Use of multiple numeration systems for architecture and design of a high performance FIR filter netlist generator

Ludovic Noury and Habib Mehrez
Université Pierre et Marie Curie
LIP6/ASIM Laboratory
75005 Paris, France
ludovic.noury@lip6.fr, habib.mehrez@lip6.fr

François Durbin and André Tissot
Commissariat à l'énergie atomique
CEA DAM/DIF
91680 Bruyeres le Chatel, France
francois.durbin@cea.fr, andre.tissot@cea.fr

Abstract

With increased complexity and shorter development cycle of hardware, designers can't afford to redesign similar subcomponents. They have to focus on the global design, and must extract a validated netlist from the specification of some classic component, thus insuring good performance and adaptation to their needs.

In this paper, we present the design methodology for a FIR filter netlist generator, the FIR filter being one of the most frequently used device in digital signal processing. Starting from mathematical equation and specification parameters, we include arithmetic knowledge in the generator, allowing architecture choices based on filter properties and realistic parameters values. We also provide a rounding option and a validation framework.

1 Introduction

Since filtering is widely used in digital signal processing we see the interest of a parameterizable digital filter generator and use it as an example of mathematical knowledge integration. We have selected one of the most commonly used digital filter, the Finite Impulse Response filter (FIR). FIR filters are time invariant linear discrete systems defined by the equation :

$$y(n) = \sum_{i=0}^{N-1} a_i x(n-i) \quad (1)$$

with $x(n)$ the input signal, $y(n)$ the filtered output, a_i the filter coefficients and N the number of coefficients. Since no real operation is instantaneous the implemented equation becomes :

$$y(n) = \sum_{i=0}^{N-1} a_i x(n-i-1) \quad (2)$$

The main difference between two filters lies in their template. For every filter the template defines the amplitude range $A(w)$ in which frequency response should be found: within $[1-\delta_p, 1+\delta_p]$ for the passband (PB) and within $[-\delta_s, \delta_s]$ for the stop-band (SB), δ_p and δ_s being the tolerances in PB and SB. From this template and N, severals algorithms allow coefficients extraction, [9] being one of the most frequently used. The designer can freely choose any coefficients, which are parameters for the FIR filter netlist generator.

The other difference are the arithmetic representation and world length of the input, output and coefficients. The input x is represented in simple position numeration on s_x bits. For the output y we use either fixed point or two's complement representation on s_y bits. The N normalized coefficients a_i are given to the generator in IEEE-754 double format with an associated maximum coding size s_a. The generator then converts them to fixed point representation using up to s_a bits. The coefficients aren't reconfigurable after the generation process.

To implement our generator we use the *Genoptim* framework [8] which allows to develop a generator independently of the target technology. We design a virtual description of the netlist, then *Genoptim* maps this virtual description onto the target library and optimizes the circuit.

2 Forms

Two forms are commonly used for FIR Filter (figure 1). The direct-form is the straightforward implementation of equation (2) whereas the indirect-form is obtained after some transformations. The direct-form has a longer critical path which limits clock rate. The indirect-form has a growing register size which increases the silicon surface and an input signal distributed over a large number of multipliers which increases power consumption due to high input capacitance and adds routing complexity.

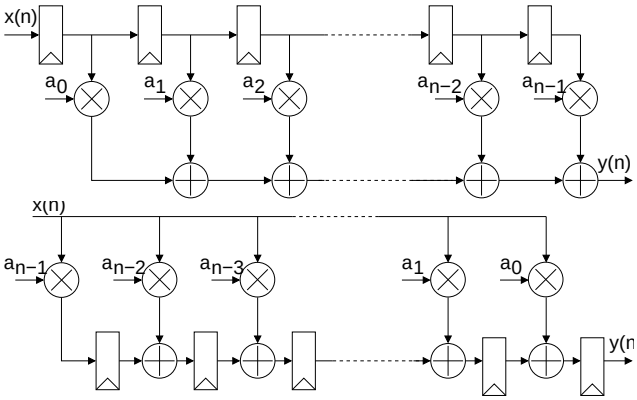

Figure 1. Direct-form (up) and indirect-form (down).

We use the direct structure prefering smaller area and lower power consumption. Moreover, this structure leaves more room for optimizations, thus reducing the clock rate limitation.

3 Constant Multiplier

Since the filter coefficients have fixed values, there is no need for a 2 operands standard multiplier, which is replaced by a new one with 1 fixed operand : a constant multiplier;also, the constants registers are removed.

A multiplier includes 2 blocks: partial products computation and summation. Since one of the operands is set, the hardware dedicated to partial products computation is greatly reduced. Therefore constant multiplier algorithms aims at minimizing the summands number.

We used the multi-base constant multiplier described in [5]. This multiplier relies on the Booth algorithm [2] adapted to constant multiplication, but uses simultaneous different bases for constant encoding. The improvement obtained from this specific operator is shown in table 1; constants in this table originate from a real application (Discrete Cosine Transform coefficients).

4 Summation

4.1 Algorithm

Wallace trees [10] are full adder trees allowing to sum efficiently n two's complement numbers, producing an output in redundant notation (carry save). We use Dadda's method [3] witch improve Wallace trees architecture, doing the minimal necessary summands matrix reduction at each level to keep the same number of levels as required by Wallace. The resulting complexity of the operator $O(\log_{\frac{3}{2}}(n))$.

Table 1. Multi-base constant multiplier compared to Booth 4 variable.variable multiplier (AMS $0,35\mu m$)

Constant	Surface μm^2		Delay ns		Power cons. $\mu W/Mhz$	
Booth4 mult.	158642	ref.	4.77	ref.	139	ref
0.707107	36939	77%	3.00	37%	29	79%
0.92388	25836	84%	2.21	54%	21	85%
0.382683	18417	88%	1.53	68%	13	91%
0.980785	16838	89%	1.61	66%	14	90%
0.83147	46387	71%	3.29	31%	35	75%
0.55557	24530	85%	1.93	60%	17	88%
0.19509	16811	89%	1.55	68%	12	91%

Back conversion to two's complement notation can be done with a standard adder.

4.2 First design

First we consider that the tree inputs are the N two's complement encoded outputs from the multipliers. The multipliers output being encoded on a $s_x + s_a - 1$ bits, the results uses $s_r = s_x + s_a - 1 + log_2 N$bits. As the inputs are signed, the sign of each multiplier output must be extended to s_r bits in order to produce a valid result. We adapt the Dadda algorithm to limit the tree growth to s_r bits.

4.3 Second design

In our previous design we did not take our application in account. The IEEE-754 double format normalized filter coefficients have been converted into a fixed point constant using up to s_a bits. Since the coefficients are normalized ($-1 < a_i < 1$), the converted values often use less than s_a bits. The input samples are unsigned and encoded on s_x bits; their maximal value is $x_{max} = 2^{s_x} - 1$. We can compute realistic maximum and minimum output values :

$$y_{max} = \sum_{a_i>0} (x_{max}.a_i) \quad y_{min} = \sum_{a_i<0} (x_{max}.a_i) \qquad (3)$$

From these values we can deduce the real minimal number of bits s_y needed to encode the output result. We now only need to extend the sign up to s_y bits and the result is on s_y bits.

$$s_y = size(\max(|y_{min}|, y_{max}) + 1 \qquad (4)$$

4.4 Final design

Reference [6] explains how to avoid partial products sign-extension in a multiplier. By extending this algorithm to the general case we can avoid sign extension in our filter. First, we anticipate the sign-extension by adding a constant

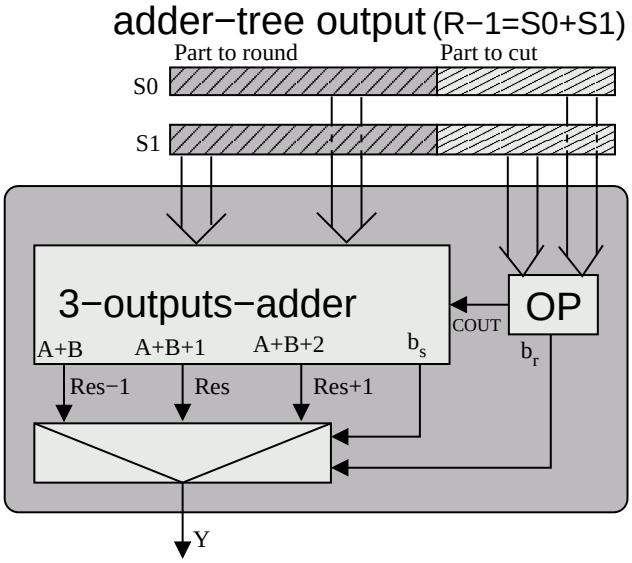

Figure 2. Rounding with the 3-outputs-adder

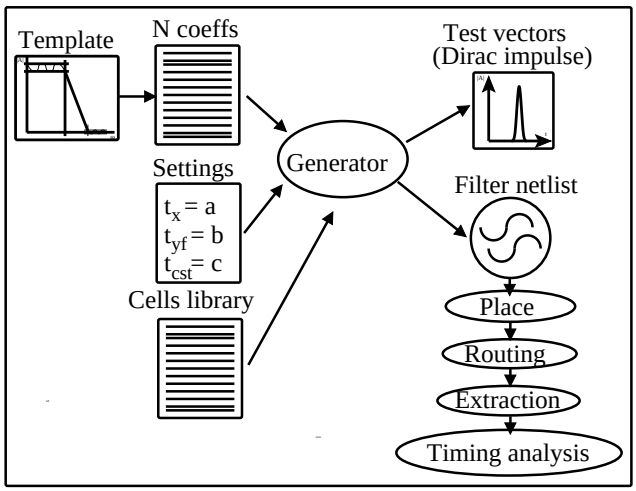

Figure 3. Generation flow

equal to the sum of all sign-extensions (including the sign bit) that would be done if all numbers were negative. Then by inverting the sign-bit of every summand, useless sign extensions are canceled.

However, in our case, we extend the sign bit of the first term to s_y bits thus forcing the sign bit on the msb (the output still being encoded on s_y bits).

If this technique has the drawback to add, in the worst case, a stage to the adder tree by adding a constant, the hardware gain in surface and power consumption widely balance the potential small delay increase (one more full-adder in the long chain). For example, with the simplest filter we generated, a $N = 19$ coefficients lowpass filter with $s_a = 16$ bits and $s_x = 10$, we switch from 442 full adders and 8 half-adders to 387 full-adders and 15 half-adders for the adder tree.

5 Rounding

With our adder tree, our output format is in redundant notation. To convert to a classical notation (two's complement, simple position numeration or fixed point) we must use an adder and eventually truncate the result.

Rather than truncating the result as we often do on DSP filters, we will take advantage of a little known particular adder which will allow us to round at nearly the same cost as a truncation, still providing more accuracy.

If we want to round the result r to r_g bits we need the b_r bit which immediately follows the r_g group and also the result bit sign b_s. Then, the rounded result being rr we have 4 possible cases:

1. $b_s = 0, b_r = 0$ then $rr = r$
2. $b_s = 0, b_r = 1$ then $rr = r + 1$
3. $b_s = 1, b_r = 0$ then $rr = r$
4. $b_s = 1, b_r = 1$ then $rr = r - 1$

To do the rounding operation we need r, $r+1$ and $r-1$. Reference [1] describes the architecture of a 3-outputs adder computing "$A + B, A + B + 1, A + B + 2$" from A and B inputs, with a complexity of the order than a Carry Look Ahead adder (CLA), one of the fastest adder $O(log_2 N)$. This special adder is constructed from a CLA with slight alterations of the last stages. We note that though the changes producing a "$A+B-1, A+B, A+B+1$" 3-outputs adder aren't complex, we however chose a simpler way. We just modify the adder tree so that the output is $A + B - 1$ by decreasing the Fadavi constant by one. The resulting architecture is represented on figure 2.

6 Results and Validation

The final generator can be used following the generation flow on figure 3. First, the coefficients are extracted from the filter template, with a tool such as *Matlab*. Then, these coefficients are transmitted to the generator together with the filter implantation parameters. The generator, using these parameters and a standard cell library produces a netlist in the user selected format along with a Dirac signal vector used later for the validation process. The resulting netlist is then placed and routed before timing analysis.

The validation flow (figure 4) consists in simulating the netlist with a Dirac signal as input, the filter output corresponding to the filter coefficients. These coefficients are

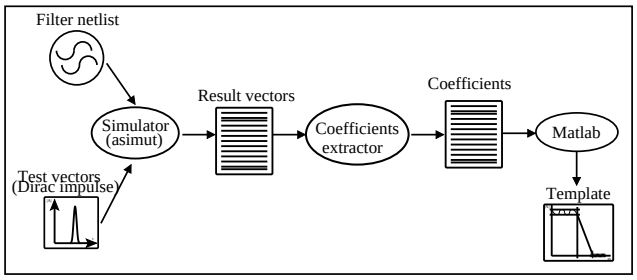

Figure 4. Validation flow

Table 2. FIR Filter generator results

Filter Class	Freq. (Mhz)	Transistors (number)	Ripple/Template (dB)	
			passband	stopband
lowpass 1	37	140701	10^{-3}	4
lowpass 2	44.6	56586	10^{-3}	1.4
bandpass	38.44	136555	10^{-2}	0.3-3.1
highpass	37.93	178898	10^{-3}	7
bandstop	39.84	129462	10^{-3}-10^{-2}	1

then forwarded to a filter analysis tool which produce the implemented filter template. Finally, we compare the obtained results with the initial template to check the filter validity.

This validation process was done with a set of typical filters templates: lowpass ($N = 51$ and $N = 19$), bandpass ($N = 51$), highpass ($N = 67$) and bandstop ($N = 47$). The benchmarking of these filters was done according to the generation flow. All filters were generated with input data encoded on 10 bits, constants on 16 bits and maximum output accuracy. The simulation, placement and routing were done with the open source *Alliance CAD System* [7] using AMS $0,35\mu m$ technology. The netlist was mapped onto *Alliance sxlib* portable standard cells library. From simulation results, filter magnitude response were extracted and compared with the corresponding refence filters (figure 5 and table 2).

7 Conclusion

Results analysis show that our generator can provide a wide range of FIR filter netlists. Resulting filters provide highly increased performances compared to the basic architecture implementation ussually obtained through synthesis tools. With these generators the designer is allowed to evaluate various architectures in short time. Time and flexibility gained by using generators show the interest of a generators framework allowing fast and safe development.

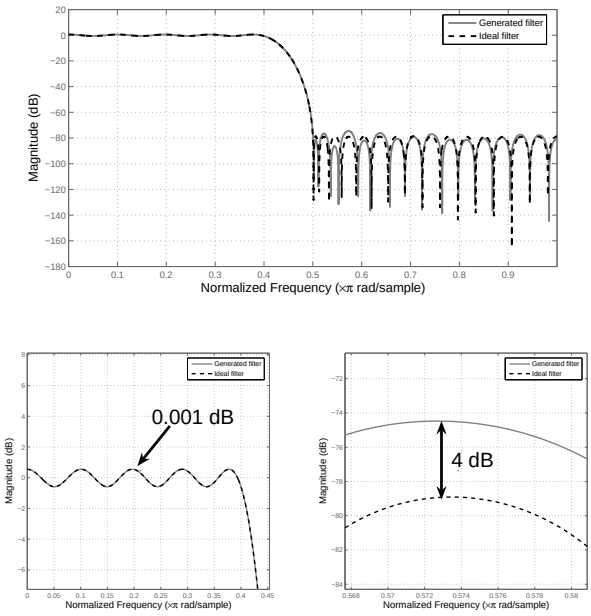

Figure 5. original and implanted lowpass filter ($N = 51$), : magnitude response (top), passband (left), outside passband (right)

References

[1] R. Avot. *Architectures matérielles pour l'arithmétique stochastique discrète.* PhD thesis, Pierre et Marie Curie university, 2003.

[2] A. Booth. A signed binary multiplication technique. *Quartely J. Mechanics and Applied Mathematics*, 4(2), 1951.

[3] L. Dadda. Some schemes for parallel multipliers. *Alta Frequenza*, 45:574–580, 1976.

[4] R. Daouphars and L.-S. Didier. Use of multiple number representation in automatic arithmetic data-path design. *in Proceedings of the SPIE - Advanced Signal Processing Algorithms*, Architectures, and Implementations XIII, 2003.

[5] Y. Dumonteix. *Optimisation des chemins de données arithmétique par l'utilisation de plusieurs systèmes de numération.* PhD thesis, Pierre et Marie Curie university, 2001.

[6] J. Fadavi-Ardekani. Mxn booth encoded multiplier generator using optimized wallace trees. *IEEE Transaction on very large integration (VLSI) Systems*, vol. 1:no. 2, 1993.

[7] A. Greiner and F. Pecheux. A complet set of cad tools for teaching vlsi design. *Third EuroChip Workshop on VLSI Design Trainning*, september 1992.

[8] A. Houelle, H. Mehrez, and N. Vaucher. On portable macrocell fpu generators using the 754-ieee standard. *ICSPAT*.

[9] Y. Lim and S. Parker. Fir fi lter design over a discrete powers-of-two coeffi cient space. *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-31:pp.583–591, june 1983.

[10] C. Wallace. A suggestion for a fast multiplier. *Prc. Techno*, 90,12(11):14–17, 1964.