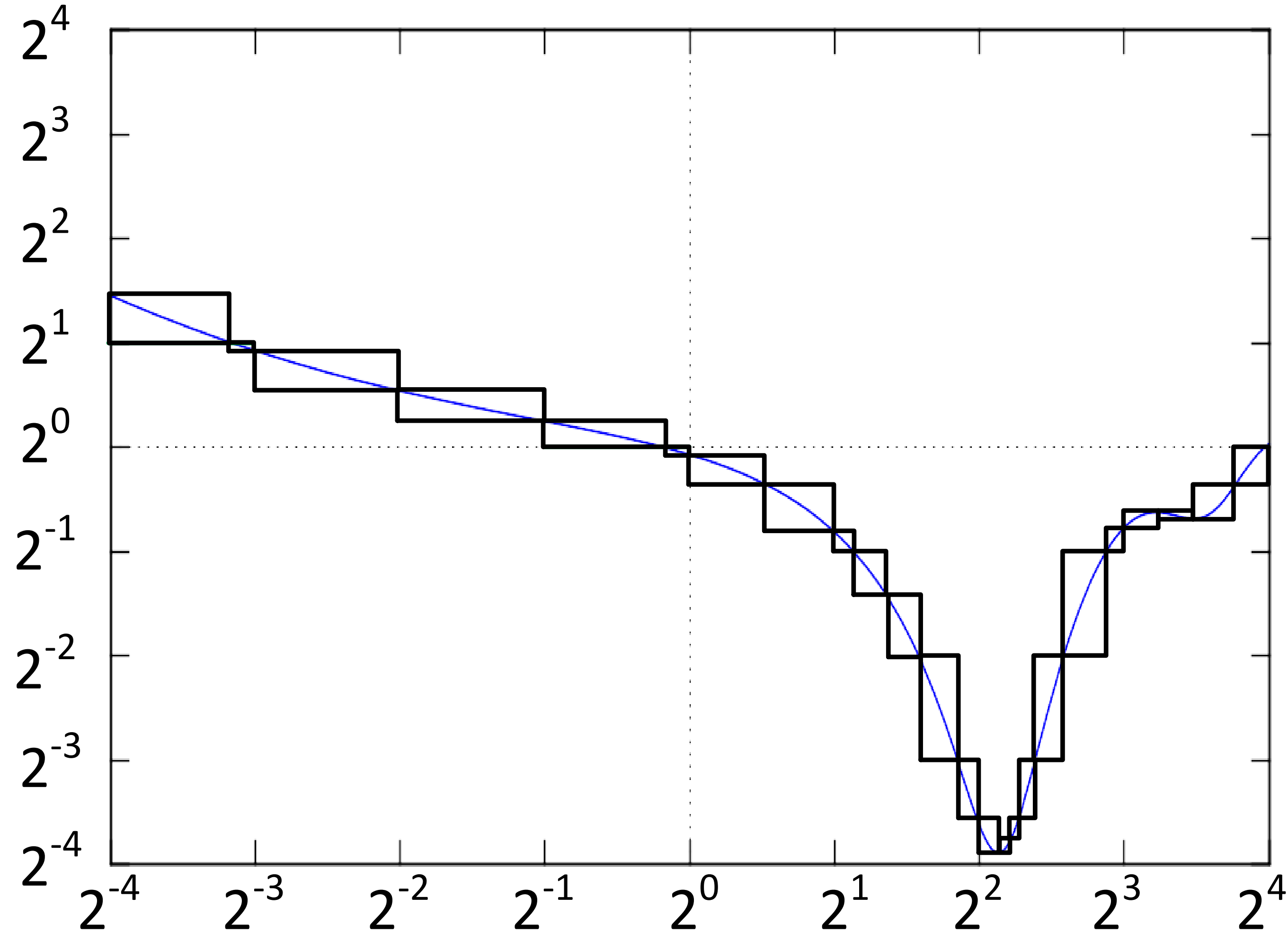
4 : Split to degree d polynomials



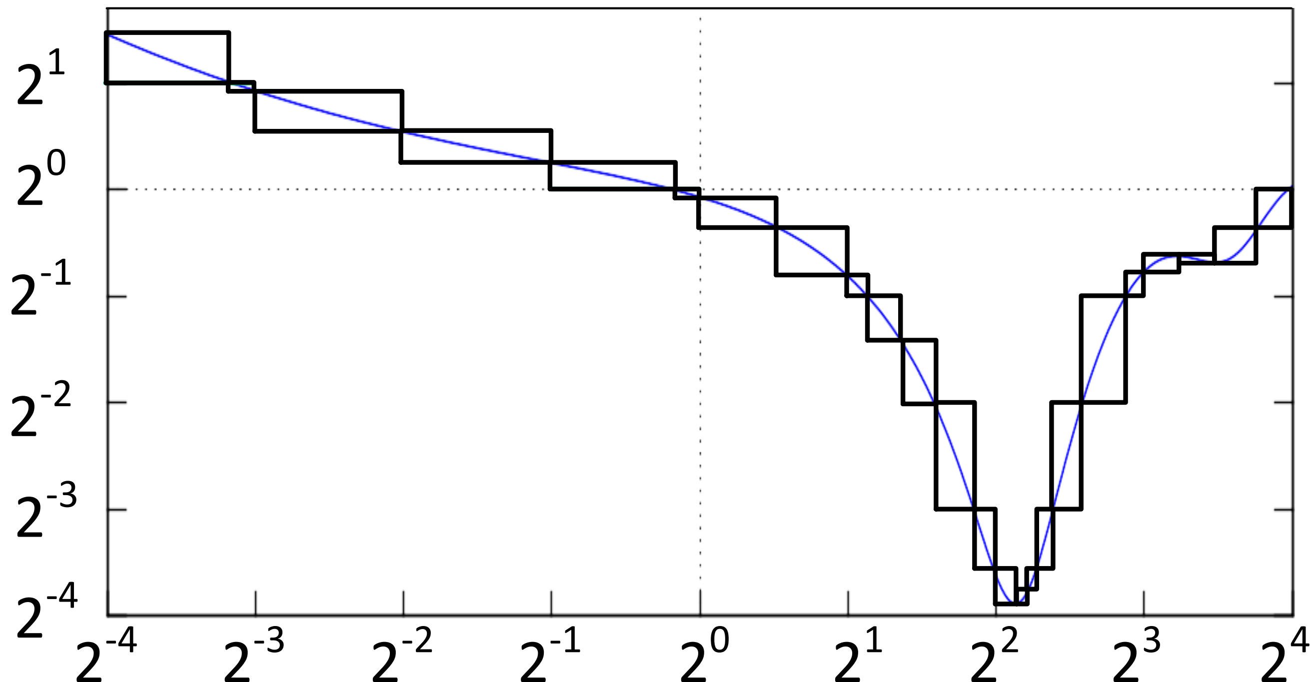
4 : Split to degree d polynomials



4 : Split to degree d polynomials



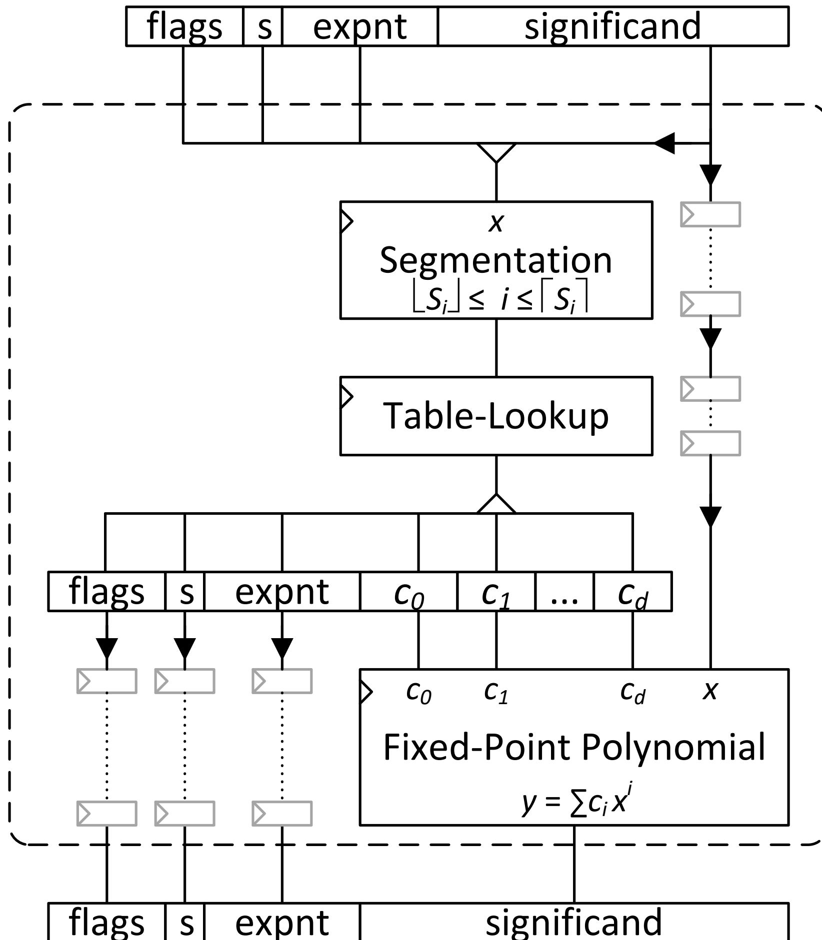
- Segments form a partition on input domain
- Segment domains and ranges cover one binade
- All segments can be faithfully calculated as degree *d* fixed-point polynomial



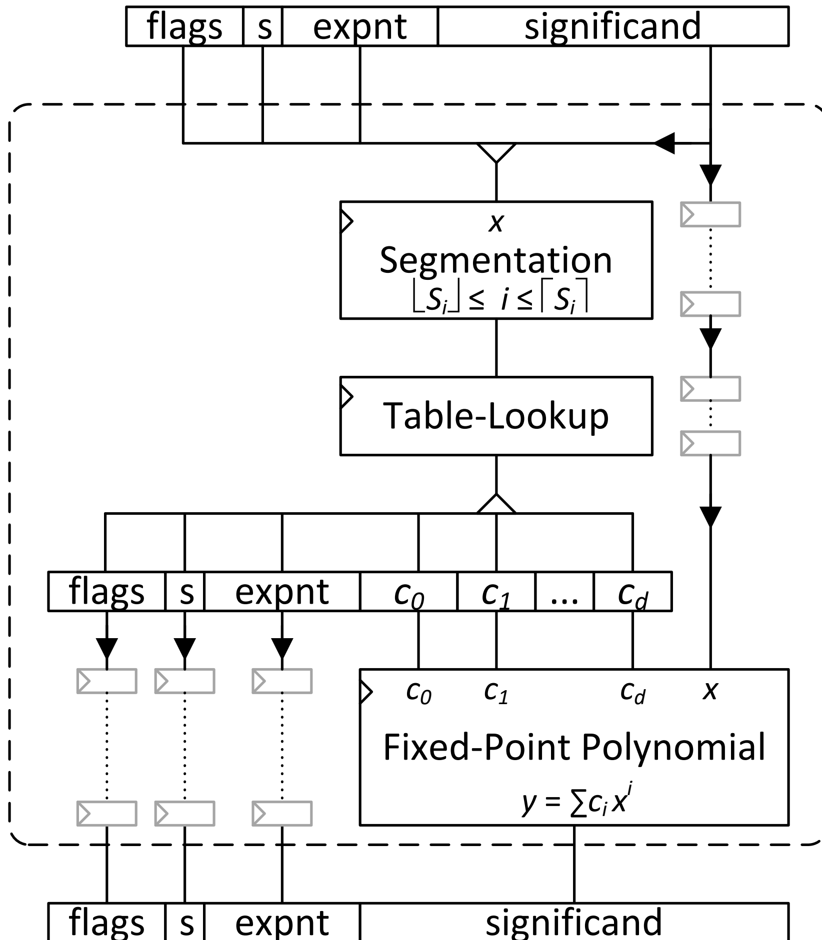
Real-world issues

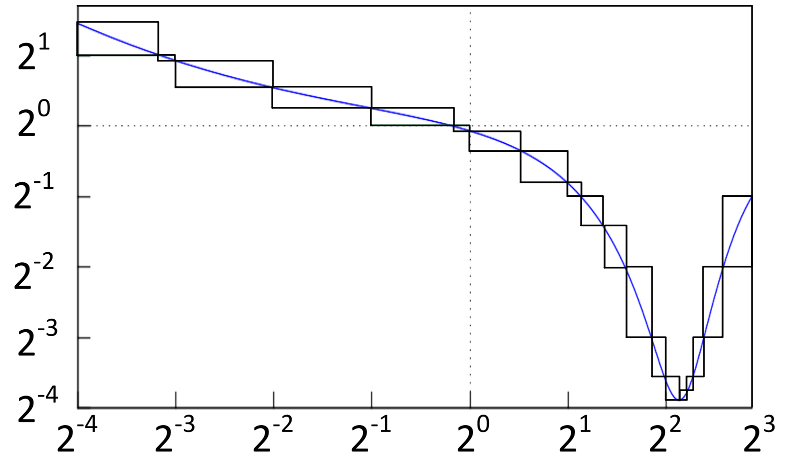
- Lots of corner cases to worry about
  - Crossing from negative to positive to NaN is fun
  - Method *should* be faithful by construction
- Calculations performed using *mpfr* and *sollya*
  - Mostly interval arithmetic via sollya
  - Occasionally bisection search in mpfr
- Speed of approximation is an issue
  - Single precision usually takes minutes
  - Double precision often takes hours

FloatApprox : Architecture

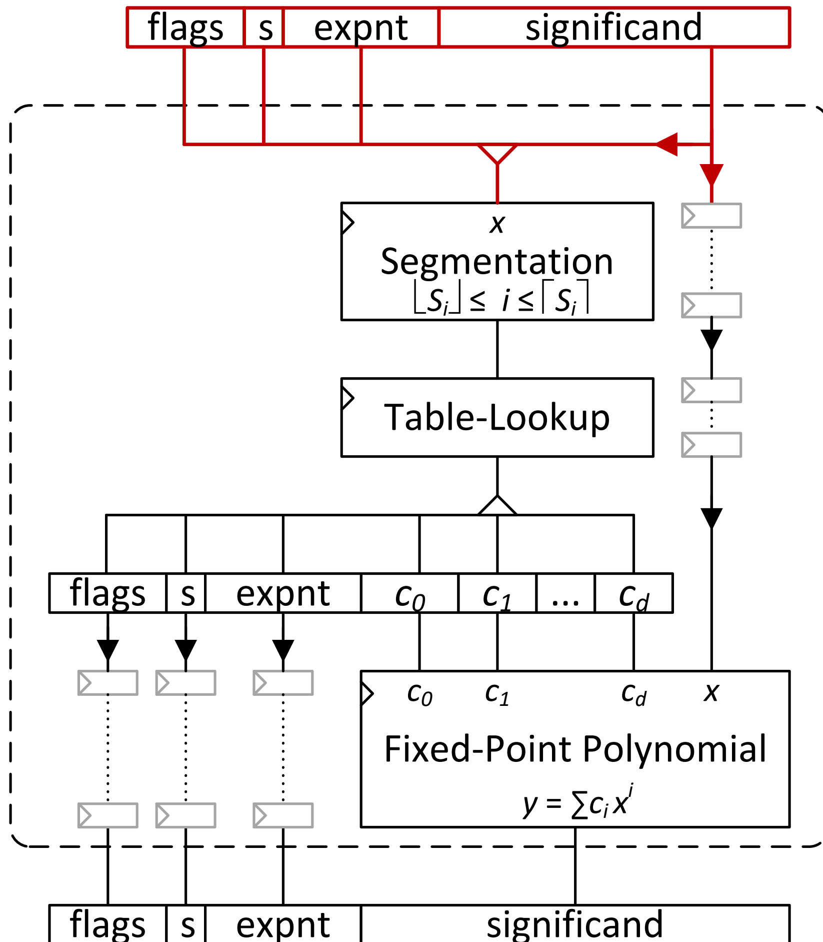


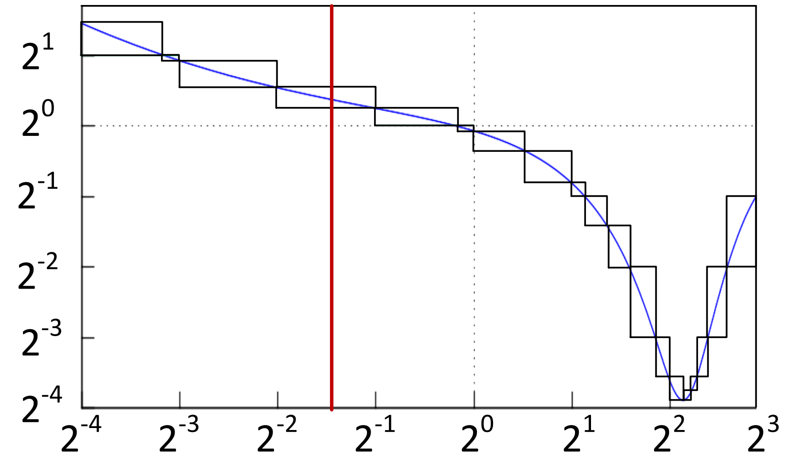
Compile-time configuration



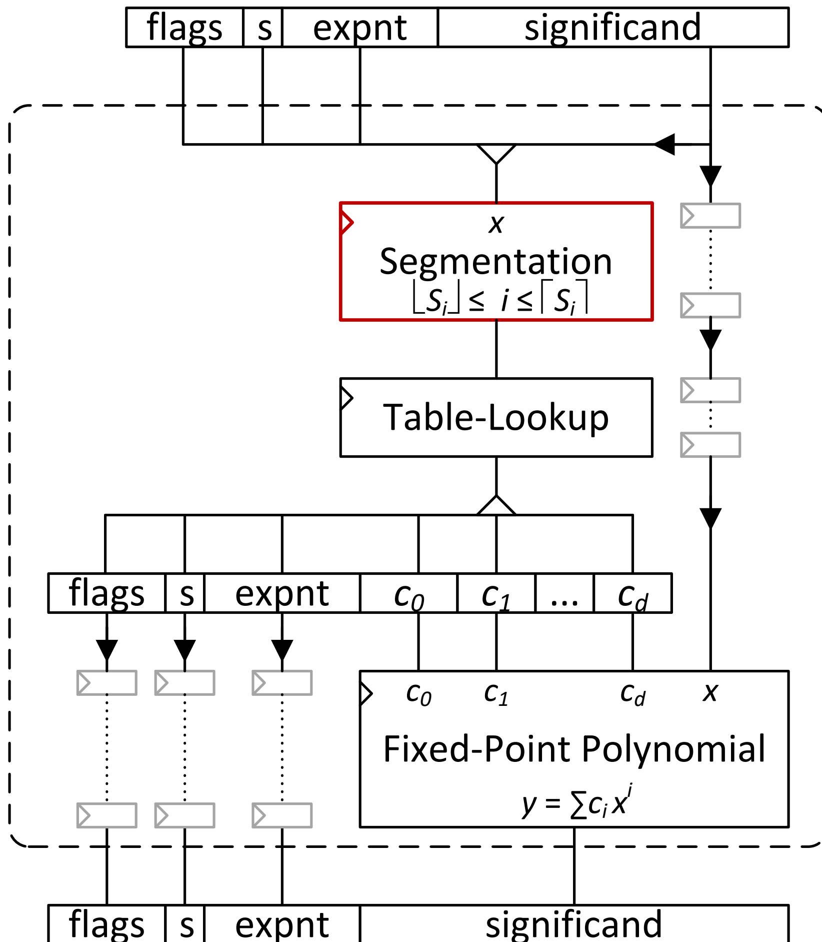


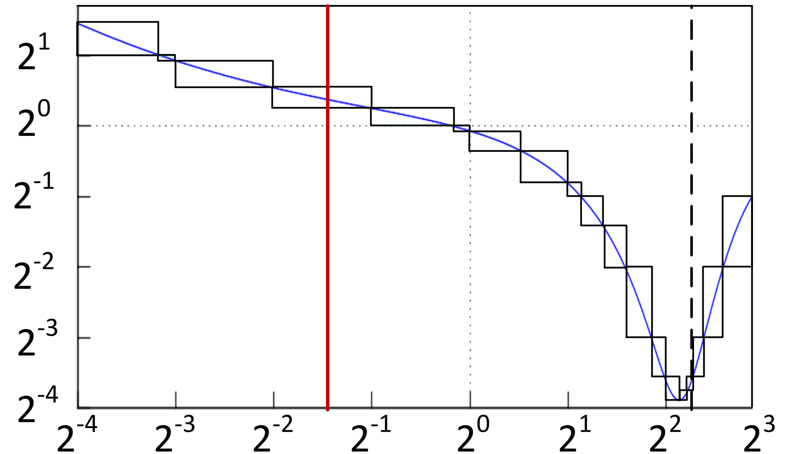
Evaluation: Input



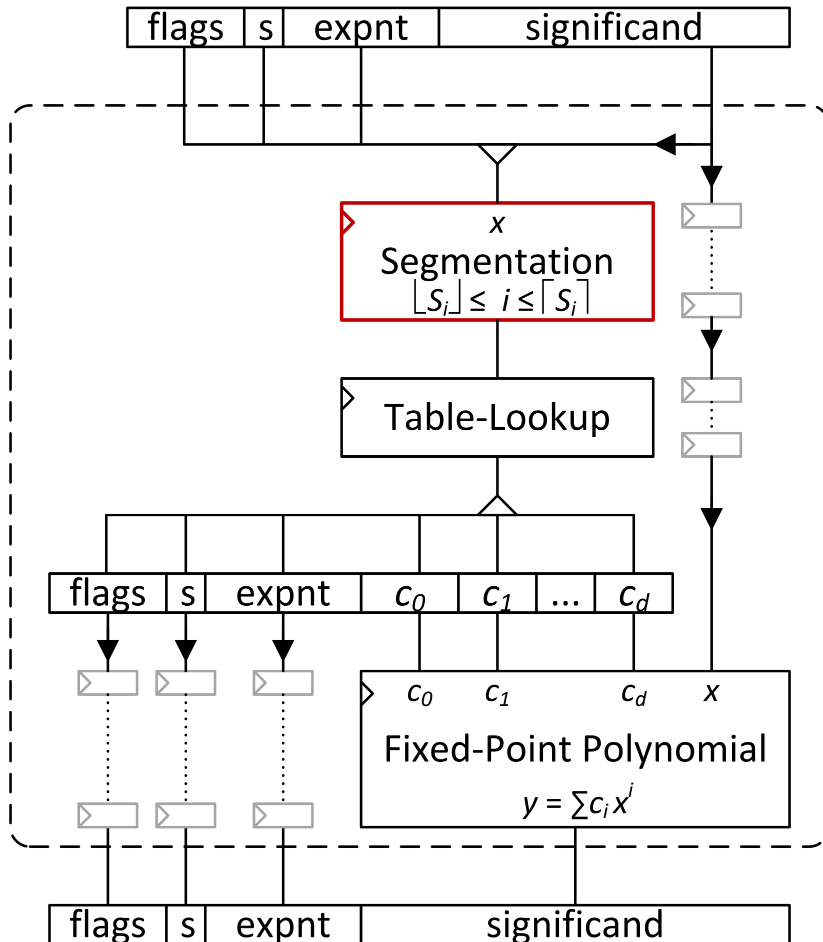


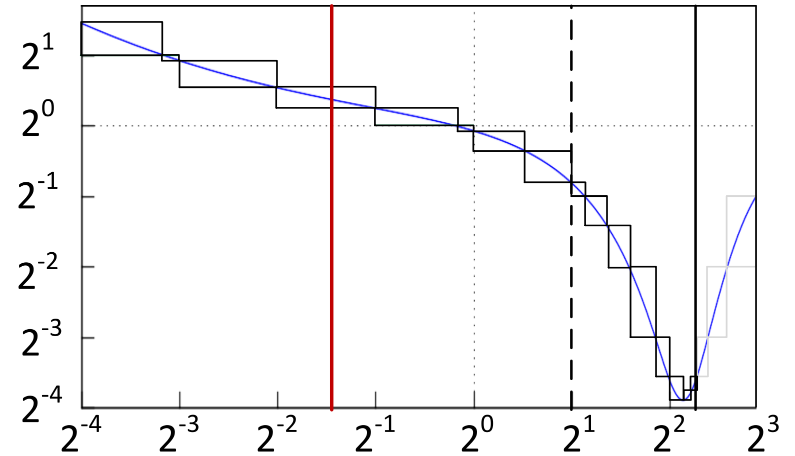
Evaluation: Segmentation



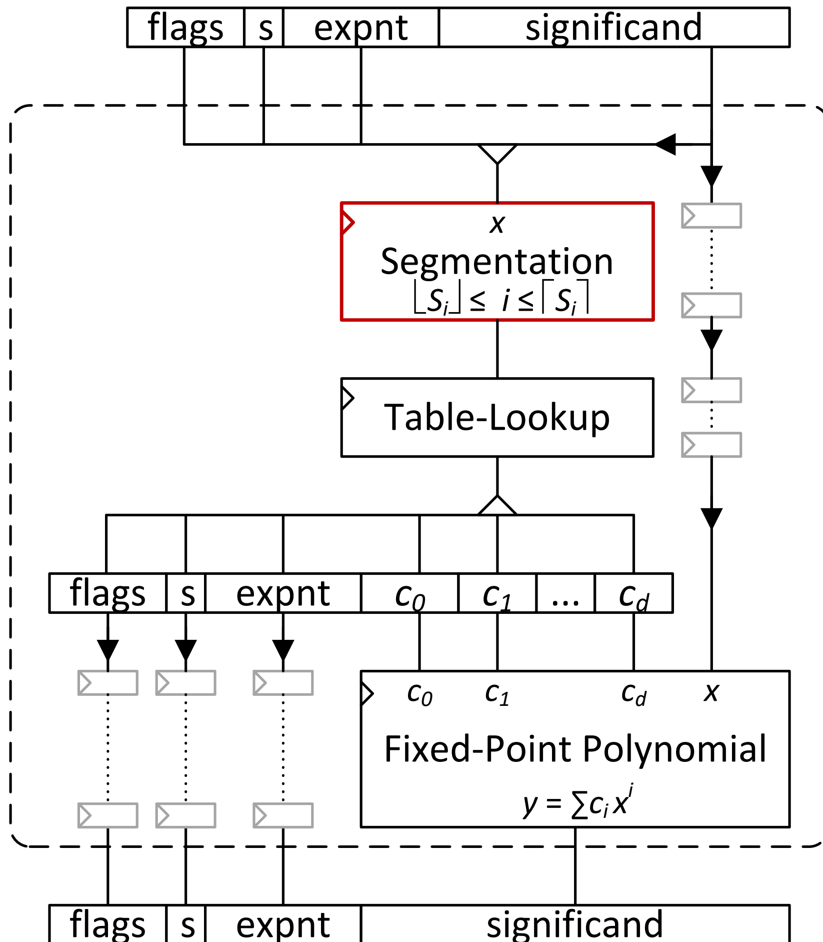


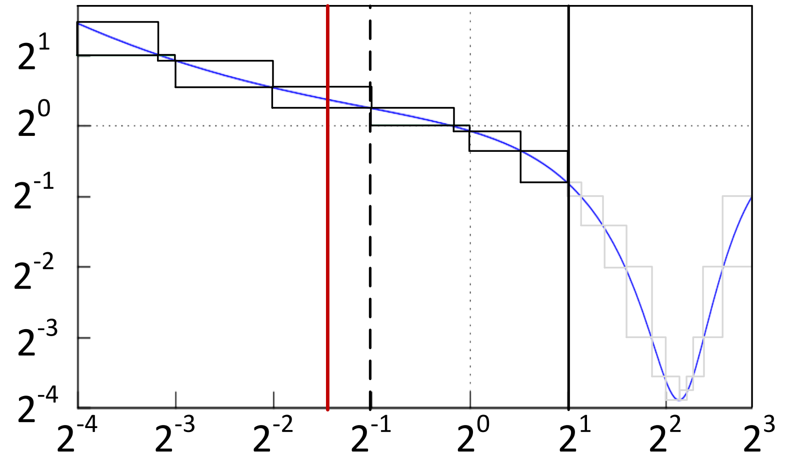
Evaluation: Segmentation



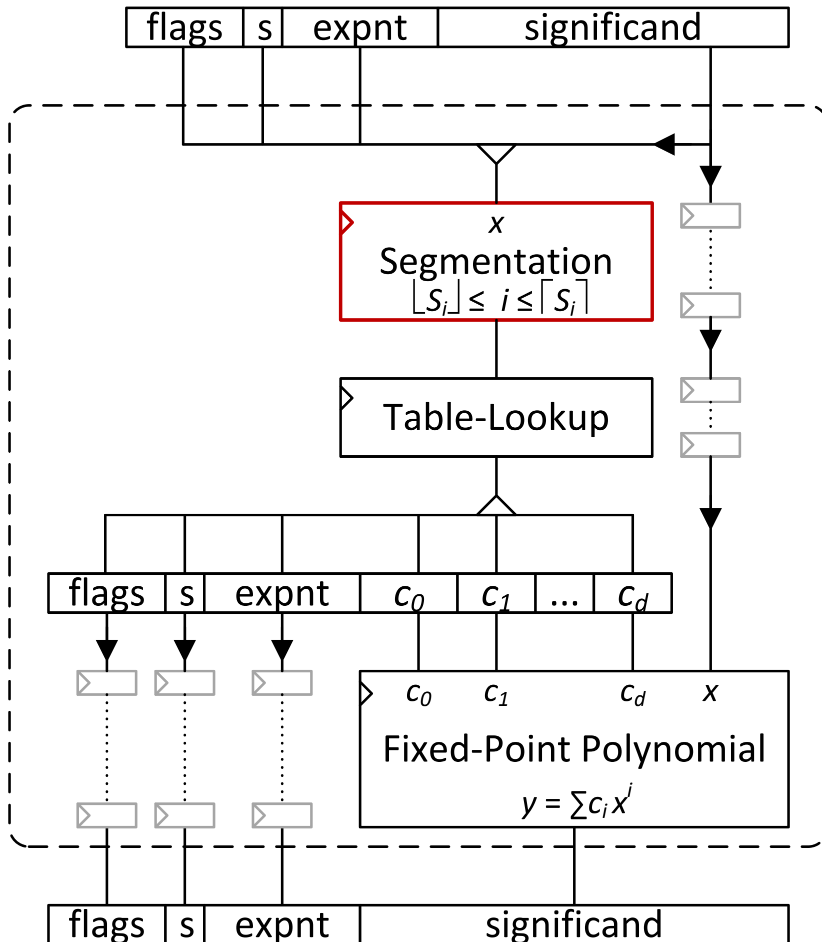


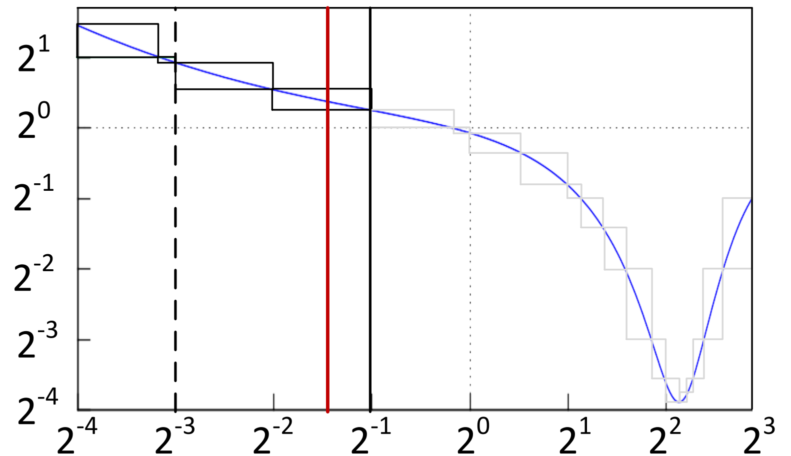
Evaluation: Segmentation



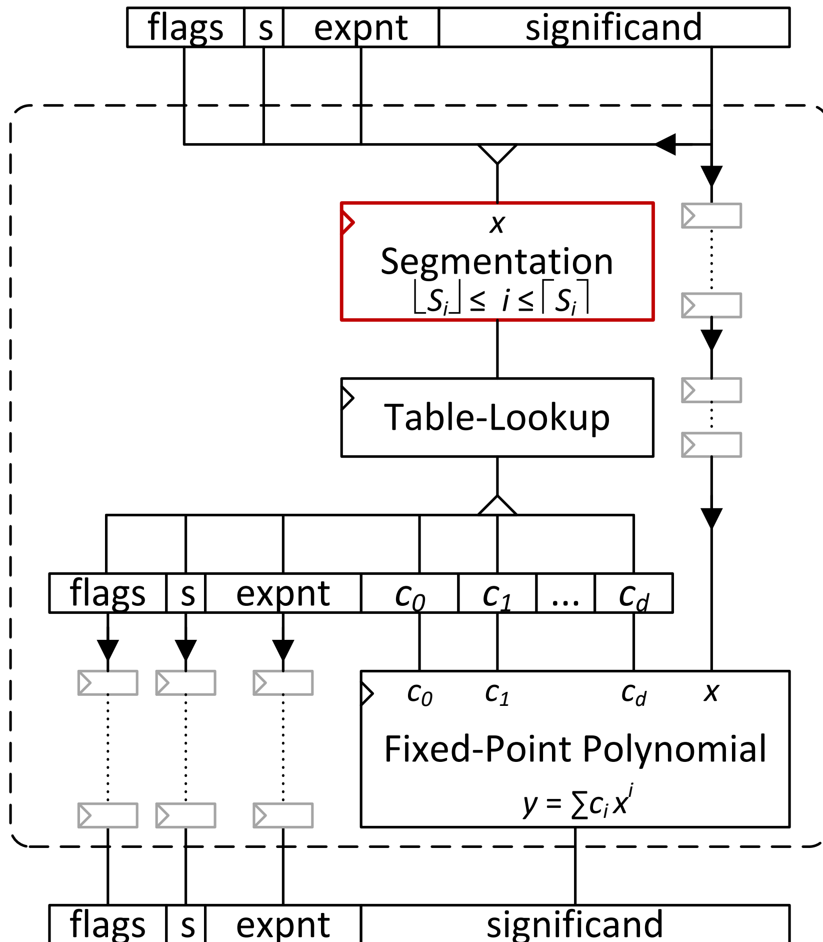


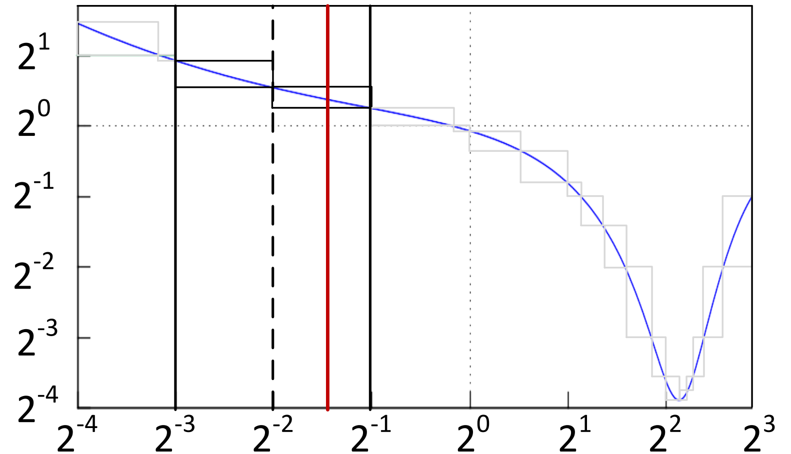
Evaluation: Segmentation



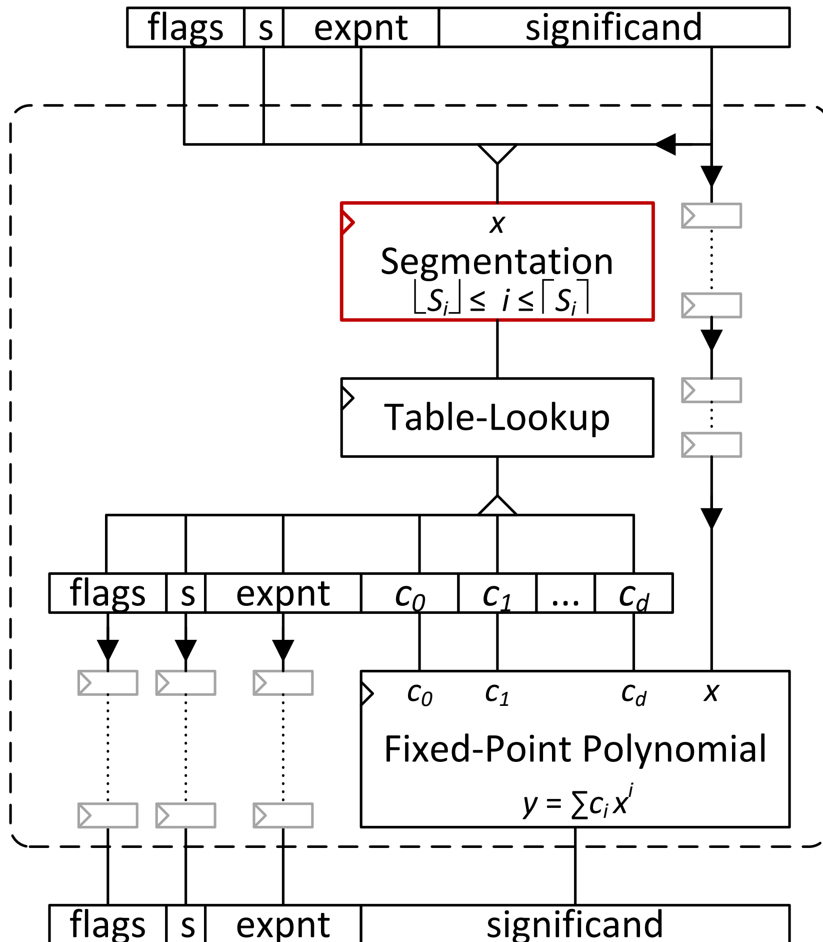


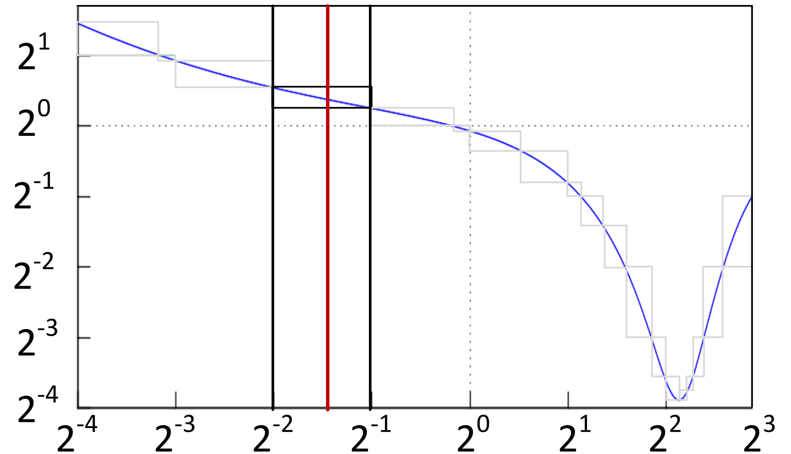
Evaluation: Segmentation



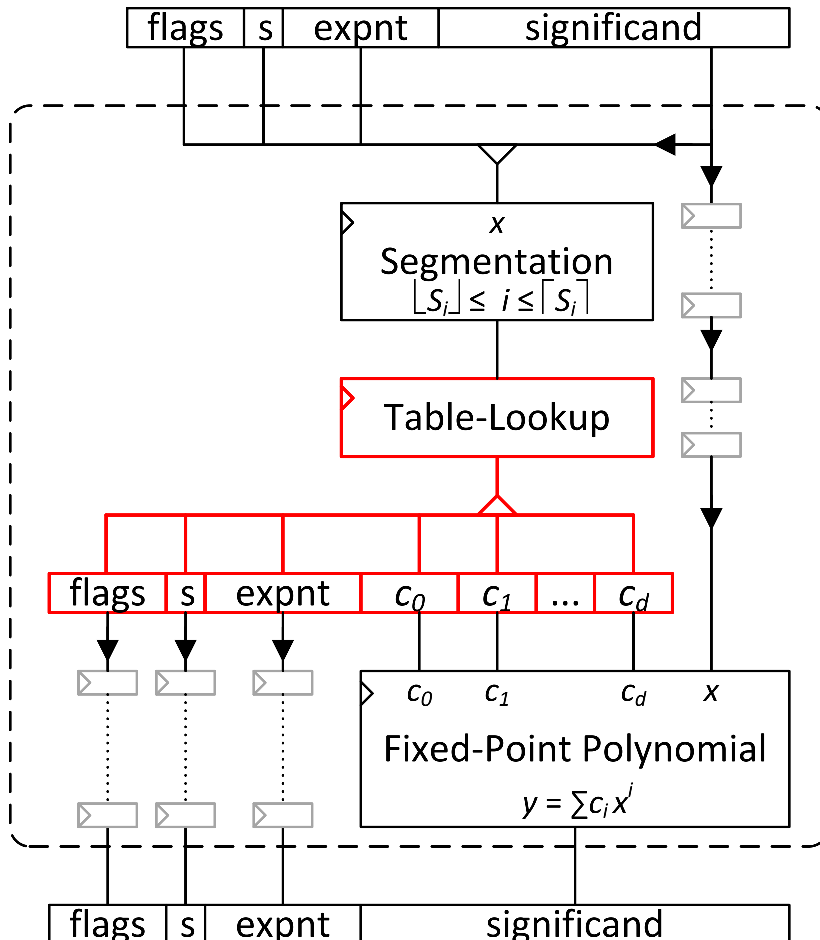


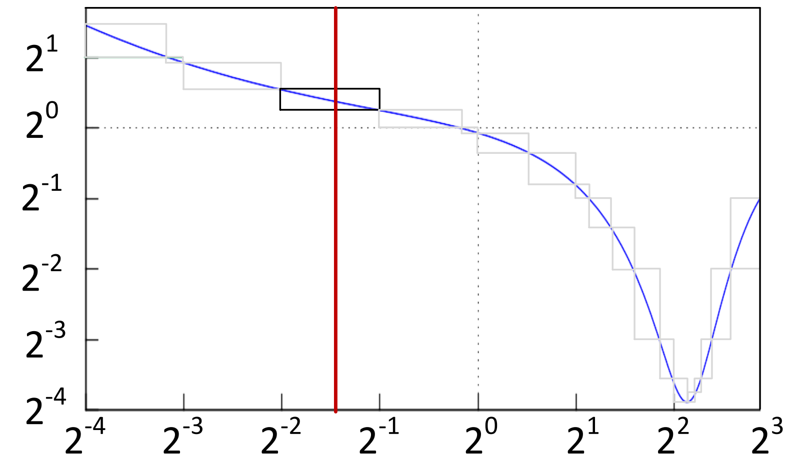
Evaluation: Segmentation

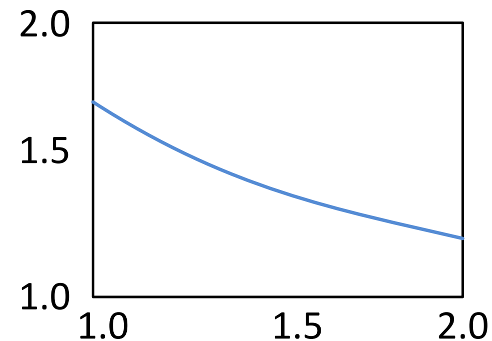




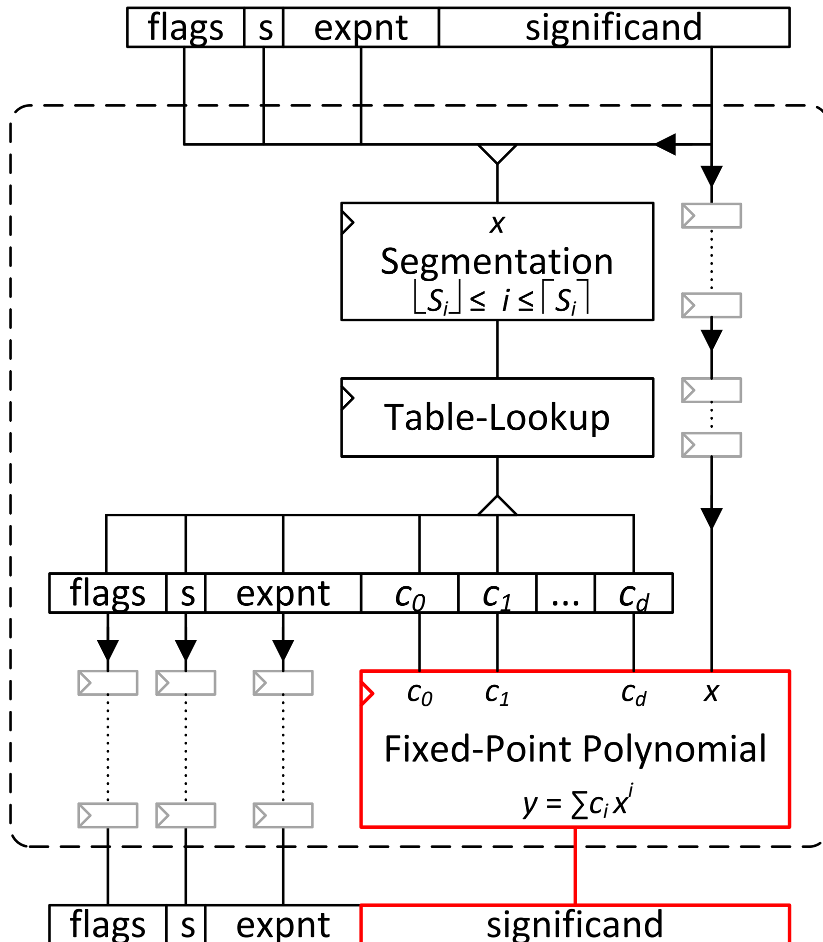
Evaluation: Table Lookup

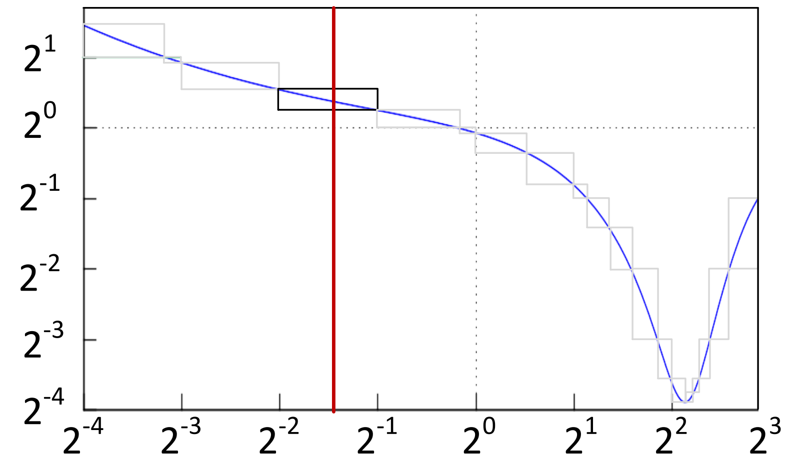


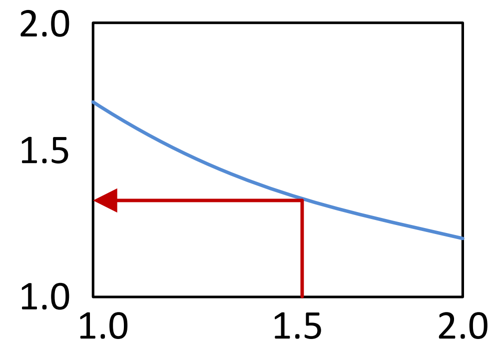




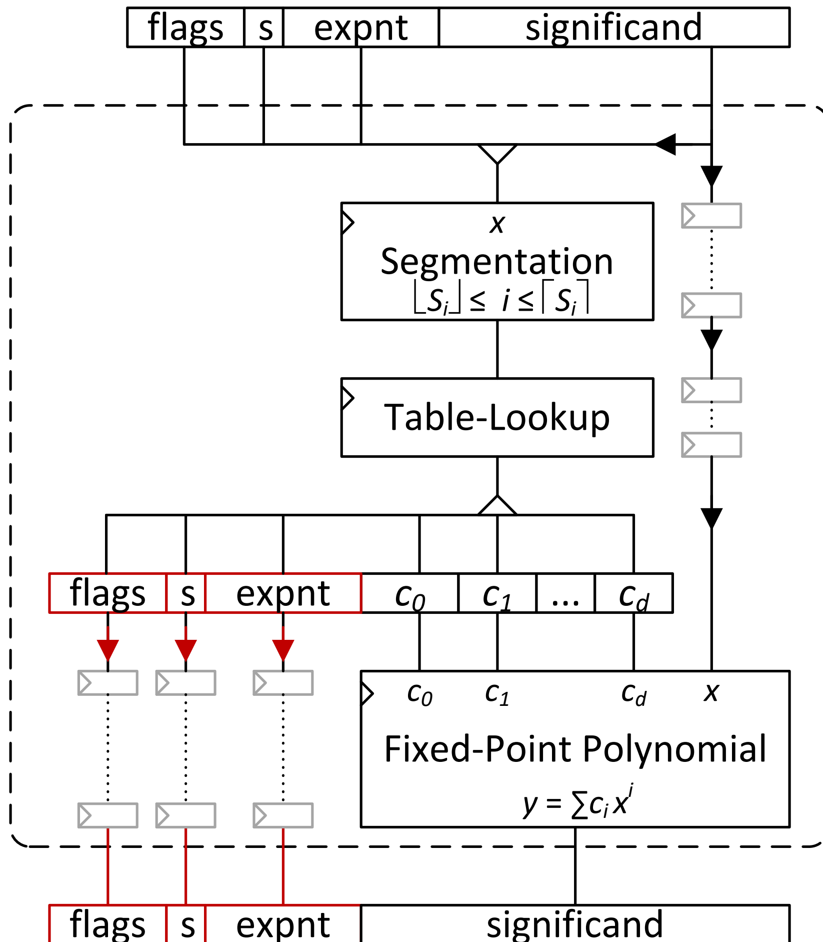
Evaluation: Fraction

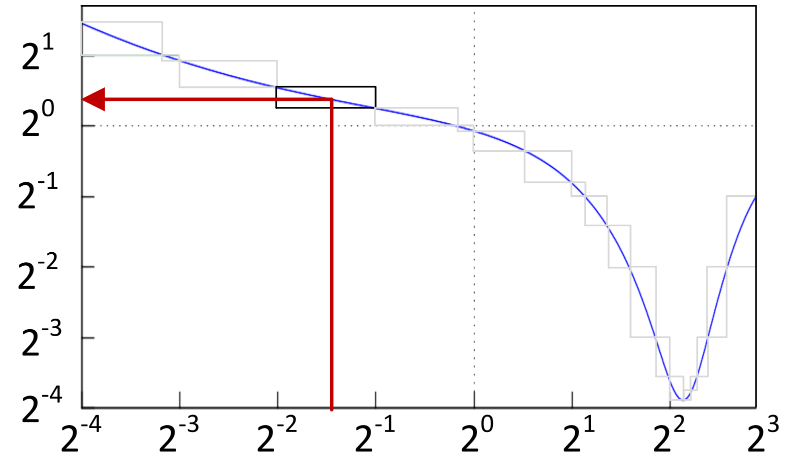




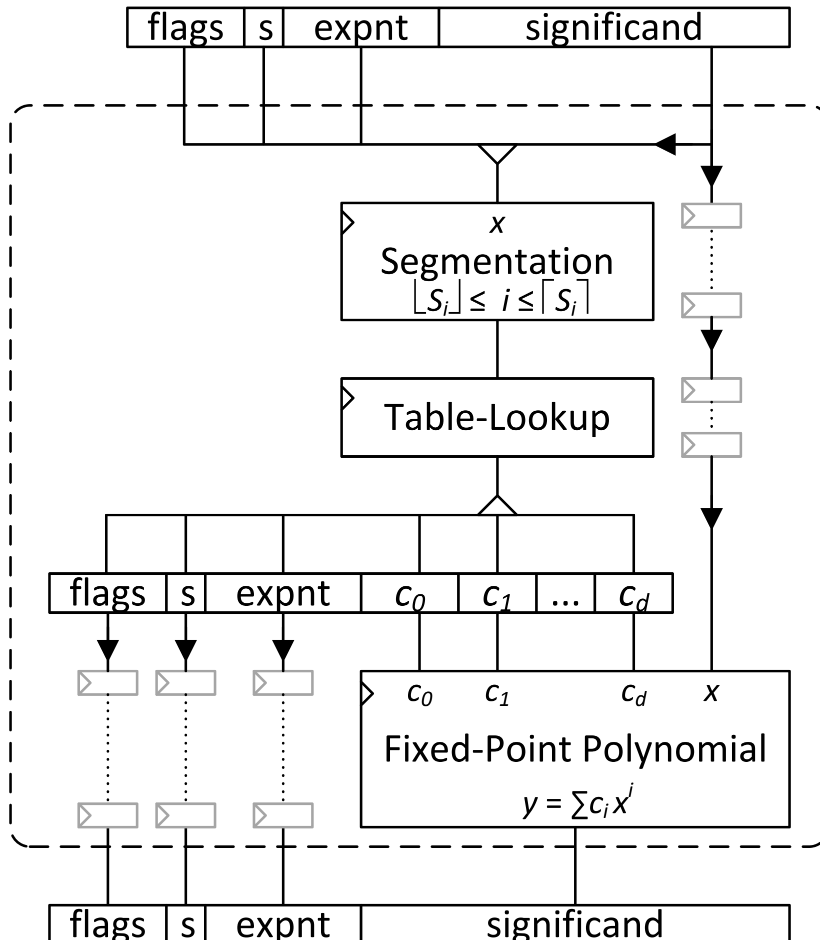


Evaluation: Flags and Exponent





FloatApprox : Architecture



/cygdrive/e/_dt10_/VMs/dev

```
Final report:
|---Entity ComparableFloatEncoded_wE5_wF10_uid4
|      Not pipelined
|    |---Entity StaticQuantiser_Table_uid8
|    |      Not pipelined
|    |---Entity StaticQuantiser_Table_uid11
|    |      Not pipelined
|    |---Entity StaticQuantiser_Table_uid14
|    |      Not pipelined
|    |---Entity StaticQuantiser_Table_uid17
|    |      Not pipelined
|    |---Entity StaticQuantiser_Table_uid20
|    |      Not pipelined
|---Entity StaticQuantiser_w17_uid6
|      Pipeline depth = 5
|---Entity FloatApprox_uid2_table
|      Not pipelined
|    |---Entity IntMultiplier_UsingDSP_12_11_0_sig
|    |      Pipeline depth = 1
|    |---Entity IntMultiplier_UsingDSP_20_11_0_sig
|    |      Pipeline depth = 1
|    |---Entity IntMultiplier_UsingDSP_20_11_0_sig
|    |      Pipeline depth = 1
|---Entity RoundingPolynomialEvaluator_d3_uid24
|      Pipeline depth = 16
Entity FloatApprox_uid2
   Pipeline depth = 22
Output file: flopoco.vhdl
vagrant@vagrant-ubuntu-trusty-64:~$ less flopoco.
```

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
- 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

	Name	Method	Interval
Primitive	log	IP core	[0,∞]
	exp	IP core	[-∞, ∞]
Composite	normpdf	exp(-x^2)/sqrt(2pi)	[-16,16]
	sigmoid	1/(1+exp(-x))	[-∞,+∞]
	log1p	log(x+1)	[-1,+∞]
	expm1	exp(x)-1	[-∞,+∞]
Approximate	sin	mul:7, add:5	[-π,+π]
	cos	mul:6, add:5	[-π,+π]
	erf	mul:7, add:7, inv:1, exp:1	[-32,+32]

Name	LUTs	BRAM	DSP	Latency
log exp	2.5x	18x	10x	4x
normpdf sigmoid log1p expm1	2x	10x	5x	1.8x
sin cos erf	0.4x	~5x	0.8x	0.7x

Name	LUTs	BRAM	DSP	Latency
log	Worse in every way			
exp				
normpdf	2x	10x	5x	1.8x
sigmoid				
log1p				
expm1				
sin	0.4x	~5x	0.8x	0.7x
cos				
erf				

<table>
<tr><th>Name</th><th>LUTs</th><th>BRAM</th><th>DSP</th><th>Latency</th></tr>
<tr><td>log</td><td rowspan="2" colspan="4">Worse in every way</td></tr>
<tr><td>exp</td></tr>
<tr><td>normpdf</td><td rowspan="4" colspan="4">Poor resource utilisation
Much better accuracy</td></tr>
<tr><td>sigmoid</td></tr>
<tr><td>log1p</td></tr>
<tr><td>expm1</td></tr>
<tr><td>sin</td><td rowspan="3">0.4x</td><td rowspan="3">~5x</td><td rowspan="3">0.8x</td><td rowspan="3">0.7x</td></tr>
<tr><td>cos</td></tr>
<tr><td>erf</td></tr>
</table>

Name	LUTs	BRAM	DSP	Latency
log	Worse in every way			
exp				
normpdf	Poor resource utilisation Much better accuracy			
sigmoid				
log1p				
expm1				
sin	More accurate and smaller except for RAMs			
cos				
erf				

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

Name	LUTs	BRAM	DSP	Latency
log	1376 (1.7x)	12 (12x)	10 (2.0x)	181 (2.2x)
exp	1372 (3.4x)	24 (24x)	9 (9.0x)	216 (5.9x)
normpdf	1498 (1.8x)	25 (25x)	11 (2.8x)	190 (2.2x)
sigmoid	1259 (0.8x)	5 (5x)	9 (9.0x)	165 (1.1x)
log1p	2203 (1.9x)	10 (10x)	17 (3.4x)	214 (1.3x)
expm1	1304 (1.8x)	12 (12x)	9 (9.0x)	173 (2.5x)
sin	1366 (0.5x)	5 (∞x)	10 (0.8x)	189 (0.8x)
cos	1220 (0.5x)	5 (∞x)	9 (0.8x)	200 (0.8x)
erf	881 (0.2x)	4 (4x)	6 (0.6x)	149 (0.4x)