10주차 결과보고서

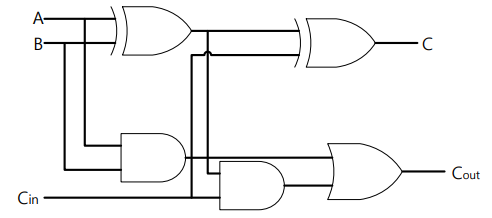
전공: 수학과 학년: 4학년 학번: 20171273 이름: 심현우

1. **4bit Binary Parallel Adder의 결과 및 simulation 과정에 대해서 설명하시오. ( Verilog source, 출력 예시, 과정 상세히 적을 것.)**

테이블이(가) 표시된 사진

자동 생성된 설명

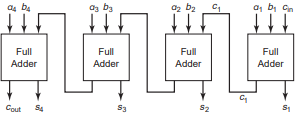
4bit Binary parallel adder는 full adder를 병렬적으로 연결하여 더 많은 자릿수의 binary bit를 계산하기 위한 회로이다. 이번 실습에서는 4자리 binary bit에 대한 계산을 수행하는 회로를 구성했다. 4자리 bit이므로 full adder 4개를 연결한다. 6주차에 구현했던 full adder의 회로와 Verilog code는 다음과 같다.



텍스트이(가) 표시된 사진

자동 생성된 설명

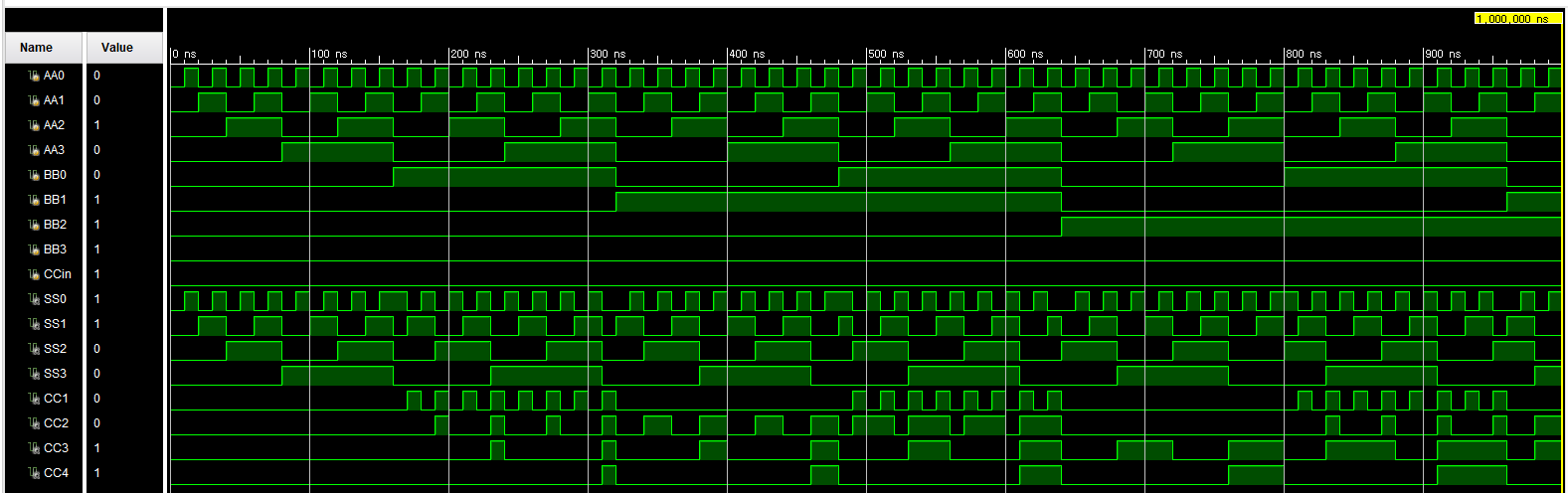
구현했던 full adder를 병렬적으로 연결한 회로와 Verilog code는 다음과 같다.



텍스트이(가) 표시된 사진

자동 생성된 설명

회로를 먼저 살펴보면 제일 하위 자릿수에서 full adder처럼 계산에 필요한 bit와 carry in이 input으로 들어온다. Full adder의 회로에 따라 sum과 carry out을 계산하고 carry out을 상위 자릿수의 carry in으로 사용한다. 이를 4bit만큼 반복한다. Verilog code로 구현 시 위의 회로 구성 그대로 Cin을 가장 하위 자릿수 carry in으로, C1~C3를 다음 자릿수에 넘겨주는 carry out으로, C4를 제일 상위 자릿수의 carry out으로 설정했다. 여기서 C1~C3는 하위 자릿수의 계산 결과로 나온 output임과 동시에 다음 자릿수의 carry in으로 들어가야하는 input이기 때문에 inout으로 설정했다. 코드는 full adder와 동일하며 각 자릿수마다 변수만 다르게 설정했다. 회로 구성은 XOR게이트와 AND, OR게이트를 사용했다. A3,A2,A1,A0와 B3,B2,B1,B0는 덧셈을 진행하는 두 4자리 bit이고 각각 16가지의 경우의 수가 생긴다. 따라서 덧셈의 총 경우의 수는 16X16으로 256가지이다. 이를 모두 truth table로 작성하는 것은 무리가 있으므로 simulation 결과를 보며 실제 덧셈을 진행하면서 확인하는 것으로 한다.



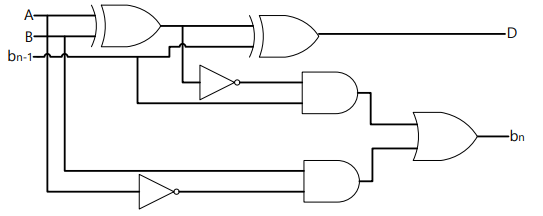
A3,A2,A1,A0와 B3,B2,B1,B0에 input 4자리 bit와 Cin으로 초기 carry bit가 있다. 초기 Cin bit는 발생하지 않은 것으로 0으로 설정했다. S3,S2,S1,S0가 덧셈의 결과이고, C1~C4가 carry가 발생한 자리를 뜻한다. 예시로 몇가지 경우를 보자. 500ps에서 A3,A2,A1,A0는 0010, B3,B2,B1,B0은 0011이다. 실제 덧셈을 진행하면 결과는 0101이고 carry는 A1,B1자리에서 발생했다. simulation결과를 확인해보면 S3,S2,S1,S0는 0101이고 C4,C3,C2,C1은 0010이다. SUM의 결과는 일치하고 C2에서 1인 것은 A1,B1자리에서 carry가 발생했다는 의미이므로 carry 역시 결과가 일치한다. 예시로 600ps하나 더 보자. 600ps에서 A3,A2,A1,A0는 1100이고 B3,B2,B1,B0는 0011이다. 실제 덧셈 계산을 진행하면 SUM 결과는 1111이고, carry는 발생하지 않는다. Simulation을 보면 S3,S2,S1,S0는 1111이고 C4,C3,C2,C1은 0000으로 결과가 일치하는 것을 확인할 수 있었다.

1. **4bit Binary Parallel Subtractor의 결과 및 simulation 과정에 대해서 설명하시오. ( Verilog source, 출력 예시, 과정 상세히 적을 것.)**

텍스트이(가) 표시된 사진

자동 생성된 설명

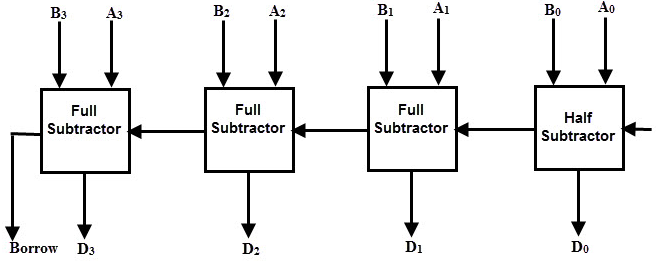
4bit Binary parallel subtractor는 full subtractor를 병렬적으로 연결하여 더 많은 자릿수의 binary bit를 계산하기 위한 회로이다. 이번 실습에서는 4자리 binary bit에 대한 계산을 수행하는 회로를 구성했다. 4자리 bit이므로 full subtractor 4개를 연결한다. 6주차에 구현했던 full subtractor의 회로와 Verilog code는 다음과 같다.



텍스트이(가) 표시된 사진

자동 생성된 설명

구현했던 full subtractor를 병렬적으로 연결한 회로와 Verilog code는 다음과 같다.



텍스트이(가) 표시된 사진

자동 생성된 설명

회로를 먼저 살펴보면 제일 하위 자릿수에서 full subtractor처럼 계산에 필요한 bit와 Burrow in이 input으로 들어온다. Full subtractor의 회로에 따라 difference와 burrow out을 계산하고 burrow out을 상위 자릿수의 burrow in으로 사용한다. 이를 4bit만큼 반복한다. Verilog code로 구현 시 위의 회로 구성 그대로 Brin을 가장 하위 자릿수 burrow in으로, Br1~Br3를 다음 자릿수에 넘겨주는 burrow out으로, Br4를 제일 상위 자릿수의 burrow out으로 설정했다. 여기서 Br1~Br3는 하위 자릿수의 계산 결과로 나온 output임과 동시에 다음 자릿수의 burrow in으로 들어가야하는 input이기 때문에 inout으로 설정했다. 코드는 full subtractor와 동일하며 각 자릿수마다 변수만 다르게 설정했다. 회로 구성은 XOR게이트와 AND, OR게이트를 사용했다. A3,A2,A1,A0와 B3,B2,B1,B0는 뺄셈을 진행하는 두 4자리 bit이고 각각 16가지의 경우의 수가 생긴다. 따라서 덧셈의 총 경우의 수는 16X16으로 256가지이다. 이를 모두 truth table로 작성하는 것은 무리가 있으므로 simulation 결과를 보며 실제 덧셈을 진행하면서 확인하는 것으로 한다.

텍스트, 시계, 시계겸용라디오이(가) 표시된 사진

자동 생성된 설명

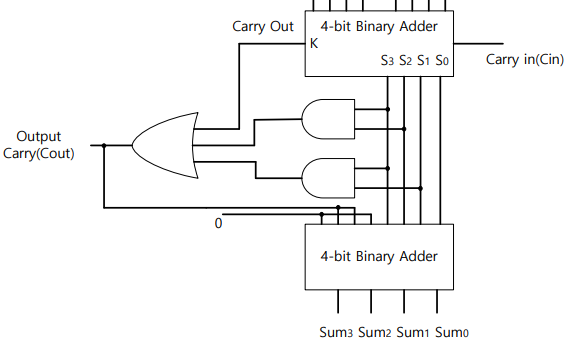
A3,A2,A1,A0와 B3,B2,B1,B0에 input 4자리 bit와 Brin으로 초기 burrow bit가 있다. 초기 Brin bit는 발생하지 않은 것으로 0으로 설정했다. D3,D2,D1,D0가 뻴셈의 결과이고, Br1~Br4가 burrow가 발생한 자리를 뜻한다. 예시로 몇가지 경우를 보자. 600ps에서 A3,A2,A1,A0는 1100, B3,B2,B1,B0은 0011이다. 실제 뺄셈을 진행하면 결과는 1001이고 burrow는 A1,B1자리와 A0,B0 자리에서 발생했다. simulation결과를 확인해보면 D3,D2,D1,D0는 1001이고 Br4,Br3,Br2,Br1은 0011이다. Difference의 결과는 일치하고 Br1과 Br2에서 1인 것은 A1,B1, A0,B0자리에서 burrow가 발생했다는 의미이므로 burrow 역시 결과가 일치한다. 예시로 300ps하나 더 보자. 300ps에서 A3,A2,A1,A0는 1110이고 B3,B2,B1,B0는 0001이다. 실제 뺄셈 계산을 진행하면 Difference 결과는 1101이고, burrow는 A0,B0자리에서 발생한다. Simulation을 보면 D3,D2,D1,D0는 1101이고 Br4,Br3,Br2,Br1은 0001으로 결과가 일치하는 것을 확인할 수 있었다.

1. **BCD Adder의 결과 및 simulation 과정에 대해서 설명하시오. ( Verilog source, 출력 예시, 과정 상세히 적을 것.)**

텍스트이(가) 표시된 사진

자동 생성된 설명

BCD adder는 4-bit binary adder를 응용하여 SUM의 결과를 BCD로 convert하는 기능을 하는 회로이다. 기본적인 회로의 변수는 binary 4-bit 두개이다. 이때, input또한 BCD로 표현하기 위하여 4-bit로 0~9범위의 숫자만 사용한다. 0000을 0으로, 1001을 9로 하여 0~9까지 사용하고 그 외 1010부터 1111까지는의 bit는 사용하지 않는다. 따라서 덧셈을 진행할 시 표현할 수 있는 숫자 범위는 0부터 18이고, 초기 Carry in을 고려하여 19까지 나타낼 수 있다. 결과는 5bit를 사용하여 나타낸다. 제일 상위 자릿수인 C와 S3,S2,S1,S0로 변수를 설정하여 나타냈는데 C는 십진법으로 10을, 나머지 S3~S0는 각각 8,4,2,1을 나타낸다. 예를 들어 18은 11000이고, 19는 11001이다. 이 C와 S3~S0의 회로를 구성하기 위해 4-bit binary adder를 두개 사용한다.



텍스트이(가) 표시된 사진

자동 생성된 설명

회로에서 위 adder는 BCD를 고려하지 않고 4-bit간 덧셈을 계산한다. 이 때, 제일 상위 자릿수에서 발생한 C4, SUM을 Z3~Z0으로 설정했다. BCD의 제일 상위 자릿수의 결과를 내기 위해 위 adder에서 계산한 결과를 이용한다. BCD의 제일 상위 자릿수는 10을 나타내므로 위 adder에서 나온 결과가 10을 넘는 경우를 생각하면 된다. 이는 위의 결과 중 C4의 값이 1이거나, Z3,Z2가 동시에 1일 때, 또는 Z3, Z1이 동시에 1인 경우이다. 따라서 위 3가지 경우를 OR게이트로 묶어 BCD의 제일 상위 자릿수로 설정한 C를 나타낸다. 이후 BCD의 하위 자릿수 4개를 구하기 위해 4-bit adder를 한번 더 사용한다. 밑 adder에서의 input은 위 adder에서 결과로 나온 Z3~Z0과 0000또는 0110이 사용된다. 0000과 0110은 BCD의 가장 상위 자릿수 결과인 C에 따라 결정된다. C가 1인 경우, 0110을 add하고, C가 0인 경우, 0000을 add한다. 우선 C가 0인 경우, 덧셈의 결과가 0~9이므로 기존 결과가 그대로 BCD의 결과로 사용될 수 있다. C가 1인 경우, 10이상의 숫자를 나타내는데 이때 하위 자릿수의 결과가 다르므로 0110을 더한다. 0110을 더하는 이유는 truth table로 확인하면 더 쉽게 알 수 있다. Truth table은 다음과 같다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **C4** | **Z3** | **Z2** | **Z1** | **Z0** | **C** | **S3** | **S2** | **S1** | **S0** | **Decimal** |
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **0** | **0** | **0** | **0** | **1** | **0** | **0** | **0** | **0** | **1** | **1** |
| **0** | **0** | **0** | **1** | **0** | **0** | **0** | **0** | **1** | **0** | **2** |
| **0** | **0** | **0** | **1** | **1** | **0** | **0** | **0** | **1** | **1** | **3** |
| **0** | **0** | **1** | **0** | **0** | **0** | **0** | **1** | **0** | **0** | **4** |
| **0** | **0** | **1** | **0** | **1** | **0** | **0** | **1** | **0** | **1** | **5** |
| **0** | **0** | **1** | **1** | **0** | **0** | **0** | **1** | **1** | **0** | **6** |
| **0** | **0** | **1** | **1** | **1** | **0** | **0** | **1** | **1** | **1** | **7** |
| **0** | **1** | **0** | **0** | **0** | **0** | **1** | **0** | **0** | **0** | **8** |
| **0** | **1** | **0** | **0** | **1** | **0** | **1** | **0** | **0** | **1** | **9** |
| **0** | **1** | **0** | **1** | **0** | **1** | **0** | **0** | **0** | **0** | **10** |
| **0** | **1** | **0** | **1** | **1** | **1** | **0** | **0** | **0** | **1** | **11** |
| **0** | **1** | **1** | **0** | **0** | **1** | **0** | **0** | **1** | **0** | **12** |
| **0** | **1** | **1** | **0** | **1** | **1** | **0** | **0** | **1** | **1** | **13** |
| **0** | **1** | **1** | **1** | **0** | **1** | **0** | **1** | **0** | **0** | **14** |
| **0** | **1** | **1** | **1** | **1** | **1** | **0** | **1** | **0** | **1** | **15** |
| **1** | **0** | **0** | **0** | **0** | **1** | **0** | **1** | **1** | **0** | **16** |
| **1** | **0** | **0** | **0** | **1** | **1** | **0** | **1** | **1** | **1** | **17** |
| **1** | **0** | **0** | **1** | **0** | **1** | **1** | **0** | **0** | **0** | **18** |
| **1** | **0** | **0** | **1** | **1** | **1** | **1** | **0** | **0** | **1** | **19** |

C가 0인 경우, 하위 자릿수 결과가 동일한 것은 위에서 설명했듯 truth table로도 확인할 수 있다. C가 1인 경우를 보면 하위 자릿수의 결과에 Z3,Z2,Z1,Z0에 0110을 add할 시 S3,S2,S1,S0가 되는 것을 확인할 수 있다. Verilog code를 구성할 때 이를 만족시키도록 4-bit adder 두개와 input을 조정했다.

Truth table를 확인하기 위해 simulation을 돌린 결과는 다음과 같다.

텍스트, 시계이(가) 표시된 사진

자동 생성된 설명

예시로 600ps를 보자. A3,A2,A1,A0은 1100이고 B3,B2,B1,B0은 0011이다. 4-bit adder의 결과는 1111이고 Z3,Z2가 동시에 1이므로 C4는 0이지만 C는 1이 된다. 따라서 BCD adder의 최종 결과는 1111과 0110을 더하여 10101이 된 것을 확인할 수 있다. 위의 truth table과도 일치한다.

1. **결과 검토 및 논의사항.**

이번 실습에서 4-bit로 확장한 adder와 subtractor, BCD adder를 구현했다. 각 truth table을 전부 구현하지 않았지만 simulation결과에 대해 직접 계산하며 결과를 검토했다. 일부분이지만 결과를 비교했을 때 구현하고자 하는 adder와 subtractor대로 결과가 잘 나온 것을 확인할 수 있었다.

1. **추가 이론 조사 및 작성.**

Adder와 subtractor 외에도 곱셈기가 있다. 곱셈기는 bit에 대한 곱셈 연산을 진행한다. 이번 실습에서 구현한 것과 같이 4-bit에 대한 곱셈 연산을 하기 위해서는 4-bit adder가 3개 필요하다. 곱셈 연산 자체는 AND 게이트를 이용하여 계산한다. 하지만 최종 결과를 위해서 각 자릿수에 대한 곱셈 연산을 한 후 덧셈 연산도 진행해야 하므로 총 3번의 덧셈 연산을 추가로 하여 결과를 낸다. 따라서 4-bit parallel adder가 3개 필요하다. 회로로 나타내면 다음과 같다.

