An Improved Micro-Architecture for Function Approximation Using Piecewise Quadratic Interpolation*

Shai Erez, Guy Even
School of Electrical Engineering, Tel-Aviv University
shaiere@gmail.com, guy@eng.tau.ac.il

November 10, 2008

Abstract

We present a new micro-architecture for evaluating functions based on piecewise quadratic interpolation. The micro-architecture consists mainly of a look-up table and two multiply-accumulate units. Previous micro-architectures based on piecewise quadratic interpolation have been shown to be efficient for small precision (e.g., single precision) computations. Moreover, they are as fast as piecewise linear interpolation while requiring smaller tables. Our main contribution is in circumventing the need for the additional squaring unit that appears in previous micro-architectures.

Based on the proposed micro-architecture, we present a detailed design of single precision reciprocal approximation $(1/x)$. Our design is based on two multiply-accumulate units that contain truncated Booth radix 4 multipliers. The number of partial products in this design is reduced by over 20% compared to previous designs using quadratic interpolation. The latency of this design is roughly the delay of 19 full-adder gates, and it can be easily pipelined into two stages each with a delay of 10 full-adder gates.

1 Introduction

Many applications, such as computer graphics and DSP require single-precision approximation of basic functions such as $1/x$, $\sqrt{x}$, $1/\sqrt{x}$, 2^x, $\log_2 x$, and trigonometric functions. According to several recent papers [LCLV08, Mul03, NSB07, OS05, POMB05, WIS05], approximation by quadratic interpolation can be computed within a single clock cycle[1]. Such designs require small look-up tables and a small amount of logic.

A block diagram of a typical piecewise quadratic interpolator is depicted in Fig. 1. The input X is partitioned into two parts; the upper bits are denoted by X_1 and the lower bits are denoted by X_2. The coefficients a_0, a_1, a_2 of the polynomial $p(X_2) = a_0 + a_1 X_2 + a_2 X_2^2$ are read from a look-up table indexed by the upper bits X_1. The circuit evaluates an approximation of $p(X_2)$. Note that the range of inputs is partitioned into 2^m subintervals where m denotes the length of X_1. Within each subinterval, the function is approximated by a quadratic polynomial, hence the term piecewise quadratic interpolation.

The evaluation of the quadratic polynomial takes place by one squaring $(Z \leftarrow X_2^2)$, and two multiplications ($a_2 Z$ and $a_1 X_2$). The latency of the squaring unit is smaller than that of the lookup and hence, roughly speaking, the latency is the sum of the delay of the look-up table and the delay of a multiplier. In addition, symmetry and truncation are employed to reduce the area of the squaring unit. In Walters and Schulte [WIS05], the issue of truncating the multipliers is explored with the aim of reducing the amount

*An preliminary version of this paper appears in ICCD 2008 IEEE INTERNATIONAL CONFERENCE ON COMPUTER DESIGN.

[1]This single cycle does not include range reductions and unpacking/packing of floating-point representation.

of logic required for squaring and multiplications. In Piñeiro et al. [POMB05] a systematic method is presented for computing the coefficients of the quadratic polynomial. In addition, in [POMB05] a unified micro-architecture is presented for evaluating multiple single-precision functions.

In this paper we suggest to evaluate the quadratic polynomial using Horner's method that requires only two multiplications. Namely, $p(X_2) = a_0 + X_2 \cdot (a_1 + a_2 \cdot X_2)$. A block diagram of the suggested micro-architecture is depicted in Fig. 2. The aim of the proposed micro-architecture is to reduce the amount of logic so as to reduce area and power consumption. The design has a higher latency and is suited for situations where the clock period allows for the extra latency. Alternatively, the micro-architecture is well suited for pipelining. We suggest how to pipeline the micro-architecture into two stages, each stage with a delay of roughly 10 full-adders.

In our detailed implementation, we use Booth radix 4 multipliers with truncation. In [POMB05] Booth radix 4 multipliers are used but without truncation. We conjecture that truncation was not used because in addition to truncating partial products, it also truncates the increment introduced by negative Booth digits. The error caused by truncating these increments is hard to bound analytically. We show by exhaustive testing that this error has a minor effect on the overall error.

For the purpose of simplicity we demonstrate the proposed micro-architecture with a detailed description for single precision reciprocal approximation ($1/x$). The argument X is in the range $[1,2)$, and is represented by 23 bits to the right of the binary point. The output is also represented in the same way. This implies that unless $X = 1$, we actually compute $2/X$, i.e., the reciprocal is normalized to the range $[1,2)$. Hence, the maximum allowed error is 2^{-24}.

2 Micro-Architecture

A detailed diagram of the proposed micro-architecture for single precision reciprocal approximation is depicted in Fig. 3. The lengths of the coefficients are 10, 18, and 26 bits. Note that the truncation boxes are trivial (i.e., zero cost and zero delay); they are depicted to emphasize the fact that bits in lower positions are discarded. The output $\tilde{p}[1:24]$ satisfies $|1/x - \tilde{p}[1:24]| < 2^{-24}$. The extra precision of one bit is required due to the normalization shift that guarantees that $2/x \in (1,2)$.

3 Design

In this section we deal with three issues. First, we bound the error introduced by the truncated multipliers. The truncation positions of both multipliers are determined by this error analysis. Second, we present the method used for computing the coefficients that are stored in the look-up tables. This method is based on [POMB05]. Finally, the bias correction is computed by exhaustive search. The bias correction consists of two parts: the most significant part is added to the coefficient a_0, and the least significant part (4 bits) is a constant that is hardwired to the addition tree of the multiply-accumulate (MAC) unit MAC-1.

We use the following notation for truncation and round-up. Let $\lfloor z \rfloor_k$ and $\lceil z \rceil_k$ denote the round-down and round-up of a number z to the closest multiple of 2^{-k}.

3.1 Truncation: analytic bounds

In this section we describe simple bounds on the errors introduced by truncations in multipliers without Booth recoding. These bounds are subtracted from the required maximum error 2^{-24} to determine the allowed error caused by the quadratic polynomial approximation. The bounds presented here are pretty tight; this means that at best one could hope to further truncate the multipliers by at most one position.

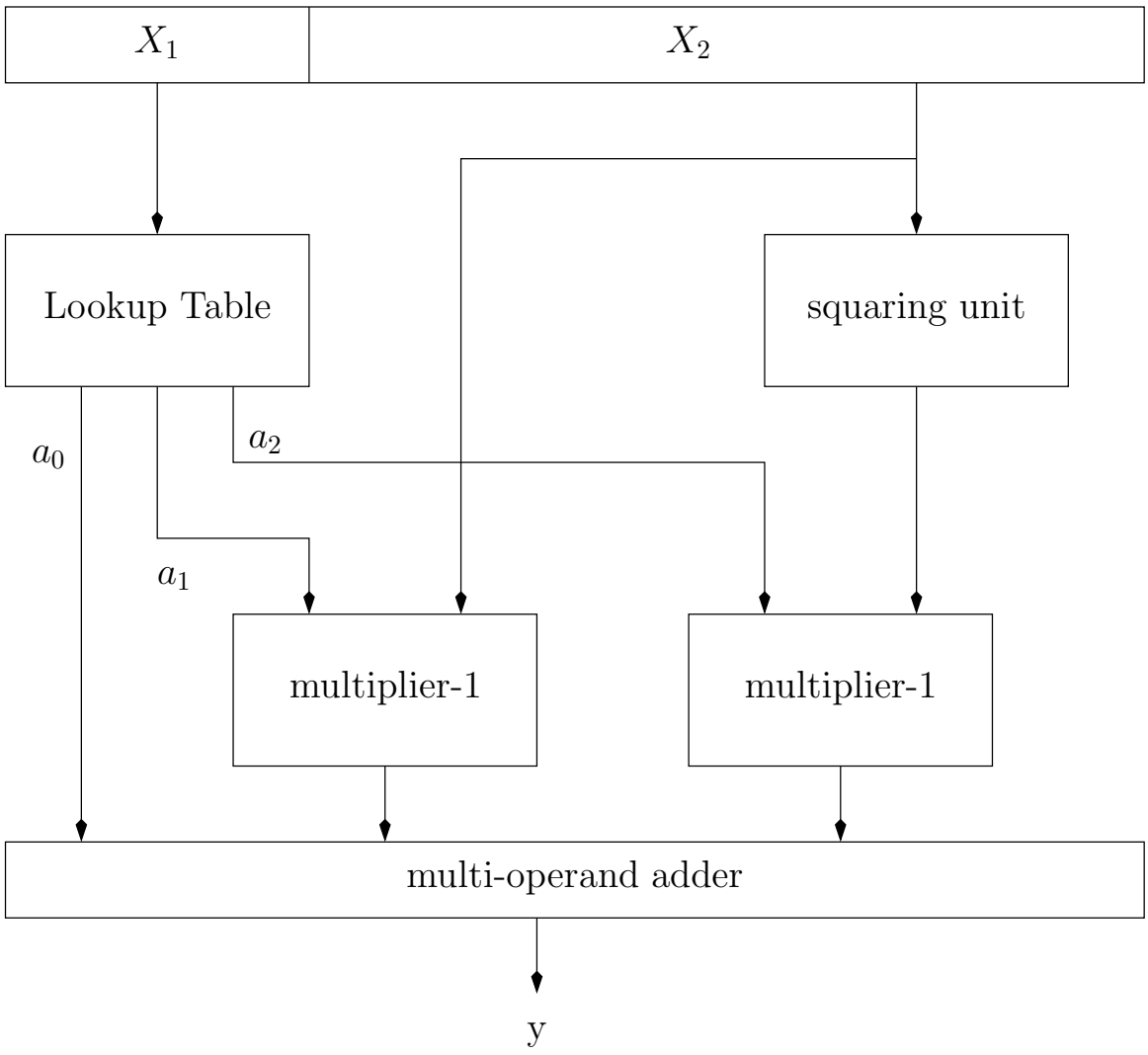

Figure 1: Quadratic interpolator block diagram [WIS05].

Consider a truncated multiplier whose multiplicands are $A[i_1 : j_1]$ and $B[i_2 : j_2]$. The exact product equals $P = \sum_{k=i_1}^{j_1} \sum_{\ell=i_2}^{j_2} A[k] \cdot B[\ell] \cdot 2^{-(k+\ell)}$. If the multiplier is truncated in positions to the right of position t, then it computes the truncated product $P_t = \sum_{k=i_1}^{j_1} \sum_{\ell=i_2}^{\min\{j_2, t+1-k\}} A[k] \cdot B[\ell] \cdot 2^{-(k+\ell)}$. The error introduced by truncation is bounded in the following claim.

Claim 1 $0 \leq P - P_t \leq \sum_{k=t+1}^{j_1+j_2} 2^{-k} \cdot (j_1 + j_2 + 1 - k) = 2^{-t} \cdot (j_1 + j_2 - t - 1) + 2^{-j_1-j_2}$.

To save hardware we evenly divide the error between the two MACs and the quadratic polynomial approximation with bounded length coefficients. Namely, the min-max error of each MAC and the quadratic polynomial is bounded by roughly $2^{-24}/3$.

The lengths of the coefficients for single precision reciprocal approximation reported in [POMB05] are $10, 16, 26$. If we wish to use the same lengths, this implies that in the MAC-2 unit we multiply $X_2[8:23]$ by $A_2[1:10]$. The outcome of MAC-2 is multiplied by $X_2[8:23]$. Hence we require that

$$\log_2 2^{-8} \cdot (2^{-t} \cdot (32 - t) + 2^{-33}) \leq 2^{-24}/3 \approx 2^{-25.584}.$$

Numerical evaluation implies that the multiplier in MAC-2 can be safely truncated in positions to the

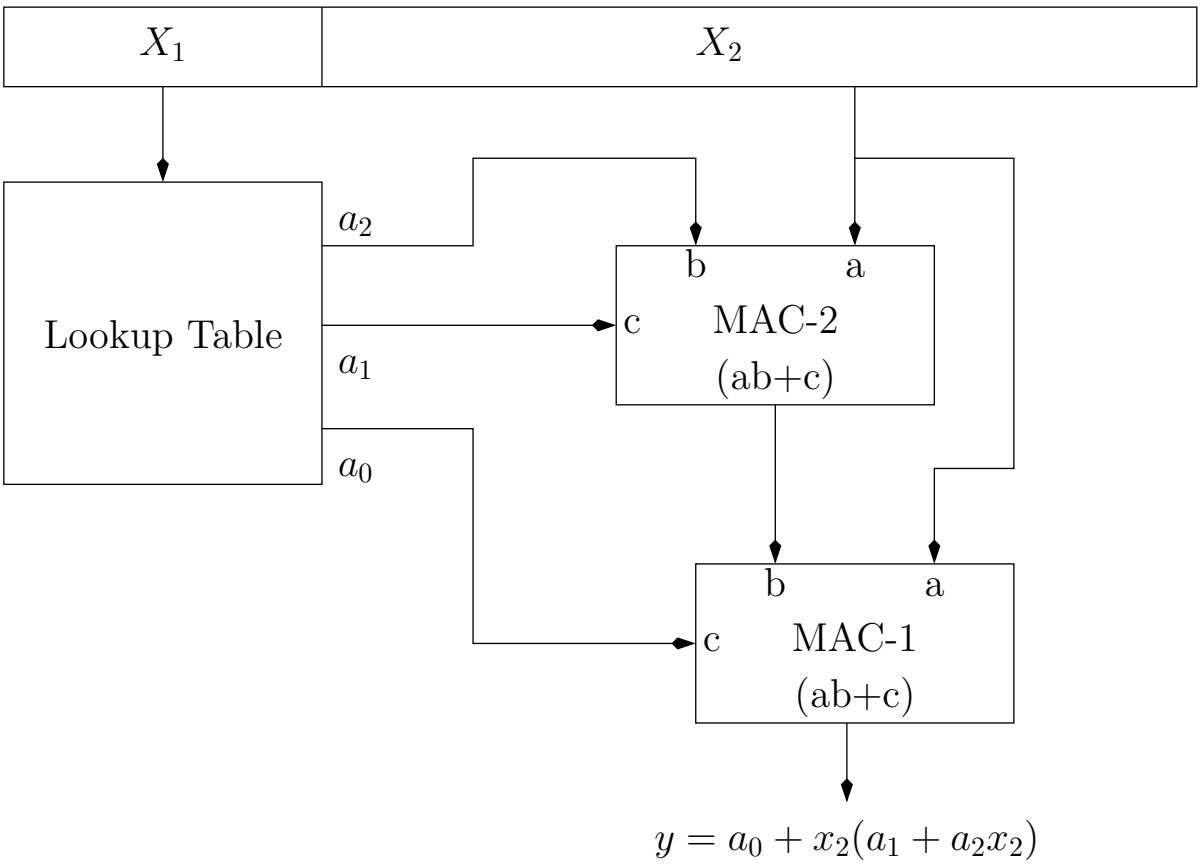

Figure 2: Proposed micro-architecture for quadratic interpolation.

right of $t = 21$. We preferred to reduce the length of the input to MAC-1 to 20 bits, and hence had to move the truncation position in MAC-2 to the right by one position, i.e., $t = 22$.

Similarly, in MAC-1 we multiply $X_2[8:23]$ by $L_{20}[1:20]$. Hence, we require that

$$\log_2 2^{-t} \cdot (42-t) + 2^{-43}) \leq 2^{-24}/3.$$

Numerical evaluation implies that the multiplier in MAC-1 can be safely truncated in positions to the right of $t = 30$.

3.2 Initial computation

The three step process for computing the truncated coefficients is listed as Algorithm 1. The parameters of the algorithm are the coefficient lengths n_0, n_1, n_2 and X_1 (which determines the subinterval). In each phase, the Remez algorithm for min-max approximation is applied [Fra65]. In the first phase (Line 1), a quadratic polynomial $q(x) = q_0 + q_1x + q_2x^2$ that approximates $1/(X_1 + x)$ is computed. To speed the execution of the min-max Remez algorithm, we begin with a quadratic least squares approximation. In Line 2 the coefficient q_1 is truncated in position n_1 to obtain $a_1(X_1)$. The second phase proceeds as follows. Having fixed $a_1(X_1)$, we substitute $z = X_2^2$ to obtain the target function $\frac{1}{X_1+\sqrt{z}} - a_1(X_1) \cdot \sqrt{z}$ that is approximated by a linear function in z. This linear function $\ell(z) = \ell_1 z + \ell_0$ is computed in Line 3. In Line 4 the coefficient ℓ_1 is truncated in position n_2 to obtain $a_2(X_1)$. In the third phase we approximate the error by a constant function simply by taking the average of the maximum error and the minimum error. In Line 5 we compute this average and denote it by *avg*. The value of *avg* is truncated in position n_0 to obtain $a_0(X_1)$. This algorithm is repeated for each of the 128 values of X_1 and determines the contents

of the table (except for the bias correction described below). The error introduced by the quadratic polynomial $a_0(X_1) + a_1(X_1) \cdot X_2 + a_2(X_1) \cdot X_2^2$ is computed by exhaustive simulation. This error should be bounded by roughly $2^{-24}/3$.

Algorithm 1 Compute coefficients a_0, a_1, a_2 with lengths n_0, n_1, n_2 with respect to X_1.

1: $(q_0 + q_1 x + q_2 x^2) \leftarrow \text{remez}_2(\frac{1}{X_1+x})$.
2: $a_1(X_1) \leftarrow \lfloor q_1 \rfloor_{n_1}$.
3: $(\ell_0 + \ell_1 z) \leftarrow \text{remez}_1(\frac{1}{X_1+\sqrt{z}} - a_1(X_1) \cdot \sqrt{z})$.
4: $a_2(X_1) \leftarrow \lfloor \ell_1 \rfloor_{n_2}$.
5:

$$avg \leftarrow \frac{1}{2} \cdot \left(\max_{X_2} \left(\frac{1}{X_1 + X_2} - (a_2(X_1) \cdot X_2^2 + a_1(X_1) \cdot X_2) \right) + \min_{X_2} \left(\frac{1}{X_1 + X_2} - (a_2(X_1) \cdot X_2^2 + a_1(X_1) \cdot X_2) \right) \right).$$

6: $a_0(X_1) \leftarrow \lfloor avg \rfloor_{n_0}$.
7: Return $a_0(X_1), a_1(X_1), a_2(X_2)$.

Figure 4 depicts one iteration of the Remez algorithm. The error associated with the initial least squares approximation is depicted as a solid line. The extreme points of the error are chosen for the first iteration. The error associated with the quadratic function after the first iteration is depicted as a dashed line. Note that the extreme points of the errors are now more "symmetric". The algorithm terminates when the absolute value of the error is the same in all the extreme points.

3.3 Bias Correction - Computation

The use of truncated multipliers and truncation of the outputs of the MAC units introduces errors that can be partly corrected by fine tuning the coefficient a_0. This fine tuning reduces the error by more than 2^{-25}.

Let $p(x)$ denote the quadratic polynomial $p(X_2) = a_0 + a_1 X_2 + a_2 X_2^2$ based on the truncated coefficients (a_0, a_1, a_2) that were computed for the prefix X_1 of X. Let $\tilde{p}(X)$ denote the number computed by the circuit. Let $\tilde{p}_{30}(X)$ denote the number computed by the $2:1$-adder before the truncation to 24 bits (see Fig. 3). Note that $\tilde{p}(X)$ does not equal $p(X)$ due to the truncated multipliers and the truncation of $a_1 + a_2 \cdot X_2$.

The error of the final 24-bit result is bounded by 2^{-24} if

$$\left\lfloor \frac{1}{X} \right\rfloor_{24} \leq \tilde{p}_{30}(X) \leq \left\lceil \frac{1}{X} \right\rceil_{24} + 2^{-24} - 2^{-30}. \tag{1}$$

(Recall that the precision of the MAC-1 unit is 30 bits to the right of the binary point).

For every value of X, define *min-bias*(X) and *max-bias*(X) as follows:

$$\textit{min-bias}(X) \triangleq \left\lfloor \frac{1}{X} \right\rfloor_{24} - \tilde{p}_{30}(X)$$
$$\textit{max-bias}(X) \triangleq \left\lceil \frac{1}{X} \right\rceil_{24} + 2^{-24} - 2^{-30} - \tilde{p}_{30}(X)$$

Hence, if $bias(X) \in [\textit{min-bias}(X), \textit{max-bias}(X)]$, then setting $a_0(X) \leftarrow a_0(X) + bias(X)$ guarantees that $\tilde{p}_{30}(X)$ satisfies Eq. 1. Since we wish to add the same bias to all numbers in the same subinterval, we

need to check, for each X_1, that the intersection $\bigcap_{X_2}[\textit{min-bias}(X_1+X_2), \textit{max-bias}(X_1+X_2)]$ is not empty. If this is the case, then we add a bias to each subinterval that is determined by X_1.

Another problem that we need to address is that one might need more than 26 bits to represent the bias; otherwise, we would need to increase the number of bits stored in the table for $a_0 + \textit{bias}$. To avoid the need to increase the table, we partition the bias into two parts $\textit{bias}[1:26]$ and $\textit{bias}[27:30]$. Our goal is to find a common 4-bit suffix $\textit{bias}[27:30]$ for all values of X. We find such a fixed suffix by exhaustive search, if one exists, and include it in the row of a_0 in the adder tree of MAC-1. A similar technique is briefly outlined in [POMB05].

3.4 Pipelined Implementation

In this section we present a pipelined implementation with two pipeline stages. In each stage, the multiply-accumulate unit uses Booth radix 4 multipliers with truncation. Moreover, in the second stage, the multiplier is input in carry-save representation and Booth recoded without 2:1-addition.

A pipelined implementation is depicted in Figure 5. Truncation is applied both to the partial products and the increments caused by negative Booth digits. Note that the lookup table is now split between two tables. In the first stage, the lookup table outputs the coefficients a_1 and a_2. In the second stage, the lookup table outputs the coefficient a_0. A row of 20×2 flip-flops is used between the two stages for storing the carry-save representation of L_{20}.

The recoding of L_{20} to Booth radix 4 digits follows Daumas and Matula [DM03]. Namely, the recoding QNP is applied to L_{20} (where each recoding Q, N, and P is a row of half-adders with some negations). The output of the QNP recoding is fed to a regular Booth radix 4 recoder.

4 Evaluation and Comparison

4.1 Table size

The length of the coefficients that we able to obtain are $10, 18$ and 26 bits. This is two bits more than the lengths reported in [POMB05]. There are tradeoffs between coefficient lengths and truncation positions. In light of the huge gap between the area of a full-adder and the area of a bit stored in the table (i.e, 35 full-adders per kilo-bit), we preferred less logic over smaller tables.

4.2 Number of partial products

In Table 1 we compare the number of partial products computed in our design with the number of partial products generated in previous micro-architectures [WIS05, POMB05]. The previous designs compute the square $Z \leftarrow X_2^2$ and then perform two multiplications $a_1 \cdot X_2$ and $a_2 \cdot Z$. The products are then added with a_0 to produce the reciprocal approximation. The cost of the squaring unit is relatively small thanks to the symmetry and truncation [WIS05]. We could not reconstruct exactly the number of partial products reported in [WIS05]; our numbers are smaller probably due to additions that take place in their design which we did not include.

4.3 Delay analysis

In this section we overview a delay optimized implementation of the proposed micro-architecture and analyze it delay. The MACs are based on truncated Booth radix 4 multipliers [KS04] to save both cost, power, and delay.

We apply the delay model used in [POMB05] to a delay optimized version of the proposed micro-architecture. According to this model, the basic unit of delay is denoted by τ and it corresponds to the delay of a full-adder. The micro-architecture is optimized as follows:

	WS05 [WIS05]	POMB05[POMB05]	here
Squaring (X_2^2)			
multiplier dimensions	15×15	16×16	none
truncation	12 bits	18 bits	0
#PP's	42	49	0
Mult. $(a_1 \cdot X_2)$			
multiplier dimensions	19×16	16×16	none
truncation	11 bits	0 bits	0
#PP's	238	256	0
Mult. $(a_2 \cdot X_2^2)$			
multiplier dimensions	14×12	14×10	none
truncation	9 bits	0 bits	0
#PP's	123	140	0
Mult. $(a_2 \cdot X_2)$			
multiplier dimensions	none	none	16×10
truncation	0 bits	0 bits	11 bits
#PP's	0	0	95
Mult. $((a_1 + a_2 \cdot X_2) \cdot X_2)$			
multiplier dimensions	none	none	20×16
truncation	0 bits	0 bits	13
#PP's	0	0	229
Total #PPs	403	445	**324**

Table 1: Comparison of the number of partial products.

1. The MAC-2 unit that computes $(a_1 + a_2 \cdot X_2)$ is designed as follows. In parallel to the table look-up, the multiplier X_2 is Booth radix 4 recoded. This reduces the number of rows in the addition tree to $1 + \lceil \frac{16+1}{2} \rceil = 10$ (the additional row is needed to add a_1). The 10 rows are reduced to a carry-save number by a sequence of $4:2$-adder, $3:2$-adder, and $4:2$-adder. Thus the delay associated with this computation is $\max\{t_{table}, t_{recode}\}(3.5\tau) + t_{pp-gen}(1\tau) + 2 \cdot t_{4:2}(3\tau) + t_{3:2}(1\tau) = 8.5\tau$.

2. The term $(a_1 + a_2 \cdot X_2)$ is output in borrow-save representation. We recode it to a Booth-4 representation using the equivalent of 3 half-adders [DM03]. After recoding, truncate the Booth-4 number and consider only the 11 most significant Booth-4 digits. We now compute $a_0 + X_2 \cdot (a_1 + a_2 \cdot X_2)$. Together with a_0, the addition tree in the multiply-accumulate unit has 12 rows. These rows are reduced to two rows by one $3:2$-adder and two $4:2$-adders. Thus the delay associated with this computation is $t_{recode}(1.5\tau) + t_{pp-gen}(1\tau) + 2 \cdot t_{4:2}(3\tau) + t_{3:2}(1\tau) = 6.5\tau$.

We need to take into account the final $2:1$-addition (whose delay is 3τ) and the register setup time (1τ). We conclude that the total delay of the design is 19τ. If a very short clock cycle is required, then our design can be easily pipelined into two stages, each with a delay of 10τ. The delay of 19τ is longer than the delay of 14.5τ reported in [POMB05]. However, the design in [POMB05] is not amenable to pipelining since all the partial products are generated simultaneously and added in a combined addition tree. No delay analysis is provided in [WIS05]; in fact, it is not clear how the intermediate products in [WIS05] are rounded.

5 Further Research

Computations based on lookup tables lead to the question whether one should precompute the table contents (as proposed in this paper) or compute the table entries "on the fly" (i.e., compress the tables). In Figure 10 the graph of the 128 values of the coefficients a_0, a_1, a_2 is depicted .This graph leads to the question whether one could compute the coefficients rather than store them.

The answer to this question depends mainly on the relative cost of read-only memory (ROM) vs. computational circuitry (i.e., multipliers and adders). The cost model used in [POMB05] estimated the area of 1Kbit to be equal to the area of 35 full-adders. In models where ROM requires more area compared to a full-adder, one may want to reduce the table size.

Consider the coefficient a_0 for which we store 26 bits per entry. We anticipate the computing the most significant bits of a_0 is rather easy using even linear interpolation. This means that instead of storing 26 bits for each entry of a_0, we could store (say) the 13 least significant bits and compute the 13 most significant bits using linear interpolation. In fact, the same circuitry (e.g. MAC-2) could be used to compute these bits. Hence, the tradeoff is actually between power and latency. We believe that these questions are of interest especially if the micro-architecture is used for approximating multiple functions.

6 Conclusions

We presented a micro-architecture for single precision approximations of functions. An implementation for approximating $1/x$ was studied in detail. We showed that the micro-architecture saves in the number of partial products. A reduction of at least 20% in the number partial products is achieved compared to previous designs. According to [POMB05], the logic costs roughly 60% of such a design[2]. The increase in the table size is less than 5%. On the other hand, the latency of the our design is 19τ compared to 14.5τ in [POMB05]. However, the design is well suited to pipelining into two stages with a clock period of 10τ.

We anticipate that this design can be useful in situations where the clock period is either at least 19τ (without pipelining) or 10τ (with pipelining). In such cases, the reduced amount of hardware will lead to smaller area and smaller power consumption.

References

[DM03] M. Daumas and D.W. Matula. Further Reducing the Redundancy of a Notation Over a Minimally Redundant Digit Set. *The Journal of VLSI Signal Processing*, 33(1):7–18, 2003.

[Fra65] W. Fraser. A survey of methods of computing minimax and near-minimax polynomial approximations for functions of a single independent variable. *J. ACM*, 12(3):295–314, 1965.

[KS04] A.A. Katkar and J.E. Stine. Modified Booth truncated multipliers. *Proceedings of the 14th ACM Great Lakes symposium on VLSI*, pages 444–447, 2004.

[LCLV08] D.U. Lee, R.C.C. Cheung, W. Luk, and J.D. Villasenor. Hardware Implementation Trade-Offs of Polynomial Approximations and Interpolations. *IEEE TRANSACTIONS ON COMPUTERS*, pages 686–701, 2008.

[2] If only one function is approximated. Approximating more functions increases the cost of the tables while hardly changing the cost of the logic.

[Mul03] J.M. Muller. Partially rounded Small-Order Approximations for Accurate, Hardware-Oriented, Table-Based Methods. *Proc. IEEE 16th Intl Symp. Computer Arithmetic (ARITH16)*, pages 114–121, 2003.

[NSB07] S. Nagayama, T. Sasao, and J.T. Butler. Design Method for Numerical Function Generators Based on Polynomial Approximation for FPGA Implementation. *Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 2007)-Volume 00*, pages 280–287, 2007.

[OS05] S.F. Oberman and M.Y. Siu. A high-performance area-efficient multifunction interpolator. *Proceedings of the 17th IEEE Symposium on Computer Arithmetic (Cap Cod, USA)*, pages 272–279, 2005.

[POMB05] J.A. Piñeiro, S.F. Oberman, J.M. Muller, and J.D. Bruguera. High-Speed Function Approximation Using a Minimax Quadratic Interpolator. *IEEE TRANSACTIONS ON COMPUTERS*, pages 304–318, 2005.

[WIS05] E.G. Walters III and M.J. Schulte. Efficient Function Approximation Using Truncated Multipliers and Squarers. *Proceedings of the 17th IEEE Symposium on Computer Arithmetic (ARITH'05)-Volume 00*, pages 232–239, 2005.

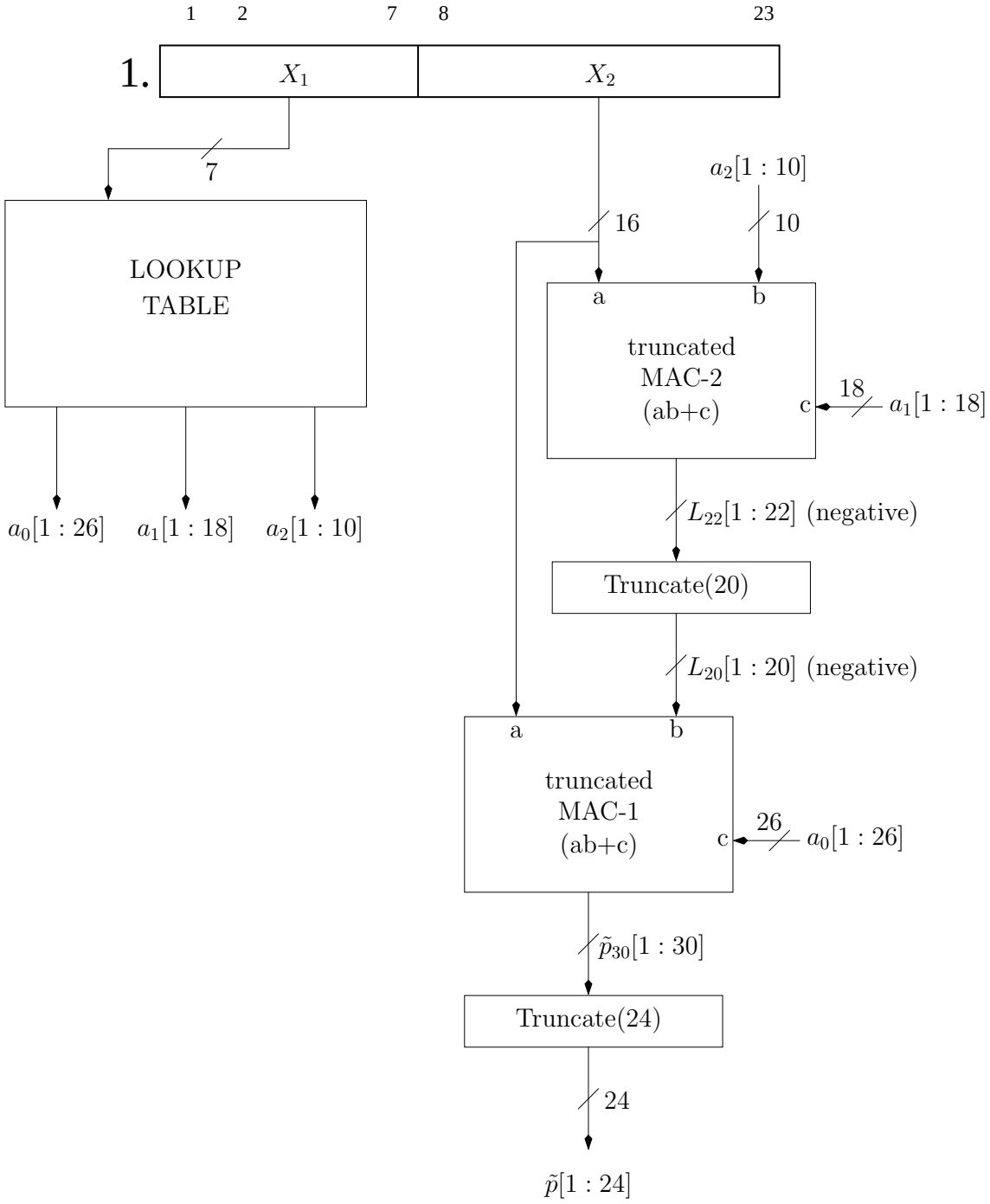

Figure 3: Detailed diagram of micro-architecture for single precision reciprocal.

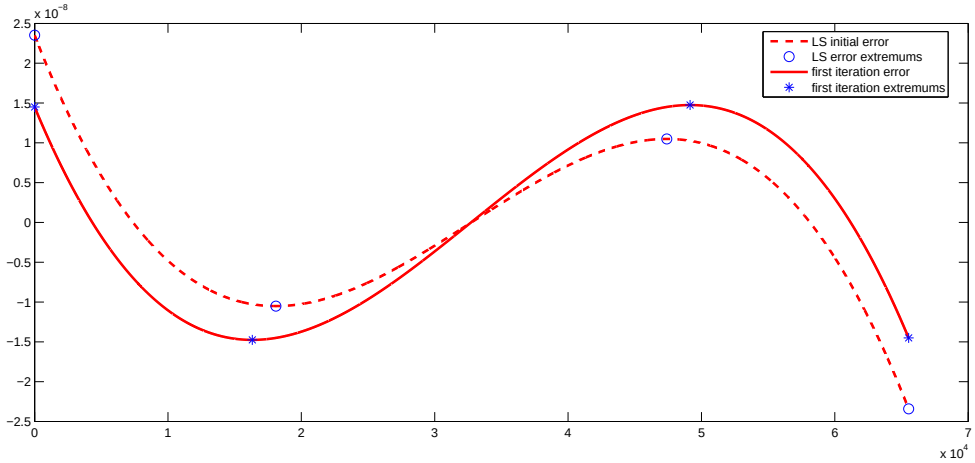

Figure 4: The error of the initial least squares approximation and the error after one iteration of Remez algorithm. Note the reduction in the min-max error after one iteration.

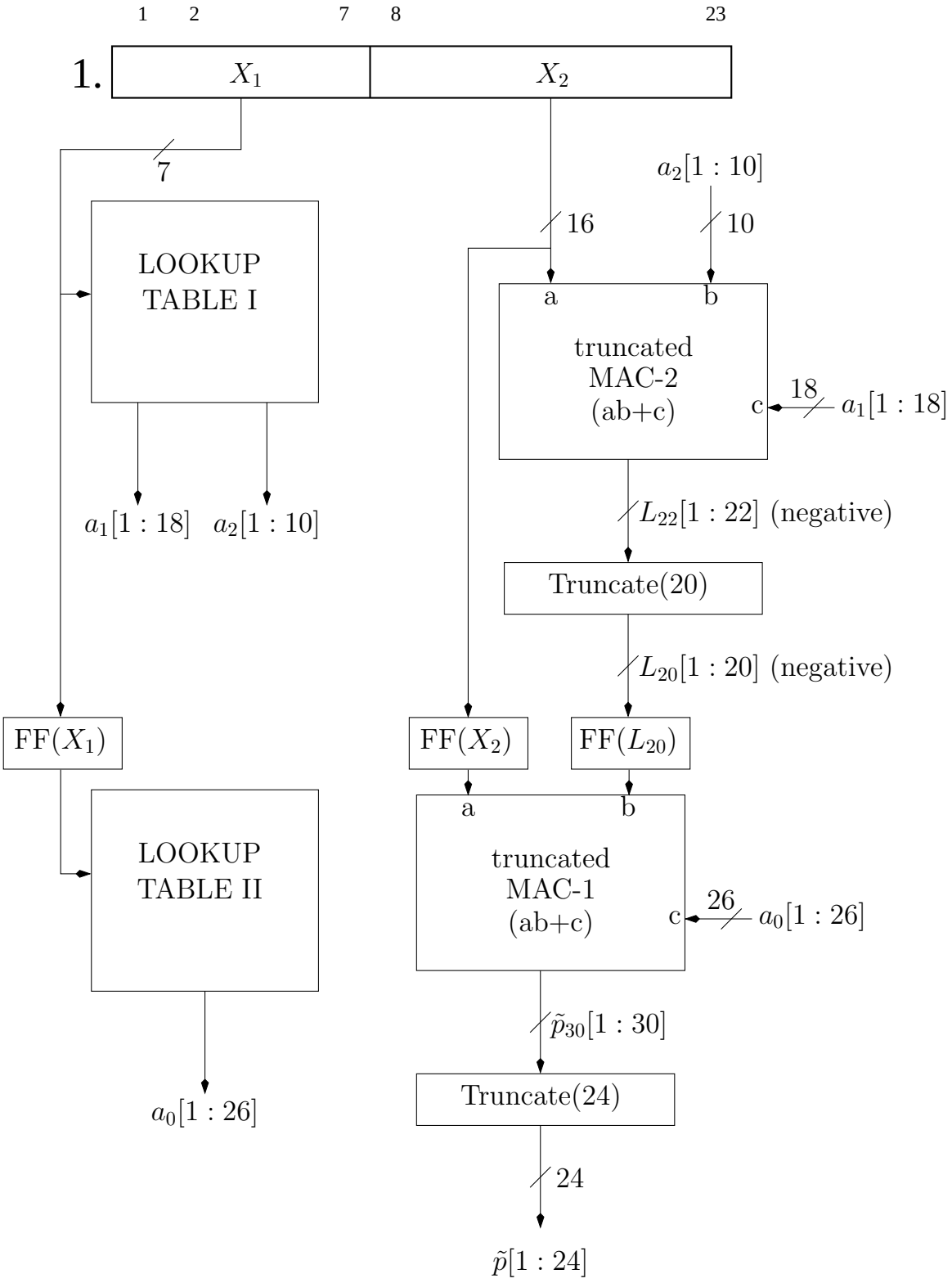

Figure 5: A two-stage pipelined micro-architecture for single precision reciprocal.

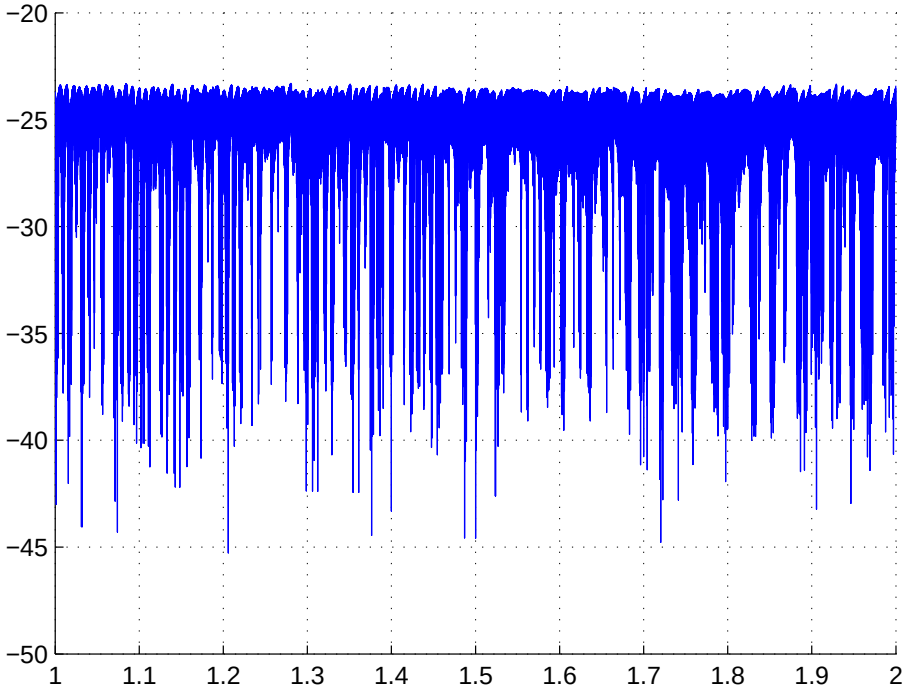

Figure 6: Error before bias correction ($\log_2$ abs(error)).

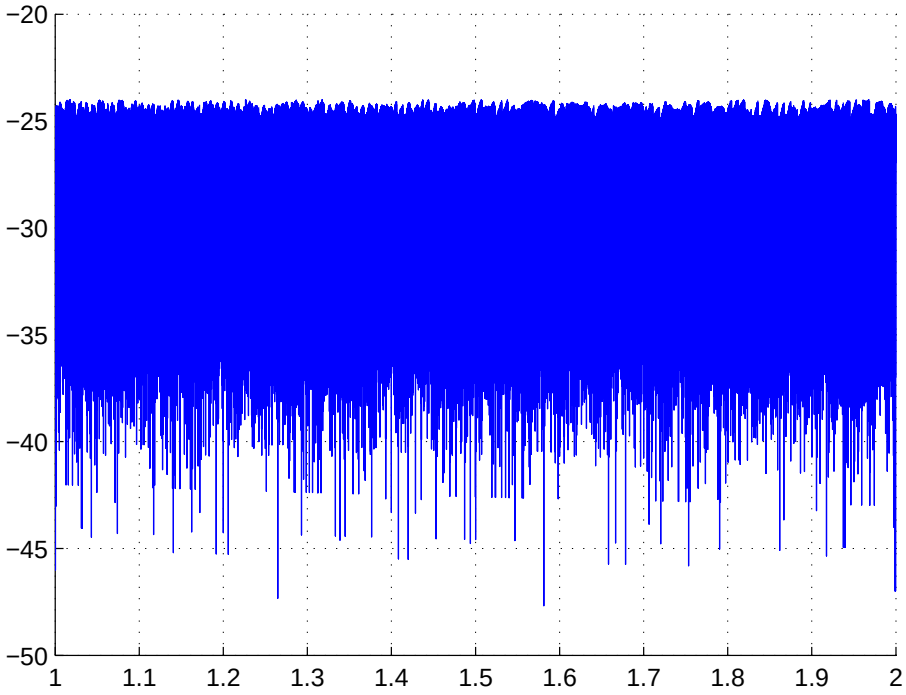

Figure 7: Error after bias correction ($\log_2$ abs(error)).

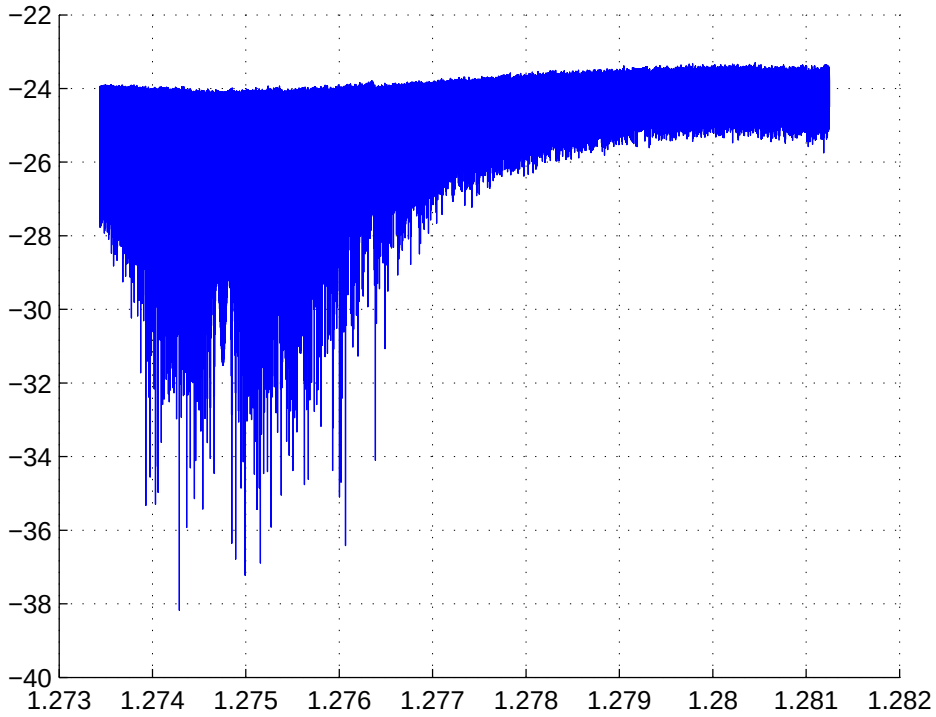

Figure 8: Error before bias correction (zoom on one interval).

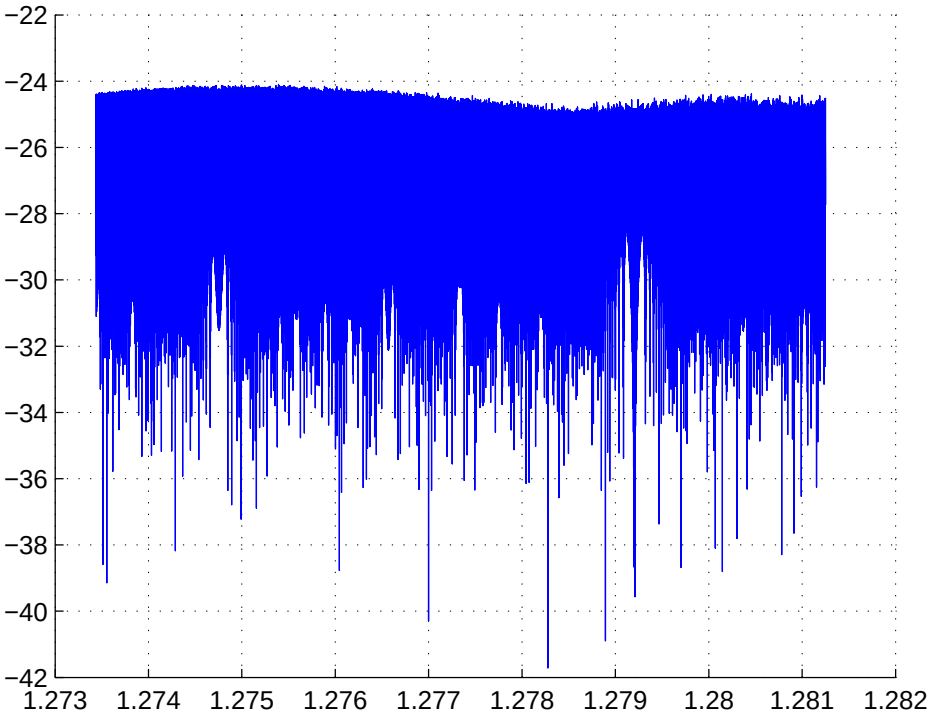

Figure 9: Error after bias correction (zoom on one interval).

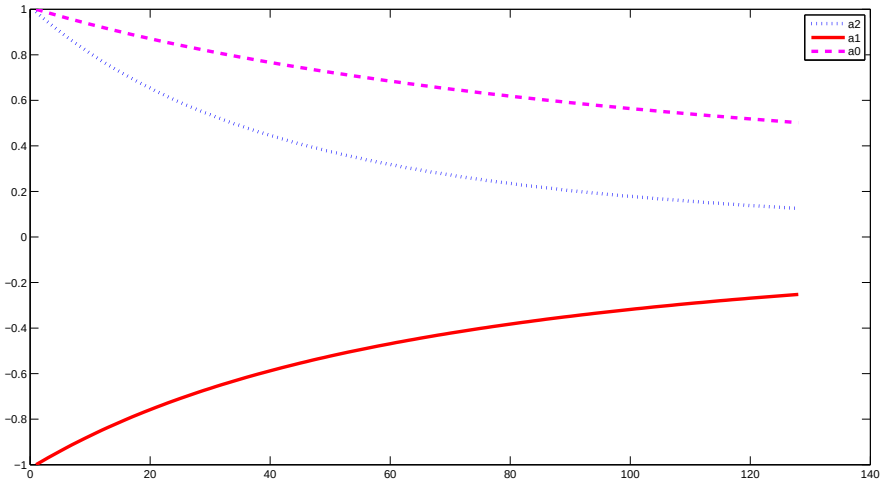

Figure 10: Values of the coefficients stored in the lookup table.