컴퓨터공학실험2 6주차 예비 보고서

전공: 컴퓨터공학 학년: 2학년 학번: 20201635 이름: 전찬

**0. 목차**

1. 전 가산기 및 반 가산기에 대해 조사한다.

2. 전 감산기 및 반 감산기에 대해 조사한다.

3. BCD 가산기에 대해 조사한다.

4. 병렬 가감산기에 대해 조사한다.

5. Carry Look-Ahead Adder을 Ripple Carry Adder와 비교해서 설명한다.

**1. 전 가산기 및 반 가산기**

가산기란 Addition(덧셈)을 수행할 수 있는 회로이다. 이중 반 가산기는 A, B 2개의 1bit input에 대해서, sum 결과와 carry 결과를 output으로 갖는 회로이다. input을 A, B 라고 했을 때, input에 대한 output S, C(carry)의 논리표는 다음과 같다.

|  |  |  |  |
| --- | --- | --- | --- |
| 반 가산기(Half Adder) | | | |
| input | | output | |
| A | B | S | C |
| 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 |

<반 가산기의 논리표>

위 결과를 토대로 S는 A, B에 대해서 XOR gate를, C는 A, B에 대해 AND gate를 적용한 형태임을 알 수 있다. (assign S = A^B; assign C = A&B; 형태) 실제로 반 가산기에 input A = 1, B = 1이 들어온다면, 그 output S = 0, C = 1 임을 파악할 수 있다.

이와 다르게 전 가산기는 이전 자리 올림의 계산 결과 또한 input으로 받게 되는 가산기이다. 반 가산기와 다르게, 여러 전 가산기를 활용하면, 1bit에 국한되지 않고 자연수의 덧셈을 구현할 수 있다는 것이다. input A, B, c에 대한 output S, C는 아래와 같다

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 전 가산기(Full Adder) | | | | |
| input | | | output | |
| A | B | c | S | C |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 |

<전 가산기의 논리표>

전 가산기를 통해서 여러 bit의 덧셈을 할 수 있는데, 한 예시로 A = 11011011(219), B = 00011010(26) 의 덧셈을 수행해 보자. 우선 첫 번째 계산에서는(계산은 LSB부터 MSB 방향으로 진행한다.) A1 = 1, B1 = 0, c1 = 0 인 계산을 수행해 output으로 S1 = 1, C1 = 0 을 갖게 된다. 여기에서 S1은 그대로 남기고, C1 을 다음 계산에 사용될 c2와 동일하게 봐준다면 두 번째 계산은 A2 = 1, B2 = 1, c2 = 0 이며 S2 = 0, C2 = 1 이 된다. 또한 c3 = 1 이 된다. 이 형태를 계속해서 반복하면 A + B의 결과가 11110101(245) 이 됨을 전 가산기의 계산을 통해서 알 수 있다.(이는 사실 병렬 가산기의 계산으로 전 가산기를 연결해서 병렬 가산기를 만들 수 있음을 파악할 수 있다.)

추가로 여기에서 S = A^B^c 형태이며, C = AB+Bc+cA 형태이다.

**2. 전 감산기 및 반 감산기**

위에서 덧셈을 수행할 수 있는 회로인 전 가산기와 반 가산기를 설명한 것처럼, 뺄셈을 수행할 수 있는 회로 또한 존재하며, 이는 감산기 이다. 감산기 또한 반 감산기와 전 감산기 두 종류가 존재하는데, 이에 대해 알아보자.

우선 반 감산기는 위에서 반 가산기를 통해 1bit 덧셈을 수행한 것과 비슷하게, 1bit 뺄셈을 수행할 수 있다. input A, B에 대해서 output D, Br이 존재하는데, 여기에서 D는 가산기에서 S의 역할이며, Br은 가산기에서 다음 자릿수에 대한 Carry가 존재하는 것처럼, 다음 자릿수에서 빌려오는 Borrow를 의미하는 변수이다. 반 감산기의 논리표는 아래와 같다.

|  |  |  |  |
| --- | --- | --- | --- |
| 반 감산기(Half Subtractor) | | | |
| input | | output | |
| A | B | D | Br |
| 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 |

<반 감산기의 논리표>

여기에서 D = A^B, Br = A’B 형태이다. 실제로 input A = 1, B = 1인 경우 1 – 1 = 0 형태로 output D = 0, Br = 0임을 파악할 수 있다.

전 감산기 또한 전 가산기와 비슷하게, 이전에 빌린 br을 추가적인 input으로 받는 회로이다. 이에 대한 논리표는 아래와 같다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 전 감산기(Full Subtractor) | | | | |
| input | | | output | |
| A | B | br | D | Br |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 |

<전 감산기의 논리표>

위 결과는 A-B-br 형태이며, 전 가산기에서 해온 과정과 비슷하게, output으로 가진 Br이 다음 input의 br이 됨을 파악할 수 있다.

실제로 A = 1101, B = 0110 의 input에 대해 A – B를 수행한다고 하자.(LSB에서 MSB 방향으로 수행한다.) 우선 A1 = 1, B1 = 0, br1 = 0 이며 D1 = 1, Br1 = 0 이다. 따라서 다음 br2 또한 0이 됨을 파악할 수 있다. 이를 적용하면 A2 = 0, B2 = 1, br2 = 0 이며 D2 = 1, Br2 = 1 임을 파악할 수 있다. 계속해서 수행하면, A – B = 0111 이 됨을 파악할 수 있다.(이는 사실 병렬 감산기의 계산으로 전 감산기를 연결해서 병렬 가산기를 만들 수 있음을 파악할 수 있다.)

**3. BCD 가산기**

BCD 란 Binary Coded Decimal의 약자로, 4bit로 10 진수 한 자리를 표현하는 형태를 의미한다. 예를 들면, 11 또한 4bit(1011)로 표현 가능하지만, 0001 0001로 한 자릿수 마다 끊어서 표현하는 것이다. 이러한 BCD는 구현과 display할 때 장점이 존재하기 때문에, 현재까지도 많이 사용되는 하나의 방법이다. BCD 또한 덧셈이 존재하는데, 그 형태는 아래와 같다.

(1) 두 BCD를 input으로 받는다.(input A, B 는 0~9 이다.)

(2) 만약 그 합이 10 이상이라면(1010), 계산 결과에 6(0110)을 더해준다.

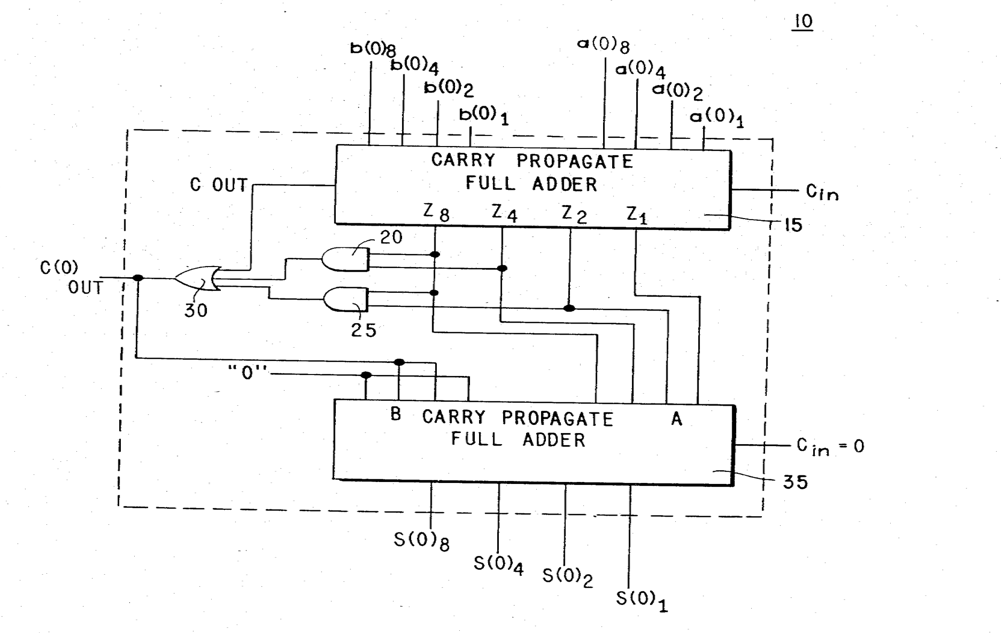
이 과정을 통해서 BCD 계산기를 구현할 수 있는데, 우선 전 가산기(4bit 병렬 가산기)를 통해서 A, B를 더해준 이후, 그 결과가 1010보다 큰지는 아래 경우들을 통해서 판단할 수 있다.

(1) 병렬 가산기의 총 output C가 1이면 그 수는 10보다 크다(10000b >= 16 > 10)

(2) 병렬 가산기의 총 output에서 S의 output 에서 MSB가 1인 경우, 2 번째 bit가 1인 경우 그 수는 10보다 크다.(11xx > 10 이기 때문에)

(3) 병렬 가산기의 총 output에서 S의 output 에서 MSB가 1인 경우, 3 번째 bit가 1인 경우 그 수는 10보다 같거나 크다.(1x1x >= 10 이기 때문에)

위 3개의 경우 중 한 경우만 만족한다면, 그 수는 10 이상이라고 할 수 있다. 따라서 위 형태를 OR gate를 통해서 묶어주며, 그 input으로 (1) 총 Carry, (2) 첫 번째와 두 번째 bit의 AND gate에 의한 output, (3) 첫 번째와 세 번째 bit의 AND gate에 의한 output을 대입해 준 이후 OR gate의 output이 1이라면, 0110을 더해주는 수행을 해주면 된다.(이는 4bit 병렬 가산기에서 다시 수행 가능하다.) 이렇게 설명한 BCD 가산기의 형태는 아래와 같다.

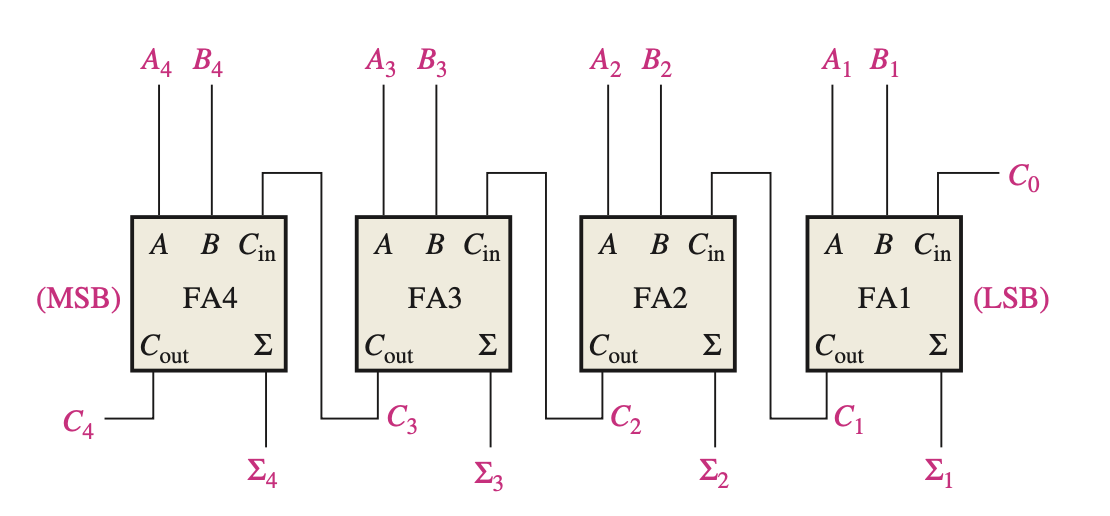


<BCD Adder의 회로 형태>1

**4. 병렬 가감산기**

위에서 전 가산기와 전 감산기의 예시를 풀어볼 때, input이 1bit가 아닌, 여러 bit(가산기에서는 8bit, 감산기에서는 4bit) 형태의 input 형태였다. 이렇게 1bit가 아닌 Bitstream 형태로 input이 들어올 때에 대해서 덧셈과 뺄셈을 수행할 수 있는데, 이를 병렬 가감산기라고 한다.

위에서 설명한 것처럼 LSB부터 시작하며, 가산기에서는 output으로 도출된 C(carry)가 다음 input의 c(carry) 가 되는 형태이다. 또한 감산기에서는 output으로 도출된 Br(borrow)가 다음 input의 br(borrow) 가 되는 형태이다. 병렬 가산기를 그림으로 표현하면 아래와 같다.



<병렬 가산기의 그림으로의 형태>2

**5. Carry Look-Ahead Adder와 Ripple Carry Adder의 비교**

4. 에서 설명한 병렬 가산기 형태는 LSB부터 MSB까지 각 carry를 구하며 계산한다는 점을 토대로 Ripple Carry Adder(리플 자리올림수 가산기) 라고 부르기도 한다. 이 병렬 가산기는 구현하기 쉽다는 장점이 존재하지만, LSB부터 MSB까지 선형적으로 계산을 수행해야 되기 때문에, 마지막 결과를 얻기까지는 많은 시간이 걸릴 수 있다. 이는 수많은 프로그램에서 수행하는 덧셈을 Ripple Carry Adder을 통해서 수행한다면 매우 비효율적인 결과가 나타날 수 있음을 의미한다.

위에서 말한 것처럼, LSB에서 MSB까지 올라오며, 선형적으로 구해내는 것은 비효율적이다. 그러면 이를 해결하기 위해서 모든 가장 MSB 부분에서의 Carry(그림에서는 병렬 가산기에서의 마지막 C4)를 한 번에 구해내는 방법이 있다면 이는 효율적이라고 할 수 있다. 실제로도 이러한 방법이 존재하는데, 바로 Carry Look-Ahead Adder(자리올림수 예측 가산기)이다.

이 방법을 간단하게 설명하자면, 모든 병렬 가산기의 일부분인 전 가산기에서는 S만 output으로 가지면 되며, 마지막 Carry만 추가적인 output으로 가진다면, 빠르게 덧셈 수행을 해낼 수 있는 형태이다. 여기에서 각 전 가산기의 S를 구하는 방법은 위에서 설명한 방법과 동일하기 때문에 생략한다. 마지막 Carry를 구해내는 방법이 중요한데, 이는 다음과 같은 아이디어를 통해서 표현해낼 수 있다.

(1) 마지막 Carry(위 그림에서 C4)는 Ripple Carry Adder의 관점에서 보자면, A4, B4, C3에서 C4 =A4B4 + B4C3 + C3A4 형태이다.

(2) 동일한 과정으로 C3 =A3B3 + B3C2 + C2A3 형태이다.

(3) C3 을 C4 의 우변에 대입하면, C4 = A4B4 + B4(A3B3 + B3C2 + C2A3) + C3A4 형태이다.

(4) 위와 같은 방법으로 계속해서 대입한다면, C0 = 0 이므로, C에 대한 항이 전부 제거되며, A, B의 조합으로 C4를 표현해낼 수 있다.

이처럼 진행해서 C4를 따로 구하고, 각각의 S1, S2, S3, S4 병렬적으로(한 순간에) 구해낸다면, 훨씬 빠른 속도의 가산기를 구현해낼 수 있다.

위 과정을 거친 Carry Look-Ahead Adder은 Ripple Carry Adder보다 훨씬 빠른 속도로 덧셈을 수행할 수 있다. 하지만 단점도 존재하는데, C4를 구해내기 위해서(input의 Bitstream 길이가 길어지면 길어질수록) 마지막 Carry를 표현하기 위한 gate가 수없이 많이 필요하게 된다. 이는 Bitsream의 길이 증가와 gate수의 증가가 선형으로 비례하는 Ripple Carry Adder보다 훨씬 더 많은 비용을 요구하는 단점이라고 할 수 있다.

**6. 출처**

1. BCD adder circuit의 특허, SCIENCE ON, 한국과학기술정보연구원,

<https://scienceon.kisti.re.kr/srch/selectPORSrchPatent.do?cn=USP1989024805131>

2. Thomas L. Floyd, Person, Digital Fundamentals Eleventh edition(global edition), 320p