논문 2005-42SP-1-11

덧셈 프로세서를 사용한 IMT-2000 인터폴레이션 필터의 저전력 설계 및 구현

(Low-power Design and Implementation of IMT-2000 Interpolation Filter using Add/Sub Processor)

장 영 범*, 이 현 정*, 문 종 범*, 이 원 상**

(Young-Beom Jang, Hyun-Jung Lee, Jong-Beom Moon, and Won-Sang Lee)

요 약

이 논문에서는 IMT-2000용 인터폴레이션 필터의 저전력 설계 및 구현 방식을 제안하였다. DA(Distributed Arithmetic) 방식의 장점인 프로세서 구조와, CSD(Canonic Signed Digit) 방식의 장점인 덧셈 연산의 최소화 방법을 함께 사용하여 각 구조의 장점을 살린 인터폴레이션 필터 구조를 제안하였다. 필터계수는 CSD형으로 나타낸 후에 4비트씩 가능한 모든 계산을 미리 수행하여 저장하고, MUX와 덧셈 프로세서를 사용하여 곱셈 연산을 수행하도록 설계하였다. 이와 더불어 기존 곱셈기 구조에서 사용되는 출력용 덧셈기와 지연소자는 1개의 덧셈기와 쉬프트 레지스터를 사용하여 효율적으로 구현될 수 있음을 보였다. IMT-2000에서 사용되는 40탭 인터폴레이션 필터에 대하여, 제안된 구조와 기존의 곱셈기를 사용한 구조를 각각 Verilog-HDL 코딩을 통하여 설계하였다. 기존의 곱셈기를 사용한 구조와 게이트 수를 비교한 결과 68.43%의 감소를 달성할 수 있었다.

Abstract

In this paper, low-power design and implementation techniques for IMT-2000 interpolation filter are proposed. Processor technique for DA(Distributed Arithmetic) filter and minimization technique for number of addition in CSD(Canonic Signed Digit) filter are utilized for low-power implementation. Proposed filter structure consists of 3 blocks. In the first CSD coefficient block, every possible 4 bit CSD coefficients are calculated and stored. In second processor block, multiplication is done by MUX and addition processor in terms of filter coefficient. Finally, in third shift register block, multiplied values are output and stored in shift register. For IMT-2000 interpolation filter, proposed and conventional structures are implemented by using Verilog-HDL coding. Gate counts for the proposed structure is reduced to 31.57% comparison with those of the conventional one.

Keywords: IMT-2000, CSD, DA, Interpolation filter

I. 서 론

IMT-2000 이동통신 시스템에서는 저전력으로 동작하는 인터폴레이션 필터가 필요하다. 여기에서 사용되는 인터폴레이션 필터는 탭 수가 크므로 이와 비례하여 곱셈 연산, 덧셈 연산, 지연 연산의 수가 증가하게 되어 구현 비용이 높아지게 된다. 이 가운데에서도 곱셈연산의 구현 비용이 가장 크므로 곱셈 연산의 효율적인 구현에 초점이 맞추어진다. 곱셈 연산의 구현은 강력한 곱셈기를 내장하고 있는 DSP 프로세서를 사용하여 구현하는 방식과 Hardwired 구현 방식이 사용될 수 있다. 이 가운데에서 IMT-2000과 같은 이동 통신 시스템에서는 매우 빠른 필터링이 요구되므로 Hardwired 방식이 많이 사용된다. 필터의 저전력 Hardwired 구현을 위

* 정회원, 상명대학교 정보통신공학과
(Dept. of Information and Telecommunication, Sangmyung University)
** 정회원, 상명대학교 일반대학원 컴퓨터정보통신공학과
(Dept. of Computer, Information, and Telecommunication, Sangmyung University)
접수일자: 2004년11월3일, 수정완료일: 2004년12월16일

하여 곱셈 연산을 곱셈기를 사용하지 않고 구현하는 것이 바람직하다. 이와 같이 곱셈 연산을 곱셈기를 사용하지 않고 덧셈 연산만으로 구현하려면 비트화 기법이 필요한데, 비트화 기법으로는 필터계수를 비트화 하는 CSD(Canonic Signed Digit) 방식과 필터 입력을 비트화 하는 DA(Distributed Arithmetic) 방식이 있다. CSD 방식은 Transposed Direct Form 필터 구조를 사용하며 입력신호에 필터계수를 곱할 때에 쉬프트 연산과 덧셈 연산을 사용하여 곱셈 연산이 이루어지도록 하는 방식이다. 이 때 덧셈 연산의 수를 줄이기 위하여 CSD형의 필터 계수가 사용된다[1][2]. 이와 같은 CSD형의 계수와 더불어 공통패턴을 공유함으로써 곱셈 연산의 수를 더욱 감소시키는 방법도 제안되었다[3][4]. 그러나 이와 같은 CSD 방식은 탭의 수가 큰 경우에 덧셈 연산의 수가 너무 증가하여 Hardwired 방식으로 구현하기에는 구현 면적이 부담이 된다. DA 방식은 필터의 곱셈 연산을 ROM과 덧셈기를 사용하여 구현하는 방식이다[5]-[7]. 이 방식은 이미 계산되어 ROM에 입력되어 있는 필터 계수의 조합을 입력신호의 비트 정보에 의해 출력 시켜서 더하는 방식이다. 그러나 4탭 정도의 단위로 ROM을 사용하여야 하므로 탭의 수가 큰 경우에는 ROM의 수가 증가하는 단점이 발생하게 된다. 본 논문에서 제안된 구조는 CSD 방식의 장점인 덧셈 연산의 감소를 이용하였으며, 동시에 DA 방식의 장점인 프로세서를 사용하여 저전력 구조를 제안한다.

Ⅱ. 제안된 덧셈 프로세서 구조의 설계

1. CSD 필터계수 블록 설계

본 논문에서는 저전력의 인터폴레이션 필터를 설계하기 위하여 3개의 블록으로 나누어 설계한다. 먼저 1:4의 인터폴레이션 회로는 그림 1과 같이 익스펜더와 인터폴레이션 필터로 구성된다.

인터폴레이션 회로는 그림 1과 같이 제로 샘플을 삽입하는 익스펜더 연산을 수행한 후에 필터링해야 하지만 실제 회로설계에서는 따로 익스펜더 회로를 만들지 않고 필터링에서 제로를 삽입한 효과를 내도록 연산을 수행하게 된다. 본 논문에서는 1:4의 인터폴레이션과

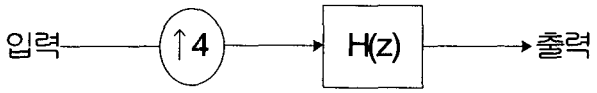

그림 1. 인터폴레이션 회로의 블록도(1:4)
Fig. 1. Interpolation block diagram(1:4).

40탭의 필터를 사용하여 저전력 구조를 설계하도록 한다. 인터폴레이션 회로의 최종 출력 y[m]은 1:4 인터폴레이션이므로 다음 식과 같이 4개의 출력의 합으로 나타낼 수 있다.

$$y[m]=g[4n]+g[4n+1]+g[4n+2]+g[4n+3] \quad (1)$$

식(1)에서 각각의 출력은 다음의 행렬식으로 나타낼 수 있다.

$$\begin{bmatrix} g[4n] \\ g[4n+1] \\ g[4n+2] \\ g[4n+3] \end{bmatrix} = \begin{bmatrix} h_0 & h_4 & h_8 & \cdots & h_{36} \\ h_1 & h_5 & h_9 & \cdots & h_{37} \\ h_2 & h_6 & h_{10} & \cdots & h_{38} \\ h_3 & h_7 & h_{11} & \cdots & h_{39} \end{bmatrix} \begin{bmatrix} x[n] \\ x[n-1] \\ x[n-2] \\ \vdots \\ x[n-9] \end{bmatrix} \quad (2)$$

식(1)과 (2)는 그림 2와 같이 4개의 필터와 1개의 MUX 회로로 구성하도록 한다.

그림 2의 4개의 필터 중에서 g[4n]을 만드는 저전력 구조를 설계하여 g[4n+1], g[4n+2], g[4n+3]의 필터도 그대로 적용하기로 한다. g[4n]의 필터는 10탭의 FIR 필터로 직접형이나 전치 직접형 등의 구조로 구현할 수 있으나, 본 논문에서는 입력신호의 계산결과를 공유하기 위하여 그림 3과 같은 전치직접형 구조를 사용한다.

그림 3에서 h_{4i}는 주어진 필터계수들이며, y_{4i}는 입력신호와 필터계수가 곱해진 값들을 나타낸다. 그림 3의 g[4n]에 대한 전치직접형 구조를 3 블록으로 나누어 저전력 구조를 설계하기로 한다. 첫 번째로 필터 계수의 곱셈연산을 덧셈연산으로 변환하기 위한 CSD 필터계수 블록의 구조를 제안한다. 본 논문의 g[4n]에서 사

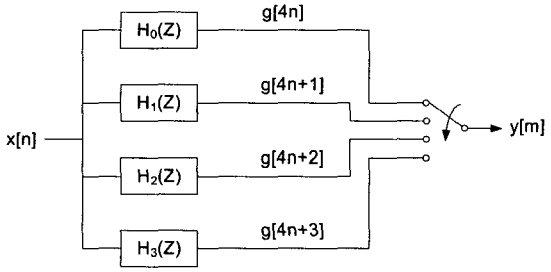

그림 2. 제안된 기본 인터폴레이션 회로의 구조 (1:4)
Fig. 2. Proposed basic interpolation diagram(1:4).

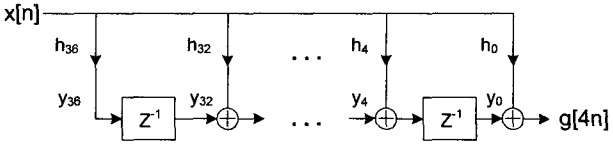

그림 3. g[4n]의 전치직접형 구조
Fig. 3. Transposed direct form structure for g[4n].

표 1. g[4n]필터의 16비트 CSD형 필터계수
Table 1. 16bit CSD form coefficients of g[4n] filter.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
h0											N					
h4								N			1				N	
h8							1			N				1		
h12						N			1							
h16					1					1		1		N		N
h20			1					N			1		1			1
h24						N		1		N					N	
h28							1		N				N			N
h32								N		1						N
h36									1		N					

표 2. 가능한 4비트 CSD형의 종류
Table 2. possible 4bit CSD types.

양수의 CSD형 4비트					음수의 CSD형 4비트				
1	2	3	4	값	1	2	3	4	값
1	0	1	0	1.25	N	0	N	0	-1.25
1	0	0	1	1.125	N	0	0	N	-1.125
1	0	0	0	1.0	N	0	0	0	-1.0
1	0	0	N	0.875	N	0	0	1	-0.875
1	0	N	0	0.75	N	0	1	0	-0.75
0	1	0	1	0.625	0	N	0	N	-0.625
0	1	0	0	0.5	0	N	0	0	-0.5
0	1	0	N	0.375	0	N	0	1	-0.375
0	0	1	0	0.25	0	0	N	0	-0.25
0	0	0	1	0.125	0	0	0	N	-0.125
0	0	0	0	0.0					

용된 필터계수는 다음과 같다.

$$\begin{array}{lll} h_0=-0.000999 & h_4=-0.003509 & h_8=0.006967 \\ h_{12}=-0.013675 & h_{16}=0.033544 & h_{20}=0.243441 \\ h_{24}=-0.025467 & h_{28}=0.011471 & h_{32}=-0.005904 \\ h_{36}=0.002957 & & \end{array} \quad (3)$$

식 (3)의 필터계수들을 덧셈 연산으로 구현하려면 2의 보수형이나 CSD형과 같은 이진수의 형으로 표현하여야 하는데, 상대적으로 1의 수가 적은 CSD 형을 선택하였다. 따라서 식 (3)의 계수들을 16비트의 CSD형으로 나타내면 표 1과 같다.

표 1에서 -1을 N으로 표기하였으며 0은 표기하지 않았다. 곱셈 연산은 입력신호 x[n]을 표 1의 1의 자리만큼 우로 쉬프트 시켜서 더해야하는데, 표 1의 2중 실선으로 표시된 것과 같이 한번에 4 비트씩 연산하는 구조를 제안한다. 즉, CSD형의 가능한 4비트의 종류를 모두 계산해 놓은 뒤에 순차적으로 더해서 최종 출력신호를 계산하는 방법이다. 표 1에서 보이는 2중 실선의 4비트의 연산을 연속적으로 수행하는 저전력 프로세서를 설계하기 위해서는 가능한 모든 CSD형의 4비트 종류를 하드웨어로 설계하여야 한다. CSD형의 4비트는 표 2와 같이 모두 21가지가 가능하다.

표 2에서 보듯이 4비트의 CSD형 표현은 1.25부터 -1.25까지 21가지 종류가 가능하다. 그러나 음수의 표현은 양수와 부호만 제외하면 같은 값이므로 하드웨어 구현에서는 따로 설계할 필요 없이 양수의 값을 그대로 사용하여 뺄셈으로 계산하면 된다. 따라서 0.125부터 1.25까지 총 10개의 종류가 가능하다. 입력신호 x[n]의 곱셈연산에 필요한 10개의 CSD형 계수의 곱셈연산은 다음 식과 같이 나타낼 수 있다.

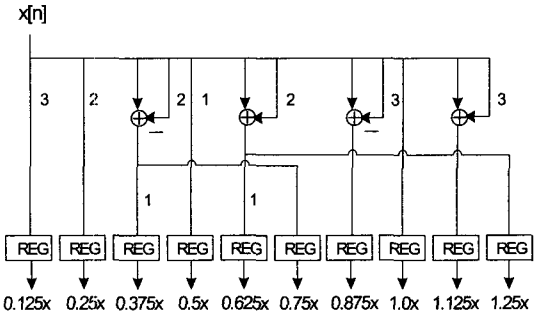

그림 4. 제안된 CSD 4비트 곱셈연산 구조
Fig. 4. Proposed CSD 4bit multiplication structure.

$$\begin{array}{ll} 1.25x = x + x\gg 2 & : (0번) \\ 1.125x = x + x\gg 3 & : (2번) \\ 1.0x = x & : (1번) \\ 0.875x = x - x\gg 3 & : (1번) \\ 0.75x = x - x\gg 2 & : (1번) \\ 0.625x = (1.25x)\gg 1 & : (2번) \\ 0.5x = x\gg 1 & : (5번) \\ 0.375x = (0.75x)\gg 1 & : (1번) \\ 0.25x = x\gg 2 & : (8번) \\ 0.125x = x\gg 3 & : (4번) \end{array} \quad (4)$$

식 (4)의 가능한 모든 연산 중에서, 표 1의 필터에서 실제로 사용되는 횟수를 식 (4)의 오른쪽 괄호에 표기하였다. 식 (4)의 괄호에서 보듯이 10개 중에는 사용되지 않는 종류가 1.25x의 1개가 있음을 알 수 있다. 제안된 식 (4)의 CSD 4비트 곱셈 구조는 그림 4와 같다.

2. 곱셈 연산용 프로세서 블록 설계

이 절에서는 곱셈을 덧셈 프로세서로 구현하는 블록의 설계방법을 제안한다. 그림 3에서 보듯이, x[n]의 입력신호는 각각 10개의 필터계수와 곱해진다. 따라서 그림 3의 곱셈 결과 값 y_{4i}는 식(4)를 이용하여 다음과 같이 쉬프트와 덧셈 연산으로 나타낼 수 있다.

$$\begin{array}{l} y_0 = -(0.25x)\gg 8 \\ y_4 = -(0.25x)\gg 12 - (0.875x)\gg 8 \\ y_8 = (0.5x)\gg 12 - (0.25x)\gg 8 + (0.125x)\gg 4 \\ y_{12} = (0.5x)\gg 8 - (0.25x)\gg 4 \\ y_{16} = -(0.625x)\gg 12 + (0.625x)\gg 8 + (0.5x)\gg 4 \\ y_{20} = (1.125x)\gg 12 + (0.25x)\gg 8 - (0.125x)\gg 4 + (0.25x) \\ y_{24} = -(0.25x)\gg 12 - (0.5x)\gg 8 - (0.375x)\gg 4 \\ y_{28} = -(1.125x)\gg 12 - (1.0x)\gg 8 + (0.25x)\gg 4 \\ y_{32} = -(0.125x)\gg 12 + (0.5x)\gg 8 - (0.125x)\gg 4 \\ y_{36} = (0.75x)\gg 8 \end{array} \quad (5)$$

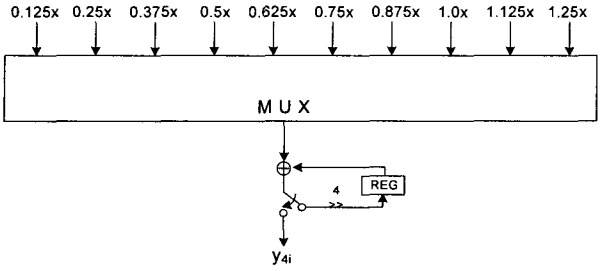

그림 5. 제안된 필터계수 곱셈연산 구조
Fig. 5. Proposed filter coefficient multiplication structure.

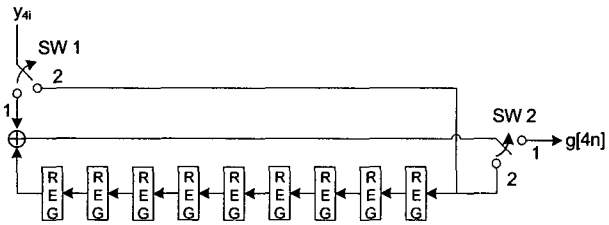

그림 6. 제안된 쉬프트 레지스터 블록의 구조
Fig. 6. Proposed shift register block structure.

식 (5)의 값들은 모두 그림 4에서 만들어진 10개의 값들의 쉬프트와 덧셈 연산으로 이루어진다. 따라서 식 (5)의 연산은 MUX 회로를 사용하여 이미 계산되어 있는 값들을 선택하도록 한 후에 덧셈과 쉬프트 연산으로 구현할 수 있다. 이와 같은 곱셈 값을 계산하는 구조는 그림 5와 같다.

그림 5에서 MUX의 select는 식 (5)의 순서로 이루어지도록 설계한다. 또한 스위치는 y_{4i}의 값들이 계산이 완료되면 아래 방향으로 보내도록 제어한다.

3. 쉬프트 레지스터 블록 설계

이 절에서는 출력용 쉬프트 레지스터 블록의 설계 방법을 제안한다. 그림 3의 레지스터 트레인은 9개의 덧셈기와 9개의 지연소자를 사용하고 있다. 그러나 본 논문의 그림 5 구조에서는 곱셈 값들이 순차적으로 출력되므로 1개의 덧셈기와 9개의 쉬프트 레지스터를 사용하는 그림 6의 구조를 제안한다.

그림 6은 10 cycle로 동작하도록 설계하였다. 첫 cycle에는 SW1과 SW2는 모두 1에 위치하여 위에서 내려온 y_0과 맨 왼쪽의 쉬프트 레지스터를 더하여 출력 g[4n]을 얻게 된다. 2 번째 cycle부터 9 번째 cycle까지는 SW1과 SW2는 각각 1과 2에 위치하도록 제어한다. 따라서 2 번째 cycle부터 9 번째 cycle 까지 위에서 내려온 y_{4i}의 값과 쉬프트 레지스터 값을 더하여 맨 오른쪽의 쉬프트 레지스터로 들여보낸다. 마지막으로 10

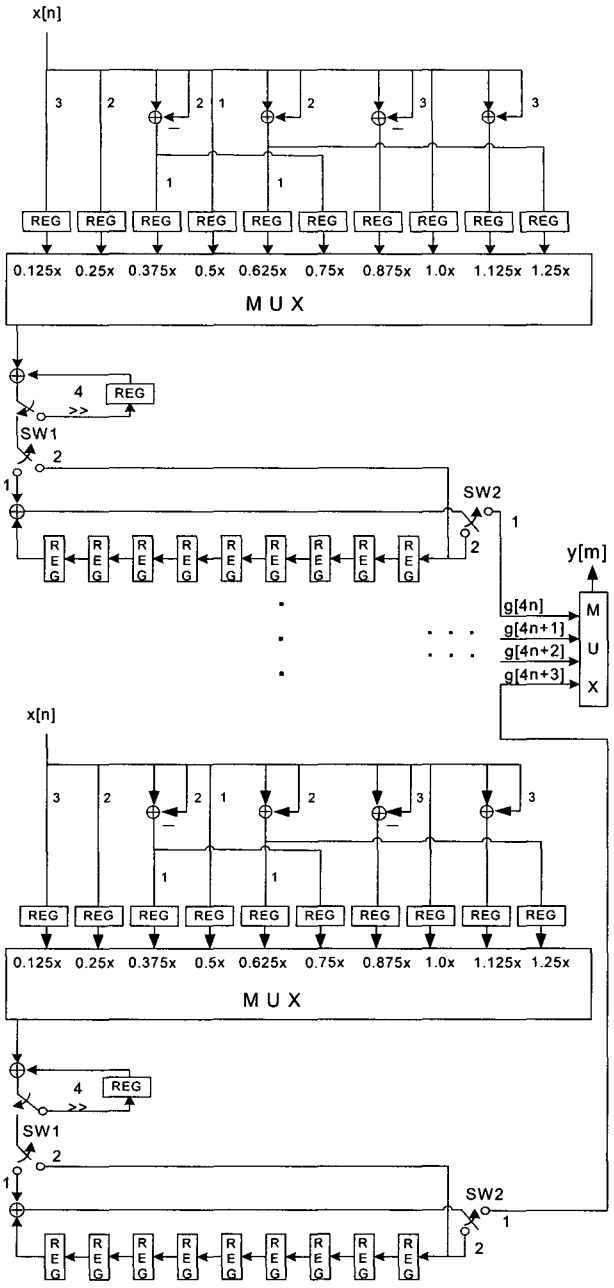

그림 7. 제안된 저전력 인터폴레이션 필터 구조
Fig. 7. Proposed interpolation block diagram of low voltage.

번째 cycle에서는 SW1과 SW2는 각각 2와 1에 위치하도록 제어한다. 따라서 y_{36}의 값은 곧바로 맨 우측의 쉬프트 레지스터로 입력된다. 이와 같은 방법으로 계산이 완료되면 모든 쉬프트 레지스터의 값들이 새로 계산된 값들로 바뀌게 된다. 지금까지 제안된 그림 4, 5, 6의 구조를 연결한 전체 구조는 그림 7과 같다.

그림 7에서 보듯이 입력신호 x[n]은 쉬프트와 덧셈기를 사용하여 0.125x[n]부터 1.25x[n]까지 10가지 종류가 만들어진 후에 레지스터에 각각 저장된다. 실질적인 연

산은 표1의 h_0의 LSB 4비트부터 시작된다. 식(5)의 10개의 필터계수 곱셈을 모두 수행하려면 덧셈의 항의 수가 25개이므로 25 clock이 필요하게 된다. 모든 종류의 값들은 미리 계산되어 있으므로 매 clock 마다 MUX를 통하여 선택하며, 완성된 y_{4i} 값들은 g[4n]과 쉬프트 레지스터 입력 값을 계산하는데 사용된다.

Ⅲ. 실험 및 고찰

본 논문에서 제안한 필터의 성능을 평가하기 위해 곱셈기를 사용하는 필터의 구조와 게이트 수를 비교하였다. 제안된 구조의 동작검증을 위해 Matlab과 C 언어를 사용하여 출력을 확인하였다. 시뮬레이션에 사용된 입력 신호는 0.5에서 -0.5까지 cosine 신호를 샘플링한 20개를 사용하였으며, 사용된 입출력 및 필터계수의 정세도는 표 3과 같다. 20개의 입력이 1:4 익스펜더를 거치면 80개의 신호가 되고, 다시 40탭의 필터를 통과하면 119개의 출력이 생성된다. 119개의 출력 중에서 y[40]부터 y[51]까지 12개의 값을 표4에 나타내었다. FPGA 구현을 위하여 Xilinx사의 ISE 6.3i를 이용하여 Verilog-HDL 코딩을 수행하였으며, Verilog-HDL 코딩의 결과가 Matlab의 시뮬레이션 결과와 같은지 확인하였다. 표 3의 사양대로 시뮬레이션한 Verilog-HDL 코딩 결과를 그림 8에 나타내었다. 그림 8에서 보듯이. 17.7us부터 y[40]이 출력됨을 볼 수 있으며, 22.1us에 y[51]이 출력되고 있음을 알 수 있다. 이 값들은 표 4에 주어진 Matlab 결과 값과 일치함을 확인하였다.

그림 8과 같이, Verilog-HDL 코딩을 통하여 구조의 동작 검증을 수행하였으며, 게이트 수를 시뮬레이션 하였다. 기존 구조와의 게이트 수를 비교하기 위하여 그림 3의 곱셈기 구조를 기존 구조로 선택하였다. 즉, 그림 3은 g[4n]의 구조이므로 이와 똑같은 g[4n+1], g[4n+2], g[4n+3]의 구조를 만들어서 4개의 구조를 결합하여 기존의 곱셈기 구조를 제작하였다. 이와 같이 만들어진 기존의 구조와 그림 7의 제안된 구조를 Xilinx사의 ISE 6.3i를 이용하여 Verilog-HDL 코딩을 수행하

표 3. simulation 사양
Table 3. simulation conditions.

	제안구조	기존 구조
입력신호	20개	
필터 계수	40tap	
입력 비트수	16비트	
필터계수 비트수	16비트	
출력 비트수	16비트	
인터폴레이션비	4	

표 4. 출력 y[m]의 matlab simulation 결과
Table 4. matlab simulation result of y[m].

y[m]	Integer	Binary
y[40]	-0.000200	1111111111111001
y[41]	0.006586	0000000011010111
y[42]	0.010944	0000000101100110
y[43]	0.012930	0000000110100111
y[44]	0.017204	0000001000110011
y[45]	0.023531	0000001100000011
y[46]	0.026390	0000001101100000
y[47]	0.028364	0000001110100001
y[48]	0.034025	0000010001011010
y[49]	0.043375	0000010110001101
y[50]	0.051416	0000011010010100
y[51]	0.058559	0000011101111110

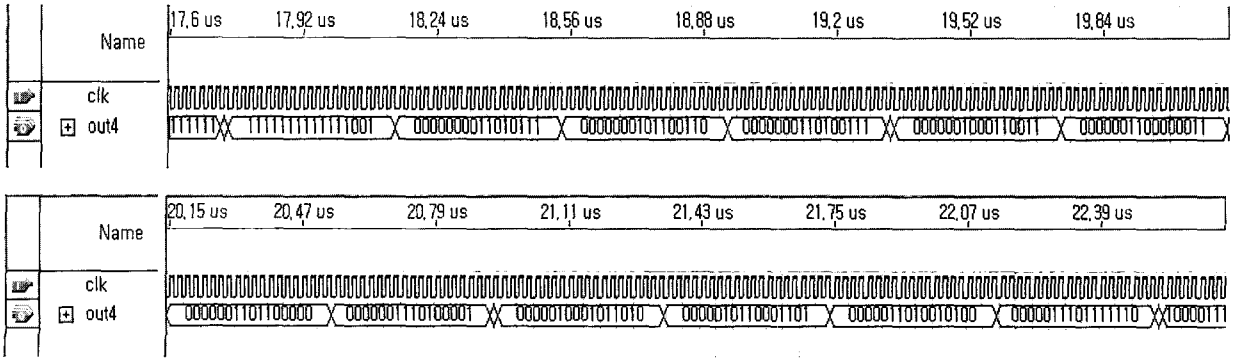

그림 8. y[m]의 Verilog-HDL simulation 결과
Fig. 8. Verilog-HDL simulation result of y[m].

표 5. 제안된 구조의 cell usage와 게이트 수 비교
Table 5. Cell usage and gate count for proposed structure.

	제안구조	기존 구조
곱셈기 수	0	40
덧셈기 수	24	36
지연소자 수	0	36
쉬프트레지스터 수	36	0
게이트 수	18,916 (31.57%)	59,925
cell usage	3,221	10,995

였다. 코딩 결과 제안 구조와 기존 곱셈기 구조의 사용된 소자의 수와 게이트 수 비교는 표 5와 같다.

표 5에서 보듯이 기존의 곱셈기 필터 구조는 곱셈 연산에 있어서 곱셈기를 10개 사용하므로 FPGA로 구현했을 때 사용된 게이트 수는 59,925개이다. 반면에 본 논문에서 제안한 구조는 18,916개의 게이트로 구현되었다. 제안 구조의 게이트 수는 기존 곱셈기 필터구조에 사용된 게이트 수와 비교하여 68.43%를 감소시킬 수 있었다.

Ⅳ. 결 론

이 논문에서는 IMT-2000용 인터폴레이션 필터의 고속/저전력 프로세서 구조를 제안하였다. CSD형으로 필터 계수를 표현하여 덧셈의 수를 최소화할 수 있었으며, 4비트씩 계산된 중간결과를 한 개의 덧셈 연산 프로세서를 사용하여 구현함으로서 구현면적을 현저히 줄일 수 있었다. 이와 더불어 기존 곱셈기 구조에서 사용되는 출력용 플립플롭 9개와 덧셈기 9개는 1개의 덧셈기와 쉬프트 레지스터를 사용하여 효율적으로 구현될 수 있음을 보였다. 구현면적의 감소효과를 알아보기 위하여 IMT-2000에서 사용되는 40탭 인터폴레이션 필터를 기존 곱셈기 구조와 제안된 구조를 사용하여 Verilog-HDL 코딩을 수행하였다. Verilog-HDL 코딩으로 구현한 결과 68.43%의 게이트 수의 감소를 달성할 수 있었다. 따라서 제안된 덧셈 프로세서 구조는 IMT-2000의 인터폴레이션 필터로서 널리 사용될 수 있을 것이다.

참 고 문 헌

[1] R.W.Reitwiesner, "Binary arithmetic," in Advances in Computers, New York: Academic, vol. 1, pp. 231-308, 1966.

[2] K. Hwang, Computer Arithmetic: Principles, Architecture, and Design, New York: Wiley, 1979.

[3] 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.

[4] 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.

[5] S.A.White, "Applications of distributed arithmetic to digital signal processing: A tutorial review," IEEE ASSP Magazine, pp. 4-19, July 1989.

[6] A. Sinha and M. Mehendale, "Improving area efficiency of FIR filters implemented using distributed arithmetic," Proc. Eleventh Interna -tional Conference on VLSI Design, pp. 104-109, 1998.

[7] 임인기, 정희범, 김경수, 김환우, "IMT-2000 시스템을 위한 승산기를 사용하지 않는 인터폴레이션 FIR 필터 구현," 한국통신학회논문지, '02-10 Vol.27 No.10C pp.1008-1014, Oct. 2002.

저 자 소 개

장 영 범(정회원)
1981년 연세대학교 전기공학과 졸업(공학사)
1990년 Polytechnic University 대학원졸업(공학석사)
1994년 Polytechnic University 대학원졸업(공학박사)
1981년~1999년 삼성전자 System LSI 사업부 수석연구원
2000년~2002년 이화여자대학교 정보통신학과 연구교수
2002년~현재 상명대학교 정보통신공학전공 교수
<주관심분야: 통신신호처리, 음성/오디오 신호처리, 통신신호처리용 SoC 설계>

문 종 범(정회원)
2005년 2월 상명대학교 정보통신 공학 졸업 예정(공학사)
<주관심분야: 통신신호처리용 SoC 설계>

이 현 정(정회원)
2005년 2월 상명대학교 정보통신 공학 졸업 예정(공학사)
<주관심분야: 통신신호처리용 SoC 설계>

이 원 상(정회원)
2004년 2월 상명대학교 컴퓨터 시스템 공학 졸업 (공학사)
2004년 2월~현재 상명대학교 대학원 컴퓨터·정보·통신공학과 석사과정
<주관심분야: 통신신호처리, 통신신호처리용 SoC 설계>