Parameterized High Throughput Function Evaluation for FPGAs

OSKAR MENCER
MAXELER Technologies, Florham Park, NJ 07932, USA

WAYNE LUK
Department of Computing, Imperial College, 180 Queen’s Gate, London SW7 2BZ, UK

Received January 28, 2002; Revised September 19, 2002; Accepted October 15, 2002

Abstract. This paper presents parameterized module-generators for pipelined function evaluation using lookup tables, adders, shifters, multipliers, and dividers. We discuss trade-offs involved between (1) full-lookup tables, (2) bipartite (lookup-add) units, (3) lookup-multiply units, (4) shift-and-add based CORDIC units, and (5) rational approximation. Our treatment mainly focuses on explaining method (3), and briefly covers the background of the other methods. For lookup-multiply units, we provide equations for estimating approximation errors and rounding errors which are used to parameterize the hardware units. The resources and performance of the resulting design can be estimated given the input parameters. A selection of the compared methods are implemented as part of the current PAM-Blox module generation environment. An example shows that the lookup-multiply unit produces competitive designs with data widths up to 20 bits when compared with shift-and-add based CORDIC units. Additionally, the lookup-multiply method or rational approximation can produce efficient designs for larger data widths when evaluating functions not supported by CORDIC.

Keywords: function approximation, FPGAs, lookup table, CORDIC, rational approximation

1. Introduction

The evaluation of differentiable functions can often be the performance bottleneck of compute-bound applications. Examples of these functions include elementary functions such as \( \sin(x) \), \( \log(x) \) or \( \sqrt{x} \), and compound functions such as \( (1 - \sin^2(x))^{1/2} \) or \( \tan^2(x) + 1 \).

An appealing way to overcome such performance bottlenecks is to use a reconfigurable computing system, which contains Field-Programmable Gate Arrays (FPGAs) to accelerate an application running on the microprocessor. In such systems, a function can be evaluated either in software on the processor, or in hardware on the FPGA. However, prior work [1] on computing elementary functions is focused on Application Specific Integrated Circuits (ASICs). Advanced FPGAs enable the development of low-cost and high-speed function evaluation units, customizable to particular applications. Such customization can take place at run time by reconfiguring the FPGA, so that different functions or precisions can be introduced according to run-time conditions.

This paper presents and compares parameterizable designs for evaluating differentiable functions using lookup tables, adders, multipliers, and dividers, which can be implemented very efficiently using FPGAs and other stream-based architectures [2–5]. We describe architectures based on full-lookup units, lookup-add units [6], lookup-multiply units [7], CORDIC units [8], and rational approximations, and compare their size and performance on FPGAs for different input data widths. We consider FPGA implementations which offer a high degree of parallelism and pipelining while allowing the user to trade off area for reduced latency. For highly pipelinable applications, such parallelism provides an edge over...
general-purpose microprocessors and can lead to significant speedups.

The function evaluators are implemented as module generators within the PAM-Blox environment [9], so that instances of particular architectures can be generated rapidly and automatically from a parameterized description. The key problem addressed in this paper is the automatic generation of parameterized function evaluation units for FPGAs. One of the challenges of automatic generation of parametrizable function evaluation units is to find the required intermediate precisions, given an input and output precision requirement. For example, if we require a 15-bit sine function from a 13-bit input, the module-generators have to create the function evaluation unit which may contain temporary values that require different precision than inputs or outputs. These intermediate precisions have to be selected to optimize speed and area of the unit.

Preliminary results suggest that there is a tradeoff between the various table lookup methods and shift-and-add based (e.g. CORDIC) units based on the number of required bits [8, 10, 11]. This paper sheds some light into the details of this tradeoff, and the applicability of the various methods to a module generation environment, which is the heart of computing with FPGAs.

2. Function Evaluation Methods

Our objective is to provide efficient circuits for differentiable function evaluation. In general, function evaluation consists of three stages. The first stage, range reduction (see for example Muller [12], Chapter 8), reduces the argument $x$ to a small interval $[a, b]$, resulting in a new argument $\bar{x}$; for instance

$$
\bar{x} = \bar{x} - kC \quad \text{(for trigonometric functions)}
$$

$$
\bar{x} = \bar{x} / C^k \quad \text{(for the logarithm function)}
$$

where $C$ is a function specific constant and $k$ is the appropriate integer to reduce the particular $x$ to the range $[a, b]$. The second stage evaluates the function $F(x)$. The third stage extrapolates $F(\bar{x})$ from $F(x)$.

In this paper, we are mainly concerned with the second stage: evaluating a function $F(x)$ where $x$ is in a small, natural evaluation interval $[a, b]$. For example, for $\sin(x)$ the evaluation interval could be $[0, \pi/2]$. For any given function, area restrictions, and timing restrictions, there exist many different architectures with different evaluation intervals $[a, b]$. Given a specific function and interval it is possible to pick the architecture which best meets the area and timing requirements of the application.

For reconfigurable datapaths, we are concerned with high throughput architectures. We therefore limit the scope of this paper to bit-parallel and pipelined FPGA implementations of function evaluation units. In the following, we describe three architectures for evaluating a given function based on lookup tables, the CORDIC method for evaluating functions using only shift and add primitives, and rational approximation.

2.1. Three Lookup-Table based Units

The first architecture, a full-lookup unit, consists of a single lookup table. While a full-lookup table is straightforward to implement, its size and latency grows very rapidly with the required precision or range.

The second architecture, a lookup-add unit (Fig. 1), is based on bipartite tables involving an addition of the results of two parallel lookups. The use of addition in conjunction with symmetric tables further reduces the required memory for the lookup tables while improving the error bounds. However, in contrast to the full lookup method, we now have to find the precision required for each of the tables and the precision for the final addition. Current techniques use simulation to find the appropriate internal bitwidths. Recently a version of this method, based on multi-partite tables [19, Muller (2001)], has been implemented in the JBits FPGA development tool.

The third architecture, which is the most promising, is a lookup-multiply unit (Fig. 2) based on affine polynomial approximation of a differentiable function. The coefficients for a polynomial approximation can be computed to minimize the average error of the

---

**Figure 1.** Bipartite (lookup-add) tables computing the function $F(x)$. $x_1$ and $x_2$ are substrings of the original binary input $x$. 