Specification:

A 4-digit keypad is to be made with the access code 2337. The inputs given are K[0]-K[8] representing buttons 1-9, K[9] for 0, K[10] for ENTER, K[11] for CLEAR, and reset and clk inputs, and outputs are *lock* and *alarm*. If the sequence of 2,3,3,7 is pressed, followed by ENTER, with no other number inputs before or after, *lock* should be LOW for 3 clock cycles, after which the keypad should be reset. If ENTER is pressed after at least one digit has been input, and the inputs are not of the sequence 2,3,3,7, the keypad should be cleared, and the input should be regarded as incorrect. On the third incorrect input, *alarm* should be set to HIGH until reset is input. A correct input should reset the alarm counter. *reset* should set the keypad to its initial state from any point. *reset* should be active-high, and all other inputs should be active-high. The output *lock* should be active-low and *alarm* should be active high.

Assumptions:

1. There will only be clk and one other input active at any given time.
2. The keypad should only register a sequence as correct if the combination 2,3,3,7 are the only digits input after a CLEAR, so 7,3,4,2,3,3,7 or 2,2,3,3,7 would count as incorrect, while 2,CLEAR,2,3,3,7 would count as correct.
3. Pressing Enter before any digits are input will count as an incorrect input.
4. Entering the correct code should reset the entire keypad after unlocking, meaning that the number of incorrect attempts should be set to zero.
5. There will be no additional inputs while the door is unlocked.
6. All inputs will only be high for one clock cycle, and will not change at clock edges.

State diagrams:

The state diagram for the keypad is broken up into four separate Moore state diagrams, for the keypad logic, alarm logic, and the unlock logic. Reset input is done through flip flop reset pin, so is not included in state diagrams.

Global Logic:

Global logic is purely combinational:

Door should unlock when code is correct and enter is pressed, and alarm is not on, so

K[10]CORalarm' = ULK

A failed attempt should be registered when code is incorrect and enter is pressed, so

K[10]COR' = FL

For Keypad logic:

Since the only digits in the combination are 2,3, and 7, all other digits, clear, and enter will reset the sequence, so we can simplify using K[3] + K[4] + K[5] + K[7] + K[8] + K[9] + K[10] + K[11] = DX

Since only one input is active at a time, these inputs can further be encoded to reduce the complexity of the state diagram using a 4-2 decoder as shown below.

Inputs				Outputs	
I[3]=K[1]	I[2]=K[2]	I[1]=K[6]	I[0]=DX	Y[1]	Y[0]
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

This introduces the issue that Y does not describe the scenario of no inputs, however, since this state diagram only changes states when there is an input, if clock = clk(DX+Y[0]+Y[1]), the state diagram can be fully reduced to two inputs.

Keypad Logic:

Inputs:

Y[1],Y[0],clock,reset

Outputs:

COR (Correct code is entered)

State descriptions:

SEQ0:

Initial state of keypad. No correct digits input.

- Input 11(2) will change to SEQ1 state (One correct input)
- Inputs 00,10,01 (DX,3,7) will not change state (Incorrect input)
- Output COR = 0 (Code is incorrect)

SEQ1:

One correct digit input.

- Input 10(3) will change to SEQ2 state (Two correct inputs)
- Inputs 00,01 (DX,7) will reset to SEQ0 (Incorrect input)
- Input 11 (2) will remain in SEQ1 state (Incorrect second input but correct first input)
- Output COR = 0 (Code is incorrect)

SEQ2:

Two correct digits input.

- Input 10(3) will change to SEQ3 state (Three correct inputs)
- Inputs 00,01 (DX,7) will reset to SEQ0 (Incorrect input)
- Input 11 (2) will set to SEQ1 state (Incorrect third input but correct first input)
- Output COR = 0 (Code is incorrect)

SEQ3:

Three correct digits input.

- Input 01(7) will change to SEQ4 state (Four correct inputs)
- Inputs 00,10 (DX,3) will reset to SEQ0 (Incorrect input)
- Input 11 (2) will set to SEQ1 state (Incorrect fourth input but correct first input)
- Output COR = 0 (Code is incorrect)

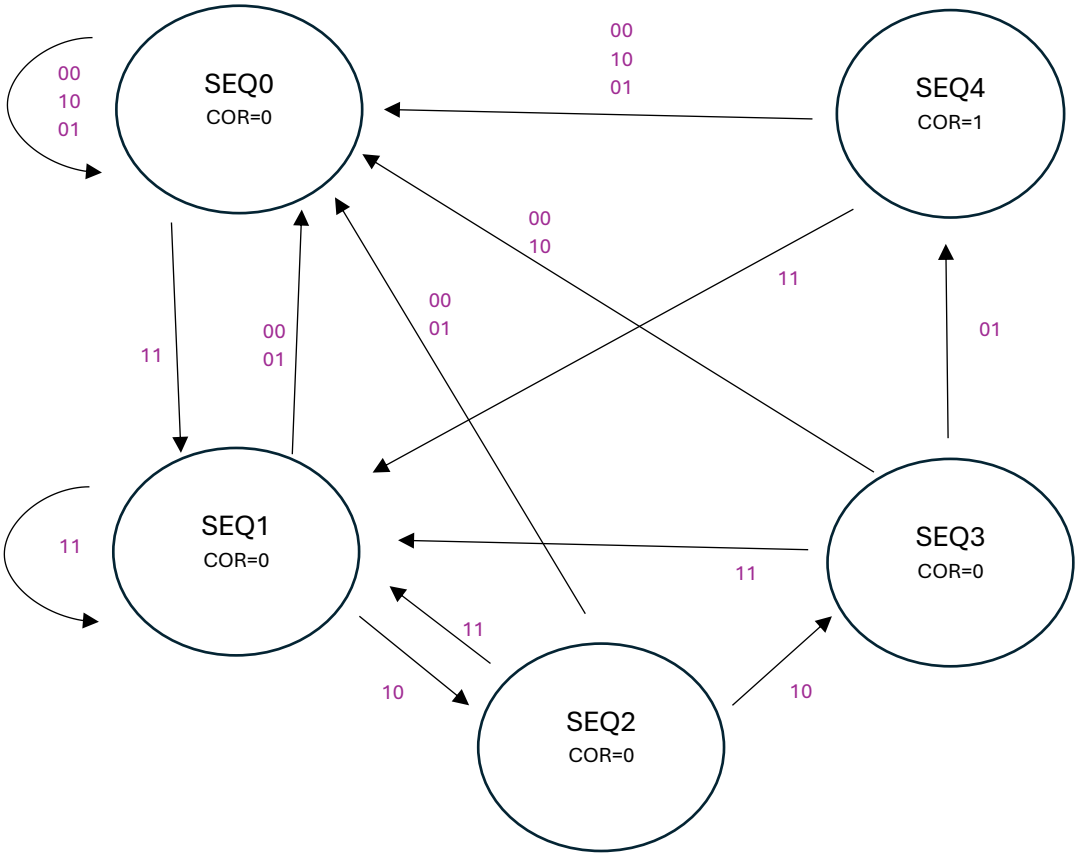
SEQ4:

Four correct digits input.

- Input 01(7) will change to SEQ4 state (Four correct inputs)
- Inputs 00,10,01 (DX,3,7) will reset to SEQ0 (Incorrect input)
- Input 11 (2) will set to SEQ1 state (Incorrect input but correct first input)
- Output COR = 1 (Code is correct)

Keypad Logic State Diagram:

Inputs Y[1],Y[0]



State Table:

Present State	Inputs		Next State	Output
	Y[1]	Y[0]		COR
SEQ0	0	0	SEQ0	0
SEQ0	0	1	SEQ0	0
SEQ0	1	0	SEQ0	0
SEQ0	1	1	SEQ1	0
SEQ1	0	0	SEQ0	0
SEQ1	0	1	SEQ0	0
SEQ1	1	0	SEQ2	0
SEQ1	1	1	SEQ1	0
SEQ2	0	0	SEQ0	0
SEQ2	0	1	SEQ0	0
SEQ2	1	0	SEQ3	0
SEQ2	1	1	SEQ1	0
SEQ3	0	0	SEQ0	0
SEQ3	0	1	SEQ4	0
SEQ3	1	0	SEQ0	0
SEQ3	1	1	SEQ1	0
SEQ4	0	0	SEQ0	1
SEQ4	0	1	SEQ0	1
SEQ4	1	0	SEQ0	1
SEQ4	1	1	SEQ1	1

State minimization:

Using implication chart

First step:

SEQ1	SEQ0=SEQ2			
SEQ2	SEQ0=SEQ3	SEQ2=SEQ3		
SEQ3	SEQ0=SEQ4	SEQ0=SEQ4 SEQ0=SEQ2	SEQ0=SEQ4 SEQ0=SEQ3	
SEQ4	X	X	X	X
	SEQ0	SEQ1	SEQ2	SEQ3

Second step:

SEQ1	SEQ0=SEQ2			
SEQ2	SEQ0=SEQ3	SEQ2=SEQ3		
SEQ3	X	X	X	
SEQ4	X	X	X	X
	SEQ0	SEQ1	SEQ2	SEQ3

Third step:

SEQ1	SEQ0=SEQ2			
SEQ2	X	X		
SEQ3	X	X	X	
SEQ4	X	X	X	X
	SEQ0	SEQ1	SEQ2	SEQ3

Fourth step:

SEQ1	X			
SEQ2	X	X		
SEQ3	X	X	X	
SEQ4	X	X	X	X
	SEQ0	SEQ1	SEQ2	SEQ3

So states cannot be reduced.

Alarm logic:

Inputs:

FL (From global logic), clk, reset

Output:

alarm

State descriptions:

FAIL0:

No failed attempts.

- Remains in state FAIL0 if FL=0 (No incorrect code)
- Changes to state FAIL1 if FL=1 (Incorrect code entered)
- Outputs alarm = 0 (Not three failed attempts)

FAIL1:

One failed attempt.

- Remains in state FAIL1 if FL=0 (No incorrect code)
- Changes to state FAIL2 if FL=1 (Second incorrect code entered)
- Outputs alarm = 0 (Not three failed attempts)

FAIL2:

Two failed attempts.

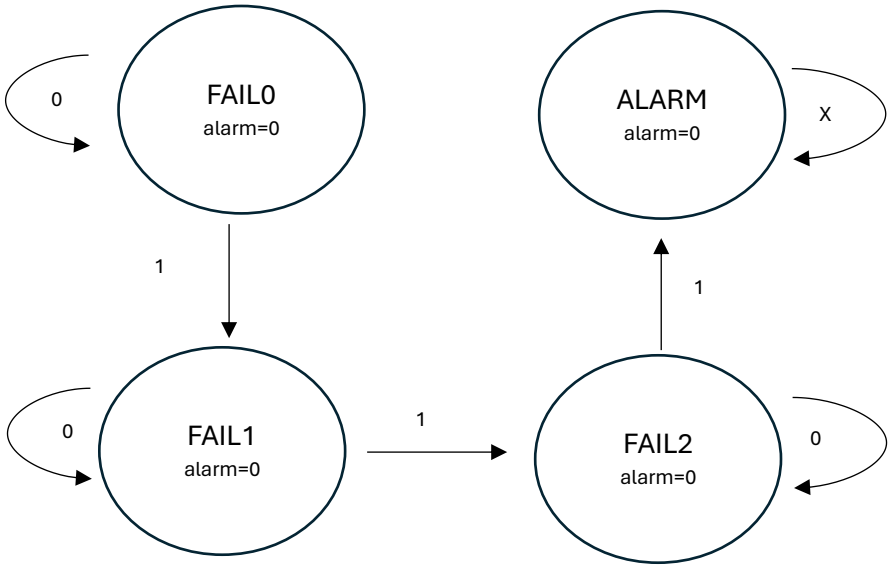
- Remains in state FAIL2 if FL=0 (No incorrect code)
- Changes to state ALARM if FL=1 (Third incorrect code entered, so alarm should sound)
- Outputs alarm = 0 (Not three failed attempts)

ALARM:

Three failed attempts. Alarm sounds and keypad needs to be reset.

- Remains in ALARM state regardless of input (Needs reset to change)
- Outputs alarm = 1 (Three consecutive failed attempts, so alarm should sound)

Alarm Logic State Diagram



State table:

Present State	Input	Next State	Output
	FL		alarm
FAIL0	0	FAIL0	0
FAIL0	1	FAIL1	0
FAIL1	0	FAIL1	0
FAIL1	1	FAIL2	0
FAIL2	0	FAIL2	0
FAIL2	1	ALARM	0
ALARM	0	ALARM	1
ALARM	1	ALARM	1

State minimization:

Using implication chart

First pass:

FAIL1	FAIL0=FAIL1 FAIL1=FAIL2		
FAIL2	FAIL0=FAIL2 FAIL1=ALARM	FAIL1=FAIL2 FAIL2=ALARM	
ALARM	X	X	X
	FAIL0	FAIL1	FAIL2

Second pass:

FAIL1	FAIL0=FAIL1 FAIL1=FAIL2		
FAIL2	X	X	
ALARM	X	X	X
	FAIL0	FAIL1	FAIL2

Third pass:

FAIL1	X		
FAIL2	X	X	
ALARM	X	X	X
	FAIL0	FAIL1	FAIL2

So the states cannot be reduced.

Unlock logic:

Inputs:

ULK (Unlock signal from global logic), clk, reset

Outputs:

lock

State descriptions:

LOCKED:

Door is locked, as correct code has not been entered.

- Remain in LOCKED state if ULK = 0 (No correct code entered)
- Change to OPEN0 state if ULK = 1 (Correct code entered)
- Output lock = 1 (Locked)

OPEN0:

Door is open, as correct code has been entered. Door is open in state OPEN0 for the first clock cycle of its opening.

- Change to OPEN1 regardless of input (1 clock cycle has passed)
- Output lock = 0 (Open)

OPEN1:

Door is open in state OPEN1 for the second clock cycle of its opening.

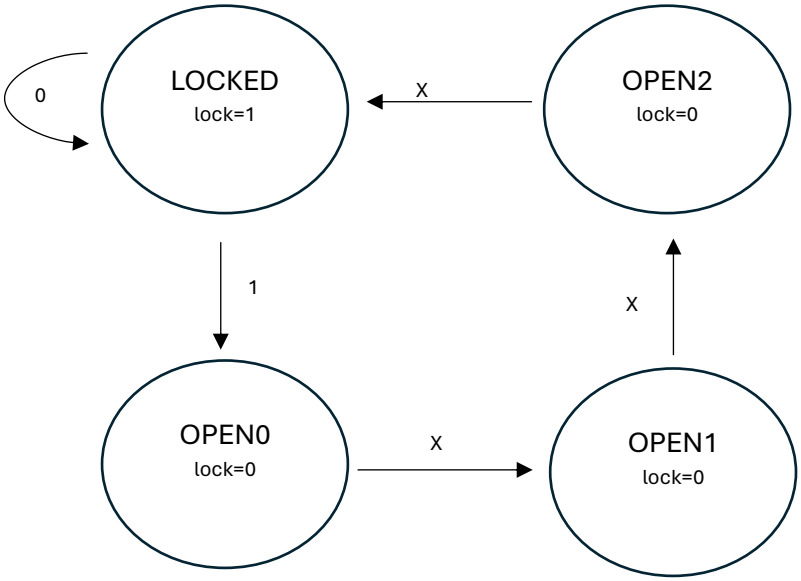
- Change to OPEN2 regardless of input (1 clock cycle has passed)
- Output lock = 0 (Open)

OPEN2:

Door is open in state OPEN2 for the third clock cycle of its opening.

- Change to LOCKED regardless of input (1 clock cycle has passed for 3 total, so door should lock)
- Output lock = 0 (Open)

Unlock logic State Diagram



State table:

Present State	Input	Next State	Outputs
	ULK		lock
LOCKED	0	LOCKED	1
LOCKED	1	OPEN0	1
OPEN0	0	OPEN1	0
OPEN0	1	OPEN1	0
OPEN1	0	OPEN2	0
OPEN1	1	OPEN2	0
OPEN2	0	LOCKED	0
OPEN2	1	LOCKED	0

State minimization:

Using implication chart

First pass:

OPEN0	X		
OPEN1	X	OPEN1=OPEN2	
OPEN2	X	X	X
	LOCKED	OPEN0	OPEN1

Second pass:

OPEN0	X		
OPEN1	X	X	
OPEN2	X	X	X
	LOCKED	OPEN0	OPEN1

So the states cannot be reduced.

State assignment:

Keypad logic:

5 states, so 3 bits A,B,C are used for state assignment. SEQ0 is initial state so it is 000.

SEQ0: ABC = 000
SEQ1: ABC = 001
SEQ2: ABC = 010
SEQ3: ABC = 011
SEQ4: ABC = 100

State table:

Minterm	Present State	Inputs		Next State	Output	Flip Flops											
	A,B,C	Y[1]	Y[0]	A,B,C	COR	Da	Db	Dc	Ja	Ka	Jb	Kb	Jc	Kc	Ta	Tb	Tc
0	000	0	0	000	0	0	0	0	0	X	0	X	0	X	0	0	0
1	000	0	1	000	0	0	0	0	0	X	0	X	0	X	0	0	0
2	000	1	0	000	0	0	0	0	0	X	0	X	0	X	0	0	0
3	000	1	1	001	0	0	0	1	0	X	0	X	1	X	0	0	1
4	001	0	0	000	0	0	0	0	0	X	0	X	X	1	0	0	1
5	001	0	1	000	0	0	0	0	0	X	0	X	X	1	0	0	1
6	001	1	0	010	0	0	1	0	0	X	1	X	X	1	0	1	1
7	001	1	1	001	0	0	0	1	0	X	0	X	X	0	0	0	0
8	010	0	0	000	0	0	0	0	0	X	X	1	0	X	0	1	0
9	010	0	1	000	0	0	0	0	0	X	X	1	0	X	0	1	0
10	010	1	0	011	0	0	1	1	0	X	X	0	1	X	0	0	1
11	010	1	1	001	0	0	0	1	0	X	X	1	1	X	0	1	1
12	011	0	0	000	0	0	0	0	0	X	X	1	X	1	0	1	1
13	011	0	1	100	0	1	0	0	1	X	X	1	X	1	1	1	1
14	011	1	0	000	0	0	0	0	0	X	X	1	X	1	0	1	1
15	011	1	1	001	0	0	0	1	0	X	X	1	X	0	0	1	0
16	100	0	0	000	1	0	0	0	X	1	0	X	0	X	1	0	0
17	100	0	1	000	1	0	0	0	X	1	0	X	0	X	1	0	0
18	100	1	0	000	1	0	0	0	X	1	0	X	0	X	1	0	0
19	100	1	1	001	1	0	0	1	X	1	0	X	1	X	1	0	1
20	101	0	0	XXX	X	X	X	X	X	X	X	X	X	X	X	X	X
21	101	0	1	XXX	X	X	X	X	X	X	X	X	X	X	X	X	X
22	101	1	0	XXX	X	X	X	X	X	X	X	X	X	X	X	X	X
23	101	1	1	XXX	X	X	X	X	X	X	X	X	X	X	X	X	X
24	110	0	0	XXX	X	X	X	X	X	X	X	X	X	X	X	X	X
25	110	0	1	XXX	X	X	X	X	X	X	X	X	X	X	X	X	X
26	110	1	0	XXX	X	X	X	X	X	X	X	X	X	X	X	X	X
27	110	1	1	XXX	X	X	X	X	X	X	X	X	X	X	X	X	X
28	111	0	0	XXX	X	X	X	X	X	X	X	X	X	X	X	X	X
29	111	0	1	XXX	X	X	X	X	X	X	X	X	X	X	X	X	X
30	111	1	0	XXX	X	X	X	X	X	X	X	X	X	X	X	X	X
31	111	1	1	XXX	X	X	X	X	X	X	X	X	X	X	X	X	X

Outputs:

COR		C,Y[1],Y[0]							
		000	001	011	010	110	111	101	100
A,B	00	0	0	0	0	0	0	0	0
	01	0	0	0	0	0	0	0	0
	11	X	X	X	X	X	X	X	X
	10	1	1	1	1	X	X	X	X

COR = A

T-Flip-Flop implementation:

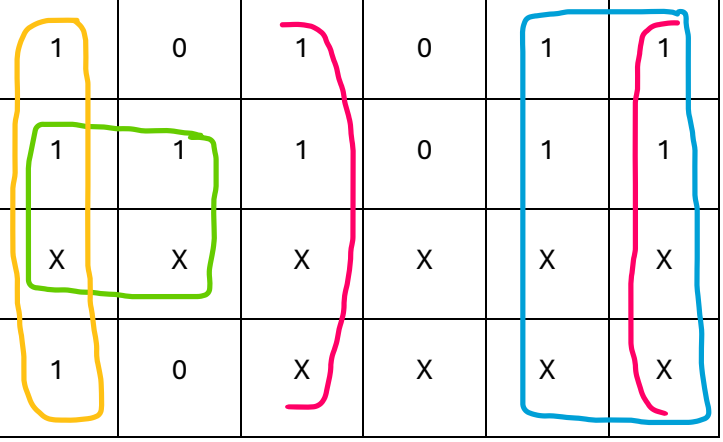
Ta		C,Y[1],Y[0]							
		000	001	011	010	110	111	101	100
A,B	00	0	0	0	0	0	0	0	0
	01	0	0	0	0	0	0	1	0
	11	X	X	X	X	X	X	X	X
	10	1	1	1	1	X	X	X	X

Ta = BCY[1]'Y[0] + A

Tb		C,Y[1],Y[0]							
		000	001	011	010	110	111	101	100
A,B	00	0	0	0	0	1	0	0	0
	01	1	1	1	0	1	1	1	1
	11	X	X	X	X	X	X	X	X
	10	0	0	0	0	X	X	X	X

Tb= CY[1]Y[0]' + BY[1]' + BY[0]

Tc		C,Y[1],Y[0]							
		000	001	011	010	110	111	101	100
A,B	00	0	0	1	0	1	0	1	1
	01	0	0	1	1	1	0	1	1
	11	X	X	X	X	X	X	X	X
	10	0	0	1	0	X	X	X	X



Tc = C'Y[1]Y[0] + CY[1]' + CY[0]' + BC'Y[1]

T-Flip-Flips:

Ta = BCY[1]'Y[0] + A
Tb= CY[1]Y[0]' + BY[1]' + BY[0]
Tc = C'Y[1]Y[0] + CY[1]' + CY[0]' + BC'Y[1]

JK-Flip-Flop implementation:

Ja		C,Y[1],Y[0]							
		000	001	011	010	110	111	101	100
A,B	00	0	0	0	0	0	0	0	0
	01	0	0	0	X	0	0	1	0
	11	X	X	X	X	X	X	X	X
	10	X	X	X	X	X	X	X	X

Ja = BCY[1]'Y[0]

Ka		C,Y[1],Y[0]							
		000	001	011	010	110	111	101	100
A,B	00	X	X	X	X	X	X	X	X
	01	X	X	X	X	X	X	X	X
	11	X	X	X	X	X	X	X	X
	10	1	1	1	1	X	X	X	X

Ka = 1

Jb		C,Y[1],Y[0]							
		000	001	011	010	110	111	101	100
A,B	00	0	0	0	0	1	0	0	0
	01	X	X	X	X	X	X	X	X
	11	X	X	X	X	X	X	X	X
	10	0	0	0	0	X	X	X	X

Jb = CY[1]Y[0]'

Kb		C,Y[1],Y[0]							
		000	001	011	010	110	111	101	100
A,B	00	X	X	X	X	X	X	X	X
	01	1	1	1	0	1	1	1	1
	11	X	X	X	X	X	X	X	X
	10	X	X	X	X	X	X	X	X

Kb = Y[1]' + Y[0] + C

Jc		C,Y[1],Y[0]							
		000	001	011	010	110	111	101	100
A,B	00	0	0	1	0	X	X	X	X
	01	0	0	1	1	X	X	X	X
	11	X	X	X	X	X	X	X	X
	10	0	0	1	0	X	X	X	X

Jc = Y[1]Y[0] + BY[1]

Kc		C,Y[1],Y[0]							
		000	001	011	010	110	111	101	100
A,B	00	X	X	X	X	1	0	1	1
	01	X	X	X	X	1	0	1	1
	11	X	X	X	X	X	X	X	X
	10	X	X	X	X	X	X	X	X

Kc = Y[1]' + Y[0]'

JK-Flip-Flops:

Ja = BCY[1]'Y[0]
Ka = 1
Jb = CY[1]Y[0]'
Kb = Y[1]' + Y[0] + C
Jc = Y[1]Y[0] + BY[1]
Kc = Y[1]' + Y[0]'

D-Flip-Flop implementation:

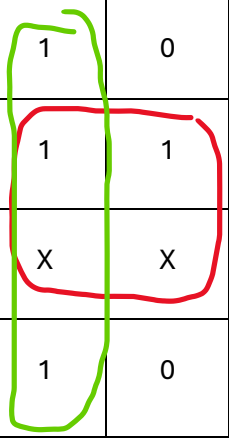
Da		C,Y[1],Y[0]							
		000	001	011	010	110	111	101	100
A,B	00	0	0	0	0	0	0	0	0
	01	0	0	0	0	0	1	0	0
	11	X	X	X	X	X	X	X	X
	10	0	0	0	0	X	X	X	X

Da = BCY[1]'Y[0]

Db		C,Y[1],Y[0]							
		000	001	011	010	110	111	101	100
A,B	00	0	0	0	0	1	0	0	0
	01	0	0	0	1	0	0	0	0
	11	X	X	X	X	X	X	X	X
	10	0	0	0	0	X	X	X	X

Db = B'CY[1]Y[0]' + BC'Y[1]Y[0]'

Dc		C,Y[1],Y[0]							
		000	001	011	010	110	111	101	100
A,B	00	0	0	1	0	0	1	0	0
	01	0	0	1	1	0	1	0	0
	11	X	X	X	X	X	X	X	X
	10	0	0	1	0	X	X	X	X



Dc = Y[1]Y[0] + BC'Y[1]

D-Flip-Flips:

Da = BCY[1]'Y[0]
Db = B'CY[1]Y[0]' + BC'Y[1]Y[0]'
Dc = Y[1]Y[0] + BC'Y[1]

Alarm logic:

4 states, so 2 bits D,E are used for state assignment. FAIL0 is initial state, so it is 00.

FAIL0: DE = 00
FAIL1: DE = 01
FAIL2: DE = 10
ALARM: DE = 11

State Table:

Minterm	Present State	Input	Next State	Output	Flip Flops							
	D,E	FL	D,E	alarm	Td	Te	Jd	Kd	Je	Ke	Dd	De
0	00	0	00	0	0	0	0	X	0	X	0	0
1	00	1	01	0	0	1	0	X	1	X	0	1
2	01	0	01	0	0	0	0	X	X	0	0	1
3	01	1	10	0	1	1	1	X	X	1	1	0
4	10	0	10	0	0	0	X	0	0	X	1	0
5	10	1	11	0	0	1	X	0	1	X	1	1
6	11	0	11	1	0	0	X	0	X	0	1	1
7	11	1	11	1	0	0	X	0	X	0	1	1

Outputs:

alarm		E,FL			
		00	01	11	10
D	0	0	0	0	0
	1	0	0	1	1

alarm = DE

T-Flip-Flop Implementation:

Td		E,FL			
		00	01	11	10
D	0	0	0	1	0
	1	0	0	0	0

Td = D'EFL

Te		E,FL			
		00	01	11	10
D	0	0	1	1	0
	1	0	1	0	0

Te = D'FL + E'FL

T-Flip-Flips:

Td = D'EFL
Te = D'FL + E'FL

JK-Flip-Flop Implementation:

Jd		E,FL			
		00	01	11	10
D	0	0	0	1	0
	1	X	X	X	X

Jd = EFL

Kd		E,FL			
		00	01	11	10
D	0	X	X	X	X
	1	0	0	0	0

Kd = 0

Je		E,FL			
		00	01	11	10
D	0	0	1	X	X
	1	0	1	X	X

Je = FL

Ke		E,FL			
		00	01	11	10
D	0	X	X	1	0
	1	X	X	0	0

Ke = D'FL

JK-Flip-Flips:

Jd = EFL
Kd = 0
Je = FL
Ke = D'FL

D-Flip-Flop Implementation:

Dd		E,FL			
		00	01	11	10
D	0	0	0	1	0
	1	1	1	1	1

Dd = D + EFL

De		E,FL			
		00	01	11	10
D	0	0	1	0	1
	1	0	1	1	1

De = E'FL + DFL + DE + EFL'

D-Flip-Flips:

Dd = D + EFL
De = E'FL + DFL + DE + EFL'

Unlock logic:

4 states, so 2 bits F,G are used for state assignment. LOCKED is initial state, so it is 00.

LOCKED: FG =00
OPEN0: FG = 01
OPEN1: FG = 10
OPEN2: FG = 11

State Table:

Present State	Input	Next State	Outputs	Flip Flops							
F,G	ULK	F,G	lock	Tf	Tg	Jf	Kf	Jg	Kg	Df	Dg
00	0	00	1	0	0	0	X	0	X	0	0
00	1	01	1	0	1	0	X	1	X	0	1
01	0	10	0	1	1	1	X	X	1	1	0
01	1	10	0	1	1	1	X	X	1	1	0
10	0	11	0	0	1	X	0	1	X	1	1
10	1	11	0	0	1	X	0	1	X	1	1
11	0	00	0	1	1	X	1	X	1	0	0
11	1	00	0	1	1	X	1	X	1	0	0

Outputs:

lock		G,ULK			
		00	01	11	10
F	0	1	1	0	0
	1	0	0	0	0

lock = F'G'

T-Flip-Flip Implementation:

Tf		G,ULK			
		00	01	11	10
F	0	0	0	1	1
	1	0	0	1	1

Tf = G

Tg		G,ULK			
		00	01	11	10
F	0	0	1	1	1
	1	1	1	1	1

Tg = F + ULK + G

T-Flip-Flips:

Tf = G
Tg = F + ULK + G

JK-Flip-Flip Implementation:

Jf		G,ULK			
		00	01	11	10
F	0	0	0	1	1
	1	X	X	X	X

Jf = G

Kg		G,ULK			
		00	01	11	10
F	0	X	X	X	X
	1	0	0	1	1

Kg = G

Jh		G,ULK			
		00	01	11	10
F	0	0	1	X	X
	1	1	1	X	X

Jh = F + ULK

Kh		G,ULK			
		00	01	11	10
F	0	X	X	1	1
	1	X	X	1	1

Kh = 1

JK-Flip-Flips:

Jf = G
Kf = G
Jg = F + ULK
Kh = 1

D-Flip-Flip Implementation:

Df		G,ULK			
		00	01	11	10
F	0	0	0	1	1
	1	1	1	0	0

Dg = F'G + FG'

Dg		G,ULK			
		00	01	11	10
F	0	0	1	0	0
	1	1	1	0	0

Dh = FG' + F'ULK

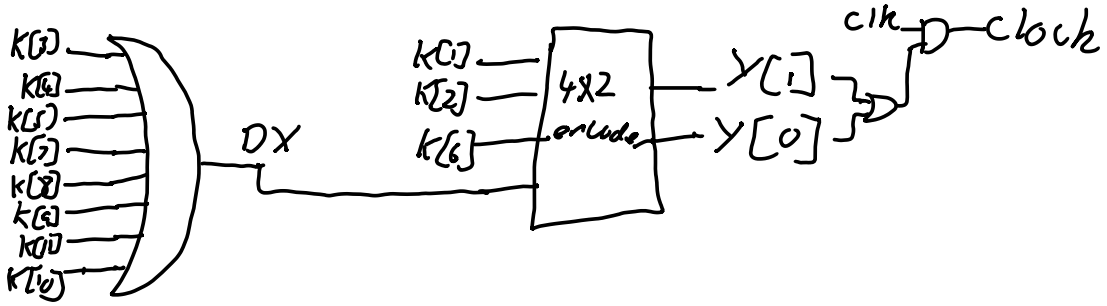
D-Flip-Flips:

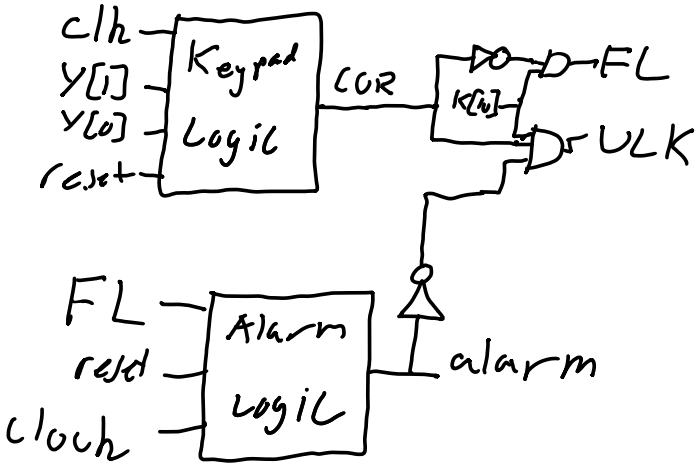
Df = F'G + FG'
Dh = FG' + F'ULK

Logic Diagrams

Global:

Logic used in all implementations:



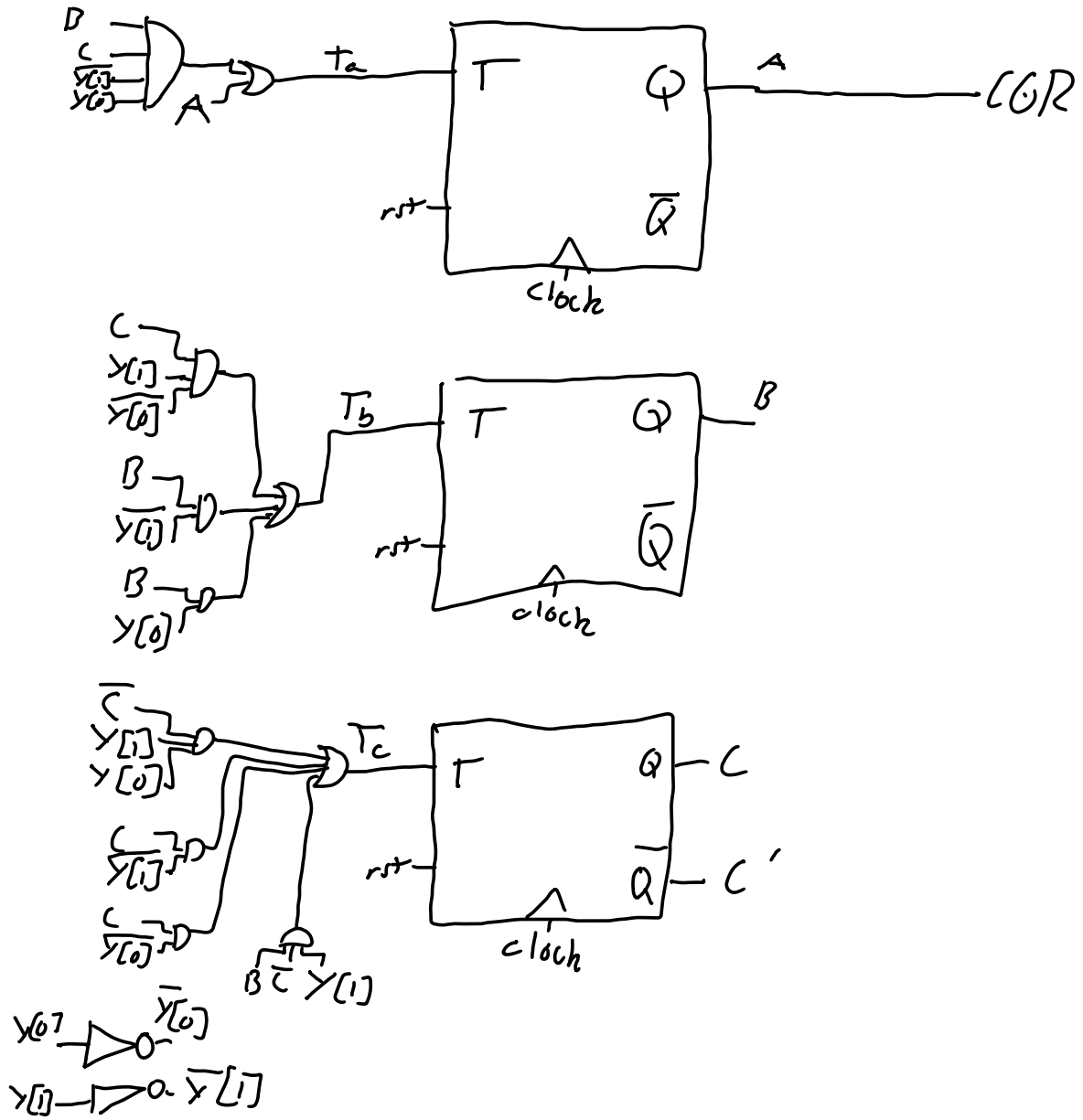


GIC = 8(OR8) + 2*2(2 OR2) + 2*2(2 AND2) + 2(2 inverters) + 4(4x2 encoder) = 22

T-Flip-Flop:

Keypad Logic:

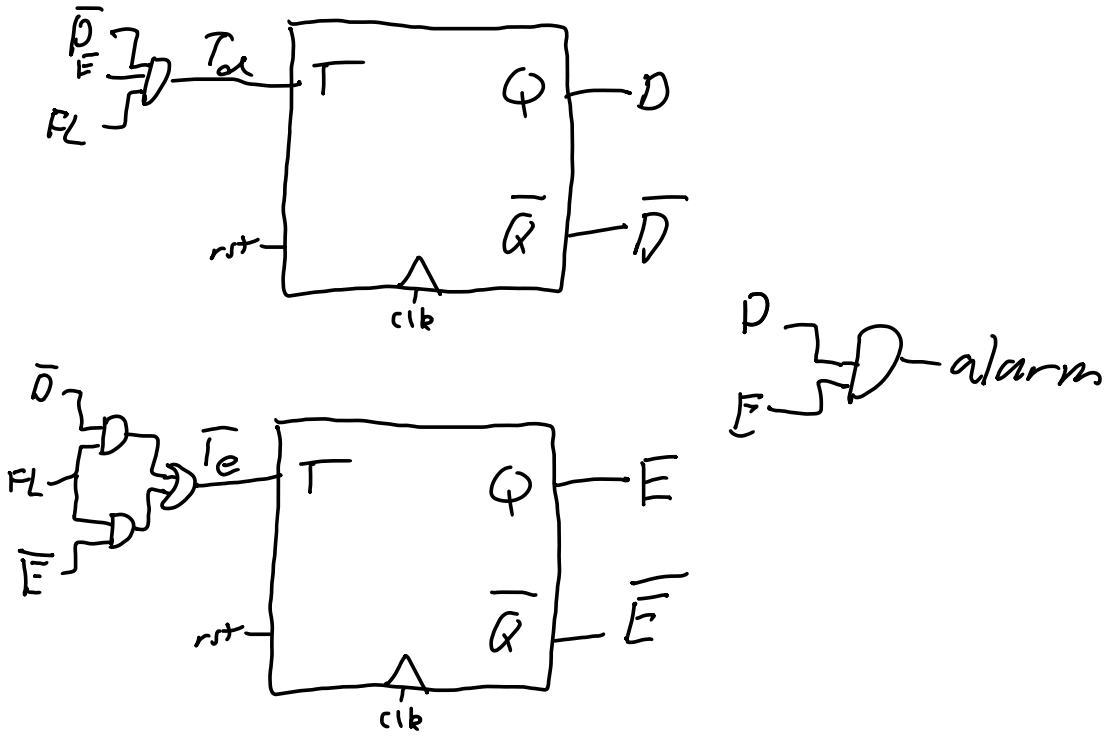
COR = A, Ta = BCY[1]'Y[0] + A, Tb= CY[1]Y[0]' + BY[1]' + BY[0], Tc = C'Y[1]Y[0] + CY[1]' + CY[0]' + BC'Y[1]



GIC = 4(AND4) + 3*3(2 AND3) + 4*2(4 AND2) + 2(2 NOT1) + 2(OR2) + 3(OR3) + 4(OR4) = 32

Alarm Logic:

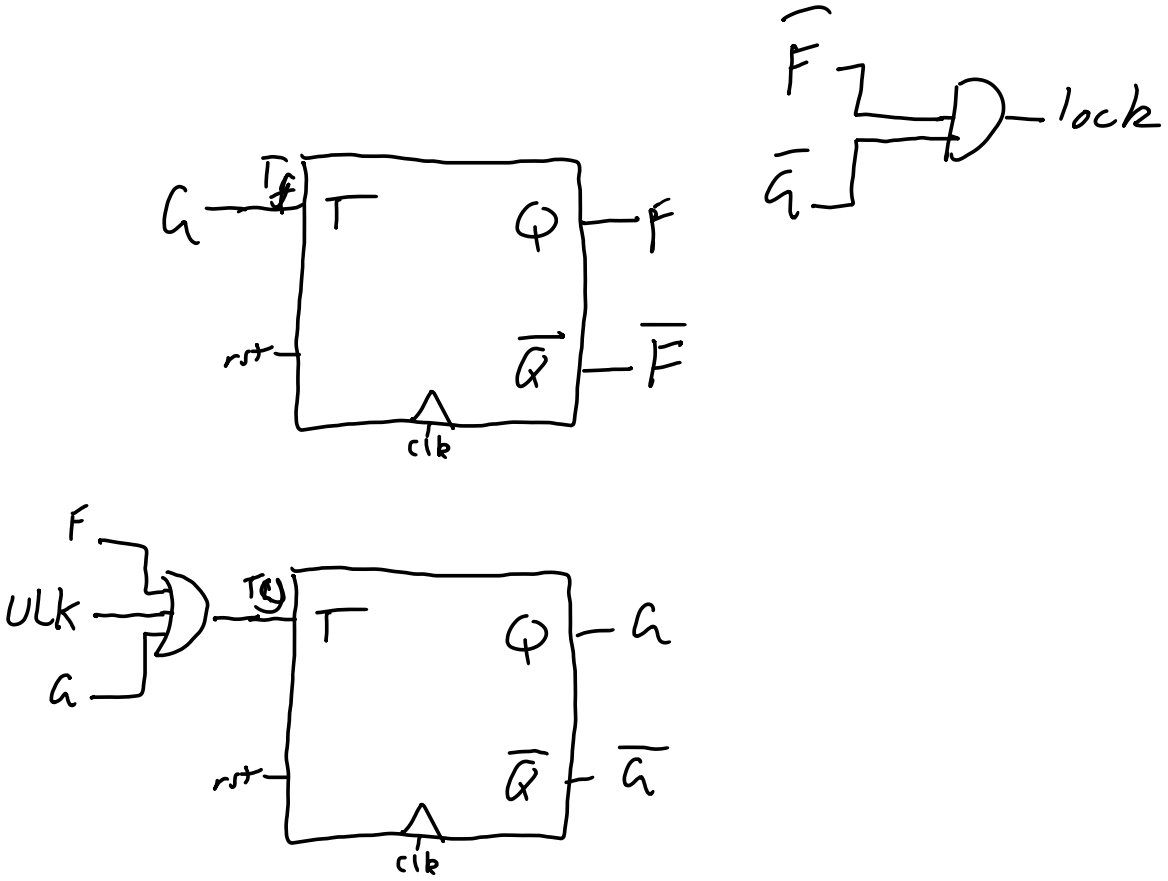
alarm = DE, Td = D’EFL, Te = D’FL + E’FL



GIC = 3*2(3 AND2) + 3(AND3) + 2(OR2) = 11

Unlock Logic:

lock = F'G', Tf = G, Tg = F + ULK + G



GIC = 2(AND2) + 3(OR3) = 5

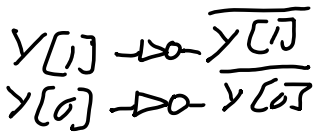
GIC:

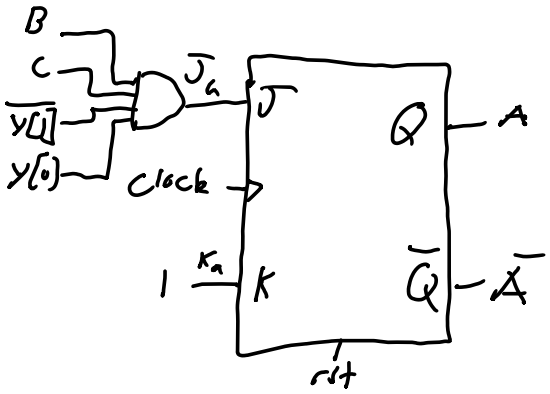
GIC(TFFs) = 32(Keypad logic) + 11(Alarm logic) + 5(Unlock logic) = 48

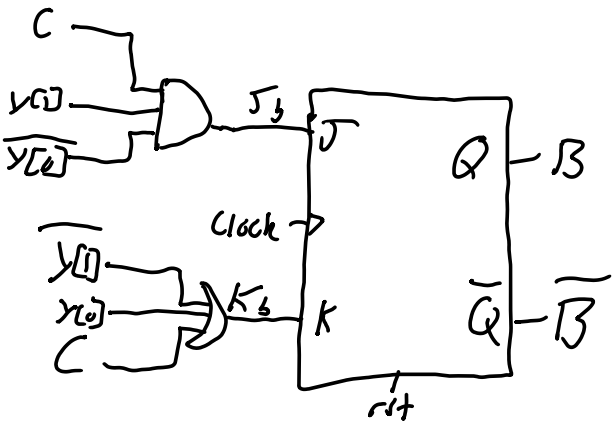
JK-Flip-Flop:

Keypad Logic:

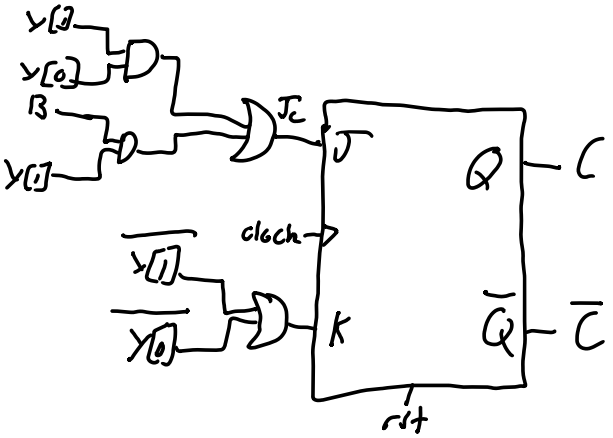
COR = A, Ja = BCY[1]'Y[0], Ka = 1, Jb = CY[1]Y[0]', Kb = Y[1]' + Y[0] + C, Jc = Y[1]Y[0] + BY[1], Kc = Y[1]' + Y[0]'







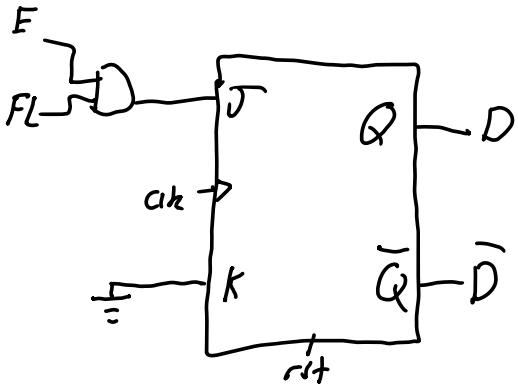
A — COR

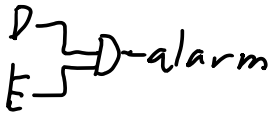


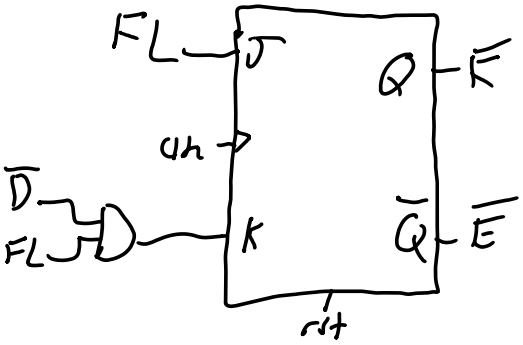
GIC = 2(2 inverters) + 4(AND4) + 3(AND3) + 2*2(2 AND2) + 3(OR3) + 2*2(2 OR2) = 20

Alarm Logic:

alarm = DE, Jd = EFL, Kd = 0, Je = FL, Ke = D’FL



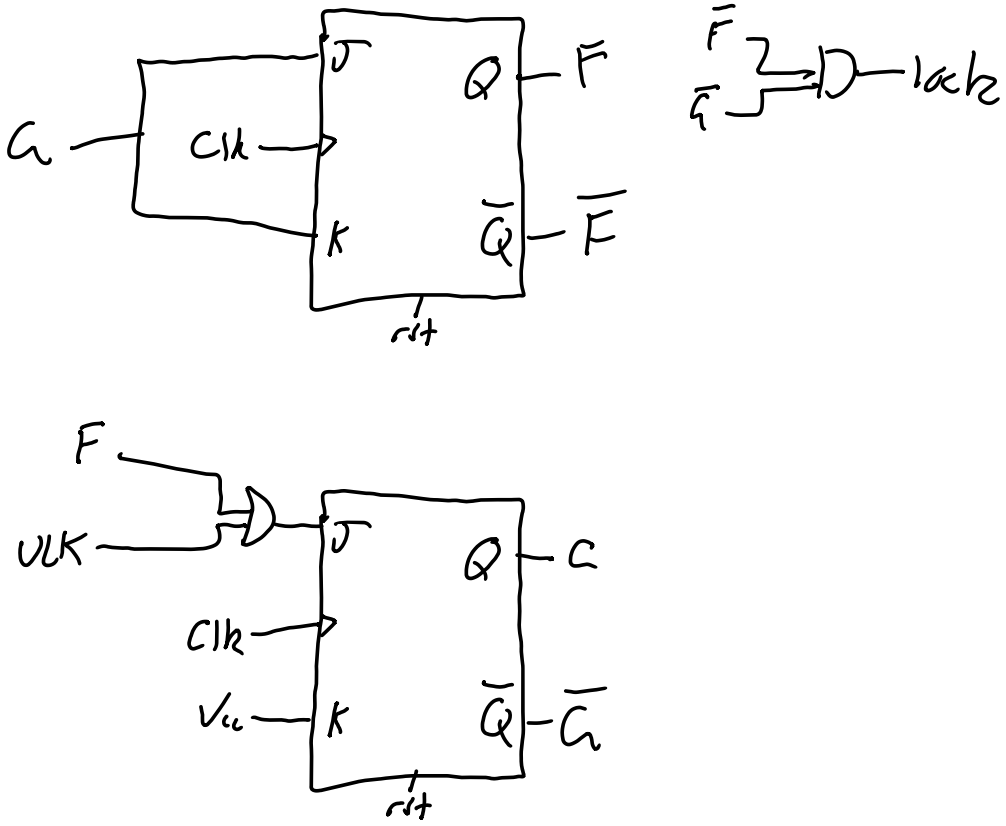




GIC = 3*2(3 AND2) = 6

Unlock Logic:

lock = F'G', Jf = G, Kf = G, Jg = F + ULK, Kg = 1



GIC = 2(AND2) + 2(OR2) = 4

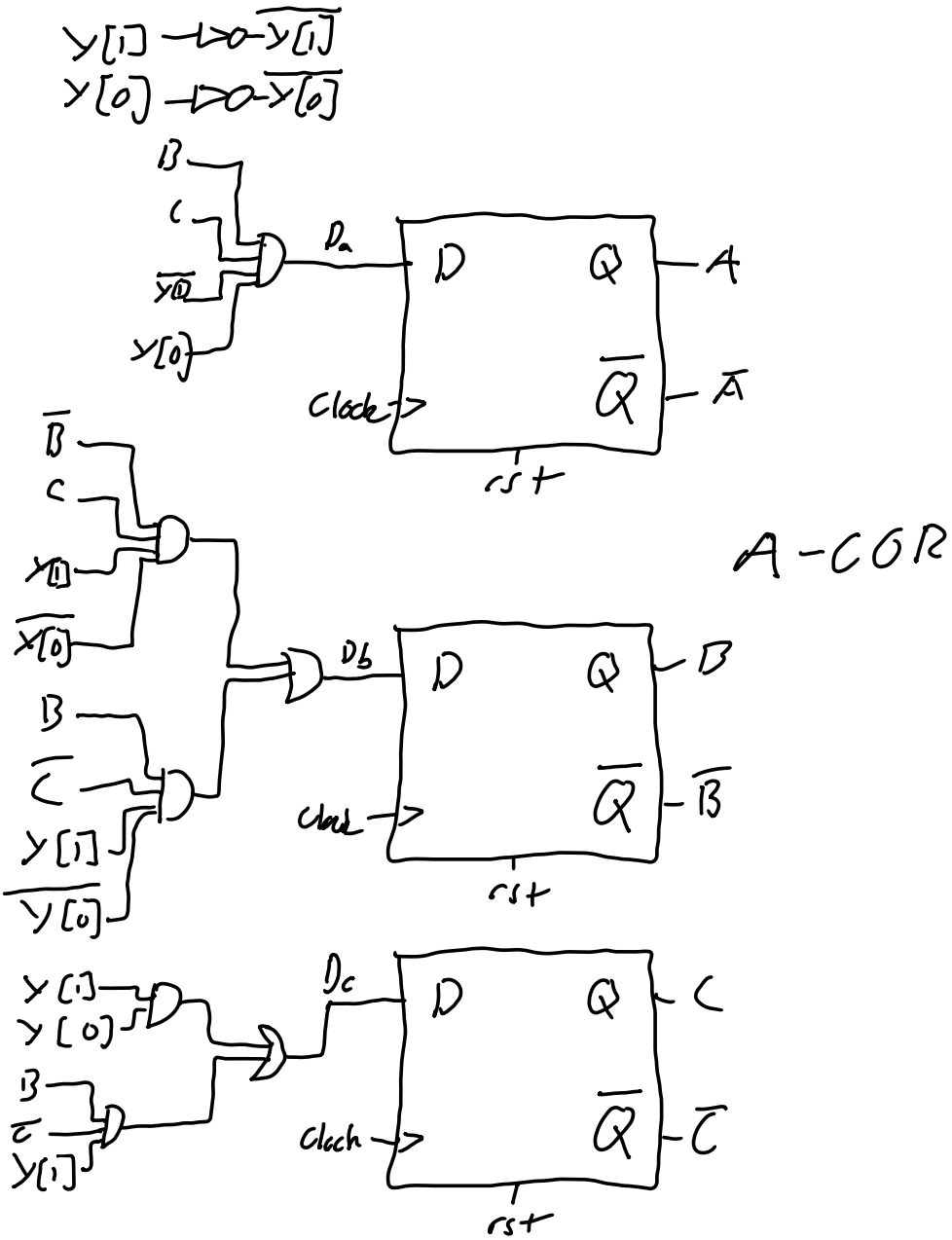
GIC:

GIC(JKFFs) = 20(Keypad logic) + 6(Alarm logic) + 4(Unlock logic) = 30

D-Flip-Flop:

Keypad Logic:

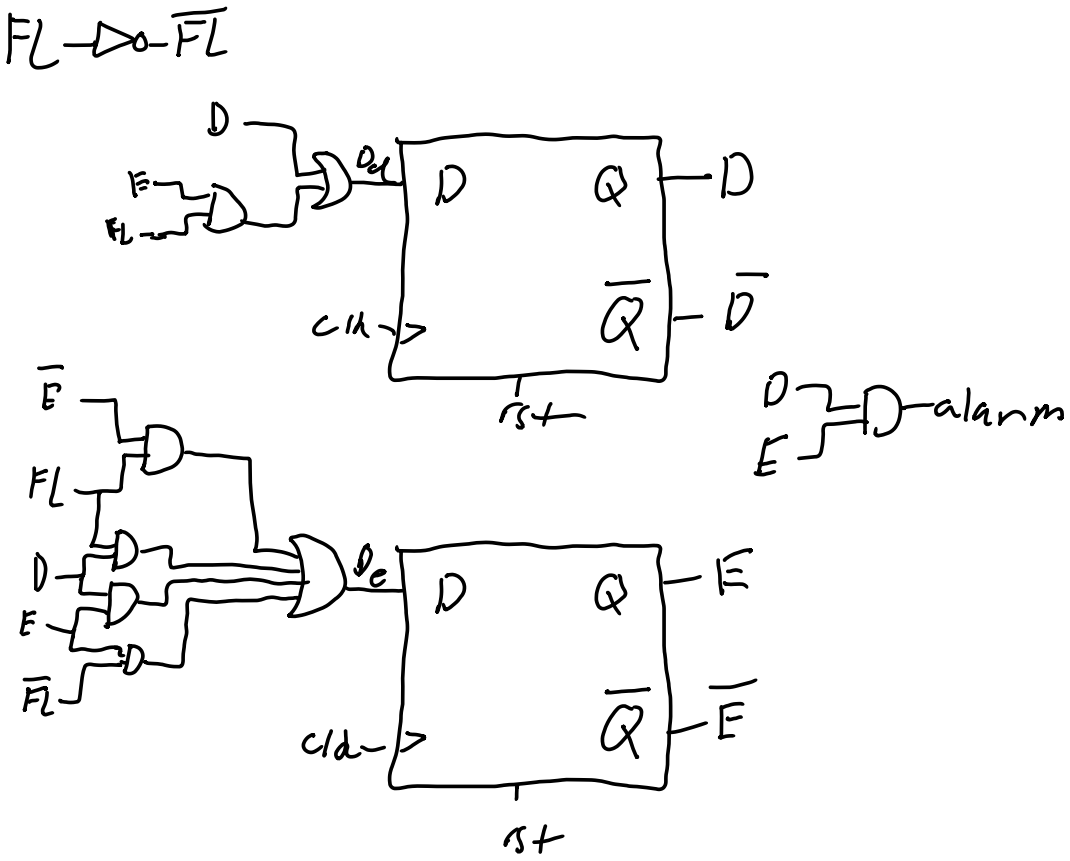
COR = A, Da = BCY[1]'Y[0], Db = B'CY[1]Y[0]' + BC'Y[1]Y[0]', Dc = Y[1]Y[0] + BC'Y[1]



GIC = 2(2 inverters) + 3*4(3 AND4) + 3(AND3) + 2(AND2) + 2*2(2 OR2) = 23

Alarm Logic:

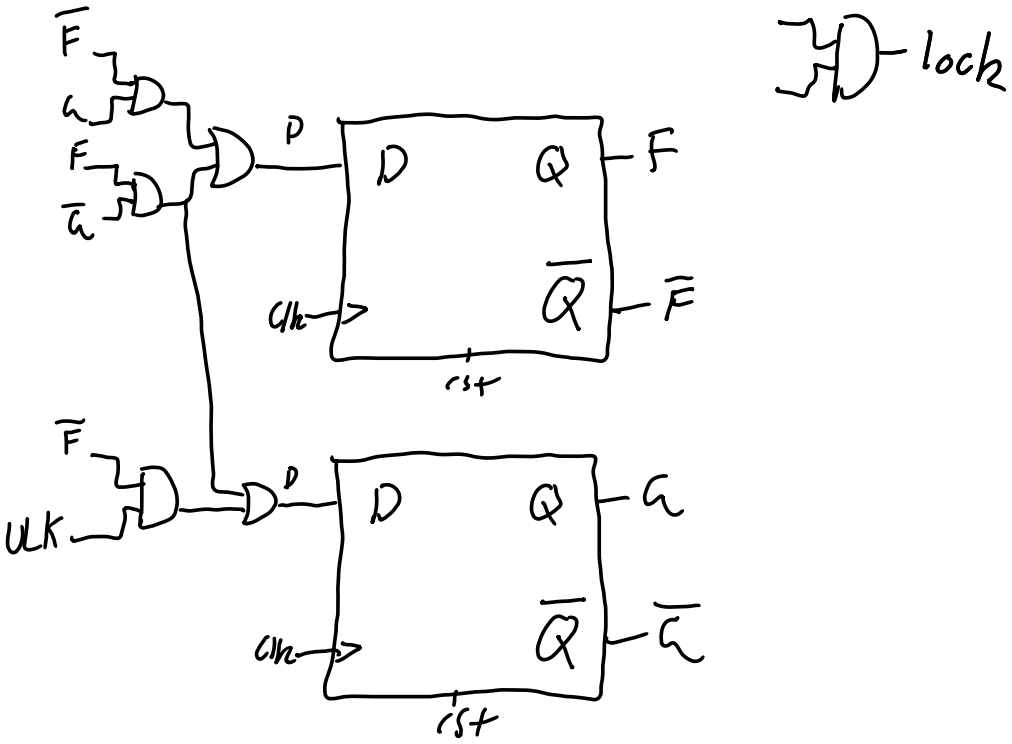
alarm = DE, Dd = D + EFL, De = E'FL + DFL + DE + EFL'



GIC = 1(inverter) + 6*2(6 AND2) + 4(OR4) + 2(OR2) = 19

Unlock Logic:

lock = F'G', Df = F'G + FG', Dh = FG' + F'ULK



GIC = 4*2(4 AND2) + 2*2(2 OR2) = 12

GIC:

GIC(DFFs) = 23(Keypad logic) + 19(Alarm logic) + 12(Unlock logic) = 54

Best Implementation:

For all three implementations, the GIC of global combinational logic is 22, but in the input and output logic for the flip flops, there is a difference in GIC. If I do not include the GIC of the flip flops themselves, the total GIC of each Flip Flop implementation is as follows:

T-Flip-Flip GIC = 22(Global) + 48(Flip flop logic) = 70

JK-Flip-Flop GIC = 22(Global) + 30(Flip flop logic) = 52

D-Flip-Flop GIC = 22(Global) + 54(Flip flop logic) = 76

Which shows that the best implementation uses JK-Flip-Flops, as the total GIC is 18 lower than when using T-Flip-Flops and 24 lower than when using D-Flip-Flops.

Verilog HDL:

Behavioural model:

My behavioral model functions the same as my flip flop implementations, but with behavioral states rather than flip flops, and is as such split into four files.

Top level global logic:

```
module ass2_behavioural(
input reset, clk,
input [11:0] keypad,
output lock, alarm
    );

   //declare registers for inputs and outputs
   reg ULK,FL,DX,clock,rst;
   wire [1:0] Y;

   //Import Encoder and logic modules.
   Encoder Enc({keypad[1], keypad[2], keypad[6], DX}, Y);
   KeypadLogic    Keypad(Y,clock,rst,COR);
   UnlockLogic    Unlock(ULK,clk,reset,lock);
   AlarmLogic     AlarmL(FL,clk,rst,alarm);

   //Module input/output logic
   always@* begin
      rst = 0;             //Set states to 0 or 1 to stop implied latch
      ULK = 0;
      FL = 0;
      DX = 1;

      //Find ULK and FL fromm keypad input and keypadlogic output as described in general logic
      if (keypad[10] == 1) begin
         if ({COR,alarm} == 2'b10)
            ULK = 1;               //ULK = 1 if enter pressed, code is correct, and alarm is not on
         else if (COR == 0)
            FL = 1;                //FL = 1 if enter pressed and code is incorrect
      end

      //find rst from reset and r(r is resetint renamed)
      if (reset)
         rst = 1;       //rst = 1 if reset = 1


      if ({keypad[0],keypad[3],keypad[4],keypad[5],keypad[7],keypad[8],keypad[9],keypad[10],keypad[11]} == 9'b000000000)
         DX = 0;        //DX = 0 if all always incorrect inputs are not on

      if(clk) begin
         clock = 1;     //stop implied latch
         if({DX,Y[0],Y[1]} == 3'b000)
            clock = 0;  //clock for keypad if clock = 1 and any inputs to keypad are 1
      end

   end


endmodule
```

Keypad Logic:

```
module KeypadLogic(
   input [1:0] Y,
   input clock, rst,
   output reg COR
    );
    //Declare localparams for all states
    localparam SEQ0 = 3'b000;
    localparam SEQ1 = 3'b001;
    localparam SEQ2 = 3'b010;
    localparam SEQ3 = 3'b011;
    localparam SEQ4 = 3'b100;

    //Signal Declaration
    reg [2:0] state_reg, state_next;                 //registers for current and next state

    //State regisster
    always@(posedge clock, posedge rst) begin
      if (rst)
         state_reg <= SEQ0;                          //reset to SEQ0 state on rst
      else
         state_reg <= state_next;                    //Push next state on clock
    end

    //Next-state logic
    always@* begin
      case(state_reg)
         SEQ0  :  begin
                     case(Y)                         //State logic for SEQ0 based on input Y[1:0] from state diagram
                        2'b00 :  state_next = SEQ0;
                        2'b01 :  state_next = SEQ0;
                        2'b10 :  state_next = SEQ0;
                        2'b11 :  state_next = SEQ1;
                     endcase
                  end
         SEQ1  :  begin
```

```
         SEQ1  :  begin
                     case(Y)                         //State logic for SEQ1 based on input Y[1:0] from state diagram
                        2'b00 :  state_next = SEQ0;
                        2'b01 :  state_next = SEQ0;
                        2'b10 :  state_next = SEQ2;
                        2'b11 :  state_next = SEQ1;
                     endcase
                  end
         SEQ2  :  begin
                     case(Y)                         //State logic for SEQ2 based on input Y[1:0] from state diagram
                        2'b00 :  state_next = SEQ0;
                        2'b01 :  state_next = SEQ0;
                        2'b10 :  state_next = SEQ3;
                        2'b11 :  state_next = SEQ1;
                     endcase
                  end
         SEQ3  :  begin
                     case(Y)                         //State logic for SEQ3 based on input Y[1:0] from state diagram
                        2'b00 :  state_next = SEQ0;
                        2'b01 :  state_next = SEQ4;
                        2'b10 :  state_next = SEQ0;
                        2'b11 :  state_next = SEQ1;
                     endcase
                  end
         SEQ4  :  begin
                     case(Y)                         //State logic for SEQ4 based on input Y[1:0] from state diagram
                        2'b00 :  state_next = SEQ0;
                        2'b01 :  state_next = SEQ0;
                        2'b10 :  state_next = SEQ0;
                        2'b11 :  state_next = SEQ1;
                     endcase
                  end
```

```
                        2'b11 :  state_next = SEQ1;
                     endcase
                  end
      endcase
    end

    //Output logic
    always@* begin
      COR = 1'b0;                                   //Set COR to 0 to stop inferred latch
      if (state_reg == SEQ4)
         COR = 1'b1;                                //COR only = 1 when in SEQ4 state
    end
endmodule
```

Alarm Logic:

```
21  module AlarmLogic(
22     input FL, clk, rst,
23     output reg alarm
24     );
25
26     //Declare localparams for all states
27     localparam FAIL0 = 2'b00;
28     localparam FAIL1 = 2'b01;
29     localparam FAIL2 = 2'b10;
30     localparam ALARM = 2'b11;
31
32     //Signal Declaration
33     reg [1:0] state_reg, state_next;                    //registers for current and next state
34     //State register
35     always@(posedge clk, posedge rst) begin
36        if (rst)
37           state_reg <= FAIL0;                           //reset to FAIL0 state on rst
38        else
39           state_reg <= state_next;                      //Push next state on clock
40     end
41
42     //Next-state logic
43     always@* begin
44        case(state_reg)
45           FAIL0 :  begin
46                       if(FL)                            //Move to FAIL1 state if FL input is 1(failed attempt), else stay at FAIL0
47                          state_next = FAIL1;
48                       else
49                          state_next = FAIL0;
50                    end
51           FAIL1 :  begin
52                       if(FL)                            //Move to FAIL2 state if FL input is 1(failed attempt 2), else stay at FAIL1
53                          state_next = FAIL2;
54                       else
55                          state_next = FAIL1;
56                    end
57           FAIL2 :  begin
58                       if(FL)                            //Move to ALARM state if FL input is 1(failed attempt 3), else stay at FAIL2
59                          state_next = ALARM;
60                       else
61                          state_next = FAIL2;
62                    end
63           ALARM : state_next = ALARM;                   //Stay at ALARM state regardless of input
64        endcase
65     end
66
67     //Output logic
68     always@* begin
69        alarm = 1'b0;                                    //Set alarm to 0 to stop inferred latch)
70        if (state_reg == ALARM)
71           alarm = 1'b1;                                 //alarm only = 1 when in ALARM state
72     end
73  endmodule
```

Unlock Logic:

```
21  module UnlockLogic(
22      input ULK, clk, rst,
23      output reg lock
24      );
25      //Declare localparams for all states
26      localparam LOCKED = 3'b000;
27      localparam OPEN0 = 3'b001;
28      localparam OPEN1 = 3'b010;
29      localparam OPEN2 = 3'b011;
30      localparam RESET = 3'b100;
31      //Signal Declaration
32      reg [2:0] state_reg, state_next;                    //registers for current and next state
33      //State register
34      always@(posedge clk, posedge rst) begin
35        if (rst)
36           state_reg <= LOCKED;                           //reset to LOCKED state on rst
37        else
38           state_reg <= state_next;                       //Push next state on clock
39      end
40
41      //Next-state logic
42      always@* begin
43        case(state_reg)
44           LOCKED    :  begin
45                           if(ULK)                       //Move to OPEN0 state if ULK input is 1, else remain locked
46                              state_next = OPEN0;
47                           else
48                              state_next = LOCKED;
49                        end
50           OPEN0     :  state_next = OPEN1;              //State OPEN1 after OPEN0 regardless of input
51           OPEN1     :  state_next = OPEN2;              //State OPEN2 after OPEN1 regardless of input
52           OPEN2     :  state_next = RESET;              //State RESET after OPEN2 regardless of input
53           RESET     :  state_next = LOCKED;             //State LOCKED after RESET regardless of input
54        endcase
55      end
56
57      //Output logic
58      always@* begin
59        lock = 0;                             //Set lock to 0 to stop inferred latch)
60        if (state_reg == RESET)
61           lock = 1'b1;
62        if (state_reg == LOCKED)
63           lock = 1'b1;                                   //lock only = 1 when in LOCKED state
64      end
65  endmodule
```

Encoder:

```
module Encoder(
input [3:0] A,
output reg [1:0] Y
    );
   always@* begin
      case(A)                           //Encode input A to Y case by case. Default case should not be needed due to design specifications
         4'b0001  :  Y = 2'b00;         //but is still included
         4'b0010  :  Y = 2'b01;
         4'b0100  :  Y = 2'b10;
         4'b1000  :  Y = 2'b11;
         default  :  Y = 2'b00;
      endcase
   end
endmodule
```

Dataflow model:

```
module ass2_structural(
input reset, clk,
input [11:0] keypad,
output lock, alarm, COR, Qa, Qb, Qc, Qd, Qe, Qf, Qg, Qh
    );
   //Logic Module input wires
   wire ULK, FL, DX, clock;
   //Encoder output wires
   wire Y1, Y0;
   //JKFF input wires
   wire Ja, Ka, Jb, Kb, Jc, Kc, Jd, Kd, Je, Ke, Jf, Kf, Jg, Kg;
   //JKFF output wires
   wire Qa, QPa, Qb, QPb, Qc, QPc, Qd, QPd, Qe, QPe, Qf, QPf, Qg, QPg, Qh;


   //Unlock logic JKFF output
   assign lock = QPf&QPg;
   assign resetint = Qf;

   //Logic module input logic
   assign ULK = keypad[10]&COR&(~alarm);
   assign FL = keypad[10]&(~COR);
   assign DX = keypad[0]|keypad[3]|keypad[4]|keypad[5]|keypad[6]|keypad[7]|keypad[8]|keypad[9]|keypad[10]|keypad[11];
   assign clock = clk&(DX|Y0|Y1);

   //Encoder logic
   assign Y1 = keypad[1]|keypad[2];
   assign Y0 = keypad[6]|keypad[1];

   //Keypad logic JKFF input
   assign Ja = Qb&Qc&(~Y1)&Y0;
   assign Ka = 1'b1;
   assign Jb = Qc&Y1&(~Y0);
   assign Kb = (~Y1)|Y0;
   assign Jc = (Y1&Y0)|(Qb&Y1);
```

```
   assign Jb = Qc&Y1&(~Y0);
   assign Kb = (~Y1)|Y0;
   assign Jc = (Y1&Y0)|(Qb&Y1);
   assign Kc = (~Y1)|(~Y0);
   //Keypad logic JKFF output
   assign COR = Qa;

   //Alarm logic JKFF input
   assign Jd = Qe&FL;
   assign Kd = 1'b0;
   assign Je = FL;
   assign Ke = QPd&FL;
   //Alarm logic JKFF output
   assign alarm = Qd&Qe;

   //Unlock logic JKFF input
   assign Jf = Qg;
   assign Kf = Qg;
   assign Jg = Qf|ULK;
   assign Kg = 1'b1;


   //Flip flop modules
   //Flip flops for keypad logic use clock as their clock signal
   JKFF A(Ja, Ka, clock, reset, Qa, QPa);
   JKFF B(Jb, Kb, clock, reset, Qb, QPb);
   JKFF C(Jc, Kc, clock, reset, Qc, QPc);

   //Flip flops for alarm and unlock logic use clk as clock signal
   JKFF D(Jd, Kd, clk, reset, Qd, QPd);
   JKFF E(Je, Ke, clk, reset, Qe, QPe);

   JKFF F(Jf, Kf, clk, reset, Qf, QPf);
   JKFF G(Jg, Kg, clk, reset, Qg, QPg);


endmodule
```

J-K-Flip-Flop implementation:

Since our states only change on the positive clock edge, a rising edge JKFF with reset is needed for this design. Below is described a NAND implementation of this JKFF in the dataflow model.

```
21  module JKFF(
22     input J,K,clk,reset,
23     output Q,QP
24  );
25     wire nand1, nand2, nand3, nand4, nand5, nand6;
26     //First layer of nands
27     assign nand1 = ~(QP&J&clk);
28     assign nand2 = ~(Q&K&clk);
29     //Second layer of nands
30     assign nand3 = ~(nand1&nand4);
31     assign nand4 = ~(nand2&nand3&(~reset));
32     //Third layer of nands
33     assign nand5 = ~(nand3&(~clk));
34     assign nand6 = ~(nand4&(~clk));
35     //Fourth layer of nands
36     assign Q = ~(QP&nand5);
37     assign QP = ~(Q&nand6&(~reset));
38  endmodule
```

Test Benches:

For this task, I used the same test bench logic to compare my two implementations to ensure both were accurate to the design specifications. All states are tested in both scenarios, but are only shown in the structural model, as I could not display the state registers as output.

Behavioural:

```
module ass2_behavioural_tb;

    // Inputs
    reg reset;
    reg clk;
    reg [11:0] keypad;

    // Outputs
    wire lock;
    wire alarm;

    // Instantiate the Unit Under Test (UUT)
    ass2_behavioural uut (
        .reset(reset),
        .clk(clk),
        .keypad(keypad),
        .lock(lock),
        .alarm(alarm)
    );


    initial begin
        // Initialize Inputs and reset keypad to start test
        reset = 1;
        clk = 0;
        keypad = 0;
        #100;
        reset = 0;
        //Test P1: Enter correct code and see if door unlocks properly.
        #25; clk = 1; #50; clk = 0; #25;
        keypad[1] = 1;                          //Input 2
        #25; clk = 1; #50; clk = 0; #25;
        keypad[1] = 0;                          //End input 2
        #25; clk = 1; #50; clk = 0; #25;
        keypad[2] = 1;                          //Input 3
        #25; clk = 1; #50; clk = 0; #25;
        keypad[2] = 0;                          //End input 3
        #25; clk = 1; #50; clk = 0; #25;
        keypad[2] = 1;                          //Input 3
        #25; clk = 1; #50; clk = 0; #25;
        keypad[2] = 0;                          //End input 3
        #25; clk = 1; #50; clk = 0; #25;
        keypad[0] = 1;                          //Input 1
        #25; clk = 1; #50; clk = 0; #25;
        keypad[0] = 0;                          //End input 1. Code is 2331, which is incorrect, so clear is pressed.
        #25; clk = 1; #50; clk = 0; #25;
        keypad[11] = 1;                         //Input clear
        #25; clk = 1; #50; clk = 0; #25;
        keypad[11] = 0;                         //End input clear. Code should be reset.
```

```
74      #25; clk = 1; #50; clk = 0; #25;
75      keypad[1] = 1;                      //Input 2
76      #25; clk = 1; #50; clk = 0; #25;
77      keypad[1] = 0;                      //End input 2
78      #25; clk = 1; #50; clk = 0; #25;
79      keypad[2] = 1;                      //Input 3
80      #25; clk = 1; #50; clk = 0; #25;
81      keypad[2] = 0;                      //End input 3
82      #25; clk = 1; #50; clk = 0; #25;
83      keypad[2] = 1;                      //Input 3
84      #25; clk = 1; #50; clk = 0; #25;
85      keypad[2] = 0;                      //End input 3
86      #25; clk = 1; #50; clk = 0; #25;
87      keypad[6] = 1;                      //Input 7
88      #25; clk = 1; #50; clk = 0; #25;
89      keypad[6] = 0;                      //End input 7. Correct code 2337 is now entered
90      #25; clk = 1; #50; clk = 0; #25;
91      keypad[10] = 1;                     //Enter Input. Door should unlock
92      #25; clk = 1; #50; clk = 0; #25;    //Clock cycle 1
93      keypad[10] = 0;                     //Enter uninput
94      #25; clk = 1; #50; clk = 0; #25;    //Clock cycle 2
95      #25; clk = 1; #50; clk = 0; #25;    //Clock cycle 3
96      #25; clk = 1; #50; clk = 0; #25;    //Clock cycle 4: Door should lock on this clock cycle.
97      #25; clk = 1; #50; clk = 0; #25;
98      #25; clk = 1; #50; clk = 0; #25;    //Buffer clock cycles
99      //Test Part 2: Enter three incorrect codes and see if alarm goes off.
100     keypad[10] = 1;                     //Enter should be incorrect code #1 if the keypad resets after unlock.
101     #25; clk = 1; #50; clk = 0; #25;
102     keypad[10] = 0;                     //End enter input
103     #25; clk = 1; #50; clk = 0; #25;
104     keypad[3] = 1;                      //Enter 4
105     #25; clk = 1; #50; clk = 0; #25;
106     keypad[3] = 0;
107     #25; clk = 1; #50; clk = 0; #25;
108     keypad[6] = 1;                      //Enter 7
109     #25; clk = 1; #50; clk = 0; #25;
110     keypad[6] = 0;
111     #25; clk = 1; #50; clk = 0; #25;
112     keypad[9] = 1;                      //Enter 0
113     #25; clk = 1; #50; clk = 0; #25;
114     keypad[9] = 0;
115     #25; clk = 1; #50; clk = 0; #25;
116     keypad[0] = 1;                      //Enter 1. Incorrect code 4701 has been entered
117     #25; clk = 1; #50; clk = 0; #25;
118     keypad[0] = 0;
119     #25; clk = 1; #50; clk = 0; #25;
120     keypad[10] = 1;                     //Enter Input. Should be incorrect code #2.
121     #25; clk = 1; #50; clk = 0; #25;
122     keypad[10] = 0;
```

```
        keypad[10] = 0;
        #25; clk = 1; #50; clk = 0; #25;
        keypad[1] = 1;                          //Input 2
        #25; clk = 1; #50; clk = 0; #25;
        keypad[1] = 0;                          //End input 2
        #25; clk = 1; #50; clk = 0; #25;
        keypad[2] = 1;                          //Input 3
        #25; clk = 1; #50; clk = 0; #25;
        keypad[2] = 0;                          //End input 3
        #25; clk = 1; #50; clk = 0; #25;
        keypad[2] = 1;                          //Input 3
        #25; clk = 1; #50; clk = 0; #25;
        keypad[2] = 0;                          //End input 3
        #25; clk = 1; #50; clk = 0; #25;
        keypad[6] = 1;                          //Input 7
        #25; clk = 1; #50; clk = 0; #25;
        keypad[6] = 0;                          //End input 7. Correct code 2337 is now entered
        #25; clk = 1; #50; clk = 0; #25;
        keypad[6] = 1;                          //Input 7
        #25; clk = 1; #50; clk = 0; #25;
        keypad[6] = 0;                          //End input 7. Incorrect code 23377 is now entered
        #25; clk = 1; #50; clk = 0; #25;
        keypad[10] = 1;                         //Enter Input. Should be incorrect code #3. Alarm should go off
        #25; clk = 1; #50; clk = 0; #25;
        keypad[10] = 0;
        //Test Part 3: Enter correct code while alarm is on to see if it can be bypassed
        #25; clk = 1; #50; clk = 0; #25;
        keypad[1] = 1;                          //Input 2
        #25; clk = 1; #50; clk = 0; #25;
        keypad[1] = 0;                          //End input 2
        #25; clk = 1; #50; clk = 0; #25;
        keypad[2] = 1;                          //Input 3
        #25; clk = 1; #50; clk = 0; #25;
        keypad[2] = 0;                          //End input 3
        #25; clk = 1; #50; clk = 0; #25;
        keypad[2] = 1;                          //Input 3
        #25; clk = 1; #50; clk = 0; #25;
        keypad[2] = 0;                          //End input 3
        #25; clk = 1; #50; clk = 0; #25;
        keypad[6] = 1;                          //Input 7
        #25; clk = 1; #50; clk = 0; #25;
        keypad[6] = 0;                          //End input 7. Correct code 2337 is now entered
        #25; clk = 1; #50; clk = 0; #25;
        keypad[10] = 1;                         //Enter Input. Should not turn off alarm.
        #25; clk = 1; #50; clk = 0; #25;
        keypad[10] = 0;
        #25; clk = 1; #50; clk = 0; #25;
        reset = 1;                              //Enter Reset. Should turn off alarm.
        #25; clk = 1; #50; clk = 0; #25;
        reset = 0;

        // Add stimulus here

    end
```

Structural:

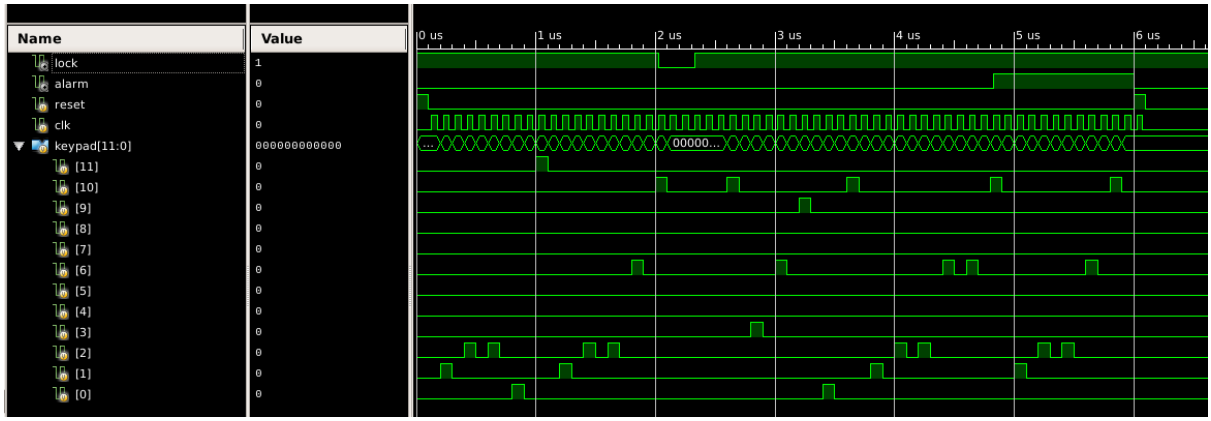
The structural testbench code gives the same inputs but receives more outputs.

```
25  module ass2_structural_tb;
26
27      // Inputs
28      reg reset;
29      reg clk;
30      reg [11:0] keypad;
31
32      // Outputs
33      wire lock;
34      wire alarm;
35      wire Qa;
36      wire Qb;
37      wire Qc;
38      wire Qd;
39      wire Qe;
40      wire Qf;
41      wire Qg;
42
43      // Instantiate the Unit Under Test (UUT)
44      ass2_structural uut (
45          .reset(reset),
46          .clk(clk),
47          .keypad(keypad),
48          .lock(lock),
49          .alarm(alarm),
50          .Qa(Qa),
51          .Qb(Qb),
52          .Qc(Qc),
53          .Qd(Qd),
54          .Qe(Qe),
55          .Qf(Qf),
56          .Qg(Qg)
57      );
```

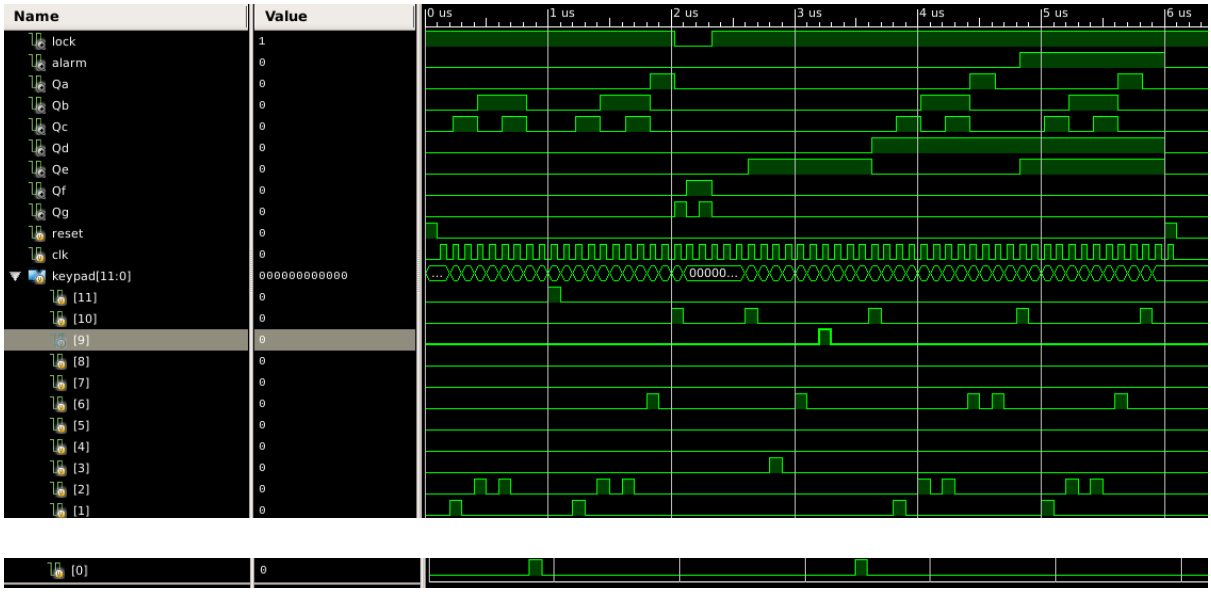
The rest of the code is the same and is thus omitted.

Simulation Results:

Behavioral:



Structural:



We can see from these models that all tests function as expected. Initially, inputting 2,3,3 sets Qb and Qc to 1, indicating the state is SEQ3, and after 1 is pressed, it resets to SEQ0, shown by ABC = 000. Clear also resets to 000. After this, 2,3,3,7 sets ABC to 100, which is SEQ4, indicating the correct code has been entered, and inputting enter correctly unlocks for 3 clock cycles, as shown by Qf and Qg counting to 11(OPEN2) before resetting to 00(LOCKED). After locking again, the first enter press increments Qd,Qe to 01(FAIL1), showing the first incorrect code has been registered. This also applies to the next two incorrect sequences, incrementing to 10(FAIL2) and 11(ALARM), after which pressing enter from the SEQ4 state does not unlock the door or turn off the alarm. The alarm turns off when reset is input, however.