6주차 결과보고서

전공: 수학과 학년: 4학년 학번: 20171273 이름: 심현우

1. **실험목적**

회로의 기본 논리게이트를 응용하여 input에 대한 다양한 연산을 진행한다. 이때, 게이트의 연산을 이용하여 input bit들에 대한 가산, 감산 연산을 구현했다. 또한 input bit에 대한 다른 code로 변환기도 구성했다. 가산과 감산을 예를 들어 보자. Bit는 0과 1의 값을 갖는다. 따라서 2진법으로 숫자라고 생각해 덧셈, 뺄셈 연산을 진행한다. 10과 01의 가산 연산을 진행하면 결과는 11이 된다. 감산 연산을 진행하면 01이 된다. Input에 대한 가산과 감산 연산은 자릿수를 나눠 진행하므로 우선 한 자리에 대한 가산, 감산 회로를 구성하고 이를 병렬로 연결하여 더 많은 자릿수에 대해 가산, 감산 연산이 가능한 회로를 구상한다. 이러한 회로를 어떻게 구성할지 고민하고, 실제 verilog를 통해 구현, 결과를 확인하는 실험을 진행한다.

1. **Full Adder 및 Half Adder의 simulation 결과 및 과정에 대해서 설명하시오. (진리표 포함)**
2. half adder

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트, 영수증, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명

우선 half adder부터 보자. Half adder는 한자릿수에 대한 가산 연산을 진행하는 회로이다. 따라서 input은 한 자릿수가 두개 들어와서 덧셈을 진행해야하므로 A, B로 두개가 들어왔다. 한 자릿수에 대한 가산 연산을 진행할 때, 결과값은 한 자릿수가 될 수도 있고, 두 자릿수가 될 수도 있다. 따라서 결과가 두 자릿수임을 감안하여 Carry와 Sum으로 output을 지정한다. 이때 carry는 1+1연산과 같이 결과가 10이 될 경우 상위 자릿수에 결과를 넘겨주는 역할을 한다. Sum은 같은 자릿수의 연산의 결과이다. 코드를 살펴보자. Carry의 경우 두 input모두 1일 경우에만 발생하므로 A&B로 구성했다. Sum의 경우 두 input값이 다른 경우, 즉 0+1이거나 1+0인 경우 발생하므로 XOR을 이용하여 구성했다. Simulation 결과를 보면 A와 B값에 따른 Carry 와 sum을 확인할 수 있다. 진리표로 작성해보자.

|  |  |  |  |
| --- | --- | --- | --- |
| In A | In B | Out C | Out S |
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

Simulation의 진리표와 우리가 구성하고자 했던 가산 연산의 결과가 일치하는 것을 알 수 있다.

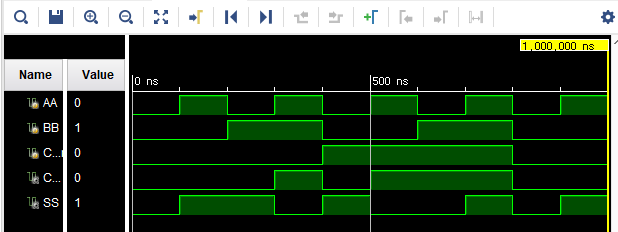
1. Full Adder

텍스트이(가) 표시된 사진

자동 생성된 설명

테이블이(가) 표시된 사진

자동 생성된 설명



full adder를 보자. full adder는 더 많은 자릿수에 대한 가산 연산을 진행하기 위한 회로이다. 따라서 input은 한 자릿수 이상 bit가 들어올 것을 가정하여 이에 대한 덧셈을 진행해야 한다. 따라서 가산 연산을 진행할 때, 고려해야할 input으로 하위 자릿수에서 발생하는 Carry가 있다. Input은 같은 자릿수에서 가산을 진행하는 두가지 bit와 하위 자릿수에서 발생한 carry까지 3개이다. 이에 대한 결과를 도출한다. 결과값은 한 자릿수가 될 수도 있고, 두 자릿수가 될 수도 있다. 따라서 결과가 두 자릿수임을 감안하여 half adder에서 구성했던 것처럼 Carry와 Sum으로 output을 지정한다. 예를 들어, 1과 0의 가산을 진행하는데 하위 자릿수에서 carry가 발생하였다고 가정하자. 그럼 결과로는 carry가 발생하고 sum은 0이 되도록 한다. 이를 구성하기 위한 코드를 살펴보자. 하위 자릿수의 carry를 Cin이라 하자. Cin이 0일 경우 가산 연산은 half adder와 동일하게 진행해주면 된다. Cin이 1인 경우 Sum은 half adder와 반대의 경우 즉 A와 B의 값이 다를 때 0을 출력한다. 같을 때는 1을 출력한다. Carry의 경우 A와 B가 모두 0일 때를 제외하고 1을 출력한다. 따라서 output carry를 cout이라 할 때 cout은 A&B | AXORB&CIN으로 구성한다. Sum의 경우 AXORB와 Cin을 XOR로 구성한다. Simulation 결과를 진리표로 작성해보자.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| in Cin | In A | In B | Out Cout | Out S |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

생각했던 가산 연산과 진리표가 일치하는 것을 확인할 수 있다.

1. **Full Substractor 및 Half Substractor의 simulation 결과 및 과정에 대해서 설명하시오. (진리표 포함)**
2. Half substractor

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트, 영수증, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 모니터, 스크린샷, 실내이(가) 표시된 사진

자동 생성된 설명

우선 half substractor부터 보자. Half substractor는 한자릿수에 대한 감산 연산을 진행하는 회로이다. 따라서 input은 한 자릿수가 두개 들어와서 뺄셈을 진행해야하므로 A, B로 두개가 들어왔다. 한 자릿수에 대한 뺄셈 연산을 진행할 때, 작은 수에서 큰 수를 뺄 수 도 있다. 따라서 이러한 경우에도 감산을 진행할 수 있도록 상위 자릿수에서 값을 빌려온다. 따라서 결과로는 빌려오는 값 burrow와 difference로 두개가 나타난다. 이때 burrow는 0-1연산과 같이 결과가 음수가 될 경우 상위 자릿수에서 1을 빌려와 10-01=01로 만들어주는 역할을 한다. Difference는 같은 자릿수의 연산의 결과이다. 코드를 살펴보자. burrow의 경우 A가 0, B가 1일 경우에만 발생하므로 ~A&B로 구성했다. difference의 경우 두 input값이 다른 경우, 즉 1-0이거나 0-1인 경우 발생하므로 XOR을 이용하여 구성했다. Simulation 결과를 보면 A와 B값에 따른 burrow 와 difference를 확인할 수 있다. 진리표로 작성해보자.

|  |  |  |  |
| --- | --- | --- | --- |
| In A | In B | Out Br | Out D |
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 |

Simulation결과와 생각했던 진리표가 일치하는 것을 알 수 있다.

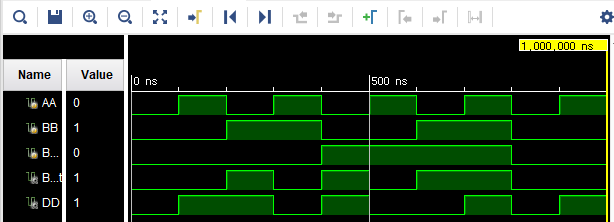
1. Full substractor

텍스트이(가) 표시된 사진

자동 생성된 설명

테이블이(가) 표시된 사진

자동 생성된 설명



full substractor를 보자. full substractor는 더 많은 자릿수에 대한 감산 연산을 진행하기 위한 회로이다. 따라서 input은 한 자릿수 이상 bit가 들어올 것을 가정하여 이에 대한 뺄셈을 진행해야 한다. 따라서 감산 연산을 진행할 때, 고려해야할 input으로 하위 자릿수에서 발생하는 burrow가 있다. Input은 같은 자릿수에서 가산을 진행하는 두가지 bit와 하위 자릿수에서 발생한 burrow까지 3개이다. 이에 대한 결과를 도출한다. 결과값은 또한 burrow가 발생할 수 있다. 따라서 결과가 두 자릿수임을 감안하여 half substractor에서 구성했던 것처럼 burrow와 difference으로 output을 지정한다. 예를 들어, 1과 1의 가산을 진행하는데 하위 자릿수에서 burrow가 발생하였다고 가정하자. 그럼 결과로는 burrow가 발생하고 difference은 0이 되도록 한다. 이를 구성하기 위한 코드를 살펴보자. 하위 자릿수의 burrow를 Brin이라 하자. Brin이 0일 경우 가산 연산은 half substrator와 동일하게 진행해주면 된다. Brin이 1인 경우 difference은 half substractor와 반대의 경우 즉 A와 B의 값이 같을 때 때 1을 출력한다. 다를 때는 0을 출력한다. burrow의 경우도 brin이 0일 때와 반대로 A가 1, B가 0일 때만 0을 출력한다. 따라서 output burrow를 brout이라 할 때 brout은 ~A&B | ~(AXORB)&BRIN으로 구성한다. difference의 경우 AXORB와 brin을 XOR로 구성한다. Simulation 결과를 진리표로 작성해보자.

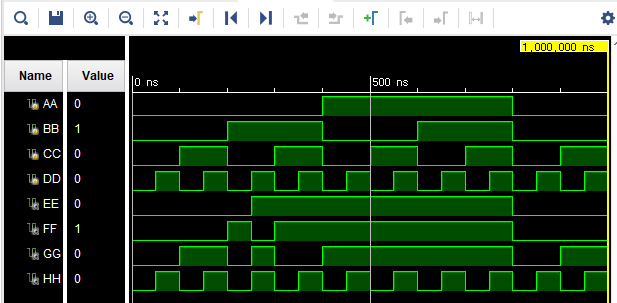
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| In A | In B | In Br | Out Br | Out D |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

Simulation의 진리표 또한 생각했던 감산 연산과 일치했다.

1. **8421(BCD)-2421 Code converter simulation 결과 및 과정에 대해서 설명하시오. (진리표 작성 및 카르노맵 SOP, POS form 포함)**

테이블이(가) 표시된 사진

자동 생성된 설명 텍스트, 영수증, 스크린샷이(가) 표시된 사진

자동 생성된 설명 

BCD code는 bit로 이루어진 숫자를 10진법으로 나타내는 것을 의미한다. 10진법의 0~9까지 숫자를 나타내기 위해 4자리의 bit를 사용하는데 4자리의 bit로는 16가지의 숫자를 나타낼 수 있다. 따라서 10개의 숫자만 사용하는 10진법에서 4자리 bit를 사용하여 10가지 숫자를 나타내는 방법에는 여러가지가 있다. 8421과 2421을 보면 자리마다 1의 숫자가 변환되는 값을 의미한다.

테이블이(가) 표시된 사진

자동 생성된 설명

위의 표처럼 각 10진법 숫자에 대한 4자리 bit의 표현이다.

이를 변환하기 위해서 8421에서 2421로의 카르노맵을 통해 회로를 구성했다. 카르노맵은 다음과 같다.

2421->E

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| AB  CD | 00 | 01 | 11 | 10 |
| 00 | **0** | **0** | **X** | **1** |
| 01 | **0** | **1** | **X** | **1** |
| 11 | **0** | **1** | **X** | **X** |
| 10 | **0** | **1** | **X** | **X** |

2421->F

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| AB  CD | 00 | 01 | 11 | 10 |
| 00 | **0** | **1** | **X** | **1** |
| 01 | **0** | **0** | **X** | **1** |
| 11 | **0** | **1** | **X** | **X** |
| 10 | **0** | **1** | **X** | **X** |

2421->G

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| AB  CD | 00 | 01 | 11 | 10 |
| 00 | **0** | **0** | **X** | **1** |
| 01 | **0** | **1** | **X** | **1** |
| 11 | **1** | **0** | **X** | **X** |
| 10 | **1** | **0** | **X** | **X** |

2421->H

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| AB  CD | 00 | 01 | 11 | 10 |
| 00 | **0** | **0** | **X** | **0** |
| 01 | **1** | **1** | **X** | **1** |
| 11 | **1** | **1** | **X** | **X** |
| 10 | **0** | **0** | **X** | **X** |

이 카르노맵을 토대로 SOP를 작성하면 다음과 같다.

E=A+BD+BC

F=BD`+A+BC

G=B’C+A+BC`D

H=D

이를 Verilog에서 작성하여 simulation을 확인해보았다.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| In A | In B | In C | In D | Out E | Out F | Out G | Out H |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |

simulation결과가 생각했던 진리표와 같은 것을 알 수 있었다.

1. **결과 검토 및 논의사항**

이번 실험에서는 가산, 감산을 진행하는 회로를 구성했다. 한 자릿수에 대한 연산과 자릿수를 확장하여 연산을 진행할 수 있도록 반, 전가감산기를 구성했다. 이를 병렬로 연결하여 두 수에 대한 연산을 진행할 수 있다. 직접 생각한 것과 verilog의 simulation를 비교하였을 때와 동일하게 결과가 나왔다. 또한 BCD code converter또한 카르노맵을 통해 구성했다. 카르노 맵을 활용하여 회로의 최소 표현식을 구하고 boolean식으로 나타낼 수 있다. 이를 이용하여 BCD code를 converter를 직접 구현하고 Verilog simulation을 통해 결과를 확인했다. 이 또한 생각했던 진리표 결과와 simulation이 동일한 것을 확인할 수 있었다.

1. **추가 이론 조사 및 작성**

카르노 맵은 회로의 boolean식을 구하기 위해 사용된다. 우선 진리표를 보고 카르노 맵을 작성한다. 카르노맵을 작성하면 인접한 1들을 묶는다. 이를 implicant라 부른다. 이때 1은 2의 k제곱의 개수가 되도록 묶는다. 이후 더 큰 묶음으로 묶는다. 이를 prime implicant라 부른다. Prime implicant의 내부의 1 중 다른 prime implicant에 포함되지 않는 1을 포함하는 implicant를 essential prime implicant라 부른다. 식을 구성할 때 이 essential prime implicant는 꼭 포함하도록 하며 나머지 1들을 포함할 수 있도록 prime implicant를 조절하여 식을 완성한다. 완성한 식은 SOP형태로 나타난다. F’을 만들기 위해 1과 0의 위치를 바꿔주고 동일한 방법으로 진행한 후 F를 구하면 POS형태의 Boolean 식을 구할 수 있다.