Filters Computing Just Right

http://flopoco.gforge.inria.fr/
Matel Țestoian, Florent de Dinechin
Socrate, CITI lab – INRIA

Precison vs. Accuracy in Hardware – The FloPoCo Philosophy

Computing Just Right

Why compute the result with more accuracy than the output can hold?
⇒ wasted accuracy that nobody can see
Why compute with less accuracy than your output can hold?
⇒ part of the result will be just noise

The FloPoCo Arithmetic Core Generator

Accurate results with minimal resources:
• generator and library for custom FPGA targets
• explore the flexibility of an FPGA

Example Operator – An Automatic Filter Generator

Output accuracy defined by the output format.

Implementing FIR Filters in Hardware

A classical solution

Sums of Products: From Theory to Implementation

Computing with precision ρ?

What we intended to compute: the mathematical result

What we actually computed:

The error:

Computing Just Right A Sum of Products by Constants

Determine the Sum’s Most Significant Bit (MSB)

The MSB of \(a_i x_i\)
• \(x_i\) bounded (fixed-point number)
• \(a_i\) known

\[
\text{msb}_{a_i x_i} = \left\lfloor \log_2(|a_i| \text{val}_{\text{max}}(x_i)) \right\rfloor
\]

The MSB of the sum
• \(a_i x_i\) bounded

\[
\text{msb}_S = \text{msb}_y = \left\lfloor \log_2 \left( \sum_{i=0}^{N-1} |a_i| \text{val}_{\text{max}}(x_i) \right) \right\rfloor
\]

Determine the Sum’s Least Significant Bit (LSB)

Suppose we use perfect multipliers: \(i_{\text{mult}} < 2^{-p - 1}\)
• after the sum:

\[
\sum_{i=0}^{N-1} a_i x_i \approx \sum_{i=0}^{N-1} \text{round}_{p_i} (a_i x_i)
\]

Need for larger intermediary precision
• \(p\) guard bits
• such that errors accumulate in the guard bits

\[
\Rightarrow \ p = \left\lfloor \log_2(N) \right\rfloor
\]

Multiplying by a Constant in a FPGA

Perfect Constant Multipliers

Basic FPGA block: look-up table (LUT) of \(n\) input bits
• tabulate all the values of \(a_i x_i\)
• ... correctly rounded to the output precision

\[
\lfloor \frac{i}{2^{k_i}} \rfloor
\]

Perfect fit for small sizes: \(x_i\) on \(i\) bits ⇒ 1 LUT/output bit

Doesn’t scale:
• \(2^n\) LUT/output bit for \(x_i\) on \(i + k\) bits

KCM Multipliers

When \(x_i > n\) bits: cut it into chunks \(d_{kj}\) of \(n\) bits

\[
x_i \approx \sum_{k=1}^{m} 2^{-k_i} d_{kj}
\]

where \(d_{kj} \in (0, ..., 2^n - 1)\)

Each \(d_{kj}\) tabulated, 1 LUT/output bit

How many output bits?

\[
a_{kj} = a_i x_k = 2^{-k_i} \sum_{k=1}^{m} 2^{-k_i} d_{kj}
\]

Sum of Products Using KCM Multipliers

Bit-heap based summation architecture

Example – Root-Raised Cosine Filter

<table>
<thead>
<tr>
<th>Naive method</th>
<th>ρ = 12</th>
<th>5.9 ns (170 MHz)</th>
<th>444 LUT</th>
<th>(\tau &gt; 2^{-9})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed method</td>
<td>ρ = 12</td>
<td>4.4 ns (227 MHz)</td>
<td>564 LUT</td>
<td>(\tau = 2^{-12})</td>
</tr>
<tr>
<td>Proposed method</td>
<td>ρ = 9</td>
<td>4.12 ns (243 MHz)</td>
<td>380 LUT</td>
<td>(\tau &lt; 2^{-9})</td>
</tr>
</tbody>
</table>