10주차 예비보고서

전공: 컴퓨터공학과 학년: 4학년 학번: 20191583 이름: 김태곤

**1. 4-Bit Adder 및 Substractor 이진 병렬 연산 기능에 대하여 조사하시오.**

4-Bit Adder와 Substractor는 디지털 시스템에서 기본적인 산술 연산을 수행하는 데 사용되는 회로이다. 이들은 이진수 체계에서 숫자들을 더하거나 빼는 데 필요한 논리 회로를 포함한다.

**1) 4-Bit Binary Parallel Adder**

4 bit Binary Parallel Adder는 네 개의 비트를 가진 두 이진수를 동시에 더할 수 있는 디지털 회로이다. 이 회로는 복수의 1-bit adder를 연결하여 구성되며, 각각의 1-bit adder는 두 비트의 합과 캐리를 출력한다. 이러한 병렬 처리 방식은 빠른 연산 속도를 가능하게 하며, 각 비트의 계산이 독립적으로 이루어짐에도 불구하고 전체적으로는 연속적인 결과를 생성한다.

4 bit Binary Parallel Adder는 네 개의 비트를 가진 두 이진수 A와 B를 더하는 과정을 통해 작동한다. 각 수는 네 개의 비트로 구성되며, 이를 각각 A3, A2, A1, A0 및 B3, B2, B1, B0으로 표현한다. 이 덧셈 과정은 네 개의 1-bit full adders를 사용하여 수행된다. 각 1-bit full adder는 독립적으로 비트 쌍을 처리하며, 예를 들어, 첫 번째 adder는 A0과 B0을 처리한다. 덧셈 작업을 시작하기 위해 가장 낮은 자리의 adder (A0와 B0을 처리하는)는 초기 캐리 입력값으로 0을 받는다. 모든 1-bit adders는 동시에 동작을 시작하여, 각 비트의 합과 해당 비트의 연산에서 발생하는 캐리를 즉각적으로 계산한다. 이렇게 연산된 캐리는 다음 자리수의 계산에 전달되어 연쇄적으로 다음 비트의 연산에 영향을 미친다. 최종적으로, 각 1-bit full adder에서 계산된 합은 최종 결과 이진수의 해당 자리수를 형성하고, 마지막 adder에서 발생한 캐리 아웃은 최종 캐리로 처리된다. 이렇게 병렬 방식으로 구성된 adder들은 각각의 연산을 동시에 수행함으로써 전체적인 덧셈 과정을 매우 빠르고 효율적으로 완료할 수 있다.

도표, 라인, 평면도이(가) 표시된 사진

자동 생성된 설명

**2) 4-Bit Binary Parallel Subtractor**

4-bit Binary Parallel Subtractor는 네 개의 비트를 가진 두 이진수 사이의 차를 계산하는 디지털 회로이다. 이 회로는 여러 1-bit subtractor를 병렬로 연결하여 구성되며, 각 1-bit subtractor는 두 비트의 차와 해당 비트의 연산에서 발생하는 borrow를 계산한다.

4-bit Binary Parallel Subtractor는 네 개의 비트로 구성된 두 수를 입력으로 받는다. 이 비트들은 A3, A2, A1, A0와 B3, B2, B1, B0로 각각 표현된다. 이 뺄셈 과정은 각 비트 쌍을 독립적으로 처리하는 네 개의 1-bit full subtractor를 사용하여 수행된다. 뺄셈 연산을 시작하기 위해, 가장 낮은 자리의 subtractor (A0과 B0을 처리하는 subtractor)에는 borrow in으로 0이 설정된다. 이 초기 값은 뺄셈 과정에서 연쇄적으로 발생할 수 있는 borrow를 계산하는 데 기초가 된다. 각 1-bit subtractor는 입력된 두 비트와, 이전 자리에서 전달된 borrow를 사용하여 해당 위치의 차와 새로운 borrow를 계산한다. 이 연산은 동시에 실행되어 각 비트 위치에서의 뺄셈이 동시에 처리된다. 결과적으로, 각 1-bit full subtractor에서 계산된 차는 최종 결과 이진수의 해당 비트를 형성하고, 마지막 subtractor에서 발생한 borrow는 최종 borrow로 기록된다. 이 회로의 병렬 구성 덕분에 모든 비트의 뺄셈이 동시에 수행되어 빠르고 효율적인 연산이 가능하다. 이와 같은 방식으로 4-bit Binary Parallel Subtractor는 두 이진수 간의 뺄셈을 정확하게 계산하며, 디지털 시스템 내에서 중요한 역할을 수행한다.

도표, 라인, 평면도이(가) 표시된 사진

자동 생성된 설명

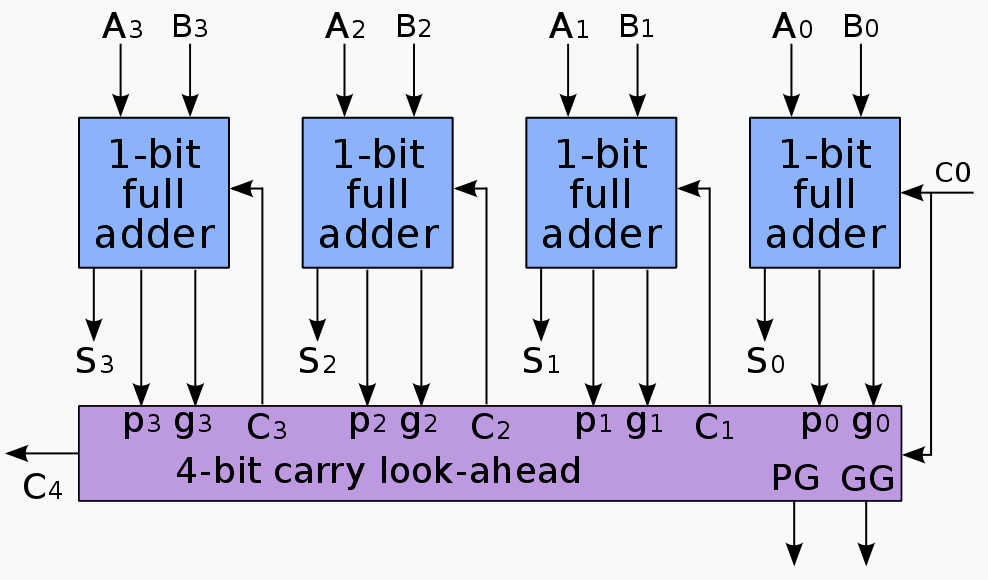
**2. Look ahead carry 대하여 조사하시오.**

Look-ahead carry generator는 덧셈 연산을 수행할 때 발생할 수 있는 캐리를 빠르게 계산하는 회로로, 덧셈 과정의 속도를 크게 향상시킨다. 기본적인 4-bit adder의 경우, 각 1-bit full adder의 캐리 출력은 다음 adder의 캐리 입력으로 전달되는데, 이로 인해 각 adder는 이전 adder의 연산이 완료될 때까지 기다려야 하며, 이는 전체 연산의 지연(delay)을 발생시킨다. Look-ahead carry logic은 이런 지연을 최소화하여 더 빠른 덧셈을 가능하게 한다.

Look-ahead carry generator는 덧셈 연산의 각 비트에서 발생할 수 있는 캐리를 사전에 예측한다. 이 회로는 각 비트의 캐리가 어떻게 발생하는지 논리적으로 분석하여, 각 비트의 연산을 기다리지 않고 캐리를 미리 계산한다. 이를 위해 두 가지 주요 신호, 즉 generate(G)와 propagate(P) 신호를 사용한다. Generate는 해당 비트 위치에서 캐리가 발생하면 G는 참이 된다. 이는 해당 비트의 두 입력이 모두 1일 경우이다. Propagate는 해당 비트 위치의 입력이 캐리를 다음 비트로 전파할 수 있으면 P는 참이 된다. 이는 두 입력 중 적어도 하나가 1일 경우이다. 이러한 G와 P 신호를 사용하여, 덧셈의 각 단계에서 캐리가 어떻게 전파될지를 미리 계산하고, 전체 덧셈 과정을 가속화한다.

Look-ahead carry의 기본적 논리는 이다. 4-bit Adder에서, look-ahead carry generator는 각 비트의 G와 P 값을 계산하고, 이를 사용하여 모든 비트의 캐리를 빠르게 계산할 수 있도록 설계되어 있다.

Look-ahead carry generator의 주요 이점은 연산 속도의 향상이다. 기존의 연쇄적 캐리 전달 방식에 비해 훨씬 빠르게 캐리를 결정하고, 따라서 전체 덧셈 연산을 가속화할 수 있다. 이는 특히 고성능 컴퓨팅이나 대규모 데이터 처리를 요구하는 응용에서 매우 중요하다.



**3. XOR을 활용한 2‘s complement 가감산에 대하여 조사하시오.**

2's complement 가감산에 XOR 게이트를 활용하는 방법은 이진 뺄셈을 덧셈 연산으로 변환하여 수행하는 효율적인 방식이다. 2's complement 시스템은 이진수에서 음수를 표현하는 표준 방법으로, 어떤 수의 2's complement를 얻기 위해선 모든 비트를 반전시킨 후 1을 더하면 된다. 이 과정에서 XOR 게이트는 비트 반전 과정에서 핵심적인 역할을 한다.

구체적으로, 수 B에서 A를 뺄 경우, B의 2's complement를 A에 더함으로써 뺄셈을 덧셈으로 변환할 수 있다. 이는 B의 모든 비트를 반전시킨 후 1을 추가함으로써 B의 음수를 형성하는 과정을 포함한다. XOR 게이트는 이 과정에서 핵심적인 역할을 한다. XOR 게이트는 두 입력 비트가 서로 다를 때만 1을 출력한다. 이 특성을 이용하여 B의 각 비트를 반전할 수 있다. 예를 들어, B의 각 비트에 1을 XOR 연산하면, 0은 1로, 1은 0으로 바뀐다. 이렇게 반전된 B에 1을 추가하면 B의 2's complement가 완성된다. 이어서, 이 결과를 A와 더한다. 이 덧셈은 XOR, AND, OR 게이트를 사용하여 수행될 수 있으며, 각 단계에서 발생하는 캐리를 다음 비트의 연산으로 전달하면서 계속 진행된다. 이 방식을 통해, 복잡한 뺄셈 연산을 단순하고 빠른 덧셈 연산으로 대체할 수 있으며, XOR 게이트를 이용한 비트 반전은 이 과정을 더욱 효율적으로 만든다. 따라서, 2's complement와 XOR 게이트를 사용하는 이 방법은 디지털 회로 설계에서 뺄셈을 수행하는 데 있어 매우 중요하며 널리 사용된다

도표, 라인, 평면도, 기술 도면이(가) 표시된 사진

자동 생성된 설명

**4. BCD 연산에 대하여 조사하시오.**

BCD(Binary Coded Decimal) 연산은 컴퓨터 시스템 내에서 십진수(Decimal)를 이진 코드(Binary)로 표현하고 처리하는 방법 중 하나이다. BCD는 각 십진수 자릿수를 4비트 이진수로 표현하여, 십진수의 숫자 0에서 9까지를 0000부터 1001까지의 이진 코드로 변환한다.

BCD 덧셈의 경우 각 자리의 십진수를 별도로 더한 다음, 결과가 9를 초과하는 경우 6을 더해서 조정한다. 이 과정은 캐리를 발생시킬 수 있으며, 다음 높은 자리로 전달된다. BCD 뺄셈은 덧셈과 유사한 방식으로 수행되지만, 보정을 위해 10의 보수를 사용한다. 결과가 음수가 될 경우, 높은 자리에서 borrow가 발생할 수 있다. BCD에서의 곱셈과 나눗셈은 기본적인 이진 곱셈 및 나눗셈을 사용하되, 각 단계 후에 결과를 BCD 형식으로 조정해야 할 수 있다. 이는 과정이 복잡하고, 추가적인 조정 단계가 필요하다.

BCD는 계산의 정확성을 유지하면서 직접적으로 십진수를 다룰 수 있어, 금융 계산에 적합하다. 또한, 십진수와 BCD 간의 변환은 단순하며, 사람이 이해하기 쉽다. 하지만 BCD는 이진 표현보다 더 많은 메모리 공간을 사용한다. 예를 들어, 최대 9의 값을 나타내는 데 4비트가 필요한 반면, 이진 표현은 이보다 적은 비트를 사용할 수 있다. 그리고 BCD 연산은 보통 이진 연산보다 느리다. 특히 복잡한 수학적 연산을 수행할 때 더 잘 드러난다.

**5. ALU의 기능에 대하여 조사하시오.**

ALU(Arithmetic Logic Unit, 산술 논리 연산 장치)는 컴퓨터의 CPU 내에 있는 핵심 구성 요소로, 다양한 산술 연산(덧셈, 뺄셈, 곱셈, 나눗셈 등)과 논리 연산(AND, OR, XOR, NOT 등)을 수행한다. ALU는 CPU가 프로그램의 명령을 실행할 때 필요한 데이터 처리 작업의 대부분을 담당한다.

ALU는 기본적인 산술 연산을 수행한다. 이에는 덧셈, 뺄셈, 곱셈, 나눗셈 등이 포함되며, 덧셈은 ALU의 가장 기본적인 연산 중 하나이다. 곱셈과 나눗셈은 더 복잡한 연산으로, 때로는 여러 단계의 덧셈, 뺄셈 및 시프트 연산을 포함할 수 있다.

ALU는 논리 연산을 통해 데이터의 비트 단위 처리를 수행합니다. 이러한 연산에는 AND, OR, XOR, NOT이 포함된다. 논리 연산은 비트 마스킹, 비트 설정 또는 클리어, 비트 테스트 등의 작업에서 중요하게 사용된다.

ALU는 데이터를 왼쪽이나 오른쪽으로 시프트하는 연산을 수행할 수 있다. 이러한 시프트 연산은 데이터의 비트 패턴을 조정하여, 곱셈이나 나눗셈을 효과적으로 수행하는 데 도움을 준다.

ALU는 두 데이터 값의 비교를 통해 크기 비교, 등등 비교 같은 조건적 연산을 수행할 수 있다. 이는 분기 명령이나 반복 구조의 조건 검사에 필수적이다. 비교 결과는 종종 프로세서의 상태 플래그에 저장되며, 이는 후속 조건 명령의 실행 흐름을 결정하는 데 사용된다.

ALU는 컴퓨터 아키텍처의 필수적인 부분으로, CPU의 성능과 효율성에 큰 영향을 미친다. 프로그램의 명령을 신속하고 정확하게 수행하기 위해 ALU는 최적화된 연산 능력을 제공해야 하며, 현대의 ALU는 더 빠른 속도와 복잡한 기능을 지원하기 위해 지속적으로 발전하고 있다. ALU의 설계와 성능은 CPU의 전반적인 처리 능력과 직접적으로 관련이 있으며, 컴퓨터의 일반적인 작업 처리 속도에 결정적인 역할을 한다.

텍스트, 도표, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

**6. 기타 이론.**

ALU는 컴퓨터의 중앙 처리 장치(CPU)에서 매우 중요한 역할을 한다. 데이터 처리와 복잡한 연산을 수행하는 역할로 ALU 외에도 FPU(Floating Point Unit)가 있다. ALU와 FPU는 CPU 내에서 다소 다른 역할을 수행하지만, 이 둘은 현대 컴퓨팅 작업에서 서로 보완적인 기능을 제공한다. ALU가 주로 정수 및 논리 연산을 담당하는 반면, FPU는 부동소수점 연산, 즉 소수점이 포함된 수의 연산을 전문적으로 처리한다. 이는 과학적 계산, 엔지니어링 모델링, 금융 분석, 그리고 복잡한 그래픽 처리와 같이 높은 수치 정밀도를 요구하는 작업에 특히 중요하다.

FPU는 고정밀도를 필요로 하는 복잡한 수학 연산을 효과적으로 수행한다. 이는 일반적인 ALU가 처리하기 어려운 고도의 연산 정밀도를 요구하는 작업이다. 또한, 덧셈, 뺄셈, 곱셈, 나눗셈과 같은 기본적인 산술 연산뿐만 아니라 제곱근 계산, 지수 연산, 로그 연산 등의 보다 복잡한 수학적 함수를 처리할 수 있다. 특히 복잡한 부동소수점 연산을 빠르게 처리함으로써 프로그램의 실행 속도를 크게 향상시킨다. 이는 연산 중 자주 발생할 수 있는 오버플로우나 언더플로우를 방지하고, 정확도를 유지하는 데도 기여한다. 다음사진은 FPU구조사진이다. FPU는 ALU옆에 부착되어있어 ALU의 지원을 극복한 계산을 해준다.

텍스트, 도표, 평면도, 기술 도면이(가) 표시된 사진

자동 생성된 설명