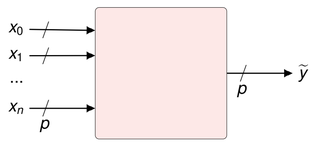
Filters Computing Just Right

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

Precision vs. Accuracy in Hardware – The FloPoCo Philosophy

Computing Just Right



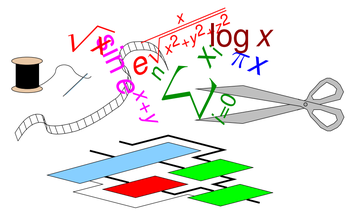
Why compute the result with **more accuracy** than the output can hold?

$\Longrightarrow$ wasted accuracy that nobody can see

Why compute with **less accuracy** than your output can hold?

$\Longrightarrow$ part of the result will be just noise

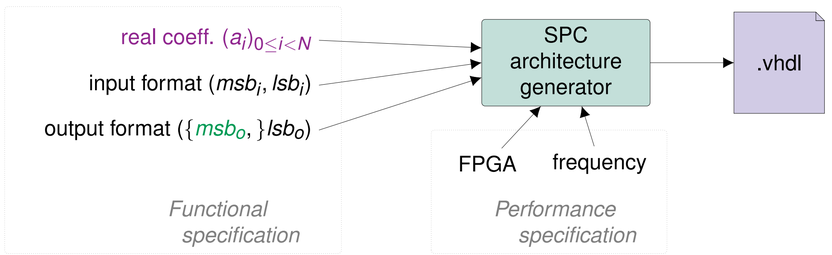
The FloPoCo Arithmetic Core Generator



Accurate results with **minimal resources**:

- **generator** and **library** for custom FPGA targets
- explore the **flexibility** of an FPGA

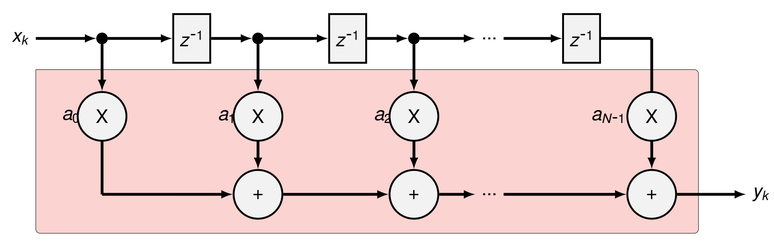
Example Operator – An Automatic Filter Generator



Output accuracy defined by the output format.

Implementing FIR Filters in Hardware

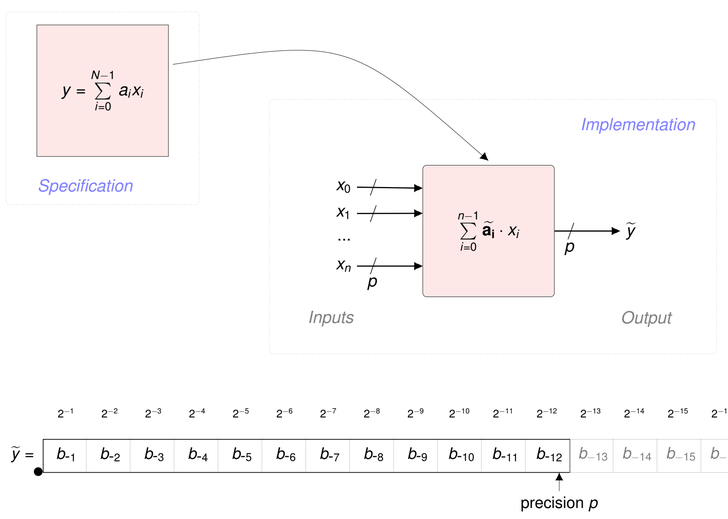
A classical solution



Each element y_k is a **sum of products**.

$$y_k = \sum_{i=0}^{N-1} a_i x_{k-i}$$

Sums of Products: From Theory to Implementation



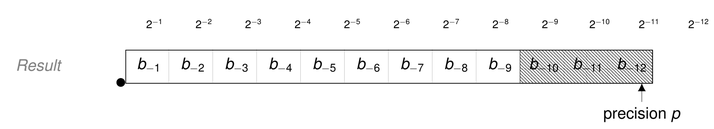
Computing with precision p?

What we **intended** to compute: the mathematical result

$$y = \sum_{i=0}^{N-1} a_i x_i$$

What we **actually** computed:

$$\tilde{y} = \sum_{i=0}^{N-1} \text{round}_p(\tilde{a}_i x_i)$$



The **error**:

$$\Longrightarrow \quad \varepsilon = \tilde{y} - y$$

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

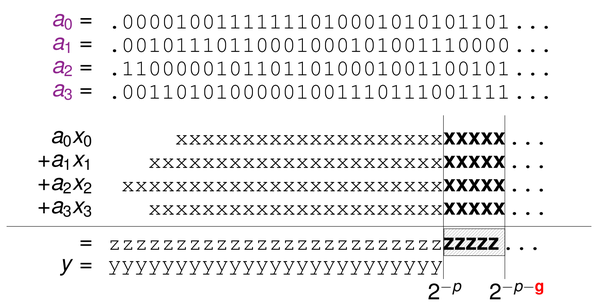
$$msb_{a_i x_i} = \lceil \log_2(|a_i| val_{max}(x_i)) \rceil$$

The MSB of the sum

- $a_i x_i$ bounded

$$msb_O = msb_y = \lceil \log_2(\sum_{i=0}^{N-1} |a_i| val_{max}(x_i)) \rceil$$

Determine the Sum's Least Significant Bit (LSB)



Suppose we use perfect multipliers: $\varepsilon_{mult} < 2^{-p-g-1}$

- after the sum:

$$\varepsilon_{y_{total}} = \sum_{i=0}^{N} \varepsilon_{mult} + \varepsilon_{final\_rounding} < N \cdot 2^{-p-g-1} + 2^{-p-1}$$

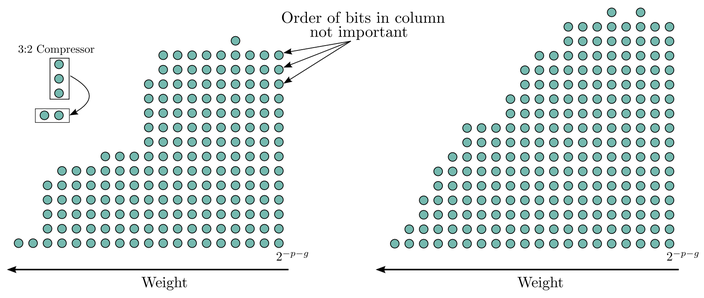
Need for larger intermediary precision

- g **guard bits**
- such that errors accumulate in the guard bits

$$\Longrightarrow g = \lceil \log_2(N) \rceil$$

Computing the Sum

- **Bit-heaps** in FloPoCo
  - bit-parallel computation of the sum
  - generalization of bit arrays used in multipliers
- Example:
  8-tap, 12-bit FIR filters used in the Zigbee protocol



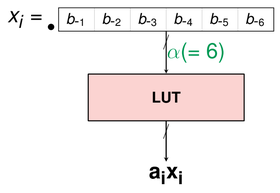
Half-Sine **Root-Raised Cosine**

Multiplying by a Constant in a FPGA

Perfect Constant Multipliers

Basic FPGA block: look-up table (**LUT**) of α input bits

- *tabulate* all the values of $\mathbf{a_i x_i}$
- ... *correctly rounded* to the output precision



Perfect fit for small sizes: x_i on α bits $\Longrightarrow$ 1 LUT/output bit

Doesn't scale:

- 2^k LUT/output bit for x_i on $(\alpha + k)$ bits

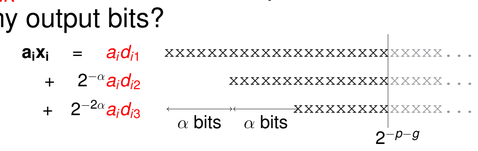
KCM Multipliers

When $x_i > \alpha$ bits: cut it into chunks d_{ik} of α bits

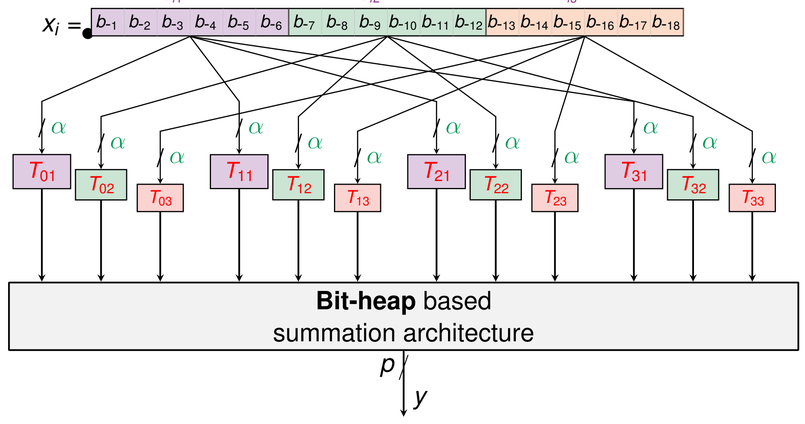
$$x_i = \sum_{k=1}^{n} 2^{-k\alpha} d_{ik} \quad \text{where} \quad d_{ik} \in \{0, ..., 2^\alpha - 1\}$$

$$\Longrightarrow \mathbf{a_i x_i} = \sum_{k=1}^{n} 2^{-k\alpha} a_i d_{ik}$$

Each $a_i d_{ik}$ tabulated, 1 LUT/output bit
How many output bits?



Sum of Products Using KCM Multipliers



Example – Root-Raised Cosine Filter

Naïve method	$p = 12$	5.9 ns (170 Mhz)	444 LUT	$\bar{\epsilon} > 2^{-9}$
Proposed method	$p = 12$	4.4 ns (227 Mhz)	564 LUT	$\bar{\epsilon} < 2^{-12}$
Proposed method	$p = 9$	4.12 ns (243 Mhz)	380 LUT	$\bar{\epsilon} < 2^{-9}$