캐리 선택과 캐리 우회 방식에 의거한 비동기 가산기의 CMOS 회로 설계

정 성 태†

요 약

본 논문에서는 캐리 선택 방식과 캐리 우회 방식에 의거한 비동기 가산기의 설계에 대하여 기술한다. 이러한 기법을 사용함으로써 본 논문의 가산기는 기존의 리플 캐리 방식의 가산기에 비하여 보다 빠른 속도로 동작한다. 본 논문에서는 CMOS 도미노 논리를 사용하여 가산기를 설계하였으며 비동기 가산기의 동작 완료를 감지할 수 있는 회로를 트리 형태로 구현함으로써 동작 완료에 소요되는 시간을 줄일 수 있도록 하였다. 실험 결과에 의하면 제안된 가산기들은 평균적으로 리플 캐리 방식에 비하여 50 퍼센트 이상의 속도 개선을 기대할 수 있음을 알 수 있다.

A Design of a CMOS Circuit of Asynchronous Adders Based on Carry Selection and Carry Bypass

Sung-Tae Jung†

ABSTRACT

This paper describes the design of asynchronous adders based on carry selection and carry bypass techniques. The designs are faster than existing asynchronous adders which are based on ripple carry technique. It is caused by reducing the carry transfer time by using carry selection and carry bypass techniques. Also, the design uses tree structure to reduce the completion sensing time. The proposed adders are designed with CMOS domino logic and experimented with HSPICE simulator. Experimental results show that the proposed adders can be faster about 50% in average cases than the previous ripple carry adders.

1. 서 론

VLSI 기술의 발전에 따라 고속의 대규모 디지털 시스템이 출현되고 있다. 그런데, 근래에 들어 동기 방식의 대규모 디지털 시스템의 설계에 있어서 전역 클럭(global clock)이 제한 요소로 대두되고 있다. 이는, 반도체 소자의 속도가 빨라지는 것에 비례하여 선의 신호 전달 속도가 빨라지지 않고 클럭이 시스템의 각 구성 요소에 동시에 전달되지 않는 클럭 스큐(clock skew) 현상을 제거하기가 어렵기 때문이다. 이러한 문제를 해결하기 위한 한 방법으로 전역 클럭을 사용하지 않는 비동기 회로(asynchronous circuit)에 대한 연구가 활발히 진행되고 있다[1, 2, 3, 4].

동기 회로가 클럭을 사용함으로써 가장 느린 경우의 속도로 동작하는 반면에 비동기 회로는 평균 속도로 동작할 수 있다는 장점을 가지고 있다. 즉, 순차 회

※ 이 논문은 1998년도 원광대학교의 교비지원에 의해서 연구됨
† 정 회 원 : 원광대학교 컴퓨터공학과 교수
논문접수 : 1998년 3월 2일, 심사완료 : 1998년 8월 25일

로에서 조합 회로의 동작 시간이 입력 데이터에 따라 나를 경우에, 동기 순차 회로에서는 가장 느린 경우의 동작 시간 동안 기다린 후에 조합 회로의 결과를 저장 장치에 저장한다. 반면에, 속도 독립 회로에서는 조합 회로의 동작이 완료되었는지를 감지하여 조합 회로의 결과를 저장 장치에 저장함으로써 평균 속도로 동작할 수 있다.

또한 비동기 회로는 저전력 회로의 구현에 적합하다는 장점을 가지고 있다. 전력 소모 요인 중에서 부하 커패시턴스의 충전과 방전에 의한 소모가 아주 큰 비중을 차지하는데, 이러한 전력 소모를 줄일 수 있는 방법으로는 전압을 낮추거나, 기생 커패시턴스를 줄이거나, 주어진 일에 필요한 게이트 출력의 전이를 줄이는 방법이 있다[5]. 전압을 낮추거나 기생 커패시턴스를 줄이는 방법은 충전과 방전 시에 소모되는 전력을 줄이는 방법이고 주어진 일에 필요한 게이트 출력의 전이를 줄이는 방법은 충전과 방전의 회수를 줄이는 방법이다. CMOS 비동기 회로에서는 회로가 유효한 작업을 하는 경우에만 게이트 출력의 전이가 일어나므로 게이트 출력의 전이 회수가 줄어들 수 있다. 즉, 전체 회로 중에서 현재 사용되지 않는 부분은 단지 누출 전류로 인한 전력만을 소모하게 된다. 이러한 이유로 전력 소모량이 회로의 활동과 직접적으로 비례하는 CMOS 회로의 경우에 비동기 회로가 저전력 회로 설계의 한 대안으로 대두되고 있다.

이러한 배경으로 비동기 논리를 이용하여 고성능 저전력 시스템을 구현하려는 노력들이 시도되고 있으며 그 결과로 디지털 시스템에 널리 사용되는 가산기의 비동기 구현 방법이 소개되었다[6,7]. 이들 가산기는 캐리 전달 방식에 있어서 리플 캐리 방식을 사용하고 있는데, 본 논문에서는 캐리 선택 방식과 캐리 우회 방식을 사용함으로써 캐리의 전달 시간이 훨씬 빠른 가산기를 설계한다. 본 논문에서는 CMOS 도미노 논리를 사용하여 가산기를 설계하였으며 비동기 가산기의 동작 완료를 감지할 수 있는 회로를 트리 형태로 구현함으로써 동작 완료에 소요되는 시간을 줄일 수 있도록 하였다. 실험 결과에 의하면 평균적으로 리플 캐리 방식에 비하여 50 퍼센트 이상의 속도 개선을 기대할 수 있음을 알 수 있다. 반면에 캐리 선택과 캐리 우회를 위하여 추가되는 회로로 인하여 회로의 면적이 증가하였고 전력 소모도 리플 캐리 방식에 비하여 증가하였음을 알 수 있다. 따라서 본 논문에서 제안된 가산기는 면적이나 전력 소모 보다 속도가 중요한 경우에 사용될 수 있을 것이다.

2. 가산기 설계

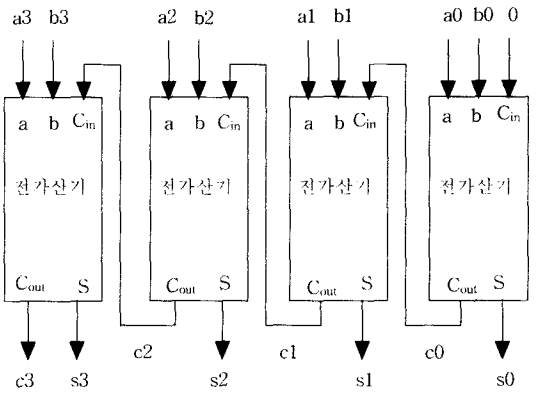
2.1 덧셈 연산

한 비트의 덧셈에는 세 입력이 필요하다. 즉, 더해질 두 비트 입력과 함께 이전의 단계로부터의 캐리 입력이 사용된다. 여기에서 캐리 입력은 이전 단계, 즉 낮은 자릿수 비트 덧셈으로부터 생성된다. 이와 같이 캐리 입력을 가지는 한 비트 가산기를 전가산기(full-adder)라 하는데, 전가산기의 진리표가 표 1에 나타나 있다.

〈표 1〉 전가산기의 진리표
〈Table 1〉 The truth table of a full-adder

입 력			출 력	
a	b	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

여러 비트의 두 수를 더하는 가산기로는 그림 1과 같이 하위 비트의 캐리 출력을 상위 비트의 캐리 입력에 순차적으로 전송하는 리플-캐리 가산기(ripple-carry adder)가 있다.



(그림 1) 4-비트 리플 캐리 가산기
(Fig. 1) 4-bit ripple carry adder

그림 1에 나타나 있듯이 리플-캐리 가산기에서는 하위 비트의 캐리 출력이 상위 비트의 캐리 입력에 연결된다. 따라서 두 수를 더하는 데 소요되는 시간은 캐리가 전달되는 속도에 의해 결정된다. 그러나 한 비트 덧셈의 캐리 출력을 생성하는데 반드시 하위 비트의 캐리 입력이 필요한 것은 아니다. 즉, 표 2에 나타나 있는 바와 같이 두 비트가 모두 0이거나 모두 1이면 캐리 입력을 기다리지 않고도 캐리 출력을 생성할 수 있다. 따라서 이와 같은 방법으로 캐리 출력을 생성하면 캐리가 전송되는 길이는 상당히 줄어들 수 있다.

〈표 2〉 전가산기의 캐리 출력
〈Table 2〉 The Carry output of a full-adder

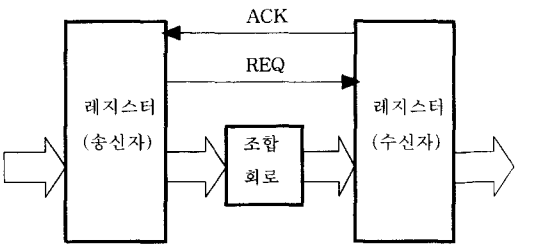
A	B	Cout
0	0	0
0	1	Cin
1	0	Cin
1	1	1

캐리는 하위 비트부터 상위 비트로 순차적으로 전송되는 것이 아니라 여러 위치에서 동시에 전송될 수 있고 캐리 전송에 소요되는 전체 시간은 연속된 캐리 전송의 길이에 달려 있다. 이론적으로는 32 비트 크기의 임의의 두수에 대한 덧셈의 평균 캐리 전송 길이는 $Log_2 32 = 4.4$이다. 그러나 대부분의 프로그램의 데이터 처리에서 피연산자들은 임의의 값을 가지지 않고 평균 캐리 전송 길이가 길어진다. 컴퓨터의 성능을 측정하는 벤치마크 프로그램인 Dhrystone의 명령어를 추적한 결과 데이터 처리에는 평균 캐리 전송의 길이가 18이었다[6].

2.2 비동기 회로 설계

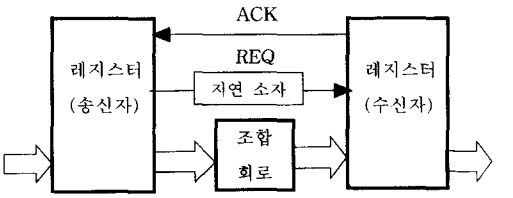
디지털 회로는 연결되어 있는 순차 회로 모듈들의 집합으로 볼 수 있는데, 동기 회로에서는 순차 회로 모듈간의 데이터의 전송이 클럭에 의해 제어된다. 즉, 한 클럭 주기 동안에 각 모듈이 입력 데이터를 처리하여 출력 데이터를 생성하도록 함으로써 자동적으로 데이터들이 모듈간에 전송되도록 할 수 있는 것이다.

비동기 회로에서는 모듈간의 데이터 전송을 제어하기 위하여 그림 2와 같이 모듈간의 통신을 사용한다. 데이터를 받는 모듈에서는 데이터를 받을 준비가 되었음을 ACK 신호로 알리고 데이터를 보내는 모듈에서는 데이터가 준비되었음을 REQ 신호로 알림으로써 데이터를 전송한다[8].

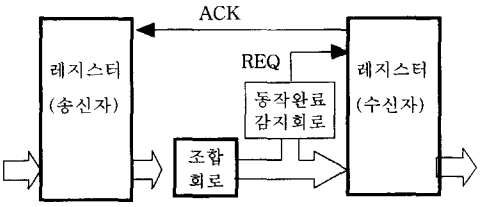


(그림 2) 비동기 회로의 데이터 전송 제어
(Fig. 2) A control method for data transfer in asynchronous sequential circuits

데이터를 전송하려는 모듈에서 데이터가 준비되었다는 것을 인지하는 방법으로는 두 가지가 사용된다[8]. 하나는 그림 3의 (a)와 같이 모듈의 동작 시간에 해당하는 지연 소자를 삽입하는 방법이다. 이 방법은 구현이 단순하고 회로의 크기가 작다는 장점이 있지만 동기 회로와 마찬가지로 각 모듈의 속도를 최대한 이용하지 못하는 단점이 있다. 다른 방법으로는



(a) 조합 회로의 최대 지연 시간과 동일한 지연 소자를 이용한 방법



(b) 동작 완료 감지 회로를 사용한 방법

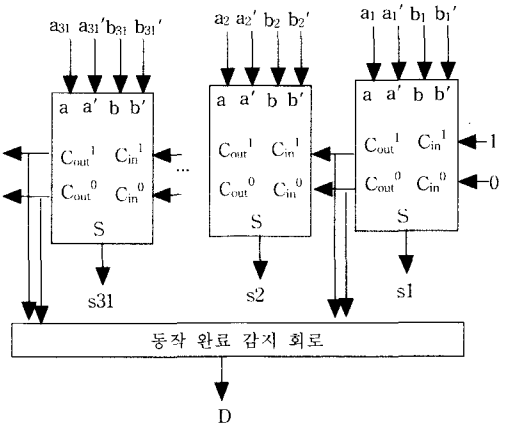
(그림 3) 동작 완료 인지 방법
(Fig. 3) Completion detection method

그림 3의 (b)와 같이 모듈의 동작이 완료되었는지를 감지하는 회로를 사용하는 방법이 있다. 이 방법은 데이터의 각 비트를 두 개의 선으로 구현해야 하고 동작 완료 감지를 위한 회로가 추가되어야 하므로 회로의 크기가 커지는 단점이 있는 반면에 각 모듈의 속도를 최대한 이용할 수 있는 장점이 있다. 본 논문에서는 이 방법을 이용하여 가산기를 설계한다.

비동기 회로의 데이터부의 구현에는 단일 레일 코드와 이중 레일 코드 두 가지 방법이 사용될 수 있다. 단일 레일 코드에서는 데이터의 각 비트를 전달하는 데에 하나의 선을 사용하고 각 선에는 두 가지의 안정된 논리 값인 0과 1이 전달된다. 이중 레일 코딩 방법에서는 두 개의 선으로 데이터의 한 비트를 전송한다. 예를 들면 "01"은 안정된 논리 값 1을 "10"은 안정된 논리값 0을 "00"은 유효한 데이터를 분리해주는 분리자를 나타나도록 할 수 있다. 이렇게 함으로써 데이터 자체로부터 데이터가 유효한지 그렇지 않은지를 알아낼 수 있는 것이다. 본 논문의 비동기 가산기에서는 캐리 전송에 이중 레일을 사용하였다.

2.3 기존의 비동기 가산기

참고문헌 [6]의 비동기 리플-캐리 가산기는 그림 4와 같은 구조로 설계되었다. 캐리 신호를 이중 레일로 구현하고 이들의 값이 모두 유효한 값을 가지는지를 검사하는 동작 완료 감지 회로를 사용함으로써 덧셈 연산이 완료되었는지를 감지한다.



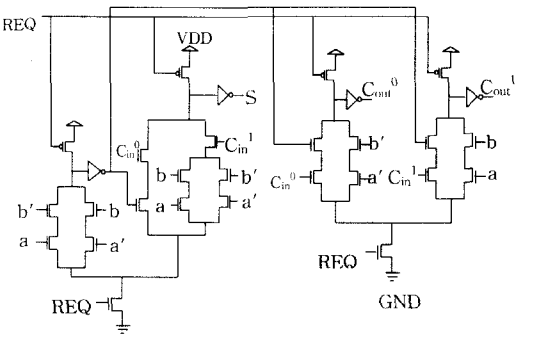
(그림 4) 비동기 리플 캐리 가산기
(Fig. 4) Asynchronous ripple carry adder

그림 5에는 참고 문헌 [6]의 가산기의 한 비트의 구조가 나타나 있다. 이 회로에 대한 논리식은 진리표로부터 간소화 과정을 거쳐서 다음과 같이 구하였다.

$$S = C_{in}^{0}(a'b + ab') + C_{in}^{1}a'b' + C_{in}^{1}ab$$

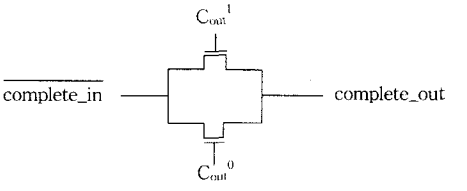
$$C_{out}^{1} = C_{in}^{1}(a'b + ab') + ab$$

$$C_{out}^{0} = C_{in}^{0}(a'b + ab') + a'b'$$



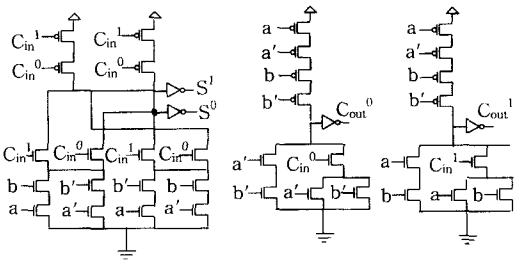
(그림 5) 전가산기의 구조[6]
(Fig. 5) The Adder Circuit[6]

캐리 값은 C_{out}^{1}, C_{out}^{0}와 같이 두 개의 신호를 사용하여 나타내는데, $C_{out}^{1}=1$ 이고 $C_{out}^{0}=0$ 이면 캐리 값이 1임을 나타내고 $C_{out}^{1}=0$ 이고 $C_{out}^{0}=1$ 이면 캐리 값이 0임을 나타낸다. 두 신호가 모두 0이면 캐리 입력이 전달되지 않았다는 것을 의미한다. 따라서 동작 완료 감지 회로에서는 그림 6과 같이 두 신호중의 하나가 1이 되었는지를 검사함으로써 동작 완료를 감지하였다. 논리적으로 그림 6의 동작 완료 감지 회로 32개를 직렬로 연결하면 전체 회로의 동작 완료를 감지할 수 있지만 이 방법은 많은 시간이 소요되므로 현실적이지 않다. 따라서 이 논문에서는 각 비트의 동작 완료 신호를 트리 구조로 전달하였다.



(그림 6) 동작 완료 감지 회로
(Fig. 6) Completion Sensing Circuit

참고 문헌 [7]의 비동기 가산기도 리플 캐리 방식을 사용하였으며 전체적인 구조는 그림 4와 같고 한 비트의 가산기의 구조는 그림 7과 같이 설계되었다. 이 가산기는 CMOS 논리를 사용하여 구현되었으며 합과 캐리 출력 둘다 이중 레일로 구현하였다. 또한 delay-insensitive 지연 모델을 만족하도록 설계하였다.

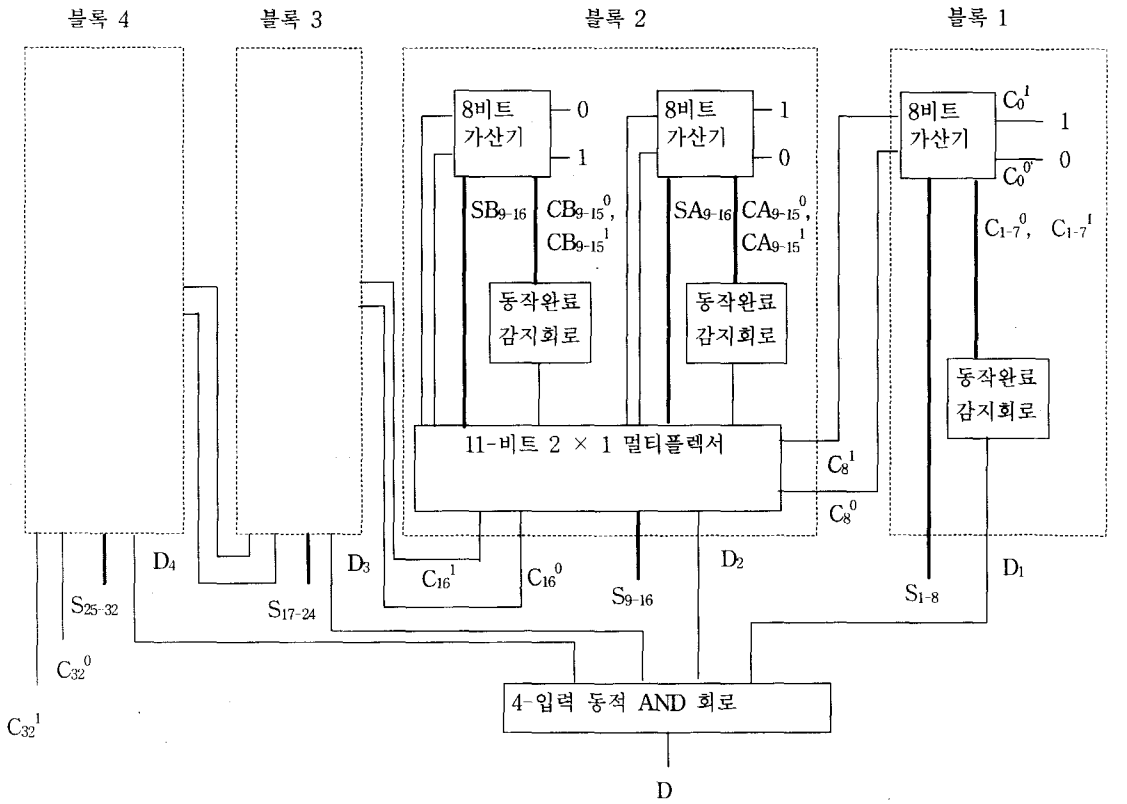


(그림 7) 전가산기의 구조[7]
(Fig. 7) Full adder circuit[7]

3. 비동기 캐리 선택 가산기

캐리 전송에 소요되는 시간을 줄이기 위한 방법으로 캐리 선택 방법을 사용할 수 있다. 본 논문에서는 그림 8에 나타나 있는 구조의 캐리 선택 가산기를 설계하였다. 이 가산기에서는 32비트를 8 비트씩 4 개의 블록으로 나누고 최하위 블록을 제외한 블록에 대하여 캐리 입력이 1인 경우와 0인 경우를 모두 계산한다. 즉, 하위 비트로부터의 캐리 입력이 0 또는 1 두 가지 중의 하나이므로 먼저 두 경우를 모두 계산한 다음에 하위 블록의 캐리 출력의 값에 따라서 두 결과 중의 하나를 선택한다. 이와 같이 4개의 블록에서 8비트 덧셈 연산을 동시에 수행하고 그 다음에 하위 블록의 캐리 출력 값에 따라서 계산 결과를 선택해 나감으로써 캐리 전송 시간을 줄일 수 있다.

그림 8의 캐리 선택 가산기에서 8비트 가산기는 전



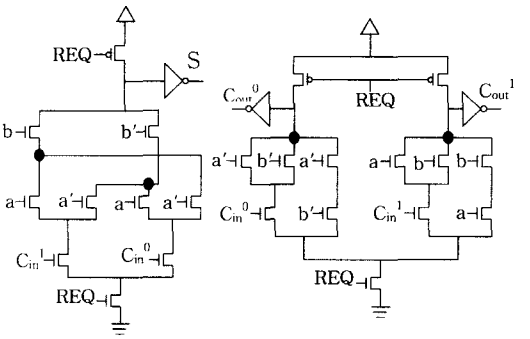
(그림 8) 비동기 캐리 선택 가산기의 구조
(Fig. 8) The structure of the asynchronous carry select adder

가산기를 리플 캐리 방식으로 연결함으로써 구현하였다. 각각의 전가산기에 대한 논리식은 진리표로부터 간소화 과정을 거쳐서 다음과 같이 구할 수 있다. 그리고 이 논리식은 그림 9와 같이 동적 논리[9]를 사용하여 구현하였다.

$$S = C_{in}^{0}a'b + C_{in}^{0}ab' + C_{in}^{1}a'b' + C_{in}^{1}ab$$

$$C_{out}^{1} = C_{in}^{1}a + C_{in}^{1}b + ab$$

$$C_{out}^{0} = C_{in}^{0}a' + C_{in}^{0}b' + a'b'$$

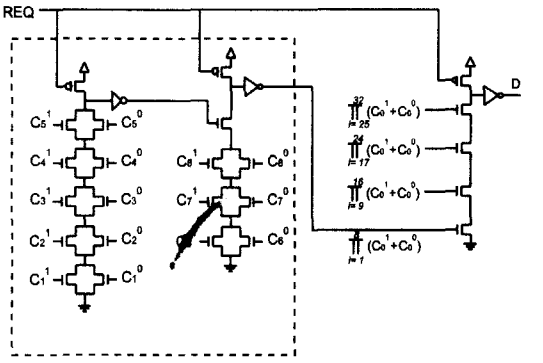


(그림 9) 전가산기의 구현
(Fig. 9) An implementation of a full-adder

그리고 각 8비트 가산기마다 동작 완료 감지 회로를 사용하여 동작 완료를 감지하였다. 첫 번째 블록의 동작 완료는 다음과 같이 모든 비트들의 캐리 출력으로부터 구할 수 있다.

$$D_1 = (C_1^0 + C_1^1)(C_2^0 + C_2^1) \cdots (C_8^0 + C_8^1)$$

본 논문에서는 동작 완료 감지 회로를 그림 10과 같이 동적 논리를 사용하여 구현하였다. 이 구현에서는 동작 완료 감지에 소요되는 시간을 줄이기 위하여 트리 형태로 구현하였고 먼저 값이 결정되는 신호들을 GND에 가까이 위치시켰다. 블록 2, 3, 4에서는 두 8비트 가산기 각각의 동작 완료 신호를 블록 1과 같은 방법으로 구한 다음에, 하위 블록의 캐리 출력에 따라 둘 중의 하나를 선택하여 D_2, D_3, D_4를 구했다. 그리고 동적 논리를 사용한 AND 게이트를 사용하여 $D = D_1D_2D_3D_4$ 를 구함으로써 전체적인 동작 완료를 감지하였다.

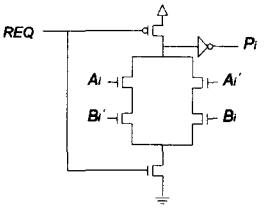


(그림 10) 동작 완료 감지 회로
(Fig. 10) An implementation of completion detection circuit

그런데, 캐리들이 병렬로 생성되지만 최종적으로 캐리가 선택되는 것은 순차적으로 수행된다. 따라서 C_{8k} (k=1,2,3) 값은 검사할 필요가 없는데, 이는 이들 값이 상위 블록의 캐리를 선택하는 데에 사용되기 때문이다. 즉, C_{32}^{1}과 C_{32}^{0} 값이 결정되기 위해서는 이들 값이 이미 결정되어 있어야 한다. 따라서, C_{32}^{1}과 C_{32}^{0} 신호가 완료되었는지 만 검사함으로써 회로를 간소화시켰다.

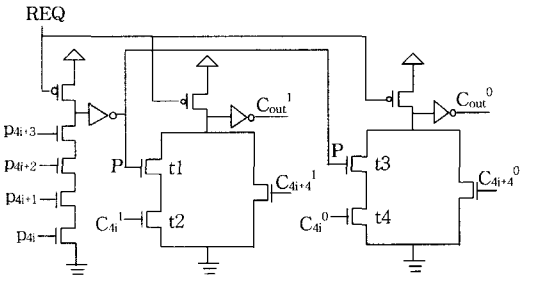
4. 비동기 캐리 우회 가산기

캐리 우회란 연속된 전가산기들이 캐리를 전송해야 하는 경우에 이들은 연속된 전가산기들을 통과시키지 않고 건너뛰어 전송하는 것을 말한다. 캐리 우회 가산기를 구현하기 위해서는 캐리의 전파 여부를 결정하는 회로를 전가산기에 추가해야 한다. 캐리를 전파해야 하는 경우는 1과 0을 더하는 경우로서 논리식으로 표현하면 $P_i = A_i \oplus B_i$ 이다. 이 회로는 그림 11과 같이 동적 논리를 사용하여 구현하였다.



(그림 11) Pi 의 구현
(Fig. 11) An implementation of Pi

본 논문에서는 4비트 단위로 캐리를 우회시키는 회로를 그림 12와 같이 구현하였다. 그림에 나타나 있는 바와 같이 4개의 전가산기의 P_i 신호가 모두 1일 경우에는 입력 캐리, $C_{4i}^1(C_{4i}^0)$가 출력 캐리, $Cout^1(Cout^0)$에 전달되도록 하였고 그렇지 않은 경우에는 각 전가산기를 거쳐서 생성된 캐리, $C_{4i+4}^1(C_{4i+4}^0)$가 전달되도록 하였다. 즉, p_{4i}, p_{4i+1}, p_{4i+2}, p_{4i+3}이 모두 1이면 4개의 연속된 비트들에서 모두 입력 캐리를 출력 캐리로 전송해야 하므로 4i 번째 비트의 입력 캐리인 $C_{4i}^1(C_{4i}^0)$를 곧바로 4i+5 번째의 캐리 입력으로 전송하는 것이다.



(그림 12) 캐리 우회 회로
(Fig. 12) Carry bypass circuit

P_i 신호가 모두 1이면 그림에서 인버터 출력 P의 값이 1이 되어서 t1과 t3가 ON 되고, $C_{4i}^1(C_{4i}^0)$의 값이 1이면 트랜지스터 t2(t4)가 ON되어서 $Cout^1(Cout^0)$의 값이 1이 되고 $C_{4i}^1(C_{4i}^0)$의 값이 0이면 트랜지스터 t2(t4)가 OFF 되어서 $Cout^1(Cout^0)$의 값이 0이 된다. 즉, P의 값이 1이면 $Cout^1(Cout^0)$의 값은 $C_{4i}^1(C_{4i}^0)$ 값과 같게된다.

P_i 신호 중의 하나가 1이 아니면 캐리가 우회되지 않고 4i+4 번째 비트의 캐리 출력이 4i+5 번째의 캐리 입력으로 전송되어야 한다. 이때에는 P 값이 0이 되므로 트랜지스터 t1와 t3가 OFF 되어서 $C_{4i}^1(C_{4i}^0)$의 값은 $Cout^1(Cout^0)$의 값에 아무런 영향을 미치지 않고 이 값은 $C_{4i+4}^1(C_{4i+4}^0)$ 신호의 값과 같게 된다.

5. 실험 결과

본 논문에서는 비동기 논리의 캐리 선택 가산기, 캐리 우회 가산기를 CMOS 도미노 논리를 이용하여 설계하였다. 표 3에는 설계한 회로에 대한 SPICE 시뮬레이션 결과가 나타나 있다. 이 실험에서는 1.2 μm 이중-메탈 공정을 사용하였으며 25°C 에서 5V 전원을 사용하였다. 트랜지스터의 크기는 대부분의 경우에 p-형 트랜지스터는 $L=1.2\mu$ $W=7.2\mu$ 를 사용하였고 n-형 트랜지스터는 $L=1.2\mu$, $W=2.4\mu$를 사용함으로써 채널 크기 비율이 3:1이 되도록 하였다.

기존의 비동기 가산기로서는 영국 맨체스터 대학에서 제안한 회로[6]와 미국의 CalTech에서 제안한 회로[7]에 대해 실험하였다. 참고 문헌에 나와있는 데이터는 오래된 공정을 사용하였기 때문에 연산 시간이나 전력 소모가 매우 커서 비교하기가 어려웠다. 따라서 본 논문에서 사용한 공정이나 트랜지스터 크기 비율 등을 똑같이 적용하여 시뮬레이션을 수행하였다.

모든 경우에 대한 실험은 불가능하므로 여기에서는 가장 긴 캐리의 길이가 다른 몇가지 경우의 덧셈을 시뮬레이션해 봄으로써 전력 소모량과 연산 시간을 비교하였다. 시뮬레이션 도구로는 HSPICE[10]를 사용하였다.

표 3의 실험 결과에 의하면 기존의 비동기 가산기의 경우에 연산 시간이 캐리의 길이에 따라 많은 차이가 있음을 알 수 있다. 그러나 캐리 우회 가산기와 캐

〈표 3〉 실험 결과
〈Table 3〉 Experimental results

A	B	연산 시간 (단위 : ns)				전력 소모량 (단위 : 10^{-11}w)				캐리 길이
		참고문헌 [6]	참고문헌 [7]	캐리 우회	캐리 선택	참고문헌 [6]	참고문헌 [7]	캐리 우회	캐리 선택	
0	0	1.89	1.94	1.97	1.96	3.25	13.4	4.16	6.68	0
0xFF000000	0	3.87	3.79	2.85	3.51	4.22	13.2	5.17	7.28	8
0xFFFF0000	0	6.53	6.07	3.53	3.51	5.17	13.6	6.16	7.89	16
0xFFFFFF00	0	9.02	9.5	4.04	3.77	6.11	14.2	7.14	8.50	24
0xFFFFFFFF	0	11.7	12.1	4.54	3.90	7.06	14.6	8.14	8.89	32

리 선택 가산기에서는 캐리가 길어질 수록 전체 비트들 중에서 병렬적으로 연산을 수행하는 곳이 많아짐으로써 캐리 길이가 증가되어도 연산 시간이 그에 비례하여 증가하지 않는 것을 알 수 있다. 그리고 캐리가 전혀 없는 경우에는 오히려 리플 캐리 가산기의 연산 시간이 보다 빠른데, 이는 캐리 우회와 캐리 선택을 위하여 부가적으로 첨가되는 회로로 인하여 지연 시간이 생기기 때문이다.

전력 소모의 경우에는 참고 문헌[6]의 리플 캐리 가산기의 전력 소모량이 가장 작고 캐리 선택 가산기가 가장 많음을 알 수 있다. 이는 캐리 우회와 캐리 선택을 위하여 추가되는 회로에서 발생하는 신호 전이로 인하여 전력 소모가 증가하기 때문이다. 그런데 세 가지 경우 모두 덧셈 연산이 필요한 경우에만 전력을 소모하지 그렇지 않은 경우에는 전력을 소모하지 않는다. 참고문헌 [7]의 가산기 경우에는 정적인 CMOS 논리를 사용함으로써 전력소모량이 증가하였다.

회로의 면적을 정확하게 비교하기 위해서는 회로의 레이아웃을 설계해야 하지만 본 논문에서는 레이아웃 단계까지 설계하지는 못하였다. 따라서 회로의 트랜지스터 수에 의하여 대략적인 면적을 비교해 보았다. 참고 문헌 [6]의 가산기의 트랜지스터 수는 1159개, 참고문헌 [7]의 가산기는 1383개, 캐리 우회 방식의 가산기는 1439개, 캐리 선택 방식의 가산기는 2006 개이다. 이와 같이 리플 캐리 가산기에 비하여 캐리 우회 방식이나 캐리 선택 방식의 가산기의 회로 면적이 증가되었지만 속도의 감소 비율보다는 적게 증가했음을 알 수 있다.

6. 결 론

본 논문에서는 동적 논리를 이용하여 비동기 캐리 우회 방식과 캐리 선택 방식의 고성능의 비동기 가산기를 설계하였다. 또한 성능향상을 위하여 동작 완료 감지 회로를 트리 구조로 설계하였으며 빨리 들어오는 입력은 GND 단자에 가깝게 위치시켰다. 실험 결과에 의하면 기존의 비동기 가산기 보다 본 논문에서 제안한 가산기들이 연산 시간 면에서 보다 빠름을 알 수 있었다. 그러나 부가되는 회로로 인하여 전력 소모는 더 증가됨을 알 수 있었다. 따라서 본 논문에서 제안한 비동기 가산기는 면적이나 전력 소모 보다 속도가 중요한 경우에 사용될 수 있을 것이다.

참 고 문 헌

[1] "Synthesis of Self-Timed VLSI Circuits from Graph Theoretic Specifications," PhD Thesis, Massachusetts Institute of Technology, 1987.

[2] T.H.-Y. Meng, R.W. Broderson and D.G. Messerchimit, "Automatic Synthesis of Asynchronous Circuits from High-level Specifications," IEEE Trans. on Computer Aided Design, Vol.8, No.11, pp.1185-1205, Nov. 1989.

[3] J. V. Woods, P. Day, S. B. Furber, J. D. Garside, N. C. Paver and S. Temple, "AMULET1: An Asynchronous ARM Processor," IEEE Trans. on Computer, Vol.46, No.4, pp.385-398, April 1997.

[4] A.J. Martin, A. Lines, R. Manohar, M. Nystroem, P. Penzes, R. Southworth and Uri Cummings, "The Design of an Asynchronous MIPS R3000 Microprocessor," Advanced Research in VLSI, pp.64-181, Sep. 1997.

[5] A. Bellaouar and M.I. Elmasry, "Low-power Digital VLSI Design," Kluwer Academic Publisher, 1995.

[6] J.D. Garside, "A CMOS VLSI Implementation of an Asynchronous ALU," IFIP Transactions, Vol. A-28, pp.181-207, 1993

[7] A.J. Martin, "Asynchronous Datapaths and the Design of an Asynchronous Adder," Technical Report CS-TR-91-08, Computer Science Department, California Institute of Technology, 1991.

[8] C.L. Seitz, "System timing," in Introduction to VLSI Systems, Addison-Wesley, 1980.

[9] N.H-E. Weste and K. Eshraghian, "Priciples of CMOS VLSI Design," Addison-Weseley, 1985.

[10] Meta-Software, Inc., "HSPICE User's Manual: Simulation and Analysis Volume 1," Meta-Software, Inc., 1996.

정 성 태

stjung@cs.wonkwang.ac.kr

1987년 2월 서울대학교 컴퓨터공학과 졸업(학사)

1989년 2월 서울대학교 대학원 컴퓨터공학과(공학석사)

1994년 8월 서울대학교 컴퓨터공학과(공학박사)

1994년 9월~1995년 2월 한국전자통신 연구소 박사후 연수연구원

1995년 3월~현재 원광대학교 컴퓨터공학과 교수

관심분야 : VLSI/CAD, 비동기 회로 설계, 컴퓨터 그래픽스