전공: 컴퓨터공학 학년: 2학년 학번: 20161603 이름: 신민준

1. 실험목적

이번 실험에서는 Adder/Subtractor를 Verilog를 사용해 구현해보면서 가산기/감산기의 개념을 이해하고자 했다. 또한, 8421 BCD Code를 2421 Code로 변환하는 Code converter를 구현해 K-Map의 minimum POS, SOP form을 구하고, 이를 실제 FPGA에 적용시켜 확인했다.

1. Full Adder 및 Half Adder의 simulation 결과 및 과정에 대해서 설명하시오.

* Half Adder

Half Adder의 Truth Table은 다음과 같다.

|  |  |  |  |
| --- | --- | --- | --- |
| Input |  | Output |  |
| a | b | s | c |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

이를 구현하기 위해 그린 K-Map은 다음에서 보이는 것과 같다.

|  |  |  |
| --- | --- | --- |
| a b | 0 | 1 |
| 0 | 0 | 1 |
| 1 | 1 | 0 |

|  |  |  |
| --- | --- | --- |
| a b | 0 | 1 |
| 0 | 0 | 0 |
| 1 | 0 | 1 |

위에서 구한 식을 사용해 Half Adder를 구현하기 위해 작성한 Verilog Code는 다음과 같다.

|  |
| --- |
| `timescale 1ns / 1ps  module half\_adder(      input a, b,      output s, c      );      assign s = a^b;      assign c = a&b;  endmodule |

이 Verilog Code의 Schematic은 다음과 같다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

Simulation을 위해 작성한 simulation code는 다음과 같다.

|  |
| --- |
| `timescale 1ns / 1ps  module sim;      reg [1:0] in;      wire s, c;      half\_adder u\_half\_adder(in[1], in[0], s, c);      initial in = 2'b00;        always in = #100 in+1;      initial begin          #1000          $finish;      end  endmodule |

해당 simulation을 수행한 결과는 다음과 같다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

* Full Adder

Full Adder의 Truth Table은 다음과 같다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input |  |  | Output |  |
| cin | b | a | s | cout |
| 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 |

이를 구현하기 위해 그린 K-Map은 다음에서 보이는 것과 같다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| c ba | 00 | 01 | 11 | 10 |
| 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| c ba | 00 | 01 | 11 | 10 |
| 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 |

위에서 구한 식을 사용해 Full Adder를 구현하기 위해 작성한 Verilog Code는 다음과 같다.

|  |
| --- |
| `timescale 1ns / 1ps  module full\_adder(      input a, b, cin,      output s, cout      );      assign s = a^b^cin;      assign cout = (a&b)|((a^b)&cin);  endmodule |

이 Verilog Code의 Schematic은 다음과 같다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

Simulation을 위해 작성한 simulation code는 다음과 같다.

|  |
| --- |
| `timescale 1ns / 1ps  module sim;      reg [2:0] in;      wire s, c;      full\_adder u\_full\_adder(in[0], in[1], in[2], s, c);      initial in = 3'b000;        always in = #100 in+1;      initial begin          #1000          $finish;      end  endmodule |

해당 simulation을 수행한 결과는 다음과 같다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

1. Full Subtractor 및 Half Subtractor의 simulation 결과 및 과정에 대해서 설명하시오.

* Half Subtractor

Half Subtractor의 Truth Table은 다음과 같다.

|  |  |  |  |
| --- | --- | --- | --- |
| a | b | borrow | D |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 |

이를 구현하기 위해 그린 K-Map은 다음에서 보이는 것과 같다.

|  |  |  |
| --- | --- | --- |
| a b | 0 | 1 |
| 0 | 0 | 1 |
| 1 | 0 | 0 |

|  |  |  |
| --- | --- | --- |
| a b | 0 | 1 |
| 0 | 0 | 1 |
| 1 | 1 | 0 |

위에서 구한 식을 사용해 Half Subtractor를 구현하기 위해 작성한 Verilog Code는 다음과 같다.

|  |
| --- |
| `timescale 1ns / 1ps  module half\_sub(      input a, b,      output borrow, d      );      assign borrow = ~a&b;      assign d = a^b;  endmodule |

이 Verilog Code의 Schematic은 다음과 같다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

Simulation을 위해 작성한 simulation code는 다음과 같다.

|  |
| --- |
| `timescale 1ns / 1ps  module sim;      reg [1:0] in;      wire borrow, d;      half\_sub u\_half\_sub(in[1], in[0], borrow, d);      initial in = 2'b00;        always in = #100 in+1;      initial begin          #1000          $finish;      end  endmodule |

해당 simulation을 수행한 결과는 다음과 같다.

스크린샷, 전자기기이(가) 표시된 사진

자동 생성된 설명

* Full Subtractor

Full Subtractor의 Truth Table은 다음과 같다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input |  |  | Output |  |
| a | b | bin | bout | 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 |

이를 구현하기 위해 그린 K-Map은 다음에서 보이는 것과 같다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a bbin | 00 | 01 | 11 | 10 |
| 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a bbin | 00 | 01 | 11 | 10 |
| 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |

위에서 구한 식을 사용해 Full Subtractor를 구현하기 위해 작성한 Verilog Code는 다음과 같다.

|  |
| --- |
| `timescale 1ns / 1ps  module full\_sub(      input a, b, bin,      output bout, d      );      assign bout = (bin&(~a|b))|(~a&b);      assign d = a^b^bin;  endmodule |

이 Verilog Code의 Schematic은 다음과 같다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

Simulation을 위해 작성한 simulation code는 다음과 같다.

|  |
| --- |
| `timescale 1ns / 1ps  module sim;      reg [2:0] in;      wire borrow, d;      full\_sub u\_full\_sub(in[2], in[1], in[0], bout, d);      initial in = 3'b000;        always in = #100 in+1;      initial begin          #1000          $finish;      end  endmodule |

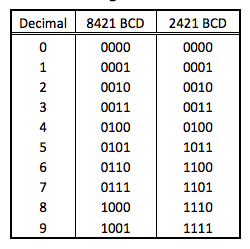
해당 simulation을 수행한 결과는 다음과 같다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

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

BCD Code와 2421 Code 사이에는 다음 표에서 보이는 것의 차이가 있다.



따라서, 이 변환기를 진리표로 나타낸다면 다음과 같다. 이 때 a, b, c, d는 8421 Code의 각 digit이며, e, f, g, h는 2421 Code의 각 digit이다.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| value | a | b | c | d | e | f | g | h |
| 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 |
| 12~15 | 1 | 1 | x | x | x | x | x | x |

이 진리표를 바탕으로 K-map을 그려 minimum SOP와 minimum POS를 구하면 다음과 같다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ab cd | 00 | 01 | 11 | 10 |
| 00 | 0 | 0 | 0 | 0 |
| 01 | 0 | 1 | 1 | 1 |
| 11 | x | x | x | x |
| 10 | 1 | 1 | x | x |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ab cd | 00 | 01 | 11 | 10 |
| 00 | 0 | 0 | 0 | 0 |
| 01 | 1 | 0 | 1 | 1 |
| 11 | x | x | x | x |
| 10 | 1 | 1 | x | x |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ab cd | 00 | 01 | 11 | 10 |
| 00 | 0 | 0 | 1 | 1 |
| 01 | 0 | 1 | 0 | 0 |
| 11 | x | x | x | x |
| 10 | 1 | 1 | x | x |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ab cd | 00 | 01 | 11 | 10 |
| 00 | 0 | 1 | 1 | 0 |
| 01 | 0 | 1 | 1 | 0 |
| 11 | x | x | x | x |
| 10 | 0 | 1 | x | x |

위 식을 implement한 Verilog Code는 다음과 같다.

|  |
| --- |
| `timescale 1ns / 1ps  module converter(      input a, b, c, d,      output e, f, g, h      );      assign e = a|(b&d)|(b&c);      assign f = a|(b&c)|(b&~d);      assign g = (b&~c&d)|a|(~b&c);      assign h = d;  endmodule  module converterPOS(      input a, b, c, d,      output e, f, g, h      );      assign e = (a|c|d)&(a|b);      assign f = (a|b)&(a|c|~d);      assign g = (a|b|c)&(a|c|d)&(~b|~c);      assign h = d;  endmodule |

이 Verilog Code의 Schematic은 다음과 같다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

Simulation을 수행하기 위해 작성한 simulation code는 다음과 같다.

|  |
| --- |
| `timescale 1ns / 1ps  module convsim;      reg [3:0] in;      wire e, f, g, h;      converter u\_converter(in[3], in[2], in[1], in[0], e, f, g, h);      initial in = 4'b0000;        always in = #100 in+1;      initial begin          #2000          $finish;      end  endmodule |

Simulation 결과는 다음과 같이 나왔다.

스크린샷, 전자기기이(가) 표시된 사진

자동 생성된 설명

1. 결과 검토 및 논의사항

Half-adder, Full-adder, Half-subtractor, Full-subtractor, 8421 code converter 모두 의도했던 바와 동일하게 작동함을 볼 수 있었기에 모든 implementation이 성공적으로 이루어졌음을 알 수 있다.

1. 추가 이론 조사 및 작성

* 8421 BCD to 2421 code converter의 경우, 4개의 함수를 동시에 구현해야 하기 때문에, PLA와 같은 회로에 적용하기 위해선 각 함수의 minimal SOP/POS form보다 더 효율적으로 회로를 구성해야 할 필요가 있을 것이다. 공유되는 term들의 갯수가 최대가 되도록 구성하면 총 회로를 구성하는데 드는 cost를 줄일 수 있다.