10주차 결과보고서

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

**1. 4bit Binary Parallel Adder 의 결과 및 Simulation 과정에 대해서 설명하시오. (verilog source, 출력 예시, 과정 상세히 적을것)**

**1) Design Source file & Testbench file**

**텍스트, 스크린샷, 소프트웨어, 웹 페이지이(가) 표시된 사진

자동 생성된 설명**

**텍스트, 스크린샷, 소프트웨어, 번호이(가) 표시된 사진

자동 생성된 설명**

**2) Schematic**

**라인, 도표, 그래프, 텍스트이(가) 표시된 사진

자동 생성된 설명**

**3) Simulation**

**스크린샷, 소프트웨어, 멀티미디어 소프트웨어, 다채로움이(가) 표시된 사진

자동 생성된 설명**

**스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**4) 결과 및 과정**

4 bit binary parallel Adder는 두 개의 4비트 이진 입력 A와 B, 그리고 하나의 캐리 입력 Cin을 받아서 덧셈 결과를 4비트의 출력S와 캐리 출력 Cout으로 반환한다. 이 모듈은 각 비트 단위의 덧셈과 캐리 전파를 처리하여 최종 결과를 이진 형식으로 제공한다.

비트별 캐리 계산을 위한 C[2:0]은 각 비트의 AND 연산 결과(A[i] & B[i])와 이전 캐리와 해당 비트의 XOR 연산 결과(C[i-1] & (A[i] ^ B[i]))의 OR 연산을 통해 계산된다. 이 과정은 각 비트별로 덧셈이 캐리를 발생시킬 수 있는지를 판단하고, 이를 다음 비트의 계산에 전달하기 위함이다.

각 비트의 합을 저장하는 S[3:0]은 해당 비트의 입력값(A[i], B[i])과 이전 비트에서 발생한 캐리(C[i-1])의 XOR 연산으로 계산된다. XOR 연산은 두 입력 간의 덧셈을 비트 단위로 수행하며, 초기 캐리(Cin)는 첫 번째 비트의 합에만 영향을 미친다.

최종 캐리를 결정하는 Cout은 마지막 비트의 덧셈 결과와 마지막 캐리(C[2])를 기반으로 결정된다. 이는 덧셈 결과가 이진 형식의 범위를 초과할 때 추가 조정이 필요한지를 결정한다.

출력 캐리 Cout은 최종적으로 계산된 캐리를 출력으로 제공한다. 이 캐리는 전체 덧셈 결과가 15를 초과하는 경우 발생한다.

최종적으로 각 S[i]는 계산된 합(S[i]), 그리고 최종 캐리 Cout를 고려하여 최종적인 결과를 도출한다. 초기값으로 A와 B는 0000, Cin은 0으로 설정되어 있으며, always 블록을 통해 A는 매 10ns마다 1씩 증가하고, B는 매 160ns마다 1씩 증가하게 구성되어 있다. 시뮬레이션은 2560ns 후에 종료된다. 이러한 설정은 Adder4 모듈이 다양한 입력 조건에서 정확한 산술 연산을 수행하는지 확인할 수 있다.

Simulation 결과를 살펴보면 input A,B의 합이 15 이하인 경우, 일반 두 수의 덧셈 결과가 나온 것을 확인할 수 있다. 또한 두 input의 합이 16 이상인 경우, Cout을 통해 제대로 된 결과값이 나오는 것을 확인할 수 있다. 예를들어 200ns의 두 수 A,B를 살펴보면 각각 4와 1의 값을 가진다. 이때 두 수의 합 4+1=5로 0의 캐리값과 0101의 값이 나오는 것을 확인할 수 있다. 또 다른 예시를 보면, 310ns에는 A의 값으로 f, B의 값으로 1이 들어와 있다. f+1 =16으로 simulation 결과를 확인해보면 캐리값 1과 sum값 0000이 나오는 것을 확인할 수 있다. 이처럼 simulation 결과를 모두 확인해 보면 4bit binary parallel Adder에 맞는 결과가 나오는 것을 알 수 있다.

따라서 구현한 4bit binary parallel Adder 는 올바르게 구현되었고, 시뮬레이션을 통해 제대로 구현되었는지 확인할 수 있었다.

**2. 4bit Binary Parallel Subtractor 의 결과 및 Simulation 과정에 대해서 설명하시오. (verilog source, 출력 예시, 과정 상세히 적을것)**

**1) Design Source file & Testbench file**

**텍스트, 스크린샷, 소프트웨어, 웹 페이지이(가) 표시된 사진

자동 생성된 설명**

**텍스트, 소프트웨어, 컴퓨터 아이콘, 번호이(가) 표시된 사진

자동 생성된 설명**

**2) Schematic**

**라인, 도표, 텍스트, 그래프이(가) 표시된 사진

자동 생성된 설명**

**3) Simulation**

**스크린샷, 멀티미디어 소프트웨어, 소프트웨어, 그래픽 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**스크린샷, 멀티미디어 소프트웨어, 소프트웨어, 텍스트이(가) 표시된 사진

자동 생성된 설명**

**4) 결과 및 과정**

4bit binary parallel subtractor인 Sub4 모듈은 두 개의 4비트 이진 입력 A와 B, 그리고 하나의 borrow 입력 BoIn을 받아서 뺄셈 결과를 4비트의 출력 D와 borrow 출력 BoOut으로 반환한다. 이 모듈은 각 비트 단위의 뺄셈과 borrow 전파를 처리하여 최종 결과를 이진 형식으로 제공한다.

비트별 borrow 계산을 위한 Bo[2:0]은 각 비트의 A[i]가 B[i]보다 작을 때 발생하는 borrow (~A[i] & B[i])와 이전 borrow 와 해당 비트의 결과가 동일할 경우 발생하는 borrow ((~(A[i] ^ B[i])) & Bo[i-1])의 OR 연산을 통해 계산된다. 이 과정은 각 비트별로 뺄셈이 borrow 를 발생시킬 수 있는지를 판단하고, 이를 다음 비트의 계산에 전달하기 위함이다.

각 비트의 차를 저장하는 D[3:0]은 해당 비트의 입력값(A[i], B[i])과 이전 비트에서 발생한 borrow (Bo[i-1])의 XOR 연산으로 계산된다. XOR 연산은 두 입력 간의 뺄셈을 비트 단위로 수행하며, 초기 borrow (BoIn)는 첫 번째 비트의 뺄셈에만 영향을 미친다.

최종 borrow 를 결정하는 BoOut은 마지막 비트의 뺄셈 결과와 마지막 borrow (Bo[2])를 기반으로 결정된다. 이는 뺄셈 결과가 이진 형식의 범위를 초과할 때 추가 조정이 필요한지를 결정한다.

출력 borrow BoOut은 최종적으로 계산된 borrow 를 출력으로 제공한다. 이 borrow 는 전체 뺄셈 결과가 음수가 되는 경우 발생한다.

최종적으로 각 D[i]는 계산된 차(D[i]), 그리고 최종 borrow BoOut를 고려하여 최종적인 결과를 도출한다. 초기값으로 A와 B는 0000, BoIn은 0으로 설정되어 있으며, always 블록을 통해 A는 매 10ns마다 1씩 증가하고, B는 매 160ns마다 1씩 증가하게 구성되어 있다. 시뮬레이션은 2560ns 후에 종료된다. 이러한 설정은 Sub4 모듈이 다양한 입력 조건에서 정확한 산술 연산을 수행하는지 확인할 수 있다.

Simulation 결과를 살펴보면 input A,B의 차가 0 이상인 경우, 일반 두 수의 뺄셈 결과가 나온 것을 확인할 수 있다. 또한 두 input의 차가 0이하인 경우, BoOut을 통해 제대로 된 결과값이 나오는 것을 확인할 수 있다. 예를들어 200ns의 두 수 A,B를 살펴보면 각각 4와 1의 값을 가진다. 이때 두 수의 합 4-1=3으로 0의 borrow값과 0011의 값이 나오는 것을 확인할 수 있다. 또 다른 예시를 보면, 320ns에는 A의 값으로 0, B의 값으로 2가 들어와 있다. 0-2 =-2로 simulation 결과를 확인해보면 Borrow값 1과 Difference값 1110이 나오는 것을 확인할 수 있다. 이처럼 simulation 결과를 모두 확인해 보면 4bit binary parallel Subtractor에 맞는 결과가 나오는 것을 알 수 있다.

따라서 구현한 4bit binary parallel Subtractor 는 올바르게 구현되었고, 시뮬레이션을 통해 제대로 구현되었는지 확인할 수 있었다.

**3. BCD Adder 의 결과 및 Simulation 과정에 대해서 설명하시오.**

**(verilog source, 출력 예시, 과정 상세히 적을것)**

**1) Design Source file & Testbench file**

**텍스트, 스크린샷, 소프트웨어, 번호이(가) 표시된 사진

자동 생성된 설명**

**텍스트, 스크린샷, 소프트웨어, 컴퓨터 아이콘이(가) 표시된 사진

자동 생성된 설명**

**2) Schematic**

**도표, 라인, 평면도, 텍스트이(가) 표시된 사진

자동 생성된 설명**

**3) Simulation**

**스크린샷, 소프트웨어, 멀티미디어 소프트웨어, 다채로움이(가) 표시된 사진

자동 생성된 설명**

**스크린샷, 텍스트, 멀티미디어 소프트웨어, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**4) 결과 및 과정**

BCDAdder 모듈은 두 개의 4비트 BCD 입력 a와 b, 그리고 하나의 캐리 입력 Cin을 받아서 덧셈 결과를 4비트의 sum 출력과 캐리 출력 cout으로 반환한다. 이 모듈은 각 비트 단위의 덧셈과 캐리 전파를 처리하여 최종 결과를 BCD 형식으로 제공한다.

비트별 캐리 계산을 위한 c[3:0]은 각 비트의 AND 연산 결과(a[i] & b[i])와 이전 캐리와 해당 비트의 XOR 연산 결과(c[i-1] & (a[i] ^ b[i]))의 OR 연산을 통해 계산된다. 이 과정은 각 비트별로 덧셈이 캐리를 발생시킬 수 있는지를 판단하고, 이를 다음 비트의 계산에 전달하기 위함이다.

각 비트의 합을 저장하는 s[3:0]은 해당 비트의 입력값(a[i], b[i])과 이전 비트에서 발생한 캐리(c[i-1])의 XOR 연산으로 계산된다. XOR 연산은 두 입력 간의 덧셈을 비트 단위로 수행하며, 초기 캐리(Cin)는 첫 번째 비트의 합에만 영향을 미친다.

최종 캐리를 결정하는 신호 T는 마지막 캐리(c[3]), 마지막 두 비트의 합(s[3] & s[2]), 그리고 마지막과 세 번째 마지막 비트의 합(s[3] & s[1])의 OR 연산으로 구성된다. 이는 덧셈 결과가 BCD 형식의 범위를 초과할 때 추가 조정이 필요한지를 결정한다.

출력 캐리 cout는 최종적으로 계산된 캐리 T를 출력으로 제공한다. 이 캐리는 전체 덧셈 결과가 9를 초과하는 경우 발생하며, 이 경우 BCD 조정이 필요하다.

추가적인 캐리 연산을 위한 배열 CA는 각 비트의 덧셈 결과에 따라 캐리 전파를 조정하는 역할을 한다. CA[0]은 초기 캐리(Cin)와 첫 번째 비트의 결과(s[0])의 XOR 연산 결과에 Cin을 AND 연산한 값으로, 첫 번째 비트에 대한 캐리 입력을 조정한다. CA[1]은 두 번째 비트의 결과(s[1])와 최종 캐리(T)를 AND 연산한 값에, 첫 번째 조정 캐리(CA[0])와 두 번째 비트의 결과(s[1])와 최종 캐리(T)의 XOR 연산을 AND 연산한 결과를 OR 연산하여 계산되며, 두 번째 비트의 캐리 전파를 처리한다. CA[2]는 세 번째 비트의 결과(s[2])와 최종 캐리(T)를 AND 연산한 값에, 두 번째 조정 캐리(CA[1])와 세 번째 비트의 결과(s[2])와 최종 캐리(T)의 XOR 연산을 AND 연산한 결과를 OR 연산하여 계산되며, 세 번째 비트에서의 캐리 전파를 처리한다.

최종적으로 각 sum[i]는 계산된 합(s[i]), 캐리 조정 배열 CA의 값, 그리고 최종 캐리 T를 고려하여 최종적인 결과를 도출한다. 초기값으로 a와 b는 0000, Cin은 0으로 설정되어 있으며, always 블록을 통해 a는 매 10ns마다 1씩 증가하고, b는 매 160ns마다 1씩 증가하게 구성되어 있다. 시뮬레이션은 2560ns 후에 종료된다. 이러한 설정은 BCDAdder 모듈이 다양한 입력 조건에서 정확한 산술 연산을 수행하는지 확인할 수 있다.

Simulation 결과를 살펴보면 input A,B의 합이 10 이하인 경우, 일반 Adder의 결과와 동일한 결과가 나온 것을 확인할 수 있다. 또한 두 input의 합이 10 이상인 경우, cout을 통해 제대로 된 결과값이 나오는 것을 확인할 수 있다. 예를들어 90ns의 두 수 A,B를 살펴보면 각각 9와 0의 값을 가진다. 이때 두 수의 합 9+0=9로 0의 캐리값과 1001의 값이 나오는 것을 확인할 수 있다. 또 다른 예시를 보면, 300ns에는 A의 값으로 e, B의 값으로 1이 들어와 있다. e+1 =15로 simulation 결과를 확인해보면 캐리값 1과 sum값 0101이 나오는 것을 확인할 수 있다. 이처럼 simulation 결과를 모두 확인해 보면 BCD Adder에 맞는 결과가 나오는 것을 알 수 있다.

따라서 구현한 BCD Adder는 올바르게 구현되었고, 시뮬레이션을 통해 제대로 구현되었는지 확인할 수 있었다.

**4. 결과 검토 및 논의 사항.**

4bit binary parallel Adder와 Subtractor 모듈은 각각 두 개의 4비트 이진 입력을 받아 덧셈과 뺄셈을 수행하고, BCD Adder는 두 개의 4비트 BCD 입력을 받아 덧셈을 수행한다. 이 모듈들은 정확한 산술 연산과 Carry 또는 Borrow 처리를 통해 각각의 연산 결과를 이진 형식으로 제공한다.

Adder 모듈에서는 Cin이 초기 캐리 입력으로 사용되고, 각 비트의 덧셈을 수행한 후 결과적인 캐리(Cout)를 출력한다. XOR과 AND 연산을 통해 캐리를 전파하면서, 모든 비트에 대해 덧셈을 수행한다. 시뮬레이션 결과, 입력 A와 B의 합이 15 이하인 경우 정상적인 덧셈 결과를 확인할 수 있으며, 16 이상에서는 Cout를 통해 올바른 결과가 출력된다.

Subtractor 모듈에서는 BoIn을 초기 borrow 입력으로 사용하고, XOR 연산을 통해 각 비트의 차를 계산한다. borrow가 필요한 경우에는 OR 연산을 통해 borrow를 전파한다. 결과적으로 BoOut은 전체 뺄셈 결과가 음수일 때 발생한다. 시뮬레이션 결과, 입력 A와 B의 차가 0 이상인 경우 정확한 뺄셈 결과가 도출되고, 0 이하인 경우 BoOut을 통해 적절한 결과가 제공된다.

BCD Adder는 BCD 형식의 입력값에 대해 덧셈을 수행하고, 결과가 10을 초과하는 경우 적절한 캐리 처리를 통해 BCD 형식을 유지한다. 이 모듈은 십진수 연산을 위해 특별히 설계되었고, 시뮬레이션에서 10 이상의 결과에 대해 캐리가 정확하게 처리되는 것을 확인할 수 있다.

이러한 모듈들의 구현과 검증을 통해, 다양한 이진 및 BCD 산술 연산이 효과적으로 수행될 수 있음을 보여준다. 각 모듈의 구현에서 borrow와 Carry 처리 로직에 대한 이해가 중요하며, 이는 특히 뺄셈과 BCD 덧셈에서 더욱 복잡해질 수 있다. 따라서 이러한 연산들을 효율적으로 수행하기 위해서는 로직의 정확성을 지속적으로 검토하고 필요한 경우 최적화하는 과정이 필요하다.

**5. 추가 이론 조사 및 작성.**

BCD Excess-3 Code Adder는 표준 BCD 코드를 사용하는 대신 각 십진 숫자에 3을 더한 값을 이진 형태로 변환하여 사용하는 Adder이다. 이렇게 변환된 코드를 Excess-3 코드라고 부르며, 예를 들어 십진수 '0'은 Excess-3 코드에서 '0011'으로, 십진수 '9'는 '1100'으로 표현된다. Excess-3 코드의 주요 특징 중 하나는 자체 보정 특성으로, 이는 덧셈 연산 결과가 BCD 허용 범위를 초과할 때 간단한 조정만으로도 올바른 결과를 얻을 수 있게 한다.

Adder 자체가 보다 간단한 회로로 구성될 수 있으며, 이는 Excess-3 코드의 구조적 특성 때문에 오버플로우 처리를 더 쉽게 할 수 있기 때문이다. 또한, Excess-3 코드는 오류 감지에도 유리하여, 코드의 구조 때문에 발생할 수 있는 일부 오류 유형을 더 명확하게 식별할 수 있다. 주로 전자 계산기나 디지털 시스템에서 사용되는 Excess-3 Adder는 이진 코드 대신 사용함으로써 특정 애플리케이션에서 덧셈 연산을 단순화하고 하드웨어 구현을 더 빠르고 효율적으로 만드는 이점을 제공한다.

텍스트, 번호, 스크린샷, 낱말맞추기 퍼즐이(가) 표시된 사진

자동 생성된 설명