**10­­­주차 예비 보고서**

20150555 남민혁

**1/3. 4-Bit Adder 및 Subtractor 이진 병렬 연산 기능에 대하여 조사하시오.**

4비트 병렬 가감산기는 4비트로 구성된 입력 두개에 대해 덧셈과 뺄셈 연산을 하는 회로로서 4개의 전가감산기를 활용하여 4비트 병렬 덧셈 또는 뺄셈이 수행된다. 즉, 과 로 표현되는 4비트 이진수 두개가 아래와 같이 4개의 전가감산기가 병렬로 연결되어 연산할 수 있다. 각각의 전가산기 (FA)는 하나의 비트씩 연산을 수행하고 합(과 자리올림수 (를 출력하는 방식으로 연산 결과를 출력한다. Schematic으로 표현하면 다음과 같다.

A picture containing text

Description automatically generated

모든 연산을 병렬적으로 수행하므로, 을 알기 위해서는 반드시 과 의 값이 확정되어 있어야만 연산이 가능하다. 따라서, 상당히 큰 비트 연산에서 특정 비트의 만 필요하더라도 이전 비트에 대하여 모든 연산이 사전에 수행되어야 하므로 지연시간이 길다는 단점이 있다. 이와 같은 단점을 해소하기 위하여 2번 문항에서 설명한 look ahead carry를 활용한 가감산기도 존재한다. 이는 낮은 비트의 연산이 완료되지 않더라도, 상위 비트에서 자리 올림수가 존재하는지 여부를 판단할 수 있다.

**2. Look ahead carry 대하여 조사하시오.**

Look ahead carry (예측 자리올림수)는 1번에서 살펴본 Ripple Carry Adder (병렬 가산기)의 단점인 캐리에 대한 긴 지연시간을 해결하기 위해 제안된 것으로서, 이전 비트에 대한 연산 수행 없이도 특정 비트의 자리올림수 을 연산할 수 있다는 점에서 차이가 있다. 이는 두개의 입력 (A, B)에 대하여 앞에서 살펴본 가산기와 다르게 자리올림수 생성 (G)와 자리올림수 전파 (P)를 통하여 각 비트의 합(S)과 자리올림수(C)를 구함으로써 가능하다.

이때 자리올림수 생성 (G)는 이전 비트의 연산결과와 상관없이 자리올림수가 반드시 생기는 경우 즉, A와 B가 모두 1인 결과를 가지는 경우를 표시한다. 한편 자리올림수 전파 (P)는 연산 결과 추가적인 자리올림수가 생기는지 여부를 검사한다. 즉, G와 P는 각각 아래와 같이 연산된다.

이를 통하여 각 비트의 합(S)과 자리올림수(C)는 아래와 같이 구할 수 있다.

이 때 는 이전 비트의 를 고려하므로, 하위 비트의 의 식을 대입하여 가장 마지막 인 이 나올 때까지 식을 전개하면 아래와 같다.

이 때 식을 모두 전개하면 만 있으므로, 연산을 위하여 의 연산을 기다릴 필요가 없이 바로 구해진다. 이러한 개념을 활용하여 구현된 아래 4비트 Carry Look-Ahead Adder (자리올림수 예측 가산기)의 Schematic 을 통해서도 각 비트의 합의 결과가 이전의 결과에 의존하지 않음을 확인할 수 있다.

A picture containing timeline

Description automatically generated

예를 들어 는 으로 표현되므로, 입력 값으로 받은 와 로 구할 수 있다.

그러나 위 식에서 알 수 있듯이 Carry Look-Ahead Adder 구현하기 위해서는 Ripple Carry Adder와 다르게 상위 비트의 결과는 하위 비트 입력 전체를 직접 입력으로 받아 연산되므로 부가적인 회로가 요구된다. 이에 따라 비용의 측면에서 Ripple Carry Adder보다 한계점이 있다.

**4. XOR을 활용한 2’s complement 가감산에 대하여 조사하시오.**

2의 보수 (2’s complement)는 비트 패턴으로 정수를 표현하는 한가지 방법으로서, 양수는 일반적인 이진수의 방법을 따르며, 음수는 다음과 같은 방법으로 표현된다. 어떤 십진수 음수 a는 a의 절댓값을 2진수로 표현한 후, 해당 결과를 모두 반전시킨 후, 1을 더함으로서 2의 보수로서 표현할 수 있다. 예를 들어 -6을 2의 보수 표현한 것은 6의 이진수인 0110(2)을 1001(2)로 반전시킨 후, 1을 더한 1010(2)이다. 2의 보수를 사용할 경우 연산시 -0이 발생하는 문제나, 자리올림 발생을 처리해야하는 문제를 해결할 수 있다는 장점이 있다.

2의 보수를 이용하면 다음 그림과 같이 덧셈과 뺄셈을 할 수 있다.

Text

Description automatically generated Text

Description automatically generated

연산의 종류에 상관이 없이 각 비트에서는 XOR의 입출력과 마찬가지로, 두개의 입력과 carry bit 입력에서 홀수 개의 1이 발생할 때만 1의 값을 가지고, 나머지는 0의 출력을 가진다. 더불어 모든 입력이 1인 경우 AND 게이트를 활용하여 carry bit를 다음 상위 비트 연산에 넘겨주는 방식이다. 즉, 아래 그림과 같이 XOR과 AND 게이트로 전가산기가 병렬적으로 연결된 형태의 회로 2의 보수의 덧셈과 뺄셈을 할 수 있다.

A picture containing shape

Description automatically generated

**5. BCD 연산에 대하여 조사하시오.**

BCD는 Binary Coded Decimal의 약자로서, 일반적인 이진수가 아니라 4개의 비트로 10진수의 하나의 숫자를 이진수로 표현하는 방식으로 예를 들어 13은 BCD코드로 0001 0011이다. 각 자리 수에서는 0부터 9까지만 존재하므로, 각 자리 수는 정확히 4비트로 표현된다. BCD 연산은 기존 이진수로 표현된 데이터를 활용하여 연산을 하는 것을 BCD로 표현된 데이터로 연산하고, BCD로 결과를 출력하는 방식이다.

가장 간단한 한자리 10진수의 연산을 고려하자. 이 경우 각 데이터는 하나의 BCD 코드로 표현되므로, 8개의 입력값 ()과 1개의 자리올림수 ()를 입력으로 받아 4개의 비트로 표현되는 합()과 1개의 자리올림수()를 출력한다. 이 때 연산의 결과가 9이하일 경우에는 BCD코드와 이진수가 동일하므로, 출력 결과는 이전에 살펴본 가산기와 동일하다. 그러나 결과가 10을 넘어가는 경우 이진 합에 0110(6)을 더하여 일의 자리수를 구하고, 자리올림수(로 다음 자릿수 연산에 반영토록 구현해야 한다.

이 때 4개의 비트를 합산하는 방식은 아래에서 살펴볼 Carry Look-Ahead Adder 과 Ripple Carry Adder 모두 가능하나, Ripple Carry Adder (병력 가산기)를 활용하면 Schematic으로 표현하면 다음과 같다.

![Diagram

Description automatically generated]()

**6. ALU의 기능에 대하여 조사하시오.**

ALU는 Arithmetic Logic Unit의 약자로서 CPU 내에서 논리와 산술 연산을 수행하는 주요 연산 장치를 의미한다. 이와 같은 ALU는 CPU 뿐만 아니라 그래픽 카드와 같은 그래픽 처리 장치, 부동소수점 장치, 디지털 신호 처리기, CD 재생기, 텔레비전 등 연산이 요구되는 모든 종류의 회로에 ALU가 포함된다. ALU는 2개의 데이터와 function을 입력으로 받아 2개의 데이터를 가지고 정해진 연산을 수행한다. 이떄 가능한 연산의 종류는 덧셈, 뺄셈과 같은 산술 연산 뿐만 아니라 배타적 논리합, 논리곱, 논리합과 같은 논리연산도 가능하다. 연산의 결과 뿐만 아니라 overflow 등의 연산 상태를 알려주기 위한 condition 출력이 존재한다. 개념적으로 아래 그림과 같은 schematic을 가진다.

Diagram, schematic

Description automatically generated

실제로 ALU는 여러 개의 집적회로 IC로 구성되며, 이에 따라 ALU가 수행할 수 있는 연산의 종류가 결정된다. 또한 위에서 각 4비트 ALU 두개를 carry로서 연결하면, 아래 그림과 같이 8비트 출력 값을 연산할 수 있는 ALU로도 확장할 수 있다.

Diagram, schematic

Description automatically generated

**7. 기타 이론**

컴퓨터는 앞서 살펴본 정수의 덧셈 방식을 기반으로 다음과 같이 수행한다.

즉, multiplier의 LSB(Least Significant Bit)와 muljtiplicand를 곱하고, 이를 결과에 더하고, multiplier는 우측으로, multiplicand는 좌측으로 이동하여 다음 연산을 진행하는 방식이다.

이러한 기본 곱셈 연산 방식을 활용할 경우 multiplier와 multiplicand가 계속 이동하므로, 32비트 연산은 64비트 저장공간과 ALU가 필요하나, multiplier가 점점 작아지는 특성 (우측으로 이동하여, 하위 비트를 고려하지 않아도 됨)을 이용할 경우 저장공간과 ALU 역시 32비트로 처리할 수 있다.이러한 처리 방식을 Optimised Multiplier라고 부른다. 구체적으로 아래 그림과 같다.

Diagram

Description automatically generated

이 밖에도 덧셈의 횟수를 최소화하기 위해 Array 형태로 연산을 구성하는 Array Multiplier도 존재한다. 즉, 기존에 각 multiplier 비트에 대해서 연산을 하고, 결과에 더하던 방식을 병렬적으로 처리하는 방식이다. 이 경우 4비트 곱셈을 최대 6회 연산으로 처리할 수 있다. 이를 schematic으로 표현하면 다음과 같다.

Diagram

Description automatically generated

**8. 참고문헌**

백지오, “컴퓨터의 정수연산”, https://skyil.tistory.com/141?category=422812

위키피디아, “Adder”, https://en.wikipedia.org/wiki/Adder\_(electronics).

위키피디아, “Arithmetic logic unit”, https://en.wikipedia.org/wiki/Arithmetic\_logic\_unit.

위키피디아, “Binary-coded decimal”, https://en.wikipedia.org/wiki/Binary-coded\_decimal.

위키피디아, “Carry-lookahead adder”, https://en.wikipedia.org/wiki/Carry-lookahead\_adder.

위키피디아, “Two's complement”, https://en.wikipedia.org/wiki/Two%27s\_complement.

장영조, “디지털공학 및 실습 8장 강의자료: 연산 논리 회로”, https://cms3.koreatech.ac.kr/sites/yjjang/down/dig\_new/chap8.pdf.