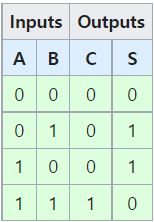
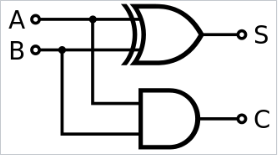
6주차 예비보고서

전공: 생명과학과 학년: 4학년 학번: 20182186 이름: 김승원

**1. 전 가산기 및 반 가산기에 대해 조사**

우선 가산기는 adder라고 쓰고, 덧셈을 하는 logic gate이다. Adder를 사용해서 산술 논리 장치로 사용할 뿐만 아니라 calculate addresses와 table indices 프로세서에서도 사용되고 있다. 여러가지 연산을 하는 Binary-coded decimal(BCD) 와 같은 가산기로 대부분 구성된다. 만약 2’s complement를 이용해서 음수를 표현하면 가산기를 가감산기로 사용하는 경우도 있다.

1) Half adder는 이진수의 단일 자리수를 연산한다. 출력은 S와 C로 출력되는데, S는 합이고, C는 carry이다. C인 carry signal은 덧셈을 진행하면서 다음 자릿수에 오버플로우를 나타낸다.



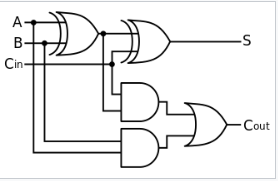
Half adder의 logic gate는 위와 같다. S에 대해서는 XOR gate가 있고, C에 대해서는 AND gate를 사용한다. Half adder는 두 개의 입력 bit를 더하고, half adder의 carry와 합을 생성한다. 이때 A와 B를 합해서 1을 초과하는 수가 나오면 carry를 생성한다.

2) Full adder는 이진수 한 자릿수를 연산한다. 그리고 하위의 carry를 포함하여 Carry out이 출력된다. 즉, 1bit Full adder는 input A, B와 Cin(carry in) 3개의 1bit 숫자를 더한다. 그리고 S와 Cout(carry out)을 출력한다.

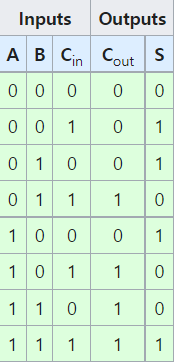
Full adder의 Boolean function은 다음과 같이 표현한다.

si = xi ⊕ yi ⊕ ci

ci+1 = xi yi + ci (xi ⊕ yi)



위는 Full adder의 logic gate이다. Full adder는 half adder 두 개를 연결해서 만들 수 있다. A와 B는 다른 half adder에 연결하고, Cout을 이외의 다른 half adder의 input에 연결한다. 그 후 half adder의 다른 input에 Cin를 연결해서 출력값이 S가 되도록 하고, Cout은 자리 올림 수에 해당하는 output을 OR연산시킨다. S는 마찬가지로 A, B, Ci 3비트로 XOR 연산 시키면 된다.

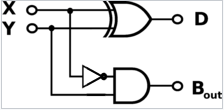


<full adder의 진리표>

**2. 전 감산기 및 반 감산기에 대해 조사**

Subtracter은 adder와 동일한 방식을 사용해서 설계한다. 이진 숫자를 가지고 뺄셈 연산을 하는 회로이다.

**1) half subtractor**



<Half subtractor gate>

Half subtractor는 2bit의 뺄셈을 할 때 사용하는 gate이다. X와 Y를 input으로 사용하고 D와 Bout을 output으로 사용한다. D는 difference이고, B는 borrow이다. 만약에 한 자리보다 더 큰 숫자의 뺄셈을 할 경우 borrow를 앞에서 가져와서 계산해야 한다.

텍스트, 옅은, 램프이(가) 표시된 사진

자동 생성된 설명

**<Half subtractor>**

D = A^B이고, B = ~A\*B이다.

**2) Full subtractor**

텍스트, 옷걸이이(가) 표시된 사진

자동 생성된 설명

**<Full subtractor gate>**

Full subtractor는 현재의 bit를 감산할 때, 하위 비트 감산에서 일어난 borrow를 고려한 다음 연산하고 차와 내림을 출력하는 gate이다.

D = X – Y – Bin – 2Bout

하얀색, 옅은, 지붕이(가) 표시된 사진

자동 생성된 설명

**<Full subtractor 진리표>**

**3. BCD 가산기**

BCD는 binary coded decimal이다. 이진수 4자리를 묶어서 십진수 한 자리로 사용하는 것이다. 이진수 네 자리가 십진수 한 자리에 바로 대응된다. 따라서 변환이 편하기는 하지만 버려지는 데이터가 많아서 같은 데이터를 저장하려고 해도 더 많은 데이터가 소모된다.

BCD 가산기는 BCD code의 가산을 수행하는 회로다. BCD에 맞도록 결과를 꼭 보정해주어야 한다. 가령, 합이 만약 10이상이 될 경우, carry signal을 동작 시켜서 합에 0110을 더해서 BCD에 맞도록 보정해야 한다.

텍스트, 하얀색, 옅은이(가) 표시된 사진

자동 생성된 설명

<8421 code>

**4. 병렬가감산기**

병렬가감산기는 병렬 가산기의 B입력을 부호 S와 XOR해서 전가산기 입력으로 사용하여 덧셈, 뺄셈이 모두 가능하게 하는 회로이다. 디지털 장치에서는 가산기에 gate를 추가하여 감산도 가능하게끔 설계한다. 이러한 신호를 add’/subtract라고 하며, 덧셈은 0, 뺄셈은 1의 값을 가진다.

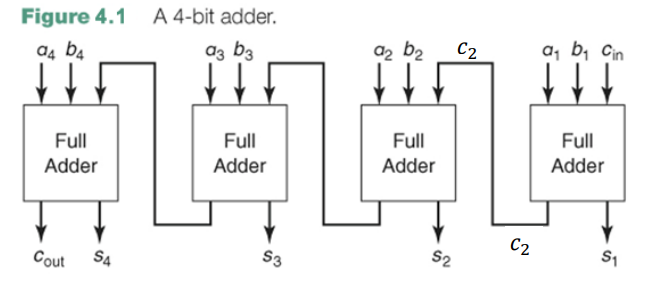
실내, 벽, 방이(가) 표시된 사진

자동 생성된 설명

**<병렬 가감산기>**

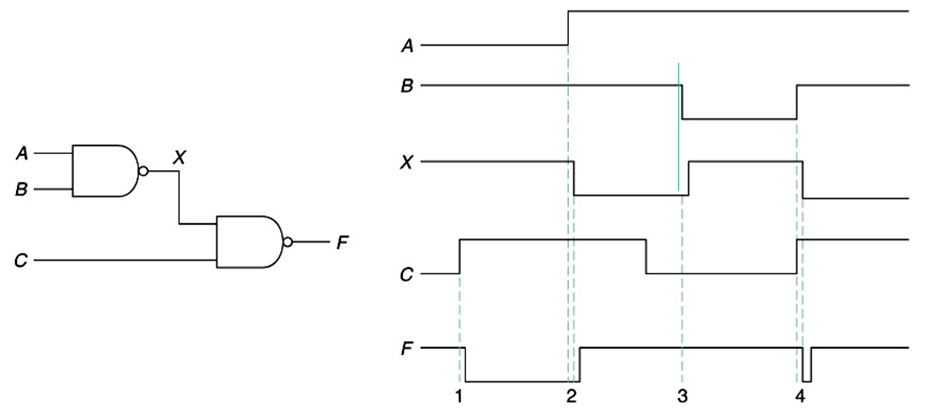
여기서 특징이 XOR gate의 동작에서 S값이 0이면 원래의 B값이 그대로 출력되고, S 값이 1이면 B의 complement가 출력된다. 이때 출력되어지는 값들은 다시 전 가산기의 input이 된다.

**5. Carry Look-ahead adder과 Ripple carry adder 비교**

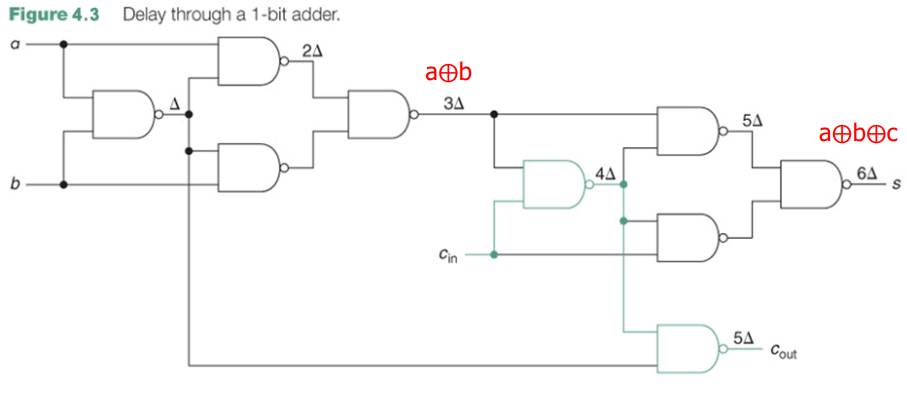


**<Ripple carry adder>**

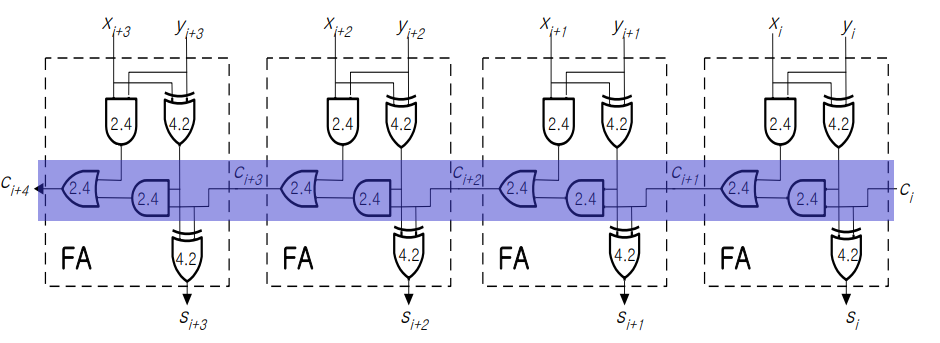
위 그림은 Ripple carry adder로 표현된 4-bit의 adder이다. Adder를 cascade로 연결을 해서 4bit의 ripple carry adder를 만든 것이다. 각각의 Full adder에 input으로 a1, b1, ci이 들어오고, s1이 합으로 나오며, Carry out은 다음 adder로 넘어가게 된다.



이때, a, b, c가 input으로 들어올 때를 확인해본다. A가 0에서 1로 변할 때 x의 값은 0이 즉시 되어야하는데, 실제로는 그렇지 않다. 이때 약간의 딜레이가 발생하게 되는데, 이를 gate delay라고 한다. Combinational system도 완전히 시간에 제약을 받지 않는 것은 아니라는 것을 알 수 있다. c 또한 1을 보면 f가 바로 0으로 down되지 않고 약간의 딜레이가 발생한 것을 확인할 수 있다. 이러한 gate delay에 의해 glitch가 생성될 수 있으므로, 딜레이를 최대한 줄이는 것이 중요한 부분이라고 할 수 있다. 4- bit ripple carry adder 라면 그래도 딜레이가 적다고 할 수 있지만, n-bit ripple carry adder가 있다고 했을 때, n의 값이 많이 커지면 딜레이 또한 많이 증가하게 될 것이다. 즉, 최종 Cout이 나오는 까지의 path를 critical path라고 하며, 이 critical path가 길어질수록 gate delay도 증가하게 된다.

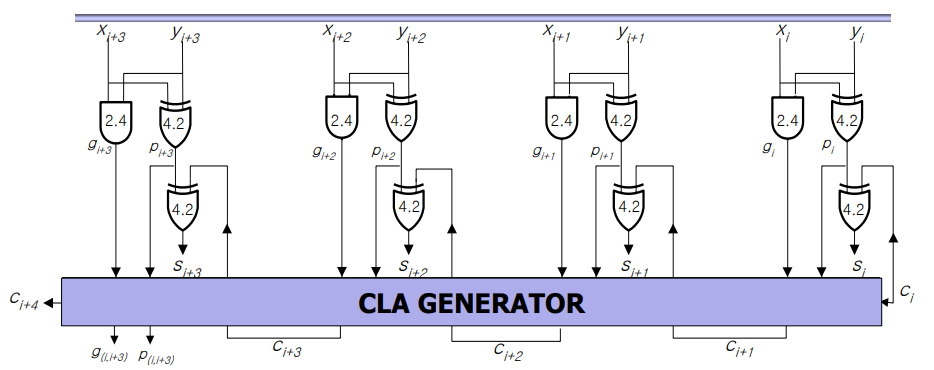


하나의 NAND gate에서 발생하는 딜레이를 Δ로 놓을 때, 1bit full adder 전체에서 보면 결국 s까지 6 Δ만큼 delay를 가지게 된다. 또한 Carry input 부터 carry out까지는 2 Δ만큼의 delay를 가진다.



**<4-bit ripple carry adder>**

결국 4-bit ripple carry adder에서 보라색 영역에 의해 딜레이가 발생하고 있는 것인데, ci부터 ci+4까지 그 자릿수에서의 carry out을 계산해서 넘겨주고 있는 형태이다. 즉 4bit이니까 4번 계산해서 옮겨주게 된다. 저 보라색영역이 system에서 가장 오래 걸리는 부분이다.



carry look-ahead adders는 ci부터 ci+4까지 하나씩 계산하는 것이 아니라, CLA generator로 이동시켜서 한 번에 계산하여 delay를 줄이게 된다. Carry look-ahead adder는 ci부터 ci+4까지 2Δ만큼의 delay를 가지게 되는데, ripple carry adder가 8Δ가 나오는 것과 비교하면 delay를 많이 줄여 줬음을 알 수 있다.

• carry-generate function gi= x~~i~~yi

• carry-propagate function pi= xi⊕yi

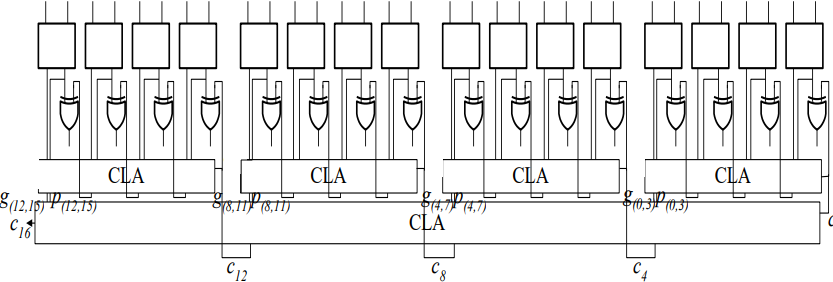
ci+1= gi+pici

ci+2=gi+1+pi+1gi+pi+1pici

ci+3=gi+2+pi+2gi+1+pi+2pi+1gi+pi+2pi+1pici

ci+4= gi+3+pi+3gi+2+pi+3pi+2gi+1+pi+3pi+2pi+1gi+pi+3pi+2pi+1pici

**6. 기타이론**



**<2-level CLA generator>**

계속해서 bit가 추가된다면, 16~64-bit carry look-ahead adder도 결국 delay가 커지게 된다. 이때, 16-bit 기준으로 CLA generator아래에 CLA generator를 하나 더 추가해주면 delay를 또 줄일 수 있다. 이때, delay를 계산해보면 G와 p부터 최종 S까지 총 8Δ 만큼 줄인 것을 확인할 수 있다. 기존의 16-bit carry look-ahead adder가 10Δ였으니 총 2Δ를 더 줄인 것이다.