CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Tillförlitlig och säker växling mellan manuell, fjärrstyrd och autonom styrning av en liten passagerarfärja

Kandidatarbete inom Informationsteknik och Datavetenskap

Albin Ryberg
Alice Moss
Ida Vranvuk
Josef Wallenlind
Theodor Castenström
Tim Tell Arvidsson

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2026

Kandidatarbete 2026

Tillförlitlig och säker växling mellan manuell, fjärrstyrd och autonom styrning av en liten passagerarfärja

Albin Ryberg
Alice Moss
Ida Vranvuk
Josef Wallenlind
Theodor Castenström
Tim Tell Arvidsson

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola
Göteborgs universitet
Göteborg, Sverige 2026

Tillförlitlig och säker växling mellan manuell, fjärrstyrd och autonom styrning av en liten passagerarfärja

Albin Ryberg
Alice Moss
Ida Vranvuk
Josef Wallenlind
Theodor Castenström
Tim Tell Arvidsson

Handledare: Örjan Askerdal, Institutionen för Data- och Informationsteknik
Examinatorer: Patrik Jansson och Arne Linde, Institutionen för Data- och Informationsteknik
Rättande lärare: Magnus Almgren, Institutionen för Data- och Informationsteknik

Kandidatarbete 2026
Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola och Göteborgs Universitet
SE-412 96 Göteborg
Telefon +46 31 772 1000

Satt med hjälp av LaTeX
Göteborg, Sverige 2026

Tillförlitlig och säker växling mellan manuell, fjärrstyrd och autonom styrning av en liten passagerarfärja

Albin Ryberg, Alice Moss, Ida Vranvuk, Josef Wallenlind, Theodor Castenström, Tim Tell Arvidsson

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola och Göteborgs Universitet

Abstract

Autonomous and remote-controlled ships are becoming gradually more common in maritime traffic. This development requires functionally safe control systems. It is especially important that the transition between control stations is implemented in a reliable way, to avoid dangerous system states. This report focuses on examining how safe transitions between control stations can be implemented for passenger ferries.

The project aims to design, implement and evaluate transition logic that ensures safe transitions, where only one control station is active at any given time. Using the monitor-actuator design pattern, the system is implemented as a state machine.

In accordance with the design pattern, the system is structured by separating logic for monitoring and transitions, where a central component is responsible for deterministic decisions. Handshake-protocols, signal validation and fail-safe mechanisms are used within the system. A combination of unit- and integration testing is used to verify the system.

Testing showed that the system can handle transitions between control stations in a safe way. The system architecture is modular and each component has a single responsibility, but it's limited by the central component being a single point of failure (SPOF). The report indicates that the monitor-actuator pattern can be designed, implemented and verified to guarantee reliable transitions between control stations. The report is written in Swedish.

Keywords: passenger ferry, functional safety, PLC, state machine, monitor-actuator, fail-safe, determinism, transition logic, maritime traffic

Sammanfattning

Autonoma och fjärrstyrda fartyg blir successivt vanligare inom sjöfart. Denna utveckling ställer stora krav på den funktionella säkerheten hos styrsystem. Särskilt viktigt är det att övergångarna mellan olika körstationer sker på ett tillförlitligt sätt för att undvika farliga systemtillstånd. Detta arbete fokuserar på att undersöka hur säkra övergångar mellan körstationer kan implementeras för en passagerarfärja.

Arbetets syfte är att designa, implementera och utvärdera övergångslogik som garanterar säkra övergångar där enbart en körstation är aktiv åt gången. Lösningen använder designmönstret monitor-actuator och implementeras som en tillståndsmaskin.

I enlighet med designmönstret bygger systemet på separerad övervakning och övergångslogik där en central komponent ansvarar för deterministiska beslut. I systemet används handshake-protokoll, signalvalidering och fail-safe-mekanismer. För att verifiera systemet utfördes enhetstester och integrationstester i en simulerad miljö.

Resultatet av testningen visar att systemet kan hantera övergångar mellan körstationer på ett säkert sätt. Systemarkitekturen är modulär och varje komponent har ett tydligt ansvarsområde, men begränsas av att den centrala komponenten utgör en single point of failure (SPOF). Arbetet indikerar att monitor-actuator mönstret kan designas, implementeras och verifieras för att säkerställa säkra övergångar mellan körstationer.
Rapporten är skriven på svenska.

Nyckelord: passagerarfärja, funktionell säkerhet, PLC, tillståndsmaskin, monitor-actuator, fail-safe, determinism, övergångslogik, sjöfart

Förord

Vi vill först uttrycka stor uppskattning för vår handledare Örjan Askerdal som har väglett oss i arbetet genom att ge värdefulla insikter och betydelsefull respons.

Samtidigt vill vi tacka uppdragsgivaren CStrider för samarbetet samt ett lärorikt studiebesök. Dessutom tackar vi elektronikföretaget IFM för vägledning och tillgång till deras hård- och mjukvara.

Albin Ryberg, Alice Moss, Ida Vranvuk, Josef Wallenlind, Theodor Castenström, Tim Tell Arvidsson, Göteborg, juni 2026

Innehåll

Figurer

Tabeller

Ordlista

Körstation	En enhet eller plats där färjan aktivt kan styra och manövreras.
PLC	Programmerbar logisk styrenhet (Programmable Logic Controller). En styrenhet som används för att övervaka och styra industriella processer.
CR710S	Central styrenhet utvecklad av IFM med två inbyggda PLC-enheter, en standard- och en safety-PLC.
IFM	Elektronikföretag som tillhandahållit hårdvara och utvecklingsmiljö för detta projekt.
CODESYS	Utvecklingsmiljö för programmering av PLC-baserade system.
SPOF	Single Point of Failure, en komponent vars fel leder till att hela systemet slutar fungera.
FMEA	Failure Mode and Effects Analysis, en metod för att systematiskt identifiera och bedöma risker i ett system.
ActuationChannel	Delen av systemet som ansvarar för att utföra styrbeslut.
MonitoringChannel	Delen av systemet som övervakar ActuationChannel och kan utlösa ett fail-safe tillstånd vid fel.
ControlUnit	Central komponent inom ActuationChannel som samlar all övergångslogik och säkerställer deterministiska beslut.
Fail-safe	En designprincip där systemet vid fel automatiskt går över till ett säkert tillstånd för att minimera skada.
Monitor-actuator	Ett designmönster som separerar styrlogik och övervakning i två oberoende kanaler.
Handshake	Ett protokoll där två parter utbyter kontrollsignaler för att koordinera en övergång eller överföring.
Tillståndsmaskin	En modell som beskriver ett system med ett begränsat antal tillstånd och definierade övergångar.
Watchdog	En övervakningskomponent som detekterar fel i ett system genom att kontrollera att systemet svarar inom en given tid.
3LSM	Three Level Safety Monitor, ett designmönster som är en vidareutveckling av monitor-actuator med ytterligare övervakningsnivåer.
OT-system	Operational Technology är system som övervakar och styr fysiska processer i industri och infrastruktur.

Heartbeat	En periodisk signal som skickas från en körstation för att indikera att den är aktiv och tillgänglig.
Styrläge/Körstation	En enhet eller ett läge som har kontroll för manövrering av färjan. Termerna används synonymt i denna rapport.
MoSCoW	Ett prioriteringsramverk med kategorierna Must have, Should have, Could have och Will not have.
Hazardanalys	En analys för att identifiera och bedöma potentiella risker och faror i ett system.
IO-Signaler	Input/Output-signaler, som används för kommunikation mellan en PLC och externa enheter.

1
Introduktion

Moderna styrsystem ställer höga krav på tillförlitlighet och säkerhet [1]. Detta gäller särskilt i system där flera körstationer eller operatörer kan ta kontroll över samma funktion, exempelvis i maritima system såsom färjor [2]. I dessa styrsystem uppstår utmaningar att säkerställa entydig och säker styrning, särskilt när det gäller att garantera att endast en körstation eller operatör är aktiv åt gången. Flera aktiva stationer kan leda till odefinierat och potentiellt farligt beteende för passagerare. På samma sätt är situationen där ingen station är aktiv lika problematisk, då det innebär att färjan saknar styrning helt [2]. Detta ställer krav på robusta mekanismer för övergång mellan körstationer och felhantering.

I detta arbete utvecklades en arkitektur för säker och tillförlitlig växling mellan olika körstationer. Arkitekturen bygger på en samling designprinciper som är tillämpbara för olika typer av körstationer. Rapporten beskriver design, implementering och verifiering av ett system som uppfyller givna säkerhetskrav samt som garanterar ett deterministiskt beteende vid övergångar mellan körstationer. För att uppnå detta utvecklades en systemlösning där kontrollbeslut centraliserades samtidigt som styrning och övervakning separerades. Lösningen har implementerats och utvärderats genom testning och verifiering.

Rapporten är strukturerad enligt följande:
Kapitel 1 introducerar bakgrund, syfte, frågeställningar och avgränsningar för arbetet. Kapitel 2 presenterar de teoridelar som arbetet bygger på och Kapitel 3 beskriver den utvecklingsprocess som följts. Vidare beskrivs arkitektur i Kapitel 4 och implementationen i Kapitel 5. I Kapitel 6 presenteras resultatet och arbetet diskuteras i Kapitel 7. Slutligen sammanställs de slutsatser som arbetet bidragit till i Kapitel 8.

1.1 Bakgrund

Utvecklingen av autonoma och fjärrstyrda fartyg, främst inom sjöfrakt, har under de senaste åren accelererat. Denna trend förväntas fortsätta i takt med den tekniska utvecklingen [1]. Under senare år har dock även autonoma passagerarfartyg, såsom mindre färjor i urbana miljöer, fått ökat intresse [3]. Dessa system förväntas bidra till mer hållbar, energieffektiv och flexibel kollektivtrafik i tätorter. Samtidigt ställer införandet av autonoma lösningar mycket höga krav på säkerhet, pålitlighet och robusthet [1].

Detta arbete är ett uppdrag av CStrider som handlar om att utveckla deras autonoma passagerarfärja Celia One. Baserat på möten med Tobias Husberg på CStri-

der samt platsbesök på färjan ges följande beskrivningar. Celia One är en elektrisk katamaran avsedd för vattenburen kollektivtrafik i stadsmiljö [4]. För att uppnå ekonomisk hållbarhet i drift av mindre autonoma färjor arbetar CStrider med autonoma lösningar. Färjan är utrustad med olika körstationer och tekniska system som tillsammans möjliggör drift. Dessa omfattar bland annat manuell styrning ombord och fjärrstyrning från land [4]. Fjärrstyrningen sker via ett Remote Operation Center (ROC), där operatörer kan övervaka och styra färjan på distans. Systemet är utformat med redundans i form av två separata kontrollstationer på land och/eller en station med redundanta kommunikationskanaler. Utöver detta finns även fjärrstyrning via mobilapplikation och två nivåer av autonom styrning: en standard autopilot med simpel rutt-följning och en avancerad med förmågan att upptäcka och hantera hinder.

Behovet av att växla mellan körstationer uppstår naturligt under drift. Exempelvis kan färjan styras autonomt i öppet vatten men kräva manuell styrning vid tilläggning eller att ROC tar över vid förändrade väderförhållanden och nödsituationer. En okontrollerad eller felaktig växling kan leda till att färjan tillfälligt saknar aktiv styrning, vilket utgör en direkt säkerhetsrisk för passagerare och omgivning. Kombinationen av mänsklig och autonom kontroll skapar säkerhetskritiska situationer som kan påverka drift, säkerhet och användarförtroende. Därför är det nödvändigt att förstå både de tekniska utmaningarna och de mänskliga faktorerna i interaktionen med autonoma system. Vid kritiska fel övergår systemet till ett fail-safe tillstånd där färjan släpper ankare. Ankaret valdes som fail-safe åtgärd då den på ett passivt och tillförlitligt sätt förhindrar att färjan driver okontrollerat, utan att kräva aktiv styrning från någon körstation.

Verifiering och testning är centrala för att säkerställa att tekniska system uppfyller sina krav. Produktverifiering innebär att fastställa att en produkt överensstämmer med specificerade krav genom test, analys, inspektion eller demonstration [5]. För autonoma passagerarfärjor innebär detta att styrsystemets funktionalitet och säkerhet kontrolleras systematiskt, vilket är särskilt viktigt i säkerhetskritiska miljöer med passagerare ombord.

Att på ett säkert sätt byta körstation är en fråga relevant både för produktutveckling och forskning. För industrin och startup-företag som CStrider bidrar forskning och utveckling inom detta område till möjligheten att införa säkra, pålitliga och accepterade autonoma färjor i stadsmiljö. Inom akademin ligger ämnet i skärningspunkten mellan kontrollsystem och människa–autonomi-interaktion, vilket är högaktuella forskningsområden. Ur ett samhällsperspektiv bidrar säkra autonoma passagerarfartyg till hållbar urban mobilitet, minskad miljöpåverkan och ökad trygghet för passagerare. Tydliga och pålitliga kontrollsystem ökar dessutom förståelsen, förtroendet och säkerheten för operatörer och användare.

1.1.1 Tidigare forskning

Kontrollövergångar i autonoma fartyg har tidigare identifierats som ett säkerhetskritiskt område inom forskning. Yang et al. [6] identifierar felscenarier i kontrollägesövergångar för autonoma fartyg med besättning ombord och konstaterar att en mer omfattande identifiering av felscenarier bidrar till en tillförlitligare och säkrare kontrollmekanism. I studien används tillståndsmaskiner för att modellera övergångar mellan kontrollägen, ett tillvägagångssätt som även tillämpas i detta arbete. Zhang et al. [2] undersöker kontrollväxlingsmekanismer i autonoma fartyg och konstaterar att industrin ännu inte har etablerat standarder för kontrollväxling, samt att enbart en agent ska ha kontroll över fartyget åt gången. Medan dessa studier bidrar med teoretiska ramverk och riskanalyser, saknas konkreta implementeringar av styrlogik som adresserar de identifierade riskerna. Detta arbete bidrar med en sådan implementering. Styrlogiken hanterar explicit de felscenarier Yang et al. identifierar genom konkreta säkerhetsmekanismer, och adresserar den lucka Zhang et al. lyfter fram gällande avsaknaden av standarder för kontrollväxling genom att säkerställa att enbart en körstation har kontroll åt gången.

1.2 Syfte

Syftet med detta arbete är att analysera, designa och utvärdera en säker och pålitlig styrlogik för växling mellan autonomt, fjärrstyrt och manuellt körstation hos en passagerarfärja. Målet med arbetet är att endast ett körstation ska kunna vara aktivt åt gången samt att övergångar mellan dessa körstationer sker säkert och kontrollerat, även vid systemfel.

1.3 Frågeställningar

För att undersöka hur säkra kontrollväxlingar uppnås, undersöker detta arbete följande frågor:

- Kan det säkerställs att endast en körstation har kontroll åt gången?
- Kan ogiltiga eller otillåtna kontrollövergångar förhindras?
- Kan utvecklingsprocessen och lösningen dokumenteras och utvärderas iterativt?
- Kan lösningen generaliseras så att den fungerar oberoende av vilken typ av körstation som integreras?

1.4 Avgränsningar

Arbetet avgränsas till att huvudsakligen behandla mjukvarulogik, vilket innebär att implementering med fysisk hårdvara inte behandlas. Istället utvärderas systemets logik i en simulerad miljö och med teoretisk analys. Arbetet fokuserar på funktio-

nell säkerhet, det vill säga att systemet bibehåller korrekt beteende även vid fel eller bortfall. Lösningen kommer inte att hantera aspekter som exempelvis cybersäkerhet.

Vidare avgränsas arbetet till en förenklad modell av signalhantering, där ett begränsat antal styrsignaler används för att representera interaktionen mellan körstationerna. Övervakningen baseras enbart på interna signaler, vilket medför att systemets utförande inte verifieras mot externa signaler. Detta resulterar i att systemet saknar återkoppling från exempelvis aktuatorer, vilket i sin tur gör att aktuatorerna inte kan verifiera att en styrsignal har verkställts eller att en tidigare aktiv station har upphört att påverka systemet.

1.5 Metod

Metoden följer en utvecklingsprocess där systemets arkitektur först designas, därefter implementeras och utvärderas. En arkitektur valdes då den definierar generella mönster och riktlinjer som ger en tydlig struktur för designen. Arkitekturen utgår från de krav och begränsningar som färjan ställer, där behovet av ett tydligt fail-safe tillstånd och funktionell säkerhet är centrala krav. Målet med arkitekturen är att den ska vara generell och fungera oberoende av vilken typ av körstation som integreras. För att uppnå detta baseras arkitekturen på designmönstret monitor-actuator. Detta mönster valdes då det möjliggör en tydlig separation mellan styrlogik och övervakning, vilket är särskilt lämpligt för system med ett väldefinierat fail-safe tillstånd. Genom att separera dessa ansvarsområden kan fel upptäckas och hanteras utan att påverka den aktiva styrlogiken.

Informationsinsamlingen baseras på vetenskapliga artiklar samt dialog med handledare och representanter från IFM och CStrider. Lösningen implementerades i CODESYS på styrenheten CR710S från IFM. Systemet modellerades som tillståndsmaskin för att hantera övergångar mellan olika körstationer. Implementeringens struktur utgår från arkitekturen och delades upp i komponenter med tydliga ansvarsområden, exempelvis beslutslogik, signalhantering och övervakning. Systemet utvärderades genom testning, där olika scenarier användes för att verifiera att kontrollövergångar skedde korrekt och att systemet uppvisade ett förutsägbart beteende.

2

Teori

2.1 Funktionell säkerhet

Enligt RISE [7] handlar funktionell säkerhet om ett systems säkerhet gällande hur dess komponenter fungerar i förhållande till den indata de tar emot. Att fokusera på denna typ av säkerhet ämnar att undvika skador på människor, miljö eller egendom.

I IEC 61508 [8], den internationella grundstandarden för funktionell säkerhet, är ett centralt krav inom funktionell säkerhet att ett system uppvisar ett förutsägbart beteende. Koncept beskrivna i detta kapitel bidrar tillsammans till att uppnå detta genom att säkerställa ett konsekvent systembeteende - något som möjliggör analys och verifiering i enlighet med standardens krav.

2.2 Determinism inom mjukvara

Determinism inom mjukvara innebär att ett system, givet samma sekvens av indata alltid producerar konsekvent utdata. Denna egenskap resulterar i ett förutsägbart och repeterbart system [9, 10]. För att kunna implementera säkra styrsystem är det viktigt att beakta deterministiska egenskaper. Detta möjliggör analys, testning och verifiering av systemets beteende, som är av stor vikt i säkerhetskritiska system [9, 11]. Mer specifikt leder determinism till att testfall ger samma resultat vid separata exekveringar [9].

2.3 Tillståndsmaskiner

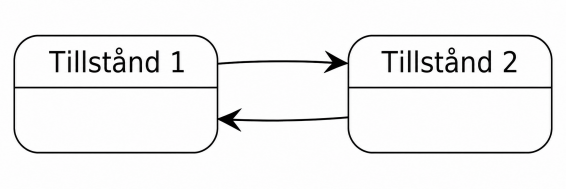
En tillståndsmaskin är en matematisk modell som används för att beskriva ett system som vid varje given tidpunkt befinner sig i enbart ett av flera definierade tillstånd [12]. I modellen övergår systemet mellan dessa tillstånd som respons på inkommande signaler eller händelser, vilket illustreras i Figur 2.1.

För att förstå hur övergångar i säkerhetskritiska system bör hanteras är det relevant att kategorisera de händelser som kan utlösa dem. Enligt Friedenthal et al. [13] kan dessa kategoriseras i fem typer. Fyra av dessa är: signalhändelser som utlöses av inkommande asynkrona meddelanden, tidshändelser vid förfluten tid, förändringshändelser när ett villkor uppfylls, samt anropshändelser vid en begäran om en operation. Den femte typen, anropshändelser vid en begäran om en operation, är inte relevant för detta arbete och behandlas därför inte vidare. Författarna skriver

också att tillståndsdiagram kan användas för att åskådliggöra övergångar mellan körstationer. Dessa illustrationer syftar till att intuitivt skildra ett systems tillstånd och övergångar mellan dessa.

Determinism är ett begrepp som är direkt kopplat till tillståndsmaskiner. I en deterministisk tillståndsmaskin innebär en kombination av tillstånd och indata alltid samma övergång och därmed samma utfall [10]. Varje insignal resulterar alltså i endast ett definierat nästkommande tillstånd.

Vidare kan tillståndsmaskiner användas i säkerhetskritiska system för att hantera farliga situationer. DoD Joint Weapon Safety Working Group [14] menar på att denna modell kan användas för säkerställa att systemet inte hamnar i ett osäkert läge. Arbetsgruppen lyfter att detta uppnås genom att definiera konkreta övergångar, även för situationer där systemet inte uppvisar korrekt beteende.



Figur 2.1: Enkel tillståndsmaskin med övergångar.

2.4 Handshake

I bokkapitlet “Acquisition of Asynchronous Data” beskriver Kaeslin [15] att ett handshake-protokoll en process som ett system använder för att säkerställa kontakt innan data överförs. Denna process möjliggör att båda parter (sändare och mottagare) kommer överens om reglerna för interaktionen. Protokollet bygger på två kontrollsignaler i motsatta riktningar (en begäran och en bekräftelse) som ser till att kommunikationen sker i rätt ordning mellan varandra. Om kommunikation inte fullföljs (svar från mottagare uteblir) inom en definierad tidsram kan processen avbrytas, vilket minskar risken för inkonsekvent kommunikation eller felaktig dataöverföring.

Handshake-protokoll kan användas i kombination med tillståndmaskiner, genom att fördelaktigt definiera varje stadie i kommunikationen som ett tillstånd. Till exempel används denna princip i nätverksprotokollet TCP (Transmission Control Protocol), som är anpassad för kommunikation utan informationsförlust [16]. I TCP används handshake-protokoll även till att skapa en tidsram för interaktioner inom systemet. Om kommunikation inte fullföljs (svar från mottagare uteblir) inom en tydlig tidsram, så kan processen initieras på nytt, för att säkerställa säker drift [16]. Detta exempel visar att handshake-protokoll kan användas för att strukturera kommunikation där parter måste vara synkroniserade.

2.5 Fail-safe

NIST [17] skriver att fail-safe är ett felsäkert sätt att avbryta systemfunktioner och övergå till ett säkert tillstånd. Detta tillstånd syftar till att minimera skada mot människor, miljö eller egendom i en situation där systemet inte uppvisar korrekt beteende. Därav förhindrar ett fail-safe att ett system fortsätter drift i potentiellt farliga tillstånd. I riktlinjer för fartyg med autonom styrning framkommer det att fail-safe-mekanismer ses som en central designprincip för att säkra maritima system [18].

2.6 Monitor-actuator

Monitor-actuator är ett designmönster som bygger på två olika kanaler; en aktuatorkanal (actuation channel) som utför huvudsakliga beräkningar och en övervakningskanal (monitoring channel) som bevakar aktuatorkanalen [11]. Designmönstret beskrivs också som lämpligt för system där det finns ett definierat fail-safe tillstånd [11]. Mer specifikt kan övervakningskanalen, om den upptäcker kritiska fel under sin bevakning, skicka en signal för att aktuatorkanalen ska övergå till detta fail-safe tillstånd. Alltså används detta designmönster för att separera logik och övervakning i säkerhetskritiska system. Genom att separera dessa funktioner kan systemet övervaka sitt eget beteende och identifiera avvikelser mellan förväntad och faktisk funktion [11].

2.7 Riskanalys och FMEA

Failure Mode and Effects Analysis (FMEA) är ett verktyg för att hitta "[...] potentiella fellägen i produkter eller processer och utvärdera deras inverkan på den övergripande kvaliteten" [19]. Denna metod hjälper till att identifiera systemrisker och bedöma deras allvarlighet. När en riskanalys utförs med hjälp av FMEA, så tilldelas risker en svårighetsgrad, förekomst och detekterbarhet [19]. Dessa attribut hjälper utvecklare att ta itu med de mest allvarliga problemen först.

2.8 Övriga koncept

I rapporten nämns även andra koncept i mindre utsträckning. Dessa koncept förklaras i detta delkapitel.

2.8.1 Integrationstester

Integrationstester syftar till att undersöka samspelet mellan olika komponenter i ett system som helhet [20]. Målet med testerna är att identifiera eventuella fel som uppstår i gränssnittet mellan de olika delarna av systemet [20].

2.8.2 Enhetstestning

Octopus Deploy beskriver att enhetstestning handlar om att isolera de minsta delarna av ett system, exempelvis funktioner och metoder, för testning [20]. Hemsidan framhåller att syftet är att testa varje enskild komponent i ett system för att säkerställa att den fungerar som avsett.

2.8.3 Heartbeat

Heartbeat-mekanismer används för att upptäcka fel i ett system där komponenter regelbundet skickar signaler för att indikera att de fortfarande är aktiva [21]. Om en heartbeat inte tas emot inom ett fördefinierat tidsintervall kan den övervakade komponenten misstänkas ha fallerat.

2.8.4 Programmable logic controller

En programmerbar logisk styrenhet (Programmable Logic Controller, PLC) används för att övervaka och styra industriella processer genom programmerbar logik [22]. En PLC samlar indata från anslutna sensorer via ingångar, utvärderar datan genom ett styrprogram och skickar utdata till aktuatorer eller andra enheter.

En central egenskap hos en PLC är att programmet exekveras i en kontinuerlig skanningscykel. Enligt Inductive Automation [22] består en cykel av: inläsning av signaler, exekvering av programlogik och skrivning av utsignaler. Under exekvering bearbetas insignaler och utsignaler bestäms. Eftersom insignaler läses in i början av cykeln så behandlas signalförändringar som uppstår under exekveringen först i nästa cykel [22]. Detta implicerar att ordningen i vilken programlogiken exekveras inom en cykel är av betydelse för systemets beteende.

IFMs hårdvarukomponent CR710S består av både en Standard PLC och en integrerad Safety PLC [23]. Standard PLC:n används för ordinarie styrlogik och kommunikation, medan Safety PLC:n används för säkerhetskritiska funktioner och övervakning. Uppdelningen möjliggör separation mellan funktionell styrning och säkerhetsrelaterad logik, vilket bidrar till ett säkrare och mer robust system.

3

Utvecklingsprocess

3.1 Kravspecifikation

En tydlig kravspecifikation är viktig för att etablera nödvändig funktionalitet hos systemet. Kraven fastställdes genom att granska uppgiftsbeskrivningen och insamlad information. Utöver detta identifierades lämpliga domänspecifika antaganden och begränsningar. Ett antagande var exempelvis att färjan har ett ankare som kan användas som fail-safe. Kravspecifikationen prioriterades med hjälp av ramverket MoSCoW (ska, bör, kan) och presenteras i Tabell 3.1.

ID	**Prioritet**	**Beskrivning**
S-01	SKA	Systemet ska stödja en manuell körstation på färjan
S-02	SKA	Systemet ska stödja en körstation på land
S-03	SKA	Systemet ska stödja övergångar mellan körstationer
S-04	SKA	Beslut inom kontrollsystemet ska loggas
S-05	SKA	Systemet ska verifieras och testas
S-06	SKA	Alla implementerade körstationer ska kunna integreras i samma system
S-07	SKA	En hazardanalys ska genomföras innan implementering av en körstation
B-01	BÖR	Systemet bör stödja styrning via mobilapplikation
B-02	BÖR	Systemet bör stödja standard autopilot
B-03	BÖR	Systemet bör stödja redundans för fjärrkontroll
B-04	BÖR	Systemets design bör inte bero på specifik hårdvara
K-01	KAN	Systemet kan stödja avancerad autopilot

Tabell 3.1: Kravspecifikation skapad via ramverket MoSCoW som prioriterar enskilda krav

3.2 Framtagning av systemarkitektur

Med kravspecifikationen som grund fastställdes en systemarkitektur. Arkitekturen beskriver övergångslogiken och användes som en mall för implementering.
Arkitekturen presenteras enligt arc42 [24], valt för dess tydliga struktur, separation av systemvyer och användarvänliga upplägg. En fullständig beskrivning ges i Bilaga A och sammanfattas i Kapitel 4. Bilagan presenterar systemet i detalj med diagram som täcker både systemets statiska komponenter, relationer och dynamiska beteenden.

3.2.1 Arbetsprocessen bakom arkitekturen

Utvecklingen av arkitekturen förankrades i kravspecifikationen och hade ett tydligt fokus på säkra och pålitliga övergångar mellan körstationer. För att kunna garantera dessa egenskaper var ett iterativt arbete viktigt. Mer specifikt utfördes riskanalyser enligt processen FMEA, med målet att kontinuerligt utvärdera systemet ur ett säkerhetsperspektiv. Denna process repeterades tills systemets risknivå ansågs godtagbar och resulterade i ett levande arkitekturarbete med designlösningar som aktivt mitigerade nyfunna risker. Risken sågs som acceptabel om den angivna konsekvensnivån som anges i Bilaga B inte överskred ett värde på två.

3.2.2 Riskanalysens påverkan på arkitekturen

Riskanalyserna utfördes genom att utvärdera varje enskild interaktion i de framtagna sekvensdiagrammen. På så sätt identifierades möjliga fel, deras orsaker samt konsekvenser för systemets säkerhet. Exempelvis identifierades risken att en övergång skulle ske när en körstation inte är i skick för styrning (SR-07 i Figur B.1). Denna risk resulterade i att en handskakningsprocess med tydliga bekräftelser mellan komponenter introducerades. För att ytterligare säkra systemet i fallet då bekräftelser uteblir (SR-08 i Figur B.1) skapades timeout-mekanismer. För presentation av den slutgiltiga riskanalysen hänvisas läsaren till Bilaga B.

3.3 Implementering av system

Under arbetet prioriterades utvecklingen och verifieringen av en MVP (minimum viable product). Denna MVP definierades genom kraven med prioritet SKA. Utöver detta utvecklades stöd för fler körstationer. Fler än två körstationer krävdes inte för att diskutera frågeställningarna, men var fördelaktiga för att utveckla en mer realistisk lösning.

Säkerhetsmekanismerna testades när alla körstationer redan var på plats. Integrationstester genomfördes däremot, i enlighet med S-06 i Tabell 3.1, för varje enskild implementering av körstationer. Implementeringen sammanfattas i Kapitel 5.

3.4 Testning

För att säkerställa att implementeringen uppfyllde kravspecifikationen och följde designbeslut genomfördes tester. Denna testning bestod av en kombination av enhetstester och integrationstester.

Arkitekturen som är dokumenterad i Kapitel 4 och implementationsbeskrivningen i Kapitel 5 analyserades för att framställa representativa testfall för enhetstestningen. Den centrala logiken ligger i `ControlUnit`, `IntegrityCheck` och `OutputProcessing`. Av denna anledning gjordes primärt enhetstester på dessa komponenter. Testerna utformades genom att skicka in manuell indata för att jämföra dess utdata med det förväntade beteendet hos komponenten. Särskilt fokus lades på att verifiera tillståndsmaskinen och dess olika övergångar mellan körstationerna för att säkerställa korrekt funktionalitet.

Integrationstester genomfördes för att garantera att samtliga komponenter i systemet samverkar på ett korrekt sätt. Integrationstester implementerades utifrån dynamiska vyer i arkitekturen, se Bilaga A.6.1. Testerna genomfördes genom att ge systemet indata och dokumentera utdata. Den utdata som systemet gav verifierades utifrån vad systemet förväntades ge ut.

En sammanställning av de olika testfallen, inklusive insignaler, utsignaler och det förväntade resultatet, finns i Bilaga D.

4

Arkitektur

4.1 Introduktion

Detta kapitel beskriver systemets struktur. Syftet med arkitekturen är att beskriva hur kontrollen för färjan överförs mellan olika körstationer. Den detaljerade arkitekturen finns i Bilaga A.

Övergripande delas systemet upp i körstationer, logik för övergångar mellan dessa, samt övervakning. Körstationerna kan begära kontrollen medan övergångslogiken godkänner eller nekar en sådan förfrågan. Övervakning används för att upptäcka avvikelser i systemets beteende eller hälsa.

Vid implementering installerades systemet på styrenheten CR710S, utvecklad av IFM, men arkitekturen är inte tänkt att vara beroende av specifik hårdvara. Därför behandlas driftsättningen endast kort i Bilaga A.7.

4.2 Kvalitetsmål

Kvalitetsmål inom arkitekturen utgick från systemets säkerhetskritiska natur. Målen som prioriterades var säkerhet, pålitlighet, underhållbarhet och kostnad. Dessa mål presenteras tydligt i Bilaga A.2.2.

Säkerhet prioriterades högst eftersom arbetets syfte är att designa ett system som åstadkommer säkra övergångar mellan körstationer. Därför var ett centralt mål att säkerställa att endast en körstation är aktiv åt gången, samt att systemet kan försättas i ett fail-safe-tillstånd vid riskfyllda situationer. Dessutom var pålitlighet viktigt, men utan att påverka säkerheten negativt. Därför kan systemets tillgänglighet inte garanteras vid kritiska fel, eftersom en övergång till ett fail-safe tillstånd prioriteras.

Underhållbarhet garanterades genom att systemet delades upp i separata komponenter med tydliga ansvar. Uppdelningen gör det lättare att utöka systemet eller ändra existerande komponenter. Dessutom var kostnad viktig för att garantera en ekonomiskt genomförbar lösning för CStrider.

4.3 Val av designmönster

Som grund för arkitekturen användes designmönstret monitor-actuator, vilket beskrivs vidare i Bilaga A.4. Mönstret valdes utifrån tillgången till ett fail-safe, tydlig modularitet och målet att minimera systemets kostnad. Gällande kostnad övervägdes andra designmönster såsom triple modular redundancy och active-passive standby. Dessa mönster har hög hårdvaruredundans [11] och valdes bort för att minimera kostnaden av systemet.

I arkitekturen hanterar `ActuationChannel` logiken för övergångar, medan `MonitoringChannel` övervakar denna logik och kan initiera fail-safe vid kritiska fel. Exempelvis kan `MonitoringChannel` genom ett heartbeat avgöra när `ActuationChannel` fysiskt slutat fungera. Denna uppdelning bidrar till modulariteten, eftersom styr- och övervakningslogik inte blandas.

4.4 Begränsningar och avgränsningar

Systemarkitekturens begränsningar och avgränsningar beskrivs i Bilagan A.3. Lösningen fokuserar på funktionell säkerhet och cybersäkerhet behandlas inte. Dessutom behandlar arbetet mjukvarulogik och därför hanteras inte realtidsaspekter eller återkoppling från aktuatorer.

4.5 Viktiga designbeslut

Ett viktigt beslut var att centralisera övergångslogiken i `ControlUnit`. Detta stärker systemets determinism genom att undvika flera olika komponenter som fattar motstridiga beslut. Därför ansvarar `ControlUnit` ensamt för att avgöra när övergångar kan genomföras. Utöver detta används komponenten `IntegrityCheck` för att validera insignaler innan de når `ControlUnit`.

Ytterligare ett arkitekturbeslut var att modellera systemets tillstånd som en tillståndsmaskin. Utifrån detta designval beskrevs tillstånden som en aktiv körstation eller faserna i en övergång. För att hantera flera samtidiga övergångar utformades en fast prioritetsordning för körstationerna. Vidare infördes en `DecisionLogger` som sparar information om pågående övergång, aktiv station, tidsangivelser, föregående aktiv station, samt de fall då en förfrågan har nekats.

4.6 Systemets dynamiska beteende

Vid en vanlig övergång skickas en kontrollbegäran av en körstation. Komponenten `IntegrityCheck` kontrollerar signalens giltighet och `ControlUnit` bedömer om för-

frågan ska godkännas. Efteråt signalerar `ControlUnit` att den aktiva körstationen ska släppa kontrollen samtidigt som att den nya ska aktiveras. Övergången kan först genomföras när den tidigare körstationen släppt kontrollen och den nya körstationen bekräftat sin tillgänglighet.

Systemarkitekturen kan också hantera flera samtidiga förfrågningar. Om flera körstationer skickar en övergångsbegäran inom samma programcykel väljer `ControlUnit` den station med högst prioritet. Denna mekanism åskådliggörs i Figur A.6.2.

För att motverka situationer där en övergång genomförs trots att en körstation är otillgänglig, eller inte släpper kontrollen, används timeout-mekanismer. Om en timeout sker avbryts övergången och felet sparas i `DecisionLogger`. Denna process presenteras detaljerat i figurerna A.6.3 och A.6.4. Systemets beteende beskrivs mer detaljerat i den dynamiska vyn i Bilaga A.6, där olika scenarion illustreras genom sekvensdiagram.

5

Implementering

5.1 Terminologi

Följande termer används genomgående i detta kapitel.

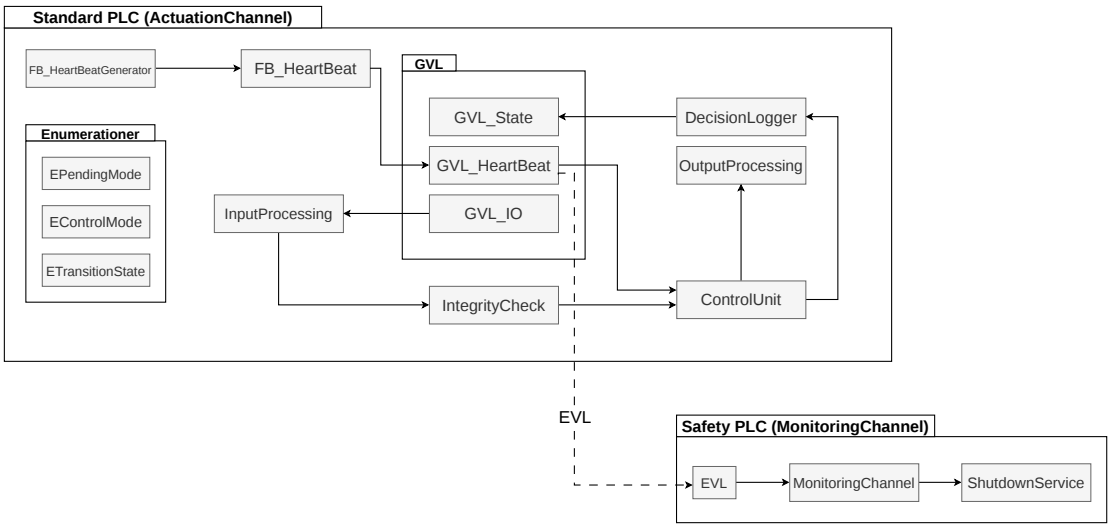
Term	**Förklaring**
Funktionsblock	En återanvändbar programenhet i CODESYS som kapslat in logik och tillstånd.
Global Variable List (GVL)	En lista för att strukturera och dela variabler mellan systemets funktionsblock.
ENUM	Enumeration är en datatyp som definierar en uppsättning av namngivna konstanter.
`RejectionReason`	Variabel som lagrar orsaken till avvisad begäran om kontrollövertag.
`RejectedRequestFlag`	Boolesk variabel som indikerar att en begäran har avvisats under den aktuella cykeln.
`MaxAckTime`	Maximal tillåten tid för att ta emot bekräftelse från den väntande station innan övergången avbryts.
`PLC_PRG`	Huvudprogram som anropar systemets funktionsblock i en fast sekvens varje skanningscykel.

Tabell 5.1: Förekommande termer inom implementeringen.

5.2 Översikt

Implementeringen baserades på den framtagna arkitekturen, där varje block ansvarar för en avgränsad del av systemets logik. `ActuationChannel`, som beskrivs i Kapitel 4, är placerad på Standard PLC och ansvarar för systemets huvudsakliga beslutslogik, medan `MonitoringChannel`, implementerad i Safety PLC, ansvarar för övervakning av systemets beteende. Uppdelningen mellan Standard PLC och Safety PLC illustreras närmare i Bilaga A.7. Samtliga funktionsblock som beskrivs i resterande avsnitt av kapitlet är installerade på Standard PLC, med undantag för blocken `MonitoringChannel` och `ShutdownService`.

`PLC_PRG` utgör systemets huvudprogram och ansvarar för att anropa samtliga funktionsblock i en fast sekvens varje skanningscykel. Sekvensen inleds med inläsning av hårdvarusignaler och avslutas med utskrift till aktuatorer. Däremellan hanteras heartbeat-övervakning, signalvalidering och beslutslogik, vilket illustreras i Figur 5.2. Denna ordning garanterar att `ControlUnit` aldrig tar emot indata som inte validerats under samma cykel. Systemets funktionsblock och deras inbördes relationer illustreras i Figur 5.1.



Figur 5.1: Komponentdiagram över systemet.

För att representera systemets möjliga tillstånd används enumerationerna `EControlMode`, `EPendingMode` och `ETransitionState`, vars värden presenteras i Tabell 5.2–5.4. Variabler som delas mellan flera funktionsblock samlas i tre GVL-block. `GVL_IO` innehåller insignaler från körstationerna, `GVL_HeartBeat` samlar heartbeat-signaler och tillhörande statusvariabler, och `GVL_State` innehåller systemets aktuella tillstånd samt statusvariabler, se Bilaga C.4.

Värde	Förklaring
INIT	Uppstartsläge, systemet inväntar den första giltiga begäran om övertag.
MANUAL	Manuell körstation är aktiv.
REMOTE	Fjärrstyrd körstation är aktiv.
AUTO	Autonom körstation är aktiv.
REMOTE_MOBILE	Mobil fjärrstyrd körstation är aktiv.
FAILSAFE	Terminalt feltillstånd, ingen körstation är aktiv.

Tabell 5.2: Värden i EControlMode

Värde	Förklaring
NONE	Ingen körstation väntar på att ta över kontrollen.
PENDING_MANUAL	Manuell körstation väntar på att ta över kontrollen.
PENDING_REMOTE	Fjärrstyrd körstation väntar på att ta över kontrollen.
PENDING_AUTO	Autonom körstation väntar på att ta över kontrollen.
PENDING_REMOTE_MOBILE	Mobil fjärrstyrd körstation väntar på att ta över kontrollen.

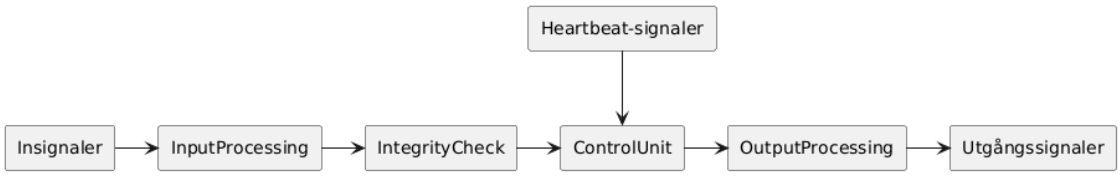
Tabell 5.3: Värden i EPendingMode

Värde	Förklaring
INIT	Uppstartsläge, systemet inväntar den första giltiga begäran.
STABLE	En körstation är aktiv, inga övergångar pågår.
PRE_TAKEOVER	Inkommande begäran om övertag valideras.
VALIDATE_TAKEOVER	Handshake-protokoll genomförs mellan körstationer.
FAILSAFE	Terminalt feltillstånd, inga övergångar tillåts.

Tabell 5.4: Värden i ETransitionState

5.3 Signalhantering

Varje körstation kommunicerar med systemet genom signaler för begäran om övertag, bekräftelse på beredskap, frisläppning samt en heartbeat-signal som indikerar att stationen är aktiv och tillgänglig. Signalhanteringen är uppdelad i tre funktionsblock som tillsammans bildar ett flöde från hårdvaruingångar till utgångssignaler. `InputProcessing` utgör ett abstraktionslager mellan hårdvaruingångarna och systemets interna logik, där signaler från de fysiska ingångarna översätts till logiska signalnamn och vidarebefordras till `IntegrityCheck` för validering, se Bilaga C.1. `IntegrityCheck` kontrollerar att kombinationerna av signalerna från körstationerna är giltiga innan de når `ControlUnit`, se Bilaga C.2. Blocket identifierar ogiltiga kombinationer av signaler från en körstation, exempelvis en samtidig begäran om övertag och frisläppning, eller en samtidig bekräftelse och frisläppning. `OutputProcessing` översätter resultat från ControlUnit till utgångssignaler, se Bilaga C.3. I tillstånden `INIT` och `FAILSAFE` sätts samtliga utgångar till `FALSE`, med undantag för heartbeat-statusar som presenteras oberoende av systemtillstånd. Ett blockdiagram över signalflödet mellan dessa block illustreras i Figur 5.2.



Figur 5.2: Blockdiagram av signalflödet mellan funktionsblocken.

5.4 Tillståndsmaskinen och heartbeat-övervakning

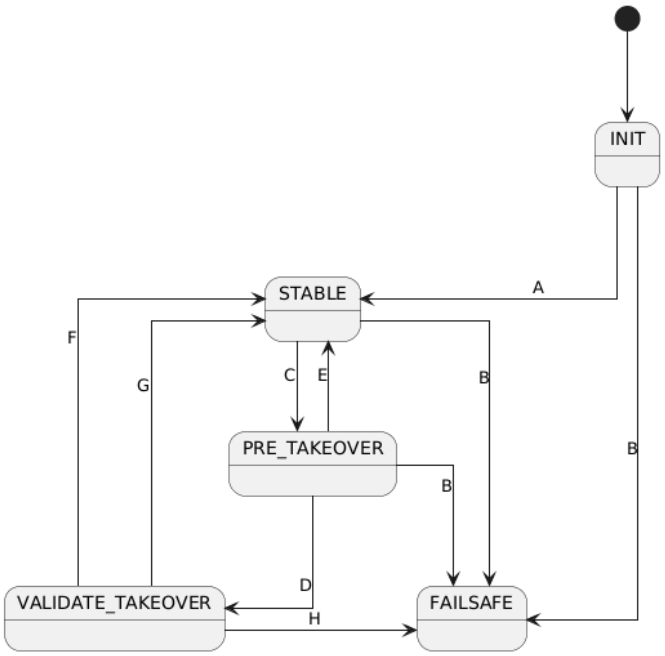
Systemets kärnlogik är implementerad i `ControlUnit`, som modellerar övergångar mellan körstationer som en tillståndsmaskin med fem tillstånd, se Bilaga C.5. Övergångarna utlöses av de händelsetyper som beskrivs i Avsnitt 2.3, däribland signalevents vid inkommande begäran, tidshändelser vid timeout och förändringshändelser när heartbeat-villkor inte längre uppfylls. Varje begäran om övertag detekteras via flankdetektering. `FAILSAFE` är ett terminalt tillstånd från vilket inga övergångar tillåts och återgång till normal drift kräver omstart av systemet.

Innan tillståndsmaskinen exekveras genomförs två säkerhetskontroller varje skanningscykel. En kontroll verifierar att den aktiva körstationens heartbeat-signal är aktiv, medan den andra kontrollerar att insignalerna har godkänts av `IntegrityCheck` via variabeln `Valid`. Vid upptäckt av fel i kontrollerna avbryts exekveringen omedelbart och systemet försätts till `FAILSAFE`, oberoende av aktuellt tillstånd.

Heartbeat-övervakningen hanteras av funktionsblocken `FB_HeartBeatGenerator` och `FB_HeartBeat`, se Bilaga C.6. `FB_HeartBeatGenerator` växlar en boolesk signal med

ett intervall på 200ms. `FB_HeartBeat` övervakar att signalen fortsätter att växla och sätter utgångsvariabeln `xOk` till `FALSE` om ingen växling detekteras inom 500ms.

Tillståndsmaskinens övergångar illustreras i Figur 5.3 och förklaras i Tabell 5.5. Vid uppstart befinner sig systemet i `INIT` och inväntar den första giltiga begäran om övertag. Därefter valideras inkommande begäran i `PRE_TAKEOVER` innan övergången får fortskrida. Om begäran godkänns genomförs ett handshake-protokoll i `VALIDATE_TAKEOVER` innan kontrollen slutligen överförs. Övergång B kan utlösas från samtliga tillstånd utom `VALIDATE_TAKEOVER`. I handshake-fasen hanteras istället heartbeat-förlust av övergångarna G och H.



Figur 5.3: Tillståndsdiagram för ControlUnit

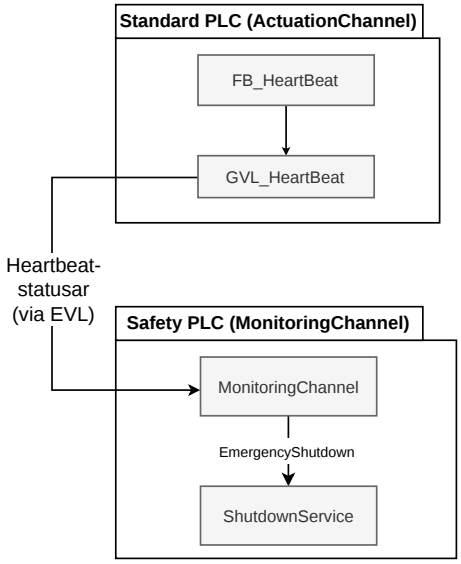
Övergång	**Beskrivning**
A	Giltig begäran om övertag, aktivt heartbeat.
B	Heartbeat-förlust hos aktiv station eller ogiltiga insignaler.
C	Begäran om övertag inkommen.
D	Begärande station ej aktiv, aktivt heartbeat.
E	Begäran avvisad.
F	Övergång slutförd, handshake genomförd.
G	Timeout (`MaxAckTime`) eller heartbeat-förlust under handshake.
H	Heartbeat-förlust eller ogiltig signal under handshake.

Tabell 5.5: Övergångar i tillståndsmaskinen

5.5 Tillståndsregistrering

Systemets spårbarhet hanteras av `DecisionLogger`, som registrerar förändringar i aktiv station, väntande station och systemtillstånd genom att varje cykel jämföra aktuella värden med föregående cykel. Vid en förändring uppdateras motsvarande variabler i `GVL_State`, där även den tidigare aktiva stationen sparas i `GVL_State.LastActiveBeforeTransition`. Avvisade begäranden registreras med tillhörande orsak i `GVL_State.RejectionReason` och `GVL_State.RejectedRequestFlag`, se Bilaga C.7.

Övervakningen av systemet på Safety PLC hanteras av `MonitoringChannel` och `ShutdownService`. Heartbeat-statusar från Standard PLC överförs till Safety PLC via en `Exchange Variable List` (EVL), som utgör kommunikationsmekanismen mellan de två PLC-enheterna, vilket illustreras i Figur 5.4. `MonitoringChannel` tar emot dessa statusar och utlöser `EmergencyShutdown` om samtliga körstationer förlorar sin heartbeat-signal samtidigt. `EmergencyShutdown` är skilt från tillståndet `FAILSAFE` i `ControlUnit`. `FAILSAFE` utlöses av enskilda fel inom ActuationChannel, såsom förlorad heartbeat eller ogiltiga signaler, medan `EmergencyShutdown` enbart utlöses vid ett totalt bortfall av samtliga körstationer. De två mekanismerna opererar därmed oberoende av varandra på separata PLC-enheter. `ShutdownService` tar emot `EmergencyShutdown` och sätter en intern variabel till `TRUE`, som förblir `TRUE` tills systemet startas om, se Bilaga C.8.



Figur 5.4: Kommunikation mellan Standard PLC och Safety PLC via EVL

6
Resultat

6.1 Resultat av testscenarion

De genomförda enhets- och integrationstesterna gav det förväntade resultatet och verifierade att implementeringen följde den avsedda funktionaliteten beskriven i Bilaga A. Detta inkluderar korrekt hantering av kontrollövergångar vid uppstart och under drift, signalvalidering samt fail-safe-beteende.

Enhetstestningen omfattade komponenterna `ControlUnit`, `IntegrityCheck` och `OutputProcessing`. Testningen av `ControlUnit` visade att systemet korrekt hanterade uppstart och överföring av kontroll mellan de olika körstationerna. Byten från uppstart till en fungerande körstation genomfördes korrekt och systemet övergick till fail-safe vid ogiltiga signaler eller heartbeat-förlust. Vidare visade testningen att den tidigare körstationen bibehöll kontrollen vid uteblivna godkännanden eller heartbeat-förlust i den övertagande körstationen, se testerna T-01-T-10 i Tabell D.1 i Bilaga D.

Testningen av `IntegrityCheck` visade att valideringen av giltiga insignaler passerade utan att bli modifierade. Resultatet visade även att `IntegrityChecks` validering inte gick igenom vid ogiltiga signalkombinationer inom en och samma körstation, exempelvis en samtidig begäran och frisläppning, eller en samtidig bekräftelse och frisläppning, se Tabell D.2 i Bilaga D. Testningen av `OutputProcessing` visade att interna tillstånd och statusvariabler stämde överens med motsvarande utsignaler, se Tabell D.3 i Bilaga D.

Integrationstesterna visade att systemets komponentinteraktioner följde arkitekturens dynamiska vyer. Mer specifikt visade testerna att systemet kunde genomföra byten mellan körstationer korrekt om en handshake hade skett. En körstation fick endast kontroll om den gett bekräftelse. Testerna visade också att ogiltiga signaler och heartbeat-förluster ledde till att systemet övergick till fail-safe, se Tabell D.4 i Bilaga D. En sammanställning av de olika testfallen, inklusive insignaler, utsignaler och det förväntade resultatet, återfinns i Bilaga D.

7

Diskussion

7.1 Uppfyllelse av syfte och krav

Systemet visar att syftet har uppfyllts genom att säkerställa deterministiska övergångar där enbart en körstation är aktiv åt gången. Ensam kontroll garanteras genom att modellera systemets tillstånd som en tillståndsmaskin där övergångar enbart kan ske om den aktiva stationen släpper kontrollen samtidigt som den nya bekräftar sin tillgänglighet. Detta besvarar den första frågeställningen beskriven i Avsnitt 1.3 om hur entydig kontroll säkerställs. Även den andra frågeställningen har bemötts när det kommer till förhindrande av ogiltiga övergångar, något som ytterligare stärks av signalvalideringen i `IntegrityCheck` och den flerstegiga valideringen i `PRE_TAKEOVER`. Centraliseringen av övergångslogiken i `ControlUnit` säkerställer därtill det deterministiska beteende som beskrivs i Avsnitt 2.2, där samma indata alltid producerar samma utdata.

Kraven S-01 och S-02 uppfylls genom att lösningen stödjer både manuella och fjärrstyrda körstationer. Dessa framgår i Tabell 3.1 och spårbarhet till implementeringen finns i Tabell 5.2. Vidare uppfylldes krav S-03, genom tillämpningen av en tillståndsmaskin för övergångar, vilket illustreras i Figur 5.3 och Tabell 5.5. Att övergångar fungerar korrekt stöds av enhetstesterna T-01 och T-10, som verifierar att systemet korrekt hanterar uppstart och byten mellan körstationer. Integrationstesterna IT-01 till IT-05 visade därtill att ogiltiga signaler och heartbeat-förluster korrekt utlöser `FAILSAFE`.

Ytterligare uppfylls krav S-04 genom implementeringen av `DecisionLogger`, se Bilaga A.4 och Avsnitt 5.5. Dessutom lever systemet upp till S-05 gällande testning genom utförandet av de enhets- och integrationstester som beskrivs i Avsnitt 6.1.

Krav S-06, som behandlar modularitet, uppfylls genom att arkitekturen och implementeringen utformas så att körstationer generaliseras. Därigenom möjliggörs användning av samma övergångslogik för flera olika typer, vilket besvarar den fjärde frågeställningen. Utöver detta uppfylls S-07 genom den riskanalys som beskrivs i Avsnitt 3.2 och Bilaga B. Detta adresserar den tredje frågeställningen om iterativ dokumentation och utvärdering.

Kraven B-02 och K-01 som behandlar standard- och avancerad autopilot implementerades inte i sin helhet. Detta berodde på att dessa, i praktiken, rör kommunikationsvägar snarare än styrlogik, vilket ligger utanför arbetets fokus. Styrlogiken för dessa körstationer följer dock samma principer som övriga implementerade körstationer. Detta innebär att de kan integreras i systemet utan ändringar i den

grundläggande övergångslogiken.

7.2 Systemets styrkor

En grundläggande styrka i arkitekturen är tillämpningen av monitor-actuator-mönstret, vilket skapar en tydlig ansvarsfördelning mellan styrlogik och övervakning. Eftersom de två kanalerna är implementerade på separata PLC-enheter inom CR710S, se Bilaga A.7 minskas risken för mjukvarufel som propagerar från `ActuationChannel` till `MonitoringChannel`. Den fysiska isolationen stärker därmed systemets säkerhetsegenskaper utöver vad en rent logisk separation hade erbjudit.

Arkitekturens fail-safe-princip innebär att systemet vid fel försätts i ett säkert tillstånd istället för att fortsätta drift under farliga förhållanden. Detta är ett medvetet val som prioriterar säkerhet över tillgänglighet och säkerställer att ett felaktigt systembeteende inte kan leda till okontrollerade situationer.

En ytterligare styrka i implementeringen är den flerstegiga valideringen, där signaler och övergångsbegäranden valideras i flera separata steg. `IntegrityCheck` validerar signalkombinationer innan de når `ControlUnit`, medan `PRE_TAKEOVER` och `VALIDATE_TAKEOVER` säkerställer att övergångar endast genomförs under definierade villkor. Denna uppdelning minskar risken för att ogiltiga tillstånd når systemets kärnlogik. Heartbeat-mekanismen bidrar dessutom till kontinuerlig övervakning av körstationernas tillgänglighet och säkerställer att systemet omedelbart övergår till `FAILSAFE` vid förlust av en aktiv stations heartbeat-signal.

Därutöver är arkitekturen hårdvaruagnostisk, vilket innebär att styrlogiken inte är bunden till specifik hårdvara och överensstämmer med Avsnitt 1.4. Detta möjliggör att systemet kan integreras med annan hårdvara i framtida projekt utan att den grundläggande logiken behöver förändras, vilket underlättar en eventuell driftsättning av systemet i en verklig miljö.

7.3 Systemets begränsningar

I Avsnitt 4.5 framgår det att arkitekturen centraliserar all övergångslogik i `ControlUnit`, vilket innebär att ett fel i denna mjukvarukomponent försätter systemet i fail-safe-tillstånd och normal drift inte kan återupptas utan omstart. Komponenten utgör därmed en SPOF för systemets tillgänglighet. Detta är en medveten avvägning i enlighet med Avsnitt 4.2, där säkerhet prioriteras över tillgänglighet till förmån för en robust och kostnadseffektiv lösning.

Ytterligare en begränsning är att `MonitoringChannel` inte övervakas av någon annan komponent och enbart övervakar körstationernas heartbeat-signaler utan att verifiera att övergångslogiken i `ActuationChannel` beter sig korrekt. Under arbetet

uppmärksammades att detta innebär att såväl felaktigt beteende i övervakningskanalen som felaktiga beslut i `ActuationChannel` riskerar att förbli oupptäckta, förutsatt att komponenterna fortsätter att fungera normalt. Separationen mellan de två kanalerna ökar därmed inte nödvändigtvis systemets förmåga att detektera logiska fel. En watchdog, designmönstret 3LSM, vilket är en vidareutveckling av monitor-actuator [11], eller en redundant implementering av övergångslogiken i `MonitoringChannel` hade kunnat adressera dessa begränsningar i framtida arbete.

Därutöver har `OutputActuatorComparator`, som enligt arkitekturen ska jämföra `ActuationChannels` beslut med aktuatorernas faktiska beteende, inte implementerats. Detta innebär att fullständig övervakning enligt monitor-actuator-mönstret inte uppnås i den nuvarande lösningen, och att avvikelser mellan styrlogikens beslut och aktuatorernas faktiska beteende inte kan detekteras.

7.4 Testningens styrkor

En styrka med testningen var att både normal funktionalitet och kritiska felfall testades. Detta bidrog till att systemets beteende kunde verifieras under olika driftförhållanden, vilket ökade systemets tillförlitlighet. Testningen bidrog även till identifieringen av flera extremfall. Exempelvis upptäcktes det under enhetstestningen att systemet inte kunde genomgå en omstart under drift vilket bidrog till att ett byte tillbaka till det initiala tillståndet inte var möjlig. Testningen användes därmed inte enbart för verifiering av systemets funktionalitet utan även för validering av systemets tillståndshantering och beteende vid dynamiska driftförhållanden. Detta i sin tur ökade implementationens robusthet.

Fortsättningsvis genomfördes testningen stegvis och deterministiskt i en simulerad miljö. CODESYS utvecklingsmiljö möjliggjorde observation av individuella PLC-cyklar, vilket underlättade analys av hur systemets tillstånd förändrades mellan dessa. Detta gjorde det möjligt att reproducera exakta testscenarion på ett kontrollerat sätt. Utöver detta bidrog integrationstestningen till att säkerställa att implementeringen stämde överens med den tidigare definierade arkitekturen. Mer specifikt kunde individuella PLC-cyklar observeras för att verifiera att systemets komponentinteraktioner var konsekventa med arkitekturens dynamiska vyer.

7.5 Testningens begränsningar

Problem identifierades gällande icke-kompatibla versioner av automatiserade testningsramverk. Arbetet begränsades av att CStrider använde en äldre version av CODESYS. Detta resulterade i att testningsramverket CoUnit inte kunde användas. Samma problem uppstod med CODESYS inbyggda testningsverktyg eftersom det inte heller kunde användas med den rådande versionen av CODESYS. Därav bestod testningen av manuella tester. Fördelen med att göra tester manuellt var att

det gav exakt kontroll över varje steg. Denna process gjorde det enklare att identifiera var problematik uppstod.

Begränsningar uppstod även gällande testningen av säkerhets-PLC:n då det inte gick att simulera testfall för den. Detta medförde att de separata komponenternas logik inte gick att verifiera.

Under integrationstestningen identifierades begränsningen att flera PLC:er inte kunde simuleras samtidigt. Detta medförde att interaktionen mellan standard- och säkerhets-PLC:n inte kunde testas. För att utföra mer omfattande integrationstestning skulle en annan testmiljö med stöd för simulering av flera PLC:er samtidigt kunna användas. Alternativt skulle testning på fysisk hårdvara där båda PLC:erna används kunna komplettera de nuvarande integrationstesterna.

7.6 Framtida arbete

Följande avsnitt beskriver begränsningar och områden för framtida utveckling, med fokus på driftsättning under verkliga förhållanden.

7.6.1 Cybersäkerhet

Den nuvarande implementationen behandlar enbart funktionell säkerhet och saknar skydd mot antagonistiska hot. Systemet autentiserar inte inkommande styrsignaler, vilket innebär att en obehörig aktör med åtkomst till kommunikationskanalen potentiellt kan injicera falska övergångsbegäranden eller bekräftelser och initiera okontrollerade kontrollövergångar. Säkerhetsrisker bör beaktas redan i design- och konceptfasen, och industriella styrsystem saknar generellt autentisering och kryptering, vilket gör dem sårbara för signalmanipulering [25]. Framtida utvecklingsprojekt bör därför inkludera autentiseringsmekanismer samt kryptering av styrsignaler för att säkerställa integriteten hos kommunikationen mellan körstationerna.

Bristande tillgänglighet utgör ytterligare en relevant säkerhetsrisk som bör adresseras i framtida arbete. Antagonistiska angrepp mot styrsystemet, exempelvis överbelastningsattacker som syftar till att hindra systemet från att behandla legitima styrsignaler, kan försätta det i ett tillstånd där kontrollövergångar blockeras, vilket i en verklig implementering kan hindra behöriga körstationer från att ta kontroll i kritiska situationer [26, 27].

7.6.2 Realtidsaspekter

Realtidsaspekter har inte behandlats inom ramen för detta projekt då implementeringen genomfördes i en simulerad miljö. I en simulerad miljö utan fysiska nätverk eller aktuatorer uppstår inte de kommunikationsfördröjningar och tidskritiska

beroenden som finns i ett verkligt driftsatt system. I ett driftsatt system är det dock nödvändigt att säkerställa att systemet reagerar på händelser inom garanterade tidsgränser. PLC:er är generellt tidskritiska och kräver deterministiska svar, där acceptabla nivåer av fördröjning är beroende av den specifika installationen [28]. I den nuvarande implementationen är tidskritiska parametrar som `MaxAckTime`, heartbeat-intervall och skanningscykelns längd satta till godtyckliga värden som inte har baserats på uppmätta kommunikationsfördröjningar. Kommande utvecklingsprojekt bör därför inkludera analys av maximal exekveringstid samt validering av dessa parametrar mot faktiska driftsförhållanden.

7.6.3 Aktuatorer och andra återkopplingssignaler

I den nuvarande implementeringen hanteras inte aktuatorer eller återkopplingssignaler från fysisk hårdvara. Detta var ett medvetet avgränsningsbeslut som möjliggjorde att styrlogiken kunde utvecklas och verifieras oberoende av specifik hårdvara. Framtida utveckling av systemet bör integrera återkopplingssignaler från aktuatorer för att möjliggöra fullständig övervakning enligt monitor-actuator-mönstret. Eftersom implementeringen är hårdvaruagnostisk, och hårdvaruspecifik logik är isolerad i `InputProcessing` och `OutputProcessing`, kan återkopplingssignaler integreras utan att den grundläggande övergångslogiken i `ControlUnit` behöver förändras.

7.6.4 Minimitid mellan övergångar

Den nuvarande implementationen saknar ett krav på minimitid mellan kontrollövergångar. Detta innebär att systemet teoretiskt kan genomföra upprepade övergångar i snabb följd, vilket kan skapa instabilitet i styrningen. I ett driftsatt system bör en minimitid definieras och upprätthållas mellan övergångar för att förhindra detta. Framtida arbete bör därför inkludera definition och implementering av en sådan parameter, baserad på de faktiska tidskraven som det specifika fartygets styrsystem ställer.

7.6.5 Testning av inbäddat system

Testningen som utförts täcker de främsta delarna av systemet, men testar inte hur själva systemet fungerar i sin helhet inbäddat i en marin båt. Därför hade det varit intressant att undersöka hur mjukvaran och hårdvaran samspelar i ett komplett system. Framtida arbete skulle därför kunna inkludera tester av systemet inbäddat i hårdvara som används på båten. Detta skulle exempelvis kunna vara styrenheten CR710S från IFM.

7.7 Samhälleliga och etiska aspekter

Systemet är avsett för säkerhetskritiska tillämpningar där fel kan få allvarliga konsekvenser för passagerare. En felaktigt hanterad kontrollövergång kan i en verklig implementation leda till att ingen eller flera körstationer har kontroll över färjan samtidigt, vilket i värsta fall kan resultera i allvarliga olyckor. Eftersom implementationen genomfördes i en simulerad miljö har ingen människa utsatts för risk under projektets gång, men de säkerhetsaspekter som beaktats speglar de krav som skulle gälla i ett verkligt driftsatt system.

En central etisk fråga är hur robust systemet är mot obehörig åtkomst och manipulation. Ett system som oavsiktligt byter körstation eller fastnar i ett odefinierat tillstånd till följd av obehörig manipulation kan leda till förlust av styrning och utsätta passagerare för fara. Den nuvarande implementationen hanterar funktionell säkerhet men saknar skydd mot antagonistiska hot, vilket är en medveten avgränsning som tidigare har diskuterats i Delkapitel 7.6.1.

Ur ett samhällsperspektiv är förtroendet för autonoma transportlösningar direkt kopplat till hur säkra och tillförlitliga systemen upplevs vara. Ur ett samhällsperspektiv är förtroendet för autonoma transportlösningar direkt kopplat till systemets upplevda säkerhet och tillförlitlighet [1]. Utmaningar kopplade till säkra kontrollövergångar är inte unika för maritima system utan utgör ett generellt problem inom autonom transport. Inom självkörande fordon har exempelvis övergångar mellan manuell och autonom styrning identifierats som säkerhetskritiska moment, där ansvarsfördelningen vid en felaktig övergång mellan fordonets tillverkare, systemet och föraren är oklar [29]. Liknande ansvarsfrågor är relevanta för autonoma passagerarfärjor och bör adresseras inför en verklig driftsättning.

En ytterligare etisk dimension är frågan om ansvar vid fel. I ett system där beslut om kontrollövergångar fattas automatiskt av styrlogiken uppstår frågan om vem som bär ansvaret om ett felaktigt beslut leder till en olycka. Denna fråga är inte löst inom ramen för detta projekt, men är viktig att bemöta inför en verklig driftsättning.

8

Slutsatser

Detta arbete har resulterat i en styrlogik för säker växling mellan körstationer hos en autonom passagerarfärja. Systemet designades med utgångspunkt i designmönstret monitor-actuator och implementerades i CODESYS och exekverades på styrenheten CR710S. Samtliga krav med prioritet SKA uppfylldes, däribland stöd för manuell och fjärrstyrd körstation, loggning av beslut samt integration av körstationer i ett gemensamt system. Arbetet resulterade även i en arkitektur som fungerar som en generell mall för implementering av olika typer av körstationer och är hårdvaruagnostisk.

Entydig kontroll säkerställs genom centraliseringen av övergångslogiken i `ControlUnit`, där tillståndsmaskinen garanterar att enbart en körstation kan vara aktiv åt gången. Ogiltiga övergångar förhindras genom signalvalidering i `IntegrityCheck`, validering med flera steg i övergångsprocessen samt timeout-mekanismer som avbryter ofullständiga övergångar. Lösningen utvärderades genom enhets- och integrationstester, vilka visade på korrekt beteende i samtliga testade scenarion inklusive felfall som heartbeat-förlust och ogiltiga signaler.

Systemets modularitet möjliggör integration av nya körstationer utan ändringar i den grundläggande styrlogiken. En central begränsning är att tillgängligheten upphör vid kritiska fel, då säkerhet prioriteras över tillgänglighet. Fullständig verifiering mot verkliga driftsförhållanden kvarstår som framtida arbete.

Referenser

[1] R. R. Negenborn *et al.*, "Autonomous ships are on the horizon: here's what we need to know," *Nature*, vol. 615, no. 7950, pp. 30–33, Mar. 2023.

[2] W. Zhang, C. Mu, Y. Wang, X. Yang, X. Zhou, X. Meng, and L. Li, "Exploring the mass-doa2 control-switching mechanism: Results from the autonomous ship guidelines review and expert survey," *Journal of Navigation*, vol. 77, no. 2, p. 276–305, 2024.

[3] A. Patlins, "Improvement of sustainability definition facilitating sustainable development of public transport system," *Procedia Engineering*, vol. 192, pp. 659–664, 2017, 12th international scientific conference of young scientists on sustainable, modern and safe transport. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877705817326607

[4] CStrider, "Innovation on water," 2023, accessed: Apr. 29, 2026. [Online]. Available: https://www.cstrider.com/

[5] NASA, "5.3 product verification," n.d., accessed: Apr. 29, 2026. [Online]. Available: https://www.nasa.gov/reference/5-3-product-verification/

[6] X. Yang, T. Zhou, X. Zhou, W. Zhang, C. Mu, and S. Xu, "A framework to identify failure scenarios in the control mode transition process for autonomous ships with dynamic autonomy," *Ocean & Coastal Management*, vol. 249, p. 107003, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0964569123005288

[7] RISE, "Functional safety," 2022, accessed: Apr. 29, 2026. [Online]. Available: https://www.ri.se/en/total-defence-and-crisis-preparedness/risk-and-security/expertise/functional-safety

[8] International Electrotechnical Commission, "Iec 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems," 2010, international standard.

[9] R. Mudduluru, J. Waataja, S. Millstein, and M. D. Ernst, "Icse 2021, proceedings of the 43rd international conference on software engineering," in *Verifying Determinism in Sequential Programs*, 2020, pp. 37–49.

[10] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, 3rd ed. Pearson, 2006.

[11] A. Armoush, "Design patterns for safety-critical embedded systems," 01 2008.

[12] Y. Hu, Y. Dan, G. Shao, S. Yu, J. Han, and B. Li, "Research on full-range autonomous navigation decision making of ships based on finite state machines," in *2023 7th International Conference on Transportation Information and Safety (ICTIS)*, 2023, pp. 94–99.

[13] S. Friedenthal, A. Moore, and R. Steiner, *A Practical Guide to SysML: The Systems Modeling Language*, 3rd ed. Morgan Kaufmann, 2015.

[14] International System Safety Society, "Guidance to perform modes, states and transitions hazard analysis," Jun. 2024, accessed: 2026-05-18. [On-

line]. Available: https://isss-tvc.org/Guidance-Modes_States_Transitions_Hazard_Analysis_2024-06-22.pdf

[15] "Chapter 8 - acquisition of asynchronous data," in *Top-Down Digital VLSI Design*, H. Kaeslin, Ed. Boston: Morgan Kaufmann, 2015, pp. 445–472. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128007303000083

[16] J. Postel, "Transmission control protocol," https://www.rfc-editor.org/rfc/rfc793, 1981, accessed: Apr. 29, 2026.

[17] NIST, "Fail safe," accessed: Apr. 29, 2026. [Online]. Available: https://csrc.nist.gov/glossary/term/fail_safe

[18] International Maritime Organization, "Msc 108/4: Report of the mass correspondence group," International Maritime Organization, Tech. Rep., 2024, accessed: 2026-05-18. [Online]. Available: https://rs-class.org/upload/iblock/01f/01f10869d178a34e3d02234862c56fad.pdf

[19] Siemens, "Failure mode and effects analysis fmea," accessed: Apr. 29, 2026. [Online]. Available: https://www.siemens.com/sv-se/technology/failure-mode-effects-analysis-fmea/

[20] Octopus Deploy, "Unit testing vs. integration testing," accessed: Apr. 30, 2026. [Online]. Available: https://octopus.com/devops/unit-testing/unit-testing-vs-integration-testing/

[21] N. Hayashibara, X. Défago, R. Yared, and T. Katayama, "The ϕ accrual failure detector," in *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems (SRDS 2004)*. IEEE, 2004, pp. 66–78. [Online]. Available: https://ieeexplore.ieee.org/document/1353004

[22] Inductive Automation, "What is a plc?" Feb. 2020, accessed: Apr. 29, 2026. [Online]. Available: https://inductiveautomation.com/resources/article/what-is-a-PLC

[23] *CR710S ecomatController/37 Technical Data*, ifm electronic gmbh, Jan. 2020, iEC 61508 SIL 2 safety controller with CODESYS 3.5.

[24] G. Starke, "arc42," accessed: Apr. 29, 2026. [Online]. Available: https://arc42.org

[25] S. Cho, E. Orye, G. Visky, and V. Prates, "Cybersecurity considerations in autonomous ships," 2022. [Online]. Available: https://ccdcoe.org/library/publications/cybersecurity-considerations-in-autonomous-ships/

[26] N. Tabish and T. Chaur-Luh, "Maritime autonomous surface ships: A review of cybersecurity challenges, countermeasures, and future perspectives," *IEEE Access*, vol. 12, pp. 17 114–17 136, 2024.

[27] V. Bolbot, G. Theotokatos, E. Boulougouris, and D. Vassalos, "A novel cyber-risk assessment method for ship systems," *Safety Science*, vol. 131, p. 104908, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925753520303052

[28] K. Stouffer, M. Pease, C. Tang, T. Zimmerman, V. Pillitteri, S. Lightman, A. Hahn, S. Saravia, A. Sherule, and M. Thompson, "Guide to operational technology (OT) security," National Institute of Standards and Technology, Tech. Rep. NIST SP 800-82r3, September 2023. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r3.pdf

[29] T. Bellet, M. Cunneen, M. Mullins, F. Murphy, F. Pütz, F. Spickermann, C. Braendle, and M. F. Baumann, "From semi to fully autonomous vehicles: New emerging risks and ethico-legal challenges for human-machine interactions," *Transportation Research Part F: Traffic Psychology and Behaviour*, vol. 63, pp. 153–164, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1369847818308556

[30] ISO, "Iso/iec 25010:2011 systems and software quality requirements and evaluation (square)," accessed: May 6, 2026. [Online]. Available: https://www.iso.org/obp/ui/en/#iso:std:iso-iec:25010:ed-2:v1:en

[31] W. Wippler, "Determinism in software – the good, the bad, and the ugly," May 2025, accessed: Apr. 29, 2026. [Online]. Available: https://thrawn01.org/posts/determinism-in-software---the-good,-the-bad,-and-the-ugly

[32] D. Harel, "Statecharts: A visual formalism for complex systems," *Science of Computer Programming*, vol. 8, no. 3, pp. 231–274, 1987, accessed: May 6, 2026. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0167642387900359

[33] Z. Molnár, "Logging the operation and enhancing the reliability of safety-critical embedded systems using self-test," *Interdisciplinary Description of Complex Systems*, vol. 17, no. 3-A, pp. 492–496, 2019, accessed: May 7, 2026. [Online]. Available: https://doaj.org/article/6b6fd5848e2a44a28e0404e7cd7ca781

[34] ifm electronic, "Cr710s - programmable controller for mobile machines," accessed: May 6, 2026. [Online]. Available: https://www.ifm.com/de/en/product/CR710S

A

Arkitektur

A.1 Introduktion

Syftet med denna bilaga är att beskriva den systemarkitektur som tagits fram i samband med kandidatarbetet Dependable and secure switching among manual, remote and autonomous control of a small ferry. Arkitekturen utgår från fallet med två körstationer, men är öppen för vidareutveckling.

Under diskussioner med uppdragsgivaren CStrider identifierades flera centrala krav för systemet, däribland funktionell säkerhet, tydlig felhantering, hög determinism samt möjligheten att försätta systemet i ett fail-safe-tillstånd vid kritiska fel. Fokus ligger därav på att redogöra för arkitekturens struktur, vars mål är att möjliggöra säkra och deterministiska övergångar mellan körstationer. Denna bilaga beskriver den framtagna arkitekturen och de designbeslut som ligger till grund för lösningen.

A.1.1 Funktionella krav

Kraven i detta delkapitel är funktionella och hämtade från kravspecifikationen angiven i Tabell 3.1 av slutrapporten.

ID	Beskrivning
S-01	Systemet ska stödja en manuell körstation på färjan
S-02	Systemet ska stödja en körstation på land
S-03	Systemet ska stödja övergångar mellan körstationer
S-04	Beslut inom kontrollsystemet ska loggas
S-05	Alla implementerade körstationer ska kunna integreras i samma system

Tabell A.1: Systemkrav

A.1.2 Kvalitetsmål

Systemet är säkerhetskritiskt och därmed har kvalitetskrav en utbredd påverkan på arkitekturen. I denna del av texten presenteras de överhängande kvalitetsmålen för arkitekturen. Säkerhet, pålitlighet och underhållbarhet är kvalitetsmål som utgår ifrån standarden ISO 25010 [30], medan kostnad är ett egenformulerat mål utifrån projektets behov.

Prioritet	**Mål**	**Förklaring**	**Exempel**
1	Säkerhet	Eftersom systemet spelar en stor roll i ett transportsystem så måste osäkra tillstånd förebyggas.	Om en kritisk komponent går sönder bör systemet gå in i ett fail-safe tillstånd
2	Pålitlighet	Trots fel bör systemet kunna fortsätta fungera.	Systemdesignen skall i största mån stödja tillgängligheten genom redundans för att hantera komponentbortfall. Säkerheten kommer dock alltid prioriteras så för vissa svåra felfall eller vid bortfall av flera komponenter, kommer systemet att anta ett felsäkert tillstånd.
3	Underhållbarhet	Systemet ska vara utformat på ett sätt som möjliggör utbyggnad och underhåll utan ändringar i grundläggande logik.	Om en ny körstation tillkommer ska den kunna integreras i lösningen utan att ändra existerande logik.
4	Kostnad	Systemet ska vara designat på ett sätt som minimerar total kostnad.	Om systemet är för dyrt att implementera så kommer CStrider inte kunna använda det.

Tabell A.2: Kvalitetsmål

A.2 Begränsningar

Systemets arkitektur är begränsad av följande faktorer. Dessa begränsningar har i stor utsträckning baserats på de avgränsningar som presenterades i slutrapporten:

- Arbetet fokuserar i första hand på mjukvara, så systemlösningar som innebär specialiserad hårdvara kommer inte att tas i beaktning.
- Arbetet fokuserar på funktionell säkerhet. Däremot kommer arbetet inte ta hänsyn till cyberhot.
- Systemet tar ej realtidsaspekter i beaktning, därför kommer prestanda inte att prioriteras.

A.3 Lösningsstrategi

Arkitekturens grundidé är att säkerställa säkra och pålitliga övergångar mellan körstationer, där enbart en station har kontroll vid varje tidpunkt. Eftersom säkerheten står i centrum har fokus legat på att skapa ett förutsägbart system med tydlig felhantering, även om tillgängligheten direkt påverkas. Av denna anledning har följande designprinciper varit väsentliga:

- Determinism: samma indata ska alltid leda till samma utdata och beteende hos systemet [31].
- Tillståndsmaskin: en modell som beskriver ett systems olika tillstånd samt hur övergångarna mellan dessa sker. [32]
- Centraliserad kontroll: logiken för kontrollövergångar ska finnas i en samlad komponent.
- Fail-safe: även vid fel, ska systemet kunna övergå till ett säkert tillstånd [17].
- Tydliga ansvarsområden: samtliga komponenter ska ha specifika och väl definierade ansvarsområden. Denna uppdelning hjälper till att underlätta underhåll av systemet.

Under arbetet övervägdes flera olika designmönster som prioriterade säkerhet. Lösningar som diskuterades var till en början främst baserade på hårdvaruredundans, såsom triple modular redundancy och active-passive standby [11]. Dessa designmönster valdes dock bort då de skulle introducera för hög kostnad. Därav tillämpades designmönstret monitor-actuator. Enligt detta mönster delas systemet upp i två separata delar [11]:

- En `ActuationChannel` som hanterar förfrågningar om kontroll från styrstationer.
- En `MonitoringChannel` som övervakar `ActuationChannel` för att säkerställa korrekt beteende.

Detta mönster visar sig passande i system med ett tydligt fail-safe tillstånd [11]. I detta fall kan ankaret släppas ned för att försätta systemet i ett sådant tillstånd. Dessutom är övervakningen separerad från övergångslogiken. Mer specifikt kan `MonitoringChannel` upptäcka fel i `ActuationChannel`, som kan utlösa fail-safe.

För att säkerställa central kontroll finns det i arkitekturen en central enhet som kallas `ControlUnit`. Denna enhet centraliserar logik för övergångar vilket bidrar till konsekventa och deterministiska beslut. Samtidigt innebär detta designbeslut att `ControlUnit` blir en single point of failure (SPOF) för systemets tillgänglighet. Däremot hindrar det nämnda fail-safe tillståndet att detta beroende blir en SPOF för hela systemet.

Systemet använder en tillståndsmaskin för att modellera beteende. Tillstånd används i denna kontext både för att beskriva en aktiv körstation, eller ett skede i en övergång. I det fall då flera körstationer begär en övergång samtidigt så används en tydlig prioritetsordning.

Vidare är systemet utformat på så sätt att kritiska fel, med hjälp av fail-safe, leder till ett säkert tillstånd. Exempel på sådana fel skulle kunna vara förlust av heart-

beat hos `ActuationChannel` eller beslut i `ControlUnit` som inte stämmer överens med indata. Användningen av fail-safe är ett medvetet designbeslut som prioriterar säkerhet över tillgänglighet.

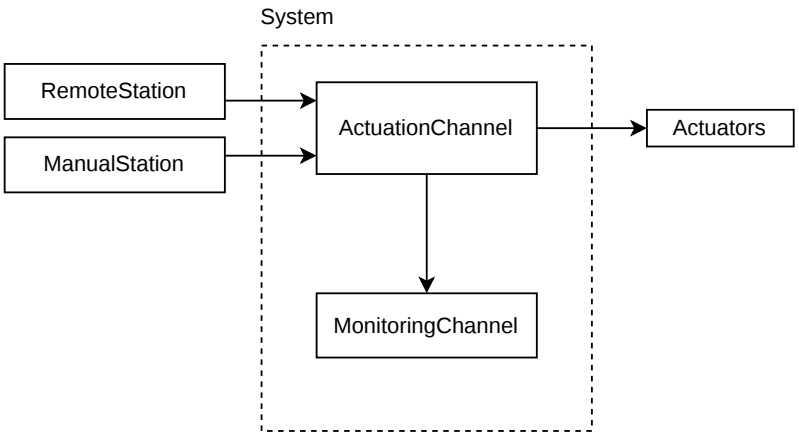
Signaler verifieras i flera steg. Insignalers rimlighet kontrolleras av komponenten `IntegrityCheck` och `ControlUnit` verifierar giltighet hos övergångar. Samtidigt bevakar `MonitoringChannel` det resulterande beteendet.

För att ytterligare öka säkerheten i systemet används en `DecisionLogger` som sparar beslut. Komponenten sparar information om aktuell station, pågående övergång, tidsstämplar, föregående aktiv station och övergångsbegäran som tidigare nekats. Tillgång till denna information ger språbarhet, vilket underlättar felsökning [33].

A.4 Statisk vy

I denna del av texten är varje komponent förklarad utifrån sitt ansvar i systemet. Alla komponenter tillsammans bidrar till säkra och deterministiska övergångar.

A.4.1 Nivå ett

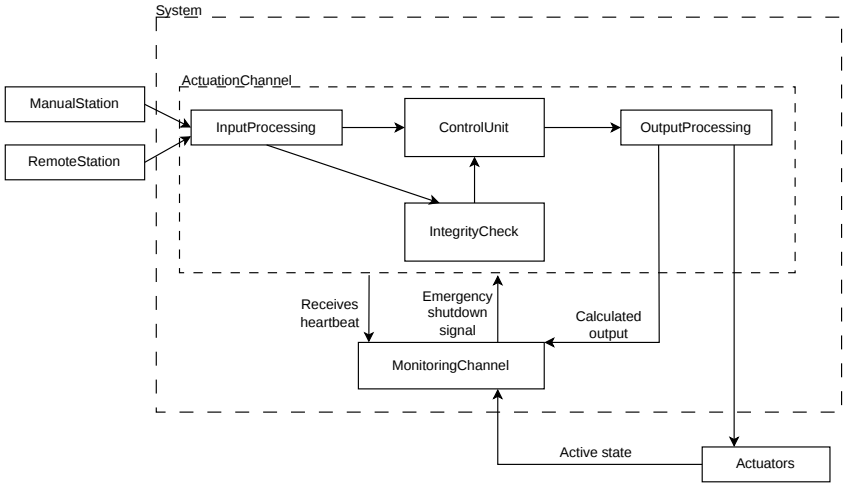


Figur A.1: Systemöverblick på hög nivå som visar separationen mellan `ActuationChannel` och `MonitoringChannel` i linje med designmönstret monitor-actuator.

Logiskt element	**Beskrivning**
`RemoteStation`	Manuell körstation på land.
`ManualStation`	Manuell körstation på båten.
`ActuationChannel`	Utför de logiska beräkningar för övergångar mellan körstationer. Den tar in data från stationer och fattar beslut om vilken som bör vara i kontroll.
`MonitoringChannel`	Bevakar den övergripande hälsan hos `ActuationChannel` och jämför dess utdata med fysiskt beteende.
`Actuators`	Ansvarar för att verkställa beslut som `ActuationChannel` tar. Kräver fysiska aktuatorer för implementering och är integrationspunkt för vidare hårdvaruarbete.

Tabell A.3: Logiska element

A.4.2 Nivå två

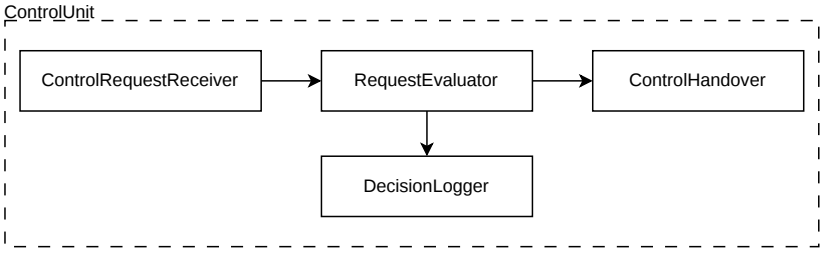


Figur A.2: Delkomponenter i `ActuationChannel` samt deras interaktion med `MonitoringChannel`.

Logiskt element	Beskrivning
`ControlUnit`	Hanterar logik för att byta mellan körstationer. Innehåller även säkerhetsmekanismer.
`IntegrityCheck`	Validerar inkommande signaler genom en rimlighetskontroll.
`InputProcessing`	Bearbetar data från körstationer så den kan användas i beräkningar.
`OutputProcessing`	Bearbetar data till faktiska signaler som hårvarunära komponenter kan använda.

Tabell A.4: Interna komponenter i `ActuationChannel`

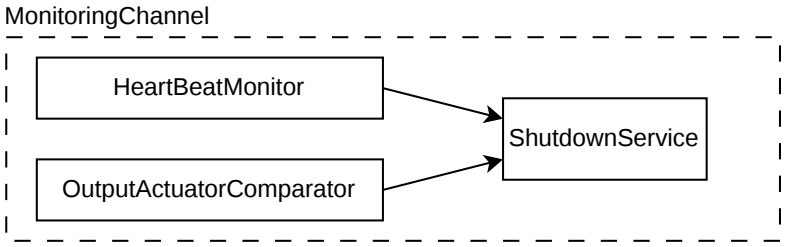
A.4.3 Nivå tre



Figur A.3: De inre komponenterna i `ControlUnit`.

Logiskt element	Beskrivning
`ControlRequestReceiver`	Tar emot förfrågningar om kontroll från olika körstationer.
`RequestEvaluator`	Utvärderar förfrågningar om kontroll utifrån etablerade villkor.
`ControlHandover`	Utför den faktiska övergången av kontroll mellan körstationer.
`DecisionLogger`	Lagrar uppgifter om nuvarande station, aktiv övergång, tidsangivelser, föregående aktiva station samt övergångsförfrågningar som tidigare har avslagits.

Tabell A.5: Delkomponenter i `ControlUnit`



Figur A.4: De inre komponenterna i `MonitoringChannel`.

Logiskt element	Beskrivning
`HeartbeatMonitor`	Övervakar `ActuationChannel` och dess hälsa genom att läsa av heartbeats.
`OutputActuatorComparator`	Jämför `ActuationChannel` och dess utdata med det faktiska beteendet hos systemet. Kräver fysiska sensorer för komplett implementation och är därmed en integrationspunkt för framtida hårdvarudriftsättning.
`ShutdownService`	Ansvarar för att skicka signaler om att gå in i fail-safe, givet fel hos systemet.

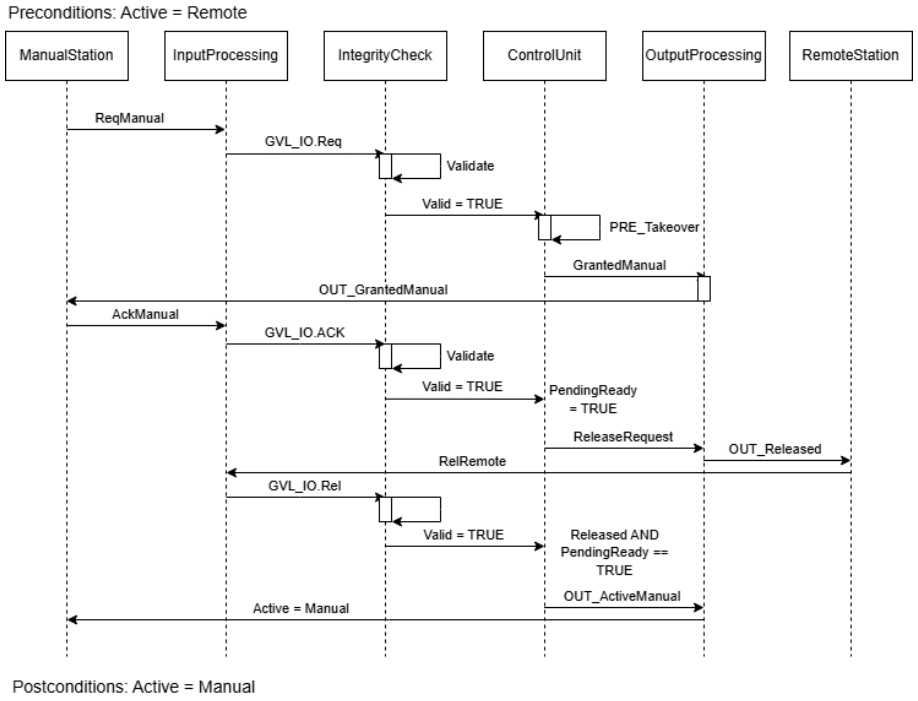
Tabell A.6: Beskrivning av delkomponenter i `MonitoringChannel`

A.5 Dynamisk vy

Detta kapitel åskådliggör systemets beteende. Mer specifikt innehåller varje delkapitel ett sekvensdiagram och tillhörande brödtext. De scenarion som valts ut visar upp några av systemets viktigaste säkerhetsmekanismer samt hur en normal övergång sker.

A.5.1 En förfrågan skickas

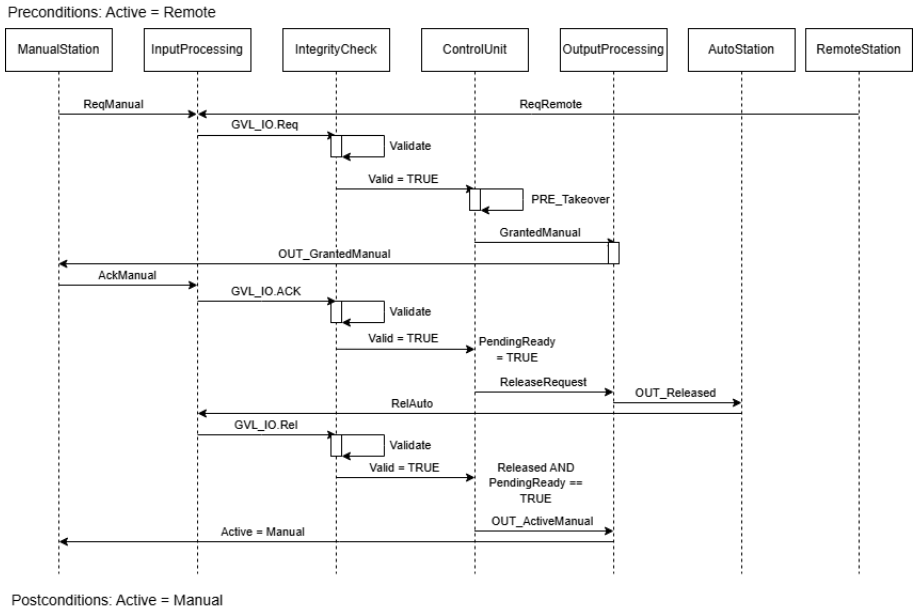
ManualStation skickar en förfrågan om att ta över kontrollen. Förfrågan valideras av `IntegrityCheck` och godkänns av `ControlUnit`. Därefter signalerar `ControlUnit` att `RemoteStation` ska släppa kontrollen samtidigt som `ManualStation` ska aktiveras. När `ManualStation` bekräftat dess tillgänglighet och `RemoteStation` släppt kontrollen, så genomförs övergången.



Figur A.5: Sekvensdiagram för en övergångsförfrågan.

A.5.2 Två förfrågningar skickas samtidigt

`ManualStation` och `RemoteStation` skickar samtidigt varsin övergångsförfrågan. `ControlUnit` utvärderar båda begäranden och väljer den med högst prioritet. Därefter fortsätter processen som vanligt tills `ManualStation` sätts som aktiv station.



Figur A.6: Sekvensdiagram för två samtidiga övergångsförfrågningar.

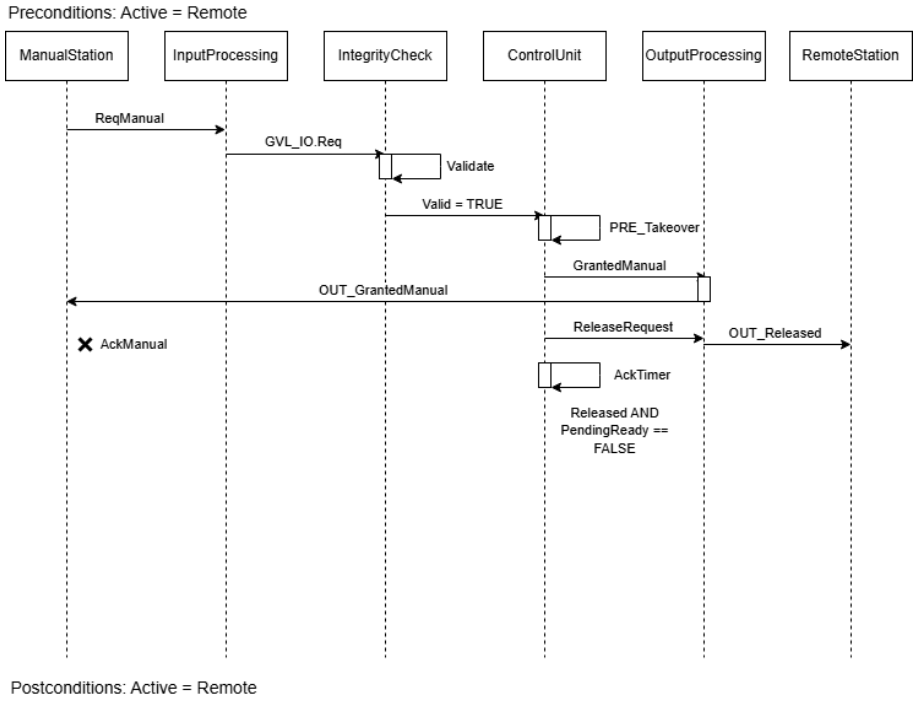
A.5.3 Körstation bekräftar inte sin tillgänglighet

`ControlUnit` godkänner `ManualStations` förfrågan, men får inget svar från stationen inom ett satt tidsintervall. Därav avbryts övergången och systemet återgår till samma tillstånd som innan processen. Avvisningen sparas i systemet där orsaken beskrivs som en timeout.

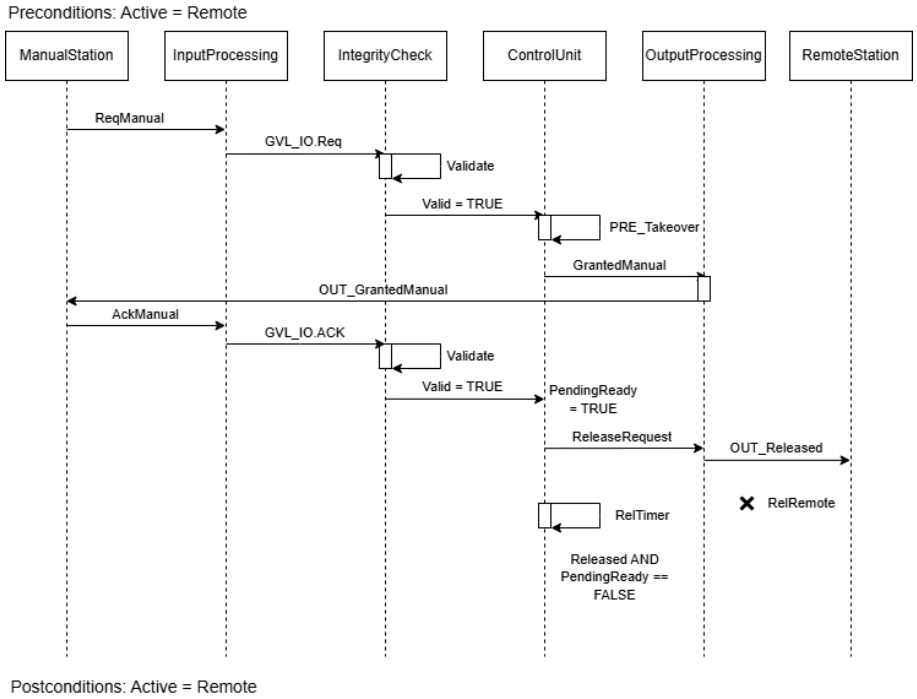
A.5.4 Körstation släpper inte kontroll

`ControlUnit` godkänner att påbörja övergången till `ManualStation`, men `RemoteStation` släpper inte kontrollen inom ett definierat tidsintervall. Övergången avbryts och systemet försätts i samma tillstånd som innan processen. När övergången stoppas sparas beslutet inom systemet med timeout som orsak.

Resultatet av detta scenario är att den gamla körstationen har fortsatt kontroll. I situationer där denna körstation inte längre är i skick för styrning blir detta problematiskt. Därför kan en hårdvarubaserad lösning implementeras, där den manuella körstationen får kontrollen och alla andra stängs ned. Denna utveckling ligger utanför projektets avgränsningar och ses som potentiellt framtida arbete.



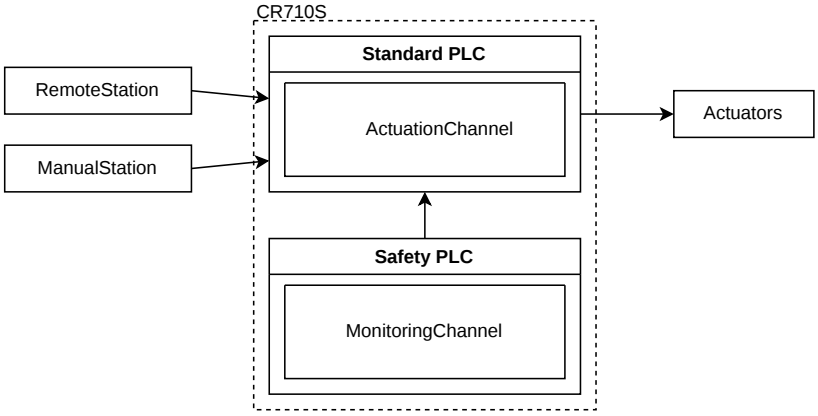
Figur A.7: Sekvensdiagram för när den körstation som skickade övergångsförfrågan inte bekräftar sin tillgänglighet.



Figur A.8: Sekvensdiagram för när den gamla körstationen inte släpper kontroll.

A.6 Driftsättningsvy

Systemet driftsätts på en CR710S utvecklad av IFM [34]. Denna styrenhet består av två PLC:er, där en är avsedd för funktionell säkerhet och den andra för standardfunktionalitet [23]. Uppdelningen syns i Figur A.9.



Figur A.9: Placering av `ActuationChannel` och `MonitoringChannel` på CR710S enskilda PLC:er.

Logisk enhet	Beskrivning
Standard PLC	Intern PLC inuti CR710S designad för att innehålla systemets standardfunktionalitet.
Safety PLC	Intern PLC inuti CR710S designad för att innehålla systemets säkerhetsfunktionalitet.

Tabell A.7: Logiska enheter i systemet

A.7 Sammanfattning av koncept

Detta kapitel sammanfattar de viktigaste designprinciperna och hur de använts i arkitekturen. Denna sektion är ett stöd till avsnitt A.4.

A.7.1 Säkerhet och fail-safe

Systemet är designat utefter tillgång till ett tydligt fail-safe tillstånd. Mer specifikt kan ankaret hos färjan släppas ned. Färjan försätts i detta tillstånd om kritiska fel upptäcks i systemet, då säkerhet prioriteras över tillgänglighet.

Fail-safe kan utlösas av två delar i systemet:

- `ControlUnit` i `ActuationChannel` om interna fel upptäcks.
- `MonitoringChannel` om avvikande externt beteende upptäcks.

I fail-safe läget kan ingen körstation ta kontroll, vilket resulterar i att systemet inte kan fortsätta drift i farliga situationer. Denna avvägning innebär att säkerhet prioriteras över tillgänglighet.

A.7.2 Monitor-actuator

Ansvarsområdet hos `ActuationChannel` är att ta emot indata från körstationer och hantera övergångar mellan dessa, medan `MonitoringChannel` övervakar denna process. `MonitoringChannel` läser av signaler från `ActuationChannel` med mål att verifiera korrekt beteende hos systemet. Vid avvikelse, såsom uteblivna beslut eller heartbeat, kan `MonitoringChannel` utlösa ett fail-safe tillstånd. Den huvudsakliga idén bakom uppdelningen är att beslut och övervakning separeras logiskt.

A.7.3 Tillståndsbaserad kontroll

Övergångarna inom systemet modelleras i `ControlUnit` som en tillståndsmaskin. Tillstånden representerar både faser i övergångar, samt vilken körstation som är aktiv. Övergångar mellan tillstånd kan i enlighet med denna modell bara ske när definierade villkor är uppfyllda. Denna process möjliggör säkra övergångar.

A.8 Större arkitekturbeslut

Tabell A.8: Arkitekturella designbeslut

Titel	Beslut	Konsekvenser
ADR-1	Central logik i `ControlUnit`	Beslut fattas deterministiskt, men ett centralt beroende skapas. Mer specifikt blir `ControlUnit` utifrån detta beslut en SPOF, vilket leder till att ett fel i denna komponent påverkar systemets helhet.
ADR-2	Användning av designmönstret monitor-actuator	Systemet delas upp i en `ActuationChannel` och en `MonitoringChannel`. Detta resulterar i tydlig separation mellan övergångslogik och övervakning. Därav kan fel i beslutslogiken lättare upptäckas, i utbyte mot att systemets komplexitet ökar.
ADR-3	Fail-safe prioriteras över tillgänglighet	Bidrar till ett säkrare system, men påverkar tillgänglighet i form av fler driftavbrott.
ADR-4	`MonitoringChannel` observerar enbart signaler från `ActuationChannel`	Lägre komplexitet men begränsar systemets förmåga att upptäcka vissa typer av fel i `ActuationChannel`.
ADR-5	Validering av signaler i `IntegrityCheck` innan de används i `ControlUnit`	Förhindrar att ogiltiga signaler når `ControlUnit`. Bidrar till ett robust system, men introducerar mer komplexitet.
ADR-6	`MonitoringChannel` installeras på säkerhets-PLCn och `ActuationChannel` på standard-PLCn hos CR710S	Skiljer `MonitoringChannel` och `ActuationChannel` fysiskt. Bidrar till tydligare separation mellan logik och övervakning, men gör systemet beroende av kommunikation mellan de olika PLC:erna.

Titel	Beslut	Konsekvenser
ADR-7	Simultana requests av olika körstationer i samma programcykel hanteras genom en fast prioritetsordning	En station med högre prioritet kan blockera lägre prioriterade stationer. Lägre prioriterade stationer får ingen explicit avvisning men händelsen loggas.
ADR-8	Övergångar valideras i flera steg, både före och under övergångsprocessen	Ökar robustheten mot oväntade tillstånd. Om ett skyddslager misslyckas triggar nästa lager fail-safe. Systemet litar inte på att tidigare steg alltid fungerar korrekt.
ADR-9	Systemet stannar i ett uppstartsläge tills en station med giltigt heartbeat skickar en request	Operatören måste aktivt ta kontroll vid uppstart, vilket minskar onödiga fail-safe-övergångar vid normal uppstart.

A.9 Arkitekturella risker

De arkitekturella riskerna identifierades iterativt under framtagningen av systemarkitekturen. Riskerna baseras på analys av designbeslut, komponentberoenden, systemets beteende vid avvikande tillstånd samt felhantering.

Tabell A.9: Identifierade risker

Risk	**Beskrivning**
`MonitoringChannel` är beroende av `ActuationChannel`	`MonitoringChannel` är direkt beroende av signaler från `ActuationChannel`. Om `ActuationChannel` havererar så kan inte `MonitoringChannel` utföra sin uppgift och därmed slutar hela systemet att fungera.
Standard PLC är en SPOF	`MonitoringChannel` kan inte fungera utan `ActuationChannel`. Därför måste systemet initiera fail-safe om standard PLC:n slutar fungera.
Begränsad felupptäckt	Eftersom `MonitoringChannel` inte duplicerar övergångslogik, så kan upptäckten av vissa logiska fel bli utebliven. Till exempel kan `MonitoringChannel` inte avgöra om rätt station väljs vid flera samtidiga kontrollförfrågningar, då den inte har tillgång till prioritetslistan.
Uppdelning av systemet i två separata PLC:er	Ökar komplexiteten i lösningen och resulterar i en svårare samt mer tidskrävande implementation.
Timeout-värden	För låga eller höga timeout-värden påverkar felhantering direkt. Mer specifikt kan för låga värden innebära en aggressiv hantering, medan höga värden kan resultera i utebliven hantering.
Simuleringsbegränsningar	Systemet kommer att simuleras i en sluten miljö. Därför kan vissa verkliga situationer inte plockas upp.

Risk	**Beskrivning**
Ingen garanti att aktiv station slutar påverka aktuatorer vid release	Released-signalen bekräftar att en aktiv station signalerat att den släpper kontrollen, men systemet kan inte verifiera att stationen faktiskt slutat skicka styrsignaler. Under övergången kan två stationer teoretiskt påverka aktuatorerna samtidigt. Aktuatorlagret måste säkerställa att endast signaler från en aktiv station verkställs.
Heartbeat-generatorns intervall och monitorns timeout är separata värden	Om enbart ena värdet ändras, så kan heartbeat-detektionen sluta fungera. Denna risk mitigeras genom konstanter i heartbeat-funktionalitet, där skillnader mellan dessa värden hålls inom satta gränser.

B

System FMEA

B.1 Introduktion

Denna bilaga innehåller den FMEA som genomfördes i samband med projektet. Resultatet presenteras i tabellform.

Värde k	Liten 1	Märkbar 2	Allvarlig 3	Mycket allvarlig 4

			Konsekvens		Riskbehandling	
ID	Felmod	Konsekvensbeskrivning	Värde K 1-4	Konsekvensnsnivå	Riskbehandling	Möjlig riskbehandling
SR-01	Flera körstationer skickar en övergångsbegäran	Eventuell konflikt där fel körstation får kontrollen.	4		Prioritetsval i ControlUnit	
SR-02	Inkorrekt hanterad indata	Beslut blir felaktigt	3		IntegrityCheck	
SR-03	Aktiv körstation släpper kontrollen sent	Situation där flera körstationer är aktiva samtidigt	4		Active := EControlMode.X i VALIDATE_TAKEOVER	
SR-04	InputProcessing blandar kommandon från flera körstationer.	Konflikt i styrning och felaktiga styrsignaler	3		Ett separat logiskt block för varje logisk komponent.	
SR-05	Körstation med lägre prioritet väljs	Fel körstation styr	3		Prioritetsordning via ELSIF i STABLE	
SR-06	Internt tillstånd (aktiv körstation) uppdateras inkonsekvent eller inte alls.	Osäkert läge.	3		setActActive := EControlMode.X i VALIDATE_TAKEOVERiveSource()	
SR-07	Körstation skickar förfrågan om kontroll utan att vara i skick för att faktiskt ta över den.	Förlust av styrning	4		Förberedelse finns, där ACK skickas från aktuell körstation.	
SR-08	ACK uteblir	Osäkerhet gällande vilken körstation som är aktiv	2		ACK-timeout	
SR-09	Körstation skickar en falsk ACK	Icke-förfrågad övergång.	4		GrantedX signal kräves innan AckX, en station kan inte bekräfta utan att ha fått Granted	
SR-10	Körstation får ej signal om att den ska ta över kontroll.	Riskerar att den aktiva körstationen släpper kontroll utan att den nya är redo att ta över.	2		ACK	
SR-10	Körstation får ej signal om att den ska släppa kontroll.	Riskerar att flera körstationer tror att de har kontrollen.	2		ACK	
SR-11	Station får inte fysisk signal som tilldelar den kontrollen.	Förlust av styrning.	4			Ej implementerad, kräver återkopplingssignal från extern aktuator. Noterat som begränsning i arc 42.
SR-12	OutputProcessing tolkar aktivering av körstation fel.	Fel körstation aktiveras.	4		Tydlig mappning av körstationer i implementering	
SR-13	Intern inkonsistens i ControlUnit	Ogiltig övergång till samma station	3		Defence in depth, PRE_TAKEOVER, VALIDATE_TAKEOVER detekterar och triggar FAILSAFE	
SR-14	Osäker övergång utan tydliga villkor	Farlig övergång.	4		validateTakeover()	
SR-15	Blandning av ansvar hos komponenter.		2		Arkitektur som är tydligt uppdelad.	
SR-16	Duplicerad request från en körstation.	Samma övergång genomförs flera gånger parallelt.	1		ControlUnit detekterar varje REQ-signal som ett enskilt knapptryck, oavsett hur länge signalen hålls aktiv. Detta förhindrar att samma request behandlas flera gånger.	
SR-17	Flera körstationer tror att de har kontroll.	Konflikt om styrning	4		Tillståndmodell, med en aktiv körstation.	
SR-18	Omstart av system i mitten av en kontrollövergång	Osäkerhet gällande vilken körstation som är aktiv	4			Fail-safe tillstånd
SR-19	Körstation skickar ACK utan aktiv kontrollövergång kopplad till den.	Osäkert tillstånd	3		Den aktiva körstationen måste släppa kontrollen för att möjliggöra en kontrollövergång.	
SR-20	Aktiv körstation skickar kommandon under övergång.	Kortsiktig konflikt i styrning	2			Ej implementerad

Figur B.1: Tabell som visar FMEA

C
Källkod

C.1 InputProcessing

```
FUNCTION_BLOCK InputProcessing
//Reads input directly from hardware and maps to logical signals
//Abstraction layer between hardware inputs and internal logic to
//decouple the system from the hardware signals
VAR_INPUT
        xReqRemote        : BOOL;
        xAckRemote         : BOOL;
        xRelRemote        : BOOL;
        xHB_Remote        : BOOL;

        xReqManual         : BOOL;
        xAckManual         : BOOL;
        xRelManual        : BOOL;
        xHB_Manual        : BOOL;

        xReqAuto        : BOOL;
        xAckAuto        : BOOL;
        xRelAuto        : BOOL;
        xHB_Auto        : BOOL;

        xReqRemoteMobile        : BOOL;
        xAckRemoteMobile        : BOOL;
        xRelRemoteMobile        : BOOL;
        xHB_Remote_Mobile        : BOOL;

END_VAR
VAR_OUTPUT
        ReqManual : BOOL;
        AckManual : BOOL;
        RelManual : BOOL;
        HB_Manual_Signal : BOOL;

        ReqRemote : BOOL;
        AckRemote : BOOL;
        RelRemote : BOOL;
        HB_Remote_Signal : BOOL;

        ReqAuto : BOOL;
        AckAuto : BOOL;
        RelAuto : BOOL;
        HB_Auto_Signal : BOOL;

        ReqRemoteMobile : BOOL;
        AckRemoteMobile : BOOL;
        RelRemoteMobile : BOOL;
        HB_Remote_Mobile_Signal : BOOL;
```

```
47
48  END_VAR
49  VAR
50  END_VAR
51
52  // Manual input - with an added abstraction layer after the hardware channels
53  ReqManual := xReqManual;
54  AckManual := xAckManual;
55  RelManual := xRelManual;
56  HB_Manual_Signal := xHB_Manual;
57
58  // Remote input - from BOOL input
59  ReqRemote := xReqRemote;
60  AckRemote := xAckRemote;
61  RelRemote := xRelRemote;
62  HB_Remote_Signal := xHB_Remote;
63
64  // Autonomous input - from BOOL input
65  ReqAuto          := xReqAuto;
66  AckAuto          := xAckAuto;
67  RelAuto          := xRelAuto;
68  HB_Auto_Signal    := xHB_Auto;
69
70  // Remote mobile - from BOOL input
71  ReqRemoteMobile := xReqRemoteMobile;
72  AckRemoteMobile := xAckRemoteMobile;
73  RelRemoteMobile := xRelRemoteMobile;
74  HB_Remote_Mobile_Signal := xHB_Remote_Mobile;
```

C.2 IntegrityCheck

```
1   FUNCTION_BLOCK IntegrityCheck
2   // Validates signal combinations before they reach ControlUnit.
3   // Blocks contradictory signals from a single station (e.g. simultaneous
4   // request and release). If invalid, all request signals are set to FALSE
5   // and Valid is set to FALSE, triggering FAILSAFE in ControlUnit.
6   VAR_INPUT
7           ReqManual         : BOOL;
8       ReqRemote         : BOOL;
9           ReqAuto                          : BOOL;
10          ReqRemoteMobile   : BOOL;
11
12      AckManual         : BOOL;
13      AckRemote         : BOOL;
14          AckAuto                          : BOOL;
15          AckRemoteMobile   : BOOL;
16
17      RelManual         : BOOL;
18      RelRemote         : BOOL;
19          RelAuto                          : BOOL;
20          RelRemoteMobile   : BOOL;
21
22      HB_Manual_Signal : BOOL;
```

```
23      HB_Remote_Signal : BOOL;
24          HB_Auto_Signal          : BOOL;
25          HB_Remote_Mobile_Signal : BOOL;
26  END_VAR
27  VAR_OUTPUT
28          Valid             : BOOL;  // All signals passed sanity check
29      ReqManual_Out     : BOOL;
30      ReqRemote_Out     : BOOL;
31          ReqAuto_Out                 : BOOL;
32          ReqRemoteMobile_Out : BOOL;
33      AckManual_Out     : BOOL;
34      AckRemote_Out     : BOOL;
35          AckAuto_Out                 : BOOL;
36          AckRemoteMobile_Out : BOOL;
37      RelManual_Out     : BOOL;
38      RelRemote_Out     : BOOL;
39          RelAuto_Out                 : BOOL;
40          RelRemoteMobile_Out : BOOL;
41      HB_Manual_Out     : BOOL;
42      HB_Remote_Out     : BOOL;
43          HB_Auto_Out                 : BOOL;
44          HB_Remote_Mobile_Out : BOOL;
45  END_VAR
46  VAR
47  END_VAR
48
49  IF (ReqManual AND ReqRemote) OR //two stations want control
50     (ReqManual AND ReqAuto)          OR
51     (ReqManual AND ReqRemoteMobile) OR
52     (ReqRemote AND ReqAuto)         OR
53     (ReqRemote AND ReqRemoteMobile) OR
54     (ReqAuto AND ReqRemoteMobile) THEN
55
56             Valid        := FALSE;
57      ReqManual_Out := FALSE;
58      ReqRemote_Out := FALSE;
59          ReqAuto_Out          := FALSE;
60          ReqRemoteMobile_Out := FALSE;
61      AckManual_Out := AckManual;
62      AckRemote_Out := AckRemote;
63          AckAuto_Out          := AckAuto;
64          AckRemoteMobile_Out := AckRemoteMobile;
65      RelManual_Out := RelManual;
66      RelRemote_Out := RelRemote;
67          RelAuto_Out          := RelAuto;
68          RelRemoteMobile_Out := RelRemoteMobile;
69      HB_Manual_Out := HB_Manual_Signal;
70          HB_Remote_Out := HB_Remote_Signal;
71          HB_Auto_Out          := HB_Auto_Signal;
72          HB_Remote_Mobile_Out := HB_Remote_Mobile_Signal;
73      RETURN;
74  END_IF
75
76  IF (ReqManual AND RelManual) OR //a station requesting and releasing at the
    ↪  same time
77     (ReqRemote AND RelRemote) OR
78     (ReqAuto AND RelAuto) OR
```

```
    (ReqRemoteMobile AND RelRemoteMobile) OR
    (AckManual AND RelManual) OR //Station acknowledging and releasing
    ↪  simultaneously
    (AckRemote AND RelRemote) OR
    (AckAuto AND RelAuto) OR
    (AckRemoteMobile AND RelRemoteMobile) THEN
            Valid        := FALSE;
     ReqManual_Out := FALSE;
     ReqRemote_Out := FALSE;
         ReqAuto_Out          := FALSE;
         ReqRemoteMobile_Out := FALSE;
     AckManual_Out := FALSE;
     AckRemote_Out := FALSE;
         AckAuto_Out          := FALSE;
         AckRemoteMobile_Out := FALSE;
     RelManual_Out := FALSE;
     RelRemote_Out := FALSE;
         RelAuto_Out          := FALSE;
         RelRemoteMobile_Out := FALSE;
     HB_Manual_Out := FALSE;
         HB_Remote_Out := FALSE;
         HB_Auto_Out          := FALSE;
         HB_Remote_Mobile_Out := FALSE;
    RETURN;
END_IF

Valid         := TRUE;
ReqManual_Out := ReqManual;
ReqRemote_Out := ReqRemote;
ReqAuto_Out          := ReqAuto;
ReqRemoteMobile_Out := ReqRemoteMobile;
AckManual_Out := AckManual;
AckRemote_Out := AckRemote;
AckAuto_Out          := AckAuto;
AckRemoteMobile_Out := AckRemoteMobile;
RelManual_Out := RelManual;
RelRemote_Out := RelRemote;
RelAuto_Out   := RelAuto;
RelRemoteMobile_Out := RelRemoteMobile;
HB_Manual_Out := HB_Manual_Signal;
HB_Remote_Out := HB_Remote_Signal;
HB_Auto_Out          := HB_Auto_Signal;
HB_Remote_Mobile_Out := HB_Remote_Mobile_Signal;
```

C.3 OutputProcessing

```
FUNCTION_BLOCK OutputProcessing
VAR_INPUT
        ReqManual       : BOOL;
    ReqRemote       : BOOL;
        ReqAuto                         : BOOL;
        ReqRemoteMobile : BOOL;
    GrantedManual   : BOOL;
```

```
8       GrantedRemote    : BOOL;
9           GrantedAuto                : BOOL;
10          GrantedRemoteMobile : BOOL;
11      Released         : BOOL;
12      PendingReady     : BOOL;
13      Active           : EControlMode;
14      State            : ETransitionState;
15      HB_Manual_OK     : BOOL;
16      HB_Remote_OK     : BOOL;
17          HB_Auto_OK                 : BOOL;
18          HB_Remote_Mobile_OK : BOOL;
19          RejectedRequestFlag : BOOL;
20  END_VAR
21  VAR_OUTPUT
22          OUT_GrantedManual : BOOL;
23          OUT_GrantedRemote : BOOL;
24          OUT_GrantedAuto          : BOOL;
25          OUT_GrantedRemoteMobile : BOOL;
26          OUT_ActiveManual  : BOOL;
27          OUT_ActiveRemote  : BOOL;
28          OUT_ActiveAuto           : BOOL;
29          OUT_ActiveRemoteMobile : BOOL;
30          OUT_Pending                   : BOOL;
31          OUT_PendingReady  : BOOL;
32          OUT_Released           : BOOL;
33          OUT_NoHB_Manual   : BOOL; //loss of heartbeat
34          OUT_NoHB_Remote   : BOOL; //loss of heartbeat
35          OUT_NoHB_Auto          : BOOL; //loss of heartbeat
36          OUT_NoHB_RemoteMobile : BOOL;
37          OUT_FailSafe           : BOOL;
38          OUT_State_Stable : BOOL;
39          OUT_State_Validate : BOOL;
40          OUT_State_FailSafe : BOOL;
41          OUT_RejectedRequestFlag : BOOL;
42
43  END_VAR
44  VAR
45  END_VAR
46
47  //No outputs are active during INIT or FAILSAFE
48  IF Active = EControlMode.INIT OR Active = EControlMode.FAILSAFE THEN
49          OUT_ActiveManual := FALSE;
50          OUT_ActiveRemote  := FALSE;
51          OUT_ActiveAuto    := FALSE;
52          OUT_ActiveRemoteMobile := FALSE;
53          OUT_GrantedManual := FALSE;
54          OUT_GrantedRemote := FALSE;
55          OUT_GrantedAuto          := FALSE;
56          OUT_GrantedRemoteMobile := FALSE;
57          OUT_Pending                   := FALSE;
58          OUT_PendingReady  := FALSE;
59          OUT_Released           := FALSE;
60  ELSE
61          OUT_GrantedManual := GrantedManual;
62          OUT_GrantedRemote := GrantedRemote;
63          OUT_GrantedAuto          := GrantedAuto;
64          OUT_GrantedRemoteMobile := GrantedRemoteMobile;
```

```
65          OUT_ActiveManual  := (Active = EControlMode.MANUAL);
66          OUT_ActiveRemote  := (Active = EControlMode.REMOTE);
67          OUT_ActiveAuto    := (Active = EControlMode.AUTO);
68          OUT_ActiveRemoteMobile := (Active = EControlMode.REMOTE_MOBILE);
69          OUT_Pending                  := (State =
            ↪  ETransitionState.VALIDATE_TAKEOVER) OR
70                                                 (State =
                                                   ↪  ETransitionState.PRE_TAKEOVER);
71          OUT_PendingReady  := PendingReady;
72          OUT_Released          := Released;
73  END_IF
74  OUT_FailSafe          := (Active = EControlMode.FAILSAFE);
75  OUT_RejectedRequestFlag := RejectedRequestFlag;
76  // Heartbeat status outputs are always active regardless of system state.
77  // This ensures operators can always monitor station availability,
78  // even during INIT or FAILSAFE.
79  //Heartbeat should be visible regardless of mode we're in
80  OUT_NoHB_Manual   := NOT HB_Manual_OK;
81  OUT_NoHB_Remote   := NOT HB_Remote_OK;
82  OUT_NoHB_Auto          := NOT HB_Auto_OK;
83  OUT_NoHB_RemoteMobile := NOT HB_Remote_Mobile_OK;
84
85  OUT_State_Stable := (State = ETransitionState.STABLE);
86  OUT_State_Validate := (State = ETransitionState.VALIDATE_TAKEOVER);
87  OUT_State_FailSafe := (State = ETransitionState.FAILSAFE);
88
```

C.4 GVL_HeartBeat, GVL_IO och GVL__State

```
1   {attribute 'qualified_only'}
2   VAR_GLOBAL CONSTANT
3           //HB_Generator_Interval must always be less than HB_Monitor_Timeout
4           //Current margin: 300ms - adjust both if either value changes
5           HB_Generator_Interval : TIME := T#200MS;
6           HB_Monitor_Timeout          : TIME := T#500MS;
7   END_VAR
8
9   VAR_GLOBAL
10          HB_Remote_Signal                : BOOL;
11          HB_Manual_Signal                : BOOL;
12          HB_Auto_Signal                       : BOOL;
13          HB_Remote_Mobile_Signal : BOOL;
14
15          HB_Manual_OK                : BOOL;
16          HB_Remote_OK                : BOOL;
17          HB_Auto_OK                       : BOOL;
18          HB_Remote_Mobile_OK : BOOL;
19  END_VAR
```

```
1   {attribute 'qualified_only'}
2   VAR_GLOBAL
3
4           //Request
5           ReqManual : BOOL;
```

```
        ReqRemote : BOOL;
        ReqAuto          : BOOL;
        ReqRemoteMobile : BOOL;

        //Acknowledge
        AckManual : BOOL;
    AckRemote : BOOL;
        AckAuto          : BOOL;
        AckRemoteMobile : BOOL;

        //Release
    RelManual : BOOL;
    RelRemote : BOOL;
        RelAuto          : BOOL;
        RelRemoteMobile : BOOL;

        ReleaseRequest_Manual : BOOL;
        ReleaseRequest_Remote : BOOL;
        Releaserequest_Auto : BOOL;
        ReleaseRequest_Remote_Mobile: BOOL;
END_VAR
```

```
{attribute 'qualified_only'}
VAR_GLOBAL
        Active        : EControlMode;      // Currently active station
    Pending       : EPendingMode;      // Station wanting to take over
    State         : ETransitionState;  // Stable / Pending_Ack

    GrantedManual : BOOL;
    GrantedRemote : BOOL;
        GrantedAuto          : BOOL;
        GrantedRemoteMobile : BOOL;

    Released      : BOOL;
    PendingReady  : BOOL;

        RejectedRequestFlag : BOOL;
        RejectionReason                : STRING;
        LastActiveBeforeTransition : EControlMode;

        TimeStamp : UDINT;

END_VAR
```

C.5 ControlUnit

```
FUNCTION_BLOCK ControlUnit
VAR_INPUT
        Valid         : BOOL;

        ReqManual                : BOOL;
        ReqRemote                : BOOL;
        ReqAuto                  : BOOL;
        ReqRemoteMobile : BOOL;
```

```
9
10          AckManual                : BOOL;
11          AckRemote                : BOOL;
12          AckAuto                  : BOOL;
13          AckRemoteMobile : BOOL;
14
15          RelManual                : BOOL;
16          RelRemote                : BOOL;
17          RelAuto                  : BOOL;
18          RelRemoteMobile : BOOL;
19
20          HB_Manual_OK                : BOOL;
21          HB_Remote_OK                : BOOL;
22          HB_Auto_OK                  : BOOL;
23          HB_Remote_Mobile_OK : BOOL;
24
25          MaxAckTime : TIME; //timeout threshold
26
27
28  END_VAR
29  VAR_OUTPUT
30
31          Active          : EControlMode := EControlMode.INIT;
32
33          GrantedManual               : BOOL;
34          GrantedRemote               : BOOL;
35          GrantedAuto                       : BOOL;
36          GrantedRemoteMobile : BOOL;
37
38          State : ETransitionState;
39
40          Released                 : BOOL;
41          PendingReady          : BOOL;
42          RejectedRequestFlag          : BOOL;
43          RejectionReason                       : STRING;
44
45          ReleaseRequest_Manual : BOOL;
46          ReleaseRequest_Remote : BOOL;
47          ReleaseRequest_Auto : BOOL;
48          ReleaseRequest_Remote_Mobile : BOOL;
49
50  END_VAR
51  VAR
52          Pending          : EPendingMode;
53          tAckTimer          : TON;
54
55          xReqRemoteOld                : BOOL := FALSE;
56          xReqManualOld                : BOOL := FALSE;
57          xReqAutoOld                  : BOOL := FALSE;
58          xReqRemoteMobileOld : BOOL := FALSE;
59
60          xReqRemotePulse                : BOOL := FALSE;
61          xReqManualPulse                : BOOL := FALSE;
62          xReqAutoPulse                        : BOOL   := FALSE;
63          xReqRemoteMobilePulse          : BOOL := FALSE;
64
65
```

```
66          xPendingManualLatch                 : BOOL := FALSE;
67          xPendingRemoteLatch                 : BOOL := FALSE;
68          xPendingAutoLatch                       : BOOL := FALSE;
69          xPendingRemoteMobileLatch         : BOOL := FALSE;
70
71  END_VAR
72
73  //Detect when a request signal goes from FALSE to TRUE
74  //Ensures each request is only processed once per button press
75  xReqRemotePulse := ReqRemote AND NOT xReqRemoteOld;
76  xReqManualPulse := ReqManual AND NOT xReqManualOld;
77  xReqAutoPulse := ReqAuto AND NOT xReqAutoOld;
78  xReqRemoteMobilePulse := ReqRemoteMobile AND NOT xReqRemoteMobileOld;
79
80  //Store previous signal state for next scan's edge detection
81  xReqRemoteOld := ReqRemote;
82  xReqManualOld := ReqManual;
83  xReqAutoOld := ReqAuto;
84  xReqRemoteMobileOld := ReqRemoteMobile;
85
86  //Safety guard - if active loses heartbeat force FAILSAFE immediately
87  //This check runs every scan before the state machine, acting as an interrupt
88  IF (Active = EControlMode.REMOTE AND NOT HB_Remote_OK) OR
89          (Active = EControlMode.MANUAL AND NOT HB_Manual_OK) OR
90          (Active = EControlMode.AUTO AND NOT HB_Auto_OK) OR
91          (Active = EControlMode.REMOTE_MOBILE AND NOT HB_Remote_Mobile_OK) THEN
92          Active          := EControlMode.FAILSAFE;
93          Pending := EPendingMode.NONE;
94          State          := ETransitionState.FAILSAFE;
95          GrantedManual                   := FALSE;
96          GrantedRemote                   := FALSE;
97          GrantedAuto                   := FALSE;
98          GrantedRemoteMobile := FALSE;
99          RETURN;
100 END_IF
101
102 //Safety guard - if IntegrityCheck detected invalid signal combination force
    ↪  FAILSAFE
103 //Prevents contradictory signals from reaching decision logic
104 IF NOT Valid THEN
105         Active                          :=  EControlMode.FAILSAFE;
106         Pending                   := EPendingMode.NONE;
107         State                           := ETransitionState.FAILSAFE;
108         PendingReady          := FALSE;
109         Released                  := FALSE;
110         tAckTimer(IN := FALSE, PT := MaxAckTime);
111         GrantedManual := FALSE;
112         GrantedRemote := FALSE;
113         GrantedAuto := FALSE;
114         GrantedRemoteMobile := FALSE;
115         RejectedRequestFlag := TRUE;
116         RejectionReason := 'FAILSAFE: INVALID_SIGNAL';
117         RETURN;
118 END_IF
119
120 //Main state machine - controls transitions between control stations
```

```
121 //States: INIT -> STABLE <-> PRE_TAKEOVER <-> VALIDATE_TAKEOVER, any state ->
    ↪  FAILSAFE
122 CASE State OF
123         //INIT - system startup, waiting for the first valid request
124         //System remains here until a station with valid heartbeat requests
            ↪  control
125         ETransitionState.INIT:
126                 IF xReqManualPulse AND HB_Manual_OK THEN
127             Active := EControlMode.MANUAL;
128             State  := ETransitionState.STABLE;
129         ELSIF xReqRemotePulse AND HB_Remote_OK THEN
130             Active := EControlMode.REMOTE;
131             State  := ETransitionState.STABLE;
132         ELSIF xReqAutoPulse AND HB_Auto_OK THEN
133             Active := EControlMode.AUTO;
134             State  := ETransitionState.STABLE;
135                 ELSIF xReqRemoteMobilePulse AND HB_Remote_Mobile_OK THEN
136                         Active := EControlMode.REMOTE_MOBILE;
137                         State := ETransitionState.STABLE;
138         END_IF
139
140         // FAILSAFE - terminal error state, no transitions allowed
141         // Entered when heartbeat is lost, signals are invalid or internal
            ↪  inconsistency detected
142         //Requires manual intervention to recover
143         ETransitionState.FAILSAFE:
144                 Pending               := EPendingMode.NONE;
145                 PendingReady        := FALSE;
146                 Released                 := FALSE;
147                 tAckTimer(IN := FALSE, PT := MaxAckTime);
148
149         //STABLE - normal operating state, one station is active, no transition
            ↪  in progress
150         // Resets rejection flags and waits for a new takeover request
151         ETransitionState.STABLE:
152                 Pending := EPendingMode.NONE;
153                 tAckTimer(IN := FALSE, PT := MaxAckTime);
154                 RejectedRequestFlag := FALSE;
155                 RejectionReason := '';
156
157                 //Priority order for simultaneous requests: MANUAL > REMOTE >
                    ↪  AUTO > REMOTE_MOBILE
158                 //Documented in ADR list- simultaneous requests from different
                    ↪  stations
159                 IF xReqManualPulse THEN
160                         xPendingManualLatch := TRUE;
161                         State := ETransitionState.PRE_TAKEOVER;
162                 ELSIF xReqRemotePulse THEN
163                         xPendingRemoteLatch := TRUE;
164                         State := ETransitionState.PRE_TAKEOVER;
165                 ELSIF xReqAutoPulse THEN
166                         xPendingAutoLatch := TRUE;
167                         State := ETransitionState.PRE_TAKEOVER;
168                 ELSIF xReqRemoteMobilePulse THEN
169                         xPendingRemoteMobileLatch := TRUE;
170                         State := ETransitionState.PRE_TAKEOVER;
171                 END_IF
```

```
172
173         //PRE_TAKEOVER - validates whether a takeover request can proceed
174         //Checks that requesting station is not already active and has valid
            ↪ heartbeat
175         //Rejected request are logged with reason
176         ETransitionState.PRE_TAKEOVER:
177                 IF xPendingManualLatch AND Active <> EControlMode.MANUAL AND
                    ↪ HB_Manual_OK THEN
178                         Pending := EPendingMode.PENDING_MANUAL;
179                         State         := ETransitionState.VALIDATE_TAKEOVER;
180                 ELSIF xPendingRemoteLatch AND Active <> EControlMode.REMOTE AND
                    ↪ HB_Remote_OK THEN
181                         Pending := EPendingMode.PENDING_REMOTE;
182                         State         := ETransitionState.VALIDATE_TAKEOVER;
183                 ELSIF xPendingAutoLatch AND Active <> EControlMode.AUTO AND
                    ↪ HB_Auto_OK THEN
184                         Pending := EPendingMode.PENDING_AUTO;
185                         State         := ETransitionState.VALIDATE_TAKEOVER;
186                 ELSIF xPendingRemoteMobileLatch AND Active <>
                    ↪ EControlMode.REMOTE_MOBILE AND HB_Remote_Mobile_OK THEN
187                         Pending := EPendingMode.PENDING_REMOTE_MOBILE;
188                         State        := ETransitionState.VALIDATE_TAKEOVER;
189                 ELSE
190                         //Request rejected - log reason for decision logger
191                         Pending := EPendingMode.NONE;
192                         State := ETransitionState.STABLE;
193                         RejectedRequestFlag := TRUE;
194                         IF (xPendingManualLatch AND NOT HB_Manual_OK) OR
195                                 (xPendingRemoteLatch AND NOT HB_Remote_OK) OR
196                                 (xPendingAutoLatch AND NOT HB_Auto_OK) OR
197                                 (xPendingRemoteMobileLatch AND NOT
                                    ↪ HB_Remote_Mobile_OK) THEN
198                                         RejectionReason := 'Request rejected:
                                            ↪ Heartbeat failure';
199                         ELSIF (xPendingManualLatch AND Active =
                            ↪ EControlMode.MANUAL) OR
200                                 (xPendingRemoteLatch AND Active =
                                    ↪ EControlMode.REMOTE) OR
201                                 (xPendingAutoLatch AND Active =
                                    ↪ EControlMode.AUTO) OR
202                                 (xPendingRemoteMobileLatch AND Active =
                                    ↪ EControlMode.REMOTE_MOBILE) THEN
203                                         RejectionReason := 'Request rejection:
                                            ↪ Station already active';
204                         ELSE
205                                         RejectionReason := 'Request rejected:
                                            ↪ Conflicting request';
206                         END_IF
207                 END_IF
208
209                 //Clear all latches regardless of outcome - each request is
                    ↪ only evaluated once
210                 xPendingManualLatch := FALSE;
211                 xPendingRemoteLatch := FALSE;
212                 xPendingAutoLatch         := FALSE;
213                 xPendingRemoteMobileLatch := FALSE;
214
```

```
215
216         //VALIDATE_TAKEOVER - handshake phase, waiting for release from active
            ↪  and ack from pending
217         //Transition completes when active station releases and pending station
            ↪  acknowledges
218         //Aborted if heartbeat is lost or acknowledgement timer expires
219         ETransitionState.VALIDATE_TAKEOVER:
220
221                 ReleaseRequest_Manual := (Active = EControlMode.MANUAL);
222                 ReleaseRequest_Remote := (Active = EControlMode.REMOTE);
223                 ReleaseRequest_Auto := (Active = EControlMode.AUTO);
224                 ReleaseRequest_Remote_Mobile := (Active =
                    ↪  EControlMode.REMOTE_MOBILE);
225
226                 tAckTimer(IN := TRUE, PT := MaxAckTime);
227
228                 //Stop transition if the new station loses connection or
                    ↪  cancels its request
229                 IF (Pending = EPendingMode.PENDING_MANUAL AND (NOT ReqManual OR
                    ↪  NOT HB_Manual_OK)) OR
230                         (Pending = EPendingMode.PENDING_REMOTE AND (NOT
                            ↪  ReqRemote OR NOT HB_Remote_OK)) OR
231                         (Pending = EPendingMode.PENDING_AUTO AND (NOT ReqAuto
                            ↪  OR NOT HB_Auto_OK)) OR
232                         (Pending = EPendingMode.PENDING_REMOTE_MOBILE AND (NOT
                            ↪  ReqRemoteMobile OR NOT HB_Remote_Mobile_OK)) THEN
233                         Pending := EPendingMode.NONE;
234                         State := ETransitionState.STABLE;
235                         tAckTimer (IN := FALSE, PT := MaxAckTime); // Reset
                            ↪  timer
236                         PendingReady := FALSE;
237                         Released := FALSE;
238                         ReleaseRequest_Manual := FALSE;
239                         ReleaseRequest_Remote := FALSE;
240                         ReleaseRequest_Auto := FALSE;
241                         ReleaseRequest_Remote_Mobile := FALSE;
242                         RejectedRequestFlag := TRUE;
243                         RejectionReason := 'Request rejected: Heartbeat loss
                            ↪  during handover';
244
245                 //Abort - pending station did not acknowledge within allowed
                    ↪  time
246                 ELSIF tAckTimer.Q THEN
247                         Pending := EPendingMode.NONE;
248                         State := ETransitionState.STABLE;
249                         tAckTimer(IN := FALSE, PT := MaxAckTime);
250                         PendingReady := FALSE;
251                         Released := FALSE;
252                         ReleaseRequest_Manual := FALSE;
253                         ReleaseRequest_Remote := FALSE;
254                         ReleaseRequest_Auto := FALSE;
255                         ReleaseRequest_Remote_Mobile := FALSE;
256                         RejectedRequestFlag := TRUE;
257                         RejectionReason := 'Request rejected: acknowledgement
                            ↪  timeout';
258
259                 ELSE
```

```
260                     //Read release signal from active station
261                     Released := FALSE;
262                     CASE Active OF
263                             EControlMode.MANUAL: Released := RelManual;
264                             EControlMode.REMOTE: Released := RelRemote;
265                             EControlMode.AUTO: Released := RelAuto;
266                             EControlMode.REMOTE_MOBILE: Released :=
                                ↪  RelRemoteMobile;
267                     END_CASE
268
269                     //Read acknowledgement from pending station
270                     PendingReady := FALSE;
271                     CASE Pending OF
272                             EPendingMode.PENDING_MANUAL: PendingReady :=
                                ↪  AckManual;
273                             EPendingMode.PENDING_REMOTE: PendingReady :=
                                ↪  AckRemote;
274                             EPendingMode.PENDING_AUTO: PendingReady :=
                                ↪  AckAuto;
275                             EPendingMode.PENDING_REMOTE_MOBILE:
                                ↪  PendingReady := AckRemoteMobile;
276                     END_CASE
277
278                     //Safety check: the new station must not be the same as
                        ↪  the current one
279                     //PRE_TAKEOVER prevents this, but if reached here it
                        ↪  indicates internal fault
280                     IF (Pending = EPendingMode.PENDING_MANUAL AND Active =
                        ↪  EControlMode.MANUAL) OR
281                             (Pending = EPendingMode.PENDING_REMOTE AND
                                ↪  Active = EControlMode.REMOTE) OR
282                             (Pending = EPendingMode.PENDING_AUTO AND Active
                                ↪  = EControlMode.AUTO) OR
283                             (Pending = EPendingMode.PENDING_REMOTE_MOBILE
                                ↪  AND Active = EControlMode.REMOTE_MOBILE)
                                ↪  THEN
284                                     Active                          :=
                                        ↪  EControlMode.FAILSAFE;
285                                     Pending                 :=
                                        ↪  EPendingMode.NONE;
286                                     State                       :=
                                        ↪  ETransitionState.FAILSAFE;
287                                     PendingReady          := FALSE;
288                                     Released                := FALSE;
289                                     tAckTimer(IN := FALSE, PT :=
                                        ↪  MaxAckTime);
290                                     GrantedManual         := FALSE;
291                                     GrantedRemote         := FALSE;
292                                     GrantedAuto         := FALSE;
293                                     GrantedRemoteMobile := FALSE;
294                                     ReleaseRequest_Manual := FALSE;
295                                     ReleaseRequest_Remote := FALSE;
296                                     ReleaseRequest_Auto := FALSE;
297                                     ReleaseRequest_Remote_Mobile := FALSE;
298                                     RejectedRequestFlag := TRUE;
```

```
299                                     RejectionReason := 'Request rejected:
                                        ↪  Integrity violation, pending
                                        ↪  matches active';
300                                     RETURN;
301                             END_IF
302
303                             //Complete transition - both conditions met
304                             IF Released AND PendingReady THEN
305                                     CASE Pending OF
306                                     EPendingMode.PENDING_MANUAL: Active :=
                                        ↪  EControlMode.MANUAL;
307                                     EPendingMode.PENDING_REMOTE: Active :=
                                        ↪  EControlMode.REMOTE;
308                                     EPendingMode.PENDING_AUTO: Active :=
                                        ↪  EControlMode.AUTO;
309                                     EPendingMode.PENDING_REMOTE_MOBILE:
                                        ↪  Active :=
                                        ↪  EControlMode.REMOTE_MOBILE;
310                                     END_CASE
311                                     Pending := EPendingMode.NONE;
312                                     State := ETransitionState.STABLE;
313                                     tAckTimer(IN := FALSE, PT := MaxAckTime);
314                                     PendingReady := FALSE;
315                                     Released := FALSE;
316                                     ReleaseRequest_Manual := FALSE;
317                                     ReleaseRequest_Remote := FALSE;
318                                     ReleaseRequest_Auto := FALSE;
319                                     ReleaseRequest_Remote_Mobile := FALSE;
320                             END_IF
321             END_IF
322     END_CASE
323
324     //Grant signals - updated each scan based on current pending mode
325     //Stations use these to know their request has been approved
326     GrantedManual := (Pending = EPendingMode.PENDING_MANUAL);
327     GrantedRemote := (Pending = EPendingMode.PENDING_REMOTE);
328     GrantedAuto   := (Pending = EPendingMode.PENDING_AUTO);
329     GrantedRemoteMobile := (Pending = EPendingMode.PENDING_REMOTE_MOBILE);
330
```

C.6 FB_HeartBeat och FB_HeartBeatGenerator

```
FUNCTION_BLOCK FB_HeartBeat
VAR_INPUT
        xSignal : BOOL;
END_VAR
VAR_OUTPUT
        xOk         : BOOL;
END_VAR
VAR
        tonTimeout        : TON;
        xPrevSignal : BOOL;
        xPulse         : BOOL;
END_VAR

xPulse := (xSignal <> xPrevSignal);
xPrevSignal := xSignal;
// Om ingen ny puls kommer, låt timern gå
tonTimeout(IN := NOT xPulse, PT := GVL_HeartBeat.HB_Monitor_Timeout);
xOk := NOT tonTimeout.Q;
```

```
FUNCTION_BLOCK FB_HeartBeatGenerator
VAR_INPUT
END_VAR
VAR_OUTPUT
        xSignal : BOOL;
END_VAR
VAR
        tonToggle : TON;
        xEnable : BOOL := TRUE;
END_VAR

//Start timer
tonToggle(IN := xEnable, PT := T#200MS);

//När timer gått, växla signal
IF tonToggle.Q THEN
        xSignal                 := NOT xSignal;
        tonToggle(IN         := FALSE, PT :=
        ↪  GVL_HeartBeat.HB_Generator_Interval);
        xEnable                 := TRUE;
END_IF;

//Heartbeat-generatorns intervall och heartbeat-monitorns
//timeout är två separata värden utan formell koppling.
// Om något värde ändras utan att det andra justeras kan
// heartbeat-detektionen sluta fungera.

```

C.7 DecisionLogger

```
FUNCTION_BLOCK DecisionLogger
VAR_INPUT
        Active : EControlMode;
        Pending : EPendingMode;
        State : ETransitionState;
        RejectedRequestFlag : BOOL;
        RejectionReason : STRING;
END_VAR
VAR_OUTPUT
END_VAR
VAR
        LastActive : EControlMode;
        LastPending : EPendingMode;
        LastState : ETransitionState;
        LastRejectedRequest : BOOL;


END_VAR

// LastActiveBeforeTransition is updated before Active changes,
// preserving the previous station for traceability purposes.
IF Active <> LastActive THEN
                GVL_State.LastActiveBeforeTransition := LastActive;
END_IF
IF Active <> LastActive OR Pending <> LastPending OR State <> LastState THEN
        GVL_State.Active := Active;
        GVL_State.Pending := Pending;
        GVL_State.State := State;

        LastActive := Active;
         LastPending := Pending;
        LastState := State;
END_IF

IF RejectedRequestFlag AND NOT LastRejectedRequest THEN
        GVL_State.RejectedRequestFlag := TRUE; //log the rejection
        GVL_State.RejectionReason := RejectionReason;
END_IF
LastRejectedRequest := RejectedRequestFlag;
```

C.8 MonitoringChannel och ShutdownService

```
FUNCTION_BLOCK MonitoringChannel
// Monitors heartbeat signals from all stations via EVL.
// EmergencyShutdown is only triggered when ALL stations lose heartbeat
// simultaneously, indicating a total system failure. This is distinct
// from FAILSAFE in ControlUnit, which triggers on individual faults.
VAR_INPUT
        HB_Manual_OK : BOOL;
        HB_Remote_OK : BOOL;
        HB_Auto_OK          : BOOL;
        HB_Remote_Mobile_OK : BOOL;


END_VAR
VAR_OUTPUT
        EmergencyShutdown : BOOL;
END_VAR
VAR
        HB_Manual_Fail : BOOL;
        HB_Remote_Fail : BOOL;
        HB_Auto_Fail   : BOOL;
        HB_Remote_Mobile_Fail : BOOL;
END_VAR

HB_Manual_Fail := NOT HB_Manual_OK;
HB_Remote_Fail := NOT HB_Remote_OK;
HB_Auto_Fail   := NOT HB_Auto_OK;
HB_Remote_Mobile_Fail := NOT HB_Remote_Mobile_OK;

IF HB_Manual_Fail AND HB_Remote_Fail AND HB_Auto_Fail AND HB_Remote_Mobile_Fail
↪  THEN
        EmergencyShutdown := TRUE;
END_IF

FUNCTION_BLOCK ShutdownService
// Alert is latched TRUE and cannot be reset during runtime.
// Recovery requires a full system restart to ensure the fault is acknowledged.
VAR_INPUT
        EmergencyShutdown : BOOL;
END_VAR
VAR_OUTPUT
END_VAR
VAR
        Alert : BOOL;
END_VAR

//Once triggered, the alert stays TRUE until system restart

IF EmergencyShutdown THEN
        Alert := TRUE;
END_IF

```

D

Testning

Detta appendix sammanställer de olika testfall som ligger till grund för verifieringen av systemets komponenter och dess samspel. Fokus låg på att verifiera systemets övergångslogik, signalhantering och fail-safe-beteende vid normal drift men även under felfall samt en korrekt utgångshantering.

D.1 Enhetstestning

Enhetstester gjordes endast på systemets `ControlUnit`, `InterityCheck` och `OutputProcessing`. Detta eftersom dessa komponenter främst ansvarar för systemets centrala beslutslogik, signalvalidering och utgångshantering. Dessa bedömdes som särskilt kritiska då fel i deras funktionalitet har en direkt påverkan på övergångar mellan körstationer, fail-safe-beteenden samt den övergripande säkerheten hos systemet.

D.1.1 ControlUnit

ID	Testfall	Indata	Förväntad utdata	Faktisk utdata
T-01	Verifierar att systemet övergår från det initiella tillståndet till manuell styrning	(1.0) Active=INIT, State=INIT, Pending=NONE (1.1) ReqManual=TRUE	(1.1) Active=MANUAL, State=STABLE, Pending=NONE	(1.1) Active=MANUAL, State=STABLE, Pending=NONE
T-02	Verifierar att en reset återställer systemet från det manuella tillståndet till det initiella	(2.0) Active=MANUAL, State=STABLE, Pending=NONE (2.1) Reset=TRUE	(2.1) Active=INIT, State=INIT, Pending=NONE	(2.1) Active=INIT, State=INIT, Pending=NONE
T-03	Verifierar att en körstations begäran blir nekad vid saknad av heartbeat i det initiella tillståndet	(3.0) Active=INIT, State=INIT (3.1) ReqRemote=TRUE, HB_Remote_OK=FALSE	(3.1) Active=INIT, State=INIT, Pending=NONE	(3.1) Active=INIT, State=INIT, Pending=NONE
T-04	Verifierar att systemet går till FAILSAFE vid ogiltiga insignaler i det initiella tillståndet	(4.0) Active=INIT, State=INIT (4.1) Valid=FALSE	(4.1) Active=FAILSAFE, State=FAILSAFE, Pending=NONE, RejectedRequestFlag=TRUE	(4.1) Active=FAILSAFE, State=FAILSAFE, Pending=NONE, RejectedRequestFlag=TRUE
T-05	Verifierar att ett byte från manuell till remote styrning genomförs korrekt	(5.0) Active=MANUAL, State=STABLE (5.1) ReqRemote=TRUE (5.2) ReqRemote=TRUE (5.3) ReqRemote=TRUE, RelManual=TRUE, AckRemote=TRUE	(5.2) Pending=PENDING_REMOTE, GrantedRemote=TRUE (5.3) Active=REMOTE, State=STABLE, Pending=NONE	(5.2) Pending=PENDING_REMOTE, GrantedRemote=TRUE (5.3) Active=REMOTE, State=STABLE, Pending=NONE
T-06	Verifierar att ett byte från manuell till remote styrning nekas vid utebliven bekräftelse från remote	(6.0) Active=MANUAL, State=STABLE (6.1) ReqRemote=TRUE (6.2) ReqRemote=TRUE (6.3) ReqRemote=TRUE, RelManual=TRUE, AckRemote=FALSE (6.4) ReqRemote=TRUE, RelManual=TRUE, AckRemote=FALSE	(6.2) Pending=PENDING_REMOTE, GrantedRemote=TRUE (6.4) Active=MANUAL, State=STABLE, Pending=NONE, RejectionRequestFlag=TRUE	(6.2) Pending=PENDING_REMOTE, GrantedRemote=TRUE (6.4) Active=MANUAL, State=STABLE, Pending=NONE, RejectionRequestFlag=TRUE
T-07	Verifierar att systemet går till failsafe vid förlust av ett heartbeat i den aktiva styrkällan under drift	(7.0) Active=REMOTE, State=STABLE (7.1) HB_Remote_OK=FALSE	(7.1) Active=FAILSAFE, State=FAILSAFE, Pending=NONE	(7.1) Active=FAILSAFE, State=FAILSAFE, Pending=NONE
T-08	Verifierar att systemet går till failsafe vid ogiltiga insignaler under drift	(8.0) Active=MANUAL, State=STABLE (8.1) Valid=FALSE	(8.1) Active=FAILSAFE, State=FAILSAFE, Pending=NONE, RejectionRequestFlag=TRUE	(8.1) Active=FAILSAFE, State=FAILSAFE, Pending=NONE, RejectionRequestFlag=TRUE
T-09	Verifierar att ett övertagande avbryts om begäran återtas under VALIDATE_TAKEOVER	(9.0) Active=MANUAL, State=STABLE (9.1) ReqRemote=TRUE (9.2) ReqRemote=TRUE (9.3) ReqRemote=FALSE	(9.2) Pending=PENDING_REMOTE, GrantedRemote=TRUE (9.3) Active=MANUAL, State=STABLE, Pending=NONE, RejectedRequestFlag=TRUE	(9.2) Pending=PENDING_REMOTE, GrantedRemote=TRUE (9.3) Active=MANUAL, State=STABLE, Pending=NONE, RejectedRequestFlag=TRUE
T-10	Verifierar att ett övertagande avbryts vid heartbeat-förlust under VALIDATE_TAKEOVER	(10.0) Active=MANUAL, State=STABLE (10.1) ReqRemote=TRUE (10.2) ReqRemote=TRUE (10.3) ReqRemote=TRUE, HB_Remote_OK=FALSE	(10.2) Pending=PENDING_REMOTE, GrantedRemote=TRUE (10.3) Active=MANUAL, State=STABLE, Pending=NONE, RejectedRequestFlag=TRUE	(10.2) Pending=PENDING_REMOTE, GrantedRemote=TRUE (10.3) Active=MANUAL, State=STABLE, Pending=NONE, RejectedRequestFlag=TRUE

Tabell D.1: Testtabell med samtliga enhetstester för `ControlUnit`

D.1.2 IntegrityCheck

ID	Testfall	Indata	Förväntad utdata	Faktisk utdata
T-01	Verifierar att giltiga insignaler passerar IntegrityCheck utan att bli modifierade	(1.0) ReqManual=TRUE, övriga requests=FALSE	(1.0) Valid=TRUE, ReqManual_Out=TRUE	(1.0) Valid=TRUE, ReqManual_Out=TRUE
T-02	Verifierar att flera request-signaler som skickas samtidigt detekteras och resulterar i att valideringen inte går igenom	(2.0) ReqManual=TRUE, ReqRemote=TRUE	(2.0) Valid=FALSE, ReqManual_Out=FALSE, ReqRemote_Out=FALSE	(2.0) Valid=FALSE, ReqManual_Out=FALSE, ReqRemote_Out=FALSE
T-03	Verifierar att en körstation inte kan släppa och begära kontroll samtidigt	(3.0) ReqManual=TRUE, RelManual=TRUE	(3.0) Valid=FALSE, ReqManual_Out=FALSE, RelManual_Out=FALSE	(3.0) Valid=FALSE, ReqManual_Out=FALSE, RelManual_Out=FALSE

Tabell D.2: Testtabell med samtliga enhetstester för `IntegrityCheck`

D.1.3 OutputProcessing

ID	Testfall	Indata	Förväntad utdata	Faktisk utdata
T-01	Verifierar att alla utgångar inaktiveras i det initiella tillståndet	(1.0) Active=INIT, State=INIT	(1.0) OUT_ActiveManual=FALSE, OUT_GrantedManual=FALSE, OUT_Pending=FALSE	(1.0) OUT_ActiveManual=FALSE, OUT_GrantedManual=FALSE, OUT_Pending=FALSE
T-02	Verifierar att korrekt aktiv utgång sätts vid det manuella läget.	(2.0) Active=MANUAL, State=STABLE	(2.0) OUT_ActiveManual=TRUE, OUT_ActiveRemote=FALSE, OUT_State_Stable=TRUE	(2.0) OUT_ActiveManual=TRUE, OUT_ActiveRemote=FALSE, OUT_State_Stable=TRUE
T-03	Verifierar att heartbeat-förlust indikeras korrekt för remote körstation	(3.0) Active=REMOTE, State=STABLE, HB_Remote_OK=FALSE	(3.0) OUT_NoHB_Remote=TRUE	(3.0) OUT_NoHB_Remote=TRUE

Tabell D.3: Testtabell med samtliga enhetstester för `OutputProcessing`

D.2 Integrationstestning

Fokuset för integrationstester är att komponenterna interagerar på ett korrekt sätt för att systemet som helhet ska fungera korrekt, därför testar integrationstesterna att `InputProcessing`, `IntegrityCheck`, `ControlUnit` och `OutputProcessing` samverkar korrekt. Mer specifikt testar integrationstesterna att:

- **InputProcessing** tar in inputs från stationerna och skickar vidare till `integritycheck`.
- **IntegrityCheck** verifierar att signalerna som kommer in från `InputProcessing` är giltiga och skickar vidare till `ControlUnit`.
- **ControlUnit** tar emot från `IntegrityCheck`, gör dess beräkningar och skickar vidare till `OutputProcessing`.
- **OutputProcessing** tar emot systemets interna beslut från `ControlUnit` och ger ut faktiska syrsignaler.

ID	Testfall	Indata	Förväntad utdata	Faktisk utdata
IT-01	Verifierar att systemet övergår från det initiella tillståndet till manuell styrning	(1.0) Active = INIT, State = INIT, (1.2) ReqManual = True	(1.0) Active = INIT, State =INIT (1.3) Active=Manual, State = STABLE	(1.0) Active = INIT, State =INIT (1.3) Active=Manual, State = STABLE
IT-02	Verifiera att Remote kan ta över kontrollen från manuell	(2.0) Active= MANUAL, State = STABLE, (2.1) ReqRemote = TRUE (2.2) RelManual = TRUE, AckRemote = TRUE	(2.1) Pending = PENDING_REMOTE, GrantedRemote = TRUE, (2.2) Active = REMOTE, State=STABLE	(2.1) Pending = PENDING_REMOTE, GrantedRemote = TRUE, (2.2) Active = REMOTE, State=STABLE
IT-03	Flera förfrågningar om att ta över leder till failsafe	(3.0) Active = Remote, State = STABLE, (3.1) ReqAuto=TRUE, ReqManual=TRUE	(3.0) Active=REMOTE, State= STABLE (3.2) Active=FAILSAFE, State=FAILSAFE	(3.0) Active=REMOTE, State= STABLE (3.2) Active=FAILSAFE, State=FAILSAFE
IT-04	Verifiera att man får timeout när station inte skickar ack	(4.0) Active=MANUAL, State=STABLE, (4.1) ReqRemote=TRUE, (4.2) RelManual=True, AckRemote=FALSE,	(4.0) Active= Manual, State=STABLE (4.1) Pending= PENDING_REMOTE, GrantedRemote=TRUE (4.2) Pending=NONE, State=STABLE, (4.3) Active= MANUAL, GrantedRemote=FALSE	(4.0) Active= Manual, State=STABLE (4.1) Pending= PENDING_REMOTE, GrantedRemote=TRUE (4.2) Pending=NONE, State=STABLE, (4.3) Active= MANUAL, GrantedRemote=FALSE

Tabell D.4: Testtabell för samtliga integrationstester