論文2002-39SD-2-2

실시간 JPEG 입력 버퍼 아키텍처

(A JPEG Input Buffer Architecture for Real-Time Applications)

任 敏 中 *

(Minjoong Rim)

요 약

USB 카메라를 이용하여 PC 화상 회의를 할 때 image sensor에서 읽어드린 동화상을 USB를 통해서 PC로 전송하게 되는데 이 때 USB의 전송 속도의 제한 때문에 동화상의 압축이 필요하다. 동화상의 압축을 위해서는 많은 양의 메모리가 필요하므로 외부 메모리를 사용하는 것이 일반적이다. 동화상 압축 알고리즘은 여러 가지가 있지만 JPEG을 사용할 경우 동화상 프레임을 모두 저장할 필요는 없으며, JPEG 압축 엔진으로 일정한 속도로 들어오는 데이터와, JPEG에서 사용되는 데이터의, 순서의 불일치를 해결해주는 JPEG 입력 버퍼만이 필요하다. JPEG 입력 버퍼는 읽고 쓰는 순서가 차이가 많이 나므로 double buffering을 사용하는 것이 일반적이지만 이 논문에서는 double buffering을 사용하지 않고 칩 안에 내장되는데 문제가 없는 적은 메모리 요구량으로 구현하는 방법을 제안한다. 제안된 메모리 아키텍처를 사용하면 별도의 외부 메모리가 필요하지 않으므로 부품 감소에 의한 전체적인 비용 절감이 가능하다.

Abstract

When a USB digital camera is used for PC video-conference applications, motion picture data need to be transferred to the PC through the USB port. Due to the mismatch between the data rates of the USB and the motion picture, data compression should be performed before the transmission from the USB. While many motion picture compression algorithms require large intermediate memory space, the JPEG algorithm does not need to store an entire frame for the compression. Instead, a relatively small buffer is required at the input of the JPEG compression engine to resolve the inconsistency between the orders of the inputted data and the consumed data. Data reordering can be easily implemented using a double buffering scheme, which still requires a considerable size of memory. In this paper, a novel memory management scheme is proposed to avoid the double buffering. The proposed memory architecture requires a small amount of memory and a simple address generation scheme, resulting in overall cost reduction.

* 正會員, 東國大學校 情報通信工學科

(Dept. of Information and Communication Engineering, Dongguk University)

※ 본 연구는 2001학년도 동국대학교 논문게재연구비 지원으로 수행되었습니다.

接受日字:2001年2月8日, 수정완료일:2001年12月20日

Ⅰ. 서 론

최근의 영상 압축 기술 및 반도체 집적회로 기술의 발달과 인터넷의 대중화로 인해 음성 뿐만 아니라 동영상도 저렴한 가격으로 PC와 네트워크를 통해 전송되고 처리될 수 있게 되었다. 이러한 추이에 맞추어 더욱

대중화되고 있는 것 중 하나가 데스크 탑 PC용 화상 회의이다. PC 화상 회의를 할 때 흔히 사용되는 USB 카메라는 CCD 혹은 CMOS 이미지 센서(image sensor)에서 받아들인 이미지 정보를 USB를 통해서 PC에 전송한다. 이 때 일반적으로 두 개의 압축이 행해지는데 하나는 PC에서 행하는 화상 회의용 압축이며 다른 하나는 USB를 위한 압축이다. 일반적으로 이미지 센서에서 발생하는 최대 데이터 속도는 USB에서 허용하는 데이터 속도를 초과하기 때문에 USB로 전송하기 전에 압축을 하고 PC에서 이 압축을 푼 다음 다시 화상 회의의 표준과 요구사항에 맞도록 압축을 하는 것이 일반적이다[1-3]. 그림 1은 흔히 사용되는 USB 카메라의 블록 다이어그램을 보여준다[1-3]. 이들 블록들은 보통 압축 엔진과 USB 인터페이스가 하나로 묶여서, 외부 메모리를 포함하여 4 칩(chip)으로 만들어지거나, 4 칩 중 두 개가 하나로 묶여서 3 칩으로 구현된다. 반도체 집적회로 기술의 발달로 칩의 수를 점차 줄여가는 추세에 있으며 부품수의 감소는 전체적인 비용 절감 효과와 시스템의 소형화를 가져올 수 있다.

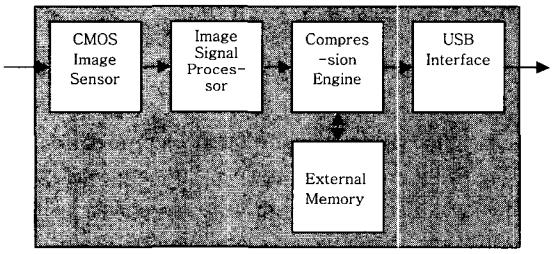


그림 1. 일반적인 USB 카메라의 block diagram
Fig. 1. Block diagram for a typical USB camera.

CMOS 이미지 센서는 1초에 최대 30 프레임(frame)의 속도로 VGA 포맷 데이터를 발생시킨다. Image Signal Processor는 받아들인 이미지의 질을 높이는 작업, CMOS 이미지 센서를 컨트롤하는 작업, 비디오 포맷을 맞추는 작업, 압축 엔진에서 처리하기 쉽도록 데이터를 포맷을 바꾸어 주는 작업 등을 한다. Image Signal Processor에서 발생되는 데이터의 최대 속도는 USB에서 지원하는 한계를 넘어 가므로 데이터를 압축하여 USB 인터페이스로 보내며 USB 인터페이스는 압축된 데이터를 USB를 통해 PC로 보낸다.

손쉽게 압축을 하는 방법은 화상 회의를 할 때의 동영상이 프레임 간에 상관관계가 매우 높다는 점을 이용하여 프레임 사이의 DPCM(differential pulse code modulation)을 하는 방법이다[4]. 이 방법은 동영상에 움직임이 많을 때는 화질의 저하가 일어 날 수 있는 단점이 있지만 그렇지 않은 경우 간단한 로직만으로 USB에서 필요한 충분한 압축율을 실현할 수 있다. 그러나 이 방법은 프레임간의 유사성을 이용하여야 하므로 프레임을 저장하기 위해 외부 메모리가 필요하여 전체 시스템의 크기 및 비용이 증가한다.

외부 메모리를 사용하지 않기 위해서 동영상을 압축할 때 프레임간의 유사성을 이용하지 않는, 즉 단일 프레임에서 압축을 하는 JPEG을 사용할 수 있다[5-6]. 그림 2는 JPEG을 사용해서 압축할 때의 CMOS Image Sensor에서 받아들인 이미지 데이터를 USB를 통해서 전송할 때까지 사용되는 블록들을 나타내며 JPEG 입력 버퍼를 제외한 다른 블록들은 그림 1에서와 동일하다. JPEG 입력 버퍼는 Image Signal Processor에서 온 데이터를 일시 저장하였다가 JPEG 압축 엔진이 압축을 할 때 순서에 맞도록 데이터를 공급한다. JPEG 입력 버퍼가 필요한 이유는 Image Signal Processor에서 공급하는 데이터의 순서와 JPEG 압축 엔진에서 필요로 하는 데이터의 순서가 틀리기 때문이다[5-6]. 그림 3은 JPEG 압축 엔진의 블록 다이어그램을 나타낸다. JPEG 압축 엔진은 8 * 8 화소(pixel) 블록을 입력으로 받아들여 2차원 DCT(discrete cosine transform)를 한 후 quantization과 entropy encoding을 이용하여 압축을 한다.

Image Signal Processor, JPEG 입력 버퍼, JPEG 압축 엔진, USB 인터페이스를 하나의 칩 안에서 구현한

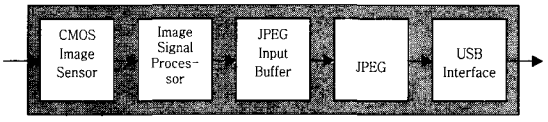


그림 2. JPEG 압축 방법을 사용한 USB 카메라의 block diagram
Fig. 2. Block diagram for a USB camera with JPEG compression.

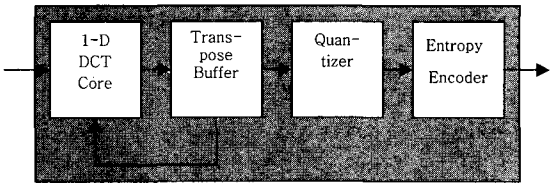


그림 3. JPEG 압축 엔진
Fig. 3. JPEG compression engine.

다고 할 때 전체 칩에서 JPEG 입력 버퍼가 차지하는 면적이 무시할 수 없으므로 효과적인 JPEG 입력 버퍼의 설계가 매우 중요하다. 이 논문에서는 USB 카메라 등에 사용할 수 있는 JPEG 입력 버퍼의 설계에 대해서 논한다. II 절에서는 JPEG 입력 버퍼의 요구 사항과 일반적인 설계 방법을 설명하며 III 절에서는 새로운 메모리 아키텍처를 제안한다. IV절에서는 토론과 함께 결론을 맺는다.

II. 일반적인 메모리 설계 방법

JPEG 입력 버퍼로 데이터가 들어올 때는 CMOS 이미지 센서가 사용하는 클록마다 하나의 화소가 들어오며 하나의 화소는 4:2:2의 경우, 번갈아 가며(Y,U) 또는(Y,V)로 구성되어 있다. VGA 포맷의 경우 하나의 수평 라인은 640 화소와 HSYNC로 구성되므로 JPEG 입력 버퍼에서는 한 줄이 640 화소로 구성되어 한 줄씩 써지게 된다. JPEG 압축 엔진으로 데이터를 읽어갈 때는 JPEG 알고리즘이 8 * 8 화소 블록 단위로 압축을 하는 특성 때문에, 4:2:2의 경우, Y088, Y188, U88, V88의 순서대로 8 * 8 화소 블록씩 읽어가게 된다[5-6].

Y_0	U_0	Y_1	V_0	Y_2	U_1	Y_3	V_1
Y_4	U_2	Y_5	V_2	Y_6	U_3	Y_7	V_3
Y_8	U_4	Y_9	V_4	Y_{10}	U_5	Y_{11}	V_5
Y_{12}	U_6	Y_{13}	V_6	Y_{14}	U_7	Y_{15}	V_7
...							

(a) Writing data

Y_0	Y_{640}	Y_{1280}	Y_{1920}	Y_{2560}	Y_{3200}	Y_{3840}	Y_{4480}
Y_1	Y_{641}	Y_{1281}	Y_{1921}	Y_{2561}	Y_{3201}	Y_{3841}	Y_{4481}
...							
Y_{15}	Y_{655}	Y_{1295}	Y_{1935}	Y_{2575}	Y_{3215}	Y_{3855}	Y_{4495}
U_0	U_{320}	U_{640}	U_{960}	U_{1280}	U_{1600}	U_{1920}	U_{2240}
...							
U_7	U_{327}	U_{647}	U_{967}	U_{1287}	U_{1607}	U_{1927}	U_{2247}
V_0	V_{320}	V_{640}	V_{960}	V_{1280}	V_{1600}	V_{1920}	V_{2240}
...							
V_7	V_{327}	V_{647}	V_{967}	V_{1287}	V_{1607}	V_{1927}	V_{2247}
Y_{16}	Y_{656}	Y_{1296}	Y_{1936}	Y_{2576}	Y_{3216}	Y_{3856}	Y_{4496}
...							

(b) Reading data

그림 4. JPEG 알고리즘을 위한 데이터의 쓰기와 읽기
Fig. 4. Writing and reading data for the JPEG algorithm.

그림 4는 VGA 포맷과 4:2:2의 포맷일 경우에 JPEG 알고리즘을 위해서 데이터가 어떻게 쓰여지고 읽는가를 보여주고 있다.

버퍼에서 데이터를 쓸 때와 읽을 때가 서로 다른 순서를 가지며, 일정한 속도로 데이터가 들어오고 나갈 때는, double buffering을 사용하는 것이 일반적이다. 데이터를 읽을 때는 8 * 8 화소 블록씩 읽어가기 때문에 640 화소 * 8 라인이 써졌을 때 데이터를 읽어가기 시작할 수 있다. 이 때 순차적으로 들어 오는, 다음 8 라인을 다른 메모리에 쓴다면 읽고 쓰기의 충돌을 피할 수 있으며 정해진 순서로 쓰기와 읽기를 반복할 수 있다. 동일한 크기의 메모리를 두 개를 만든 다음 하나의 메모리에 데이터를 쓰는 동안 다른 메모리에서는 데이터를 읽어 내고, 쓰고 있는 메모리에 데이터가 다 차면, 쓰는 메모리와 읽는 메모리를 바꾼다. 그림 5는 double buffering을 사용했을 때의 메모리에서 데이터의 쓰고 읽는 순서의 위치를 표시한다. 하나의 화소의 데이터인 Y 데이터와 UV 데이터가 순차적으로 쓰여지는 것으로 표시하였으므로 한 라인은 640 * 2 = 1280 개의 Y 또는 UV의 데이터를 가진다. 이 방법은 4:2:2의 경우 16 bit/pixel * 640 pixel/line * 8 line/memory * 2 memory = 20 Kbytes의 메모리를, 4:2:0의 경우에는 16 라인을 저장한 후 읽어야 하므로 16 bit/pixel * 640 pixel/line * 16 line/memory * 2 memory = 40

0	1	2	3	4	1277
1280	1281	1282	1283	1284	2559
2560	2561	2562	2563	2564	3839
...					
8960	8961	8962	8963	8964	10239

(a)데이터를 쓰는 buffer에서의 데이터의 쓰기의 순서
The order of writing to the buffer

0	128	8	136	16	10232
1	129	9	137	17	10233
...	...				...
6	134	14	142	22	10238
7	135	15	143	23	10239

(b)데이터를 읽는 buffer에서의 데이터의 읽기의 순서
The order of reading from the buffer

그림 5. Double buffering을 할 때의 데이터의 쓰기와 읽기
Fig. 5. Writing and reading data for the double buffering memory structure.

Kbytes를 차지하게 된다. 이와 같은 크기의 메모리는 전체 칩에서 무시하기 힘든 부분을 차지하게 된다. Double buffering 방법은 쓰는 부분과 읽는 부분이 분리되어 있어 주소 발생 방법과 읽고 쓰기가 간단하지만, 메모리의 낭비가 심하다.

무한히 복잡한 주소 발생 방법을 사용할 수 있다면, 데이터를 읽어낼 때마다 그 자리에 새로운 데이터를 쓸 수 있으므로 이론적으로는 4:2:2의 경우 7 라인 (640 pixel/line * 7 line * 16 bit/pixel = 8960 bytes), 4:2:0의 경우에는 15 라인 (640 pixel/line * 15 line * 16 bit/pixel = 19200 bytes) 의 크기의 메모리만을 필요하게 된다. 그러나 데이터를 읽고 쓰는 방법의 차이가 많이 나기 때문에 최소 메모리만을 사용한다면 주소 발생 방법이 매우 어려워지게 될 것이다.

Ⅲ. 새로운 메모리 설계 방법

이 논문에서는 double buffering의 반의 메모리만으로 메모리를 구성하면서 주소 발생을 위한 하드웨어가 증가하지 않도록 하는 메모리 아키텍처를 제안한다. 메모리를 절약하는 방법은 데이터를 읽어낸 부분에, 즉 유효한 데이터가 있지 않은 부분에 새로운 데이터를 쓰는데 있다. 그러나 읽고 쓰는 방법의 차이가 심하기 때문에 읽은 부분에 바로 새로운 데이터를 쓴다면 주소 발생 방법이 매우 복잡하게 된다. 최근에 읽은 부분에 새로운 데이터를 쓴다는 개념은 유지하지만 약간의 지연을 둠으로써 주소 발생을 간단하게 한다.

새로운 메모리 아키텍처의 기본 개념은 메모리를 논리적으로 Y 메모리와 UV 메모리로 나눈 후 다시 각 메모리를 3차원 메모리로 구성하는데 있다. 그림 6은 2개의 3차원 메모리를 보여준다. 3차원 메모리의 인덱스를 i, j, k라고 하고 Y 메모리와 UV 메모리를 구별하기 위해 인덱스 n (Y 메모리는 0, UV 메모리는 1) 을 사용한다고 하자. 3차원 메모리에서 2개의 차원 (i, j) 은 8 * 8 (4:2:0 포맷에서는 16 * 16) 의 정사각형 모양을 취하게 한 다음 매 8 라인마다 대칭인 두 개의 축이 서로 바뀌도록 한다. 데이터를 쓸 때에는 항상 k가 증가하는 방향으로 먼저 쓴다. 처음 8 라인 동안에는 k가 최대치(VGA 포맷의 경우 80)가 되면 j를 증가시킨다. 그림에서 각 열이 640 화소를 가지는 한 라인을 나타낸다. 7 라인이 다 써지고 8 라인이 써지기 시작하면 데이터를 읽을 수 있다. 데이터를 읽을 때는 그림 6에서 8개의 행 중 첫 번째 행부터 읽어내게 된다. JPEG에서 필요한 데이터는 Y0, Y1, U, V의 순서대로 8 * 8 화소 블록을 읽는 것이므로 행의 순서대로 읽는 것은 아니지만 메모리에서 필요한 모든 데이터가 8개 중 첫 번째 행에 위치하게 되므로 쓰는 주소와 충돌하지 않는다. 첫 번째 행이 모두 읽어지면 첫 번째 행에 9번째 라인을 쓴다. 다시 말해서 행과 열이 바뀌어서(i와 j가 바뀌어서) 데이터가 쓰여지게 된다. 이 때 읽는 부분은 8개의 행 중 두 번째 행에 위치하게 되므로 읽고 쓰기의 충돌은 없다. 데이터를 읽은 자리에 바로 데이터를 쓰는 것은 아니며 데이터의 읽기와 쓰기는 한 라인만큼 차이를 가진다. 즉 이론적인 최소 메모리에 비해서 한 라인만큼 메모리를 낭비한다고 볼 수 있다. 그러나 메모리의 낭비가 심하지 않으면서 읽고 쓰기의 충돌을 쉽게 피할 수 있으며 주소 발생 방법이 간단하여 주소

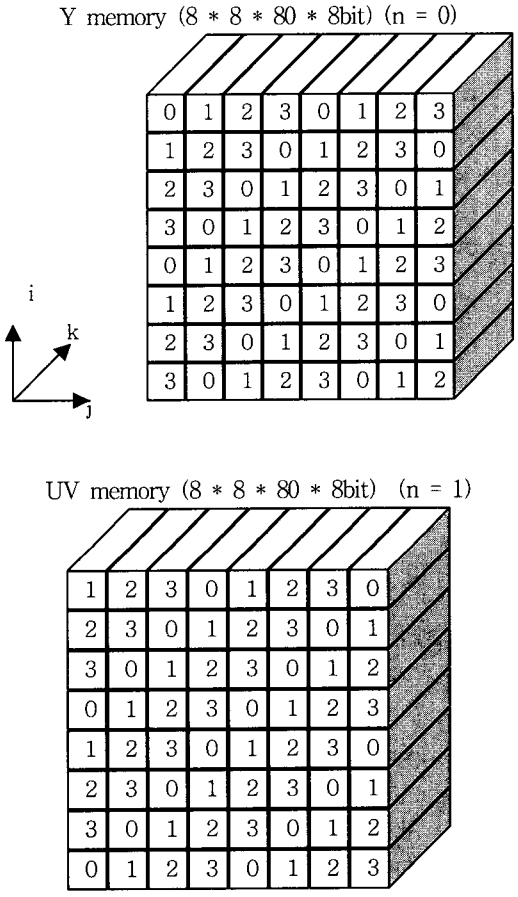


그림 6. JPEG 입력 버퍼의 3차원 메모리의 구성도
Fig. 6. 3-dimensional memory architecture for a JPEG input buffer.

발생 방법의 구현으로 인한 오버헤드가 적게 된다. 이와 같이 행과 열을 바꾸어 가며 데이터를 읽고 쓰는 것이 가능한 것은 메모리를 논리적으로 3차원으로 만들었기 때문이다.

Double buffering을 사용하지 않음으로써 생기는 문제점은 하나의 메모리에서 데이터의 읽고 쓰기를 하므로 고속의 클록을 상용하지 않으면서 single-port RAM을 사용할 경우, 읽고 쓰기의 충돌을 피하기 위해서 메모리를 쪼개야 한다는 것이다. 특히 JPEG에서 사용되는 2차원 DCT에서는 데이터가 일정한 속도로 공급되는 것이 아니고 그림 3에서와 같이 데이터를 빠른 속도로 읽어 내어 1차원 DCT를 8번 수행한 후 transpose하여 다시 1차원 DCT를 8번 수행하므로 double buffering을 하더라도 (고속 클록을 사용하지 않는다면) 한 번에 여러 개의 데이터를 읽고 쓰는 것이 필요하다. 그림 6의 숫자는 물리적인 메모리의 번호로서 한 번에 최대 4개의 데이터를 동시에 읽을 때, 즉 4개의 single-port RAM을 사용하는 물리적인 메모리를 사용할 때 논리적 메모리가 물리적 메모리에 어떻게 mapping되는 가를 나타낸다. 한 번에 4개의 데이터를 읽어내기 위해서는 읽어내는 4개의 데이터가 모두 다른 물리적인 메모리에 속해 있어야 한다. 이 논문의 메모리는 매 8 라인마다 두 개의 축이 뒤바뀌므로 가로 축과 세로 축 모두, 4개의 데이터를 읽을 때, 4개의 데이터가 서로 다른 메모리에 있도록 배치한다. 이를 해결하는 방법으로 (i + j + n) modulo (# of memory blocks)를 취하여 물리적 메모리에 mapping시켰다.

제안된 메모리 아키텍처는 개념적으로는 조금 복잡하지만 Appendix의 pseudo code에서 볼 수 있듯이 메모리 주소 발생이 카운터와 약간의 부가 로직으로 구현 가능하다. 그림 7은 제안된 메모리 아키텍처의 주소 발생기의 구조를 나타낸다. Double buffering을 사용하였어도 주소 발생을 위해서 비슷한 크기의 카운터는 필요한 것이었으므로 제안된 메모리 아키텍처를 위한 주소 발생의 부가 로직은 double buffering에 비해 큰 차이가 없다. 이에 반해서 메모리의 양은 반으로 준다.

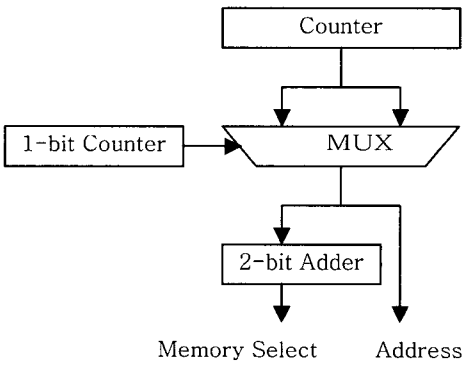


그림 7. 새로운 메모리 구성을 위한 주소 발생기
Fig. 7. Address generator for the proposed memory structure.

Ⅳ. 토의 및 결론

PC 화상 회의의 보급이 가속화되면서 더 저렴한 PC 화상 회의용 칩에 대한 필요성이 대두되었다. 이 논문은 PC 화상 회의에서 많이 사용되는 USB 카메라의 단일 칩 구현을 위해서 사용될 수 있는 실시간 JPEG 입력 버퍼의 메모리 구성 방법에 대해서 설명하였다. 표 1은 제안된 메모리 구성 방법과 double buffering 방법, 이론적인 최소 메모리 구성 방법을 비교한 것이다. 제안된 메모리 구성 방법은 이론적인 최소 메모리 요구량에는 미치지 못하지만 일반적인 double buffering 방법에 비해서 절반의 메모리를 사용한다. 주소 발생을 위해서 사용하는 카운터는 일반적인 카운터에 비해서 순서는 바뀌어 발생이 되지만 카운터 이외의 부가 로직은 별로 없으므로 하드웨어로 구성할 때의 오버헤드는 거의 없다.

표 1. 제안된 방법과 double buffering 방법, 이론적인 최소 메모리 구성법과의 비교
Table 1. Comparison of the proposed, the double buffering, and the theoretical minimum memory structures.

	메모리 요구량	
	4:2:2 format	4:2:0 format
Double buffering	20,480 Bytes	40,960 Bytes
제안된 방법	10,240 Bytes	20,480 Bytes
이론적인 최소치	8,960 Bytes	19,200 Bytes

논문에서 제안된 방법을 사용하면 메모리에 대한 부담을 대폭 줄일 수 있으므로 외부 메모리를 사용하지 않고 화상 회의용 USB 카메라 칩을 구현할 수 있다. 제안된 방법은 USB 카메라 칩 이외에도 JPEG 정지 카메라와 같은 JPEG 응용 칩에 적용될 수 있을 것이다.

Appendix

이 절에서는 보다 구체적인 예를 들어 3절에서 설명한 개념을 정리한다. VGA 포맷을 사용한다고 가정했을 때, 그리고 YUV가 4:2:2인 경우의 메모리의 구성을 보다 자세히 나타내면 다음과 같다. 먼저 논리적인 메모리의 구성은 다음과 같은 특징을 가진다.

▶i (=0..7), j (=0..7), k(=0..79)의 3차원 메모리가 2개 n(=0,1) 있는 구조를 가진다.

▶n이 0에는 Y를 n이 1에는 UV를 쓴다.

j * 80 + k는 각 라인의 640 화소 중 하나를 나타낸다.

▶i는 라인을 나타낸다.

▶읽는 동작은 쓰는 동작보다 한 라인 앞서 간다.

▶매 8 라인마다 i와 j가 바뀐다.

또한 물리적인 메모리의 구성은 다음과 같다.

▶4개의 데이터를 동시에 읽을 수 있게 하기 위해 4개의 2560 byte의 메모리로 구성된다. 그림 6의 숫자는 80 byte의 메모리 블록이 어느 물리적인 메모리에 해당되는가를 보여준다.

▶논리적 주소 i, j, k, n은 (i + j + n) % 4의 물리적인 메모리의 주소 k * 32 + n * 16 + j * 2 + i / 4 에 해당된다. 즉, (k:n:j:i)의 주소를 구성한 후 2 LSB(least significant bit)는 무시한다.

이 개념을 pseudo code로 나타내면 다음과 같다.

```
#define counter(count,size,inc,carry)    ₩
    count = count + inc;  ₩
    carry = (count >= size);  ₩
    count = count % max;

writeOperation(int dataIn[2])
{
static int i = 0;     /* i = 0..7 */
static int j = 0;     /* j = 0..7 */
static int k = 0;     /* k = 0..79 */
static int iteration = 0;  /* iteration = 0..1 */
int c;          /* carry of counters */
writeToMemory(dataIn, i, j, k, iteration);
counter(k, 80, 1, c);
counter(j, 8, c, c);
counter(i, 8, c, c);
counter(iteration, 2, c, c);
}

writeToMemory(int dataIn[2], int i, int j, int k, int iteration)
{
  int n;       /* 0  Y, 1  UV */
int address;
for (n = 0; n < 2; n++) {
address = (iteration == 0) ? (k:n:j:i[2]) : (k:n:i:j[2]);
mem[(i + j + n) % 4][address] = dataIn[n];
}
}

readOperation(int dataOut[4])
{
int i = 1;      /* i = 0..7, read operation starts from the second row */
int jj = 0;     /* jj = 0,4 */
int m = 0;      /* m = 0..15 */
int n = 0;      /* 0  Y, 1  UV */
int iteration = 0;  /* iteration = 0,1 */
int c;          /* carry of counters */
k = (n == 0) ? (o[2:0]:m[3:0]) : (o[2:0]:m[2:0]:m[3]);
readFromMemory(dataOut, i, jj, k, n, iteration);
counter(jj, 8, 4, c);
counter(m, 16, c, c);
counter(n, 2, c, c);
counter(o, 5, c, c);
counter(i, 8, c, c);
counter(iteration, 2, c, c);
}

readFromMemory(int dataOut[4], int i, int jj, int k, int n, int iteration)
{
  int j, m;
for (m = 0; m < 4; m++) {
  j = jj + m;
   address = (iteration == 0) ? (k:n:j:i[2]) : (k:n:i:j[2]);
   dataOut[m] = mem[(i + j + n) % 4][address];
}
}
```

위의 예는 VGA의 경우에 대해서 보였는데 그 이유는 USB 카메라 칩은 여러 포맷을 지원하여야 하며 그 중에서 VGA를 위한 메모리의 요구량이 가장 크기 때문이다. VGA가 아닌 CIF, SIF, QCIF 등의 경우에도 위와 유사한 방법을 사용하여 해결할 수 있다. CIF의 경우에는 한 라인의 수가 8로 나누어 떨어지지 않지만 약간의 메모리와 클록을 낭비함으로써 이를 해결할 수 있다.

참 고 문 헌

[1] OV511 Data Sheet, OmniVision.

[2] OV7610/7110 Data Sheet, OmniVision.

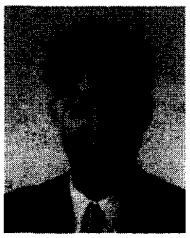
[3] SAA8115HL : Digital Camera USB Interface, Philips.

[4] J.G. Proakis, Digital Communications, McGraw-Hill, Inc., 1989.

[5] G.K. Wallace, The JPEG Still Picture Compression Standard, Communication of the ACM, April 1991.

[6] A.N. Netravali, B.G. Haskell, Digital Pictures : Representation, Compression, and Standards, Plenum Press.

저 자 소 개



任 敏 中(正會員)

1987년 2월 : 서울대학교 전자공학과 공학사. 1990년 5월 : University of Wisconsin-Madison 공학석사. 1993년 8월 : University of Wisconsin-Madison 공학박사. 1993년 9월~2000년 2월 : 삼성전자 선임연구원. 2000년 3월~현재 : 동국대학교 정보통신공학과 전임강사. <주관심분야> 통신 VLSI, 이동통신