**6주차 결과 보고서**

20150555 남민혁

**1. 실험 목적**

6주차 실험의 목적은 아래와 같다.

* Adder/Subtractor 의 개념 이해
* Code converter의 개념 이해
* Verilog를 사용하여 다양한 Adder와 Subtractor 구현
* Verilog를 사용하여 다양한 Code converter 구현

**2. Full Adder 및 Half Adder 의 simulation 결과 및 과정에 대해서 설명하시오.**

**1) Full Adder**

**-Truth Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input | | | Output | |
| A | B |  | 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 |

**-Karnaugh Map (S)**

|  |  |  |
| --- | --- | --- |
| AB | 0 | 1 |
| 00 | 0 | 1 |
| 01 | 1 | 0 |
| 11 | 0 | 1 |
| 10 | 1 | 0 |

**-Karnaugh Map ()**

|  |  |  |
| --- | --- | --- |
| AB | 0 | 1 |
| 00 | 0 | 0 |
| 01 | 0 | 1 |
| 11 | 1 | 1 |
| 10 | 0 | 1 |

**-최소화된 Boolean 함수**

**-Verilog 코딩**

|  |  |
| --- | --- |
| Verilog | `timescale 1ns / 1ps  module full\_adder(  input a,  input b,  input cin,  output s,  output cout  );  assign s = (a ^ b) ^ cin;  assign cout = cin & (a ^ b) | (a & b);  endmodule |
| Test Bench | `timescale 1ns / 1ps  module full\_adder\_tb;  reg aa;  reg bb;  reg ccin;  wire ss;  wire ccout;  full\_adder u\_full\_adder(  .a (aa),  .b (bb),  .cin (ccin),  .s (ss),  .cout (ccout)  );  initial aa = 1'b0;  initial bb = 1'b0;  initial ccin = 1'b0;  always aa = #400 ~ aa;  always bb = #200 ~bb;  always ccin = #100 ~ccin;  initial begin  #800  $finish;  end  endmodule |

**-Simulation 출력 결과 비교**

**Graphical user interface

Description automatically generated**

구현 결과가 진리표와 동일하게 출력됨을 확인하였다.

**-구현된 Schematic**

Diagram, schematic

Description automatically generated

**2) Half Adder**

**-Truth Table**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Input | | | Output | | |
| A | B | S | | C |
| 0 | 0 | 0 | | 0 |
| 0 | 1 | 1 | | 0 |
| 1 | 0 | 1 | | 0 |
| 1 | 1 | 0 | | 1 |

**-Karnaugh Map (S)**

|  |  |  |
| --- | --- | --- |
| B  A | 0 | 1 |
| 0 | 0 | 1 |
| 1 | 1 | 0 |

**-Karnaugh Map (C)**

|  |  |  |
| --- | --- | --- |
| B  A | 0 | 1 |
| 0 | 0 | 0 |
| 1 | 0 | 1 |

**-최소화된 Boolean 함수**

**-Verilog 코딩**

|  |  |
| --- | --- |
| Verilog | `timescale 1ns / 1ps  module half\_adder(  input a,  input b,  output s,  output cout  );  assign s = a ^ b;  assign cout = a & b;  endmodule |
| Test Bench | `timescale 1ns / 1ps  module half\_adder\_tb;  reg aa;  reg bb;  wire ss;  wire ccout;  half\_adder u\_half\_adder(  .a (aa),  .b (bb),  .s (ss),  .cout (ccout)  );  initial aa = 1'b0;  initial bb = 1'b0;  always aa = #200 ~ aa;  always bb = #100 ~bb;  initial begin  #400  $finish;  end  endmodule |

**-Simulation 출력 결과 비교**

**A picture containing sitting, green, player, holding

Description automatically generated**

구현 결과가 진리표와 동일하게 출력됨을 확인하였다.

**-구현된 Schematic**

Diagram

Description automatically generated

**3. Full Subtractor 및 Half Subtracter 의 simulation 결과 및 과정에 대해서 설명하시오.**

**1) Full Subtractor**

**-Truth Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input | | | Output | |
| A | B |  | D |  |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

**-Karnaugh Map (D)**

|  |  |  |
| --- | --- | --- |
| AB | 0 | 1 |
| 00 | 0 | 1 |
| 01 | 1 | 0 |
| 11 | 0 | 1 |
| 10 | 1 | 0 |

**-Karnaugh Map ()**

|  |  |  |
| --- | --- | --- |
| AB | 0 | 1 |
| 00 | 0 | 1 |
| 01 | 1 | 1 |
| 11 | 0 | 1 |
| 10 | 0 | 0 |

**-최소화된 Boolean 함수**

**-Verilog 코딩**

|  |  |
| --- | --- |
| Verilog | `timescale 1ns / 1ps  module full\_subtractor(  input a,  input b,  input bin,  output d,  output bout  );  assign d = (a ^ b) ^ bin;  assign bout = (~(a ^ b) & bin) | (~a & b);  endmodule |
| Test Bench | `timescale 1ns / 1ps  module full\_subtractor\_tb;  reg aa;  reg bb;  reg bbin;  wire dd;  wire bbout;  full\_subtractor u\_full\_subtractor(  .a (aa),  .b (bb),  .bin (bbin),  .d (dd),  .bout (bbout)  );  initial aa = 1'b0;  initial bb = 1'b0;  initial bbin = 1'b0;  always aa = #400 ~ aa;  always bb = #200 ~bb;  always bbin = #100 ~bbin;  initial begin  #800  $finish;  end  endmodule |

**-Simulation 출력 결과 비교**

**Graphical user interface

Description automatically generated**

구현 결과가 진리표와 동일하게 출력됨을 확인하였다.

**-구현된 Schematic**

Diagram, schematic

Description automatically generated

**2) Half Subtractor**

**-Truth Table**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Input | | | Output | | |
| A | B | D | |  |
| 0 | 0 | 0 | | 0 |
| 0 | 1 | 1 | | 1 |
| 1 | 0 | 1 | | 0 |
| 1 | 1 | 0 | | 0 |

**-Karnaugh Map (D)**

|  |  |  |
| --- | --- | --- |
| B  A | 0 | 1 |
| 0 | 0 | 1 |
| 1 | 1 | 0 |

**-Karnaugh Map ()**

|  |  |  |
| --- | --- | --- |
| B  A | 0 | 1 |
| 0 | 0 | 1 |
| 1 | 0 | 0 |

**-최소화된 Boolean 함수**

**-Verilog 코딩**

|  |  |
| --- | --- |
| Verilog | `timescale 1ns / 1ps  module half\_subtractor(  input a,  input b,  output d,  output bout  );  assign d = a ^ b;  assign bout = ~a & b;  endmodule |
| Test Bench | `timescale 1ns / 1ps  module half\_subtractor\_tb;  reg aa;  reg bb;  wire dd;  wire bbout;  half\_subtractor u\_half\_subtractor(  .a (aa),  .b (bb),  .d (dd),  .bout (bbout)  );  initial aa = 1'b0;  initial bb = 1'b0;  always aa = #200 ~ aa;  always bb = #100 ~bb;  initial begin  #400  $finish;  end  endmodule |

**-Simulation 출력 결과 비교**

**A picture containing graphical user interface

Description automatically generated**

구현 결과가 진리표와 동일하게 출력됨을 확인하였다.

**-구현된 Schematic**

Diagram

Description automatically generated

**4. 8421(BCD)-2421 Code converter simulation 결과 및 과정에 대해서 설명하시오. (진리표 작성 및 카로노맵 SOP form, POS form 포함)**

**-Truth Table**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input (8421BCD)** | | | | **Output (2421BCD)** | | | |
| A | B | C | D | W | X | Y | Z |
| 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 |

**-Karnaugh Map (W)**

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

**-Karnaugh Map (X)**

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

**-Karnaugh Map (Y)**

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

**-Karnaugh Map (Z)**

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

**-최소화된 Boolean 함수**

**-Verilog 코딩**

|  |  |
| --- | --- |
| Verilog | `timescale 1ns / 1ps  module code\_converter(  input a,  input b,  input c,  input d,  output w,  output x,  output y,  output z  );  assign w = a | (b & (c | d));  assign x = a | (b & (c | ~d));  assign y = a | (~b & c) | (b & ~c & d);  assign z = d;  endmodule |
| Test Bench | `timescale 1ns / 1ps  module code\_converter\_tb;  reg aa;  reg bb;  reg cc;  reg dd;  wire ww;  wire xx;  wire yy;  wire zz;  code\_converter u\_code\_converter(  .a (aa),  .b (bb),  .c (cc),  .d (dd),  .w (ww),  .x (xx),  .y (yy),  .z (zz)  );  initial aa = 1'b0;  initial bb = 1'b0;  initial cc = 1'b0;  initial dd = 1'b0;  always aa = #400 ~ aa;  always bb = #200 ~bb;  always cc = #100 ~cc;  always dd = #50 ~dd;  initial begin  #800  $finish;  end  endmodule |

**-Simulation 출력 결과 비교**

**Graphical user interface, chart

Description automatically generated**

구현 결과가 진리표와 동일하게 출력됨을 확인하였다.

**-구현된 Schematic**

Diagram, schematic

Description automatically generated

**5. 결과 검토 및 논의사항**

이번주 실험에서 가산기, 감산기와 Code converter를 구현하였다. 구현을 위하여 주어진 논리에 대하여 진리표를 구성한 후, 이를 활용하여 Karnaugh Map를 이용하여 SOP와 POS 형태로 최소화 하였다. 끝으로 최소화한 식을 Verilog 코드로 작성하고 모든 가능한 입력에 대해 simulate 해보았다. 그 결과 모든 디자인이 의도한 진리표와 다르지 않고 동일하게 결과가 출력됨을 확인하였다.

**6. 추가 이론 조사 및 작성**

본 실험에서는 하나의 비트로 구성된 입력값 두개에 대하여 더하고, 빼는 연산을 수행하는 가감산기를 제작하였다. 그러나 일반적으로 우리는 N비트로 표현된 두개의 입력값을 연산하며, 이는 Full adder와 Full subtractor를 N개 병렬적으로 연결함으로써 간단하게 N비트 연산기로 확장할 수 있다.

그러나 상위 비트 연산을 위해서는 반드시 하위 비트의 연산이 완료되어 carry나 borrow bit이 생성되어야만 연산이 가능하여, 연산 시간의 측면에서 느리다. 이를 개선한 것이 carry lookahead adder로서, 주어진 n개의 input을 활용하여 미리 carry를 계산하고, 각 비트는 주어진 비트의 입력만으로 개별적으로 연산을 수행할 수 있도록 고안되었다.

예를 들어 4비트 carry lookahead adder에 대하여 각 출력값의 불리안 식은 다음과 같다.

이를 바탕으로 Carry bit를 출력하는 Schematic은 다음과 같이 구현할 수 있다.

Diagram, schematic

Description automatically generated

**7. 참고 문헌**

Geekforgeeks, “Carry Look-Ahead Adder”, https://www.geeksforgeeks.org/carry-look-ahead-adder/.

강석태, “Verilog HDL Summary”, http://vlsi.hongik.ac.kr/lecture/%EC%8B%A4%ED%97%98/Verilog\_Summary.pdf.

한국기술교육대학교 전기전자통신공학부, “디지털시스템 설계 및 실습” Verilog HDL 문법 강의자료, https://cms3.koreatech.ac.kr/sites/yjjang/down/dsys11/M01\_VerilogHDL01.pdf.

장영조, “디지털공학 및 실습 6장 강의자료: 부울함수의 간소화”, https://cms3.koreatech.ac.kr/sites/yjjang/down/digi09/ch06.pdf.