FX-SCORE: A Framework for Fixed-Point Compilation of SPICE Device Models using Gappa++

Hélène Martorell
Institut National Polytechnique de Toulouse
Toulouse, France
helene.martorell@gmail.com

Nachiket Kapre
Imperial College London
London, United Kingdom
nachiket@imperial.ac.uk

***Abstract*—Automated, offline precision-analysis of dataflow computation containing elementary functions (*e.g.* exp) and if-then-else control flow operations enables accurate fixed-point FPGA implementation of SPICE device equations. We perform interval analysis of these equations using Gappa++ to statically compare error bounds of fixed-point and double-precision implementations. This is possible due to the limited dynamic range of physical voltage, current and conductance quantities in a SPICE simulation of real-world circuits. In contrast to previous custom-precision SPICE device mappings, our fixed-point implementation has the same accuracy as double-precision implementation when compared to ideal arithmetic (reals). To deliver these implementations we develop FX-SCORE, a high-level framework based on the SCORE streaming FPGA framework, that automatically generates Gappa++ scripts and AutoESL circuits to explore the cost-quality tradeoffs of FiXed-point FPGA implementations. Using our methodology, we can determine whether fixed-point is always better than a double-precision implementation at the same relative error. We demonstrate 35% geometric mean area improvement for different SPICE device models such as Diode, Level-1 MOSFET and an Approximate MOSFET when comparing custom fixed-point implementations with standard double-precision realizations.**

I. Introduction

FPGA-based application accelerators can outperform conventional architectures *e.g.* multi-core CPUs, GPUs by orders of magnitude for parallel applications that are amenable to spatial implementation. A combination of factors explain these speedups including pipelined, application-specific datapaths, concurrent access to distributed memories, and localized movement of data over wires. FPGA performance is enhanced if we can unroll the dataflow computation as a fully spatial circuit. However, for many complex, irregular and large problems *e.g.* high-performance computing (HPC) problems, it is not always possible to fully exploit spatial parallelism due to FPGA capacity limits. In these cases, we can explore precision customization of the datapath circuits to deliver a smaller spatial implementation. We can perform this tradeoff for numerical computations where we can define and compare the accuracy of datapath evaluation. Usually, this is achieved through paper-pencil analysis, extensive simulations or a compiler optimization for simple, signal-processing applications.

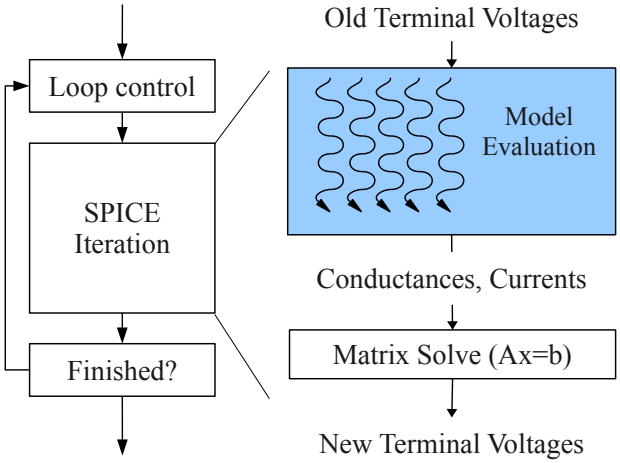


Fig. 1: Flowchart of a SPICE Simulator

We use the Model-Evaluation phase of the SPICE circuit simulator as a case study to demonstrate the potential for automated precision analysis. SPICE (Simulation Program with Integrated Circuit Emphasis) [15] is a numerical program that is floating-point intensive and challenging to implement on parallel hardware. It models static and dynamic analog behavior of electronic circuits by iteratively solving non-linear, differential circuit equations. It consists of two phases per iteration as shown in Figure 1: **Model Evaluation** followed by **Matrix Solve** ($A\vec{x} = \vec{b}$). The Model Evaluation phase is a data-parallel computation requiring concurrent evaluation of different device equations to construct A and $\vec{b}$. A double-precision FPGA implementation of Model-Evaluation can be expensive to implement on the FPGA. Hence, we investigate the potential for fixed-point compilation of such device models to the FPGA fabric.

Recent advances in static analysis tools such as Gappa++ [1], [14] offer an opportunity to automate the exploration of precision for challenging applications such as SPICE. In contrast with other precision analysis tools currently available, Gappa++ is able to analyze dataflow graphs containing elementary functions *e.g.* `exp`. However, we must still write verbose, low-level Gappa++ scripts that

encode the arithmetic properties we wish to analyze. This can be tedious, error-prone and would have to be managed by the already over-burdened hardware designer. In this paper, we develop a high-level toolflow called FX-SCORE, based on the SCORE [7] framework, to automatically generate appropriate fixed-point implementations of double-precision datapaths using Gappa++, AutoESL [3], Flopoco [4] library and the Xilinx Coregen library. While we demonstrate this toolflow on SPICE device models, the framework is general and can be reused in other applications.

In this paper, we make the following contributions:

- Development of a high-level methodology for compiling SCORE programs to low-level Gappa++ scripts for static computation of relative error in SPICE devices.
- Compiler support for handling if-then-else control flow operations in Gappa++ scripts.
- Design of a hardware backend that compiles SCORE programs to AutoESL C circuits using Flopoco and Coregen libraries for elementary functions (*e.g.* `exp`).
- Demonstration of area-error tradeoffs of using this toolflow for devices such as Diode, Level-1 MOSFET and Approximate MOSFET (can be extended to other device models as well).

II. Background

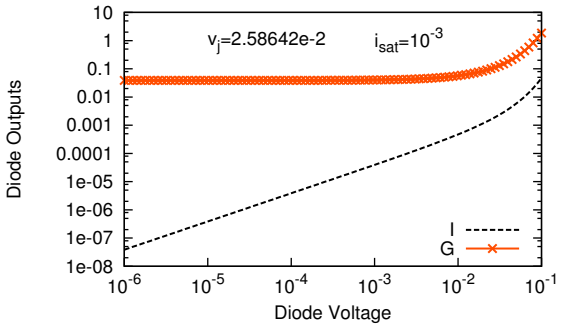
A. SPICE Simulator

SPICE simulates the dynamic analog behavior of a circuit described by its constituent non-linear differential equations. The circuit equations model the linear (*e.g.* resistors, capacitors, inductors) and non-linear (*e.g.* diodes, transistors) behavior of devices and the conservation constraints (*i.e.* Kirchoff's current laws—KCL) at the different nodes and branches of the circuit. SPICE solves the non-linear circuit equations by alternately computing small-signal linear operating-point approximations for the non-linear elements and solving the resulting system of linear equations until it reaches a quiescent operating point. In the Model-Evaluation phase, the simulator computes conductances and currents through different elements of the circuit and updates corresponding entries in the matrix with those values.

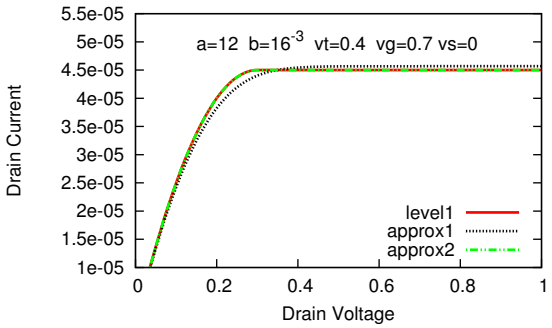
B. Non-Linear Models

In this paper, we focus on three devices to illustrate the impact of precision analysis. Each of these examples allows us to illustrate unique cases for the use of FX-SCORE in customizing precision for hardware implementation. We show range-limited I-V characteristics of these devices in Figure 2.

- **Diode** A diode is a simple two-terminal non-linear device modeled in SPICE. We represent the current and conductance expressions for a simplified diode as shown in Table I. The equation for diode current and conductance contains the `exp` elementary function which is expensive to implement on an FPGA (See Table IV. *Can precision analysis of equations with elementary functions help reduce implementation cost?*



(a) Diode



(b) MOSFET

Fig. 2: I-V Characteristics of different devices

- **Level-1 Schichman-Hodges MOSFET Model** The Level-1 MOSFET model is the simplest transistor model available in SPICE. We show expression for drain current and gate-drain conductance in Table I. This model contains conditional control flow operations in the form of if-then-else statements. We suggest an approach for handling these statements later in Section III-C. The operating regions of the transistor, in each branch, are valid over different input ranges which can offer an opportunity for precision customization. *Can we customize the precision of each branch of the IF-statement to improve FPGA area requirements compared to a homogeneous-precision implementation?*
- **Approximate MOSFET Model** For many circuit simulations, it is important to obtain proper subthreshold current characteristics that are not captured in the Level-1 model. We may choose a simple approximate model from IBM that delivers this behavior with few operations. This model contains two `log` and `exp` operations for each current, as shown in Table I, that are valid over a small input range. As we scale to finer geometries we need to start enhancing SPICE device equations with models of extra physical effects. *What is the cost of modeling subthreshold effects in this manner when implemented in fixed-point?*

C. SCORE Framework

SCORE [2], [7] is a high-level system architecture that is well-suited for high-performance FPGA implementation of SPICE device models as well as other parallel, streaming computations. A SCORE program consists of a graph of operators (compute) and segments (memory) linked to each

Device	Equations
Diode	$I = i_{sat} \times (\exp^{V/v_j} - 1)$ $G = i_{sat} \times \exp(V/v_j)/v_j$
Level-1 MOSFET (level1)	**if** $(Vgs < vt)$ $I_d = 0$ **elseif** $(Vds > Vgs - vt)$ $I_d = b \times (Vgs - vt)^2/2$ **else** $I_d = b \times (Vgs - vt - (Vds/2)) \times Vds$
Approximate MOSFET (approx1)	$term_1 = \log(1 + \exp^{a(Vgs-vt)})/a$ $term_2 = \log(1 + \exp^{a(Vgd-vt)})/a$ $I_d = b \times (term_1^2 - term_2^2)/2$
Approximate MOSFET (approx2)	$term_1 = (Vgs - vt > 0)?(Vgs - vt) : 0$ $term_2 = (Vgd - vt > 0)?(Vgd - vt) : 0$ $I_d = b \times (term_1^2 - term_2^2)/2$

TABLE I: SPICE Device Equations

other via streams (interconnect). Streams provide point-to-point communication and are realized as single-source, single-sink FIFO queues of unbounded length. For SPICE devices, non-linear device is a streaming operator with `voltage` inputs and `conductance` and `current` outputs.

D. Gappa++

Gappa [1] and Gappa++ [14] are proof assistants that enables static analysis of arithmetic properties of numerical programs. Gappa performs interval analysis of numerical programs and can bound intervals of numerical variables by providing intervals on inputs and/or outputs. It can also compute absolute and relative error in different implementations. We pick Gappa++ instead of other tools (*e.g.* System Generator) because it supports elementary functions that are present in SPICE device models. Gappa++ [14] has been used to compare CPU, FPGA and GPU implementations of some example programs using low-level Gappa++ scripting and manual implementation on each target. Our approach automates the manual creation of verbose Gappa++ scripts and also generates FPGA implementations from the same high-level description of computation.

E. Previous Work

In Table II, we list previous attempts at implementing SPICE phases in parallel hardware. AWSIM [13] and TINA [17] implement device evaluation using TBMA approximation [12] (table-lookup with interpolation) with a permissible error tolerance (1% relative error for currents down to $10^{-11}A$ compared to nominal double-precision implementation on Sun-3/60). Our approach avoids any accuracy tradeoffs and delivers the same quality as double-precision implementation of the device models. If required, we can easily adapt our approach to deliver even cheaper implementations at lower accuracy. In [18], a fixed-point, table-lookup implementation (without interpolation) is used to implement a circuit-specific simulation accelerator without reporting accuracy tradeoffs. In [6], a transformed and discretized macromodel of device equations is implemented on the FPGA in fixed-point arithmetic with no quality comparisons with traditional SPICE simulation. SILCA [20] performs a combination of piecewise weakly non-linear (PWNL) approximation of device evaluation and chordal conductance approximation, among other

Ref.	Name	Key Idea	Hardware	Accuracy	Tradeoff
[13]	AWSIM-3	Compiled-Code VLIW	Custom prototype	1% rel. error (10^{-11}A)	Table-lookup
[17]	TINA	Based on AWSIM-3	23 Xilinx XC4005 chips	1% rel. error (10^{-11}A)	Table-lookup
[18]	-	Partial-evaluation	Altera Flex10K	not reported	Fixed-point
[6]	SPO	Analytical Transform	Xilinx Spartan 3	not reported	Fixed-point
[20]	SILCA	Approximation	-	SPICE-like (not precise)	PWNL, etc
[9]	Nascentric	Data-parallelism	NVIDIA 8800 GTS	4.8×10^{-5} rel. error	Single-Precision
[10]	SPICE^2	Custom-VLIW	Xilinx V6 lX760	Same as spice	Double-Precision

TABLE II: Parallel SPICE Approaches with Precision/Accuracy Tradeoffs

transformations, to deliver "SPICE-like" accuracy of the overall simulation. We have a more precise definition of accuracy and deliver the same accuracy as that of a double-precision Model-Evaluation implementation. Nascentric [9] implements device evaluation on single-precision NVIDIA GPU with a large relative error of 4.8×10^{-5}, compared to double-precision CPU, across few benchmark simulations. We compute relative error with reference to ideal arithmetic using static analysis. In [10], a resource-shared FPGA implementation of the SPICE device models delivers 6.5× speedup across different device models when implemented in double-precision arithmetic. We hope to lower the implementation precision to avoid resource sharing on the FPGA fabric.

III. Precision-Analysis of SPICE Device Models

In this section, we describe how we use FX-SCORE and Gappa++ to perform precision analysis of SPICE devices.

A. Accuracy

Both fixed-point and double-precision implementations of computation can have error due to *rounding*. Rounding errors are introduced due to insufficient precision to represent a number (*e.g.* $\frac{1}{3} = 0.333\ldots$). We can bound these errors through exhaustive simulations of our numerical programs. This is intractable for SPICE device models (*e.g.* multiple 64-bit inputs). Instead, we use Gappa++ to statically analyze error bounds on the outputs of the device models. Gappa++ can allow us to examine two kinds of errors. Absolute error, shown in Equation 1, (1) is the difference between the given and reference values while Relative error, shown in Equation 2, (2) is Absolute error scaled with the reference value. We compare relative error of alternative implementations instead of absolute error since we have small, but non-trivial dynamic ranges for our physical variables. We perform a sweep of different precisions and pick one that provides the same or lower relative error than the double-precision implementation (See Equation 3).

$$ABSERR_{precision}(f) = (f_{precision} - f_{ideal}) \qquad (1)$$

$$RELERR_{precision}(f) = (f_{precision} - f_{ideal})/f_{ideal} \quad (2)$$

$$RELERR_{fixed-point}(f) \leq RELERR_{ieee\_double}(f) \quad (3)$$

B. High-Level FX-SCORE Framework

We extend the open-source SCORE framework [11] with two new code-generation backends, one for Gappa++ and another for AutoESL. We also add language support for specifying input stream intervals in SCORE syntax. The intervals are propagated to Gappa++ for computation of relative error. SCORE already supports custom bitwidth types which are provided to AutoESL for hardware generation. As Gappa++ does not naturally support if-then-else statements, we devise a simple strategy described later in Section III-C.

We can better understand the mechanics of the FX-SCORE framework through an example. We show how the framework compiles a SCORE diode description into Gappa++ scripts and AutoESL C code.

SCORE Code: In Listing 1, we observe that the SCORE description of the diode consists of two main parts: (1) SCORE operator definition which lists all inputs (`voltage`) and outputs (only showing `current` for simplicity) with optional interval information (Line 2–3 in Listing 1), and (2) SCORE operator body which captures the dataflow description of the computation into individual states (Line 4–5 in Listing 1). For SPICE device models, we can also specify instance-specific device constants (*e.g.* a, b, c, v_j, vt and i_{sat} parameters from Table I) using SCORE `param` type (not shown in listing). These values are encoded as constants for Gappa++ and AutoESL. When generating code, we must be careful to note that while Gappa++ fixed-point representation only requires the LSB weight (*i.e.* fraction bits), AutoESL requires specification of both LSB and MSB weights (*i.e.* integer bits and fraction bits)

Gappa++ Script: In Listing 2, the Gappa++ script contains three main parts: (1) expression of computation under real ($\_m$), double ($\_dbl$) and fixed-point ($\_fx$) versions (Line 3–5 in Listing 2), (2) specification of intervals for the variables (Line 7–8 in Listing 2), and (3) questions about the relative error between double-ideal and fixed-ideal implementations (Line 9–10 in Listing 2). In (2), we must also specify a bound on the smallest output quantity required which in the example is machine ϵ for double-precision numbers. It is clear that writing these low-level Gappa++ scripts can be easily avoidable with automation.

AutoESL C: In Listing 3, the AutoESL C code contains three parts: (1) `ap_fixed` data-type specification of fixed-point format (Line 1), (2) function with inputs and outputs (passed as a pointer like in CUDA [16]) that match the streaming SCORE operator (Line 2), and (3) function body with dataflow expression for output (Line 4). We also use AutoESL for floating-point mapping by replacing `ap_fixed` type with `double`.

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

Listing 1: SCORE (User input)

```
@fx = fixed<-32,ne>;
@dbl = float<ieee_64,ne>;
i_m = isat_m*(exp(v_m/vj_m)-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] /\
  |i_m| >= 0x1p-53 ->
  (i_dbl-i_m)/i_m in ? /\
  (i_fx-i_m)/i_m in ?
}
```

Listing 2: Gappa++ (Auto-Generated)

```
typedef ap_fixed<32,8> data_t;
void diode(data_t v, data_t *i)
{
  *i = isat*(exp(v/vj)-1);
}
```

Listing 3: AutoESL (Auto-Generated)

TABLE III: Code Listings for Diode Current

```
y_1 = (cond_1)*option_1 + (1-cond_1)*option_2
y_2 = (cond_2)*option_1 + (1-cond_2)*option_2
{
  (cond_1 in [0,0] \/ cond_1 in [1,1]) /\
  (cond_2 in [0,1])
  ->
  y_1 in ? /\
  y_2 in ?
}
```

Listing 4: Solution for IF-statement in Gappa++

C. IF-statement handling

Gappa++ does not naturally support expression of if-then-else type statements. For SPICE Level-1 equations as shown in Table I, we must specify drain current in three operating regions separately and combine them based on drain-source voltage. To support this, we construct a formulation shown in Listing 4. We then rewrite the if-then-else construct as a binary select operator. We then assign integer 0,1 constraints to the condition variable to capture its boolean behavior. We experimented with mutually exclusive range constraints ($[0,0]$ or $[1,1]$) and inclusive range constraints ($[0,1]$). We observed that we consistently got tighter results with the mutually exclusive range constraints at the expense of additional exploration time by Gappa++. We realize that for complex dataflow computations with multiple IFs, Gappa++ will have to exhaustively explore 2^N range permutations for the N condition variables. For the examples in this paper, we report the tighter error measurements.

IV. Experimental Setup

In Figure 3, we show the compilation flow for using the FX-SCORE framework for mapping SPICE Device Equations to fixed-point circuits.

The SCORE compiler accepts range-annotated input descriptions of device equation computations. It performs simple optimizations such as constant-folding and strength-reduction to simplify certain operations (*e.g.* divide by constant v_j gets converted into multiplication with its inverse). We supply input ranges to the SPICE physical voltage quantities from realistic SPICE circuit simulations of CMOS digital circuits. For the diode, we constrain the input voltage to 10^{-6}V$\rightarrow$ 0.1V. This ensures the output currents stay within reasonable physical limits (*e.g.* V=1 results in an I=$6.8 \cdot 10^{13}$. This also captures the behavior of the SPICE lim_exp (limited exponential) function that avoids runaway outputs of resulting current and conductance values. For the transistor circuits, we separately constrain drain voltage to 1.9V$\rightarrow$ 2.1V (pullup transistor topology) and source voltage to 10^{-6}V$\rightarrow$ 0.1V (pulldown transistor topology). Gate voltages can go between these two extremes. We constrain drain and source voltages to tight intervals based on expected usage scenario in a digital CMOS circuit. For both pullup and pulldown cases, we observe similar error behavior. We do range sensitivity analysis later in Section V-A. We also limit supplied input precision to double-precision for all implementations. This is because the Model-Evaluation phase of SPICE operates in tandem with Matrix-Solve phase which is implemented in double-precision. This constraint may be lifted as part of future work.

We explore different fixed-point bitwidths to generate relative error information and choose the minimum implementation bitwidth accordingly. Our framework currently supports homogeneous fixed-point representation. However, for certain examples, we modify the generated Gappa++ and AutoESL programs to support hybrid precision implementations. While SCORE can generate spatial fixed-point Verilog circuits, we use AutoESL because of its floating-point operator support. For the hardware implementation, we tabulate the resource costs of double-precision operators in Table IV. We use Flopoco to support custom `exp` and `log` operators in fixed-point implementations. The number format used in these functions is custom floating-point with exponent fixed to 8 bits. While the simulation ranges can be accommodated with even fewer exponent bits, we are constrained by Flopoco's to a minimum of 8 exponent bits. We also supply appropriate number format conversion wrappers around these cores with the help of Coregen. These cores are limited to a maximum of 64 fraction bits. We also replace the fixed-point divider core generated by AutoESL with Coregen-supplied fixed-point divider due to AutoESL's pessimistic integer bitwidth calculation. These in turn are limited to 54 bits of fraction bits. The bitwidth limits of the hardware cores constrain our experimental data to a limited range of possibilities. However, we are still able to deliver same relative error implementations with hybrid precision implementations.

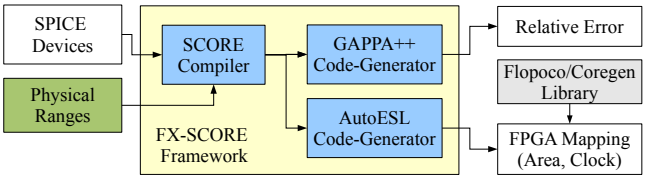


Fig. 3: FX-SCORE Flow

Operator	Area			Latency	Speed	Ref.
	Slices	DSP	RAM	clocks	MHz	
Multiply	238	10	0	10	308	[19]
Add	360	3	0	8	370	[19]
Divide	1896	0	0	57	190	[19]
Exponential	841	0	5	36	172	[5]
Logarithm	934	0	12	40	109	[8]

TABLE IV: Double-Precision Cost Model (V6LX760)

V. Evaluation

In this section, we show the error and area results of static analysis of SPICE device equations using FX-SCORE.

A. Impact of Range on Error

The exact value of relative error in the computation will depend on the tightness of the input range. In Figure 4, we show the impact of changing the upper limit of the input voltage range on the relative error in the current I of the diode ($10^{-3} \rightarrow 10^{1}$]) where the lower limit is set to 10^{-6}. We observe that relative error at a given precision increases as we increase the upper limit of the input range. This is expected as a wider range will require more bits to represent correctly. At 48-bit precision, the fixed-point implementation error is much larger than the nominal double-precision error for all range combinations. However, as we increase this to 72-bit, we are able to meet or do better than double-precision across all range combinations. When we limit the upper range to under 10^{-1}V, we observe that the double-precision error is now larger than the 72-bit fixed-point error. This confirms that we need to reasonably bound the input range to achieve narrower bitwidths.

B. Impact of Precision on Error

We now investigate the impact of changing precision of the datapath on the relative error of the different device models as shown in Figure 5. We are interested in using this precision sweep to identify the smallest fixed-point precision (crossover

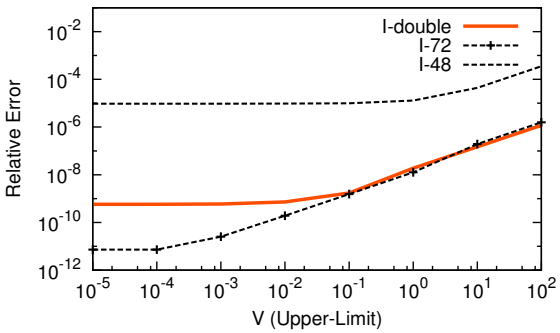


Fig. 4: Impact of Input Range on Diode I Equation

precision) that matches or does better than the relative error of double-precision implementation.

For the diode, shown in Figure 5(a), we observe that the current (I) and conductance (G) error scaling trends are very similar and decrease monotonically as we increase fixed-point bitwidth. The crossover precision for both quantities is at 72-bits. We do not observe any further improvement in error at higher fixed-point precisions as we consider voltage inputs to be available in double-precision for all implementations.

For the Level-1 MOSFET trends in Figure 5(b), we notice similar improvement in error as we increase fixed-point bitwidth. We analyze precision requirements of the linear and saturation region separately. We observe that the fixed-double crossover is at 60 bits for the saturation region and at 70 bits for the linear region. This difference can be explained by counting operations in each branch, the saturation region only requires one multiplication and a subtraction while the linear region requires two subtractions and a multiplication. This hybrid precision assignment may potentially offer an opportunity to save FPGA area.

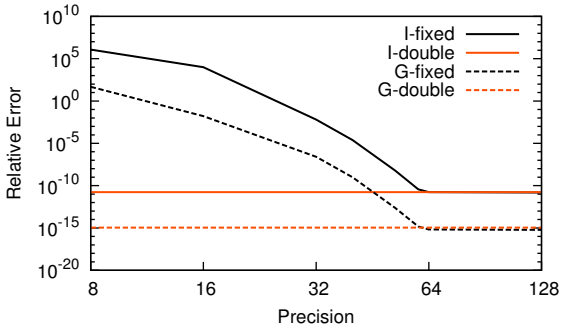
Finally, we consider the error scaling trends for the two Approximate MOSFET implementations in Figure 5(c). We observe that the approximation with subthreshold behavior exhibits larger relative error and has a crossover bitwidth of 96 bits. As expected, the less-accurate, simpler approximation without subthreshold behavior has smaller relative error and a crossover bitwidth of 72 bits. We attribute this difference due to the simplicity of operations in the second approximation, two multiplications and three subtractions. In contrast, subthreshold modeling requires the presence of elementary functions like `exp` and `log`.

C. Impact of Precision on FPGA Implementation

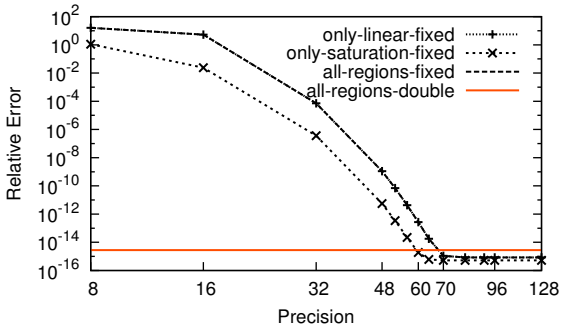
Next, we evaluate the impact of precision on the FPGA resource requirement of different SPICE device models together with relative error. We are primarily interested in identifying FPGA resource costs at the crossover bitwidth. We show the area-error trends for the different devices in Figure 6 and tabulate FPGA implementations details in Table V.

For the diode implementation, shown in Figure 6(a), we observe that area scaling data is only available up to 64 bits due to core generation limitations. We build a hybrid precision 64-72 bit implementation to deliver better relative error than the reference double-precision implementation. In this case, only the portion of the dataflow expression following the `exp` function is implemented in 72-bit hardware. Unfortunately, our hybrid fixed-point implementation requires 10% more area than the reference double-precision mapping. This is because total area is dominated by `exp` operator and the associated format conversion wrappers.

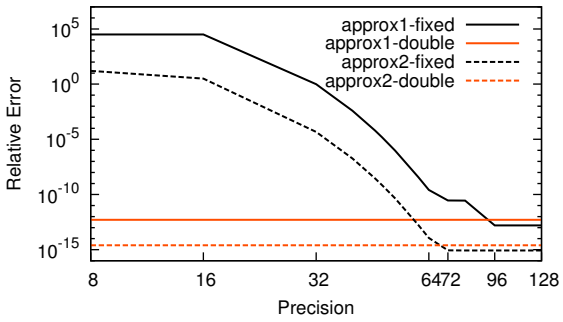
The Level-1 MOSFET without elementary functions shows more promise. In Figure 6(b), we observe that the fixed-point implementation at the crossover bitwidth of 72 bits is 70% smaller than the reference double-precision implementation. A hybrid precision implementation is possible where we customize the precision in each branch at 60 bits (saturation)



(a) Diode



(b) Level-1 MOSFET

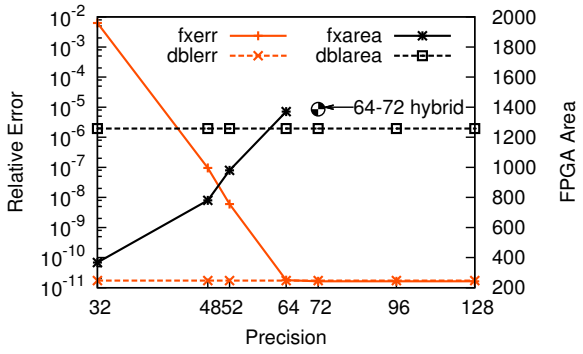


(c) Approximate MOSFET

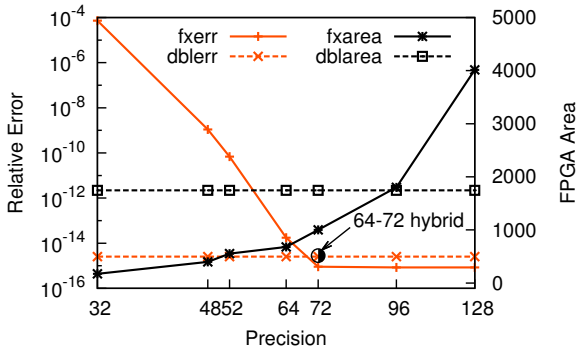
Fig. 5: Gappa++ Error for different devices

and 70 bits (linear). This implementation is able to surpass the reference double-precision implementation by $3.3\times$. Since we have a common subexpression $V_{gs} - V_t$ between the two branches, AutoESL is able to resource-share and reduce the area of the hardware implementations.

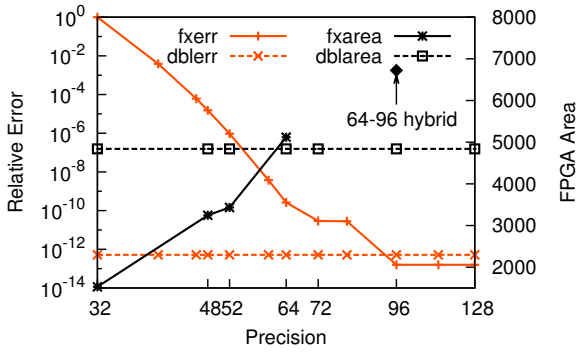
For the Approximate MOSFET implementation with subthreshold effects, shown in Figure 6(c), we observe that we are again limited by the expensive hardware implementations of the `exp` and `log` operators. Due to core generation limits, we can only generate hardware up to 64 bits. We develop a hybrid-precision implementation that keeps the `exp` and `log` operators at 64 bits while increasing the precision of the downstream operations to the crossover bitwidth of 96. We used Gappa++ to certify that this hybrid implementation had better relative error than the reference double-precision mapping. Our hybrid implementation is 40% larger than the



(a) Diode



(b) Level-1 MOSFET



(c) Approximate MOSFET

Fig. 6: Gappa++ Area-Error Trends for different devices

double-precision mapping. We later show in Table V, that the fixed-point implementation of the simpler approximate MOSFET delivers an area benefit of 60% at the smaller crossover bitwidth of 72.

In Table V, we note that the AutoESL clock period for the fixed-point mappings is relatively high. The timing report suggests a insufficient pipelining of DSP48 chains for high bitwidths multipliers. We also report that the Gappa++ runtimes for all our device models is less than a one second.

VI. Discussion

In Figure 7, we show the Area-Error tradeoffs for the different device models. The FX-SCORE framework can allow the hardware developer to explore the entire range of Area-Error tradeoffs when designing and composing the hardware system. We deliver fixed-point implementations of all device models with relative error that is better than reference double-precision implementations. For the Level-1 MOSFET and Approximate MOSFET ($approx2$), we were able to deliver area improvements when compared to the double-precision mapping due to simpler operations and hybrid-precision tuning of different IF-statement branches. This was not possible in the case of Diode and Approximate MOSFET ($approx1$) due to the overheads of expensive implementations of elementary functions `exp` and `log`. Of course, we can lower evaluation accuracy to achieve better area.

In Figure 8, we show the normalized throughput possible on a Xilinx V6LX760 by tiling multiple instances of the different device implementations to fit 118560 slices. Normalized throughput is the number of device evaluations per unit time across multiple tiles ($\frac{118560}{SLICE} \times \frac{1}{T_{clk}}$). We show significantly improved throughput compared to a resource-shared VLIW implementation from [10]. Under ideal pipelining of fixed-point multiplier DSP chains, our fixed-point devices are able to outperform double-precision Level-1 MOSFET and Approximate MOSFET ($approx2$) implementations by $3.4\times$ and $3.8\times$ respectively. For the Diode and Approximate MOSFET ($approx1$) our implementation has 10% less throughput in both cases.

Can precision analysis of equations with elementary functions help reduce implementation cost? For our examples, we have to revert to double-precision mapping of devices with `exp` and `log` operators. The large dynamic ranges around these operators (especially subthreshold currents down to $[10^{-12} : 10^{-6}]$A) makes it difficult for purely fixed-point mappings to surpass double-precision implementations.

Can we customize the precision of each branch of the IF-statement to improve FPGA area requirements compared to a homogeneous-precision implementation? For the Level-1 MOSFET, we were able to tune the precision of each branch to further improve area reduction from $1.7\times$ to $3.3\times$ when compared to a double-precision mapping. This translates into a saving of $1.9\times$ compared to the homogeneous case.

What is the cost of modeling subthreshold effects when implemented in fixed-point? Modeling of subthreshold MOSFET effects results in variables with a large dynamic ranges. This coupled with the presence of elementary functions makes it expensive to model on FPGAs. For fixed-point processing we pay an overhead of $7\times$ while for double-precision we pay an overhead of $3\times$ (see Table V).

Larger device models with different distributions of arithmetic operators can potentially deliver better results than the small `diode` and `approx1` examples. We also expect better support for smaller exponent bitwidths in the `exp` and `log` operators to further reduce area required. For timing, better pipelining of inter-DSP interconnect by the tools or substitution with pipelined fixed-point multipliers can deliver better results. A quick experiment with Flopoco's [4] 70-bit integer multiplier suggests that a $T_{clk} = 2.5ns$ is easily achievable.

SPICE Device	Relative Error (I)	Double-Precision				Bitwidth		Fixed-Point				Area Saving	Gappa++ Runtime
		SLICE	DSP	RAM	T_{clk}	Ideal	Feasible	SLICE	DSP	RAM	T_{clk}		
diode	$1.6 \cdot 10^{-11}$	1258	22	10	3.5	72	64-72	1386	68	5	8.5	0.9	0.6s
level1	$9 \cdot 10^{-16}$	1745	17	0	3.8	72	72	1002	108	0	10.2	1.7	0.7s
level1_{hybrid}	$9 \cdot 10^{-16}$	-	-	-	-	-	60-70	515	49	0	8.4	3.3	0.7s
level1_{linear}	$8.9 \cdot 10^{-16}$	737	17	0	3.5	72	72	493	54	0	8.9	1.4	0.5s
$\text{level1}_{satur.}$	$6.1 \cdot 10^{-16}$	1473	17	0	3.5	64	64	403	33	0	8.8	3.6	0.5s
approx1	$2.9 \cdot 10^{-14}$	4840	78	70	4.4	96	64-96	6722	210	42	10.2	0.7	0.8s
approx2	$8.8 \cdot 10^{-16}$	1604	34	0	3.5	72	72	950	68	0	9	1.6	1s

TABLE V: Comparing different FPGA Implementations of SPICE devices (T_{clk} in ns)

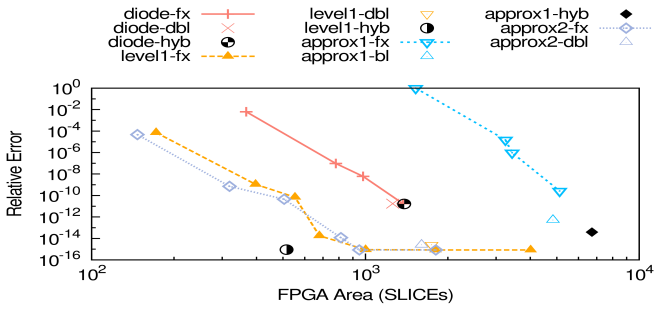


Fig. 7: Error-vs-Area for different devices

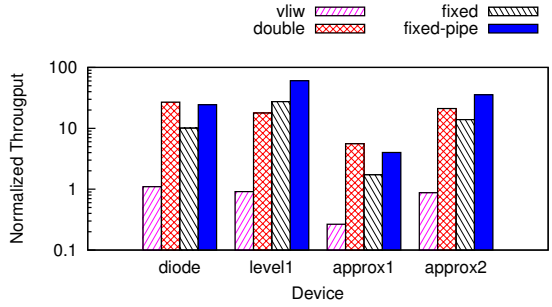


Fig. 8: FPGA Throughput for different devices

VII. Conclusions

The FX-SCORE framework delivers a $\approx 1.35\times$ geomean (0.7–3.3×) area reduction for custom fixed-point implementation of SPICE device models when compared to the reference double-precision implementations with same or better relative error. We are able to provide this analysis by compiling parallel SCORE descriptions of the devices into low-level Gappa++ scripts for evaluating relative error and AutoESL circuits for estimating FPGA implementation costs. For devices dominated by `exp` and `log` operations such as Diode and Approximate MOSFET (with subthreshold effects), our fixed-point mappings presently cannot offer efficient alternatives to double-precision implementations. However, for the Level-1 MOSFET and Approximate MOSFET (no subthreshold effects), our fixed-point mappings provided up to 3.3× area reductions. In the future, we expect this framework to enable us to deliver a complete application-level fixed-point analysis and area-error tradeoffs for complex applications like SPICE.

VIII. Appendix

The tools used in this study are open-source and available for download online [11]. We would like to thank Eylon Caspi, André DeHon, Michael Linderman and Kuen Hung Tsoi.

References

[1] S. Boldo, J. Filliâtre, and G. Melquiond. Combining Coq and Gappa for certifying floating-point programs. *Intelligent Computer Mathematics*, pages 59–74, 2009.

[2] E. Caspi. *Design Automation for Streaming Systems*. Phd, University of California, Berkeley, 2005.

[3] J. Cong, B. Liu, S. Neuendorffer, and J. Noguera. High-Level Synthesis for FPGAs: From Prototyping to Deployment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(4):473–491, 2011.

[4] F. de Dinechin. FloPoCo, 2011.

[5] F. De Dinechin and B. Pasca. Floating-point exponential functions for DSP-enabled FPGAs. In *International Conference on Field-Programmable Technology*, pages 110–117. IEEE, 2010.

[6] B. Deepaksubramanyan, P. Parakh, Zhenhua Chen, H. Diab, D. Marcy, and F. Schlereth. An FPGA-based MOS circuit simulator. In *48th Midwest Symposium on Circuits and Systems*, pages 655–658 Vol. 1, 2005.

[7] 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. *Microprocessors and Microsystems*, 30(6):334–354, Sept. 2006.

[8] J. Detrey, F. De Dinechin, and X. Pujol. Return of the hardware floating-point elementary function. In *IEEE Symposium on Computer Arithmetic*, pages 161–168. IEEE, 2007.

[9] K. Gulati, J. F. Croix, S. P. Khatri, and R. Shastry. Fast circuit simulation on graphics processing units. In *Asia and South Pacific Design Automation Conference*, pages 403–408, 2009.

[10] N. Kapre. *SPICE^2 - A Spatial Parallel Architecture for Accelerating the SPICE Circuit Simulator*. Phd, California Institute of Technology, 2010.

[11] N. Kapre, E. Caspi, and A. DeHon. SCORE. https://github.com/nachiket/tdfc, 2011.

[12] D. Lewis. Device model approximation using 2N trees. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 9(1):30–38, 1990.

[13] D. Lewis. A compiled-code hardware accelerator for circuit simulation. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 555 – 565, 1992.

[14] M. Linderman, M. Ho, D. Dill, T. Meng, and G. Nolan. Towards program optimization through automated analysis of numerical precision. In *IEEE/ACM international symposium on Code Generation and Optimization*, pages 230–237, New York, New York, USA, 2010. ACM.

[15] L. W. Nagel. *SPICE2: A Computer Program to Simulate Semiconductor Circuits*. PhD thesis, University of California Berkeley, 1975.

[16] NVIDIA. *Nvidia CUDA C Programming Guide*. NVIDIA, 2011.

[17] M. van Iersel. *Circuit Simulation on a Field Programmable Accelerator*. PhD thesis, University of Toronto, 1995.

[18] Q. Wang and D. M. Lewis. Automated Field-Programmable Compute Accelerator Design Using Partial Evaluation. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 145 – 154, Napa Valley, 1997.

[19] Xilinx. LogiCORE IP Floating-Point Operator v6.0. Technical report, Xilinx, 2011.

[20] Zhao Li and C.-J. Shi. SILCA: SPICE-accurate iterative linear-centric analysis for efficient time-domain Simulation of VLSI circuits with strong parasitic couplings. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(6):1087–1103, 2006.