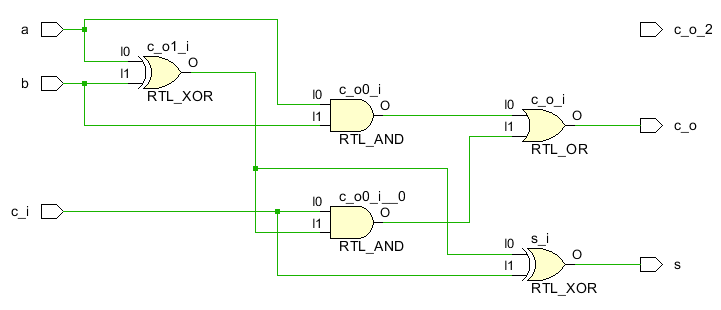
10주차 예비보고서

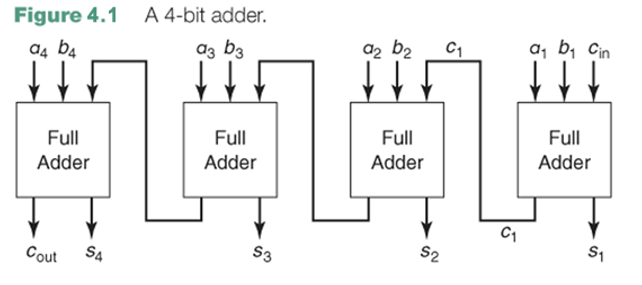
전공: 컴퓨터공학과 학년: 2학년 학번: 20191619 이름: 이동석

**1. 4-bit Adder 및 Subtractor 이진 병렬 연산 기능**

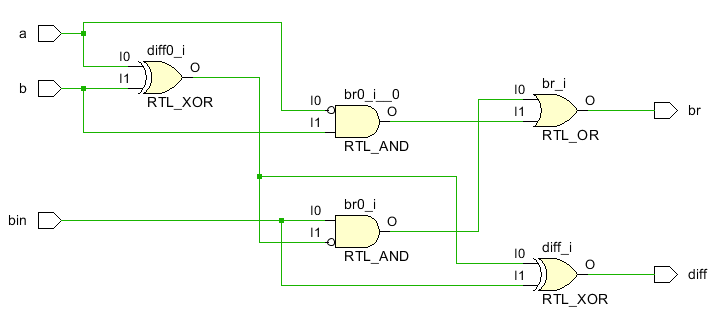
전가산기(Full Adder)와 반가산기(Half Adder)에 대해 공부하면서 4-bit 가산기에 대해 잠깐 언급한적이 있다. 4-bit 전가산기의 경우 4개의 비트에 대해서 가산기의 역할을하는 회로를 의미한다. 가산기란 입력이 들어왔을 때 각 자리의 비트의 합(sum)과 자리올림(Carry)을 출력하는 논리회로이다. 이때, 전 가산기는 이전 자리 올림까지 고려하는 가산기이다.



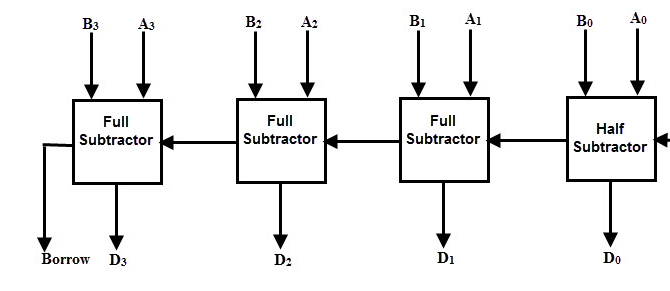
위는 1-bit 전가산기 이다. (두 개의 반가산기와 하나의 OR게이트) 4-bit에 대해 가산기의 역할을 하는 회로는, 4개의 1-bit 전가산기를 병렬로 연결하면 된다. 참고적으로, 1개의 전가산기에서 C-out = BC + AC + AB or AB + AC + BC ( A + A’) 이며 S = A⊕(B⊕C) 이다.



Parallel Adder에서 가장 간단한 형태로 알려진 Ripple Carry Adder이다.



마찬가지로, 4-bit 감산기도 동일하다. 위의 1-bit 전감산기 4개를 연결하면, 4-bit에 대한 감산기의 역할을 수행할 수 있다. 감산기란, 두 개의 이상의 이진수를 입력받았을 때, 입력 값의 차(Difference)와 빌림 수(Borrow)을 출력으로 하는 논리 회로이다. 참고적으로, 1개의 전감산기에서 Diff = A⊕B⊕Bin, Br = A’Bin + A’B + BBin or Bin (A XOR B)’ + A’B가 된다.



**2. Look ahead carry**

Ripple Carry Adder는 앞서 소개한 바와 같이 4개의 전가산기를 연결한 병렬 가산기의 일종이다. 자릿수가 ripple(물결)치듯 움직인다 하여 이런 이름이 붙었다. 또한, 이렇게 연속적으로 순서로 연결되어있는 방식을 cascade로 연결되어있다(carry chain)라고 한다. 이런 방식은 간단하게 구현이 가능하지만, 이전 carry의 연산이 다음 비트의 연산에 관여를 하기 때문에 최종 출력이 나오기 까지 시간이 많이 걸린다. 당연하게도 delay시간이 늘어나며, (이를 Critical path라고도 부른다.) 만큼의 delay가 생긴다. 이런 캐리 전파 지연 속도를 개선하면, 다른 산술 연산 속도 역시 빨라지게 되기 때문에 delay를 줄이는 것은 매우 중요하다. 따라서, 속도를 빠르게 하기 위해 Carry Look Ahead(미리 예측하는) 가 등장했다.

Carry Look-Ahead Adder의 기본 아이디어는, 입력받는 신호를 기반으로 캐리 신호를 **미리 계산**하여 문제를 해결하는 것이다. 생각해보면, 두 개의 입력값 A\_i와 B\_i의 캐리 여부는 바로 접근 가능하다. 따라서, 이 계산을 위해서 이전 비트의 캐리 값을 기다릴 필요가 없다.

들어가기 앞서 여러 정의와 표현을 이해하고 넘어가야 한다.

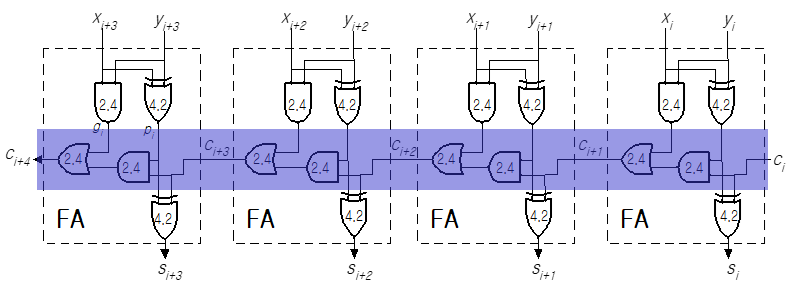
1. Carry-generate function : G\_i = A\_iB\_i -> A\_i와 B\_i 모두 1이면 이전 비트와 관계없이 무조건 캐리비트를 생성한다.
2. Carry Propagation function : p\_i = A\_i⊕B\_i -> 둘 중 하나만 1이면 이전 비트의 캐리(C-in)이 캐리(C-out)로 출력

이를 바탕으로 다음과 같은 표현이 가능하다.

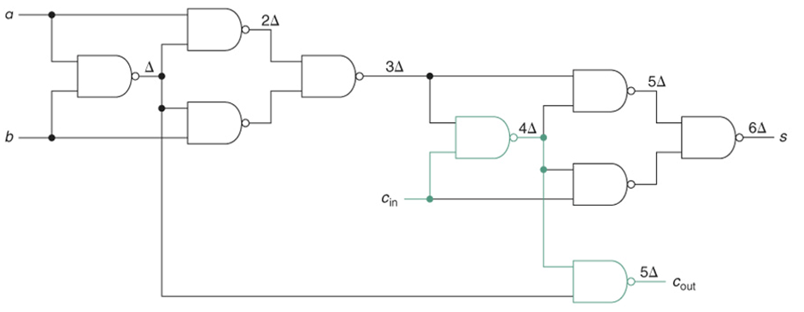
텍스트이(가) 표시된 사진

자동 생성된 설명

최종적으로 회로는 더욱 더 복잡하게 되지만, 속도는 향상된 다음과 같은 모습을 띈다. (C\_(i+4) 를 Carry-look-ahead-generator는 2 gate delay만 걸린다.)



하지만, 계산에 AND/OR게이트가 추가적으로 필요하므로 fan-in문제가 나타나곤 한다. (총 (n(n+1)/2 만큼의 and/or 게이트가 필요)



전가산기 하나를 추가할 때 마다 게이트 딜레이가 2△증가한다. Carry look ahead adder에서 g\_i와 p\_i는 1△ 증가한다. 그래서 1개의 Carry look ahead adder의 총 게이트 딜레이는, 1 + c\_i + s\_i 인데 c\_i는 2, s\_i는 3만큼의 딜레이가 있으므로 총 6만큼의 게이트 딜레이가 생긴다.

**3. XOR 2’s complement 가감산**

2의 보수는 어떤 비트 A에 대해 보수를 취한 뒤(1의 보수) 1을 더해주는 것이다. 이렇게 나온 값을 원래 A의 음수값으로 생각한다. 이런 Complment 방식과 XOR게이트를 사용하면, 앞서 설명했던 4-bit adder/subtractor를 한번에 할 수 있는 회로를 구현할 수 있다.

우선 뺄셈을 덧셈으로 생각해주는 방법을 알아야한다. A – B = A + (-B) 와 동일하기 때문이다. 예를 들어, 이진수 1111 과 0100의 뺄셈을 해보자. 이때, 0100의 2의 보수는 1011 + 1 = 1100 이다. 즉, 1111(2) + 1100(2) = 1011(2) = 15 + (– 4) = 11 로 생각할 수 있는 것이다. (4-bit 연산이므로, 넘어가는 bit는 무시한다.)

추가적으로 +연산을 할지 -연산을 할지 정하는 시그널 비트 S가 추가 된다. XOR은 1⊕A=A’, 0⊕A=A가 성립한다.

벽, 실내, 방이(가) 표시된 사진

자동 생성된 설명

위 회로에서 시그널 비트 S가 0일 때는 B\_i⊕0 = B\_i 이고 C\_0도 0이므로 일반적인 4-bit Adder와 동일할 것이다. 반면, S가 1일 때는 B\_i⊕1 = B\_i’이 되어 1의보수가 되고, C\_0가 1이 되어 최종적으로 2의 보수가 된다. 즉, 전체 비트 계산은 A-B = A + B’ + 1의 연산이 수행된다.

**4. BCD연산**

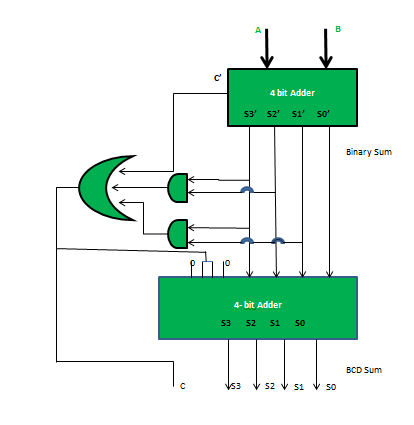
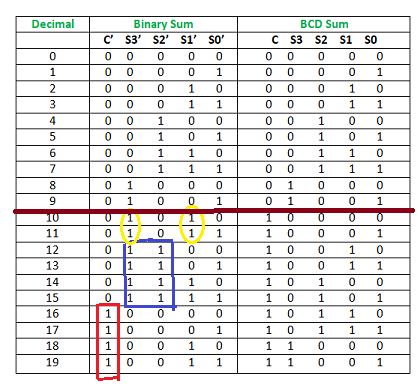
BCD는 Binary coded decimal로 이진 코드 십진법을 의미한다. 단순히 0~9를 4bit를 사용해 표현하는 것이다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 십진법 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| BCD | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 |

이런 표현 방식은 변환이 편하나, 0000과 같이 사용하지 않는 데이터가 많아 데이터 크기가 커진다.

BCD 가산기는 10진수를 BCD로 표현한 것을 입력으로 받고 가산기의 역할을 수행한다. 이때 0~9를 벗어나는 이진수 즉, 1010과 같은 결과가 나오게 되면 추가적인 작업이필요하다. 예를 들어, A = 1001 과 B = 0011을 BCD가산기를 이용해 더한다고 생각해보자. 이진 합계는 9 + 3 = 12 이므로, 일반적인 가산기를 통해 합산한다면 본래 1100(9보다 큰 값)을 출력해야 하지만 BCD가산기의 경우 0001 0010 으로 출력한다. 그렇다면 A = 0011 과 B = 0100 일때는 어떻게 될까? 이는 일반 이진 합과 동일하게 출력하며 결과는 3 + 4 = 7(9보다 작은 값) 이므로 0000 0111 이다. 결론적으로 BCD가산기와 일반 이진합의 차이는 결과가 9보다 커질 때, 2진수로 0110만큼 더해야 함을 알 수 있다.

다음 표를 참고하면 한눈에 볼 수 있다. 이를 통해 , C’(캐리) + S3’S2’ + S3’S1’ = 1임을 알 수 있다. (10 ~ 19가 되는 경우) 다음은, BCD가산기를 병렬가산기를 이용해 나타낸 것이다.



BCD연산의 뺄셈은 덧셈보다 조금 더 까다롭다. EAC(End-around-carry)를 이용한다. EAC는 마지막에 캐리가 발생하면 1, 아니면 0이며 이때 결과 역시 1일 때 +, 0일 때 -가 된다.

우선, 0010 0001 0110을 0101 0100 0001에서 뺄셈을 진행한다고 생각해보자. 이때 빼려고 하는 이진수에 1의 보수를 취해준 뒤 서로 덧셈을 진행한다.

0101 0100 0001

+ 1101 1110 1001 -> 1의 보수

= 1(EAC) 0011 0010 **1010** + 1(EAC) 이때 최종적으로 Carry된 비트는 무시된다.

또한 **1010**은 캐리를 가지지 않기 때문에 1010을 더해준다.(캐리가 발생했을 경우 가운데 4-bit에 1010을 더해준다.) 따라서, 최종적으로

0011 0010 1011

+ 0000 0000 1010

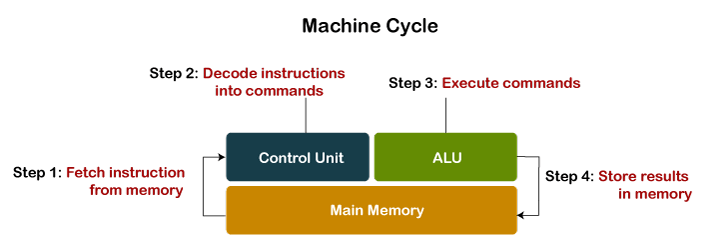
= 0011 0010 **1**0101 이 되며, carry는 무시된다.

이는 541 – 216 = 325로 계산되어 올바른 값을 가지는 것을 확인할 수 있다.

**5. ALU**

CPU에서 연산을 수행하며, Arithmetic and logical unit의 약자로 산술 논리 장치를 의미한다. 보통 ALU의 크기가 CPU의 크기를 결정한다. 크게 내부장치와 레지스터로 이루어져있다. 내부장치에는 이진 산술연산 중 덧셈과 뺄셈을 하기위해서 앞서 설명했던 n-bit 감가산기가 있다. 이외에도 Shift, 비교, 보수 , 논리연산을 수행한다.(보통 정수계산을 수행) 또한, 오버플로우 검출기도 있어 Overflow(레지스터 용량 초과)의 유무를 확인할 수 있다.

보통 레지스터나 메모리로부터 데이터를 입력받고(LOAD), 데이터를 처리한 결과가 레지스터 혹은 메모리에 저장된다. (STORE, AC라는 누산기에 임시적으로 보관하기도 함)



ALU내에서 n개의 선택 변수들이 2^n개의 출력으로 디코드 되어 많은 마이크로 동작을 수행할 수 있다.

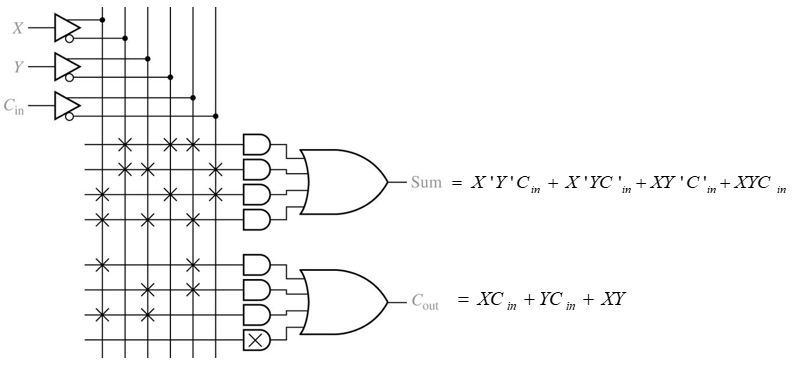
**6. 기타이론**

Programmable Logic Device로 불리는 설계가능논리 소자다. 이는 논리함수의 동작을 직접프로그래밍화 하는 기술 중 하나이다. 내부 구조에 배선이 있어 가능하다. 용량과 구조에 따라서, PLA, PAL, CPLD, FPGA로 나뉜다. 생산시간의 단축과 비용 절감, 부품 절감 등의 이유로 많이 사용되어왔다.

전 가산기를 이런 PAL방식과 PLA방식으로 구현할 수 있다. 둘의 가장 큰 차이점은 AND게이트와 OR게이트 어레이의 프로그래밍 유무이다.



위 그림은 PLA방식이다. 상호 연결된 격자 배선을 연결하여 회로를 구성한다. 이때, AND와 OR Array를 모두 프로그래밍 가능하다. PLA방식으로 구현한 Full Adder 이다.



반면 위 그림은 PLA방식임을 알 수 있다. PLA와 다르게 AND 어레이만 프로그래밍 가능하며, OR어레이가 고정되어 있음을 알 수 있다. 이런점에서 PAL보다는 생산비용이 조금 더 적게 들고 빠르다. 마찬가지로, PLA로 구현한 Full Adder이다.

**7. 참고문헌**

디지털 회로개론 강의자료 chap4 Designing Combinatinal Systems. 김주호 교수

<https://www.javatpoint.com/addition-and-subtraction-using-1s-complement-in-digital-electronics>

<https://www.javatpoint.com/what-is-alu>