

논문 2010-47TC-12-5

Radix-16 Modified Booth 알고리즘을 이용한 저전력 Horizontal DA 필터 구조

(Low-power Horizontal DA Filter Structure Using Radix-16 Modified Booth Algorithm)

신 지 혜*, 장 영 범**

(Ji Hye Shin and Young Beom Jang)

요 약

이 논문에서는 디지털 필터의 저전력 구현을 위한 새로운 DA(Distributed Arithmetic) 필터 구조를 제안한다. 제안된 구조는 입력샘플 비트 포맷에서 수직 방향으로 연산하는 기존의 DA 구조와는 달리 입력샘플 비트를 수평 방향으로 연산하여 ROM이 필요 없으며 Modified booth 알고리즘의 적용이 가능한 저전력 필터 구조이다. 이와 더불어 제안된 필터 구조는 ROM이 필요 없게 되므로 고정된 필터 계수용 필터 뿐 아니라 변하는 필터계수를 갖는 필터 구현에 적용이 가능하다. 제안된 DA 구조와 기존의 DA 구조를 사용하여 20 탭 필터를 Verilog-HDL을 사용하여 구현하였으며, Synopsis로 논리합성한 결과 기존 구조에 비하여 41.6%의 구현 면적 감소효과를 얻을 수 있었다.

Abstract

In this paper, a new DA(Distributed Arithmetic) filter implementation technique has been proposed. Contrary to vertical directional calculation of input sample bit format in the conventional DA implementation technique, proposed implementation technique utilizes horizontal directional calculation of input sample bit format. Since proposed technique calculates in horizontal direction, it does not need ROM and utilizes the Modified Booth algorithm. Furthermore proposed technique can be applied to implement the variable coefficients filters in addition to the fixed coefficients filters. Using conventional and proposed techniques, a 20 tap filter is implemented by Verilog-HDL coding. Through Synopsis synthesis tool, it has been shown that 41.6% area reduction can be achieved.

Keywords : DA(Distributed Arithmetic), Radix-16, Modified Booth multiplier

I. 서 론

디지털 필터는 곱셈기(multiplier), 덧셈기(adder), 지연소자(delay element)의 3가지 소자로 구성된다. 일반적으로 SoC 반도체의 필터는 1개의 곱셈기가 내장되어

필터의 탭 수만큼 반복하여 곱셈연산을 수행하게 된다. 필터의 구현은 곱셈기의 구현 비용이 가장 크므로 곱셈기의 효율적인 구현에 초점이 맞추어진다.

필터 계수가 고정된 경우에는 곱셈연산을 덧셈기를 사용하여 구현하면 효율적이다. 이와 같이 덧셈기를 사용하여 필터를 구현하는 방식으로는 필터계수를 비트화하는 CSD(Canonic Signed Digit) 방식과 입력샘플을 비트화하는 DA(Distributed Arithmetic) 방식이 널리 사용되고 있다. 또한 필터 계수가 변하는 경우의 필터 구현에는 Booth 알고리즘을 이용한 Modified Booth 곱셈기가 널리 사용되고 있다.

* 학생회원, 상명대학교 컴퓨터정보통신공학과
(Graduate School, Sangmyung University)

** 정희원(교신저자), 상명대학교 정보통신공학과
(College of Engineering, Sangmyung University)

※ 이 논문은 2010년도 상명대학교 교내연구비지원에
의하여 연구되었음.

접수일자: 2010년7월20일, 수정완료일: 2010년12월10일

필터 계수를 비트화하는 CSD 방식은 필터 계수를 CSD 형으로 나타낸 후 공통패턴을 공유하여 필터의 구현면적을 줄이는 방식이다.^[1~3] 그러나 CSD 방식은 텁의 수가 매우 큰 경우에 덧셈 연산의 수가 크게 증가하여 Hardwired 방식으로 구현하기 어렵다.

입력샘플을 비트화하는 DA 방식은 필터의 곱셈 연산을 ROM과 덧셈기를 사용하여 구현하는 방식이다.^[4~7] 이 방식은 이미 계산되어 ROM에 입력되어 있는 필터 계수의 조합을 입력신호의 비트 정보에 의해 출력시켜서 더하는 방식이다. DA 구조는 반드시 ROM을 사용하여야 하므로 텁 수가 큰 필터의 구현에는 ROM의 구현면적이 커지게 된다. 또한 필터 계수를 ROM에 저장해 놓고 사용하므로 고정된 필터 계수만 사용이 가능하다는 단점이 있다.

이 논문에서 제안된 필터 구조는 입력샘플의 비트 포맷을 수평방향으로 이용함으로써 미리 필터계수의 조합을 저장할 필요가 없게 되어 ROM이 필요 없는 단순한 필터 구조를 제안한다. ROM이 필요 없게 되므로 고정된 필터 계수 뿐 아니라 변하는 필터계수를 갖는 필터에 적용이 가능하다. 이와 더불어 수평방향의 비트 포맷은 Modified Booth 알고리즘을 이용할 수 있게 되어 더욱 구현이 단순화될 수 있음을 보인다.

이 논문의 II장에서는 기존의 DA 구조를 살펴보고 III장에서는 Modified Booth 알고리즘을 이용한 Horizontal DA 구조를 제안한다. IV장에서는 기존 DA 구조와 제안된 구조를 각각 구현하여 그 면적을 비교하고 V장에서 결론을 맺는다.

II. 기존의 DA 필터 구조

DA 필터는 필터의 곱셈 연산을 ROM과 덧셈기, 쉬프트 연산만으로 처리하는 구현 방식이다.^[4] 즉 ROM에 저장되어 있는 필터 계수의 조합을 입력신호의 비트 정보에 의해 출력시켜서 더하는 필터 구현방식이다. 이 논문에서 제안하는 Horizontal DA 필터와의 비교를 위하여 다음 식의 K 텁 필터에 대한 기존 DA 필터의 원리를 먼저 알아본다.

$$y = \sum_{k=1}^K h_k x_k \quad (1)$$

식 (1)은 FIR 필터의 컨볼류션식으로서 하나의 출력 샘플만을 구하는 식이며, h_k 는 필터 계수이고 x_k 는 입

력 샘플을 나타낸다. 식 (1)에서 입력 샘플 x_k 의 절대 값이 1보다 작다고 가정하면 다음과 같이 2의 보수형 (2's complement)으로 나타낼 수 있다.

$$x_k = -b_{k0} + \sum_{n=1}^{N-1} b_{kn} 2^{-n} \quad (2)$$

식 (2)에서 b_{kn} 은 입력 샘플의 한 비트이므로 0 또는 1의 값이다. b_{k0} 는 입력 샘플의 MSB(Most Significant Bit)인 sign-bit이며, $b_{k,N-1}$ 은 LSB(Least Significant Bit)를 나타낸다. 식 (2)를 식 (1)에 대입하면 출력 샘플 y 를 x_k 의 비트를 사용하여 나타내면 다음과 같은 식으로 표현할 수 있다.

$$\begin{aligned} y &= \sum_{k=1}^K h_k \left[-b_{k0} + \sum_{n=1}^{N-1} b_{kn} 2^{-n} \right] \\ &= \sum_{n=1}^{N-1} \left[\sum_{k=1}^K h_k b_{kn} \right] 2^{-n} + \sum_{k=1}^K h_k (-b_{k0}) \end{aligned} \quad (3)$$

위의 식 (3)을 DA라고 정의한다. 식 (3)의 팔호인 $\left[\sum_{k=1}^K h_k b_{kn} \right]$ 는 한 clock마다 계산되며, b_{kn} 은 오직 0과 1의 값만을 가지므로 이 팔호의 결과 값은 2^K 개의 서로 다른 값을 가질 수 있다. 따라서 매 clock마다 이 값을 계산하는 대신 계산 된 값을 ROM에 저장해 놓고 사용한다. 이때 입력 신호의 비트는 ROM의 어드레스가 되고 ROM의 출력은 $\sum_{k=1}^K h_k b_{kn}$ 이 된다. 이러한 계산

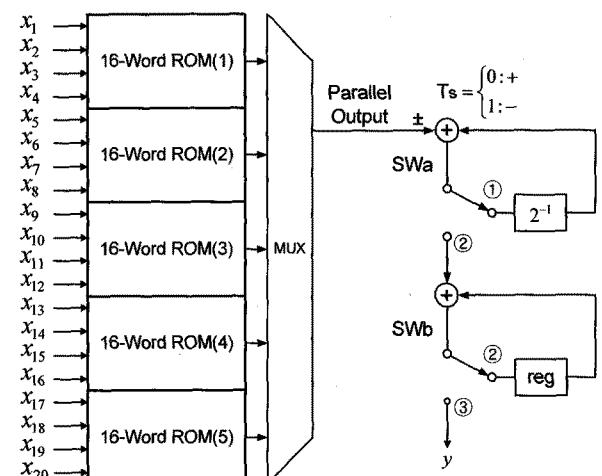


그림 1. 기존의 DA 필터 구조(20텝)

Fig. 1. Conventional DA Filter Structure(20 tap).

을 N clock 반복하면 최종 출력 y 를 얻게 된다. 이 계산에서 K 는 필터의 탭 수로서, 너무 크면 ROM의 크기가 매우 증가 한다. 예를 들어 10 탭 필터인 경우에 ROM은 1024개의 수를 저장하여야 한다. 따라서 필터 탭 수가 큰 경우에는 ROM size를 고려하여 적당한 크기의 탭으로 분리하여 구현한다. 예를 들어 20 탭 필터의 경우에 4 탭으로 분리하여 구현하면 그림 1과 같다.

그림 1은 4 탭의 DA 필터 5개를 사용한 20 탭 DA 필터 구조이다. DA 필터는 탭의 수가 커지면 ROM의 크기도 비례해서 커지기 때문에 적당한 탭의 필터 여러 개로 구성하는 것이 바람직하다. 그림 1에서 20 탭의 필터 계수는 4 탭씩 분리하여 구현하므로 각각의 ROM은 16-word로 구성한다. 필터 계수의 조합인 ROM 출력은 MUX를 통하여 덧셈기로 들어가도록 제어한다. b_{kn} 이 16 비트로 구성된 경우에 첫 번째 clock의 ROM 출력은 $\sum_{k=1}^K h_k b_{k,N-1}$ 이 되며 이 결과 값은 1비트 우로 쉬프트시킨 후에 2번째 clock에서 $\sum_{k=1}^K h_k b_{k,N-2}$ 의 ROM 출력과 더해진다. 15번째 clock의 ROM 출력은 $\sum_{k=1}^K h_k b_{k1}$ 이 된다. 마지막 16번째 clock에서는 식 (3)의 sign-bit 인 $\sum_{k=1}^K h_k (-b_{k0})$ 를 계산하여야 하므로 Add/Sub을

Subtracter로 동작하도록 제어한다. 즉 T_s 제어신호는 sign-bit를 제어하는 신호로 sign-bit를 계산하는 동안에만 1이 되도록 제어한다. 그림 1에서 MUX를 통하여 5개의 4 탭 DA 필터를 제어한다. 예를 들어 16 비트 정세도의 입력을 사용하는 DA 필터에서는 하나의 4 탭 DA 필터의 계산이 완료되는 16 clock마다 MUX가 동작하게 된다. 스위치 SW_a는 ①의 위치에 있다가 하나의 4 탭 DA 필터의 계산이 완료되는 16 clock마다 1 clock 동안 ②의 위치로 스위칭 하도록 제어한다. 스위치 SW_b는 ②의 위치에 있다가 4 탭 필터 5개가 완전히 계산이 끝나는 80 clock 후에 ③의 위치로 스위칭 하도록 제어한다.

< 기존 DA 구조를 사용한 4 탭 필터 계산 예제>

다음과 같은 4 탭 DA 필터 예제를 사용하여 연산 과정을 자세히 알아보자.

$$y = \sum_{k=1}^4 h_k x_k$$

$$\begin{aligned} h_1 &= 0.38, \quad x_1 = -0.89946 \\ h_2 &= 0.27, \quad x_2 = 0.20574 \\ h_3 &= 0.23, \quad x_3 = -0.88163 \\ h_4 &= 0.1, \quad x_4 = 0.58842 \end{aligned} \quad (4)$$

먼저 4개의 입력 샘플에 대하여 16 비트 정세도의 비트로 나타내면 표 1과 같다.

입력신호들은 표 1의 이중실선과 같이 vertical 방향으로 LSB부터 한 비트 씩 입력되면서 ROM의 어드레스로 동작하게 된다. 즉 첫 번째 clock에서는 입력 샘플의 LSB인 1111이 ROM의 어드레스로 동작하여 $h_1 + h_2 + h_3 + h_4 = 0.98$ 을 출력시킨다. 두 번째 clock에서는 LSB-1인 1010이 ROM의 어드레스로 동작하여 $h_1 + h_3 = 0.61$ 을 출력시킨다. 따라서 16-Word ROM의 어드레스별 저장 값은 표 2와 같다.

이 예제의 경우에 clock 별 선택되는 어드레스와

표 1. 예제의 입력을 사용하는 4 탭 DA 필터의 입력 신호 구조(16 비트)

Table 1. Input data structure of 4 tap conventional DA Filter using 16bit input data.

x_1	1	0	0	0	1	1	0	0	1	1	0	1	1	1	1	1
x_2	0	0	0	1	1	0	1	0	0	1	0	1	0	1	0	1
x_3	1	0	0	0	1	1	1	1	0	0	1	0	0	1	1	1
x_4	0	1	0	0	1	0	1	1	0	1	0	1	0	0	0	1

표 2. 예제의 16-Word ROM의 저장값

Table 2. Memory of 16-Word ROM in this example.

address	b_{1n}	b_{2n}	b_{3n}	b_{4n}	16-Word ROM values
0	0	0	0	0	0
1	0	0	0	1	$h_4 = 0.1$
2	0	0	1	0	$h_4 = 0.23$
3	0	0	1	1	$h_3 + h_4 = 0.33$
4	0	1	0	0	$h_2 = 0.27$
5	0	1	0	1	$h_2 + h_4 = 0.37$
6	0	1	1	0	$h_2 + h_3 = 0.5$
7	0	1	1	1	$h_2 + h_3 + h_4 = 0.6$
8	1	0	0	0	$h_1 = 0.38$
9	1	0	0	1	$h_1 + h_4 = 0.48$
10	1	0	1	0	$h_1 + h_3 = 0.61$
11	1	0	1	1	$h_1 + h_3 + h_4 = 0.71$
12	1	1	0	0	$h_1 + h_2 = 0.65$
13	1	1	0	1	$h_1 + h_2 + h_4 = 0.75$
14	1	1	1	0	$h_1 + h_2 + h_3 = 0.88$
15	1	1	1	1	$h_1 + h_2 + h_3 + h_4 = 0.98$

표 3. 이 예제의 clock 별 어드레스와 ROM 출력
Table 3. Address and ROM output in this example.

clock	어드레스	ROM 출력	clock별 A/S 출력
0	15	0.98	0.98
1	10	0.61	1.1
2	14	0.88	1.43
3	8	0.38	1.095
4	13	0.75	1.2975
5	2	0.23	0.87875
6	13	0.75	1.189375
7	8	0.38	0.9746875
8	3	0.33	0.81734375
9	7	0.6	1.008671875
10	10	0.61	1.114335938
11	15	0.98	1.537167969
12	4	0.27	1.038583985
13	0	0	0.5192919925
14	1	0.1	0.3596459963
15	10	0.61	-0.4301770018

ROM 출력 값은 표 3과 같다. 표 3에서 보듯이 16 clock 후의 최종 필터 출력은 -0.4301770018이 된다.

III. 제안된 Horizontal DA 필터 구조

1. Radix-16 Booth 알고리즘

Booth 알고리즘은 곱셈을 수행할 때, 승수의 값이 0인 자리들에 대해서는 그 곱을 더 할 필요 없이 쉬프트만 하면 되고, 2^k 에서 2^m 까지의 값이 1인 자리들은 $2^{k+1} - 2^m$ 과 동등하게 취급할 수 있다는 점을 이용한다. 예를 들어 승수가 001111인 경우 010000 - 000001 = 010001과 같이 표현 할 수 있다. 1의 수가 4개에서 2개로 감소되었으므로 하드웨어의 구현비용을 줄일 수 있다. 이와 같이 0과 1만을 사용하는 2의 보수형 수보다 1, 0, -1을 사용하는 CSD 수는 1 또는 -1의 수가 적어지는 장점을 갖는다.^[2] 그러나 Booth 알고리즘은 승수(multiplier)가 계속 변하는 경우에는 구조가 균일성(regularity)을 보장하지 못하므로 구현 비용이 증가할 수도 있다. 이러한 단점을 보완하여 부분 곱(partial product)의 수도 감소하며 균일한 구조를 만들 수 있는 Modified Booth 알고리즘이 곱셈기에 사용되고 있다.^[8] 기존 DA 필터에서는 보통 입력샘플 비트를 4비트씩 연산하므로 이 논문에서도 표 4와 같이 입력샘플 비트를 4비트씩 연산하도록 하였다.

기존 DA 필터의 입력샘플 비트 포맷과 다른 점은 기존 DA 필터 구조에서는 입력샘플 비트 표에서 수직방향으로 연산하는데 반하여 제안된 DA 구조에서는 입력

표 4. 제안된 DA 필터 구조의 입력 샘플 비트 포맷
(16 bit 정세도의 경우)

Table 4. Input sample bit format in the proposed DA filter structure (16 bit resolution).

x_1	1	0	0	0	1	1	0	0	1	1	0	1	1	1	1	1
x_2	0	0	0	1	1	0	1	0	0	1	0	1	0	1	0	1
x_3	1	0	0	0	1	1	1	1	0	0	1	0	0	1	1	1
x_4	0	1	0	0	1	0	1	1	0	1	0	1	0	0	0	1

그림 2. 16 bit 입력의 Radix-16 Modified Booth Grouping
Fig. 2. 16 bit input data of Radix-16 Modified Booth's Grouping.

	$x_{i+3}x_{i+2}x_{i+1}x_i$	x_{i-1}	Case	$x_{i+3}x_{i+2}x_{i+1}x_i$	x_{i-1}	Case
Case 1	0 0 0 0	0	$0 \rightarrow 0H$	Case 18	0 1 1 0	0 1 1 0
Case 2	0 0 0 0	1 1 1	$= 0 0 0 1 0 0 \bar{1} \rightarrow +1H$	Case 19	0 1 1 0 0 0 1 0	$= 1 0 1 0 1 1 0 \rightarrow -7H$
Case 3	0 0 0 1	0	$\rightarrow +1H$	Case 20	0 1 1 0 0 0 1 1 0	$= 1 0 1 0 1 1 0 \bar{1} 0 \rightarrow -6H$
Case 4	0 0 0 1	1 0	$= 0 0 0 1 0 1 0 \rightarrow +2H$	Case 21	0 1 1 0 1 0 0 0	$= 1 0 1 1 1 1 0 \rightarrow -6H$
Case 5	0 0 1 0	0	$\rightarrow +2H$	Case 22	0 1 1 0 1 0 0 1 1 0	$= 1 0 1 1 1 1 0 \bar{1} 0 \rightarrow -5H$
Case 6	0 0 1 0	1 1 0	$= 0 0 1 1 0 \bar{1} 0 \rightarrow +3H$	Case 23	0 1 1 0 1 0 1 1 0	$= 1 0 1 1 1 0 1 1 0 \rightarrow -5H$
Case 7	0 0 1 1	0	$\rightarrow +3H$	Case 24	0 1 1 0 1 1 1 0 1 0	$= 1 0 1 1 1 0 1 1 0 \bar{1} 0 \rightarrow -4H$
Case 8	0 0 1 1	1 0	$= 0 1 0 1 1 1 0 \bar{1} 0 \rightarrow +4H$	Case 25	0 1 1 1 1 0 0 0 0	$= 1 0 1 0 1 1 0 0 0 \rightarrow -4H$
Case 9	0 1 0 0	0	$\rightarrow +4H$	Case 26	0 1 1 1 1 0 0 1 1 0	$= 1 0 1 0 1 1 0 0 1 1 0 \rightarrow -3H$
Case 10	0 1 0 0	1 1 0	$= 0 1 0 1 1 1 0 \bar{1} 0 \rightarrow +5H$	Case 27	0 1 1 1 1 0 1 1 0	$= 1 0 1 0 1 1 0 1 1 0 \rightarrow -3H$
Case 11	0 1 0 1	0	$\rightarrow +5H$	Case 28	0 1 1 1 1 1 0 1 1 0	$= 1 0 0 1 1 1 1 0 1 1 0 \bar{1} 0 \rightarrow -2H$
Case 12	0 1 0 1	1 0	$= 0 1 1 0 1 1 0 \bar{1} 0 \rightarrow +6H$	Case 29	0 1 1 1 1 1 1 0 1 0	$= 1 0 0 1 1 1 1 1 0 1 0 \rightarrow -2H$
Case 13	0 1 1 0	0	$= 1 0 1 0 1 1 0 \bar{1} 0 \rightarrow +6H$	Case 30	0 1 1 1 1 1 1 0 1 1 0	$= 1 0 0 0 1 1 1 1 0 1 1 0 \bar{1} 0 \rightarrow -1H$
Case 14	0 1 1 0	1 1 0	$= 1 0 1 1 1 1 0 \bar{1} 0 \rightarrow +7H$	Case 31	0 1 1 1 1 1 1 1 0	$= 1 0 0 0 1 1 1 1 1 0 \bar{1} 0 \rightarrow -1H$
Case 15	0 1 1 1	0	$= 1 0 0 1 1 1 1 0 \bar{1} 0 \rightarrow +7H$	Case 32	0 1 1 1 1 1 1 1 1 0	$= 1 0 0 0 1 1 1 1 1 1 0 \bar{1} 0 \rightarrow 0H$
Case 16	0 1 1 1	1 0	$= 1 0 0 0 1 1 1 1 0 \bar{1} 0 \rightarrow +8H$			
Case 17	0 1 1 1	0 0	$= 1 0 0 0 1 1 1 1 0 \bar{1} 0 \rightarrow -8H$			

그림 3. Radix-16 Modified Booth 알고리즘 승수 Y의 case 별 Partial Product

Fig. 3. Partial Product by case of Multiplier Y Using Radix-16 Modified Booth Algorithm.

샘플 비트 표를 표 4의 이중 실선과 같이 수평 방향으로 4비트씩 관찰하여 연산한다. 따라서 제안 구조를 하는 Horizontal DA 필터 구조로 명명하였다. 제안 구조에서 입력신호를 수평 방향으로 비트화한 것은 Modified Booth 알고리즘을 적용하기 위함이다. 즉 표 4와 같이 4비트씩 grouping하므로 Modified Booth 알

고리즘을 적용하기 위하여 실제로는 그림 2와 같이 5비트씩 관찰하여야 한다.

그림 2에서 보듯이 16 비트 정제도 입력의 경우에 4비트씩 연산하므로 4개의 부분 곱이 발생하게 된다. 이와 같이 4개의 비트를 한 번에 연산하는 알고리즘을 Radix-16 Modified Booth 알고리즘이라고 부르며 실제로는 그림 3과 같이 5비트를 관찰하게 된다.

실제로 5비트씩 관찰하므로 Radix-16 Booth 알고리즘은 그림 3과 같이 모두 32가지의 경우가 발생한다. 입력샘플을 비트 포맷으로 나타냈으므로 부분곱은 필터계수 H 의 가중치로 나타낼 수 있으며 32가지의 각각의 경우에 대한 가중치는 그림 3과 같다. 예를 들어서 case 2에서 보듯이 00001은 00011이 되어 LSB인 1을 제외하면 1H가 된다. 또한 case 32의 11111의 경우에는 00001이 되어 역시 LSB인 1을 제외하면 0H가 된다. 이와 같이 Modified Booth 알고리즘을 이용하여 부분곱을 연산하는 하드웨어를 Modified Booth Encoder라고 부른다. 따라서 제안된 구조에서는 Radix-16 Modified Booth Encoder가 필요하다.

2. 제안된 Horizontal DA 필터의 전체 구조

지난 절에서 논한 Radix-16 Modified Booth encoder를 포함한 제안된 Horizontal DA 필터 구조는 그림 4와 같다.

그림 4는 20 탭의 FIR 필터에 대한 제안 구조이다. 2개의 MUX는 각각 입력신호와 필터계수를 4 clock마다 하나씩 선택하도록 설계한다. 따라서 첫 4 clock 동안에

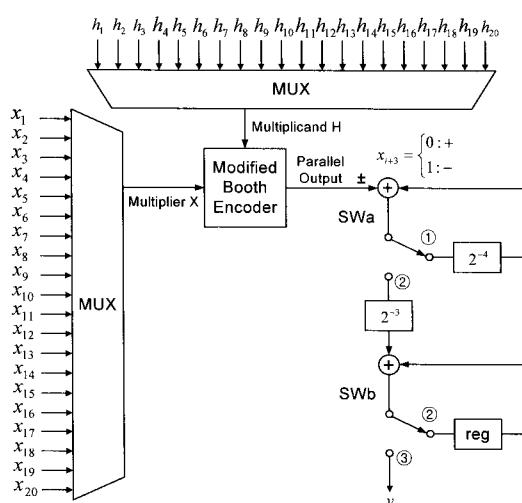


그림 4. 제안된 20 탭 Horizontal DA 필터 구조
Fig. 4. Proposed 20 tap Horizontal DA Filter Structure.

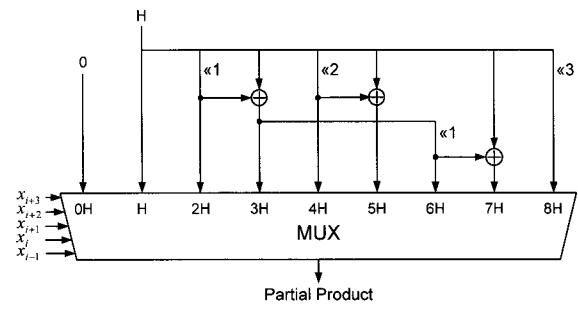


그림 5. Radix-16 Modified Booth Encoder의 세부 구조
Fig. 5. Radix-16 Modified Booth Encoder Structure.

는 입력용 MUX와 계수용 MUX가 각각 x_1 과 h_1 을 선택한다. 입력용과 계수용 MUX의 input은 각각 20개씩 이므로 각각의 MUX가 input을 모두 받아들이는 데에 총 80 clock이 필요하다. 입력용 MUX의 제어신호로는 up counter를 사용하여 x_1 부터 x_{20} 을 순차적으로 선택하도록 제어한다. 계수용 MUX도 같은 방법으로 h_1 부터 h_{20} 까지 순차적으로 선택된다.

선택된 하나의 입력신호 x_n 과 필터계수 h_n 은 Modified Booth Encoder로 입력되어 4 clock 동안 Booth 알고리즘을 통한 곱셈 연산을 수행하게 된다. Radix-16 Modified Booth Encoder의 내부 하드웨어 구조는 그림 5와 같다.

그림 5에서 보듯이 필터계수 샘플 H 가 입력되면 동시에 9가지의 부분 곱이 만들어지도록 설계하였다. 2H는 H 를 1 비트 좌로 쉬프트 시켜서 만들고 3H는 2H와 H 를 더하여 만든다. 이와 같은 방식으로 덧셈기 3개와 쉬프트기 4개를 사용하여 9가지 부분 곱을 얻는다.

그림 5의 MUX에서는 입력샘플 비트가 selector로 동작하도록 설계하였다. 즉 32가지 입력샘플 비트에 따른 MUX의 선택을 요약하면 표 5와 같다.

표 5에서 보듯이 원래 Radix-16 Modified Booth Encoder는 0H, $\pm 1H \sim \pm 8H$ 의 17가지의 부분 곱을 생성하여야 한다. 그러나 우리가 설계한 Encoder는 9가지의 부분 곱만을 출력하도록 설계하였다. 32가지 case들은 입력 비트의 조합에 따라 case 1부터 case 16까지는 양의 부분 곱을 생성하고 case 17부터 case 32까지는 음의 부분 곱을 생성한다. 그러나 음의 부분 곱은 양의 부분 곱을 그대로 사용할 수 있다. 즉 입력샘플 비트 $x_{i+3}x_{i+2}x_{i+1}x_ix_{i-1}$ 을 관찰하면 x_{i+3} 이 1이면 부분 곱이 음수가 됨을 알 수 있다. 따라서 x_{i+3} 의 비트를 판별하여 0인 경우에는 Add/Sub을 Adder로 동작하

표 5. Modified Booth Encoder MUX의 입력샘플 비트 별 부분곱

Table 5. Partial product for input sample bit in the Modified Booth Encoder MUX.

case	x_{i+3}	x_{i+2}	x_{i+1}	x_i	x_{i-1}	partial product
1	0	0	0	0	0	0H
2	0	0	0	0	1	H
3	0	0	0	1	0	H
4	0	0	0	1	1	2H
5	0	0	1	0	0	2H
6	0	0	1	0	1	3H
7	0	0	1	1	0	3H
8	0	0	1	1	1	4H
9	0	1	0	0	0	4H
10	0	1	0	0	1	5H
11	0	1	0	1	0	5H
12	0	1	0	1	1	6H
13	0	1	1	0	0	6H
14	0	1	1	0	1	7H
15	0	1	1	1	0	7H
16	0	1	1	1	1	8H
17	1	0	0	0	0	-8H
18	1	0	0	0	1	-7H
19	1	0	0	1	0	-7H
20	1	0	0	1	1	-6H
21	1	0	1	0	0	-6H
22	1	0	1	0	1	-5H
23	1	0	1	1	0	-5H
24	1	0	1	1	1	-4H
25	1	1	0	0	0	-4H
26	1	1	0	0	1	-3H
27	1	1	0	1	0	-3H
28	1	1	0	1	1	-2H
29	1	1	1	0	0	-2H
30	1	1	1	0	1	-H
31	1	1	1	1	0	-H
32	1	1	1	1	1	0H

록 제어하고 1인 경우 Subracter로 동작하도록 제어함으로써 회로를 단순화시켰다. 즉 -H부터 -8H까지의 음의 부분 곱은 부가적인 하드웨어를 사용하지 않고, 그림 4의 Add/Sub 회로가 Subracter로 동작하도록 설계하였다. 이렇게 함으로써 Modified Booth Encoder 회로의 구현 면적을 감소시킬 수 있다.

이렇게 만들어진 가중치는 Add/Sub 루프를 3번 돌고 4번째 부분 곱과 더해지면 완전한 1 템 곱셈 결과값을 만들어내도록 설계하였다. 즉 첫 번째 부분 곱은 루프를 통해 우로 4 비트 쉬프트시켜서 1/16로 만든 후에 다음 부분 곱과 더하고 이 값은 다시 1/16로 만든 후 다음 부분 곱과 더한다. 스위치 SW_a는 ①의 위치에 있다가 하나의 입력신호와 필터계수의 곱셈 연산이 완료되는 4 clock마다 ②의 위치로 스위칭하여 1 템 곱셈 결과를 아래 덧셈기로 내려 보내도록 설계하였다. 이때 1 템 필터 결과값은 정수(integer number)이므로 십진수(decimal number)로 변환하여야 한다. 따라서 1/8로 스케일링 하기위해서 2^{-3} 하드웨어를 추가하였다. 스위치 SW_b는 ②에 위치하며 4 clock마다 하나의 곱셈결과를

받아서 20번을 더하면 완전한 출력을 얻게 된다. 따라서 모든 계산이 완료된 80번째 clock에서 ③의 위치로 스위칭하여 최종 결과를 출력하도록 설계하였다.

< 제안 구조의 4 템 필터 계산 예제 >

식 4에서 사용한 예제를 다시 사용하여 제안 구조의 연산 과정을 살펴보기로 한다. 4 템 필터이므로 총 16 clock이 필요하며 각 clock 별 계산 결과는 표 6과 같다. 표 6에서 보듯이 4 clock 후에 1 템의 필터 계수와 입력 샘플의 곱이 구해진다. 같은 방법으로 16 clock이 지나면 4 템 필터의 결과를 얻을 수 있다.

표 6에서 보듯이 2nd adder input 4개를 더하면 -0.430177002를 얻을 수 있다. 이 결과값은 기존 DA필터 구조의 계산 예제의 결과값과 같음을 알 수 있다.

표 6. 제안구조의 clock 별 부분곱과 Add/Sub 출력

Table 6. Partial product and Add/Sub output in the proposed structure.

clock	H	partial product	Add/Sub output	2nd adder input
1	0.38	-H=-0.38	-0.38	
2		-2H=-0.76	-0.78375	
3		-3H=-1.14	-1.188984375	
4		-7H=-2.66	-2.7343115234375	-0.34178894
5	0.27	5H=1.35	1.35	
6		5H=1.35	1.434375	
7		-6H=-1.62	-1.5303515625	
8		2H=0.54	0.44435302734375	0.055544128
9	0.23	7H=1.61	1.61	
10		2H=0.46	0.560625	
11		-H=-0.23	-0.1949609375	
12		-7H=-1.61	-1.62218505859375	-0.202773132
13	0.1	H=0.1	0.1	
14		5H=0.5	0.50625	
15		-5H=-0.5	-0.468359375	
16		5H=0.5	0.4707275390625	0.058840942

IV. 구 현

이 절에서는 제안된 구조의 성능을 평가하기 위해 기존 DA 필터 구조와 제안된 DA 필터 구조를 구현하여 각각의 면적을 비교한다.

먼저 20 템의 필터 계수와 20개의 입력신호 샘플을 사용하여 테스트 벡터를 만든다. 사용된 필터계수와 입력샘플은 그림 6과 같다.

검증을 위한 테스트벡터는 Matlab을 통해 -1에서 1까지의 20개의 입력 신호를 랜덤으로 만들어 사용하였

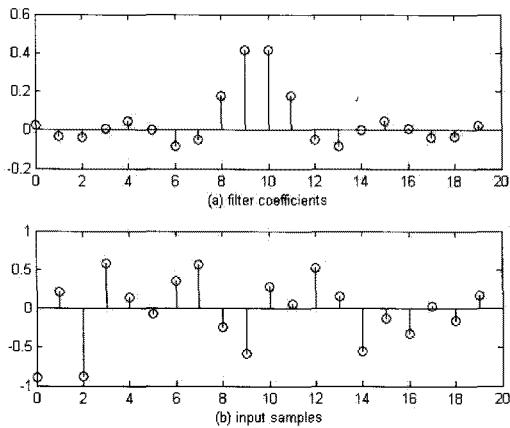


그림 6. 사용된 20탭 필터계수와 20개의 입력신호 샘플
Fig. 6. 20 tap filter coefficients and 20 input samples.



그림 7. Radix-16 Horizontal DA 필터의 simulation 결과
Fig. 7. Simulation result of Radix-16 Horizontal DA Filter.

표 7. 제안구조의 면적비교
Table 7. Gate count comparison for proposed structure.

	기존 구조	제안구조
ports	659	675
nets	2576	1909
cells	1613	1099
Total area(μm^2)	66541	38859
면적 비율	100%	58.4%

고 필터 계수는 저역통과 필터의 필터계수 20개를 사용하였다. 출력력 및 필터계수의 정세도는 각각 16 비트를 사용하였다. 20개의 입력이 20 탭의 필터계수와 MAC(Multiplication and Accumulation) 연산이 수행되면 한 개의 출력이 생성된다.

MatLab을 사용하여 연산한 결과값은 -0.059854364 (십진수) 또는 1111100001010111(2진수)이다.

각각의 구조에 대하여 모두 Xilinx사의 ISE 7.1i를 이용하여 Verilog-HDL로 코딩하였으며 Function 시뮬레이션을 위해 Modelsim을 사용하였다. Verilog-HDL 시뮬레이션 결과는 그림 7에서 보듯이 1111100000110100으로서 Matlab 시뮬레이션 결과와 같음을 알 수 있다.

마지막으로 각각의 구조에 대하여 Synopsis Design Compiler Tool을 사용하여 합성한 후, 면적을 비교하였다. Chartered 0.18 μm CMOS 셀 라이브러리를 사용하였으며 각 구조마다 동일한 constraints를 적용하여 실

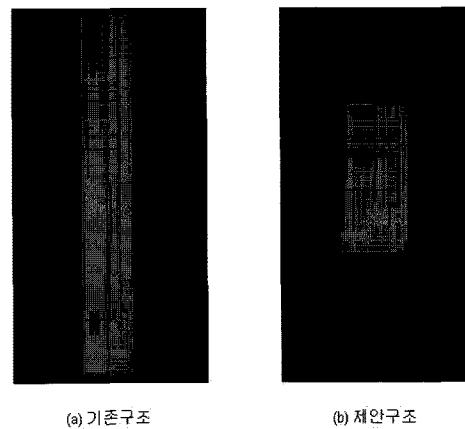


그림 8. 제안구조의 Schematic Design
Fig. 8. Schematic Design for proposed structure.

험하였다. 각각의 구조의 합성결과에 대한 Area_Report는 표 7과 같고 Schematic은 그림 8과 같다.

표 7에서 보듯이 기존 Vertical DA 필터 구조의 경우 합성 후 면적이 66541 μm^2 로 계산되었으며 제안된 Horizontal DA 필터 구조는 합성 후 면적이 38859 μm^2 로 계산되었다. 제안된 구조가 기존의 구조에 비하여 41.6%의 면적 감소효과가 있음을 볼 수 있다.

V. 결 론

이 논문에서는 기존의 필터 구현방식중 하나인 DA 필터 구조를 개선한 새로운 DA 필터 구조를 제안하였다. 우리는 제안된 방식을 Horizontal DA 필터 구조로 명명하였으며 이 제안된 구조는 입력샘플의 비트 포맷을 수평방향으로 연산하므로 기존 DA 필터 구조에서 필요하던 ROM을 사용하지 않아도 되며, Modified Booth 알고리즘을 이용할 수 있는 장점이 있다. 따라서 제안된 필터 구조는 기존 DA 필터 구조와 비교하여 구현 면적이 감소됨을 입증하였다.

참 고 문 헌

- [1] K. Hwang, Computer Arithmetic: Principles, Architecture, and Design, New York: Wiley, 1979.
- [2] R. I. Hartley, "Subexpression sharing in filters using canonic signed digit multipliers," IEEE Trans. Circuits and Systems-II: Analog and Digital Signal Processing, vol. 43, No. 10, pp. 677-688, Oct. 1996.

- [3] M. Yagyu, A. Nishihara, and N. Fujii, "Fast FIR digital filter structures using minimal number of adders and its application to filter design," IEICE Trans. Fundamentals of Electronics Communications & Computer Sciences, vol. E79-A No. 8, pp. 1120-1129, Aug. 1996.
- [4] S. A. White, "Applications of distributed arithmetic to digital signal processing: A tutorial review," IEEE ASSP Magazine, pp. 4-19, July 1989.
- [5] A. Sinha and M. Mehendale, "Improving area efficiency of FIR filters implemented using distributed arithmetic," Proc. Eleventh International Conference on VLSI Design, pp. 104-109, 1998.
- [6] 임인기, 정희범, 김경수, 김환우, "IMT-2000 시스템을 위한 승산기를 사용하지 않는 인터플레이션 FIR 필터 구현," 한국통신학회논문지, '02-10 Vol.27 No.10C pp.1008-1014, 2002년 10월.
- [7] 장영범, 이현정, 문종범, 이원상, "덧셈 프로세서를 사용한 IMT-2000 인터플레이션 필터의 저전력 설계 및 구현," 전자공학회논문지, 제42권 SP편, 제1호, 79-85쪽, 2005년 1월.
- [8] 조경주, 김원관, 정진균, "Booth 인코더 출력을 이용한 저오차 고정길이 modified Booth 곱셈기 설계," 한국통신학회논문지, '04-2 Vol.29 No.2C pp.298-305, 2004년 2월.

저 자 소 개



장 영 범(정희원)-교신저자
 1981년 연세대학교 전기공학과
 졸업.(공학사)
 1990년 Polytechnic University
 대학원 졸업.(공학석사)
 1994년 Polytechnic University
 대학원 졸업.(공학박사)

1981년~1999년 삼성전자 System LSI 사업부
 수석연구원.
 2000년~2002년 이화여자대학교 정보통신학과
 연구교수.
 2002년~현재 상명대학교 정보통신공학과 교수.
 <주관심분야 : 통신신호처리, 비디오신호처리,
 SoC 설계>



신 지 혜(학생회원)
 2008년 상명대학교 정보통신
 공학과 졸업.(공학사)
 2009년~현재 상명대학교 대학원
 컴퓨터 정보통신공학과
 석사과정.

<주관심분야 : 통신신호처리, SoC 설계>