NTNU

TTK 4235

Group 49

Rapport Heislab

Forfattere:
Andreas von Brandis
Sondre Holth Rognlien

March 1, 2020

Introduksjon

Denne rapporten skal inneholde dokumentasjon av designvalg til heisprosjektet i faget TTK4235.

Innholdsfortegnelse

1 Overordnet arkitektur **1**

1.1 Klassediagram 2

1.2 Sekvensdiagram 3

1.3 Tilstandsdiagram 10

1.4 Use case 11

1.5 Designvalg av arkitektur 12

2 Moduldesign **12**

2.1 Main 12

2.2 Floors 12

2.3 Order handler 13

2.4 Door logic 13

2.5 Timer 14

3 Testing **14**

3.1 Enhetstesting 14

3.2 Integrasjonstesting 14

4 Diskusjon **15**

1 Overordnet arkitektur

På et høy nivå kan styresystemet deles opp i 6 moduler. En modul som inneholder åpning og lukking av døren. En modul som holder styr på tiden i forbindelse med åpning av dør eller aktivering av stoppmekanismer. En modul som holder oversiken over hvor heisen til enhver tid befinner seg og en modul som står for håndtering av bestillinger/ordre. Dette innebærer blant å ta i mot ordre, samt prioritere rekkefølgen de utføres i via et køsystem. Denne modulen inneholder også logikk for håntering av lys i forbindelse med bestillinger.

Vi har også en egen modul for hardware. Vi velger vise denne frem, selv om den ble utgitt i oppgaveteksten, ettersom det er en kritisk del av systemt og mer eller mindre alle modulene til en viss grad er avhengig av denne. Vi har valgt å ikke vise frem kode/moduler for input og output samt "Channels.h" som også ble utgitt ettersom dette blir indirekte brukt via hardware og kan gjemmes vekk.

Til slutt har vi en main modul som binder alt sammen. Denne er i all hovedsak bygd opp som en endelig tilstandsmaskin.

1.1 Klassediagram

Under ser dere et klassediagram som beskriver hver av modulene som inngår i designet av heissystemet og hvordan disse modulene relaterer til hverandre.

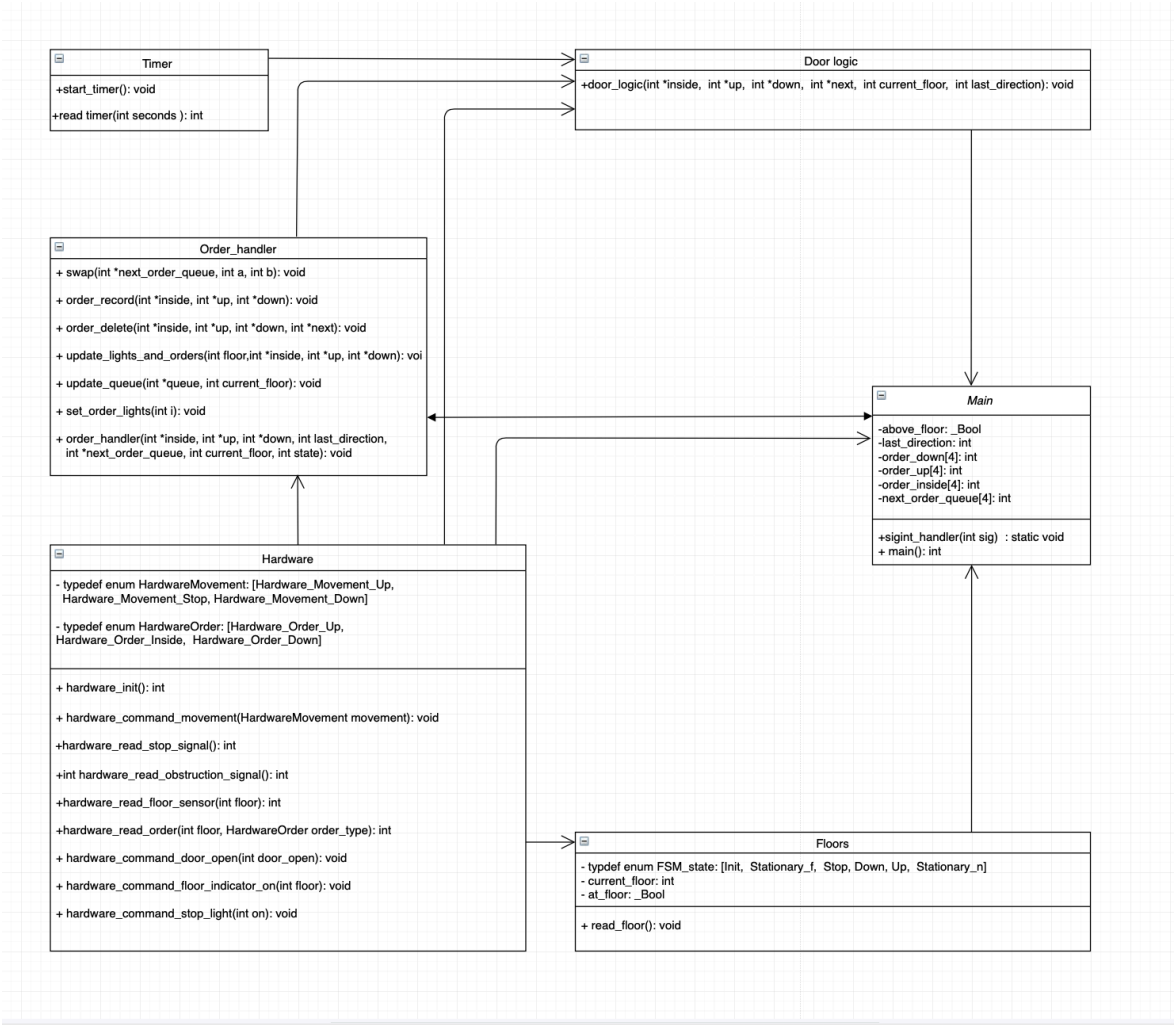
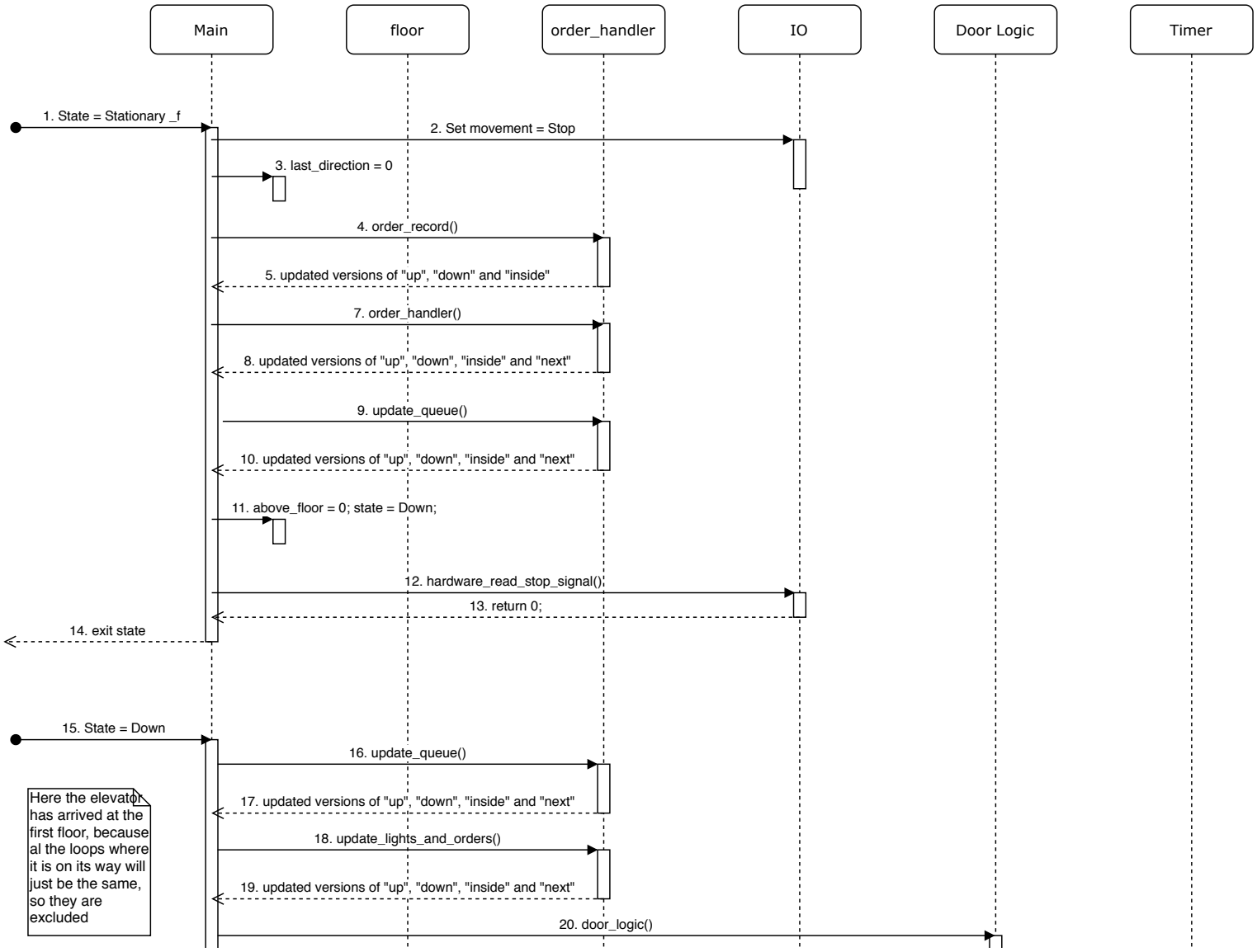


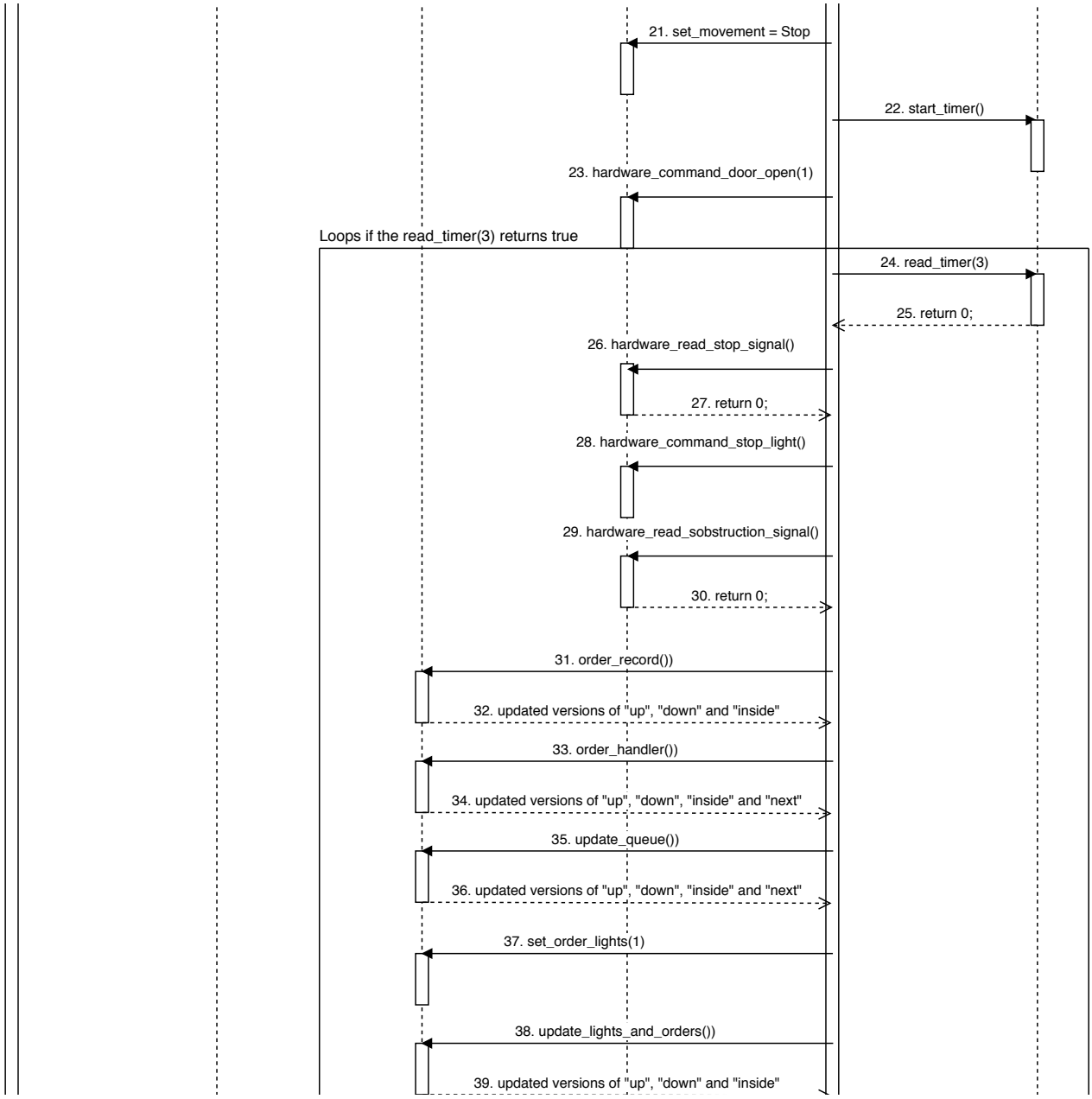
Figure 1: Klassediagram

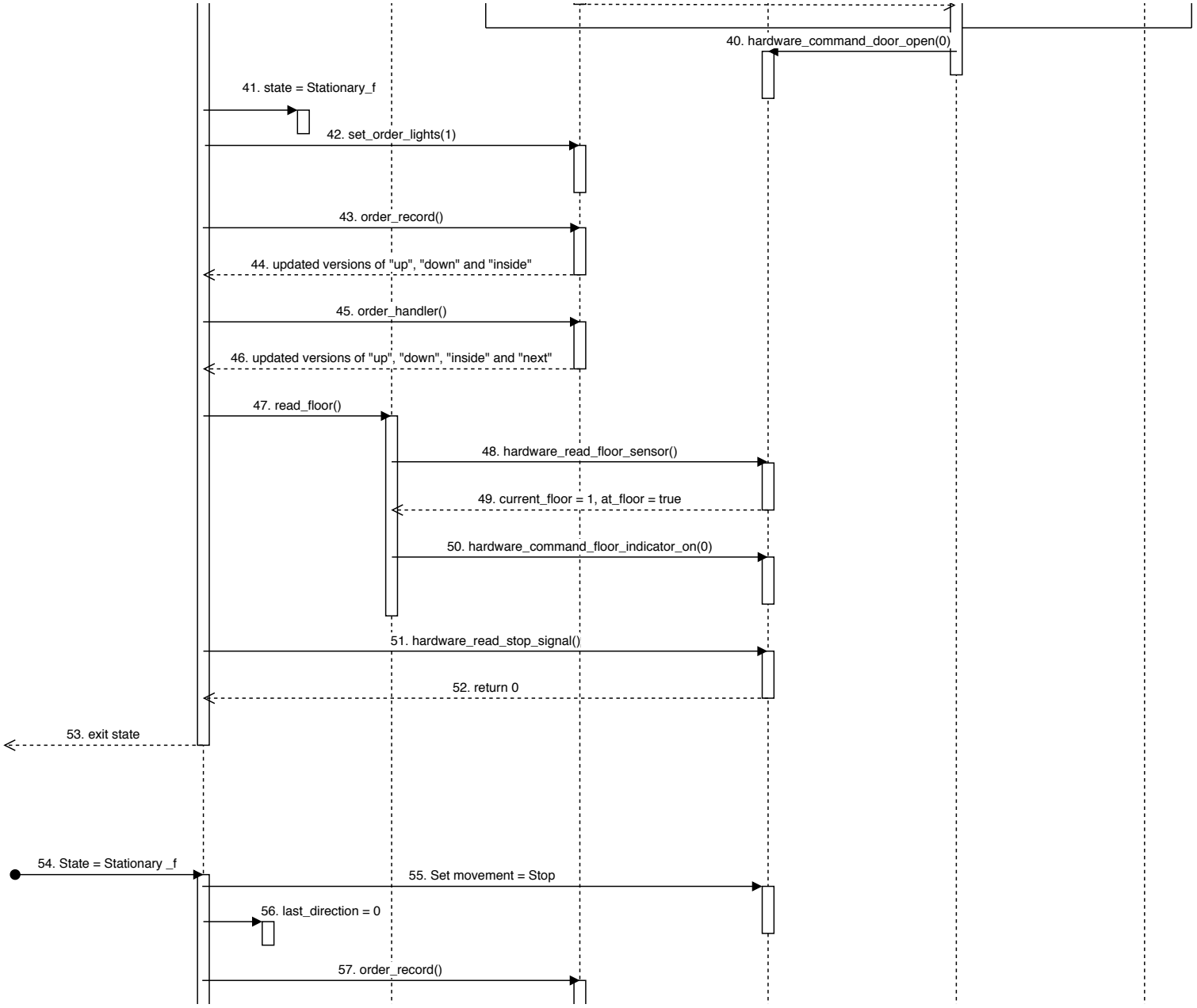
1.2 Sekvensdiagram

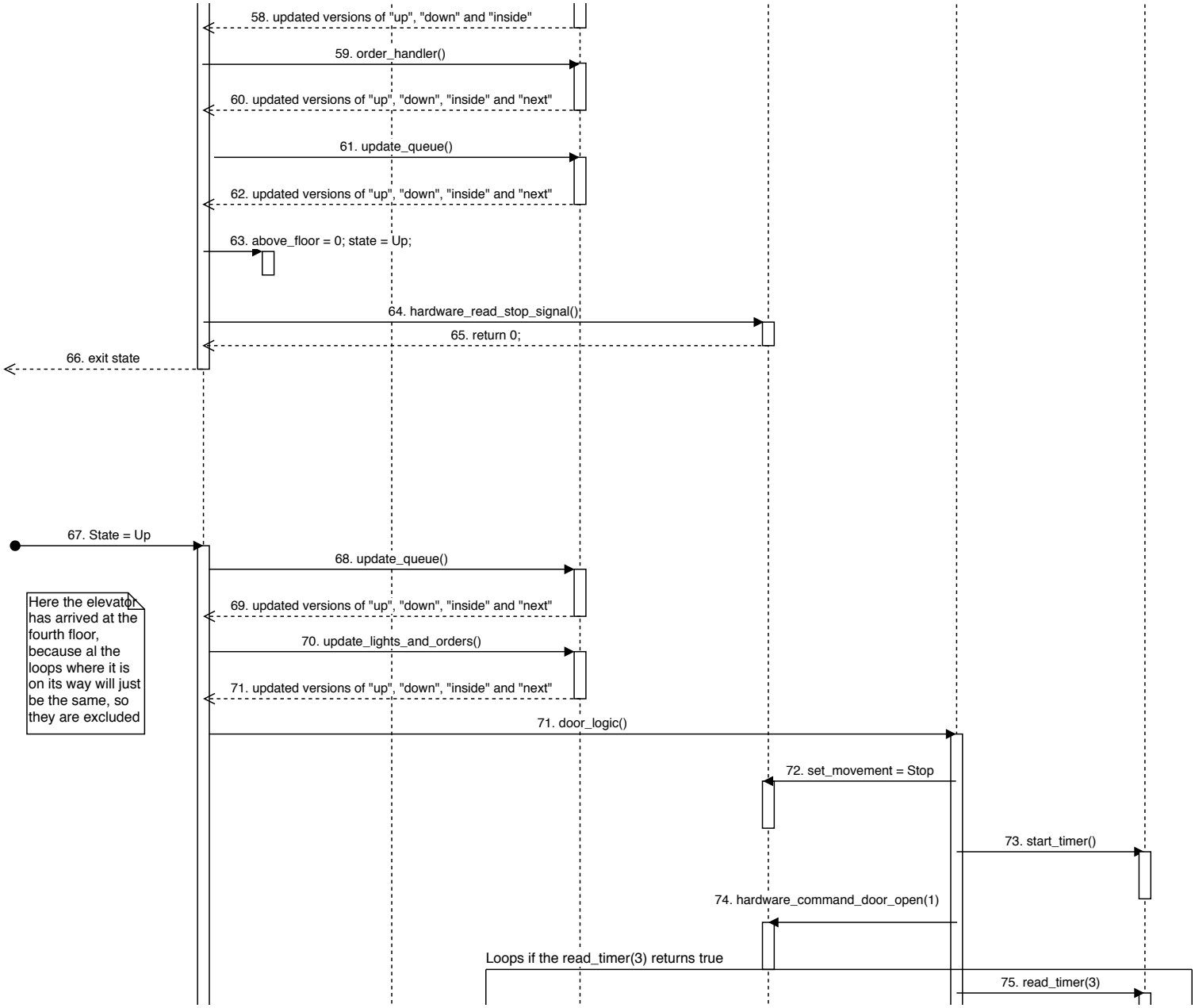
Under ser dere et sekvensdiagram som illustrerer hvordan modulene i systemet fungerer sammen. Sekvensen som vises er den som er oppgitt i oppgavetesten:

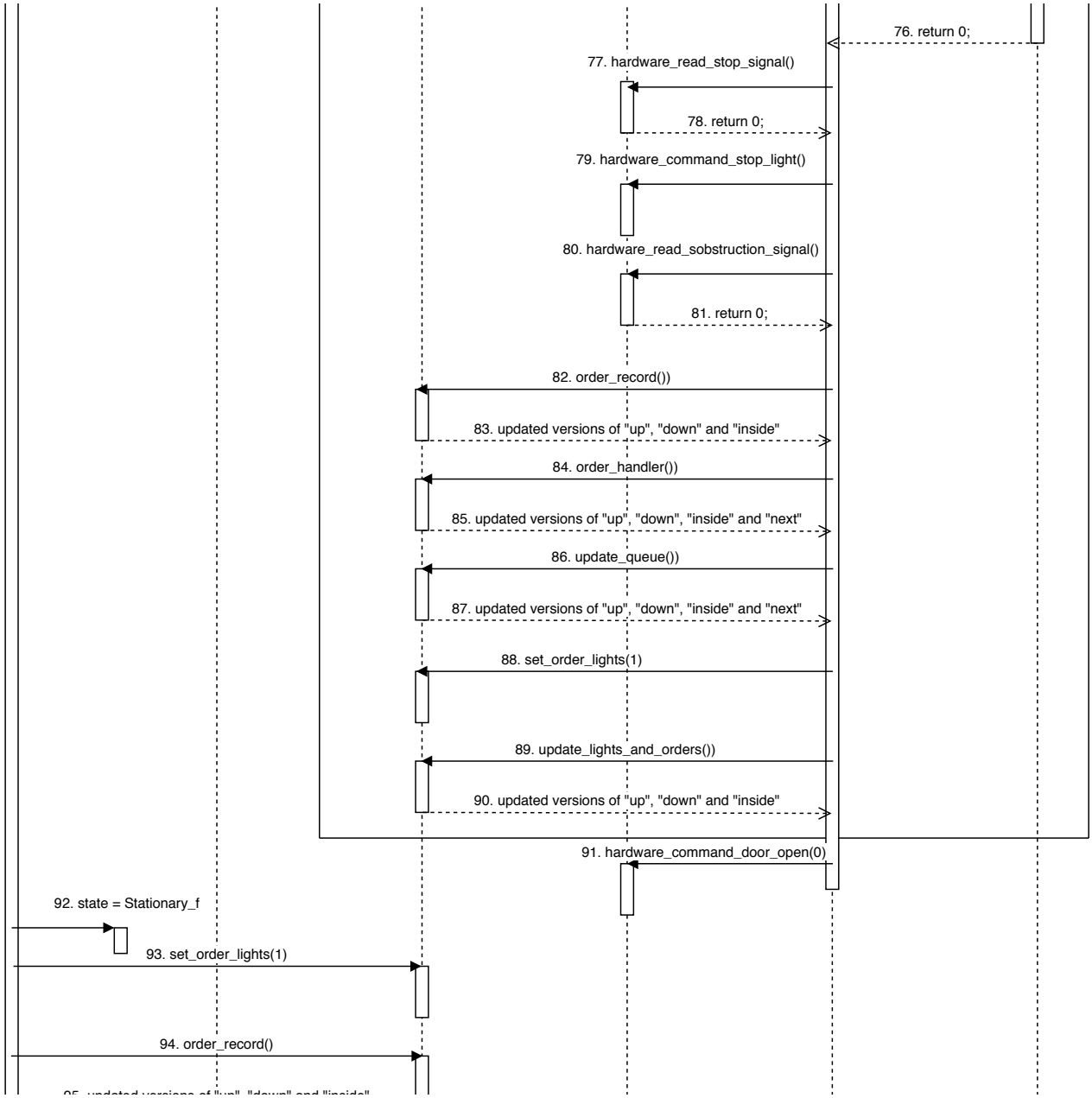
1. Heisen står stille i 2. etasje med døren lukket.
2. En person bestiller heisen fra 1. etasje.
3. Når heisen ankommer, går personen inn i heisen og bestiller 4. etasje.
4. Heisen ankommer 4. etasje, og personen går av.
5. Etter 3 sekunder lukker dørene til heisen seg.

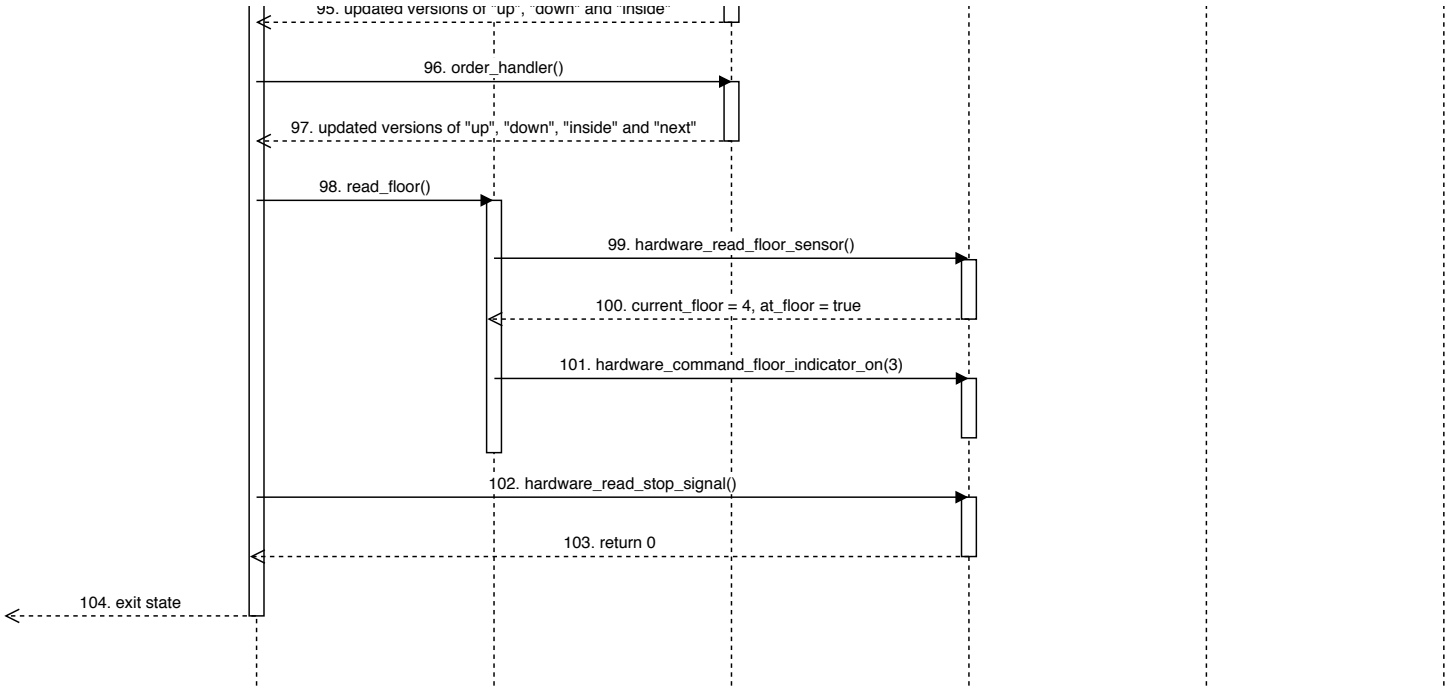

Main
floor
order_handler
IO
Door Logic
Timer
1. State = Stationary _f
2. Set movement = Stop
3. last_direction = 0
4. order_record()
5. updated versions of "up", "down" and "inside"
7. order_handler()
8. updated versions of "up", "down", "inside" and "next"
9. update_queue()
10. updated versions of "up", "down", "inside" and "next"
11. above_floor = 0; state = Down;
12. hardware_read_stop_signal()
13. return 0;
14. exit state
15. State = Down
16. update_queue()
17. updated versions of "up", "down", "inside" and "next"
18. update_lights_and_orders()
19. updated versions of "up", "down", "inside" and "next"
20. door_logic()
Here the elevator has arrived at the first floor, because al the loops where it is on its way will just be the same, so they are excluded


21. set_movement = Stop
22. start_timer()
23. hardware_command_door_open(1)
Loops if the read_timer(3) returns true
24. read_timer(3)
25. return 0;
26. hardware_read_stop_signal()
27. return 0;
28. hardware_command_stop_light()
29. hardware_read_sobstruction_signal()
30. return 0;
31. order_record())
32. updated versions of "up", "down" and "inside"
33. order_handler())
34. updated versions of "up", "down", "inside" and "next"
35. update_queue())
36. updated versions of "up", "down", "inside" and "next"
37. set_order_lights(1)
38. update_lights_and_orders())
39. updated versions of "up", "down" and "inside"


40. hardware_command_door_open(0)
41. state = Stationary_f
42. set_order_lights(1)
43. order_record()
44. updated versions of "up", "down" and "inside"
45. order_handler()
46. updated versions of "up", "down", "inside" and "next"
47. read_floor()
48. hardware_read_floor_sensor()
49. current_floor = 1, at_floor = true
50. hardware_command_floor_indicator_on(0)
51. hardware_read_stop_signal()
52. return 0
53. exit state
54. State = Stationary _f
55. Set movement = Stop
56. last_direction = 0
57. order_record()


58. updated versions of "up", "down" and "inside"
59. order_handler()
60. updated versions of "up", "down", "inside" and "next"
61. update_queue()
62. updated versions of "up", "down", "inside" and "next"
63. above_floor = 0; state = Up;
64. hardware_read_stop_signal();
65. return 0;
66. exit state
67. State = Up
68. update_queue()
69. updated versions of "up", "down", "inside" and "next"
70. update_lights_and_orders()
71. updated versions of "up", "down", "inside" and "next"
71. door_logic()
72. set_movement = Stop
73. start_timer()
74. hardware_command_door_open(1)
Loops if the read_timer(3) returns true
75. read_timer(3)
Here the elevator has arrived at the fourth floor, because al the loops where it is on its way will just be the same, so they are excluded


76. return 0;
77. hardware_read_stop_signal()
78. return 0;
79. hardware_command_stop_light()
80. hardware_read_sobstruction_signal()
81. return 0;
82. order_record())
83. updated versions of "up", "down" and "inside"
84. order_handler())
85. updated versions of "up", "down", "inside" and "next"
86. update_queue())
87. updated versions of "up", "down", "inside" and "next"
88. set_order_lights(1)
89. update_lights_and_orders())
90. updated versions of "up", "down" and "inside"
91. hardware_command_door_open(0)
92. state = Stationary_f
93. set_order_lights(1)
94. order_record()


95. updated versions of "up", "down" and "inside"
96. order_handler()
97. updated versions of "up", "down", "inside" and "next"
98. read_floor()
99. hardware_read_floor_sensor()
100. current_floor = 4, at_floor = true
101. hardware_command_floor_indicator_on(3)
102. hardware_read_stop_signal()
103. return 0
104. exit state

1.3 Tilstandsdiagram

Under ser dere et tilstandsdiagram som viser hvordan heisen oppfører seg til enhver tid avhengig av hvilke input den får inn og hvilke tilsand den befinner seg i.

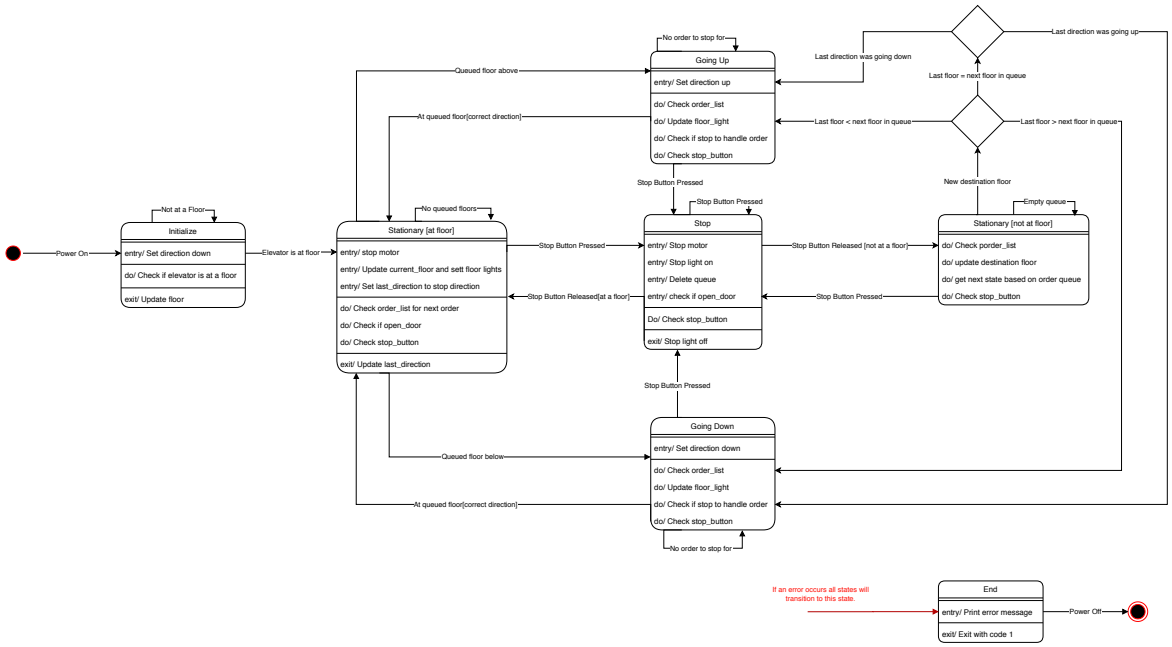


Figure 2: FSM

1.4 Use case

Under ser dere et Use case diagram av systemt. Dette er ment som en simpel og oversiktlig representasjon av en brukers forskjellige interaksjoner med systemet.

Figure 3: Use Case

1.5 Designvalg av arkitektur

I designet av systemt virket det naturlig og oversiktlig å dele systemet opp i konkrete moduler. Det gjorde også lettere å teste hver del for seg. Det virket også ryddig å ha en main som samlepunkt for modulene. Main er bygd opp som en tilstandsmaskin som utgjør kjernen i koden vår. Dette er naturlig da heisen fungerer som en funksjon som kjører helt til programmet termineres utenifra.

2 Moduldesign

Denne seksjonen vil ta for seg modulene i litt større detalj. Hvordan de er bygd opp og implementert samt litt tanker rundt hvorfor vi har tenkt slik vi gjorde.

2.1 Main

Denne modulen er på en måte kjernen i systemet vårt. Her initialiserer vi en del startvariable og sjekker om heisen er i en definert/lovlig tilsand før systemet starter.

Main er i hovedsak bygd opp som en endelig tilstandsmaskin som hele tiden sjekker hvilke state den er i. Den kan enten være i State = "Init" som setter heisen til en definert tilstand. "Stationary f" som sier at heisen står stille og er i en etasje. "Stationary n" som sier at heisen står stille og ikke er i en etasje. "Up" som sier at heisen er på vei oppover og tilsvarende for "Down" bare nedover. Vi har også state = "Stop". Hver av disse tilstandene har en definert oppførsel gitt hvilke inputparemetere systemet får.

2.2 Floors

Denne modulen holder styr på hvor heisen til enhver tid er gjennom variable som gir beskjed om nåværende etasje og om den er på en etasje eller ikke. Modulen består i hovedsak av funksjonen "read floor" som setter disse variablene avhengig av hvilke etasje som blir lest av etasjesensorene.

Her har vi også en enum for de forskjellige tilstandene til heisen.

2.3 Order handler

Denne modulen håndterer alle ordre som styresystemet tar inn og setter tilsvarende lys på eller av. Den baserer seg i hovedsak på at vi har flere forskjellige arrays for hver type ordre som vi aksesserer vi pekere. Et array for ordre nedover, et for oppover, et for ordre fra innsiden og et som inneholder rekkefølgen på de ordrene som skal utføres.

Den har en funksjon "order record" som itererer gjennom alle etasjene og lagrer alle ordre som blir gitt via knappene i de forskjellige arrayene.

Den har også en funksjon kalt "order delete" som sletter alle ordre. Den itererer gjennom og setter alle elementene i de forskjellige arrayene til 0.

Funksjonen "set order lights" slår av eller på alle ordrelys når den tilsvarende knappen for den enkelte etasje blir trykket. Den har en innparameter som er 1 dersom lysene skal bli slått på og 0 dersom de skal slås av.

Vi har også "update lights and orders" som slår av lys når man stopper på korresponderende etasje for å vise at bestillingen er tatt. Den fjerner også bestillingen.

"update queue" sletter første orderen fra arrayet som innholder de neste ordrene som skal utføres. På denne måten vil ikke heisen gå til samme etasje to ganger. Den vil så skifte alle ordre ett hakk til venstre i arrayet, dvs den oppdaterer køen.

Til slutt har vi funksjonen "order handler" som fungerer som køsystemet i heisen vår. Den tar inn alle ordre og returnerer en liste med de neste ordrene i rekkefølge. Den bestemmer altså prioriteten på ordrene. Den baserer seg i hovedsak på iterasjon gjennom alle ordrene for derretter å oppdatere arraryet med rekkefølgen på ordrene. Dersom visse krav er tilfredsstilt så vil rekkefølgen på ordrene bli omstokket. F.eks. dersom heisen er på vei mot 4 etasje, men på veien så får den ordre om å gå til 3 etasje. Dersom heisen da befinner seg under 3 etasje, vil den stoppe i 3 før den går videre til 4. Den vi altså ta ordre fra etasjer som er på vei til etasjen den i utganspuntket skulle til.

2.4 Door logic

Denne modulen inneholder logikk for åpning og lukking av døren. Den benytter seg av Timer modulen og stopper i 3 sekunder når døren åpnes. Så lenge stoppsignalet eller obstruskjonesbryteren aktivert mens døren er åpen, vil den forbli åpen. Døren vil også holde seg åpen i 3 sekunder etter at stoppsignalet har gått lavt.

2.5 Timer

Denne modulen setter opp en timer som brukes av door logic modulen. Den består av to funksjoner. Den første "start timer" initialiserer en tid siden en gitt dato (1/1 - 1970). Den andre funksjonen "read timer" sammenlikner differansen mellom tiden initialisert av "start timer" og den nåværende tiden siden den gitte datoen 1/1 - 1970 med tiden gitt av innparamenteren, her 3 sekunder. Den returener 1 dersom differansen er mindre eller lik 3 sek og 0 ellers.

3 Testing

Denne seksjonen tar for seg hvordan vi testet systemet under implementasjon. Vi delte opp dette i enhetstesting og integrasjonstesting.

3.1 Enhetstesting

Etter at vi var kommet fram til hvordan arkitekturen og moduldesignet skulle se ut, kom vi fram til at vi først skulle lage alle de forskjellige modulene hver for seg og teste disse individuelt før vi satte alt sammen til et helhetlig system.
Her gikk vi frem med enkle tester i GDB, samt et par printf her og der. Her testet vi stort sett ut funksjon for funksjon som f.eks. at "read floor" ga ut riktige verdier på variablene "current floor" og "at floor" avhengig av hvilke etasjesensorer som ble lest.

3.2 Integrasjonstesting

Når vi hadde laget alle modulene og endelig fått de til å kompilere, begynte vi å sette sammen de modulene som snakket med hverandre. Her benyttet vi oss av klasse, sekvens og tilstandsdiagrammene vi hadde laget til å få en klarere oversikt over hvordan vi skulle sette systemet sammen. Testprosedyren vår gikk stort sett ut på å gå gjennom alle punktetene i kravspesifikasjonene en for en. Så vi testet systemet stort sett kun for en spesfikk ting av gangen. Noe av det første vi gjorde var å se om heisen kjørte til en definert tilstand ved oppstart. Dersom den var mellom to etasjer skulle den stoppe på etasjen under. Så gikk vi over til å teste om lysene fungerte slik de skulle ved å kjøre heisen til alle de forskjellige etasjene. Videre implementerte vi dørlogikken i systemet, og så at heisen ventet i 3 sekunder før den kjørte videre. Her oppdaget vi blant annet at heisen ikke kunne ta imot bestillinger i tidsintervallet på 3 sekund der døren var åpen. Etter å ha fikset dette gikk vi over til å teste hvordan ordre håndetreingen og

køsystemet fungerete. Dette gjorde vi ved å skrive ut arrayet som viste rekkefølgen av ordre i terminalen mens koden kjørte. Så testet vi alle mulig tenkelige sekvenser for ordre med ulike starttilstander og så om prioriteringen av rekkefølgen på ordre endret seg slik den skulle. Her oppdaget vi mye rar oppførsel som vi heldigvis klarte å fikse på. En av de "morsomste" feilene vi hadde var om vi trykket på stoppknappen flere ganger mens vi var mellom to etasjer. Da hadde heisen nemlig en tendens til å snu retning.

4 Diskusjon

Så er systemt vårt perfekt? Langt ifra! Koden fungerer og det er mye som er bra, men i ettertid hadde det kanskje vært lurt å gjøre et par ting annerledes. Men dette er noe som ikke er så lett og se direkte, og når man først begynner å bli obs på enkelte ting som kunne vært gjort annerledes føles det ofte litt for seint. Koden bærer dessverre litt preg av "spagetti", spesielt "order handler", men vi konkluderete med at det ble litt mye jobb å skrive om hele koden når det i utganspuntket fungerete slik det skulle. Til kodens forsvar har vi prøvd å få "spagettien" til å se mest mulig ryddig ut. Vi fant også ut at det ble litt overkill å definere 10 forskjellige get funksjoner for alle mulige variable og at dette egentlig ikke kom til å minimere koden så veldig mye.
I etterpåklokskap kunne vi kanskje også ha laget en helt egen modul for køsystemet, ettersom "Order Handler" er en modul som endte opp med å gjøre veldig mye. På en annen side mener vi at denne modulen er relativt minimalistisk i forhold til alt den gjør.
Ellers prøvde vi å lage modulene mest mulig selvestedige. Dette viste seg å ikke alltid være så lett, men med god bruk av klassediagrammet kom vi fram til hvilke moduler som måtte snakke med hverandre og hvilke som kunne la være.
Vi kunne også kanskje vært litt mer bevisst på navnsetting av variable. Vi kunne f.eks. skrevet g- foran variable som var globale og p- foran pekervariable osv. Ellers føler vi de fleste funksjonene har relativt selvforklarende navn, noe som øker lesbarheten på koden.