6주차 결과보고서

전공: 아트&테크놀로지학과 학년: 4학년 학번: 20191048 이름: 김도솔

**1.**

Adder(가산기), Subtractor(감산기), Code converter(부호 변환기)의 개념을 이해하고, Verilog를 사용해 직접 다양한 Adder, Subtractor, Code converter를 구현한다. 또한 FPGA를 통해 Verilog로 구현된 회로의 동작을 확인한다.

**2.**

**1) Half Adder**

도표, 스케치, 라인, 그림이(가) 표시된 사진

자동 생성된 설명

(A)

* (A)의 Verilog 코딩

|  |  |
| --- | --- |
| (A)’s Design Source Code | (A)’s Simulation Source Code |
| `timescale 1ns / 1ps  module ha(  input x, y,  output s, c  );  assign s = x^y;  assign c = x&y;  endmodule | `timescale 1ns / 1ps  module ha\_tb;  reg xx, yy;  wire s, c;  ha u\_ha (  .x (xx ),  .y (yy ),  .s (s ),  .c (c )  );  initial begin  xx = 1'b0;  yy = 1'b0;  end  always xx = #100 ~xx;  always yy = #150 ~yy;  initial begin  #1000  $finish;  end  endmodule |

ha 모듈은 2개의 입력(x, y)과 2개의 출력(s, c)을 가지고 있다. 이때 s는 합 출력을 나타내며, c는 자리올림 출력을 나타낸다. s는 x와 y의 XOR(배타적 OR) 연산 결과로, 두 입력이 서로 다를 때 1을 출력하며, 같을 때는 0을 출력한다. c는 x와 y의 AND(논리 곱) 연산 결과로, 두 입력이 모두 1일 때 1을 출력하며, 그 외의 경우에는 0을 출력한다.

* Schematic Diagram

도표, 평면도, 라인, 그래프이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 verilog 코드로 Schematic Diagram을 생성한 결과이다. (A)의 구조와 동일하게 두 input이 AND 게이트로 묶여 c(자리올림)을 출력하고, XOR 게이트로 묶여 s(합)을 출력하는 형태임을 확인할 수 있다.

* Simulation 결과

스크린샷, 사각형이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드를 시뮬레이션 돌린 결과이다. 시간에 따라 입력 x, y의 값을 바꾸면 그에 따라 출력 s, c의 값이 변화하는 것을 확인할 수 있다. 출력 s는 입력 x, y의 값이 다른 경우에만 1을 출력하고 두 입력 값이 같은 경우 0을 출력한다. 출력 c는 입력 x, y가 모두 1인 경우에 1을 출력한다. 이는 1과 1을 더하는 경우에만 합이 2로 자리올림이 발생하기 때문이다. 이를 통해 구현한 반가산기가 이론에서 공부한, 라는 논리 함수와 동일하게 동작한다는 것을 실험적으로 확인할 수 있었다. 시뮬레이션 결과를 진리표로 정리한 결과는 다음과 같다.

* 진리표

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | | **Output** | |
| **X** | **Y** | **S** | **C** |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

**2) Full Adder**

도표, 스케치, 기술 도면, 라인이(가) 표시된 사진

자동 생성된 설명

(A)

* (A)의 Verilog 코딩

|  |  |
| --- | --- |
| (A)’s Design Source Code | (A)’s Simulation Source Code |
| `timescale 1ns / 1ps  module fa(  input x, y, cin,  output s, cout  );  assign s = (x^y)^cin;  assign cout = (cin&(x^y))|(x&y);  endmodule | `timescale 1ns / 1ps  module fa\_tb;  reg xx, yy, cin;  wire s, cout;  fa u\_fa (  .x (xx ),  .y (yy ),  .cin (cin ),  .s (s ),  .cout (cout )  );  initial begin  xx = 1'b0;  yy = 1'b0;  cin = 1'b0;  end  always xx = #100 ~xx;  always yy = #120 ~yy;  always cin = #140 ~cin;  initial begin  #1000  $finish;  end  endmodule |

fa 모듈은 3개의 입력(x, y, cin)과 2개의 출력(s, cout)을 가지고 있다. 이때 cin은 자리올림 입력을, cout은 자리올림 출력을 의미하며, s는 합 출력을 나타낸다. s는 (x^y)^cin의 결과이다. x^y는 x와 y의 XOR(배타적 OR) 연산 결과로서 두 입력 중 하나만 1인 경우 1을 출력한다. 그리고 이 결과와 cin의 XOR 연산을 통해 s가 계산된다. cout은 (cin&(x^y))|(x&y)의 결과이다. x^y는 x와 y의 XOR 연산 결과이며, cin과 함께 AND 연산을 하여 자리올림 입력까지 고려한다. 그리고 x와 y의 AND 연산을 통해 새로운 자리올림이 발생하는 경우를 확인하고 두 결과를 OR 연산하여 cout을 계산한다.

* Schematic Diagram

도표, 라인, 평면도, 폰트이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 verilog 코드로 Schematic Diagram을 생성한 결과이다. (A)의 구조와 동일하게 두 입력 x, y를 XOR 게이트로 연결하고, 그 출력과 cin을 XOR 게이트로 연결해 s(합)를 출력하는 형태이다. 또한 입력 x, y를 XOR 게이트로 연결하고 그 출력과 cin을 AND 게이트로 연결한다. 이 출력 값과 x, y를 AND 게이트로 연결한 출력 값을 다시 한번 OR 게이트로 묶어 cout(자리올림)을 출력한다. 이는 반가산기 두개와 OR 게이트를 연결한 것으로도 볼 수 있다. 먼저 두 개의 입력을 첫 번째 반가산기에 연결하고, 출력된 Sum을 두 번째 반가산기의 입력으로 연결한다. 다음으로 두 번째 반가산기의 다른 입력에 Carry-in을 연결하면 두 번째 반가산기가 최종 Sum을 출력하고, 두 반가산기의 자리올림 수 출력을 OR 연산하면 Carry-out이 된다.

* Simulation 결과

스크린샷, 사각형이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드를 시뮬레이션 돌린 결과이다. 시간에 따라 입력 x, y, cin의 값을 바꾸면 그에 따라 출력 s, cout의 값이 변화하는 것을 확인할 수 있다. 출력 s는 세 입력 x, y, cin 중 참인 입력이 홀수 개일 때 1을 출력한다. 출력 c는 세 입력 x, y, cin 중 참인 입력이 두 개 이상일 때 1을 출력한다. 이는 합이 2 이상인 경우에 자리올림이 발생하기 때문이다. 이를 통해 구현한 전가산기가 이론에서 공부한 , 라는 논리 함수와 동일하게 동작한다는 것을 실험적으로 확인할 수 있었다. 시뮬레이션 결과를 진리표로 정리한 결과는 다음과 같다.

* 진리표

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | | | **Output** | |
| **X** | **Y** |  | **S** |  |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

**3.**

**1) Half Subtracter**

도표, 스케치, 라인, 그림이(가) 표시된 사진

자동 생성된 설명

(A)

* (A)의 Verilog 코딩

|  |  |
| --- | --- |
| (A)’s Design Source Code | (A)’s Simulation Source Code |
| `timescale 1ns / 1ps  module hs(  input x, y,  output D, b  );  assign D = x^y;  assign b = ~x&y;  endmodule | `timescale 1ns / 1ps  module hs\_tb;  reg xx, yy;  wire D, b;  hs u\_hs (  .x (xx ),  .y (yy ),  .D (D ),  .b (b )  );  initial begin  xx = 1'b0;  yy = 1'b0;  end  always xx = #100 ~xx;  always yy = #150 ~yy;  initial begin  #1000  $finish;  end  endmodule |

hs 모듈은 2개의 입력(x, y)과 2개의 출력(D, b)을 가지고 있다. 이때 D는 차이 출력을 나타내며, b는 자리빌림 출력을 나타낸다. D는 x와 y의 XOR(배타적 OR) 연산 결과로, 두 입력이 서로 다를 때 1을 출력하며, 같을 때는 0을 출력한다. 다. b는 x의 보수(~x)와 y의 AND(논리 곱) 연산 결과이다.

* Schematic Diagram

도표, 평면도, 라인, 그래프이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 verilog 코드로 Schematic Diagram을 생성한 결과이다. (A)의 구조와 동일하게 한 input에 NOT을 취한 결과와 다른 input이 AND 게이트로 묶여 b(자리빌림)을 출력하고, 두 input이 XOR 게이트로 묶여 s(합)을 출력하는 형태임을 확인할 수 있다.

* Simulation 결과

스크린샷, 사각형이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드를 시뮬레이션 돌린 결과이다. 시간에 따라 입력 x, y의 값을 바꾸면 그에 따라 출력 D, b의 값이 변화하는 것을 확인할 수 있다. 출력 D는 입력 x, y의 값이 다른 경우에만 1을 출력하고 두 입력 값이 같은 경우 0을 출력한다. 출력 b는 입력 x가 0, y가 1인 경우에 1을 출력한다. 이는 0에서 1을 빼는 경우에만 차이가 -1로 자리빌림이 발생하기 때문이다. 이를 통해 구현한 반감산기가 이론에서 공부한, 라는 논리 함수와 동일하게 동작한다는 것을 실험적으로 확인할 수 있었다. 시뮬레이션 결과를 진리표로 정리한 결과는 다음과 같다.

* 진리표

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | | **Output** | |
| **X** | **Y** | **b** | **D** |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 |

**2) Full Subtractor**

도표, 스케치, 라인, 평면도이(가) 표시된 사진

자동 생성된 설명

(A)

* (A)의 Verilog 코딩

|  |  |
| --- | --- |
| (A)’s Design Source Code | (A)’s Simulation Source Code |
| `timescale 1ns / 1ps  module fs(  input x, y, bin,  output D, bout  );  assign D = (x^y)^bin;  assign bout = ((~(x^y))&bin)|(~x&y);  endmodule | `timescale 1ns / 1ps  module fs\_tb;  reg xx, yy, bin;  wire D, bout;  fs u\_fs (  .x (xx ),  .y (yy ),  .bin (bin ),  .D (D ),  .bout (bout )  );  initial begin  xx = 1'b0;  yy = 1'b0;  bin = 1'b0;  end  always xx = #100 ~xx;  always yy = #120 ~yy;  always bin = #140 ~bin;  initial begin  #1000  $finish;  end  endmodule |

fs 모듈은 3개의 입력(x, y, bin)과 2개의 출력(D, bout)을 가지고 있다. 이때 bin은 자리빌림 입력을, bout은 자리빌림 출력을 의미하며, D는 차이 출력을 나타낸다. D는 (x^y)^bin의 결과이다. x^y는 x와 y의 XOR(배타적 OR) 연산 결과로서 두 입력 중 하나만 1인 경우 1을 출력한다. 그리고 이 결과와 bin의 XOR 연산을 통해 차이(D)가 계산된다. bout은 (~(x^y))&bin와 (~x&y)의 OR(논리 합) 연산 결과이다. x^y는 x와 y의 XOR 연산 결과이며, 여기에 NOT을 취해 bin과 함께 AND 연산을 하여 자리빌림 입력까지 고려한다. 이 결과 값과 x의 보수와 y의 AND 연산 결과를 OR 연산하여 새로운 자리빌림(bout)을 계산한다.

* Schematic Diagram

도표, 텍스트, 라인, 폰트이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 verilog 코드로 Schematic Diagram을 생성한 결과이다. (A)의 구조와 동일하게 두 입력 x, y를 XOR 게이트로 연결하고, 그 출력 값과 bin을 XOR 게이트로 연결해 D(차이)를 출력하는 형태이다. 또한 입력 x, y를 XOR 게이트로 연결하고 그 출력에 NOT을 취한 결과와 bin을 AND 게이트로 연결한다. 이 출력 값과 x에 NOT을 취한 결과와, y를 AND 게이트로 연결한 출력 값을 다시 한번 OR 게이트로 묶어 bout(자리빌림)을 출력한다. 이는 반감산기 두개와 OR 게이트를 연결한 것으로도 볼 수 있다. 우선 두 개의 입력 A, B를 첫 번째 반감산기에 연결하고, 그 Difference를 두 번째 반감산기의 입력으로 연결한다. 다음으로 두 번째 반감산기의 다른 입력에 Borrow-in을 연결하면 두 번째 반감산기가 최종 Difference를 출력하고, 두 반감산기의 Borrow 출력을 OR 연산하면 Borrow-out이 된다.

* Simulation 결과

스크린샷, 사각형이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드를 시뮬레이션 돌린 결과이다. 시간에 따라 입력 x, y, bin의 값을 바꾸면 그에 따라 출력 D, bout의 값이 변화하는 것을 확인할 수 있다. 출력 D는 세 입력 x, y, bin 중 참인 입력이 홀수 개일 때 1을 출력한다. 출력 bout는 세 입력 x, y, cin가 각각 (0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 1, 1)인 경우 1을 출력한다. 이를 통해 구현한 전감산기가 이론에서 공부한 , 라는 논리 함수와 동일하게 동작한다는 것을 실험적으로 확인할 수 있었다. 시뮬레이션 결과를 진리표로 정리한 결과는 다음과 같다.

* 진리표

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | | | **Output** | |
| **X** | **Y** |  |  | **D** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

**4.**

**8421(BCD)-2421 Code converter**

텍스트, 번호, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

(A)

* (A)의 표를 이용하여 진리표 작성

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Decimal** | **8421 Code** | | | | **2421 Code** | | | |
| **A** | **B** | **C** | **D** | **a** | **b** | **c** | **d** |
| **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| **2** | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| **3** | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| **4** | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| **5** | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| **6** | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| **7** | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| **8** | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| **9** | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| **10** | 1 | 0 | 1 | 0 | X | X | X | X |
| **11** | 1 | 0 | 1 | 1 | X | X | X | X |
| **12** | 1 | 1 | 0 | 0 | X | X | X | X |
| **13** | 1 | 1 | 0 | 1 | X | X | X | X |
| **14** | 1 | 1 | 1 | 0 | X | X | X | X |
| **15** | 1 | 1 | 1 | 1 | X | X | X | X |

8421 Code와 2421 Code는 (A)의 표에서 보여지는 것과 같이 차이가 있다. 위의 표는 이를 진리표로 나타낸 것이다. A, B, C, D는 8421 Code의 각 digit을, a, b, c, d는 2421 Code의 각 digit을 나타낸다. 이 진리표를 바탕으로 K-map을 작성해 minimum SOP와 minimum POS를 구할 수 있다.

* K-map(4개)과 Boolean 함수 작성 (SOP form, POS form)

|  |  |
| --- | --- |
| **(a)** | **(b)** |
|  |  |
| SOP form: A+BC+BD  POS form: (A+C+D)(A+B) | SOP form: A+BC+BD’  POS form: (A+B)(A+C+D’) |
| **(c)** | **(d)** |
|  |  |
| SOP form: A+BC’D+B’C  POS form: (A+B+C)(A+C+D)(B’+C’) | SOP form: D  POS form: D |

* Verilog 코딩

|  |  |
| --- | --- |
| (A)’s Design Source Code | (A)’s Simulation Source Code |
| `timescale 1ns / 1ps  module conv(  input ain, bin, cin, din,  output aout, bout, cout, dout  );  assign aout = ain|(bin&cin)|(bin&din);  assign bout = ain|(bin&cin)|(bin&~din);  assign cout = ain|(bin&~cin&din)|(~bin&cin);  assign dout = din;  endmodule | `timescale 1ns / 1ps  module conv\_tb;  reg ain, bin, cin, din;  wire aout, bout, cout, dout;  conv u\_conv (  .ain (ain ),  .bin (bin ),  .cin (cin ),  .din (din ),  .aout (aout ),  .bout (bout ),  .cout (cout ),  .dout (dout )  );  initial begin  ain = 1'b0;  bin = 1'b0;  cin = 1'b0;  din = 1'b0;  end  always ain = #20 ~ain;  always bin = #30 ~bin;  always cin = #50 ~cin;  always din = #70 ~din;  initial begin  #1000  $finish;  end  endmodule |

위에서 K-map을 통해 구한 minimum SOP를 바탕으로 8421(BCD)-2421 Code converter의 코드를 작성할 수 있다. conv 모듈은 4개의 입력(ain, bin, cin, din)과 4개의 출력(aout, bout, cout, dout)을 가지고 있다. 4개의 입력은 8421 Code의 각 자릿수를 나타내고, 4개의 출력은 2421 Code의 각 자릿수를 나타낸다. aout은 ain|(bin&cin)|(bin&din)의 결과이고, bout은 ain|(bin&cin)|(bin&~din)의 결과이고, cout은 ain|(bin&~cin&din)| (~bin&cin)의 결과이고 dout은 din이다.

* Schematic Diagram

도표, 텍스트, 평면도, 라인이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 verilog 코드로 Schematic Diagram을 생성한 결과이다. 이를 통해 작성한 코드가 회로에 어떻게 구성되었는지 확인할 수 있다.

* Simulation 결과

스크린샷, 사각형, 컴퓨터이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드를 시뮬레이션 돌린 결과이다. 시간에 따라 입력 ain, bin, cin, din의 값을 바꾸면 그에 따라 출력 aout, bout, cout, dout의 값이 변화하는 것을 확인할 수 있다. 각 입력 값에 대한 출력 값을 관찰하면(특히 입력 값이 5 이상) 위에서 작성한 진리표와 동일하게 동작하는 것을 알 수 있다. 이를 통해 K-map 작성으로 얻은 minimum SOP가 의도한 대로 잘 동작한다는 것을 실험적으로 확인할 수 있었다.

**5.**

Half-adder, Full-adder, Half-subtractor, Full-subtractor를 예비 보고서에서 미리 조사해 각 회로가 어떻게 동작할지 예측할 수 있었고, 모두 의도했던 바와 동일하게 작동하는 것을 확인해 모든 구현이 성공적으로 이루어졌음을 알 수 있었다. 8421-2421 code converter 구현 과정에서는 정해진 논리식이 없는 경우 K-map 작성을 통해 논리식을 구할 수 있다는 것을 알게 되었다. 이때 NAND, NOR 게이트만 사용하여 회로를 만들면 비용 절감의 효과가 있어 구현해보고 싶었지만 논리식을 변형하는데 시간 소모가 커서 시도하지 못했다. 후에 NAND, NOR 게이트만 이용해 논리식을 구성하는 연습을 해본다면 실력 향상에 도움이 될 것 같다.

**6.**

코드 변환기는 데이터의 형식을 변환하거나 표현하는 데 도움을 주는 장치를 말한다. 아래는 이번 실습에서 다룬 8421-2421 코드 변환기 외에 다양한 코드 변환기들을 조사한 것이다.

* 10진수 - 2진수 변환기: 10진수 숫자를 2진수로 변환하는데 사용된다. 주로 컴퓨터에서 사용하는 데이터 형식을 이해하는 데 도움이 된다.
* 2진수 - 10진수 변환기: 2진수로 표현된 숫자를 10진수로 변환하는데 사용된다. 컴퓨터 프로그램에서 2진수 데이터를 사람이 이해할 수 있는 형식으로 표시하는 데 유용하다.
* 16진수 - 2진수 변환기: 16진수로 표현된 숫자를 2진수로 변환하는데 사용된다. 이 변환은 컴퓨터에서 사용되는 데이터 형식을 표현하는 데 편리하다.
* 2진수 - 16진수 변환기: 2진수로 표현된 숫자를 16진수로 변환하는데 사용된다. 이 변환은 데이터를 간결하게 표현하고 읽기 쉽게 만들 수 있다.
* BCD - 이진 변환기: 8421 BCD 코드를 2진수로 변환하는데 사용된다. 이때 BCD 코드는 10진수 숫자를 4비트 이진 코드로 나타내는 데 사용된다.
* ASCII - 2진수 변환기: ASCII 문자를 8비트 이진 코드로 변환하는데 사용된다. 이 변환은 문자열 데이터를 컴퓨터에서 저장하거나 전송하는 데 필요하다.
* Gray 코드 - 2진수 변환기: Gray 코드를 2진수로 변환하는데 사용된다. 이때 Gray 코드는 이진 코드 간의 전이가 한 비트만 변경되는 특별한 숫자 체계이다.
* BCD - Excess-3 변환기: BCD 코드를 Excess-3 코드로 변환하는데 사용된다. Excess-3 코드는 8421 BCD 코드를 수정한 4비트 10진수 코드이다.
* Excess-3 - BCD 변환기: Excess-3 코드를 BCD 코드로 변환하는데 사용된다. 이 변환은 디지털 회로 및 표시 장치에서 사용된다.
* Binary Coded Gray (BCG) - 2진수 변환기: BCG 코드를 2진수로 변환하는데 사용된다. BCG 코드는 Gray 코드를 이진 코드로 나타내는 방법 중 하나이다.