Exploiting input parameter uncertainty for reducing datapath precision of SPICE device models

Kapre, Nachiket

2013

Kapre, N. (2013). Exploiting Input Parameter Uncertainty for Reducing Datapath Precision of SPICE Device Models. 2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines, pp.189-197.

https://hdl.handle.net/10356/98267

https://doi.org/10.1109/FCCM.2013.28

Exploiting Input Parameter Uncertainty for Reducing Datapath Precision of SPICE Device Models

Nachiket Kapre
Nanyang Technological University
Singapore
nachiket@ieee.org

***Abstract*— Double-precision computations operating on inputs with uncertainty margins can be compiled to lower precision fixed-point datapaths with no loss in output accuracy. We observe that ideal SPICE model equations based on device physics include process parameters which must be matched with real-world measurements on specific silicon manufacturing processes through a noisy data-fitting process. We expose this uncertainty information to the open-source FX-SCORE compiler to enable automated error analysis using the Gappa++ backend and hardware circuit generation using Vivado HLS. We construct an error model based on interval analysis to statically identify sufficient fixed-point precision in the presence of uncertainty as compared to reference double-precision design. We demonstrate 1–16× LUT count improvements, 0.5–2.4× DSP count reductions and 0.9–4× FPGA power reduction for SPICE devices such as Diode, Level-1 MOSFET and an Approximate MOSFET designs. We generate confidence in our approach using Monte-Carlo simulations with auto-generated Matlab models of the SPICE device equations.**

I. Introduction

The spatial parallelism and customizability of hardware circuits allows reconfigurable architectures such as FPGAs to offer orders of magnitude performance improvements and power reductions compared to traditional computer organizations. Datapath bitwidth is a key parameter of a reconfigurable circuit that can be tuned to achieve these savings. Arithmetic circuits with smaller bitwidth require fewer resources (LUTs, Block RAMs, DSP blocks) than ones with wider bitwidths. For example, a double-precision floating-point multiplier requires ≈3x more LUTs and ≈1.2× more DSP blocks than a corresponding 54-bit fixed-point circuit (See Table III). This customization is not easily possible on conventional architectures such as CPUs (64-bit integer datapaths) and GPUs (single or double-precision) where it is fixed at fabrication time. There are exceptions in the form of lower precision MMX instructions and smaller bitwidth embedded systems but it does not offer the complete precision flexibility possible on an FPGA fabric.

So how does one select datapath bitwidth effectively? Previous research [8], [9] has shown how to customize bitwidth (also referred to as wordlength) through a combination of empirical approaches and static interval analysis of arithmetic error properties of the implementations. The intuition behind these techniques is the observation that most arithmetic computations operating on real-world models have a limited dynamic range and only need to evaluate correctly over that range. Consequently circuit implementations may not need the full precision made available by the double-precision architectures to operate correctly. However, it has been argued that the potential benefits from these precision tweaks is limited to small, single-digit factors. Are there other sources of information that we can use to further improve the impact of precision reduction? Can this allow us to stretch the compute density advantage of fixed-point compilation beyond what is currently possible? In this paper, we propose using uncertainty in input parameters (constants) as additional information in the interval analysis process for selecting bitwidth. We observe that physical calculations such as circuit simulation are often based on process parameters that have inherent measurement limits and data-fitting noise. Circuit designers that rely on SPICE still run computations on full double-precision CPU hardware despite these input uncertainties. We build upon our existing open-source FX-SCORE [7] framework which is a high-level streaming abstraction for generating FPGA circuits. It uses Gappa++ for the error analysis backend and Vivado HLS for the hardware generation backend to simplify the bitwidth selection and implementation process. Our framework is equally applicable to other computations where input uncertainties are known in advance.

In this paper, we make the following contributions-

- Develop language and compiler extension for our open-source FX-SCORE framework that captures input uncertainty information for constants.
- Formulate an error model for datapaths with input parameter uncertainty.
- Develop Monte-Carlo simulation framework for SCORE compiler that can build confidence in our precision optimizations.
- Demonstrate FPGA resource utilization reduction (up to 16× LUT count reductions and 2.4× DSP count savings) and power usage improvements (up to 4×) across a range of benchmark circuits such as SPICE devices.

Device	Equation	Ranges	Parameters	Add	Mult	Div	Exp	Log
Diode	$I = i_{sat} \cdot (\exp^{V/v_j} - 1)$	$V \in [1e^{-6}, 0.1]$	$i_{sat} = 1e^{-3}$ $v_j = 2.58e^{-2}$	1	1	1	1	0
Level-1 MOSFET (level1)	**if** $(Vgs < vt)$ ← *cutoff* $I_d = 0$ **elseif** $(Vds > Vgs - vt)$ ← *saturation* $I_d = b \cdot (Vgs - vt)^2 / 2$ **else** ← *linear* $I_d = b \cdot (Vgs - vt - Vds/2) \cdot Vds$	$V_g \in [1e^{-6}, 2.0]$ $V_d \in [1.9, 2.1]$ $V_s \in [1e^{-6}, 0.1]$	$vt = 0.4$ $a = 12$ $b = 1e^{-3}$	5	4	0	0	0
Approx. MOSFET (approx1)	$t_1 = (Vgs > vt)?\, Vgs - vt : 0$ $t_2 = (Vgd > vt)?\, Vgd - vt : 0$ $I_d = b \cdot (t_1^2 - t_2^2)/2$	$V_g \in [1e^{-6}, 2.0]$ $V_d \in [1.9, 2.1]$ $V_s \in [1e^{-6}, 0.1]$	$vt = 0.4$ $a = 12$ $b = 1e^{-3}$	3	3	0	0	0
Approx. MOSFET (approx2)	$t_1 = \log(1 + \exp^{a(Vgs-vt)}) / a$ $t_2 = \log(1 + \exp^{a(Vgd-vt)}) / a$ $I_d = b \cdot (t_1^2 - t_2^2)/2$	$V_g \in [1e^{-6}, 2.0]$ $V_d \in [1.9, 2.1]$ $V_s \in [1e^{-6}, 0.1]$	$vt = 0.4$ $a = 12$ $b = 1e^{-3}$	5	5	0	2	2

TABLE I: SPICE Device Current Equations

II. Background

A. FX-SCORE

SCORE [3] is a high-level stream programming framework designed to compose scalable FPGA applications using stream interconnections between parallel dataflow operators. It is well suited for development of scalable FPGA implementations that exploit different forms of parallelism amenable to efficient circuit implementations. FX-SCORE [7] is an extension to the SCORE framework to support automated bitwidth selection of stream data-types using interval analysis. It integrates the proof assistant Gappa++ [1], [6] as a backend for proving error properties of the fixed-point implementation. It also generates intermediate AutoESL code for hardware compilation. It demonstrated 2-3x area savings for simple SPICE device models when compiling fixed-point circuits having equivalent error properties compared to double-precision hardware. We develop our precision analysis pass as part of the FX-SCORE compiler.

B. Gappa and Gappa++

Gappa [1] is a proof assistant capable of generating formal certificates for error bounds of floating-point and fixed- point arithmetic computations. Gappa++ [6] is an extension to Gappa that adds interval propagation support for *exp* and *log* functions at the expense of less formalism. It also permits affine analysis of expressions (which are unused in this work). A Gappa script contains a simple dataflow description of the arithmetic expression and user-supplied input bounds based on domain knowledge. A user requests Gappa to provide output bounds and also compute bounds on absolute and relative errors. An FPGA system designer can then use these bounds to figure out the required fixed-point precision to stay within error bounds.

C. Device Models

In this paper, we use a set of SPICE device models to demonstrate the impact of our compiler enhancements. We tabulate the equation complexity for these different devices in Table I. We also provide the expected input operating ranges for these different devices based on simulation scenarios for CMOS digital circuits. Our benchmarks contain a mixture of floating-point add, multiply, divide operators as well as the *exp* and *log* elementary functions. While our examples appear modest in size (handful of floating-point operations as seen in Table I), they are relatively large for the FPGA compile sweeps conducted in this work.

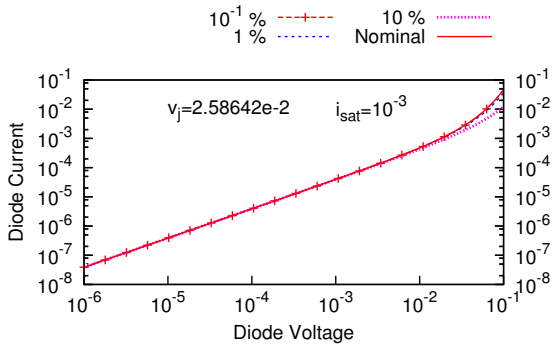


Fig. 1: I-V Characteristics of a Diode with Input Parameter Uncertainty

Let us consider the impact of uncertainty on the operation of a diode through an example. In Figure 1, we show the impact of input parameter uncertainty in v_g and i_{sat} on diode current. As we can see, uncertainty percentages below 1% make no appreciable impact (intuitively) on the result. As we will see later in Section V, that level of uncertainty corresponds to a non-trivial 3× reduction in bitwidth.

D. Previous Work

Previous academic and commercial tools and accelerators for SPICE have used approximations and precision reductions in ad-hoc manner without any a priori static analysis of potential error or quality of the resulting simulation. We do not discuss these further as an excellent review is available in our previous paper [7]. In [10], the authors develop an interval SPICE simulator that propagates input intervals through the numerical operations of the entire simulation to generate outputs with corresponding intervals. This results in a 3-4× increase in compute work and offers a substitute to the traditional exhaustive process corner simulation that may be otherwise needed. While one of these techniques is inevitable for large input uncertainty ranges, a fixed-point substitute to the traditional double-precision simulation offers a cheaper alternative in cases where the uncertainties may be

small. It should be noted that the fixed-point substitute in its present form will not offer the same quality of result as an expensive interval simulation but can replace the oblivious double-precision simulations already used in industry. In [4], the authors quantify the extent of uncertainty in model parameters due to measurement noise and data-fitting errors and observe uncertainties as large as a 1–10% percent for certain parameters. Another source of uncertainty in circuits is parameter variation which can be as high as 4% for 60-65nm Xilinx FPGA devices [5]. Previously, in FX-SCORE [7], we have demonstrated an approach using static analysis for bounding relative errors in their fixed-point implementations based purely on the intervals of the perfect inputs.

III. Uncertainty-Driven Precision-Analysis in the FX-SCORE Compiler

In this section, we explain our error formulation and explain how we adapt the FX-SCORE framework to capture and use input parameter uncertainty information.

A. Error Formulation

The FX-SCORE framework exploited the observation that double-precision implementations of computation (f_{double}) have some inherent error in their outputs when compared to infinite precision implementations (f_{ideal}) due to rounding errors in the arithmetic operations. We represent this in Equation 1, 2. This shows that if we carefully pick a fixed-point implementation (f_{fixed}) that has lower relative error than the inherent error in double-precision implementation (f_{double}), our fixed-point FPGA circuit implemented at that precision will be more accurate than the double-precision implementation. Equation 3 describes the simple optimization expression used to determine this ideal bitwidth. FX-SCORE currently implements this optimization via a brute force search across all bits within the range. As we see later in Figure 3(b), the error behavior can be erratically non-monotonic with precision preventing a trivial binary search implementation of bitwidth search.

$$err_{double}(f) = (f_{double} - f_{ideal})/f_{ideal} \tag{1}$$

$$err_{fixed(bits)}(f) = (f_{fixed(bits)} - f_{ideal})/f_{ideal} \tag{2}$$

$$\begin{aligned} \text{minimize} \quad & bits \\ \text{subject to} \quad & err_{fixed(bits)} \leq err_{double}, \\ & bits \in [16, 128]. \end{aligned} \tag{3}$$

$$err_{uncertain}(f) = (f_{uncertain} - f_{ideal})/f_{ideal} \tag{4}$$

$$err_{double(u)}(f) = (f_{double} - f_{uncertain})/f_{uncertain} \tag{5}$$

$$err_{fixed(u,bits)}(f) = (f_{fixed(u,bits)} - f_{uncertain})/f_{uncertain} \tag{6}$$

$$\begin{aligned} \text{minimize} \quad & bits \\ \text{subject to} \quad & err_{fixed(u,bits)} \leq err_{double(u)}, \\ & bits \in [16, 128], \\ & u \in [10^{-16}\%, 1\%]. \end{aligned} \tag{7}$$

However, FX-SCORE did not consider the impact of uncertainty. We consider input uncertainty in our analysis. First, we measure the relative error inherent in the infinite precision implementation in Equation 4 . This changes our reference design to be the one implementing infinite precision calculation in presence of input uncertainty. Next, we show our revised formulae in Equation 5, 6, and 7. Uncertainty (u) is a user-specified value as a percentage around the input parameter. Values of $u < 10^{-16}$ are too small to be representable as double-precision numbers and are consequently irrelevant in our implementation. The optimization described in Equation 7 is also implemented as a sweep across different bitwidths.

B. Adapting the FX-SCORE Compiler

FX-SCORE compiler accepts streaming parallel descriptions of computation and generates code for multiple backends. In Table II, we show code listings for a diode and the codes generated for the three backends that are necessary for this work. We also highlight the language modifications and adaptation required to make the system operate with uncertainty information.

- **Frontend**: The FX-SCORE compiler [7] extends SCORE input syntax by permitting the programmer to provide input intervals. We extend the FX-SCORE compiler to accept parameter inputs (boxed Line 2 in Listing 1) with suitable initialization constant values along with uncertainty interval percentage. In this example, we specify the diode junction voltage ($v_j = 2.58e^{-2}$) and saturation current ($i_{sat} = 10^{-3}$). Uncertainty information is propagated using a compiler option.
- **Gappa++**: The FX-SCORE compiler [7] generates a Gappa++ script with appropriate input intervals and type information for static analysis. We modify the Gappa++ backend script generator to convert uncertainty input into interval for the parameter inputs (boxed Line 9 in Listing 2). For example, we generate an interval ($[2.55e^{-2}, 2.60e^{-2}]$) for diode junction voltage around a mean of $2.58e^{-2}$ with an uncertainty of 1%. We also modify the relative error expressions to match the Equations 5 and 6.
- **Matlab**: FX-SCORE [7] does not provide empirical evidence to confirm the correctness of bitwidth selection using their framework. In principle, we should trust Gappa++ transforms, but interval analysis is known to generate highly pessimistic bounds. We provide a new Matlab Monte-Carlo backend (example code template in Listing 3) that allows confirmation that the bound calculations are reasonable and allows experimental validation of the fixed-point precision reduction. The Matlab code computes a pseudo-random distribution of input parameter combinations in the uncertainty interval and evaluates the dataflow equations

```
diode(
param vj=2.58e-2, param isat=1e-3
input double v [1e-7,0.1],
output double i) {
  state dfg(v):
    i = isat*(exp(v/vj)-1);
}
```

Listing 1: SCORE with Input Parameters (Uncertainty is compiler option)

```
@fx = fixed<-64,ne>;
@dbl = float<ieee_64,ne>;
i_m = isat_m*(exp(v_m/vj_m)-1);
i_u = isat_u*(exp(v_m/vj_u)-1);
i_dbl dbl = isat_dbl*(exp(v_dbl/vj_dbl)-1);
i_fx fx = isat_fx*(exp(v_fx/vj_fx)-1);
{
  v in [1e-7,0.1] /\
  vj in [2.55e-2,2.60e-2]
  |i_u| >= 0x1p-53 ->
  (i_dbl-i_u)/i_u in ? /\
  (i_fx-i_u)/i_u in ?
}
```

Listing 2: Gappa++ Script

```
function i = diode( vj, isat, v)
  i = isat*(exp(v/vj))-1;

function i_fx = diode_fx( vj, isat, v)
  i_fx = fi(isat*exp(v/vj))-1, 1, 64, 56);

function diode_monte_carlo()
  vj=unifrnd(2.58e-2 ± 1%);
  isat=unifrnd(1e-3 ± 1%);
  v=linspace(1e-6,1,10);
  inputs = allprod(vj, isat, v);
  i = arrayfun (@diode, inputs);
  i_fx = arrayfun (@diode_fx, inputs);
```

Listing 3: Matlab Sketch of Monte-Carlo

TABLE II: Code Listings for Diode Current

across the product set of input parameter combinations. The generated output distribution is then compared against the statically-predicted bounds from Gappa++. This empirical validation improves programmer confidence in our compiler transforms.

IV. Experimental Setup

In this section, we describe our experimental flow in terms of our software infrastructure and the hardware library framework used. We use and adapt the FX-SCORE streaming compiler and its three backends to perform experiments performed in this paper. In Figure 2, we show the corresponding three components of the FX-SCORE compiler.

A. Precision Analysis

We now accept uncertainty information on the parametric inputs in addition to ranges on the variable inputs such as *voltage*. The programmer can supply uncertainty value in these parameters as a single percentage for all input parameters during the compilation process using a compiler

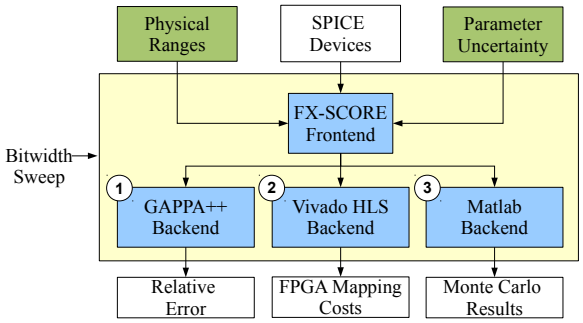


Fig. 2: FX-SCORE Compile Flow (with Uncertainty)

	IEEE-64			54-bit			24-bit		
	LUT	Reg	DSP	LUT	Reg	DSP	LUT	Reg	DSP
Basic Math Operators									
Add	1019	1074	3	54	54	15	24	24	6
Mult	305	927	11	120	653	9	9	30	2
Elementary Functions									
Exp	1559	2475	12	1960	2910	9	595	664	1
Log	4792	4095	12	3235	4004	9	948	1218	3
Divide	3301	6119	0	481	1828	18	193	811	10
Format Conversion Wrappers									
FL2Flco	103	0	0	102	0	0	52	0	0
Flco2FL	111	0	0	104	0	0	49	0	0
FL2Fix	-	-	-	135	374	0	70	174	0
Fix2FL	-	-	-	139	424	0	37	168	0

TABLE III: Hardware Operators Cost Model (V7LX160T)

switch `(-egappa_uncertain <value>)`. As part of a future release, we will allow per-parameter specification of uncertainty in the FX-SCORE input language. We upgraded the Gappa++ backend to use newer versions of Gappa v0.16. These required modification to the disjunction expressions for handling **`if`**/**`then`**/**`else`** statements in the FX-SCORE input.

B. Monte-Carlo Simulation Backend

We wrote a new Matlab backend to experimentally confirm our compilation output. Our Matlab backend uses the Statistics toolbox (**`unifrnd`**`()`) for generating the pseudo-random input parameter combinations and the Fixed-Point toolbox (**`fi`**`()`) to evaluated fixed-point implementations of our equations. We use the Matlab (**`arrayfun`**`()`) model for vector evaluation of our device equations across the input combinations. We were able to run the double-precision **`arrayfun`**`()` evaluations in parallel $10\times$ faster using an NVIDIA C2075 platform when compared to Intel Core i7-3770 3.4GHz CPU using a minor code-generation modification in the FX-SCORE Matlab backend. The ease of transferring our Monte-Carlo simulations to the GPU made us pause and reflect on the vision of using FPGAs for a similar purpose one day in the not too distant future.

C. Hardware Operators

We use the latest fixed-point and floating-point hardware cores based on the new release of Flopoco [2] v2.4.0 as well as the latest release of Vivado HLS v2012.4 to ensure up-to-date FPGA mapping cost data. For certain operators like the *exp* and *log* we have no fixed-point equivalents. For these operators, we use the Flopoco-generated floating-point cores with an exponent width of 8. We observed that

the fixed-point division cores inferred by Vivado HLS were pessimistic with their bitwidth analysis and hence we replaced them with Coregen-supplied fixed-point dividers. We tabulate the resource utilization numbers in Table III. We generated the resource utilization numbers post place-and-route and compute power using XPower assuming default 12.5% FF activity and 50% BRAM activity. These operators are able to compile correctly with a 4ns performance target with some double-precision and small bitwidth fixed-point mappings satisfying a more aggressive 3ns target. We can observe that the fixed-point mappings can often be as much as 20× smaller that the equivalent double-precision implementations. They can sometimes require more DSP blocks than the double-precision reference at precisions above 64-bits (not shown in Table). In addition to LUTs and DSP blocks, the exp and log operators also require BRAM memory blocks to store lookup tables. The range bitwidths possible for our FPGA mapping experiments is some subset from 16-bits to 64-bits due to core generation constraints in Flopoco as well as Xilinx Coregen. We generate and use format conversion wrappers to integrate the dividers and Flopoco cores with the rest of the datapath.

D. Production Use

For our experimental flow, we perform an exhaustive Gappa++ analysis sweep as well as a Vivado HLS implementation sweep across different bitwidths. In a production environment this will be quite expensive in terms of total CAD time. In fact, we only need the end user to invoke our compilation flow with an exhaustive sweep across different bitwidth when calling the Gappa++ backend. That will provide the minimum precision required to match the reference double-precision design in a matter of seconds. The slower Vivado HLS backend can be called exactly once with this bitwidth.

V. Evaluation

We now describe the results of our experimental evaluation of our modifications and enhancements to the FX-SCORE framework. We will first illustrate the application of our flow on a diode and then discuss broader results on our benchmark set of SPICE devices.

A. Impact of Parametric Uncertainty

Earlier in Figure 1, we visually built an intuition that input parameter uncertainty can cause small changes in the outputs. We saw the extent of output error depending on the amount of input parameter uncertainty in the computation. Higher the uncertainty, larger the error. Can we formalize this intuition? In Figure 3(a), we show the impact of different uncertainty factors on relative error in diode current (y-axis) for different values of diode voltage (x-axis). As we can see, the amount of relative error in the output exceeds the relative error introduced by the double-precision implementation when considering uncertainty. This suggests that the double-precision implementation is generating results that are too accurate with respect to the relative error that will be introduced purely from input parameter uncertainty. *Caveat*: You may notice that the uncertainty case for $u = 10^{-10}$ below $V_d = 10^{-3}$V seems to introduce less error than the Gappa++ double-precision bound. Of course, the Gappa++ bound is the worst case error over the entire interval $[10^{-5} - 0.1]$ rather than the instantaneous error values at a given voltage and thus appears to provide the illusion of having larger error below $V_d = 10^{-3}$. In practice, all our calculations are based on worst case error.

B. Relative Error Analysis using Gappa++

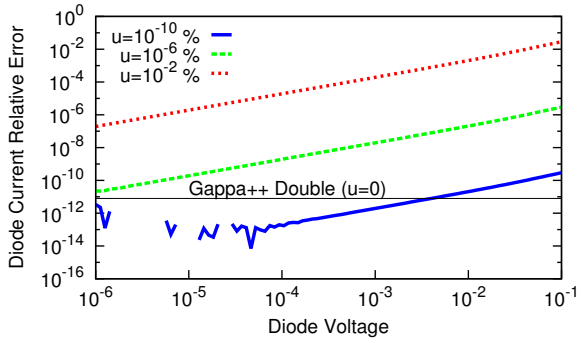
In Figure 3(b), we show how we compute the sufficient fixed-point precision required to implement our circuit. As we can see, there is an inherent relative error in the baseline double-precision implementation. As we increase fixed-point precision, the relative error value reduces until it matches (or does slightly better than) the reference double-precision error threshold (horizontal line at $\approx 10^{-11}$). The smallest precision when this happens (64 bit) is our bitwidth for the fixed-point mapping. Now, in the presence of uncertainty, we observe that this crossover happens at a lower precision as the reference double-precision error level rises with uncertainty. Thus, for the diode, the crossover occurs at 40 bits for a $u = 10^{-10}$.

C. Understanding Crossover Bitwidth

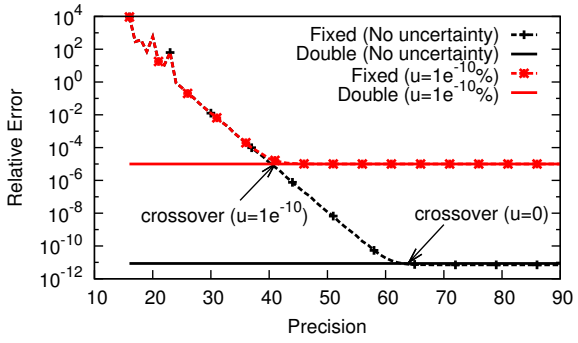
How does the crossover bitwidth change with parametric uncertainty? To answer this question, we sweep the uncertainty percentage from 10^1 to 10^{-16} in Figure 3(c). We observe a wide range of relative errors trends in our fixed-point implementations that are bounded by the double-precision reference design. For a large portion of the uncertainty range, we note that the fixed-point relative error can do as well as the double-precision threshold provided we use a large enough bitwidth. Thus, for sensible uncertainty ranges, it is possible to tradeoff FPGA implementation area to target a certain expectation of relative output error. Large uncertainties of 10% or more will introduce a large amount of error that will be unacceptable in circuit simulations at any precision.

D. Monte-Carlo Validation of our FX-SCORE's Decisions

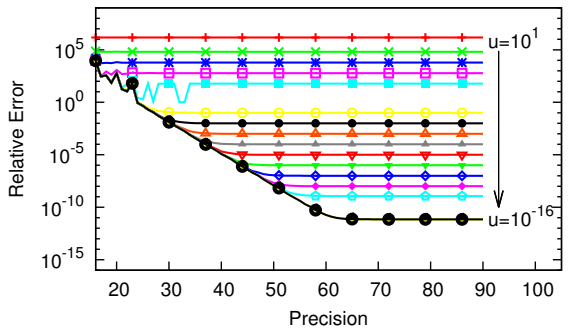
Now you might wonder if our statically analyzed bounds are too pessimistic (or large). In Figure 3(d), we show how the amount of relative error in the diode current (y-axis) as a function of uncertainty (x-axis) for different precisions computed using Monte-Carlo simulations. The reference double- precision bound is calculated using Gappa++ by considering the impact on uncertainty (This is $f_{double(u)}$ from Equation 5). As we had hoped, all Monte-Carlo simulations generate an error spread that is bounded by the static analysis (The fixed-point ranges are $f_{fixed(u,bits)}$ from Equation 6). The Monte-Carlo simulations are based on input parameter ranges and assume uniform random distribution. A 16-bit implementation of the circuit can handle high uncertainties in the input parameters above 10^{-11} in a manner that is competitive with a double-precision implementation. If we raise the precision to 32 bits, we can tolerate uncertainties above 10^{-15}. Only at 64 bits can our fixed-point implementation be completely bounded by the double-precision relative error across all uncertainties in



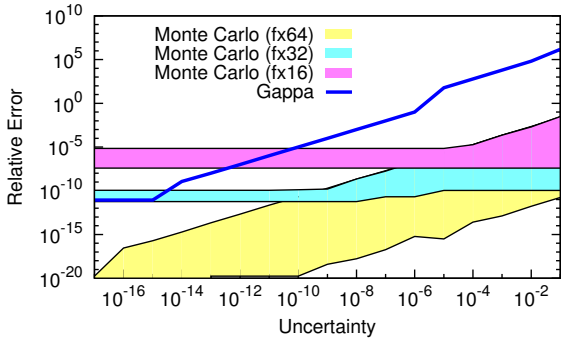
(a) Relative Error I-V Characteristics of Diode



(b) Relative Error Crossover



(c) Relative Error for Different Uncertainty Percentages



(d) Monte-Carlo Simulation

Fig. 3: Applying FX-SCORE Uncertainty Extensions to a Diode

the range $[10^{-16} : 10^{-1}]$. This data shows that the crossover precision for matching double-precision mappings will change with the amount of uncertainty in the input parameters. The Gappa++ static analysis bounds are going to be somewhat pessimistic compared to Monte-Carlo data as they have to be conservative in their assumptions to guarantee robust results across a range of calculations.

E. Impact of Device Model on Crossover Bitwidth

Now that we understand the application of our framework to the diode, how does it handle different devices? In Figure 4, we show the crossover fixed-point precision across different uncertainty percentages for different device models. As we can see, we can reduce datapath precision significantly (2-3 $\times$) as we increase uncertainty (0.1%) in our input parameters. The crossover precisions stay relatively flat at small values of uncertainty (roughly below 10^{-10}). This is because these small uncertainties have limited influence on output intervals. For devices such as $approx2$, we note that the crossover precision threshold stays particularly flat and only drops above an unusually high uncertainty of 10^{-4}. We attribute this to the fact that the arithmetic expression in this example includes a log that compresses the output intervals thereby limiting the influence of input parameter uncertainty.

F. Resource Utilization of Vivado HLS Implementations

We now discuss the results of compiling fixed-point circuits using Vivado HLS toolflow. In Figure 5, we plot LUT count,

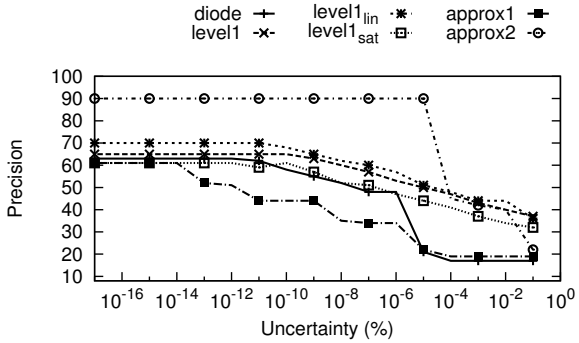


Fig. 4: Crossover Precision for Different Devices

DSP count and FF usage statistics for a diode. We sweep bitwidths from 24 bits to 64 bits and generate FPGA mappings for every bit in the interval. We observe a large dynamic range in resource utilization between 1-5$\times$ suggesting a broad opportunity for area savings in presence of uncertainty-driven precision reduction. In contrast with vanilla FX-SCORE [7], the design space exploration approach in this paper is not arbitrarily quantized and provides a better understanding of cost scaling trends.

G. Impact of Uncertainty on FPGA Mapping Costs

As observed in Figure 4, we are able to reduce the precision of fixed-point implementation in presence of input parameter uncertainty. How does that affect FPGA resource utilization? In Figure 6, we show the FPGA LUT count growth as a

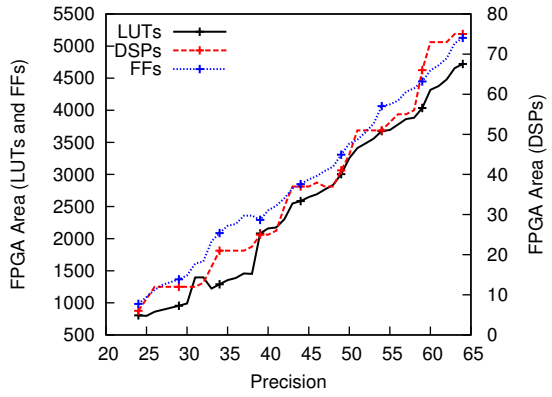


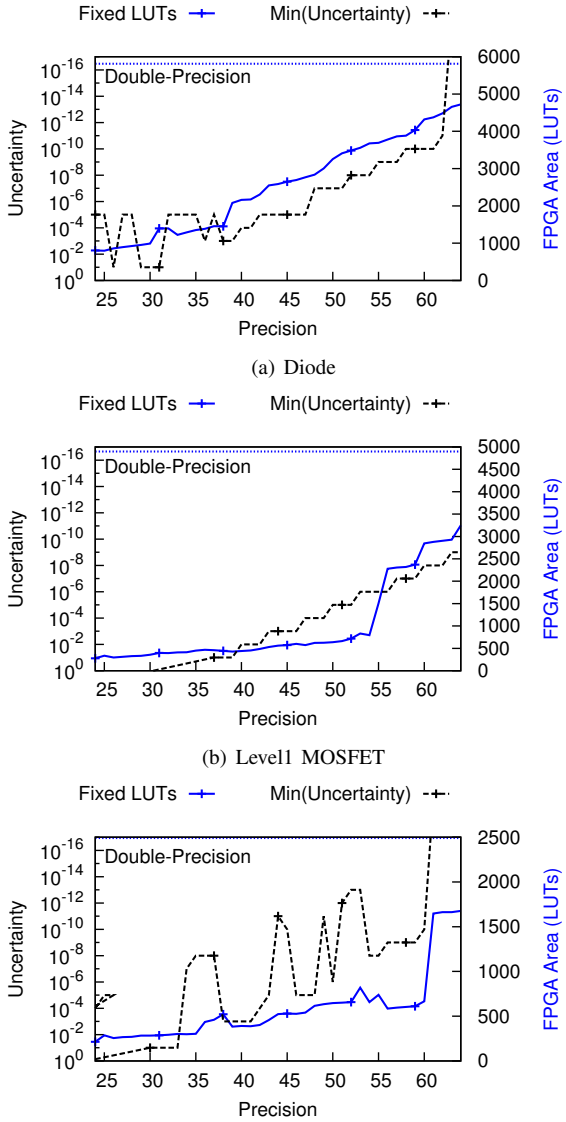
Fig. 5: Implementation Scaling Costs for Diode

function of precision as well as the equivalent minimum tolerable uncertainty in our inputs. The minimum tolerance is the smallest value of uncertainty that can be supported at a given bitwidth. You may recollect that we need increasingly larger datapath precision as we reduce uncertainty margins on the input parameters. When the input parameter uncertainty gets smaller than 10^{-16}, it is smaller than what can be represented in double-precision arithmetic and thus corresponds to the the costliest fixed-point FPGA implementation.

Ultimately, we are interested in understanding the impact of uncertainty on FPGA resource utilization and power usage when compared to the reference double-precision implementation. In Figure 7, we show the savings in LUT and DSP usage as well as the reduction in power utilization for our implemented circuits. As expected, in Figure 7(a), we observe area savings increase dramatically as we increase input parameter uncertainty. Higher uncertainty values permit lower precision implementations that are able to deliver as much as 16× reduction in LUT counts. This is the basis of our "order of magnitude" resource improvement claim in Section I. It is important to note that any significant savings only become apparent for uncertainties above 10^{-10} and escalates quickly above that. This can be explained from Figure 4 where the crossover precision tends to stay high for the very small uncertainties and only starts decreasing when there is sufficient uncertainty in the inputs. In Figure 7(b), we see a spread from 0.4–2.5× reduction in DSP counts. Approximately below $u = 10^{-10}$, we seem to require more DSPs than the double-precision mapping. We attribute this partly to larger internal precision generated by Vivado HLS and the lack of suitable DSP-LUT-balancing high-level synthesis algorithms in generation of the fixed-point cores. However, at a large enough uncertainty, we do achieve non-trivial DSP count reductions. We also used XPower to generate the dynamic power numbers in Figure 7(c). We see substantial savings due to reduced LUT count usage. As discussed earlier, the missing datapoints in Figure 7 are due to core bitwidth limitations.

H. Cost of Ignoring Uncertainty for Fixed-Point Mappings

In Table IV, we quantify the impact of ignoring uncertainty information in the FX-SCORE compiler [7] for a represen-



(a) Diode

(b) Level1 MOSFET

(c) Approximate Transistor

Fig. 6: Impact of Input Uncertainty on Crossover Precision and FPGA LUTs

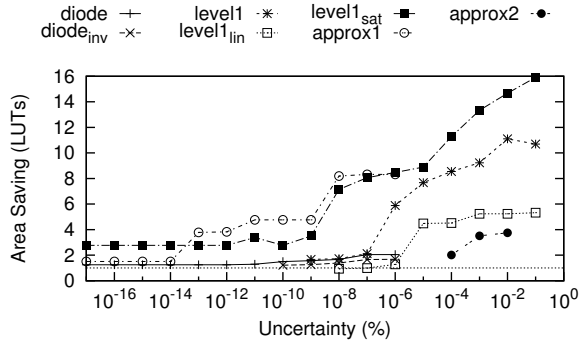
tative uncertainty of 10^{-6}. In contrast with Figure 7, where we compare against a double-precision reference baseline, we now compare against the oblivious fixed-point compilation that assumes $u = 0$. We see 1–5.5× LUT count savings, 1–4.4× DSP count reductions and 1–5.5× power usage improvements.

VI. Discussion and Future Work

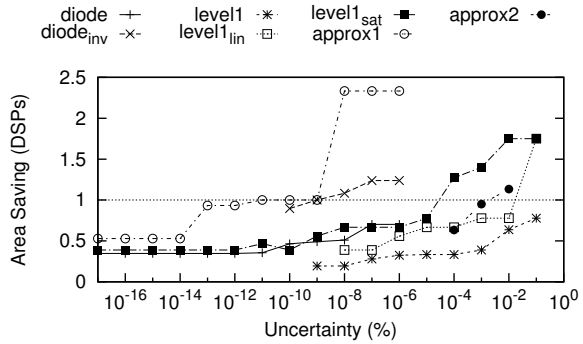
Fixed-point implementations of computation have been shown to deliver modest, single-digit resource reductions compared to oblivious double-precision mappings in the past. While this has been criticized by some as being too little, we believe precision tuning can continue to deliver increasing density benefits by exploring multiple aspects of system composition. While this paper focuses on *input parameter uncertainty*, we believe custom precision selection can deliver even greater improvements through a combination of additional optimizations such as *hybrid fixed/float precision*

Device	Crossover Bitwidth			FPGA LUTs				FPGA DSPs				FPGA Power			
	Obliv.	Uncert.	*Diff.*	Dbl.	Obliv.	Uncert.	*Ratio*	Dbl.	Obliv.	Uncert.	*Ratio*	Dbl.	Obliv.	Uncert.	*Ratio*
Diode	63	48	16	5811	4657	2836	1.6×	26	75	37	2×	105	36	17	2.1×
Level1	64	53	9	4898	3249	832	3.9×	14	80	43	1.9×	41	61	44	1.3×
Level1$_{lin.}$	64	57	13	1246	1324	973	1.3×	14	36	25	1.4×	26	-	-	-
Level1$_{sat.}$	64	47	23	4534	1856	535	3.4×	14	40	21	1.9×	97	-	-	-
Approx1	61	34	25	2490	1647	299	5.5×	28	53	12	4.4×	33	55	10	5.5×
Approx2	90	90	0	21813	21965	21965	1×	76	235	235	1×	338	276	276	1×

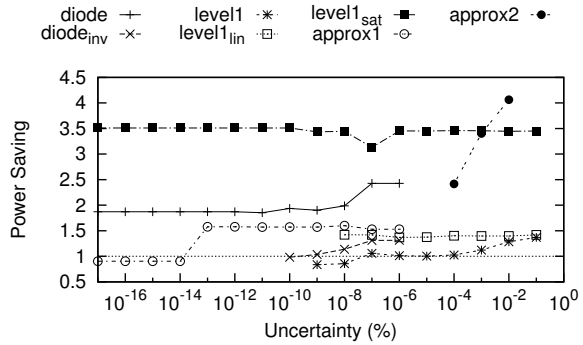
TABLE IV: FX-SCORE Before and After Extensions for Uncertainty (u=10^{-6})



(a) LUT Savings



(b) DSP Savings



(c) Power Savings

Fig. 7: Comparing FPGA Implementation Costs

selection, *runtime precision adaptation* or *number system customization*. It is important to build and refine toolflows like FX-SCORE with newer ideas and compiler transformations that can automate this bitwidth selection process.

VII. Conclusions

We show how to get greater than an order of magnitude reduction in LUTs if we exploit input parameter uncertainty in the context of SPICE device equations. We modify FX-SCORE to support input parameter uncertainty and observe up to 16× reduction in LUT counts, 2.5× reduction in DSP blocks and 4× reduction in dynamic power utilization of SPICE devices when compared to oblivious double-precision mappings. We also observed resource utilization and power usage improvements up to ≈5x compared to the uncertainty-oblivious fixed-point mappings. The larger precision reductions are possible for input uncertainties larger than 10^{-10} and they continue to increase with higher uncertainties. In contrast, uncertainties smaller than 10^{-10} deliver more modest benefits as they are too small to have measurable impact on output error ranges. We further observed that the extent of these improvements are tied to the type of arithmetic operations and specifically elementary functions like log limit the opportunity for aggressive precision reduction. Simpler datapaths using add and $multiply$ operations on bounded input ranges are more amenable to substantial resource savings.

VIII. Acknowledgments

I would like to acknowledge the contributions of Scott Hauck (for raising a related question at FCCM 2012), Michael Linderman (for introducing Gappa++), David Boland (for periodic consultations) and Chua Ngee Tat (for IT infrastructure support) in the delivery of this paper. Source code is openly available at http://www.github.com/nachiket/tdfc.

References

[1] S. Boldo, J.-C. Filliâtre, and G. Melquiond. Combining Coq and Gappa for Certifying Floating-Point Programs. In *Intelligent Computer Mathematics*. 2009.

[2] F. de Dinechin, C. Klein, and B. Pasca. Generating high-performance custom floating-point pipelines. In *2009 International Conference on Field Programmable Logic and Applications (FPL)*, 2009.

[3] A. DeHon, Y. Markovsky, E. Caspi, M. Chu, R. Huang, S. Perissakis, L. Pozzi, J. Yeh, and J. Wawrzynek. Stream Computations Organized for Reconfigurable Execution. *Journal of Microprocessors and Microsystems*, 2006.

[4] C. Fager, L. J. P. Linner, and J. C. Pedro. Optimal parameter extraction and uncertainty estimation in intrinsic FET small-signal models. *Microwave Theory and Techniques, IEEE Transactions on*, 2002.

[5] B. Gojman, S. Nalmela, N. Mehta, N. Howarth, and A. DeHon. GROK-LAB: Generating Real On-chip Knowledge for Intra-cluster Delays Using Timing Extraction. In *FPGA '13: Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*, 2013.

[6] M. Linderman, M. Ho, and D. Dill. Towards program optimization through automated analysis of numerical precision. *Proceedings of the 8th annual IEEE/ACM international symposium on Code generation and optimization*, 2010.

[7] H. Martorell and N. Kapre. FX-SCORE: A Framework for Fixed-Point Compilation of SPICE Device Models using Gappa++. In *FCCM '12: Proceedings of the 2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, 2012.

[8] M. Stephenson, J. Babb, and S. Amarasinghe. Bidwidth analysis with application to silicon compilation. In *PLDI '00: Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation*, 2000.

[9] R. Strzodka and D. Goddeke. Pipelined Mixed Precision Algorithms on FPGAs for Fast and Accurate PDE Solvers from Low Precision Components. In *FCCM '06: Proceedings of the 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2006.

[10] Q. Y. Tang and C. J. Spanos. Interval-Value Based Circuit Simulation for Statistical Circuit Design. In *Proc. SPIE 7275, Design for Manufacturability through Design-Process Integration*, 2009.