

A Serial Logarithmic Number System ALU

Mark G. Arnold
Lehigh University
marnold@eecs.lehigh.edu

Panagiotis D. Vouzis
Lehigh University
vouzis@lehigh.edu

Abstract

Serial arithmetic uses less hardware than parallel arithmetic. Serial floating point (FP) is slower than parallel FP. The Logarithmic Number System (LNS) simplifies operations, but a fast serial implementation of LNS has never been proposed previously. This paper presents a fast bit-serial LNS that combines a novel serial implementation of Mitchell's method and a new error correction method that is compatible with least-significant-bit-first serial arithmetic. Keywords: Serial Arithmetic, Logarithmic Number System.

1. Introduction

Bit-serial arithmetic [6] trades computational speed for reduced area by processing one bit at a time. This paper looks at resource-constrained systems that process the logarithms of the numbers (rather than the numbers themselves) using bit-serial arithmetic. Although bit-parallel logarithmic arithmetic has been studied quite extensively, only one previous work [7] has ever considered using bit-serial logarithmic arithmetic, and that implementation, although very accurate, is extremely slow. In contrast, the design approach considered here strives for high-speed albeit low-precision bit-serial logarithmic arithmetic suitable for control applications [3].

1.1. Logarithmic Number System

The format of the Logarithmic Number System (LNS) [9] has a base- b (usually, $b = 2$) logarithm (consisting of signed integer and fractional parts) to represent the absolute value of a real number and often an additional sign bit to allow for that real to be negative. To compute LNS sums, both the log and antilog are combined into a single special function, known as $s_b(z) = \log_b(1 + b^z)$. Starting with $x = \log_b(X)$ and $y = \log_b(Y)$ already in LNS format, the result, $t = \log_b(X + Y)$ is computed as $t = \max(x, y) + s_b(-|x - y|)$, which is justified by the fact

that $T = X + Y = \max(X, Y)(\min(X, Y)/\max(X, Y) + 1)$. There is a similar function, d_b , to compute the logarithm of differences not considered here. This paper starts with a bit-parallel algorithm for LNS addition, known as Low-Precision Very-Insignificant-Power (LPVIP) [2], that is based on Mitchell's method [8].

1.2. Mitchell's Method

Mitchell [8] described an efficient approximation for the base-2 logarithm and antilogarithm without tables or iteration. In this paper, we will not actually use a hardware implementation of Mitchell's logarithm; however, this approximation, $\log_2(X) \approx M(X)$, is useful in the analysis of our proposed method. For $X \leq 1.0$, $M(X) \leq 0.0$ is:

$$M(X) = 2^n \cdot X - n - 1, 2^{-n} \leq X \leq 2^{-n+1}. \quad (1)$$

The other method proposed by Mitchell, which is the basis for the serial circuit in this paper, is Mitchell's antilogarithm:

$$2^x \approx M^{-1}(x) = 2^{\text{int}(x)} \cdot (\text{frac}(x) + 1). \quad (2)$$

It is important to note $M(M^{-1}(x)) = x$ exactly.

2. Serial Mitchell's Method

Parallel arithmetic usually only requires combinational logic, whereas serial arithmetic demands state machines that keep track of timing information. We will assume that the Mitchell circuit can operate in a pipelined fashion. It can output serially the result of the last computation as it is accepting the input for the next computation.

Fig. 1 shows a block diagram for a possible approach to implement Mitchell's antilogarithm in a serial fashion, with globally-generated `clock` and `sync` and a serial-data input stream (`in`). Each clock cycle defines the time to transmit one bit. Because Mitchell's method involves integer and fractional parts, the global `sync` signal gives timing information that says when the serial-data stream belongs to the

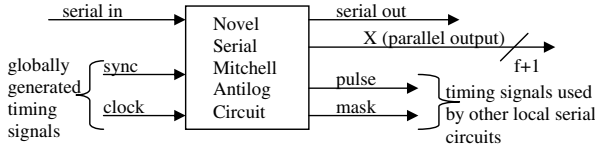


Figure 1. Mitchell's antilogarithm circuit.

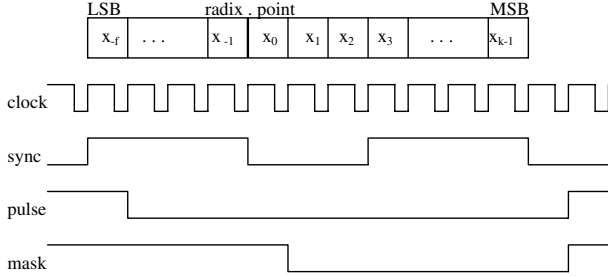


Figure 2. Timing.

integer and fractional parts. We further assume that the integer part of the input allows for sufficiently negative numbers, $x < -f$, such that $M^{-1}(x) = 0$ due to truncation to f bits. This means the `sync` pulse also needs to encode the timing for the most significant integer bits.

Fig. 2 shows the relationship of the timing signals to the serial-data stream. The input consists of $k + f$ bits, starting with the least significant bit, x_{-f} . During the f clock cycles that the fractional part is transmitted, `sync` is asserted. When the first bit of the integer part, x_0 , is transmitted, `sync` becomes zero. During the time that the first $\lceil \log_2(f) \rceil$ bits are transmitted, `sync` remains zero. Fig. 2 assumes $f \leq 8$ so that `sync` is asserted again when x_3 is transmitted. x_3 , or the following bits, could trigger the clearing of the shift register in the event that $M^{-1}(x) = 0$ due to truncation to f bits.

This circuit assumes the input is never positive ($x \leq 0$) which in turn means the output, $X = M^{-1}(x) \leq 1.0$, is positive but never larger than unity. The timing for potentially non-zero serial output occurs during the $f + 1$ clock cycles when input bits $x_{-f} \dots x_0$ are accepted. As illustrated in Fig. 2, this timing coincides with the signal `mask`. The serial output is zero except during the time indicated by `mask`.

The main Mitchell circuit consists of a Verilog `reg x[F:0]` and a state machine that operates in four phases, corresponding to two complete cycles of the `sync` signal. In the first phase, corresponding to the first time `sync` is one, fractional bits, x_i for $-f \leq i \leq -1$, are shifted into the Verilog `reg x[F:0]`. In the second phase, corresponding to the first time `sync` is zero, low-order integer bits, x_i for $0 \leq i \leq \lceil \log_2(f) \rceil$, indicate whether or not to shift the Verilog `reg x[F:0]` right by 2^i . In the third phase, corresponding to the second time `sync` is one, high-order

Table 1. Serial-Mitchell unit synthesis ($k = 6$).

f	CLB	LUT	FF	Freq. (MHz)	Equiv. MHz
5	21	33	18	313	28.4
6	22	35	19	298	24.8
7	22	36	20	320	24.6
8	23	37	21	301	21.5

integer bits, x_i for $\lceil \log_2(f) \rceil < i \leq k$, indicate whether or not to clear the Verilog `reg x[F:0]` as explained earlier. In the final phase, corresponding to the second time `sync` is zero, the input bits are ignored; and the Mitchell output, X , is held in the Verilog `reg x[F:0]`. Because of the pipelined nature of this circuit, when these four phases repeat, the current serial output, X_i , will be produced while simultaneously the new serial input, x_i , is received.

For moderately sized f , a compromise between barrel- and single-bit-shifters is possible, in which the shift register allows variable shifts for short distances, and the state machine uses multiple cycles for longer distances. This compromise achieves both low cost and one-output-bit per cycle because the time for high-order integer bits in the third phase can be used for the multiple-cycle shifts, provided that the state machine simultaneously monitors whether the shift register needs to be cleared.

Because the input, x , is a negative two's-complement number, the shift amount specified by (2) ranges from -1 to $-f$. It is more convenient to perform the shifting in the range from 0 to $-f + 1$, and then shift one additional time at the completion of the process to achieve the shift given in (2). When the input, x_i , is zero, shift by 2^i ; otherwise, leave the shift register alone.

Fig. 3 shows an Algorithmic State Machine (ASM) chart that defines the four-phase Mitchell algorithm described above. In ASM chart notation [1], each state begins with a rectangle. A state continues processing in parallel through decisions (diamonds) and Mealy commands (ovals) until a new state is entered at the next clock edge. Inside the ovals is the Verilog non-blocking assignment, which takes effect in the next state. Concatenation, shown as a list in curly brackets, joins bits together; bit selection, shown as a colon in brackets, indicates the left and right positions of the bits to be kept.

The state machine of Fig. 3 was coded in implicit Verilog and transformed into a one-hot implementation [1] by the Verilog Implicit To One-hot (VITO) tool (www.verilog.vito.com). Table 1 presents synthesis results for a Xilinx Virtex-IV FPGA using Xilinx Webpack 9.1i for various f . One-hot encoding makes the machine fast, but costs eight flip flops. Together with $f + 1$ flip flops for `x[F:0]` and four flip flops for `zero, oldzero`,

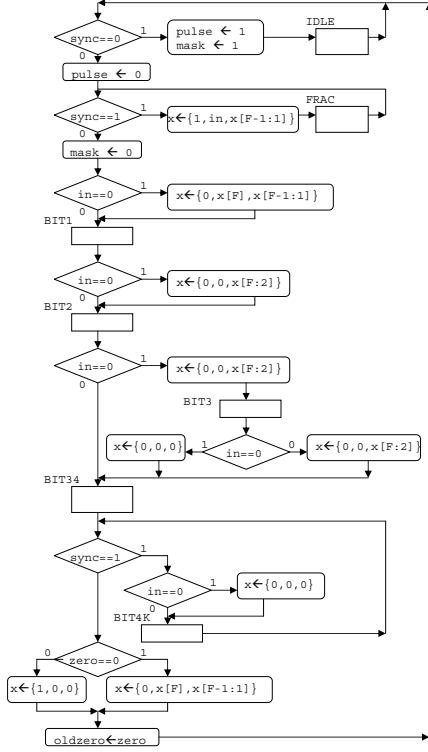


Figure 3. ASM Chart.

mask and pulse, the flip flops total $f+13$. The equivalent parallel-arithmetic frequency shown is the serial-arithmetic frequency divided by $f+k$. Serial-arithmetic clock frequencies of nearly 300MHz are equivalent to parallel-arithmetic clock frequencies around 30-20Mhz for the f shown.

3. LPVIP s_b

The LPVIP approach uses the fact that

$$s_2(z) \approx 2^z \quad (3)$$

for $z \leq 0$. It was shown in [2] that this approximation is about as accurate as the Mitchell's antilogarithm method. Combining (3) with (2), we have

$$s_2(z) \approx 2^{\text{int}(z)} \cdot (\text{frac}(z) + 1). \quad (4)$$

The error can be expressed as a function of z

$$\hat{E}_s(z) = s_2(z) - M^{-1}(z), \quad (5)$$

as was described in [2]; however, defining it in this way requires the entire z to be saved, which would add to the cost of bit-serial implementation of error correction. This paper will present a novel technique, to reduce this error at the cost of a small ROM, that yields

$$\begin{aligned} \bar{E}_s(X) &= \hat{E}_s(M(X)) = s_2(M(X)) - M^{-1}(M(X)) \\ &= s_2(M(X)) - X. \end{aligned} \quad (6)$$

Combining (6) with (1), we have the desired LPVIP correction:

$$\bar{E}_s(X) = s_2(2^n \cdot X - n - 1) - X, 2^{-n} \leq X \leq 2^{-n+1}.$$

Fig. 4(a) shows a plot of $\bar{E}_s(X)$.

4. Interpolation of $\bar{E}(X)$

Prior methods for LNS addition using interpolation [5] have mostly taken z as input and produced $s_b(z)$ as interpolated output. Combining LPVIP with interpolation requires a different approach as the output of interpolation is not $s_b(z)$, but rather is $\hat{E}_s(z)$ [2]. This paper suggests a novel variation in which the input is not z , but instead is X , and the output is $\bar{E}_s(X)$. Unlike [2], the approach proposed here generalizes to arbitrary precision at the cost of increased memory. We approximate the error correction as:

$$\bar{E}_s(X_H + X_L) \approx \bar{E}_s(X_H) + \bar{E}'_s(X_H + \Delta/2) \cdot X_L, \quad (7)$$

where X_H and X_L are the high-order and low-order parts of the parallel output, X , from Mitchell's method, X_H is a multiple of $\Delta = 2^{-W}$ and $X_L < \Delta$, where 2^W is the number of words in the \bar{E}_s and \bar{E}'_s memories. $\bar{E}_s(X_H + X_L)$ can be output in $f-4$ bits. The worst-case error in first-order interpolation of a continuous function is proportional to the second derivative:

$$\bar{E}_s''(X) = 2^{2n} \cdot s_2''(2^n \cdot X - n - 1), 2^{-n} < X < 2^{-n+1}. \quad (8)$$

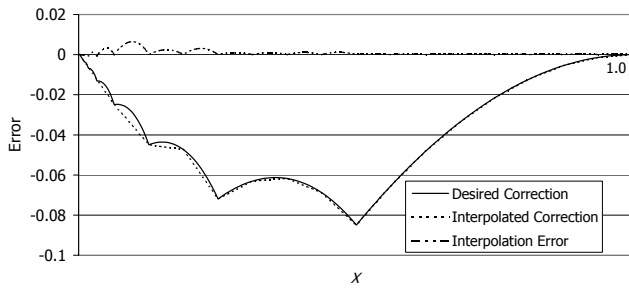
By looking at the value of the second derivative at the right endpoints, $X_{\text{right}} = 2^{-n+1}$, in each of the n cases,

$$\begin{aligned} E_{\text{max}}''(n) &= 2^{2n} \cdot s_2''(2^n \cdot 2^{-n+1} - n - 1) \\ &= 2^{2n} \cdot s_2''(1 - n), \end{aligned} \quad (9)$$

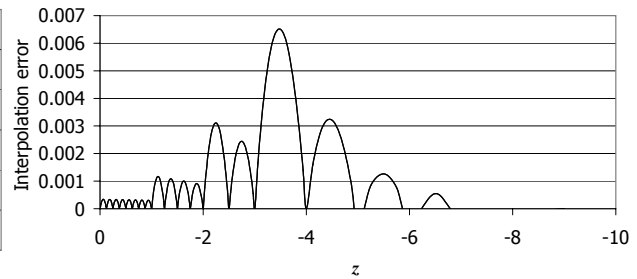
we can find the n that maximizes (9), subject to the constraint $n \leq W$, which keeps the interpolation line within a single continuous case where the derivative will be meaningful for estimating the interpolation error. The result of (9) at that n bounds the error of linear interpolation of $\bar{E}_s(X)$ as $E_{\text{max}}''(n)\Delta^2/8$. Some algebra shows the bound on the interpolation error is

$$E_{\text{max}}''\Delta^2/8 \approx 2^{-W-3}. \quad (10)$$

Fig. 4 gives an example of applying the proposed correction method with interpolation for $f = 8$ using a 4×16 memory for $\bar{E}_s(X_H)$ and a 2×16 memory for $\bar{E}'_s(X_H)$. This requires only three CLBs plus a small (2×4) parallel multiplier for X_L . Fig. 4(a) depicts $\bar{E}_s(X)$ (desired correction), its interpolation, and the difference between the desired and interpolated correction, which as predicted by (10) for this $W = 4$ case is (at worst) on the order of 2^{-7} . Fig. 4(b) depicts end-user perception of this approximation by looking at this difference as a function of z , i.e., the difference between the desired $\hat{E}_s(z)$ minus its interpolated value.



(a) The accuracy of the interpolation of the correction function.



(b) Interpolation error.

Figure 4. The error behavior of the correction method.

5. Serial LNS Adder

Fig. 5 shows a bit-serial LNS ALU composed of two Mitchell units, a one-bit mux, correction logic, two $(k + f)$ -bit First-In-First-Out (FIFO) serial registers (to hold x and y for pipelined operation), a bit-serial adder and two bit-serial subtractors (forming z and $-z$ in parallel). Because this circuit uses least-significant-bit-first formatting, it will not be known which of these two logarithms is negative (as required by Mitchell's method) until the last cycle. To cope with this, two instances of the Mitchell's method described earlier are provided. One of these will produce $X = 0$; the other will produce $X = M^{-1}(z)$. ORing these two $f + 1$ -bit parallel outputs (with $f + 1$ OR gates) is guaranteed to produce $X = M^{-1}(z)$, which can then produce the correction $\bar{E}_s(X)$ described earlier. At the completion of the bit-serial subtract, the status bit for $x < y$ can be latched to select the proper bit stream for $\max(x, y)$, which is added serially to the other terms to produce the LNS result.

6. Conclusion

This paper has described LNS summation using the LPVIP approximation of s_b with a novel seven-state serial implementation of Mitchell's method, needing only global clock and sync signals, which simplify routing within an FPGA. The method is compatible with pipelining, and able to operate at nearly 300MHz (equivalent to around 20MHz for parallel LNS) on a Virtex FPGA. We have proposed a novel correction for obtaining s_b using interpolation based on the Mitchell approximation as input (rather than on the original argument as input) which makes the unit accurate enough for the control application we have in mind [3]. By trading speed for reduced area, serial LNS opens up sophisticated algorithms demanding difficult real arithmetic on limited-resource systems such as microrobots [4].

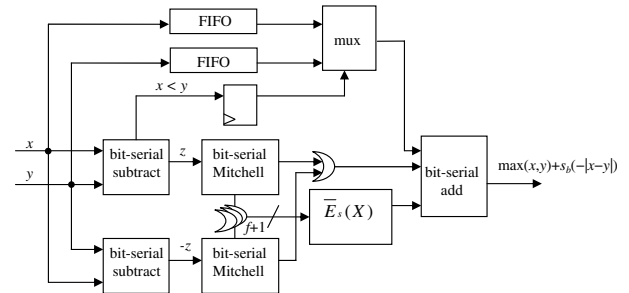


Figure 5. The architecture of a Mitchell bit-serial LNS adder.

References

- [1] M. G. Arnold. *Verilog Digital Computer Design: Algorithms into Hardware*. PTR Prentice Hall, NJ, 1999.
- [2] M. G. Arnold. LPVIP: A Low-power ROM-Less ALU for Low-Precision LNS. In *Proc. of the 14th Int. Workshop on Power and Timing Modeling, Optimization and Simulation LNCS 3254*, pages 675–684, Santorini, Greece, Sept. 2004.
- [3] L. G. Bleris, J. G. Garcia, M. V. Kothare, and M. G. Arnold. Towards Embedded Model Predictive Control for System-on-a-Chip Applications. *Journal of Process Control*, 16(4):255–264, March 2006.
- [4] D. Coleman, J. Spletzer, and M. G. Arnold. Target-Logic Circuits Built with Holonomic Field Programmable Robot Arrays. In *Proc. of the Work-in-Progress Session of 32nd EuroMicro Conference on Software Engineering and Advanced Applications*, pages 53–54, Cavtat, Croatia, Sept. 2006.
- [5] S. Collange, F. de Dinechin, and J. Detrey. Floating Point or LNS: Choosing the Right Arithmetic on an Application Basis. In *Proc. of the 9th EuroMicro Digital System Design*, pages 197–203, Dubrovnik, Croatia, 30 Aug.–1 Sept. 2006.
- [6] P. Jenne and M. A. Viredaz. Bit-Serial Multipliers and Squarers. *IEEE Trans. on Computers*, 43(12):1445–1450, 1994.
- [7] P. P. Li. The Serial Log Machine. Technical Report 4517, Cal. Inst. of Tech, CS Dept., May 30 1981.
- [8] J. N. Mitchell. Computer Multiplication and Division Using Binary Logarithms. *IEEE Trans. on Electronic Computers*, EC-11:512–517, Aug. 1962.
- [9] E. E. Swartzlander and A. G. Alexopoulos. The Sign/Logarithm Number System. *IEEE Trans. on Computers*, 24(12):1238–1242, Dec. 1975.