High Performance Architecture for Reciprocal Function Evaluation on Virtex II FPGA

M. Anane, H. Bessalah and N. Anane
Centre de Développement des Technologies Avancées
Haouch Oukil Baba Hassen, Alger, Algérie
e-mail: m.anane@cdta.dz

Abstract

The fast and accurate evaluation of elementary functions is vitally important in many fields of scientific computing. We describe, in this paper, a method to calculate powering ($X^p$) which employs table lookups and polynomial approximation, a second-order Taylor series expansion. We also present the architecture for evaluating the reciprocal on virtex-II FPGA. The virtex-II FPGA family incorporates large Blocks memories SelectRAM and provides fast arithmetic carry logic capability; we exploit these resources to implement our architecture on the XC2V80(-5) FPGA circuit with operating frequency over 31 MHz.

1. Introduction

High-speed evaluation of elementary functions was all the time considered like a very important way to accelerate the scientific computation in digital signal and image processing applications. Among of these the powering function $X^p$ which is very interesting for applications such as computer 3D graphics … It can also be a very efficient way to compute other important functions like: the reciprocal ($X^{-1}$), the square root ($X^{1/2}$), the inverse square root ($X^{-1/2}$) and the reciprocal cube ($X^{-3}$). When the precision of the input and output operands is relatively low, it is possible to employ direct table lookup to store the function values. However, the amount of memory required for the tables becomes quickly prohibitive with the increase of the operands precision. The use of this method to evaluate an elementary function whose argument is in simple precision floating point format requires a table memory of $2^{24} \times 25$ bits that make it inefficient.

Methods that employ two or more parallel table lookups followed by addition have recently been developed for approximating the elementary functions [3], [4], [5], [6]. These methods are especially attractive due to recent advances made in VLSI technology. Compared to conventional table lookups, they require significantly less memory. They also have simple hardware requirements since they only need the memory for the tables and a carry-propagate or multi-operand adder for addition. However these remain very big and impossible to implement on FPGA circuits for the simple and double precisions of the IEEE 754 norm [7]. The most efficient methods for computing powering function are table-driven method algorithms. Thus, it has been traditionally computed by means of a linear approximation for single-precision format. The piecewise linear approximation [8], [9] is an efficient method for generating a power of an operand in rather low precision. The two coefficients of the linear function are read out of a lookup table. A multiplication and an addition are required besides a table lookup. When the $m$ most significant bits of an operand are used as the index of the lookup table, about $2m$-bits accuracy is obtained.

Second degree approximation based algorithms [10], [11] allow the reduction of memory requirements, at the expense of increasing the complexity of computing the polynomial approximation.

Some of today FPGAs are very well suited for the implementation of tables as they are based on small programmable memories. In this paper we use the Virtex-II Xilinx FPGA [12] to design high performance architecture to compute reciprocal. The virtex-II family incorporates dedicated carry logic for high-speed arithmetic and large Block SelectRAM memories. Our contribution in this work is the use of Virtex-II capability to implement the algorithm that computes the reciprocal. The method is based on the second-order Taylor series expansion to compute the powering ($X^p$) as $C_2X^2 + C_1X + C_0$, where $X_2$ is the lower part of $X$. The three coefficients, $C_0$, $C_1$, $C_2$, are stored in lookup tables, and select by $X_1$, the upper part of $X$. 

Several techniques have been developed to reduce the memory needed to accurately evaluate the elementary functions. Some are based on the utilization of addition and shifts (it is the case of the CORDIC algorithm [1]). Others use a polynomial or rational approximation of the function to evaluate [2]. The computing by these last requires an elevated development degree that results in an elevated number of additions and multiplications.
2. Piecewise Taylor interpolation

We assume that the operand X is a (n+1)-bit binary number in range \(1 \leq X < 2 \) Namely X is represented as \([1.x_1,x_2,\ldots,x_n]\) \((x_i \in \{0, 1\})\). We split X into two parts, X1 and X2, where X1= \([1.x_1,x_2,\ldots,x_m]\) and X2= \([x_{m+1},x_{m+2},\ldots,x_n]\) \times 2^{-m}.

By the second-order Taylor expansion, in the range X1 \leq X < X1 + 2^{-m}, Xp of X can be approximated as:

\[
X_p = (X_1+2^{-m}-1)p + (X_2-2^{-m}-1).p.(X_1+2^{-m}-1)p-1 + 1/2.(X_2-2^{-m}-1)^2.p(p-1) + \epsilon_{\text{appro}}
\]

The piecewise approximation based on the second order Taylor expansion adopts a quadratic function \(C_0 + C_1.X_2 + C_2.X_2^2\), where:

\[
C_0 = (X_1 + 2^{-m}-1)^p - 2^{-m}.p(X_1 + 2^{-m}-1)^{p-1} + p(p-1)(X_1+2^{-m}-1)^{p-2}.2^{-2m-3}
\]

\[
C_1 = p(X_1 + 2^{-m}-1)^{p+1} - 2^{-m}.p(X_1 + 2^{-m}-1)^{p-1} + p(p-1)(X_1+2^{-m}-1)^{p-2}
\]

\[
C_2 = 1/2.p(p-1).X_2 + 2^{-m}.p(p-1)(X_1+2^{-m}-1)^{p-2}
\]

The three coefficients \(C_0, C_1\) and \(C_2\) are read through table lookup addressable by \(X_1\) (without the leading 1). The lookup table keeps the coefficients for \(2^m\) intervals of X. The diagram block of this method is presented on the figure 1.

The bound for the error of this second-degree Taylor approximation \(\epsilon_{\text{appro}}\) is expressed as follows:

\[
\epsilon_{\text{appro}} \leq 1/3.2^{-3m-2}.p(p-1)(p-2).max(\xi^{(p-3)})
\]

where \(\xi \in [1, 2]\)

As the bound obtained for the approximation error is about \(2^{3m}\) (the exact value depends on the parameter p), about 3m bits of precision can be obtained by using m bits to address the lookup tables.

The total error of this method comes not only from the second-degree Taylor approximation employed in the algorithm, but also from the limited precision in calculation of the function \(C_2.X_2^2 + C_1.X_2 + C_0\), which includes the rounding errors of coefficients to be stored in lookup tables, the errors owed to intermediate computations and the errors of the final result rounding. The total error can be expressed as follows:

\[
\epsilon_{\text{total}} = \epsilon_{\text{appro}} + \epsilon_{\text{inter}} + \epsilon_{\text{round}}
\]

If the \(\epsilon_{\text{appro}} \leq 2^n\) then m (the width of the table address) can be calculated as follows:

\[
m \geq \frac{n+\log{p}+\log{p}-1+\log{p-2}+3+\log(6)+\max(0,p-3)}{3}
\]

In the case where, we compute a reciprocal (p=−1) with precision of 1ulp (one unit in last place), and the argument is a floating-point simple precision number. Then \(\epsilon_{\text{total}} \leq 2^{-23}, \epsilon_{\text{round}} \leq 2^{-24}, \epsilon_{\text{inter}} \leq 2^{-25}, \epsilon_{\text{appro}} \leq 2^{-25}\) and \(m = 8\).

3. Hardware implementation of the reciprocal

The increase of the Taylor approximation order drives certainly to reduce the total size of lookup tables that is an essential element, as we target to implement our architecture on FPGA circuit (which is a limited resource). Nevertheless, this introduces supplementary computations that can degrade the computation delay of the elementary function considered \((X^{-1})\). According to the architecture presented on figure 1, the computation delay is estimated as:

\[
T_{\text{eval}} = T_{\text{squaring}} + T_{\text{mul}} + T_{\text{add}}
\]

\(T_{\text{squaring}}\) : Execution time of the squaring \((X_2^2)\).

\(T_{\text{mul}}\) : Execution time of a multiplication.

\(T_{\text{add}}\) : Execution time of a three numbers addition.

To reduce the evaluation delay, first of all, we thought to eliminate the carry propagation present at the end of the two multiplications \(C_1.X_2\) and \(C_2.X_2^2\) (which will be done according to the modified Booth algorithm [13]). The main idea is to employ a unified tree to accumulate the
partial products of both $C_1X_2$ and $C_2X_2^2$, plus the coefficient $C_0$. For that, we reduce the partial products of these multiplications, and represent their results in Carry-Save notation [14]. Then the result, in C-S notation, is converted to binary representation by a fast adder. The resulting tree has almost the same delay as standard multiplier. As illustrated on figure 2, we split our architecture in three stages, where each one will include operations that can take place in parallel. The first stage includes, reading coefficients from tables $C_0$, $C_1$, $C_2$, and compute the $X_2$ square (the result will be represented in C-S). The second stage includes generating the partial products of $C_1X_2$ and $C_2X_2^2$, as well as reducing them plus the coefficient $C_0$ to a number in C-S notation. The third stage is composed of a fast adder.

3.1. Memory block.

The three polynomial coefficients $C_0$, $C_1$, $C_2$ of the Taylor approximation are stored in tables addressed by the $m=8$ bits word $X_1$, and their minimum word length is set by the computation, in order to guarantee faithful rounding to the nearest of the results. For the reciprocal function, $C_0$, $C_1$, $C_2$ word lengths are 27, 20 and 12, respectively, which leads to total table size of $2^8\times(27+20+12)=15104$ bits. These coefficients are used in the second stage for generating the partial products of $C_1X_2$ and $C_2X_2^2$ multiplications.

The tables values of $C_0$, $C_1$ and $C_2$ coefficients are computed using C++ program that performs all the computations and the error determination which provides the word lengths of these coefficients.

The virtex-II FPGA Series provides dedicated memory blocks. The block SelectRam memory resources are 18 k-bits of True Dual-Port Ram, programmable from $16\times1$bit to $512\times36$bits, in various depth and width configurations. Our tables is a $256\times59$bits, it can be implemented in only two blocks SelectRAM. The virtex-II circuit XC2V80 makes available 8 blocks.

3.2. Computation of $X_2^2$

Some strategies can be employed to reduce the amounts of CLBs and delay of the unit, which computes the squaring of the lower part of the input operand, $X_2$. These

![Diagram](image-url)
strategies [11] are essentially based on the three following properties:

- \( x_i x_i = x_i \)
- \( x_i x_j = x_j x_i \)
- \( (x_i x_j + x_j x_i)2^k = x_j x_i 2^{k+1} \) where \( k \) designates of the bit rank.

An example for the calculation of \( A^2 \) \((A = a_3 a_2 a_1 a_0)\) is given on figure 3.

\[
\begin{array}{cccc}
  a_3 & a_2 & a_1 & a_0 \\
\times & a_3 & a_2 & a_1 & a_0 \\
  \hline
  a_3 a_0 & a_2 a_0 & a_1 a_0 & a_0 a_0 \\
  a_3 a_1 & a_2 a_1 & a_1 a_1 & a_0 a_1 \\
  a_3 a_2 & a_2 a_2 & a_1 a_2 & a_0 a_2 \\
  a_3 a_3 & a_2 a_3 & a_1 a_3 & a_0 a_3 \\
\end{array}
\]

(Original matrix)

\[
\begin{array}{cccc}
  a_3 a_2 & a_2 a_0 & a_1 a_0 & a_0 a_0 & 0 & a_0 \\
  a_3 & 0 & a_2 a_1 & 0 & a_0 & 0 \\
\end{array}
\]

(Equivalent matrix)

Fig. 3 Matrices transformation for the squaring of A

To further reduce the circuit delay that computes \( X_2^2 \), we let the result in C-S (Carry-Save) representation then the computation is made without carry propagation, which represents the significant delay of the computation operation. The size of the equivalent matrix is also reduced by truncating the partial products and the value of \( X_2^2 \) at the 2^{-28} position, in order to have a precision of 1ulp at the end of reciprocal computation.

3.3. CS-Booth recoding

As \( X_2^2 \) value will be used as operand in the multiplication \( C_2 X_2^2 \) which will be made according to the modified-Booth algorithm. A CS-Booth recoding unit can generate the Booth coding of \( X_2^2 \) needed to obtain the partial products of \( C_1 X_2 \). This is used advantageously in the computation of \( X_2^2 \) in carry save form.

3.4. Binary-Booth recoding

A Binary to Booth recoding unit is needed to generate the Booth coding of \( X_2 \) required to obtain the partials products of \( C_1 X_2 \).

3.5. Partial Products reduction

One of the main features of our architecture is the use of an unified tree to reduce the partial products of \( C_1 X_2 \), \( C_2 X_2^2 \) plus \( C_0 \). This allows us to use only one final adder (with carry propagation) for performing the C-S to binary conversion instead of three.

For the reciprocal \((m=8)\), \( X_2 \) will be so written on 15 significant bits. Therefore, \( X_2^2 \) (without 16 bits leading zeros) is represented on 12 bits. Employing Booth recoding, this word length lead to generation of 8 partial products to compute \( C_1 X_2 \) and 7 partial products to compute \( C_2 X_2^2 \). The total number of products to be computed is 16 \((8+7+1)\), the last one is the coefficient \( C_0 \), and only three levels of 4:2 CSA are needed to perform the whole computation. The critical path of the unit is:

\[
T_{p.p.r} = T_{pp-gen} + 3.T_{4:2\ CSA}
\]

3.6. Final Addition

The final addition is used to convert the result from redundant arithmetic (carry save representation to binary representation). The best way to perform this addition, on Xilinx FPGA, with minimum delay, is the use of the dedicated carry chain available in CLB (Configurable Logic Block).

4. Implementation results

The architecture that we have described in the last section has been designed using Foundation series 4.1 Xilinx environment. All logic blocks have been described in VHDL except the block for the final addition. This one has been generated with the CORE generator system. This is a design tool that delivers parameterized COREs optimized for Xilinx FPGA. To guarantee correct behavior, our architecture has been simulated using ModelSim XE 5.5b. Then synthesized employing FPGA express tool. Our architecture has been mapped, placed and routed into Xilinx XC2V80(-5)cs144 device. The resulting implementation are given as follows:

<table>
<thead>
<tr>
<th>Total accumulated area:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of CLBs</td>
</tr>
<tr>
<td>Number of RAM blocks</td>
</tr>
<tr>
<td>Delay:</td>
</tr>
</tbody>
</table>

The implementation results show that, our circuit can operate at over 31 MHz with latency equal to one and consumes only 446 CLBs. This is compared favorably to the implementation reported in [15] which operates at 33 MHz with latency of 3, and consumes 1130 CLBs.

5. Conclusion

In this paper, a FPGA implementation of a method for computation of reciprocal for single precision floating-point format has been presented. We used for this
implementation a circuit from Virtex-II family, which is developed for high performance designs. We have exploited the dedicated resources provided by this family, like SelectRAM block and carry chain propagation to design high performance architecture for reciprocal function evaluation. In our implementation, we have used two SelectRAM blocks among of eight, the remains blocks can be used to implement three additional functions \(X^0, X^2, X^3\) by duplicate the table size and inserting multiplexers to select the corresponding functions.

6. References


