Adaptive Range Reduction for Hardware Function Evaluation

Dong-U Lee, Altaf Abdul Gaffar, Oskar Mencer and Wayne Luk
Department of Computing, Imperial College
London, United Kingdom
{dong.lee, altaf.gaffar, o.mencer, w.luk}@ic.ac.uk

Abstract

Function evaluation $f(x)$ typically consists of range reduction and the actual function evaluation on a small interval. In this paper, we investigate optimization of range reduction given the range and precision of x and $f(x)$. For every function evaluation there exists a convenient interval such as $[0, \pi/2)$ for $\sin(x)$. The adaptive range reduction method, which we propose in this work, involves deciding whether range reduction can be used effectively for a particular design. The decision depends on the function being evaluated, precision, and optimization metrics such as area, latency and throughput. In addition, the input and output range has an impact on the preferable function evaluation method such as polynomial, table-based, or combinations of the two. We explore this vast design space of adaptive range reduction for fixed-point $\sin(x)$, $\log(x)$ and $\sqrt{x}$ accurate to one unit in the last place using MATLAB and ASC, A Stream Compiler. These tools enable us to study over 1000 designs resulting in over 40 million Xilinx equivalent circuit gates, in a few hours' time. The final objective is to progress towards a fully automated library that provides optimal function evaluation hardware units given input/output range and precision.

1 Introduction

The evaluation of functions is often the performance bottleneck of many compute-bound applications. Examples of these functions include elementary functions such as $\log(x)$, $\sin(x)$ and $\sqrt{x}$. Computing these functions quickly and accurately is a major goal in computer arithmetic. Software implementations are often too slow for numerically intensive or real-time applications. The performance of such applications depends on the design of a hardware function evaluator. Advanced FPGAs enable the development of low-cost and high-speed function evaluation units, customizable to particular applications. The challenge is to provide a programming tool or library for FPGAs that delivers the optimal function evaluation unit for a given function, with the associated input/output range and precision.

Recent work [6, 9] shows the connection between precision and function evaluation methods. This paper focuses on adaptive range reduction. The main contributions of this paper are:

- Framework for adaptive range reduction based on a parametric function evaluation library, and on function approximation by polynomials and tables and precomputing all possible input/output ranges.
- Implementation of design space exploration for adaptive range reduction, using MATLAB in producing function evaluation parameters for hardware designs targeting the ASC system.
- Evaluation of the proposed approach by exploring various effects of range reduction of several arithmetic functions such as $\sin(x)$ and $\log(x)$ on throughput, latency and area for FPGA designs.

The rest of this paper is organized as follows. Section 2 covers overview and background material. Section 3 shows the design of the adaptive function evaluation library for ASC. Section 4 presents the implementation of the algorithmic design space exploration with MATLAB, ASC library code generation, and the automation of the ASC design space exploration process optimizing area, latency or throughput. Section 5 discusses results, and Section 6 offers conclusions and thoughts on future work.

2 Background

Consider an elementary function $f(x)$, where x and $f(x)$ have a given range $[a, b]$ and precision requirement. The evaluation $f(x)$ typically consists of three steps [12]:

(1) range reduction, reducing x over the interval $[a, b]$ to a more convenient y over a smaller interval $[a', b']$,

(2) function evaluation on the reduced interval, and

(3) range reconstruction: expansion of the result back to the original result range.

There are two main types of range reduction:

- additive reduction: y is equal to $x - mC$;
- multiplicative reduction: y is equal to x/C^m

where integer m and a constant C are defined by the evaluated function.

Range reduction is widely studied, especially for CORDIC [13] and floating-point number systems on microprocessors [1]. Li et. al. [8] present theorems that prove the correctness and effectiveness of commonly used range reduction techniques. Lefévre and Muller [7] suggest a method for performing range reduction on the fly: overlapping the computation with the reception of the input bits for bit-serial systems. Defour et. al. [2] present an algorithm suitable for small and medium sized arguments in IEEE double precision. Their method is significantly faster than Payne and Hanek's modular range reduction method [12], at the expense of larger table sizes.

In contrast, range reduction which adapts to different input ranges and precisions has received little attention. To the best of our knowledge, this is the first paper that deals with this issue.

There are numerous methods to approximate a function on a reduced interval, and the optimal method depends on the precision of the input and output as shown in [9]. Direct table look-ups are impractical for precisions higher than a few bits. Symmetric table addition methods [4] are fast with reasonable table sizes for precisions lower than 20 bits, but are perhaps inappropriate for larger precisions due to its large table sizes at high precisions. CORDIC function evaluation provides a popular research topic, involving only shift and add operations. However, CORDICs have an execution time which is linearly proportional to the number of operands, and is not suitable for applications requiring high accuracy and speed. Of course the tradeoffs depend on the optimization metric as well. In this paper we show that input and output ranges form another consideration when choosing the optimal method. We demonstrate the principles with polynomial-only and table-with-polynomial methods with a varying number of coefficients.

3 Design

This section describes our approach for adaptive range reduction. Section 3.1 provides an overview. Section 3.2 describes the degrees of freedom for choosing different parameters in our method.

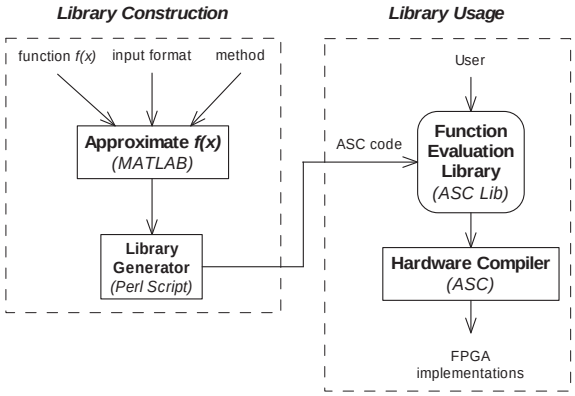

Figure 1: *Work flow: MATLAB generates all the ASC code for the library. The user simply indexes into the library to obtain the specific function approximation unit.*

3.1 Design Overview

Figure 1 shows the work flow of our approach. The function of interest, its input range and precision, and evaluation method are supplied to our MATLAB program, which automatically designs the function approximator and produces its hardware description. In our case, MATLAB produces code for ASC, A Stream Compiler for FPGAs [10]. This large collection of ASC functions is then transformed by a Perl script into an ASC function evaluation library (ASC lib). ASC then takes care of design space exploration on the architecture level, the arithmetic level, and the gate level of abstraction. The result is an optimized function evaluation library for computing with FPGAs.

Given a function $f(x)$ and an interval $[a, b]$ we approximate the function with polynomials and tables. Tasks in designing a function evaluation library include automating the selection of range reduction, the selection and design of the function evaluation method, and area, latency and throughput optimizations on the lower levels of abstraction. This section shows how we design a function evaluation library that contains optimized implementations for a large number of range/precision combinations.

The conventional way of implementing function evaluation is shown below for the three functions evaluated in this paper. We use ASC code notation [10] in Figure 2 to show various methods of function evaluation including range reduction and range reconstruction, which follow the ideas presented in [11] and [13].

The code in Figure 2 shows as an example a different function evaluation method for each function. In reality, we use many combinations of evaluation methods and functions. $\sin(x)$ is an instance of additive reduction, whereas $\log(x)$ and $\sqrt{x}$ are instances of multiplicative reduction.

The central contribution of this paper lies in recon-

Evaluating $f(x) = \sin(x)$

```
// Range Reduction
x1 = abs(x) % (2*pi);
x2 = IF(x1>pi, x1-pi, x1);
y = IF(x2>(pi/2), pi-x2, x2);

// Evaluation Method
// f(y) where y = [0,pi/2)
// e.g. polynomial-only (po)
f1 = (a*y+b)*y+c;

// Range Reconstruction
f = IF(x1>pi, f1, -f1);
```

Evaluating $f(x) = \log(x)$

```
// Range Reduction
exp = LeadingOneDetect(x)-FracWidth(x);
y = x << exp;

// Evaluation Method
// f(y) where y = [0.5,1)
// e.g. table+degree-1-polynomial (tp1)
f1 = Table1[y]*y+Table2[y];

// Range Reconstruction
f = f1+exp*log(2);
```

Evaluating $f(x) = \sqrt{x}$

```
// Range Reduction
exp = LeadingOneDetect(x)-FracWidth(x);
x1 = x << exp;
y = IF(exp[0], x1 >> 1, x1);

// Evaluation Method
// f(y) where y = [0.25,1)
// e.g. table+degree-2-polynomial (tp2)
f1 = (Table1[y]*y+Table2[y])*y+Table3[y];

// Range Reconstruction
exp1 = IF(exp[0], exp+1 >> 1, exp >> 1);
f = f1 << exp1;
```

Figure 2: *Description of range reduction, evaluation method and range reconstruction for the three functions* $\sin(x)$, $\log(x)$ *and* $\sqrt{x}$.

sidering the above structure for user-defined fixed-point bitwidths. When programming FPGAs one can select any bitwidth for the integer part and the fractional part of the fixed point number. As a consequence, a function evaluation library obtains the range and precision of the input and can use this information to produce an optimized function evaluation unit. Previous work [9] shows the subproblem of how to select function evaluation methods based on precision. In this work we add the issue of input/output range and range reduction. Based on input range and precision we now have the following degrees of freedom:

1. applicability of range reduction
2. evaluation method selection
3. evaluation method design
   - find minimal bitwidths
   - find minimal polynomial degree (for polynomial-only method)
   - find minimal segments (for table-with-polynomial method)
4. optimize: area, latency or throughput

The ASC function evaluation library takes the range, precision and optimization metric and instantiates one of many instances of the corresponding function evaluation unit.

3.2 Degrees of Freedom

Applicability of Range Reduction

Assume we require a hardware unit to compute $\sin(x)$ where x is a fixed point variable with four integer bits and eight fraction bits. Then the range of the input is $[0, 16)$ and the expected range of the output is $[-1, 1]$. The precision of the input and output is 2^{-8} which also sets the ulp (unit in last place). Given a particular function that we want to evaluate, we can decide whether it is necessary to implement range reduction or not. In order to make the correct decision we need to consider the optimization metric (area, latency or throughput), design a function evaluation unit with and without range reduction, and select the more optimal one.

In practice, we actually pre-compute all possible input ranges and store for each function a particular range r so that for all input ranges smaller than r we do not use range reduction, and for all input ranges above r we use range reduction. We obtain the graphs which determine r after place-and-route. Section 5 shows the detailed graphs of this step.

Evaluation Method Selection

There are many possible function evaluation methods, such as symmetric table addition methods, CORDIC, rational approximation, polynomial-only methods and table-with-polynomial methods. In this paper we explore polynomial-only (*po*) and table-with-polynomial methods with polynomials of degree two to four (*tp2-4*). The architecture for an approximation unit with a table-with-polynomial scheme is shown in Figure 4. The polynomial coefficients are found in a minimax sense that minimizes the maximum absolute error.

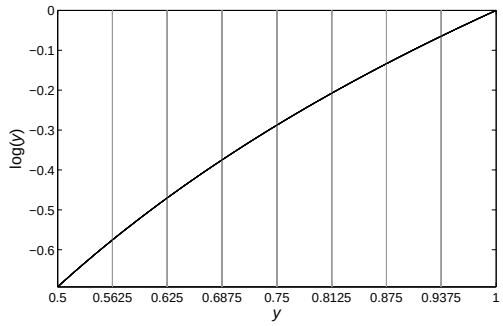

Figure 3: *Segmentation for evaluating* $\log(y)$ *with eight uniform segments. The leftmost three bits of the inputs are used as the segment index.*

For the table-with-polynomial approach, the input interval is split into 2^k equally sized segments. The k leftmost bits of the argument y serve as the index into the table, which holds the coefficients for that particular interval. Note that for the polynomial-only approach, there would be just one entry (coefficient) in the table and no addressing bits. Segmentation for evaluating $\log(y)$ with eight uniform segments ($k = 3$) is illustrated in Figure 3. We observe that the range reduced interval is relatively linear, and hence the use of uniform segmentation is sufficient. However, for non-linear functions such as certain compound functions, one should use non-linear segmentation techniques such as the hierarchical segmentation method [5]. Note that for the polynomial-only approach, there would be just one entry (coefficient) in the table and no addressing bits.

The table-with-polynomial (*tp*) methods trade off table area versus polynomial area, while the polynomial-only method results in precision and polynomial degree related to the precision of the input and output of the function evaluation.

Evaluation Method Design

Once we know which method to use, we need to design the optimized unit. For the polynomial-only method we can find the minimal degree of the polynomial that will satisfy the required output precision. Then we need to find the optimized bitwidths of the computation inside the function evaluation units for all the methods.

Optimize: Area, Latency or Throughput

While the options or selections of the previous degrees of freedom are pre-computed with MATLAB, the area, latency and throughput optimizations on the arithmetic and gate-levels can be left for the compiler to worry about. The next section about the implementation contains details on how this is achieved.

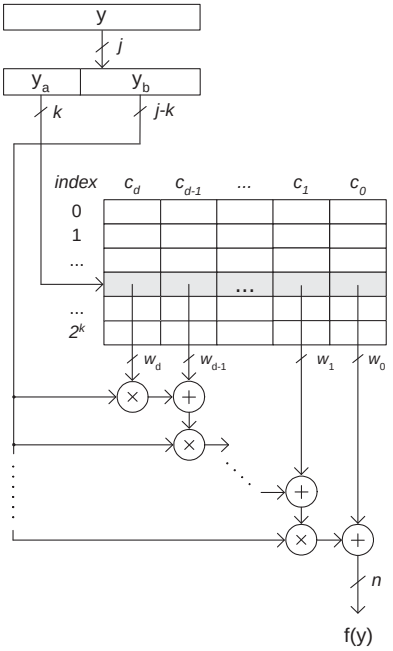

Figure 4: *Architecture of table-with-polynomial unit for degree* d *polynomials. Horner's rule is used to evaluate the polynomials.*

4 Implementation

This section presents the implementation of the algorithmic design space exploration with MATLAB, and ASC code generation and optimization.

4.1 Algorithmic Design Space Exploration

We use MATLAB to generate a large number of implementations for function evaluation. We consider several function evaluation methods: polynomial-only (*po*), and table-with-polynomial of degree two to four (*tp2*-*tp4*). For a given function and any range/precision pair, the MATLAB code generates polynomial coefficients which form entries of the lookup tables based on the Remez method [12]. This method computes the minimax coefficients that minimize the maximum absolute error over an interval. The range and precision are represented by the integer and fraction bitwidths respectively. In this fashion we also obtain minimal bitwidths and the minimal number of polynomial terms for the po method. For the *tp* methods, we find the minimal table-size and the coefficient bitwidths for the given range and precision.

The following structure is used to produce 2000 lines of MATLAB code for design exploration.

```
// for a given function f, input format i,
// method m and polynomial degree d

if (m=='po') // for polynomial-only
        // find minimum polynomial degree
        min_degree = find_min_degree(f,i);
        // find minimum internal fraction bitwidth
        int_bw =  find_min_int_bw(f,i,min_degree);
        // generate polynomial coefficients
        coeffs = gen_coeffs(f,i,min_degree,int_bw);
        // generate ASC code
        gen_ASC(f,i,min_degree,int_bw,coeffs);

elseif (m=='tp') // for table-with-polynomial
        // find minimum number of segments
        min_segs = find_min_segs(f,i,d);
       // find minimum internal fraction bitwidth
        int_bw = find_min_int_bw(f,i,d,min_segs);
       // generate coefficient lookup table
        table = gen_table(f,i,d,int_bw,min_segs);
       // generate ASC code
        gen_ASC(f,i,d,int_bw,table,min_segs);
end
```

For this implementation we use uniform bitwidths for the internal datapath fraction bitwidth. This minimum bitwidth is found using a binary search method.

4.2 ASC Code Generation and Optimizations

ASC, A Stream Compiler [10], provides a programming environment for FPGAs. ASC code makes use of C++ syntax and ASC semantics which allow the user to program on the architecture-level, the arithmetic-level and the gate-level. As a consequence ASC code provides the productivity of high-level hardware design tools and the performance of low-level optimized hardware design. ASC provides types and operators to enable research on custom data representation and arithmetic. Currently supported types are HWint, HWfix and HWfloat. For this paper we use the HWfix type which is defined as follows:

```
HWfix x(TMP,size,fract_size,sign_mode);
```

All results in this paper are given for sign-magnitude representation which makes most sense for range reduction. ASC provides operator-level optimizations of area, latency, and throughput, which is refered to below as the optimization mode.

As a result of this work, ASC provides a function evaluation library call of the form:

```
y = HWsin(x);
```

In order to create an optimizing function evaluation library, we utilize MATLAB to generate a vast amount of ASC code. This ASC code forms a two-dimensional matrix, which is indexed by range and precision of the argument to the function evaluation call. Each matrix entry consists of a pointer to an ASC function which is called for the particular input x.

Note that for each function we determine two design selection matrices: for minimum area (Figure 13) and for minimum latency (Figure 14) as shown in Section 5. The `HWsin(x)` call indexes into the matrix to find the optimized ASC implementation. For instance, from Figure 13, a $\sqrt{x}$ design with 12-bit range and 16-bit precision, the smallest implementation would be *tp3*.

The function evaluation code, for example for $\log(x)$, then indexes into the matrix of function pointers (`HWlog_matrix`) and accesses the correct function based on input range and precision:

```
HWfix &HWlog( HWfix &x ){
  return HWlog_matrix[x.range][x.precision](x);
}
```

All together, the 2000 lines of MATLAB code generates 300,000 lines of ASC code, resulting in over 1000 designs with a total of over 40 million Xilinx equivalent circuit gates.

5 Results

The method in Section 4.2 produces 1000 distinct designs, which are placed and routed on a Xilinx Virtex-II XC2V6000-6 device. These result in over 150 graphs/figures. We summarize all the results in two matrices which show the Pareto-optimal solutions in Figure 13 for area and Figure 14 for latency. In essence, these matrices tell us, for each combination of range and precision of the input, which method to use for the three functions to get the minimum area or latency. Note that we use the term range reduction to also include range reconstruction.

The remaining result figures show a sample of the graphs that we have to consider for making the decisions presented in the matrices above. For latency optimization, ASC simply generates a purely combinatorial circuit with no pipeline registers. For area optimization, a single adder is used repeatedly for multiplication. Note that latency in our work refers to the combinatorial delay.

To decide when to use range reduction, we consider as an example $\sin(x)$ and $\log(x)$ evaluated with range reduction (WRR) and without range reduction (WOR). Figures 5, 6, 7 and 8 show the area and latency results. The results cover various ranges with the precision fixed at eight bits. In the case of evaluating with WOR, we approximate the function over the entire user defined range with the given methods (*tp2*,*tp3*,*tp4*). With *po*, it is difficult to approximate with WOR, because of the non-linearities of the functions over the user defined range. Considering the area for $\sin(x)$, WOR has a lower LUT usage than WRR when the range is less then six bits. In the case of $\log(x)$, we observe that even for ranges as low as two bits, the LUT usage for WOR is significantly higher than WRR and this gap increases with range. This is due to the non-linear region of

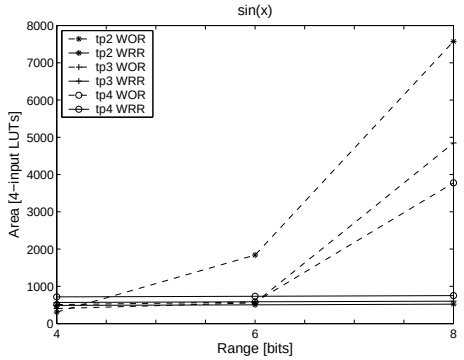

Figure 5: *Area for* $\sin(x)$ *with precision of eight bits for different methods with (WRR, solid line) and without (WOR, dashed line) range reduction, with the designs optimized for area.*

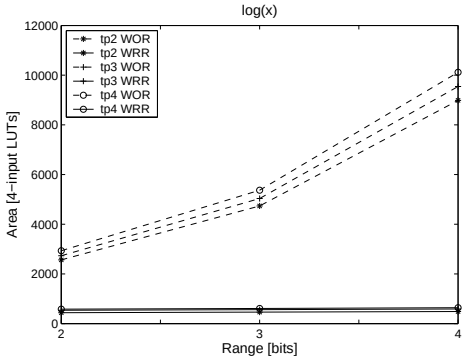

Figure 7: *Area for* $\log(x)$ *with precision of eight bits for different methods with (WRR, solid line) and without (WOR, dashed line) range reduction, with the designs optimized for area.*

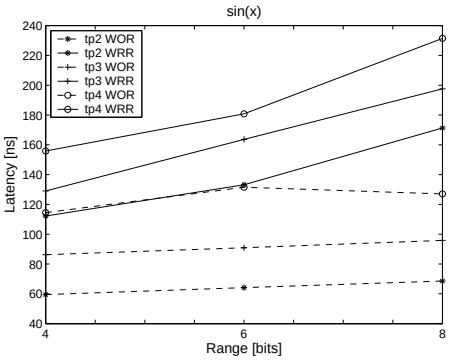

Figure 6: *Latency for* $\sin(x)$ *with precision of eight bits for different methods with (WRR, solid line) and without (WOR, dashed line) range reduction, with the designs optimized for latency.*

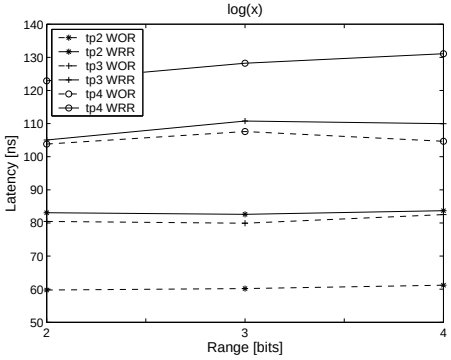

Figure 8: *Latency for* $\sin(x)$ *with precision of eight bits for different methods with (WRR, solid line) and without (WOR, dashed line) range reduction, with the designs optimized for latency.*

$\log(x)$ near zero which requires more segments to approximate with WOR. There is little change in the area results for WOR with varying ranges. This is because there are small changes in the range reduction circuits (modulus for $\sin(x)$ and barrel shifter for $\log(x)$) for such small bit differences. Looking at the latency results for $\sin(x)$ and $\log(x)$, WOR is always faster than the corresponding WRR method. This is due to the absence of the range reduction step.

Figures 9 and 10 show the area cost of range reduction for $\sin(x)$ and $\log(x)$, with the approximation circuit implemented using *tp3*. The lower part of the bars shows LUTs used for function evaluation, and the small upper part shows the LUTs used for range reduction. On average, the percentage area used by range reduction for $\sin(x)$ and $\log(x)$ are 41% and 22% respectively. The cost of range reduction with increasing range is more significant for $\sin(x)$, due to the use of the modulus operation which incorporates a divider. In contrast, $\log(x)$ uses a barrel shifter to perform range reduction.

Figures 11 and 12 highlight the area and latency tradeoffs of various functions and methods, when we consider the range while keeping the precision fixed at eight bits. The area results indicate that generally, the *po* and *tp4* methods occupy the largest area due to their large polynomials. The area requirements for $\sqrt{x}$ is the largest, because the approximation interval is slightly more non-linear than the other two functions.

Considering the latency results, we observe that $\sin(x)$ is significantly slower, which is due to the increase in the divider circuit with bitwidth. By looking at these figures along with the other figures which do not fit into this paper, we are able to create the resulting matrices in Figure 13 and Figure 14.

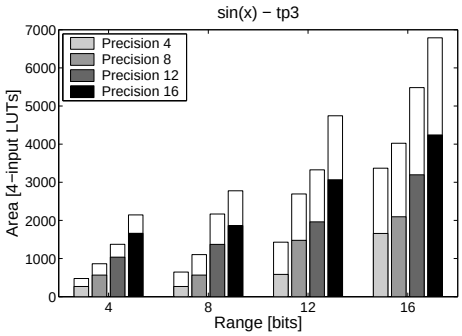

Figure 9: *Area cost of range reduction (upper part) for* $\sin(x)$ *implemented using* tp3 *with the designs optimized for area.*

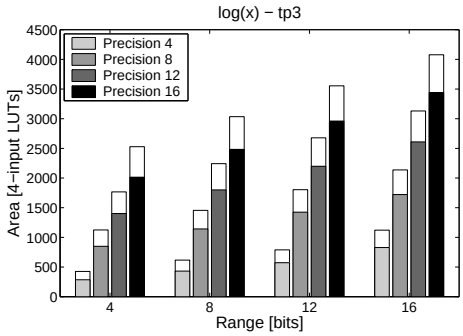

Figure 10: *Area cost of range reduction (upper part) for* $\log(x)$ *implemented using* tp3 *with the designs optimized for area.*

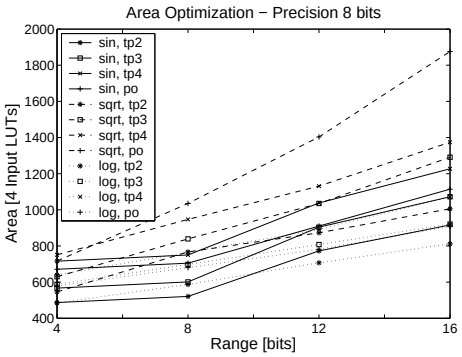

Figure 11: *Area versus range for all three functions using different methods with the precision fixed at eight bits optimized for area.*

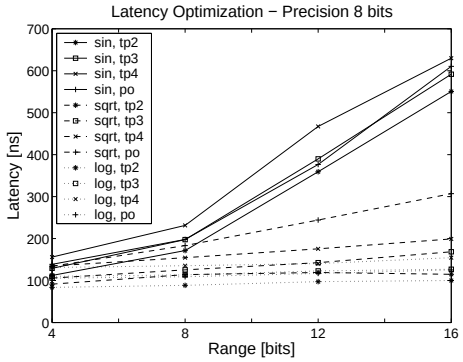

Figure 12: *Latency versus range for all three functions using different methods with the precision fixed at eight bits optimized for latency.*

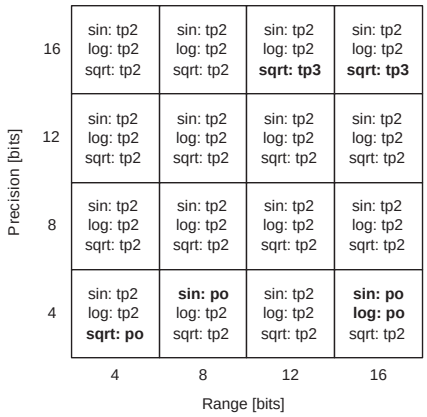

Figure 13: *Area matrix showing, for each input range/precision combination, the design with minimum area.*

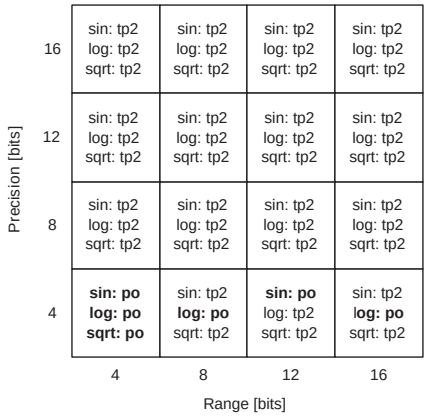

Figure 14: *Latency matrix showing, for each input range/precision combination, the design with minimum latency.*

6 Conclusions

This paper shows the design space exploration of function evaluation with custom range and precision. The result is an optimizing function evaluation library for A Stream Compiler, ASC. The novel aspect of this work is the method and range reduction selection based on range and precision of the input/output variables. The detailed research issues to which this paper contributes are:

- exploration of the area and speed tradeoffs of function evaluation with and without range reduction, using ASC;
- given a function, its input/output range/precision, and an optimization metric, we automate the decision about whether range reduction helps to optimize the metric by pre-computing a large library of function evaluation generators;
- given the above and a decision regarding range reduction, we automate the decision which is the optimal evaluation method to use by looking at the range/precision/method space and selecting the best method in each case;
- given the method, we automate the decision about which bitwidths and number of polynomial terms to use by constructing the function evaluation generators via MATLAB simulation and computation.

In addition, we show the productivity which we obtain from combining MATLAB with ASC, exploring over 40 million Xilinx equivalent circuit gates in a relatively short amount of time.

Future work includes implementing other elementary functions, exploring other evaluation methods such as rational approximation and symmetric table addition methods, and utilizing block RAMs and embedded multipliers. We also hope to optimize our designs further by employing non-uniform bitwidth minimization techniques [3]. The final objective is to progress towards a fully automated library that provides optimal function evaluation hardware units given input/output range and precision.

Acknowledgment

The authors thank Ray C.C. Cheung and David J. Pearce for their assistance. The support of Xilinx Inc. and the U.K. Engineering and Physical Sciences Research Council (Grant number GR/N 66599, GR/R 55931 and GR/R 31409) is gratefully acknowledged.

References

[1] W.J. Cody and W. Waite. *Software Manual for the Elementary Functions*. Prentice Hall, 1980.

[2] D. Defour, P. Kornerup, J.M. Muller, and N. Revol. A new range reduction algorithm. In *Proc. IEEE Asilomar Conf. on Sig., Syst. and Comput.*, volume 2, pages 1656–1660, 2001.

[3] A. Abdul Gaffar, O. Mencer, W. Luk, and P.Y.K. Cheung. Unifying bit-width optimisation for fixed-point and floating-point designs. In *Proc. IEEE Symp. on Field-Prog. Cust. Comput. Mach. (FCCM)*, 2004.

[4] J.E. Stine and M.J. Schulte. The symmetric table addition method for accurate function approximation. *J. of VLSI Sig. Proc.*, 32(2):167–177, 1999.

[5] D. Lee, W. Luk, J.D. Villasenor, and P.Y.K. Cheung. Hierarchical segmentation schemes for function evaluation. In *Proc. IEEE Int. Conf. on Field-Prog. Tech. (FPT)*, pages 92–99, 2003.

[6] D. Lee, O. Mencer, D.J. Pearce, and W. Luk. Automating optimized table-with-polynomial function evaluation for FPGAs. In *Proc. Int. Conf. on Field-Prog. Logic and App. (FPL)*, LNCS 3203, pages 364–373. Springer-Verlag, 2004.

[7] V. Lefévre and J.M. Muller. On-the-fly range reduction. *J. of VLSI Sig. Proc.*, 33:31–35, 2003.

[8] R.C. Li, S. Boldo, and M. Daumas. Theorems on efficient argument reductions. In *Proc. IEEE Symp. on Comput. Arith.*, pages 129–136, 2003.

[9] O. Mencer and W. Luk. Parameterized high throughput function evaluation for FPGAs. *J. of VLSI Sig. Proc. Syst.*, 36(1):17–25, 2004.

[10] O. Mencer, D.J. Pearce, L.W. Howes, and W. Luk. Design space exploration with A Stream Compiler. In *Proc. IEEE Int. Conf. on Field-Prog. Tech. (FPT)*, pages 270–277, 2003.

[11] M.J. Schulte and E.E. Swartzlander Jr. Hardware designs for exactly rounded elementary functions. *IEEE Trans. on Comput.*, 43(8):964–973, 1994.

[12] J.M. Muller. *Elementary Functions: Algorithms and Implementation*. Birkhauser Verlag AG, 1997.

[13] J.S. Walther. A unified algorithm for elementary functions. In *Proc. AFIPS Spring Joint Comput. Conf.*, pages 379–385, 1971.