<table>
<thead>
<tr>
<th><strong>Title</strong></th>
<th>MixFX-SCORE: Heterogeneous Fixed-Point Compilation of Dataflow Computations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Author(s)</strong></td>
<td>Ye, Deheng; Kapre, Nachiket</td>
</tr>
<tr>
<td><strong>Date</strong></td>
<td>2014-05</td>
</tr>
<tr>
<td><strong>URL</strong></td>
<td><a href="http://hdl.handle.net/10220/39194">http://hdl.handle.net/10220/39194</a></td>
</tr>
<tr>
<td><strong>Rights</strong></td>
<td>© 2014 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. The published version is available at: [<a href="http://dx.doi.org/10.1109/FCCM.2014.64">http://dx.doi.org/10.1109/FCCM.2014.64</a>].</td>
</tr>
</tbody>
</table>
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 \( i = i_{sat} \cdot \left( e^{v_{d}} - 1 \right) \) (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
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.

\[
\begin{align*}
    err_{\text{double}}(f) &= (f_{\text{double}} - f_{\text{ideal}})/f_{\text{ideal}} \quad (1) \\
    err_{\text{homogeneous}}(f) &= (f_{\text{homogeneous}} - f_{\text{ideal}})/f_{\text{ideal}} \quad (2) \\
    err_{\text{heterogeneous}}(f) &= (f_{\text{heterogeneous}} - f_{\text{ideal}})/f_{\text{ideal}} \quad (3)
\end{align*}
\]

**minimize** \( \text{area}_{\text{heterogeneous}} \)  
**subject to** \( err_{\text{heterogeneous}} \leq err_{\text{double}} \) \( (4) \)

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.

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.

### Listing 1: SCORE Description of Diode Model

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

### Listing 2: SCORE Program & Input Range

```
SCORE Program + Input Range
```

### Fig. 1: MixFX-SCORE Flow

![MixFX-SCORE Flow Diagram](image-url)
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 to 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 changes on a per-application basis and covers the range 52–73 bits.

Listing 2: Auto Generated Gappa++ DFG Sample

```plaintext
@fx1 = fixed<−64,ne>;' ... ; @fx7 = fixed<−64,ne>;
@dbl = float<ieee_64,ne>;
i_m = sat<si>(exp(v_m/vj_m)−1);
i_dbl_dbl = sat<(exp(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 = bx6 (exp_9−1);
i_fx = fx7 (isat−minus_10);
{
  v in [1e−6, 0.1] /
  |i_m| >= 0xλp−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.

\[
int_{\text{bits}} = \left\lceil \log_2(|\max(|x_{\text{max}}|, |x_{\text{min}}|) + 1) \right\rceil + 1
\] (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...
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