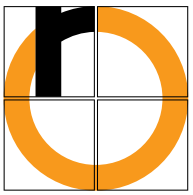
Hochschule **Rosenheim**
University of Applied Sciences



MASTERARBEIT

Entwicklung eines Microcontroller-basierten Prototypen für die echtzeitfähige Sensordatenverarbeitung im Rahmen des Hagelflieger-Projekts RO-BERTA

Autor: Dipl.-Ing(FH) Andreas Bernhardt
Fachbereich: Elektro- und Informationstechnik
Spezialisierung: Kommunikationstechnik, Mess- und Regelungstechnik

Erstprüfer: Prof. Dr.-Ing. Peter Zentgraf
Zweitprüfer: Prof. Dr.-Ing. Martin Versen

Abgabedatum: 25.09.2012

Erklärung gemäß § 31,5 RaPo

Hiermit erkläre ich, dass ich die vorliegende Masterarbeit

Entwicklung eines Microcontroller-basierten Prototypen für die echtzeitfähige Sensordatenverarbeitung im Rahmen des Hagelflieger-Projekts RO-BERTA

selbständig verfasst, noch nicht anderweitig zu Prüfungszwecken vorgelegt, keine anderen als die vorgegebenen Quellen und Hilfsmittel benutzt, sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

_____ _____

Ort, Datum Andreas Bernhardt

Danksagung

Ich bedanke mich bei Prof. Dr.-Ing. Martin Versen für seine Zusage zur Betreuung meiner Masterarbeit sowie für den von ihm stammenden Vorschlag, dem Call for Papers zur europäschen Fachkonferenz im Bereich digitale Signalverarbeitung, EDERC2012, zu folgen.

Dank gilt außerdem den Mitarbeitern der Hochschule Rosenheim, v.a. dem RO-BERTA-Kernteam, bestehend aus dem Teamleiter Prof. Dr.-Ing. Peter Zentgraf, welcher auch die Funktion des Erstbetreuers dieser Arbeit übernommen hat, Dipl.-Ing.(FH) Martin Heigl und Dipl.-Ing.(FH) Peter Viehhauser für die hervorragende Zusammenarbeit, sowie Herrn Stefan Kipfelsberger für seine Unterstützung in schaltungstechnischen Fragen.

Martin Laake, als studentische Hilfskraft im Labor für Mess- und Regelungstechnik tätig, hat durch seinen Entwurf des EAGLE-Platinenlayouts für den HAILsens Prototypen, sowie weiterer Platinen für das Boardsystem HAIL ebenfalls einen anerkennenswerten Beitrag geleistet, wofür ich ihm sehr dankbar bin.

Beeindruckt hat mich auch die Hilfsbereitschaft, das Engagement und die Kompetenz der Texas Instruments Mitarbeiter und der Ingenieure im E2E-Forum, welche mir einige Male Auswege aus entwicklungstechnischen Sackgassen geschaffen haben.

Rosenheim, 25.09.2012

Inhaltsverzeichnis

Abkürzungsverzeichnis

Abkürzung	Definition
A/D	Analog-to-Digital
ADC	Analog-to-Digital Converter
ASCII	American Standard Code for Information Interchange
API	Application Program Interface
BIOS	Basic Input Output System
CAN	Controller Area Network
CCS	Code Composer Studio
CMOS	Complementary Metal Oxide Semiconductor
CPU	Central Processing Unit
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
DIP	Dual In-Line Package
DMP	Digital Motion Processor
DTP	Domino Transfer Protocol
DWD	Deutscher Wetterdienst
EAGLE	Einfach Anzuwendender Grafischer Layout Editor
FAT	File Allocation Table
FET	Feldeffekttransistor
GND	Ground (elektrisches Massepotential)
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
HAIL	Hagelabwehr-Server in der Luft
HASE	Hagelabwehr-Server auf der Erde
HWI	Hardware-Interrupt Service-Routine
I/O	Input Output
I^2C	Inter Integrated Circuit
IC	Integrated Circuit
IDE	Integrated Development Environment
IIR	Infinite Impulse Response
IP	Internet Protocol
ISR	Interrupt Service-Routine
LAN	Local Area Network
LCD	Liquid Crystal Display
LED	Light-Emitting Diode
LSB	Least Significant Bit
MAC	Media Access Control
MCU	Microcontroller Unit
MEMS	Micro Mechanical Electronical System
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor
MOSI	Master Out, Slave In
MPU	Motion Processing Unit

Abkürzung Definition

MSB	Most Significant Bit
NDK	Network Development Kit
OSI	Open System Interconnection
PAP	Programmablaufplan
RAM	Random Access Memory (Arbeitsspeicher)
RO-BERTA	Rosenheims meteorologische Besonderheiten: eine regelungstechnische Aufgabe
RTOS	Real-Time Operating System
SCK	Serial Clock
SCLK	Serial Clock
SD	Secure Digital
SDA	Serial Data
SDK	Softare Development Kit
SMD	Surface Mounted Device
SOMI	Slave Out, Master In
SPI	Serial Peripheral Interface
SSI	Synchronous Serial Interface
SWI	Software-Interrupt Service-Routine
$\overline{SS}$	Slave Select
TCP	Transmission Control Protocol
TI	Texas Instruments
TWI	Two-Wire Interface
UDP	User Datagram Protocol
USB	Universal Serial Bus
UTC	Universal Time Coordinated, koordinierte Weltzeit
WWW	World Wide Web

Formelzeichen Dimension Größe

a	$\left[\frac{m}{s^2} = 0,10197\,g\right]$	Beschleunigung
B	$[Gs = 10^{-4}\,T]$	Magnetische Flussdichte
f	$\left[Hz = \left[\frac{1}{s}\right]\right)$	Frequenz
I	$[A]$	Strom
Out	$[\,]$	Digitaler Wert in dezimaler Darstellung
P	$[bar = 10^5\,Pa]$	Absoluter Luftdruck
P	$[W]$	Leistung
φ	$[°]$	Winkel
RF	$[\%]$	Relative Feuchtigkeit
t	$[s]$	Zeit
T	$[s]$	Periodendauer
T	$[°C]$	Temperatur
V	$[V]$	Spannung

Index	Definition
B	Basis
BE	Basis-Emitter
C	Collector
F	Forward
Gier	Drehung um die vertikale Achse
HAILsens	Daten von HAILsens
max	Maximaler Wert
min	Minimaler Wert
Nick	Drehung um die Querachse
OH	Out HIGH
OL	Out LOW
P	Absoluter Luftdruck
rel	Relativ
RF	Relative Feuchtigkeit
Roll	Drehung um die Längsachse
S	Supply, Versorgung
Station	Daten der Wetterstation
Stell	Am Stellwinkel eingestellter Wert
T	Temperatur
XYZ	Repräsentation von Werten für drei Raumrichtungen

Notation	Erklärung
`functionXY()`	Name einer allgemeinen C-Funkion
`tskXFxnYZ()`	Bezeichner für eine SYS/BIOS Task-Funktion
`swiXFxnYZ()`	Bezeichner für eine SYS/BIOS Software-Interrupt Service-Routine
`clkXFxnYZ()`	Bezeichner für eine SYS/BIOS Clock-Funktion
`hwiXFxnYZ()`	Bezeichner für eine SYS/BIOS Hardware-Interrupt Service-Routine
`variable`	Variablenname
`value`	Wert einer Variable
`keyword`	Wert einer Variable
`code`	Allgemeines C Code-Statement
header.h	C-Headerdatei
source.c	C-Quelcodedatei
*.xy	Dateinamenserweiterung mit generischem Dateinamen (*)
C:\user\folder	Windows Dateipfad
0*xAFFE*	Zahl in hexadezimaler Darstellung

1 Hinführung zum Thema

In der Region rund um Rosenheim, sowie auch einigen angrenzenden Regionen Oberbayerns und Österreichs, entstehen durch Hagel-Unwetter immer wieder verheerende Schäden an Fahrzeugen, Häusern und Pflanzen. Aus diesem Grund gibt es hier eine über hundertjährige Geschichte diverser Ansätze zur Hagelabwehr. Das Grundprinzip dabei ist, Silberjodid in eine potentiell gefährliche Gewitterwolke einzubringen. Die Chemikalie soll Kristallisationskeime für das in der Wolke gefrierende Wasser bereit stellen. Erreicht wird dadurch die Entstehung vieler kleiner anstatt weniger großer Hagelkörner, die dann teilweise sogar schon auf ihrem weg zur Erde abschmelzen und somit ungefährlich sind.
In früheren Zeiten wurde das Silberjodid mit Kanonen von der Erde aus in den Himmel geschossen. Seit Mitte der 80er Jahre befinden sich zwei zweimotorige Partenavia P68 Flugzeuge im Einsatz, welche gezielt und effektiv Silberjodid in den Aufwindstrom einer Gewitterwolke einbringen können [5].

Seit dem Jahr 2007 arbeitet das Labor für Mess- und Regelungstechnik der Hochschule Rosenheim mit dem Verein zur Erforschung der Wirksamkeit der Hagelbekämpfung e.V. zusammen. Ziel der Zusammenarbeit ist eine Optimierung der Effizienz des Hagelabwehrflugs, u.a. durch die Erfassung und adäquate Aufbereitung von meteorologischen Daten, sowie Bewegungs-, Lage- und Positionsdaten im Flugzeug [5].

1.1 Das Projekt RO-BERTA

Das aktuelle Projekt RO-BERTA stellt eine Erweiterung des ursprünglichen Projekts RO-BERT dar. RO-BERTA bezweckt einerseits die interaktive Visualisierung von Wetterdaten, welche beispielsweise vom DWD[1] bereit gestellt werden, für den Piloten im Cockpit des Flugzeugs zur besseren Lokalisierung des Hagelzentrums. Andererseits werden die an bord gesammelten Daten für weitere empirische Studien zur Bodenstation HASE[2] übermittelt. Der Boden-Server schreibt die Informationen aus dem Flugzeug in eine Datenbank, von wo aus sie auch den Vereinsmittgliedern entsprechend grafisch aufbereitet über eine Website zur Verfügung stehen [6].

[1] DWD: Deutschen Wetterdienst
[2] HASE: Hagelabwehr-Server auf der Erde

1.2 Boardsystem HAIL

Das Boardsystem HAIL[3] sammelt und verarbeitet GPS-, Lage- und Bewegungsdaten, sowie meteorologische Messwerte in Echtzeit. Dabei erfasst das Subsystem HAILacquisition zusätzlich zu den unter Abschnitt 2.2 aufgeführten, von HAILsens übernommenen Messaufgaben noch den dynamischen Luftdruck, die GPS-Position mit zugehöriger UTC[4] und die in die beiden Brenner eingespritzte Menge Silberjodid. HAILscout stellt einerseits die digitale Funkverbindung für den Datenaustausch mit der Bodenstation HASE[5] zur Verfügung, andererseits werden aufgenommene Messwerte und Wetterdaten für die interaktive grafische Darstellung prozessiert. HAILpower schließlich ist für die elektrische Energieversorgung des Board-Systems HAIL zuständig [3].

Eine Übersicht über die Komponenten des Boardsystems ist in Abbildung 1.1 gegeben.

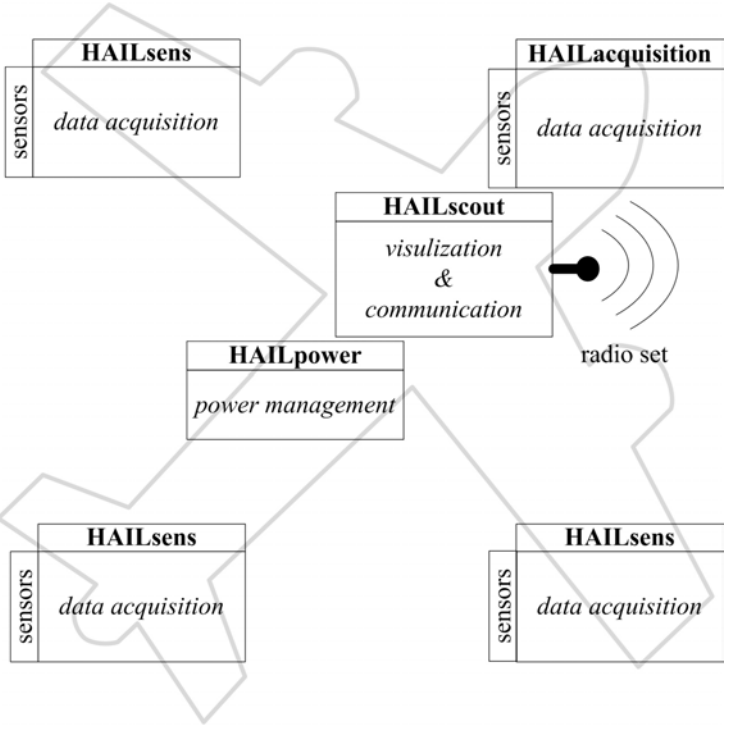


Abbildung 1.1: Boardsystem HAIL, Übersicht (Eigene Darstellung nach [3])

Die Kommunikation der Subsysteme HAILsens und HAILacquisition mit den jeweils angeschlossenen Sensoren läuft über den I²C-Bus[6]. Die aufgenommenen Messdaten werden dann in einem speziell entwickelten, dreischichtigen Übertragungsprotokoll über Ethernet und UDP[7] zur zentralen Verarbeitungseinheit HAILscout übermittelt [3].

[3] HAIL: Hagelabwehr-Server in der Luft
[4] UTC: Universal Time Coordinated, koordinierte Weltzeit
[5] HASE: Hagelabwehr-Server auf der Erde
[6] I²C: Inter Inegrated Circuit
[7] UDP: User Datagram Protocol

1.3 Aufgabenstellung: Prototypen-Entwicklung für das Subsystem HAILsens

Für die Aufnahme und Verarbeitung von Lage- und Bewegungsdaten, sowie meteorologischen Messwerten in Echtzeit soll das Boardsubsystem HAILsens sorgen. Herzstück des Systems ist der Microcontroller F28M35H52C1 aus der Concerto-Serie von Texas Instruments (TI). Warum genau diese MCU[8] gewählt wurde, wird im Abschnitt 2.1 erläutert. Abbildung 1.2 zeigt die Komponenten von HAILsens im Überblick.

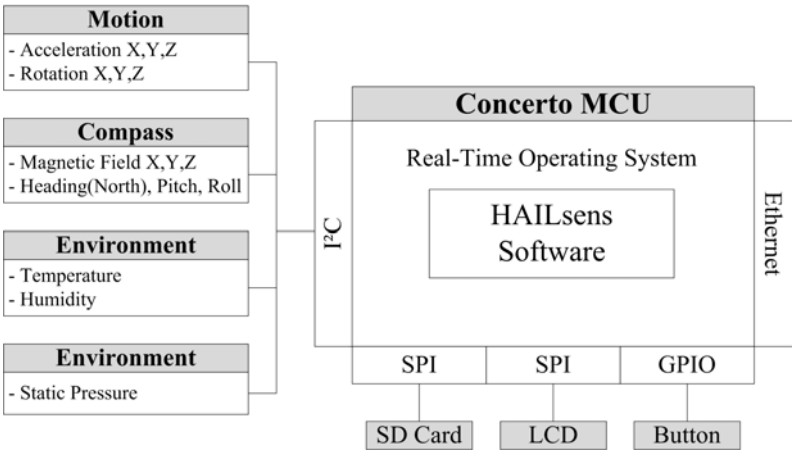


Abbildung 1.2: Blockschaltbild des Subsystems HAILsens (Eigene Darstellung nach [3])

Eine Hauptanforderung an HAILsens ist es, alle Messungen innerhalb exakt zeitdeterministischer Messzyklen durchzuführen. Aus diesem Grund liegt die Einbettung aller Funktionen zur Datenerfassung in ein Echtzeit-Betriebssystem nahe. Ausführungen dazu finden sich in den Abschnitten 4.3.1 und 5.4. Des weiteren macht die Minimalimplementierung eines User-Interfaces Sinn. So kann der Status der Sensoren leicht über einen Button und ein LCD[9] überprüft werden. Im Normalbetrieb, also währen des Hagelabwehrfluges ist diese Display allerdings nicht in Betrieb, da HAILsens in Tragflächen und Heck des Flugzeugs montiert wird und so dem Benutzer während des Einsatzes nicht zugänglich ist.

Neben der Echtzeitfähigkeit der Messroutinen ist die Herausforderung bei der Entwicklung des Systems HAILsens die Programmierung mit dem Hintergrund größtmöglicher Effizienz, sowohl hinsichtlich des Platzbedarfs für den Code, als auch in Bezug auf die Performanz des Programmablaufs. Diese Designkriterien sind aufgrund des begrenzten Speicherplatzes auf der MCU notwendig. Auch die Einbindung verschiedener Busverbindungen wie Ethernet, SPI und I^2C (siehe Kapitel 3) machen die Entwicklungs-Aufgabe interessant. Die lokale Speicherung der aufgenommenen Messdaten auf einer SD[10] Karte runden den Funktionsumfang des Systems HAILsens ab.

[8] MCU: Microcontroller Unit
[9] LCD: Liquid Crystal Display
[10] SD: Secure Digital

2 Hardwarekomponenten

Dieses Kapitel erläutert die Hauptelemente des Boardsubsytems HAILsens. Auf schaltungstechnische Elemente mit untergeordneter Funktion wie Vorwiderstände oder Potentiometer wird hier nicht näher eingegangen.

2.1 H52C1 Concerto controlCARD

Aus einer Vielzahl geeigneter MCUs verschiedener Anbieter fiel die Auswahlentscheidung zugunsten des 32 *bit* Dual-Core-Microcontrollers F28M35H52C1 aus der Concerto-Serie von Texas Instruments. Der Wunsch des Projektteams nach einer aktuellen und performanten Technologie, welche sich durch eine Vielzahl von Hardware-Schnittstellen wie Ethernet, I²C, SPI[1] oder CAN[2] auszeichnet und so auch für zukünftig geplante Anwendungen einsetzen lässt. Auch der auf Gleitkomma-Operationen spezialisierte C28x Rechenkern soll Raum für künftige Entwicklungen sichern. Für HAILsens wurde allerdings ausschließlich der Cortex M3-Kern eingebunden, da das System vornehmlich binäre Rohdaten prozessiert. Nur für die relativ seltenen Display-Ausgaben der Messungen sind die Rohdaten in Gleitkomma-Werte umzurechnen. Abbildung 2.1 zeigt die prinzipielle Beschaffenheit einer Concerto MCU. Für den F28M35H52C1 gelten jeweils die angegebenen Höchststwerte für die Taktfrequenzen der beiden Kerne und die Größe des Flash-Speichers der beiden Subsysteme (siehe auch Seite 71 im Anhang) [33].

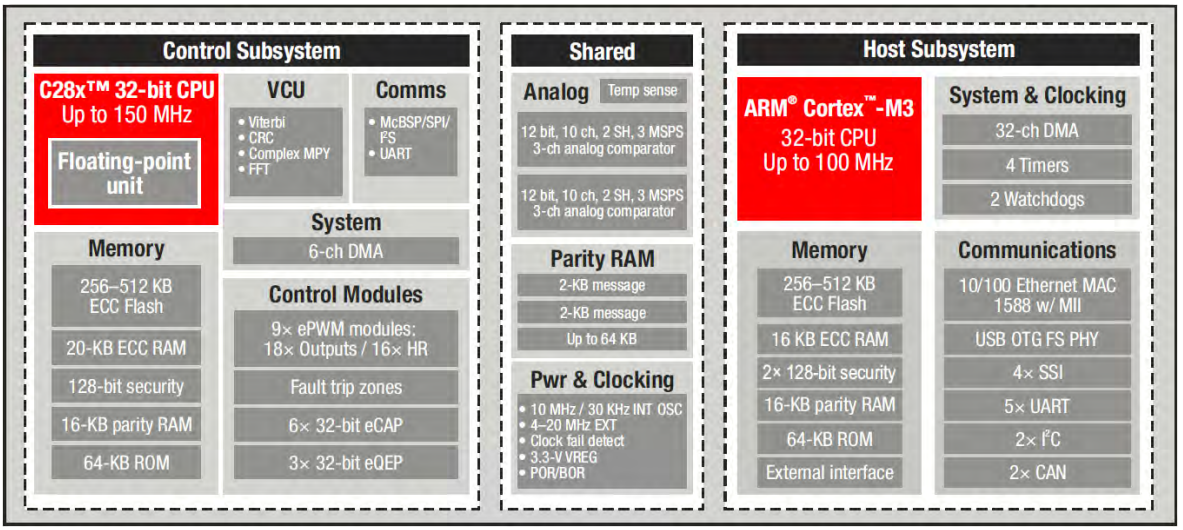


Abbildung 2.1: Blockdiagramm des Concerto Microcontrollers [34]

[1] SPI: Serial Peripheral Interface
[2] CAN: Controller Area Network

Als Entwicklungswerkzeug für den Controller bietet TI die relativ kostengünstige H52C1 Concerto controlCARD an. Die $90\,mm\,x\,70\,mm$ große controlCARD integriert einen F28M35H52C1 Microcontroller zusammen mit einem Micro-SD Kartenslot mit zugehörigem SPI-Controller, eine RJ-45 Buchse mit Ethernet-Controller, eine USB[3] Micro AB-Buchse über die das Board auch mit Energie versorgt werden kann und eine USB-ISO JTAG-Schnittstelle für Programmier- und Debug-Zwecke. [36]

In Kombination mit einer USB Docking-Station als H52C1 Experimenter Kit gibt es zusätzlich die Möglichkeit, $+5\,V$ und $+3,3\,V$ ohne gesonderte Spannungsquellen über Pins, die vom USB-Anschluss eines PCs gespeist werden, zu beziehen. Die unter den Abschnitten 2.2 und 2.3 aufgeführten externen Systemkomponenten können dadurch auf einfachem Weg mit Energie versorgt werden. Die signalführenden MCU-Pins sind auf die Kontakte einer Stiftleiste geführt und können auf diese Art einfach angeschlossen werden. Das Experimenter Kit ist in Abbildung 2.2 noch einmal grafisch beschrieben [37].

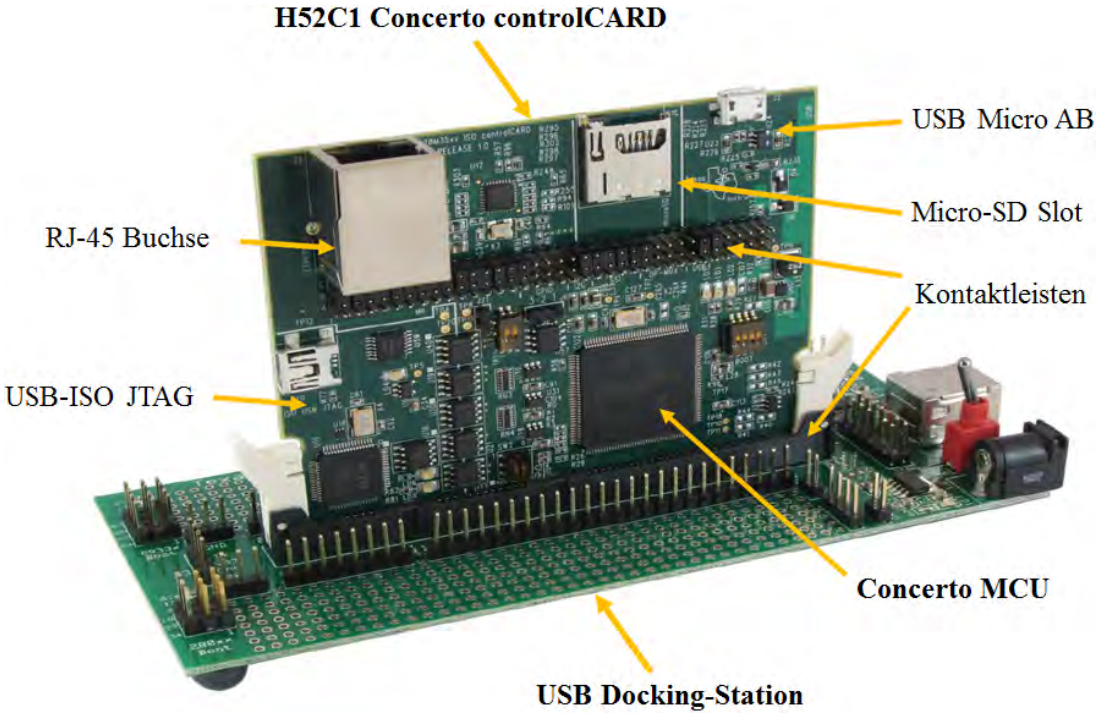


Abbildung 2.2: H52C1 Concerto Experimenter Kit (Eigene Darstellung nach [41])

Da eine eigene, komplexe Schaltungsentwicklung unter fachgerechter Integration aller notwendigen Schnittstellentreiber, sowie das damit verbundene Platinenlayout durch die Verwendung der Control Card vorerst entfällt, kann eine nicht unerhebliche Menge an Entwicklungszeit eingespart werden. Die für das Subsystem HAILsens benutzten Concerto IO-Pins zum Anschluss der Sensoren, des LCD-Moduls, des Tasters zum Umschalten Sensoranzeigen, des Ethernet-Transceivers und des Micro SD Speicherkartenslots sind in Abbildung A.3.1 im Anhang auf der Seite 70 zusammengefasst.

[3]USB: Universal Sirial Bus

2.2 Sensoren

Eine Vorauswahl der verwendeten ICs[4] wurde bereits von Dipl.-Ing.(FH) Mathias Frey im Rahmen seiner Diplomarbeit mit dem Thema “Integration einer Sensorik zur Messung von Umweltparametern in eine Flugzeugnase“ getroffen (vgl. [1]). Dort sind auch Ausführungen zu den physikalischen Messprinzipien der einzelnen Sensoren zu finden. Abweichend von dieser Auswahl ist in der vorliegenden Masterarbeit die Bauform des in Abschnitt 2.2.1 beschriebenen Drucksensors, sowie der Feuchte- und Temperatursensor (Abschnitt 2.2.2) aufgrund schlechter Erfahrungen mit dem I²C-Interfacewurde des ursprünglich von Herrn Frey bestimmten ICs.

Alle Sensoren verfügen über eine I²C-Schnittstelle für deren Parametrierung und die Messdatenabfrage. Der Abfrageablauf variiert jedoch von Sensor zu Sensor teilweise erheblich. Lediglich beim Bewegungssensor und beim elektronischen Kompass sind die Abfragesequenzen ähnlich, weshalb hier die selben C-Funktionen für beide Sensor-ICs benutzt werden können (vgl. Abschnitte 2.2.3, 2.2.4 und 5.2.2). Einzelheiten zur konkreten I²C-Kommunikation finden sich in den Abschnitten 3.1.3 und 5.2.2. Die Reihenfolge, in der die Sensoren hier aufgeführt sind, korrespondiert mit der der zeitlichen Abfolge ihrer Integration in das Subsystem HAILsens, sowie mit einer steigenden Komplexität der ICs in puncto Parametrierung und Datenkommunikation.

2.2.1 Drucksensor HDI0611ARZ8P5

Abbildung 2.3: Drucksensor HDI [26]

Der Messung des absoluten Luftdrucks dient eine Gerätevariante aus der HDI-Serie von Sensortechnics. Die Erweiterung 0611ARZ8P5 in der Bezeichnung bezieht sich auf den Messbereich (0611 ≙ $600\,mbar$ ... $1100\,mbar$), den Messmodus (A ≙ absoluter Druck), die Kalibrierung (R ≙ barometrisch), die Bauform (Z ≙ DIP[5]-Gehäuse, axiale Messöffnung), die Portierung (8 ≙ Straight), die Qualität (P ≙ Prime Grade) und die Versorgungsspannung (5 ≙ $V_S = +5\,V$). Das Gerät liefert die aktuelle Messung über ein analoges Spannungssignal, sowie zusätzlich in Form einer $15\,bit$-Information. Der digitale Wert Out_P wird alle $250\,\mu s$ im I²C-Ausgangsregister des ICs upgedated. Die untere Messbereichsgrenze $P_{min} = 600\,mbar$ entspricht einem digitalen Wert von $Out_{P_{min}} = 3277$, die Messbereichsobergrenze $P_{max} = 1100\,mbar$ einem Wert von $Out_{P_{max}} = 29490$. Die Genauigkeit des ausgegebenen Messwerts liegt bei einer Umgebungstemperatur von $0\,°C$ bis $85\,°C$ bei $\pm 1\,\%$ des Skalenbereichs. Da der HDI intern mit einem $12\,bit$-ADC[6] arbeitet, liegt die Auflösung des Sensors nach Datenblattangaben bei maximal 11 bis $12\,bit$. Für nähere Angaben zur Auflösung verweist die Dokumentation zum Drucksensor auf den Support von Sensortechnics [26, 27].

Aus diesen Daten und mit dem Zusammenhang $1\,mbar = 1\,hPa$ lässt sich eine Formel für die Umrechnung des digitalen Werts Out_P in den physikalischen Luftdruck P angeben:

[4] IC: Integrated Circuit
[5] DIP: Dual In-Line Package
[6] ADC: Analog-to-Digital Converter

$$P = \frac{P_{max} - P_{min}}{Out_{P_{max}} - Out_{P_{min}}} \cdot (Out_P - Out_{P_{min}}) + P_{min} \pm 0,01 \cdot (P_{max} - P_{min}) = \ldots$$
$$= 0,01907\,hPa \cdot Out_P - 537,50\,hPa \pm 5,0\,hPa \quad (2.1)$$

Die Berechnung des Druckwerts nach Gleichung 2.1 ist für die Anzeige auf dem LCD-Modul in der C-Funktion `HDI0611CalcPressure()` implementiert (vgl. Abschnitt 5.2.2). Die Messunsicherheit von $\pm 5\,mbar$ bleibt dabei unberücksichtigt.

Der Sensor benötigt eine Versorgungsspannung von $V_S = +5\,V$ bei einem Strom von $I_S = 5\,mA$. Der logische HIGH-Pegel der I²C-Busschnittstelle liegt ebenfalls bei $+5\,V$. Um diesen Pegel auf die $+3,3\,V$ der übrigen Busteilnehmer anpassen zu können, wird der in Abschnitt 2.3.3 beschriebene Level-Converter verwendet. Für die Beschaffung einer HDI-Variante für $V_S = +3,3\,V$ musste zum Zeitpunkt der Bestellung mit unverhältnismäßig langen Lieferzeiten gerechnet werden. Um den Drucksensor zeitnah in das Subsystem HAILsens zu integrieren wird deshalb das oben beschriebene Harware-Setup mit $V_S = +5\,V$ und Level-Converter gewählt [26, 27, 46].

2.2.2 Feuchte- und Temperatursensor SHT21

Mit Abmessungen von nur $3,0\,mm\,x\,3,0\,mm\,x\,1,1\,mm$ bringt der SMD[7]-Sensor SHT21 ebenfalls eine voll funktionsfähige, bidirektionale Bus-Schnittstelle mit. Über dieses I²C-Interface kann der IC u.a. parametriert werden. Beispielsweise ist die Auflösung der vom Sensor gelieferten digitalen Werte für relative Luftfeuchtigkeit und Temperatur einstellbar. Für die Messung beider physikalischer Größen wurde jeweils die höchste verfügbare Auflösung gewählt, was eine relativ lange Messdauer für einen einzelnen Messwert zur Folge hat [25].

Abbildung 2.4: SHT21 [25]

Relative Luftfeuchtigkeit

Gemessen wird die Luftfeuchtigkeit mit einer Auflösung von $12\,bit$ (bzw. $0,04\,\%$ relative Feuchte), wodurch sich eine maximale zu erwartende Messdauer von $29\,ms$ ergibt. Der Messbereich des SHT21 erstreckt sich von $0\,\%$ bis $100\,\%$ relativer Feuchte mit einer Genauigkeit von $\pm 2,0\,\%$ vom Skalenendwert. Aus dem digitalen Wert Out_{RF} berechnet sich der physikalische Wert RF für die relative Feuchtigkeit in Prozent nach [25] zu

$$RF = (-6.0 + \frac{125.0}{2^{16}} \cdot Out_{RF} \pm 2,0)\,\%$$
$$= (-6.0 + 0,0019074 \cdot Out_{RF} \pm 2,0)\,\% \ . \quad (2.2)$$

[7] SMD: Surface Mounted Device

Temperatur

Gewählt wurde für den Temperaturwert die Auflösung von $14\,bit$ (bzw. $0{,}01\,°C$), wobei mit einer Messdauer von bis zu $85\,ms$ zu rechnen ist. Im Messbereich von $-40\,°C$ bis $+125\,°C$ beträgt die Genauigkeit typischerweise $\pm 0.3\,°C$. Der physikalische Temperaturwert T ergibt sich demnach laut [25] aus dem digitalen Dezimalwert Out_T durch folgenden mathematischen Ausdruck:

$$\begin{aligned} T &= \left(-46{,}85 + \frac{175{,}72}{2^{16}} \cdot Out_T \pm 0{,}3\right) °C \\ &= \left(-46{,}85 + 0{,}0026813 \cdot Out_T \pm 0{,}3\right) °C \end{aligned} \tag{2.3}$$

Die Berechnung der physikalischen Werte aus den Gleichungen 2.2 und 2.3 ist für deren Display-Ausgabe in den C-Funktionen `SHT21CalcHumidity()` und `SHT21CalcTemperature()` implementiert (vgl. Abschnitt 5.2.2), jedoch ebenfalls wieder ohne die in den Ausdrücken vorkommende Messunsicherheit. Der Vollständigkeit halber sei noch erwähnt, dass die Genauigkeit beider Messwerte schwach temperaturabhängig ist. Die Umweltbedingungen während eines Hagelabwehr-Fluges befinden sich in der Regel jedoch in einem Rahmen, in dem dieser Einfluss vernachlässigt werden kann [25].

Versorgt wird der SHT21 mit $V_S = +3{,}3\,V$. Der HIGH-Pegel für den I²C-Bus ist ebenfalls auf diesem Spannungsniveau. Die äußerst geringe Stromaufnahme von $I_S = 300\,\mu A$ garantiert einen energieverbrauchsarmen Betrieb des ICs. Wichtig ist, dass nach dem Einschalten der Spannungversorgung mindestens $15\,ms$ gewartet wird, um die vollständige Initialisierung des SHT21 zu gewährleisten [25].

2.2.3 Elektronischer Kompass HMC6342

In MEMS[8]-Ausführung findet man beim HMC6343 integrierte Sensorik für die Ausrichtung zum magnetischen Nordpol der Erde, sowie für die Neigung auf zwei Achsen und Magnetfeldmessung in drei Raumrichtungen. Ebenfalls auf dem $9{,}0\,mm\,x\,9{,}0\,mm\,x\,1{,}9\,mm$ großen Chip befindet sich ein Microprozessor mit Algorithmen zur Neigungskorrektur, wodurch der HMC6343 auch in einer von der Horizontalen abweichenden Lage Kompasswerte mit relativ hoher Genauigkeit erzeugt [28]. Gewählt wurde eine Liefervariante des ICs auf dem so genannten Breakout Board (vgl. Abbildung 2.5). Dies erleichtert die elektrische Kontaktierung des SMD-Bausteins mit Versorgungsspannung und I²-Bussignalen. Es entfällt dadurch die Notwendigkeit der

Abbildung 2.5: Kompassmodul HMC6343

[8] MEMS: Micro Mechanical Electronical System

Erstellung eines eigenen Platinenlayouts, was wieder wertvolle Entwicklungszeit einspart.

Bereitgestellt werden alle Messwerte in Form eines $16\,bit$ Zweierkomplements. Für die Umrechnung des Zweierkomplements in eine dezimale Darstellung muss zuerst das höchstwertige Bit (MSB[9]) betrachtet werden, welches der Indikator für das Vorzeichen der Dezimalzahl ist. Falls gilt $MSB = 0$, handelt es sich um eine positive Zahl, welche direkt aus dem Binärsystem in das Dezimalsystem umgerechnet werden kann. Wenn $MSB = 1$ ist, wird zuerst 1 subtrahiert und danach alle Bits invertiert. Erst dann kann bei gesetztem Vorzeichen-Bit in die dezimale Zahlendarstellung umgewandelt werden. Falls ein vorzeichenbehafteter C-Datentyp für eine Zahl deklariert wurde, interpretiert der C-Compiler allerdings das zugehörige Bitmuster standardmäßig als Zweierkomplements-Darstellung und eine explizite Umwandlung muss nicht implementiert werden [4, 24]. Es sind einstellbare Update-Raten der Messwerte von $1\,Hz$, $5\,Hz$ oder $10\,Hz$ wählbar. HAILsens übernimmt, ohne spezielle Konfiguration dieser Rate, den Default-Wert von $5\,Hz$, da dieser in idealer Weise zur Messaufgabe passt (vgl. Abschnitt 5.4).

Ausrichtung zum magnetischen Nordpol: Gierwinkel

Abbildung 2.6 zeigt die Orientierung der Bezugsachsen des HMC6343. Die X-Achse (roter Pfeil) zeigt in die Vorwärts-Bewegungsrichtung des Hagelflugzeugs, der gebogene schwarze Pfeil deutet eine Abweichung vom der Nord-Ausrichtung an. Diesen Drehwinkel um die Z-Achse, in der Avionik als Gierwinkel bezeichnet, liefert der Sensor in einem Wertebereich von $0°$ bis $360°$, wobei $0°$ bedeutet, dass die X-Achse des ICs genau in Richtung des magnetischen Nordpols zeigt [28].

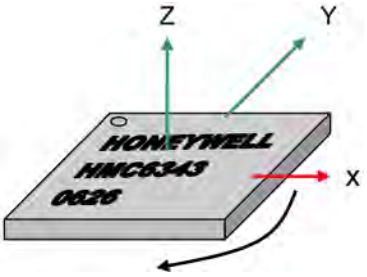


Abbildung 2.6: Dreiachsen-System des HMC6343 [28]

Die in Form eines Zweierkomplements ausgelesenen Rohdaten Out_{Gier} im Bereich von $Out_{Gier_{min}} = 0$ bis $Out_{Gier_{max}} = 3600$ repräsentieren die Nord-Orientierung als Ganzzahlwert in Zehntelgrad. Die Auflösung der Kompass-Information liegt demnach bei $0,1°$. Sind X- und Y-Achse horizontale ausgerichtet, ist der ausgegebene Kompasswert mit einer Genauigkeit von $\pm 2,0°$, bei einer Neigung bis $\pm 15°$ mit $\pm 3,0°$ und bei einer Neigung bis $\pm 60°$ mit $\pm 4,0°$ spezifiziert. Für Display-Ausgaben des Gierwinkels φ_{Gier} müssen die Rohdaten Out_{Gier} durch den Faktor 10 geteilt werden [28]:

[9] MSB: Most Significant Bit

$$\varphi_{Gier} = (\frac{Out_{Gier}}{10,0} \pm 3,0)^\circ \tag{2.4}$$

Die in dieser Umrechnungsformel angegebene Messunsicherheit ist als Mittelwert der für die verschiedenen Neigungen spezifizierten Werte zu sehen. Gleichung 2.4 findet strukturell gleich auch für die Berechnung der Werte von Nickwinkel und Rollwinkel aus den Rohdaten Out_{Nick} und Out_{Roll} Verwendung (Gleichungen 2.5 und 2.6). Die C-Funktion `TwosComplement2Float()` zur Umwandlung eines Ganzzahlwerts in eine Gleitkommazahl kann somit mehrfach benutzt werden (siehe auch 5.2.2).

Es ist die Anwendung eines im HMC6343 integrierten, auf Werte zwischen 0 und 15 konfigurierbaren IIR[10]-Filters auf die ausgegebenen Werte für Out_{Gier} möglich. Dadurch würde der Messwerteverlauf geglättet, was bei schnellen Bewegungen Sinn machen kann. Da die Partenavia sich für eine Update-Rate von $5Hz$ relativ träge bewegt, wird die Default-Einstellung von 0 beibehalten, also dieser IIR-Filter nicht angewendet [28].

Nickwinkel und Rollwinkel

Ebenfalls mit einer Auflösung von $0,1^\circ$ erzeugt der HMC6343 die Rohdaten der Drehwinkel-Abweichungen von einer horizontalen Lage der X- und Y-Achse. Diese Winkel werden in der Avionik, wie auch in den folgenden Ausführungen, mit Nickwinkel (Drehung um die Querachse Y) und Rollwinkel (Drehung um die Längsachse X) bezeichnet (vgl. Abbildung 2.6). Der Messbereich dieser Winkel ist im Datenblatt des Sensors mit $\pm 80^\circ$ angegeben. Die Genauigkeit der Messwerte variiert mit der Abweichung von der waagrechten Lage ($\hat{=} 0^\circ$). Zwischen 0° und 15° ist sie für beide Drehachsen mit $\pm 1^\circ$, zwischen 15° und 60° mit $\pm 2^\circ$ angegeben [28].

Der Wertebereich von Out_{Nick} und Out_{Roll} liegt jeweils zwischen ±900 und entspricht einer Winkelangabe in Zehntelgrad. Aus diesen Rohdaten werden die Werte für Ausgaben des Nickwinkels φ_{Nick} und des Rollwinkels φ_{Roll} am LCD-Modul mit den Formeln

$$\varphi_{Nick} = (\frac{Out_{Nick}}{10,0} \pm 1,5)^\circ \tag{2.5}$$

und

$$\varphi_{Roll} = (\frac{Out_{Roll}}{10,0} \pm 1,5)^\circ \tag{2.6}$$

berechnet, allerdings wieder ohne die angegebene mittlere Messunsicherheit. Wie oben bereits erwähnt, dient dieser Umrechnung die C-Funktion `TwosComplement2Float()`.

[10] IIR: Infinite Impulse Response

Magnetfeld

Der Messsbereich des Magnetometers ist im Datenblatt des HMC6343 mit typischerweise $\pm 1\,Gs$ und maximal $\pm 2\,Gs$ angegeben. Zu Auflösung und Genauigkeit findet man hier keine weiteren Spezifikationen. Der HMC6343 wird als Kompassmodul vertrieben. Die Magnetfeldmessung wird vom Produzenten als Hilfsangabe angesehen und ist deshalb nicht im Detail spezifiziert. Auf Anfrage hat Honeywell die Information heraus gegeben, dass im HMC6343 drei Einachs-Magnetometer vom Typ HMC1041Z verbaut sind. Die weiteren Angaben beziehen sich deshalb auf die in dessen Datenblatt angegebenen Werte. Es ist dort eine Auflösung von $0,16\,mGs$ aufgeführt. Die Genauigkeit ist nicht explizit angegeben, lediglich ein vom Messbereich abhängiger Linearitätsfehler, welcher für $\pm 1\,Gs$ bei $0,17\,\%$ des Vollausschlags liegt [28, 29].

Mit dem Wissen aus [12], dass das Erdmagnetfeld in Mitteleuropa in horizontaler Richtung eine Stärke von etwa $20\,\mu T$ und in vertikaler Richtung von etwa $44\,\mu T$ hat, konnte durch Versuchsreihen, mit Hilfe der Angaben aus beiden Datenblättern und mit dem Zusammenhang $1\,Ga = 10^{-4}\,T = 100\,\mu T$ folgende Gleichung für die Berechnung der magnetischen Flussdichte B_{XYZ} in den drei Richtungen eines rechtwinkligen Koordinatensystems aus den vom Sensor gelieferten binären Rohdaten $Out_{B_{XYZ}}$ abgeleitet werden:

$$B_{XYZ} = \left(\frac{Out_{B_{XYZ}}}{100,0} \pm 0,17\right)\mu T \tag{2.7}$$

Dieser mathematische Zusammenhang beinhaltet jedoch, wie oben angedeutet, keine absolut exakte Beschreibung des Magnetometer-Verhaltens. Der Faktor $100,0$ im Nenner wurde anhand der vorher genannten Annahmen ermittelt. Wie die Ausführungen in Abschnitt 6.1.2 zeigen, treffen diese Annahmen jedoch zumindest für magnetische Flussdichten in der Größenordnung des Erdmagnetfelds zu. Es ist deshalb davon auszugehen, dass die $16\,bit$-Werte für jeweils eine Raumrichtung die Stärke der magnetischen Flussdichte in Hundertstel μT repräsentiert. Gleichung 2.7 kann somit für Ausgaben der Messwerte an einem Display ebenfalls mit der C-Funktion `TwosComplement2Float()` ohne die Messunsicherheitsangabe implementiert werden (vgl. 5.2.2).

Der Sensor ist in horizontaler Lage montiert. Durch entsprechende Konfiguration über die I²C-Schnittstelle kann prinzipiell auch eine Positionierung gewählt werden, bei welcher die X-Achse oder die Y-Achse vertikal ausgerichtet sind. Für eine kompaktere Integration aller vom Subsystem HAILsens verwendeten Sensoren auf eine einzige Platine kann eine derartige Einstellung am IC für kommende Entwicklungsschritte notwendig werden [28].

Versorgt wird der HMC6343 mit $V_S = +3,3\,V$, was auch den I²C-HIGH-Level auf diesen Wert fest legt. Die Stromaufnahme im Normalbetrieb beträgt $I_S = 4,5\,mA$. Nach Aktivierung der Spannungsversorgung benötigt der Sensor-IC eine Startup-Zeit von $500\,ms$ für die vollständige Initialisierung aller Systemkomponenten [28].

2.2.4 Bewegungssensor MPU-6050

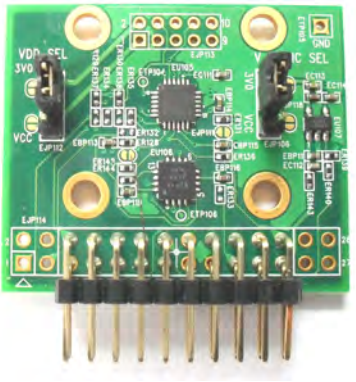


Abbildung 2.7: Evaluation Board mit MPU-6050

Für die Bestimmung von Beschleunigung und Drehraten in drei Raumrichtungen ist die Verarbeitungseinheit MPU[11]-6050 im Einsatz. Zur schnellen Integration in das Subsystem HAILsens wegen der einfachen elektrischen Kontaktierungsmöglichkeiten, ist die in Abbildung 2.7 gezeigte Variante des Chips auf dem Evaluation Board MPU-6050EVB eingekauft worden. Der MPU-6050 selbst ist der obere der beiden Chips in der Mitte der Abbildung des Evaluation Boards. Die Messeinrichtungen des ICs sind als eingebettetes dreiachs MEMS-Gyroskop und MEMS-Beschleunigungsaufnehmer implementiert. Die insgesamt sechs Messwerte werden von integrierten ADCs digitalisiert und sind so als $16\,bit$-aufgelöste Werte in Zweierkomplements-Darstellung über den I^{2}C-Bus auslesbar. Des weiteren verfügt die MPU über eine DMP[12] Hardwarebeschleunigungs-Engine, welche für die recht kompakten Bauteilabmessungen des Sensor-ICs von $4,0\,mm\,x\,4,0\,mm\,x\,0,9\,mm$ eine Fülle von Funktionalitäten, wie beispielweise einen speziellen Interrupt bei freiem Fall oder die Möglichkeit der Anbindung externer Zusatzsensorik, bietet. Der DMP entlastet zusätzlich durch Vorverarbeitung der Messdaten den Host-Processor, in diesem Fall den Concerto Microcontroller [30].

Am Beschleunigungssensor (Akzelerometer) können Messbereiche von $\pm 2\,g$, $\pm 4\,g$, $\pm 8\,g$ oder $\pm 16\,g$, am Gyroskop Messbereiche von $\pm 250\,\frac{°}{s}$, $\pm 500\,\frac{°}{s}$, $\pm 1000\,\frac{°}{s}$ oder $\pm 2000\,\frac{°}{s}$ eingestellt werden. Je weiter man den Messbereich wählt, desto geringer ist die Empfindlichkeit und daraus Abgeleitet auch die Genauigkeit des ausgegebenen Messwerts (siehe Gleichung 2.8). Es wurden aus diesem Grund nach Absprache mit dem Piloten der P68 die kleinstmöglichen Einstellungen für die Weite der Messbereiche verwendet, um bei praxistauglichen Ober- und Untergrenzen größtmögliche Genauigkeit der Messungen zu erhalten. Um den Einfluss der durch den Flugzeug-Motor hervorgerufenen Vibrationen auf die Messwerte zu verringern, kann ein digitaler Tiefpassfilter mit Grenzfrequenzen zwischen $5\,Hz$ und $256\,Hz$ programmiert werden. HAILsens verwendet eine Grenzfrequenz von $5\,Hz$, was die starken Fluktuationen der am Display angezeigten Werte für Beschleunigung und Drehraten weitgehend eliminiert (vgl. Kapitel 6.1.3) [30].

Akzelerometer

Die die in HAILsens implementierte Beschleunigungsmessung verwendet einen Messbereich von $\pm 4\,g$, woraus nach [30] die Empfindlichkeit $8192\,LSB/g$[13] resultiert. Die Auflösung von $16\,bit$ entspricht für diese Einstellung demnach

[11] MPU: Motion Processing Unit
[12] DMP: Digital Motion Prozessor
[13] LSB: Least Significant Bit

$$\pm\frac{1}{2}LSB \;\hat{=}\; \pm\frac{1}{2}LSB\cdot\frac{1}{8192}g/LSB \;=\; \pm\frac{1}{16384}g \;=\; \pm 61{,}04\cdot10^{-6}g\ . \tag{2.8}$$

Im Datenblatt [30] sind noch die weiteren Faktoren Temperaturdrift ($\pm 0{,}02\,\%/°C$), Cross-Axis Sensitivity ($\pm 2\,\%$), eine Nichtlinearität von $\pm 0{,}5\,\%$ und die anfängliche Kalibrierungstoleranz von $\pm 3\,\%$ angegeben, welche die Mesunsicherheit zusätzlich vergrößern. Ob sich diese Angaben auf den Messwert, den Skalenendwert oder den Skalierungsfaktor beziehen, geht aus dem Dokument nicht hervor. Die in Gleichung 2.9 verwendeten $\pm 1{,}5\,\%$ (Faktor $0{,}015$) sind als aus Erfahrungswerten abgeleitete Abschätzung für eine obere Schranke der im Datenblatt aufgeführten Toleranzen, bezogen auf den Skalenendwert zu sehen. Aus dem digitalen Ausgabewehrt $Out_{a_{XYZ}}$ lässt sich der Wert für die physikalische Beschleunigung a_{XYZ} in $\frac{m}{s^2}$ mit dem Zusammenhang $1\,g = 9{,}80665\,\frac{m}{s^2}$ [11] dann ausdrücken durch die Formel

$$\begin{aligned} a_{XYZ} &= \left(\frac{Out_{a_{XYZ}}}{8192} \pm \frac{1}{16384} \pm 0{,}015\cdot 4\right)\cdot 9{,}80665\,\frac{m}{s^2} \\ &= \left(\frac{Out_{a_{XYZ}}}{835.352} \pm 0{,}589\right)\frac{m}{s^2}\ . \end{aligned} \tag{2.9}$$

Diese Formel kann für Ausgaben der Messwerte über das LCD-Modul wieder mit der C-Funkion `TwosComplement2Float()` umgesetzt werden, da diese Funktion einen Divisor (Nenner 835.352 in Gleichung 2.9) als Parameter hat (vgl. 5.2.2). Die abgeschätzte Messunsicherheit bleibt für die Displayausgaben unberücksichtigt. Es zeigt sich, dass bei der Anzeige der gerundet angegebenen Erdbeschleunigung von $9{,}81\,\frac{m}{s^2}$ die Werte auf die zweite Nachkommastelle genau dargestellt werden, was für Projekt RO-BERTA ausreichend ist (vgl. Abschnitt 6.1.3). Die Auswertung der während eines Hagelabwehr-Fluges aufgenommenen Messreihen wird zeigen, ob eine genauere Betrachtung der Spezifikationen notwendig ist [30].

Gyroskop

Die Drehratemessung findet in einem Bereich von $\pm 250\,\frac{°}{s}$ statt, was für die relativ trägen Drehbewegungen der Partenavia ausreicht. Durch diese Einstellung ergibt sich ein Skalierungsfaktor von $131\,LSB/\frac{°}{s}$, welcher mit einer Toleranz von $\pm 3\,\%$ bei $25\,°C$ spezifiziert ist und zusätzlich einen Temperaturdrift von $\pm 2\,\%$ hat. Es ergibt sich also eine Gesamttoleranz der Empfindlichkeit von $\pm 5\,\%$.Die weiteren Angaben einer Nichtlinearität von $0.2\,\%$ und einer eine Cross-Axis Sensitivity von $\pm 2\,\%$ aus dem Datenblatt sind, wie beim Akzelerometer, wieder schwer in Beziehung zu setzen und werden deswegen hier nicht weiter berücksichtigt. Die aus Gleichung 2.10 für die Berechnung der Drehrate ω_{XYZ} aus den digitalen Rohdaten $Out_{\omega_{XYZ}}$ resultierende Messunsicherheit muss deshalb als Mindestwert betrachtet werden [30].

$$\omega_{XYZ} = \left(\frac{Out_{\omega_{XYZ}}}{131 \pm 0{,}05 \cdot 131}\right)\frac{^\circ}{s} \tag{2.10}$$

Diese Umrechnungsformel ist ebenfalls mit der Funktion `TwosComplement2Float()` im C-Programm implementierbar. Die am Display angezeigten physikalischen Werte für die Drehraten um drei Raumachsen lässt die Messunsicherheit außer acht.

Der MPU-6050 selbst wird an $V_S = 3{,}3\,V$ bei einer je nach Betriebsart variierenden Stromaufnahme von etwa $V_S = 3{,}9\,mA$ betrieben. Dieser befindet sich jedoch mit einiger zusätzlicher Elektronik auf dem Evaluation Board. Das verwendete Evaluation Board benötigt eine Versorgungsspannung von $V_S = 5\,V$, die Stromaufnahme ist in dessen Datenblatt nicht angegeben. Die mit einem Fluke Multimeter gemessene Stromaufnahme des MPU-6050EVB im Nennbetrieb beträgt $I_S = 3{,}7\,mA$. Der I^2C-HIGH-Pegel ist für bei Hardware-Konfigurationen mit $3{,}3\,V$ angegeben, was eine Problemlose Systemintegration in HAILsens ermöglicht. Eine Startup-Time von maximal $100\,ms$ nach Einschalten der Spannungsversorgung gewährleistet die vollständige Initialisierung aller IC-Komponenten [30].

2.2.5 Überblick Sensoren

Im folgenden sind die in den Abschnitten 2.2.1 bis 2.2.4 ausgeführten technischen Daten der verwendeten Sensorik noch einmal zusammengefasst.

Messtechnische Spezifikationen

Tabelle 2.1 gibt einen Überblick über die Messspezifikationen der von den ICs ausgegebenen und umgerechneten physikalischen Größen.

Messgröße	Messbereich	Empfindlichkeit	Genauigkeit
Luftdruck	600 ... 1100 hPa	0,01907 hPa/LSB	±5,0 hPa
Luftfeuchte	0 ... 100 %	0,0019074 $\%/LSB$	±2,0 %
Temperatur	−40 ... 125 °C	0,0026813 °C/LSB	±0,3 °C
Gier	0 ... 360 °	0,1 °$/LSB$	±3,0 °
Nick, Roll	−80 ... +80 °	0,1 °$/LSB$	±1,5 °
Mag. Flussdichte	−100 ... +100 μT	0,01 $\mu T/LSB$	$\gtrsim$ ±0,17 μT
Beschleunigung	−39,23 ... +−39,23 $\frac{m}{s^2}$	0,001197 $\frac{m}{s^2}/LSB$	$\lesssim$ ±0,589 $\frac{m}{s^2}$
Drehraten	−250 ... +250 $\frac{^\circ}{s}$	0,007634 $\frac{^\circ}{s}/LSB$	$\lesssim$ ±5 % vom Messwert

Tabelle 2.1: Spezifikationen der verwendeten Sensorik [26, 25, 28, 29, 30]

Elektrischer Anschluss

Abbildung 2.8 zeigt die elektrischen Anschlüsse der ICs in Form eines mit der Software EAGLE[14] erstellten Schaltplans. Zu sehen sind Steckbuchsen (CON1 bis CON4) für die Verbindung der Sensoren mit Versorgungsspannung und Bussignalen (SDA[15] und SDC[16]). Die Beschaffenheit des I^2-Busses wird in Abschnitt 3 beschrieben. Der Level-Converter für die Angleichung des vom Drucksensor verwendeten HIGH-Pegels wird in Abschnitt 2.3.3 beschrieben.

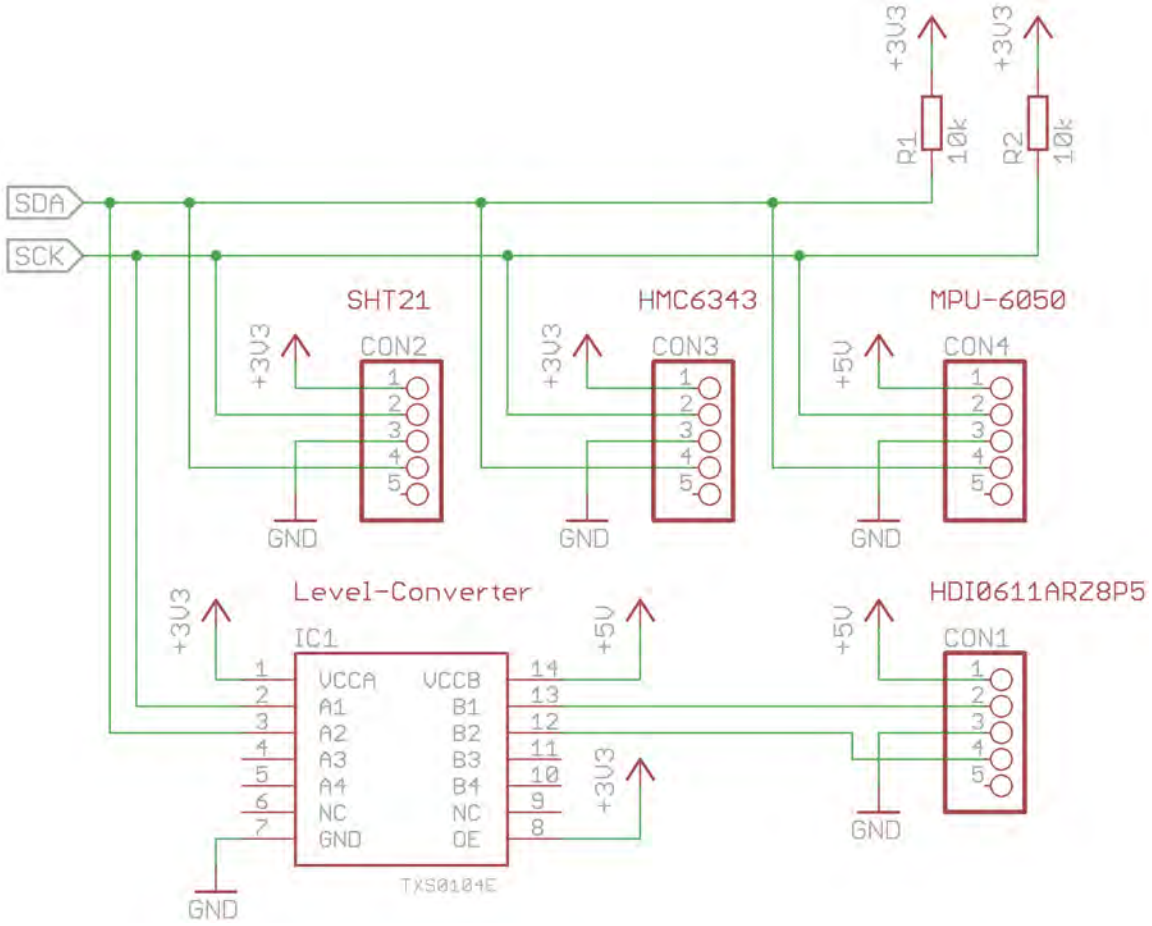


Abbildung 2.8: Anschluss der Sensor-ICs an Spannungsversorgung und I^2-Bus

Tabelle 2.2 listet die jeweils am Ende der Abschnitte 2.2.1 bis 2.2.4 erwähnten elektrischen Anschlussdaten auf.

[14] Einfach Anzuwendender Grafischer Layout Editor
[15] SDA: Serial Data
[16] SDC: Serial Clock

Sensor	Versorgungsspannung	Stromaufnahme	I^2 HIGH-Pegel
HDI0611ARZ8P5	$+5\,V$	$5\,mA$	$+5\,V$
SHT21	$+3,3\,V$	$0,3\,mA$	$+3,3\,V$
HMC6342	$+3,3\,V$	$4,5\,mA$	$+3,3\,V$
MPU6050EVB	$+5\,V$	$3,7\,mA$	$+3,3\,V$

Tabelle 2.2: Elektrische Anschlussdaten der Sensoren [26, 25, 28, 29, 30]

2.3 Sonstige Bauteile

Neben einigen Bauelementen mit untergeordneter Bedeutung, wie beispielsweise ein Taster, Verbindungsleitungen (USB, Ethernet, usw.) und Vorwiderstände, wurden die folgenden Komponenten für die schaltungstechnische Umsetzung der Messaufgabe ausgewählt und verwendet.

2.3.1 Flüssigkristall-Display DIP204B-4NLW

Um den Status des Messsystems HAILsens auf einfachem Weg schnell einsehen zu können, wird neben den Signal-LEDs an Board der Concerto controlCARD ein vierzeiliges LCD-Modul der Firma ELECTRONIC ASSEMBLY mit 20 Zeichen pro Zeile am Microcontroller angeschlossen. Die aktuell gelieferten Messdaten aller Sensoren können über einen Taster ausgewählt und programmatisch in Echtzeit am Display ausgegeben werden (vgl. Abschnitt 5.4), wodurch eine Überprüfung der Funktion der Sensor-ICs und des groben Programmzustandes möglich ist. Abbildung 2.9 zeigt das LCD-Modul mit dessen darauf dargestellten Basisdaten.

Abbildung 2.9: LCD-Modul DIP204B-4NLW [48]

Das DIP204B-4NLW verfügt über den integrierten Display-Controller KS00073 von Samsung, welcher die Kommunikation mit dem Modul über eine SPI-Schnittstelle und die Steuerung der Punktmatrix des Displays implementiert. Neben einigen nicht genutzten und deshalb in dieser Arbeit auch nicht ausgeführten Features, wie beispielsweise frei programmierbare Zeichen, ist die Konfiguration des Displays und und die Kontrolle der Anzeigedaten in Abschnitt 3.2.3 detailliert beschrieben [48, 49].

Eine Spannungversorgung von $V_S = +3,3\,V$, sowie die SPI-Anschlüsse MOSI[17], SCK und CS[18] (vgl. Abschnitt 3.2) sind für den Betrieb des Displays erforderlich. Um die serielle Datenübertragung zu aktivieren, muss auf der Rückseite des Moduls unbedingt die im unteren, mittleren Bereich befindliche Brücke auf das mit SPI beschriftete SMD-Pad umgelötet werden. Zusätzlich gibt es separate Eingänge für Kontrastspannung und LED-Hintergrundbeleuchtung des LCD-Moduls. Die Schaltung der LED-Beleuchtung durch den Concerto Microcontroller wird im folgenden Abschnitt 2.3.2 genauer erläutert [48]. Der Display-Treiber benötigt eine Startup-Zeit von mindestens $20\,ms$ [48, 49].

2.3.2 Bipolar-Transistor BC846

Die weiße LED[19]-Hintergrundbeleuchtung des Flüssigkristall-Displays hat eine in [48] angegebene maximale Stromaufnahme von $45\,mA$. Die GPIOs[20] des F28M35H52C1 können zwar im C-Programm unter anderem als Open-Drain Ausgänge konfiguriert (sog. Pin Muxing) und so prinzipiell zum Schalten verwendet werden, allerdings ist im Datenblatt [39] ein maximal schaltbarer Strom von $4\,mA$ angegeben. Für das direkte Schalten ist der Strom durch die LED also zu hoch. Deshalb wird für den verwendeten GPIO8 (Pin15 des Concerto Controllers, vgl. Anhang Seite 70) eine Konfiguration als Push-Pull-Ausgang gewählt, welcher $V_{OH} = 3,3\,V$ und $V_{OL} = 0\,V$ ausgibt. Dieser Ausgang ist mit der Basis eines NPN-Transistors der Firma Fairchild Semiconductor verbunden (vgl. Abbildung 2.10). Der Basisstrom beträgt für eine Basis-Emitter-Spannung von $V_{BE} = 3,3\,V$ nach [50] nur $I_B = 100\,\mu A$, was eine sichere Verschaltung mit dem Controller-Ausgang zulässt [48, 50].

Die Kathode der LED kann dann auf den so genannten Open-Collector-Ausgang des Transistors gelegt werden. In diesem Setup hat der Bipolartransistor die Funktion eines über die Spannung am Basisanschluss steuerbaren elektrischen Schalters. Der Kollektorstrom darf nach Datenblatt maximal $I_C = 100\,mA$ betragen, was für den Betrieb der LED-Hintergrundbeleuchtung des Displays ausreichend ist. Der Vorwiderstand $R_1 = 100\,k\Omega$ schützt den Basis-Anschluss des Transistors vor Überstrom. Der Widerstand $R_2 = 2,2\,\Omega$ vor der LED sorgt durch einen geeignet großen Spannungsabfall von etwa $V_{R2} = 0,1\,V$ für eine notwendige Vorwärtsspannung von $V_F = 3.2\,V$ durch die Diode. Zusätzlich wird dadurch der Strom durch die Leuchtdiode auf die in [48] geforderten

$$I_F = \frac{V_{R2}}{R_2} = \frac{0,1\,V}{2,2\,\Omega} \approx 45\,mA \tag{2.11}$$

begrenzt und damit einer Zerstörung des Bauteils vorgebeugt [48, 50].

[17] MOSI: Master Out Slave In
[18] CS: Chip Select
[19] LED: Light-Emitting Diode
[20] GPIO: General Purpose Input/Output

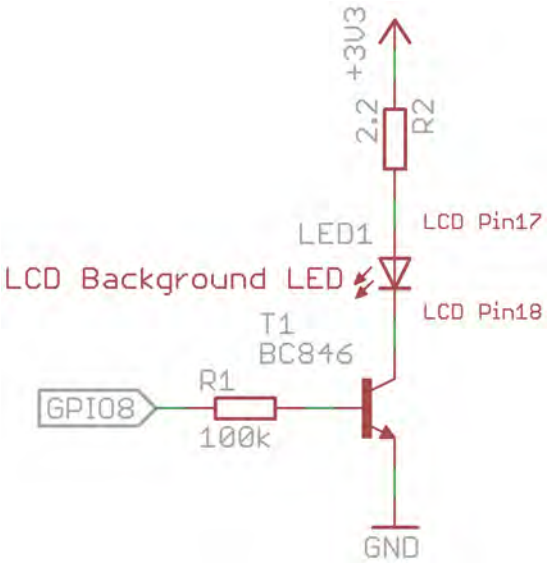


Abbildung 2.10: Über Transistor geschtaltene LED-Beleuchtung des LCD-Moduls

2.3.3 Bidirektionaler Level-Converter TXS0104E

Abbildung 2.11: TXS0104E Pegelwandler [46]

Die Aufgabe des Texas Instruments Pegelwandlers TXS0104E ist es, den I^{2}C-HIGH-Pegel des Drucksensors (vgl. Abschnitt 2.2.1) von $5V$ auf die von den übrigen Sensoren benutzten $+3,3V$ anzugleichen. Da die Kommunikation über den Bus bidirektional ist, muss in beide Richtungen umgesetzt werden, also von $5V$ nach $+3,3V$ und umgekehrt. Die Besonderheit dieses speziellen Level-Converters ist, dass kein separates Signal für die Festlegung der Konvertierungsrichtung notwendig ist. Der IC löst diese Aufgabe durch eine spezielle interne Beschaltung. Möglich ist die parallele Konvertierung von bis zu vier angeschlossenen Signalen. Für den I^2-Bus werden jedoch nur die beiden Signalpfade für SDA und SCL benötigt. Die schaltungstechnische Integration des Pegelwandlers wurde in Abbildung 2.8 auf Seite 15 bereits gezeigt [47].

2.3.4 Micro SD Speicherkarte

Die über HAILsens von den Sensoren aufgenommenen Messwerte sind in Rohdatenform im speziell für das Projekt RO-BERTA vom Teammitglied Dipl.-Ing.(FH) Martin Heigl entwickelten Übertragungsprotokoll DTP[21] organisiert (vgl. Abschnitte 3.3.1, 5.4.2 und Anhang A.3.3). Um einem Datenverlust bei der Ethernet-Übertragung der gesammelten Rohwerte an die zentrale Verarbeitungseinheit HAILscout an bord des Flugzeugs (vgl. Abbildung 1.1 auf Seite 2) wegen eventueller Übertragungsfehler vorzubeugen, werden alle erstellten DTP-Frames zusätzlich auf einer $2\,GB$ Micro SD Speicherkarte von SanDisk gespeichert (siehe Abbildung 2.12, rechter Teil).

[21] DTP: Domino Transfer Protocol

Abbildung 2.12: Micro SD Speicherkarte mit SD-Adapter

Das Speichermedium sitzt während des Betriebes von HAILsens im Micro SD-Slot der Concerto controlCARD (vgl. Abbildung 2.2 auf Seite 5). Als Schnittstelle zwischen SD Karte und Concerto Controller dient ein SPI-Interface des Slots (vgl. Abschnitt 3.2.3). Die pro Sekunde produzierten Frames werden, gesteuert von der HAILsensRTapp Software (vgl. Abschnitt 5.4.1), jeweils in einer eigenen Datei mit inkrementeller Nummerierung im Dateinamen abgespeichert. Die einzelnen Dateien mit der Dateinamenserweiterung ROB (für RO-BERTA) haben jeweils eine Größe von rund $4,79\,KB$. Das in der HAILsensRTapp implementierte FAT16[22]-Dateisystem hat eine Obergrenze von

$$2\,GB \;=\; 2\cdot 2^{30}\,Byte \;=\; 2^{21}\,KB \;=\; 2097152\,KB \tag{2.12}$$

adressierbarem physikalischem Speicher. Es können demnach theoretisch für eine Dauer von

$$\frac{2097152\,KB}{4,79\,\frac{KB}{s}} \;\approx\; 437819\,s \;\approx\; 122\,h \tag{2.13}$$

Messdaten gespeichert werden. Ein Hagelabwehr-Flug dauert im Durchschnitt etwa drei Stunden, weshalb das Speichervolumen von $2\,GB$ der verwendeten SanDisk-Karte ausreichend ist. Mit dem in Abbildung 2.12 dargestellten SD-Adapter kann die Karte in den SD-Slot eines PCs eingeführt und so die erstellten Dateien auf diesen übertragen werden. Mit spezieller Software, beispielsweise einem Hex-Editor wie [51] in Abschnitt 6.3 beschrieben, können die gespeicherten Dateien dann weiter untersucht oder verarbeitet werden.

[22] FAT: File Allocation Table

3 Buskommunikation

Für die Datenkommunikation der in den vorangegangenen Abschnitten 2.1 bis 2.3 aufgeführten intelligenten Peripheriegeräte von HAILsens sind verschiedene serielle Busse im Einsatz. Wesentlich für Bussysteme ist, dass die Busteilnehmer auf ein gemeinsames Übertragungsmedium zugreifen. Das Protokoll bzw. die Spezifikation eines Busses regelt den Zugang zum Medium, um eine Kollision der von den angeschlossenen Geräten auf dem Bus übertragenen Daten zu vermeiden. Wie diese Übertragungssicherheit erreicht wird und wie sich der Ablauf einer Datenübertragung prinzipiell gestaltet beschreibt dieses Kapitel, gefolgt von der Anwendungen der einzelnen Bussysteme im entwickelten Prototypen [8].

3.1 I²C

Die Anzahl der Pins eines Microcontrollers ist begrenzt. Damit trotz dieser Limitierung viele Peripheriegeräte, wie beispielsweise die oben beschriebenen Sensor-ICs angeschlossen werden können, empfiehlt sich der Einsatz eines seriellen Datenbusses mit möglichst wenig Steuerleitungen.

Der I²C-Bus eignet sich ideal für die Kommunikation zwischen integrierten Schaltungen (ICs). Daher auch dessen Name, ausgeschrieben Inter-Integrated Circuit. Es handelt sich um einen Bus mit nur zwei Leitungen (deshalb auch TWI[1]), der die in der Regel raren Pins eines ICs bzw. eines Microcontrollers effektiv nutzt [14, 53].

3.1.1 Protokollbeschreibung

Dieser Bus implementiert eine Master-Slave-Architektur. In der Regel gibt es einen Master (auch Multimaster-Mode möglich), der das Taktsignal erzeugt und damit auch das Übertragungsmedium durch eine Steuerung des Pegels auf der Taktleitung SCK kontrollieren kann. Übertragen werden die Daten zwischen dem Master und den meist mehreren Slaves seriell in Paketen von 8 *bit* über die SDA-Leitung des Busses[14, 53].

Der I²C-Master, im Fall von HAILsens ist das der Concerto Microcontroller, initialisiert den Übertragungsablauf durch das Erzwingen einer START-Bedingung. Nach jedem übertragenen *Byte* setzt der jeweilige Empfänger ein Acknowledge-Signal auf dem Bus, um den Erhalt der Daten zu bestätigen. Der Empfänger zieht dafür die vom Sender in den IDLE-Zustand (Leerlauf) entlassene Datenleitung SDA auf LOW. Der Master erzeugt für für das Acknowledge-Bit nach jedem *Byte* einen zusätzlichen Taktimpuls [27].

[1] TWI: Two-Wire Interface

Abbildung 3.1 zeigt den Begin eines Datentransfers und die Bestätigung des Datenempfangs durch das Acknowledge-Bit.

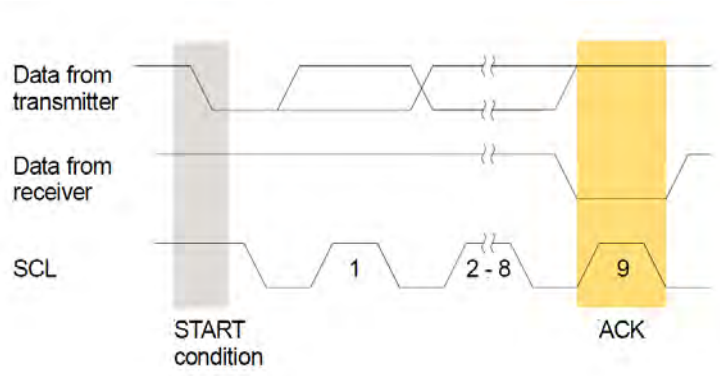


Abbildung 3.1: I²C START-Bedingung und Acknowledge-Bit [27]

Jedem Busteilnehmer ist eine 7 *bit* lange Adresse zugeordnet, die diesen eindeutig innerhalb des Busses identifiziert. Grundsätzlich existieren also $2^7 = 128$ verschiedene Busadressen. Allerdings sind 16 dieser Adressen für Sonderzwecke reserviert, was die Anzahl der frei adressierbaren Busknoten auf 112 reduziert. Die Slave-Adresse sendet der Master im ersten *Byte*, dessen achtes *bit* (R/$\overline{W}$, vgl. Abbildung 3.2) die Übertragungsrichtung signalisiert, d.h. ob Daten zum Slave geschrieben oder welche von ihm gelesen werden sollen. Danach beginnt der eigentliche Datentransfer, welchen ein negiertes Acknowledge (NACK) gefolgt von einer STOP-Bedingung abschließt. In Abbildung 3.2 ist ein kompletter I²C-Bustransfer grafisch dargestellt [27].

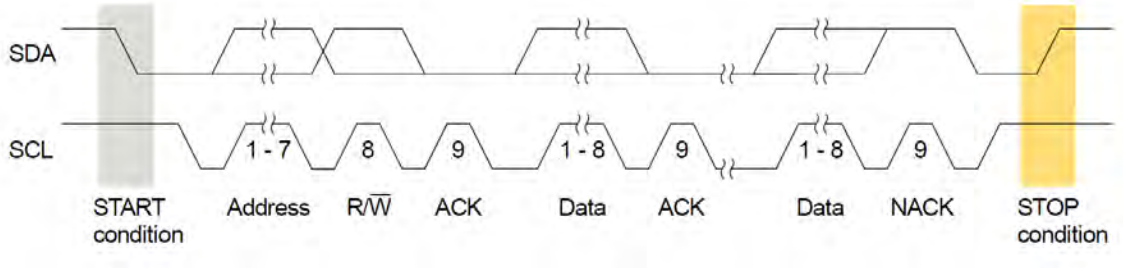


Abbildung 3.2: Kompletter I²C Bustransfer [27]

Es sind Übertragungsraten von $100\,kHz$ (Standard Mode), $400\,kHz$ (Fast Mode), $1\,MHz$ (Fast Mode Plus) und $3{,}4\,MHz$ (High Speed Mode) erlaubt [14].

Ein Nachteil des I²C-Protokolls ist, dass die Slaves nicht selbstständig eine Datenübertragung einleiten können. Dies spielt jedoch für die Anwendung in HAILsens keine Rolle.

Weitere detaillierte Beschreibungen der I²C-Buskommunikation finden sich u.a. in den Sensor-Dokumentationen [25], [26], [27] und [30].

3.1.2 Elektrischer Anschluss

Die I²C-Busspezifikation schreibt vor, dass das SDA- und das SCK-Signal im IDLE-Zustand auf HIGH-Potential befinden muss (Active LOW). Um dieses Verhalten elektrisch zu erreichen, werden die beiden Leitungen über Pullup-Widerstände mit dem HIGH-Potential verbunden (vgl. Abbildung 3.3). Wenn durch die Widerstände R_P kein Strom fließt ($I = 0$), fällt an ihnen wegen $U = R \cdot I$ auch keine Spannung ab. Das Potential an der Verbindung des Widerstandes zur Busleitung wird dann nach HIGH bzw. V_{DD} gezogen [53].

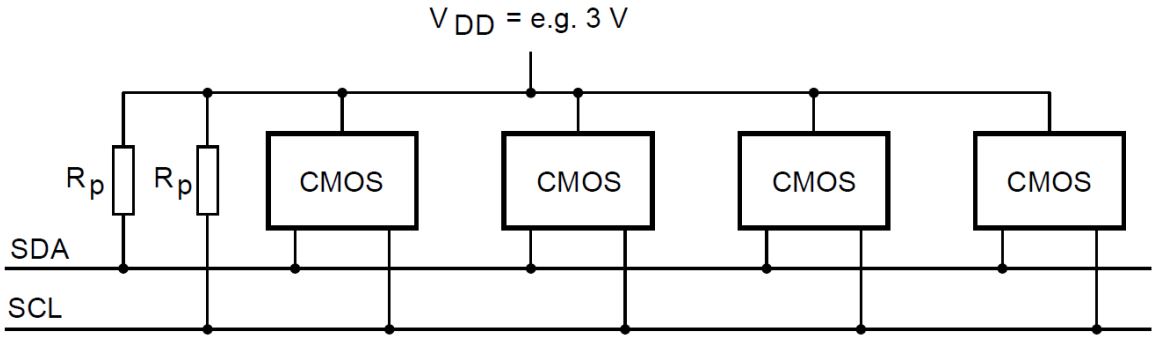


Abbildung 3.3: Elektrische Verschaltung von I²-Busteilnehmern [53]

Die in Abbildung 3.3 dargestellten Busteilnehmer sind Mitglieder der CMOS[2]-Logikfamilie. Die Nutzung von MOSFET[3]-Ein- und -Ausgängen in Halbleitern ist bei ICs weit verbreitet und findet auch beim Microcontroller und den Sensoren des Subsystems HAILsens Anwendung [18, 9].

Abhängig von der Versorgungsspannung V_{DD}, der Kapazität des Busses und den angeschlossenen Busteilnehmern ist die Wahl der Größe des Pullup-Widerstandes. Für die meisten Fälle passend ist in der Praxis ein Wert von etwa $R_P = 5\,k\Omega$, welcher auch am I²C-Bus von HAILsens verwendet wird [53].

3.1.3 Anwendung

Allen Sensor ICs, die mehr als eine physikalische Größe liefern können, muss mitgeteilt werden, welche Messgröße beim nächsten Lesezyklus über den Bus geschickt werden soll. Beim SHT21 beispielsweise gibt es Trigger-Kommandos für Luftfeuchte- oder Temperaturmessungen. Für die Triggerung wird zum Slave geschrieben, weshalb das Lese-Bit (vgl. R/$\overline{\text{W}}$ in Abbildung 3.2) im ersten übertragenen *Byte* nicht gesetzt ist. Die Triggerung ist beim Drucksensor nicht notwendig, da dieser ohnehin lediglich Druckwerte zur Abfrage bietet.

[2]CMOS: Complementary Metal Oxide Semiconductor
[3]MOSFET: Metal-Oxide-Semiconductor Field-Effect Transistor

Abgerufen werden die Messwerte bei allen Sensoren, indem der Master die Slave-Adresse mit gesetzten Lese-Bit über den Bus schickt (vgl. Abbildung 3.4). Nachdem der betreffende Slave diese Information aufgenommen hat, beginnt er mit dem Datentransfer. Tabelle 3.1 gibt einen Überblick zu den Busadressen der Sensor-ICs und den Kommandos zur Initialisierung einer Messung.

Sensor	**Adresse**	**Messgröße**	**Triggerung**	**Anz.** *Byte*
HDI0611ARZ8P5	$0x78$	Luftdruck	nicht notwendig	2
SHT21	$0x40$	Luftfeuchte	$0xF5$	2
		Temperatur	$0xF3$	2
HMC6342	$0x19$	Ausrichtung	$0x50$	6
		Magnetfeld	$0x45$	6
MPU-6050EVB	$0x68$	Beschleunigung	$0x3B$	6
		Drehraten	$0x43$	6

Tabelle 3.1: I²C-Adressen und -Kommandos [27, 25, 28, 31]

Die Anzahl der nach einem Abfragekommando vom entsprechenden Slave gesendeten *Byte* variiert. Jeder Messwert wird zwar bei allen Sensor-ICs innerhalb von $2\,Byte$ auf dem Bus übertragen, falls es sich jedoch um Messungen in den drei Raumrichtungen handelt, schickt der angesprochene IC diese drei Messwerte (insgesamt $6\,Byte$) nacheinander in Folge [27, 25, 28, 31].

Drucksensor

Abbildung 3.4 zeigt den Ablauf einer Messwert-Abfrage am Beispiel des Drucksensors, der die $15\,bit$ Druckinformation für einen Messwert in $2\,Byte$ aufgeteilt überträgt.

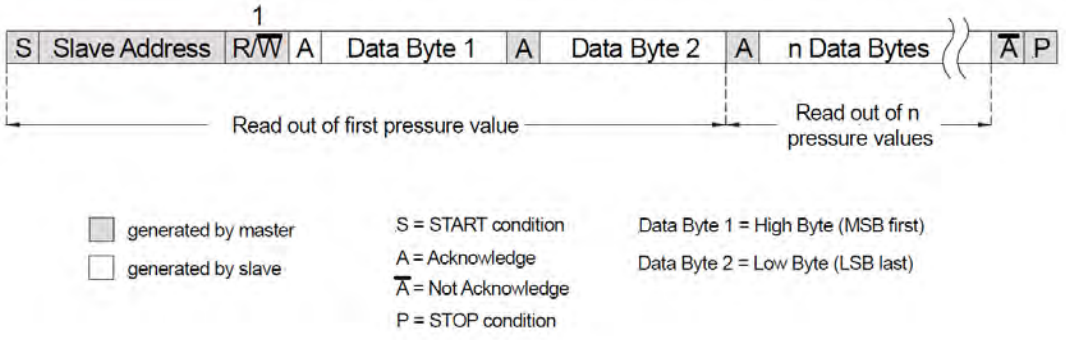


Abbildung 3.4: I²C Datenabfrage am Beispiel des Drucksensors [27]

Die C-Funktion `HDI0611GetAcquiredData()` implementiert diese Datenabfrage. Der abgefragte Messwert wird als Rohdatum in $2\,Byte$ eines Puffer-Arrays für die weitere Verarbeitung gespeichert (vgl. Abschnitte 5.2.2 und 5.4.3).

Feuchte- und Temperatursensor

Die Triggerung der Messungen ist in der C-Funktionen `SHT21StartMeasurementCycle()` umgesetzt, welcher der Typ des Messwerts als Parameter übergeben wird. Abgerufen können Feuchte- oder Temperaturwert dann mit Hilfe der Funktion `SHT21GetAcquiredData()` nach dem in Abbildung 3.4 dargestellten und oben beschriebenem Schema. Wie unter Abschnitt 2.2.2 aufgeführt, verwendet HAILsens die Default-Einstellungen des SHT21. Dadurch ist eine gesonderte Initialisierungssequenz für den IC nicht notwendig. Im Hauptprogramm wird der Sensor beim Systemstart einmalig mit Hilfe der Funktion `SHT21SoftReset()` zurückgesetzt.

Elektronischer Kompass und Bewegungssensor

Die Extraktion der Messwerte beim HMC6343 und beim MPU-6050 ähneln sich weitestgehend. Die Triggerung einer Messung wurde in der C-Funkion `RequestXYZData()`, die Datenabfrage in der Funktion `GetXYZData()` implementiert. Die Funktionen haben u.a. die Busadresse des jeweiligen ICs und den Messwerttyp als Parameter. Dadurch ist eine Mehrfachnutzung beider Funktionen möglich (vgl. Abschnitte 5.2.2 und 5.4.3). HAILsens verwendet die Standardwerte aller möglichen Einstellungen beim HMC6343. Eine spezielle Konfigurationsroutine ist nur für den MPU-6050 in der Funktion `MPU6050Configure()` programmiert, um die im Abschnitt 2.2.4 erwähnten Messbereichseinstellungen und den digitalen Tiefpassfilter einzustellen [28, 31].

Als Taktrate für die I²C-Schnittstelle des Concerto Microcontrollers wurde $100\,kHz$ gewählt. Diese Rate empfehlen die Datenblätter einiger verwendeter Sensor-ICs. Abschnitt 6.2 diskutiert später noch, dass trotz dieser relativ niedrigen Übertragungsgeschwindigkeit keinerlei Einschränkungen im Timing der Programmausführung von HAILsensR-Tapp entstehen.

3.2 SPI

Beim SPI[4] handelt es sich ebenfalls um einen serieller Bus nach dem Master-Slave-Prinzip, jedoch mit einer separaten Slave-Select-Leitung ($\overline{SS}$, Active LOW) zur Ansteuerung der einzelnen Slaves. Eine eigene Adressierung der Peripheriegeräte, so wie beim I²C-Bus, entfällt deshalb. Der dadurch entstehende Nachteil ist, dass die Anzahl der Steuerleitungen mit der Anzahl der angeschlossenen Busteilnehmer steigt, da jedem Slave eine eigene Aktivierungsleitung benötigt (vgl. Abbildung 3.5). Der Master-IC muss dafür entsprechend viele Pins zur Verfügung stellen können. Da am Concerto Microcontroller (Master) jeweils nur ein einziger Slave über ein Bus-Interface kommuniziert, fällt dieser Nachteil bei der HAILsens Hardwarekonfiguration nicht ins Gewicht [21].

[4] SPI: Serial Peripheral Interface

3.2.1 Protokollbeschreibung

Die SPI-Busspezifikationen sind insgesamt recht locker formuliert. Das Taktsignal SCLK[5] wird vom einzigen Bus-Master erzeugt und damit auch kontrolliert. Es sind Taktraten bis in den MHz-Bereich erlaubt. Für den Datenaustausch gibt es zwei Leitungen. Über den Anschluss MOSI[6] werden Informationen vom Master zum Slave, über den SOMI[7]-Anschluss vom Slave zum Master übermittelt. Dadurch ist der SPI-Bus vollduplexfähig, das heißt, alle Busteilnehmer können gleichzeitig senden und empfangen. Notwendig sind dafür jedoch die zwei o.g. separaten Leitungen, was sich u.U. bei einer geringen IC-Pinanzahl als Nachteil erweisen kann [21].

Aktiviert wird ein Slave, indem der Master dessen $\overline{SS}$-Leitung gegen Masse zieht. Dann kann der Datentransfer beginnen. Für die Übertragung existieren vier verschiedene Modi (Mode 0 bis 3), welche sich aus einer Kombination der beiden Parameter CPOL[8] und CPHA[9] ergeben. HAILsens benutzt Mode 0, wodurch bei jeder steigenden Flanke des SCLK-Signals der Pegel des Datensignals (MOSI oder SOMI) übernommen wird. Es ergibt sich eine Übertragungsrate von einem *bit* pro Taktimpuls [21].

3.2.2 Elektrischer Anschluss

Die elektrische Verschaltung der Busteilnehmer ist in Abbildung 3.5 dargestellt.

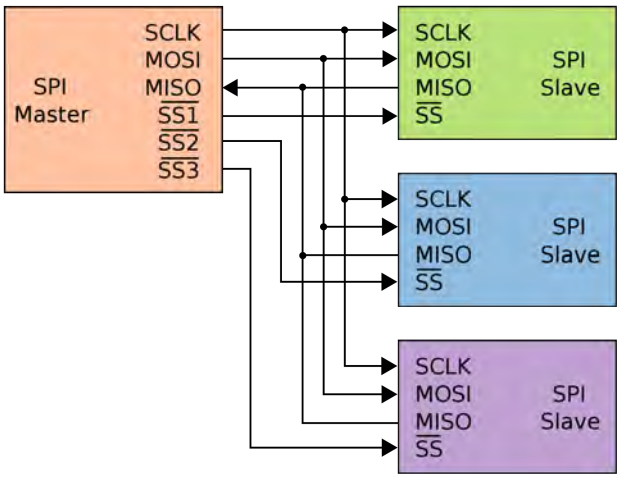


Abbildung 3.5: SPI-Hardwarekonfiguration mit drei Slaves [21]

Besonderheiten wie Pullup-Widerstände und dergleichen gibt es hier nicht.

[5] SCLK: Serial Clock
[6] MOSI: Master Out, Slave In
[7] SOMI: Slave Out, Master In
[8] CPOL: Clock Polarity
[9] CPHA: Clock Phase

3.2.3 Anwendung

Im Subsystem HAILsens werden zwei separate, vom F28M35H52C1 bereit gestellte SPI-Transceiver zur Kommunikation mit dem LCD-Modul und mit der Micro SD Speicherkarte benutzt. Prinzipiell wäre natürlich ein Master-Interface für die beiden Slaves ausreichend. Der Kartenslot auf der Concerto controlCARD ist jedoch mit den entsprechenden Pins des Concerto SSI0[10]-Peripherals platinenintern verdrahtet. Außerdem verwendet der SPI-SD-Kartentreiber des MCU SDK[11] (siehe Abschnitt 4.3) spezielle Interrupts und andere Konfigurationen für SSI0, welche nicht mit der Datenkommunikation zum Display kompatibel sind. Die Schnittstelle zum LCD-Modul nutzt deshalb einen eigenen SPI-Port des Microcontrollers (SSI1), um kommunikationstechnische Konflikte zu vermeiden (vgl. auch Tabelle in Abbildung A.3.1 auf Seite 70 im Anhang) [39].

LCD-Modul

Die Taktrate für den Datenaustausch mit dem LCD-Modul wurde in der Funktion `InitLCDperipherals()` auf $100\,kHz$ gesetzt. Der Display-Controller verlangt ein spezielles Übertragungsprotokoll. Jedes zu sendende *Byte* muss zuerst gespiegelt (LSB first) und in zwei $4\,bit$ breite *Nibble* aufgeteilt werden. Die beiden *Nibble* werden danach, jeweils gefolgt von einem Null-*Nibble*, in zwei separaten *Byte* übertragen wird. Diese Sequenz zur Datenaufbereitung und für das anschließende Senden ist in der C-Funktion `LCDsendByte()` implementiert, der einfach das zu sendende *Byte* übergeben werden kann. Diese Funktion, sowie weitere komfortable Treiberroutinen für die Display-Kommunikation, sind in der Bibliotheksdatei *lcd.h* bzw. *lcd.c* zusammengefasst, welche in Abschnitt 5.2.1 noch genauer besprochen werden. Der SPI-Bus findet sowohl für die Konfiguration des LCD-Moduls (z.B. Cursoranzeige aus), wie auch für die Übertragung der anzuzeigenden Daten Verwendung. Da sich an SSI1 nur dieser eine Slave befindet, ist die $\overline{SS}$-Leitung dauerhaft mit $0\,V$-Massepotential verbunden.

Micro SD Kartenslot

Die Taktrate der Übertragung liegt hier bei $400\,kHz$. Genauere Details zur Kommunikation mit dem Micro SD Slot auf der Concerto controlCARD, bzw. der Micro SD Speicherkarte, wurden im Rahmen dieser Masterarbeit nicht untersucht. Die im MCU SDK beinhalteten Treiber-Bibliotheken bieten einfach zu handhabende C-Funktionen für die Initialisierung des Speichermediums, sowie das Lesen, Schreiben, Erstellen usw. von Dateien. Die in diesem Zusammenhang etwas komplexere Konfiguration des Seriellen Interfaces ist durch die von TI gelieferten SD Karten Bibliotheks-Funktionen gekapselt und die Untersuchung des exakten Kommunikationsablaufs nicht mehr notwendig.

[10] SSI: Synchronous Serial Interface
[11] MCU SDK: Microcontroller Unit Software Development Kit

3.3 Ethernet mit UDP

Ethernet dürfte die allgemein bekannteste, im Rahmen der Prototypenentwicklung für HAILsens verwendete Schnittstelle sein. Diese Technologie ist auf hohe Übertragungsraten und große Kabellängen ausgelegt. Abhängig von der verwendeten Variante sind beispielsweise bei $10\,Gbit/s$ Leitungslängen bis $220\,m$, bei $100\,Mbit/s$ sogar bis $2000\,m$ möglich. Eingesetzt wird Ethernet meist im Kontext von lokalen Computernetzwerken (LAN[12]) [13].

3.3.1 Protokollbeschreibung

Auch die Ethernet-Schnittstelle ist durch zwei separate Leitungen für das Senden und Empfangen von Daten vollduplexfähig. Die Datenübertragung auf dem Bus geschieht mittels digitalem Zeitmultiplex. Jeder Netzwerkteilnehmer verfügt über einen global eindeutigen Schlüssel, die so genannte MAC[13]-Adresse. Der Zugriff der auf das gemeinsame Übertragungsmedium wird über den CSMA/CD[14]-Algorithmus kontrolliert. Dieses asynchrone Medienzugriffsverfahren ist in der einschlägigen Literatur beschrieben. Im Rahmen dieser Arbeit wird auf Ausführungen zum Verfahren verzichtet. Der Ethernet-Treiber des MCU SDK regelt die Zugriffskontrolle intern, wodurch der Programmierer sich um die Details der Theorie und der Anwendung des Verfahrens nicht kümmern muss [10, 13, 16].

Ethernet ist als Basis für Netzwerkprotokolle wie TCP/IP[15] oder UDP[16] geeignet. Diese Protokolle regeln u.a. die Übertragungsart, Flusskontrolle oder Fehlersicherung. Die einzelnen vom Netzwerk übertragenen Protokollframes beinhalten dafür diverse Flags und Prüfsummen, sowie die MAC- und IP-Adressen der Kommunikationspartner [15, 19, 22, 23].

3.3.2 Elektrischer Anschluss

Ethernet-Teilnehmer werden durch genormte RJ-45 Stecker bzw. Steckbuchsen (vgl. Abbildung2.2 auf Seite 5), meist mit Cat-5 oder Cat-7 Kabel und ggf. über Router, Switches oder Bridges miteinander verbunden. Auf die Belegung des RJ-45 Steckers und die Funktion der einzelnen Leitungen wird im Rahmen dieser Arbeit nicht näher eingegangen. Die RJ-45 Buchse ist auf der Concerto controlCARD fest verdrahtet, was den Anwender von einer genaueren Untersuchung der Signale und Leitungen entbindet. Auf Seite 70 im Anhang befindet sich eine Auflistung der am Concerto Microcontroller für die Ansteuerung des Ethernet-Transceivers verwendeten Pins [13].

[12] LAN: Local Area Network
[13] MAC: Media Access Control
[14] CSMA/CD: Carrier Sense Multiple Access with Collision Detection
[15] TCP/IP: Transmission Control Protocol/Internet Protocol
[16] UDP: User Datagram Protocol

3.3.3 Anwendung

HAILsens benutzt die Ethernet-Schnittstelle, um die im Ausgangspuffer gesammelten Rohdaten aller Sensoren zur zentralen Verarbeitungseinheit HAILscout an bord des Hagelabwehr-Flugzeugs zu senden (vgl. Abbildung 1.1 auf Seite 2). Als minimales Übertragungsprotokoll wurde UDP gewählt, wobei HAILsens die Rolle des Servers und HAILscout die des UDP-Clients übernimmt. Der nicht unerhebliche Protokoll-Overhead von TCP/IP für Flusskontrolle und Fehlersicherung ist im relativ kleinen Bord-LAN des Flugzeuges mit lediglich sechs Busteilnehmern überflüssig. Aufwendiges Routing (wie im WWW[17]) oder das erneute Senden eventuell fehlerhafter Datenpakete ist nicht gefordert. Des weiteren ist die Kenntnis der exakten Framestruktur und Beschaffenheit der Datenübertragung mittels UDP für die Entwicklungsaufgabe nicht relevant, da TI im MCU SDK auch hier komfortable Bibliotheksfunktion bietet, welche einen Großteil der Komplexität der Schnittstellenkommunikation, sowie die Initialisierung und Kommunikation mit dem Ethernet-Transceiver auf der Concerto controlCARD für den Programmierer kapseln. Ethernet mit UDP fungiert in HAILsens somit als einfacher Container für den Datentransport [15, 22, 23].

Die IP-Adressen der Teilnehmer des Boardnetzes standen zum Zeitpunkt der abgeschlossenen Prototypenentwicklung noch nicht endgültig fest. Für erste Tests wurde an HAILsens 141.60.140.73 und an den HAILscout-Prototypen 141.60.140.70 vergeben. Die Übertragungsrate der Ethernet-Schnittstelle des Concerto Microcontrollers beträgt $100 Mbit/s$ [39].

Spezielles Übertragungsprotokoll: DTP

Im Rahmen des Forschungsprojektes RO-BERTA entwickelte Dipl.-Ing.(FH) Martin Heigl eine eigene Protokollstruktur. Für das aufwendige und weit verteilte Gesamtnetz des Projekts mit unterschiedlichsten Elementen (Relaisstation, Datenbank, diverse Subnetze usw.) kann damit u.a. eine elegante Zuordnung aller Hard- und Software-Komponenten als so bezeichnete Items geschehen. Die Sensordaten des Subsystems müssen in diese Protokoll- bzw. Framestruktur verpackt und mit den vom Protokoll geforderten Referenz-IDs, Prüfsummen und Error-Tags in den jeweiligen Headern ergänzt werden (vgl. Anhang A.3.3 auf Seite 72). Erst dann können die Rohdaten in den DTP-Frames über UDP an HAILscout gesendet werden. Die Berechnung aller Prüfsummen ist in der Software-ISR[18] `swi0FxnBuildTXframe()` implementiert, welche alle $100 ms$ von einer Funktion des Echtzeit-Betriebssystems getriggert bzw. aufgerufen wird (vgl. Abschnitt 5.4.2). Der Task `tsk0FxnIOhandler()` versendet dann die fertigen Frames mit der Funktion `sendto()` über einen UDP-Socket (vgl. Abschnitt 5.4.1) [2, 3].

[17] WWW: World Wide Web
[18] Interrupt Service-Routine

4 Software-Entwicklungswerkzeuge

4.1 controlSUITE

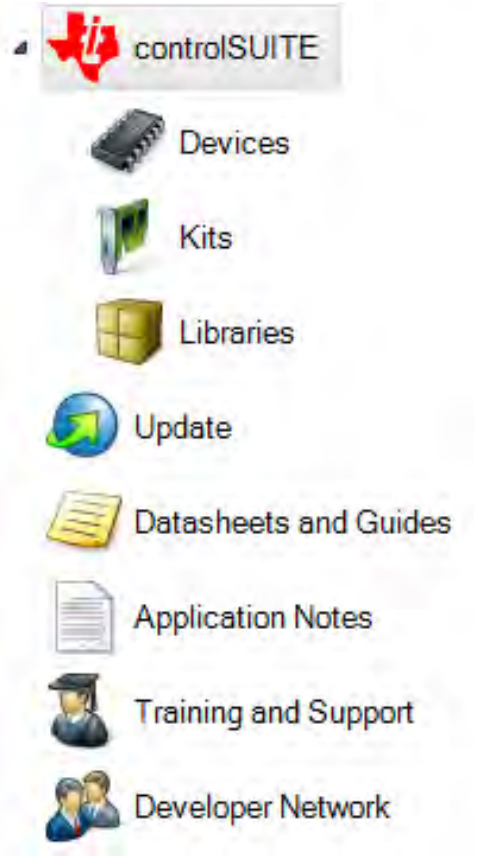


Abbildung 4.1: Ausschnitt aus dem controlSUITE-Hauptmenü

Die Software controlSUITE, welche kostenlos von TI zur Verfügung gestellt wird, bietet eine Infrastruktur von kompletten und lauffähigen Code-Beispielen, Hard- und Softwaredokumentationen, Online-Trainings und diversen anderen Hilfestellungen für die Entwicklungsarbeiten mit C2000 Microcontrollern. Dadurch entfällt die mitunter aufwändige Recherche zu Anwendungsbeispielen und Spezifikationen größtenteils und der Entwicklungsprozess kann effektiv beschleunigt werden, was eine erhebliche Zeit- und damit auch Kostenersparnis mit sich bringt. Abbildung 4.1 zeigt einen Ausschnitt des controlSUITE-Hauptmenüs [45].

Alle mit der Software gelieferten Dokumente und Beispielanwendungen können auch direkt im korrespondierenden Dateisystem, in der Regel unter dem Pfad *C:\TI\controlSUITE* angelegt, gefunden werden. Hier sind u.a. das Platinenlayout der Concerto controlCARD oder der USB Docking-Station in diversen Dateiformaten und mit Bauteillisten angelegt. Gerätespezifische C-Bibliotheken und Treiber für F28M35x MCUs sind hier ebenfalls abgelegt und können mit einer Entwicklungsumgebung (IDE[1]) wie beispielsweise dem TI Code Composer Studio in ein Programmier-Projekt integriert werden.

4.2 Code Composer Studio IDE

Die Eclipse-basierte Code Composer Studio (CCS) IDE stellt eine Fülle von Werkzeugen für die Software-Entwicklung mit TI MCUs und DSPs bereit. Die grafische Benutzeroberfläche kann modular konfiguriert und damit auf die individuellen Bedürfnisse des Benutzers angepasst werden. Der CCS-Einstieg in die C2000-Programmierung fällt mit dem Tutorial [42] aus controlSUITE relativ leicht. Nach dessen Abarbeitung kön-

[1] IDE: Integrated Development Environment

nen Code-Beispiele für die Ansteuerung von auf der Concerto controlCARD befindlichen LEDs, Bus-Schnittstellen oder einfachen IO-Pins zugig zum Laufen gebracht und deren Programmablauf untersucht werden. Die zwei Hauptmodi CCS Edit für die Programmierung in C oder C++ und CCS Debug für In-Circuit-Analysen des auf dem jeweiligen Controller laufenden Codes. Abbildung 4.2 zeigt die gebräuchlichsten Fensterbereiche im Edit-Modus.

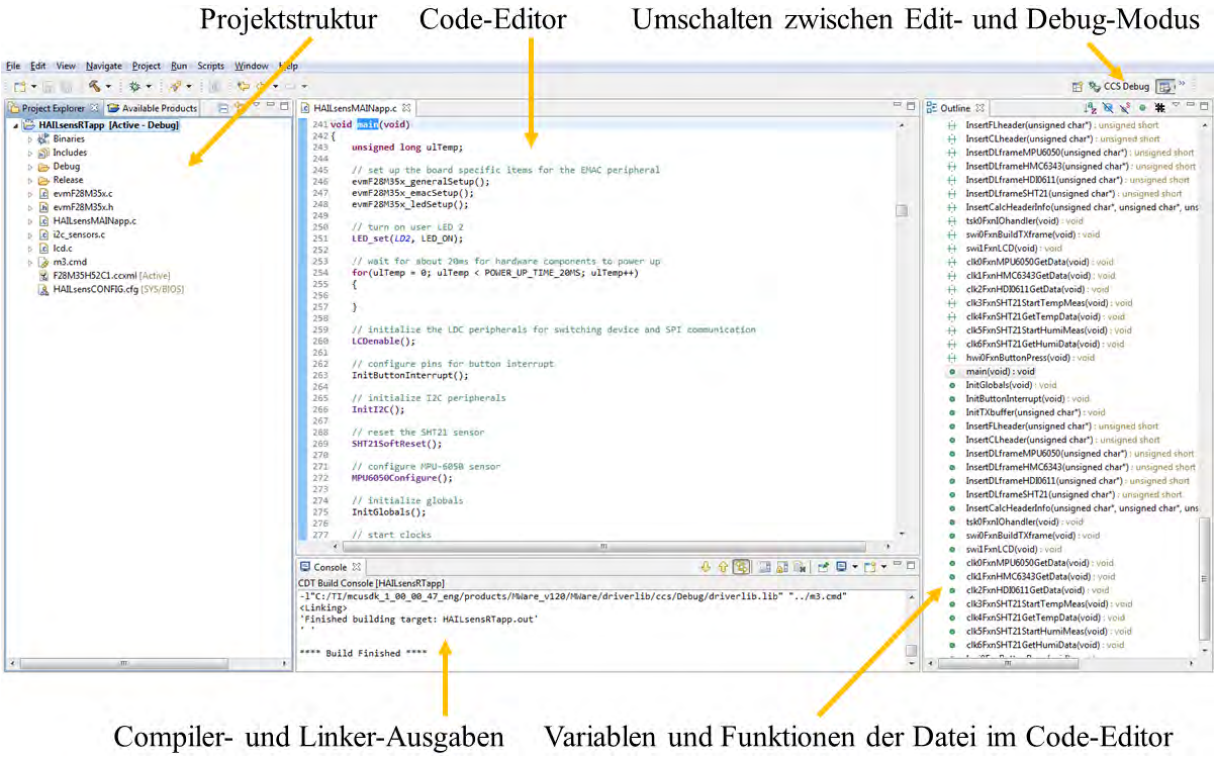


Abbildung 4.2: Fenster des Edit-Modus von CCS

Der Project Explorer (links in Abbildung 4.2) gibt einen Überblick zu den im Projekt benutzten Header- (*.h) und Quellcode-Dateien (*.c). Des weiteren sind hier auch andere, im Kontext der hardwarenahen C-Programmierung notwendigen Dateien direkt oder als Verlinkung zugänglich. Beispielsweise die Memory Map (*.cmd), welche die Verwendung der Speicherbereiche auf der MCU aufteilt und zuweist, die Konfigurationsdatei der Zielhardware (*.ccxml) oder die Konfiguration des Echtzeitbetriebssystems SYS/BIOS (*.cfg, vgl. auch Abschnitt 4.3.1). Das Tutorial [42] erläutert die Einbindung und Anwendung dieser in einem Programmier-Projekt benötigten Bausteine eingehend.

Im mittleren Fensterbereich ist der Code-Editor zu sehen. Hier werden Programme erstellt, oder die o.g. Projektdateien editiert. Für die C- und C++-Programmierung stehen gängige Editoroptionen, wie beispielsweise die farbliche Kennzeichnung von Datentypen, Funktionsnamen oder Strings zur Verfügung. Der rechte Fensterbereich (Outline) zeigt alle Variablen und Funktionen der im Editor geöffneten Datei, jeweils mit Deklaration und Definition. Der Konsolen-Output im unteren Fensterbereich gibt während der Kompilierung Informationen zu den gegenwärtigen Compiler und Linker-Aktivitäten.

Getestet wird die mit CCS erstellte Software im Debug-Modus. Durch die ISO USB JTAG-Schnittstelle auf der Concerto controlCARD gibt es die Möglichkeit, den auf dem Microcontroller laufenden Code detailliert und schrittweise zu untersuchen. Neben Standard-IDE-Optionen wie Breakpoints zur Prüfung des generellen Programmablaufs, kann die Veränderung des Zustandes eines Ausdrucks oder einer Variablen beobachtet, Registerwerte eingesehen oder in Kombination mit SYS/BIOS auch die Ausführung der am Echtzeit-Scheduling beteiligten Tasks über entsprechende Grafen dargestellt werden.

Abbildung 4.3 zeigt CCS im Debug-Modus. Die auf dem Concerto Controller laufende HAILsensRTapp hält gerade an einem Breakpoint (links unten im Bild) in der Routine für die Organisation der Datensendung über Ethernet (vgl. Abschnitt 5.4.1).

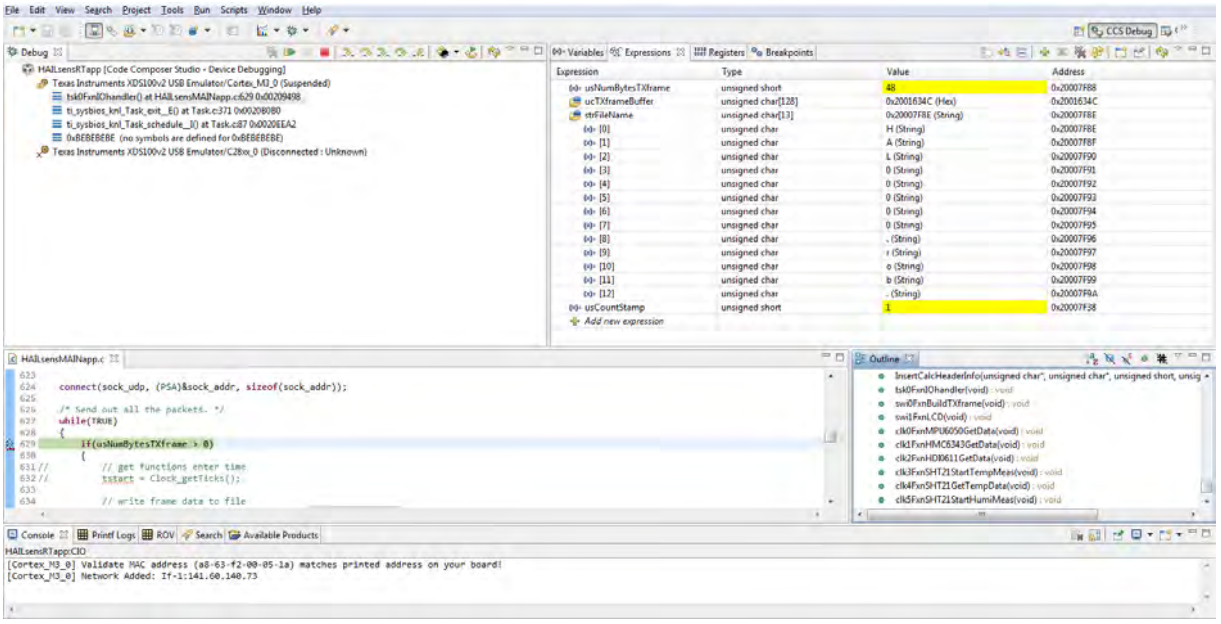


Abbildung 4.3: CCS im Debug-Modus

Im rechten oberen Bereich von Abbildung 4.3 ist die von CCS gelb markierte Variable zu sehen, welche den Füllstand des Sendepuffers anzeigt und deren Wert sich gerade von 0 auf 48 verändert hat. Da der Puffer nun gefüllt ist, werden die Daten im nächsten Schritt über einen UDP-Socket an HAILscout versendet.

Begonnen wurde die Entwicklungsaufgabe mit der CCS-Version 4.2.5.5 (CCSv4), welche im Rahmen der Nutzung mit dem H52C1 Experimenter Kit frei lizensiert ist. Es gab jedoch erhebliche Schwierigkeiten mit der Integration von Drittanbieter-Software für die Ethernet-Kommunikation in die TI-Echtzeitumgebung SYS/BIOS. Nach einer Anfrage zur Problematik im TI E2E-Forum machten TI-Mitarbeiter das Angebot, ein zum damaligen Zeitpunkt noch nicht veröffentlichtes Alpha-Release des MCU SDK (vgl. Abschnitt 4.3) zur Verfügung zu stellen. Das MCU SDK beinhaltet u.a. TI-eigene C-Bibliotheken für Ethernet-Kommunikation, läuft aber nur innerhalb CCS Version 5. Aus diesem Grund bildete der Umstieg auf CCSv5 (Version 5.1.0.9) die Grundlage für die weiteren Entwicklungsschritte.

4.3 MCU SDK

Das MCU Software Development Kit bildet eine Zusammenstellung der wichtigsten Tools für die Anwendungsprogrammierung mit CCSv5 in C mit Fokus auf TI Concerto Devices. Neben Mechanismen für die Interprozess-Kommunikation und diversen Schnittstellen Treibern, beispielsweise für die Ethernet-Kommunikation, bietet das SDK noch viele andere umfangreiche C-Bibliotheksdateien. Die in den Bibliotheken enthaltenen Funktionen regeln etwa die Organisation des FAT-Dateisystems auf der SD Karte im Slot der Concerto controlCARD oder das Zusammenspiel der einzelnen Komponenten unter dem Echtzeitbetriebssystem SYS/BIOS. Im Software Development Kit enthalten sind auch viele Beispiele, welche die Verwendung und die Interaktion der verschiedenen Software-Bausteine demonstrieren [43].

Abbildung 4.4 zeigt einen Überblick zu den Komponenten des MCU SDK, welcher sich in CCSv5 nach Doppelklick auf die im Project Explorer aufgelistete Konfigurationsdatei *HAILsensCONFIG.cfg* im Editor-Fenster öffnet. Die für die HAILsensRTapp verwendeten Bausteine sind am grünen Haken innerhalb der blauen Blöcke erkennbar.

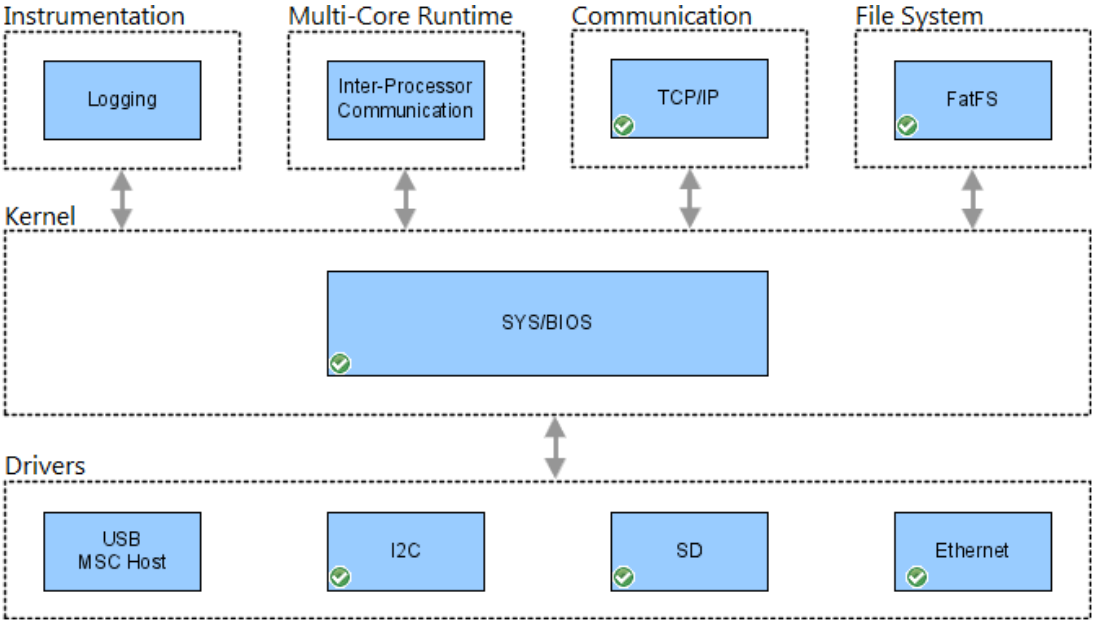


Abbildung 4.4: Überblick zu den Komponenten des MCU SDK in CCSv5

Nach Doppelklick auf die einzelnen Blöcke können die Module des MCU SDK detailiert und mit Hilfe einer grafischen Benutzeroberfläche komfortabel konfiguriert werden. Die Editierung der Datei *HAILsensCONFIG.cfg* ist nach Rechtsklick auf den Dateinamen im Project Explorer, dann der Auswahl *Open With→Text Editor* in den erscheinenden Popup-Menüs, auch mit einem textbasierten Editor möglich. Diese Option kann in einigen Anwendungsfällen einen Vorteil bringen.

4.3.1 SYS/BIOS

Das Echtzeit-Betriebssystem (RTOS[2]) SYS/BIOS wird von TI ebenfalls kostenlos angeboten. Abbildung 4.5 gibt einen Überblick zu den Komponenten von SYS/BIOS. Die Darstellung stammt aus dem XGCONFIG-Editor von CCSv5 für die grafische Editierung der Datei *HAILsensCONFIG.cfg*.

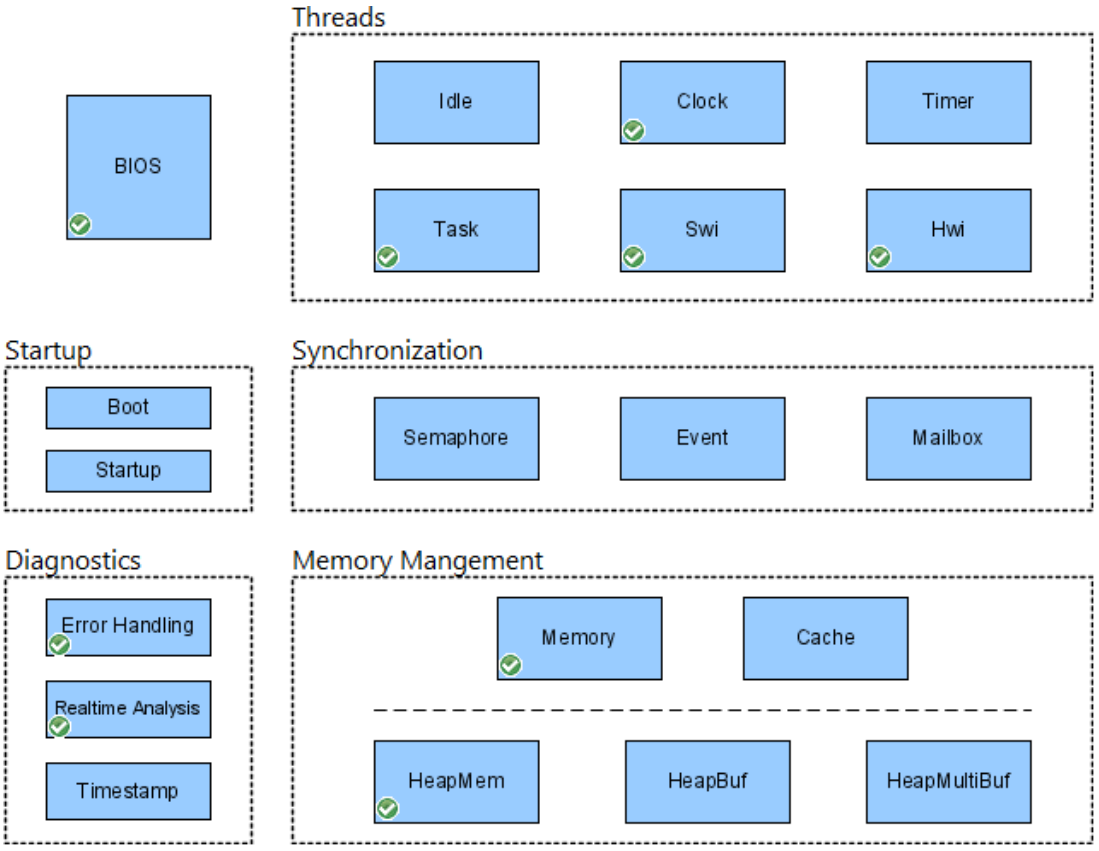


Abbildung 4.5: Überblick zu den Komponenten von SYS/BIOS in CCSv5

Diese Ansicht öffnet sich nach einem Doppelklick auf den Block *SYS/BIOS* aus Abbildung 4.4 und der anschließenden Auswahl von *System Overview* über den entsprechenden Button im Editor. Die in der HAILsens-Software benutzen Bausteine, welche in Abschnitt 5.4 noch genauer erläutert werden, sind von CCSv5 wieder mit einem grünen Haken markiert worden. Die einzelnen Blöcke können auch hier nach Doppelklick weiter editiert werden.

[2]RTOS: Real-Time Operating System

Standardmäßig läuft auf einem Microcontroller ein einziger Task mit einer Endlosschleife (z.B. `while(1)`). Die Steuerung des Prozessors durch ein Betriebssystem mit präemptivem, prioritätsgesteuertem Scheduler (RTOS) ermöglicht es jedoch, mehrere Tasks quasiparallel laufen zu lassen. Jeder Task hat eine eigene Priorität. Es sind beispielsweise, wie in SYS/BIOS, Prioritätslevel zwischen 0 und 31 vom Programmierer wählbar. Dem Task mit der höchsten Priorität, welcher für die Ausführung bereit ist, wird der Prozessor zugeteilt. Ein Task A mit höherer Priorität kann einen Task B mit niedrigerer Priorität zudem aus dem Prozessor verdrängen. Bei einem solchen Task-Switch wird der Prozessorkontext von Task B gespeichert. Wenn alle Tasks mit höherer Priorität als der von Task B beendet sind, der Prozessor für Task B also wieder frei ist, lädt der Scheduler wieder dessen Prozessorkontext. Der niedriger priorisierte Task B kann dann an der Stelle weiterverarbeitet werden, wo er zuvor vom höher priorisierten Task A unterbrochen wurde. Was dadurch erreicht wird, ist so genanntes Echtzeit-Verhalten. Der Software-Entwickler kann durch ein RTOS den Programmablauf so planen, dass alle Tasks einer Anwendung innerhalb strikter Zeitgrenzen abgearbeitet werden.

Für die Koordination der Sensordatenaufnahme mit festen Raten (z.B. Beschleunigungsmessung exakt alle $200\,ms$), des Managements der Sensordaten-Organisation in Frames, der Datenspeicherung auf SD Karte und der Anzeige der Messwerte über das LCD-Modul in einer einzigen MCU-Anwendung bringt die Benutzung eines Echtzeitbetriebssystems entschiedene Vorteile. Die Strukturierung der Anwendung HAILsensRTapp, v.a. hinsichtlich der oben beschriebenen Echtzeitfähigkeit, wird in Kapitel 5.4 ausführlich beschrieben.

4.3.2 NDK

Eine API[3] für die Ethernet-Kommunikation findet sich im MCU SDK-Modul NDK[4]. Implementiert sind hier Übertragungsprotokolle aus den Schichten 3 bis 7 des OSI[5]-Referenzmodells. Die Protokolle können über entsprechende Bibliotheksfunktionen leicht in eine eigene Anwendung eingebunden werden. Programmbeispiele für die Verwendung der Netzwerkprotokolle sind im MCU SDK ebenfalls enthalten [20].

In Abbildung 4.6 ist ein Überblick zu den Funktionalitäten des NDK gegeben. Die Darstellung ist an das OSI-Modell angelehnt. Die abgebildete GUI, welche die Schnittstelle für die Konfiguration des Networking-Moduls im MCU SDK bildet, erscheint im XGCONFIG-Editor nach Doppelklick auf den Block *Communication* bzw. *TCP/IP* aus Abbildung 4.4.

[3] API: Application Program Interface
[4] NDK: Network Development Kit
[5] OSI: Open System Interconnection

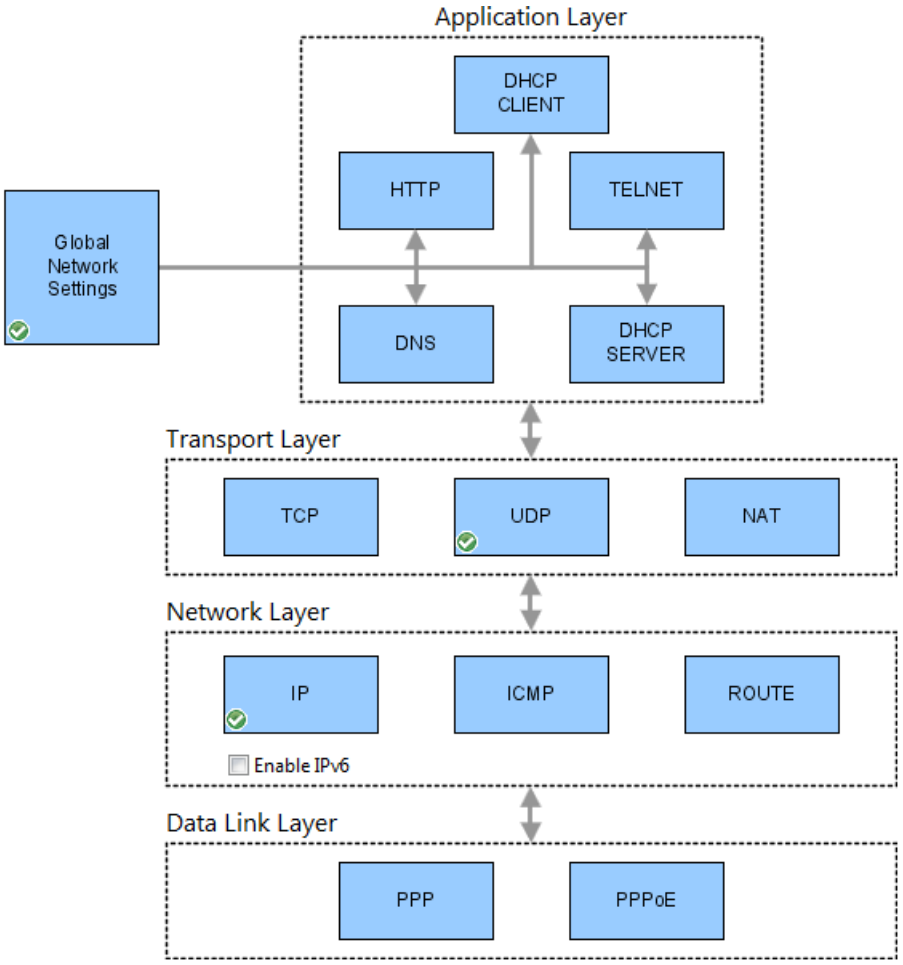


Abbildung 4.6: Überblick zu den Komponenten des NDK in CCSv5

HAILsens benutzt, wie in Abschnitt 3.3 bereits erwähnt, die Ethernet-Schnittstelle mit der Protokollvariante UDP aus der OSI-Transportschicht. Die dafür aktivierten NDK-Module sind in Abbildung 4.6 mit einem grünen Haken markiert.

5 Struktur des C-Programms HAILsensRTapp

Dieses Kapitel beschreibt den Kern der Entwicklungsarbeit für das Subsystem HAILsens. Der generelle Ablauf des Programms HAILsensRTapp, sowie die durch das SYS/BIOS RTOS ermöglichte zeitliche Planung der unterschiedlich priorisierten Tasks innerhalb des Echtzeit-Scheduling-Prozesses soll im folgenden erläutert werden.

Der gesamte, für die HAILsensRTapp geschriebene C-Code ist ausführlich mit Kommentaren versehen worden. Auch auf aussagekräftige Variablennamen und sonstige strukturierende Maßnahmen wurde bei der Erstellung der Software großer Wert gelegt, um die Lesbarkeit des Quellcodes weitestgehend zu erleichtern. Der Datenträger im Anhang dieser Masterarbeit beinhaltet u.a. die in diesem Kapitel beschriebenen Programmodule vollständig. Die **.h*- und **.c*-Dateien lassen sich mit allen gängigen ANSI-C-Editoren öffnen und untersuchen, weshalb auf einen zusätzlichen schriftlichen Ausdruck des Quellcodes im Anhang dieser Arbeit verzichtet wurde.

5.1 Genereller Ablauf

Auch im C-Code, der auf dem Concerto Microcontroller unter dem Echtzeit-Betriebssystem SYS/BIOS läuft, gibt es die C-Hauptfunktion `main()`. Eine Endlos-Schleife, wie sie in der klassischen Microcontroller-Programmierung zur Steuerung eines gesamten Programmablaufs üblich ist, sucht man hier jedoch vergeblich. Es werden in `main()` zwar der M3-Kern und die in der Anwendung benutzte Controller-Peripherie (I^2C, SPI, GPIOs usw.), sowie die angeschlossenen Sensoren und das LCD-Modul initialisiert und die in Abschnitt 5.4.3 noch näher beschriebenen Clock-Funktionen gestartet, das eigentliche Scheduling der echtzeit-relevanten Programm-Tasks beginnt jedoch erst mit dem Befehl `BIOS_start()` am Ende der Routine `main()`.

Nach der Initialisierung von Hardware und Programmvariablen (vgl. Abschnitt 5.3) erfüllt die Anwendung HAILsensRTapp im Wesentlichen die folgenden Aufgaben:

- Abfrage und Pufferung aller physikalischen Messgrößen innerhalb fester, zeitlich deterministischer Zyklen (Abschnitt 5.4.3)
- Arrangieren der aufgenommenen Sensor-Rohdaten in DTP-Übertragungsframes (Abschnitt 5.4.2)
- Regelmäßige, lokale Speicherung der generierten Frames in Form einzelner Dateien auf der Micro SD Karte, sowie das Senden dieser Frames über Ethernet-Datagramme (Abschnitt 5.4.1)

- Reagieren auf das asynchrone, vom Benutzer über einen Taster getriggerte Hardware-Interrupt-Ereignis zur Steuerung der Display-Ausgaben (Abschnitte 5.4.2 und 5.4.4)
- Ggf. Umrechnung der gepufferten Rohdaten mit anschließender Aufbereitung und Anzeige der aktuellen physikalischen Messwerte über das LCD-Modul (Abschnitte5.2.1, 5.2.2 und 5.4.2)

Die o.g. Aufgaben bzw. die betreffenden Tasks der Software sind in ein RTOS eingebettet. Sie werden somit quasiparallel ausgeführt. Aus diesem Grund ist die sonst im Allgemeinen übersichtliche Darstellung in einem Ablaufdiagramm schwierig, weshalb an dieser Stelle darauf verzichtet wird. Allerdings sind in den Unterpunkten ab Abschnitt 5.4 beispielhaft aussagekräftige Diagramme zur Beschreibung der Subroutinenabläufe enthalten. Damit, und mit der im weiteren erläuterten Verteilung der Task-Prioritäten für die Planung des Programmablaufs im RTOS, sollte sich dem Leser das teilweise etwas komplexere Zusammenwirken der Komponenten der Software HAILsensRTapp erschließen.

5.2 Erstellte Bibliotheken

Der Zweck der hier beschriebenen Bibliotheksfunktionen besteht in erster Linie in der Gewährleistung eines möglichst modularen Programmaufbaus der HAILsensRTapp. Wiederholt im Code auftauchende Befehlssequenzen wurden dafür jeweils in eine eigene Funktion ausgelagert. Gebündelt sind die Funktionen entsprechend ihrer Verwendung in Bibliotheksdateien (**.c* und **.h*). Vorteile dieses Ansatzes sind neben der erzielten Modularität und der daraus resultierenden Wiederverwendbarkeit der Funktionen in künftigen Projekten auch die aussagekräftigen (durch die gewählten Funktionsnamen) und kompakten Funktionsaufrufe im Hauptprogramm. Dadurch erhöhen sich die Übersichtlichkeit und Lesbarkeit des Programmcodes wesentlich.

5.2.1 SPI-Kommunikation mit dem Display

Zur Erleichterung der Kommunikation mit dem LCD-Modul über SPI wurden die Dateien *lcd.c* und *lcd.h* geschaffen. Damit steht eine einfach zu handhabende Programmierschnittstelle (API) für Displaykonfiguration und Displayausgaben zur Verfügung. Die unten aufgeführten Funktionen verwenden zum Senden eines *Byte* intern die von der TI API für den Concerto m3-Kern bereitgestellten Routinen. Damit wird ein *Byte* im Ausgangsregister des Controllers platziert und anschließend gesendet. Die Datenübertragung findet nur vom Concerto SPI-Master zum LCD-Modul (Slave) statt, da beispielsweise die eine Abfrage der Position des Cursors auf dem Display nicht notwendig ist. Des weiteren ist die $\overline{SS}$-Leitung des LCD-Moduls dauerhaft mit Masse verbunden, da auf diesem Bus nur der Concerto-Master und der Display-Controller-Slave unidirektional kommunizieren.

In der Bibliotheksdatei *lcd.c* sind die folgenden Funktionen enthalten:

`InitLCDperipherals()`

- → Aktivierung des Concerto SSI1 Peripherals
- → Zuweisung der Kontrolle über die für SPI verwendeten Pins an den M3-Kern
- → Einstellung der Pinfunktionen für SPI (vgl. Anhang A.3.1 auf Seite 70)
- → Aktivierung von Port B, Pin 0 als Push-Pull-Ausgang zum Schalten der LED-Hintergrundbeleuchtung (vgl. Abschnitt 2.3.2)
- → Parametrierung für *byte*weise Übertragung mit einer Rate von $100\,kHz$

`LCDenable()`

- → Aufruf von `InitLCDperipherals()` (s.o.)
- → Einstellung des LCD-Moduls für *byte*weise Übertragung, Anzeige in vier Zeilen und Cursor-Auto-Inkrementierung
- → Einschalten des Displays (Power Down Mode disable)
- → Löschen des Display-Inhalts
- → Setzen des Cursors auf Home-Position (links oben)

`LCDdisable()`

- → Display-Controller deaktivieren (Power Down Mode enable)
- → LED-Hintergrundbeleuchtung ausschalten

`LCDbacklightOn()`

- → LED-Hintergrundbeleuchtung einschalten

`LCDbacklightOff()`

- → LED-Hintergrundbeleuchtung ausschalten

`LCDclear()`

→ Löschen des Display-Inhalts

→ Setzen des Cursors auf Home-Position (links oben)

`LCDsendByte()`

→ *Byte* spiegeln

→ Oberes *Nibble* im ersten *Byte* an das LCD-Modul senden

→ Unteres *Nibble* im zweiten *Byte* senden

`LCDprintString()`

→ Ausgeben eines ASCII[1]-Strings an einer vorgebbaren Displayposition durch Aufrufen von `LCDsendByte()` in einer Schleife bis zum String-Ende-Zeichen (ASCII '\0')

`Float2String()`

→ Konvertierung eines Gleitkommawertes in einen ASCII-String mit vorgebbarer Anzahl an Nachkommastellen

`Long2String()`

→ In `Float2String()` verwendete Hilfsfunktion

Die zugehörige Headerdatei *lcd.h* beinhaltet neben den Prototypen-Deklarationen der Funktionen aus *lcd.c* zusätzlich diverse Defines, welche die Implementierung des Kommunikationsprotokolls für das LCD-Modul besser lesbar machen. Um beispielsweise das Display zu löschen und den Cursor in die Home-Position zu bringen, muss zum Display-Controller das *Byte* $0x01$ übertragen werden. Selbsterklärender als das C-Statement `LCDsendByte(0x01)` ist jedoch der Aufruf der Funktion mit einem entsprechenden Define, nämlich `LCDsendByte(CLEAR_DISPLAY)`.

Erläuterungen zu den Parameterlisten und Datentypen der einzelnen Funktionen finden sich in den ausführlich mit Kommentaren versehenen Dateien *lcd.c* bzw. *lcd.h* auf dem dieser Masterarbeit beiliegenden Datenträger.

[1] ASCII: American Standard Code for Information Interchange

5.2.2 I²C-Kommunikation mit den Sensor-ICs

Auch die teilweise stark unterschiedlichen I²C-Kommunikationsprotokolle der Sensor-ICs legen die Erstellung einer eigenen C-Bibliothek nahe. Die Triggerung und Abfrage von Messungen durch diese API lässt sich damit übersichtlicher in die zyklischen Tasks des Echtzeit-Betriebssystems integrieren (vgl. Abschnitt 5.4.3).

Die Datei *i2c_sensors.c* beinhaltet die folgenden Funktionen:

`InitI2C()`

- → Aktivierung des Concerto I2C0 Peripherals
- → Zuweisung der Kontrolle über die für I²C verwendeten Pins an den M3-Kern
- → Einstellung der Pinfunktionen für das I²C-Interfacce
  (vgl. Anhang A.3.1 auf Seite 70)
- → Setup des I²C-Master-Moduls mit einer Übertragungsrate von $100\,kHz$

`SHT21SoftReset()`

- → Software-Reset des Sensors SHT21 (Reset bei aktiver Spannungsversorgung)

`SHT21StartMeasurementCycle()`

- → Initialisierung einer Temperatur- oder einer Feuchtemessung
  (Messwerttyp wird als Parameter übergeben)

`SHT21GetAcquiredData()`

- → Sequentielles Lesen von zwei *Byte* eines Temperatur- oder eines Feuchtewerts vom Ausgangsregister des SHT21 (12 bzw. $14\,bit$-Information, vgl. Abschnitt 2.2.2)
- → Speicherung der gelesenen Rohdaten in einem Datenpuffer

`SHT21CalcTemperature()` und `SHT21CalcHumidity()`

- → Umrechnung der eingelesenen Rohdaten des SHT21 in Gleitkommawerte nach den Gleichungen 2.3 und 2.2 (siehe Seiten 8, 7) für Displayausgaben

`HDI0611GetAcquiredData()`

→ Sequentielles Lesen von zwei *Byte* der aktuellen Druckmessung vom Ausgangsregister des HDI0611 (15 *bit*-Information, vgl. Abschnitt 2.2.1)

→ Speicherung der gelesenen Rohdaten in einem Datenpuffer

`HDI0611CalcPressure()`

→ Umrechnung der eingelesenen Rohdaten des Drucksensors in einen Gleitkommawert nach Gleichung 2.1 (siehe Seite 7) für Displayausgaben

`MPU6050Configure()`

→ Setzen der Bits im Power Management Register des MPU6050 zum Beenden des Sleep-Modus, zur Deaktivierung des integrierten Temperatursensors und für deiWahl der sensorinternen Taktquelle als Referenz für das X-Achsen-Gyroskop.

→ Einstellung im allgemeinen MPU6050 Konfigurationsregisters für die Deaktivierung der Synchronisation und zur Wahl einer Eckfrequenz des digitalen Tiefpassfilters von $5\,Hz$

→ Einstellung von Messbereichsgrenzen des Gyroskops bei $\pm 250\,\frac{°}{s}$

→ Deaktivierung des digitalen Hochpassfilters des Akzelerometers und Wahl von Messbereichsgrenzen bei $\pm 4\,g$

`RequestXYZData()`

→ Festlegung der zum folgenden Abfragezyklus im I^2C-Ausgangsregister des HMC6343 oder des MPU6050 bereitzustellenden Messwertetripel
(Sensoradresse und Messwerttyp sind Parameter; vgl. Tabelle 3.1 auf Seite 23)

`GetXYZData()`

→ Sequentielles Lesen von drei mal zwei *Byte* aus dem Ausgangsregister des HMC6343 oder des MPU6050 (Sensoradresse ist Funktionsparameter)

→ Speicherung von jeweils zwei *Byte* für X-, Y- und Z-Achse bei Magnetfeld-, Beschleunigungs- und Drehratenmessungen, bzw. für Gier, Nick und Roll bei Messungen der Ausrichtung, in einem eigenen Pufferarray

`TwosComplement2Float()`

→ Umwandlung der von den ICs HMC6343 und MPU6050 gelieferten Rohdaten aus der Zweierkomplements- in eine Gleitkommadarstellung

In der Headerdatei *i2c_sensors.h* sind neben den Prototypen-Deklarationen der Funktionen aus *i2c_sensors.c* auch etliche Defines für das einfachere Lesen des Kommunikationsprotokolls im C-Code beinhaltet. Nähere Erläuterungen zu den Parameterlisten und Datentypen der einzelnen Funktionen finden sich in den ausführlich mit Kommentaren versehenen Dateien *i2c_sensors.c* bzw. *i2c_sensors.h* auf dem dieser Masterarbeit beiliegenden Datenträger.

5.2.3 Framestrukturierung und -organisation

Die Grundgerüste der drei möglichen DTP-Frametopologien für die Datenübertragung an HAILscout (vgl. Abschnitt 5.4.2) sind in Form von Strukturvariablen in der Datei *eth_com.h* hinterlegt. Dadurch können in der die Hauptanwendung implementierenden Quelldatei, *HAILsensMAINapp.c*, unter Inkludierung dieser Headerdatei globale Variable für die Datenpufferung und -übertragung angelegt werden, ohne die Quelldatei mit Definition der Strukturvariablen überfrachten zu müssen. Zusätzlich beinhaltet *eth_com.h* Defines für DTP-Frame- und -Subframe-Größen, DTP-Referenz-IDs und Error-Tags für das selbsterklärende Error-Handling in den entsprechenden Subroutinen [2].

5.3 Hilfsfunktionen im Hauptprogramm

Neben den oben aufgeführten und in die Bibliotheksdateien ausgelagerten APIs für Kommunikationszwecke existieren in der Hauptanwendung *HAILsensMAINapp.c* einige Hilfsfunktionen für allgemeine Initialisierungszwecke, sowie für die Berechnung der Prüfsummen in den DTP-Headern und das Einsetzen der Command Layer, Data Layer und Frame Layer Frames in den Übertragungspuffer (vgl. Anhang A.3.3 auf Seite 72). Diese Hilfsroutinen haben im Detail die folgenden Aufgaben:

`InitGlobals()`

Initialisierung der in der Hauptanwendung benutzen globalen Variablen für

→ Anzeige- und Umrechnungszwecke in Zusammenhang mit dem LCD-Modul

→ DTP-Frame-Pufferung

→ Speichermanagement

`InitButtonInterrupt()`

- → Spezifizierung von Port D, Pin 4 als Push-Pull-Eingang mit kleinem Pull-Up-Widerstand (vgl. Anhang A.3.1)
- → Aktivierung des Interrupts auf diesem Pin mit Triggerung auf die fallende Flanke
- → Deaktivierung der Interrupts auf den übrigen Port D Pins
- → Löschen aller Interrupts in der Pipeline für Pin 4

`InitTXbuffer()`

- → Initialisierung der Elemente des Sendepuffer-Arrays mit $0x00$

`InsertFLheader()` und `InsertCLheader()`

- → Einsetzen des Frame Layer bzw. Command Layer Headers in den Übertragungspuffer
- → Summierung der *Byte* im jeweiligen Header
- → Rückgabe der Anzahl der geschriebenen *Byte*

`InsertDLframeMPU6050()`, `InsertDLframeHMC6342()`, `InsertDLframeHDI0611()` und `InsertDLframeSHT21()`

- → Einsetzen der Date Layer Frames in den Übertragungspuffer (falls *Error*-Tag gesetzt, leeren Frame erzeugen)
- → Summierung der *Byte* im jeweiligen Frame
- → Rückgabe der Anzahl der geschriebenen *Byte*

`InsertCalcHeaderInfo()`

- → Einsetzen des aktuellen CountStamps in den Command Layer Header
- → Einsetzen von SizeFrameData und CountDataItem in den Frame Layer Header
- → CheckSum berechnen und in den Frame Layer Header einsetzen

5.4 Echtzeit-Scheduling und Timing

Wie in Abschnitt 4.3.1 bereits ausgeführt, nutzt die HAILsensRTapp das Echtzeit-Betriebssystem SYS/BIOS. Die dadurch erreichte Erleichterung einer exakten zeitlichen Ablaufplanung durch den prioritätsgesteuerten, präemptiven Scheduler wird in diesem Abschnitt erläutert. Grundlage für die Planung ist eine systematische Zuweisung von Ausführungsprioritäten an Subroutinen (allgemein Thread bzw. Task). Die verschieden priorisierten, für die Implementierung der HAILsens-Funktionalitäten genutzten Task-Kategorien (reiner Task, SWI[2], Clock-Funktion bzw. Clock-ISR, HWI[3]) sind mit einer gewählten Auflösung des System-Ticks von $1\,ms$ für die Anwendung ausreichend zeitdeterministisch. Eine Übersicht der am Echtzeit-Scheduling teilnehmenden Tasks ist in Tabelle A.2 im Anhang auf Seite 74 gegeben [44].

Die Zuweisung der Prioritätslevel erfolgt über den grafischen XGCONFIG-Editor von CCS, der sich nach Doppelklick auf den Dateinamen *HAILsensCONFIG.cfg* im Project Explorer öffnet (vgl. Abschnitt 4.3, Seite 32 ff.). Diese Konfiguration eines Tasks kann nach Doppelklick auf den in Abbildung 4.4 dargestellten *SYS/BIOS*-Block mit anschließender Auswahl eines den Task-Kategorien zugeordneten *Thread*-Blocks durchgeführt werden (vgl. Abbildung 4.5). Hier sind auch andere, teilweise optionalen Einstellungen wie die Task-Stackgröße, eine Task-Handle, die Periode T des Timer-Interrupts für Glock-ISRs oder der Thread-Kontext einstellbar (vgl. Abbildung 5.1 auf Seite 47). Der Task ist damit in das RTOS SYS/BIOS eigebettet.

5.4.1 Task

Für Task-Funktionen sind in SYS/BIOS die niedrigsten 16 Prioritätslevel reserviert. Gewählt werden können Levelwerte von 1 bis 15 für aktive Tasks. Inaktiven Tasks haben die Priorität –1. Die Level liegen unter denen von SWIs, Clock-ISRs und HWIs. Es gibt nur eine einzige reine Task-Funktion im HAILsens-RTOS. Die Subroutine `tsk0FxnIOhandler()` hat die niedrigste Priorität (Level 1) im Scheduling-Prozess.

Abbildung A.4 auf Seite 75 im Anhang A.3.5 zeigt den PAP [4] dieses Tasks, der für die Steuerung der Datenspeicherung und der Ethernet-Kommunikation zuständig ist. Solange kein Task mit höherer Priorität für die Ausführung bereit ist, läuft nach einer einmaligen Initialisierungssequenz für den Kommunikationsaufbau mit der SD Karte und mit dem UDP-Socket die darauf folgende Endlosschleife (`while(TRUE)`) immer im Hintergrund der Hauptanwendung (vgl. Anhang A.3.4, Seite 73). Im Falle eines Interrupts höherer Priorität (Timer-, Software- oder Hardware-Interrupt) speichert der Scheduler den Prozessorkontext und arbeitet zuerst die zugehörige Service-Routine ab. Auch geschachtelte Interruptverarbeitung ist möglich. Anschließend wird der Prozessorkontext wieder geladen und die Ausführung von `tsk0FxnIOhandler()` fortgesetzt.

[2] SWI: Software-Interrupt Service-Routine
[3] HWI: Hardware-Interrupt Service-Routine
[4] PAP: Programmablaufplan

5.4.2 SWIs

SYS/BIOS verwendet für Software-Interrupt Service-Routinen ebenfalls 16 verschiedene Prioritätslevel. Innerhalb der SWIs können die Levelwerte 0 bis 15 vergeben werden, was auf den gesamten Scheduling-Prozess bezogen eine Priorität von 16 bis 31 bedeutet. Diese Prioritäten sind höher als die von Task-Funktionen und niedriger als die der Clock-ISRs und HWIs, wobei Clock-ISRs einen Sonderfall der SWIs darstellen (vgl. Abschnitt 5.4.3) [44].

Die beiden in diesem Kontext in der HAILsensRTapp vorkommenden SWIs, `swi0FxnBuildTXframe()` (Priorität 18) und `swi1FxnLCD()` (Priorität 16), beinhalten jeweils einen in Form eines `switch-case`-Konstrukts implementierten Zustandsautomaten. Getriggert werden die ISRs von den in Abschnitt 5.4.3 noch näher beschriebenen Clock-Funktionen. Dadurch ist auch die Ausführung der SWIs zyklisch und mit einer Auflösung von $1\,ms$ exakt zeitdeterministisch. Die folgenden beiden Unterpunkte beschreiben die Funktionalität dieser Service-Routinen genauer.

`swi0FxnBuildTXframe()`

Wie bereits in Abschnitt 3.3.3 angedeutet, werden die von den Sensor-ICs abgefragten und in der Concerto MCU gepufferten Messdaten für die Kommunikation mit der zentralen Verarbeitungseinheit HAILscout vor deren Übertragung in DTP-Frames arrangiert. Das DTP-Protokoll besteht aus drei Schichten (Layer), jeweils mit einem eigenen Satz an Header-Information pro Layer (vgl. Abbildung A.2 auf Seite 72 im Anhang). Die layerspezifischen Error-Tags, Referenz-IDs und Prüfsummen müssen zur Laufzeit berechnet und dann an der dafür vorgesehenen Stelle in die Layer-Header eingefügt werden. Realiesert sind diese Funktionalitäten in der Software-ISR `swi0FxnBuildTXframe()`.

Die Erstellung eines neuen Frames wird mit dem Statement `Swi_post(swi0)` in der Clock-Funktion `clk0FxnMPU6050GetData()` alle $100\,ms$ getriggert (vgl. Abschnitt 5.4.3). Die Task-Handle `swi0` ist dem Task `swi0FxnBuildTXframe()` über die Datei *HAILsensCONFIG.cfg* zugeordnet. Die Abfrage der vier Sensor-ICs findet mit unterschiedlichen Raten statt. Die höchste Rate bzw. das kürzeste Abfrageintervall von $100\,ms$ gilt für den Datentransfer vom Bewegungssensor MPU6050. Die Datenabfrage selbst ist in der zyklisch aufgerufenen Clock-Funktion `clk0FxnMPU6050GetData()` umgesetzt (siehe Abschnitt 5.4.3), weshalb diese Funktion auch den der ISR `swi0FxnBuildTXframe()` zugeordneten Interrupt triggert. Das nächstgrößere Intervall von $200\,ms$ gilt für Abfrage der Lage- und Ausrichtungsdaten des HMC6343, gefolgt von der Aufnahme der meteorologischen Daten des SHT21 und des HDI0611 mit jeweils $1000\,ms$ (vgl. dazu auch Abbildung A.3 und Tabelle A.2 auf Seite 73 im Anhang). Daraus ergeben sich drei mögliche Konstellationen von Data Layer Frames in einem DTP-Frame, die in einer zyklisch wiederholten Sequenz erstellt werden. In Tabelle sind die Einzelelemente dieser Sequenzierung zusammengefasst.

Sensordaten im Frame	Größe $[Byte]$	Erstellungsintervall	Sequenzschritt
MPU6050	72	$1000\,ms$	1
HMC6343			
SHT21			
HDI0611			
MPU6050	56	$200\,ms$	3,5,7, und 9
HMC6343			
MPU6050	39	$100\,ms$	2,4,6,8 und 10

Tabelle 5.1: Sequenzierung der Frame-Erstellung in `swi0FxnBuildTXframe()`

Dem in *eth_com.h* hinterlegten `enum`-Varablentyp `FRAME_NUMBER`, welcher die Werte `NoFrame`, `Frame01`, `Frame02`, ... , `Frame10` annehmen kann, sind die Nummern der zehn Sequenzschritte zugeordnet. Der Wert `NoFrame` ist für Initialisierungszwecke und den Fall eines I^2C-Übertragungsfehlers reserviert. Die übrigen zehn Werte dienen der Steuerung des Zustandsautomaten in `swi0FxnBuildTXframe()`. Da `FRAME_NUMBER` einem eigenen Variablentyp entspricht (wie `int` oder `float`), muss in der Hauptanwendung *HAILsensMAINapp.c* zuerst eine Variable diesen Typs angelegt werden. Der Name dieser Variablen ist `FrameNumber`. Daraus ergibt sich das einleitende Statement `switch(FrameNumber)` für die `switch-case`-Anweisung.

Abbildung A.5 auf Seite 76 im Anhang A.3.5 zeigt den kompletten Ablauf der Software-ISR `swi0FxnBuildTXframe()` am Beispiel des `case Frame03` in Form eines PAPs. Die Initialisierung der lokalen `static`-Variablen ist in dem Ablaufplan nicht berücksichtigt. Nach der Fertigstellung eines Frames im Ethernet-Sendepuffer wird die Anzahl der im Puffer befindlichen *Byte* also Indikator für die Sendebereitschaft der Sensordaten in einer globalen Variablen gespeichert. Ebenfalls als Indikator, jedoch für die Speicherorganisation, inkrementiert die Software-ISR ggf. noch die Nummer der **.rob*-Datei (vgl. auch Abbildung A.4 und Abschnitt 5.5).

`swi1FxnLCD()`

Die Aufbereitung und Anzeige der Messdaten wird ebenfalls alle $100\,ms$ in der Clock-Funktion `clk0FxnMPU6050GetData()` getriggert mit dem SYS/BIOS-Aufruf `Swi_post(swi1)`. Das Updaten der am Display angezeigten Messwerte (vgl. Abschnitt 2.2) mit den in Tabelle 5.1 angegebenen Raten genügt dadurch Echtzeit-Anforderungen. Der `enum`-Variablentyp `DISPLAY_STATE` ist in der Header-Datei *lcd.h* hinterlegt. Er kann die Werte `Off`, `Meteo`, `Heading`, `Mag`, `Accel` und `Rot` annehmen. Die globale Variable `LCDstate` vom Typ `DISPLAY_STATE` wird für das Umschalten des Anzeigestatus benutzt (`switch(LCDstate)`). Die in Abschnitt 5.4.4 noch beschriebene, von einem Taster getriggerte Hardware-ISR inkrementiert `LCDstate`.

Der PAP in Abbildung A.6 auf Seite 77 im Anhang beschreibt den Ablauf der SWI exemplarisch für den Übergang von ausgeschaltetem Display auf die Anzeige der meteorologischen Daten, also den `case Meteo`.

5.4.3 Clock-Funktionen

Alle Messungen sollen in exakten Zyklen wiederholt werden. Für diesen Zweck bieten sich die sog. Clock-Funktionen von SYS/BIOS an. Für diesen periodischen Tasktyp kann durch Editierung der Datei *HAILsensCONFIG.cfg* eine Task-Handle, ein einmaliges Timeout beim Programmstart und die Zykluszeit in *ms*, sowie Argumente der Funktion gewählt werden (optional). Abbildung 5.1 zeigt die Kofigurationsoptionen im XGCONFIG-Editor am Beispiel der Clock-Instanz `clk0FxnMPU6050GetData()`.

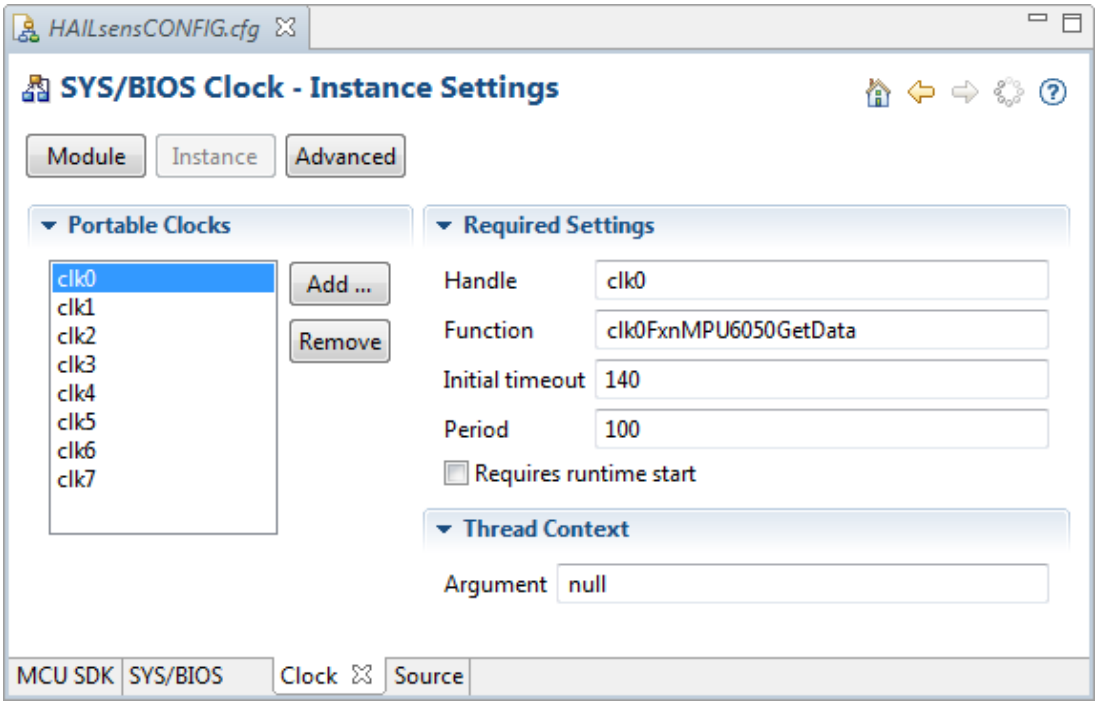


Abbildung 5.1: Konfiguration der Clock-Instanz `clk0FxnMPU6050GetData()`

Für alle in der HAILsensRTapp benutzten Clock-Funktionen sind diese Einstellungen ähnlich. Eine Übersicht dazu findet sich in Tabelle A.2 auf Seite 74 im Anhang. Den einzigen grundlegenden Unterschied gibt es bei einer Einstellung von `clk7FxnHWIdebounce()`. Diese Funktion dient dem softwaremäßigen Entprellen des Tasters für das Umschalten der LCD-Anzeige. Der Taster-Interrupt ist ein asynchrones, nicht periodisches Ereignis. Deshalb ist im Feld *Period* der Wert 0 eingetragen. Die Hardware-ISR `hwi0FxnButtonPress()` (vgl. Abschnitt 5.4.4) schaltet Interrupts am MCU-Pin, an dem der Taster angeschlossen ist ab und startet einen Timer. Nach 200 *ms* läuft der Timer ab, was die Ausführung von `clk7FxnHWIdebounce()` anstößt. Hier werden Interrupts dann wieder global aktiviert. Die während eines einzelnen Tasterdrucks mehrfach auftretenden fallenden Flanken durch unsauberes Schließen des elektrischen Kontakts können dadurch keine erneuten Interrups mehr auslösen und ein unbeabsichtigtem Mehrfachumschalten des Displays wird eliminiert.

Wie bereits am Anfang von Abschnitt 5.4.2 erwähnt, stellen Clock-ISRs einen Sonderfall der SWIs dar. Clock-ISRs haben innerhalb der SWIs die höchste Priorität von 31. Sie sind alle gleich priorisiert, wodurch die ISRs sich nicht gegenseitig aus dem Prozessor verdrängen können. Dadurch kann eine sichere Abarbeitung der Routinten gewährleistet werden. Lediglich der in Abschnitt 5.4.4 beschriebene, relativ selten vorkommende Hardware-Interrupt hat eine höhere Priorität.

Auch die Abläufe der für die Sensor-Datenabfrage eingesetzten Funktionen mit den Handles `clk0` bis `clk7` ähneln sich weitestgehend. Da die Temperatur- und Luftfeuchtemessungen des SHT21 relativ viel Zeit in Anspruch nehmen (vgl. Anschnitt 2.2.2), wurde hier jeweils das Anstoßen einer Messung und die eigentliche Datenabfrage auf zwei Clock-ISRs aufgeteilt. In `clk3FxnSHT21StartTempMeas()` und in `clk3FxnSHT21StartHumiMeas()` gibt es deshalb nur ein einziges C-Statement, und zwar der Aufruf von `SHT21StartMeasurementCycle()` aus *i2c_sensors.c*. Des weiteren ist in `clk0FxnMPU6050GetData`, einer Funktion mit der kürzesten Ausführungsperiode von $100\,ms$, sowohl das Inkrementieren der globalen Variable `usCountStamp` aus einem DTP-Header, von `FrameNumber` für die DTP-Frame-Erstellung, als auch die Triggerung der in Abschnitt 5.4.2 aufgeführten Software-ISRs implementiert. Eine weitere Besonderheit ist das Togglen der auf der controlCARD befindlichen roten LED in `clk6FxnSHT21GetHumiData()` mit einer Periode von $1000\,ms$, sozusagen als Anzeige für den Herzschlag des Systems HAILsens.

Abbildung A.7 auf Seite 78 im Anhang zeigt einen exemplarischen PAP für die Clock-Funktion `clk0FxnMPU6050GetData`.

5.4.4 HWI

Als einzige, explitit für die Behandlung eines Hardware-Interrupts programmierte Service-Routine, hat `hwi0FxnButtonPress()` die folgenden Aufgaben:

- → Deaktivierung der Interrupts an Port D, Pin 0
- → Löschen vorhandener Interrupt-Requests für Port D, Pin 0
- → Umschalten des Display-Status
- → Starten des $200\,ms$-Timers für die Triggerung von `clk7FxnHWIdebounce()` zum Entprellen des Tasters
- → Posten eines Interrupts für `swi1FxnLCD()`, damit die Änderung des Displaystatus frühestmöglich übernommen wird

Abbildung A.8 auf Seite 79 zeigt den Ablauf der Funktion übersichtlicher in Form eines PAPs. Wie bereits oben erwähnt, werden die Interrupts für Port D, Pin 0 in der Clock-Funktion `clk7FxnHWIdebounce()` wieder reaktiviert.

5.5 Überblick Datenmanagement

In Abbildung 5.2 sei für dieses Kapitel abschließend noch einen Überblick zum Fluss der vom Subsystem HAILsens aufgenommenen Messdaten gegeben.

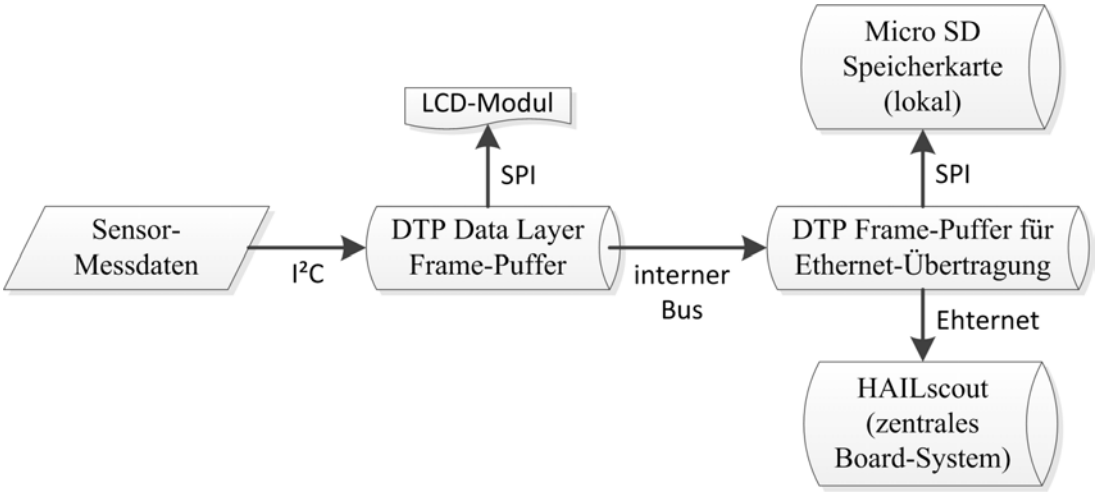


Abbildung 5.2: Datenfluss Im Subsystem HAILsens

6 Systemtests

Dieses Kapitel erläutert die Verifikation der verschiedenen System-Aspekte. Geprüft wurden unter anderem die Sensor-Messdaten auf Plausibilität, das Timing des geplanten Echtzeit-Taskings, die Ethernet-Übertragung sowie die auf der SD Karte gespeicherten DTP-Frames auf Vollständigkeit.

6.1 Verifikation der Sensor-Messdaten

Nicht für alle Sensoren stehen geeignete Messmittel zur Verfügung, um die gelieferten Messwerte wissenschaftlich exakt beurteilen zu können. Die Messungen wären mit höherer Genauigkeit als die Originaldaten im selben Milieu durchzuführen. Eine derartige Verifikation ist nicht Teil der Prototypenentwicklung von HAILsens. Eine Größenordnungsmäßige Betrachtung durch alternative Messungen und Referenzwerte ist im Rahmen der Entwicklungsaufgabe jedoch durchgeführt worden. Es ist in der Regel außerdem davon auszugehen, dass die qualitativ hochwertigen und deshalb auch relativ teuren Sensor-ICs Werte liefern, die sich in einem in den jeweiligen Datenblättern angegebenen Bereichen bewegen.

6.1.1 Meteorologische Daten

Die von den Sensor-ICs HDI0611 und SHT21 gelieferten Messwerte (vgl. Abschnitte 2.2.1 und 2.2.2) konnten in zufriedenstellender Weise verifiziert werden. Einerseits kommen die Daten von der seriös wirkenden privaten Wetterstation in Rosenheim/Pösling [7]. Hier werden Wetterdaten seit dem Jahr 2007 aufgezeichnet und archiviert. Die archivierten Wetterdaten sind monats- oder jahreweise käuflich erwerbbar. Auf der Homepage des Betreibers können die aktuellen Daten, minütlich upgedatet, beobachtet werden. Nachdem HAILsens sich am 22.09.2012 ab 9:10 Uhr in Rosenheim/Pang etwa eine Stunde an der frischen Luft auf die klimatischen Außenbedingungen einstellen konnte, wurden die in Abbildung gezeigten Messwerte für Temperatur, relative Feuchte und barometrischen Druck am Display angezeigt.

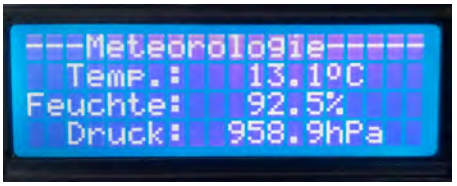


Abbildung 6.1: HAILsens-Wetterdaten vom 22.09.2012, 10:10 Uhr, gemessen in Pang

Etwa zur gleichen Zeit lieferte die $1,5\,km$ entfernte Wetterstation in Pösling die Daten in Abbildung 6.2. Die Vergleichswerte sind in der Darstellung blau umrandet.

Aktuelle Wetterwerte vom 22. September 2012 um 10:09 Uhr

	Temperatur	Taupunkt	Wind	Luftdruck	Niederschlag	Globalstrahlung	Sonne
aktuell	12.6 °C	11.1 °C (91% r.F.)	8 km/h aus SSO Böen bis 10 km/h	958.3 hPa 1013.9 hPa (NN)	24h: 12.7 mm 1h: 3.6 mm	40 W/m²	Nein
heute	Mittel 12.7 °C Max 13.9 °C Min 11.7 °C	Mittel 9.9 °C	Mittel 6.3 km/h Max 29 km/h	Max 958.4 hPa Min 954.6 hPa	Summe 12.7 mm Max 0.2 mm/min	Summe 0.06 kWh/m² Max 49 W/m²	0.0 Std.
Grafiken	letzte 24 Std. letzte 7 Tage	letzte 24 Std. letzte 7 Tage	letzte 24 Std. letzte 7 Tage	letzte 24 Std. letzte 7 Tage	letzte 24 Std. letzte 7 Tage	letzte 24 Std. letzte 7 Tage	

Abbildung 6.2: Homepage der Wetterstation Pösling, 22.09.2012, 10:09 Uhr

Die relativen Abweichungen dieser Daten vom jeweiligen HAILsens Messwert in Prozent werden in den Gleichungen 6.1 bis 6.3 ausgedrückt.

$$\Delta T_{rel} = \frac{T_{HAILsens} - T_{Station}}{T_{HAILsens}} \cdot 100\,\% = \frac{13.1\,°C - 12,6\,°C}{13.1\,°C} \cdot 100\% = 3,817\,\% \quad (6.1)$$

$$\Delta RF_{rel} = \frac{RF_{HAILsens} - RF_{Station}}{RF_{HAILsens}} \cdot 100\,\% = \frac{92,5\,\% - 91.0\,\%}{92,5\,\%} \cdot 100\% = 3,053\,\% \quad (6.2)$$

$$\Delta P_{rel} = \frac{P_{HAILsens} - P_{Station}}{P_{HAILsens}} \cdot 100\,\% = \frac{958,9\,hPa - 958,3\,hPa}{958,9\,hPa} \cdot 100\% = 0,063\,\% \quad (6.3)$$

Hinsichtlich der vorhandenen Messunsicherheiten, sowie den $1,5\,km$ entfernten Messorten sind die relativen Abweichungen sehr klein, bzw. die Übereinstimmungen der mit den unterschiedlichen Messmitteln aufgenommenen Wetterdaten sehr groß. Die Genauigkeit der von der Wetterstation eingesetzten Messgeräte ist nicht bekannt, wird jedoch als recht hoch eingeschätzt.

Plausibilitätsbetrachtungen der von HAILsens gelieferten Temperatur- und Druckwerte wurden gleichzeitig auch mit einem handelsüblichen, niedrigpreisigem Zimmer-Thermo-Hygrometer der Firma TFA Dostman (Modell *Style*) im selben, oben beschrieben Milieu durchgeführt. Die Übereinstimmung der Messwerte war hierbei erwartungsgemäß etwas geringer als bei der Wetterstation Pösling.

6.1.2 Elektronischer Kompass

Der Prüfung für die von HAILsens ausgegebenen Nick- und Rollwinkel (vgl. Abschnitt 2.2.3) dient ein Stellwinkel, auf dem der Sensor HMC6343 platziert werden kann. Für $\varphi_{Stell} = 45{,}0°$ und einen Sensor-IC, der so montiert ist, dass seine Y-Achse (vgl. Abbildung 2.6) mit der Achse der beiden Schenkel des Stellwinkels parallel verläuft, zeigt das Display die in Abbildung 6.3 dargestellten Werte.

Abbildung 6.3: Ausgegebener Nickwinkel bei $\varphi_{Nick_{Stell}} = 45{,}0°$

Ist die X-Achse des Sensors parallel zur Stellwinkel-Achse orientiert, werden die folgenden Werte angezeigt:

Abbildung 6.4: Ausgegebener Rollwinkel bei $\varphi_{Roll_{Stell}} = 45{,}0°$

Dies entspricht den in den Gleichungen 6.4 und 6.5 angegebenen relativen Abweichungen vom jeweiligen HAILsens-Messwert.

$$\Delta\varphi_{Nick_{rel}} = \frac{\varphi_{Nick_{HAILsens}} - \varphi_{Nick_{Stell}}}{\varphi_{Nick_{HAILsens}}} \cdot 100\,\% = \frac{45{,}6° - 45{,}0°}{45{,}6°} \cdot 100\% = 1{,}316\,\% \quad (6.4)$$

$$\Delta\varphi_{Roll_{rel}} = \frac{\varphi_{Roll_{HAILsens}} - \varphi_{Roll_{Stell}}}{\varphi_{Roll_{HAILsens}}} \cdot 100\,\% = \frac{45{,}4° - 45{,}0°}{45{,}6°} \cdot 100\% = 0{,}881\,\% \quad (6.5)$$

Die angegeben, gemessenen Abweichungen des ausgegebenen Nick- und des Rollwinkels vom Montagewinkel sind als relativ gering einzuschätzen. Die Genauigkeit des Stellwinkels ist nicht bekannt. Eine Abschätzung dieser Genauigkeit im Zehntelgrad-Bereich dürfte realistisch sein.

Die Ausgaben des Kompass-Werts für die Abweichung der Ausrichtung vom magnetischen Nordpol der Erde wurden bisher lediglich anhand eines handelsüblichen Kompasses erfolgreich auf Plausibilität geprüft. Orientiert man jedoch des HMC6343 auf den von ihm selbst gemessenen Nord-Nullwinkel, ist eine Beurteilung der ausgegebenen Magnetfeldmessungen in den drei Raumrichtungen möglich. Der Sensor-IC misst dann die folgenden Werte:

Abbildung 6.5: Messung des Erdmagnetfelds

Die Stärke des Erdmagnetfeldvektors in Mitteleuropa wird in [12] mit etwa $48\,\mu T$ angegeben, mit $20\,\mu T$ horizontaler und $40\,\mu T$ vertikaler Komponente. Die von HAILsens gemessenen Werte sind in dieser Größenordnung. Wie auf Seite 11 bereits angedeutet, sollten die Magnetfeld-Daten des HMC6343 kritisch betrachtet werden. Die Untersuchung der während eines Flugeinsatzes von den Subsystemen aufgenommenen Daten ist als experimenteller Ansatz gedacht. Ob die Magnetfelddaten tatsächlich für Forschungszwecke verwendet werden können, wird sich nach den ersten Testflügen mit anschließender Auswertung der aufgenommenen Daten zeigen.

6.1.3 Bewegungssensor

Referenzwerte für die Beschleunigungs- und Drehratendaten des MPU-6050 gibt es bisher nicht. Eine um drei Raumachsen drehbare Aufhängung mit Winkelskalen wurde von Matthias Frey in seiner Diplomarbeit [1] zur Drehratenbestimmung entwickelt und aufgebaut. Die Bestückung der Apparatur mit einem HAILsens-Prototypen steht derzeit noch aus. Auf Plausibilität geprüft können lediglich die Beschleunigungswerte des Sensors für die drei Raumrichtungen. Die bekannte, mittlere Erdbeschleunigung von rund $9,81\,\frac{m}{s^2}$ wird auf allen drei Sensorachsen näherungsweise gemessen, je nachdem, ob sich die X-, Y- oder Z-Achse in senkrechter Orientierung befinden. Abbildung 6.6 zeigt die Display-Ausgabe bei waagrechter Ausrichtung des Sensor-ICs. Der angezeigte Z-Wert weicht weniger als $2\,\%$ vom Erwartungswert ab.

Abbildung 6.6: Beschleunigungsmessung bei waagrechter Montage des MPU-6050

6.2 Timing-Benchmarks

Auf alle durchgeführten Untersuchungen, die das zeitliche Verhalten von HAILsens chrakterisieren, wird in diesem Abschnitt näher eingegangen.

6.2.1 Tastersignal und HWI

Schwierigkeiten gab es mit dem Button zum Umschalten der Messwertgruppen auf dem Flüssigkristall-Display und der damit zusammenhängenden Interrupt Service-Routine. Die im Zustandsautomaten in der Subroutine `swi1FxnLCD()` programmtechnisch festgelegte Sequenz der einzelnen Anzeigegruppen (vgl. Seite 46 unten) wurde vom System zuerst nicht eingehalten. Eine nicht reproduzierbare Anzahl von Sequenzschritten wurde übersprungen. Dies legte die Vermutung nahe, dass ein einzelner Tastendruck mehrere Interrupts in kurzer Folge hintereinander auslöst. Die Untersuchung mit dem digitalen HAMEG-Oszilloskop HMO1024 liefert den in Abbildung 6.7 dargestellten, typischen Signalverlauf beim Drücken des Tasters.

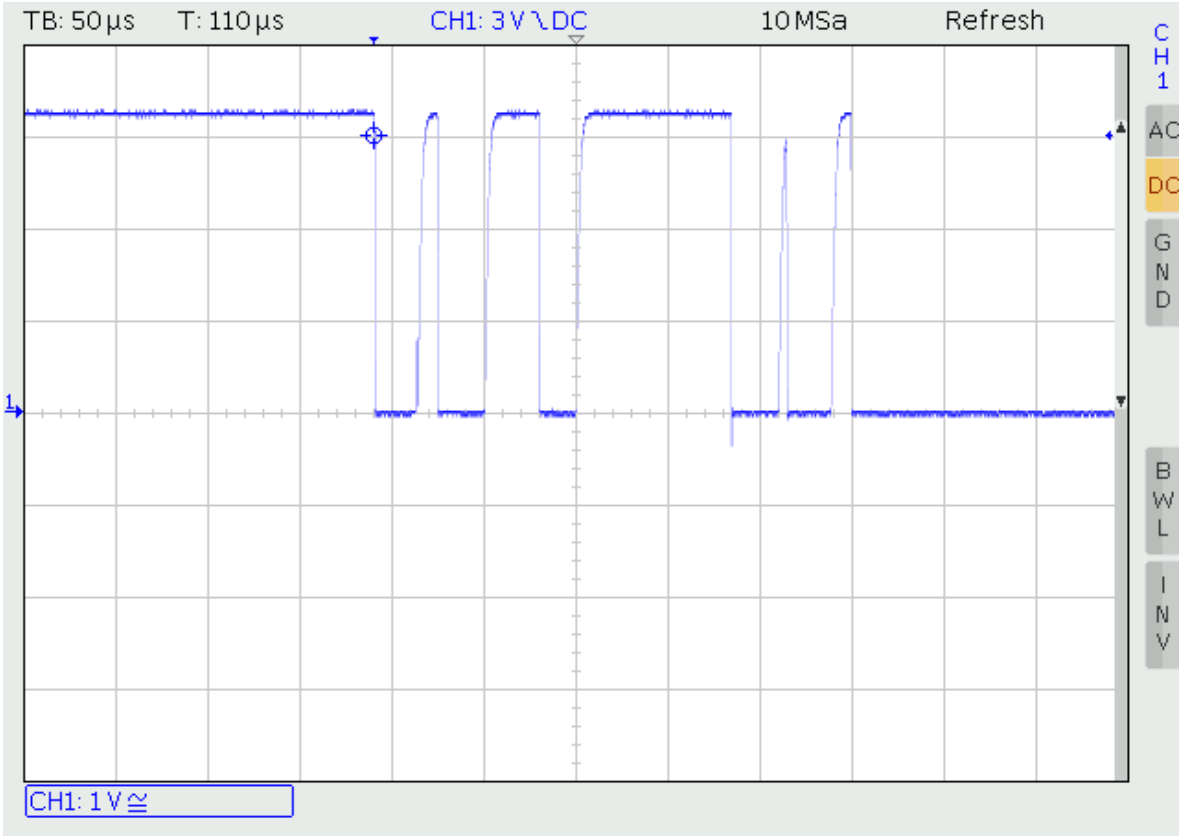


Abbildung 6.7: Signalverlauf am Oszilloskop während eines Tasterdrucks

Das Drücken des Tasters zieht die Spannung am digitalen Eingang des Concerto Controllers von $3,3V$ auf $0V$. Deshalb wurde das Trigger-Level des Oszilloskops auf $3,0V$ gesetzt und auf die fallende Flanke getriggert. Zu sehen ist in Abbildung 6.7, dass während einer Zeit von ungefähr $250\mu s$ nach der ersten fallenden Flanke weitere fünf falende Flanken folgen. Jede dieser insgesamt sechs fallenden Flanken löst am digitalen Eingang der Concerto MCU einen Interrupt aus. Die Anzahl der Interrupts wird gespeichert und die Hardware-ISR (welche den Display-Status inkrementiert, vgl. Abschnitt 5.4.4) dann so oft abgearbeitet, bis die Interrupt-Queue wieder leer ist. Gelöst wird das Problem durch softwaremäßiges Entprellen des Tasters. Dies geschieht mittels globaler Abschaltung von Hardware-Interrupts im Controller durch die ISR `hwi0FxnButtonPress()`. Nach $200ms$ ist das Rücksetzen des Counters und die globale Reaktivierung von Interrupts in dere bereits in Abschnitt 5.4.3 beschriebenen Clock-Funktion `clk7FxnHWIdebounce()` implementiert.

6.2.2 Echtzeit-Tasking

Zur Untersuchungen der Ausführungsdauer von Task-Funktion, der Software-ISRs und Hardware-ISR wurden diverse Hilfsmittel von CCS und SYS/BIOS eingesetzt. Die im Anhang auf den Seiten 73 und 74 diesbezüglich zusammengefassten Informationen sind durch Aufnahme des aktuellen System-Ticks am Beginn und am Ende einer Echtzeit-Subroutine mit der Funktion `Clock_getTicks()` erstellt worden. Die in Tabelle A.2 aufgeführten Werte entsprechen demnach einer System-Tick-Differenz. Die Auflösung des Ticks von $1ms$ ist für die Überprüfung des Scheduling-Ablaufs ausreichend genau. Es stellt sich heraus, das alle Subroutinen über genügend große Zeitschlitze für deren Abarbeitung verfügen. Der Scheduling-Prozess verläuft demnach planmäßig.

6.3 Datenkommunikation

Auf der Micro SD Karte konnte die korrekte Speicherung der DTP-Frames in den **.rob*-Dateien mit Hilfe der Freeware Hex-Editor MX von NEXT-Soft erfolgreich nachgewiesen werden. Jede etwa $4,79KB$ große Datei enthält 100 Frames. Ein neuer Frame wird alle $100ms$ erstellt, eine neue Datei mit eigener Dateinummer alle $10s$. Die Framespeicherung erfolgt lückenlos, was am inkrementierten CountStamp-Wert aufeinanderfolgender Frames einer Datei ersichtlich ist. Abbildung 6.8 zeigt den mit dem Hex-Editor geöffneten Inhalt der von HAILsens erstellten Datei *HAL00003.ROB*. Es ist die vierte, während eines Messeinsatzes erstellte Datei (Zählbeginn bei 0). Das erstellende HAILsens-System sitzt im ersten Hagelabwehr-Flugzeug (A) im linken Flügel (L).

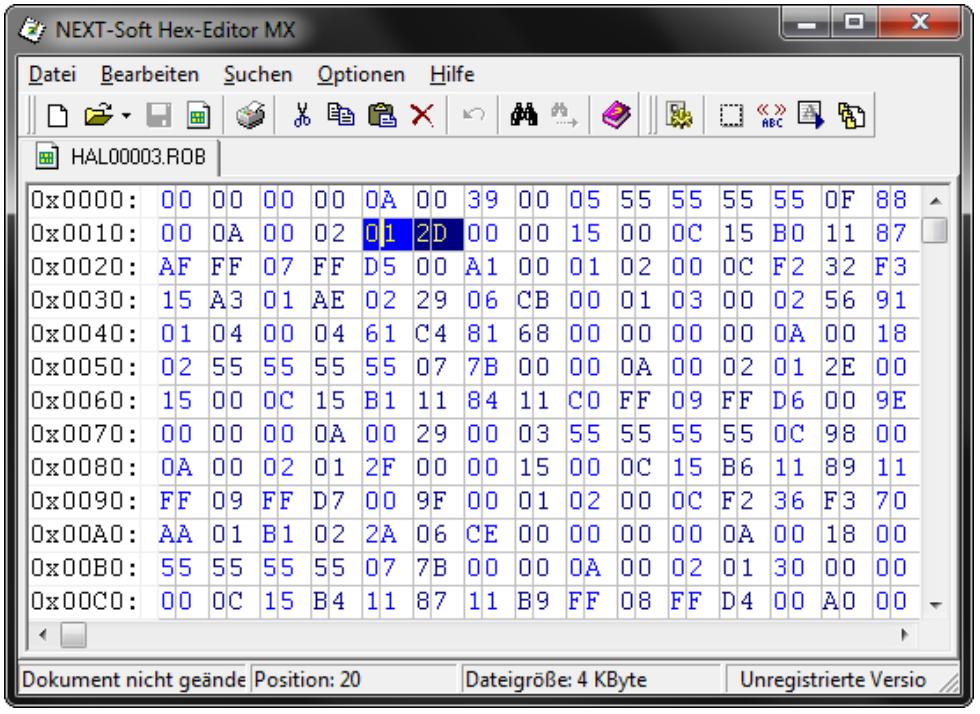


Abbildung 6.8: Auf SD Karte geloggte Daten, untersucht mit Hex-Editor MX

Markiert ist hier der CountStamp (vgl. Anhang A.3.3 auf Seite 72) des ersten DTP-Frames in der Datei der den hexadezimalen Wert $0x012D$ hat, was in dezimaler Notation dem Wert 301 entspricht. Der CountStamp folgt immer $7\,Byte$ nach dem $4\,Byte$ großen DEBUG-Feld mit dem Wert $0x55555555$. $7\,Byte$ nach dem zweiten DEBUG-Feld in Zeile $0x0050$ sieht man den CountStamp des zweiten, in der Datei gespeicherten Frames mit dem Wert $0x012E$, 302 in dezimaler Darstellung. Auch die anderen Headerfelder der DTP-Frames wurden erfolgreich auf Korrektheit geprüft.

Die Analyse der über Ethernet mit UDP übertragenen Daten mit Hilfe der von der HW-group bereitgestellten Freeware Hercules SETUP utility bestätigt die lückenlose und korrekte Übertragung der DTP-Frames. Abbildung 6.9 zeigt eine Beispieldatenaufnahme über die Ethernet-Schnittstelle eines PCs. Die Untersuchung der in den UDP-Datagrammen übermittelten Information gestaltet sich ähnlich wie oben für den Hex-Editor. Orientieren kann man sich auch hier wieder am DEBUG-Feld ($0x55555555$). $7\,Byte$ nach diesem Feld ist jeweils der CountStamp mit den Werten $0xB4$, $0xB5$, $0xB6$ und $0xB7$ eigetragen. Damit ist die lückenlose Übertragung verifiziert, welche zuerst, ohne Verwendung des MCU SDK (vgl. Seite 4.2 f.), große Probleme bereitet hatte.

Auch die RO-BERTA eigene, von Martin Heigl entwickelte Software HAILagent konnte die ordnungsgemäße Übertragung der Sensordaten über Ethernet und UDP verifizieren. HAILagent extrahiert die Rohdaten, berechnet die physikalischen Messwerte und schreibt diese anschließend mit Zeitstempel in eine SQL-Datenbank. Auch hier kommen die Frames lückenlos an. Die in der Datenbank einsehbaren Sensor-Werte für die verschiedenen Messungen sind in Größe und Verlauf plausibel.

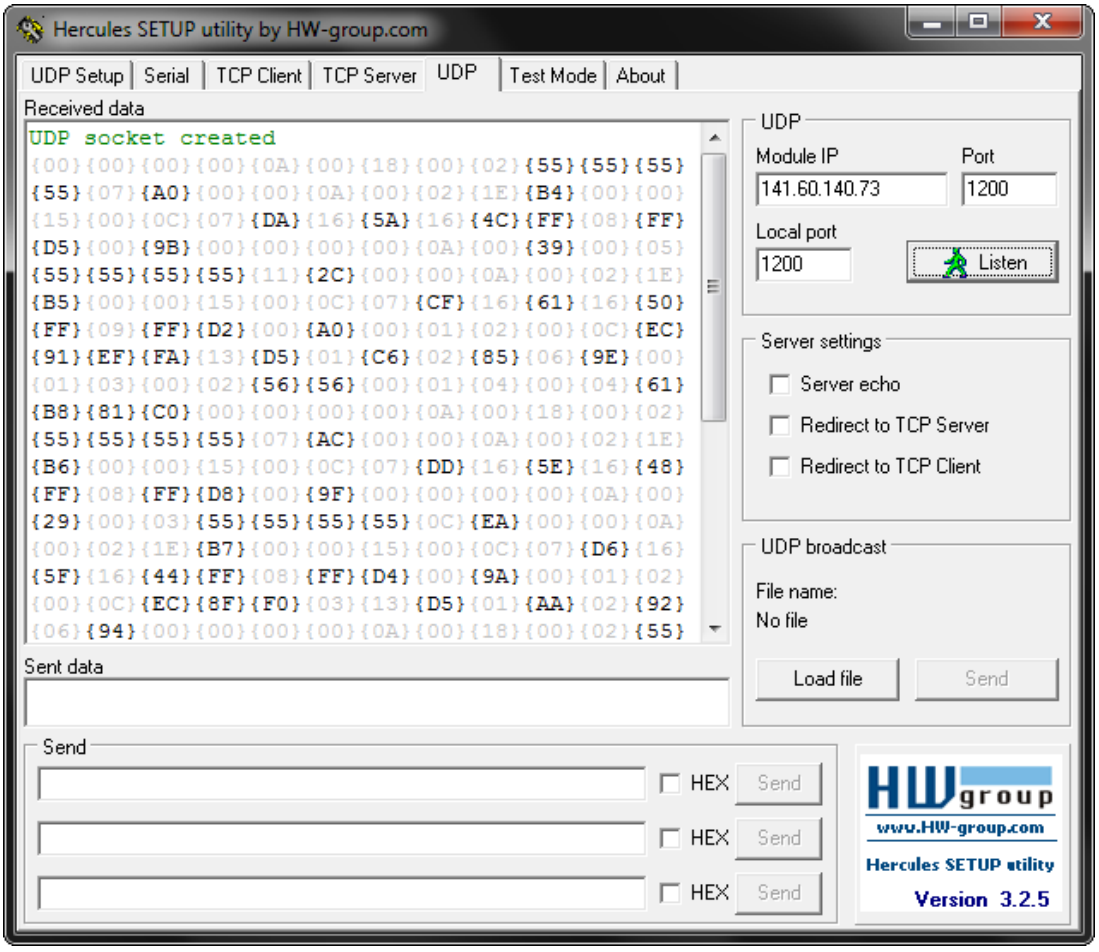


Abbildung 6.9: Untersuchung der Ethernet-Übertragung mit der Hercules SETUP utility

6.4 Betriebssicherheit des Programms

Die Integration der Ethernet-Kommunikation mittels Drittanbieter-Software bzw. TCP/IP-Stack in das TI RTOS SYS/BIOS bereitete zuerst große Schwierigkeiten. Das Programm HAILsenRTapp stürzte nach einer nicht reproduzierbaren Zeit ab. Auch tagelange Debugversuche scheiterten, da CCS nur eine begrenzte Anzahl von Breakpoints zur Verfügung stellt und die extrem verschachtelten Funktionsaufrufe für eine UDP-Übertragung so nicht bis zur Fehlerquelle verfolgt werden können. Der Programm-Pointer zeigte beim Absturz dann an eine leere Speicherstelle. Unter Benutzung des MCU SDK mit TI-eigener Ethernet-API läuft die HAILsenRTapp nun stabil. Mehrere 24-stündige Testläufe verliefen erfolgreich. In Anbetracht der Tatsache, dass ein Hagelflug-Einsatz maximal etwa 3 Stunden dauert, ist die garantierte 24-Stunden-Stabilität des Systems HAILsens mehr als ausreichend.

6.5 Energieverbrauch des Subsystems

Die pro forma Betrachtung der Leistungsaufnahme von HAILsens lieferte bei ausgeschaltetem Display, dies ist die Standard-Betriebsart, einen Wert von $750\,mW$. Ist das LCD-Modul aktiv, werden rund $850\,mW$ vom System aufgenommen. Diese Werte von unter $1\,W$ belasten die Energieversorgungseinheit an bord eines Hagelflugzeugs nicht nennenswert. Trotzdem kann die Leistungsaufnahme von HAILsens wahrscheinlich noch erheblich reduziert werden, indem man die Sleep- und Standby-Modi der einzelnen IC-Komponenten nutzt. Dies bedeutet dann allerding zusätzlichen, nicht zu unterschätzenden Zeit- und Programmier-Aufwand. Im Rahmen der Systementwicklung für das Boardsystem HAIL ist eine energieeffiziente Implementierung irrelevant.

Abbildung 6.10 zeigt den aktuellen Stand des HAILsens Prototyps mit eigener, von Martin Laake gelayouteter Backplane-Platine in einem Demonstrations-Aufbau. Die Concerto controlCARD wird weiterhin verwendet, steckt allerdings nicht mehr in der USB Docking-Station von TI. Der abgebildete Akku ist Stellvertretend für die $12\,V$ Gleichspannungsversorgung an bord des Flugzeugs angeschlossen. Lineare DC/DC-Wandler stellen die vom System benötigten Spannungspegel $3,3\,V$ und $5\,V$ zur Verfügung.

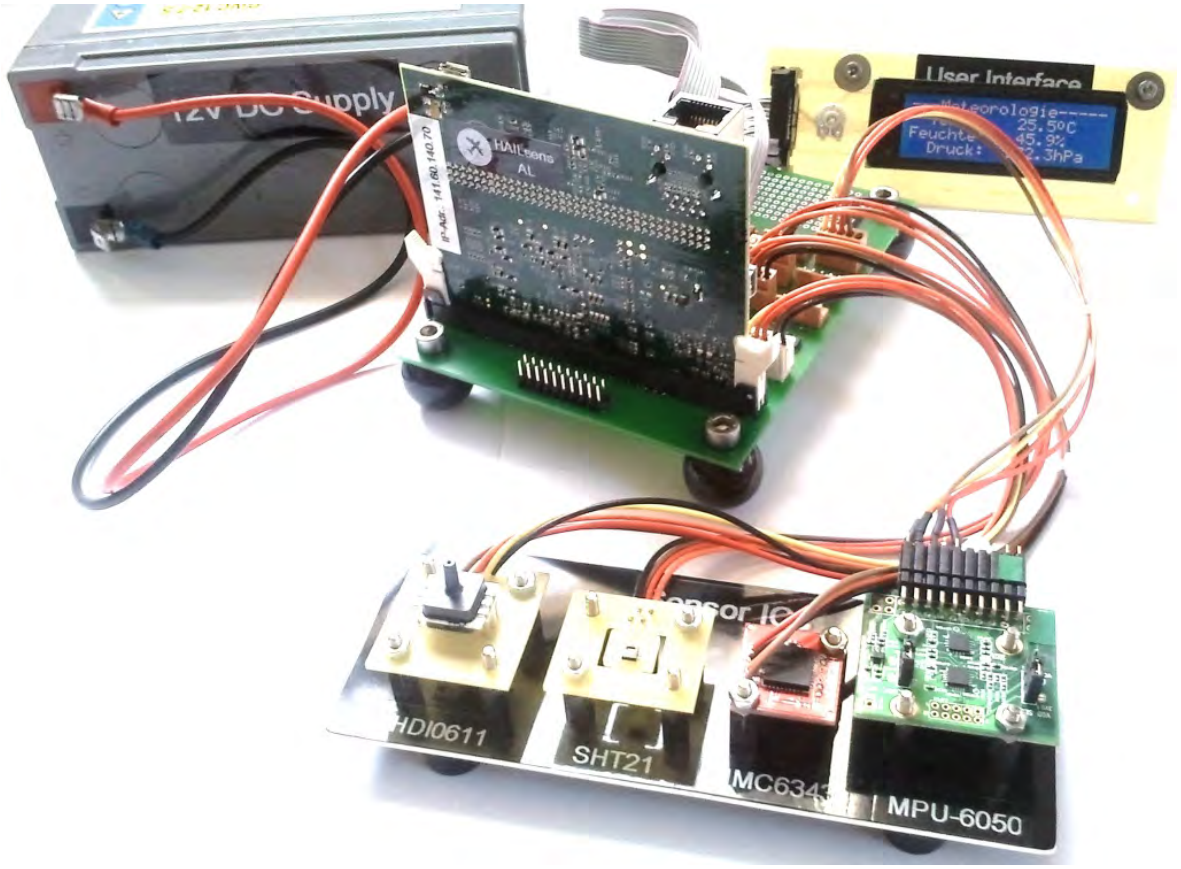


Abbildung 6.10: HAILsens Prototyp, Version 2.0

7 Zusammenfassung und Ausblick

Die Aufgabenstellung der Prototypenentwicklung für das HAIL-Subsystem HAILsens konnte im Rahmen dieser Masterarbeit zufriedenstellend abgeschlossen werden. HAILsens erfüllt alle Punkte der Anforderungsliste. Diese beinhaltet Echtzeit-Fähigkeit, die Minimalimplemtierung eines User Interfaces bestehend aus Taster und Flüssigkristall-Display, die lokale Speicherung der Messdaten auf SD Karte und die Datenübertragung über die Ethernet-Schnittstelle. Die einzelnen Funktionen, sowie die Betriebssicherheit und korrekte Arbeitsweise des Gesamtsystems wurden eingehend mit positiven Ergebnissen geprüft.

Verbessert werden kann, wie oben bereits erwähnt, die Energieeffizienz des Systems, welche aber für die Integration in das Boardsystem HAIL keine Rolle spielt. Auch der Integrationsgrad der Systemkomponenten kann verbessert werden. Durch ein kompakteres Platinenlayout, indem auch die relevanten Komponenten der controlCARD integriert werden, sind die Geräteabmessungen erheblich reduzierbar. Der geschätzte Zeitaufwand hierfür beträgt mehrere Wochen. Der Montageraum in der Heckflosse und den Tragflächen des Flugzeugs bietet HAILsens jedoch auch in seinem gegenwärtigen Zustand genug Platz, wodurch der zeitaufwändige Platinenentwurf nicht notwendig ist und deshalb entfallen kann.

Sehr wichtig ist der Entwurf und die Konstruktion eines adäquaten Gehäuses für die elektronischen Komponenten. Diese Aufgabe sollte nicht unterschätzt werden. Beispielweise muss der Temperatursensor so untergebracht werden, dass er wirklich die Umgebungstemperatur misst und nicht vom Wind-Chill-Effekt beeinflusst wird. Auch der Bewegungssensor ist weitestgehend mechanisch zu entkoppeln, um Erschütterungen oder vom Flugzeugmotor hervorgerufene Vibrationen weitestgehend von der messtechnischen Erfassung auszuschließen. Geplant ist, den Gehäuseentwurf demnächst in eine Bachelorarbeit an der Hochschule Rosenheim auszulagern.

Eine bereichernde Erfahrung war die Teilnahme an der diesjährigen europäischen Konferenz für digitale Signalverarbeitung in Amsterdam am 13. und 14. September 2012, der EDERC2012. Hier gab es im Vorfeld einen so genannten Call for Papers Aufruf. Ein fünfseitiges Manuskript wurde hierfür angefertigt, das die Implementierung und Funktionalität von HAILsens beschreibt. Das Paper wurde angenommen und wird nach einer mündlichen Präsentation auf der Konferenz in der internationalen digitalen Bibliothek IEEE Xplore veröffentlicht. Der Index des veröffentlichten Artikels ist derzeit noch unbekannt, da die Organisatoren der erst etwa eine Woche zurückliegenden Konferenz bisher noch keine Information dazu bereit gestellt haben. Unter dem Titel der Arbeit, SYSTEM FOR REAL-TIME DATA ACQUISITION AND PROCESSING ON A HAIL-FIGHTER AEROPLANE, sollte die Veröffentlichung jedoch innerhalb der nächsten Wochen auf der IEEE Xplore Homepage zu finden sein.

Abschließend ist zu bemerken, das der Erfahrungsgewinn bei der Implementierung der Systemanforderungen in jeder Hinsicht bereichernd ist. Die Programmierung eines aktuellen Microcontrollers von einem der Marktführer in diesem Bereich, die Verwendung eines Echtzeit-Betriebsystems, die Anwendung diverser Schnitstellen-Protokolle und nicht zuletzt die elektrotechnische Einbindung aller Systemkomponenten bringt auch für künftige Entwicklungsaufgaben anderer eingebetteter System enorme Vorteile. Im Moment wird im Labor für Mess- und Regelungstechnik unter anderem an einer Toolchain für das Rapid Prototyping mit TI Delfino und MSP430 Controllern gearbeitet. Dadurch kann direkt aus MATLAB-Code oder Simulink-Modellen eine auf dem jeweiligen DSP lauffähiger Code erzeugt werden. Ein Regelalgorithmus muss so nicht zuerst in hochspezialisierten C-Code umgesetzt werden, wodurch eine erhebliche Zeitersparnis möglich ist. Auch andere Projekte, wie beispielsweise die Errichtung eines interaktiven Netzwerks von Wetterstationen stehen an, bei denen der Einsatz von TI-Hardware geplant ist. Die im Rahmen dieser Masterarbeit gewonnenen Erfahrungen dürften also weiter gewinnbringend eingesetzt werden können.

Literaturverzeichnis

[1] M. FREY:
Integration einer Sensorik zur Messung von Umweltparametern in eine Flugzeugnase
Diplomarbeit, Rosenheim, Oktober 2011

[2] M. HEIGL:
Domino Transfer Protocol
Interne Unterlagen zum RO-BERTA-Kommunikationsprotokoll, Rosenheim, März 2012

[3] M. HEIGL, P. VIEHHAUSER, A. BERNHARDT, P. ZENTGRAF:
Projekt RO-BERTA, Zwischenbericht Oktober 2011 - März 2012
Interne Unterlagen zur Projekt-Dokumentation, Rosenheim, März 2012

[4] S. HEITSIEK:
Das Einsteigerseminar C
1. Auflage, verlag moderne industrie Buch AG & Co. KG, Landsberg 2002

[5] HAGELABWEHR ROSENHEIM:
`http://www.hagelabwehr-rosenheim.de/index.html`
(Homepage des Vereins zur Erforschung der Wirksamkeit der Hagelbekämpfung im Raum Rosenheim e.V.)
aufgerufen am 30.03.2012

[6] HOCHSCHULE ROSENHEIM:
`http://hochfelln.fh-rosenheim.de/~hoegl/roberta/texte/aktuelles.htm`
(Internet-Auftritt des Projekts RO-BERTA)
aufgerufen am 30.03.2012

[7] PRIVATE WETTERSTATION ROSENHEIM/PÖSLING:
`http://hochfelln.fh-rosenheim.de/~hoegl/roberta/texte/aktuelles.htm`
(Startseite der private Wetterstation Rosenheim/Pösling)
aufgerufen am 22.09.2012

[8] WIKIPEDIA:
`http://de.wikipedia.org/wiki/Bus_(Datenverarbeitung)`
(Begriffserklärung Bussystem)
aufgerufen am 12.06.2012

[9] WIKIPEDIA:
`http://de.wikipedia.org/wiki/Cmos`
(Beschreibung der CMOS-Technologie)
aufgerufen am 25.06.2012

[10] WIKIPEDIA:
http://de.wikipedia.org/wiki/Carrier_Sense_Multiple_Access/Collision_Detection
(Ausführungen zum CSMA/CD Medienzugriffsverfahren)
aufgerufen am 28.06.2012

[11] WIKIPEDIA:
http://de.wikipedia.org/wiki/Erdbeschleunigung
(Ausführungen zur Erdbeschleunigung)
aufgerufen am 05.06.2012

[12] WIKIPEDIA:
http://de.wikipedia.org/wiki/Erdmagnetfeld
(Ausführungen zum Erdmagnetfeld)
aufgerufen am 04.06.2012

[13] WIKIPEDIA:
http://de.wikipedia.org/wiki/Ethernet
(Beschreibung der Ethernet-Technologie)
aufgerufen am 12.06.2012

[14] WIKIPEDIA:
http://de.wikipedia.org/wiki/I2c
(Zusammenfassung der I^2-Busspezifikation)
aufgerufen am 12.06.2012

[15] WIKIPEDIA:
http://de.wikipedia.org/wiki/Internet_Protocol
(Beschreibung IP)
aufgerufen am 28.06.2012

[16] WIKIPEDIA:
http://de.wikipedia.org/wiki/MAC-Adresse
(Beschreibung MAC-Adresse)
aufgerufen am 28.06.2012

[17] WIKIPEDIA:
http://de.wikipedia.org/wiki/Master-Slave
(Ausführungen zum Master-Slave-Prinzip)
aufgerufen am 12.06.2012

[18] WIKIPEDIA:
http://de.wikipedia.org/wiki/MOSFET
(Beschreibung MOSFET)
aufgerufen am 26.06.2012

[19] WIKIPEDIA:
http://de.wikipedia.org/wiki/Netzwerkprotokoll
(Beschreibung Netzwerkprotokoll)
aufgerufen am 28.06.2012

[20] WIKIPEDIA:
http://de.wikipedia.org/wiki/OSI-Modell
(Beschreibung OSI-Modell)
aufgerufen am 09.07.2012

[21] WIKIPEDIA:
http://de.wikipedia.org/wiki/Serial_Peripheral_Interface
(Zusammenfassung der SPI-Busspezifikation)
aufgerufen am 12.06.2012

[22] WIKIPEDIA:
http://de.wikipedia.org/wiki/Transmission_Control_Protocol
(Beschreibung TCP)
aufgerufen am 28.06.2012

[23] WIKIPEDIA:
http://de.wikipedia.org/wiki/User_Datagram_Protocol
(Beschreibung UDP)
aufgerufen am 28.06.2012

[24] WIKIPEDIA:
http://de.wikipedia.org/wiki/Zweierkomplement
(Beschreibung der binären Zahlendarstellung als Zweierkomplement)
aufgerufen am 02.06.2012

[25] SENSIRION AG:
http://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/Humidity/Sensirion_Humidity_SHT21_Datasheet_V3.pdf
(Produktseite SHT21)
heruntergeladen am 05.06.2012

[26] SENSORTECHNICS GMBH:
http://www.sensortechnics.com/cms/upload/datasheets/DS_Standard-HDI_E_11650.pdf
(Datenblatt HDI0611)
heruntergeladen am 30.03.2012

[27] SENSORTECHNICS GMBH:
http://www.sensortechnics.com/cms/upload/appnotes/AN_I2C-Bus-HDI-HCLA-HCA-SSI_E_11155.pdf
(I^2C-Bus-Kommunikation HDI0611)
heruntergeladen am 30.03.2012

[28] HONEYWELL INTERNATIONAL INC.:
http://www51.honeywell.com/aero/common/documents/myaerospacecatalog-documents/Missiles-Munitions/HMC6343.pdf
(Datenblatt HMC6343)
heruntergeladen am 30.03.2012

[29] HONEYWELL INTERNATIONAL INC.:
http://www51.honeywell.com/aero/common/documents/myaerospacecatalog-documents/Missiles-Munitions/HMC1041Z.pdf
(Datenblatt HMC1041Z)
heruntergeladen am 04.06.2012

[30] INVENSENSE INC.:
http://invensense.com/mems/gyro/documents/PS-MPU-6000A.pdf
(Product Specification MPU-6050)
heruntergeladen am 30.03.2012

[31] INVENSENSE INC.:
http://invensense.com/mems/gyro/documents/RM-MPU-6000A.pdf
(Register Map MPU-6050)
heruntergeladen am 30.03.2012

[32] INVENSENSE INC.:
http://invensense.com/mems/gyro/documents/AN-MPU-6000EVB.pdf
(User Guide MPU-6050)
heruntergeladen am 08.06.2012

[33] TEXAS INSTRUMENTS INC.:
http://www.ti.com/product/f28m35h52c
(Produktseite des F28M35H52C Concerto Microcontroller's)
aufgerufen am 15.05.2012

[34] TEXAS INSTRUMENTS INC.:
http://www.ti.com/lit/ml/sprb203a/sprb203a.pdf
(Concerto Brochure)
heruntergeladen am 30.03.2012

[35] TEXAS INSTRUMENTS INC.:
http://www.ti.com/lit/wp/spry174/spry174.pdf
(Concerto White Paper)
heruntergeladen am 30.03.2012

[36] TEXAS INSTRUMENTS INC.:
http://www.ti.com/tool/tmdxcncdh52c1
(Produktseite der H52C1 Concerto controlCARD)
aufgerufen am 15.05.2012

[37] TEXAS INSTRUMENTS INC.:
`http://www.ti.com/tool/tmdxdockh52c1`
(Produktseite des H52C1 Concerto Experimenter Kit)
aufgerufen am 15.05.2012

[38] TEXAS INSTRUMENTS INC.:
`http://www.ti.com/lit/ug/spruh22b/spruh22b.pdf`
(Concerto F28M35x Technical Reference Manual)
heruntergeladen am 30.03.2012

[39] TEXAS INSTRUMENTS INC.:
`http://www.ti.com/lit/ds/sprs742c/sprs742c.pdf`
(Concerto F28M35x Datasheet)
heruntergeladen am 30.03.2012

[40] TEXAS INSTRUMENTS INC.:
`http://focus.ti.com/graphics/mcu/concerto_mcu_func_block.gif`
(Concerto MCU Blockdiagramm)
heruntergeladen am 29.06.2012

[41] TEXAS INSTRUMENTS INC.:
F28M35xx_InfoSheet_Rev1 0_09_07_11 Release.pdf
(Experimenter Kit Info Sheet)
enthalten in der Software controlSUITE

[42] TEXAS INSTRUMENTS INC.:
F28M35x-FRM-EX-UG.pdf
(F28M35x Firmware Development Package USER'S GUIDE)
enthalten in der Software controlSUITE
Stand 07.07.2012

[43] TEXAS INSTRUMENTS INC.:
mcusdk_1_00_00_47_eng_release_notes.html
(MCU SDK 1.00.00.47 Release Notes)
enthalten im MCU SDK
Stand 07.07.2012

[44] TEXAS INSTRUMENTS INC.:
`http://www.ti.com/litv/pdf/spruex3k`
(SYS/BIOS User's Guide)
heruntergeladen am 16.08.2012

[45] TEXAS INSTRUMENTS INC.:
`http://www.ti.com/lsds/ti/microcontroller/32-bit_c2000/software.page`
(Seite mit Informationen zur Software controlSUITE und Code Composer Studio)
aufgerufen am 29.06.2012

[46] TEXAS INSTRUMENTS INC.:
http://www.ti.com/product/txs0104e
(Produktseite des TXS0104E Level-Translators)
aufgerufen am 05.06.2012

[47] TEXAS INSTRUMENTS INC.:
http://www.ti.com/lit/ds/symlink/txs0104e.pdf
(Datenblatt des TXS0104E Level-Translators)
heruntergeladen am 05.06.2012

[48] ELECTRONIC ASSEMBLY GMBH:
http://www.lcd-module.de/pdf/doma/dip204-4.pdf
(Datenblatt des DIP204B-4NLW LCD-Moduls)
heruntergeladen am 09.06.2012

[49] SAMSUNG ELECTRONICS GMBH:
http://www.lcd-module.de/eng/pdf/zubehoer/ks0073.pdf
(Datenblatt des KS0073 Display-Treibers)
heruntergeladen am 11.06.2012

[50] FAIRCHILD SEMICONDUCTOR CORPORATION:
http://www.fairchildsemi.com/ds/BC/BC846.pdf
(Datenblatt des Bipolar-Transistors BC846)
heruntergeladen am 09.06.2012

[51] NEXT-SOFT:
http://hexedit.nextsoft.de/
(Produktseite des Hex-Editor MX)
aufgerufen am 11.06.2012

[52] HW GROUP S.R.O.:
http://www.hw-group.com/products/hercules/index_en.html
(Download-Seite der Hercules SETUP utility)
aufgerufen am 11.06.2012

[53] PHILIPS SEMICONDUCTORS:
http://i2c2p.twibright.com:8080/spec/i2c.pdf
(The I^2-bus specification)
heruntergeladen am 12.06.2012

A Anhang

A.1 Abbildungsverzeichnis

A.2 Tabellenverzeichnis

A.3 Übersichten und Pläne

A.3.1 HAILsens Pinning

Pinname	Port/Pin	Funktion	Peripheral	Besonderheiten
GPIO8	B/0	Push-Pull out	GPIOB	LCD Backlight
GPIO10	B/2	I2C0SCL	I2C0	I^2C-Interface
GPIO11	B/3	I2C0SDA	I2C0	I^2C-Interface
GPIO12	B/4	SSI1TX	SSI1	SPI-Interface
GPIO13	B/5	SSI1RX	SSI1	SPI-Interface
GPIO14	B/6	SSI1CLK	SSI1	SPI-Interface
GPIO16	D/0	SSI0TX	SSI0	SPI-SD-Interface
GPIO17	D/1	SSI0RX	SSI0	SPI-SD-Interface
GPIO18	D/2	SSI0Clk	SSI0	SPI-SD-Interface
GPIO19	D/3	SSI0FSS	SSIO	SPI-SD-Interface
GPIO68	C/4	MII_TXD3	ETH	*)
GPIO70	C/6	Push-Pull out	GPIOC	LED7 auf Board
GPIO71	C/7	Push-Pull out	GPIOC	LED6 auf Board
GPIO20	D/4	Push-Pull in	GPIOD	Button Interrupt
GPIO30	E/6	MII_MDIO	ETH	*) zu Ethernet Transceiver auf Concerto controlCARD
GPIO37	F/5	MII_RXD3		
GPIO40	G/0	MII_RXD2		
GPIO41	G/1	MII_RXD1		
GPIO43	G/3	MII_RXDV		
GPIO47	G/7	MII_TXER		
GPIO49	H/1	MII_RXD0		
GPIO51	H/3	MII_TXD2		
GPIO52	H/4	MII_TXD1		
GPIO53	H/5	MII_TXD0		
GPIO54	H/6	MII_TXEN		
GPIO55	H/7	MII_TXCK		
GPIO56	J/0	MII_RXER		
GPIO58	J/2	MII_RXCK		
GPIO59	J/3	MII_MDC		
GPIO60	J/4	MII_COL		
GPIO61	J/5	MII_CRS		
GPIO62	J/6	MII_PHYINTR		
GPIO63	J/7	MII_PHYRST		
GPIO70	C/6	Push-Pull out	GPIOC	LED2

Tabelle A.1: Für HAILsens verwendete IO-Pins des F28M35H52C1 Microcontrollers

A.3.2 Concerto Blockdiagramm

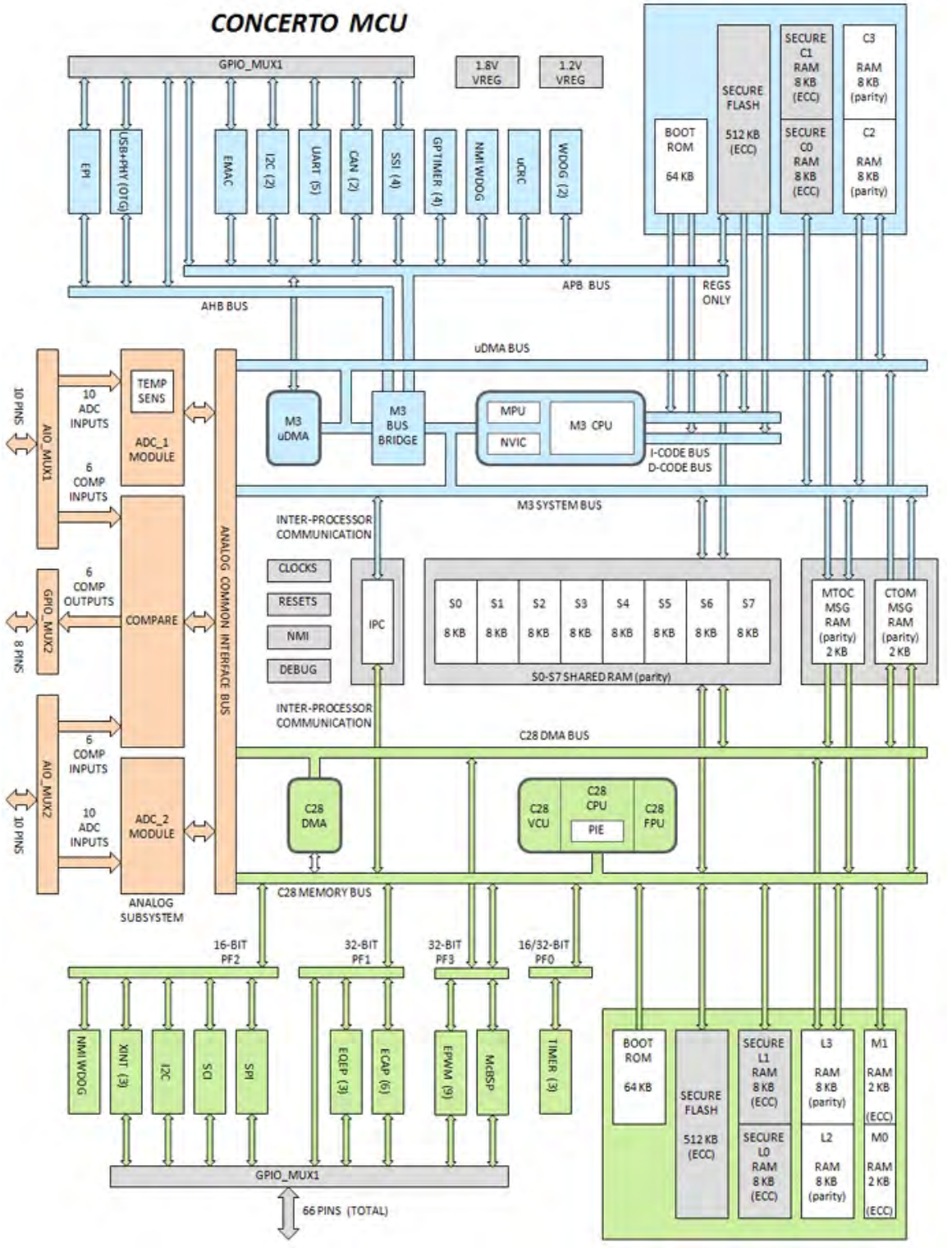


Abbildung A.1: Detailliertes Blockdiagramm der Concerto MCU [40]

A.3.3 Domino Transfer Protocol

Abbildung A.2: DTP am Beispiel zweier integrierter Sensoren [2]

A.3.4 HAILsens Echtzeit-Timing

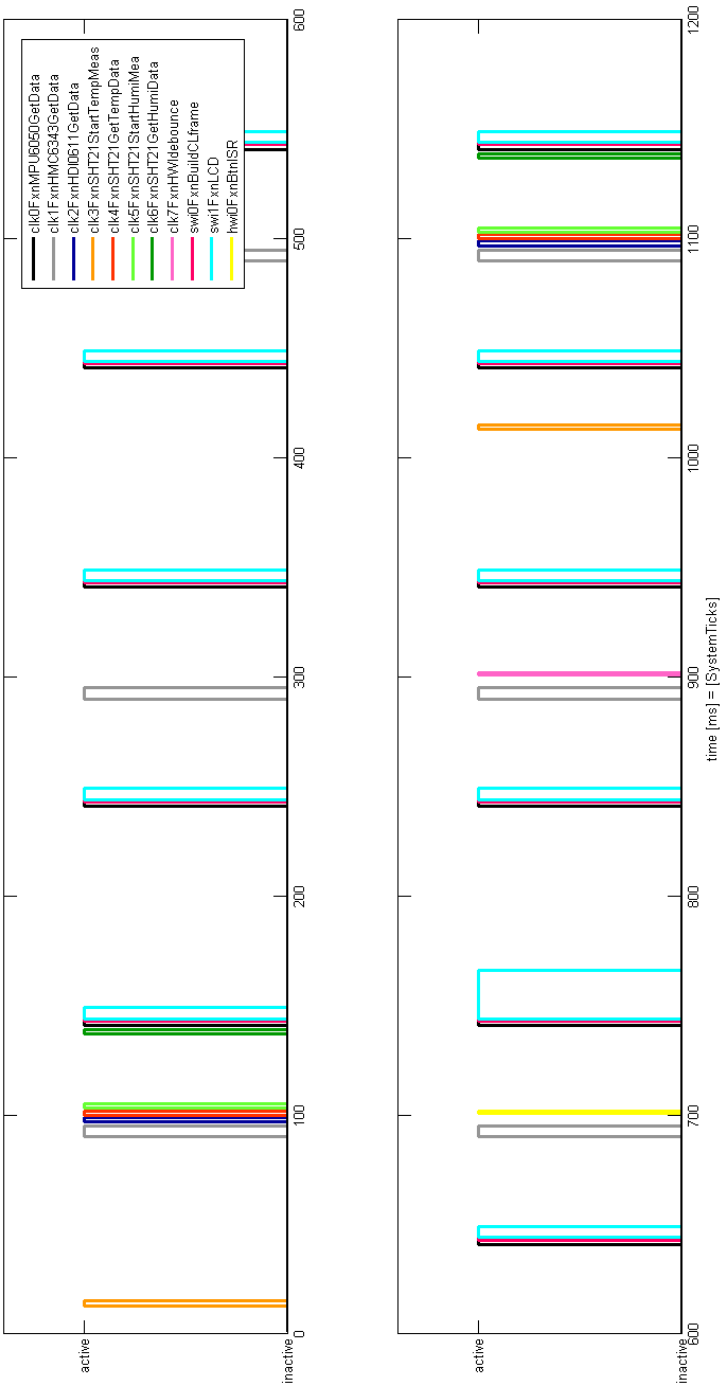


Abbildung A.3: Timing-Diagramm des Echtzeit-Schedulings in der HAILsensRTapp

Der Hintergrund-Task `tsk0FxnIOhandler()` läuft immer in den freien Zeitschlitzen dieser mit MATLAB erzeugten Grafik, ist hier also nicht explizit aufgeführt. Bei $t = 700\,ms$ ist die Reaktion der HWI-Routine (gelb) mit dem darin getriggerten Reaktivieren des Interrupts nach $200\,ms$ (rosa) zur Taster-Entprellung beispielhaft simuliert.

Taskname	Priorität	Verzögerung nach `BIOS_start()` $[ms]$	Periode $[ms]$	Aufgabe der Funktion	besondere Eigenschaften
`tsk0FxnUDPhandler`	1	–	–	Senden der in `swi0Fxn` erstellten Datenframes über UDP und SD-Karten-Management	Endlosschleife, läuft immer im Hintergrund
`swi0FxnBuildTXframe`	18	< 142	100	Organisation der Mess*byte* in einem Frame	kürzeste Ausführungsdauer, getriggert in `clk0Fxn`
`swi1FxnLCD`	16	< 143	100	Steuern der LCD-Aktivität	getriggert in `clk0Fxn`
`clk0FxnMPU6050GetData`	31	140	100	Beschleunigung und Drehraten abfragen	triggert Framebildung und LCD-Steuerung
`clk1FxnHMC6343GetData`	31	90	200	Ausrichtung und Magnetfeld abfragen	–
`clk2FxnHDI0611GetData`	31	97	1000	Luftdruck abfragen	–
`clk3FxnSHT21StartTempMeas`	31	13	1000	Temperaturmessung starten	–
`clk4FxnSHT21GetTempData`	31	100	1000	Temperaturdaten abfragen	–
`clk5FxnSHT21StartHumiMeas`	31	103	1000	Feuchtemessung starten	–
`clk6FxnSHT21GetHumiData`	31	137	1000	Feuchtedaten abfragen	schalten LED7 auf controlCARD
`clk7FxnHWIdebounce`	31	unbestimmt	nicht periodisch	softwaremäßige Taster-Entprellung	getriggert in `hwi0Fxn` `hwi0Fxn`
`hwi0FxnBtnISR`	> 31	asynchroner Interrupt	asynchroner Interrupt	Umschalten des LCD-Status	höchste Priorität, triggert `clk7Fxn`

Tabelle A.2: Zusammenfassung des Zeitplans für die Echtzeit-Tasks

A.3.5 Subroutinen-Abläufe

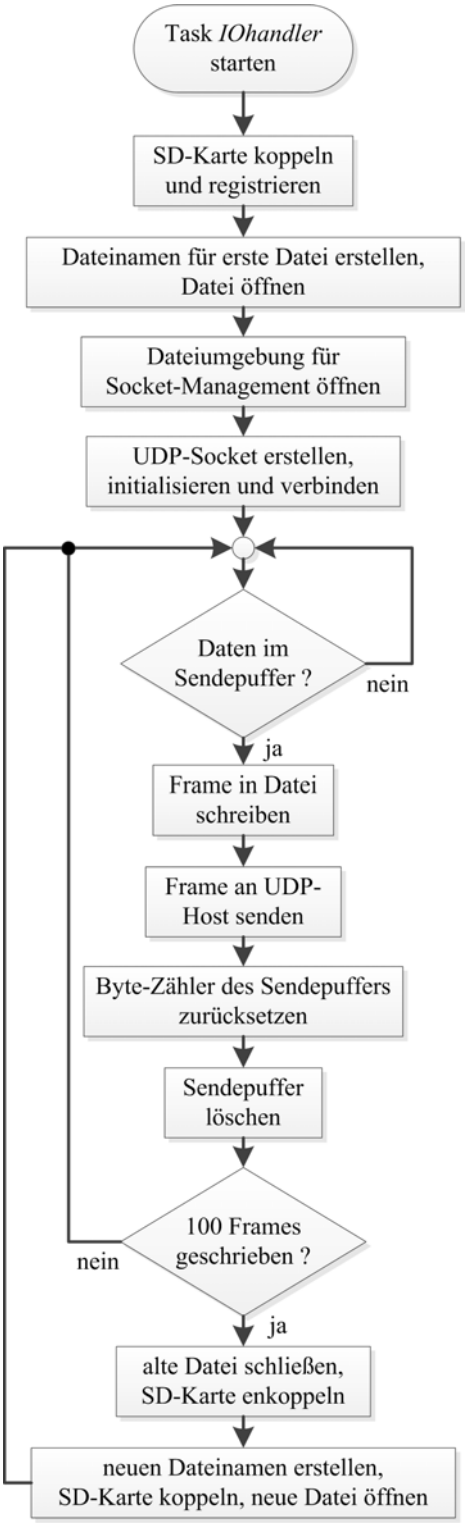


Abbildung A.4: PAP `tsk0FxnIOhandler()`

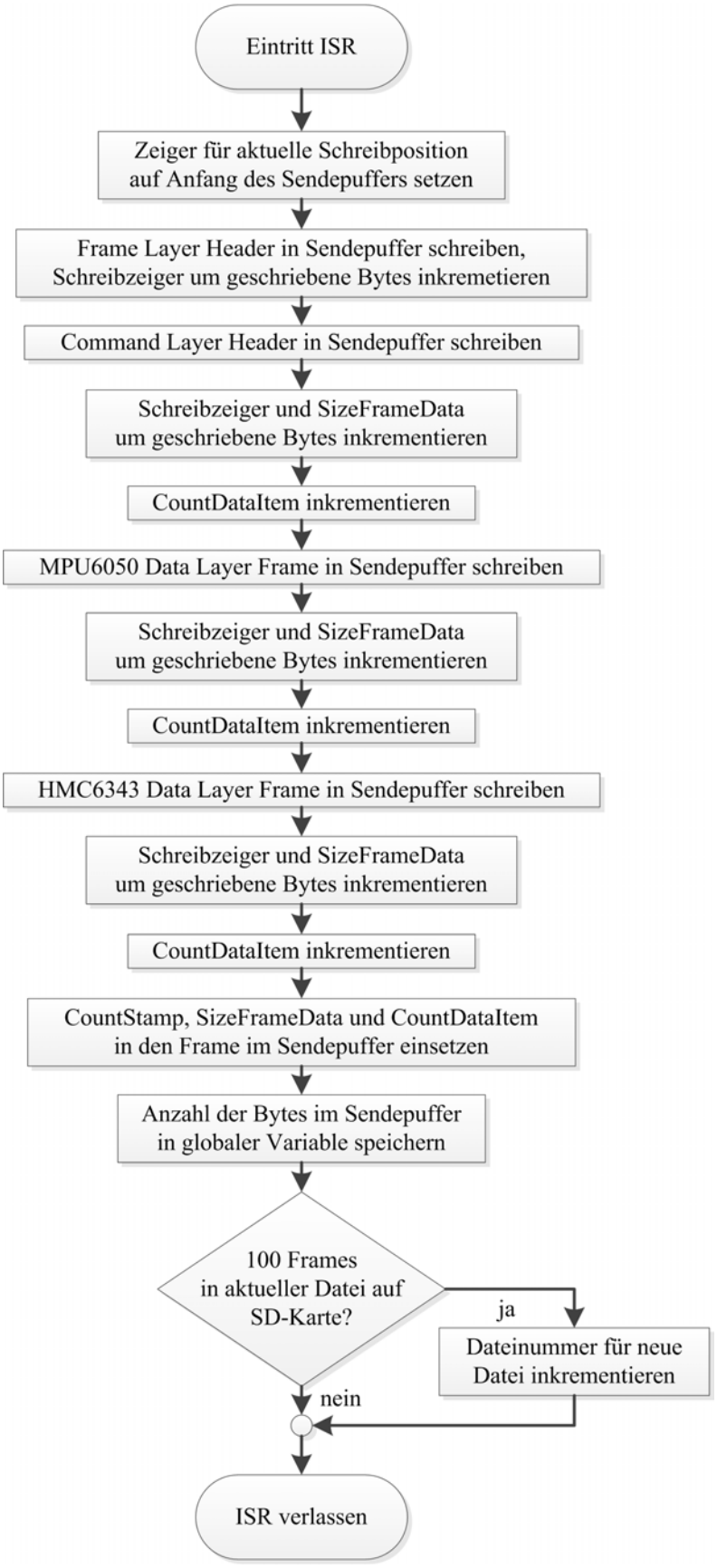


Abbildung A.5: PAP der ISR `swi0FxnBuildTXframe()` für den `case Frame03`

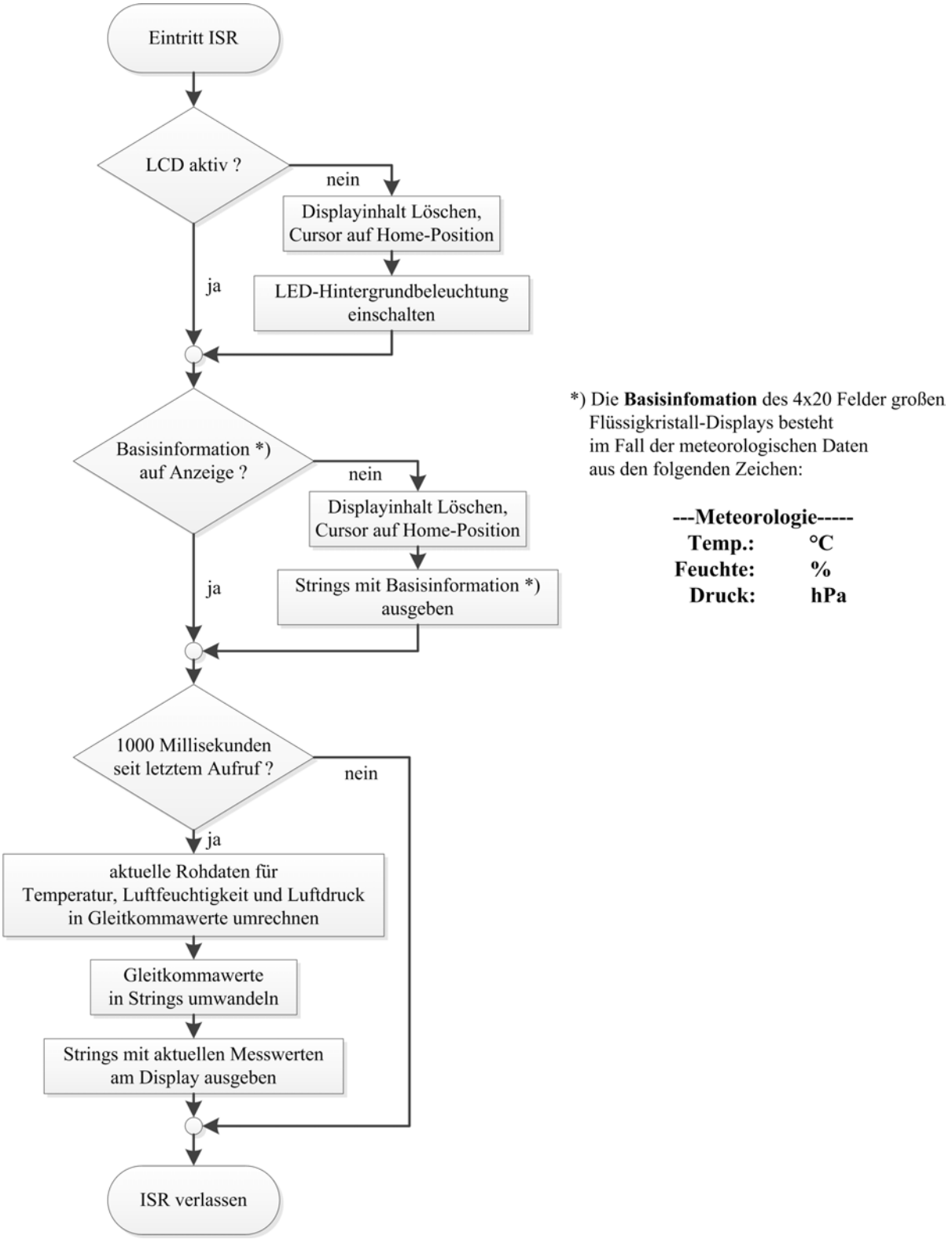


Abbildung A.6: PAP der ISR `swi1FxnLCD()` für den `case Meteo`

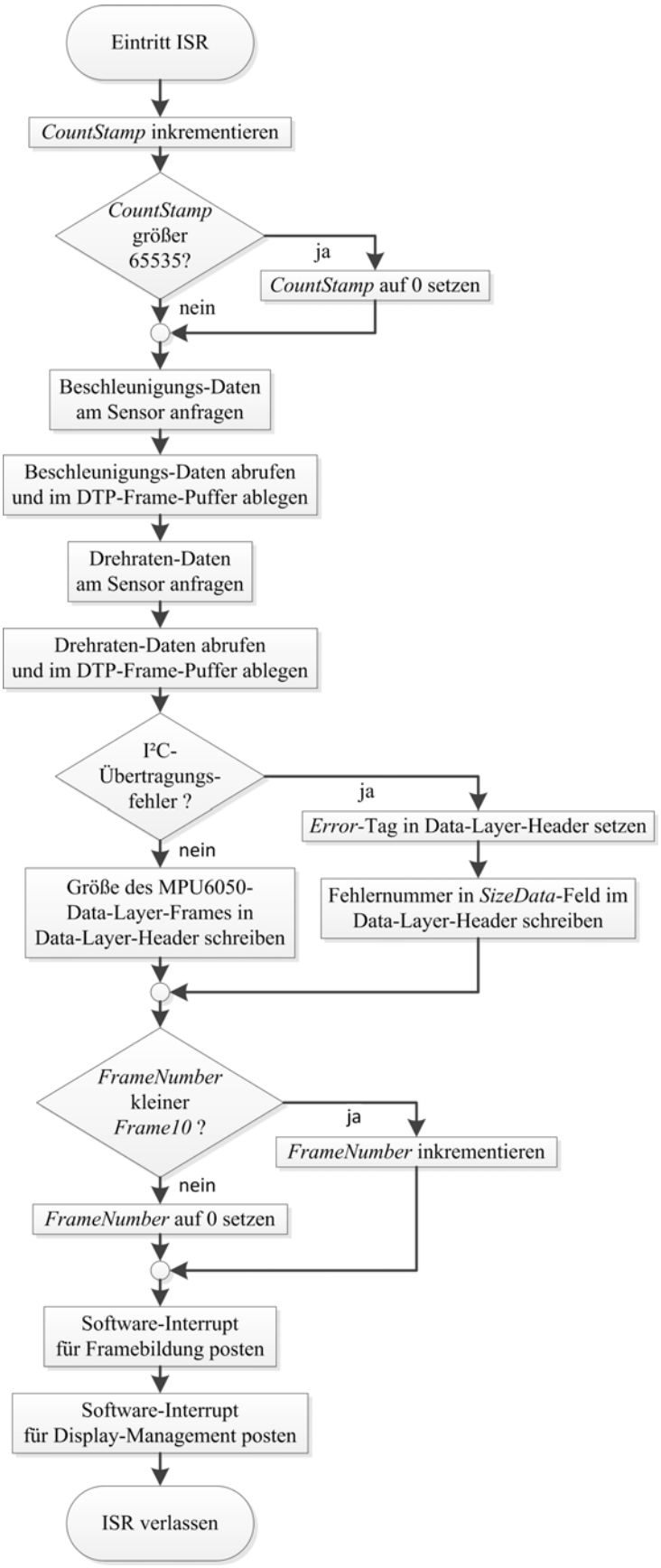


Abbildung A.7: PAP der Clock-ISR `clk0FxnMPU6050GetData`

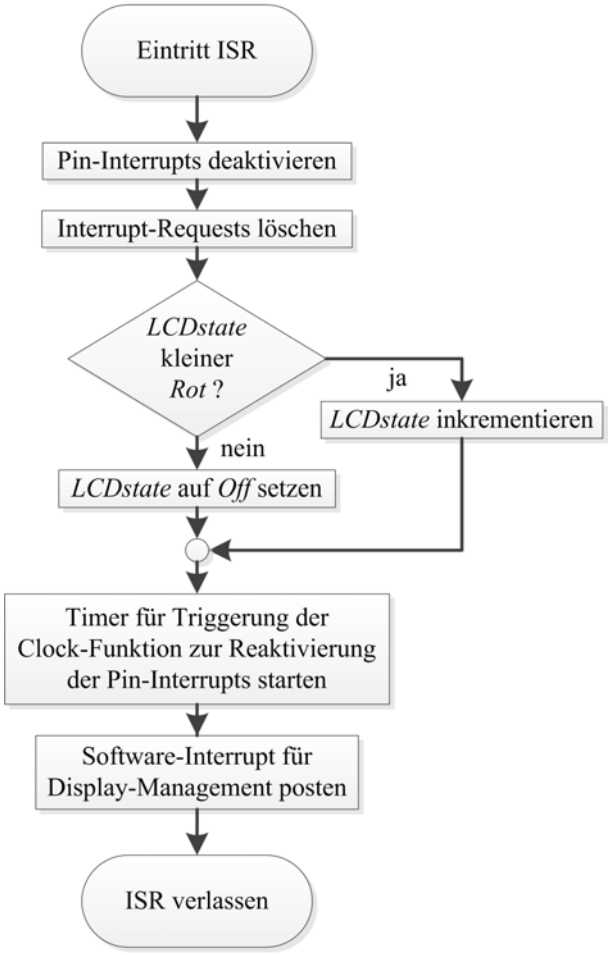


Abbildung A.8: PAP der Hardware-Interrupt Service-Routine `hwi0FxnButtonPress`