Test #1

```
void process_block(const unsigned char* src, const unsigned int width, const unsigned int height,
unsigned char* dst);
```

특정한 DSP 코어에서 동작하는 위와 같이 정의된 함수가 있다고 하자, 이때, src, dst 는 그 DSP 코어에 위치한 빠른 R/W가 가능한 메모리 주소를 받는다고 가정하고, 그 메모리의 크기는 16KB 이며 시작 주소는 DSP_LOCAL_MEM 이라고 한다.

또한 DMA를 위한 함수는 아래와 같이 정의되어 있다고 가정한다.

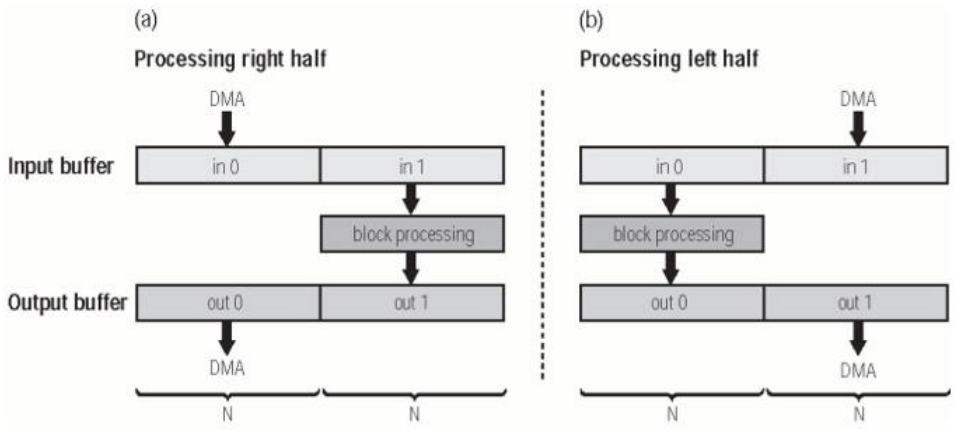
```
DMA_handle_t DMA_copy(const unsigned char* src, unsigned char* dst, const unsigned int size)
// asynchronous 하게 동작하기 때문에 wait을 위한 handle을 반환한다.
bool DMA_wait(DMA_handle_t dma_handle);
```

위 DDR 위치한 256x256 크기를 가지는 입력 영상(g_src)에 대해서 DMA를 활용하여 process_block 함수를 적용하는 간단한 구현은 아래와 같다.

```
unsigned char* g_src;
unsigned char* g_dst;
static const unsigned int SRC_WIDTH = 256;
static const unsigned int SRC_HEIGHT = 256;

static const unsigned int DSP_LOCAL_MEM_SIZE = 16384;

// DSP_LOCAL_MEM을 절반으로 나눠서 src,dst를 위한 메모리로 지정
unsigned char* local_src = DSP_LOCAL_MEM;
unsigned char* local_dst = local_src + (DSP_LOCAL_MEM_SIZE / 2);

// 한번에 처리할 block size와 block의 개수를 계산
unsigned int block_width = SRC_WIDTH;
unsigned int block_height = (DSP_LOCAL_MEM_SIZE / 2) / block_width;
unsigned int loop_count = SRC_HEIGHT / block_height;

// block 의 개수만큼 나누어서 처리
for (int i = 0; i < loop_count; i++) {
 DMA_handle_t dma_handle = DMA copy(g_src + i * block_width * block_height, local_src,
block_width * block_height);
   DMA_wait(dma_handle);
   process_block(local_src, block_width, block_height, local_dst);
   dma_handle = DMA_copy(local_dst, block_width, block_height, g_dst + i * block_width *
```

```
block_height);
   DMA_wait(dma_handle);
}
```

1. 위의 구현은 cache 관련된 잠재적인 bug가 있다.
   A. 잠재적인 bug를 찾아내고 설명하시오.
   B. cache_wb, cache_inv 함수는 적절히 정의하고 이를 이용하여 bug를 고친 구현을 작성하시오.
2. 아래 그림에 설명된 DMA ping-pong 기법을 참조하여 위의 구현이 더 빠르게 동작하도록 개선하시오.



Test #2

메모리상에 아래와 같은 패턴으로 저장되어 있는 BGR 영상이 있다고 할 때,

B0	G0	R0	B1	G1	R1	B2	G2	R2
B3	G3	R3	B4	G4	R4	B5	G5	R5
B6	G6	R6	B7	G7	R7	B8	G8	R8

이를 각각 아래와 같이 B, G, R 채널로 추출하는 함수를 단순히 구현하면 아래와 같다.

```
void split(const unsigned char* bgr, unsigned char* b, unsigned char* g, unsigned char* r, unsigned int size)
{
   for (int i = 0; i < size; i++) {
      b[i] = bgr[i * 3 + 0];
      g[i] = bgr[i * 3 + 1];
      r[i] = bgr[i * 3 + 2];
```

```
    }
}
```

이와 동일하게 동작하는 함수를 아래의 4개 X86 Intrinsic을 사용하여 최적화된 함수를 구현하시오(편의상 size는 16의 배수라고 가정한다.).

__m128i _mm_load_si128 (__m128i const* mem_addr)

movdqa

Synopsis

```
__m128i _mm_load_si128 (__m128i const* mem_addr)
#include <emmintrin.h>
Instruction: movdqa xmm, m128
CPUID Flags: SSE2
```

Description

Load 128-bits of integer data from memory into dst. mem_addr must be aligned on a 16-byte boundary or a general-protection exception may be generated.

Operation

```
dst[127:0] := MEM[mem_addr+127:mem_addr]
```

void _mm_store_si128 (__m128i* mem_addr, __m128i a)

movdqa

Synopsis

```
void _mm_store_si128 (__m128i* mem_addr, __m128i a)
#include <emmintrin.h>
Instruction: movdqa m128, xmm
CPUID Flags: SSE2
```

Description

Store 128-bits of integer data from a into memory. mem_addr must be aligned on a 16-byte boundary or a general-protection exception may be generated.

Operation

```
MEM[mem_addr+127:mem_addr] := a[127:0]
```

__m128i _mm_blendv_epi8 (__m128i a, __m128i b, __m128i mask)

pblendvb

Synopsis

```
__m128i _mm_blendv_epi8 (__m128i a, __m128i b, __m128i mask)
#include <smmintrin.h>
Instruction: pblendvb xmm, xmm
CPUID Flags: SSE4.1
```

Description

Blend packed 8-bit integers from a and b using mask, and store the results in dst.

Operation

```
FOR j := 0 to 15
        i := j*8
        IF mask[i+7]
                dst[i+7:i] := b[i+7:i]
        ELSE
                dst[i+7:i] := a[i+7:i]
        FI
ENDFOR
```

__m128i _mm_shuffle_epi8 (__m128i a, __m128i b)

pshufb

Synopsis

```
__m128i _mm_shuffle_epi8 (__m128i a, __m128i b)
#include <tmmintrin.h>
Instruction: pshufb mm, mm
CPUID Flags: SSSE3
```

Description

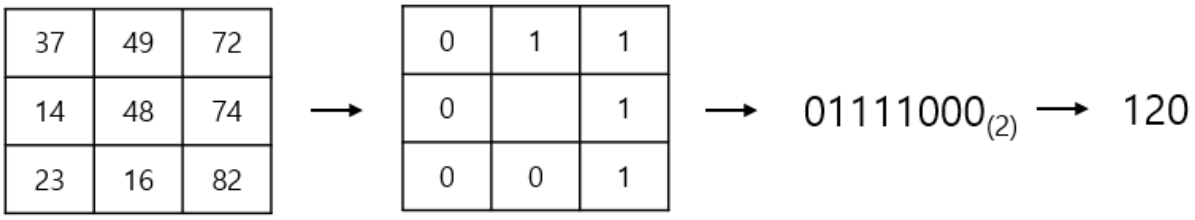
Shuffle packed 8-bit integers in a according to shuffle control mask in the corresponding 8-bit element of b, and store the results in dst.

Operation

```
FOR j := 0 to 15
        i := j*8
        IF b[i+7] == 1
                dst[i+7:i] := 0
        ELSE
                index[3:0] := b[i+3:i]
                dst[i+7:i] := a[index*8+7:index*8]
        FI
ENDFOR
```

Test #3

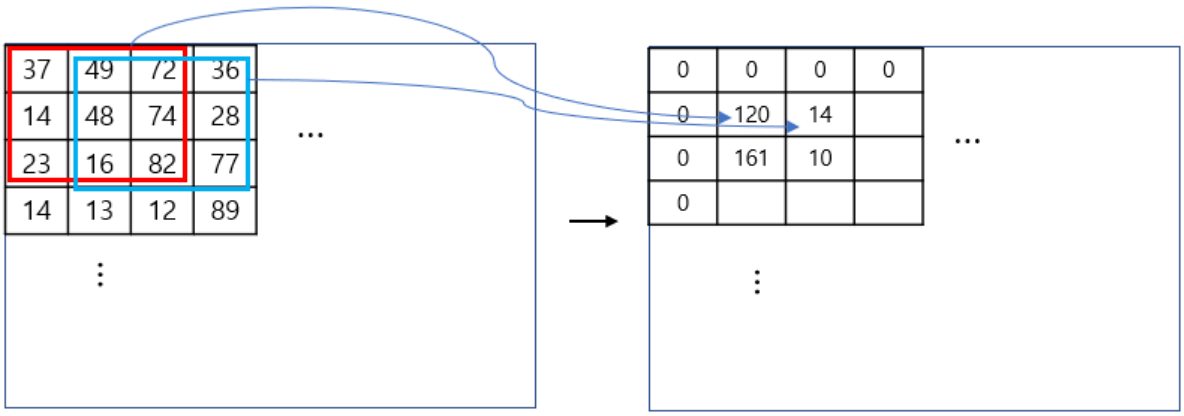
1) 10 <= a <=245 범위의 임의의 값(unsigned char) 가지는 3*3 배열(예, 아래 그림(1))에 대하여 그 배열의 중앙값과 주변 8개의 값을 비교하여 크면 1, 그렇지 않으면 0으로 변환을 한다고 하자 ((예, 아래 그림(2)). 이를 왼쪽상단부터 시계방향으로 그 값을 이진수로 표현을 하면 아래 그림 (3)과 같고, 이를 십진수로 표현하면 아래 그림 (4)가 된다. unsigned char 3*3 배열에 임의의 수를 랜덤하게 입력하고, 최종 변환된 십진수 값과 입력 값을 출력하세요.



(1)입력 (2) 변환 (3) 이진수 표현 (4) 십진수

2) 가로와 세로의 크기가 각각 3x3 보다 큰 임의의 NxM 크기를 가지는 이차원 배열의 모든 위치에 대하여 위 1)의 연산을 수행한다고 하자. 3x3 연산에 대한 결과는 아래 그림과 같이 연산자의 중심 위치에서 결과 배열에 10 진수 값으로 저장한다. (3x3 필터의 중심이 배열의 영역을 넘어가는 외각영역은 0 으로 처리를 한다.) 임의의 크기에 대하여 임의의 값으로 주어졌을 때 변환하는 함수를 구현하고, 이 결과를 출력하세요.

```
void convert(unsigned char* source_data, int N, int M, unsigned char* converted_data)
{
}
```



(1) source_data (2) converted_data

3) 앞의 2)번에서 NxM 크기별(10x10, 100x100, 500x500, 1000x1000)로 convert 함수의 처리속도를 측정해보세요. 처리속도향상을 위하여 노력한 것이 있으면 최대한 언급하고, 향상된 처리속도도 함께 리포트를 해주세요. (처리속도 측정은 debug 모드가 아닌 release 모드에서 측정)

4) 만일 source_data 에 -10 <= k <= 10 범위의 임의의 작은 값(k)을 모든 배열 원소에 동일한 값(k)으로 더하거나 뺐을 때 converted_data 는 어떻게 변하는가요? 그 이유는 무엇인가요?

Test #4

Operating System 의 Kernel 에 대해 매우 자세하게 설명 하십시오.

수고 많으셨습니다. 감사합니다.