Redundant 십진코드를 이용하여 십진 자리간 Carry 전파를 제거한 십진 Adder 설계

論 文
55D-11-4

A Design of the Redundant Binary Coded Decimal Adder for the Carry-Free Binary Coded Decimal Addition

諸 政 玟* · 鄭 台 相†
(Jung-Min Je · Tae-Sang Chung)

Abstract - In the adder design, reduction of the delay of the carry propagation or ripple is the most important consideration. Previously, it was introduced that, if a redundant number system is adopted, the carry propagation is completely eliminated, with which addition can be done in a constant time, without regarding to the count of the digits of numbers involved in addition. In this paper, a RBCD(Redundant Binary Coded Decimal) is adopted to code 0 to 11, and an efficient and economic carry-free BCD adder is designed.

Key Words : carry-free addition, redundant number systems, binary coded decimal number, BCD adder

1. 서 론

컴퓨터 연산의 기본은 가산이며, 우리가 수작업으로 가산을 할 경우와 같은 하드웨어 구조에서는 ripple carry에 의하여 두 수의 bit 수에 선형으로 비례하는 연산시간이 걸리는데, 따라서 빠른 연산을 위하여 carry의 ripple에 따른 지연을 줄이는 방법이 연산이론 혹은 가산기 설계 연구의 주를 이루고 있다. 가산기 설계에 있어서 carry의 처리와 관련하여 정립된 중요 연구 결과로 carry-look-ahead adder, carry-skip adder, carry-select adder 등이 있다[1-4]. 이 방식들은 carry ripple에 따른 지연을 줄이기 위하여 추가적인 하드웨어를 사용하지만, 처리하는 수의 bit 수를 n이라고 할 경우, 일반적으로 bit 수에 따라 log(n) 지연 시간을 가지게 되어, ripple carry 방식의 kn 지연 시간 보다 월등히 좋은 결과를 준다.

Ripple carry 문제를 근본적으로 해결하기 위하여 RNS (Redundant Number System)가 개발되었다[5]. 이 수 법칙에 의하면 연산 속도는 연산이 처리하는 수의 bit 수에 전혀 관련이 없게 된다[6-7]. 전통적인 non-redundant 수 체계에서는 base가 b라고 할 경우 각 자리는 0부터 b-1까지 꼭 b개의 계수를 가지게 되고, 어떠한 수도 유일하게 표현된다. 반면, b 진법에서 계수가 b 개 보다 더 많이 가질 수 있게 하는 수 체계를 고려하여 보자. 예를 들어 십진법 수 체계에서 계수를 0부터 9까지 대신에 0부터 11까지 허용하면 수 표현이 유일하지 않게 되며, 이러한 수 체계를 RNS이라고 한다.

RNS 연산에서 ripple carry 문제가 완전히 제거되는 것을 **그림 1**을 이용하여 다음과 같이 간략하게 설명할 수 있다.

0부터 11까지의 계수를 허용하는 redundant 십진법을 고려하여 보자. 연산이 n digit의 두수를 처리한다고 하는 경우, 첫 단계로 각 자리에서는 하단(오른편)에서 전달될 수 있는 carry를 포함하지 않고 각각 독립적으로 가산을 하면 0부터 22까지의 사이의 수가 된다(**그림 1**의 "4bit Adder" block). 이 결과를 중간합 이라고 하자.

이 계산에 소요되는 시간은 두수의 자리의 수와 전혀 관련이 없다. 다음 단계에서, 중간합 즉 0부터 22까지의 수를 상단(왼편)으로 전달되는 carry[0..2]와 조정합[0..9]을 만들어준다(**그림 1**의 "Intermediate Reformation" block). redundant 수 체계를 도입함으로써 이러한 재구성이 가능하고, 재구성 방법은 **표 1**에 나타나 있다.

중간합을 carry와 조정합으로 재구성하는 하드웨어의 과정도 모든 자리가 각각 서로 이웃한 자리와 관계없이 상수 시간 안에 처리된다. 마지막 단계는 하단에서 올라오는 carry[0..2]를 조정합[0..9]에 가산하여 최종합[0..11]을 구하는 것으로(**그림 1**의 "Carry Adder" block), 이는 허용한

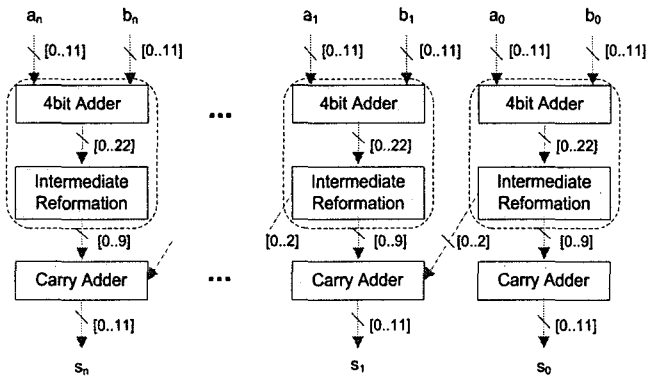

그림 1 10진수 Carry-Free Adder의 구조
Fig. 1 Decimal Carry-Free Adder Structure

† 교신저자, 正 會 員 : 중앙대학교 전기전자공학부 교수
E-mail : tschung@digital.cau.ac.kr
* 學生會員 : 중앙대학교 전기전자공학부 석사과정
接受日字 : 2006年 9月 13日
最終完了 : 2006年 10月 10日

표 1 RBCD 중간합의 재구성
Table 1 Reformation of Intermediate RBCD Sum

Intermediate Sum			Reformation		Intermediate Sum			Reformation	
dec	cb	sb	C_{RBCD}	S_{RBCD}	dec	cb	sb	C_{RBCD}	S_{RBCD}
0	0	0000	0	0000	12	0	1100	1	0010
1	0	0001	0	0001	13	0	1101	1	0011
2	0	0010	0	0010	14	0	1110	1	0100
3	0	0011	0	0011	15	0	1111	1	0101
4	0	0100	0	0100	16	1	0000	1	0110
5	0	0101	0	0101	17	1	0001	1	0111
6	0	0110	0	0110	18	1	0010	1	1000
7	0	0111	0	0111	19	1	0011	1	1001
8	0	1000	0	1000	20	1	0100	2	0000
9	0	1001	0	1001	21	1	0101	2	0001
10	0	1010	1	0000	22	1	0110	2	0010
11	0	1011	1	0001					

redundant 수의 계수 범위에 들게 되어 상단으로 가는 추가의 carry가 발생 없이 합산이 종결된다. 이 과정은 모든 자리가 각각 하단의 carry를 사용하지만 이 carry들은 동시에 만들어짐으로 이웃한 자리와 관계없이 상수 시간 안에 처리된다.

위 RNS을 이용한 carry-free 가산의 설명에서 몇 가지의 사항을 지적하면 다음과 같다.

첫째, 위 문장에서 중간합과 중간합의 재구성 부분을 직렬로 구분하는 것으로 설명하였으나, 이는 단지 이해를 돕기 위한 것으로 실제로는 한 과정으로 처리할 수 있는 연산 구조가 된다. 둘째, 연산을 위하여 외부에서 입력되는 non-RNS 숫자는 내부에 합당한 RNS 숫자로 변환되어야 하는데, 이 경우 일반적으로 redundancy를 위하여 수 표현에 있어서 bit 수를 추가하여야 한다. 예를 들어 0부터 15까지를 사용하는 16진법의 수 체계는 한자리 수를 표현하기 위하여 4 bit를 요구하지만 0부터 19까지를 허용하는 redundant 16진법은 5 bit을 요구하게 된다. 셋째로 RNS 연산에서는 컴퓨터 내부에서는 모든 수가 RNS 형태로 저장되며, 연산의 결과도 RNS가 되며, 위에서 설명한데로 RNS 하에서 연산이 반복될 경우에만 carry-free 가산의 장점이 충분히 나타난다. 하지만 반복되는 연산의 최종 결과를 외부에 출력할 경우, RNS를 non-RNS로 바꾸어야 하는데, 이 경우는, 필요하다면 기존의 carry-look-ahead 방법 등을 사용함으로써 carry의 지연 속도를 어느 정도 줄일 수는 있지만, carry의 ripple을 완전히 피할 수는 없다.

인간은 일반적으로 수를 십진법으로 인지하도록 교육되고 훈련되었으므로, 십진법은 인간에게 가장 편한 수 체계이다. 더구나 십진수 표현법은 직관적으로 값을 알 필요가 있거나, 십진수에서 십진 이외의 수체계로의 변환, 연산, 그리고 다시 십진수로 환원하는 과정에서 오차를 허용하지 않는 분야, 혹은 십진수의 고정 소숫점 표현법을 사용하는 분야(fixed-point number system) 등, 예를 들어 금융 분야나 상업 분야에서 널리 사용되고 있다. 컴퓨터에서 십진수 연산을 위하여 4 bit의 binary code를 사용하는데 대표적인 것이 BCD (Binary-Coded-Decimal)이다.

Shirazi, Yun 그리고 Zhang[8]에 의하여 -7부터 7까지 15개의 숫자를 사용하는 RBCD(Redundant Binary Coded Decimal) 가산기의 설계가 이루어 졌다. 이 논문의 특징은 signed-digit 형태를 취하여 각 자리에 음수를 허용하고 있으나, 그렇지 않은 것에 비하여 장점이 없으며, 또한 허용 숫자의 범위가 일반 BCD(0부터 9까지)의 수 범위를 포함하지 않으므로, 외부 BCD 입력을 내부 RBCD로 변환하는 과정이 필요하며, 출력을 위하여 RBCD를 BCD로 변화함에 있어서 signed-digit인 관계로 회로가 복잡하여진다.

일반 BCD는 가능한 4 비트 code 16개 중 10개만을 사용하므로, 이 범위를 포함하는 범위는, 예를 들어 0부터 11까지를 사용하면, 비트 수의 추가가 없이 redundant 십진수를 구현할 수 있으며, 이 경우 BCD 입력을 RBCD로 변환하는 과정이 필요없다. 본 논문은 0부터 11까지를 사용하는 RBCD 가산기를 설계한다. 하드웨어 설계는 연산의 논리를 이해하고 신호의 흐름을 잘 파악하기 위하여 gate level에서 schematic으로 구현하였으며, Altera사의 FPGA 구현 프로그램인 MAX+plus II[9] 이용하여 성능, 면적 등을 검증하였다.

본 논문의 본론에서 RBCD Adder의 구조와 RBCD-BCD 변환기의 구조를 기술하고, 기존에 발표된 다른 Redundant BCD Adder[8]과의 성능 및 하드웨어 복잡도(면적)의 비교 분석을 통해서 본 논문에서 제안한 방식의 장점을 설명한다.

2. 본 론

2.1 RBCD 가산기의 구조 및 설계

BCD 수 표현법은 4 비트로 구성되어 있다. 4 비트로 표현 가능한 code는 16가지이므로 BCD에서는 6개의 code가 남아 있으므로 비트 수의 추가 없이 4 비트만으로 redundant 십진수를 수용할 수 있는데, 본 논문에서는 최소한의 redundancy만을 허용하는 0과 11까지의 code를 사용하고자 한다. 수 0과 11까지의 RBCD는 0과 9까지의 수를 포함하므로, 외부에서 입력되는 일반 BCD 수는 바로 RBCD가 되어 추가적인 변환 하드웨어가 필요하지 않다. 본 논문에서 사용하는 RBCD는 일반 BCD를 포함하며 **표 2**에 나타나 있다.

표 2 RBCD 표현
Table 2 RBCD Representation

Decimal	RBCD	Decimal	RBCD	Decimal	RBCD
0	0000	4	0110	8	1000
1	0001	5	0111	9	1001
2	0010	6	1000	10	1010
3	0011	7	1001	11	1011

최소 redundancy를 허용하는 0부터 10까지를 사용하는 RBCD[0..10] 대신에, 본 논문에서 0부터 11까지 사용하는 RBCD[0..11]을 채택한 이유는 RBCD[0..10]의 경우는 carry-free 가산을 할 수 없기 때문이며, 이에 대한 설명은 다음과 같다. 가산하는 두 개의 수가 각각 10인 경우 그 중간합은 20이 되며, 이를 carry와 조정합으로 재구성하는 경

우, carry 2에 조정합 0이 된다. 즉 carry가 2가 발생할 수 있음을 알 수 있다. 제 3단계에서 조정합과 하단에서 올라오는 carry(2가 가능)를 합산하여 추가로 발생하는 carry가 없이 최종합이 RBCD[0..10]의 범위에 있기 위하여서는 조정합이 0부터 8사이의 수가 되어야만 한다. 그런데 연산에서 제1단의 중간합이 19가 되는 경우도 있는데, 이 중간합은 제2단에서 carry 1과 조정합 9로 재구성할 수밖에 없다. 이를 정리하면 제3단에서 조정합이 9가 되고, 올라오는 carry가 2가 될 수도 있기 때문에 이 경우 최종합이 11이 되어 정해진 RBCD[0..10]의 범위를 벗어나므로 추가적인 carry가 발생하게 된다. 따라서 RBCD[0..10]은 carry-free 가산이 불가능하여짐을 알 수 있고, 따라서, 0부터 11까지를 사용하는 RBCD[0..11]가 carry-free 연산을 가능하게 하는 최소 redundancy를 사용함을 알 수 있다. 물론 0부터 12이상을 사용하는 RBCD를 구현할 수 있으며, 또한 논리식 간략화에 있어서 약간의 하드웨어적인 장단점을 찾을 수는 있지만, 이론상의 근본적인 장점은 찾을 수 없다.

채택한 RBCD는 0부터 11까지의 범위로 4 비트를 사용하여 표현하므로, carry-look-ahead 방법을 사용하는 빠른 4 비트 binary adder, 즉 74LS283 chip 구조를 사용하여 두 개의 RBCD 수의 합을 구할 수 있으며, 그 결과는 0부터 22까지 범위의 중간합이 되어 1 비트 carry c_b와 4 비트 sum s_b 등 합계 5 비트의 결과를 얻을 수 있다. 각 10진 자릿수 간의 carry 전달을 없애기 위하여 이 중간합을 **표 1**과 같이 10진 carry c_{RBCD}와 10진 조정합 s_{RBCD}값으로 구분할 수 있다. 여기서 조정합은 RBCD의 허용 최대수 11에서 최대 10진 carry 값 2를 수용하는 것을 고려하여 9까지를 허용하여야 한다.

표 1에서 중간합 0부터 22까지의 5 비트 binary 수에서 10진 carry와 조정합을 구하는 방법은 5변수 논리식 축약법(random logic)을 사용하는 것이 회로를 가장 간략화 할 수 있다. 그러나, BCD에서 carry와 합의 관계를 잘 이해하기 위하여 다음과 같이 합의 구간을 decoding하여 중간합의 결과에 0(0000), 6(0110) 혹은 12(1100)를 더하여 주어도 된다. 첫째 중간합이 0부터 9까지의 경우(x)는 조정이 필요 없다. 즉 0(0000)을 더하면 된다. 둘째, 중간합이 10부터 19까지의 경우(y)는 c_{RBCD}가 1이 되며, s_{RBCD}는 s_b에 6 즉 0110을 더하면 된다. 마지막으로 합이 20에서 22까지의 경우(z)는 c_{RBCD}가 2가 되며, s_{RBCD}는 s_b에 12 즉 1100을 더하면 된다.

예로서 중간합 21은 binary로 10101이므로 c_b=1과 s_b=0101이 되며, 이를 BCD carry 2와 조정합은 1로 재구성하여야 하는데, 조정합은 s_b=0101 즉 5에 12를 더하여 17이 되나 carry 16을 무시하고 4 비트 결과인 1만을 취하면 된다.
이를 종합하면 조정합은 아래의 4 비트 합산으로 가능하다. 여기서 decoding 논리식 y와 z만이 필요하다.

$$S_{RBCD} = S_{b(4bit)} + \{(z)(y+z)(y)(0)\}_{(4bit)}$$

제3단계에서는 BCD의 각 자리에서 하단에서 오는 carry[0..2]를 더하여 결과합을 구하는 회로이다. 중간합이 0부터 9까지이므로 최대 carry 2를 더하여도 그 결과는 0부터 11이 되어 정하여진 RBCD범위에 속하므로 추가적인 carry chain이 형성되지 않는다. 즉 carry-free 가산이 가능하다. 여기서 carry가 0, 1 혹은 2가 되므로 중간합에 carry

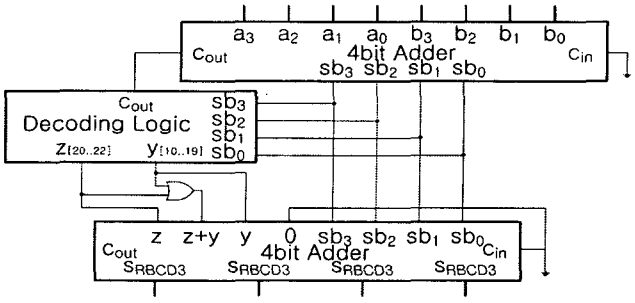

그림 2 '중간합의 재구성'회로의 다른 접근방식
Fig. 2 Another approach of 'Intermediate Reforamtion' Circuit

의 가산은 일반 가산기를 채택하는 것 보다는 간략한 increment회로를 사용하는 것이 효과적이다. 즉 1을 더하는 것은 일반 increment가 되고, 2를 더하는 것은 2^1 자리 즉 두 번째 자리에서부터 increment 하는 회로로 처리하면 된다.

2.2 RBCD-BCD 변환

외부에서 입력되는 BCD는 채택한 RBCD의 수 표현 범위 안에 있으므로, 별도의 변환이 없이 자체가 RBCD가 된다. 이런 RBCD 수들은 위의 연산방법에 의하여 carry-free 가산 즉 자릿수의 개수에 관계없이 상수 시간이 소요되는 가산을 할 수 있으며, 모든 수는 RBCD형태로 컴퓨터 메모리에 저장되고 반복적으로 처리된다. 하지만 이 결과를 외부에 출력하는 경우, 이를 유효한 BCD로 변환하여야 하는데, RBCD의 모든 자리는 0부터 11까지 범위의 수인데 변환과정에서, 어느 한 특정 자리에서 보면, 하단에서 오는 carry는 0 혹은 1이 되므로, 합하여 0부터 12까지의 범위의 수가 되며, 이를 상단으로 보내는 carry를 0 혹은 1과 자리 값으로 0부터 9를 남겨야 한다. 이런 과정이 최하단 으로부터 반복하여 연결하여야 하므로 변환기의 연결구조는 **그림 3**과 같이 carry ripple chain 형태가 되고, 따라서 RBCD에서 BCD로의 변환은 숫자의 자리의 수에 비례하는 시간이 소요되며 이를 피할 수는 없다.

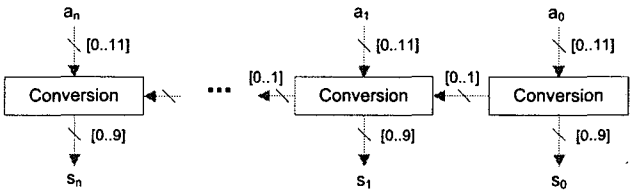

그림 3 RBCD에서 BCD로의 변환
Fig. 3 RBCD to BCD Conversion

2.3 검증 및 성능평가

설계한 RBCD 가산기는 Altera사의 MAX+plus II [9]를 이용하여 FPGA(FLEX10K EPF10K30BC356-3)로 구현하여 동작을 검증하였다. 가산기의 기능을 검증하기 위하여 임의의 두 값을 더하여 올바른 결과가 나오는지를 확인하였고, 가산기의 효용성(회로설계의 면적이나 회로 동작의 지연시간)은 Report파일과 Timing Analysis를 통해 사용된 LC의 개수와 최고 지연시간을 기준으로 작성하였다.

본 논문에서 제안한 RBCD[0..11]과 논문[8]에서 제안한 RBCD[-7..7], 그리고 ripple carry 연결방식의 BCD Adder의 회로를 구현하고 simulation을 통하여 가산을 할 경우의 측정된 지연시간과 회로의 사용면적이 **그림 4, 그림 5**에 그래프로 나타나있다.

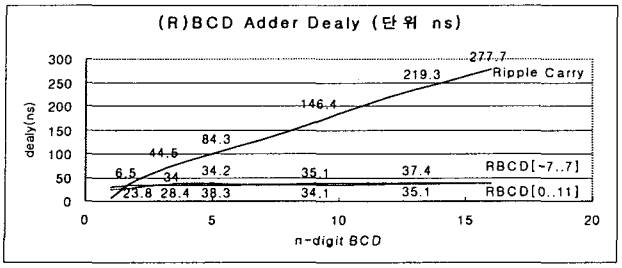

그림 4 3가지 종류의 (R)BCD Adder 지연시간 비교
Fig. 4 Delay comparison of three types (R)BCD adder

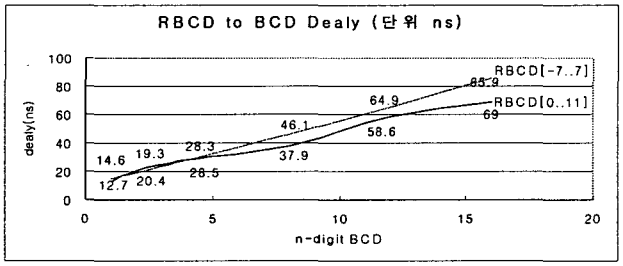

그림 5 3가지 종류의 (R)BCD Adder 사용면적 비교
Fig. 5 Area comparison of three types (R)BCD adder

그림 4에서 보면, 연산시간은 ripple-carry BCD 방식의 경우는 자리수의 증가에 따라 연산시간이 선형으로 증가함을 보여주는데 이는 이론적으로 예측한 것과 같다. 이에 반하여 본 논문에서 설계한 RBCD[0..11]은 두수의 합은 각 숫자의 자리수의 증가와 관계가 없이 일정함을 보여주는데, 이는 carry-free 가산의 이론상 합당한 것이다. 마찬가지로 RBCD[-7..7]도 같은 결과를 보여 주여 carry-free 가산임을 알 수 있으며, RBCD[0..11]과 RBCD[-7..7] 사이에는 지연시간의 큰 차이가 없다.

그림 5에서 3가지 방식으로 설계된 BCD 가산기들의 면적을 비교하여 보았다. 먼저, carry-free 방식이든, ripple carry 방식이든 각 digit의 회로는 동일하므로, 3방식 모두가 digit수가 늘어남에 따라 선형적으로 면적이 늘어남을 알 수 있다. 그러나 RBCD는 redundant code를 사용하는 관계로, 그리고 중간합을 재구성하는 과정과 중간합과 carry를 더하는 과정이 포함되므로 RBCD는 BCD보다 더 많은 면적을 차지함을 그래프 상으로 확인할 수 있다. 그리고 RBCD중 RBCD[0..11]이 RBCD[-7..7]보다 작은 면적을 사용함을 보여주고 있는데, 이는 사용하는 수의 범위가 다르고, 또한 redundancy의 여유도에 차이가 있는데, RBCD[0..11]은 2의 여유도인 반면에, RBCD[-7..7]은 5의 여유도가 있으므로, 중간합의 재구성을 위한 회로의 decoding 논리식이 RBCD[0..11]이 RBCD[-7..7] 보다 간단하게 되기 때문이다.

반복된 연산의 결과를 외부로 출력할 때 사용하는 RBCD to BCD회로에서는 RBCD[0..11]이 RBCD[-7..7]에 비해 16자릿수의 덧셈인 경우 기준으로 1ns/digit의 이점이 있었고 면적의 경운 2LC/digit의 이점이 있다. 이것역시 수 범위의 차이에서 오는 decoding 논리식이 RBCD[0..11]이 RBCD[-7..7]보다 간단하기 때문이다.

마지막으로 RBCD[-7..7]은 BCD to RBCD로 변환하는 회로가 필요한데 여기서 지연시간은 평균 24ns 면적은 1-digit당 14LC정도를 사용하여 본 논문에서 제안한 RBCD[0..11] 방식이 RBCD[-7..7]보다 효율적인 회로 구성임을 보여준다.

3. 결 론

BCD 연산은 금융 분야와 상업 분야가 커질수록 더 많이 사용될 가능성이 높은 분야이다. 그러나 일반적인 BCD 연산 방법은 carry의 ripple로 인해서 숫자가 커질수록 최종단의 결과까지 지연시간이 길어진다. 그러나 RNS(Redundant Number System)을 이용하여 carry의 전파가 필요없는 BCD 가산기를 설계할 수 있어, 고속의 병렬 연산이 가능하다. 본 논문에서 제안한 방식은 기존에 나온 다른 논문[8]과는 RBCD 가산기와 RBCD-BCD변환에서는 실행 속도에서 약간의 이득이 있으나 근본적이 차이로는 볼 수가 없다. 그러나 회로의 면적 자체에서 이점이 있었고, 본 논문에서 제안한 RBCD가 기존의 BCD를 포함하므로, 설계 구조가 쉽게 이해되며, 회로의 구성이 더 조직적이 되어, BCD-RBCD로의 변환 과정이 필요 없어서 최초 RBCD값으로의 변환에 필요한 면적 및 실행시간을 단축할 수 있다.

참 고 문 헌

[1] Ling, H., "High-Speed Binary Adder," IBM J. Research and Development, Vol. 25, No. 3, pp. 156-66, 1981
[2] Lehman, M., and N. Burla, "Skip Techniques for High-Speed Carry Propagation I Binary Arithmetic Units," IRE Trans. Electronic Computers, Vol. 10, pp. 691-8, 1961
[3] Bedriji, O.J., "Carry-Select Adder," IRE Trans. Electronic Computers, Vol. 11, pp. 340-6, 1962
[4] Kantabutra, V., "A Recursive Carry-Lookahead/Carry-Select Hybrid Adder," IEEE Trans. Computers, Vol. 42, No. 12, pp. 1495-9, 1993
[5] Metze, G., and J.E., Robertson, "Elimination of Carry Propagation in Digital Computers," Information Processing '59(Preceedings of a UNESCO Conference), 1960, pp. 389-396.
[6] Phahami, B., "Generalized Signed-Digit Number Systems: A Unifying Framework for Redundant Number Representation," IEEE Trans. Computers, Vol. 39, No. 1, pp. 89-98, 1990
[7] Phatak, D.S., "Hybrid Signed-Digit Number Systems: A Unifying Framework for Redundant Number Representations with Bounded Carry Propagation Chains," IEEE Trans. Computers, Vol. 43, No. 8, pp. 880-91, 1994
[8] Shirazi, B. and Yun, D.Y.Y. and Zhang, C.N., "RBCD: redundant binary coded decimal adder" IEE Proceedings Computers and Digital Techniques, Volume 136, Issue 2, Pages 156-160, Mar 1989
[9] "MaxPlusII Getting Started" http://www.altera.com/literature/manual/81_gs.pdf