Natural logarithm implementation for BFloat16 FPU

Andrea Buffoli
Matteo Giacomello
Artem Glukhov
Giacomo Ticchi

May 9, 2020

Abstract

The goal of the project is to add the support for the natural logarithm operation to a Floating Point Unit and verify the correctness against the results produced by the corresponding C function. Thus we designed a module which calculates the natural logarithm of a number in bfloat16 floating-point format (Brain Floating Point) and provides the result in the same format.

Chapter 1

Algorithm and Architecture

Considering only the case where the input X is a valid positive bfloat16 number, i.e.

$$X = F * 2^{(E-E_0)}$$

If we calculate its natural logarithm, we obtain:

$$R = log(X) = log(F) + (E - E_0) * log(2)$$

Therefore the evaluation of the result is divided in two main parts:

- computation of $log(F)$ with $F \in [0, 2)$
- computation of the product $(E - E_0) * log(2)$

In order to get a better accuracy the output range of $log(F)$ is centered around zero in the following way:

$$R = \begin{cases} log(F) + (E - E_0)log(2) & when \text{ F} \in [0, \sqrt{2}) \\ log(\frac{F}{2}) + (1 + E - E_0)log(2) & when \text{ F} \in [\sqrt{2}, 2) \end{cases}$$

Input operators different from the X specified above are handeled as special cases, in particular:

Input		**Output**
sign bit	**special cases**	**special cases**
1(negative)	Any value	NaN
0(positive)	Zero	Negative Infinite
	Positive Infinite	Positive Infinite
	NaN	NaN

The architecture we implemented is the following (slightly re-arranged from the paper by Florent de Dinechin and Jérémie Detrey[1]):

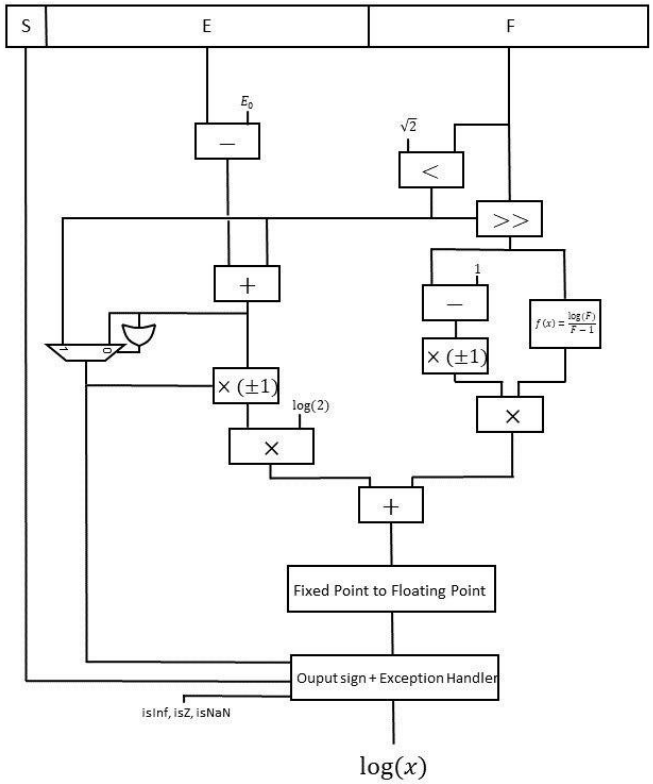

Figure 1.1: Block design of the implemented algorithm

Some considerations:

- The function $f(x) = \dfrac{log(F)}{F-1}$ is implemented through a simple LUT with 8 input bits and 12 output bits.
- The conversion from Fixed Point format to Bfloat16 is made with a mask.
- The sign of the result is the sign of $(E - E_0)$ (its MSB) when $(E - E_0) \neq 0$; otherwise it is given by the comparison between F and $\sqrt{2}$.
- Overflow and Underflow conditions are never met because the non-infinite output range goes from $[-92.0; +88.5]$ and the smallest ouput is $3.9e^{-3}$ (in modulus) so both the largest and smallest possible outputs are representable in Bfloat16 format.
- The module supports denormal numbers with respect to the one proposed by the paper: the LUT is implemented with an input range between $[\frac{\sqrt{2}}{2}; \sqrt{2}]$ while we have used an input range between $[2^{-7}; \sqrt{2}]$ which solves the problem.

[1] Parameterized floating-point logarithm and exponential functions for FPGAs

Chapter 2

TestBench and Simulation

Before instantiating the module in lampFPU_top, a simulation was mandatory in order to check all possible errors and bugs.
It was decided to simulate the module on Vivado with an ad hoc testbench, so to fix in first approximation the main problems of the algorithm. To test the precision, a simulation with all of the acceptable inputs was executed (the full range from $S = 0$ $E = 0x00$ $F = 0x00$ to $S = 0$ $E = 0xFF$ $F = 0x7F$, 32768 values), the results were dumped on a csv file and parsed by means of a MATLAB script.
In the following figures, an error histogram is plotted, and the average and percentage errors are computed. This script takes into account all the possible special cases in input, i.e. SNan, QNan. The errors are more significant with respect to the simulations based on the DPI, because MATLAB is much more precise.

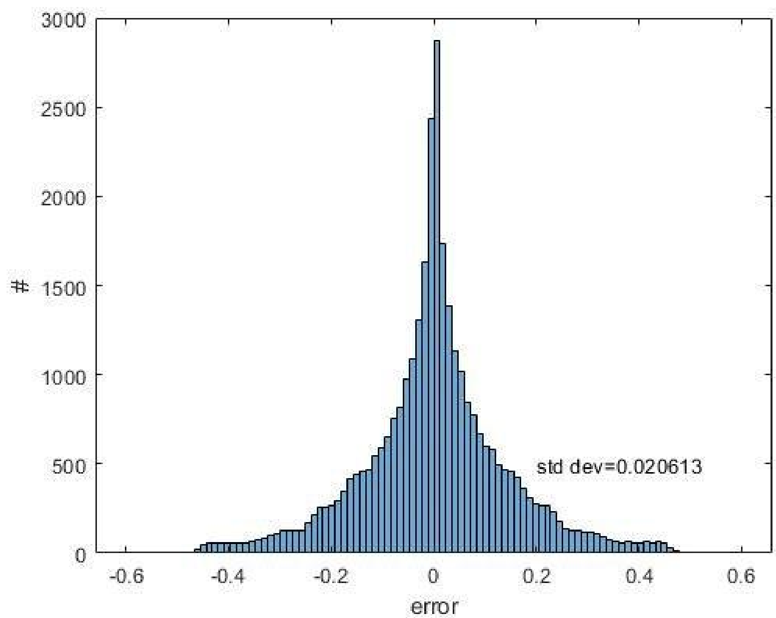

Figure 2.1: Histogram of the error.

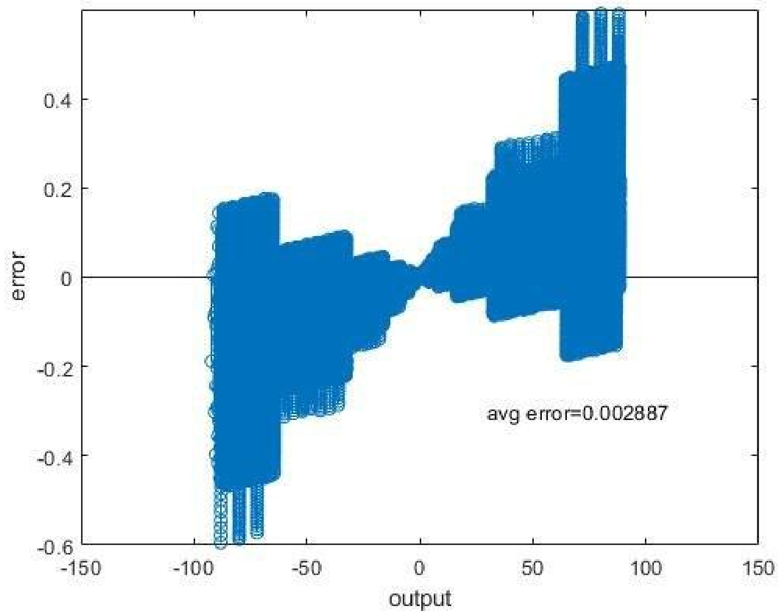

Figure 2.2: Errors with respect to Matlab results.

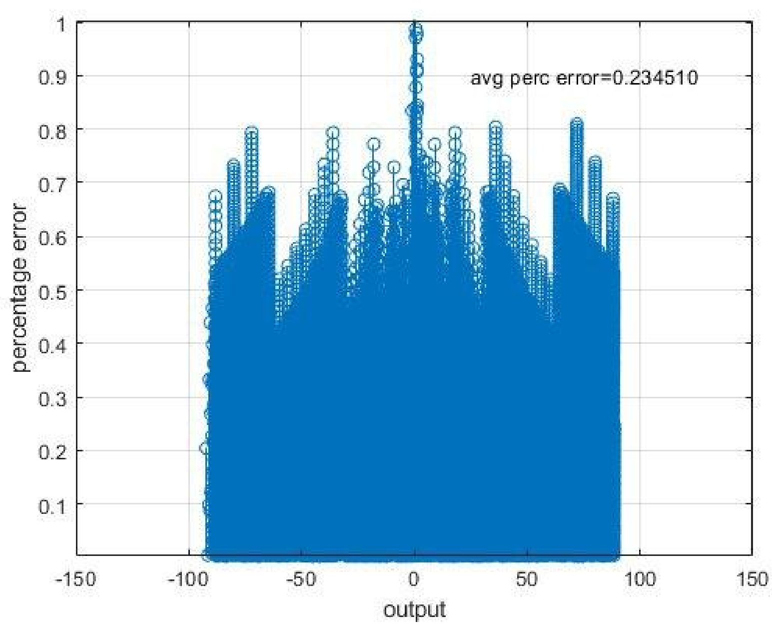

Figure 2.3: Percentage error.

Chapter 3

Integration

The aim of the project was to implement the natural logarithm module inside an already existing Floating Point Unit (FPU) provided by Andrea Galimberti and Davide Zoni [1]

The signals used in the module are:

INPUT:	COMMENT:
clk	clock signal
rst	reset signal
doLog_i	activates the module
s_op_i	sign of the input operand
extE_op1_i	extended exponent of the input operand
extF_op1_i	extended fractional of the input operand
isZ_op_i	flag: true if input is zero
isInf_op_i	flag: true if input is infinite
isSNAN_op_i	flag: true if input is SNaN
isQNAN_op_i	flag: true if input is QNaN
OUTPUT:	**COMMENT:**
s_res_o	sign of the output
e_res_o	exponent of the output
f_res_o	fractional of the output
valid_o	asserted to 1 when the result is ready
isOverflow_o	asserted to 1 when an overflow on the output is detected
ifUnderflow_o	asserted to 1 when an underflow on the output is detected
isToRound	1 if the output must be rounded

The integration part consisted in the connection of the submodule to the top module, and the adding of the new state into the Finite State Machine logic.
Also the Package was modified, with the addition of the folllowing functions:
LUT_log, FUNC_fix2float_log, FUNC_calcInfNanResLog
The first is a 8 bit look-up table used to calculate $\frac{log(F)}{F-1}$ [2];
The second is a function used to transform a fixed point number into a $BFloat_{16}$ SEF notation;
The last one is a function used to raise the correct output basing on the input flags.

[1] LAMP - BFloat16FPU
[2] See Chapter 1 for more details

3.1 Performances

The whole algorithm performs within 6 clock cycles while from the *doLog_i* assertion to the raise of *isResultValid_o* register happens after just 3 clock cycles.

The algorithm had to be divided into a finite state machine with two states, because the multiplication operation between internal registers was too slow to perform within one clock cycle and so caused the presence of a negative slack.

	W\O LOG	W\ LOG
LUT	1065 (1.68%)	1302 (2.05%)
FF	376 (0.3%)	399 (0.31%)
DSP	2 (0.83%)	4 (1.67%)
IO	91 (43.33%)	91 (43.33%)
BUFG	1 (3.13%)	1 (3.13%)
WNS	0.448ns	0.218ns

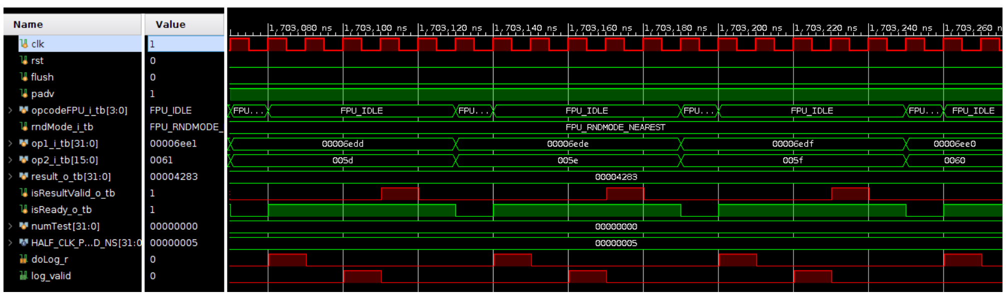

Figure 3.1: example waveforms of the Log module.

The simulation was done with the support of a Direct Programming Interface (DPI) that calculates the natural logarithm with the *math.h* C library performed on a 32bit float value, and returns a 32bit floating point value, that then is transformed to bfloat16 by truncation. This translates into a little discrepancy between some results calculated by the DPI and the ones calculated by the FPU-RTL module due to the rounding to the next even performed on the FPU-RTL output.