An Alternative Carry-Save Arithmetic for New Generation Field Programmable Logic Arrays

Uğur ÇİNİ

ugurcini@yahoo.com

Trakya University, Dept. of Electrical & Electronics Engg.

Muhendislik Fakultesi Edirne Türkiye

Mustafa AKTAN

aktanmus@gmail.com

Terahz Elektronik Ltd.

Ankara Üniversitesi Teknoloji Geliştirme Bölgesi

D Blok Zemin Kat No:6 Gölbaşı Ankara Türkiye

Avni MORGÜL

amorgul@fsm.edu.tr

Fatih Sultan Mehmet University, Dept. of Biomedical Engg.

Haliç Yerleşkesi, Sütlüce Mah. Karaağaç Cad. No:12 Beyoğlu İstanbul Türkiye

An Alternative Carry-Save Arithmetic for New Generation Field Programmable Logic Arrays

Abstract

In this work, a double carry-save addition operation is proposed which is efficiently synthesized for 6-input LUT based FPGAs. The proposed arithmetic operation is based on redundant number representation and provides carry propagation free addition. Using the proposed arithmetic operation, a very compact and fast multiply and accumulate unit is designed. The proposed design provides the fastest multiply-add operation for 6-input LUT based FPGA systems according to our knowledge. An Finite Impulse Response (FIR) filter implementation is given as an example to show the performance of the proposed structure. The proposed impementation provides a dramatic performance increase, which is at least two times faster than conventional binary multiply-add implementations.

1. Introduction

Generic field programmable gate array (FPGA) devices are based on 4-input look-up table based logic elements. High performance FPGA devices offer 6-input look-up table (LUT) based logic elements by which more complex functions can be realized with higher performance [1, 2]. In this work, an extra redundant arithmetic scheme is proposed to further exploit 6-input LUT devices.

Redundant architectures are based on signed-digit systems and carry-save arithmetic both of which provide carry-free addition schemes [3-5]. In carry-save arithmetic, each digit of a number is represented by two bits, namely carry (c) and sum (s) [6], whereas in conventional binary (e.g. 2's complement) representation, each digit is represented by a single bit. The redundancy in number representation provides carry-free arithmetic implementations. In this work, redundancy is increased by using double carry-save representation, where each digit of an arbitrary number is represented by three bits.

Increased redundancy enables addition operation to be handled within a single LUT delay on a 6-input LUT based architecture, which is not possible with a conventional redundant carry-save addition scheme. Addition of two double carry-save mode numbers can be done using a parallel array of (6, 3) counter circuits. Using the proposed arithmetic, addition of two redundant numbers takes a single LUT delay in 6-input LUT structures which is the core of the paper.

(6,3) counters are the best suited multi-operand addition schemes for 6-input LUT FPGAs [7]. In multiplication, (6,3) counters can be used to reduce six partial products to three. Together with provided double carry-save arithmetic, both addition and multiplication operations can be handled using only (6,3) counters, which provides a very regular structure. A multiply-accumulate operation based on the proposed system takes 2 LUT delays if the coefficients of multiplications are 12-bit wide. 12-bit also refers (6, 3) counters since canonic signed-digit (CSD) [8-11] recording of 12-bit coefficients corresponds to at most 6 non-zero partial products which is also suitable for (6, 3) counters. Higher bit widths are also possible with the increase of the critical path.

In this paper, a multiply-accumulate unit is designed and used to realize a constant coefficient FIR filter. In the proposed system, the multiplication of each constant coefficient is achieved through (6,3) counter array with redundant outputs. Moreover, backward sign-extension is implemented for the removal of extra sign-bit in the system which will be explained in the following sections. After the multiplication phase, the redundant addition operation, i.e. accumualtion phase, is also implemented by a single stage (6,3) counter array yielding a regular multiply-accumulate structure in FIR filters [12]. As a result, a multiply-accumulate operation is accomplished in two stages.

Redundant architectures implemented on FPGAs are not very common. Recent publications related to FPGA arithmetic can be found in [13-17]. However, only [17] focuses on 6-input LUT structures. Our example filter implementation is compared with the filter implementation of [17] in Section 3, where performance of the proposed design is shown to be much higher.

In the next section, the proposed redundant arithmetic scheme will be explained. Section 3 deals with implementation methodology and comparison with regular implementations. The paper is finalized with a conclusion section.

2. (6,3) counters and double carry-save arithmetic

Counter circuits are generally used to reduce the number of operands in a multi-operand addition operation such as reduction of partial products in multiplication. A counter circuit basically counts non-zero input operands and converts the result in radix-2 format [18, 19]. For example a (6,3) counter produces the results $(000)_2$ and $(110)_2$ for the 6-input binary inputs 000000 and 111111, respectively.

For a (6,3) counter the result is represented by three bits as the name (6,3) implies such that six inputs are reduced to three. The representation of a (6,3) counter is depicted in Figure 1(a). The multiple operand addition scheme is shown in Figure 1(b), where six 8-bit binary numbers X_0 ... X_5 are reduced to S_0, S_1, S_2. The conventional binary result is obtained by adding up the three outputs S_0, S_1, and S_2 by shifting each output relatively to their bit weights as seen in Figure 1(b). (6,3) counters are especially useful for 6-input LUT based FPGAs, where the operation can be handled in a single LUT stage. A (6,3) counter for a single vertical slice can be implemented by three 6-input LUTs, since the function has three outputs.

In carry-save arithmetic, each number is represented in combination of two numbers $Z = (S, C)$ [6]. In this work, double carry-save architecture is employed such that:

$$Z^a = S_0 \quad (1.a)$$

$$Z^b = S_1 << 1 \quad (1.b)$$

$$Z^c = S_2 << 2 \quad (1.c)$$

$$Z = (Z^a, Z^b, Z^c) \quad (1.d)$$

The operator "<<*k*" represents *k*-bit left shift. As shown in Figure 1, each output is relatively shifted by one bit. The redundant number Z represents the addition of six binary numbers. The redundant number Z is always a composition of three binary numbers (Z^a, Z^b, Z^c) and whenever normal binary (i.e. conventional 2's complement binary representation) is required three operand addition is accomplished such that $S = Z^a + Z^b + Z^c$.

To summarize, in double carry-save representation, each number is a composition of three radix-2 numbers. And the digit set of each redundant number Z_i is in the set of {0, 1, 2, 3}, where each digit can take values between 0 ($z^a_i + z^b_i + z^c_i = 0$) to 3 ($z^a_i + z^b_i + z^c_i = 3$). Here, each digit of Z_i is represented by three bits, i.e. $z_i = (z^a_i , z^b_i , z^c_i)$. In this structure, each summation digit is also represented by three bits.

The redundant addition of two double carry-save numbers can be accomplished by a single (6,3) counter array. Proposed redundant addition and subtraction operations of X and Y with the result of Z are depicted in Figure 2 (a) and 2 (b), respectively. Here, each digit of X, Y and Z are represented as x_i, y_i, z_i, respectively.

Double carry-save addition operation can be written as:

$$Z = X \underline{\oplus} Y \tag{2}$$

Double carry-save subtraction of $X - Y$ can be written as:

$$Z = X \underline{\oplus} \overline{Y} + 3 \tag{3}$$

Here, $\underline{\oplus}$ represents (6,3) reduction, $\overline{Y}$ represents inverting each of the bits of the redundant number *Y*, and, at the end, constant 3 is added up to the number, since Y is a composition of three normal binary numbers. As an analogy to 2's complement, where $-A$ is represented as $\overline{A}+1$, here, constant 3 is added up since each digit is a composition of three bits in proposed redundant representation. Since each number is represented by a set of three numbers, the double carry representation is defined as $-Y = \overline{Y} + 3$. Figure 2(b) shows the redundant subtraction.

To summarize, the output of (6,3) counter with six normal binary numbers at the input results in a single double carry-save format output, as shown in Figure 1(b). Two double carry-save format numbers can be added up using a single stage (6,3) counter circuit, as shown in Fig 2(a).

It should also be noted that, addition of three normal binary numbers (i.e. conventional 2's complement binary number) with the output of double carry-save output is so trivial that, all three is written down together, without any operation at all. As an example, if three numbers X_0, X_1 and X_2 is to be added, $Z^a = X_0$, $Z^b = X_1$, $Z^c = X_2$, then the output is

$Z = (Z^a, Z^b, Z^c)$. If two numbers is to be added up i.e. X_0 and X_1, then $Z^a = X_0$, $Z^b = X_1$, $Z^c = 0$ then the result is again $Z = (Z^a, Z^b, Z^c)$, same as in (1.d).

For comparison, the proposed double carry-save, conventional redundant carry-save, and normal binary (i.e. conventional 2's complement binary) addition are implemented using 6-input LUT devices. Redundant carry-save representation is also named as CS2 [5, 20] representation in the literature since each digit is represented by two bits. The redundant carry-save addition operation is represented in Figure 3. The (3, 2) blocks represent counters for reducing three operands to two which have equivalent function of full adders. Two stages of (3, 2) counters implement the function of (4, 2) reduction. Here, numbers Y and X are added giving the result Z, where each of X, Y and Z is represented by a combination of (X^S, X^C), (Y^S, Y^C), and (Z^S, Z^C), respectively. Here, addition operation is actually a (4, 2) reduction where four operands are reduced to two.

Comparison of the three different addition schemes is depicted in Figure 4. As can be seen from the table, carry propagate addition delay increases with adder size. Normally, the delay is directly proportional to the adder size in the carry-propagate addition. There is a slight nonlinearity after the synthesis. The conventional carry-save operation and proposed double carry-save representation delays are independent of the digit lengths. However, the conventional redundant carry-save operation requires 2 LUT delay whereas the proposed double carry-save operation requires only a single LUT delay. The area requirement of normal carry-save and double carry-save is the same. Both implementations have approximately twice the area of a carry-propagate adder. It should also be noted here that, for short adder sizes, carry-propagate addition operation

is quite fast because of the fast carry logic blocks of the FPGA fabric. However, double carry-save method is still much faster than both of the implementations for any adder size. The timing analyses of the adders are measured using TimeQuest Timing Analyzer of Altera™ QuartusII platform for StratixII FPGAs.

3. Implementation

In this section, a useful example for the double carry-save arithmetic will be given. As mentioned in the previous section, the proposed arithmetic operation is especially useful for recursive multiply-accumulate operations. One of the mostly used applications for this type of implementation is digital filters. These filters require many cascades of multiply-add operations and the proposed double carry-save structure is extremely useful to reach high performance targets. For this reason, a fixed coefficient based FIR filter will be designed based on fixed coefficient multiply and add unit.

3.1 FIR Filter Generation

FIR filters are used for shaping the input signal with the desired frequency response. Discrete time domain representation for an *N*-tap FIR filter is given as:

$$y[n] = \sum_{k=0}^{N-1} h[k]x[n-k] \qquad (4)$$

Here, *x*, *y*, *h* are the input, output and the transfer function of the filter, respectively. The frequency response $H(\omega)$ of a linear-phase FIR filter with impulse response $h[n]$ and length *N* is:

$$H(\omega) = \sum_{n=0}^{N-1} h[n] e^{-j\omega n} \tag{5}$$

Here, the requirement for a linear filter is that, the filter coefficients must be symmetric or anti-symmetric, as a result, the filter coefficients can also be written in terms of the amplitude $A(\omega)$ and phase terms as:

$$H(\omega) = A(\omega) e^{-j\omega(M-1)} \tag{6}$$

where for symmetric filter M is approximately half the length of the filter tap count, here, M is given as

$$M = \left\lfloor \frac{N}{2} \right\rfloor + 1 \tag{7}$$

The amplitude $A(\omega)$ is a real function of frequency given by

$$A(\omega) = \sum_{m=0}^{M-1} h[m] T_m(\omega) \tag{8}$$

where $T_m(\omega)$ is a trigonometric function determined by the length and type of symmetry of the filter. The values of $T_m(\omega)$ for the four possible types of linear phase FIR filters are given in Table 1 [21].

The transposed FIR filter minimizes the critical path of the FIR operation to a single multiply-add operation as seen in Figure 5(a). The linear phase implementation of an odd length filter realized by symmetric or anti-symmetric coefficients can be realized as in Figure 5(b).

3.2 Realization of multiply-add operation

Here, the proposed arithmetic will be implemented by generating a multiply-add unit. If the filter specifications are defined, corresponding filter coefficients and number of taps for the filter can be generated using any FIR filter generation algorithm [21]. Since the filter characteristics are predefined, fixed multiplication coefficients are used to generate fixed multiplication blocks.

After the filter coefficients are calculated, the multiply and accumulate operations can be implemented according to the redundant arithmetic as defined in Section 2. The fixed coefficient multiplication for the generated filter coefficients are realized by properly tiling the variables. A multiplication scheme for a 12-bit fixed coefficient and 12-bit data input word-length is depicted in Figure 6(a). In the example, the coefficient is given as ($0\bar{1}0101\bar{1}0101\bar{1}$). Here, we rename each non-zero digit in the coefficient as c_i, where $c_0 = -1,\ c_1 = 1,\ c_2 = -1,\ c_3 = 1,\ c_4 = 1$, and $c_5 = -1$ in the example (enumeration $c_0 \ldots\ c_5$ is based on the non-zero digit from the right to left of the example coefficient). At most six non-zero digits are allowed in any of the coefficients, which is always the general case for 12-bit canonic signed-digit (CSD) coded coefficients. A 12-bit CSD coded number has at most 6 non-zero digits, since number of non-zero digits in CSD format is at most the half of the word length [12]. In this representation, higher input data word-length is allowed without increasing the critical path delay. The s_i in each line shown in Figure 6(a) also represents the sign bit of any non-zero bit for the corresponding coefficient.

The diagram in Figure 6(a) is just an example and is not the generalized case. In the generalized case, the shift operations of the coefficients can be arbitrary amounts, depending on the positions of the non-zero digits. Arithmetic right shift operation is applied to the partial product by (n - i) times, where n is the coefficient word-length and i is the position of the non-zero digit in the coefficient. Here, e_i is the most significant bit (MSB) of input variable X, if the corresponding coefficient digit is 1. On the other hand, e_i is the complement of MSB of input variable X if the corresponding coefficient digit is -1. In other words, $e_i = x_{MSB} \oplus s_i$ [19, 22]. Here, s_i is the sign of the incident non-zero coefficient digit. As can be seen from Figure 6(a), the sign bit of the most significant digit of the coefficient (s_5) increases the number of partial products by one, i.e. seven products exist in worst case and the partial products cannot be fed into a single stage (6,3) counter. The problem is solved using backward sign extension. The (6,3) reduction is done after this operation. Figure 6(b) shows the application of backward sign extension.

The multiplication operation for fixed-coefficient with input data is realized by reduction of the partial products generated as shown in Figure 6(b). For the redundant double carry-save representation, each number is a composition of three normal binary numbers. As a result, the six partial products need to be reduced to three, to make the number compatible with double carry representation. For six partial products generated, multiplication with redundant outputs is shown in Figure 7(a). In the figure it can be seen that the multiplication phase consists of a single stage (n+1) digit (6,3) reduction scheme. Here, n is equal to the the length of the coefficient and length of the variable, i.e. n = (*coef_wordlength* + *data_wordlength*), which is 24 for the given example. The +

1 in the (n+1) definition is for the residue reduction as shown in Figure 6(b) and Figure 7(a). Still after the reduction, there are residue bits existing together with the multiplier result. The end result is obtained at the accumulation step. The accumulation step is also an (n+1) digit (6,3) reduction scheme. The other input for the accumulation step comes from the previous tap of the designed filter. The whole multiply-accumulate operation is accomplished in two stages of (n+1) digit (6,3) counter arrays. At the multiply-add operation output, the result appears in double carry-save format, that is composed of three binary numbers.

When there exists three or less non-zero digits in the coefficients, the multiply phase gets even simpler, as (6, 3) reduction for the multiply phase is removed. The multiplication phase for the multiply-add operation only consists of arithmetic shifts and sign bit padding operations which is shown in Figure 7(b). In this case, the hardware cost for the multiply-accumulate phase is halved, which greatly reduces the hardware cost for the construction of the related filter tap. As a result, the reduction of non-zero digits in coefficients plays an important role in the filter design procedure.

3.3 Filter Realization

For the performance measurement, a low-pass filter is implemented using the proposed multiply accumulate unit.. The frequency response characteristics of the filter are given in Table 2. The pass-band and stop-band frequencies are normalized to the sampling frequency in the example. The magnitude and phase response plots of the filter are shown in Figure 8.

Given the specifications in Table 2, the filter can be synthesized using any of mathematical tools, such as Matlab™ or other kind of filter design software. However, if the multiplication is to be done in a single stage, the coefficients must be digitized to 12-bit word-length. The word-length of the example filter is optimized using the algorithm in [10]. The example filter could be synthesized with 25 taps and 12 digit coefficients. The synthesized coefficients for the filter are given in Table 3. In the table, the coefficients that have three or less than three non-zero digits are marked with asterisks. The coefficients with three or less non-zero digits save space in synthesis and increase performance.

The last stage of the redundant multiply-add operation is pipelined with registers, and between the pipeline, a three operand addition is handled to convert double carry-save representation to normal two's complement binary output in the end. The representation of the multiply-accumulate operation is shown in Figure 9. The representation only shows the last multiply-add operation of the Figure 5(b). As the figure reveals, the redundant output of the system is converted to normal binary by insertion of a three operand adder circuit after the last tap of the multiply-add operation.

For the comparison, the filter is designed in three ways. The first implementation is generated using fixed coefficient multipliers and carry-propagate adders in normal binary mode using totally 6-input FPGA fabric. In the second implementation, firm multipliers are used, which are generally recommended for DSP operations in FPGA fabrics. For the accumulate phase, carry-propagate adders are used in the second

implementation. In the third implementation, proposed double carry-save implementation is handled for multiply and accumulate units. In each of the implementations, the architecture in Figure 5(b) is used.

Fixed coefficient multipliers are designed using the Altera™ QuartusII Megafunctions Wizard. Each of the implementations is synthesized using Altera™ Quartus II software. The comparison of hardware cost and maximum operating speed is given in Table 4. A fast redundant signed-digit based FIR filter [17] is also added for comparison. The implementation in [17] has also redundant representation with carry-free arithmetic utilizing 8-bit coefficients with 8-bit data word-length and 16 taps, where the recorded speed is 293 MHz in a 6-input LUT based FPGA. Our implementation performance is far beyond the referenced implementation, besides, our implementation has larger coefficient and data widths (both of which are 12 bits wide) with 25 taps. The hardware resources used is not reported in the implementation [17]. The speed performance comparison of the filter implementations are shown in Figure 10. The synthesized filter is functionally tested on Altera™ Stratix II DSP Development Kit by filtering out the high performance component of a mixture of two sinusoidal signals.

Various FIR filter implementations and detailed analysis of filter design is planned as a future study. However, we experience that as the number of taps in the design is increased, the maximum speed of the system will gradually reduce. The reason is the loading effect of the input signal. Since the input signal is connected to every tap in the transposed FIR filter structure, as the number of taps increase, capacitive loading of the interconnections dominate. This effect is also incident in conventional soft multipliers.

It should be mentioned that, FIR filter is only a good example for the implementation. Together with fixed-coefficient architectures, variable coefficient multiply and add units and other arithmetic structures that require multiply and add operations can be tailored according to the requirements. The arithmetic operations proposed here can be applied to high performance systems that recursively require addition and multiplication operations such as digital filtering, matrix multiplication and other similar structures.

4. Conclusion

In this work, double carry-save arithmetic is presented for carry-propagation free operations. Using the proposed scheme, the critical path for each multiply-add operation is reduced to only two LUT cascades, one for multiply and the other for the accumulate operation. The hardware is generated using only (6, 3) counters, giving a very regular structure.

For testing the performance, a fixed coefficient FIR filter implementation methodology suitable for 6-input LUT based FPGAs is presented. The proposed implementation resulted in more than 100 percent speed improvement over conventional fixed-coefficient multiplication based FIR filtering schemes. The filter frequency is recorded as 440 MHz whereas any of arithmetic circuit such as a simple AND gate delay is limited to 500 MHz for the StratixII FPGAs. By exploiting the compatibility of the number system with the FPGA hardware, an extreme speed advantage is gained. To our knowledge, the implementation presented here provides fastest speed in 6-input LUT based FPGAs for the multiply-add operations.

References

[1] Xilinx Inc. (2009), Xilinx Virtex-5 Family Overview.

[2] Altera Corporation (2009), Altera Stratix II Device Handbook.

[3] Parhami, B. (Jan. 1990). Generalized Signed-Digit Number Systems: A Unifying Framework for Redundant Number Representations. *IEEE Trans. on Computers*, Vol. 39, No: 1, 89-98.

[4] Ercegovac, M., Lang, T. (2004), Digital Arithmetic, Morgan Kauffman.

[5] Dhananjay, S. P., Goff, T., Koren, I. (2001), Constant-Time Addition and Simultaneous Format Conversion Based on Redundant Binary Representations, IEEE Trans. on Computers, Vol. 50., 1267-1278.

[6] Noll, T. G. (1991). Carry-Save Architectures for High-Speed Digital Signal Processing. *Journal of VLSI Signal Processing*, Vol. 3, 121-140.

[7] Altera Corporation (2009), Advanced Synthesis Cookbook: A Design Guide for Stratix II, Stratix III, and Stratix IV Devices.

[8] Samueli, H. (1989), An improved search algorithm for the design of multiplierless FIR filters with powers-of-two coefficients, IEEE Trans. On Circuits & Systems, Vol. 39, 1655-1668.

[9] Yurdakul, A., Dündar, G. (2008), Fast and efficient algorithm for the multiplierless realization of linear DSP transforms, Circuits, Devices and Systems, IEE Proceedings, Vol. 149, 205-211.

[10] Aktan, M., Yurdakul, A., Dündar, G. (2008). An Algorithm for the Design of Low-Power Hardware Efficient FIR Filters. accepted for publication in *IEEE Trans. Circuits Syst. I, Reg. Papers I.*

[11] Hartley, R. (1996), Subexpression sharing in filters using canonic signed digit multipliers, IEEE Trans. on Circuits and Systems II, Vol. 43, 677-688.

[12] Parhi, K (1999), VLSI Digital Signal Processing Systems: Design and Implementation, John Wiley & Sons, Inc.

[13] Hormigo, J., Ortiz, M. (2009), Efficient implementations of carry-save adders in FPGAs, IEEE Int. Conf. on application-specific systems, architectures and processors, 207-210

[14] Kamp, W., Bainbridge, S. A. (2007), Multiply accumulate unit optimized for fast dot-product evaluation, International Conference on Field-Programmable Technology, 349-352.

[15] Parandeh-Afshar, H., Brisk, H. P., Ienne, P. (2008), Improving synthesis of compressor trees on FPGAs via Integer Linear Programming, DATE Conference, 1256-1261.

[16] Kamp, W., Bainbridge, S. A., Hayes, M. (2009), Efficient implementation of fast redundant number adders for long word-lengths in FPGAs, Conference on Field-Programmable Technology.

[17] Cardarilli, G.C., Pontarelli, S., Re, M., Salsano, A. (2008). On the use of Signed Digit Arithmetic for the New 6-Inputs LUT Based FPGAs. *International Conference on Electronics, Circuits and Systems*, 602-605.

[18] Koren I. (2002), Computer Arithmetic Algorithms, AK Peters.

[19] Parhami, B. (2008), Computer Arithmetic: Algorithms and Hardware Designs, 2nd Edition, Oxford University Press.

[20] Gonzalez, A. F., Mazumder, P. (2000), Redundant arithmetic, algorithms and implementations, Integration, the VLSI Journal, Vol. 30, 13-53.

[21] Mitra, S. (2001), Digital Signal Processing: A Computer Based Approach, 2nd Edition, Mc-Graw Hill.

[22] Weste, N., Harris, D. (2010), CMOS VLSI Design: A Circuits and Systems Perspective, 4th Edition, Addison-Wesley.

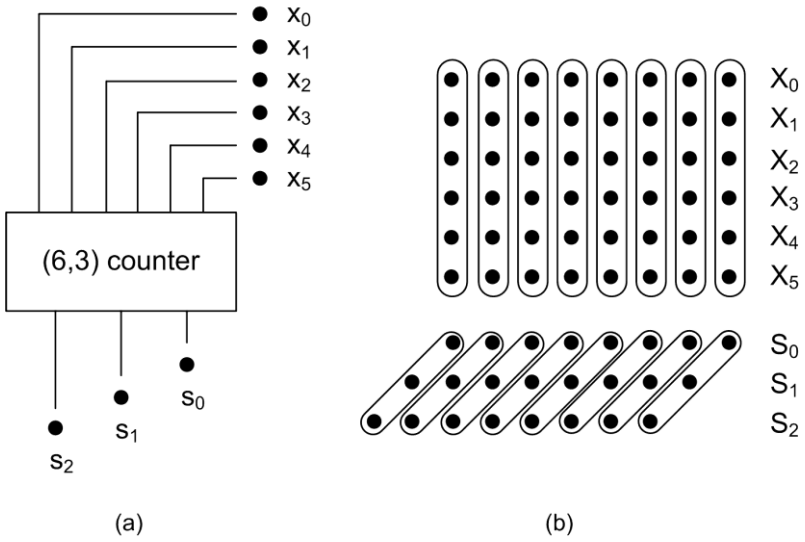
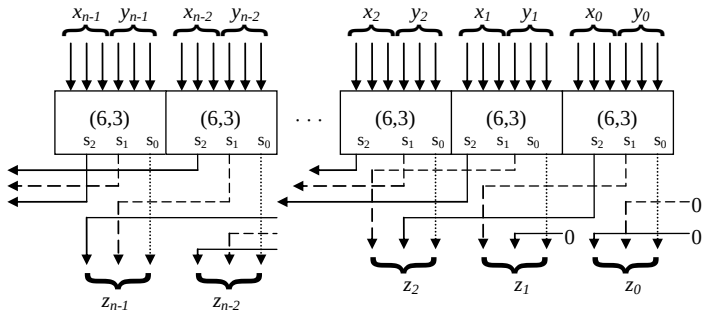
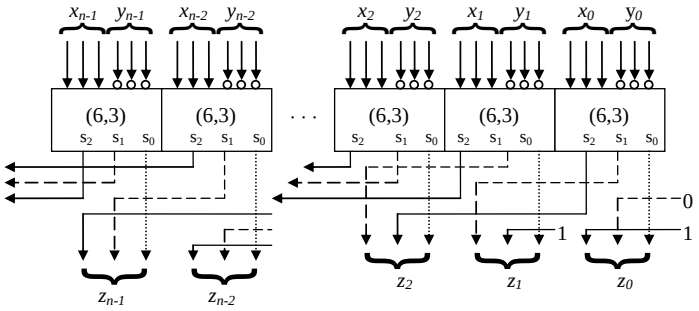


Figure 1. (6, 3) counter: (a) single bit structure (b) addition of multiple operands using counters



(a)



(b)

Figure 2. Redundant Double Carry-Save Arithmetic Operations: (a) Addition; (b) Subtraction

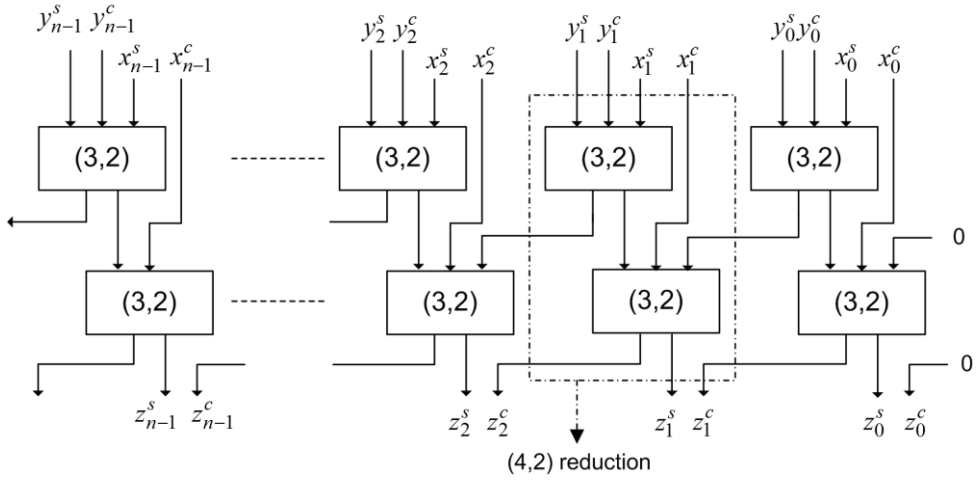


Figure 3. Conventional redundant carry-save addition

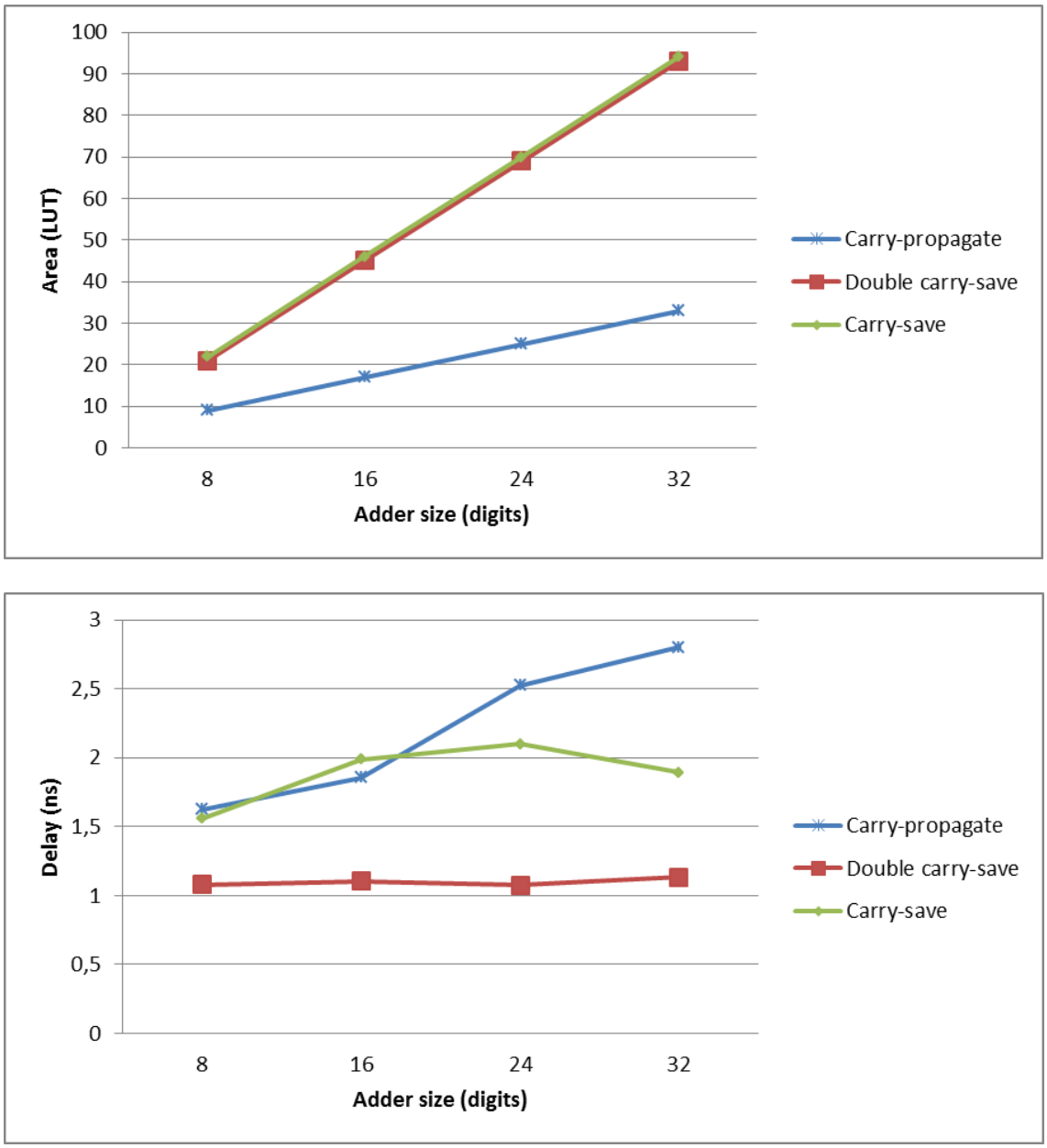
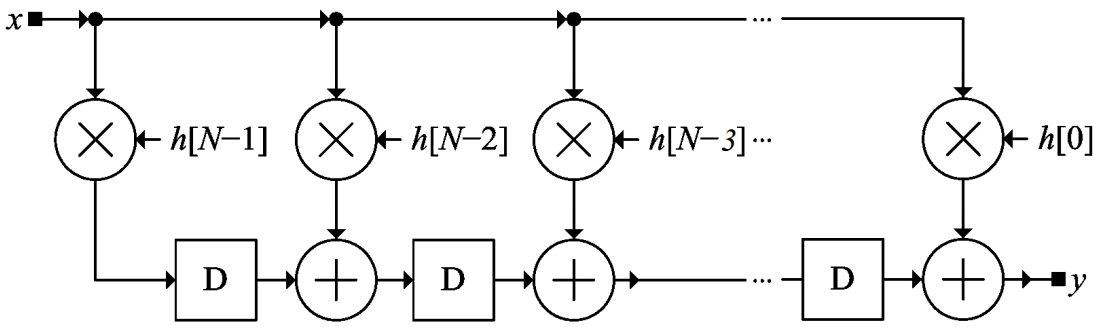
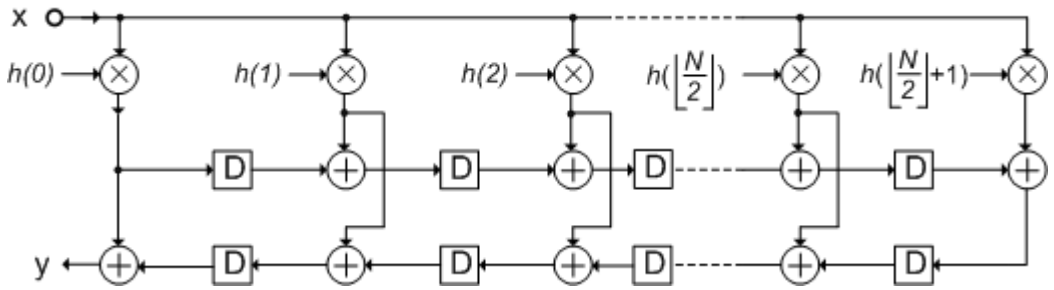


Figure 4. Different adder implementations on a 6-input LUT FPGA



(a)



(b)

Figure 5. FIR filter implementation: (a) Transposed form; (b) sharing the coefficients

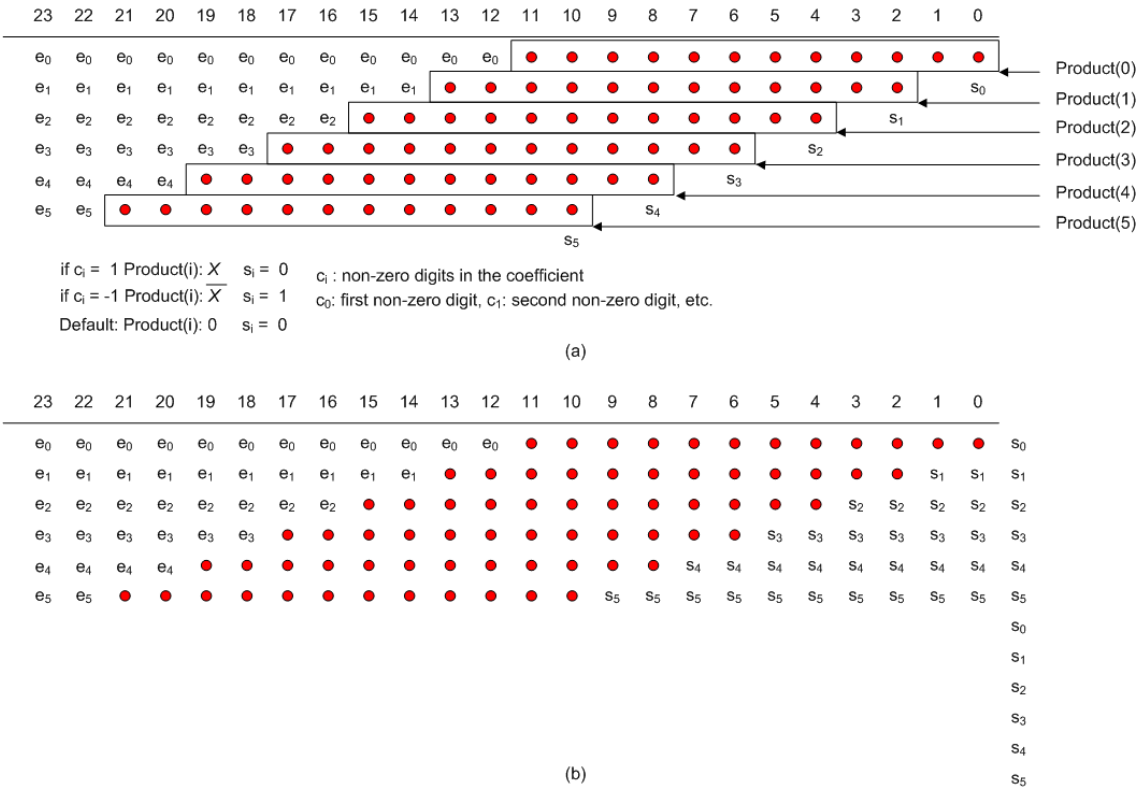
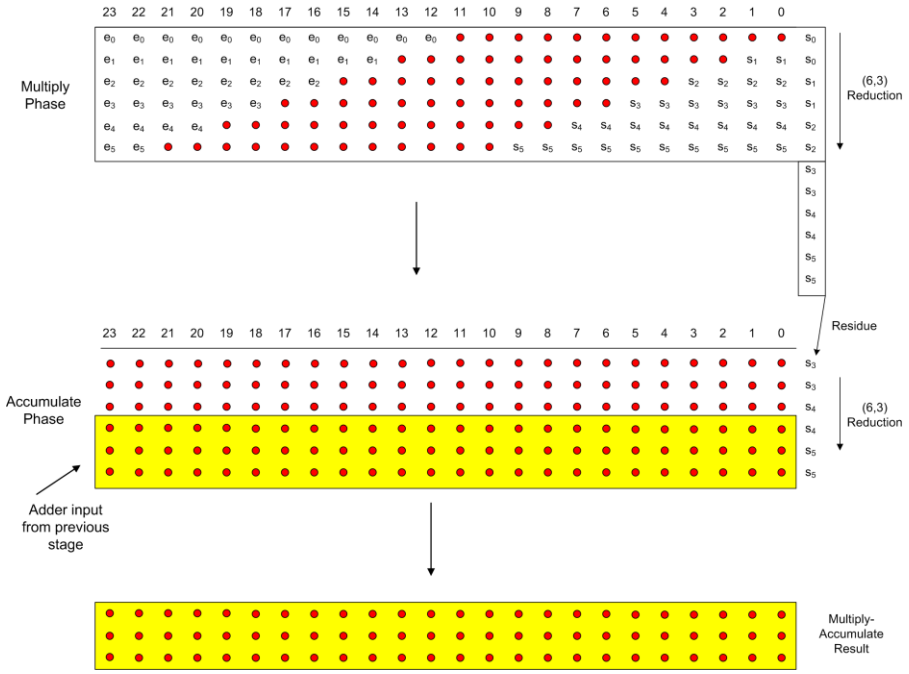
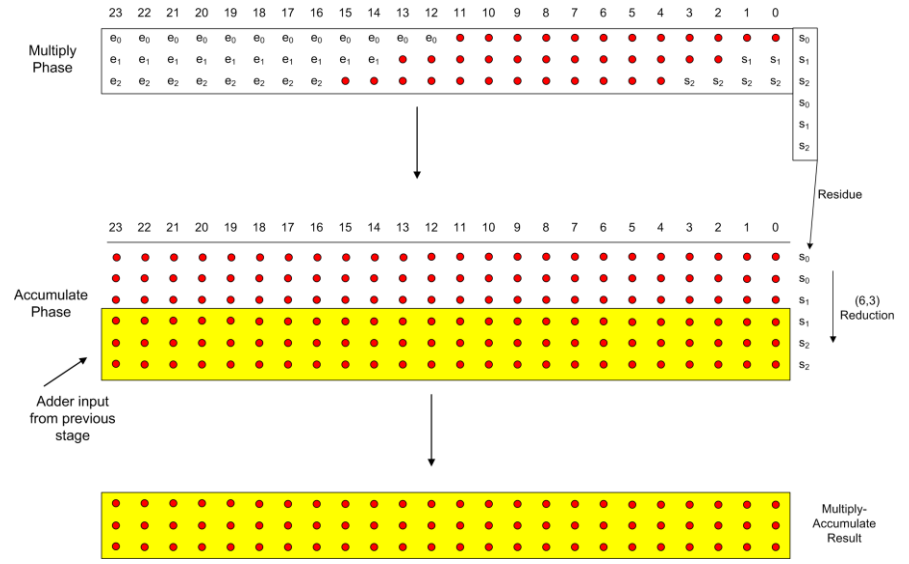


Figure 6. (a) Generation of the partial products; **(b)** backward-sign extension



(a)



(b)

Figure 7. Multiply- add operation: **(a)** Generalized case; **(b)** Up to 3 nonzero bits in the coefficient

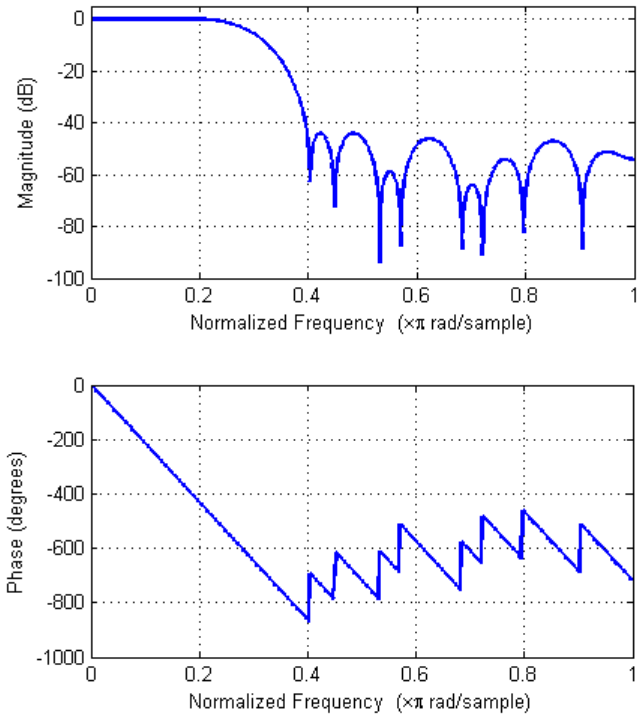


Figure 8. Frequency response of the sample filter.

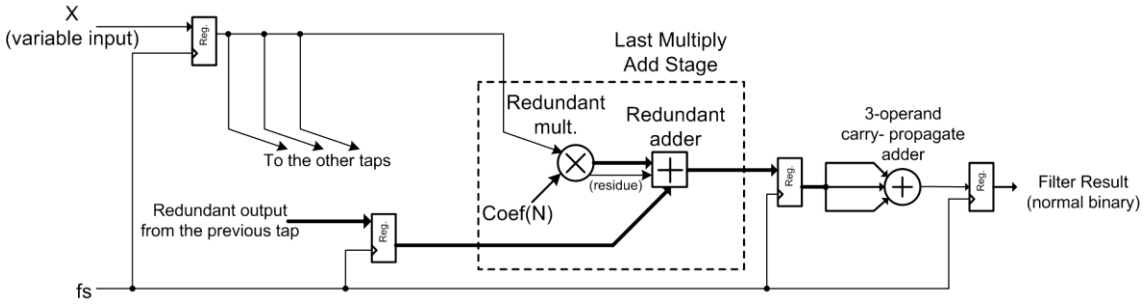


Figure 9. Representation of multiply-add operation and conversion to normal binary

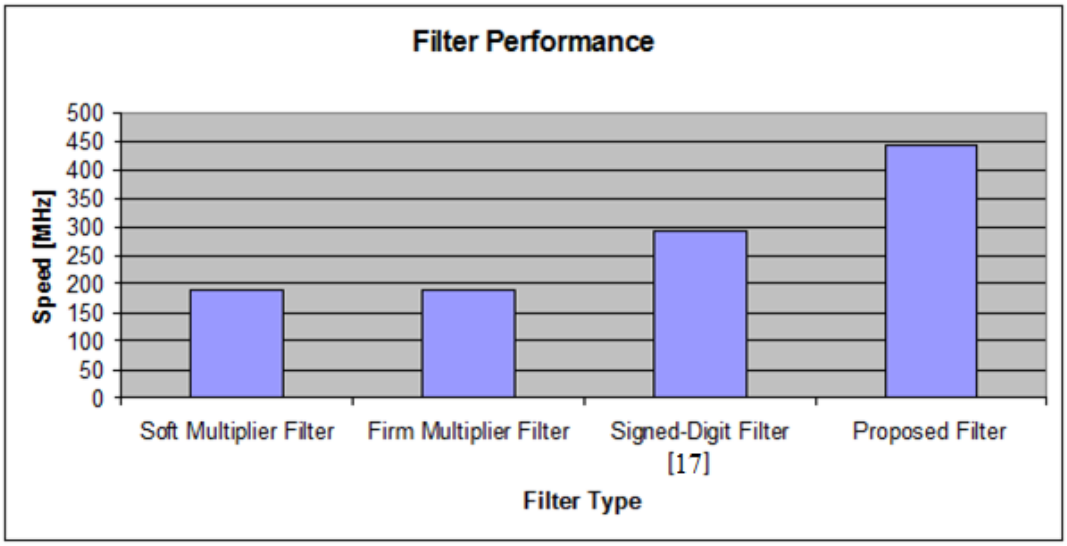


Figure 10. Filter Performance Comparison

Table 1. $T_m(\omega)$ for different types of linear-phase FIR filters

Type	N	Symmetry	$T_m(\omega)$
1	Odd	Symmetric	$\begin{cases} 1 & m = M-1 \\ 2\cos((M-m-1)\omega) & \text{otherwise} \end{cases}$
2	Even	Symmetric	$2\cos((M-m-0.5)\omega)$
3	Odd	Anti-symmetric	$2\sin((M-m-1)\omega)$
4	Even	Anti-symmetric	$2\sin((M-m-0.5)\omega)$

Table 2. Frequency response characteristics of the example filter

Pass-band Ripple (dB)	Pass-band Frequency	Stop-band Attenuation	Stop-band Frequency
0.05	0.2	-44	0.4

Table 3. Optimized coefficients of the filter.

COEFFICIENTS OF FILTER F1 WITH *N*=25, *B*=12			
$h(0)= -2^{-8}$	(*)	$h(7)= -2^{-4} + 2^{-6} - 2^{-8} + 2^{-10}$	
$h(1)= -2^{-7} + 2^{-9}$	(*)	$h(8)= -2^{-5} - 2^{-7} - 2^{-10}$	(*)
$h(2)= -2^{-11}$	(*)	$h(9)= 2^{-5} - 2^{-9}$	(*)
$h(3)= 2^{-6} - 2^{-8} – 2^{-11}$	(*)	$h(10)= 2^{-3} + 2^{-6} + 2^{-8}$	(*)
$h(4)= 2^{-6} + 2^{-8}$	(*)	$h(11)= 2^{-2} + 2^{-8} + 2^{-10}$	(*)
$h(5)= 2^{-7} + 2^{-10}$	(*)	$h(12)= 2^{-2} + 2^{-4} – 2^{-6} + 2^{-8}$	
$h(6)= -2^{-6} - 2^{-8} – 2^{-10}$	(*)		
$h(n)=h(24-n)$ for n=13, 14, …, 24			

Table 4. Comparison of various implementations of the filters

Implementation	**Hardware Cost**		**Maximum Speed (MHz)**
	ALUT + DSP Blocks	**Register Count**	
Fixed–coefficient soft multiplier	814 ALUT	576	188
Firm multiplier	576 ALUT + 26 DSP Blocks	600	190
Proposed method	**1515 ALUT**	**1491**	**446**