Automated Optimization of Look-Up-Table Implementation for Function Evaluation on FPGAs

L. Deng[a], C. Chakrabarti[a], N. Pitsianis[bc], X. Sun[b]

[a]School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ, 85287, U.S.A.;
[b]Department of Computer Science, Duke University, Durham, NC, 27708, U.S.A.
[c]Department of Electrical and Computer Engineering, Aristotle University, Thessaloniki, 54124, Greece

ABSTRACT

This paper presents a systematic approach for automatic generation of look-up-table (LUT) for function evaluations and minimization in hardware resource on field programmable gate arrays (FPGAs). The class of functions supported by this approach includes sine, cosine, exponentials, Gaussians, the central B-splines, and certain cylinder functions that are frequently used in applications for signal and image processing and data processing. In order to meet customer requirements in accuracy and speed as well as constraints on the use of area and on-chip memory, the function evaluation is based on numerical approximation with Taylor polynomials. Customized data precisions are supported in both fixed point and floating point representations. The optimization procedure involves a search in three-dimensional design space of data precision, sampling density and approximation degree. It utilizes both model-based estimates and gradient-based information gathered during the search. The approach was tested with actual synthesis results on the Xilinx Virtex-2Pro FPGA platform.

Keywords: FPGAs, function evaluation, look-up table, automatic generation, resource minimization

1. INTRODUCTION

On-chip function evaluation is an indispensable component for various embedded systems utilizing modern integrated circuitry technology and techniques for real-time digital signal and image processing (DSIP) or on-line data processing. We are interested particularly in function evaluation on field programmable gate array (FPGA) platforms, which offers the special means to resolving customization issues on size, weight and energy consumption as well as processing speed. FPGA implementation of function evaluation is more involved than its software counterpart on a commercially off the shelf (COTS) computer. We present in this paper an approach to providing automatic aid to such a complex design and development process. This approach is developed toward a few objectives. The first objective is to provide the flexibility of evaluating functions of a broad family, subject to certain important constraints. The other objectives are more common, namely, to increase processing efficiency or throughput and decrease the use of hardware resources, such as slices and block random access memories (BRAMs). The primary constraints are specified explicitly in terms of thresholds, or inequalities, on accuracy in function evaluation and on available hardware resources. In accuracy analysis, both analytical truncation errors and architectural truncations errors, a.k.a. the round-off errors, are taken into consideration.

We review first and briefly some existing approaches for function evaluation that are precursors to the approach introduced here. There is the function-specific design approach. For example, the CORDIC algorithms[1–3] for trigonometric and hyperbolic functions can be implemented in FPGA very efficiently and economically using fixed-point (FXP) data representation and arithmetic. For other functions, or the same set of functions in floating-point (FLP) data representation and arithmetic, one may use the *direct* look-up table (LUT) approach, in which function values are pre-calculated at certain sample points and stored in memory. The function values at non-sample points are approximated by that at nearby sample points. Additional means are used to

Further author information: (Send correspondence to L. Deng)
L. Deng, C. Chakrabarti: E-mail: {ldeng2, chaitali}@asu.edu
N. Pitsianis, X. Sun: E-mail: {nikos, xiaobai}@cs.duke.edu

reduce power consumption and the use of resources for memory and processing elements such as adders and multipliers. Such means are once again specific to a small set of functions and restricted to fixed point arithmetic operations.[1–3] The direct LUT approach also imposes a tradeoff between accuracy and resource use. The approximation accuracy depends on the sampling strategy. As the sampling density increases to improve approximation accuracy, the use of memory space for the LUT and resource for table lookup increase. To address this issue, a fair amount of work has been on combining the use of LUTs with local correction on non-sample points via polynomial approximation. For example, the use of LUT is combined with a second-order polynomial approximation in,[4] in single-precision FLP data representation and arithmetic. Polynomial approximation is primarily based on the Taylor expansion of sufficiently smooth functions.[2,5–10] The polynomial evaluation is via efficient use of adders and multipliers. For example, a single multiplier is used in[6] for evaluating power functions, x^p.

Evaluating various functions such as the exponential, Bessel and Gamma functions has been considered and implemented in.[7] Only analytical truncation errors in function approximation are considered. Round-off errors, together with the truncation errors, are taken into considerations in[5,8] for a small set of special functions such as the exponential, the natural logarithm, and base-2 power functions. The LUT size and polynomial order have been further reduced via additional techniques such as data folding, scaling and geometric tiling in,[9,10] for certain special functions. A recent work in[3] provides a systematic way to support trigonometric, logarithmic, sigmoidal and square root functions by reducing LUT size with non-uniform sampling, with FXP arithmetic. The data precision is determined to meet a specified accuracy requirement.

The approach introduced in this paper integrates many advantages in the precursor approaches and makes further advances toward automatic optimization in accurate and efficient function evaluation on FPGA platform for a broader family of sufficiently smooth functions. The basic approach is based on the combined use of LUTs and Taylor polynomial approximation. The central design issue is to coherently determine the sampling density, the degree of Taylor polynomials and the data representation precision. Formally, this design process involves a systematic search in a configuration space of three dimensions along sampling density (the LUT length), approximation degree (the LUT width) and the bit-width in data representation (the LUT depth). Both FXP and FLP data representations and arithmetic are supported. The degree of Taylor polynomials affects the use of resource for adders and multipliers in polynomial evaluation. In accuracy estimates, both analytical truncation errors and architectural round-off errors are taken into consideration. The output of the LUT design includes close estimates on efficiency, accuracy, the use of area and memory space at seven LUT configurations. The central configuration is ideally the best option for the (currently) specified objective and constraints. The estimates at the six neighbor configurations, two neighbor configurations along each dimension, provide the gradient information that can be very useful for iterative integration with other modules beside function evaluation. The output design is ready for generation of the LUT tables and the code in hardware description language (HDL) for polynomial approximations and their mapping onto the FPGA platform.

The rest of the paper is organized as follows. In Section 2 we describe the basic analysis on analytical trunction errors as well as on architectural truncation errors due to finite-precision data representation and arithmetic. We describe also the basic models for estimating the resource consumption in relation to the LUT-based function evaluation. In section 3 we introduce the systematic arrpoarch in its design flow and implementation. We demonstrate in Section 4 the effectiveness of the approach and associated tools.

2. BACKGROUND

2.1 Function approximation by truncated Taylor expansion

We describe first the role of truncated Taylor expansions in the LUT-based evaluation of a smooth function $f(x)$,

$$T_n(x) = f(c) + \sum_{k=1}^{d} \frac{f^{(k)}(c)}{k!}(x-c)^k, \tag{1}$$

where $f(c)$ and $f^{[k]}(c)$ are the values of the function and its derivatives at a reference point c, x is the evaluation point, d is the degree of the Taylor polynomial. The function f is assumed smooth enough at c. The truncation

error can be described as follow

$$R_n(x) = \frac{f^{(d+1)}(\xi)}{(d+1)!}(x-c)^{d+1}, \tag{2}$$

where ξ is some point between the reference point c and the evaluation point x.

This approximated function evaluation can be carried out by looking up into a table for the coefficients associated with $(x-c)^k$ followed by the evaluation of the polynomial in $x-c$. The look-up table contains m chosen reference points or sample points in a pre-specified range of the variable x. When the samples are equally spaced, the space between two consecutive sample points is called the *sampling interval*, denoted in this paper by Δ. Associated with every sample point is a row of $d+2$ entries for the sample point and the coefficients at this point associated with $(x-c)^k$, $k = 0 : d$. We may say that the LUT is of length m and width $(d+2)$. When $d = 0$, the LUT is direct, rendering piecewise constant approximation, without requiring additional calculation.

2.2 Round-off errors

In addition to the analytical truncation error in the Taylor polynomial, as described in (2), one must consider also round-off errors due to architectural truncations in data representation and arithmetic operations in finite precision. For function approximation with Taylor polynomials, we may put round-off errors into two groups based on where they are introduced. The round-off errors in the first group are in the LUT entries. These errors depend on both the polynomial coefficients and the chosen data representation in FXP or FLP format as well as in bit-length. We view the bit-length as the third dimension or depth of the LUT.

2.2.1 Round-off errors in the LUT entries

The round-off errors in the LUT in FLP representation have different properties from that in the FXP representation. The FLP format can be specified in two bit fields, FLP(E, M), where E is for the number of exponent bits and M for the mantissa bits. The FXP format can be specified by FXP(I, F), where I is for the number of integer bits and F for the fraction bits. At the LUT instantiation, each LUT entry is converted into a chosen finite precision format. The errors in an efficient β in the corresponding finite representation can be described as follows,

$$(\beta)_{\mathrm{FL}} = \beta \cdot (1+\varepsilon_r), \quad \text{where} \quad |\varepsilon_r| \le 2^{-\mathrm{M}}$$
$$(\beta)_{\mathrm{FX}} = \beta + \varepsilon_a, \quad \text{where} \quad |\varepsilon_a| \le 2^{-(\mathrm{F}+1)}$$

We note that the round-off errors in the FLP format are relative and that in the FXP format are absolute. The round-off errors in the LUT entries will be accumulated in the evaluation of a Taylor polynomial.

2.2.2 Round-off errors during polynomial evaluation

A polynomial can be evaluated efficiently using the Horner's rule. Let $p_d(x)$ be a polynomial of degree d with the variable translated to a sample point c,

$$p(x) = p_0 + p_1 * (x-c) + p_2 * (x-c)^2 + \cdots + p_d * (x-c)^d, \quad p_d \neq 0.$$

The evaluation procedure by the Horner's rule is as follows,

$$\begin{aligned} q_d &:= p_d; \\ q_{k-1} &:= q_k * (x-c) + p_{k-1}; \quad k = d : -1 : 1, \end{aligned}$$

the function value $p(x)$ is rendered by q_0. With the finite-precision arithmetic , we have

$$\begin{aligned} (q_d)_{\mathrm{FL}} &:= (p_d)_{FL}; \\ (q_{k-1})_{\mathrm{FL}} &:= [(q_k)_{\mathrm{FL}} * (x-c)(1+2\epsilon_k) + (p_{k-1})_{\mathrm{FL}}](1+\eta_k); \quad k = d : -1 : 1, \end{aligned}$$

where ϵ_k represent the round-error introduced in the subtraction $x-c$ followed by the multiplication and η_k is the error introduced in the summation at step k. One shall notice not only the occurrence of the errors but also their propagation during the accumulation process.

We may determine first the LUT entries based on the analytical truncation error estimates. Once the sampling interval, the approximation degree, the data format and the numerical values of the entries become avalable, we can estimate the accumualted round-errors. This estimation process can be implemented in MATLAB. We omit the detail. The total round-errors can be reduced, when necessary, by changing the LUT depth, length or width. The goal is to meet the accuracy requirement with both analytical truncation errors and the accumulated round-errors taken into consideration.

2.3 Estimation of resource usage for LUTs

On FPGA platforms, area and memory consumption for a LUT depend on the approximation degree d, sampling interval Δ and data representation. Memory space for a LUT increases with each of the three LUT dimensions. The last two dimensions also affect the area needed for polynomial evaluation. To avoid time-consuming synthesis, placement and routing (P&R) procedure, we have developed models for estimating the area in terms of the number of slices and the memory space in terms of the number of block RAMs, on Xilinx Virtex-2Pro FPGA.[11]

In estimation of slice usage, we assume a fully pipelined implementation for Taylor polynomial evaluation via the Horner's rule. Specifically, the hardware consists of d adders (subtractors), d multipliers and a memory address generator. The address generator is treated as an adder, a float-to-fixed conversion (only needed for floating point data), a shifter and a 'round' function. The total number of slices can be estimated as follows,

$$\text{Nslices}_{\text{total}} = \alpha(\text{utilization}) \times \sum_{\text{comp.}} \mathit{Slice}(\text{comp.}), \tag{3}$$

where the components include adders(subtractors), multipliers, and rounding unit, and α is an empirical scaling function that describes how well the slices are utilized in the polynomial evaluation design. The detail can be found in.[11]

In estimation of memory space in terms of the number of Block RAMs, we denote by m m the number of sampling points and d the degree of the Taylor polynomial. The lookup table contains m rows and $d+2$ columns, with the i-th row contains the information at the i-th sample point. We store each column in a separate block RAM (BRAM) so that once the memory address is generated, all the polynomial coefficients can be fetched in the same clock cycle. The 18K-bit BRAM supports limited configurations. Suppose a BRAM configuration can take p entries and q bits per entry. There are (E+M+1) bits per entry in FLP representation. We need $\lceil \frac{m}{p} \rceil \cdot \lceil \frac{E+M+1}{q} \rceil \cdot (d+2)$ BRAMs in total for the lookup table.

3. LUT DESIGN PROCEDURE

We give first a brief description of the design procedure at the top level. We search in the three-dimensional design space for a LUT configuration with minimal data precision (as the first priority) and minimal polynomial degree (the second priority), subject to the accuracy requirement. Next, we maximize the sampling interval, without compromising the accuracy, to reduce memory usage. In order to make the search efficient, we use the estimation models and tools we have developed. We illustrate the design procedure with the FLP format.

Figure 1 shows the block diagram of the LUT automation flow. At the input are descriptions for the kernel function to be implemented, accuracy bound, and thresholds on area and memory. The input is provided through a graphical user interface (GUI). The output contains a central LUT configuration and six neighbor configurations. It also contains an indicator on whether or not the central configuration is a feasible solution. The initial values of the parameters such as for data precision (E, M), polynomial degree d and sampling interval Δ are determined. The procedure searches among all the feasible solutions that satisfy accuracy requirement for a design with minimal resource usage. It searches along data precision and polynomial order separately and in iterative fashion. At each iteration, there is a check on whether or not the accuracy is maintained. The rest of the section elaborates the details of the design blocks.

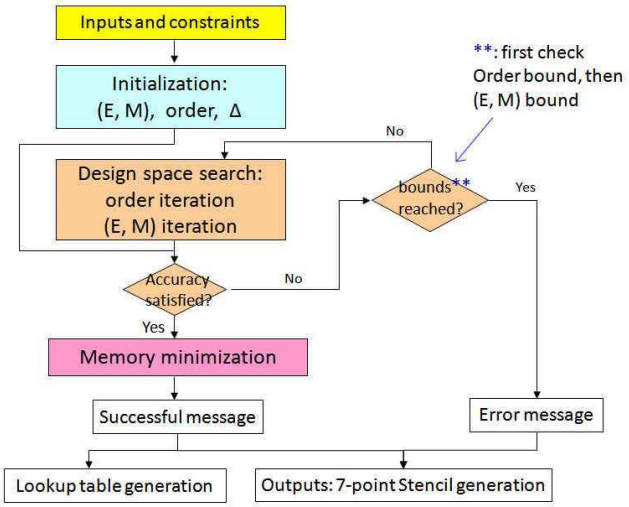

Figure 1. General flow of LUT automation tool.

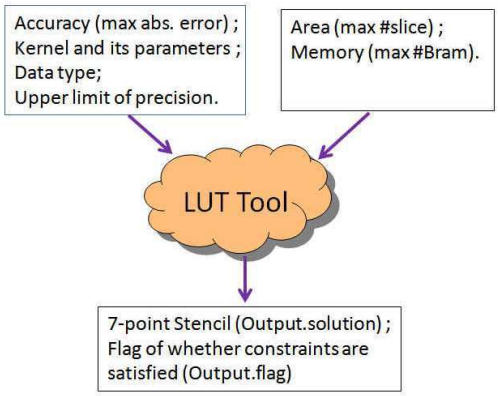

Figure 2. Input and Output for LUT automation tool.

3.1 Input and Output

Figure 2 shows the detail in the input and output. With the aid of the GUI, the user specifies the required accuracy in terms of tolerance on the absolute error, the function to be implemented and its parameters, the data type (FL or FX), the upper limits on data precision (max (E, M) or (I, F)), the area (max number of slices) and the memory (max number of BRAMs). The special functions supported in the current system library include the exponential $a \cdot e^{-b \cdot x}$, Gaussian $a \cdot e^{-b \cdot x^2}$, zero-order Bessel function of the first kind $J_0(x)$, powers of Sinc $sinc^n(x)$, central B-splines $\beta^n(x)$, sine, cosine and atan(x). Here 'a', 'b' and 'n' are user-specified kernel parameters. These functions have special roles in many applications and posses special structures that can be further exploited in the LUT-based evaluation approach. For other sufficiently smooth functions not in the library, the LUT generation tool uses the symbolic computation tools to obtain their derivatives.

At the ouput, the design system generates multiple configurations in a 7-point stencil, see Figure 3. Each solution configuration consists of design parameters such as data precision, polynomial degree, sampling interval, the corresponding area and memory estimates of the hardware implementation, and the estimated accuracy. The area and memory estimates are based on the models in Section 2.3, the accuracy esimates are in terms of the error upper bounds derived from the equations in Section 2. The errors between the hardware simulation results and the Matlab double-precision reference results are also provided at simulation points in a denser distribution than that of the LUT. The center configuration is found by the search as one using minimal resouce. There are two neighbor solutions along each of the three LUT dimensions in data format, polynomial degree and sampling density. The LUT at the central solution is fully provided.

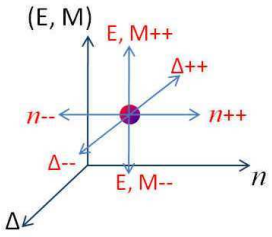

Figure 3. Generation of 7-point Stencil.

3.2 Parameter Initialization

The procedure for parameter initialization computes the upper bounds and initial values for data precision (E, M), polynomial order d and sampling interval Δ. First the bound and initial value of data precision is computed. The user provides the upper limits for data precision using the GUI. With Xilinx IP cores, there are additional limitations on the data precision. For instance, in FLP format, the ranges of M and E are not independent. For instance, if $E = 6$, M can change from 4 to 29; The initial value of M is calculated by substituting the accuracy requirement into the error bound equations in Section 2. The initial value of E is set to be 4, which is the minimal bits allowed by Xilinx IP. We check also on the maximum internal value for E.

The dynamic range of a free variable can be provided by the user through the GUI. For certain functions used in many applications, such as Gaussians and the central B-splines used for non-uniform Fourier transforms, we provide automatic estimate of the dynamic range. We note first that the dynamic range is not irrelevant to the data precision. For a given data precision, one shall determine the threshold η on "machine zeros" $\max_{\eta} 1+\eta = 1$. For a decaying function, we determine the "vanish zone", $\min_{\tilde{x}} f(\tilde{x}) \leq f(0)\cdot\eta$, where $f(0)$ is assumed the maximal value of the function. The LUT tool sets the input dynamic range as $(0, \tilde{x})$.

The upper bound on Taylor polynomial degree is calculated based on Eqn. 3 in Section 2.3. The initial degree is set to 0. The bound on the sampling interval Δ is calculated last. As described in Section 2.3, once we know the configurations of block RAM, the data precision and the polynomial order, we can calculate the maximum number of entries, or the maximum number of sampling points. We divide the input dynamic range by the number of sampling points and round the result to the nearest power of 2, which is the minimal Δ. The initial value for sampling interval is set to be this minimal value.

3.3 Search for minimal-resource solutions

Once the area and memory constraints are translated to the upper bounds of data precision and polynomial order as described earlier, we can search within a bounded three-dimensional (3-D) design space spanning (E, M), d and Δ to find the 'minimal-resource design' under the accuracy requirement. Keep in mind that the design process at the top level is iterative. At each iteration, there is a specific set of parameters for data precision and polynomial degree.

The search at each iteration operates in the following way. We start with the minimum sampling interval Δ, within the memory space allowed. If the accuracy requirement is not met, we increase the polynomial order and adjust the sampling interval accordingly. When the round-off errors become dominant, we change (E,M). If no feasible configuration is found, estimates of additional resource are provided to the user. When a feasible solution is found, we proceed to minimizing the degree or maximizing the sampling interval under the accuracy constraint. Figure 4 shows the block diagram of the design space search flow. Note that if the data precision (E, M) has to be increased, then the input dynamic range and the bounds and initial values for polynomial order and sampling interval need to be re-adjusted. The memory minimization procedure, shown in the right part of Figure 4, is visited only when a feasible solution is found. In each iteration through this loop, it increases the sampling interval till the accuracy condition is violated.

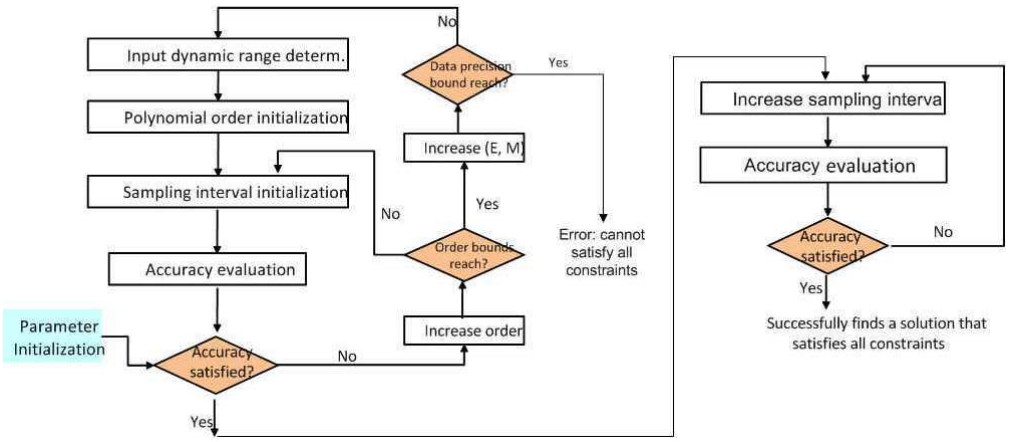

Figure 4. Detailed flow for the design space search.

4. EXPERIMENTAL RESULTS

We demonstrate in this section the working of the LUT automation tool with three groups of experiments. First, we present the results of two convolution kernels, Gaussian and Bessel, under different area, memory and accuracy constraints. Next we show how the rendered solutions in 7-point stencil provide gradient information for adjustment in resource allocation at a higher design level. Third, we introduce the use of sampling points and LUTs for polynomial evaluation with reduced use of hardware resource in total.

4.1 Exploration Tool Results

We illustrate with two special functions that are frequently signal and image processing: $J_0(x)$ the zero-order Bessel function of the first kind, and $a \cdot e^{-b \cdot x^2}$ the Gaussian function, say, with parameter a=2 and b=1/32. The upper limit on data precision has been set to single-precision floating point: FL-(8, 23). The other constraints such as max slices, max BRAMs and accuracy requirement are specified in each case.

Table 1. LUT designs for Bessel function J_0

Constraints	Slice	2000	2000	2000	2000	2000	2000	1000	1500	2000
	BRAM	10	10	10	5	15	25	10	10	10
	Accuracy	1e-5	1e-6	1e-7	1e-7	1e-7	1e-7	1e-7	1e-7	1e-7
Design Results	FL-(E,M)	(6,19)	(6,22)	(6,23)	(6,23)	(6,23)	(6,23)	(6,23)	(6,23)	(6,23)
	order	1	2	2	2	2	1	1	2	2
	Δ	2^{-7}	2^{-5}	2^{-8}	2^{-5}	2^{-8}	2^{-10}	2^{-8}	2^{-5}	2^{-5}
	Estim.Slice	839	1484	1534	1534	1534	953	953	1534	1534
	Synth.Slice	783	1382	1444	1444	1444	891	891	1444	1444
	Error	7.15%	7.38%	6.23%	6.23%	6.23%	6.96%	6.96%	6.23%	6.23%
	Estim.BRAM	3	4	8	4	8	24	6	8	8
	Synth.BRAM	3	4	8	4	8	24	6	8	8
	Max Sim.Err	3.68e-6	4.76e-7	1.35e-7	3.61e-7	1.35e-7	1.33e-7	9.28e-7	1.35e-7	1.35e-7

Tables 1 and 2 show the minimal-resource solutions for the two functions at 9 different constraint specifications. Columns 3-5 correspond to the case where the area and memory constraints are the same but the accuracy requirement is different. As the accuracy requirement increases, the data precision (E,M) and polynomial order increases, thus consuming more slices. Also, as the accuracy increases from 10^{-5} to 10^{-6} for J_0 (Table 1), the order increases and Δ increases from 2^{-7} to 2^{-5} due to the memory minimization. Unfortunately, this is not reflected in the number of BRAMs which increase because different order of derivatives are stored in separate BRAMs. As the accuracy further increases to 10^{-7}, for J_0 example, the order does not increase and so Δ reduces to 2^{-8}.

Table 2. LUT designs the Gaussian function $2 \cdot e^{-x^2/32}$

Constraints	Slice	2000	2000	2000	2000	2000	2000	1500	2500	3500
	BRAM	10	10	10	5	15	25	10	10	10
	Accuracy	1e-5	1e-6	1e-7	1e-7	1e-7	1e-7	1e-7	1e-7	1e-7
Design Result	FL-(E,M)	(6,17)	(6,21)	(6,23)	(6,23)	(6,23)	(6,23)	(6,23)	(6,23)	(6,23)
	order	2	2	3	3	3	2	2	3	3
	Δ	2^{-3}	2^{-3}	2^{-4}	2^{-3}	2^{-4}	2^{-5}	2^{-4}	2^{-4}	2^{-4}
	Estim.Slice	1103	1436	2111	2111	2111	1534	1534	2111	2111
	Synth.Slice	1146	1352	1998	1998	1998	1444	1444	1998	1998
	Error	-3.75%	6.21%	5.66%	5.66%	5.66%	6.23%	6.23%	5.66%	5.66%
	Estim.BRAM	4	4	10	5	10	16	8	10	10
	Synth.BRAM	4	4	10	5	10	16	8	10	10
	Max Sim.Err	1.03e-5	2.12e-6	1.33e-7	2.32e-7	1.33e-7	1.37e-7	3.20e-7	1.34e-7	1.34e-7

In columns 6-8, area and accuracy constraints are the same but BRAM memory increases. Since the data precision, which is decided by the accuracy is the same, and there is more available memory, the sampling interval decreases. Also, the polynomial order decreases and so the overall designs consume more BRAMs but less slices. In columns 9-11, memory and accuracy constraints are kept the same and the number of slices increase. The data precision remains the same and for J_0 (Table 1), the polynomial order increases and so Δ increases. The designs consumes more slices because of higher order and more BRAMs because of overhead of data arrangement in BRAMs.

In all the experiments, accuracy requirement has been satisfied. We observe that the consumed slices and BRAMs, which are estimated by the LUT tool, are close to that with the actual synthesis data. The average error is 6.13% in the number of slices, and there is no mismatch between the estimated and synthesized number of BRAMs. Note that since the fixed point memory address generator (Section 2.3) is the same for all the presented experiments, and to better demonstrate the effect of polynomial orders on the area results, we have ignored the number of slices due to memory address generator in Tables 1 and 2.

4.2 Solution rendering with gradient information

Next we show how the gradient information in the rendered 7-point stencil solutions can help guide the settings of the user-defined constraints for another pass through the LUT tool. We choose e^{-x}, the Exponential function to illustrate this. The upper limit on data precision is FL-(8, 23) and the accuracy requirement is set to 10^{-8}. The maximum slices is initially set to be 2000 and the maximum BRAMs to be 15.

Table 3. Sample of 7-point Stencil solutions for Exponential function

7-point	'Best'	E,M++	E,M–	Δ++	Δ–	order++	order–
FL-(E,M)	(6,23)	(6,24)	(6,22)	(6,23)	(6,23)	(6,23)	(6,23)
degree n	3	3	3	3	3	4	2
Δ	2^{-4}	2^{-4}	2^{-4}	2^{-3}	2^{-5}	2^{-4}	2^{-4}
Estim.Slice	2111	2190	2040	2111	2111	2687	1534
Estim.BRAM	10	10	10	5	20	12	8
Max Sim.Err	1.69e-7	1.09e-7	2.88e-7	1.17e-6	1.13e-7	1.15e-7	1.09e-5

Table 3 presents the 7-point stencil solutions. The solutions do not meet the accuracy requirement. They are not within the slice constraint either. They are within the the BRAM constraint. They indicate however that relaxing on the slice constraint may lead to a feasible solution. This is indeed the case. When the maximum slices is set to 3000 instead, all the constraints are satisfied. A minimal-resource solution is found at data precision (6, 23), degree 4, $\Delta = 2^{-4}$, with 2525 slides in estimate, and 12 BRAMs in estimate. The errors are bounded by $4.74e-8$.

4.3 Polynomial evaluation with resource reduction

So far, the Taylor polynomials are used for approximating sufficiently smooth functions. In this section, we show how to re-write polynomials in reference to pre-chosen sample points in a LUT for reduction in resource

usage. The degree is preserved in the re-writing and there is no analytical truncation errors. We illustrate this approach with the central B-spline of order n, $\beta^n(x)$, which is a piecewise polynomial. A naive evaluation of B-splines has been previously implemented in.[12]

Table 4. Design configurations for central B-splines of different orders

$\beta^n(x):\ n$		8	9	10	11	12
Naive Method	FL-(E,M)	(7,28)	(7,30)	(7,32)	(7,33)	(7,35)
	# Slice	5982	6932	8047	9012	10271
	# BRAM	8	9	10	11	12
	Max Sim.Err	5.20e-6	3.46e-6	2.20e-6	3.26e-6	2.14e-6
Re-writing w.r.t. sampling points	FL-(E,M)	(6,16)	(6,16)	(6,16)	(6,16)	(6,16)
	# Slice	3345	3726	4108	4315	4870
	# BRAM	8	9	10	11	12
	Max Sim.Err	6.00e-6	6.14e-6	6.77e-6	8.26e-6	7.01e-6

Table 4 gathers some experimental results for central B-splines of different orders, showing the comparison between the naive method and the LUT-based method. The accuracy requirement is set to be 10^{-5}. The LUT-based method has a much better control of the errors. It saves 1 exponent bit (E) and 16 mantissa bits (M) on average. The LUT-based implementations require on average 48.5% fewer slices than the naive method.

REFERENCES

[1] Muller, J. M., "Partially rounded small-order approximations for accurate, hardware-oriented, table-based methods," *Proceedings of the 16th IEEE Symposium on Computer Arithmetic - ARITH-16'2003* , 114–121 (2003).

[2] Cao, J., Wei, B. W. Y., and Cheng, J., "High-performance architectures for elementary function generation," *Proceedings of the 15th IEEE Symposium on Computer Arithmetic* , 136–144 (2001).

[3] Sasao, T., Nagayama, S., and Butler, J., "Numerical function generators using LUT cascades," *IEEE Transactions on Computers* **56**, 826–838 (2007).

[4] Pineiro, J., Bruguera, J., and Muller, J., "Faithful powering computation using table look-up and a fused accumulation tree," *Proceedings of IEEE Symposium on Computer Arithmetic* , 40–47 (2001).

[5] Tang, P. P., "Table-lookup algorithms for elementary functions and their error analysis," *Proceedings of IEEE Symposium on Computer Arithmetic* , 232–236 (1991).

[6] Takagi, N., "Powering by a table look-up and a multiplication with operand modification," *IEEE Transactions on Computers* **47**, 1216–1222 (1998).

[7] Cody, W. J. and Stoltz, L., "The use of Taylor series to test accuracy of function programs," *ACM Transactions on Mathematical Software (TOMS)* **17**, 55–63 (1991).

[8] Tang, P. P., "Table-driven implementation of the exponential function in IEEE floating-point arithmetic," *ACM Transactions on Mathematical Software (TOMS)* **15**, 144–157 (1989).

[9] Sobti, K. et al., "Efficient function evaluations with lookup tables for structured matrix operations," *Proceedings of IEEE Workshop on Signal Processing Systems* , 463–468 (2007).

[10] Bui, H. and Tahar, S., "Design and synthesis of an IEEE-754 exponential function," *IEEE Canadian Conference on Electrical and Computer Engineering* **1**, 450–455 (1999).

[11] Deng, L. et al., "Accurate models for estimating area and power of FPGA implementations," *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing - ICASSP'2008* , 1417–1420 (2008).

[12] Unser, M., "Splines: a perfect fit for signal and image processing," *IEEE Signal Processing Magazine* **16**, 22–38 (1999).