August 4, 2021

Università degli Studi di Padova

Hardware implementation of base 2 logarithm

Authors:
Gabriele Bortolato

1 Methods

1.1 Look-up table

Given an integer input x (from 0 to $2^{64}-1$), this method follows three steps:

- find the MSB position of x with a priority encoder (PE_{out})
- compute the 'rest' (r)
- rest is converted to the fraction part of the logarithm with a look-up table (LUT)

Once the fractional width is selected the LUT output can be computed; greater the fractional width grater the precision of the results, but higher will be the resources requirement.

The MSB position is extracted with a priority encoder shown below.

$$PE_{out} = \log_2 x \tag{1}$$

```
always @(posedge clk)   // Priority encoder
begin
    if      (x[63]) prienc <=   6'd63;
            //                  //
            //                  //
    else if (x[2])  prienc <=   6'd2;
    else if (x[1])  prienc <=   6'd1;
    else            prienc <=   6'd0;
end
```

Secondly the rest is computed and its width is matched with the LUT input width.

$$r = x - 2^{PE_{out}} \tag{2}$$

```
r <= ((x-(16'b1<<prienc))<<(IN_W-prienc)>>IN_W-F_W));
```

Finally this rest is injected to the precomputed LUT and the fractional part is found.

$$frac = LUT(r) \tag{3}$$

```
case(r)     // frac = LUT [r]
    10'd0:frac  <= 10'd0;
    10'd1:frac  <= 10'd1;
        //      //
        //      //
    10'd1022:frac   <= 10'd1023;
    10'd1023:frac   <= 10'd1023;
endcase
```

Finally the result is obtained with a simple sum.

$$\log_2 x = PE_{out} + \frac{frac}{2^{frac\_width}} \tag{4}$$

```
y <= (prienc<<F_W) + frac;
```

1.2 Taylor expansion

This method follow the same first step of the previous one, but the fractional part is computed with a Taylor expansion.

$$\begin{aligned} \log_2(x) &= \log_2\left(\frac{x \cdot 2^{PE_{out}}}{2^{PE_{out}}}\right) \\ &= PE_{out} + \log_2\left(\frac{x}{2^{PE_{out}}}\right) \\ &= PE_{out} + \log_2(1+r) \end{aligned} \tag{5}$$

Where r is

$$r = \frac{x - 2^{PE_{out}}}{2^{PE_{out}}} \in [0, 1[\tag{6}$$

Now the second logarithm is computed with the following expansion.

$$\log_2(1+r) = \frac{1}{\ln(2)}\left(r - \frac{r^2}{2} + \frac{r^3}{3} + \cdots\right) \tag{7}$$

```
localparam [9:0] i_ln2    = 10'd739;  // 1/(ln2) 10_fix_9
localparam [9:0] i_ln2_2  = 10'd369;  // 1/(2*ln2)
localparam [9:0] i_ln2_3  = 10'd246;  // 1/(3*ln2)

I_order_1   <= ((x-(64'b1<<prienc))<<(IN_W-prienc))>>(IN_W-F_W);

I_order_2   <= I_order_1;
II_order_2  <= I_order_1*I_order_1;

I_order_3   <= I_order_2;
II_order_3  <= II_order_2[2*F_W-1:F_W];
III_order_3 <= II_order_2[2*F_W-1:F_W]*I_order_2;

y_1  <= prienc<<(F_W+9);
y_2  <= y_1 + I_order_1[2*F_W-1:F_W]  *i_ln2  ;
y_3  <= y_2 - II_order_2[2*F_W-1:F_W] *i_ln2_2;
y_4  <= y_3 + III_order_3[2*F_W-1:F_W]*i_ln2_3;
```

Each correction order implies the introduction of two DSPs (multipliers) and they are limited in number, for example in the PYNQ-Z2 board (Zynq7020) 120 of them are present.

2 Hardware utilization

It will be utilized a PYNQ-Z2 board for testing purposes. A DMA will be employed to transfer the data from the PS (Processing System) to the PL (Programmable Logic) and viceversa. The PL log result will be compared with the numpy PS result.

In this example a 64 bits integer input will be fed in to the two module, the result is in fixed point format with 10 fractional bits (16_fix_10).

Method	FF	LUT	BRAM	DSP	Delay [C.C.]
Look-up table	122	208	0.5	0	3
Taylor expansion	106	202	0	5	4

As it clearly visible both methods use roughly the same amount of flip-flops and look-up tables, but the LUT method uses a BRAM block (larger the fractional width is, larger the RAM) and the expansion method uses DSPs. The first method has a fixed delay of a 3 clock cycles [1], instead the second method produces a delay related to the order correction [2].

3 PYNQ-Z2 Deploying

3.1 Look-up table

In the FIG:1 is shown a 10 bits LUT response. This output is precomputed via a python script that makes use of the np.log2 and np.round numpy functions.

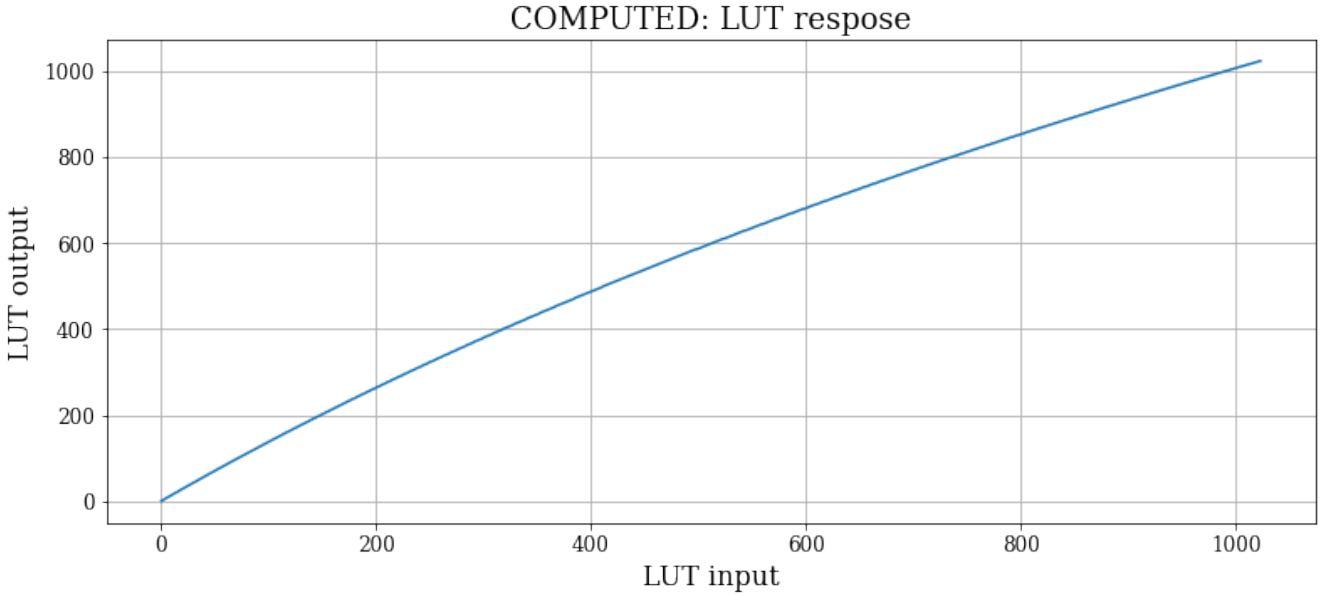
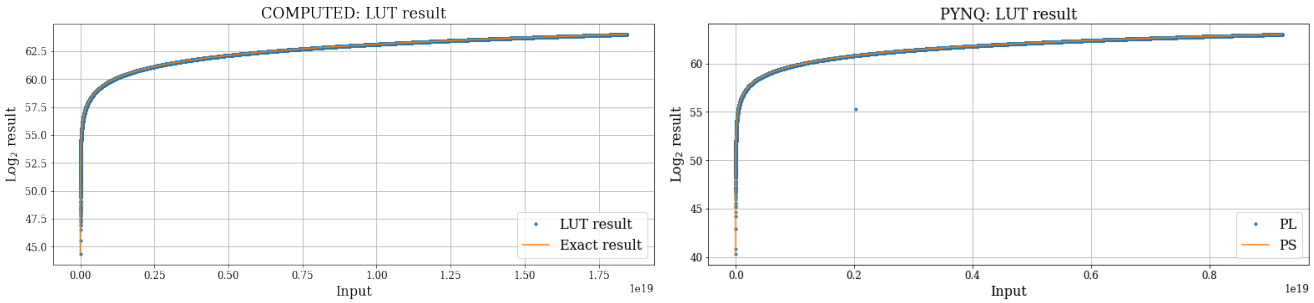


Figure 1: LUT employed for the fractional part. In this example a 10 bits wide LUT is employed, its value are generated by a python script.

[1] LUT input evaluation, LUT output and adder

[2] rest evaluation and one clock cycle per order correction

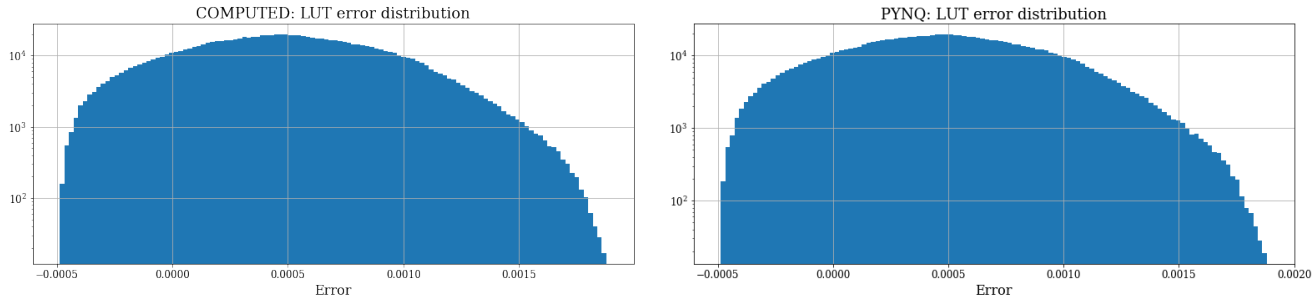
In FIG:2 is shown the output of 2^{20} random integer samples computed with np.log function and LUT method.



(a) Expected module response (b) Hardware implementation response

Figure 2: Comparisons between expected and actual "LOG LUT" module response with the exact logarithm value, in both cases 2^{20} random input samples are considered.

The FIG:3 highlight the error distribution introduced by this method, the ladder depends on the rounding method used and the width selected.



(a) Expected module response (b) Hardware implementation response

Figure 3: Error distributions.

3.2 Taylor expansion

In FIG:4 show the taylor expansion response in function of the input rest, it is shown the III order approximation, the average of the III order and II order and the exact response.

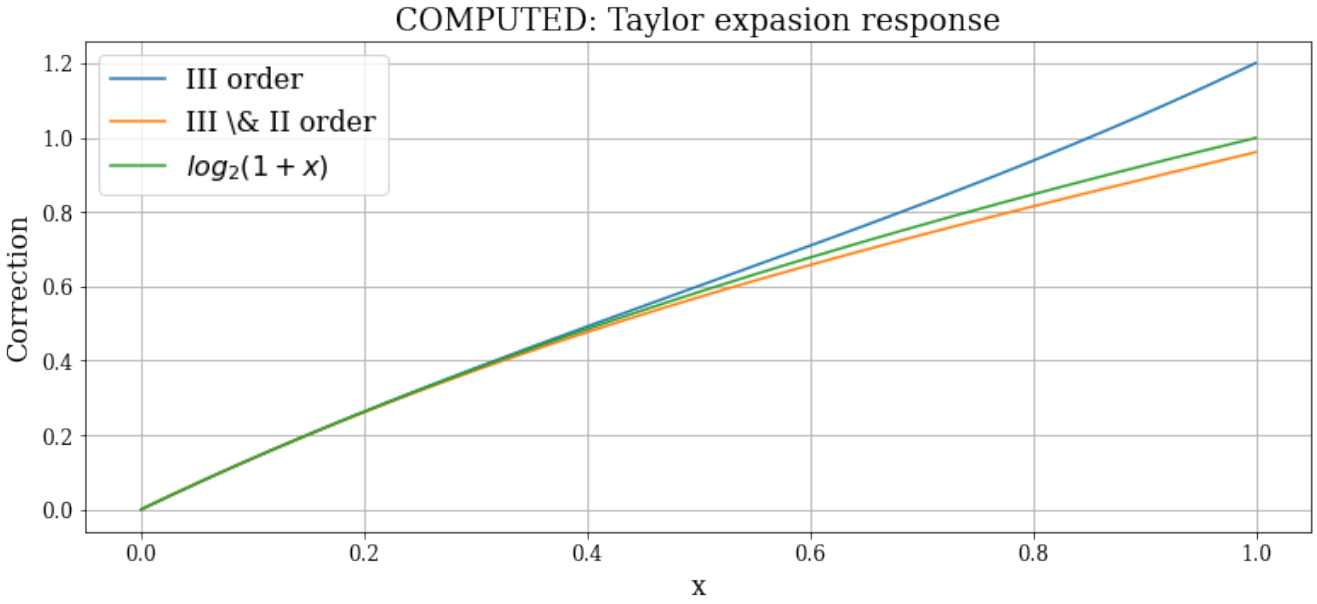
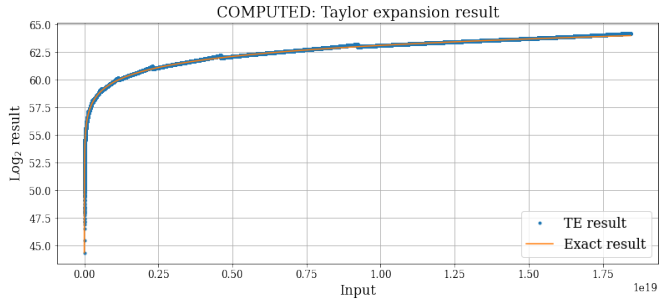
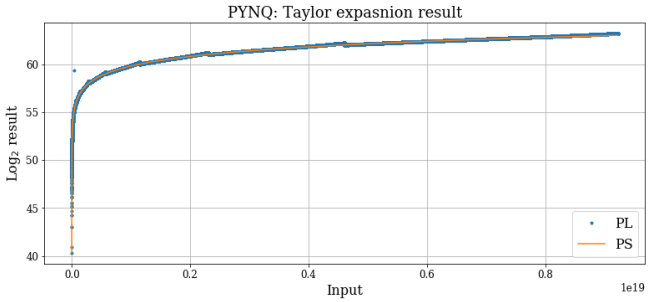


Figure 4: Fractional part estimation with Taylor expansion.

Similar to the previous method in FIG:5 is shown the response of the taylor expansion method, the various steps are caused by the PE_{out} change.



(a) Expected module response



(b) Hardware implementation response

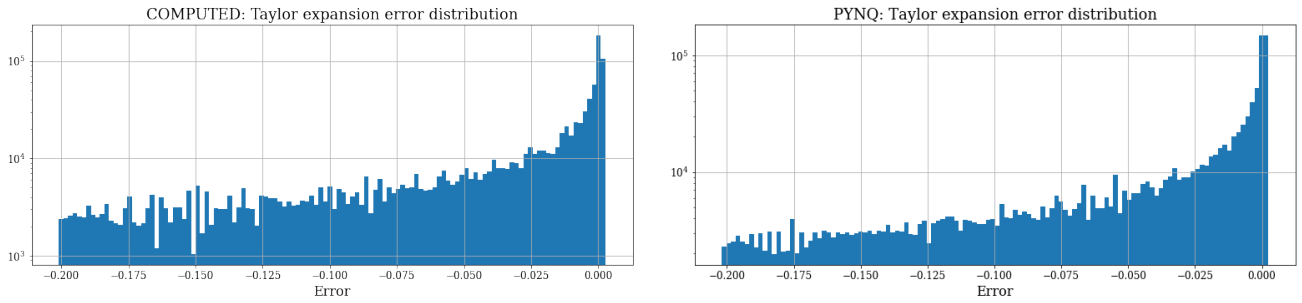
Figure 5: Comparisons between expected and actual "LOG TE" module response with the exact logarithm value, in both cases 2^{20} random input samples are considered.

The error distribution in FIG:6 is only negative as the III order expansion is always grater than the log result (II order is always positive instead).

3.3 Execution time

Here a simple test is produced to evaluate the time performance of the modules.

For comparison the PS with the ARM clocked at 650 MHz produced a computation time per sample of 345 ns.



(a) Expected module response

(b) Hardware implementation response

Figure 6: Error distributions.

Method	10MHz	25MHz	50MHz	100MHz	166.67MHz	200MHz	250MHz
Look-up table	101ns	41ns	21ns	11ns	7ns	7ns	FAILED
Taylor expansion	101ns	41ns	21ns	11ns	7ns	7ns	FAILED

4 Conclusions

The errors introduced by the two methods are now presented. The LUT method introduces an

	COMPUTED		PYNQ implementation	
Method	Mean error	STD	Mean error	STD
Look-up table	4.9×10^{-4}	4.1×10^{-4}	4.9×10^{-4}	4.1×10^{-4}
Taylor expansion	-4.3×10^{-2}	5.5×10^{-2}	-4.3×10^{-2}	5.6×10^{-2}

error two order of magnitude smaller than the taylor expansion method but it requires a script to generate the LUT output, this means that it is not possible to change its parameter in the vivado framework (input width, fractional width and so on), it also uses BRAM blocks. The second method has the only advantage to be customizable in vivado, but it has worse performance in terms of precision and delay and it employs DSPs which are limited in number.