Implementation av kollisionsvarnare med full autobroms och detektering av fotgängare i AUTOSAR

Mikael Lindén
34164

Kandidatavhandling
Handledare: Johan Ersfolk
Datateknik
Institutionen för informationsteknik
Åbo Akademi
2013

Referat

Mängden datorutrustning i bilar ökar konstant. I samma takt ökar också komplexiteten av datorutrustningen. Nya innovationer i biltekniken gör det allt svårare att skapa en fungerande helhet av all elektronik i bilarna. Som en lösning till detta har ledande tillverkare inom bilindustrin utvecklat en standardiserad mjukvaruarkitektur AUTOSAR (AUTomotive Open System ARchitecture). Denna avhandling presenterar hur en komplex innovation, kollisionsvarnare med full autobroms och detektering av fotgängare (CWAB-PD), implementeras i AUTOSAR. Första delen koncentrerar sig på teorin bakom CWAB-PD och hur den implementeras i en traditionell miljö, samt redogör för funktionaliteten och olika komponenter i AUTOSAR. Andra delen tar upp vilka aspekter måste tas i beaktande då ett traditionellt system implementeras i en standardiserad miljö.

Nyckelord: *AUTOSAR, migration, kollisionsvarnare, autobroms, CWAB-PD, körsäkerhet*

Innehåll

Figurer

1 Introduktion

Coelingh, Eidehall och Bengtsson [1] presenterar i sin artikel en innovation som finns implementerad i några bilmodeller tillverkade av Volvo Cars. Kollisionsvarnare med full autobroms och detektering av fotgängare är ett komplext körsäkerhetssystem bestående av en stor mängd elektroniska styrenheter och sensorer samt aktuatorer kopplade till dem. Mjukvaran på dessa styrenheter är starkt hårdvarukopplade och måste modifieras redan vid minsta förändring av mjukvaran eller systemets logiska modell. Mängden styrenheter ökar komplexiteten av systemet och som naturlig följd av detta ökar komplexiteten av bilens hela elektronikkrets.

Som lösning till den konstant ökande komplexiteten hos bilarnas elektroniska system har ledande tillverkare inom bilindustrin utvecklat en standardiserad mjukvaruarkitektur AUTOSAR. Arkitekturens grundidé är att öka portabilitet och hårdvaruoberoenhet hos bilarnas elektroniska system. Mjukvarukomponenter, som baserar sig på arkitekturen, kan distribueras på olika elektroniska styrenheter och komponenter från olika system kan köras parallellt på samma styrenheter. Som följd av detta kan antalet styrenheter i en bil minskas, vilket leder till märkvärda besparingar i tillverkningskostnader.

I denna avhandling redogörs tekniken och funktionaliteten bakom kollisionsvarnare med full autobroms och detektering av fotgängare. Vidare presenteras mjukvaruarkitekturen AUTOSAR och dess fundamentala innehåll. Detta följs av en teoretisk implementation av körsäkerhetssystemet i AUTOSAR. Avhandlingen avslutas med en konklusion.

2 Bakgrund

I det här kapitlet presenteras hur kollisionsvarnaren med full autobroms är implementerad i en traditionell omgivning, vilka tekniker det baserar sig på samt hurdana krav det finns för systemet. Vidare redogörs grundidéerna och tekniken bakom den standardiserade mjukvaruarkitekturen AUTOSAR.

2.1 CWAB-PD

Kollisionsvarnare med full autobroms och detektion av fotgängare (Collision Warning with Full Auto Brake and pedestrian detection, CWAB-PD) är ett körsäkerhetssystem för bilar utvecklad av Volvo Car Corporation. Systemet grundar sig på kollisionsvarnare (Forward Collision Warning, FCW) och full autobroms (Automatic Emergency Brake, AEB), som tillsammans förebygger påkörningar bakifrån i trafiken. [1]

FCW är ett komplex system som består av tre huvudkomponenter: framåtriktad kamera (Forward Looking Camera, FLC), framåtriktad radar (Forward Looking Radar, FLR) och en separat styrenhet (Forward Sensing Module, FSM). Relevanta objekt framför bilen identifieras som fordon eller fotgängare från videoflödet från FLC. Med information från FLR kalkylerar FSM rörelseriktning och hastighet för varje objekt som identifierats och varnar för eventuell risk för kollision. [1]

Full autobroms (Automatic Emergency Braking, AEB) använder information från avancerade hjälpsystem så som FCW och retarderar bilen autonomiskt eller stöder chauffören i inbromsningen i situationer då det finns överhängande risk för påkörning. Stöd i form av omedelbar maximal bromskraft ges i situationer då chauffören själv hinner observera faran och börjar retardationen. [1]

2.1.1 Matematiska och fysikaliska teorin bakom CWAB-PD

För att kunna använda den data som FLC och FLR skapar, behövs det matematiska och fysikaliska modeller som ligger som grund för risk-kalkylationerna. Modellen som används i CWAB-PD grundar sig på antagandet att båda objekten, värden och målet, har samma rörelseriktning eller att målet har hastighet lika med noll. En viktig faktor i kalkyleringen av kollisionsrisken är tid till kollision (TTK). Ekvationerna nedan beskriver TTK i alla möjliga fall i den ovannämnda modellen. $\tilde{P}_x, \tilde{v}_x$ och $\tilde{a}_x$ står för relativa läget, hastigheten och accelerationen mellan värden och målet.

$$ttk = \begin{cases} -\dfrac{\tilde{p}_x}{\tilde{v}_x}, & \tilde{v}_x < 0 \; och \; \tilde{a}_x = 0 \\ -\dfrac{\tilde{v}_x}{\tilde{a}_x} - \dfrac{\sqrt{\tilde{v}_x^2 - 2\tilde{p}_x \cdot \tilde{a}_x}}{\tilde{a}_x}, & \tilde{v}_x < 0 \; och \; \tilde{a}_x \neq 0 \\ -\dfrac{\tilde{v}_x}{\tilde{a}_x} + \dfrac{\sqrt{\tilde{v}_x^2 - 2\tilde{p}_x \cdot \tilde{a}_x}}{\tilde{a}_x}, & \tilde{v}_x \geq 0 \; och \; \tilde{a}_x < 0 \\ odefinierat & \tilde{v}_x \geq 0 \; och \; \tilde{a}_x \geq 0 \\ odefinierat & \tilde{v}_x - 2\tilde{p}_x \cdot \tilde{a}_x < 0 \end{cases}$$

För att kunna beräkna tidpunkten då inbromsningen senast måste börjas behövs tiden som inbromsningen tar.

$$t_{broms} = -\frac{\tilde{v}_x}{a_{x,värd}}$$

Med hjälp av inbromsningstiden beräknas sträckan som värden fortskrider under inbromsningen.

$$\tilde{p}_{x.broms} = -\frac{a_{x,värd} \cdot t_{broms}^2}{2} = -\frac{\tilde{v}_{x,0}^2}{2a_{x,värd}}$$

Tidpunkten då inbromsningen senast måste börjas fås från ekvationen nedan.

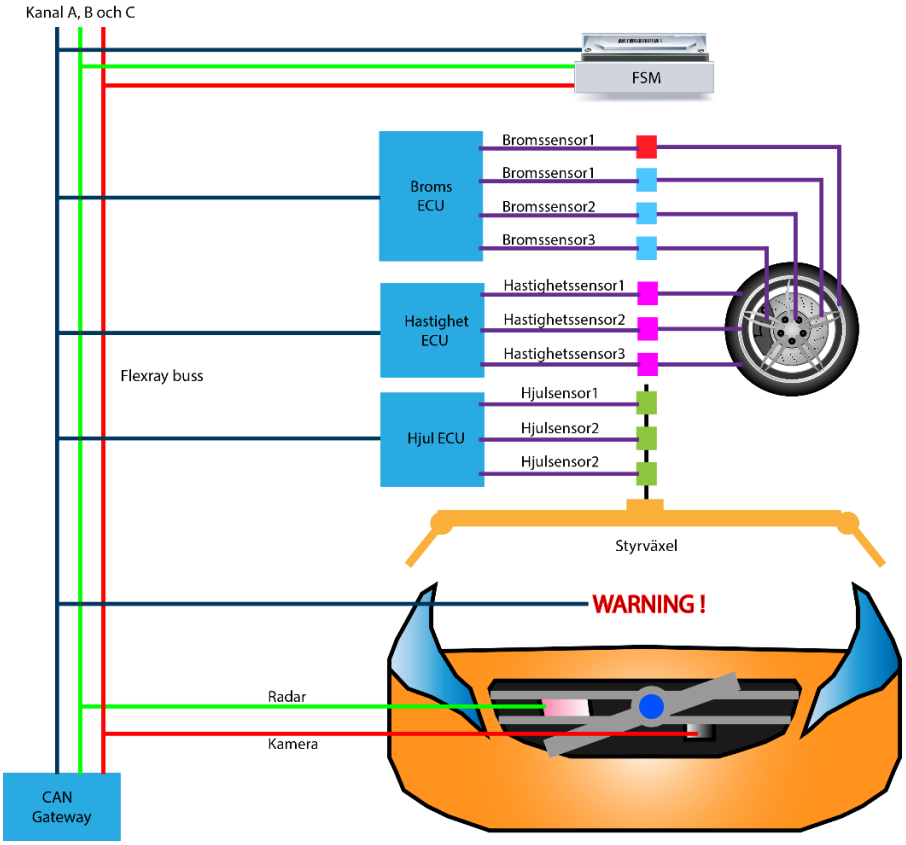
$$ttk_{broms} = -\frac{\tilde{p}_{x.broms}}{\tilde{v}_{x,o}}$$

Förutom inbromsning kan värden också köra runt målet. Senaste tidpunkten då styrmanövern måste börjas fås från följande ekvation. I ekvationen står $a_{y,värd}$ för värdens laterala acceleration, $b_{värd}$ och $b_{mål}$ för respektive bredd för värden och målet. [1]

$$ttk_{styr} = min\left(\sqrt{\frac{2}{a_{y,värd}}\left(y_0 \pm \frac{b_{värd} + b_{mål}}{2}\right)}\right)$$

2.1.2 Traditionell implementation av CWAB-PD

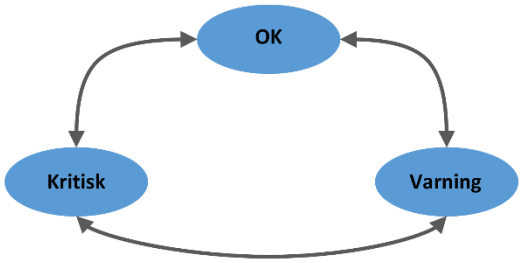
Figuren 2.2 presenterar hur CWAB-PD kan implementeras i en traditionell omgivning. Systemet består av olika elektroniska styrenheter (Electronic Control Unit, ECU), sensorer och aktuatorer som kommunicerar med varandra genom Flexray CAN-bussar och (Control Area Network Bus). De passiva sensorerna och aktuatorerna är kopplade till en och samma CAN-bus, medan data från radarn och kameran överförs genom skilda bussar på grund av datamängden. [1, 2]



Figur 2.1: Traditionell implementation av CWAB-PD.

2.1.3 Säkerhetskrav

CWAB-PD är ett körsäkerhetssystem och kategoriseras per definition som säkerhetskritisk system. Som följd av detta har man byggt in redundans i systemet, vilket skapar feltolerans hos vissa komponenter. Komponenternas och vidare hela systemets driftstatus måste kunna värderas kontinuerligt. För detta ändamål definieras olika operativa tillstånd för varje ECU. Figuren 2.2 redogör de olika tillstånden hos ECU:n [2]



Tillstånd Hastighet-/Hjul-ECU	Tillstånd	Sensorer
OK	0	3
Varning	1	2
Kritisk	2	1

Tillstånd Broms-ECU	Tillstånd	Sensorer	Aktuatorer
OK	0	3	1
Varning	1	2	1
Kritisk	2	1	1
Kritisk	2	1/2/3	0

Tillstånd FSM	Tillstånd	Ha-ECU	B-ECU	Hj-ECU	Kamera	Radar
OK	0	0	0	0	0	0
Varning	1	0/1	0/1	0/1	0	0
Kritisk	2	0/1/2	0/1/2	0/1/2	0/2	0/2

Figur 2.2: De olika tillstånden för ECU:n i CWAB-PD implementerat på traditionellt sätt. [2]

Tillståndet ”OK” står för normalt drifttillstånd hos ECU:n. Alla till ECU:n kopplade sensorerna och aktuatorerna är funktionsdugliga och ECU:n själv fungerar normalt. I ”Varning”-tillståndet har en sensor av de passiva sensorerna slutat fungera enligt fördefinierade krav. ECU:n kan ännu fortsätta normalt men en varningsflagga initieras, vilken sedan fångas av bilens interna diagnostiksystem. Tillståndet ”Kritisk” står för det lägsta möjliga drifttillstånd. ECU:n, och vidare hela systemet, har förlorat sin funktionsduglighet och hela systemet kopplas av. [2]

2.2 AUTOSAR

Under de senaste femton åren har mängden elektroniska styrenheter i bilar ökat kraftigt. Detta är en följd av kraftig ökning av datoriserade funktioner i bilar samt av att styrenheternas mjukvara har varit starkt hårdvarukopplad. Denna starka koppling mellan mjuk- och hårdvara har gjort återanvändningen av mjukvara väldigt svårt eller omöjligt. Samtidigt som mängden ECU:s ökar, ökar också komplexiteten av hela elektronikkretsen. Ytterligare finns det växande krav för systemtäckande kommunikation mellan ECU:s. Som naturlig följd ökar mångfalden av styrenheter och nätverkskomponenter. [3, 4]

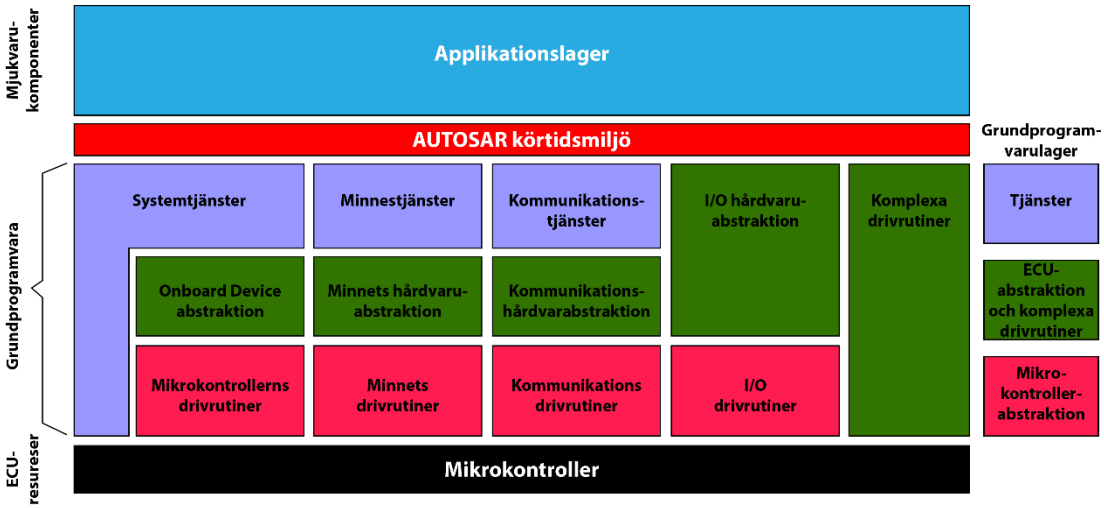
För att lösa ovannämnda problem grundade ledande bil- och komponenttillverkare AUTOSAR (AUTomotive Open System ARchitecture) år 2003. Huvudmålen i projektet formulerades som

- Standardisering och portabilitet av grundprogramvara
- Skalbarhet mellan olika typer av fordon och system
- Integration av olika funktioner i samma modul
- Besparingar som följd av standardisering
- Förenkling av service och underhåll
- Möjliggöra tillväxt och ökad komplexitet i bilarnas datorutrustning [3]

Genom att erbjuda en standardiserad mjukvaruarkitektur samt en standardiserad utvecklingsprocess för utvecklingen av system till det, förenklas samarbetet mellan bil- och komponenttillverkare. När mjukvarans portabilitet ökar, minskar det på utvecklingskostnaderna för ett visst system för en viss bilmodell. Likaså minskar komplexiteten av bilarnas hela elektronikkrets när mängden ECU:n minskar till följd av att olika mjukvarukomponenter kan köras parallellt på en och samma ECU. Mjukvarukomponenternas hårdvaruoberoenhet gör det också möjligt att använda COTS-hårdvara (Commercial-off-the-shelf) istället för beställningsutvecklad hårdvara. Med detta nås avsevärda besparingar i tillverkningskostnaderna. Ytterligare förenklas service och underhåll av bilar, när det är möjligt att använda standardreservdelar.

2.2.1 Mjukvaruarkitektur

AUTOSAR mjukvaruarkitektur består av tre olika lager: från uppifrån neråt finns grundprogramvaran, körtidsmiljön och applikationslagret. Modellen för arkitekturen presenteras i figur 2.3. I applikationslagret implementeras själva mjukvaran för et visst system. Mjukvarukomponenterna i applikationslagret kommunicerar med varandra virtuellt via körtidsmiljön oberoende av om de befinner sig på samma ECU eller inte. Grundprogramvarulagret sköter kommunikationen mellan körtidsmiljön och hårdvaran. [4]



Figur 2.3: AUTOSAR mjukvaruarkitektur. [4]

Grundprogramvara

Grundprogramvarulagret har standardiserad struktur men är inte själv hårdvaruoberoende. Den standardiserade strukturen möjliggör de högre programvarunivåernas hårdvaruoberoende via API (Application Programming Interface). Lagret består av hårdvaruberoende och -oberoende moduler som är vidare indelade i fyra underlager: tjänstelagret, lagret för ECU-abstraktion, lagret för komplexa drivrutiner samt lagret för mikrokontrollerabstraktion. Lagret för mikrokontrollerabstraktion (Microcontroller Abstraction Layer, MCAL) är det nedersta underlagret och består av olika typer av drivrutiner. Drivrutinerna kommunicerar direkt med mikrokontrollernas hårdvara, vilket gör dessa

hårdvaruberoende. Ovanför MCAL ligger lagret för ECU-abstraktion (ECU Abstraction Layer, ECUAL). Modulerna i ECUAL använder MCAL:s gränssnitt för att utnyttja drivrutinernas tjänster och är som följd delvis hårdvaruoberoende. Lagret för mikrokontrollerabstraktion (Complex Drivers Layer, CDL) sträcker sig ända från hårdvarunivån upp till API:n. Den används för att implementera funktioner som inte är stödda i AUTOSAR eller som har strikta tidsbegränsningar. Tjänstelagret (Service Layer, SL) består av systemtjänster i form av operativsystemsfunktioner och diagnostik, nätverkshantering och minnestjänster. [5]

AUTOSAR körtidsmiljö

Mellan grundprogramvaru- och applikationslagret befinner sig AUTOSAR:s standardiserade körtidsmiljö. Den består av en buss (Virtual Functional Bus, VFB) som länkar applikationskomponenterna med varandra, oberoende av om de befinner sig på samma ECU eller inte, samt med de underliggande lagren. Till dess uppgifter hör också schemaläggning av processer. VFB möjliggör applikationslagrets absoluta hårvaruoberoende. [5]

Applikationslagret

Applikationslagret skiljer sig väsentligt från grundprogramvarulagret och körtidsmiljön; den består av ostandardiserade mjukvarukomponenter av tre olika slag: applikationskomponenter, sensor-/aktuatorkomponenter samt kalibreringskomponenter. Kommunikationen mellan dessa komponenter måste dock vara standardiserad intra-ECU- eller inter-ECU-kommunikation (kommunikation innanför ECU:n eller mellan olika ECU:n). I mjukvarukomponenterna implementeras själva funktionaliteten av ett system. För applikationskomponenter är det ingen skillnad på vilken ECU de mappas, medan sensor- och aktuatorkomponenter måste alltid mappas på ECU:n som är kopplad till motsvarande sensor/aktuator. [5]

3 Implementation av CWAB-PD i AUTOSAR

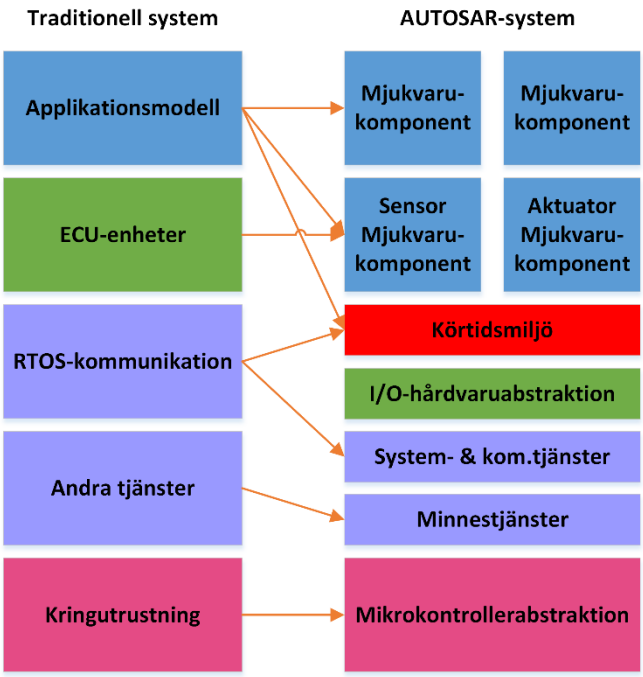
I det här kapitlet presenteras en möjlig implementation av CWAB-PD i den standardiserade mjukvaruarkitekturen AUTOSAR. Utvecklingsprocessen följer AUTOSAR:s utvecklingsmetodik och är i teorin helt genomförbar. Slutprodukten har dock inte validerats eller testats, utan presenteras som ett teoretiskt exempel på en möjlig lösning.

Utveckling av ett system i AUTOSAR är modellbaserat och genomförs via standardiserade utvecklingssteg. I det första steget konstrueras mjukvarukomponenterna för systemet och hur de kommunicerar med varandra via VFB. Detta följs av systemdesign på ECU-nivå; mjukvarukomponenternas krav på ECU:n hanteras och information från denna process används i det följande steget. När mjukvarukomponenternas krav är hanterade, allokeras de på olika ECU:n som passar den specifika mjukvarukomponentens uppgift. I det sista steget konfigureras grundprogramvaran och körtidsmiljön så att de är anpassade för mjukvarukomponenternas krav. [6]

I den nuvarande versionen av AUTOSAR-utvecklingsmetodiken finns det inte ett standardiserat utvecklingssteg för hantering av processernas realtidskrav. I kommande versioner kommer tidskravshanteringen att standardiseras, men i den här avhandlingen hanteras tidskraven enligt rådande kutym, det vill säga tidskraven beaktas i ett relativt sent skede av utvecklingsprocessen. Detta kan dock ha som följd att antalet utvecklingsiterationer ökar om man hittar fel i systemet sent i utvecklingsprocessen. [2]

Migration från ett äldre system skiljer sig givetvis från en helt ny design. Figur 3.1 redogör hur uppgifterna av de olika komponenterna i ett äldre system överförs och allokeras på komponenter i ett AUTOSAR-baserat system. Denna avhandling

koncentrerar sig på applikationsnivån av CWAB-PD. Vi skall alltså se hur delar av ett äldre system omvandlas till mjukvarukomponenter i en AUTOSAR-miljö.

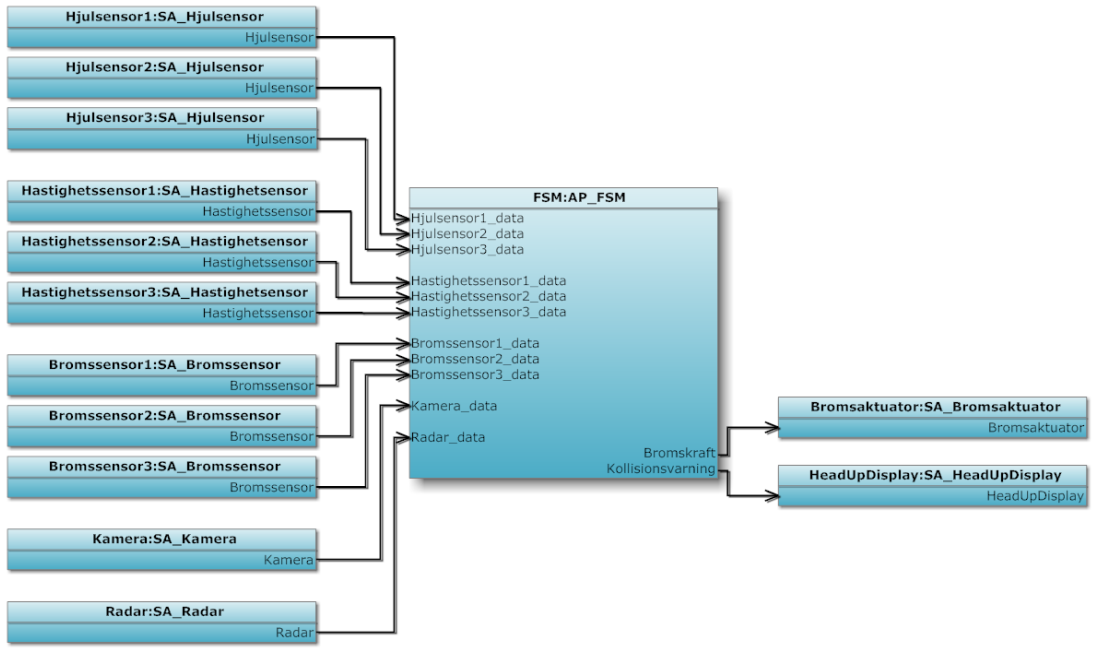


Figur 3.1: Migration från ett traditionellt system till ett AUTOSAR-system. [6]

3.1 Konstruktion av mjukvarukomponenterna

Utvecklingsprocessen initialiseras med att mjukvarukomponenterna för hela systemet konstrueras. Eftersom det i vårt fall är frågan om migration från ett existerande traditionellt implementerat system till ett system implementerat i AUTOSAR, är det första steget att plocka isär alla systemkomponenterna och analysera dem. Applikationsmjukvaran från det äldre systemet kan innehålla grundprogramvarukomponenter så som hårdvaruabstraktioner. Dessa skall avlägsnas, så att den resulterande applikationsmjukvaran bara innehåller logisk information. Den logiska informationen används sedan för att konstruera och definiera de olika mjukvarukomponenterna. Vidare definieras hur de olika komponenterna integreras med varandra via VFB. [2, 6]

Figuren 3.2 presenterar mjukvarukomponenternas sammansättning som en logisk presentation, där sensorkomponenterna ligger till vänster, applikationskomponenten i mitten och aktuatorkomponenterna till höger. Pilarna visar dataflöden mellan de olika komponenterna. Från figuren ses att varje hjul-, hastighet- och bromssensorer har varsin mjukvarukomponent. Kameran och radarn har ytterligare varsin mjukvarukomponent. [2]



Figur 3.2: Mjukvarukomponenternas sammansättning.

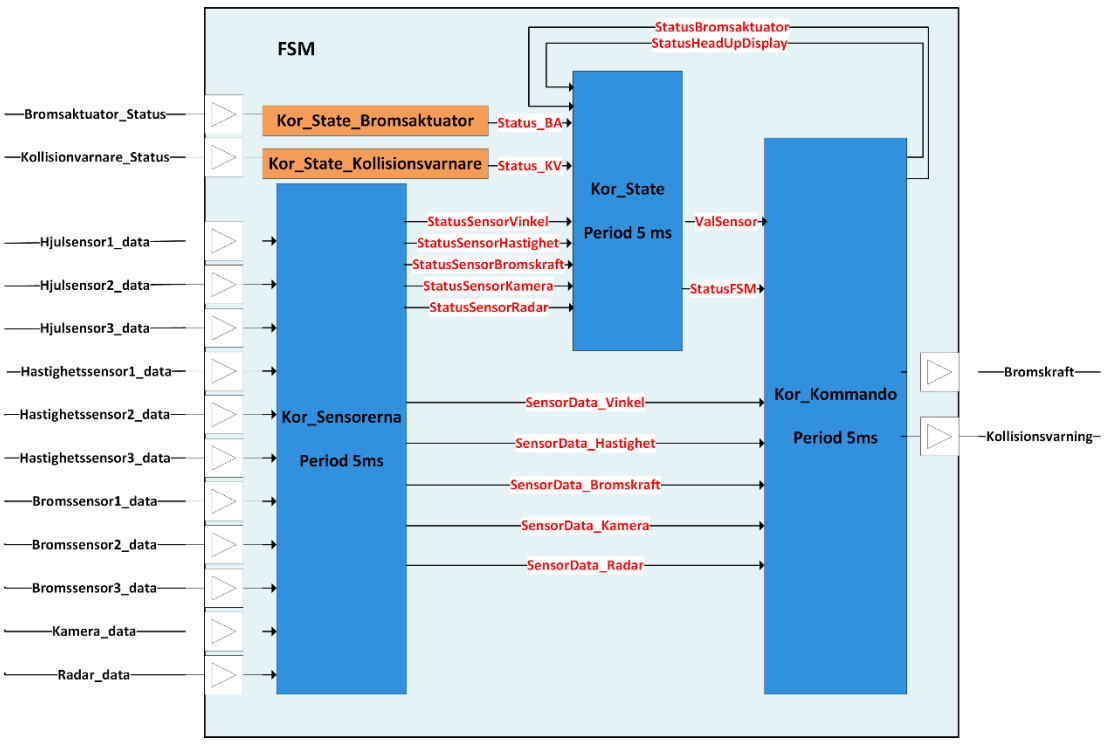
Dataflödena från hjulsensorkomponenterna för information om styrväxelns, eller med andra ord framhjulens, vinkel i relation till bilens horisontalplan. Hastighetssensorkomponenterna använder information om framhjulens vinkelhastighet för att kalkylera bilens hastighet i relation till vägen och skickar denna information vidare till applikationskomponenten. Bromsensorkomponenterna för sin del använder information om trycket i bromssystemet för att kalkylera bromskraften vid en viss tidpunkt. Kamera- och radarkomponenten förser applikationskomponenten med realtidsinformation från varsin sensor.

Från applikationskomponenten kommer det ut två olika dataflöden; signaler till bromsaktuatorkomponenten och head-up-display:n. Signalerna till dessa komponenter

är beroende av chaufförens åtgärder. Vid en risksituation skickas det först signal till head-up-display:n som vidare varnar chauffören om kollision. Ifall chauffören inte reagerar på denna varning, skickar applikationskomponenten bromssignal till bromsaktuatorn och bilen börjar retardera. I situationer då chauffören agerar för långsamt, stöder systemet bromsningen genom att ge maximal bromskraft genast då bromspedalen trycks ner.

3.2 Mjukvarukomponenternas interna struktur

Följande steg i utvecklingsprocessen är att konstruera och definiera vad som händer inuti de olika mjukvarukomponenterna. Den inre strukturen består av olika körbara entiteter som är kopplade med varandra enligt mjukvarukomponentens funktionalitet. Figuren 3.3 redogör FSM:s inre struktur och funktionalitet, det vill säga vilka körbara entiteter komponenten består av och hurdana relationer det finns mellan entiteterna. De blåa kategoriseras som schemalagda entiteter medan orangefärgade är datahändelseentiteter. Alla schemalagda entiteter i FSM har en period på fem millisekunder. [2]



Figur 3.3: FSM:s inre struktur och funktionalitet.

Dataflöden från aktuatorkomponenterna kommer in i FSM via portar som är kopplade till datahändelseentiteterna ”Kor_State_Bromsaktuator” och ”Kor_State_Kollisionsvarnare”. Dessa skickar information om aktuatornas tillstånd till ”Kor_State”-entiteten (”Kor” syftar på ”Kör” och är översatt från engelskans ”Run”).

Dataflöden från sensorkomponenterna kommer in i FSM via portar som är kopplade till den körbara entiteten ”Kor_Sensorerna”. Entiteten skickar vidare information om sensorernas tillstånd till ”Kor_State”-entiteten som använder den för att bestämma tillståndet för hela applikationskomponenten. Ytterligare jämförs data från olika sensorer av samma typ och information om vilken signal som är mest pålitlig vid en viss tidpunkt skickas vidare till ”Kor_Kommando”-entiteten. De olika tillstånden och deras relation till de andra mjukvarukomponenternas tillstånd presenterades i figur 2.2. Om FSM hamnar i ett kritiskt tillstånd kopplas hela systemet av och en varningsflagga initieras. Denna fångas av bilens interna diagnostiksystem. [2, 6]

Entiteten ”Kor_Kommando” behandlar information från kameran och radarn och evaluerar relevanta objekt framför bilen samt bestämmer deras hastighet och rörelseriktning. Denna lägesbild kopplas sedan ihop med informationen från de andra sensorerna för att evaluera risken för kollision. Ifall det uppstår en risksituation, skickar entiteten en signal till kollisionsvarnaren och/eller bromsaktuatorn beroende på situationen och chaufförens agerande. Informationen från ”Kor_State”-entiteten används för att välja de pålitligaste sensordataflöden. [2]

3.3 Allokering av mjukvarukomponenterna

I det här steget allokeras de olika mjukvarukomponenterna till lämpliga ECU:n. Som redan tidigare konstaterades, är det i teorin ingen skillnad på vilken ECU en mjukvarukomponent allokeras; VFB integrationen sköter om kommunikationen mellan mjukvarukomponenterna. Det finns dock vissa restriktioner: sensor- och aktuatorkomponenterna måste allokeras på ECU:n som har direkt förbindelse till motsvarande sensor eller aktuator. [2]

3.4 Konfigurering av grundprogramvaran

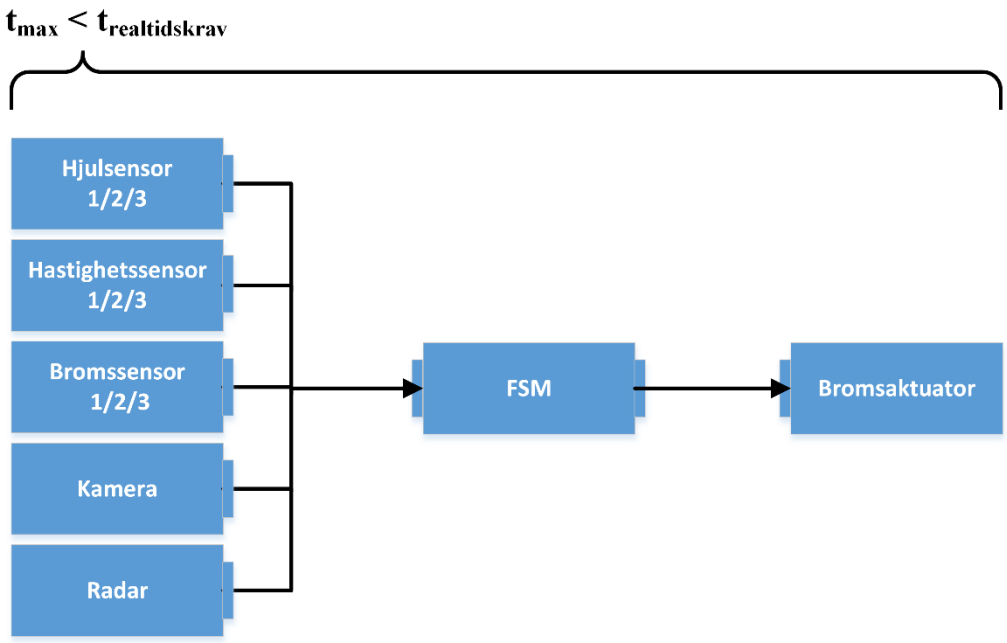
Det sista steget i den standardiserade utvecklingsprocessen innehåller konfigurering av grundprogramvaran. Till detta hör bland annat definiering av uppgifter (tasks) och konfigurering av kommunikationstjänsterna. För FSM definieras tre olika uppgifter: en för ”Kor_Kommando”- och ”Kor_State”-entiteter, en för ”Kor_Sensorerna”-entiteten och en för datahändelseentiteterna. Närmare behandling av konfigureringen av kommunikationstjänsterna lämnas utanför denna avhandling. [2]

3.5 Hantering av tidskrav

Som redan tidigare konstaterades, är CWAB-PD ett säkerhetskritiskt system, vilket har som följd att den måste möta säkerhets- och realtidskrav. Vidare konstaterades att den standardiserade utvecklingsprocessen saknar ett steg var systemets realtidskrav skulle hanteras. För att säkerställa systemets pålitlighet identifierar vi några tidskrav för systemet på olika nivåer. Dessa tidskrav skall valideras genom testning, men detta lämnas utanför avhandlingen.

Realtidskrav på VFB-nivån

På VFB-nivån identifieras åtminstone två olika realtidskrav. Det första är den tid som det tar för bromsaktuatormjukvarukomponenten att agera efter att kameramjukvarukomponenten och radarmjukvarukomponenten har detekterat ett objekt framför bilen. Det andra är den tid som det tar för kollisionsvarnaren att visa en varning efter att kameran och radarn har detekterat ett objekt framför bilen. Dessa tidskrav är svåra att hantera eftersom FSM:s funktionalitet är beroende av alla sensorer. Figuren 3.4 demonstrerar händelsekedjan för det förstnämnda tidskravet. För att systemet kan fungera felfritt i realtid, får tiden t_{max} inte överstiga systemets realtidskrav $t_{realtidskrav}$. Tiden t_{max} är den maximala tiden som det tar för signalen att fara genom hela systemet från sensormjukvarukomponenten till aktuatormjukvarukomponenten. [2]



Figur 3.4: Realtidskrav på VFB-nivå över hela systemet.

Realtidskrav på mjukvarukomponentnivå

På mjukvarukomponentnivån finns det realtidskrav i form av synkroniseringskrav. Applikationsmjukvarukomponenten behöver ett synkroniserat dataflöde från sensorerna för att skapa den redundans som eftersträvas. Ytterligare behövs det synkronisering mellan de körbara entiteterna. ”Kor_Kommando”-entiteten är beroende av information som ”Kor_State” och ”Kor_Sensorerna”. Likaså är ”Kor_State”-entitetens funktionalitet beroende av ”Kor_State_Bromsaktuator”, ”Kor_State_Kollisionsvarnare” och ”Kor_Kommando”. Det är tydligt att elementen inuti mjukvarukomponenterna måste operera synkroniserat för att möjliggöra systemets stabilitet. [2]

Realtidskrav på systemnivå

Realtidskraven på systemnivå liknar kraven på VFB-nivån med vissa skillnader; förutom tiden som det tar för signalen att flöda genom mjukvarukomponenterna behöver man beakta sensorernas och aktuatorernas eftersläpning. Ytterligare beaktas den eftersläpning som orsakas av de fysiska kopplingarna mellan ECU:n och sensorerna. [2]

4 Avslutning

I denna avhandling presenterades en möjlig implementation av körsäkerhetssystem CWAB-PD i den standardiserade mjukvaruarkitekturen AUTOSAR. Utvecklingsprocessen följer de steg som definieras i AUTOSAR:s utvecklingsmetodologi. Som redan tidigare nämnts är den lösning som presenteras i denna avhandling rent teoretisk. Slutprodukten har inte blivit validerad eller testad, och om CWAB-PD i framtiden blir implementerad rent fysiskt i verkligheten kan den skilja sig avsevärt från den som presenteras här. Ytterligare kan konstateras att redundansen som finns inbyggd i systemet till viss mån är paradoksal; systemet klarar av att operera i situationer då enskilda sensorer saknar funktionalitet, men om applikationsmjukvarukomponentens ECU tar skada, förlorar hela systemet sin funktionalitet.

Här bör ännu påpekas att AUTOSAR:s utveckling håller på kontinuerligt och att den version av mjukvaruarkitekturen, som var ute under skrivandet av denna avhandling, är inte en slutgiltig produkt. Till exempel hanteringen av tidskrav är bristfälligt definierad i utvecklingsmetodologin. Av denna orsak har den sköts enligt de principer som presenterades av Kum, Park Lee och Jung i sin artikel [6] om migration från ett äldre system till AUTOSAR. I framtiden kommer arkitekturen dock med stor sannolikhet att säkerställa sin position som den grundläggande standarden för bilarnas mjukvaruarkitektur. Detta garanteras av att de ledande bil- och komponenttillverkarna satsar märkvärda mängder tid och kapital i utvecklingen av standarden. Ytterligare fungerar de sannolika framtida besparingarna i tillverkningskostnaderna som en kraftig drivfaktor i utvecklingen.

Litteratur

[1] Erik Coelingh, Andreas Eidehall och Mattias Bengtsson, “Collision Warning with Full Auto Brake and Pedestrian Detection – a practical example of Automatic Emergency Braking”, i *13th International IEEE Conference on Intelligent Transportation Systems*, ITSC 2010, (Funchal, Madeira, Portugal), sid. 155-160, 19-22 September, 2010.

[2] Khaled Chaaban, Patrick Leserf och Sébastien Saudrais, “Steer-By-Wire System Development Using AUTOSAR Methodology”, i 2009 *IEEE Conference on Emerging Technologies & Factory Automation*, ETFA 2009, (Palma de Mallorca, Spanien), sid. 1110-1117, 22-25 September 2009.

[3] Stefan Voget, Michael Golm, Bernard Sanchez och Friedhelm Stappert, “Application of the AUTOSAR Standard”, *Automotive Embedded Systems Handbook*, sid. 2-1 – 2-25, December 2008.

[4] Stefan Bunzel, “AUTOSAR – the Standardized Software Architecture”, *Informatik-Spektrum*, vol. 34, sid. 79-83, Januari 2011.

[5] Jesper Melin och Daniel Boström, ”Applying AUTOSAR in Practice: Available Development Tools and Migration Paths”, 2011.

[6] Daehyun Kum, Gwang-Min Park, Seonghun Lee och Wooyoung Jung, ”AUTOSAR Migration from Existing Automotive Software”, i *2008 International Conference on Control, Automation and Systems*, ICCAS 2008, (Seoul, Korea), sid. 558-562, 14-17 Oktober, 2008.