MixFX-SCORE: Heterogeneous Fixed-Point Compilation of Dataflow Computations

Deheng Ye
Nanyang Technological University
Singapore
ye0014ng@e.ntu.edu.sg

Nachiket Kapre
Nanyang Technological University
Singapore
nachiket@ieee.org

Abstract—
Mixed-precision implementation of computation can deliver area, throughput and power improvements for dataflow computations over homogeneous fixed-precision circuits without any loss in accuracy. When designing circuits for reconfigurable hardware, we can exercise independent control over bitwidth selection of each variable in the computation. However, selecting the best precision for each variable is an NP-hard problem. While traditional solutions use automated heuristics like simulated annealing or integer linear programming, they still rely on the manual formulation of resource models, which can be tedious, and potentially inaccurate due to the unpredictable interactions between different stages of the FPGA CAD flow. We develop MixFX-SCORE, an automated tool-flow based on FX-SCORE fixed-point compilation framework and simulated annealing, to address this challenge. We outsource error analysis (Gappa++) and resource model generation (Vivado HLS, Logic Synthesis, Xilinx Place-and-Route) to external tools that offer a more accurate representation of error behavior (backed by proofs) and resource usage (based on actual utilization). We demonstrate 1.1–3.5x LUTs count savings, 1–1.8x DSP count reductions, and 1–3.9x dynamic power improvements while still satisfying the accuracy constraints when compared to homogeneous fixed-point implementations.

I. Introduction

Reconfigurable devices such as FPGAs allow the designer to have complete freedom in customizing the implementation of computation in hardware. This includes the ability to select the minimum number of bits necessary to represent variables as demanded by the application. When compared to ISA-based processors with a fixed set of data types (*e.g.* short, int, long, float, double), FPGAs allow bit-level tuning of circuit precision for a given application. This can deliver combined area, throughput and power improvements over traditional processor implementations. However, we can only reduce bitwidth as long as certain user-supplied error constraints are satisfied. The objective of the bitwidth minimization problem, which we explore in this paper through MixFX-SCORE, is to find the optimal combination of bitwidth for all variables along the datapath subject to user-supplied error bound constraints in a reasonable amount of time.

Bitwidth allocation is an NP-hard problem [2]. For a small 7-variable `diode` device model $i = i_{sat} \cdot (e^{v/v_j} - 1)$ (see line 5–11 in Listing 2), we will have to consider $(128 - 16)^7$ combinations of bitwidth allocations in the range 16–128 over a non-convex optimization space. Existing approaches for solving this problem rely on a combination of techniques including empirical observations and measurements, paper-pencil analysis, and automated heuristic search (interval and affine analysis with simulated annealing, integer linear programming and some other custom heuristics). These methods require building analytical error and area models to estimate precision properties and hardware costs respectively. A key research question we investigate in this paper is the fidelity of the resource models used in this search. As we climb the design abstraction stack to high-level synthesis (HLS) flows, we must prepare to handle unpredictable interactions between HLS compiler, the logic synthesis and place-and-route steps in the FPGA CAD flow. This makes it tricky to construct analytical resource models that fully capture unpredictable impacts of optimization in each CAD step. Inaccuracies in the model result in incorrect conclusions by the search heuristic. If we rely purely on analytical models to drive the search, we observed consistently inferior results. Our results indicate superior solution quality when using feedback from the logic synthesis stage.

FX-SCORE [8], [5] shows how to find equivalent homogeneous fixed-point implementations of double-precision floating-point circuits that include control flow as well as elementary functions. Analysis tools such as Gappa [1] and Gappa++ [7] are a critical component of this framework. They enable automated mathematical analysis of relative error models supported by proofs of correctness. FX-SCORE also relies on Vivado HLS for generation of synthesizable hardware for calculating resource utilization, power and performance of the resulting hardware. However, FX-SCORE framework only automates homogeneous fixed-point computations, *e.g.* a single bitwidth for the entire computation. In this paper, we describe an improved MixFX-SCORE framework built upon FX-SCORE that uses adaptive simulated annealing (ASA) [4], [3] for selecting global optimized mixed precision bitwidths and is based on multiple resource models (analytic, post-HLS, post-logic-synthesis).

We make the following contributions:

- Integration of FX-SCORE framework and a custom adaptive simulated annealing (ASA) cost function supported by Gappa++ for estimating relative error of mixed precision implementations.
- Development of a dataflow engine for FX-SCORE that

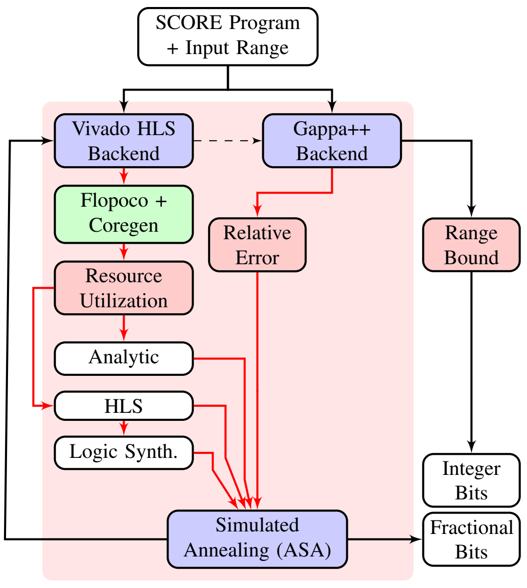

Fig. 1: MixFX-SCORE Flow

supports the Vivado HLS and Gappa++ code-generation backends.

- Bitwidth optimization support for elementary functions and control flows.
- Quantification of resource, delay and dynamic power cost of mixed-precision computations when compared to floating-point and iso-quality homogeneous fixed-point computations across a variety of benchmarks.

II. MixFX-SCORE Framework

A. Error Formulation

Double-precision implementations of computation have inherent errors due to rounding and truncation of intermediate variables in the computation. We can measure this error in the form of absolute or relative error metrics as required by the application. FX-SCORE, and other bitwidth tuning techniques, use relative error metric as reference for lowering and customizing precision. In contrast with the original optimization formulation in FX-SCORE, we pick a heterogeneous mixture of precisions such that the relative error for the resulting combination is lower than the reference error in double-precision mapping as shown in Equation 4. The minimization goal now requires a suitably accurate area model.

$$err_{double}(f) = (f_{double} - f_{ideal})/f_{ideal} \tag{1}$$
$$err_{homogeneous}(f) = (f_{homogeneous} - f_{ideal})/f_{ideal} \tag{2}$$
$$err_{heterogeneous}(f) = (f_{heterogeneous} - f_{ideal})/f_{ideal} \tag{3}$$

$$\begin{aligned} \text{minimize} \quad & area_{heterogeneous} \\ \text{subject to} \quad & err_{heterogeneous} \leq err_{double} \end{aligned} \tag{4}$$

B. Tool Flow

The FX-SCORE framework integrates multiple external tools (such as Gappa++, Flopoco, Xilinx Coregen), and a compiler backend (Vivado HLS). We further extend FX-SCORE by supporting an Adaptive Simulated Annealing (ASA) backend that generates custom cost function code which internally invokes three different area models. We adapt the heterogeneous precision code generation using a DFG engine. In Figure 1, we show an overview of this MixFX-SCORE compilation framework.

At a high level, the MixFX-SCORE framework runs two types of analysis to determine precision (1) fraction-width analysis, and (2) integer-width analysis. The mathematical analytics is handled by Gappa++ through a combination of simple interval analysis (for integer width) and error model formulation (for fraction width). The framework accepts SCORE programs annotated with interval information on the input variables. We then translate the computation into Gappa++ scripts that encode these inputs to generate intervals for intermediate variables as well as relative error statistics for the outputs of the computation. We also generate Vivado HLS code from the same intermediate representation with custom types for each variable.

Listing 1: SCORE Description of Diode Model

```
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 ); }
```

C. Fraction-width Analysis

We show a DFG code example automatically generated from our framework in Listing 2 using the `diode` device model SCORE implementation shown in Listing 1. FX-SCORE compiles the whole computation dataflow equation into one single Gappa and C++ expression (see sample code in [8]) with homogeneous precision. In contrast, MixFX-SCORE generates code that is split into atomic statements for every binary and unary arithmetic expressions (line 5–11 from Listing 2). This allows us to specify a unique precision for individual operation independently. Like some earlier studies [6], we use Adaptive Simulated Annealing[4], [3], to heuristically explore the optimization space quickly. We integrate ASA into our flow by auto-generating the `asa_usr_cost_fn` function customized for each input problem. We use the relative error result of the Gappa++ invocation as the constraint to enforce legality of the solutions based on Equation 4 We build the ASA cost function based on calling the Vivado flow and using models at different levels of the compilation process. For fast compilations, we use an analytical area calculation, while resorting to post-HLS and post-logic synthesis results for increasing the accuracy of the model at the cost of substantially larger runtime.

For basic mathematical operators, we build the area model based on the following rule, which is similar to [6], [9]. Area of adder is approximated as sum of input bitwidths, area of multiplier is approximated to product of input bitwidths, area of elementary functions is derived from its Taylor approximation while for division we build a database-driven model through a compile sweep.

We optimize the runtime of the ASA search by a combination of strategies. A certain bitwidth combination is only validated when it satisfies the relative error constraints established by the Gappa++ invocation. Combinations that fail the relative error test do not need to run the hardware backend flow. We restrict the range of legal bitwidths for certain operators based on the hardware-generation limits of Coregen and Flopoco We use the result of the homogeneous FX-SCORE framework result as our starting point to begin the optimization. We control the range of allowed bitwidths carefully to avoid fruitless search of the solution space. Due the non-monotonic nature of the optimization problem, we sometimes have to allow intermediate variables to have larger precision than the recorded homogeneous precision. We provide a 2-bit guard band above this crossover limit as the initial upper bound across all variables in the program. To compute the lower bound for the search, we start from such a guarded bitwidth and decrease the precision of each variable one-by-one until it fails to meet the relative error criteria. Now we finish bounding the lowest range. We replace the bitwidth of each variable with the lowest bit individually. This combination of bitwidth may not satisfy our relative error constraints. We then start increasing all the variables' bitwidths simultaneously until it succeeds to meet the error constraints. Once we do this, we collect a vector of bitwidths ranges that are tight enough as the respective selection limits for the individual variables during the ASA search. This heuristic gives an interval for each ASA input parameter is sufficient to cover the solution for the benchmark set we use. We also empirically choose the solution of the analytical area model as the starting annealing state for our post-HLS and post-logic synthesis invocations to reduce annealing iterations.

Listing 2: Auto Generated Gappa++ DFG Sample

```
@fx1 = fixed<−64,ne>; ... ... ; @fx7 = fixed<−64,ne>;
@dbl = float<ieee_64,ne>;
i_m =isat_m∗(exp(v_m/vj_m)−1);
i_dbl dbl = isat∗(exp(v/vj)−1);
vj = fx1 (0.0258);
isat = fx2 (0.001);
v_fx = fx3 (v);
divide_7 = fx4 (v_fx/vj);
exp_9 = fx5 (exp(divide_7));
minus_10 = fx6 (exp_9 −1);
i_fx = fx7 (isat∗minus_10);
{
    v in [1e−6, 0.1] /\
    |i_m| >= 0x1p−53 /\
    (i_dbl − i_m) / i_m in ? /\
    (i_fx − i_m) / i_m in ?
}
```

D. Integer Bitwidth Selection

FX-SCORE is customized to handle SPICE device models and assumed a uniform 8-bit integer portion for all variables based on worst-case interval analysis of voltage and current values. For MixFX-SCORE, we consider many more benchmarks, and even within a benchmark, we fully customize the precision of individual variables. As described earlier, we chose the fraction bits based on relative error models. For choosing the integer bits, we obtain the dynamic range of all variables, and then calculate the integer portion using Equation 5.

$$int\_bits = \lceil \log_2(\lfloor \max(|x_{max}|, |x_{min}|) + 1 \rfloor) \rceil + 1 \quad (5)$$

III. Methodology

A. Benchmarks

We evaluate our framework using a variety of benchmarks, including all the MiniBit [6] computations (rewritten in SCORE syntax) and all the SPICE device models used in FX-SCORE [8]. For the MiniBit benchmarks, we borrow ideas from [10] to properly allocate bitwidths for fractional input constants. For RGB_to_YCbCr, and DCT8x8, the input signals are unsigned integers which changes the relative error reference baseline. We compile these inputs and the derived intermediate variables into Vivado $uint$ data type. For other benchmarks, IEEE754 double-precision is the reference precision.The mixed fixed-point precision implementations are compiled to Vivado ap_fixed data type.

B. Tools

We upgrade the Gappa++ backend to support newer version Gappa (v1.0.0). We use Vivado Design Suite v2013.2 for hardware compilation. We compile three versions of design (double, homogeneous, heterogeneous) to the Kintex 7 FPGA device xc7k160. We use the latest versions of core generators (Flopoco v2.5.0, Coregen floating-point core v6.1, Coregen divider core v4.0). For operators such as exp and log with no direct fixed-point compilation support in Vivado HLS, we use a post-synthesis drop-in replacement. We compute FPGA dynamic power after PAR assuming default 12.5% toggle rate and 50% BRAM activity using XPower.

IV. Evaluation

In this section, we evaluate the experimental results of our MixFX-SCORE framework. We define *crossover* as the minimum homogeneous fractional bit-width required that leads to lower or equal relative error over double precision implementation obtained by invoking the original FX-SCORE compiler on the benchmark. We tabulate the crossover bits for all the benchmarks in Table I. We can see that crossover bit changes on a per-application basis and covers the range 52–73 bits.

A. MixFX-SCORE Results

We now compare the resource utilization, power consumption and circuit delay achieved by our MixFX-SCORE framework using post-PAR data (ASA driven using post-logic-synthesis resource models) with the existing best solutions possible using the FX-SCORE framework. We calculate heterogeneous implementation savings using the equation $(old - new)/old * 100\%$. Even though we use MiniBit [6] benchmarks, a direct comparison with MiniBit results is not possible due to different baseline (error in fixed-point homogeneous implementation) and technology (FPGAs

TABLE I: Comparing MixFX-SCORE Results with Fx-SCORE Implementations

Benchmarks	**Crossover**	**FPGA LUTs**				**FPGA DSPs**				**FPGA Dynamic Power(mW)**			
	Bit	Dbl.	Fxscore	Heter.	Savings	Dbl.	Fxscore	Heter.	Savings	Dbl.	Fxscore	Heter.	Savings
Polynomial	54	4121	1217	931	23%	14	48	27	43%	267	129	93	27%
RGB_to_YCbCr	52	3821	939	805	14%	75	35	35	0%	136	27	27	0%
2x2 Matrix Mult	73	4644	7466	5655	24%	100	200	136	32%	101	380	273	28%
Bspine	58	9122	4670	3041	34%	39	176	132	25%	610	414	299	27%
DCT8x8	52	10296	5362	4849	9%	100	150	150	0%	537	376	372	1.1%
Diode	63	5656	5105	4036	20%	26	85	67	21%	460	234	181	22.6%
Level1	60	6878	1435	929	35%	31	48	36	25%	388	123	95	22%
Level1$_{linear}$	62	1199	564	434	23%	14	28	18	35%	29	2	1	50%
Level1$_{saturation}$	61	4372	855	494	42%	14	28	18	35%	325	68	61	10%
Approx1	62	21813	21914	6293	71%	76	128	77	39%	1368	1261	326	74%
Approx2	61	2017	992	731	26%	25	44	41	6%	66	128	104	18%
Geomean Savings					25.5%				14.3%				14.8%

TABLE II: Comparing MixFX-SCORE Results with FX-SCORE Implementations

Bmarks.	**FPGA FFs**				**Clock (ns)**				**Figure-of-Merit**			
	Dbl.	Fxscore	Heter.	Savings	Dbl.	Fxscore	Heter.	Savings	Dbl.	Fxscore	Heter.	Ratio
Polynomial	6486	2191	1972	10%	2.916	2.712	2.893	-6.6%	3.43e6	7.62e5	3.96e5	1.9x
RGB_to_YCbCr	6757	2046	1879	8.2%	2.743	2.608	2.840	-7.6%	1.98e6	1.15e5	1.77e5	0.7x
2x2 Matrix Mult	9517	1298	1186	8.6%	3.052	2.512	2.331	8%	2.05e6	1.09e7	5.33e6	2x
Bspine	13916	8806	7483	15%	3.052	3.353	2.936	12.5%	1.84e7	1.14e7	4.99e6	2.3x
DCT8x8	17621	6476	5897	8.9%	3.052	14.755	14.533	1.5%	2.02e7	4.64e7	4.24e7	1.1x
Diode	10379	4593	3532	23.1%	7.252	4.725	5.336	-12.7%	2.09e7	7.7e6	5.33e6	1.5x
Level1	10256	2787	2263	18.8%	3.052	2.682	2.747	-2.2%	8.88e6	7.9e5	4.3e5	1.8x
Level1$_{linear}$	1799	1199	1048	12.5%	2.400	2.272	2.805	-21%	1.03e5	5.1e3	2.2e3	2.3x
Level1$_{saturation}$	6602	1513	1060	29.9%	3.052	2.734	2.831	-3.7%	4.61e6	2.63e5	1.47e5	1.8x
Approx1	26285	20953	8823	57.9%	7.506	13.670	6.086	53.8%	2.44e8	4.33e8	1.59e7	27x
Approx2	3454	1815	1444	20.4%	3.052	2.743	2.753	0%	5.07e5	6.57e5	4.42e5	1.6x
Geomean Savings				16%				0.84%				2.1x

without DSP blocks). In Table I, II, we observe LUT count savings from 9.6–71.3%, FF count improvements from 8.2%–57.9%, DSP count reductions of 0–43.8% and dynamic power savings of 10.3–74.1% when compared to the homogeneous implementations of FX-SCORE. Since we desire low power, small area, short delay design simultaneously, we formulate Figure-of-Merit (FoM) metric which is the product of $area$, $clock\ period$, and $dynamic\ power$ to evaluate the overall quality of MixFX-SCORE results. Here we compute $area$ as LUTs+20×(DSPs+BRAMs). We see our framework delivers a geometric mean 2.1x improvement when compared to FX-SCORE when using this FoM metric.

V. CONCLUSIONS

The MixFX-SCORE framework performs heterogeneous fixed-point compilation of streaming dataflow computations to deliver 1.1-3.5x area improvements, 1-1.8x DSP count savings and 1-3.9x FPGA dynamic power reductions across a range of benchmarks when compared to the homogeneous fixed-point implementations. We integrate a carefully-tuned Adaptive Simulated Annealing (ASA) bitwidth search with FX-SCORE. We support wordlength optimizations for complex non-linear computations and control flows.

REFERENCES

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

[2] G. Constantinides and G. Woeginger. The complexity of multiple wordlength assignment. *Applied Mathematics Letters*, 15(2):137 – 140, 2002.

[3] L. Ingber. Asa package. http://www.ingber.com/#ASA.

[4] L. Ingber. Very fast simulated re-annealing. *Mathematical and computer modelling*, 12(8):967–973, 1989.

[5] N. Kapre. Exploiting input parameter uncertainty for reducing datapath precision of spice device models. In *Int. Symp. on Field-Programmable Custom Computing Machines (FCCM)*, pages 189–197, 2013.

[6] D.-U. Lee, A. A. Gaffar, R. C. C. Cheung, O. Mencer, W. Luk, and G. A. Constantinides. Accuracy-guaranteed bit-width optimization. *IEEE Trans. on CAD of Integrated Circuits and Sys.*, 25(10):1990–2000, 2006.

[7] M. D. Linderman, M. Ho, D. L. Dill, T. H. Meng, and G. P. Nolan. Towards program optimization through automated analysis of numerical precision. In *Proc. IEEE Int. Symp. on Code Generation and Optimization*, pages 230–237, New York, NY, USA, 2010. ACM.

[8] H. Martorell and N. Kapre. Fx-score: A framework for fixed-point compilation of spice device models using gappa++. In *Int. Symp. on Field-Programmable Custom Computing Machines (FCCM)*, pages 77–84, 2012.

[9] W. G. Osborne, R. C. C. Cheung, J. Coutinho, W. Luk, and O. Mencer. Automatic accuracy-guaranteed bit-width optimization for fixed and floating-point systems. In *Int. Conf. on Field Programmable Logic and Applications (FPL)*, pages 617–620, 2007.

[10] S. Vakili, J. Langlois, and G. Bois. Enhanced precision analysis for accuracy-aware bit-width optimization using affine arithmetic. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 32(12):1853–1865, 2013.