ARCHITECTURES FOR FUNCTION EVALUATION ON FPGAs

Nalin Sidahao, George A. Constantinides**, Peter Y.K. Cheung***

*Department of Electronic Engineering, Mahanakorn University of Technology, Bangkok 10530, Thailand
**Department of Electrical and Electronic Engineering, Imperial College, London SW7 2BT, UK

ABSTRACT

This paper presents a new family of architectures for multi-cycle area-efficient evaluation of elementary and composite functions, and an exploration of the design tradeoffs for implementation on Field Programmable Gate Arrays (FPGAs). The method is exemplified with two common functions, sine and power-of-2. To test the performance of each design, we compare the proposed architecture to an established table-based method for several different input word-lengths and output precision requirements. FPGA-based results are presented, illustrating both the technology-independent and the technology-specific attributes of the tradeoff of area and speed between the proposed techniques.

1. INTRODUCTION

Due to the increasing need for high-performance Digital Signal Processing (DSP) and scientific computing applications, hardware-based implementation of elementary functions have been proposed in the past [1-3].

Our approach to simple and regular hardware evaluation of elementary functions is based on minimax polynomial approximations of these functions, i.e. approximations that minimize the worst-case error [4]. An important aspect of our work is that the proposed structures can perform an arbitrary function with minor hardware changes. Moreover, we propose a family of architectures, allowing a different number of terms of the polynomial to be computed per clock cycle, and leading to a controlled latency / area tradeoff.

Table-based methods are often employed to implement functions in hardware. For conventional table lookup methods, when the required function precision increases, the memory size grows exponentially. To obtain memory savings, a bipartite table technique [5] and an improved symmetric bipartite table method (SBTM) [6, 7] have been proposed in the past. These techniques can reduce the size of the tables by factor of two compared to previous methods for bipartite table approximations, however the hardware area cost still scales rapidly with the precision requirement.

The area requirement for high precision is thus an essential factor with which to be concerned. This paper proposes a new architecture to evaluate functions for implementation on FPGAs, based on polynomial approximations. The method is illustrated through the implementation of sine and power-of-2. Compared to the table-based SBTM method, for high precision, the proposed architecture provides a lower area requirement for implementation in the FPGA, at the cost of a slower evaluation. Our aim is therefore to target non-critical portions of the arithmetic design to achieve a multi-cycle area-efficient implementation. Results are presented illustrating the design performance tradeoff between area and speed. We believe that the detailed quantitative data presented in this paper for current FPGA families [10] can be helpful in guiding designers or designing automation tools in their selection of function approximations, based on speed, area, and precision requirements.

In summary, the contributions of this paper are:

- The development, and details, of a family of multi-cycle polynomial evaluation architectures
- FPGA performance comparisons between the proposed approach and table-based methods, in order to aid future design and design automation

The rest of this paper is organized as follows: after a description of the proposed architecture in Section 2, the implementation details and performance measurements compared with SBTM are presented in Section 3. Finally, conclusions are drawn, and future work is suggested in Section 4.

2. POLYNOMIAL APPROXIMATIONS

2.1 ALGORITHM

The method described in this paper uses polynomials based on minimax approximations to approximate the value of a function. Consider a general polynomial functional form with real coefficients and real variable

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n \qquad (1)$$

where $0 \le n \in Z$, $a_j, x \in R$, $j = 0,1,\ldots,n$, $a_n \neq 0$.

The core arithmetic operations to be performed in polynomial evaluation are multiplication and addition.

Each member of the proposed family of evaluation architectures allows a different number of terms to be calculated each cycle. We refer to an architecture capable of evaluating k terms per cycle as a k-stage architecture.

As an example, consider the following derivation of a 2-stage implementation. We could consider Equation (1) as representing a fully unfolded parallel implementation. Rearranging Equation

0-7803-7761-3/03/$17.00 ©2003 IEEE

(1) leads to

$$f(x) = a_0 + a_1 x + a_2 x^2 + x^2\left(a_3 x + \ldots + a_n x^{n-2}\right) \quad (2)$$

If we have: $f_{n-2}(x) = a_2 + a_3 x + a_4 x^2 + \ldots + a_n x^{n-2}$ (3)

then from (2) and (3), we can obtain

$$f(x) = a_0 + a_1 x + f_{n-2}(x) x^2 \quad (4)$$

We can consider implementing (4) as a single multiply accumulate, together with a feedback loop, as shown in Fig 1.

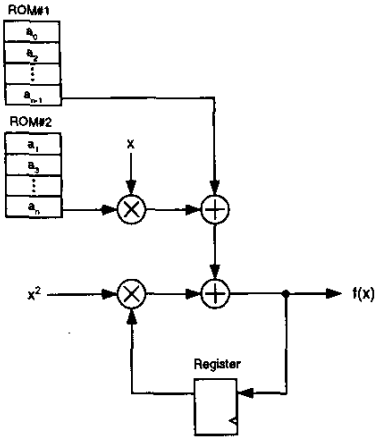

Figure 1: A 2-stage Architecture of n-Degree Polynomial Evaluation

Developing in the same manner, the equation of k-stage of n-degree polynomial can be expressed as

$$f(x) = a_0 + a_1 x + \ldots + a_{k-1} x^{k-1} + f_{n-k}(x) x^k \quad (5)$$

2.2 ARCHITECTURE AND IMPLEMENTATION

Generalizing this algorithmic approach, and pipelining the associated feed-forward network leads to the general polynomial evaluation architecture, shown in Fig. 2.

The diagram is a straightforward realization of equation itself. Two arithmetic operations of multipliers and adders are exploited. The input and output are assumed to require the same precision, p bits. Each ROM contains the coefficients stored as $(p+g)$-bit signed number in 2's complement (g is the number of guard bits). The number of guard bits provides a degree of accuracy control, and can be selected depending on each approximated function and the required bound on approximation error. The requirement is that the generated approximation $\tilde{f}(x)$ differ from its true value $f(x)$ by less than one unit in the least [significant] position, or *ulp* [8]. Due to word-length growth, we truncate the results $\tilde{f}(x)$ before feeding back to $(p+g)$-bit register providing current output for next cycle multiplication with x^k.

Many DSP systems use the fixed-point number system due to power, speed and area advantages over floating-point equivalents [9]. Likewise we assume all coefficient values are in the range [-1, 1). In addition to the datapath, some overheads of a counter, registers for pipelining p-bit inputs x to x^k and a p-bit output $\tilde{f}(x)$, clear and reset circuits, and rounding part at output are required. A deep pipeline is required for the calculation of the required power of x, as in order to maintain the feedback loop as the critical path, each multiplier must include a pipeline register.

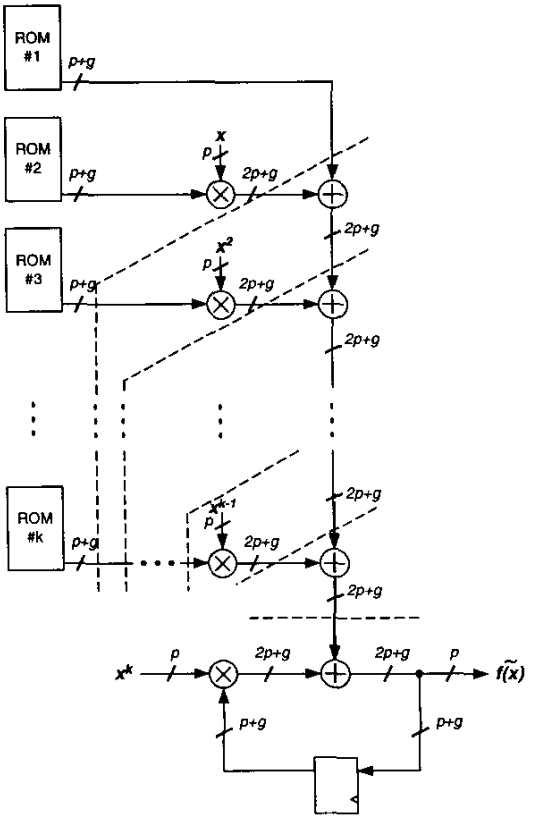

Figure 2: General Architecture of n-Degree Polynomial Evaluation (pipeline stages shown as dashed lines)

For a 1-stage architecture, only ROM#1 and a feedback loop are needed. The ROM#1 contains all polynomial coefficients, and provides an output sequence at each clock cycle starting from a_n and ending with a_0. For a k-stage design, the coefficient values for each ROM are summarized in the following table:

Given $m = \lceil (n+1)/k \rceil - 1$ (6)

	Last cycle ↓				First cycle ↓
ROM#1	a_0,	a_k,	a_{2k},	…,	a_{mk}
ROM#2	a_1,	a_{k+1},	a_{2k+1},	…,	a_{mk+1}
ROM#3	a_2,	a_{k+2},	a_{2k+2},	…,	a_{mk+2}
…			…		
ROM#k	a_{k-1},	a_{2k-1},	a_{3k-1},	…,	a_n

Table 1: Content of ROMs

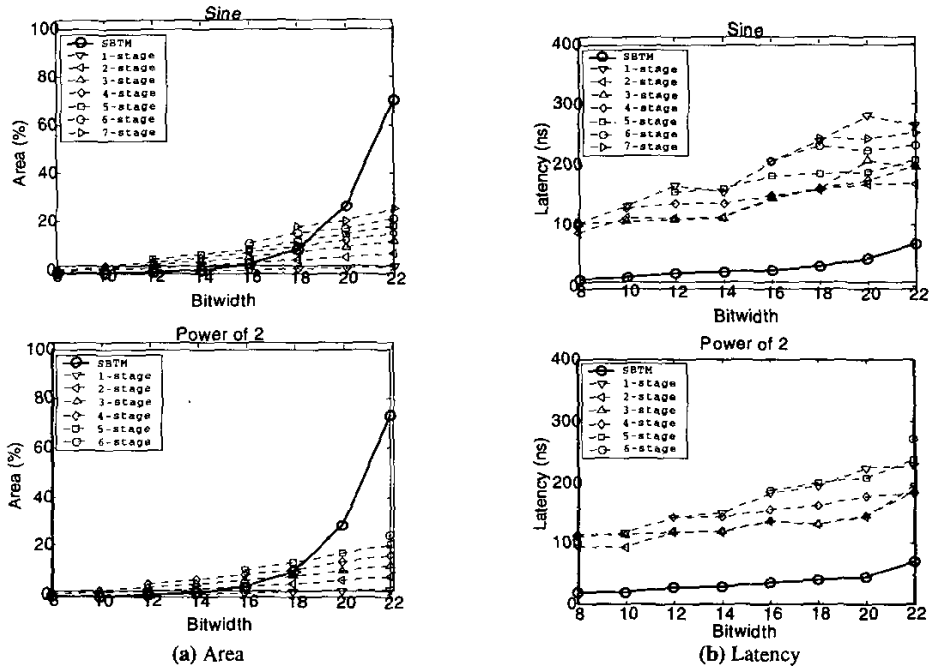

Figure 3: Performance Results against Bitwidth

For non-integer $(n+1)/k$, the addition of higher order term(s) with zero coefficients is required to fill the ROMs appropriately. This increases the hardware requirement by a small amount in order to maintain a regular and simple hardware structure.

Rather than pipelining the output of each ROM, the stored coefficients are rotated by of the degree to which it would be pipelined. For example, in Fig. 2, ROM#3 coefficients are rotated from right to left by one address position.

There are three main aspects of this structure:

- The critical path of the overall computation lies on the feedback loop. This is caused by the propagation delay of a $p\times(p+g)$-bit multiplier and a $(2p+g)$-bit adder.
- A polynomial of n-degree with k-stages can produce a complete evaluation every $\lceil (n+1)/k \rceil$ cycles. Thus fewer stages lead to lower throughput. In addition, fewer stages tend to require more guard bits in order to maintain acceptable approximation results.
- Due to the pipelining of the feed-forward network, the total latency for a k-stage design is $\lceil (n+1)/k \rceil + k$ cycles. If latency is more important than throughput for a particular design, attention must be drawn to the fact that there is a lowest latency achievable of approximately $2\sqrt{(n+1)}$ cycles, for $k = \sqrt{(n+1)}$ and integer $\sqrt{(n+1)}$.

The number of polynomial terms obtained from minimax polynomial approximations is based on the maximum error acceptable between the approximation and the true function value. The acceptable maximum error must be less than one *ulp* for each precision; therefore lower precision requires fewer terms. This implies that very high stage implementations are superfluous for low precision requirements.

3. RESULTS

For analyzing performance of the hardware, an implementation of the proposed architectures on a Xilinx Virtex device [10] (V1600E) is considered. For comparison, SBTM tools were obtained from [11]. In all cases, the number of guard bits was exhaustively simulated through MATLAB [12] to ensure comparable error performance between the two methods.

Fig. 3 demonstrates the performance results against word-length. The overall results show the tradeoffs that can be make in terms of area and speed for different precision requirements. Fig. 3(a) shows percentage slice usages of total 15,552 slices (each slice consists of two 4-input look-up tables (LUTs) and two registers). The area usages of our proposed architecture grows linearly with word-length, whereas the SBTM is likely to increase exponentially. Considering each particular stage provides important information about the range of precisions where the proposed method requires less area. In both cases of functions of Fig 3(a), this area cross-over occurs between the 14- and 22-bit approximations for a 1-stage design. The cross-overs for 2- and 3-stage designs are at 18 to 22 bits. Finally, upwards of 4-stage are at 20- to 22-bit approximations.

Fig. 3(b) illustrates the effect on latency of increasing word-length. As the word-length increases, the latency of all cases

increases. It is clear that for all precisions the proposed method has a significantly longer latency than the SBTM approach. The more complex interaction between the number of stages and latency are also clear from Fig. 3(b): both the sine and power-of-2 architectures have minimum latency for a 2-stage design.

Fig. 4 illustrates the variation of throughput for our proposed method with the number of stages. A similar plot for the SBTM approach is not provided, as the latter can be pipelined to an arbitrary depth.

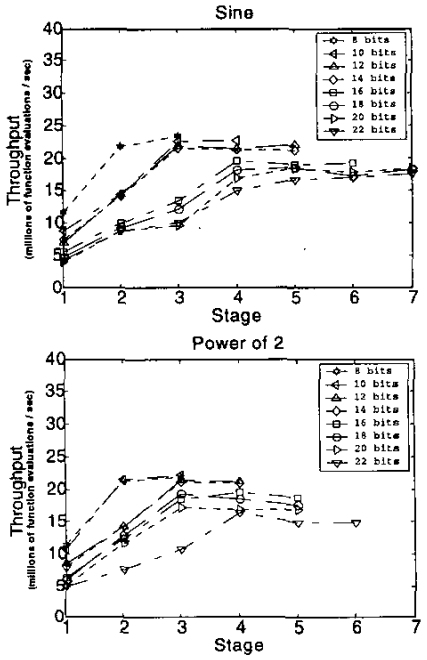

Figure 4: Throughput Results against Stage

Mainly as a result of fewer polynomial computation terms required for low precision calculations, the throughput for low precision tends to be greater than that for high precision. As the number of stages increases, the throughput rises linearly until a $\lceil (n+1)/2 \rceil$-stage design. After this, the throughput tails off. This is because the $\lceil (n+1)/2 \rceil$-stage design computes half the number of polynomial terms by a cycle which is the most possible for full utilization of all resources during all cycles. This number of stages requires two cycles to complete all polynomial terms. Although using more stages can compute more than half the terms each cycle, these designs still require two cycles to complete all terms. For instance, for the 6^{th} order polynomial approximation to the 16-bit sine function, a 4-stage design would achieve the peak throughput, whereas 5- and 6-stage designs would be inferior, providing approximately the same throughput with a greater area requirement and longer latency.

4. CONCLUSIONS

The proposed architecture provides designers the flexibility of a family of architectures for function approximation in order to achieve a multi-cycle area-efficient implementation. We have also presented the important aspect of bit precision to area and speed performance of our work compared to a table-based method.

The restrictions of our proposal are the following: the representation used for evaluation the elementary function employs a signed fixed-point fractional number system; the tradeoffs for floating point or alternative numbers may be different. In addition, we limit the coefficients of polynomial in the range [-1, 1). Therefore, if the original minimax coefficients do not satisfy such a range, these will be scaled down by an appropriate constant value α. That means when actually implementing the approximation, the final outputs of $\tilde{f}(x)$ need to be multiplied back with α (this may involve only shift and add operations).

Current effort involves extending the presented architecture for other function approximations and incorporating them within an electronic design automation framework. In addition, we are investigating shaping the precision and overflow-protection requirements.

5. REFERENCES

[1] J. Cao, B.W.Y. Wei and J. Cheng, "High-performance architectures for elementary function generation," in *Proceedings of the 15th IEEE Symposium on Computer Arithmetic*, Vail, Colorado, pp. 136-144, June, 2001.

[2] Y. H. Hu, "CORDIC-based VLSI architectures for digital signal processing," *IEEE Signal Processing Magazine*, vol. 9, pp. 16-35, July 1992.

[3] J. L. Aravena and S. T. Soh, "Architectures for polynomial evaluation," *Proceedings of the Twenty-First Southeastern Symposium on System Theory*, pp. 190-194, March 26-28, 1989.

[4] J. M. Muller, *Elementary Functions: Algorithms and Implementation*, Birkhäuser, 1997.

[5] D. D. Sarma and D. W. Matula, "Faithful Bipartite ROM Reciprocal Tables," *Proceedings of the 12th IEEE Symposium on Computer Arithmetic*, Bath, England, pp. 17-28, July, 1995.

[6] M. J. Schulte and J. E. Stine, "Symmetric Bipartite Tables for Accurate Function Approximation," *Proceedings of the 13th IEEE Symposium on Computer Arithmetic*, Pacific Grove, California, pp. 175-183, July, 1997.

[7] M. J. Schulte and J. E. Stine, "Approximating Elementary Functions with Symmetric Bipartite Tables," *IEEE Transactions on Computers*, vol. 48, no. 8, pp. 842-847, August, 1999.

[8] I. Koren, *Computer Arithmetic Algorithms*. Englewood Cliffs, New Jersey: Prentice Hall, 1993.

[9] C. Inacio and D. Ombres, "The DSP decision: fixed point or floating?," *IEEE Spectrum*, vol. 33, pp. 72-74, September 1996.

[10] Xilinx, San Jose, *DataSource™ CD-ROM*, Rev. 7, Third Quarter, 2002.

[11] M. J. Schulte and James E. Stine, "Internet Tools for Symmetric Bipartite Tables for Accurate Function Approximation," http://www.eecs.lehigh.edu/~caar/SBTM.html.

[12] MathWorks, "MATLAB," http://www.mathworks.com.