Project. 4층 엘리베이터

20200516 김규빈

1. 구현한 상태전이표, 무어 혹은 밀리 머신 도식, 그리고 상세한 설명

- 엘리베이터 상태 전이도

U := $u_1u_2u_3u_4$ ↑ 방향 버튼 ex) 1110 이면 1,2,3층 의 ↑ 버튼이 켜진 상태

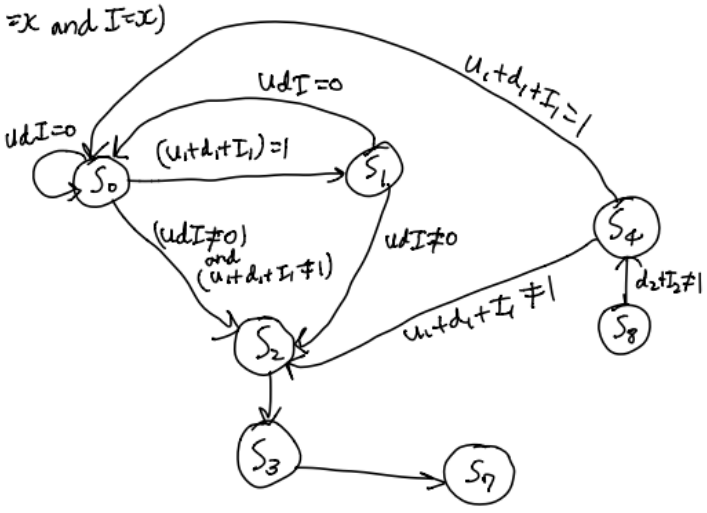
d := $d_1d_2d_3d_4$ ↓ 방향 버튼 ex) 0111 이면 2,3,4층 의 ↓ 버튼이 켜진 상태

I := $I_1I_2I_3I_4$ 내부 버튼 ex) 1010 이면 엘리베이터 내부의 1,3층 버튼이 켜진 상태

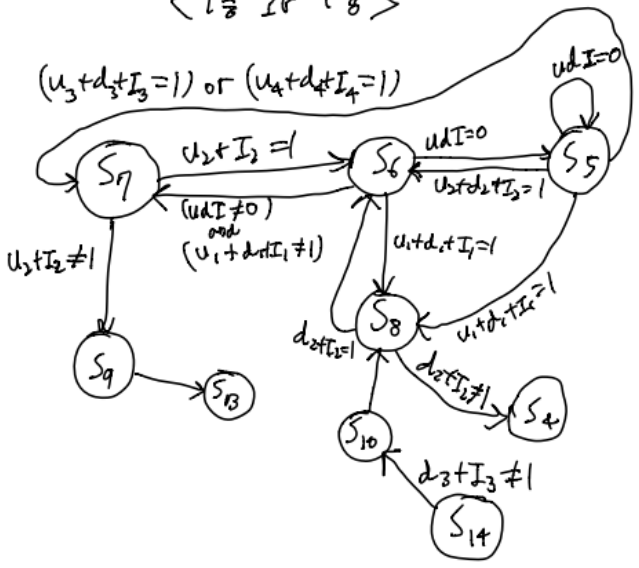
(udI=x) := (u=x and d=x and I=x)

(udI≠x) := ¬(udI=x)

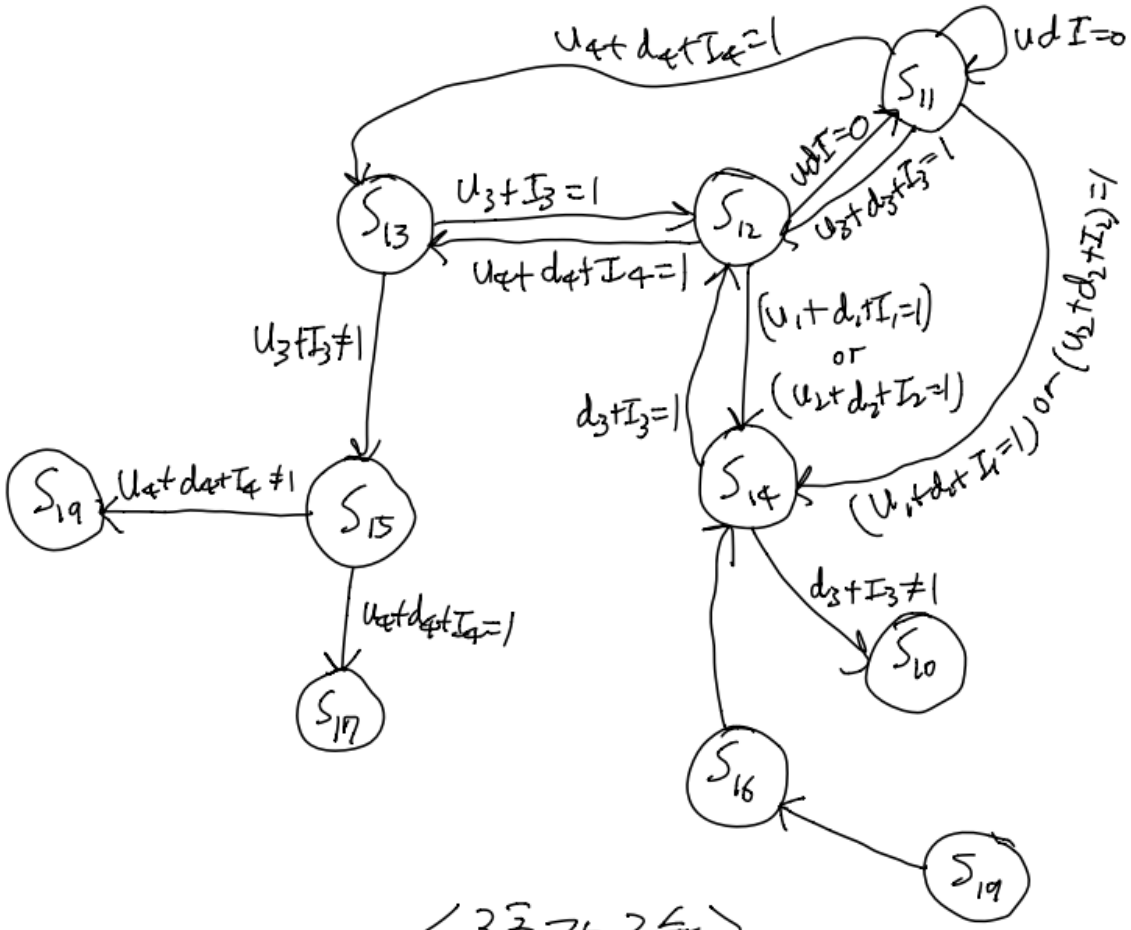
위치	방향	묻
S_0	1층	- 0
S_1	1층	- 1
S_2	1층	↑ 0
S_3	1층	↑ 0
S_4	1층	↓ 0
S_5	2층	- 0
S_6	2층	- 1
S_7	2층	↑ 0
S_8	2층	↓ 0
S_9	2층	↑ 0
S_{10}	2층	↓ 0
S_{11}	3층	- 0
S_{12}	3층	- 1
S_{13}	3층	↑ 0
S_{14}	3층	↓ 0
S_{15}	3층	↑ 0
S_{16}	3층	↓ 0
S_{17}	4층	- 0
S_{18}	4층	- 1
S_{19}	4층	↓ 0



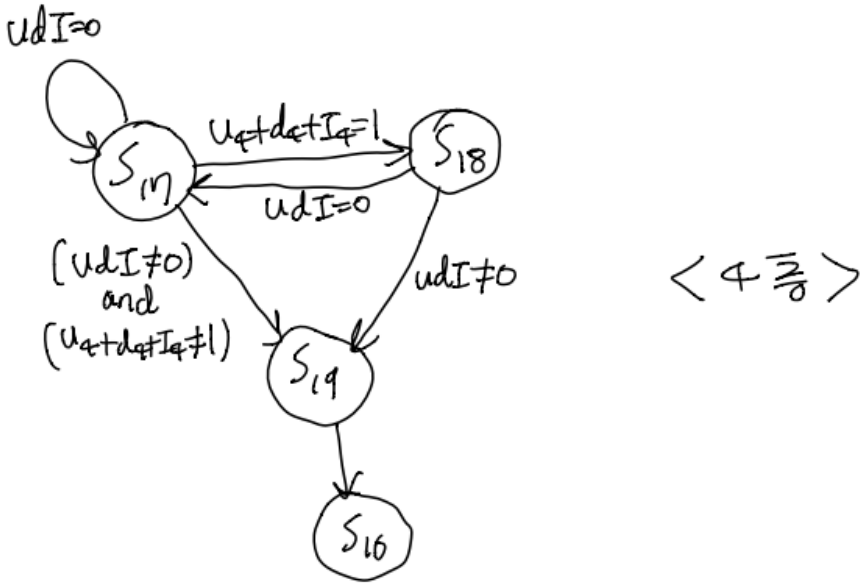
<1층 과 1층>



<2층과 2층>

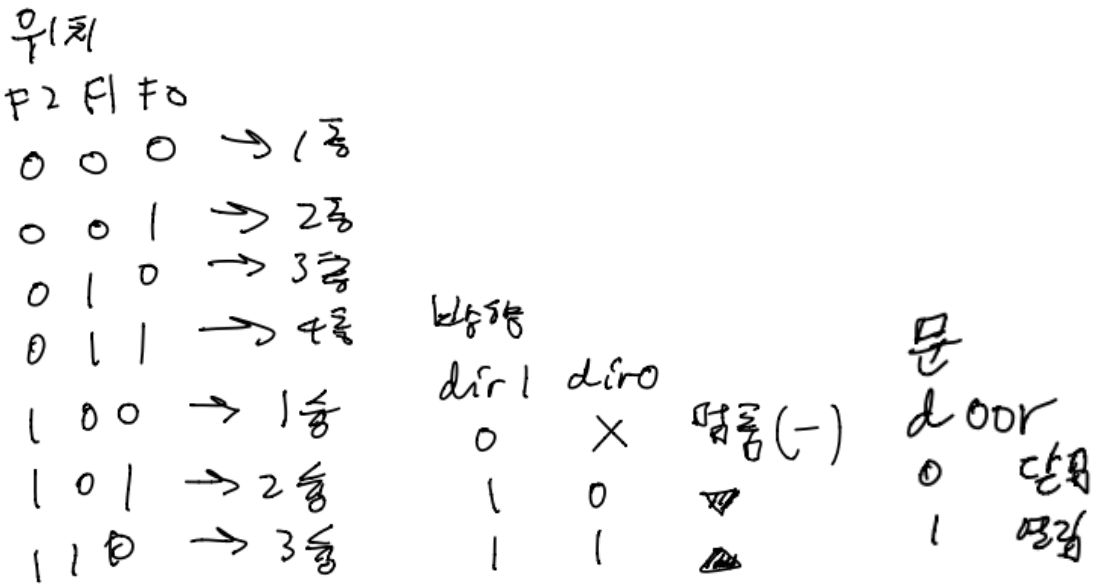


<3층과 3층>



<4층>

위와 같이 무어머신의 상태전이도를 먼저 그렸다. 그리고 state를 정의하는 위치, 방향, 문을 다음과 같이 설정했다.



따라서 u, d, I는 이 무어머신의 input을 의미하고, 위치, 방향, 문은 state를 나타낸다. 그리고 output은 6비트로 설정하여 각 비트가 순서대로 F2, F1, F0, dir1, dir0, door이 되도록 했다.

나는 D flip flop을 사용하여 이 머신을 구현하기로 했다. D FF의 input을 D_in이라고 하면, D_in은 next state가 1이면 1, next state가 0이면 0이다. 상태전이표 전체를 그리는 것은 state의 개수가 많고, input과 state의 비트 수도 크므로 상태전이표 전체를 그리는 대신 D_in이 1이되는 state와 input의 조건을 찾아서 기록했다. (present->next : condition 방식으로 기록, condition이 x라는 것은 어떤 input이든 상관없다는 뜻)

F2는 슝이 next state일때 1이되므로 다음과 같다.

S2->S3 : x

S8->S4 : d2+I2 != 1

S7->S9 : u2+I2 != 1

S14->S10 : d3+I3 != 1

S13->S15 : u3+I3 != 1

S19->S16 : x

F1은 3층, 4층, 3슝이 next state일때 1이되므로 다음과 같다.

present state가 4층, 3슝인 경우는 항상 가능

present state가 3층인 경우는 S14->S10을 제외하고 모두 가능

present state가 S9(2슝up)인 경우도 가능

F0은 2층, 4층, 2슝이 next state일때 1이되므로 다음과 같다.

present state가 S3(1슝up)인 경우도 가능

present state가 2층인 경우는 S8->S4를 제외하고 모두 가능

present state가 S10(2슝down)인 경우도 가능

S14->S10 : d3+I3 != 1

present state가 S15(3슝up)인 경우도 가능

present state가 4층인 경우는 S19를 제외하고 모두 가능

dir1은 0이 되는 경우가 개수가 훨씬 적으므로 0이 되는 경우들을 찾은 후 코드를 작성할 때는 not(A or B or C)와 같은 방식을 사용하여 1이되는 경우를 얻었다. dir1은 멈춤(-)이 next state일 경우 0이 되므로 다음과 같다.

S0->S0 : udI = 0 S0->S1 : u1+d1+I1=1

S1->S0 : udI = 0

S4->S0 : u1+d1+I1=1

S5->S5 : udI = 0 S5->S6 : u2+d2+I2=1

S6->S5 : udI = 0

S7->S6 : u2+I2 = 1

S8->S6 : d2+I2 = 1

S11->S11 : udI = 0 S11->S12 : u3+d3+I3 = 1

S12->S11 : udI = 0

S13->S12 : u3+l3 = 1

S14->S12 : d3+l3 = 1

S15->S17 : u4+d4+l4 = 1

S17->S17 : udl = 0 S17->S18 : u4+d4+l4 = 1

S18->S17 : udl = 0

dir0은 up이 next state일때 1이되므로 다음과 같다.

S0->S2 : !(udl=0), u1+d1+l1 != 1

S1->S2 : !(udl=0)

S4->S2 : u1+d1+l1 != 1

S2->S3 : x

S3->S7 : x

S5->S7 : (u3+d3+l3 = 1) or (u4+d4+l4 = 1)

S6->S7 : !(udl=0), u1+d1+l1 != 1

S7->S9 : u2+l2 != 1

S9->S13 : x

S11->S13 : u4+d4+l4 = 1

S12->S13 : u4+d4+l4 = 1

S13->S15 : u3+l3 != 1

door는 S1, S6, S12, S18 중 하나가 next state일때 1이되므로 다음과 같다.

S0->S1 : u1+d1+l1 = 1

S5->S6 : u2+d2+l2 = 1

S7->S6 : u2+l2 = 1

S8->S6 : d2+l2 = 1

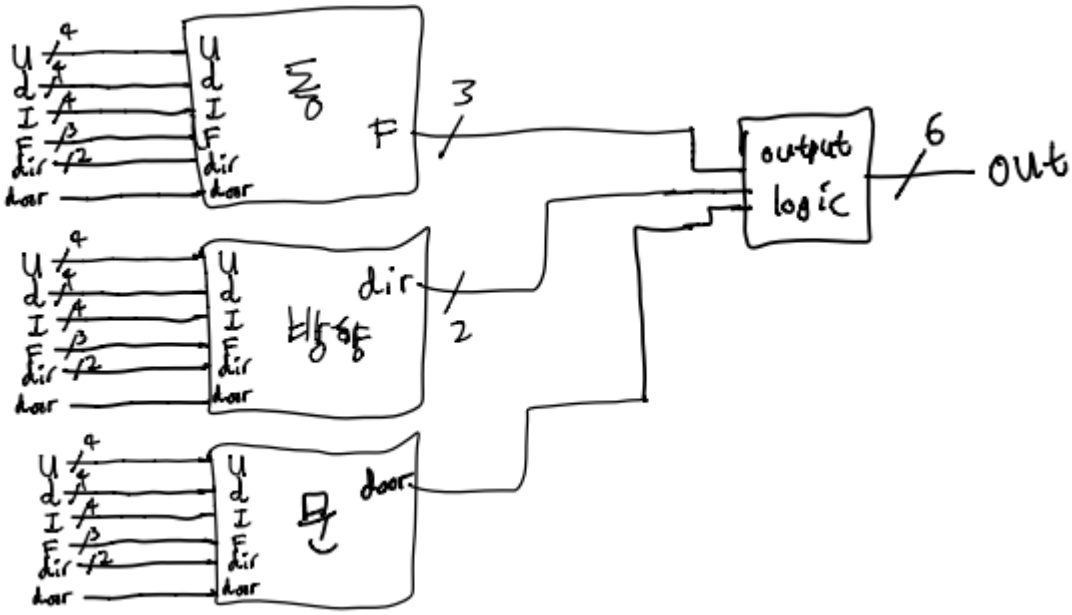
S11->S12 : u3+d3+l3 = 1

S13->S12 : u3+I3 = 1

S14->S12 : d3+I3 = 1

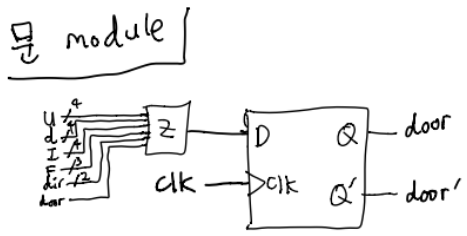
S17->S18 : u4+d4+I4 = 1

이제 이를 통해 각 state value에 들어가는 D FF의 input 값을 얻어낼 수 있다.

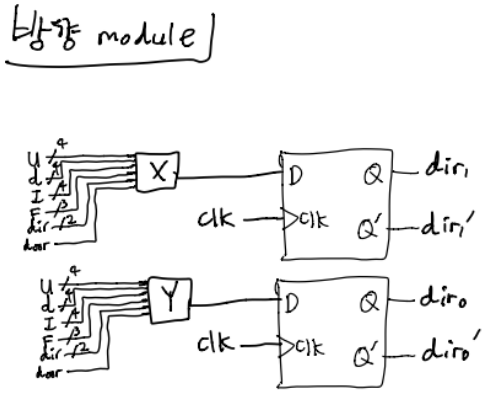


무어머신의 도식은 다음과 같다. '층', '방향', '문' 모듈이 input인 u, d, I 그리고 현재 state를 나타내는 F, dir, door를 통해 next state를 얻는다. 그리고 output logic은 out이 state를 나타내도록 되어 있다. 즉, out[5] = F[2], out[4] = F[1], out[3] = F[0], out[2] = dir[1], out[1] = dir[0], out[0] = door를 만족하게 연결되어 있다. 즉, 등호의 좌변과 우변이 연결되어 있다.

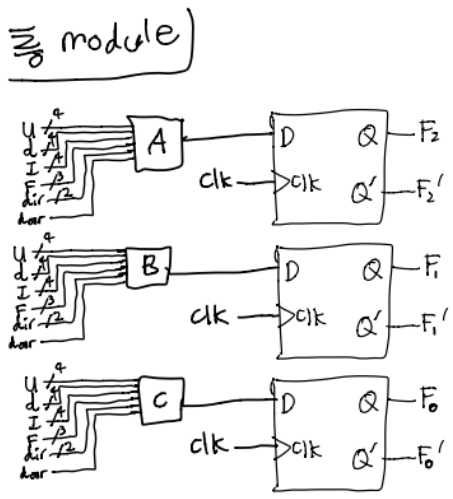
2. 각 기능을 모듈로 나타낸 간단한 회로도



'문'의 회로도는 다음과 같다. Z는 D_in을 얻는 로직을 의미한다. 위에서 찾은 door가 1이되는 조건들을 or로 묶어주는 부분이 Z에 들어간다.



'방향'의 회로도는 다음과 같다. X, Y는 dir1과 dir0의 D_in을 얻는 로직을 의미한다. 위에서 찾은 dir1, dir0이 1이 되는 조건들을 or로 묶어주는 부분이 X, Y에 들어간다.



'층'의 회로도는 다음과 같다. A, B, C는 F2, F1, F0의 D_in을 얻는 로직을 의미한다. 위에서 찾은 F2, F1, F0이 1이 되는 조건들을 or로 묶어주는 부분이 A, B, C에 들어간다.

3. 논의

처음에는 상당히 많은 state, 큰 비트의 state와 input으로 인해 어떻게 시작해야할지 막막했다. 하지만 차근차근 상태전이표를 만들었고, 최종적으로 완성할 수 있었다. 엘리베이터는 엄청나게 복잡한 장치가 아님에도 이정도로 복잡하게 회로도가 만들어진다는 점에서, K-map이나 Quine-McCluskey algorithm과 같은 단순화 과정이 얼마나 중요한지 다시한번 느낄 수 있었다.