LDLAB5

수리과학부 2019-16022 박채연

0. Discussion

(1) Concepts that I learned during the lab session

이론 수업시간에 FPGA에 대해 다룬 적이 있는데 프로그래밍을 통해 회로 동작을 설계한다는 것이 구체적으로 무엇을 의미하는지 정확히 이해하지 못했었는데, 본 랩 시간에 FPGA에 직접적으로 프로그래밍을 하는 것이 아니라 컴퓨터에 연결하여 프로그래밍을 하는 것임을 배웠다. 또한, FPGA가 SNU Logic Design Board 전체를 의미하는 것이 아니라 칩처럼 생긴 부품을 의미하는 것임을 알게 되었다.

과제 파트를 수행하면서 1bit ALU가 동작하는 방식에 대해 배웠다. 교수님께서 이론 시간에 결과값을 모두 미리 계산해두고 control input에 따라 output을 출력하는 방식으로 ALU가 동작한다고 설명하신 적이 있다. 실제로 1bit ALU를 베릴로그로 코딩하면서 해당 의미를 조금 더 자세히 이해할 수 있었다. 예를 들어 M=0, S1=0, S0=0이고 A=1이면 P87에 led 불빛이 들어와야하는데, M=0, S1=0, S0=0 값이 지정되었을 때 A=1인지 확인하여 output을 출력하는 것이 아니라 미리 control input 조합에 따른 결과 값을 계산해두고 M=0, S1=0, S0=0이면 미리 계산해둔 결과값 중 control input에 해당되는 결과값 하나를 출력하는 방식이다.

(2) Any errors that I made

SNU Logic Design Board를 컴퓨터에 연결하여 초기 테스트를 진행해보았지만 Xilinx에서 Cable이 인식되지 않는다는 에러 메시지가 지속해서 떴다. 프로젝트 설정이 잘못되었을 수도 있다고 생각하여 프로젝트를 다시 만들어보기도 하고 컴퓨터를 껐다 켜보기도 했지만 작동하지 않았다.

1bit ALU를 구현하는 과정에서 빠르게 진행하고자 하다보니 output이 1개가 나오도록 하는 실수를 했다. 즉, M=0, S1=0, S0=0일 때 A 값이 1이면 P87에 led 불빛이 들어오고, M=0, S1=0, S0=1일 때 A 값이 0이면 P88에 led 불빛이 들어오도록 output을 각각 지정해야하는데, M, S1, S0의 조합에 따른 8가지 모든 경우에 해당하는 결과를 or로 연결하여 8가지 조건 중 하나가 성립하면 P87에 불이 들어오도록 프로그래밍을 했다. PPT 슬라이드를 보니 output을 위해서는 총 6개의 led (P87, P88, P90, P91, P92, P93)를 사용해야한다고 되어있어서 잘못 프로그래밍했다는 것을 발견하였고 해당 부분을 수정하였다. 시뮬레이션 결과가 정확하게 나오는 것을 확인하고 회로에서 Dip Switch를 조작해가며 32가지 모든 경우에 대해 사진까지 찍었지만 팀원 중 한 명이 결과가 이상한 것 같다고 하였다. 확인해보니 총 6개의 led 가운데 처음 2개의 led를 제외한 나머지 led에서는 불빛이 들어오지 않은 것을 확인하였다. 마지막에 결국 해당 오류 발생 원인을 발견하여 정상적으로 동작하는 것을 확인하였다.

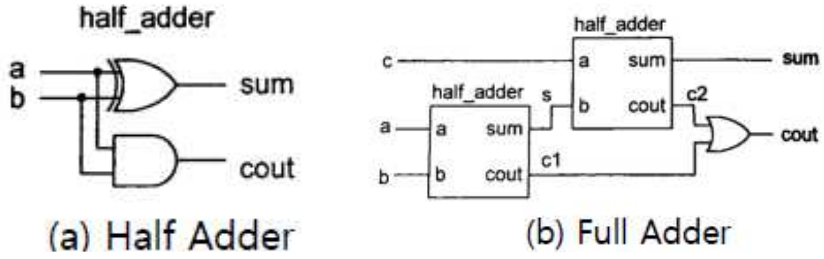
(3) How to correct my errors

조교님께서 Xilinx에서 Cable이 인식되지 않는 문제가 컴퓨터와 FPGA가 매우 오래되었기 때문이라고 하셨다. 새로운 케이블을 주셔서 다시 시도해보았지만 여전히 문제는 지속되었고 옆 자리 빈 컴퓨터에서 처음부터 다시 시도하였는데 드디어 제대로 동작하는 것을 확인할 수 있었다.

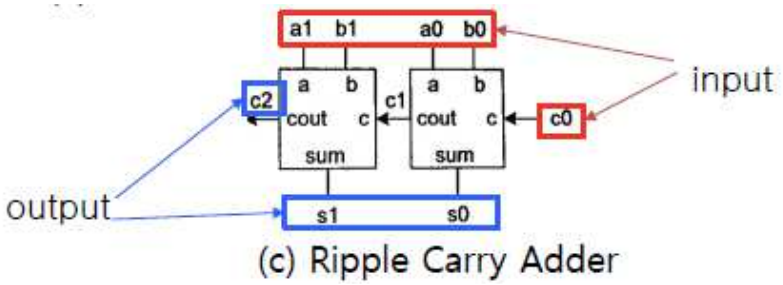
1bit ALU에 대해 output led를 총 6개 사용해야 한다는 것을 알고 빠르게 수정하였다. 시뮬레이션 결과 32가지 모든 경우의 수에 대해 정상적으로 동작하였다. 그러나 실제로 회로의 Tactile Switch를 눌러보면 총 6개의 led 가운데 맨 앞 두 개만 불이 들어왔는데, 마지막 input(=S1)에 대한 DIP switch의 포트 번호를 41번이 아니라 51번으로 매칭시켜주었기 때문임을 발견하였다. 포트 번호를 바꿔주니 시뮬레이션 뿐만 아니라 SNU Logic Design Board 역시 올바르게 동작하는 것을 확인하였다.

1. Lab Part

Implement ripple-carry adder, shown below, in Verilog, and program it on the FPGA board.

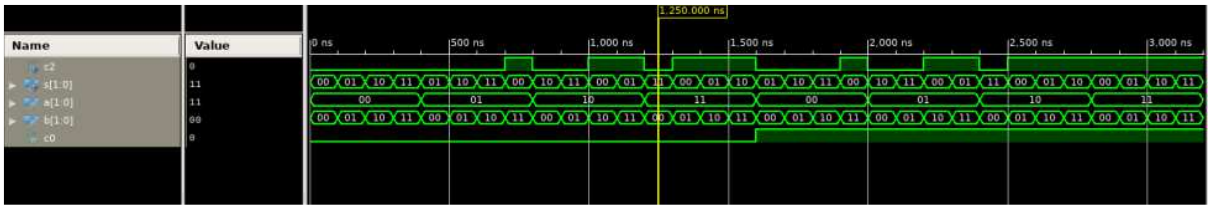


우선 위의 이미지를 참고하여 half adder를 구현하였고, half adder를 이용하여 full_adder를 구현하였다. 관련 Verilog 파일은 각각 'half_adder.v'와 'full_adder.v'이다.



다음으로, 구현한 full adder를 두 개 사용하여 ripple carry adder를 'ripple_carry_adder.v' 파일에 구현하였다. 이때, half adder와 full adder, ripple carry adder는 각각 Structural Description 방법으로 구현하였다.

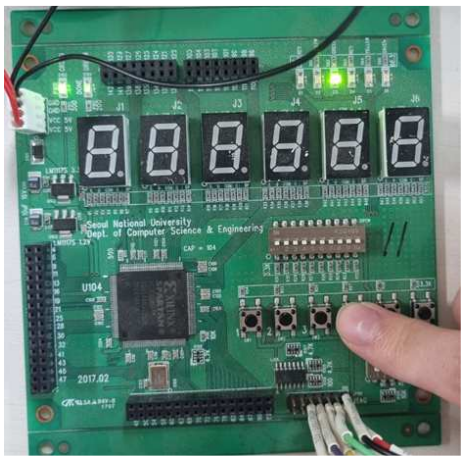
'ripple_carry_adder_test.v' 파일을 생성하여 앞에서 구현한 ripple carry adder의 동작을 5개의 input(a1, b1, a0, b0, c0) 조합에 따른 모든 경우의 수에 대해 검증하는 코드를 작성하였다. ripple carry adder의 시뮬레이션 결과는 아래와 같다.



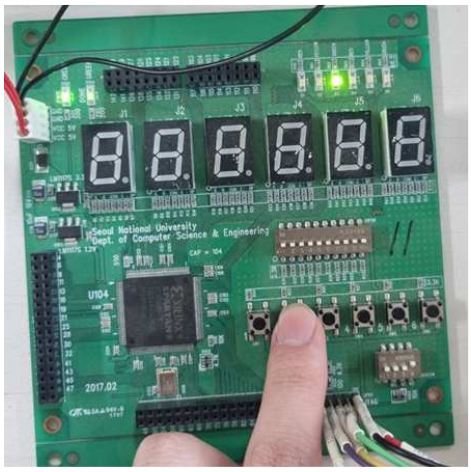
구현한 ripple carry adder를 FPGA board에 program한 결과는 아래와 같다. 이때, 총 5개의 tactile switch 를 사용하였는데, 왼쪽부터 순서대로 a1, b1, a0, b0, c0 를 나타낸다. 또한, 3개의 led 를 사용하였는데, 붉은 색은 c2, 주황 색은 s1, 노란 색은 s0를 의미한다. FPGA board의 동작 결과가 위의 시뮬레이션 결과와 정확히 일치하는 것을 확인할 수 있다.

(1) a1 = 0, b1 = 0, a0 = 0, b0 = 0, c0 = 0인 경우

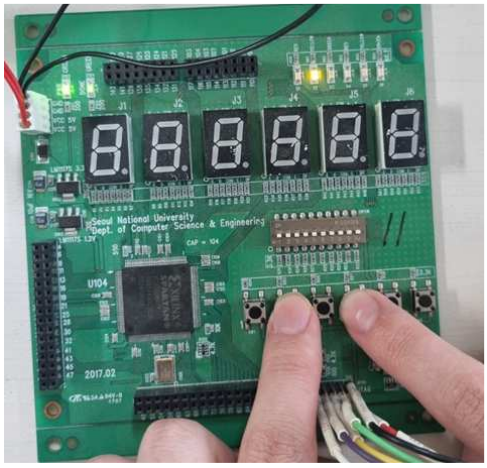
(2) a1 = 0, b1 = 0, a0 = 0, b0 = 1, c0 = 0인 경우



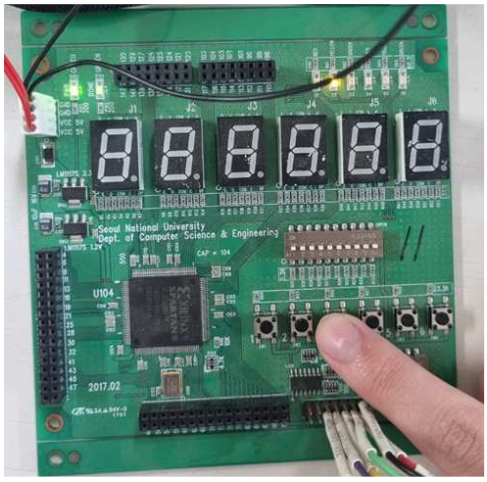
(3) a1 = 0, b1 = 1, a0 = 0, b0 = 0, c0 = 0인 경우



(4) a1 = 0, b1 = 1, a0 = 0, b0 = 1, c0 = 0인 경우



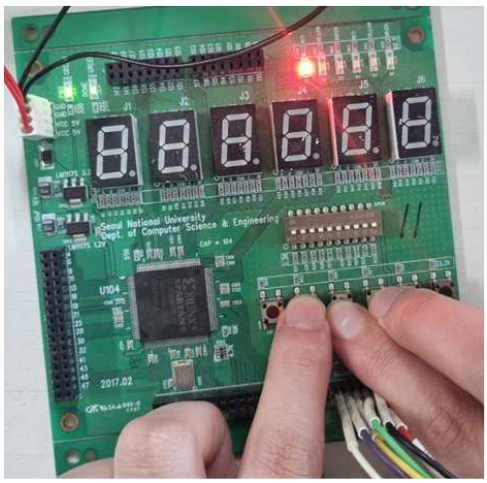
(5) a1 = 0, b1 = 0, a0 = 1, b0 = 0, c0 = 0인 경우



(6) a1 = 0, b1 = 0, a0 = 1, b0 = 1, c0 = 0인 경우

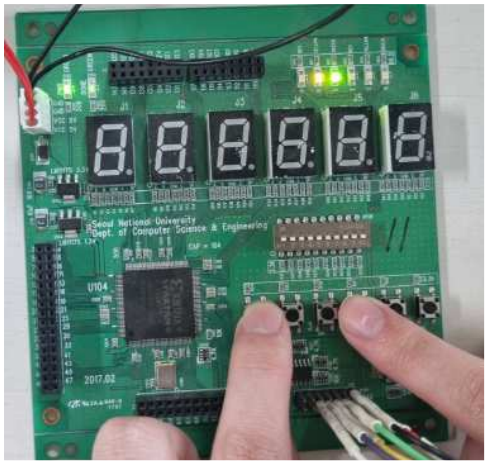
(7) a1 = 0, b1 = 1, a0 = 1, b0 = 0, c0 = 0인 경우

(8) a1 = 0, b1 = 1, a0 = 1, b0 = 1, c0 = 0인 경우



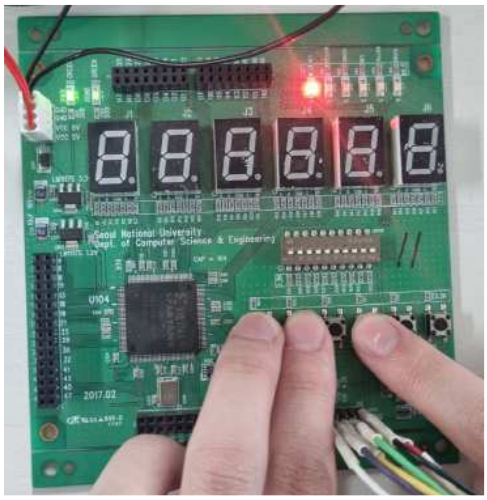
(9) a1 = 1, b1 = 0, a0 = 0, b0 = 0, c0 = 0인 경우

(10) a1 = 1, b1 = 0, a0 = 0, b0 = 1, c0 = 0인 경우

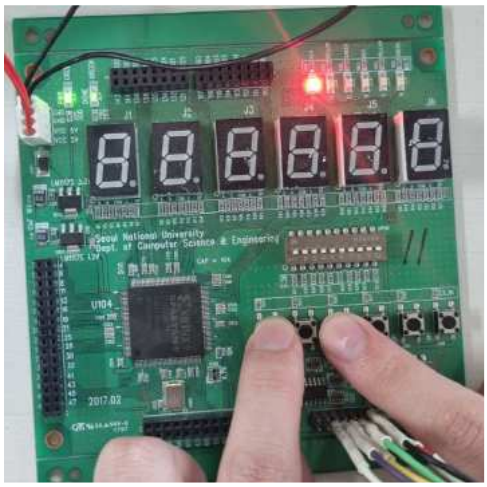


(11) a1 = 1, b1 = 1, a0 = 0, b0 = 0, c0 = 0인 경우

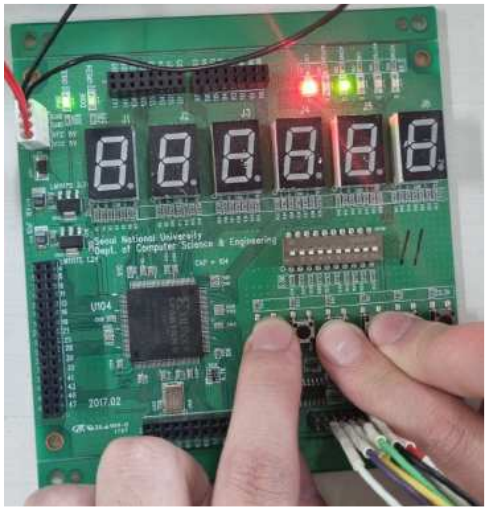
(12) a1 = 1, b1 = 1, a0 = 0, b0 = 1, c0 = 0인 경우



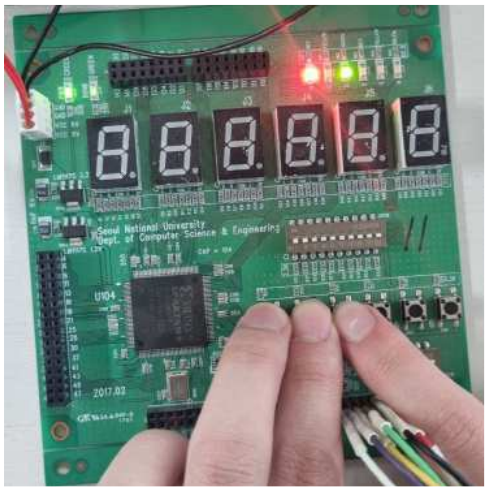
(13) a1 = 1. b1 = 0, a0 = 1, b0 = 0, c0 = 0인 경우



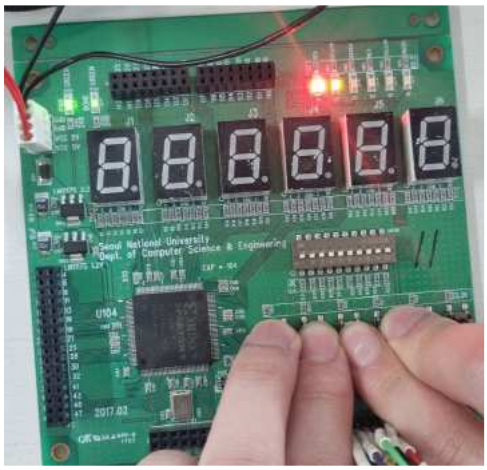
(14) a1 = 1. b1 = 0, a0 = 1, b0 = 1, c0 = 0인 경우



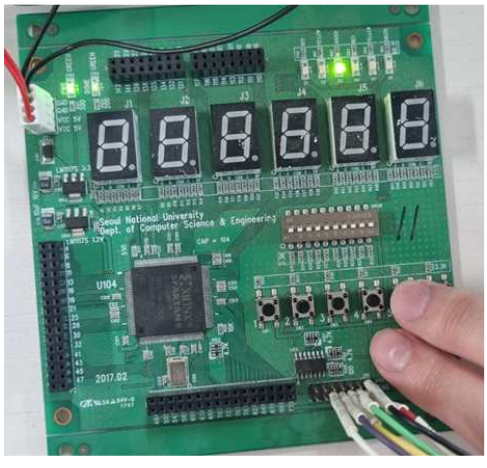
(15) a1 = 1. b1 = 1, a0 = 1, b0 = 0, c0 = 0인 경우



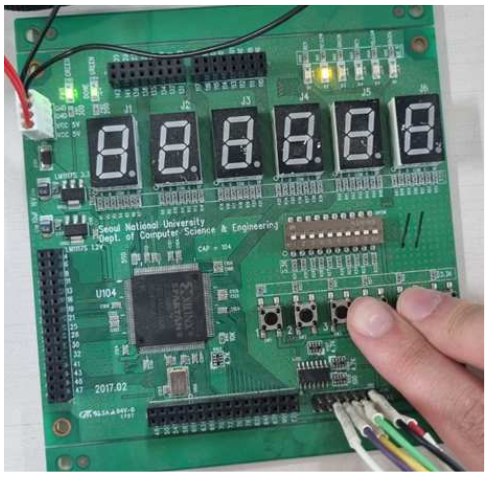
(16) a1 = 1. b1 = 1, a0 = 1, b0 = 1, c0 = 0인 경우



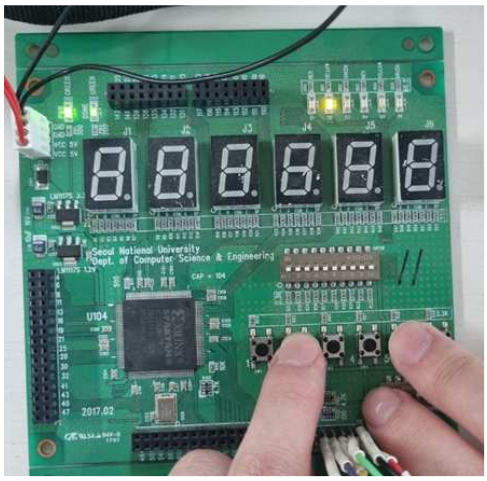
(17) a1 = 0, b1 = 0, a0 = 0, b0 = 0, c0 = 1인 경우



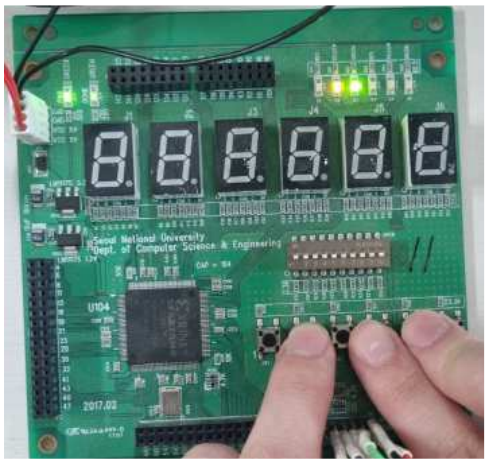
(18) a1 = 0, b1 = 0, a0 = 0, b0 = 1, c0 = 1인 경우



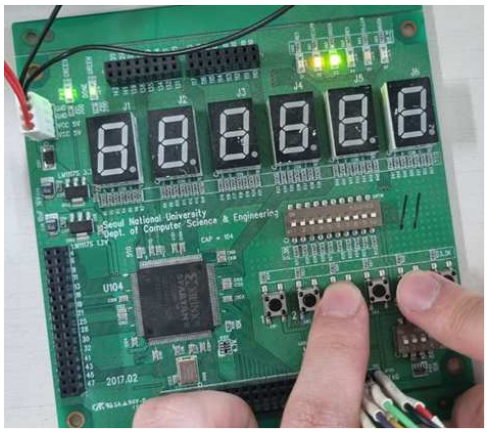
(19) a1 = 0, b1 = 1, a0 = 0, b0 = 0, c0 = 1인 경우



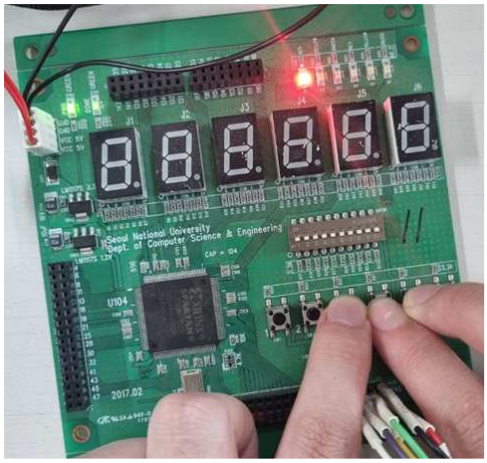
(20) a1 = 0, b1 = 1, a0 = 0, b0 = 1, c0 = 1인 경우



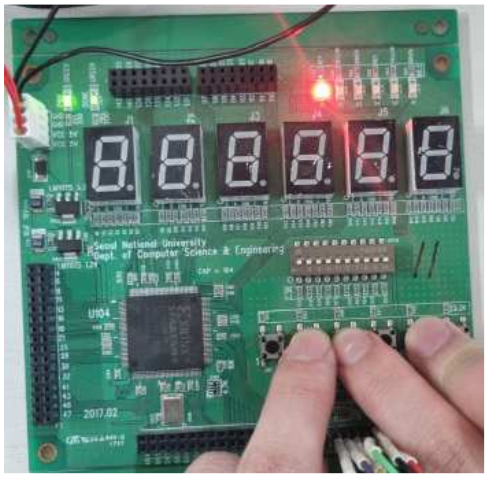
(21) a1 = 0, b1 = 0, a0 = 1, b0 = 0, c0 = 1인 경우



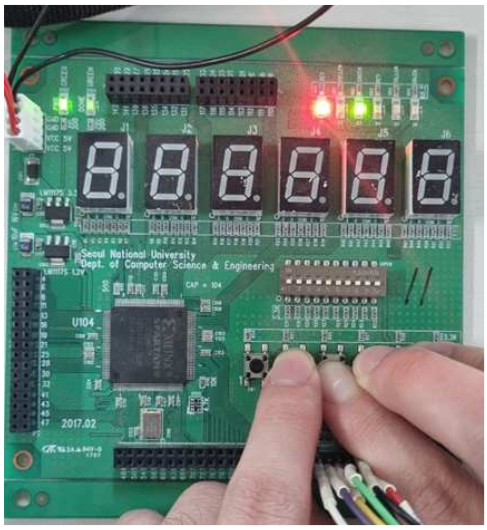
(22) a1 = 0, b1 = 0, a0 = 1, b0 = 1, c0 = 1인 경우



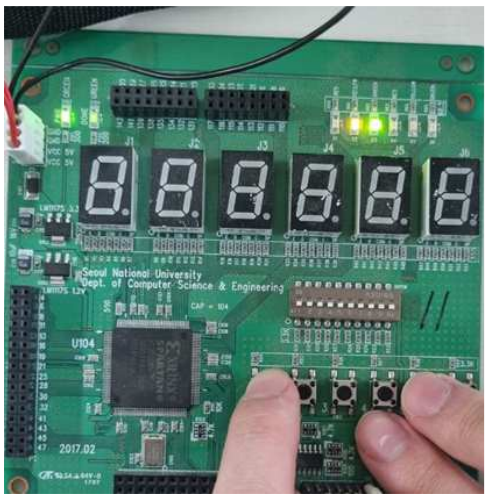
(23) a1 = 0, b1 = 1, a0 = 1, b0 = 0, c0 = 1인 경우



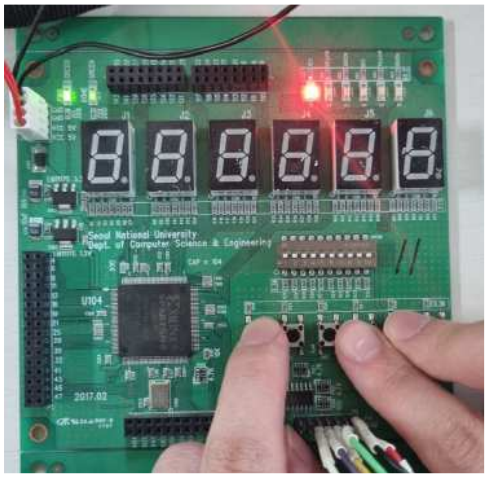
(24) a1 = 0, b1 = 1, a0 = 1, b0 = 1, c0 = 1인 경우



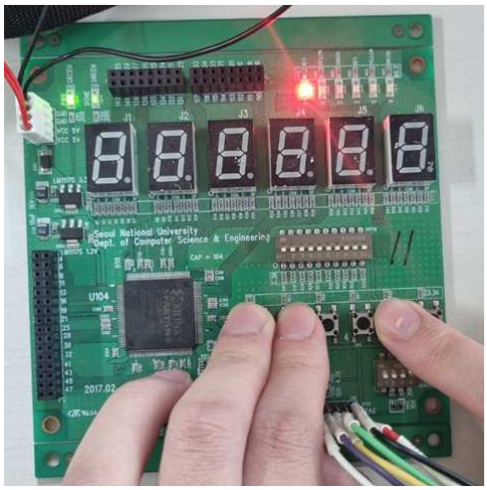
(25) a1 = 1, b1 = 0, a0 = 0, b0 = 0, c0 = 1인 경우



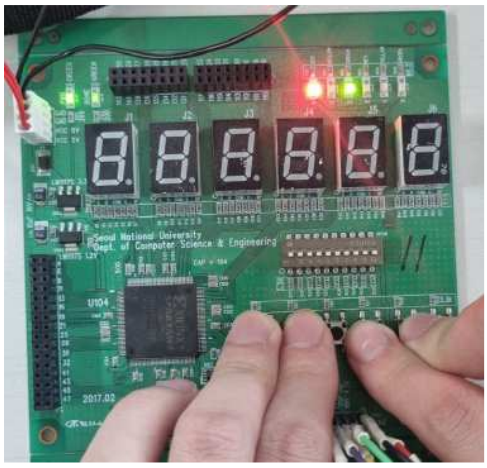
(26) a1 = 1, b1 = 0, a0 = 0, b0 = 1, c0 = 1인 경우



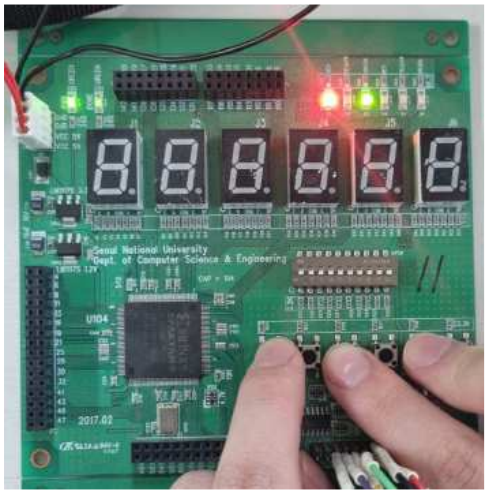
(27) a1 = 1, b1 = 1, a0 = 0, b0 = 0, c0 = 1인 경우



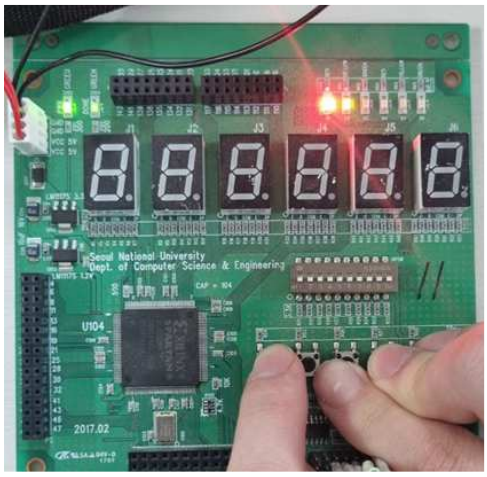
(28) a1 = 1, b1 = 1, a0 = 0, b0 = 1, c0 = 1인 경우



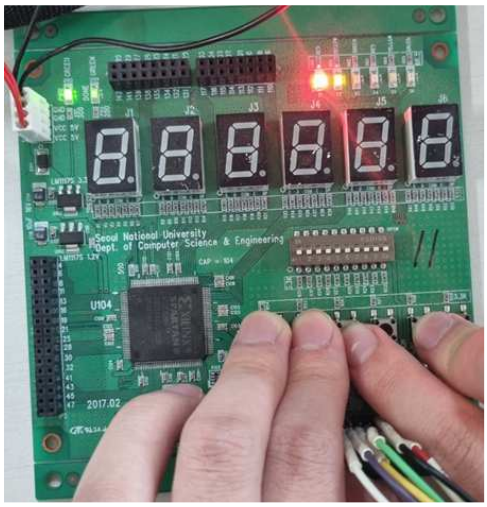
(29) a1 = 1, b1 = 0, a0 = 1, b0 = 0, c0 = 1인 경우



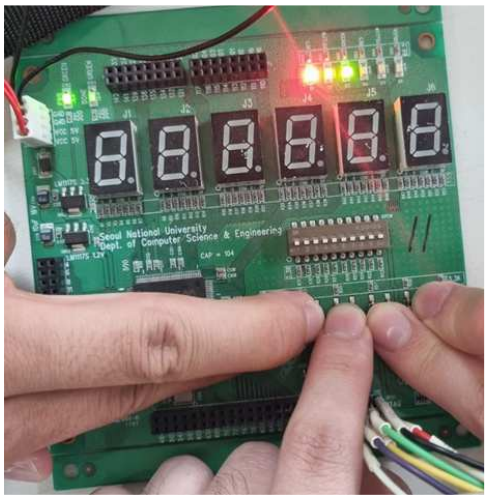
(30) a1 = 1, b1 = 0, a0 = 1, b0 = 1, c0 = 1인 경우



(31) a1 = 1, b1 = 1, a0 = 1, b0 = 0, c0 = 1인 경우



(32) a1 = 1, b1 = 1, a0 = 1, b0 = 1, c0 = 1인 경우



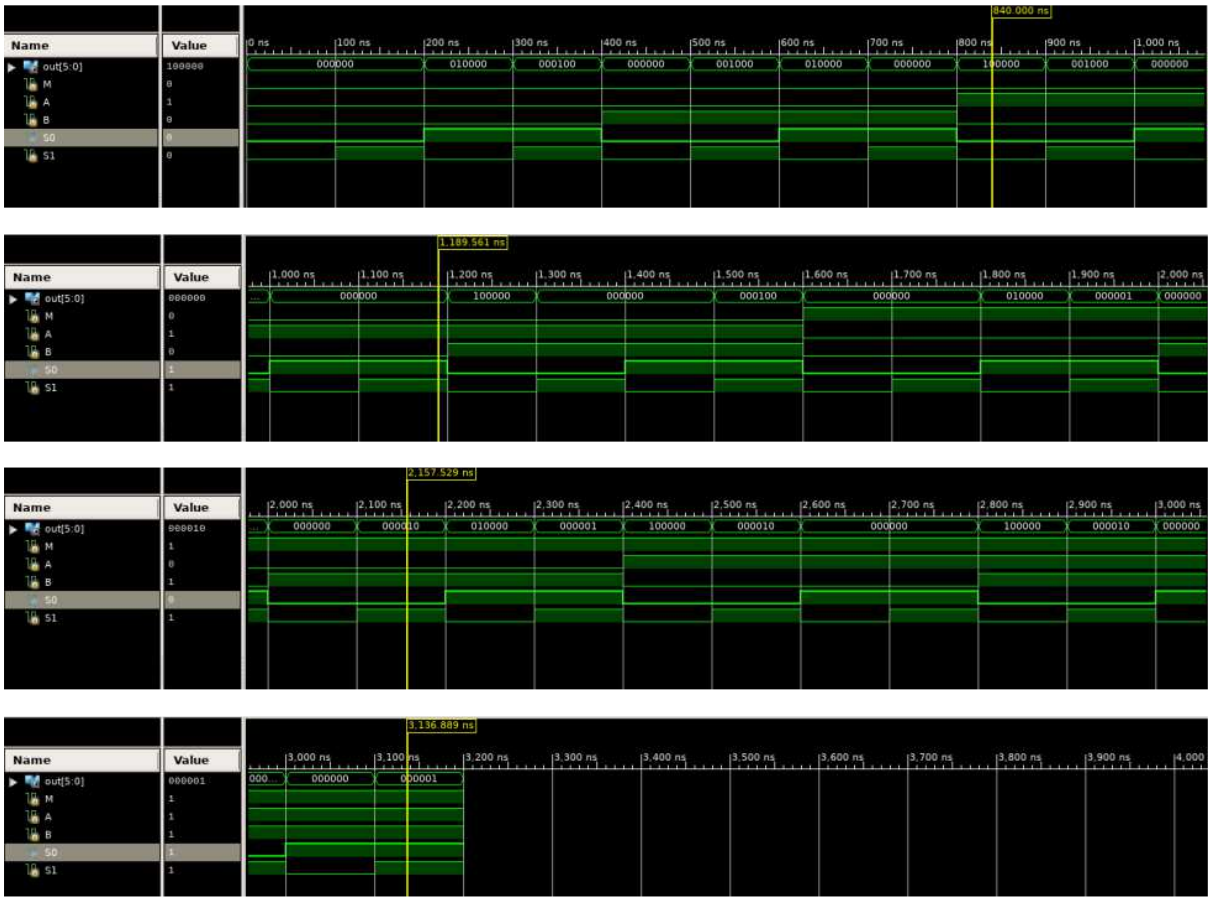
2. Homework part

Follow specification(a), Implement in Verilog. (M1, S1, S0) are control inputs, (Ai, Bi) are data inputs, and (Fi) is an output. Use the Dip switch for the inputs and simulate all possible outputs using Verilog test bench. Program on FPGA board and test all possible outputs.

M	S1	S0	Function	Comment	led
0	0	0	$F_i = A_i$	load	P87
0	0	1	$F_i = \sim A_i$	complement	P88
0	1	0	$F_i = A_i \oplus B_i$	xor	P90
0	1	1	$F_i = \sim(A_i \oplus B_i)$	xnor	P91
1	0	0	$F_i = A_i$	load	P87
1	0	1	$F_i = \sim A_i$	complement	P88
1	1	0	$F_i = A_i + B_i$	Add (ignore carry)	P92
1	1	1	$F_i = \sim A_i + B_i$	Complement and add (ignore carry)	P93

(a) specification

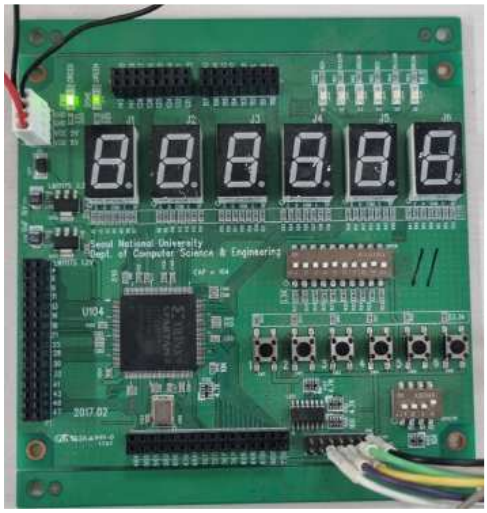
위의 specification 이미지를 참고하여 Structural Description 방법으로 one bit adder를 구현하였다. 관련 Verilog 파일은 'alu_onebit_structural.v' 이다. 또한, 'alu_onebit_test.v' 파일을 생성하여 앞에서 구현한 1bit ALU의 동작을 5개의 input(A, B, M, S0, S1) 조합에 따른 모든 경우의 수에 대해 검증하는 코드를 작성하였다. 1 bit ALU의 시뮬레이션 결과는 아래와 같다.



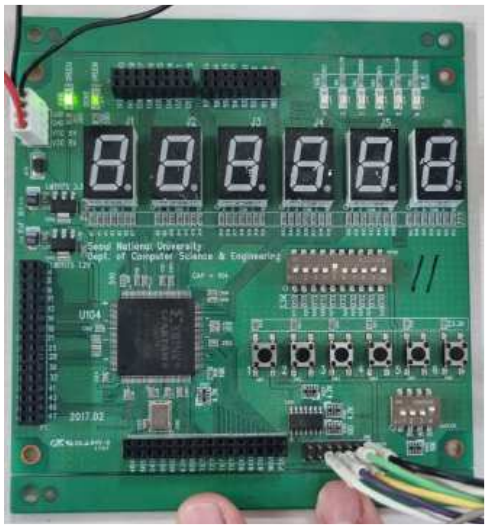
구현한 1bit ALU를 FPGA board에 program한 결과는 아래와 같다. 이때, 총 5개의 Dip Switch를 사용하였는데, 왼쪽부터 순서대로 A, B, M, S0, S1을 나타낸다. 또한, 총 6개의 LED를 사용하였는데, 왼쪽부터 순서대로 앞의 specification에서 명시된 P87, P88, P90, P91, P92, P93에 해당하는 led이

다. FPGA board의 동작 결과가 위의 시뮬레이션 결과와 정확히 일치하는 것을 확인할 수 있다.

(1) A = 0, B = 0, M = 0, S0 = 0, S1 = 0인 경우

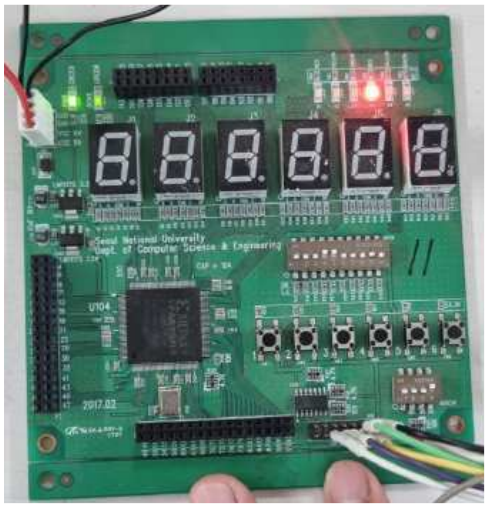


(2) A = 0, B = 0, M = 0, S0 = 0, S1 = 1인 경우

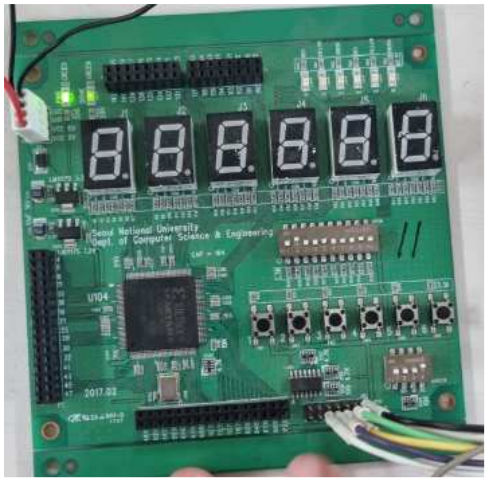


(3) A = 0, B = 0, M = 0, S0 = 1, S1 = 0인 경우

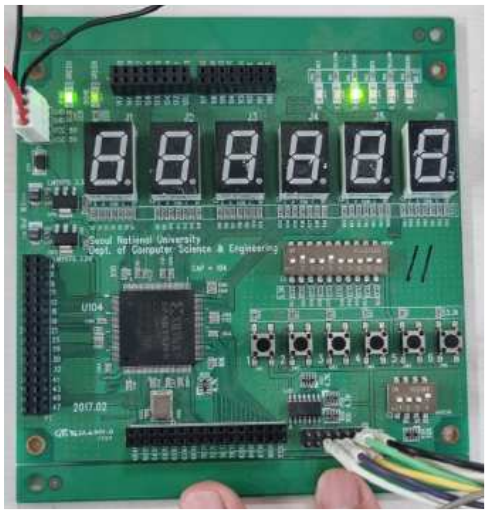
(4) A = 0, B = 0, M = 0, S0 = 1, S1 = 1인 경우



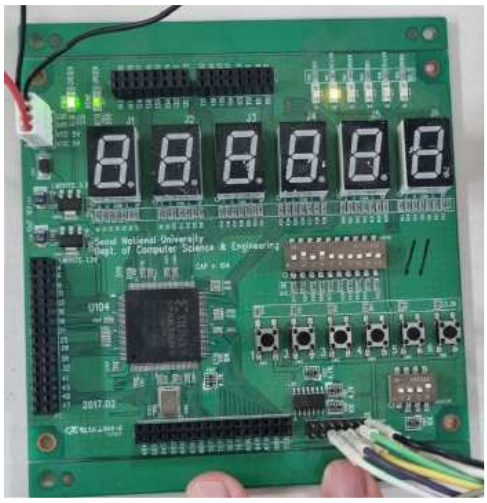
(5) A = 0, B = 1, M = 0, S0 = 0, S1 = 0인 경우



(6) A = 0, B = 1, M = 0, S0 = 0, S1 = 1인 경우



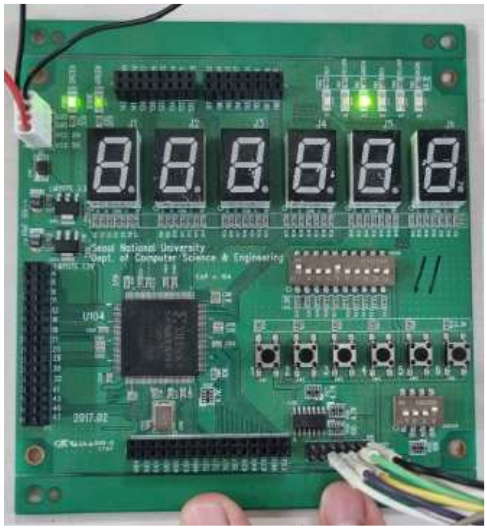
(7) A = 0, B = 1, M = 0, S0 = 1, S1 = 0인 경우



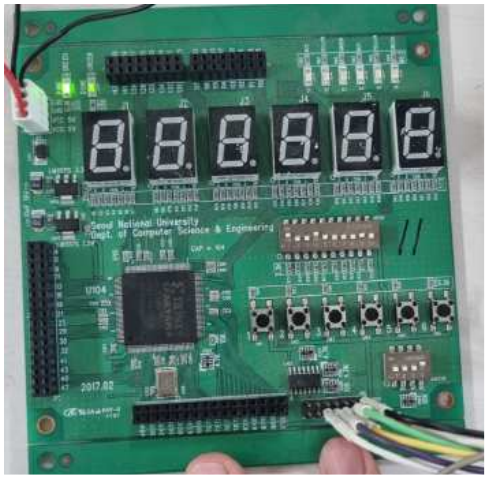
(8) A = 0, B = 1, M = 0, S0 = 1, S1 = 1인 경우

(9) A = 1, B = 0, M = 0, S0 = 0, S1 = 0인 경우

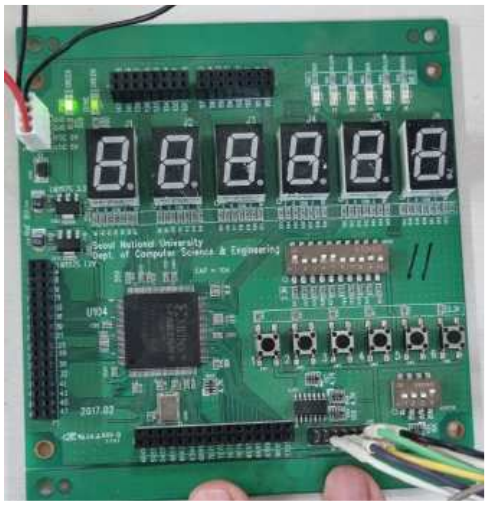
(10) A = 1, B = 0, M = 0, S0 = 0, S1 = 1인 경우



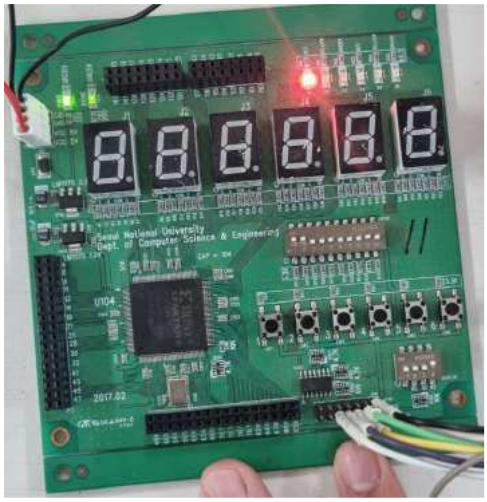
(11) A = 1, B = 0, M = 0, S0 = 1, S1 = 0인 경우



(12) A = 1, B = 0, M = 0, S0 = 1, S1 = 1인 경우



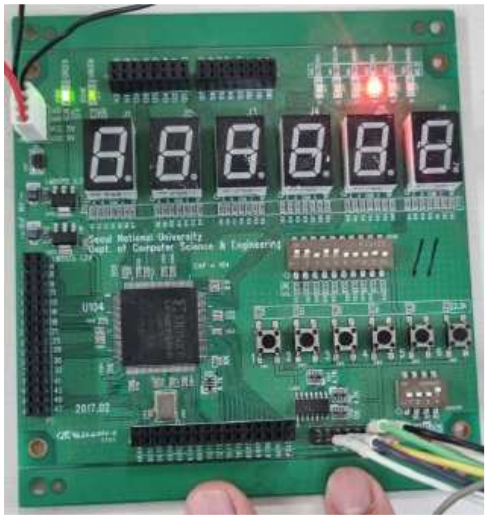
(13) A = 1, B = 1, M = 0, S0 = 0, S1 = 0인 경우



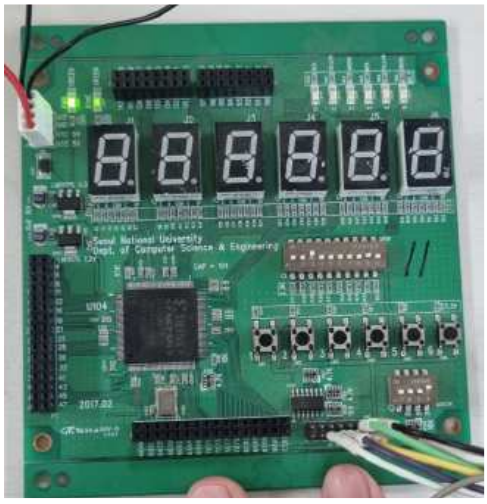
(14) A = 1, B = 1, M = 0, S0 = 0, S1 = 1인 경우

(15) A = 1, B = 1, M = 0, S0 = 1, S1 = 0인 경우

(16) A = 1, B = 1, M = 0, S0 = 1, S1 = 1인 경우

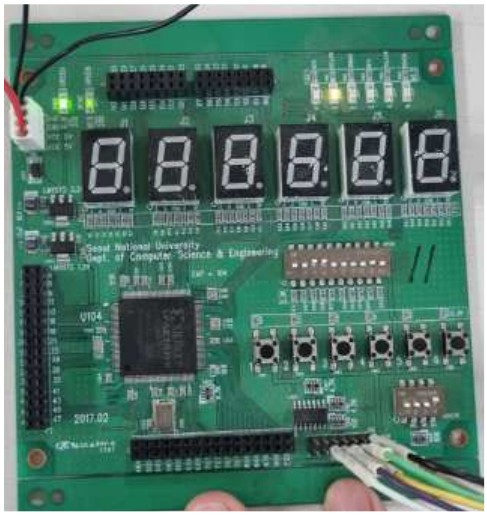


(17) A = 0, B = 0, M = 1, S0 = 0, S1 = 0인 경우

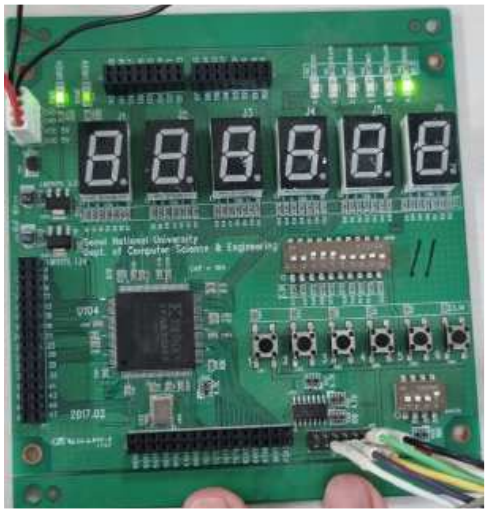


(18) A = 0, B = 0, M = 1, S0 = 0, S1 = 1인 경우

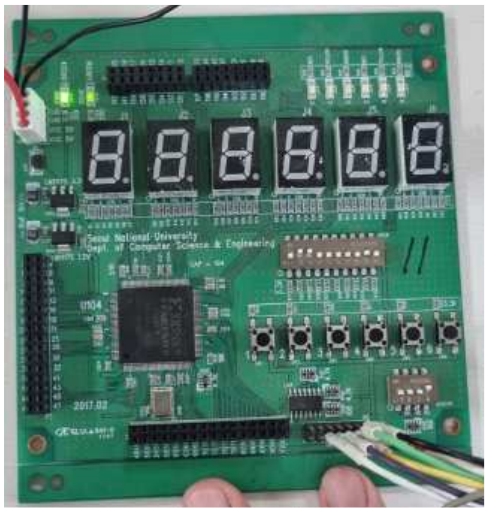
(19) A = 0, B = 0, M = 1, S0 = 1, S1 = 0인 경우



(20) A = 0, B = 0, M = 1, S0 = 1, S1 = 1인 경우

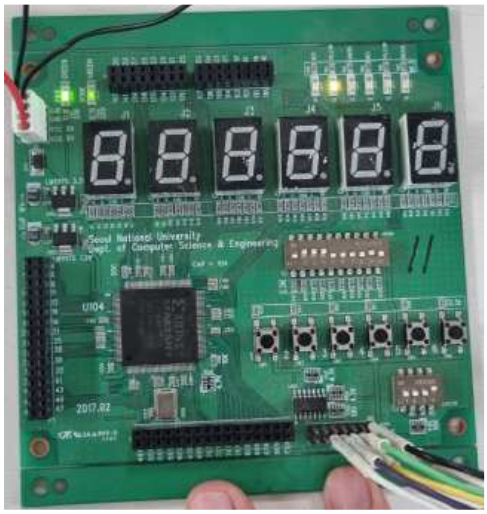


(21) A = 0, B = 1, M = 1, S0 = 0, S1 = 0인 경우

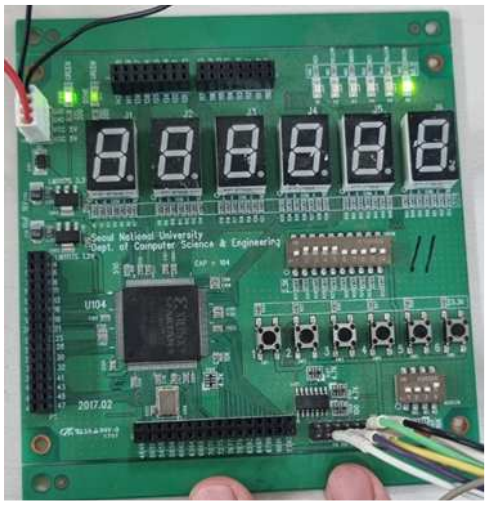


(22) A = 0, B = 1, M = 1, S0 = 0, S1 = 1인 경우

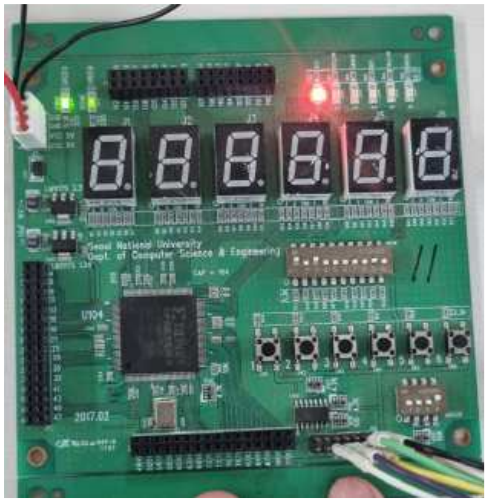
(23) A = 0, B = 1, M = 1, S0 = 1, S1 = 0인 경우



(24) A = 0, B = 1, M = 1, S0 = 1, S1 = 1인 경우



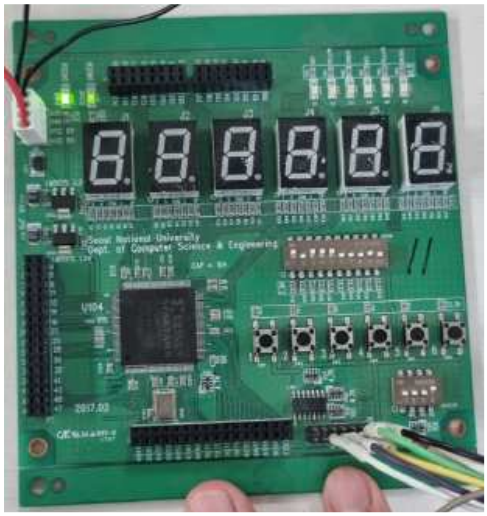
(25) A = 1, B = 0, M = 1, S0 = 0, S1 = 0인 경우



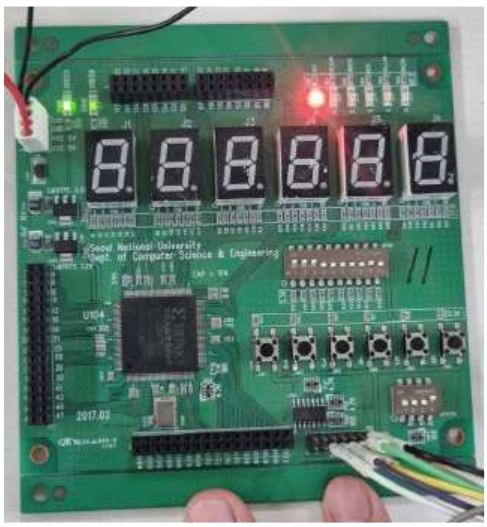
(26) A = 1, B = 0, M = 1, S0 = 0, S1 = 1인 경우

(27) A = 1, B = 0, M = 1, S0 = 1, S1 = 0인 경우

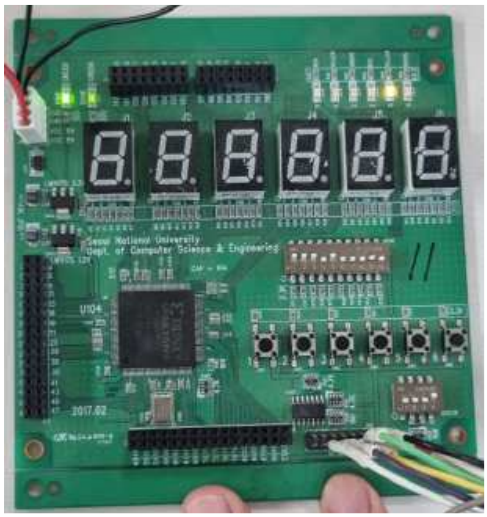
(28) A = 1, B = 0, M = 1, S0 = 1, S1 = 1인 경우



(29) A = 1, B = 1, M = 1, S0 = 0, S1 = 0인 경우



(30) A = 1, B = 1, M = 1, S0 = 0, S1 = 1인 경우



(31) A = 1, B = 1, M = 1, S0 = 1, S1 = 0인 경우

(32) A = 1, B = 1, M = 1, S0 = 1, S1 = 1인 경우