SYSTEM DESIGN DES BARRACUDA FLIGHT CONTROL SYSTEMS

G. Jarasch, M. Friedrich, K. Harth, M. Momberg, S. Schärer, A. Schönhoff, D. Wetteborn
EADS Military Air Systems
Deutschland

1. EINFÜHRUNG

Der Erstflug des unbemannten Demonstrators Barracuda am 2. April 2006 stellt einen großen Meilenstein in der Entwicklung unbemannter Fluggeräte im europäischen Raum dar.

Der Barracuda wurde von EADS Military Air Systems (MAS) entwickelt, um Schlüsseltechnologien für das unbemannte militärische Fliegen zu erproben. Die Zulassung erfolgte nach Kategorie 2 der LTF 1550 [3], jedoch mit Einschränkungen wie für Kategorie 1. Der Barracuda darf somit vorerst nur in gesperrten Lufträumen über unbewohntem Gebiet geflogen werden.

Im folgenden Beitrag wird die Entwicklung des Flight Control Systems für den Barracuda beschrieben. Dabei werden neue Vorgehensweisen und Techniken vorgestellt sowie Besonderheiten, die sich im Kontext eines Experimentalprogramms ergeben.

Systemdesign und Softwareentwicklung wurde in diesem Projekt als eine eng vernetzte Aufgabe verstanden. Die gegenseitige Einflussnahme trug wesentlich zum Erfolg des Projekts bei.

2. DAS BARRACUDA FCS

Das Flight Control System (FCS) des Barracuda stellt jene Funktionen zur Verfügung, die zur sicheren Durchführung des Fluges notwendig sind. Das FCS besteht aus drei Subsystem-Gruppen:

– der Sensorik zur Erfassung des Flugzustands,
– dem Flight Control Computer (FCC) [5], auf dem unter anderem die Flugführungsalgorithmen implementiert sind und
– der Aktuatorik für die Steuerflächen, das Bugrad, die Bremse und das Triebwerk.

Einen Überblick über die Einbauorte der FCS-Subsystem-Gruppen bietet BILD 1.

Als Schnittstelle zur Missionsführung, zu der auch das Starten einer Mission gehört, dient der Advanced Mission Computer (AMC) [11]. Er stellt die Verbindung zur Bodenstation über eine Funkverbindung her [4].

Bei der Entwicklung des Demonstrators Barracuda kamen dem FCS neben seiner originären Aufgabe, die Flugführung und -regelung zu gewährleisten, noch weitere Aufgaben zu, die zum Teil durch den Demonstratorcharakter wie auch durch das Nichtvorhandensein eines Piloten notwendig wurden.

Durch das Fehlen eines Piloten musste schon von Beginn an ein Autonomiekonzept mit integriertem Health Monitoring (HM) konzipiert werden, da eine lückenlose und zuverlässige Überwachung der Flugzeugsysteme im Flug anderweitig nicht gewährleistet werden kann. Der Demonstrator Barracuda wird momentan nur unter Kategorie-1-Bedingungen nach der LTF 1550 [3] betrieben. Aus Zulassungssicht wird der Flugzeugverlust nicht als sicherheitskritisch eingestuft, aus EADS-Firmensicht jedoch schon, weswegen firmenintern höhere Ansprüche an die Sicherheit gestellt wurden als von Zulassungsseite gefordert.

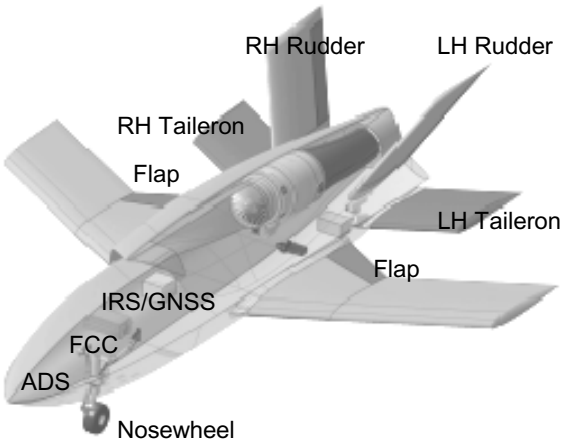

BILD 1: Überblick FCS Einbauorte

Die Gesamtarchitektur ist zum Teil durch neu entwickelte Geräte, wie auch durch günstig verfügbare Commercial-off-the-Shelf-Produkte (COTS) geprägt. Es finden sich in der aktuellen Architektur Konzepte aus dem zivilen Umfeld wie auch aus der militärischen Entwicklung. Dies trifft insbesondere für die Redundanzstrategien und die verwendeten Bussysteme zu.

Dem Entwicklungsprozess folgend werden zuerst die Aspekte des funktionalen Designs beschrieben, die dann in der Systemdesign-Phase bis auf Geräteebene detailliert werden.

2.1. Funktionales Design

Im Entwicklungsschritt des funktionalen Designs werden die rein funktionalen Aspekte noch ohne Berücksichtigung von physikalischer Allokation oder von Implementierungs-Randbedingungen festgelegt.

Hier werden im Besonderen die Aspekte des Redundanzkonzepts, die Health-Monitoring-Funktionalität und das Autonomiekonzept vorgestellt.

2.1.1. Funktionaler Überblick

Für den Betrieb des Barracuda lassen sich im wesentlichen zwei Phasen mit unterschiedlichen Funktionsketten unterscheiden:

– das Herstellen der Startbereitschaft und
– der eigentliche Flug.

Gerade der erste Aspekt zeigt auf, dass bei einem unbemannten Flugzeug andere Strategien gewählt werden müssen als bei konventionellen Flugzeugen. Bei einem unbemannten Flugzeug sind die Interaktionsmöglichkeiten zwischen Personal und den Bordsystemen limitiert, da eine Kommunikation ausschließlich über die Bodenstation, über spezielles Bodengerät oder über Schnittstellen am Flugzeug erfolgt. Selbst die Bodenstation stellt kein Cockpitersatz dar, weil über den Datenlink nicht die Menge von Informationen transportiert werden kann wie sie in einem Cockpit im Flugzeug zusammenlaufen.

Beim Barracuda übernimmt das FCS die Koordination der Systeminitialisierung für

– das Navigationssystem,
– das Luftdatensystem,
– das Elektriksystem,
– die Aktuatorik und
– das Triebwerk.

Lediglich das Anschalten der einzelnen Komponenten erfolgt über das Startup Panel (siehe BILD 2). Das Anlassen des Triebwerkes wird ebenfalls über das Startup Panel durchgeführt und vom ersten Wart über einen extern angeschlossenen Laptop, der die Triebwerksdaten anzeigt, überwacht.

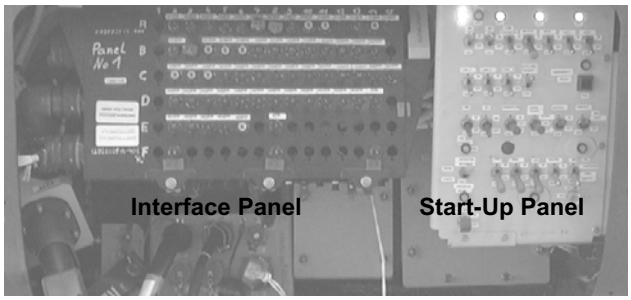

BILD 2: Startup Panel

Erst wenn alle Systeme erfolgreich hochgefahren sind, das Triebwerk angelassen wurde und der Actuator Movement Check durchgeführt wurde, tritt eine zweite funktionale Kette in den Vordergrund (siehe BILD 8).

Mit Hilfe des Navigations- und Luftdatensystems wird der Flugzustandsvektor bestimmt. Durch Kenntnis der vorprogrammierten Wegpunkte werden vom FCC für die Flugregelung Sollkommandos generiert. Der Aufbau der Flugregelung wird in [8] detailliert dargelegt.

Neben diesen Flugführungsmodulen gibt es noch weitere Funktionen, die für ein FCS notwendig sind, wie z. B.

– das Management des Echtzeitverhaltens,
– die Überwachung der Grundsysteme,
– das Redundanzmanagement und
– das Health Monitoring.

Die letzten beiden Funktionen werden im Folgenden beschrieben.

2.1.2. Redundanzkonzept

Bei unbemannten, autonomen Flugzeugen wie dem Barracuda kommt der Wahl des Redundanzkonzepts eine besondere Bedeutung zu. Im Fehlerfall muss das System selbst

– den Fehler erkennen,
– analysieren, wodurch der Fehler verursacht wurde bzw. welches Subsystem fehlerhaft ist, und
– nur fehlerfreie und gültige Signale weiter verarbeiten.

Im Rahmen der Systemauslegung muss spezifiziert werden, wie das System im Fehlerfall zu reagieren hat. Unterschieden werden muss dabei die Reaktion des Systems auf einen Erst- und Zweitfehler, abhängig vom Redundanzgrad.

Weiterhin wird zwischen dem für den Flug kritischen und absolut relevanten System (Systemkern) und den für einen sicheren Weiterflug nicht essentiellen Subsystemen differenziert.

Der Systemkern des Barracuda-FCS ist fail-operable (kurz fail-op) ausgelegt. Im Fall eines Erstfehlers bleibt die Funktionalität erhalten. Gewährleistet wird das durch Abschalten des fehlerhaften Moduls. Nach der Neukonfiguration aufgrund eines Erstfehlers ist der Systemkern fail-indicate (kurz fail-ind). Die nicht essentiellen Systeme sind fail-safe ausgelegt. Im Fehlerfall wird auf ihre Funktion verzichtet.

Zum Systemkern des Barracuda gehören: der FCC, die Navigationsplattformen, die Laserhöhenmesser, die Aktuatorik (bis auf die Bugradsteuerung und die Triebwerksansteuerung) und das Luftdatensystem. Nicht essentielle Systeme sind die Bugradlenkung, der Datenlink zum AMC und die Triebwerksansteuerung.

Die für einen sicheren Weiterflug essentiellen Systeme, wie der FCC und das Navigationssystem, sind triplex ausgelegt. Bezüglich des Redundanzkonzepts mussten im Entwicklungsprozess aus Kosten- und Verfügbarkeitsgründen Abstriche gemacht werden. Aktuatorik-System wie auch das Luftdaten-System sind nur duplex. Zur Ausgleichung des Redundanzsprunges sind die Schnittstellen der Aktuatorik auf alle drei FCC-Kanäle verteilt. Für das Luftdaten-System wird auf dem FCC einen Backup-Lösung berechnet.

Die Schnittstelle zur Bugradlenkung beispielsweise ist nur simplex vorhanden. Zur sicheren Erkennung einer Fehlfunktion wurde hier ein Kontroll-Monitor implementiert, mit dem die Bugradlenkung im Fehlerfall in den frei beweglichen Modus (engl. free castor mode) geschaltet wird.

2.1.3. Health Monitoring

Die FCS-Health-Monitoring-Funktion überwacht alle Funktionen innerhalb des FCS. Es lassen sich drei Kategorien von überwachten Informationen unterscheiden:

– Geräteinformationen,
– Informationen aus dem Redundanzmanagement und
– Informationen aus der Fehlerselbsterkennung der FCC-Software.

Zum einen gibt es Zustandsinformation, die von Subsystemen selbst erzeugt werden. Im Allgemeinen trifft das für Systeme zu, die vollständig in einem eigenständigen Gerät integriert sind. Als Beispiel sei hier die integrierte Navigationssensorik erwähnt, die einen hardwarespezifischen Fehler wie Übertemperatur aber auch funktionelle Fehler wie ein Divergieren der hybriden Navigationslösung, auf Grund von sich widersprechenden Sensorsignalen, selbständig detektiert und zur Anzeige bringt.

Auf diese Weise werden jedoch gerade systematische Fehler nicht entdeckt. Aus diesem Grund wurde, wie bereits beschreiben, ein Redundanzmanagement-Konzept implementiert. Durch die gewählten Voting/Monitoring-Strategien ist es möglich, Fehler zu erkennen, die auf Geräteebene nicht erkannt bzw. überwacht werden können.

Die verwendete Softwareentwicklungstrategie des Defensive Programming basiert zudem nicht auf Fehlerkorrektur sondern auf Fehlervermeidung. So werden beispielsweise bedingte Kontrollflußänderungen stehts von der Erfüllung einer Bedingung eingeleitet (engl. guarded command), während die letzte Alternative in einer Kontrollfluß-Verzweigung (der else- oder default-Fall) stets als Fehlerfall angesehen wird, da keine der vorangegangenen Bedingungen erfüllt war. Hierdurch ist beispielsweise die Reaktion auf Speicherfehler durch Höhenstrahlung (sog. Single Bit Upsets) möglich. Solche Fehlerinformationen werden ebenfalls vom Health Monitoring überwacht.

Die zentrale Aufgabe des Health-Monitoring-Moduls ist neben der Sammlung der Informationen die Zusammenfassung dieser Daten zu einem kombinierten Fehlerzustand pro Subsystem.

Zur Interpretation der jeweiligen Fehlerinformationen bedarf es auch der Berücksichtigung des Gesamtsystemzustandes, zum Beispiel, ob einzelne Systeme gerade initialisiert werden oder ob sich das System bereits im Flugzustand befindet. Da es sich hierbei um eine sehr große Menge von Signalen und Abhängigkeiten von Zuständen handelt, ist dies ein gutes Beispiel dafür, dass sich eine Modellierung dieser Funktionalität in datenbankgestützten Spezifikationsmodell deutlich leichter realisieren und vor allem verifizieren lässt.

Die Beurteilung von Signalqualitäten wurde in diesem Projekt der jeweiligen Funktion überlassen, die das Signal generiert, und nicht im Health Monitor mitimplementiert. Das Auslegungsprinzip ist, dass die Signalqualität als ein weiteres Attribut des Signales angesehen wird, das vorzugsweise von der generierenden Quelle selbst beurteilt werden kann.

Die Auswertung der durchgeführten Taxi Tests hat gezeigt, dass selbst eine geschulte Testmannschaft nicht in der Lage ist, alle Systeminformationen lückenlos zu überwachen. Erschwerend kommt hinzu, dass die Abtastrate der Flugversuchsinstrumentierung (engl. flight test instrumentation, FTI) keine lückenlose Überwachung zulässt, da sehr kurzzeitige Ereignisse nicht abgetastet werden. Hier hat sich das Zusammenspiel des Health Monitors mit dem Autonomiekonzept bewährt, welches im Folgenden beschrieben wird.

2.1.4. Autonomiekonzept

Die Einführung von autonomen Funktionen ist dort zwingend notwendig wo eine ganzheitliche Situationsrepräsentation am Ort der Entscheidungfindung (z.B. am Boden) nicht möglich ist. Bei unbemannten Fluggeräten kann nicht immer gewährleistet werden, dass der Gesamtzustand am Boden sicher dargestellt wird [12]. Dies ist der Treiber dafür, auch in dieser frühen Ausbaustufe schon einige Entscheidungen autonom an Bord zu treffen.

Bei Barracuda beschränken sich in der jetzigen Ausbaustufe die autonomen Entscheidungen auf jene Aspekte, die einen sicheren Flug gefährden.

Die aus dieser Sicht kritischste Phase ist die Beschleunigung des Flugzeugs bis kurz vor der Rotation. Werden hier Systemfehler im flugkritischen Teil, dem FCS, festgestellt, so führt das zu einem autonomen Startabbruch. Die einzige Ausnahme ist eine Fehlfunktion der Bremse.

Da weder die FTI noch der reguläre Führungsdatenlink sicherheitskritisch ausgelegt wurden, ist weder sichergestellt, dass während der Beschleunigung alle Fehler sicher erkannt werden können, noch, dass entsprechende Reaktionen vom Boden aus kommandiert werden können. Allerdings hat der Operateur in der Bodenstation dennoch die Möglichkeit, auch vom Boden einen Startabbruch zu kommandieren, jedoch nur bis zu einer definierten Rollgeschwindigkeit.

Während der Erstflugerprobung wird im Fehlerfall von einem automatischen Abbruch der Mission und einem sofortigen Rückflug abgesehen, da die Erstflugtrajektorie und das "Return To Base"- Manöver sich vom Zeitgewinn kaum unterscheiden und somit die Zeitdauer erhöhten Risikos (engl. time at risk) mit einem degradierten System nicht nennenswert reduziert werden kann.

2.2. Systemdesign

Ist die Funktionalität vollständig beschrieben und spezifiziert, erfolgt die Definition der Subsysteme und die Allokation der Funktionen auf die Subsysteme.

Nach dem Überblick über die gewählte Systemarchitektur wird die Anbindung der Sensorik und Aktuatorik an den FCC vorgestellt, der das Integrationszentrum dieses Systems darstellt.

2.2.1. Systemarchitektur

Die interne Kommunikation des FCCs mit den Subsystemen ist über verschiedene digitale Interfaces (MIL-Std-1553B, ARINC 429, RS 485) gewährleistet (siehe BILD 3).

Über den FCS-Bus nach MIL-Std-1533B kommuniziert der FCC über Remote Interface Units (RIU) mit den Grundsystemen, mit den Bremscomputern (BCU), den Navigations-

plattformen (engl. inertial reference system / global navigation satellite system, kurz IRS/GNSS) und den Luftdatencomputern (engl. air data computer, ADC). Die Laserhöhenmesser sind über RS 485 und das Aktuatoriksystem über ARINC 429 mit dem FCC verbunden. Die Bugradsteuerung und das Triebwerk werden mittels diskreter bzw. analoger Signale gesteuert, die vom FCC erzeugt werden. Der AMC ist über den Avionic-System-Bus (AVS-Bus) nach MIL-Std-1553B mit dem FCC verbunden.

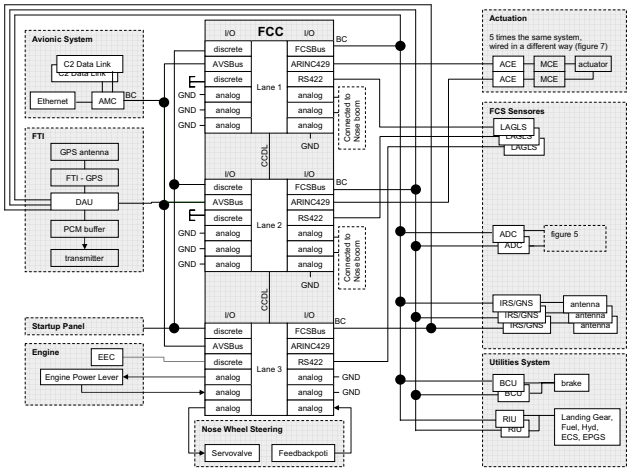

BILD 3: Systemarchitektur (aus [13])

Die Bedienerschnittstelle zum Operateur ist über den AMC gegeben. Der AMC kommuniziert über einen Datenlink mit der Ground Control Station (GCS). Von der GCS werden vom Operateur eingegebene Kommandos (High Level Commands) zum AMC gesendet und Kontrolldaten empfangen.

Um den Entwicklungsaufwand zu minimieren, wurde das Flugzeug so konzipiert, dass es auch als Rig verwendbar ist.

BILD 4: Barracuda als Rig

Im Rigbetrieb werden für Hardware-in-the-Loop-Simulationen die realen Sensoren und Geräte überbrückt, so dass den auf dem FCC implementierten Funktionen simulierte Daten zugeführt werden können. Hierdurch können auch Fehler im Gesamtsystem stimuliert und getestet werden. BILD 4 zeigt den Barracuda mit angeschlossener Simulation.

2.2.2. Sensorik

Die Sensorik dient der Erfassung des Flugzustandsvektors. Zum Sensorsystem des Barracuda gehören Luftdaten- und Navigationssystem.

Das Luftdatensystem (siehe BILD 5) besteht hardwareseitig aus dem Noseboom mit Sensoren für den Anströmwinkel und den Staudruck, zwei ADCs und einer Sonde zur Messung der Lufttemperatur (engl. total air temperature, TAT). Die ADCs sind über den FCS-Bus mit dem FCC verbunden. Auf dem FCC sind Funktionen zur Luftdaten-Konsolidierung und eine analytische Backup-Lösung für den Fall eines Sensorausfalls implementiert.

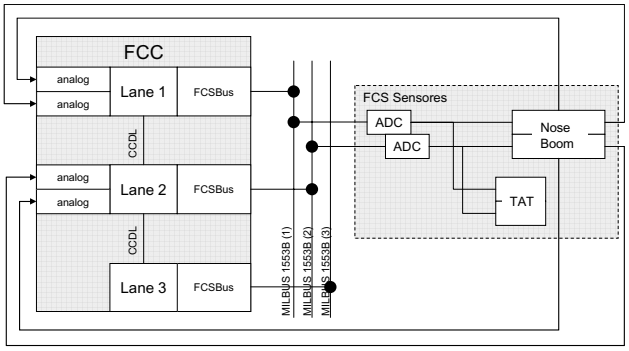

BILD 5: Luftdatensystem (aus [13])

Das Navigationssystem (siehe BILD 3) besteht hardwareseitig aus drei integrierten, DGPS-gestützten Inertialplattformen und drei Laserhöhenmessern. Es handelt sich um ein triplex-redundantes System. Die drei Plattformen sind über den FCS-Bus und die Laseraltimeter über RS 485 mit dem FCC verbunden. Auf dem FCC sind Funktionen zur Konsolidierung der Navigationsdaten implementiert.

2.2.3. FCC

Das Design des Flight Control Computers des Barracuda hat die Aufgabe, das definierte funktionale Redundanzkonzept auch in der Hardware abzubilden. Daher wurde in der FCC-Hardware eine Triplex-Architektur realisiert (siehe BILD 6)

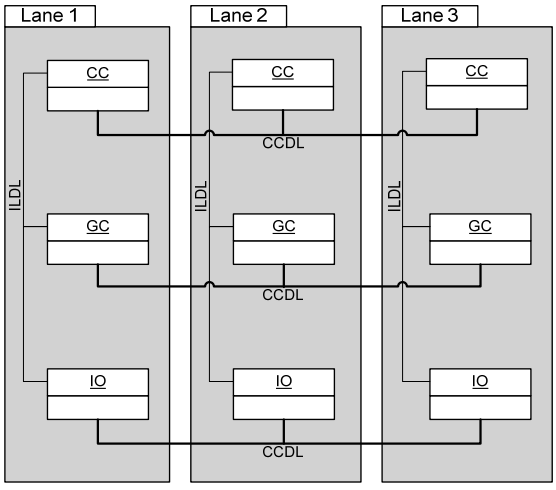

BILD 6: FCC-Architektur

Die gewählte Architektur besitzt die folgende Merkmale:

– Die funktionale Triplex-Architektur wird durch drei identische Kanäle im FCC abgebildet, die zusammen mit den zugehörigen Inertialsensor-Kanälen sogenannte Bahnen (engl. lanes) bilden, ähnlich den Fahrspuren einer Straße. Der Begriff Lane wird im folgenden auch synonym für Kanal verwendet.
– Nachdem für die Wahl aller anderen FCS-Geräte 'off-the-shelf' als oberstes Prinzip galt, verfügt der FCC über eine Vielzahl von digitalen Schnittstellen wie ARINC 429, MIL-Std-1553B und RS 485 sowie diskrete und analoge Ein- und Ausgänge.
– Jede der drei Lanes verfügt über zwei Microcontroller MPC565 zur Ausführung der Applikation (Control Computer, CC; Guidance Computer, GC) und jeweils einen weiteren MPC565 für die Verwaltung der diskreten und analogen Ein- und Ausgänge und für die ARINC-429-Schnittstelle.
– Sowohl zur Intra-Lane- als auch zur Cross-Lane-Kommunikation (engl. Intra-Lane Data Link, ILDL bzw. Cross-Lane Data Link, CCDL) wurde ein MTU-proprietäres, serielles Busprotokoll in Punkt-zu-Multipunkt-Topologie verwendet.

Für weiterführende Erläuterungen zur FCC-Hardware sei auf [4] verwiesen.

Die Software-/Software-Schnittstelle zu der vom FCC-Zulieferer MTU erstellten hardwarenahen Software wurde so gewählt, dass die MTU mit Ausnahme des MIL-Std-1553B-Services die Ein-/Ausgabe-Services für ARINC 429 und RS 485 sowie alle Built-In-Tests implementierte.

2.2.4. Aktuatorik

Das Bugfahrwerk des Barracuda wurde "off-the-shelf" vom Alphajet übernommen. Der Regelalgorithmus der Lenkbewegung ist auf dem FCC implementiert.

Die Ansteuerung des Triebwerkes erfolgt über eine elektromechanische Ansteuerung des Schubhebels.

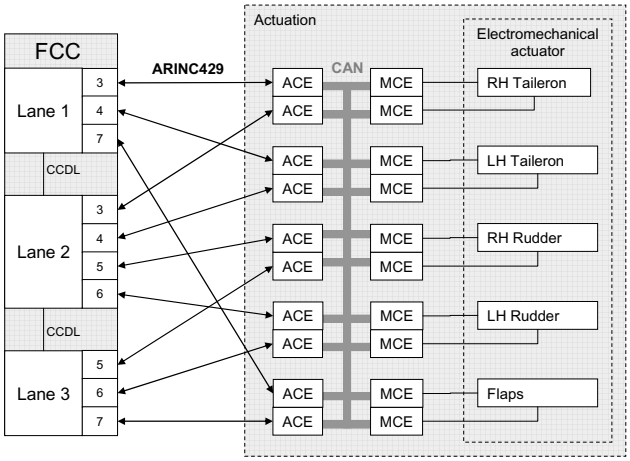

BILD 7: Aktuatorsystem (aus [13])

Die elektromechanischen Aktuatoren für die aerodynamischen Steuerflächen besitzen jeweils eine eigene duplexredundante Positionsregelungselektronik (engl. actuation control electronic, ACE; motor control equipment, MCE) (siehe BILD 7). Sie erhält die Positionvorgaben des FCC über ARINC 429.

Aufgrund des bereits erwähnten Redundanzsprunges von triplex auf duplex sind nur jeweils zwei FCC-Lanes mit den zwei Kanälen der Positionregelungselektronik für eine Steuerfläche verbunden.

2.3. Software Design

In der Flugsteuerungssoftware des Barracuda sind die oben genannten funktionalen Komponenten der Flugführung, Signalverarbeitungsalgorithmik, des Redundanzmanagements sowie des Health Monitoring implementiert. Desweiteren muss die Software verschiedene Infrastrukturdienste zur Verfügung stellen, wie zum Beispiel die Ansteuerung der Datenübertragung über die verschiedenen Bussysteme.

Die Randbedingungen bei der Auslegung sind:

– Abbildung der funktionalen Systemarchitektur (siehe BILD 8: Funktionaler Datenfluß
– Korrekte, deterministische und sichere Implementierung der digitalen Algorithmik

Aus diesen Randbedingungen resultierten die folgenden grundlegenden Design-Entscheidungen.

Die Software ist, ebenso wie die Rechnerarchitektur, triplex-similar ausgelegt. Bei diesem, in der militärischen Entwicklung geläufigen Ansatz kommt in allen redundanten Kanälen die identische Software zum Einsatz. Desweiteren gibt es keine Unterscheidung zwischen aktiven und „stand-by“ Komponenten. Alle Kanäle arbeiten gleichberechtigt im synchronen Parallelbetrieb und jeder Kanal steuert seinen eigenen Satz von Ausgängen an.

Zur Implementierung einer komplexen digitalen Regelung mit hohen Abtastraten in Kombination mit der Ausführung längerlaufender Prozesse wurde ein gemischt nicht-preemptiver/preemptiver periodischer tabellengestützter Scheduling-Algorithmus mit zur Laufzeit statischen Scheduling-Tabellen gewählt. Damit zu jedem Zeitpunkt die absolute Gleichheit der Rechenergebnisse der drei Kanäle gewährleistet werden kann, ist zusätzlich eine strikte Synchronisation der Lanes erforderlich.

Die funktionale System- und Subsystem-Unterteilung wird in der Software-Architektur mittels so genannter Domänen abgebildet. Die Domänen spiegeln sich in der Namensgebung von Parametern und Softwarekomponenten wider, bezeichnen aber ebenso Grenzen der Arbeitsteilung beziehungsweise Zuständigkeiten zwischen verschiedenen Systementwicklern und Disziplinen. So schließt z. B. die Domäne „ADS“ alle zum Luftdatensystem gehörigen Softwarefunktionalitäten ein, „NAV“ die der Navigation und „MIL“ den Dienst und Gerätetreiber für den MIL-Std-1553B Bus.

Bezüglich des Datenflusses zwischen verschiedenen Modulen sind zwei Konzepte erwähnenswert:

Zwischen einzelnen Softwaremodulen wird der Datenaustausch grundsätzlich nicht über globale Variablen durchgeführt sondern durch ein „Pull-Prinzip“ entkoppelt. Benötigt

ein Modul ein Datum, so muss es sich dieses von der Datenquelle mittels Aufruf einer Accessorfunktion abholen. Eigene Daten, z. B. Berechnungsergebnisse, werden wiederum anderen Modulen über eigene Accessoren zur Abholung bereitgestellt.

Findet ein Datenaustausch über eine Systemgrenze (Intra-Lane, Cross-Lane oder nach extern) hinweg statt, wird dieser Schritt von der Applikationssoftware durch eine sogenannte Daten-Präsentations-Schicht (engl. data presentation layer) entkoppelt. In dieser Schicht werden alle medienspezifischen Aufgaben abgewickelt:

– Bedienung von Bus-Protokollen,
– Low-Level-Dekodierung und -Kodierung,
– Fixed-Point-Skalierungen und
– Umrechnung von den gerätespezifischen Einheiten in die in der FCC-Software durchgängig verwendeten SI-Einheiten.

Zur Applikationssoftware hin findet eine Zwischenspeicherung aller Daten statt. Auf diese Daten kann dann ebenfalls über Accessor- und Mutatorfunktionen zugegriffen werden.

Eine bestimmte Softwarekomponente benötigt durch diese Kapselung der Schnittstellen zur Hardware keine Kenntnis über die Quelle eines Signals oder über dessen Ausprägung, z. B. bei einem Bustransfer; nur der Signalname muss bekannt sein. Auch eine Reallozierung von Modulen von einem Prozessor zum anderen innerhalb der Lane ist leicht und ohne Änderung der Komponente selbst durchführbar.

Eine besondere Synergie wird zusätzlich erreicht, indem bestimmte Gerätetreiber (u. a. für MIL-Std-1553B) in eine generische und eine hardwarespezifische Schicht geteilt werden, die gemeinsam einen Dienst (engl. service) zur Kommunikation über diese Schnittstelle gegenüber der Anwendungssoftware anbieten. Somit kann der Quellcode der Daten-Präsentations-Schicht, welcher nur die generische Diensteschicht anspricht, sowohl in der FCC-Software als auch in den verschiedenen Testsystemen ohne Änderungen verwendet werden.

3. DER ENTWICKLUNGSPROZESS

Für die militärische Flugzeugentwicklung ist das V-Modell ein besonders wichtiger Standard [1]. Dieses Entwicklungsmodell ist hinreichend beschrieben, und es soll an dieser Stelle auch nicht näher darauf eingegangen werden. Es sei nur angemerkt, dass das V-Modell einen vorwärtsgerichteten Ablauf in der Spezifizierung vom Gesamtflugzeug bis hinunter zur Geräteebene vorsieht und bei dem hier vorgestellten integrierten Prozess von Anfang an Randbedingungen der Software-Entwicklung berücksichtigt werden müssen [10].

3.1. System-Entwicklungsprozess

Die Grundzüge des System-Entwicklungsprozesses unterscheiden sich nicht wesentlich vom klassischen Vorgehen des V-Modells [6]. Abweichend davon wurde jedoch bei der Definition des Begriffs „Low-Level-Anforderung“ (engl. low-level requirement) eine innovative Methode der Notation gewählt. Darauf wird bei der Beschreibung des Software-Entwicklungsprozesses näher eingegangen.

Im Folgenden wird für Vorgehen und Methoden die gewählte Strategie vorgestellt und erläutert, an welchen Stellen, bedingt durch den experimentellen Charakter des Projekts, davon abgewichen wurde. Anschließend werden die verwendeten Werkzeuge vorgestellt.

3.1.1. Vorgehensweise

Wie beispielsweise in [6] beschrieben, werden ausgehend von den Top-Level-Anforderungen die Systemanforderungen abgeleitet. Diese werden soweit heruntergebrochen, bis der vollständige Satz an Low-Level-Anforderungen für die Gerätespezifikationen oder die Softwarespezifikation vorliegt.

Für jede Anforderung wird festgehalten, aus welcher übergeordneten Anforderung sie sich ableitet und warum sie genau so abgeleitet wurde. Diese Verfolgbarkeit (engl. traceability) und der Schritt der Validierung kann bei einem Experimentalprojekt nicht immer konsequent durchgehalten werden, da es für die Wahl des Designs auch Lösungen gibt, die zwar eine einmal gestellte Top-Level-Anforderung verletzten, diese Verletzung aus Projektsicht aber trotzdem akzeptabel ist.

Im Rahmen der Integrationstests wird nachgewiesen, dass jede Anforderung auch eingehalten wird.

Setzt man voraus, dass jede einzelne Anforderung als korrekt validiert wurde, und alle Anforderungen korrekt implementiert wurden, dann ergibt sich, dass man das gewünschte System vor sich hat.

3.1.2. Methoden

Auch wenn an manchen Stellen das Systemdesign vor der Analyse der Anforderungen gegeben war, zum Beispiel, weil Projektpartner bereits fertige Systeme eingebracht hatten [5], so wurde für das gesamte FCS eine vollständige und konsistente Anforderungsbasis erstellt.

Die Validierung erfolgte im selben Dokument jeweils mit einem beschreibenden Absatz. Anforderungen wurden in einer Anforderungsdatenbank verwaltet, auf die weiter unten noch eingegangen wird.

Als Nachweis der Testabdeckung wurde eine Verifikation-Matrix etabliert, in der jede Anforderung mit der durchzuführenden Nachweisart, z. B. Simulation, Analyse, Test oder Review, attributiert und mit dem Nachweis-Ergebnis verlinkt wurde.

3.1.3. Werkzeuge

Zur Erfassung der System- und Schnittstellen-Anforderungen wurde das Werkzeug DOORS 6.0 der Firma Telelogic verwendet.

Die Werkzeuge zur Modellierung der Low–Level-Anforderungen werden der Werkzeugumgebung der Software-Entwicklung zugeordnet und dort beschrieben.

DOORS bedient sich der Vorteile einer Datenbank, in der für jedes System eine Datenbasis erstellt wird, in der jeder Datensatz eine eindeutig identifizierbare Anforderung repräsentiert. Aus der Anforderungsanalyse abgeleitete Anforderungen können über Verlinkungen mit ihrer übergeordneten Anforderung verbunden werden, um eine Nachverfolgbarkeitsanalyse (engl. traceability analysis) zu ermöglichen.

Gleiches gilt für die Nachweisdokumente. Hier wird jeder Test als einzelner Datensatz definiert und somit für jede getestete Anforderung ein Verweis erstellt. Sind alle Anforderungen vollständig mit der Nachweismatrix verlinkt, dann ist der Nachweis der vollständigen Testabdeckung automatisch erbracht.

In BILD 12 ist ein Ausschnitt aus der DOORS-Beschreibung der Health-Monitor-Funktionalität dargestellt.

3.2. Software-Entwicklungsprozess

Der Entwicklungsprozess für die FCC-Software des Barracuda muss im wesentlichen zwei Anforderungsgruppen gerecht werden.

Zum einen muss die Software, und damit der Prozess, die extrem hohen Systemanforderungen an Sicherheit und Fehlerfreiheit der Implementierung erfüllen. In diesem Zusammenhang liegt ein Hauptaugenmerk auf der Zertifizierbarkeit der Software. Auch wenn die FCS-Software im jetzigen Ausbauzustand keiner formalen Zertifizierung unterlag, wurden doch Prozess und Software so konzipiert, dass eine Zulassung nach den hohen Anforderungen des Zulassungsstandards für Onboard-Software, RTCA DO-178B [9], jederzeit möglich ist.

Zum anderen muss die Software-Entwicklung aber in der Lage sein, die hohe Dynamik einer innovativen Prototypenentwicklung und die kurzen Entwicklungszyklen mitzutragen. Hierbei treten vor allem die Aspekte der schnellen, effizienten und fehlerfreien Code-Erstellung, der Wartbarkeit und der Änderbarkeit in den Vordergrund.

Für beide Themengruppen sind eine übersichtliche Strukturierung, durchgängige Benamung von Bezeichnern, die Konformität zu Software-Standards sowie eine gute Wartbarkeit des Codes unverzichtbar.

3.2.1. Anwendung der RTCA DO-178B

Der Prozess, nach dem die FCC-Software des Barracuda entwickelt wurde, leitet sich unmittelbar aus den Zulassungsanforderungen ab. Der anzuwendende Zulassungsstandard für Onboard-Software ist die RTCA DO-178B. Durch die Zulassung des Luftfahrzeugs Barracuda nach LTF 1550, Kategorie 2, jedoch unter Kategorie-1-Bedingungen für die erste Phase der Erprobung, ist eine dedizierte Zulassung des FCS und somit der FCC-Software nicht gefordert. Da jedoch eine vollständige Zulassung nach Kategorie 2 vorgesehen ist, wurde durch das Entwicklungsteam ein Software-Level D* für die FCC-Software festgelegt, in dem die Ziele (engl. Objectives) nach DO-178B für den Software-Entwicklungsprozess nach Level A zu erfüllen sind, die Verifikationsziele jedoch nur nach Level D.

Die Reduktion der Anforderungen an die Verifikationsziele leitet sich unmittelbar aus den reduzierten Zulassungsanforderungen an das FCS für den Betrieb unter Kategorie-1-Bedingungen ab. In den Verifikationsaktivitäten wurde stattdessen ein Schwerpunkt auf die Entwicklung von dedizierten Verifikationsmethoden und den Aufbau von Werkzeugketten zur automatisierten Verifikation gelegt, die die Grundlage für eine spätere vollständige Zertifizierung nach Kategorie 2 schaffen.

Der Software Configuration Management Prozess wurde für alle Artefakte der Software-Entwicklung und -Verifikation bis in den System-Entwicklungsprozess hinein umgesetzt, um eine vollständige Nachvollziehbarkeit der Software-Bauzustände zu gewährleisten.

Die Instanziierung des Software Quality Assurance Prozesses sowie des Certification Liaison Prozesses wurden aufgrund der reduzierten Anforderungen beim Betrieb unter Kategorie-1-Bedingungen an das FCS nicht durchgeführt.

Für eine vollständige Zulassung des Luftfahrzeugs nach Kategorie 2 wird mit einer Zulassung der FCC-Software nach DO-178B, Level C gerechnet. Die hierfür notwendigen Voraussetzungen wurden in der abgeschlossenen Entwicklungs- und Erprobungsphase erarbeitet und werden für die vollständige Kategorie-2-Zulassung in die Produktionsreife überführt.

3.2.2. Vorgehensweise

Als Vorgehensweise für die Software-Entwicklung wurde das Evolutionäre Prototyping von Softwarekomponenten (EPS) gewählt. Das EPS unterscheidet sich vom konventionellen Evolutionären Prototyping [1] dadurch, dass das Prototyping separat auf die einzelnen Software-Subsysteme angewendet wird und nicht auf das gesamte Software-System. Der Software-Entwicklungsprozess wird also lokal zu jedem und separat für jedes Software-Subsystem - teilweise mehrfach - durchlaufen, bevor eine Integration des gesamten Software-Systems durchgeführt wird. Zudem setzt sich das EPS bis in den System-Entwicklungsprozess hinauf fort, in dem wesentliche Komponenten der Sensorsignal-Vorverarbeitung und Konsolidierung sowie der digitalen Regelung graphisch entwickelt, gegen die ihnen zugrundeliegenden Anforderungen verifiziert und zur Validierung dieser Anforderungen im Rahmen von Simulationen verwendet werden.

Die Technologie, die die Anwendung des EPS erst ermöglicht, ist die modellbasierte Entwicklung (engl. model-based design, MBD) mit automatisierter Codegenerierung. Hierdurch können einerseits die System-Entwickler auf einer adäquaten Abstraktionsebene des Modells ihre Design-, Verifikations- und Validierungsaktivitäten durchführen und andererseits die Software-Entwickler das Modell effizient in Quellcode hoher Qualität überführen. Gerade die automatisierte Codegenerierung verlagert wesentliche Teile des Software-Designs und –Codings in die Konfiguration der Codegeneratoren, in der die Abbildungsregeln von Modell zu Quellcode festgelegt werden. Hier werden zentrale Software-Design-Entscheidungen getroffen und

somit die Qualität des generierten Quellcodes festgelegt. Dies betrifft die Verifizierbarkeit und Wartbarkeit des Quellcodes, aber auch die Behebung von Software-Fehlern, da die Korrektur der Konfiguration an der fehlerhaften Stelle sofortige Auswirkung auf alle Modellelement-Instanzen hat und somit implizit alle fehlerhaften Stellen im Quellcode korrigiert sind.

Design- und Coding-Entscheidungen, die nicht in der Konfiguration der Codegeneratoren umgesetzt werden können, weil sie von der Art der Verwendung der Modellelemente abhängen, werden in Form von Software Model Standards (in Ergänzung der DO-178B Software Standards) den System-Entwicklern zur Verfügung gestellt. Die Einhaltung der Software Model Standards wird bei der Übernahme eines Modells in den Software-Entwicklungsprozess durch einen computer-unterstützten Modell-Review als Eingangskriterium überprüft.

Es tritt somit eine Entkopplung der Software-Design- und -Coding-Aktivitäten von der Bereitstellung der System-Anforderungen oder des System-Designs auf, da erstere in die Codegenerator-Konfiguration oder die Model-Standards einfliessen und letztere hauptsächlich die Nutzung der Codegeneratoren anstossen.

Im Bereich der Verifikation wurde automatisiert generierter Quellcode wie manuell erstellter Quellcode behandelt. Eine Werkzeugqualifizierung der Codegeneratoren wurde nicht vorgenommen, so dass eine Reduktion des Verifikationsaufwandes von automatisiert generiertem Quellcode gegenüber manuell erstellem Quellcode nicht zu verzeichnen war.

3.2.3. Methoden

Als grundlegende Software-Entwicklungsmethode wurde das objektbasierte Design verwendet, das durch die folgenden, im Design der FCC-Software direkt umgesetzten Paradigmen charakterisiert wird:

- Objekt- und Klassenbildung: Gemeinsame Repräsentation des Zustands und Verhaltens von Funktionalität in Objekten und der Zusammenfassung gleicher Zustands- und Verhaltensbeschreibungen zu Klassen.
- Abstraktion: Verdeckung der Implementierungskomplexität von Teil-Funktionen durch deren Repräsentation als Klassen mit einem internen Zustand und Verhalten.
- Kapselung: Bereitstellung der Funktionalität und Daten einer Klasse ausschliesslich über deren Methoden (Funktionen) und Accessoren bzw. Mutatoren, d. h. keine direkte Zugreifbarkeit auf Daten von aussen z. B. durch deren globale Deklaration.

Die dem objektorientierten Design zuzuordnenden Paradigmen Vererbung, Polymorphismus und späte Bindung blieben unberücksichtigt, da ihre Zertifizierbarkeit im Kontext von DO-178B nicht oder nur mit grossem Aufwand sicherzustellen ist.

Basierend auf objektbasiertem Design wurden die einzelnen verwendeten Software-Entwicklungsmethoden durch die Typen der zu entwickelnden Software-Komponenten bestimmt. Zu unterscheiden sind hier:

- Prozess-Scheduler, Mathematik-Bibliothek, Teile des Health Monitors, FTI-Funktionen, generische Funktionen und Treiber für MIL-Std-1553B, ARINC 429, RS 485: Manuelles, objektbasiertes Design und Codegenerierung,
- Prozess-Schedules und MIL-Std-1553B-Buslisten: automatisierte Schedule-Planung aus einem Prozessmodell,
- Anpassbare Parameter und Wegpunktlisten: System-Design und automatisierte Codegenerierung aus Matlab-Skripten,
- Datenpräsentationsschicht für MIL-Std-1553B, ARINC 429, digitale und analoge Ein-/Ausgänge, CCDL und ILDL: System-Design und automatisierte Codegenerierung aus Interface Control Dokumenten (ICDs),
- Teile des Health Monitors: System-Design und automatisierte Codegenerierung aus einer abstrakten Spezifikation,
- Inertial-, Laseraltimeter- und Luftdaten-Sensorsignal-Vorverarbeitung und -Konsolidierung, Flugregelung, Autopilot, Moding, Flugführung: System-Design und automatisierte Codegenerierung aus regelungstechnischen Datenfluss- und Zustandsübergangsmodellen,
- Software-Architektur und System-Moding: objektbasiertes Design und automatisierte Codegenerierung aus UML-Modellen unter Verwendung von Klassendiagrammen und Statecharts.

Insgesamt wurde etwa 95% der gesamten FCC-Software automatisiert generiert.

Die Integration der Software-Komponenten fand rein auf der Quellcode-Ebene statt. Daher war die strikte Einhaltung von gemeinsamen Namensregeln bis hinauf zum System-Design notwendig.

Die verwendeten Software-Verifikationsmethoden entsprechen denen des DO-178B:

- Reviews: die Modelle, ICDs und der Quellcode wurden Reviews unterzogen, bei denen jene Ziele des DO-178B für Level D überprüft wurden, die nicht bereits durch die Anwendung der verbleibenden Verifikationsmethoden abgedeckt wurden.
- Analysen: es wurden computerunterstützte Schedulability-Analysen durchgeführt, um nachzuweisen, dass sowohl die berechneten Prozess-Schedules als auch die MIL-Std-1553B-Buslisten alle an sie gestellten Anforderungen erfüllen; statische Quellcode-Analysen wurden durchgeführt, um die Konformität des Codes mit den Softwarestandards für den automatisiert überprüfbaren Teil der Regeln nachzuweisen.
- Tests: für selektierte kritische Software-Komponenten wurden Low-Level-Tests auf einem Prätarget (Single-Board-Computer) mit instrumentierungsfreier Analyse der strukturellen Statement- und Branch-Abdeckung durchgeführt; die regelungstechnischen Datenfluss- und Zustandsübergangsmodelle wurden Integrationstests auf dem Prätarget mit struktureller Abdeckungsanalyse unterzogen; Hardware-/Software-Integrationstests auf dem FCC wurden im Rahmen der System-Integrationstests durchgeführt.

Ziel der abgeschlossenen Entwicklungsphase des Barracuda in Bezug auf die Verifikationsmethoden war deren Entwicklung und Verfeinerung, so dass nicht alle der beschriebenen Methoden auf alle Software-Komponenten

angewendet wurden. Vielmehr erfolgte ihre Anwendung zum Teil exklusive, oder die aus Systemintegrationstests gewonnenen Erkenntnisse über die Integrität der Software-Komponenten wurden als in dieser Phase ausreichend eingestuft. Von einer vollständigen Durchführung der Verifikationsaktivitäten für alle Software-Komponenten wird für die vollständige Kategorie-2-Zertifizierung ausgegangen.

3.2.4. Werkzeuge

Die Hauptaspekte bei der Prozessimplementierung sowie bei der Auswahl und Konfiguration kommerzieller Werkzeuge und der Eigenentwicklung von Tools waren:

- Erfüllung der Sicherheits- und Qualitätsanforderungen and den Software-Code
- Integrierbarkeit des Codes aus Teilprozessen in der Gesamt-Software
- Höchstmöglicher Grad an Automatisierung
- Durchgängige Verfolgbarkeit von Änderungen durch Einsatz von werkzeuggestütztem Konfigurationsmanagement
- "Single Source“-Ansatz, d.h. für jeden Sachverhalt immer nur eine Spezifikation (Modell, Datenbankmodul, etc.) zur Vermeidung von Fehlern und Mehraufwänden

Die in 3.2.3 beschriebenen Entwicklungsmethoden wurden im Wesentlichen unter Verwendung der folgenden Werkzeuge umgesetzt:

- Design von UML-Modellen und Codegenerierung mittels Rhapsody (I-Logix)
- Design regelungstechnischer Datenflussmodelle in MATLAB/Simulink (MathWorks), Codegenerierung mittlels Targetlink (dSPACE)
- Spezifikation der Datenpräsentationsschicht und des Health Monitor Logiknetzwerkes Anforderungs-Management-Werkzeug DOORS (Telelogic), Codegenerierung mittels der DOORS-eigenen Skriptprogrammiersprache "dxl".

Neben den beiden Teilprozessketten Simulink/Targetlink und Rhapsody, bei denen die Anpassung der Codegenerierung sowie die Erstellung von Software Model Standards vorgenommen werden mussten, handelt es sich bei allen anderen Teilprozessen der Software-Erzeugung und der Integration um Eigenentwicklungen.

Die FCC-Software des Barracuda wurde in der Programmiersprache C erstellt. Dabei kam aus Aspekten der Software-Sicherheit nur eine Untermenge von C zum Einsatz.

3.3. Synergieeffekte zwischen System- und Softwareentwicklung

Voraussetzungen für eine effiziente und fehlerfreie Entwicklung sicherheitskritischer Systeme und Software in dem hier beschriebenen Umfeld sind ein definierter Entwicklungsprozess sowie der Einsatz leistungsfähiger Methoden und Werkzeuge.

Der durchweg positive Verlauf des Projekts hat gezeigt, dass der bei der Entwicklung des Barracuda-FCS etablierte Prozess vielfältige Vorteile bietet.

An erster Stelle ist in diesem Kontext der Übergang von der klassischen Software-Entwicklung hin zu modellbasierten Techniken zu nennen. Durch die Verwendung von Modellen als Beschreibungsmittel im Entwicklungsprozess entsteht eine gemeinsame Sprache, auf deren Basis die Kommunikation zwischen System- und Software-Entwicklern harmonisiert wurde.

Der höhere Abstraktionsgrad bei der Darstellung von komplexen Zusammenhängen mittels eines Modells, beispielsweise als UML-Statechart (siehe BILD 11), ermöglicht ein besseres, intuitives Verständnis der Sachlage für einen größeren Kreis von Entwicklern. Dieser Vorteil und die Tatsache, dass bei dem hier vorgestellten, integrierten Entwicklungsprozess die Modelle teilweise selbst Anforderungen darstellen, ermöglichen es, Charakteristiken guter Anforderungen, insbesondere Eindeutigkeit, Konsistenz und Vollständigkeit, leichter sicherstellen.

Es hat sich gezeigt, dass solche Modelle darüberhinaus eine ideale Diskussionsgrundlage für diziplinenübergreifende Aktivitäten darstellen. Reviews lassen sich somit deutlich zeit- und kosteneffizienter und mit höherer Qualität durchführen.

Da die Modelle zudem als Grundlage für eine automatisierte Dokumenten- und Codegenerierung dienen, lassen sich aus der traditionellen Prozesskette manuelle und damit auch potentiell fehlerbehaftete Aktivitäten weitestgehend eliminieren. Verfolgbarkeit und Konsistenz sind bei der automatisierten Generierung implizit gegeben, wenn auch der zulassungsrelevante Nachweis noch zu erbringen ist.

4. AUSBLICK

Es konnte gezeigt werden, dass die entwickelten Architekturen sowie die gewählten Vorgehensweisen, Methoden und Werkzeuge sich im Rahmen eines Prototypen-Entwicklungsprogramms als geeignet erwiesen haben.

Es gelang, einen kosten- und zeiteffizienten Entwicklungsprozess umzusetzen.

Trotz des augenscheinlichen Widerspruchs zwischen Kosteneffizienz und Qualität konnte ein hoher Anspruch bezüglich Sicherheit und Zuverlässigkeit erfüllt werden.

Die entwickelten Verfahren, Methoden und Werkzeuge können wegweisend für zukünftige Entwicklungen eingesetzt werden.

Der integrierte Ansatz der System- und Software-Entwicklung führte zu einem transparenten Entwicklungsprozess und zu einem vor allem zeiteffizienten Projektverlauf.

Die Entwicklung des Barracuda ist noch nicht abgeschlossen. Zukünftig erfolgt der Ausbau zum Versuchsträger für zukünftige, agile autonome Einsatzsysteme.

5. REFERENZEN

[1] Anderson, Dorfman: “Aerospace Software Engineering: A Collection of Concepts”, Progress in Astronautics and Aeronautics Series, V-136, AIAA, 1991
[2] BMVg: "Entwicklungsstandard für IT-Systeme des Bundes – Vorgehensmodell Juni 1997", 1997
[3] BMVg, WTD 61: "LTF1550-001 - Sonderbestimmungen bei Prüfung und Zulassung unbemannter Luftfahrzeuge der Bundeswehr", 2004
[4] Gottmann, T: "Auslegung des Barracuda Gesamtsystems", Deutscher Luft- und Raumfahrtkongress 2006, Braunschweig
[5] Hierlwimmer, R., et al.: "Systemkomponenten für den Barracuda – Entwicklung eines Flight Control Computers und der Schubdüse", Deutscher Luft- und Raumfahrtkongress 2006, Braunschweig
[6] IEEE: "IEEE STD 1220-1998 – Standard for Application and Management of the System Engineering Process", IEEE, New York, 1999
[7] Jarasch, G., Schönhoff, A.: "Integration von UAVs in den kontrollierten Luftraum", Deutscher Luft- und Raumfahrtkongress 2002, Stuttgart
[8] Moormann, D. et al: "Autonome Flugregelung des UAV-Demonstrators Barracuda", Deutscher Luft- und Raumfahrtkongress 2006, Braunschweig
[9] RTCA: „Software Considerations in Airborne Systems and Equipment Certification”, DO-178B, Washington D.C., USA, 1992
[10] Schönhoff, A., et al.:"Moderner Entwicklungsprozeß für sicherheitskritische Software", Deutscher Luft- und Raumfahrtkongress 2001, Hamburg
[11] Seitz, R., Westerbuhr, F.:"Advanced Mission Computer", Deutscher Luft- und Raumfahrtkongress 2002, Stuttgart
[12] Stütz, P., Schulte, A.: "Missionsmanagement für ein hochfliegendes, unbemanntes Luftfahrzeug im kontrollierten Luftraum", DGLR-Workshop T5 UAV, Braunschweig, 2002
[13] Wetteborn, D.: "Analyse der internen Kommunikationsarchitektur eines Flugführungssystems", Studienarbeit 2005, München

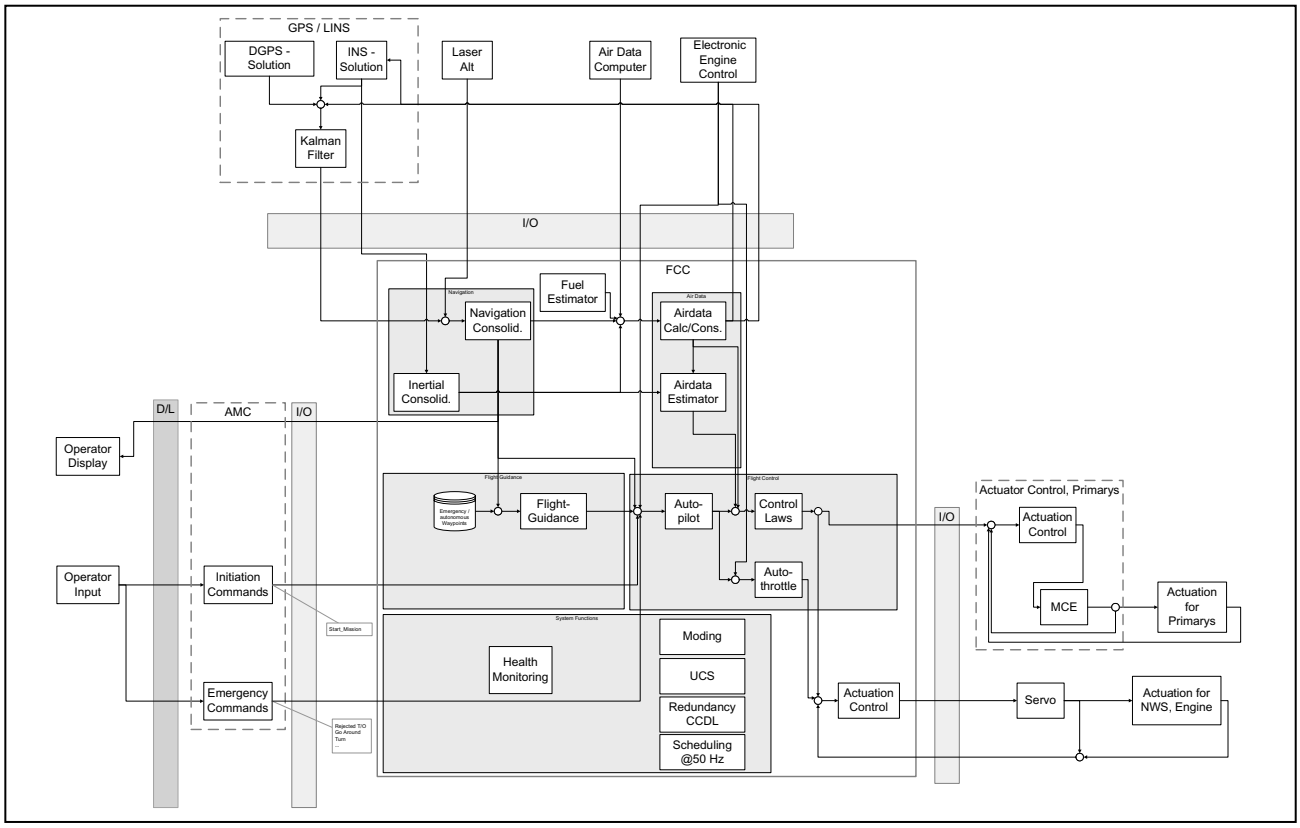

BILD 8: Funktionaler Datenfluß

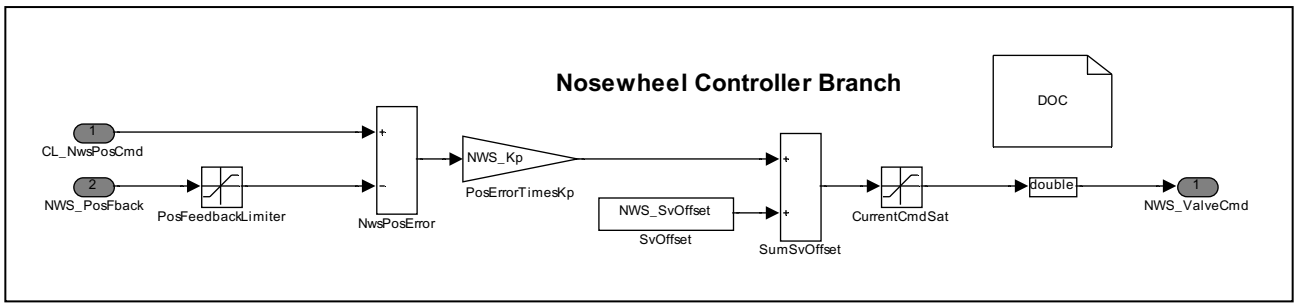

BILD 9: Ausschnitt aus einem Simulink-Modell

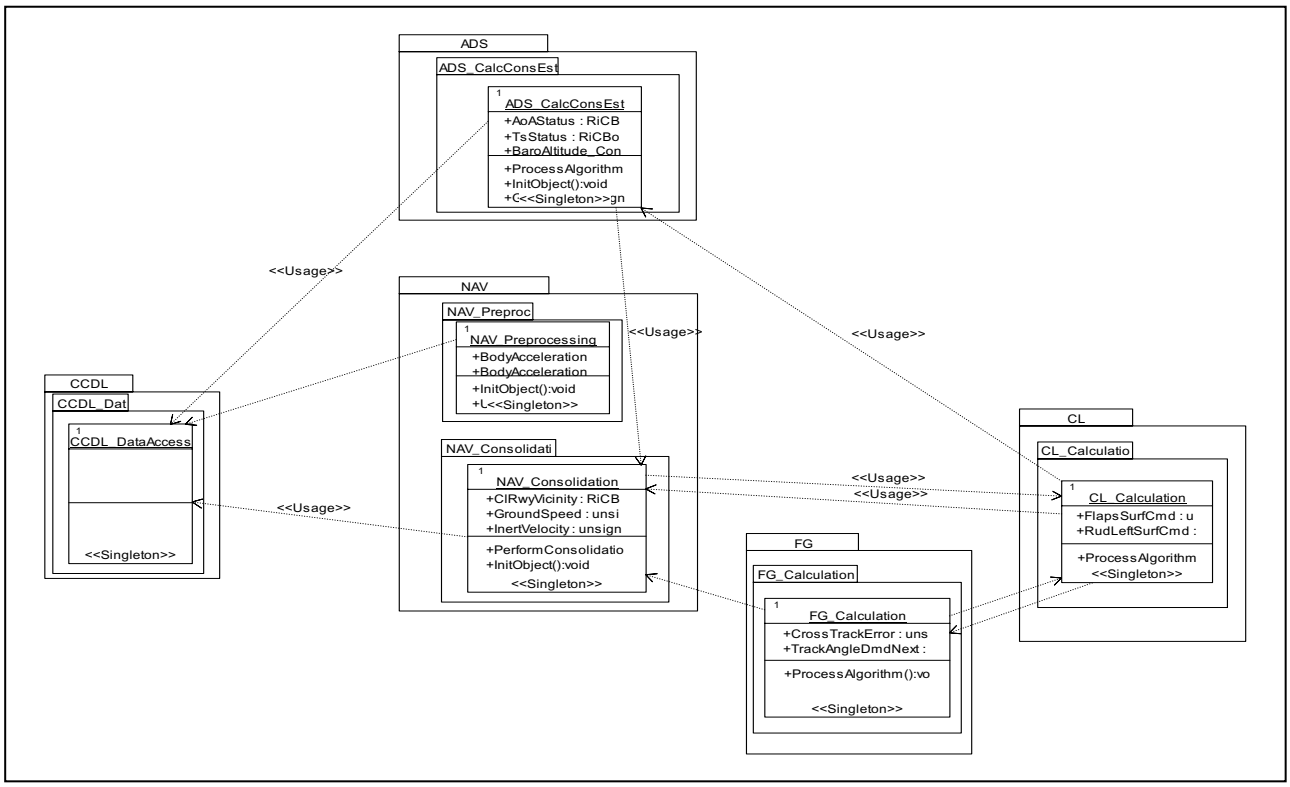

BILD 10: Ausschnitt aus einem Rhapsody-Klassendiagramm

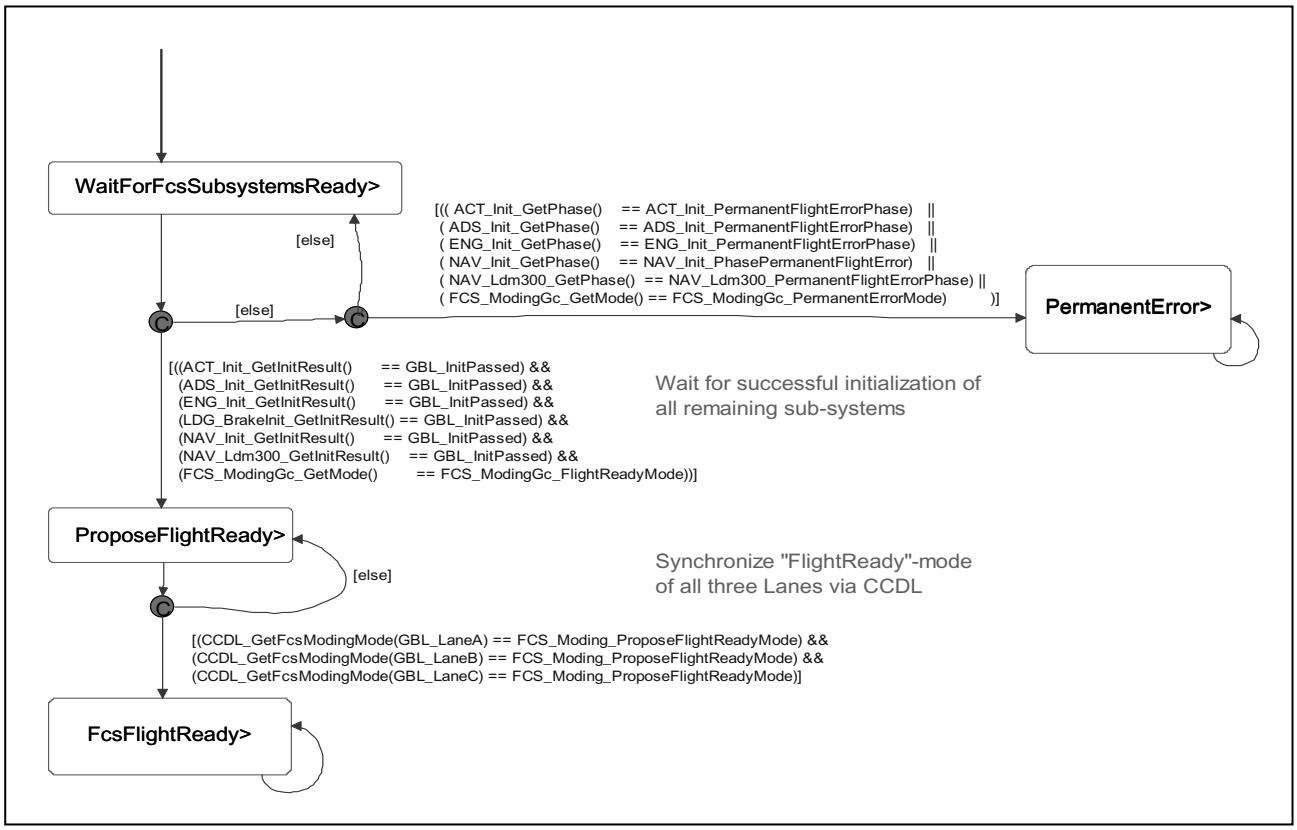

BILD 11: Ausschnitt aus einem Rhapsody-Zustandsübergangsdiagramm

Spezifizierung der Eingangssignale:

ICD-U-279-M-0007 - Incoming	Domain	Class	Signal Name	Comp	Comp Value	Guard Condition
2.9.2 IRS/GNNS	NAV					
2.9.2.1 IRS Failed	NAV	IrsDiscrete	Failed	!=	GBL_NotFailed	NAV_Init_GetInitResult() == GBL_InitPassed
2.9.2.2 Over Temperature	NAV		OverTemperature	!=	GBL_False	NAV_Init_GetInitResult() == GBL_InitPassed
2.9.3 LDM300	NAV	Ldm300				
2.9.3.1 Init Result	NAV	Ldm300	InitResult	!=	GBL_InitPassed	NAV_Ldm300_GetPhase() == NAV_Ldm300_InitPassedPhase || NAV_Ldm300_GetPhase() == NAV_Ldm300_PermanentFlightErrorPhase
2.9.3.2 Error	NAV	Ldm300	Error	!=	GBL_NoError	NAV_Ldm300_GetInitResult() == GBL_InitPassed
2.9.4 Nav_Consolidation	NAV	Consolidation				
2.9.4.1 Status Hybride Latitude [0]	NAV	Consolidation	StatusHybLatitude	!=	0	

Spezifizierung der Ausgangssignale als logische Verknüpfung der Eingangssignale:

Outgoing Signal	Output Name	Latched	Boolean Expression of Preproc. Inputs (True means fail)
1.1 Nav State	NavState	False	NAV_Init_InitResult | NAV_IrsDiscrete_Failed | NAV_OverTemperature | NAV_Ldm300_InitResult | NAV_Ldm300_Error | NAV_Consolidation_StatusHybLatitude | NAV_Consolidation_StatusHybLongitude | NAV_Consolidation_StatusHybAltitude | NAV_Consolidation_StatusHybVelocityN | NAV_Consolidation_StatusHybVelocityE | NAV_Consolidation_StatusHybVelocityD | NAV_Consolidation_StatusInertPhi | NAV_Consolidation_StatusInertTheta | NAV_Consolidation_StatusInertPsi | NAV_Consolidation_StatusInertRollRate | NAV_Consolidation_StatusInertPitchRate | NAV_Consolidation_StatusInertYawRate | NAV_Consolidation_StatusInertNx | NAV_Consolidation_StatusInertNy | NAV_Consolidation_StatusInertNz | NAV_Consolidation_StatusLaser
1.2 Ads State	AdsState	False	ADS_Init_InitResult | ADS_AdcFailure | ADS_PowerSplyCardFail | ADS_PtCardFail | ADS_PsCardFail | ADS_CpuCardFail | ADS_TatInputFail | ADS_CalcConsEst_AoAStatus | ADS_CalcConsEst_MachStatus | ADS_CalcConsEst_PsStatus | ADS_CalcConsEst_TsStatus | ADS_Noseboom_HeatMon

BILD 12: Ausschnitt aus einem DOORS-ICD