비트 분할 데이터 시프트 및 다양한 형식 변환이 가능한 데이터 처리기의 VLSI 설계

정회원 유 재 희*

VLSI Design of Data Manipulation Unit capable of bit partitioned shifts and various data type conversions

Jae-Hee You* *Regular Member*

요 약

일반적인 시프트 연산과 더불어, 비트 분할 시프트 및 멀티미디어 데이터의 다양한 형식변환이 가능한 데이터 처리기가 제안되었다. 데이터 형식 변환 연산과 시프트 연산의 유사점을 최대한 이용하여, Barrel 시프터를 변형하여, 약간의 interconnection을 추가함으로써, 최소의 하드웨어로써 두 개의 연산을 통합 처리 가능하도록 하였다. 제안된 데이터 처리기는 크게 일반적인 시프트 연산과 pack 연산을 수행하는 시프터 블록과 unpack 연산 등을 수행하는 블록으로 구성된다. 제안된 데이터 처리기는 Verilog HDL를 사용하여 설계되었으며, Compass 0.6㎛ standard cell library를 사용하여 VLSI 구현된 결과에 대하여 논의된다.

ABSTRACT

A data manipulation unit capable of bit partitioned shift and various multimedia data type conversions in addition to conventional shift, is presented. Utilizing the similarity between the data type conversion and the shift, the addition of small amount of interconnections to conventional barrel shifter enables data type conversion as well as shift operations with minimal hardware overhead. The presented data manipulation unit is composed of the shifter block for conventional shift and a pack and a unpack block. It has been designed with verilog HDL and the VLSI implementation results using compass 0.6 um standard cell are discussed.

I. 서 론

화상, 음성을 포함하는 멀티미디어 데이터의 중요성이 부각되고 있는 최근의 정보통신의 흐름에 따라 이를 수행하기 위한 효율적인 하드웨어 구현 방법에 대한 연구가 활발히 진행되고 있다. 특히, 멀티미디어 데이터는 8, 16비트로 구성되어 있으나, 프로세서 연산 유닛의 데이터 폭은 32 또는 64비트로 이루어져 있어, 이러한 데이터 폭의 차이는 하드웨어의 효율성 및 연산성능을 저하시키게 되어, 데이터 형식의 변환 처리가 매우 중요하다. [1, 2]에서는 이러한 데이터 형식변환 연산을 위해 프로세서 연산 유닛의 덧셈기 또는 승산기를 사용하게 되어, 동시에 덧셈 또는 곱셈 명령어 수행이 불가능하며, 이를 해결하기 위해서는 추가의 하드웨어가 필요하게 된다. [2]에서는, 이러한 데이터 처리를 위해, 많은 양의 추가 하드웨어를 설계하였으며, 이는 멀티미디어 데이터가 아닌 일반 데이터 처리시 하드웨어 효용도를 크게 저하시키게 된다. 따라서 본 논문에서는 데이터 처리연산이 일반적인 시프트연산

* 홍익대학교 전자전기 공학부
논문번호: 010076-0426, 접수일자: 2001년 4월 26일
※ 본 연구는 정보통신부 대학기초연구사업 출연과제 "화상 및 음성정보처리에 우수한 Signal Processor Execution unit 설계" (과제번호 : 96188-BT-11) 의 지원을 받아 수행되었음

과 유사점이 많은 점을 최대한 이용하여, 시프터를 변형하고, pack연산을 위한 interconnection과 unpack, expand, merge 연산 등을 통합 처리하기 위해 최종단에 적은 양의 하드웨어를 추가함으로써 시프트 및 다양한 데이터 형식 변환연산을 통합 수행할 수 있는 데이터 처리기의 구현 방안을 제안하였다. Ⅱ장에서는, 데이터 형식 변환 명령어 및 수행방법에 대해 기술되었고, Ⅲ장에서는 시프트 및 형식 변환 연산의 통합처리 방안에 대해 기술되었다. Ⅳ장에서는 데이터 처리기의 VLSI 아키텍쳐에 대하여, Ⅴ장에서는 VLSI 설계결과에 대하여 논의된다.

Ⅱ. 데이터 형식변환 연산

화상 멀티미디어 데이터 중에서 많은 부분을 차지하는 화소는 unsigned 8비트 데이터로 구성된다. 이를 처리하기 위해서는, 32 비트의 워드단위 데이터 중에서 특정 위치의 8 또는 16비트를 지정된 비트 위치로 이동하여 추출한 후, 같은 방법에 의하여 추출된 또 다른 8비트의 데이터들을 조합하여 새로운 32비트의 워드 데이터를 생성하는 pack연산과 pack에 의하여 생성된 32비트 데이터를 풀어 2개의 32비트 데이터를 생성하는 unpack연산 등이 필요하다. 이와 같은 데이터 형식변환 연산을 수행하기 위하여 UltraSPARC VIS[1]에서는 표 1과 같은 명령어들을 정의하고 있다. 표 1에 나타낸 pack연산 중의 하나인 PACKFIX는 그림 1에 나타낸바와 같이 2개의 32비트 데이터 각각에서 4비트로 구성된 scale_factor에 따라, 시프트를 한 후 2개의 16비트를 추출하여 32비트의 데이터를 생성하는 연산이다. 즉, rs1과 rs2부분 데이터를 scale_factor만큼 logical left shift를 한 후 16~31비트 사이의 16비트를 추출하여 rd의 0~15비트와 16~31비트에 각각 위치시킨다.

표 1. 데이터 형식변환 명령어.

Instruction	동작
PACK16	Four 16-bit packs
PACK32	Two 32-bit packs
PACKFIX	Four 16-bit packs
EXPAND	Four 16-bit expands
MERGE	Two 32-bit merges

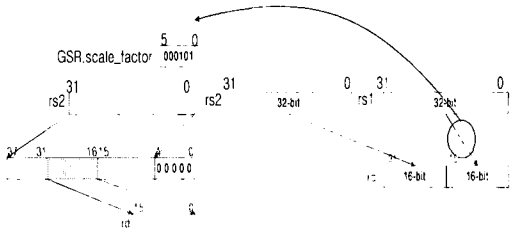

그림 1. PACKFIX의 연산방법.

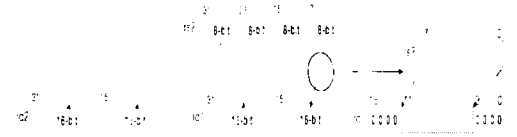

그림 2. EXPAND의 연산방법.

EXPAND는 그림 2에 나타낸 바와 같이 1개의 32비트 데이터를 구성하는 4개의 8비트 데이터 각각을 4개의 16비트 데이터로 확장하여 최종적으로 1개의 64비트 데이터를 생성하는 연산이다.

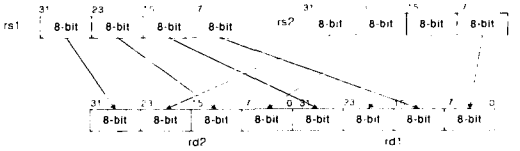

그림 3. MERGE의 연산방법.

rs2의 4개의 8비트 데이터를 rd1, 2 각각의 0~15, 16~31비트 위치에 있는 총 4개의 16비트 데이터에, 4~11, 20~27비트에 lsb를 위치시킨다. MERGE는 그림 3과 같이 2개의 32비트 데이터를 구성하는 8개의 8비트 데이터에 대해 교체 연산을 통해 새로운 8개의 8비트 데이터로 구성된 64비트 데이터를 생성한다.

Ⅲ. 데이터 시프트와 형식 변환 연산의 통합 처리 방안

기존의 32비트 워드단위 시프트와 더불어, 8, 16, 32, 64 비트 단위의 다양한 시프트 및 형식변환 연산을 처리하기 위한 데이터 처리기의 구조를 그림 4에 나타내었다. 이미 설명된 바와 같이, 데이터 형식 변환 연산은 특정한 자리 위치에 있는 데이터의 추출, 0 padding, 2개의 32비트 워드단위 데이터로

부터 데이터 처리를 거쳐 64비트 더블워드 단위의 데이터 생성 등이 요구된다. 그림 4의 데이터 처리기는, 두 개의 독립적인 32비트 연산 유닛이 프로세서 안에 있을 때, 각각의 연산유닛에 시프터가 있을 경우를 나타낸다. 64비트 단위 더블워드 데이터 생성을 위해서, 32비트 시프터 2개가 필요하며, EXPAND, MERGE 등의 수행을 위하여, 두 유닛 간의 연결이 필요하다. 따라서, 하위 시프터의 msb와 상위 32비트 시프터의 lsb간의 연결 경로를 구현하였다. 이와 같은 제안된 시프터간의 연결구조를 통하여, 일반적인 더블워드 64 비트 단위의 시프트를 처리 가능하다. 이를 위해 각각의 시프터는 그림 4 의 MUX-JOIN 하드웨어에 의하여, 64비트 데이터 처리기로 전환 가능하도록 설계되었다. 두 개 이상의 연산 유닛의 경우에 있어서도, 동일한 개념을 확장 적용 가능하다.

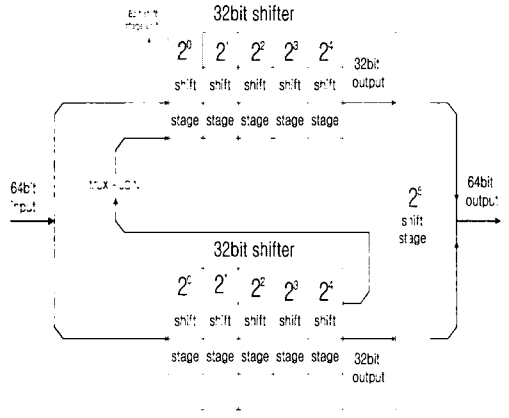

그림 4. 제안된 데이터 처리기의 구조

일반적인 시프트 연산에 있어서는, 두 개의 상위와 하위 32비트 barrel 시프터를 바탕으로 독립적인 두 개의 32비트 데이터에 대해 독립적인 시프트 연산을 수행할 수 있도록 하였다. Byte, Halfword 등의, 데이터는 각각 2^0, 2^1, 2^2 스테이지, 2^0, 2^1, 2^2, 2^3 스테이지까지의 시프트 연산을 수행하고 각각 2^3, 2^4 스테이지는 bypass하여 수행 가능하다. 이 경우, left 시프트, right 시프트 등의 상, 하위 비트의 빈자리는 적절한 MUX에 의해 각 스테이지에 0 또는 입력 데이터 비트를 입력하여 처리 가능하며, 64비트 더블워드 단위 시프트는 그림 4의 MUX-JOIN에 의해, 0 대신 하위 32비트 시프터 입력을 입력시켜 처리한다. 표 2는 데이터처리기가 지원하는 연산을 나타낸다. 데이터 형식 변환과 시프트 연산을 통합 처리하기 위한 연산 과정중 데이터 추출 및 자리이동은 scale-factor에 따라 워드단위 데이터를 시프트 시켜 데이터를 추출하여 수행가능하며, 0 padding은 시프트 처리된 최종 데이터에 0을 추가하여 생성 가능하다. 제안된 64비트 데이터 처리기를 이용하여, 그림 5에 나타낸 바와 같이 두 개의 연속된 32비트 데이터로 이루어지는 64비트 데이터 내의 정렬되어 있지 않은 임의의 위치의 연속된 32비트에 대한 데이터 추출을 수행하는 PACK64를 용이하게 구현할 수 있도록 하였다. PACK64를 위해서는 1~31비트의 시프트가 요구되므로 5비트의 pack_factor를 사용하였다.

표 2. 데이터처리기가 지원하는 연산

구 분	기 능	동 작
데이터 형식변환 연산	PACK16	Four 16-bit packs
	PACK32	Two 32-bit packs
	PACK64	One 64-bit pack
	PACKFIX	Four 16-bit packs
	EXPAND	Four 16-bit expands
	MERGE	Two 32-bit merges
일반적인 시프트 연산	SLL	Logical left shift
	SRL	Logical right shift
	SRA	5Arithmetic right shift

PACK64의 연산방법은 두 개의 32비트, 총 64비트 입력 데이터를 그림 4에 나타낸 연결구조를 바탕으로 pack_factor만큼 logical left 시프트를 한 후에 상위 32비트(32 - 64비트 위치)를 추출하게 된다. 앞에서 설명된 PACK16, 32, FIX연산에서는 pack_factor의 최상위 비트를 '0'으로 설정하고 하위 4비트만을 사용하였다. 각각의 32비트 데이터 처리기가 일반적인 시프트연산과 더불어, PACK16, PACK32, PACFIX 등의 다양한 형식 변환 연산을 처리 가능하게 하기 위해, 각 연산에 필요한 데이터 이동경로를 분석하여 보면 다음과 같다. 데이터 처리기에 입력되는 두 개의 32비트 데이터는 식(1)과 같이 표현할 수 있다.

$$X = \sum_{i=0}^{31} 2^i x[i] \quad (0 \le i \le 31, x[i] \in \{0,1\}) \quad (1)$$

여기서, x[i]는 입력 데이터를 나타낸다. 기존의 일반적인 시프트연산에 있어, 입력된 데이터가 시프트 하여야 할 거리는 일반적으로 Barrel 시프터에서

식(2)와 같이 표현할 수 있다.

$$\text{shift distance} = n0 \cdot 2^0 + n1 \cdot 2^1 + n2 \cdot 2^2 + n3 \cdot 2^3 + n4 \cdot 2^4 + n5 \cdot 2^5 \quad (n0, n1, n2, n3, n4, n5 \in \{0,1\}) \quad (2)$$

여기서, n0, n1, n2, n3, n4, n5는 각각 Barrel 시프터 스테이지 1, 2, 3, 4, 5에 있어서 시프트 여부를 나타낸다. 이미 설명된 바 있는 PACK16, PACK32, PACK64, PACKFIX 각 경우에 있어서, 64비트 입력 (x[i])을 통해 32비트 출력 (y[i])를 산출하기 위하여 요구되는 연산은 각각 식(3 - 6)과 같이 표현될 수 있다.

$$\begin{aligned}
&y[i]\ (24 \le i \le 31) = x[i-(n0 \cdot 2^0 + n1 \cdot 2^1 + n2 \cdot 2^2 + n3 \cdot 2^3 + n4 \cdot 2^4) + 2^0] \\
&y[i]\ (16 \le i \le 23) = x[i-(n0 \cdot 2^0 + n1 \cdot 2^1 + n2 \cdot 2^2 + n3 \cdot 2^3 + n4 \cdot 2^4) + (2^0 + 2^3)] \\
&y[i]\ (8 \le i \le 15) = x[(i+2^6)-(n0 \cdot 2^0 + n1 \cdot 2^1 + n2 \cdot 2^2 + n3 \cdot 2^3 + n4 \cdot 2^4) + (2^0 + 2^4)] \\
&y[i](0 \le i \le 7) = x[(i+2^6)-(n0 \cdot 2^0 + n1 \cdot 2^1 + n2 \cdot 2^2 + n3 \cdot 2^3 + n4 \cdot 2^4) + (2^0 + 2^3 + 2^4)]
\end{aligned} \quad (3)$$

$$\begin{aligned}
&y[i]\ (8 \le i \le 31) = x[i-(2^3)] \\
&y[i]\ (0 \le i \le 7) = x[(i+2^6)-(n0 \cdot 2^0 + n1 \cdot 2^1 + n2 \cdot 2^2 + n3 \cdot 2^3 + n4 \cdot 2^4) + (2^0 + 2^3 + 2^4)]
\end{aligned} \quad (4)$$

$$\begin{aligned}
&y[i]\ (32 \le i \le 63) = x[i-(n0 \cdot 2^0 + n1 \cdot 2^1 + n2 \cdot 2^2 + n3 \cdot 2^3 + n4 \cdot 2^4)] \\
&y[i]\ (0 \le i \le 31) = x[(i+2^6)-(n0 \cdot 2^0 + n1 \cdot 2^1 + n2 \cdot 2^2 + n3 \cdot 2^3 + n4 \cdot 2^4)]
\end{aligned} \quad (5)$$

$$\begin{aligned}
&y[i]\ (32 \le i \le 63) = x[i-(n0 \cdot 2^0 + n1 \cdot 2^1 + n2 \cdot 2^2 + n3 \cdot 2^3 + n4 \cdot 2^4)] \\
&y[i]\ (0 \le i \le 31) = x[(i+2^6)-(n0 \cdot 2^0 + n1 \cdot 2^1 + n2 \cdot 2^2 + n3 \cdot 2^3 + n4 \cdot 2^4) + 2^4]
\end{aligned} \quad (6)$$

식(3 - 6)에서 나타낸 바와 같이 위치별로 분할된 각 데이터 변환 처리연산은 많은 부분이 기존의 시프트 연산인 식(2)와 유사한 형태를 가지고 있음을 알 수 있다. 이와 같은 공통점을 최대한 활용하고, 데이터 형식변환을 위한 최소의 추가 시프트 경로를 살펴보면 다음과 같다. 식(3 - 6)에서 $(i + 2^6)$ 항은 그림 4에 나타낸 MUX-JOIN에 의하여 구현 가능하며, PACK64에서는 충돌문제가 발생하지 않는다. 식(3)에 나타낸 PACK16연산은 입력된 4개의 16비트에서 4개의 8비트를 추출하는 연산으로, pack_factor에 의해 결정되는 n0 ~ n4에 의해 기존의 $2^0 \sim 2^3$ 시프트 스테이지를 이용하여 시프트 연산을 수행한다. 그러나 식 (3)에 나타낸 바와 같이, 이 연산 이외로 2^0, $2^0 + 2^3$, $2^0 + 2^4$, $2^0 + 2^3 + 2^4$만큼의 시프트 연산은 설명된 pack_factor에 의한 연산과 시프트경로에서 충돌이 발생한다. 즉, 입력 데이터의 비트 위치 $(63 \le i \le 0)$중 네 번째 16비트 $(63 \le i \le 48)$는 pack_factor만큼 시프트를 수행할 경우 0 ~ 15비트 시프트 연산이 가능하고 추가로 2^0만큼의 시프트를 수행하여야 하므로 총 1 ~ 16비트 범위의 시프트를 수행하여야 한다. 또한, 세 번째 16비트 $(47 \le i \le 32)$, 두 번째 16비트 $(31 \le i \le 16)$, 첫 번째 16비트 $(15 \le i \le 0)$는 각각 $2^0 + 2^3$, $2^0 + 2^4$, $2^0 + 2^3 + 2^4$ 시프트를 추가로 요구하므로 각각 9 ~ 24비트, 17 ~ 32비트, 25 ~ 40비트의 시프트 연산이 수행될 필요가 있다. 즉, 네 개의 16비트 데이터 블록은 서로 독립적인 시프트를 필요로 하므로 두 번째 16비트의 시프트 범위중 16 ~ 24비트만큼의 시프트 연산은 첫 번째 16비트의 시프트 연산경로와 충돌이 발생한다. 유사하게 다른 16비트 데이터의 시프트간에도 충돌이 발생하게 되어, 이를 위한 추가의 시프트 경로가 설계되었다. PACK32와 PACKFIX연산의 경우, 유사한 시프트 경로 충돌문제가 발생하게 되어, 이를 위한 추가 경로가 설계되었다. 64비트 데이터에서 32비트 데이터를 정해진 위치에서 추출할 경우, 경로의 충돌 문제점을 해결하기 위하여 다음의 두 가지 방법이 고려되었다. 첫 번째로 2^4 과 2^5 시프트 스테이지사이에 앞서 설명된 2^0, $2^0 + 2^3$, $2^0 + 2^4$, $2^0 + 2^3 + 2^4$를 위한 경로를 추가하는 방법이다. 두 번째로 충돌이 일어나는 경로를 시프트 스테이지마다 별도의 경로를 추가하여 해결하는 방안이다. 첫 번째 방법의 장점은 규칙성을 유지할 수 있고, 제어가 용이하다는 장점이 있으나 추가 시프트 경로가 증가하는 단점이 있다. 두 번째 방법은 추가 경로의 양은 적으나, 각 시프트 스테이지의 제어를 위한 로직이 비교 적 복잡하다는 문제점이 있다. 동일한 모듈의 반복 및 interconnection의 규칙성을 유지하기 위하여, 첫 번째 방법을 사용하였다. 이 경우 추가되는 interconnection의 전체 양은 적으며 이에 대하여서는 Ⅳ에서 논의된다. 그림 6은 PACK16 연산수 행 방안을 나타낸다. 32비트 데이

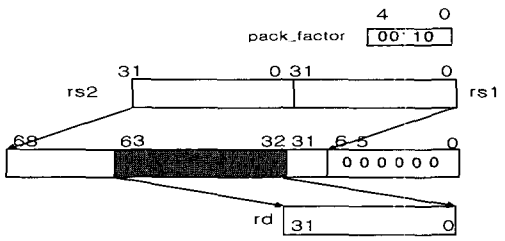

그림 5. PACK64 연산방법

터 2개에 있어 앞서 설명된 총 4개의 16비트(H16 ~ H31, H0 ~ H15와 L16 ~ L31, L0 ~ L15)데이터가 시프터로 입력된 후, 이미 설명된 바와 같이 pack_factor 하위 4비트만을 사용하여 2^0 ~ 2^4 의 시프트 스테이지를 통하여 1 ~ 15비트까지 시프트를 수행한다. 이후 각각의 16비트 데이터에 대한 데이터 추출 최하위 비트 위치인 2^4 시프트 stage의 H23 ~ H30, H7 ~ H14와 L23 ~ L30, L7 ~ L14비트 위치의 데이터를 추가로 설계된 interconnection을 통해, 2^5 시프트 stage의 H24 ~ H31, H16 ~ H23, H8 ~ H15, H0 ~ H7의 비트 위치 출력하여 32비트 결과데이터를 생성하게 된다.

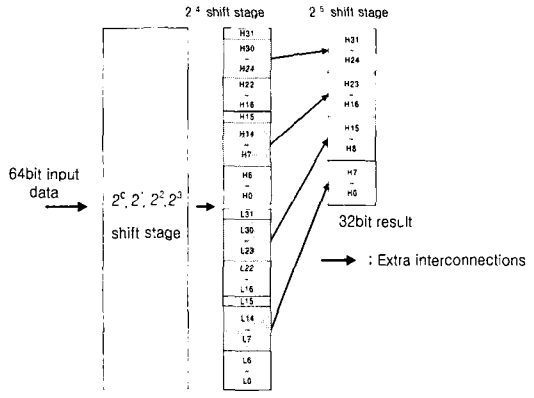

그림 6. PACK16을 위한 추가 연결구조

Unpack연산은 Ⅲ에서 설명한 바와 같이 EXPAND와 MERGE 연산이 있다. EXPAND와 MERGE 연산은 시프트 연산을 요구하지 않고 교체 연산을 필요로 하므로 각 시프트 스테이지에 있어, 시프트 연산 없이 통과시킨 후 시프터의 맨 마지막 스테이지에 새로이 추가된 MUX로 구성된 블록을 통하여 처리하도록 하였다. 이 블록에 입력된 데이터는 그림 7에 나타낸 바와 같이 EXPAND 또는 MERGE에 의해 생성된 MUX 제어신호에 따라 교체 연산을 수행하고 EXPAND의 경우에는 0 padding이 수행된다.

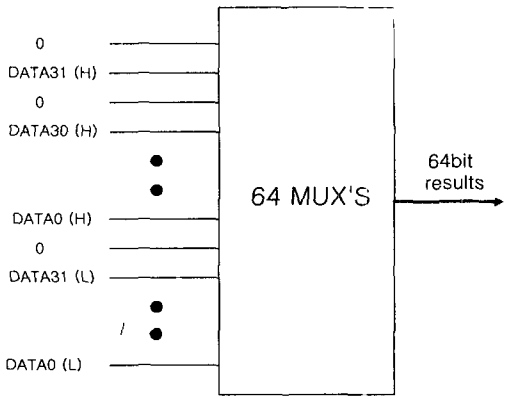

그림 7. EXPAND, MERGE 처리방안

Ⅳ. VLSI 아키텍처

데이터 처리기의 세부아키텍처는 그림 8에 나타낸 바와 같이 기존의 시프트 연산과 데이터 형식 변환연산 등을 위한 2^0 ~ 2^5 시프트 스테이지로 구성된다. 이 시프트 스테이지는 left와 right 시프터로 이루어진 32비트 단위의 barrel 시프터 2개 (Shift 32 high, shift 32 low)로 이루어져 있다. 또한 각각에 있어 최종 스테이지에 EXPAND, MERGE 등을 위한 블록, 그리고 barrel 시프터와 EXPAND, MERGE등 출력 중에 최종출력을 선택하는 MUX가 구현되어 있다. Shift 32 high, shift 32 low에는 Ⅲ에서 설명된 바와 같이 형식 변환을 위한 추가의 interconnection이 설계 되었다. 각각의 기존 시프트 수행 시에는 그림 4 에 나타낸 바와 같이 MUX에 의해, left 시프트를 위해 사용되는 '0'을 shift

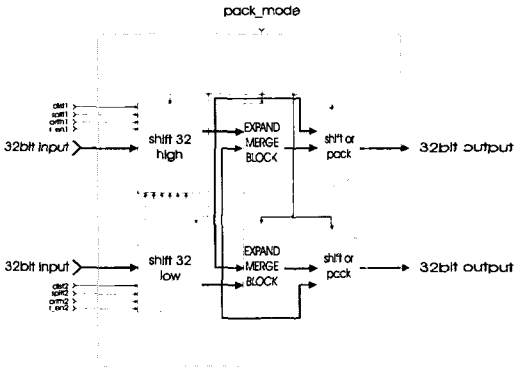

그림 8. 데이터 형식 변환 및 시프터의 구조도

32 high의 최하위 비트로 입력하여, shift 32 high와 shift 32 low가 분리되어 두 개의 독립된 시프트가 수행된다.

데이터 형식 변환 및 64비트 데이터의 기존 시프트는 shift 32 low의 상위 비트가 shift 32 high의 하위 비트 MUX에 의해 연산별로 선택된 데이터의 입력을 통하여 수행된다. 32비트 시프터인 shift 32 high와 shift 32 low는 각각 8비트 단위로 비트 분할하여 4개의 8비트, 2개의 16비트, 1개의 32비트 데이터가 시프트를 수행하게 된다. 데이터 형식변환 연산 중 pack연산은 shift 32 high, shift 32 low 연산을 수행하고, EXPAND, MERGE등도 shift 32 high, shift 32 low를 bypass 하여 EXPAND, MERGE블록에서 데이터 추가 또는 자릿수 이동 등을 통하여 수행된다. 64비트 데이터 처리를 위해, shift 32 low의 최상위 비트들을 shift 32 high의 최하위 비트들로 입력시키기 위한 그림 4의 MUX-JOIN에 해당되는 연결구조를 단지 left shift의 경우에 있어 그림 9에 나타내었다. shift 32 high의 최하위 입력데이터인 DATA0(H)와 shift 32 low의 최상위 데이터인 DATA31(L)가 제어신호(join)에 의하여 2^0 시프트 스테이지의 최하위 MUX인 H0에 의해 선택된다. 동일한 방법으로 shift 32 high의 2^0 시프트 스테이지 H0과 H1 MUX의 출력 데이터와 하위 32비트 시프터의 2^0 시프트 스테이지 L31과 L30 MUX의 출력 데이터가 join신호에 의하여 상위 32비트 시프터의 2^1 시프트 스테이지의 최하위 MUX인 H0과 H1로 입력이 된다. 이후 2^2, 2^3, 2^4 시프트 스테이지도 같은 방법에 의하여 두 개의 시프터 블록을 연결한다. 2^5 시프트 스테이지는 shift32 low의 2^4 시프트 스테이지의 출력 전부와 shift 32 high의 2^4 시프트 스테이지의 출력전부가 입력되어 최종적으로 32비트 시프트를 수행하게 된다. 표 3은 기존의 barrel 시프트에 데이터 형식 변환 연산을 위해서 필요한 추가 interconnection및 MUX의 개수를 나타내었다. 표 3에 나타낸 바와 같이, 추가 interconnection과 MUX는 각각 185개, 62개로써 각각 약 14.8%, 약 9% 정도 증가하게 된다. [4]의 기존의 integer unit 칩사진을 통하여 알 수 있듯이, 데이터 형식 변환 연산과 시프터의 하드웨어양은, 거의 유사하나, 제안된 데이터 처리유닛은, 단지 interconnection과 MUX부분의 10%정도의 적은 양의 칩 면적 증가로 기존의 시프트 연산 뿐 아니라 각종 멀티미디어 데이터 처리를 위한 하드웨어를 용이하게 구현할 수 있음을 알 수 있다. 이와 같은 개선점은 여러개의 연산유닛으로 구성된, superscalar 프로세서의 경우 기존의 방법보다 보다 큰 차이가 있게 되며, 기존의 ALU등의 연산유닛을 사용한 방법보다 훨씬 향상된 하드웨어 효용도를 얻을 수 있다.

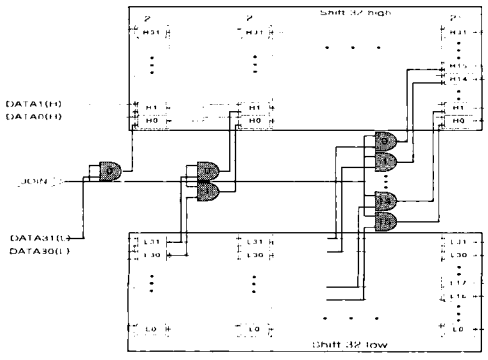

그림 9. MUX-JOIN을 위한 연결구조

V. VLSI 설계

제안된 데이터처리기는 Verilog HDL을 이용하여 설계되고, Verilog-XL를 이용하여 검증이 수행되었다. 그림 10에 시뮬레이션 결과를 나타내었다. op_a1 (op_a2)은 shift 32 high (shift 32 low)의 입력이고, Finout1 (Finout2)은 shift 32 high (shift 32 low)의 출력 값을 나타내며, op_b1 (op_b2)은 shift 32 high와 shift 32 low의 시프트 거리 또는, pack_factor를 나타낸다. Pack은 일반적인 시프트 연산과 데이터 형식변환 연산을 구별하는 신호이다. 첫 번째 Finout1은 op_a1를 8비트 단위로 분할하여

표 3. 추가 Interconnection 및 MUX의 개수

구 분	Interconnection 개수				MUX의 개수
기 존 Barrel shifter	1,251				704
PACK연산을 위해 추가된 interconnection	PACK 16:32	PACK 32:8	PACK 64:1	PACKFIX:16	0
Total	57				0
UNPACK 연산을 위한 Interconnection 및 MUX	EXPAND:64		MERGE:64		64
Total	128				64
하드웨어증가량	14.8 %				9 %

7비트 SLL (shift left logical)을 실행한 결과이고, 첫 번째 하위 Finout2는 op_a2를 비트 분할 없이 6 비트 SLL을 한 결과이다. 두 번째 연산은 PACK16 연산을 한 결과이다. 검증된 Verilog HDL를 바탕으로 Compass를 이용하여 회로합성 후, critical path에 대해 버퍼링등을 통한 실행 시간의 최적화를 수행하였다. 0.6㎛ standard cell를 이용하여 placement & routing을 수행하였으며 그림 11에 최종 layout 결과를 나타내었다.

표 4에 최종 VLSI 설계된 데이터 처리유닛의 설계 결과 요약을 나타내었다. 표 4에 나타낸바와 같이 적은 양의 하드웨어로, 높은 효율의 모든 데이터 처리가 가능한 하드웨어가 설계 가능하다.

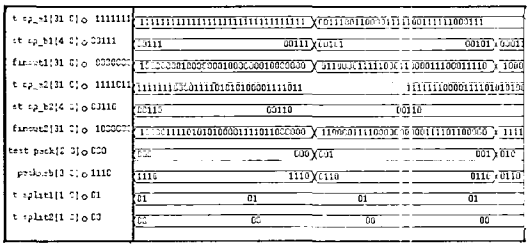

그림 10. 시뮬레이션 결과.

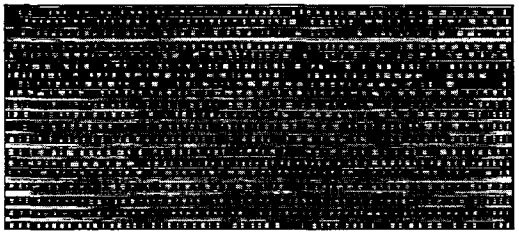

그림 11. 데이터 처리 유닛의 layout.

표 4. VLSI 설계 결과 요약.

공 정	0.6 ㎛ 3metal CMOS
면 적	1.5 ㎜ × 1.0 ㎜
속 도	50 MHz
Gate 수	4,594
트랜지스터 수	18,377
Power 소모	218.01 ㎽
VDD	5 V

Ⅵ. 결 론

기존의 산술 및 논리 워드 단위 시프트와 더불어 비트 분할 또는 통합을 통해 8, 16, 32비트 및 64 비트 데이터에 대한 시프트 및 멀티미디어를 위한 다양한 데이터 형식을 통합적으로 수행 할 수 있는 데이터 처리기가 제안되었다. 데이터 처리기는 형식변환 연산을 위해 약간의 interconnection이 추가된 2개의 32비트 Barrel 시프터, Barrel 시프터간의 선택적 연결 블록, 최종 stage에 2^5 시프트 스테이지 및 EXPAND, MERGE 등을 처리하기 위한 블록으로 구성된다. 전용 데이터 형식변환 하드웨어 대신 기존의 시프터에 약 14.8%와 9%의 추가 interconnection 및 MUX를 통해 구현 가능한 데이터 처리 유닛을 통해 다양한 시프트 및 데이터 형식변환 연산기능을 통합적으로 수행하도록 함으로써 면적 감소와 하드웨어 활용도 향상을 기대할 수 있다. 설계된 데이터 처리기는 Verilog HDL을 사용하여 설계, 동작이 검증되었으며 0.6㎛ standard cell을 바탕으로 하는 Compass를 통해 합성 및 layout 되었으며, 이에 관한 설계 결과가 논의되었다.

참 고 문 헌

[1] L.Kohn, *The Visual Instruction Set(VIS) in UltraSPARC™*, IEEE, 1995

[2] TM320C80 (MVP) Master Processor, Digital Signal Processing Products, Texas Instrument, 1995

[3] Craig Hansen, *Architecture of a Broadband MediaProcessor*, COMPCON96, 1996

[4] Lavi A. Lev et. al., " A 64-b Microprocessor with Multimedia Support", IEEE J. of SSCJ, vol. 30, No. 11, Nov. 1995.

유 재 희(Jae-Hee You)

1963年 3月 3日生. 1985年 2月 서울대학교 전자공학과 학사. 1990年 2月 Cornell 대학교 전기공학과 공학박사. 1990年 3月 ~ 1991年 3月 Texas Instruments. Dallas, VLSI Design Laboratory MTS. 1991年 3月 ~ 현재, 홍익대학교 전자전기 공학부 부교수, P&K, Global Communication Technologies, Primenet 기술 고문. 주 관심 분야는 Multimedia VLSI 아키텍쳐 및 시스템 설계, Home Networking 시스템, DRAM 회로 설계 등임.