論文98-35C-8-3

ASIC 용 메모리 컴파일러 설계

(Design of a Memory Compiler for ASIC)

金 政 範 *, 權 五 瑩 **, 洪 性 濟 **

(Jeong Beom Kim, Oh Hyeong Kwon, and Sung Je Hong)

요 약

본 논문에서는 실리콘 컴파일러의 특수한 경우로서 ASIC 칩에 내장되는 RAM과 ROM을 생성하는 메모리 컴파일러를 제시한다. 먼저, 메모리 구현에 필요한 단위셀들을 설계하였다. 본 논문에서 제시한 메모리 컴파일러는 단위셀들을 이용하여 타일링 방식으로 64~4096워드, 4~256비트 사이의 모든 내장형 RAM과 ROM을 생성한다. 본 논문의 단위셀들은 한 개의 폴리 층과 두 개의 메탈 층을 갖는 0.8μm CMOS 공정을 이용하여 설계하였으며, HSPICE를 이용하여 어드레스 억세스 시간 및 전력소모 등을 모의실험 하였다.

Abstract

In this paper, we propose a memory compiler to generate embedded RAMs and ROMs for ASIC chips. We design the leaf cells to be composed of memory blocks. The compiler is built using tile-based method to simplify routing. The compiler can generate any memory layouts to satisfy 64 to 4096 words and 4 to 256 bits per word. The technology we used here is 0.8 μm single poly double metal CMOS process. The address access time and power consumption are verified through the HSPICE simulation.

I. 서 론

오늘날, 반도체 공정 및 설계기술의 눈부신 발전으로 칩들의 복잡도가 증가하여 단일 칩 상에 수백 만개 이상의 트랜지스터가 내장될 수 있게 되었다. 그러나 대규모 집적회로를 인간의 회로설계 능력만으로 단기간 내에 설계하여 동작속도 및 면적의 제한조건을 만족하는 회로를 구현하기는 매우 어려운 일이다. 이러한 제한조건을 극복하기 위해서 자동 회로설계에 대한 많은 연구가 진행되어 있다. 이러한 연구의 한 방편으로, 메모리 컴파일러에 대한 활발한 연구가 진행되어 왔다[1-6]. 메모리 컴파일러는 RAM과 ROM의 레이아웃을 생성하는 도구이다. 메모리는 반복구조를 갖기 때문에 메모리 칩에서 사용되는 각종 셀들을 설계한 후 어드레스 크기와 워드 크기에 따라 반복배치하기 때문에 자동화에 적합하다.

현재, ASIC(Application Specific Integrated Circuit) 칩들의 개발에 중요한 요소는 설계기간의 단축과 높은 신뢰성이다. 사용자의 다양한 요구를 수용하기 위해서 제품의 라이프 사이클이 과거에 비해 단축되고 있다. 따라서 높은 신뢰성을 가지는 칩을 최단시일 내에 개발하는 것이 요구된다. RAM과 ROM 등의 메모리 요소는 MCU(Micro Computer Unit)와 DSP(Digital Signal Processor) 칩뿐만 아니라 많은 ASIC 칩들에 내장되고 있다. ASIC 칩에 내장

* 正會員, 忠北大學校 電氣電子工學部
(School of Electrical and Electronics Engineering, Chungbuk National University)
** 正會員, 浦港工科大學校 電子計算學科
(Dept. of Computer Science and Engineering, Pohang University of Science and Technology)
接受日字: 1998年3月12日, 수정완료일: 1998年7月28日

되는 메모리 요소들은 그 사용용도와 칩 면적의 최소화를 위해서 다양한 용량의 메모리 요소가 필요하다. 이러한 ASIC 용 메모리를 효과적으로 신속하게 생성시킬 수 있다면, 메모리 컴파일러는 칩 설계기간을 단축할 수 있을 뿐만 아니라 칩의 신뢰도를 향상시킬 수 있기 때문에 칩 설계에 유용하게 사용될 수 있다.

지금까지 메모리 컴파일러에 대한 많은 연구가 진행되어 왔다[3,4]. Shinohara[3]는 데이터 패스(data-path) 모듈에 적합한 다중포트(multi-port) RAM 컴파일러를 개발하였다. 이 컴파일러는 워드 당 비트 수가 64, 워드 크기가 2K로 한정되며, 총 비트 수가 32K 비트 이하의 RAM이 선택적으로 구현 가능하다. 따라서 다중포트(1~6 포트) RAM 구현이 가능하다는 장점이 있는 반면, 총 비트 수가 32K 비트로 한정되어 고용량의 RAM 구현이 불가능하다는 단점이 있다. Tou[4]는 게이트 어레이(gate array) 방식에 내장되는 RAM 컴파일러를 개발하였다. 이 컴파일러는 Mentor Graphic 사의 GDT 환경에서 L 언어로 구현한 것으로서, 워드 당 비트 수가 256, 워드크기가 16K로 한정되며, 총 비트 수가 256K 비트 이하의 RAM이 선택적으로 구현 가능하다. 따라서 이중포트(1~2 포트) RAM 구현이 가능하고 Shinohara[3]의 컴파일러에 비해 고용량의 RAM 생성이 가능하지만, 총 비트 수가 256K 비트로 한정되어 고용량의 RAM 구현이 불가능하다는 단점이 있다. 현재, ASIC 칩에는 1M 비트 정도의 고용량 RAM이 내장되는 수요가 증가하고 있기 때문에 메모리 컴파일러는 고용량 RAM을 구현할 수 있어야 한다. Shinohara[3] 와 Tou[4]의 컴파일러의 공통된 단점은 구현 가능한 총 비트 수내에서 선택적으로 RAM 구현이 가능하다는 점과 총 비트 수가 256K 비트 이하로 한정되어 고용량의 RAM 구현이 불가능하다는 점이다. 이러한 문제점을 해결하기 위해서, 본 논문에서는 워드 당 비트 수가 4~256, 워드크기가 64~4K로, 총 비트 수가 1M 비트 범위내의 모든 RAM과 ROM의 레이아웃이 생성 가능한 메모리 컴파일러를 설계하였다. 따라서 본 논문에서 설계한 메모리 컴파일러는 최소용량 64 x 4 비트, 최대용량 4K x 256 비트 사이의 모든 RAM과 ROM들을 생성 가능하다.

논문 구성은 II장에서 타일링에 대하여 서술하며, III장에서는 메모리 컴파일러의 구성에 대하여 서술한다. IV장은 RAM과 ROM의 설계에 대하여 설명한다. V장은 구현 및 결과비교이며, VI장은 결론이다.

II. 타일링

설계에 사용되는 기본단위를 셀(cell)로 정의한다. 셀은 트랜지스터, 게이트 또는 여러 부품을 내포하는 회로를 지칭한다. 셀은 다시 다른 셀을 포함할 수 있다. 예로 NAND 게이트 셀이 nMOS와 pMOS 트랜지스터 셀을 포함하는 경우처럼 셀은 계층구조를 갖는다. 셀의 계층구조에서 최하위 계층의 셀을 단위셀(leaf cell)이라 한다. 셀들의 복합체를 블록이라 하며, 동일한 셀 뿐만 아니라 다른 셀들을 포함할 수 있다.

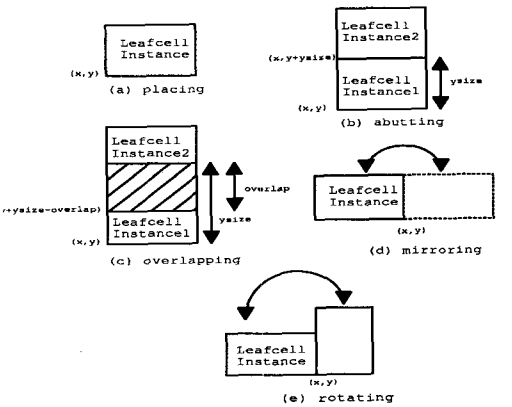


그림 1. 타일 속성
Fig. 1. Tile attribute.

회로설계에 사용되는 단위셀들을 미리 준비하고 이 셀들을 배열함으로서 원하는 레이아웃을 얻는 기법을 타일링이라고 한다. 타일은 단위셀 또는 셀을 지칭하는 것으로 타일과 셀은 본 논문에서 동일한 것으로 간주한다. 타일은 레이아웃의 절대좌표 및 상대좌표에 따라 배열되며 타일들의 접속방법에 따라 속성이 정의된다. 타일의 속성으로는 배치(placing), 접속 (abutting), 공유(overlapping), 반사(mirroring), 회전(rotating)등이 있다. 배치속성은 타일을 원하는 절대좌표에 그대로 타일을 위치시키는 속성이다. 그림 1(a)는 절대좌표(x,y)에 타일을 배치한 그림이다. 그림 1(b)는 타일들 사이에 한 면을 일치시킴으로서 복합셀을 구성하는 접속속성이다. 그림 1(c)는 타일과 타일 사이에 공유 가능한 면적을 중복시켜 배치하는 공유속성이다. 그림 1(d)는 타일이 x축 또는 y축으로 반사변환되는 속성이다. 회전속성은 타일을 기준좌표 예로

타일의 좌측 하단의 좌표에 대하여 시계 또는 반시계 방향으로 회전변환하는 속성이다. 그림 1(e)는 타일이 좌표(x,y)에 대하여 반시계 방향으로 90도 회전한 예다. 이러한 단위셀의 접합과정으로 블록이 형성된다.

Ⅲ. 메모리 컴파일러 구성

메모리 컴파일러는 사용자가 최종 출력하고자 하는 메모리 종류와 크기에 대한 정보만으로 메모리 레이아웃을 생성한다. 본 장은 메모리 컴파일러 개발과 컴파일러를 사용한 레이아웃 생성에 필요한 입력자료와 메모리 컴파일러의 구조, 즉 설계 흐름 과정에 대하여 서술한다.

1. 입력자료

메모리 컴파일러에 사용되는 입력은 두 가지로 구분된다. 첫째 입력자료는 단위셀 레이아웃과 단위셀 명세서이다. 다시 말해서, 물리적인 관점과 논리적인 관점의 입력자료이다. 물리적 관점에서는 단위셀 회로에 대한 레이아웃, 즉 단위셀 레이아웃 라이브러리(library)다. 논리적 관점에서 단위셀 명세서는 단위셀 크기와 단위셀에 필요한 입출력, 전원위치 등에 대한 좌표를 나열한 데이터 베이스다. 두 번째는 RAM과 ROM 레이아웃 생성에 요구되는 매개변수 값이다. 여기서 매개변수 값은 메모리 컴파일러가 생성할 RAM과 ROM의 레이아웃 이름, 워드 크기, 데이터 크기와 내장될 메모리의 형태로 가로(폭), 세로(길이) 비율에 대한 자료가 된다. ROM 설계의 경우 ROM의 마스킹 데이터가 입력자료로 추가된다. 그림 2는 메모리 컴파일러의 입력자료에 대한 그림이다.

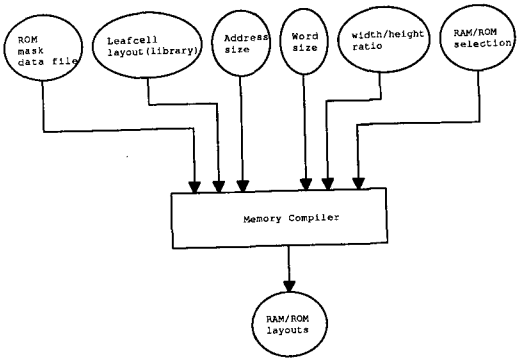


그림 2. 입력 데이터
Fig. 2. Input data.

2. 메모리 컴파일러의 흐름 과정

메모리 크기에 대한 입력자료로부터 메모리 각 블록의 위치를 결정한다. 각 블록은 메모리 크기에 따라 단위셀의 배치 및 복제로 블록들의 레이아웃을 생성한다. 다음 블록들간의 라우팅을 수행함으로서 최종 메모리 레이아웃이 생성된다. 그림 3은 입력으로부터 레이아웃 생성되기까지 메모리 컴파일러의 동작 흐름도이다. 본 논문에서는 블록 레이아웃을 생성하기 위한 단위셀의 배치 및 복제방법으로써 타일링 방법을 사용한다.

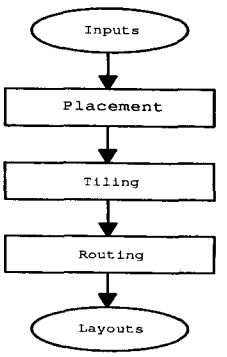


그림 3. 흐름도
Fig. 3. Flow Chart.

Ⅳ. RAM, ROM 설계

1. 단위셀

단위셀들은 그 자체가 의미 있는 회로인 경우와 단순히 라우팅을 간략화하기 위한 셀들로 구분된다. 일반적으로 타일링 설계방식은 다양한 셀을 보유한 경우, 좋은 결과를 산출한다. 즉, 여러 제약조건에 따라 필요한 단위셀을 선택함으로서 보다 사용자의 목적에 적합한 결과를 얻을 수 있다. 본 논문에서 설계한 메모리 컴파일러는 최소용량 64 x 4 비트, 최대용량 4K x 256 비트 사이의 모든 RAM과 ROM들을 생성 가능하도록 다양한 형태의 단위셀들을 설계하여, 컴파일러는 생성하려는 메모리의 용량에 적합한 셀들을 테이블 색인(table look-up) 방식에 의해 선택하여 레이아웃을 생성한다. 또한 단위셀의 표준화가 컴파일러 설계에 중요한 역할을 한다. 표준화는 단위셀의 전원, 클럭 등의 위치를 일정하게 유지함으로서 타일링만으로 라우팅의 간략화가 실현된다.

2. RAM 구조 및 구현 회로

RAM은 예비충전(precharge) 회로, 행 데코더(row decoder) 회로, 열 데코더(column decoder) 회로, 데이터 실렉터(data selector) 회로, 감지 증폭기(sense amplifier), RAM 셀 어레이(cell array) 회로, 입출력 회로로 구성된다. 예비충전 회로는 클럭이 0인 구간에서 모든 데이터 연결선을 1로 설정하는 역할을 한다. 행 데코더 회로는 어드레스 입력에 의해 하나의 워드 연결선을 활성화시키는 역할을 한다. 행 데코더 회로로 본 논문에서는 도미노(domino) 논리 게이트를 이용한다. 열 데코더는 어드레스 입력에 의해 데이터 실렉터 회로를 활성화시키는 역할을 하며 선택된 데이터를 감지 증폭기로 보내는 역할을 한다. 열 데코더 회로 역시 도미노 논리 게이트로 구현한다. 데이터 실렉터 회로는 패스(pass) 트랜지스터를 이용한다. 출력단으로부터 멀리 떨어진 곳에 위치하는 RAM 셀의 경우 출력단과 RAM 셀 사이의 신호 감쇄 가능성이 있다. 이러한 단점을 해결하기 위해서 감지 증폭기가 사용된다. 본 논문에서 RAM 셀은 가장 보편적으로 사용되는 6개의 트랜지스터 구조로 설계하였다. 입출력 장치는 데이터를 입력하거나 선택된 데이터를 외부 버스에 전송하는 회로다.

3. RAM 설계

레이아웃은 사용자가 제시한 폭, 길이 비율에 가장 근접하게 타일들을 배치하기 위하여 어드레스를 MSB에 근접한 상위 어드레스와 LSB에 근접한 하위 어드레스 두 부분으로 구분한다. 상위 어드레스는 RAM 셀 어레이의 행을 표현하며 하위 어드레스는 열을 나타낸다. 어드레스의 상위, 하위 구분은 레이아웃의 폭과 길이의 비율에 따라 자동적으로 나누어진다. 목표로 하는 RAM을 행 데코더 블록, 열 데코더 블록, 메모리 블록, 실렉터 블록, 어드레스 버퍼 블록, 입출력 제어 블록과 클럭 블록등 7개의 블록으로 구분하며, 각 블록은 여러 단위셀의 배열로 구현한다. 어드레스 버퍼 블록과 데코더 블록 사이의 어드레스 연결선 라우팅, 또 각 블록에 공급해야 될 V_{dd}, *GND*, 입출력 신호 및 클럭을 위한 라우팅이 요구된다. 각 블록 내에서는 단위셀의 접합만으로 라우팅이 된다.

1) 블록배치

RAM 전체 배치는 좌표(0,0)을 기준으로 행 데코더, 열 데코더, 메모리, 실렉터 블록이 상하좌우에 배치된다. 그 외 어드레스 버퍼 블록은 어드레스 라우팅이 용이하도록 행 데코더와 열 데코더 블록에 근접하게 배치한다. 입출력 제어 블록과 클럭 블록은 실렉터 블록에 근접하게 배치한다. 어드레스 버퍼 블록, 입출력 제어 블록과 클럭 블록은 나머지 4개 블록들과 라우팅에 필요한 채널을 유지한다. 어드레스 버퍼 블록은 행, 열 데코더 블록과 어드레스 연결선의 라우팅이 요구되기 때문에 채널 폭은 전원과 어드레스 연결선 배치를 위한 최소영역이 확보되는 x좌표를 지정한다. 반면에, 어드레스 버퍼 블록의 y좌표는 행, 열 데코더 블록 두 길이의 합 중심에 놓이도록 y좌표를 정한다. 입출력 제어 블록은 실렉터 블록에 입출력 신호를 공급하기 때문에 신호 공급이 용이하게 좌표를 정한다. 클럭 블록은 클럭 신호선의 전체 길이가 가능한 짧도록 배치하기 위해서 실렉터 블록의 우측 하단에 접속하도록 클럭 블록의 좌표를 지정한다. 사용자가 반사복제된 레이아웃을 원하거나 사용자가 제시한 폭, 길이 비율에 적합한 배치로 반사복제된 레이아웃을 생성할 수 있도록 RAM 배치를 한다. 이 때 y축을 기준으로 클럭 및 입출력 제어 블록을 제외한 나머지 블록들을 반사복제한다.

행 데코더 블록은 행 어드레스, 즉 어드레스의 상위 부분만을 데코딩하여 해당 어드레스의 메모리 셀들을 선택하는 기능을 갖는다. 이 블록은 좌표(0,0)의 좌측 상단에 배치한다. 행 데코더의 최하단은 어드레스 00...0xx...x, 최상단은 11...1xx...x에 일치된다. 여기서, xx...x는 하위 어드레스를 표현한 것으로 x는 0 또는 1을 나타낸다. 행 데코더 블록은 다음 단위셀들로 구성된다. 도미노 회로의 pMOS와 인버터 게이트를 단위셀로, nMOS 트랜지스터를 또 하나의 단위셀로 정한다. 또한, *GND*가 연결된 nMOS 트랜지스터를 하나의 단위셀로 정한다. 행 데코더의 타일 배치 알고리즘은 다음과 같다.

Algorithm mosaic_decoder(row, pos_x, pos_y)
입력 : 데코더 블록 기준좌표, 어드레스 비트 수
출력 : 데코더 레이아웃
1. **for** i = 0 **to** 2^(row-1) do
2. temp = i
3. put row_b on (pos_x, pos_y)
4. pos_x = pos_x + width(row_b)
5. **for** j = 0 **to** row - 1 **do**
6. r = temp % 2
7. temp = temp / 2

```
8.      if (r = 0)
          then
9.            put row_m on (pos_x, pos_y)
10.                 put row_p on (pos_x +
                      width(row_m), pos_y)
          else
11.           put row_p on (pos_x, pos_y)
12.                 put row_m on (pos_x +
                      width(row_p), pos_y)
13.       pos_x = pos_x + width(row_m) +
                    width(row_p)
14.   put row_buf on (pos_x, pos_y)
15.   pos_y = pos_y + height(row_b)
```

알고리즘 mosic_decoder에서 변수 row는 상위 어드레스 비트 수이며, pos_x, pos_y는 행 데코더 블록에서 좌측 하단의 *x, y*좌표를 갖는 변수다. 알고리즘에서 라인 1-7은 변수 row의 값을 2진수로 변환하는 부분이다. 라인 8-15는 단위셀을 배치하는 부분이다.

열 데코더 블록은 하위 어드레스를 데코딩하는 기능을 갖는다. 본 설계에서 열 데코더 블록의 최상단은 어드레스 xx...x00...0, 최하단은 xx...x11...1이 되도록 한다. 여기서, xx...x는 상위 어드레스를 표현한 것이다. 열 데코더 블록은 행 데코더 블록의 단위셀과 단지 크기만 다른 동종의 단위셀을 갖는다. 또한 열 데코더 블록의 타일·배치 알고리즘은 행 데코더 블록의 타일 배치 알고리즘과 동일하나 블록의 상단에서 하단으로 어드레스가 증가하는 배열을 갖는 차이가 있다. 열 데코더 블록은 좌표(0,0)의 좌측 하단에 위치한다.

메모리 블록은 데이터를 저장하는 기능을 갖는다. 메모리 블록은 1 비트 RAM 단위셀이 복제된 셀 어레이 부분 블록, 예비충전 부분 블록과 클럭이 1인 구간에서 *GND*로의 패스(path)를 설정하기 위한 산출(evaluation) 부분 블록으로 구성된다. 메모리 블록은 좌표(0,0)의 우측 상단에 위치한다. 메모리 블록 내의 가장 위쪽에 예비충전 부분 블록을 배치한다. 또 우측의 세로로 긴 사각형에 해당되는 블록이 산출 부분 블록이다. 나머지 부분에 셀 어레이 부분 블록이 위치한다. 메모리 블록의 폭과 길이에 대한 산출식은 식(1)과 같다. 식(1)에서 *col*은 하위 어드레스 비트 수, *row*는 상위 어드레스 비트 수이며, *word*는 메모리 워드 크기로 워드당 비트 수를 나타낸다.

$$\begin{cases} (\text{폭}) = (2^{col} * word) * (\text{RAM 단위셀의 폭}) + (\text{산출 단위셀의 폭}) \\ (\text{길이}) = 2^{row} * (\text{RAM 단위셀의 길이}) + (\text{예비충전 단위셀의 길이}) \end{cases} \quad (1)$$

실렉터 블록은 열 데코더의 출력을 입력으로 받아 하위 어드레스에 일치하는 데이터 비트를 선택하는 것이 주 기능이다. 따라서 열 데코더의 출력과 실렉터 블록의 입력의 연결이 용이하도록 근접하게 배치한다. 실렉터 블록은 좌표(0,0)의 우측 하단에 위치한다. 실렉터 블록은 입출력 버퍼, 데이터 실렉터, 실렉터 통로와 산출 부분 블록들로 구성된다. 입출력 버퍼 단위셀은 데이터의 읽기 및 쓰기 신호에 따라 데이터를 일시 저장하는 플립플롭과 감지 증폭기를 포함한다. 실렉터 단위셀은 하위 어드레스에 일치하는 RAM 셀을 선택하기 위한 목적으로 설계된 단위셀이다. 따라서 열 데코더의 출력이 실렉터 단위 셀의 입력단자에 연결된다. 실렉터 통로 단위셀은 실렉터 단위셀들과 입출력 버퍼 단위셀 사이에 신호전달의 연결통로를 제공하기 위해 단순히 메탈만으로 구성된다. 실렉터 블록의 산출 단위셀 역시 클럭이 1인 구간에서 *GND*로의 패스를 형성하기 위한 목적이다. 실렉터 블록 내에서 부분 블록들의 위치는 다음과 같다. 상단에는 실렉터 부분 블록, 하단에는 입출력 버퍼 부분 블록이, 그 중간에는 실렉터 통로 부분 블록이 위치하며, 우측에 산출 부분 블록이 위치한다. 실렉터 블록의 폭과 길이는 식(2)와 같이 산출된다. 실렉터 블록의 폭은 메모리 블록의 폭과 같다. 식(2)에서 *col*은 하위 어드레스의 비트 수이며, *word*는 메모리의 워드 크기를 의미한다.

$$\begin{cases} (\text{폭}) = (2^{col} * word) * (\text{실렉터 단위셀 폭}) + (\text{산출 단위셀 폭}) \\ (\text{길이}) = 2^{col} * (\text{실렉터 단위셀 길이}) + (\text{실렉터 통로 길이}) + (\text{입출력 버퍼 길이}) \end{cases} \quad (2)$$

어드레스 버퍼 블록의 기능은 어드레스 입력 A_i로부터 A_i와 $\overline{A_i}$을 동시에 출력한다. RAM 배치에서 가장 좌측, 혹은 중앙에 위치하며, 어드레스 버퍼 단위셀 한 종류만으로 구현된다. 어드레스 버퍼 블록은 어드레스 비트 수만큼 상하로 단위셀을 배치한다. 블록의 최하단은 하위 어드레스(A_0)를 나타내는 것으로 열 데코더의 입력으로 연결된다. 반면에 블록의 상위 부분은 행 데코더의 입력으로 연결된다. 따라서 어드레스 버퍼 블록과 행, 열 데코더 블록 사이에 어드레스 연결선의 라우팅이 필요하다. 또한 V_{dd}와 *GND*를 어드레스 버퍼 블록에 연결하는 라우팅이 필요하다. 어드레스 버퍼 블록과 행, 열 데코더 블록 사이의 필요한 채널은 적어도 식(3)에서 산출되는 채널 폭을 확보해야 한다. 위에서 서술한 바와 같이 어드레스 버퍼

블록은 어드레스 입력 A_i에 대해 출력 A_i와 $\overline{A_i}$ 가 동시에 출력되기 때문에 식(3)에서 어드레스 비트 수의 두 배에 해당하는 채널 폭이 필요하다. Δ_1, Δ_2는 각각 메탈 폭과 메탈과 메탈 사이의 간격이다. 식(3)에서 *row*와 *col*은 각각 상위 어드레스와 하위 어드레스 비트 수를 나타내는 변수다. 식(3)에서 상수 2와 3이 더해진 것은 V_{dd}와 *GND* 라우팅 때문이다. 식(3)에 따라 어드레스 버퍼 블록의 *x*좌표가 결정된다. 어드레스 버퍼 블록의 *y*좌표는 행 데코더와 열 데코더 두 길이 합의 중심에 위치하도록 어드레스 버퍼 블록의 좌측 하단의 좌표를 정한다.

$$(\text{채널 폭}) = (2*\max\{row, col\}+2)*\Delta_1 + (2*\max\{row, col\}+3)*\Delta_2 \quad (3)$$

입출력 제어 블록은 입력 제어 신호와 출력 제어 신호를 출력한다. 입출력 제어 블록의 출력은 실렉터 블록의 입출력 버퍼 부분 블록의 읽기 및 쓰기 입력단으로 연결된다. 따라서 실렉터 블록의 좌측 하단, 즉 입출력 버퍼 부분 블록의 첫 번째 좌측 단위셀에 근접하게 배치한다. 입출력 제어 블록은 입력 제어 단위셀과 출력 제어 단위셀로 구성된다.

클럭 블록은 외부 클럭 입력으로 부터 내부 클럭 신호를 출력한다. 클럭 지연 시간 방지를 위해 가능한 클럭 연결선 길이를 최소화하도록 클럭 블록의 위치를 정한다. 실렉터 블록 내의 산출 부분 블록에 클럭 신호가 연결되어야 하기 때문에 산출 부분 블록의 최하단과 클록 블록을 근접하게 배치하여 클럭선의 길이를 감소시킨다.

2) 상위/하위 어드레스 배분

어드레스 배분은 사용자가 원하는 RAM 레이아웃의 폭/길이 비율을 근거로 산출된다. 여기서 원하는 비율을 최대한 만족하면서 최소면적의 레이아웃을 생성하도록 한다. 메모리 컴파일러에서 출력되는 레이아웃은 하위 어드레스가 2 비트일 경우 실렉터 블록의 면적이 최소가 된다. 그러나 어드레스 크기가 크며, 하위 어드레스를 2 비트로 설정할 경우 폭에 비해 길이가 상대적으로 긴 RAM 레이아웃이 생성되는 문제가 있다. 반면에, 하위 어드레스가 3 비트 이상인 경우, 1 비트 증가마다 실렉터 블록 내의 실렉터 단위셀이 지수 배씩 증가되어 점유면적이 증가한다. 따라서 가능한 하위 어드레스 비트 수를 적게 하면서 폭/길이 비율에 적합한 배치를 얻도록 어드레스를 배분한다. 어드레스를 상위/하위 어드레스로 배분하는 알고리즘은 다음과 같다.

```
Algorithm divide_row_col(addr, word, ratio)
  입력 : 어드레스 크기, 워드 크기, 폭/길이 비율
  출력 : 상위/하위 어드레스 비트 수
1. col = 2
2. row = addr - col
3. not_find = true
4. while (not_find and row > 1) do
5.       calculate width of layout
6.       calculate height of layout
7.       now = abs(width - (ratio * height))
8.       if (now < before)
9.         then row_before = row
10.              col_before = col
11.              row = row - 1
12.              col = col + 1
13.              before = now
14.        else not_find = false
15.             row = row_before
16.             col = col_before
17. return(row, col)
```

알고리즘 divide_row_col은 라인 2에서 하위 어드레스 비트 수를 2로 초기화하고, 라인 4-16에서 1씩 하위 어드레스 비트 수를 증가시키며 전체 레이아웃의 폭/길이 비율과 입력으로 주어진 변수 ratio와 근접한 어드레스 배분을 찾는다.

4. ROM 구조 및 구현 회로

ROM은 예비충전 회로, 행 데코더 회로, 열 데코더 회로, 데이터 실렉터 회로, ROM 셀 어레이 회로, 출력 회로로 구성된다. 예비충전 회로, 행 데코더 회로, 열 데코더 회로, 데이터 실렉터 회로는 RAM의 경우와 동일하다. ROM 셀은 NOR 구조로 설계하였다. ROM 셀 어레이에 대한 프로그램은 ROM implantor를 이용한다.

5. ROM 설계

어드레스 크기와 워드 크기 및 폭/길이 비율로부터 내장형 ROM 레이아웃이 생성된다. 폭/길이 비율에 따라 상위 어드레스와 하위 어드레스로 구분하고 상위 어드레스는 ROM 레이아웃의 길이에 영향을 미치며, 하위 어드레스는 폭에 영향을 미친다. RAM과 마찬가지로 상위, 하위 어드레스 구분은 자동적으로 폭/길이

비율에 따라 조정된다. ROM 레이아웃의 완료 후 ROM에 입력될 마스킹(masking) 데이터가 ROM implantor로 레이아웃에 추가 입력된다. 따라서 ROM 설계를 수행하는 사용자는 마스킹을 위한 별도의 데이터 파일을 준비해야 한다. 본 논문에서는 ROM을 행 데코더 블록, 열 데코더 블록, 메모리 블록, 실렉터 블록, 어드레스 버퍼 블록과 클럭 블록등 6개 블록으로 구분한다. 블록들 간에는 어드레스 연결선, V_{dd}, *GND* 및 클럭 신호 공급을 위한 라우팅이 요구된다.

1) 블록배치

ROM 배치는 클럭 블록 배치를 제외하고 RAM 배치와 동일하다. 좌표(0,0)을 기준으로 행 데코더, 열 데코더, 메모리, 실렉터 블록이 상하좌우에 배치된다. 어드레스 버퍼 블록은 행 데코더와 열 데코더 블록에 근접하게 배치한다. 클럭 블록은 행 데코더 블록 우측 상단에 근접하게 배치한다. 어드레스 버퍼 블록과 클럭 블록은 타 블록들과 접촉되지 않은 블록배치를 한다. 따라서 좌표(0,0)을 기준으로 하여 상하좌우의 4개 블록들과 라우팅이 요구된다. 어드레스 버퍼 블록의 기준좌표는 이 블록의 좌측 하단이 된다. 기준점의 x 좌표는 어드레스 버퍼 블록과 데코더 블록 사이에 어드레스 연결선 라우팅 및 전원을 위한 채널을 유지하도록 지정된다. y좌표는 행, 열 데코더 블록의 두 길이의 합 중심에 놓이도록 좌표를 정한다. 사용자의 요구 또는 폭/길이의 비율에 따라 y축을 기준으로 클럭 블록을 제외한 나머지 블록들을 반사복제한다.

행 데코더 블록은 하위 어드레스를 무시하고, 상위 어드레스만을 고려했을 때 상위 어드레스의 짝수, 홀수에 따라 두 개의 부분 블록으로 분리한다. 행 데코더 블록 내의 우측이 홀수 상위 어드레스 데코딩에 사용되는 부분 블록이다. 좌측은 짝수 상위 어드레스 데코딩에 사용된다. 짝수 행 데코더의 최하단은 어드레스 00...0xx...x, 최상단은 11...0xx...x이 된다. 홀수 행 데코더의 최하단은 어드레스 00...1xx...x, 최상단은 11...1xx...x 이다. 여기서, xx...x는 하위 어드레스다. 행 데코더 타일 배치 알고리즘은 RAM의 행 데코더 타일 배치 알고리즘 mosaic_decoder와 유사하다. ROM의 행 데코더 타일 배치 알고리즘은 mosaic_decoder에서 라인 1의 for 루프에서 변수 i가 2씩 증가하도록 한다.

열 데코더는 하위 어드레스만을 데코딩한다. RAM과 동일하게 열 데코더는 위로부터 아래로 하위 어드레스가 증가하도록 배치한다. 즉 열 데코더의 최상단이 어드레스 xx...x00...0, 최하단이 xx...x11...1 이다. 여기서, xx...x는 상위 어드레스가 된다. 열 데코더 블록의 타일배치는 RAM의 경우와 동일하다.

메모리 셀 부분 블록의 상단에 예비충전 부분 블록, 우측에 산출 부분 블록이, 나머지 부분에 ROM 셀 부분 블록이 위치한다. 메모리 블록 내의 타일 배치는 RAM에서 메모리 블록과 동일하다.

실렉터 블록은 좌표(0,0)의 우측 하단에 위치한다. 출력 버퍼 부분 블록, 실렉터 통로 블록, 실렉터 부분 블록과 산출 부분 블록으로 구성된다. 상단에는 실렉터, 하단에는 출력 버퍼가, 그 중간에는 실렉터 통로 블록이 위치하며, 우측에 산출 부분 블록이 놓인다. 역시 RAM의 실렉터 블록의 타일배치와 동일하다.

어드레스 버퍼 블록은 가장 좌측에 위치하며 다른 셀들과 접촉되지 않는 별도의 블록을 형성한다. 어드레스 버퍼 블록은 RAM과 동일하다. 어드레스 연결선이 어드레스 버퍼 블록과 행, 열 데코더 블록 사이에 요구되며, 어드레스 버퍼 블록에 V_{dd}와 *GND*가 공급되어야 한다. 따라서 이 사이의 채널은 적어도 식(3)에서 산출된 채널 폭을 확보해야 한다.

클럭의 지연시간 방지를 위해 가능한 전체 클럭 연결선 길이를 최소로 하고 ROM 배치에서 빈 공간을 활용할 수 있는 클럭 블록 위치를 정한다. 셀 어레이 블록의 예비충전 단위셀과 근접한 위치에 클럭 블록의 위치를 선정한다. 클럭 블록은 RAM의 클럭 블록과 동일한 회로다.

2) 상위/하위 어드레스 배분

사용자가 원하는 레이아웃의 폭/길이 비율에 적합한 레이아웃이 출력되도록 어드레스를 배분한다. RAM과 마찬가지로 하위 어드레스가 2 비트일 경우 최소면적을 갖는 레이아웃이 얻어진다. 하위 어드레스의 비트수가 1 씩 증가마다 실렉터 블록의 실렉트 단위셀이 지수 배씩 증가된다. 따라서 하위 어드레스 비트 수를 감소시켜 폭/길이 비율에 근접한 레이아웃이 출력되도록 한다. ROM 단위셀은 nMOS 트랜지스터만으로 구현되기 때문에 상대적으로 ROM 셀 어레이 블록은 다른 블록에 비해 전체 레이아웃에서 차지하는 면적이 작다. ROM 레이아웃 배치에서 반사복제를 하면 ROM 셀 어레이 면적보다 나머지 블록이 차지하는 영역이 증가하게 된다. 따라서 최소면적의 ROM 레이아웃을 얻기 위해 반사복제를 피하는 방향으로 ROM

레이아웃을 출력한다.

3) ROM 마스킹 데이터 파일

ROM 마스킹 데이터는 텍스트 파일로 라인 당 1워드를 나타내고, 어드레스 크기만큼의 행을 갖는다. 한 라인에서는 오른쪽이 MSB이며, 왼쪽이 LSB이다. 첫 번째 라인은 어드레스 0을, 마지막 라인은 어드레스 (2^n-1) 위치의 데이터이다.

V. 구현 및 결과비교

단위셀 레이아웃은 한 개의 폴리층과 두 개의 메탈층을 갖는 0.8 μm CMOS 공정을 이용하여 설계하였다. 메모리 컴파일러는 Cadence 사의 SKILL 언어로 구현하였으며, 레이아웃은 Cadence 사의 설계 도구인 layoutPlus를 이용하였다[7]. 메모리 컴파일러는 사용자가 원하는 메모리의 크기, RAM 또는 ROM 구분 및 단위셀 라이브러리를 쉽게 입력하도록 GUI (Graphic User Interface)를 제공한다.

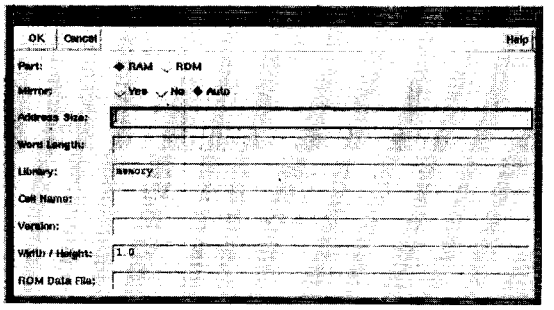


그림 4. 초기화면
Fig. 4. Memory compiler user interface.

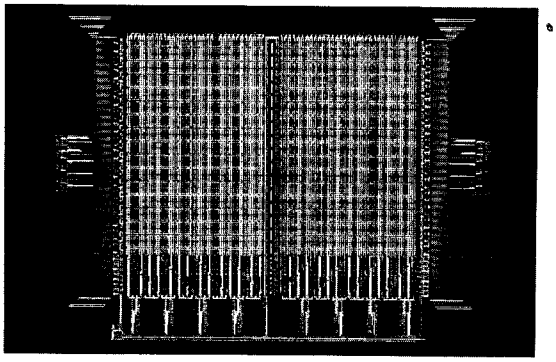


그림 5. 1K x 4 RAM의 레이아웃
Fig. 5. Layout of 1K x 4 RAM.

그림 4는 구현한 메모리 컴파일러 실행할 때의 초기 화면이다. 초기 화면의 Mirror 항목에서 Auto는 메모리 컴파일러가 반사복제한 레이아웃의 출력여부를 결정하는 선택사양이다. 그림 5는 본 논문에서 제시한 메모리 컴파일러가 생성한 1K x 4 RAM의 레이아웃 출력이다. 표 1은 워드 크기가 8 비트이고 폭/길이 비율을 1로 정했을 때 어드레스 크기에 따른 메모리 레이아웃의 크기를 나타낸 표이다.

표 1. 메모리 레이아웃 크기(8비트, 폭/길이=1)
Table 1. Layout size of memory.

	address size	width(um)	height(um)
RAM	64	538.0	524.5
	128	554.4	833.3
	256	906.4	894.7
	512	1174.4	1450.9
	1K	1626.8	1635.1
	2K	3034.8	1880.7
	4K	3051.2	3115.9
ROM	64	447.9	200.6
	128	471.1	262.6
	256	494.3	379.0
	512	517.5	604.2
	1K	540.7	1047.0
	2K	1127.8	1054.6
	4K	1174.2	1932.6

표 2. 비교표
Table 2. Comparision table.

	Shinohara[3]	Tou[4]	본 논문
구 조	다중포트(1-6) RAM	단일/이중포트 RAM	단일포트 RAM/ROM
워드당 비트 수	1 - 64	1 - 256	4 - 256
워드 수	4 - 2K	4 - 16K	64 - 4K
총 비트 수	32K	256K	1M
공 정	1um double metal CMOS	0.7um L_{eff} triple metal CMOS	0.8um double metal CMOS
설계 방식	datapath	gate array	cell based
구현 환경	-	Mentor Graphics GDT L language	Cadence Skill Language
address access time	-	6.2 nsec	5.6 nsec
Max. power consumption at 40 MHz	-	100.5 mW	109.4 mW

본 논문에서 제시한 메모리 컴파일러가 생성한 1K x 8 RAM의 레이아웃으로부터 SPICE netlist를 추출한 후, SPARC SERVER 1000 workstation에서 HSPICE로 모의실험 하였다. 모의실험은 어드레스 억세스 시간(address access time)과 최대 전력소모(maximum power consumption)에 대해 실행하였다. 여기서 어드레스 억세스 시간이란 RAM의 read 동작시의 동작속도를 측정하는 파라미터로서, 어드레스가 지정되어 있고 read 신호가 활성화되어 있는 상태에서 클럭이 0 상태에서 1 상태로 변할 때 출력단자에서 유효한 데이터가 인출되기까지의 시간을 의미한다. 어드레스 억세스 시간에 대한 모의실험은 정상상태(V_{dd}=5V, Temp=25'C)에서 실행하였으며, 결과는 5.6 nsec로서 기존논문의 결과 6.2 nsec보다 9.7% 향상되었다. 또한, 최대 전력소모에 대한 모의실험은 최상상태(V_{dd}=5.5V, Temp=-40'C)에서 40 MHz의 동작속도 조건에서 실행하였으며, 결과는 109.4 mW로서 기존논문의 결과 100.5mW보다 8.9% 증가하였다. 기존논문과 비교한 내용을 표2에 정리하였다. 본 논문에서 제시한 메모리 컴파일러는 총 비트 수가 1M로써, 기존논문의 컴파일러보다 고용량 메모리 생성에 유용하다. 또한 최소용량 64 x 4 비트, 최대용량 4K x 256 비트 사이의 모든 RAM과 ROM들을 생성 가능하며, 어드레스 억세스 시간 및 최대 전력소모등에 대한 모의실험 결과를 통하여 기존논문의 결과보다 우수함을 확인할 수 있었다.

VI. 결 론

본 논문에서는 메모리 구현에 필요한 단위셀들을 설계한 후, 타일링 방식을 이용하여 내장형 RAM과 ROM을 생성하는 메모리 컴파일러를 설계하였다. 본 논문에서 제시한 메모리 컴파일러는 총 비트 수가 1M로써, 고용량 RAM, ROM 생성에 유용하며, 최소용량 64 x 4 비트, 최대용량 4K x 256 비트 사이의 모든 RAM과 ROM들을 생성 가능하다. 또한, 어드레스 억세스 시간 특성이 기존논문의 결과보다 우수함을 HSPICE 모의실험을 통하여 확인하였다. 그러나 생성 가능한 RAM과 ROM이 단일 포트라는 단점을 갖는다. 따라서 생성되는 RAM과 ROM을 다중포트로 확장하는 문제, 저 전력소모를 위한 셀 및 구조에 대한 연구 및 내장형 테스트 회로(built-in test circuit)의 추가 등에 대한 연구가 수행되어져야 할 것이다.

참 고 문 헌

[1] 홍성제, 권오형, 김정범, "ASIC용 ROM/RAM Compiler의 개발", 최종 연구 보고서, 현대전자, 1995. 6

[2] R. W. Brodersen, *Anatomy of a Silicon Compiler*, Kluwer Academic Pub., Boston, 1992.

[3] H. Shinohara, *et al.*, "A Flexible-Port RAM Compiler for Datapath," *in Proc. CICC,* pp. 16.5.1-16.5.4, 1990.

[4] J. C. Tou, *et al.*, "A Submicronmeter CMOS Embedded SRAM Compiler," *IEEE J. Solid State Circuits,* vol. 27, No. 3, pp. 417-424, Mar. 1992.

[5] N. Dutt, and D. Gajski, "Design Synthesis and Silicon Compilation," *IEEE Design & Test of Computers,* pp. 8-23, Dec. 1990.

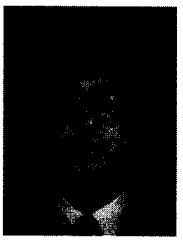
[6] G. Suzuki, *et al.*, "MOSAIC: A Tile-Based Datapath Layout Generator," *in Proc. ICCAD,* pp. 166-170, 1992.

[7] Cadence, *SKILL Language User Guide* 4.3, Cadence Inc., 1994.

저 자 소 개

金 政 範(正會員)

1985년 2월 인하대학교 전자공학과(학사). 1987년 2월 인하대학교 대학원 전자공학과(석사). 1997년 2월 포항공과대학교 대학원 전자전기공학과(박사). 1987년 1월 ~ 1992년 5월 금성반도체(현 LG반도체) 중앙연구소 선임연구원. 1994년 8월 ~ 1997년 9월 현대전자 시스템 IC 연구소 책임연구원. 1997년 9월 ~ 현재 충북대학교 전기전자공학부. 관심분야는 VLSI 설계, CAD, Multi-Valued Logic



洪 性 濟(正會員)

1973년 서울대학교 전자공학과(학사). 1979년 Iowa State University 전자계산학과(석사). 1983년 The University of Illinois at Urbana-Champaign 전자계산학과(박사). 1973년 ~ 1975년 중앙 경리단. 1976년 ~ 1977년 동양컴퓨터 엔지니어링, 1983년 ~ 1989년 GE 연구원. 1989년 ~ 현재 포항공과대학교 전자계산학과 교수. 관심분야는 VLSI 설계, Test Generation, 스위칭 이론, 병렬처리

權 五 瑩(正會員)

1992년 3월 ~ 현재 포항공과대학교 전자계산학과 박사과정 재학중. 관심분야는 논리합성 및 설계자동화