SAFAR
EINE FLY-BY-WIRE STEUERERUNG FÜR EIN FLUGZEUG DER GENERAL AVIATION (DIAMOND DA42)

Sebastian Polenz, Florin Cake, Simon Görke, Reinmund Küke*, Reinhard Reichel
Institut für Luftfahrtsysteme (ILS), Universität Stuttgart
*Rheinmetall Defence Electronics GmbH (RDE), Bremen
{sebastian.polenz, florin.cake, simon.goerke, reinhard.reichel}@ils.uni-stuttgart.de

Zusammenfassung

Im Rahmen des EU-Projektes SAFAR (2008-2011, Koordinator RDE) wird für ein Flugzeug der General Aviation, eine Diamond DA42, eine Fly-by-Wire Steuerung entwickelt. Dies ist für pilotierte Flugzeuge ein erster Schritt zu einem so genannten Easy Handling System, welches umfassende autonome Steuerfähigkeiten aufweist und das Steuern von Flugzeugen der hier betrachteten Klasse selbst unter IMC deutlich vereinfacht. Das Fly-by-Wire System wird zurzeit auf seinen Erstflug hin im Testrig am ILS verifiziert und parallel dazu von Diamond Aircraft Industries in eine DA42 eingerüstet. Der Erstflug ist für Mitte Q4 2011 geplant.

Aufgrund der umfassenden Funktion des Fly-by-Wire Systems steuert es alle Steuerklappen und die Triebwerke an, umfasst u.a. ein fehlertolerantes integriertes Attitude/Navigationssystem, redundante Air Data Probes und weist Erweiterungsmöglichkeiten zur Ansteuerung des Fahrwerks einschließlich Bremsen und Lenken auf. Es benötigt nach der vollständigen Qualifikation kein mechanisches Backup.

Flugzeuge der hier betrachteten Klasse unterliegen extremen Kosteneinschränkungen. Dafür entwickelt das ILS seit 2003 für fehlertolerante Netzwerke die Technologie einer flexiblen Plattform, übertragbar auf Fly-by-Wire Systeme. Eine Plattform stellt dabei ein Netzwerk mit flexibler Hardware- (inkl. Sensorik und Aktuatorik), Kommunikations- und Software-Architektur dar. Kernpunkt ist dabei die Middleware, die alle Aufgaben wie Signal-/Netzwerkkommunikation, Fehlermanagement, Herstellung des byzantinischen Agreements sowie das gesamte Plattformbetriebsmanagement übernimmt. Für Applikationen wie die Flugregelung bleiben Netzwerkkomplexität, Fehlertoleranz und Redundanz verborgen. Gegenüber Applikationen bildet die Middleware eine Plattforminstanz als ein virtuelles, nicht verteiltes Simplexsystem ab. Die Applikationen schrumpfen so wieder auf ihre ureigene Aufgabe. Die Komplexität des Netzwerks einschließlich Fehlertoleranz etc. bildet sich in der Middleware ab. Durch die Einführung von Abstraktionsebenen ist es im Rahmen von SAFAR erstmals gelungen, die Middleware so aufzubauen, dass sie sich mittels Spezialisierung generieren (instanzieren) lässt - ein signifikanter Vorteil im Vergleich zu heutigen Fly-by-Wire Entwicklungen. Damit lässt sich der Entwicklungsaufwand für ein Fly-by-Wire System zukünftig signifikant reduzieren – eine absolute Voraussetzung für die hier betrachtete Flugzeugklasse.

Inhalt dieses Beitrags ist eine grundlegende Darstellung des Fly-by-Wire Systems, entwickelt im Rahmen des EU-Projektes SAFAR, und der Grundzüge der Plattformtechnologie.

1. SMALL AIRCRAFT FUTURE AVIONICS ARCITECTURE

1.1. Motivation

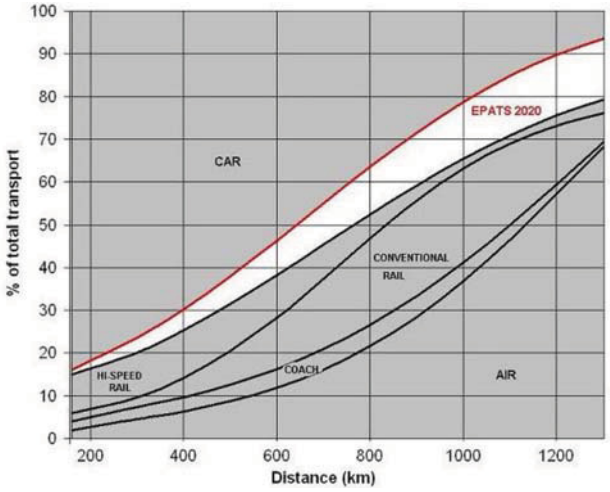

BILD 1. Zukunftsprognose der European Personal Air Transportation System STUDY (EPATS) über Individualverkehr 2020 [3].

Verschiedene internationale Studien und Einrichtungen (ACARE 2020, EPATS, NASA, EU [1][2][3]) sagen einen wachsenden Bedarf von individuellen Punkt zu Punkt Verbindungen im Luftverkehr für die nächsten 10-20 Jahre voraus. Grund hierfür ist zum einen ein erhöhtes Mobilitätsbedürfnis für eine wachsende Zahl von Personen (Privat und Business), zum anderen aber auch der Bedarf nach individuellem Gütertransport.

Flüge unter IFR stellen hohe Ansprüche an Piloten. Dies ist ein entscheidender Grund dafür, dass nur ein kleiner Prozentsatz der Privatpiloten IFR Berechtigung aufweist [4]. Flüge unter VFR können aufgrund ihrer Wetterabhängigkeit nur in sehr eingeschränktem Maße ausgeführt werden. Dies wird gemäß [4] dadurch bestätigt, dass heute der überwiegende Teil von Flügen, pilotiert von Privatpiloten, freizeitsportlicher Natur ist. Somit stellen Flugzeuge der allgemeinen Luftfahrt in ihrer Masse heute kein zuverlässiges und flexibles Verkehrsmittel dar. Dies sind aber zentrale Eigenschaften eines Individualverkehrsmittels wie dem Auto [4]. Aber auch in anderen Punkten wie Sicherheit hinken heute privat pilotierte Flugzeuge der allgemeinen Luftfahrt in der Masse weit hinter den entsprechenden Eigenschaften des Automobils hinterher.

Um den Anforderungen an ein Individualverkehrsmittel wie dem Automobil gerecht zu werden, zielt das Easy Handling System auf hochgradige Automatisierung von Flugsteuerung/-führung bis hin zu autonomen Eigenschaften ab. Fehlerbedingte Funktions-Degradationen werden als absolut sicherheitskritisch betrachtet. Ein derartiger Ansatz bedarf aber einer vollen Fly-by-Wire Steuerung für Flugzeuge der allgemeinen Luftfahrt.

1.2. Projektziel

Schwerpunkt dieses Projektes ist die Entwicklung eines allgemeinen Fly-by-Wire Ansatzes, geeignet für Flugzeuge der allgemeinen Luftfahrt. Dies ist ein notwendiger Schritt in Richtung Easy Handling System. Die Entwicklung von Funktionen in für eine Easy Handling Charakteristik steht zunächst hingegen im Hintergrund. So wird schließlich ein Fly-by-Wire System für eine Diamond DA42 ausgelegt, integriert und flugerprobt.

Spezielle Anforderungen der allgemeinen Luftfahrt, beispielsweise Kosten, zwingen zu neuen Ansätzen. Um dieser Anforderung zu genügen basiert das Fly-by-Wire System auf dem Ansatz einer so genannten flexiblen Plattform. Diese Technologie ist über mehrere Forschungsprojekte hinweg vom ILS entwickelt worden und dient als Basis für die Realisierung des Fly-by-Wire Systems in diesem Vorhaben.

1.3. Projektpartner

SAFAR ist ein internationales Projekt im Rahmen des 7. Rahmenforschungsprogramms der EU. Partner und deren Aufgaben sind:

- Rheinmetall Defence Electronics (D)
  → Koordinator
- TU Delft (NL)
  → Flugregelung
- Deutsche Flugsicherung (D)
- Septentrio Satellite Navigation (B), Honeywell International (CZ), GMV (E)
  → Integrierte Navigation
- Institut für Luftfahrtsysteme, Uni Stuttgart
  → FlyByWire Plattform

2. ZENTRALE ANFORDERUNGEN AN ZUKÜNFTIGE AVIONIK ARCHITEKTUREN FÜR KLEINFLUGZEUGE

Zentrale Anforderungen an ein Fly-by-Wire System für ein Flugzeug der allgemeinen Luftfahrt sind:

- Integration komplexer Funktionen zur Umsetzung der Easy Handling Charakteristik
- Kosteneffizienz
- Ausfallsicherheit.

Das im Projekt betrachtete Flugzeug, die DA42, ist der Klasse CS23/Class II [5] zugeordnet. Damit ergeben sich für ein Primärsystem folgende sicherheitsrelevante Anforderungen:

- $P(Fail_{System} \rightarrow Catastrophic) < 10^{-7} / h$
- DAL C

Darüber hinaus gibt es von verschiedenen Herstellern zusätzliche Anforderungen zur Erhöhung der Robustheit eines Fly-by-Wire Systems gegenüber „Common Cause Failures“.

3. TRENNUNG ZWISCHEN MANAGEMENT UND FUNKTIONALITÄT

Die Technologie „flexible Plattform“ basiert auf einer strikten Trennung zwischen Funktionalität auf der einen und Management (Operating-, Netzwerk-, Fehler-, Redundanzmanagement) auf der anderen Seite. Die Plattform ist dabei ein flexibles Netzwerk aus:

- Hardware-Komponenten (Kernmodule, Aktuatorik, Sensor, etc.)
- einer spezialisierbaren Middleware

Die eigentlichen Systemfunktionen SFu(x) werden in Applikationen realisiert und so auf diese Plattform integriert (siehe BILD 2).

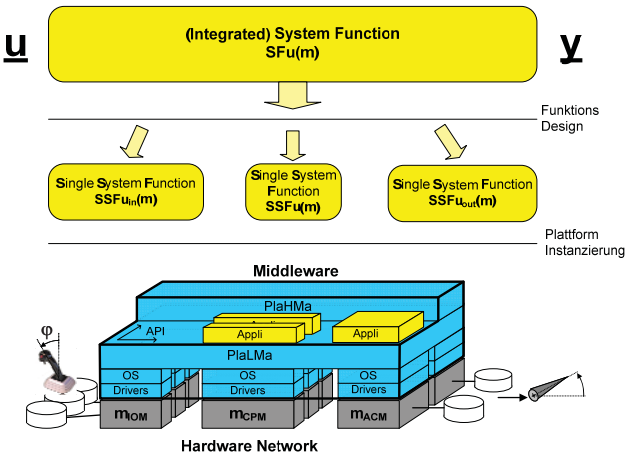

BILD 2. Design-Prozess

Ziel ist es die Komplexität der verteilen FBW-Steuerung (redundanter Datenfluss, Plattform Management, redundantes Signal Handling, etc.) zu abstrahieren und somit gegenüber der Applikation als virtuelles Simplex-System abzubilden. Die Applikationen schrumpfen dabei wieder auf ihre ureigene Aufgabe der Funktionalität.

4. DESIGN SYSTEMFUNKTIONEN (SFU)

Die eigentliche Funktionalität, wie Flugregelung oder Navigation, wird in sog. Systemfunktionen (SFu) abstrahiert. Eine SFu(x) stellt für das System (z.B. Flugzeug) eine Transferfunktion von Eingangsvektor zu Ausgangsvektor dar. Das Funktionsdesign kann dabei von einem Simplex-System ausgehen, da sämtliche Redundanz in der generischen Plattform behandelt wird („Simplex Minded Functional Design“). Falls nötig kann die Systemfunktion in weitere Unterfunktionen (Single System Funktion) unterteilt werden um beispielsweise spezielle Sensor-Anbindungen (simplex) zu implementieren. Mit der Plattform-Instanzierung werden diese SFu(x) in Applikationen realisiert (BILD 2).

Typische sicherheitsrelevante Anforderungen an die Systemfunktion werden im Folgenden beschrieben.

SFu(x) sei eine von mehreren Systemfunktionen, die auf der Plattform ausgeführt wird. Charakterisiert das Tupel $(SFu(x), z_{FA})$ den Zustand der Systemfunktion SFu(x), so seien:

$(SFu(x), z_{FA} = k)$, $k \in \{Deg0, Deg1, ...\}$, $k \notin \{DegN\}$: SFu(x) arbeitet korrekt aber ggf. degradiert (Deg1, …)

$(SFu(x), z_{FA} = z_{fo}\})$: SFu(x) arbeitet fehlerhaft und mindestens eine ihrer sicherheitsrelevanten Ausgangsgrößen hat keinen sicheren Defaultwert angenommen, verhält sich quasi „außer Kontrolle“

Damit ergibt sich folgende *generische Plattform-Anforderung 1*:

(1) $P(SFu(x), z_{FA} \neq k) \leq P_{f\max}$, $P_{f\max} \in \{10^{-5}..10^{-11}\}$

(2) $P(SFu(x), z_{FA} = z_{fo}) \leq P_{fo\max}$,

$P_{fo\max} \in \{10^{-8}..10^{-11}\}$

Aus Kapitel 2 ergibt sich ebenfalls die Anforderung, die Plattform robust gegenüber „Common Cause Failure“ auszulegen. Der klassische Ansatz hierbei wäre der dissimilare Aufbau aller Replikationen. Institutsinterne Untersuchungen zeigen, dass Redundanzstrukturen mit ausschließlich aktiven Replikationen, typisch für klassische Triplex- oder Quadruplexstrukturen, keinen hohen Grad an Dissimilarität erlauben und somit keine glaubhafte Robustheit gegen Common Design Faults durch Dissimilarität erwarten lassen. Anders verhält es sich mit Replikationen, die in einer Art „Active/Standby“-Betriebsweise arbeiten und bis auf wenige Größen keinen Replikationsgleichlauf erfordern.

Somit folgt die *generische Plattform-Anforderung 2*:

Die Plattform muss Potential für einen hohen Grad an Dissimilarität durch Active/Standby-Replica aufweisen.

5. DESIGN PLATTFORM

5.1. Plattform Elemente

Zentrale Plattform-Elemente sind ein flexibles Hardware-Netzwerk und eine spezialisierbare Middleware, die das Netzwerk und dessen Redundanzstrukturen u.a. verwaltet und der Applikation als virtuelles Simplex-System abstrahiert.

5.1.1. Flexibles Hardware Netzwerk

Die *generische Plattform Anforderung 1* zusammen mit der Forderung nach möglichst geringen Kosten führt zu der Forderung nach einer hohen Flexibilität des Hardware-Netzwerkes der Plattform-Instanz selbst. Die generische Plattform legt sich also nicht auf eine bestimmte Hardware

fest, sondern beinhaltet folgende allgemeine Komponenten:

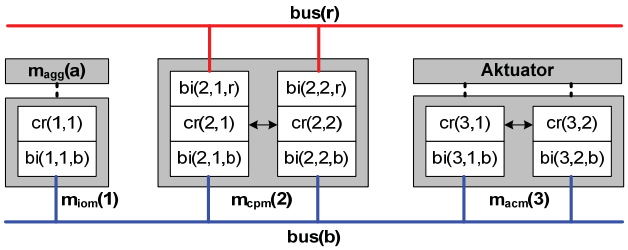

BILD 3. Hardware Komponenten der Plattform

bus(x): Bussystem

- Jede Plattforminstanz besteht generell aus zwei voneinander unabhängigen Bussystemen, die in sich wiederum redundant aufgebaut sein können. Bussysteme können sein: FlexRay, AFDX, etc.

$m_{cpm}(k)$: Computing Module

- Alle $m_{cpm}(k)$ der hier betrachteten Plattforminstanz weisen eine Duplexstruktur auf, bestehen also aus zwei Lanes: la(1), la(2).
- Jede Lane umfasst :
  - einen Processing-Kern cr($m_{cpm}(k)$,la(i)) mit Stromversorgung,
  - *zwei* voneinander unabhängige Businterfaces bi, eines pro Bus.

$m_{iom}(k)$: Input/Output Modul

- Alle $m_{iom}(k)$ der hier betrachteten Plattforminstanz weisen eine Simplexstruktur auf und bestehen demnach aus nur einer Lane la(1). Eine Duplexstruktur ähnlich dem m_{acm} ist ebenfalls möglich.
- Die Lane umfasst
  - einen Processing-Kern cr($m_{iom}(k)$,la(i)) mit Stromversorgung,
  - *ein* Businterface bi,
  - Interfaces für IO-Aufgaben und zur Ankopplung externer Aggregate

$m_{acm}(k)$: Actuator Control Modul

- Alle $m_{acm}(k)$ der hier betrachteten Plattforminstanz weisen eine Duplexstruktur auf. Jede Lane umfasst
  - einen Processing-Kern cr($m_{acm}(k)$,la(i)) mit Stromversorgung,
  - *ein* Businterface bi,
  - Interfaces zur Aktuatoransteuerung und sicheren Passivierung des Aktuators im Fehlerfall

$m_{agg}(k)$: Aggregate

- Aggregate sind Sensoren oder einfache Stellelemente, die nicht an den Plattformbus angekoppelt, sondern via IOM verbunden sind.

Um die Flexibilitäts-Anforderung auch an die Netzwerk-Topologie sicherzustellen, können die o.g. Hardware-Komponenten zu verschiedenen Netzwerk-Topologien kombiniert werden. Als Beispiel soll eine einzelne SFu(x) mit unterschiedlichen Verfügbarkeitsanforderungen auf eine Plattform-Instanz integriert werden (Sensorik und Aktuatorik wird zur Veranschaulichung vernachlässigt):

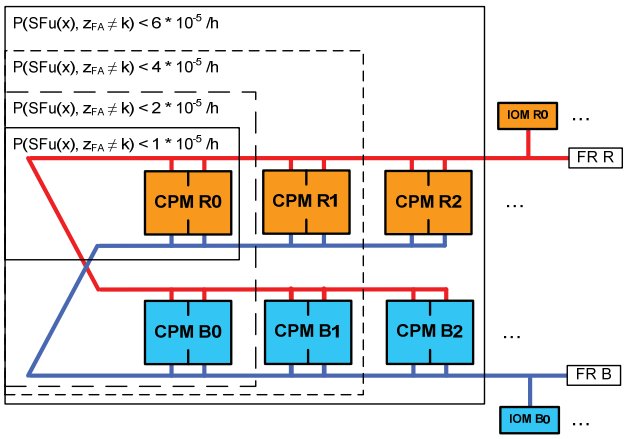

BILD 4. Beispiel für das Flexibles Hardware-Netwerk der generischen Plattform

5.1.2. Generische Middleware

Ziel der Middleware ist es die Komplexität der Plattform-Instanz gegenüber den Systemfunktionen zu verbergen (BILD 5). Die Applikation sieht nur ein „virtuelles Simplexsystem“.

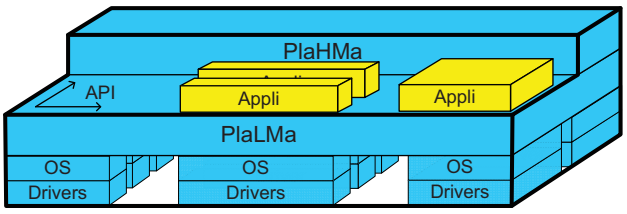

BILD 5. Generische Middleware

Die Middleware im klassischen Sinne bezeichnet in der Informatik anwendungsneutrale Programme, die so zwischen Anwendungen vermitteln, dass die Komplexität dieser Applikationen und ihre Infrastruktur verborgen wird [8]. Das Plattform-Management umfasst jedoch weit mehr Aufgaben. Eine generische Middleware im Sinne einer redundanten Plattform umfasst:

- Kommunikation / „Reliable Broadcast“ innerhalb der Plattform
- fehlertolerante Reduktion redundanter Daten
- Fehlererkennung und Zuordnung von Fehlern auf Granulate[1]
- Consensusbildung auf allen Hierarchieebenen
- Durchführung von Rekonfigurationen

Die Middleware umfasst also auch ein komplexes (Redundanz-) Management. Da diese Middleware im hohen Maße spezialisierbar sein muss, wird diese Komplexität noch erhöht. Es ist also eine klare

[1] Ein Granulat ist die kleinste Einheit von Hardware oder Funktionen, die aus Sicht des Plattformmanagements nicht weiter unterteilt wird

Strukturierung der Middleware mit einfachen Schnittstellen zwischen den einzelnen Elementen notwendig. Die Middleware wird zunächst unterteilt (siehe auch BILD 5):

- Plattform-Highlevel-Management (PlaHMa)
  - Verwaltung der plattforminternen Kommunikation
  - Realokierung von Funktionen
  - Consensbildung auf Plattform-Ebene
- Plattform-Lowlevel-Management (PlaLMa)
  - Durchführung der plattforminternen Kommunikation
  - Consensbildung auf Modul-Ebene

PlaHMa soll in diesem Artikel nicht weiter vertieft werden. PlaLMa wird im Folgenden bezüglich der Applikations-Kommunikation erläutert.

Die Herausforderung für eine flexible Plattform liegt auch in der Heterogenität der verwendeten Kommunikation. So ist für die allgemeine Luftfahrt neben zahlreichen Analog-Schnittstellen ein CAN-Bus ebenso denkbar wie FlexRay oder AFDX. Für eine generische Middleware muss also eine einheitliche Darstellung und Behandlung dieser vielfältigen Schnittstellen gewährleistet werden. Dies umfasst die Mechanismen zur Erkennung von fehlerhaften Signalen/Daten auf Simplex- und Redundanzpfaden, Zuordnung von Fehlern auf Granulatebene sowie Reduktion von redundanten Signalen auf virtuelle Simplexsignale mit entsprechenden Statusinformationen für die Applikation. Dazu wurde ein Schichtenmodell (ähnlich dem OSI-Modell [9]) u.a. auf Signalebene eingeführt.

Dieses Modell umfasst innerhalb eines Moduls zunächst folgende Schichten:

Index	*Schicht*
L7	Application Layer
L6	Presentation Layer
L5	Redundancy Layer
L4	Granulation Layer
L3	Unification Layer
L2	Data Link Layer
L1	Physical Layer

TAB 1. Schichten der Plattformmanagement-Signalebene

- **L1** und **L2** entsprechen im Wesentlichen dem OSI-Modell [9].
- **L3**: Zentrale Aufgabe dieser Schicht ist die Standardisierung der Darstellung aller heterogenen Signaldaten, Statusdaten sowie Statusinformationen zu diesen Daten, i.e. im Wesentlichen Fehlerklassifizierungen.
- **L4**: Diese Schicht ordnet die einzelnen Datenpakete der Schicht L3 und deren Statusinformation Quellgranulaten zu. Ein Quellgranulat kann sein: ein m_{agg}, ein Teil eines m_{agg}, oder ein Cluster von m_{agg}.
- **L5**: Reduziert redundante Daten auf virtuelle Simplexdaten mit effizienten Fehlermonitoring. Hier kommen typische Mehrheits-Voting/Monitoring Verfahren zum Einsatz. Es erfolgt eine Ergänzung der Statusinformation.
- **L6**: Hier erfolgt die Darstellungsanpassung der Daten für die Applikation.
- **L7**: Stellt analog zum OSI-Modell die API für die Applikation zur Verfügung.

Für Signale von der Applikation zum Empfänger wird dieses Modell entsprechend rückwärts durchlaufen.

Weitet man dieses Schichtenmodell auf die gesamte Plattform aus, so kann es zwischen Schicht 4 und 5 oder zwischen Schicht 5 und 6 zum Zweck der Plattform-internen Datenübertragung unterbrochen werden. Es ergibt sich folgende Schichten-basierte Kommunikation:

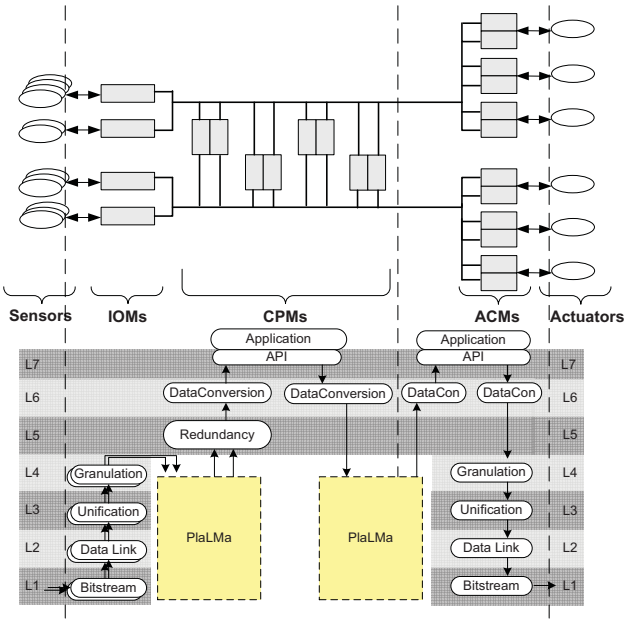

BILD 6. Schichtenbasierte Signalkommunikation auf der Plattform

5.2. Ausgewählte Plattform-Mechanismen

Im Folgenden wird auf einige Plattformmechanismen detailliert eingegangen um die Funktionsweise der generischen Plattform zu veranschaulichen. Dazu wird von folgendem Plattform-Beispiel ausgegangen:

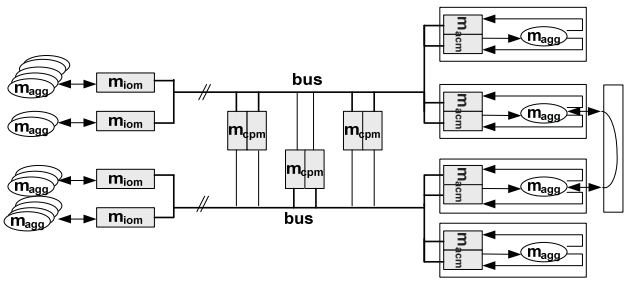

BILD 7. Beispiel einer Plattform-Instanz

5.2.1. Consensus im Plattformkern

Die Plattform weist generell eine hierarchische Redundanzstruktur auf: Aggregate sind redundant, Module sind teilweise sowohl intern redundant – die Plattform lässt auch Triplex- und Quadruplexmodule zu – als auch zueinander redundant. Auf allen Redundanzhierarchien bedarf es eines Consensus zwischen allen Replikationen. Betrachtet wird im Weiteren allerdings nur der Consensus auf höchster Plattformebene, nämlich der Consensus zwischen den m_{cpm}.

5.2.2. Datenübertragung mittels Frames fd [2]

Der Austausch von Daten erfolgt über so genannte Duplex-Frames fd, bestehend aus zwei redundanten Frames f.

- fd(m(s),bus(x)) ist ein Duplexframe mit folgender Charakteristik:
  - Frame f(m(s),la(**1**),bus(x)) wird von Modul m(s), Lane **1** generiert und via bus(x) übertragen.
  - Frame f(m(s),la(**2**),bus(x)) wird von Modul m(s), Lane **2** generiert und über den gleichen bus(x) übertragen.
  - Die Payload pf von frame f enthält einen Zeitstempel.
  - pf(m(s),la(1),bus(x)) = pf(m(s),la(2),bus(x)).

- Ein empfangener fd gilt als korrekt, wenn gilt:
  - beide f sind korrekt im Zeitbereich,
  - beide f sind korrekt im „Code"-Bereich,
  - beide f haben identische Payload

- Jedes m(s) überträgt seine fd über beide Bussysteme: bus(b), bus(r).
- Ein empfangendes m(e) akzeptiert nur korrekte fd.
- Empfängt ein m(e) auf einem der beiden Busse ein korrektes fd, so gilt die Datenübertragung als korrekt.

5.2.3. „Failure"- Hypothese

Entscheidend für die Sicherstellung des Consensus zwischen den intakten m im Plattformkern ist die Fehlerhypothese für den Plattformkern. Diese Hypothese wird getragen von folgenden Eigenschaften:

- Eine fehlerhafte Lane la(m(k),la(i)) kann das Verhalten seiner intakten Nachbar-Lane la(m(k),la(¬i)) nicht manipulieren.
- Ein fehlerhaftes bi(m(k),la(i),bus(x))[3] kann nur bus(x) manipulieren. Die Manipulation darf *beliebiger* Art sein.
- Ein fehlerhaftes bi(m(k),la(i),bus(x)) kann beide redundanten Frames f eines beliebigen fd nie so manipulieren, dass dieser fd gemäß o.a. Definition weiterhin als korrekt gilt, obwohl einzelne Werte seiner „Payload" verfälscht sind.
- Ein fehlerhafter Kern cr(m(k),la(i)) darf sich *beliebig* fehlerhaft verhalten, kann allerdings bi(m(k),la(i),bus(x)) und bi(m(k),la(i),bus(¬x)) seiner lane nur so weit manipulieren, dass sie
  - Frames f mit inkorrekter Payload pf übertragen oder
  - keinen Frame übertragen.

5.2.4. Consensus zwischen allen korrekten m (=m_{cpm})

Plattformstruktur, Busübertragung und die gewählte Übertragungsform stellen für den Fall eines einzigen fehlerhaften Granulates[4] sicher:

- Ein fehlerhaftes Modul (Modul mit einem fehlerhaften Granulat) kann die zyklisch ablaufende Kommunikation zwischen den anderen intakten Modulen in keiner Weise stören.
- Ist im fehlerhaften Modul der Kern cr(m(s),la(i)) fehlerhaft, so wird dieser „Failure" von allen intakten Modulen in konsistenter Weise wahrgenommen. Alle intakten Module ermitteln so für die Payload des fehlerhaften Modul m(s) denselben Wert, wobei dieser auch „nil" sein kann.
- Ist im fehlerhaften Modul das Businterface bi(m(s),la(i),bus(x)) fehlerhaft, so ermitteln alle intakten Module für die von m(s) gesendete Payload
  - denselben Wert und
  - den Wert des Senders, also m(s).

Mit dieser Definition des Consensus ist sichergestellt, dass unter der o.a. Fehlerhypothese Entscheidungen von den m_{cpm} des Plattformkerns in zueinander konsistenter Weise erfolgen.

6. DIE PLATTFORM-INSTANZ FÜR SAFAR

Ausgehend von der oben beschriebenen Plattform und den in Kapitel 2 formulierten Anforderungen wird eine Plattform-Instanz für SAFAR abgeleitet.

6.1. Abbilden der Funktionen auf SFu

Funktionell wird von Projektpartnern zunächst eine herkömmliche Flugsteuerung auf diese Plattform integriert um die grundsätzliche Anwendbarkeit derselben zu demonstrieren. Konkret wird ein AttitudeMode[5] und ein DirectMode[6] realisiert. Der DirectMode stellt dabei eine funktionale Degradation des AttitudeMode dar.
Dafür werden zwei Grundfunktionalitäten benötigt. Zum einen ist dies die Flugregelung, welche Piloten-Kommandos via Side-Stick zusammen mit Lage- bzw. Luftdateninformationen in Stellkommandos für Aktuatoren an den Hauptsteuerflächen und die Motoren umsetzt. Des Weiteren wird eine integrierte Navigation realisiert, die Lage und Positionsdaten aus unterschiedlichen (redundant vorhandenen) Sensoren (GNSS, IMU, ADS) via Kalmanfilter fusioniert und eine erweiterte Fehlererkennung beinhaltet.

[2] In diesen Abschnitt steht m immer für m_{cpm}.
[3] Bus-interface in Module m(k),, Lane la(1), an bus(x).
[4] Modulgranulate: cr(m_{CPM}(k),la(i)), bi(x)
[5] Im Attidue-Mode gibt der Pilot via 2-Achsen Stick die Lage des Flugzeugs vor. Das Seitenruder wird automatisch angesteuert. Dafür werden v.a. Lagedaten des Flugzeugs verwendet.
[6] Im Direct-Mode steuert der Pilot via 2-Achsen Stick direkt die Aktuatoren für Höhen- und Querruder an.

Da die Zielsetzung einer „Easy Handling Charakteristik“ das fliegen im DirectMode ausschließt, wird der Verlust des AttitudeMode (hier ersatzweise für EHC) als „Catastrophic“ (nach CS23) eingestuft.

Die beschriebenen Funktionen (Flugregelung, integrierte Navigation) werden auf die SFu(FCL) sowie die SFu(INAV) abgeleitet.

Aus Hardware-Performance Gründen wird jede SFu in einer eigenen map[7] ausgeführt:

- SFu(FCL) → map(fcl)
- SFu(INAV) → map(inav)

Da bereits der Verlust von SFu(INAV) (→ Verlust des AttitudeModes) als CATASTROPHIC zu bewerten ist, folgt mittels beispielhafter Sicherheitsbudgetierung:

(3) $$P\left(\left(SFu(FCL), z_{FA} \neq k\right) \cup \left(SFu(INAV), z_{FA} \neq k\right)\right) < 0.5*10^{-7}/h$$

(4) $$P\left(\left(SFu(FCL), z_{FA} = z_{fo}\right) \cup \left(SFu(INAV), z_{FA} = z_{fo}\right)\right) < 0.5*10^{-7}/h$$

6.2. Plattform Topologie

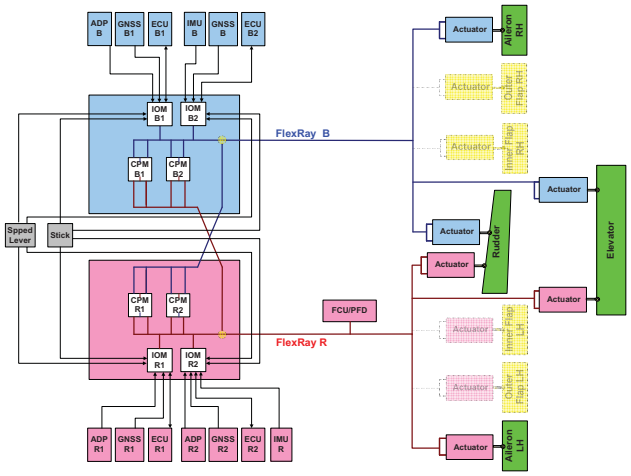

BILD 8. Plattform-Instanz SAFAR (Topologie)

Die o.g. Anforderungen werden mit einem 4CPM / 4IOM Plattformkern erreicht. Zusätzlich ist eine Sensorik mit der Redundanz jeweils mindestens Triplex[8][9] notwendig.

Die Aktuatorik bietet im Verbund mit dem Plattformkern bereits eine hohe Integrität (lediglich ein Blockieren nach der Kupplung kann nicht passiviert werden). Deswegen wird wegen der dominanteren Verfügbarkeitsforderung ein Doppelaktuatoren-System pro Steuerfläche realisiert.

Die daraus folgende Plattform-Topologie ist in BILD 8 dargestellt.

7. UMSETZUNG -- IRON BIRD & FLUGVERSUCHE

7.1. Einschränkungen am Demonstrator

Für das „Permit for Flight“ wurde von den Behörden für den Notfall eine sichere Entkopplung der FBW-Steuerung von den Original-Systemen des Versuchsflugzeugs verlangt. Außerdem sollten Original-Systeme (z.B. Stromversorgung) nicht verändert werden.

Dementsprechend wurden folgende Maßnahmen umgesetzt:

- Das Flugzeug wird von zwei Piloten geflogen; einer für die mechanische Backup-Steuerung, einer für die FBW-Steuerung
- Ein „Not-Aus“ öffnet die Stromversorgung der Aktuatorik, deren elektromagnetischen Kupplungen daraufhin öffnen und die FBW-Steuerung vom Orginial-System trennen. Dieser „Not-Aus“ kann von beiden Piloten betätigt werden.
- Um die Original-Stromversorgung im Flugzeug nicht zu verändern, wurde lediglich eine Simplex-Versorgung der FBW-Steuerung realisiert.

7.2. Entwickelte Hardware

Im Laufe des Projekts wurden Kernmodule neu- bzw. weiterentwickelt.

7.2.1. Smart-Aktuator

BILD 9. Aktuator mit Aktuator-Control-Module (Smart Aktuator)

Zur Ansteuerung der Steuerflächen wurde ein Smart-Aktuator entwickelt. „Smart“ bezeichnet hierbei die Eigenschaft die meiste Intelligenz zur Ansteuerung und (Redundanz-) Verwaltung im Modul selbst zu halten. Nach außen hin steht ein relativ einfaches Interface (via FlexRay) zur Verfügung.

[7] Mega-Applikation: Ein bis n SFu(x) können in einer map zusammengefasst werden. Eine map ist im fehlerfreien Fall immer auf mehreren cpm aktiv (Active/Standby). Durch eine aktive Realokierung dieser maps wird im Fall eines fehlerhaften cpm die kritische map längstmöglich ausgeführt.

[8] Die Integrität eines Simplex-Sensors kann mit ca. 90% (C=0.9) angegeben werden. Eine Degradation von Duplex auf Simplex ist wegen (3) also nicht zulässig. Um trotzdem die erforderliche Verfügbarkeit zu erreichen ist mindestens Triplex notwendig.

[9] Da die SFu(INAV) aus unterschiedlichen Sensor-Typen (IMU, GNSS, ADS) ähnliche Daten zur Fusion verwendet, kann von der Triplex-Forderung abgewichen werden.

Der Smart-Aktuator selbst wird dabei in den eigentlichen Aktuator und die Kontrolleinheit unterteilt.
Der Aktuator ist ein Gleichstrom-Synchronmotor, dem ein Getriebe mit einem Verhältnis von 120:1 nachgelagert ist. Um den Aktuator im Fehlerfall sicher von der Steuerfläche trennen zu können, ist eine elektromagnetische Kupplung, die im stromlosen Zustand öffnet, implementiert.

Die Kontrolleinheit ist intern duplex aufgebaut. Sie kommandiert eine ebenfalls integrierte Leistungselektronik, steuert die Kupplung auf redundanten Pfaden an und sichert eine korrekte Kommunikation mit den Plattformkern (m_{cpm}).

7.2.2. Kern-Module

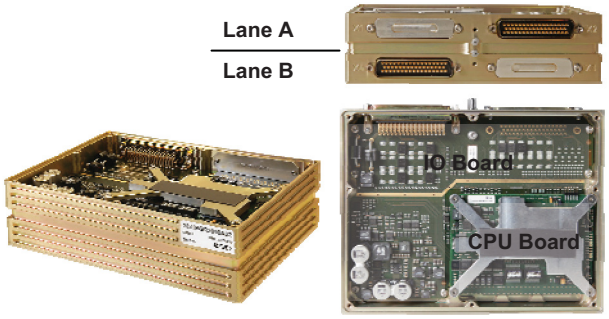

BILD 10. Kernmodule m_{cpm} und m_{iom}

Im Fokus der Entwicklung für die Kernmodule m_{cpm} und m_{iom} lag ein hoher Grad an wieder verwendbaren Hardware Komponenten. So verwenden alle Module/Lanes eine gleich aufgebaute Prozessor-Platine (cr, CPU-Board), die neben einem Mikrocontroller von Freescale auch zusätzliche Schnittstellen und Speicher enthält. Die Kernmodule unterscheiden sich lediglich bezüglich der IO-Platine, die auf die verschiedenen Schnittstellenanforderungen für den jeweiligen Anwendungsfall angepasst sind. Die Module sind aus zwei gleichen, jedoch voneinander separierten Bereichen aufgebaut. Im Falle eines Duplex-Moduls (z.B. m_{cpm}) kommunizieren beide über eine interne Ethernet-Verbindung. Im Fall eines simplex Designs (m_{iom}) enthält die Box dann zwei weitestgehend separierte Module.

Dieses Hardware-Design stellt auch einen ersten Schritt in Richtung Zertifizierbarkeit dar, da bereits ein Großteil der Anforderungen aus DO160 erfüllt wird.

7.3. Closed Loop Simulation und IronBird

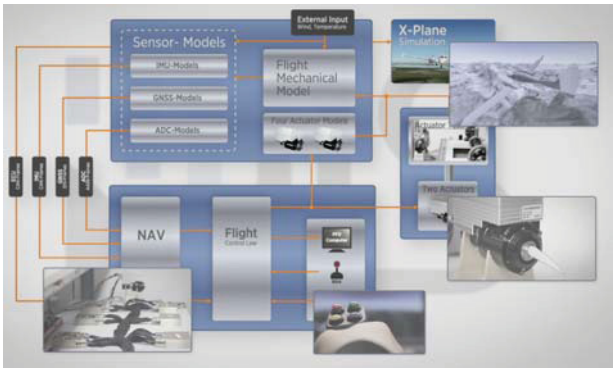

BILD 11. Closed Loop Simulation

Um das richtige Zusammenspiel aller Komponenten (HW+SW) bereits vor den Flügen sicherstellen zu können, wurde ein IronBird am ILS realisiert. Dabei laufen alle Kern-Module, ein Teil der Aktuatorik und das HMI (Stick+Display) als hardware-in-the-loop, während Flugzeug, Sensorik und die restliche Aktuatorik im Testsystem simuliert werden.

7.4. Flugversuche

BILD 12. SAFAR Versuchsflugzeug Diamond DA42

Die Umsetzbarkeit einer solch komplexen FBW-Steuerung mit den o.g. Konzepten wird gegen Projektende (Q4 2011) in geplanten 50 Flugstunden demonstriert. Die FBW-Steuerung wird dabei auf eine Diamond DA42 (siehe Bild) „on top“ aufgesetzt, d.h. ein mechanisches Backup bleibt bestehen.

Nach der Zertifizierung der hier entwickelten FBW-Steuerung kann auf das mechanische Backup verzichtet werden. Dies wird mit SAFAR zunächst jedoch nicht umgesetzt.

Ende 2010 wurden bereits erste Sensor-Messflüge zur Parametrisierung der integrierten Navigation durchgeführt.

8. AUSBLICK

Im Bereich der allgemeinen Luftfahrt wird in zukünftigen Projekten der Easy Handling Aspekt weiter voran getrieben. Ziel sind dabei auch optional pilotierte Flugzeuge. Ebenfalls wird die Plattform auf Anwendungsfälle im Bereich der CS25 hin untersucht und erweitert. Für den Bereich der nicht absolut sicherheitskritischen Systeme (z.B. Kabinensysteme) wird die Möglichkeit einer selbstständigen Spezialisierung (Selbst-Adaption) der Plattform untersucht.

9. LITERATUR

[1] Future Flight, A Review of the Small Aircraft Transportation System Concept, Special Report 263, Transportation Research Board, 2002
[2] ACARE 2008 Addendum to the Strategic Research Agenda, ACARE, 2008
[3] EPATS, European Personal Air Transportation System STUDY, www.epats.eu (online: 05.08.2011)
[4] G. Konrad, Flugzeuge der Allgemeinen Luftfahrt als zukünftiges Individualverkehrsmittel, Dissertationsschrift, noch nicht veröffentlicht
[5] JAR CS23, Advisory Circular 1309, EQUIPMENT, SYSTEMS, AND INSTALLATIONS IN PART 23 AIRPLANES
[6] N.N.: DO178-B: Software Considerations in Airborne Systems and Equipment Certification. Washington D.C.: RTCA, 1993.
[7] N.N.: DO254: Design Assurance Guidance For

Airborne Electronic Hardware. Washington D.C.: RTCA, 2000.
[8] W. Ruh u. a.: Enterprise Application Integration. Wiley, 2001
[9] Zimmermann, H: *OSI Reference Model−The ISO Model of Architecture for Open Systems Interconnection*. IEEE Transactions on Communications. (1980) 28-4, ISSN 0090-6778, S. 425-432.

10. ABKÜRZUNGEN

VFR	Visual Flight Rules (Sichtflug)
SAFAR	Small Aircraft Future Avionics Architecture
ILS	Institut für Luftfahrtsysteme
GNSS	Global Navigation Satellite System
IMU	Inertial Measurement Unit
ADS	Air Data System
HMI	Human-Machine-Interface
EHC	Easy Handling Charakteristik
FBW	Fly-By-Wire