Novel Area-Efficient FPGA Architectures for FIR Filtering With Symmetric Signal Extension

A Benkrid[1] and K Benkrid[2]
[1]School of Computer Science, The Queen's University of Belfast, Belfast BT7 1NN, UK
[2] The University of Edinburgh, School of Engineering and Electronics, Edinburgh, EH9 3JL, UK

Abstract

This paper presents four novel area-efficient FPGA bit-parallel architectures of Finite Impulse Response (FIR) filters that smartly support the technique of symmetric signal extension while processing finite length signals at their boundaries. The key to this is a clever use of variable-depth shift registers which are efficiently implemented in Xilinx FPGAs in the form of SRL16 component. Comparisons with the conventional architecture of FIR filter with symmetric boundary processing show considerable area saving especially with long-tap filters. For instance, our architecture implementation of the 8-tap low Daubechies-8 FIR filter achieves ~30% reduction in the area requirement (in terms of slices) compared to the conventional architecture while maintaining the same throughput. Two of the above-cited novel architectures are dedicated to the special case of symmetric FIR filters. The first architecture is highly area-efficient but requires a clock frequency doubler. While this reduces the overall processing speed, it does maintain a high enough throughput to achieve real time performance. Moreover, this speed penalty is cancelled in bi-phase filters which are widely used in multirate architectures (e.g. wavelets). Our second symmetric FIR filter architecture saves less logic than the first architecture (e.g. 10% with the 9-tap low Biorthogonal 9&7 symmetric filter instead of 37% with the first architecture) but overcomes its speed penalty as it matches the throughput of the conventional architecture.

1. Introduction

Finite Impulse Response (FIR) filters are widely used in digital signal processing. A K-tap FIR filter is defined by the following input-output equation [1]:

$$\text{out}(n) = \sum_{i=0}^{K-1} x(n-i).h(i) \qquad (1)$$

where {h(i): i = 0,…, K-1} are the filter coefficients.

Figure 1 shows the two conventional architectures (the direct and the inverse form) of a general K-tap FIR filter hardware implementation [2]. Both architectures seek to align the products *x(n-i).h(i)* in Eq.1 in time before being accumulated. In architecture (a), the chained input samples delays (Δ) align in time the input samples *x(n-i)* before *parallel* multiplication and accumulation. Whereas, in architecture (b), the products *x(n-i).h(i)* are aligned in time through the internal delays (Δ) of the *adder chain*.

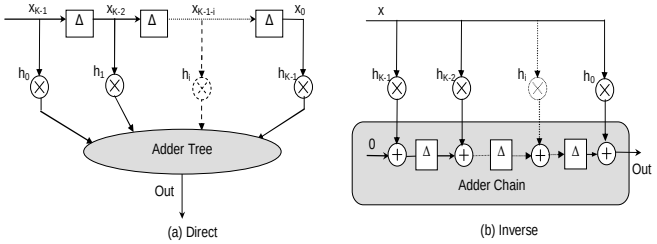

Figure 1. Two conventional FIR filter architectures

An FIR filter implements a convolution operation [3], which is often built on the assumption of *infinite length signals* e.g. continuous audio signal. *Finite length signals* (e.g. images) on the other hand, have discontinuities at the boundaries. At this point the problem of which values to use at these regions emerges. Although, this problem could be ignored for a one-stage convolution, it cannot be discarded when implementing a multi-stage convolution as in subband coding and Filter banks which are widely used in Speech , Image and Video processing standards and applications [4, NEW REFERENCE]. A commonly recommended solution (called *symmetric extension*) to this problem is to extend each row by reflection at the signal boundary [5] as shown in Fig.2.

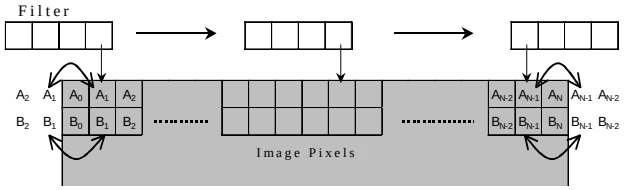

Figure 2. A finite 2-D signal (image) with symmetric boundary extension

With *length-preserved* filtering using a K-tap general FIR filter, the minimum number of extra samples to be introduced is constant and equal to *K-1*. This is because *P+(K-1)* input samples are required to generate *P* output samples. However, the number of samples to be added at the left border of the input signal (referred to by α) and the right one (referred to by μ) can be variable, i.e. not a constant.

To handle the problem of boundary filtering in hardware, Chakrabati [6] proposed the use of a router (or switcher) to feed the appropriate data, in parallel, to the multipliers (see Fig.3).

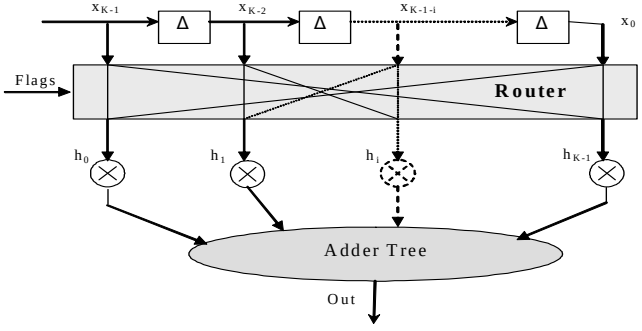

Figure 3. A conventional FIR architecture with a hard-router to handle the boundary processing

The router (referred to by *Hard-Router*) configuration is detailed in [6]. It is implemented using multiplexers where a controller is needed to drive their appropriate selection signals. For a K-tap FIR filter, a minimum of $W(\sum_{i=2}^{\alpha+1}\lceil i/2\rceil+\sum_{j=2}^{K-\alpha}\lceil j/2\rceil)$ four-to-one Look Up Tables (LUTs) are required to implement the Hard-Router when the input signal is extended by α samples at its left side [7]. This represents an $O(K^2W)$ hardware complexity[1]. Consequently, the Hard-Router solution requires considerable area and routing resources which will inevitably degrade the speed performance.

To reduce primarily the high area cost of the Hard-Router, we present in this paper novel hardware architectures for FIR filters tailored to Xilinx FPGAs. These architectures cleverly exploit the variable-depth Shift Register Logic component: SRL16 [8], [9] implemented by each LUT in a number of Xilinx FPGA series (see Fig. 4). These stretch from early Xilinx Virtex series, through to Virtex-II, Virtex-IIPro, Virtex-4 and Virtex-5 series and their low cost equivalents (e.g. Spartan-II, Spratan-IIE and Spartan-3 series). Indeed, in all of these FPGAs, each slice LUT can be configured to create a shift register (SRL16) that varies in length from 1 to 16 bits. The SRL16 configuration consists of a chained delay with a multiplexer at the output. The input address *Addr[3:0]* selects which bit in the chained delay to be output hence controlling the length of the shift register from 1 to 16 (see Fig.4). Longer shift register length can be implemented with multiple chained SRL16. Note that each SRL16 can be immediately pipelined by using the flip-flop available at the same slice logic cell.

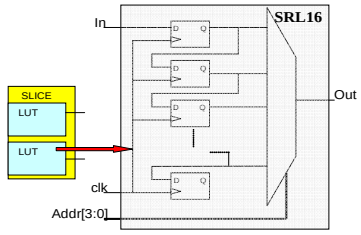

Figure 4. SRL16 configuration of a LUT

In the remainder of this paper, we first present our novel hardware architecture for a general FIR filter regardless of the signal boundaries extension. We then show how our architecture can be efficiently used to handle symmetric signal extension with little hardware penalty. A detailed scheduling algorithm is presented. We then compare the area requirement of our architectures to that of the conventional "Hard-Router" based FIR architectures. A case study showing real timing and area measurements is included. In section 3, we tailor the results given in [10] to the special case of symmetric FIR filters. Two novel architectures are presented. Similarly, area cost evaluation and a dedicated case study are provided. Finally, conclusions are drawn. It is worth noting that this paper is an enhancement and completion of previous work published by the authors in [7] [10] [11]. This includes a novel architecture for symmetric FIR filters which overcomes previous speed penalty reported in [7] as well as a detailed description of the algorithm used in symmetric signal boundary handling for both general FIR filtering with adder chain accumulator structure and symmetric FIR filters.

Throughout the rest of this paper, we assume the use of bit parallel arithmetic. The term SRL corresponds to either one SRL16 component or a chained SRL16 components if needed. The abbreviation 'cc' denotes the term clock cycle and the term $SRL_{(i)}$ refers to the SRL associated with the filter coefficient h_i.

[1] $\sum_{i=1}^{K} i = K(K+1)/2 = O(K^2)$

2. General FIR Filter

This section presents our novel hardware architecture of a general K-tap FIR filter. It explains how symmetric boundary signal processing is cleverly handled. The area efficiency of our novel architecture is demonstrated through a comparison of the logic area requirements of our architecture compared to the conventional architecture. A case study is included at the end to quantify the area savings as well as speed performance.

2.1. A Novel Architecture of a General K-tap FIR Filter

Figure 5 shows our novel architecture of a general K-tap FIR filter. The accumulator can have an *adder chain* or *adder tree* structure. The novelty of this architecture lies in the introduction of a SRLs layer.

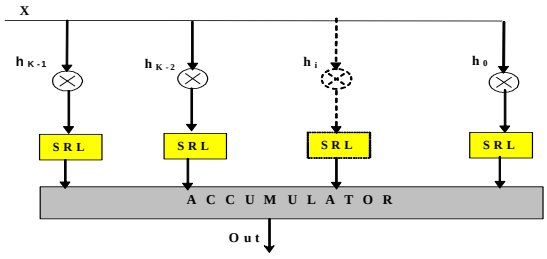

Figure 5. Novel architecture of a General FIR Filter

The input data samples *x* in the architecture are first multiplied in parallel with the filter coefficients. Then, the SRLs skew and align the multiplication results *x(n-i).h(i)* properly in time as shown in Fig.6. The skewed products are then summed up to produce the filter outputs.

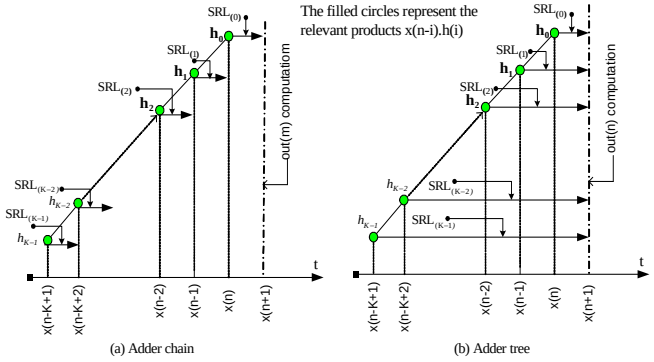

Figure 6. Data Dependence Graph (DDG) of a K-tap general FIR filter. The horizontal arrows show the SRLs' skew lengths.

Note that the SRLs in Fig.5 can be placed before or after the multipliers. Thus skewing the products *x(n-i).h(i)* or the multiplicands *x*. If the filter coefficients are fractional and truncation is carried out at the output of the multipliers, the wordlengths at the outputs of the multipliers could be smaller than at their inputs. Thus, for area optimisation, the SRLs should be placed at the multipliers' outputs. On the other hand, if the filter coefficients values are greater than 1, the wordlengths of the outputs of the multipliers will be greater than at their inputs. Then, placing the SRLs before the multipliers saves us more logic. As a consequence, our architecture of Fig.5 can have four varieties:

1. Adder tree with pre-multipliers SRLs
2. Adder chain with pre-multipliers SRLs

3. Adder tree with post-multipliers SRLs
4. Adder chain with post-multipliers SRLs

In the following, we consider only case 3 and 4. The same analysis and results still hold valid for case 1 and 2 where the raw input samples instead of the products are skewed. In fact, skewing a product: *x(n-i).h(i)* by *L* clock cycles (cc) is equivalent to skewing *x(n-i)* by the same number of clock cycles.

2.1.1 Filtering with no Boundary Extension

When the accumulator structure is an *adder chain*, each product is delayed by only one clock cycle (cc) through the relevant SRL (see Fig.6.a). In fact, with the exception of the SRL layer, the architecture of our filter is similar to that of Fig.1.b, which aims to supply the products to the adder chain as soon as they are computed. The products are aligned in time through the internal delays (Δ) of the adder chain structure and not necessarily through the SRL layer. The latter increases only the FIR latency. On the other hand, it is up to the added SRL layer to skew the products and align them properly in time before parallel accumulation as shown in Fig.6.b if the accumulator structure is an adder tree. This is because unlike the structure of Fig.1.a, the structure of Fig.5 does not include an input samples delay chain.

Figure 6 will be used to deduce each SRL skew length. This is given in table 1. With an adder chain accumulator structure, the SRLs' skew lengths are all equal to one[2] (see Fig.6.a). Whereas with an adder tree structure, each SRL skew length is equal to the time interval between the SRL product instant and the filter output computation instant (see Fig.6.b)

Accumulator Type	$SRL_{(0)}$	$SRL_{(1)}$	$SRL_{(2)}$	….	$SRL_{(K-2)}$	$SRL_{(K-1)}$
Adder Tree	1	2	3		K-1	K
Adder Chain	1	1	1	…	1	1

Table 1. The SRLs' skew length for a general FIR filter with No boundary processing

Besides being able to implement a convolution operation, our architecture seeks primarily to handle signal boundaries processing. This is handled efficiently with very little hardware overhead, thanks to the SRLs *dynamic skewing* feature as shown in the following section.

Throughout, the term P_i refers to the FIR Data Dependency Graph (DDG) product associated with the multiplier h_i. In particular, P_{K-1} (P_0) refers to the first (last) FIR DDG's node product.

2.1.2 Filtering with Symmetric Boundary Extension

To implement in hardware filtering with symmetric boundary extension, no alteration on the architecture of Fig.5 is necessary. The boundary filtering is simply implemented by a proper skewing of the products P_i through *a dynamic SRL addressing.*

Figure 7 shows for instance a 4-tap general FIR filter DDG at two consecutive sequences boundary region (the deduction of such graph will be detailed later in the section). In this figure, $Sequence_I$ refers to the current sequence of input samples to be filtered, the negative values (-1, -2, -3…) the instants which precede the start of this sequence, the shaded rectangle shows the boundary region between the two consecutive sequences (I-1) and I, and the dashed arrows show where the input symmetric extension is applied. The boundary DDGs of $Sequence_I$ (shown in dashed lines) represent the *possible* DDGs at the *left-hand* side signal boundary, whereas the ones associated with $sequence_{(I-1)}$ represent the possible DDGs at the signal *right-hand* side boundary. We say possible as this depends on the location of the symmetry-filtering axis which is determined by the number of input samples (α) introduced at the left-hand side of the input signal and the number of input samples (μ) introduced at the signal right-hand side (see section 1).

From Fig.7 and as a general rule we can see that the *regular* DDG (a straight line as depicted in Fig.6) of a general K-tap FIR filter ends at the P_{K-1} of instant "*–K*" (= -4 in Fig.7), and starts from the P_0 of time "*K-1*" (= 3 in Fig.7). Between these two values, the DDGs become irregular. Because of the DDG irregularity, the SRLs skew lengths given in table 1 are no more valid.

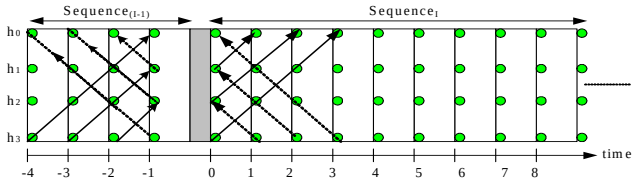

Figure 7. DDGs of a 4-tap FIR filter (K=4) using symmetric boundary extension

To find the required SRLs' skew lengths, and subsequently the required addresses, according to the filter length K and the symmetry-filtering axis, we suggest in the following a dedicated approach. In this paper, we consider only the filter with adder chain accumulator structure. Details regarding the adder tree accumulator case can be found in [10].

Throughout, the term *Hub* refers to the *irregular* DDG deflection point. The term P_{Hub} denotes the *Hub* associated product whereas P_{Hub-i} (P_{Hub+i}) represents the i^{th} product that comes before (after) the *Hub* where *i* is a positive integer.

- **An Approach to Determine the SRLs' Skews lengths when Using the Symmetric Signal Extension**

When using an adder chain accumulator structure and as already shown in Fig.6.a, the relevant products P_i for each filter output should be fed to the accumulator regularly in time such that the product P_j is fed to the j^{th} adder *(j-i)* cc's *before* feeding the product P_i to the i^{th} adder where j>i. Based on this rule (referred to **rule 1**), the required new SRLs' skew lengths when handling boundary filtering using symmetric extension can be deduced. This is achieved by following the next two steps.

a) Left Side Signal Boundary Extension

Figure 8 shows the DDG at the left side boundary of the input $Sequence_I$ when α input samples are introduced through symmetry reflection (see the arced arrows). This was necessary since the P_{Hub-i} multiplicands at this boundary region correspond to the previous $Sequence_{(I-1)}$ input samples (see the dashed line). The resulting boundary DDG becomes irregular, i.e. not a straight line.

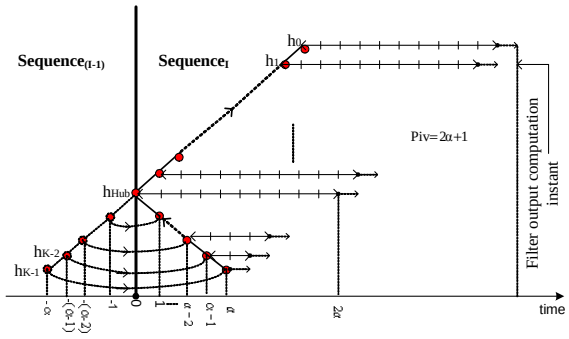

Figure 8. The irregular DDG for an FIR filter at the left side signal boundary when using symmetric extension.

[2] Actually the SRLs' skew lengths can have any value, a value greater than one will only increase the filter's latency

Since the input signal is extended at its left side by the first α samples of $Sequence_I$, we need to wait for α cc's before starting to accumulate the necessary products of the *first output*. The first partial accumulation is initiated one *cc* after computing the product P_{K-1} which coincides with instant α (see the dashed arrow in Fig.8). This corresponds to a P_{K-1} delay of one *cc*. According to rule 1, the product P_{K-2} should feed the adder chain one *cc* after feeding P_{K-1} to the adder chain. As such, the product P_{K-2} of instant (α-1) is skewed by a three (and not just one!) *cc*'s delay (see Fig.8). This represents a 2 *cc*'s increment to the regular P_{K-2} skew length (see Fig.6.a). This *compensation* is necessary since the product P_{K-2} in the irregular DDG is computed one cc earlier than P_{K-1} instead of one cc later.

By following the same reasoning and because of the DDG linearity, we can deduce easily that each P_{Hub-i} needs to be delayed by a 2 extra cc's than its predecessor $P_{Hub-i-1}$. Therefore, the P_{Hub} skew length is equal to ($1+2.\alpha$) cc's. We refer to this value by the term *Piv*, i.e.

Piv = $2\alpha+1$. This skew value should be applied on all the P_{Hub+i} products since they are, along with P_{Hub}, regularly and adequately separated in time, i.e. P_{Hub+i} is produced *i* cc's after P_{Hub} (see Fig.6.a and 8). For instance, according to rule 1, P_{Hub+1} should be fed into the adder chain one *cc* after P_{Hub}. Since P_{Hub+1} is computed one *cc* after P_{Hub} (see Fig.8), P_{Hub+1} should be skewed by the same P_{Hub} skew length, i.e. *Piv cc*'s.

From what precedes, we can conclude that if a finite length signal is extended by α samples at its left side, the new DDG's products skews for the *first* filter output can be represented by a skew list L_1 such that:

L_1=[1, 3, 5, 7,…, Piv-2, Piv, Piv, Piv, …, Piv], where Piv=$2\alpha+1$.

The underlined element in the list refers to the skew length of the hub's SRL. The skew values in the above list are applied subsequently on the filter's SRLs from left to right, regularly delayed in time by one cc (see Fig.8).

The irregularity of the filter's DDGs does not occur only with the first filter output of $Sequence_I$, but rather with all the first α outputs. Consequently, we need to determine the remaining SRLs skew lists associated with the next "α-1" outputs.

From figure 9, we can see that the computations of the P_{Hub-i} products of the second filter output are done one cc *earlier* than with the first filter output. For instance, the product P_{K-1} of the second filter output is computed at instant "α-1", a one *cc* earlier than with the first filter output. Inversely, the computations of the P_{Hub+i} products of the second filter output are done one *cc later* than with the first filter output.

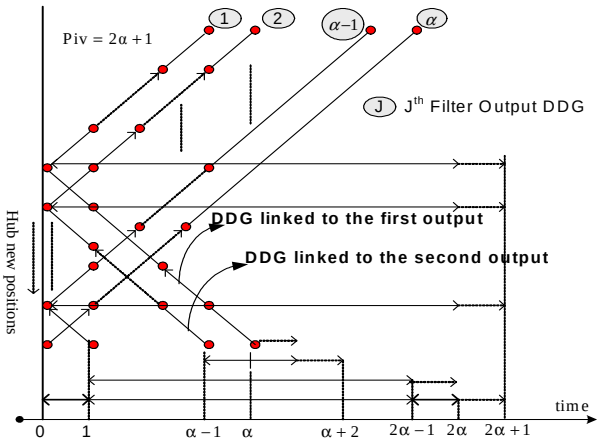

Figure 9. The irregular DDGs for an FIR filter at the left side signal boundary when using symmetric extension

Since the accumulation of each filter output products should be initiated at each new *cc*, the product P_{K-1} of the second filter output should be fed to the adder chain one *cc* after feeding the accumulator with the P_{K-1} of the first filter output. This corresponds to the instant "α+2" (see Fig.9). Therefore, the required SRL_{K-1} skew length for the second filter output is equal to 3 cc's delay (see Fig.9). The remaining SRLs skew lengths of this second filter output can be deduced as already explained with the first filter output since the associated DDGs have the same shape. The reader can easily verify that the skew delay list L_2 associated with the second output is:

L_2=[3, 5, 7,…, Piv-4, Piv-2, Piv, Piv, Piv, Piv, …, Piv]

The above list can be interpreted as the *left shift* of the list L_1 where the right side is filled with the value *Piv*. This rule can be extrapolated to the remaining boundary filter outputs skew lists. As a consequence, all the DDGs hubs are delayed by the same number of clock cycles (*Piv*) since for each new irregular DDG, the hub moves one position backward (see Fig.9), as does its associated skew list index. Therefore, we can conclude that the skew list for the last irregular output (P_{K-1} of instant 1, see Fig.9) is:

L_α=[Piv-2, Piv, Piv,…, Piv]

The first output with a *regular* DDG corresponds to P_{K-1} of instant 0 (see Fig.9). It should be delayed by *Piv* cc's as it has to be fed to the adder chain one *cc* after P_{K-1} of instant 1 (see Fig.9). Logically, this value is applied equally to the rest of the regular DDG's products. We refer to the resulting list by the *regular skew list RegSkew*, where

RegSkew = [Piv, Piv, Piv,…, Piv]

This list remains valid through the entire non-boundary regions.

Therefore when the input signal is extended by symmetry at its left side by α samples, the required SRLs skew lengths at the boundary region can be represented with a *Left-Matrix* matrix expression where:

Left-Matrix = [α_Matrix | Piv*I(α,K-α-1)].

I(α, K-α-1) refers to the identity matrix of size [α, K-α-1], and α_Matrix is of size [α, α+1], such that:

$$\alpha\_Matrix=\begin{bmatrix} \mathbf{1} & \mathbf{3} & \mathbf{5} & \mathbf{7} & ... & \mathbf{Piv-4} & \mathbf{Piv-2} & \underline{Piv} \\ \mathbf{3} & \mathbf{5} & \mathbf{7} & \mathbf{9} & ... & \mathbf{Piv-2} & \underline{Piv} & Piv \\ \mathbf{5} & \mathbf{7} & \mathbf{9} & \mathbf{11} & ... & \underline{Piv} & Piv & Piv \\ . & . & . & . & & & . & . \\ . & . & . & & & . & . & . \\ \mathbf{Piv-6} & \mathbf{Piv-4} & \mathbf{Piv-2} & \underline{Piv} & Piv & ... & Piv & Piv \\ \mathbf{Piv-4} & \mathbf{Piv-2} & \underline{Piv} & Piv & ... & Piv & Piv & Piv \\ \mathbf{Piv-2} & \underline{Piv} & Piv & ... & Piv & Piv & Piv & Piv \end{bmatrix}$$

The underlined elements in this matrix refer to the skew length of the hub's SRL. All the matrix elements put on bold refer to the pre-hub SRLs skew lengths where irregularity is occurring. They form an upper triangular matrix.

b) Right Side Signal Boundary Processing

Figure 10 shows the DDG at the right side signal boundary where β samples are introduced through symmetric reflection (see the arced arrows). This was needed since the P_{Hub+i} multiplicands at this boundary region correspond to samples from $sequence_{(I+1)}$ samples (see the dashed line). The resulting boundary DDG is irregular, i.e. not a straight line.

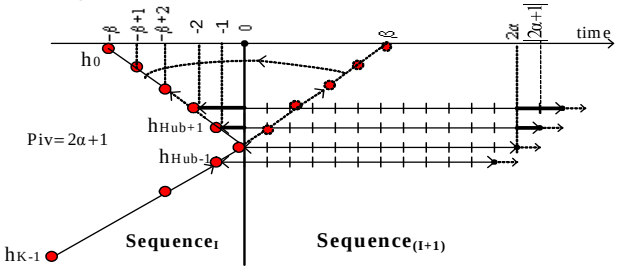

Figure 10. The irregular DDG of a general FIR filter at the right side signal boundary when using symmetric extension.

As already explained in the last section, all the regular DDGs products will be delayed by *Piv* cc's. This is portrayed through the product P_0 skew length in *the last* regular DDG in Fig.11. The next P_0 is supplied to the adder chain at instant "*Piv+1*" cc, and corresponds to the first *irregular* DDG. The hub of the latter should be fed into the adder chain one *cc* earlier according to rule 1, i.e. at instant *Piv*. Since this hub product is computed at instant 0 (see Fig.11), the skew length of the irregular DDG hub is equal to *Piv*, which is equal to the skew length of all the *precedents* DDGs hubs (see section a).

By following the same reasoning on the subsequent irregular DDGs in Fig.11, we can easily deduce that the required skew length of the remaining irregular DDGs' hubs is also equal to *Piv* cc's. This result will be used to deduce the skew delays of the remaining irregular DDGs products.

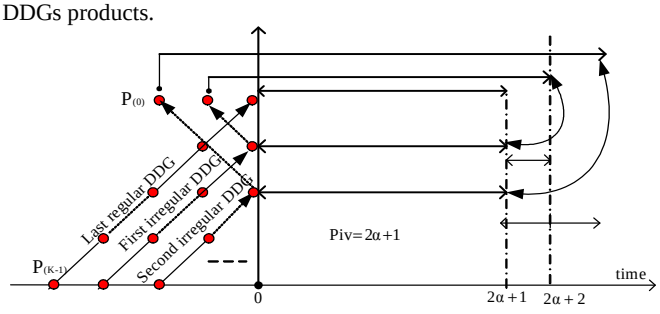

Figure 11. The irregular DDGs of an FIR filter at the right side signal boundary when using symmetric extension.

According to Fig.11, all P_{Hub-i} products are separated in time as required by rule 1. Therefore they need to be skewed with the same P_{Hub} skew length value, i.e. *Piv*. On counter part, the skew length of the post-hub products P_{Hub+i} need to be updated as rule 1 is broken.

If the signal is extended by β samples from the right, the product P_{hub+1} has to be skewed by *Piv+2* cc's delay (see Fig.11). This is because P_{hub+1} product should be supplied to the adder chain one *cc* after P_{hub}. However P_{hub+1} is computed one cc earlier (see Fig.11). To cope with this irregularity, 2 cc's delay should be then added to the P_{hub} skew delay, and that is where the value *Piv+2* comes from. The same reasoning once applied on the subsequent products shows that the skew length of $P_{hub+i+1}$ should be 2 cc's greater than of P_{hub+i}.

Consequently, when symmetrically extending the signal by β-sample from its right side, the skew lengths of P_{hub+i} can be grouped into a skew list R_1 such that

R_1= [Piv+0, Piv+2, Piv+4, Piv+6,…., Piv+2β]

This list values are applied from left to right on the filter's $SRLs_{hub+i}$ ($0 \le i \le \beta$) elements. Since the value of β ranges from 1 to μ (see section 1), the skew lengths associated with the irregular DDGs at the right signal boundary can be represented with a *Right-Matrix* matrix expression where:

Right-Matrix = [Piv*I(μ, K-μ-1) | μ_Matrix].

I(μ, K-μ-1) refers to the identity matrix of size [μ, K-μ-1], and μ_Matrix is of size [μ, μ+1] such that:

$$\mu\_Matrix=\begin{bmatrix} Piv & Piv & Piv & Piv & Piv & \dots & \underline{Piv} & \mathbf{Piv}+2 \\ Piv & Piv & Piv & Piv & \dots & \underline{Piv} & \mathbf{Piv}+2 & \mathbf{Piv}+4 \\ Piv & Piv & Piv & \dots & \underline{Piv} & \mathbf{Piv}+2 & \mathbf{Piv}+4 & \mathbf{Piv}+6 \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ Piv & Piv & \underline{Piv} & \mathbf{Piv}+2 & \mathbf{Piv}+4 & \dots & \mathbf{Piv}+2\mu\text{-}6 & \mathbf{Piv}+2\mu\text{-}4 \\ Piv & \underline{Piv} & \mathbf{Piv}+2 & \mathbf{Piv}+4 & \mathbf{Piv}+6 & \dots & \mathbf{Piv}+2\mu\text{-}4 & \mathbf{Piv}+2\mu\text{-}2 \\ \underline{Piv} & \mathbf{Piv}+2 & \mathbf{Piv}+4 & \mathbf{Piv}+6 & \dots & \mathbf{Piv}+2\mu\text{-}4 & \mathbf{Piv}+2\mu-2 & \mathbf{Piv}+2\mu \end{bmatrix}$$

The underlined elements in this matrix refer to the hubs' SRLs skew lengths. All the matrix elements put in bold refer to the post-hub SRLs' skew lengths. They form a lower triangular matrix.

The above matrix along with *RegSkew* list and *Left_Matrix* determine the required skew lengths of the SRLs in the architecture of Fig.5 with *adder chain* accumulator structure when extending the input signal boundaries by symmetry. The expression of these matrices for the case of *adder tree* accumulator structure can be found in [10].

2.2 Area Comparisons

In the following, we compare the area cost of Fig.3 and Fig.5 architectures. Table 2 lists the FPGA logic resources used by these two architectures. It is worth noting that when using Xilinx FPGAs, the internal delays of the adder chain are implemented using the "free" flip flops available in the slices. The LUTs of those slices will be used to implement the combinatorial adders of the adder chain. Thus the cost of the delays in the adder chain can be considered null if the cost of adders has been already considered as it is done in table 2.

	Figure 3 architecture	Figure 5 architecture
Number of Multipliers	K	K
Number of Word Delays	K-1	0
Number of Adders	AdderTree(K)	Accumulator (K)
Router Hardware (LUTs)	$W(\sum_{i=2}^{\alpha+1}\lceil i/2\rceil+\sum_{j=2}^{K-\alpha}\lceil j/2\rceil)$	SRLs Layer

Table 2. Logic resources consumed by different FIR filter architectures. K is the filter length and W is the input wordlength.

From table 2, we can see clearly that our architecture with its two varieties (adder chain and adder tree accumulator structure) does not necessitate input samples delays, thus saving (*K-1*) parallel word delays, and consumes the same number of multipliers and nearly the same number of adders as the architecture of Fig.3. Indeed, our architecture uses either a K-input *adder chain* or *adder tree*. Those two types have been grouped in table 2 under the term *Accumulator (K)*. It is well known that both of these accumulators' structures consume nearly the same number of (equivalent) adders, i.e. K. The router's logic resources requirements given in table 2 need now to be inspected carefully.

Table 2 gives the number of LUTs consumed by the hard router as explained in section 1. With our architecture, the routing functionality is implemented through the SRL layer. If the required SRL's skew length is less or equal to 16, one SRL16 (one LUT) can be used. However, if more depth is needed, more SRL16 should be chained, thus increasing the required number of LUTs. Therefore, the *Left_Matrix*, *Right_Matrix*, and *RegSkew* expressions given above are used to determine the number of SRL's LUTs per multiplier. This is equal to $\lceil DL/16 \rceil$ where DL represents the maximum skew length (DL) value of an SRL. If we omit the area cost of the SRLs' address generators (which can be considered negligible in comparison with the final filter area), our SRL layer area cost will depend solely on the number of SRLs used. Figure 12 plots the router area cost evolution for a K-tap filter [11]. It shows clearly that the hard router consumes much more area than our suggested architecture. The SRL layer cost has a reverse bell shape when using an adder tree accumulator structure. It is minimal for α values equal to $\lfloor \frac{K-1}{2} \rfloor$ and $\lceil \frac{K+1}{2} \rceil$. On the other hand, the SRL layer cost has a stair-wise shape when using the adder chain. It consumes less logic than with the adder tree structure only for α values much smaller than $\lfloor \frac{K-1}{2} \rfloor$ (see Fig.12).

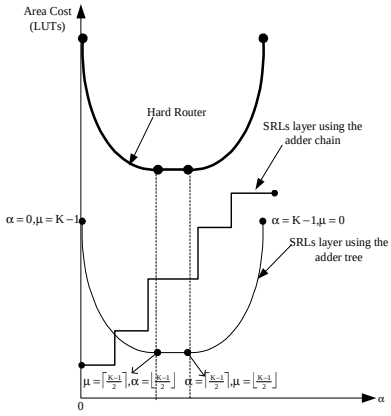

Figure 12. Soft and Hard Router area cost comparison for K>9♦

From what precedes, we can conclude that our novel architecture consumes fewer logic resources than the conventional architecture of Fig.3. This is shown clearly in Fig.13. This figure does not include the cost of the multipliers as it is the same in all the architectures and allocates the same number of LUTs to a W bit parallel delay and W-bit parallel adder. This is valid thanks to the dedicated fast carry logic in Xilinx FPGAs [8][9]. Fig.13 shows how different α values favour the adder tree or adder chain accumulator structure. It is worth noting that this graph has been normalised in term of the input wordlength (W). Thus the real difference in LUTs between the architectures should be multiplied by W, which will favour our architectures even further.

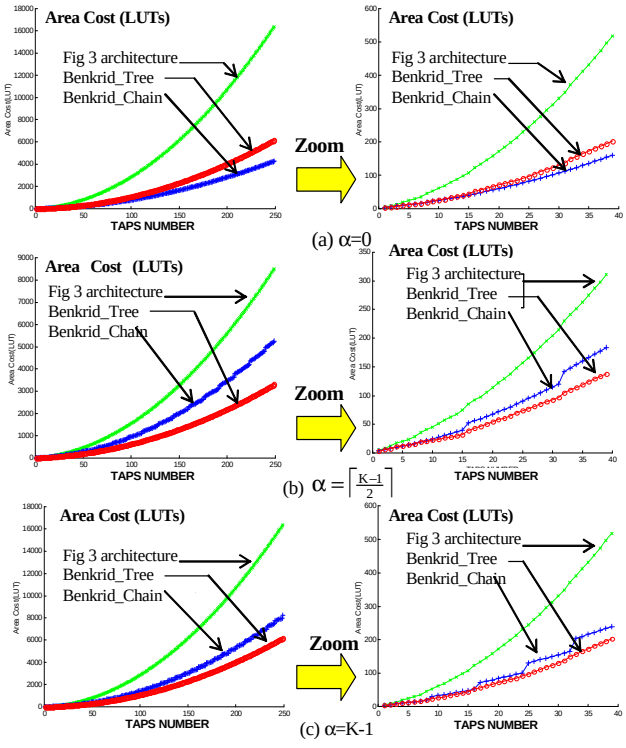

Figure 13. Normalised Area cost comparisons of different FIR architectures. Benkrid_Tree/Benkrid_Chain refers to Fig.5 architecture with adder tree/adder chain accumulator structure.

♦ Note that for K≤9, the SRL layer area cost is reduced to a constant value much smaller than the hard router area cost

2.3 Timing and Area Measurement: Case Study

In this case study, we present the real hardware implementation results of the standard low filter Daubechies-8 wavelet (8 taps) [4] on Xilinx Virtex-E FPGAs using our architectures and the architecture of Fig.3. The filter has been implemented using bit parallel arithmetic with word level pipelining seeking therefore the maximum speed. To favour the architecture of Fig.3, the value of α is set to 3, a value for which the area cost of the *Hard-Router* is minimal (see Fig. 12). The adders and multipliers were implemented using the dedicated carry logic of the FPGA slices. Since the filter coefficients are constant, we used a Canonic Signed Digit (CSD) representation based approach for the multiplier design [12].

The implementation of the SRLs' address generators is straightforward. Each SRL addresses can be subdivided into two sets which correspond to the:

- **Boundary region:** which defines a set of *K-1* values per an SRL corresponding to the *Left_Matrix* and *RightMatrix* rows. These values are stored in the slices distributed RAMs.
- **Non-boundary region**: where the SRL address is constant corresponding to a *RegSkew* value.

A counter line and multiplexer are needed to switch between the two sets of values.

Table 3 and 4 give the achieved results with 9-bit input word length, 8-bit coded coefficients and a 2-bit intermediate and final precision results. The design has been captured in structured VHDL and synthesised using Xilinx ISE software. Timing constraints were applied to determine the maximum achievable frequency. Table 3 shows first the performances achieved from implementing the Daubechies-8 FIR filter with *no boundary processing*. The architectures of Fig.1 are used as well as ours (Fig.5). We can see that when using Fig.1.a architecture, the implementation delivers higher speed but requires more area compared to the inverse form architecture of Fig.1.b. This is expected as the architecture of Fig.1.a uses input samples delays which automatically increase the filter area and its speed as it does not require long routing line to feed the multipliers. Our architecture with *no boundary processing* delivers the same speed as the conventional inverse FIR architecture. However, it consumes more area because of the SRL layer.

	Area (Slices)	Speed (Mhz)
Fig 1.a architecture (Direct)	147	~167
Fig 1.b architecture (Inverse)	113	~159
Fig 5 architecture with adder tree	146	~159
Fig 5 architecture with chain adder	142	~159

Table 3. Performance of a low Daubechies-8 FIR filter implementation using different architectures on Xilinx XCVE50-8 FPGA with **no boundary processing.**

Table 4 shows the performances achieved from implementing the same filter using *symmetric boundary extension*. The architectures of Fig.3 are used as well as ours (Fig.5). The first row of table 3 and 4 shows the effect of the *Hard-Router* on the architecture of Fig.1.a. It implies the use of 78 extra slices leading to ~6 MHz speed penalty. On the other hand, by comparing the performance of our architecture with and without boundary processing, we can see clearly that the dynamic skewing of the SRL layer does introduce a slight area penalty (12 slices) but with no speed penalty.

As stated in section 1, our architecture has been developed mainly to handle the boundary filtering more efficiently than the architecture of Fig.3. Table 4 confirms the superiority of our architecture as almost 70 slices have been saved while maintaining nearly the same throughput.

	Area (slices)	Speed(Mhz)
Fig. 3 architecture	225	~161
Fig. 5 architecture with adder tree	158	~159
Fig. 5 architecture with adder chain	154	~159

Table 4. Performance of a low Daubechies-8 FIR filter implementation using two different architectures on Xilinx XCVE50-8 FPGA **with boundary processing**.

3. Linear Phase FIR Filters

In this section, we suggest novel architectures to handle the problem of boundary processing when using linear phase FIR filters. These are frequently used in digital signal processing since they don't distort the input signal phase [5].

To obtain linear phase filters, symmetry relationship is imposed on the filter coefficients such as (L is the filter length):

$$h(i) = h(L-1-i), \quad 0 \le i \le L-1$$

or

$$h(i) = -h(L-1-i), \quad 0 \le i \le L-1$$

The FIR filter is called *symmetric* (anti-symmetric) if it satisfies the first (second) above condition. Without loss of generality, we focus in the following only on the odd-length symmetric FIR filters i.e. the filter length *L* is odd. The extension of the results to the remaining categories is straightforward.

When implementing linear phase FIR filters in hardware [2], the symmetric coefficients property is often exploited to *halve* the number of multipliers used (see Fig.14).

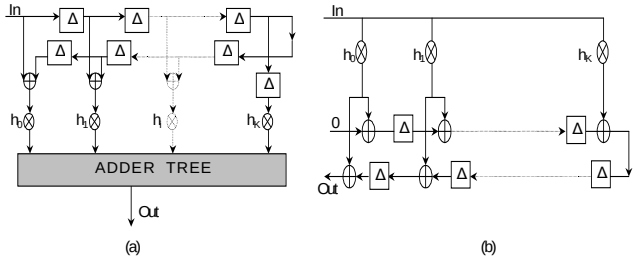

Figure 14. Two conventional (2K+1)-tap symmetric FIR filter architectures

A generic *2K+1* taps symmetric filter output satisfies:

$$out(n) = \sum_{i=-K}^{K} h(i).x(n-i) \qquad (2)$$

where h(i) are the filter coefficients.

As with the general FIR filter, Chakrabarti [6] proposes the use of a hard-router to handle the signal boundary filtering (see Fig.15). To avoid mainly the highly area cost of the hard router implementation (see section 1), we suggest in the next section novel architectures.

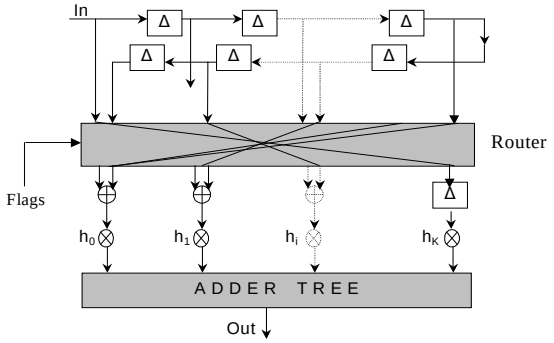

Figure 15. A conventional symmetric FIR filter architecture with a hard-router to handle the signal boundaries

3.1 Novel Architectures of Linear Phase FIR Filters

Equation 2 can be re-written as:

$$out(n) = \sum_{i=-K}^{-1} h(i).x(n-i) + \sum_{i=0}^{K} h(i).x(n-i) \qquad (3)$$

Since h(-i) is equal to h(i) in a symmetric filter, Eq.3 is equivalent to:

$$out(n) = \underbrace{\sum_{i=0}^{K} h(i)x(n-i)}_{SUM_1} + \underbrace{\sum_{i=1}^{K} h(i)x(n+i)}_{SUM_2} \qquad (4)$$

In the next section, we explain how Eq.4 is implemented using our novel architectures. Two main architectures are suggested. The first architecture (*multi-clocked*) is considerably more area-efficient but requires a clock frequency doubler, which could nearly halve the filter throughput. Whereas the second architecture is a single clocked architecture allowing a much higher throughput but with replicated logic. In the following, the multi-clocked architecture (being the more complex) implementation is thoroughly detailed. The results are then easily extended to the single clocked architecture case.

3.1.1 Filtering with No Boundary Processing

- **Multi-Clocked Architecture**

Figure 16 shows two variants of our novel multi-clocked symmetric FIR architecture.

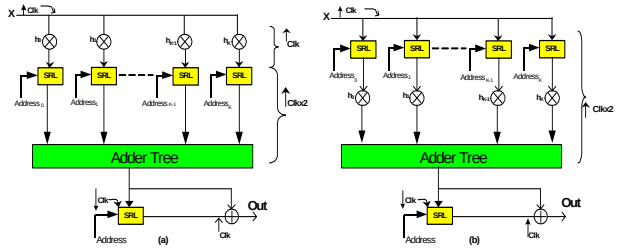

Figure 16. Two variants of our novel multi-clocked symmetric FIR architecture

Except from skewing the input samples instead of the products results, the functionality of Fig.16.b is highly similar to the architecture of Fig.16.a. To avoid redundancy, only the architecture of Fig.16.a is considered. In the architecture of Fig.16.a, the input data samples *x* are first multiplied in parallel with the filter coefficients. Then, the SRLs skew the products *x(n-i).h(i)* to align them properly in time before parallel accumulation (see Fig.17). The adder tree generates alternatively the sums SUM_1 and SUM_2 of Eq.4. A dedicated SRL (referred to as *sink SRL*) is placed at the output of the adder tree to align in time SUM_1 with SUM_2 before addition to generate the filter output.

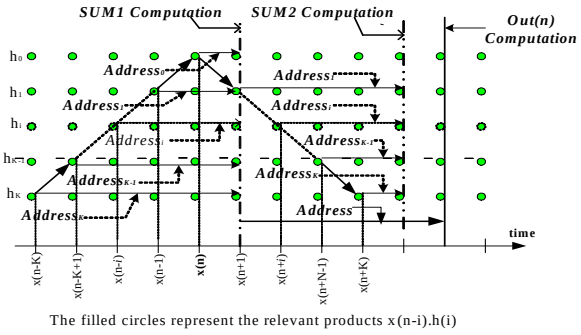

Figure 17. The DDG of a 2K+1 Symmetric FIR filter

However, since the computation of SUM_1 and SUM_2 can be initiated at the same cycle (see Fig.18) and since Xilinx FPGA slices accepts one clock polarity (either rising edge or falling edge), the logic

involving SUM_1 and SUM_2 computation should be clocked (at least) with *double the input clock frequency*. The FPGA on-chip DLL/DCM components can be used to generate this clock frequency [8][9]. As such, the first half of the master clock cycle generates SUM_1 whereas the sum SUM_2 is generated in the second half clock cycle.

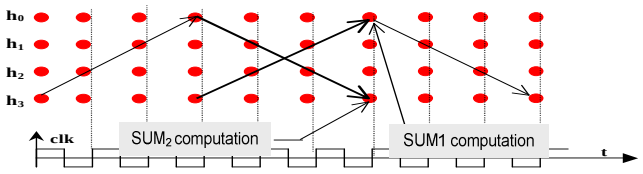

Figure 18. An example showing how SUM_1 and SUM_2 computation coincide in time in a 7-tap symmetric FIR filter

In figure 16, we show the required clocking at each node of the architecture. Upward and downward arrows refer to the rising and the falling edge of the clock respectively. *Clk* is the input clock (Master clock), whereas *Clkx2* runs at double the *Clk* frequency rate. In both architectures and except the sink SRL, the address generators of the SRLs are clocked by *Clkx2* whereas the SRLs inputs are clocked with the master clock *Clk*. The sink SRL is clocked at the falling edge of the master clock to generate the filter outputs at the rising edge of the master clock as illustrated in Fig.19.

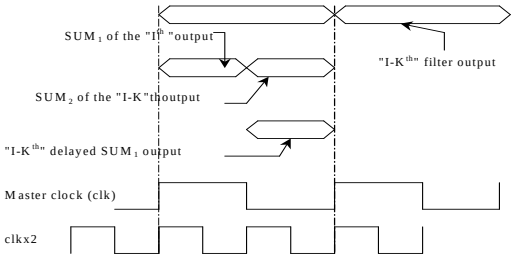

Figure 19. Partial sums synchronisation in our novel architectures of Fig.16

In order to ensure the timely synchronisation shown in Fig.19 at the "sink SRL" input. The delay in cycles of the logic that follows the SRL layer has to be considered to identify at which master clock edge the operands of SUM_1 and SUM_2 should be available. In fact, if this delay is even, the operands of SUM_1 and SUM_2 should be available at the rising and falling clock edge respectively. In contrast, if this delay is odd, the operands of SUM_1 and SUM_2 should be available at the falling and the rising clock edge respectively.

Although the control for our architecture looks more complex than the conventional architectures of Fig.14, it is actually easily parameterised. Indeed, Fig.20 shows the products skew flow graph for our generic 2K+1 taps symmetric FIR filter

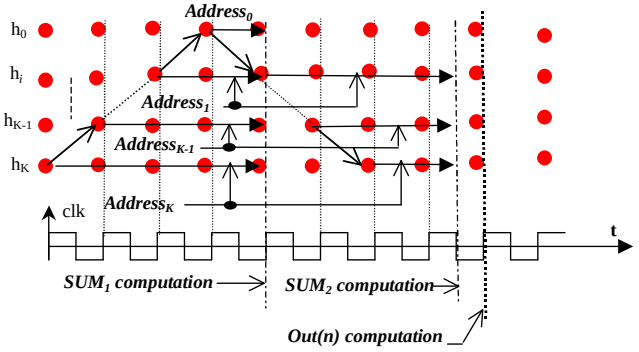

Figure 20. The DDG of the architecture of Fig.16 with the different SRLs skew lengths

Table 5 gives the SRLs skew length deduced from Fig.20. Each SRL skew length, being related to SUM_1 or SUM_2 is equal to the interval of time between the SRL product instant and SUM_1 or SUM_2 computation instant.

	SUM_1	SUM_2
Skew $Length_1$	2+ν	K+(0.5+ν)
Skew $Length_2$	3+ν	K-1+(0.5+ν)
Skew $Length_3$	4+ν	K-2+(0.5+ν)
-------	------	--------
Skew $Length_{K-1}$	K+ν	2+(0.5+ν)
Skew $Length_K$	K+1+ν	1+(0.5+ν)

Table 5. Regular SRLs' skew lengths for the (2K+1) taps symmetric multi-clocked FIR filter of Fig 16 (with **no boundary processing**)

The variable ν in table 5 is equal to:

- 0, if the delay in cycles of the logic that follows the SRLs layer is even.
- 0.5, if the delay in cycles of the logic that follows the SRLs layer is odd.

By following the same geometrical interpretations, we can easily conclude that the skew length of the sink SRL is constant and equal to K.

The SRLs addresses can be easily implemented. In fact according to table 5, each SRL's bit address could either have a constant value (0 or 1), or toggle between 0 and 1 value where a simple toggling flip-flop can be used. It is worth noting that for an odd length symmetric filter, the SRL linked to the tap "h_0" is actually a simple multiplexer, which outputs its input data and a zero value alternatively. Depending on the multipliers output versus input wordlength and as explained in section 2.1, one of Fig.16 architectures consumes less hardware. However, since more logic (multipliers) is clocked at double the master clock frequency in architecture (b) than in architecture (a), the throughput of architecture (b) is expected to be lower than it is with architecture (a).

- **Single-clocked Architecture**

Using the architecture of Fig.16, a clock frequency doubler was necessary to interleave correctly the computation of the partial sums SUM_1 and SUM_2 since those have to be computed at each master clock cycle through a single adder tree structure. This required a careful and tricky synchronisation as shown in Fig.19-20. However, the use of clock doubler will definitely decrease the filter throughput possibly to its half. Owing to this shortcoming, we suggest the architecture of Fig.21 which allocates a dedicated SRL layer and an adder tree for each partial SUM_i. This architecture can be seen as a combination of two parallel filters having the architecture of Fig.5 with shared multipliers logic.

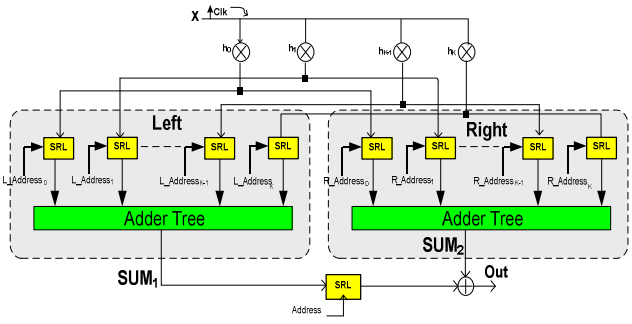

Figure 21. Our novel single-clocked symmetric FIR architecture

Table 6 gives the necessary SRLs skew lengths. Those have been deduced from table 5 by simply removing the unnecessary factor *v* and (*0.5+v*) since each partial sum SUM_i can now be generated at the rising edge of the master clock (see Fig.17). The user can verify that the sink address skew length is equal to K.

	SUM_1	SUM_2
Skew Length $_1$	2	K
Skew Length $_2$	3	K-1
Skew Length $_3$	4	K-2
-------	------	--------
Skew Length $_{K-1}$	K	2
Skew Length $_K$	K+1	1

Table 6. Regular SRLs' skews lengths for a (2K+1) taps symmetric single-clocked FIR filter (with no boundary processing)

Besides being able to implement the convolution operation with no boundary processing, the architectures of Fig.16 and Fig.21 seek primarily to handle the latter problem. The next section details how the above architectures are used to achieve this goal. Throughout, the architecture of Fig.16.a is first considered. The results are then extended to the architecture of Fig.21. In the following, this notation is used:

Pivot: the point of the DDG graph reflection (see x(n) axis in Fig.17).
Left_Line: the DDG's line associated with SUM_1 computation
Right_Line: the DDG's line associated with SUM_2 computation.

3.1.2 Filtering with Symmetric Boundary Extension

- **Multi-Clocked Architecture**

To handle the boundary processing more efficiently than it is suggested in Fig.15, we update slightly our architecture of Fig.16 through a proper dynamic SRLs addressing as explained in the following. Throughput, we assume the number of samples introduced at the left side and the right side input signal be equal to K (i.e. $\alpha=\mu=K$) as it is usually done in practise since any other symmetry axis location leads to samples redundancy in the filter outputs.

Figure 22 shows the 7-tap symmetric FIR filter DDG at the boundary region. The dashed lines in this figure show where the reflection occurs, whereas the shaded rectangle shows the boundary region between two consecutive input sequences: $Sequence_I$ and $Sequence_{(I-1)}$. We denote by negative values (-1, -2, -3…) the instants which precede the instant 0 of $Sequence_I$.

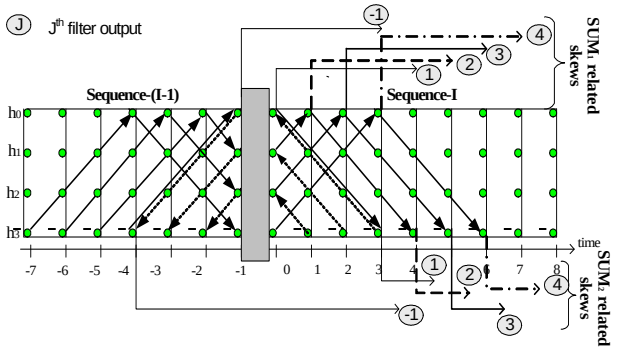

Figure 22. DDG of a 7-tap symmetric FIR filter using boundary symmetric extension (K=3)

From Fig.22, and as a general rule we can see that the *regular* DDG (as depicted in Fig.17) of a generic (2K+1)-tap symmetric FIR filter ends at the pivot of instant "–(*K+1*)" (= -4 in Fig.22), and starts from the pivot of instant *K* (= 3 in Fig.22). Between these two values, the DDG is irregular. Because of the DDG irregularity at the boundary region, the SRLs skew lengths given in table 5 should be updated. The updated SRLs' skew lengths could be deduced using the following approach.

- **An Approach to determine the SRLs' skew lengths when using Symmetric Signal Extension**

Figure 22 shows clearly that when filtering at the left side signal boundary, only the *Left_Line* of the filter's DDG is altered, whereas it is the *Right_Line*, which is altered when filtering at the right side signal boundary. To get the necessary skew lengths when extending the signal by symmetry, we consider separately the *Left_line* and the *Right_line* of the filter's DDG. This allows us to use the skew matrices and skew lists given in [10].

a) Left Side Signal Boundary Extension

By comparing Fig.22 and Fig.7 (which is valid for both accumulator structures), we can conclude that the *Left_Line* DDG of a (2K+1)-Tap symmetric filter is similar to the DDG of a (K+1)-Tap FIR filter when the input signal is extended by *K* samples at its left side (the *Left_Line* DDG is regular in $sequence_{(I-1)}$). The *Left_Matrix* and *RegSkew* expressions given in [10] can then be used. However depending on the delay parity of the logic that follows the SRL layer and as explained in section 3.1.1, the pivot of the symmetric filter should feed the adder tree *1* cc or *1.5* cc after being computed, instead of *1* cc as assumed with the general (K+1)-Tap filter [10]. It is equal to *1* cc for even logic delay parity. Therefore, the pivot as well as the *Left_Line* skew length given in [10] should be updated by a *Left_Upd* value such that:

$$\text{Left\_Upd} = \begin{cases} 0.5 & \text{, If the logic delay parity after the SRL is odd} \\ 0 & \text{, If the logic delay parity after the SRL is even} \end{cases}$$

Consequently, the reader can verify easily, that the SRLs' skew lengths at the left side boundary regions are given by a *K_Matrix* (since α = K) of size equal to [K, K+1], such that:

$$\text{K\_Matrix} = \begin{bmatrix} 1 & 2 & 3 & . & ... & K-1 & K & \underline{K+1} \\ 3 & 4 & 5 & . & ... & K+1 & \underline{K+2} & K+1 \\ 5 & 6 & 7 & . & ... & \underline{K+3} & K+2 & K+1 \\ . & . & . & . & ... & . & . & . \\ . & . & . & . & ... & . & . & . \\ 2K-5 & 2K-4 & 2K-3 & 2K-2 & ... & K+3 & K+2 & K+1 \\ 2K-3 & 2K-2 & \underline{2K-1} & 2K-2 & ... & K+3 & K+2 & K+1 \\ 2K-1 & \underline{2K} & 2K-1 & 2K-2 & ... & K+3 & K+2 & K+1 \end{bmatrix} + \text{Left\_Upd}$$

On the other hand, the *Left_Line* SRLs' skew lengths at the non-boundary regions are given by the *LeftRegSkew* list such that:

LeftRegSkew = [2K+1, 2K, 2K-1,…,K+3, K+2, K+1] + Left_Upd

This list is deduced by using the *RegSkew* list expression given [10], where ξ=K.

Unlike the *Left_Line*, the DDGs *Right_Line* at the left side signal boundary are all regular. Thus the values given in table 5 will be used:

RightRegSkew = [K+(0.5+v), K-1+(0.5+v),…, 1+(0.5+v)]

b) Right Side Signal Boundary

When filtering through the non-boundary regions, the DDG *Left_Line* and the *Right_Line* products are skewed respectively by the *LeftRegSkew* and *RightRegSkew* lists given above. However unlike the *Left_Line*, the *Right_Line* is no longer a straight line when filtering through the right side signal edge. Therefore, the *LeftRegSkew* list values are still valid at the right boundary regions whereas the *Right_Line* skew list should be updated.

By comparing Fig.7 and Fig.22 at the right signal boundary, we can conclude that the *Right_Line* for a (2K+1) symmetric FIR filter is equivalent to the DDG of a (K+1)-tap filter where the left extension value α is equal to 0. Therefore, the *Right_Matrix* expression given in [10] can be used where ξ=0.

Similarly, depending on the delay parity of the logic that follows the SRL and according to section 3.1.1, the product $P_{(2K+1)}$ should feed the adder tree 1.5 cc or 2 cc after it has been computed, instead of +1 cc as assumed with the general K-Tap filter [10]. It is equal to 1.5 cc for even logic delay parity. Therefore, the $P_{(2K+1)}$ as well as the *Right_Line* skew lengths given in [10] related to a K-Tap general FIR filter should be updated by a *Right_Upd* value such that:

$$\text{Right\_Upd} = \begin{cases} 1 & \text{, If the logic delay parity after the SRL is odd} \\ 0.5 & \text{, If the logic delay parity after the SRL is even} \end{cases}$$

Consequently, the reader can verify easily, while taking into account the opposite direction of the arrows in Fig.7 and Fig.22, that the SRLs' skew lengths at the left side boundary regions are given by a *K_Matrix* (since $\alpha = K$) of size equal to [K, K], such that[3]:

$$\text{K\_Matrix} = \begin{bmatrix} K & K-1 & K-2 & \dots & 3 & 2 & 3 \\ K & K-1 & K-2 & \dots & 3 & 4 & 5 \\ K & K-1 & K-2 & \dots & 5 & 6 & 7 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ K & K-1 & K & \dots & 2K-5 & 2K-4 & 2K-3 \\ K & K+1 & K+2 & \dots & 2K-3 & 2K-2 & 2K-1 \\ K+2 & K+3 & K+4 & \dots & 2K-1 & 2K & 2K+1 \end{bmatrix} + \text{Right\_Upd}$$

The reader can verify easily that the sink SRL skew length becomes equal to 0 (see Fig.22). This SRL can then be replaced with a Flip-Flop clocked at the falling edge.

- **Single-Clocked Architecture**

The approach explained in the previous section can be easily extended to the architecture of Fig.21. The required SRLs skew lengths are deduced by simply removing the factors *Left_Upd*, *Right_Upd*, and $(0.5+v)$ from the expression of *K_Matrices* , *LeftRegSkew* and *RightRegSkew* given above. Those updates are no more necessary as explained previously since SUM_1 and SUM_2 operands and computation are generated now at the rising edge of the master clock. In fact, the architecture of Fig.21 is a parallel combination of the architecture of Fig.5. The latter's required skew lengths have to be applied separately on SUM_1 and SUM_2 SRLs layer. On the other hand, the reader can verify easily that the sink SRL skew length is equal to 0 (see Fig.22). This SRL can then be replaced with a Flip-Flop clocked at the rising edge.

3.2 Area and Speed Comparisons

Table 7 gives a comparison of the resources used by our architectures (Fig.16, 21) and the conventional architecture of Fig.15 with a W-bit data input. The cost of the Hard Router in table 7 has been deduced using its formula given in section 1 with $\alpha = \mu = K$.

	Fig 15 architecture	Fig 16 architecture	Fig 21 architecture
Number of Multipliers	K+1	K+1	K+1
Number of Word Delays	2K+1	1 (sink SRL)	1 (sink SRL)
Number of Adders	K+ adderTree(K+1)	1+adderTree(K+1)	1+2*adderTree(K+1)
Router Hardware (LUTs)	$2W\sum_{i=2}^{i=K+1}\lceil i/2 \rceil$	SRL Layer	2*SRL Layer

Table 7. Logic resources consumed by different (2K+1) Taps symmetric FIR architectures with Boundary Processing

[3] The leftmost column in *Right_Matrix* expression has been removed since it is related to the pivot which has been already considered in the *Left_Line* skew length expression

If we exclude the area cost of the SRLs' address generators and the multipliers (which is the same in all architectures), Fig.23 depicts the evolution of the normalised area cost of all architectures in relation to the filter length. Note that the graphs have been normalised in term of the input wordlength (W). Thus the real difference in slices between the three architectures should be multiplied by W, which favours our architectures further. From this figure, we can see clearly that our novel architectures are much more compact than the conventional architecture of Fig.15. The architecture of Fig.16 is the most area efficient. However, since it uses a clock frequency doubler, the speed of the architecture will be very much limited by the speed of the logic clocked at Clkx2. We expect Fig.16 architecture to be significantly slower than the conventional architecture of Fig.15 and Fig.21. Table 8 gives the results achieved from the implementation of the standard low Biorthogonal 9&7 wavelet filter (9 taps) [4], using bit parallel arithmetic. The FIR was implemented using the same constraints stated in the case study of section 2.3.

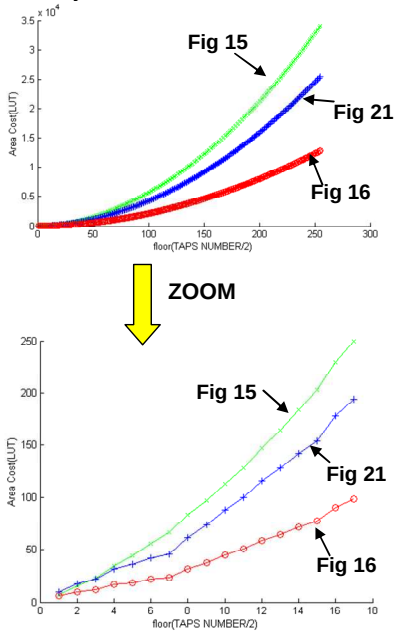

Figure 23. Area cost comparisons of different symmetric FIR architectures with normalised wordlength.

	Area (Slices)	Speed (MHz)
Fig.15 architecture	257	~155
Fig.16.a architecture	160	~90
Fig.21 architecture	230	~155

Table 8. Performance of the low Biorthogonal 9&7 FIR filter implementations on Xilinx XCVE50-8 FPGA with symmetric boundary extension

From table 7, we conclude that the hard-router solution introduces a considerable area penalty, whereas the speed performance of the architecture of Fig.16 is significantly lower, although still high enough to ensure a real time implementation. Our architecture of Fig.21 gives an in-between speed/area performance. It is worth noting that if we implement the same filter under the same constraints, but with sub-sampled output (by a factor of 2), the speed penalty in Fig.16 disappears as shown in table 9.

	Area (Slices)	Speed (MHz)
Fig.15 architecture	222	~162
Fig.16.a architecture	149	~161

Table 9. Performance of the low Biorthogonal 9&7 FIR filter implementation with decimated output on Xilinx XCVE50-8 FPGA with symmetric boundary extension

Note that the change in each filter area between table 8 and 9 is due to the decimation by a factor of 2, which means that fewer inputs are multiplexed or skewed than in the non-decimated version. In fact, to find the SRLs' skew lengths (either for a general or a symmetric FIR filter) when the output is decimated, the expression of the skew lists and Matrices given above can be used. With bi-phase FIR filters [13], the odd (even) coefficients are multiplied with the odd (even) input samples only. Therefore the above matrices should be sampled, where each odd (even) order SRL is associated with an odd (even) skew matrix row. Moreover, since the multipliers are enabled only once every over two cycles, the SRLs' skew length given in these matrices should be divided by 2. As a consequence, there is no need to use a clock frequency doubler when using the architecture of Fig.16 to implement a symmetric filter with decimated outputs. As such the architecture of Fig.16 delivers a similar speed range to the conventional filters but with considerable area saving. This architecture is hence more suitable in multirate architectures such as those found in Discrete Wavelet Transforms [4].

4 Conclusion

This paper addressed an important signal processing technique which is often lightly considered in hardware implementation, namely: signal boundary extension in finite length signals processing. The technique of symmetric extension has been particularly investigated as it is the most widely used technique in practice. To this end, the design and implementation of four novel bit-parallel FIR filters architectures have been detailed. The architectures cleverly harness the SRL16 components of Xilinx FPGAs to achieve significant area savings compared to conventional FIR architectures. Detailed scheduling algorithms were presented making the architectures fully scalable and parameterisable.

For a general FIR filter, we presented two architectures (see Fig 5 with two varying accumulator types) that lead to a very compact FPGA configuration compared to conventional architectures while maintaining the same throughput. The special case of symmetric FIR filters was further considered. In it, two novel architectures (see Fig 16 and Fig 21) were also devised. The first architecture (Fig 16) delivered considerable area savings albeit at the expense of a clock frequency doubler and lower throughput. Nonetheless the overall processing speed was still high enough to achieve real time performance e.g. for wavelet transformation of HDTV. Furthermore, we noted that this architecture can match the speed of the conventional architecture if the filter output is going to be decimated, as it is the case in multirate applications (e.g. wavelets) [4]. The second symmetric filter architecture in Fig 21 replicates some logic to avoid the need for a clock frequency doubler. This allowed us to match the conventional architecture throughput, albeit with less area savings compared to our first symmetric filter architecture.

Another advantage of our architectures compared to conventional ones resides in the fact that they can readily harness sub-expression sharing (i.e. sharing multiplier blocks between different terms of the FIR filter) which can lead to even more area savings.

Finally, it is worth noting that the role of SRL16 presented in this paper can be played by distributed dual port RAM configurations of the FPGAs' LUTs, something which makes the benefits of our architectures possible on a wider range of FPGAs chips including Xilinx XC4k series as well as Lattice Semiconductor's FPGAs. The only drawback is that a 16 bit dual RAM port configuration consumes two 4->1 LUTs instead of one LUT for an SRL16 configuration. Nonetheless, despite this cost, our architectures still require less LUTs overall compared to conventional FIR architectures. The benefits of our architectures are hence applicable to a wide range of FPGAs. The fact that we have presented implementation results for only Xilinx Virtex-E FPGAs does not narrow the scope of our conclusions. In fact, further optimisations could have been made on our architectures including harnessing sub-expression sharing and using embedded arithmetic blocks in the latest FPGAs (e.g. hardwired multipliers and Xtreme DSP blocks on Xilinx Virtex-4 FPGAs) which would have favoured our architectures even further. Nonetheless, we wanted to focus this paper on the novel algorithm devised, which concerns the optimisation of the SRL layer, as opposed to the hard router, in order to skew the FIR input data appropriately before subsequent multiplication. Any subsequent multiplier/adder optimisation can be readily applied to our architectures.

References

[1] Proakis. J.G., Manolakis .D.G, *"Introduction to Digital Signal Processing,"* McMillan Publishing, USA, 1989.

[2] Peter Pirsch, *"Architectures for Digital Signal Processing,"* John Wiley & Sons, 1999.

[3] Oppenheim A V, Schafer R W, *"Discrete-time signal processing,"* Englewood Cliffs, N.J.: Prentice Hall, 1989.

[4] Vetterli M, Kovacevic M, *"Wavelets and Subband Coding,"* Prentice Hall, New Jersey, USA, 1995

[5] Smith M.J.T, Eddins S *"Analysis/Synthesis techniques for subband image coding,"* IEEE Trans On Acoustics, Speech and Signal Processing, 1446-1456, August 1990

[6] Chakrabarti C, *"A DWT based encoder architecture for symmetrically extended images,"* Proceedings of the International Symposium on Circuits and Systems, 1999.

[7] Benkrid A, Benkrid K, Crookes D, *"Design and Implementation of a Novel Architecture for Symmetric FIR filters with Boundary Handling on Xilinx Virtex FPGAs,"* IEEE Conference on Field-Programmable Technology, FPT'2002, December 16, 2002

[8] http://www.xilinx.com/partinfo/ds022.pdf

[9] http://direct.xilinx.com/bvdocs/publications/ds100.pdf

[10] Benkrid A, Benkrid K, Crookes D, *"Design and Implementation of a Novel FIR Filter Architecture with Boundary Handling on Xilinx VIRTEX FPGAs"* 13th International Conference on Field Programmable Logic and Applications (FPL 2003), 1st - 3rd Sept. 2003, in Lisbon, Portugal

[11] Benkrid A, Benkrid K, Crookes D, *"A Novel FIR Filter Architecture for Efficient Signal Boundary Handling on Xilinx VIRTEX FPGAs"*, The 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines FCCM'03, Napa, California 8 - 11 April 2003

[12] Keshab K. Parhi, *"VLSI Digital Signal Processing Systems: Design and Implementation,"* John Wiley & Sons, 1999

[13] Vaidyanathan P P, *"Multirate Systems and Filterbanks,"* Prentice-Hall Inc., USA, 1993

[14] A.Benkrid, K.Benkrid, 'Handling finite length signals borders in two-channel filter banks for perfect reconstruction', Journal of Signal Processing, Elsevier, Vol. 86. pp 375-387, February 2006