

# Entwicklung eines Evaluierungswerkzeuges für barometrische Luftdruck-Sensoren

## Bachelorarbeit

im Studiengang Informatik



Vorgelegt von: Benedikt Kleinmeier

Fakultät: Informatik und Mathematik

Matrikelnummer: 03 05 98 08

Prüferin: Prof. Dr. Heidi Anlauff

6. März 2012

## Zusammenfassung

Die Infineon Technologies AG, ein Entwickler und Hersteller von Halbleiterprodukten, produziert Sensoren zur Messung des barometrischen Luftdrucks. Um Kunden bei der Entwicklung von Produkten auf Basis dieser Luftdruck-Sensoren zu unterstützen, soll ein Evaluierungswerkzeug entwickelt werden, das die Druckdaten grafisch anzeigt und protokolliert. Das Evaluierungswerkzeug besteht aus einem Drucksensor, einem Programmiergerät und einem PC, auf dem eine grafische Benutzeroberfläche läuft. Es werden digitale wie analoge Drucksensoren unterstützt. Daneben noch zwei Programmiergeräte, die jeweils mit einem Mikrocontroller ausgestattet sind. Der PC nutzt das Programmiergerät, um mit dem Sensor zu kommunizieren. Dazu muss im Laufe eines Softwareentwicklungsprozesses für die Mikrocontroller eine Firmware in der Programmiersprache C geschrieben werden. Des Weiteren eine grafische Benutzeroberfläche in C# für den PC. Beide Software-Komponenten sollen dabei modular und generisch gehalten werden, damit sie auch in zukünftigen Projekten wiederverwendet werden können. Am Ende des Softwareentwicklungsprozesses steht das fertige Evaluierungswerkzeug. Der Kunde erhält die dokumentierte grafische Benutzeroberfläche, die durch ein Installationsprogramm leicht installiert werden kann.

**Themengebiete:** Grafische Benutzeroberfläche, Luftdruck-Sensoren, Mikrocontroller-Firmware, Programmiersprachen C und C#, .NET-Framework

## **Vorwort**

Ich möchte mich an dieser Stelle ganz herzlich bei der Infineon Technologies AG bedanken. Sie schenkte mir das Vertrauen, das in dieser Bachelorarbeit beschriebene Projekt, in einer spannenden und überaus arbeitnehmerfreundlichen Umgebung umzusetzen. Ein ganz besonderer Dank gilt meinem Betreuer, Dr. Daniel Bichler, welcher mir mit seinem umfangreichen Wissen jederzeit hilfreich zur Seite stand.

Dieses Projekt gab mir die Möglichkeit meine bisher erworbenen Fähigkeiten gewinnbringend einzusetzen und mich fachlich, wie auch persönlich weiterzuentwickeln. Bin für diese Gelegenheit sehr dankbar und auch für die Menschen und Persönlichkeiten, die ich durch dieses Projekt kennenlernen durfte, darunter: Matthias Böhm, Dr. Michael Brauer, Leonardo Cruz Velasquez, Bochang Lee, Martin Mangstl, Richard Moser, Alfred Niklas, Dr. Michael Wycisk, Sigmund Zaruba.

Vielen Dank.

Benedikt Kleinmeier

# Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>6</b>
<b>Tabellenverzeichnis</b>	<b>8</b>
<b>Listingsverzeichnis</b>	<b>9</b>
<b>1 Einführung</b>	<b>10</b>
1.1 Übersicht des Evaluierungswerkzeuges . . . . .	10
1.1.1 Sensor . . . . .	11
1.1.2 Programmiergerät . . . . .	14
1.1.3 PC (Grafische Benutzeroberfläche) . . . . .	15
1.2 Der Projektumfang . . . . .	16
<b>2 Die Komponenten des Evaluierungswerkzeuges</b>	<b>18</b>
2.1 Unterstützte Sensoren . . . . .	18
2.1.1 Digitale Sensoren . . . . .	18
2.1.2 Analoge Sensoren . . . . .	29
2.2 Unterstützte Programmiergeräte . . . . .	34
2.2.1 PGSISI2 . . . . .	35
2.2.2 UWLink . . . . .	39
2.3 PC (Grafische Benutzeroberfläche) . . . . .	42
2.3.1 Hardware . . . . .	43
2.3.2 Software . . . . .	43
<b>3 Der Softwareentwicklungsprozess</b>	<b>44</b>
3.1 Die Mikrocontroller-Firmware . . . . .	45
3.1.1 Firmware für XC886CLM-8FF (UWLink) . . . . .	51
3.1.2 Firmware für XC164CS-16F (PGSISI2) . . . . .	59
3.2 Die Schnittstelle zwischen Firmware und GUI: Das Protokoll . . . . .	66
3.3 Die grafische Benutzeroberfläche . . . . .	69
3.3.1 GUI . . . . .	70
3.3.2 Installationsprogramm . . . . .	84
3.3.3 Benutzerhandbuch . . . . .	86
<b>4 Ergebnis: Die grafische Benutzeroberfläche</b>	<b>88</b>
4.1 Die drei Kontrollelemente der GUI . . . . .	89
4.1.1 Menüleiste . . . . .	90
4.1.2 Seitenleiste . . . . .	92
4.1.3 Bereich für Sensordaten . . . . .	92

5 Zusammenfassung und Ausblick	96
Glossar	100
Literatur	102

## Abbildungsverzeichnis

1	Datenfluss zwischen den Komponenten des Evaluierungswerkzeuges	11	14	Beispielverkabelung zwischen digitalem Sensor und Mikrocontroller . . . . .	29
2	Größenvergleich zwischen digitalem Drucksensor mit acht Pins und 1-Cent-Münze	12	15	Pin-Konfiguration der analogen Drucksensoren KP234, KP235 und KP236 . . . . .	30
3	Transfer-Funktion eines digitalen Drucksensors . . . . .	13	16	Blockdiagramm zum internen Aufbau der analogen Drucksensoren . . . . .	32
4	Die unterstützten Programmiergeräte des Evaluierungswerkzeuges . . . . .	15	17	Beispiel für lineare Transfer-Funktion (Druck) des KP236 . . . . .	33
5	Pin-Konfiguration der digitalen Drucksensoren KP253, KP254 und KP256 . . . . .	19	18	Beispielverkabelung zwischen analogem Sensor und Mikrocontroller . . . . .	34
6	Blockdiagramm zum internen Aufbau der digitalen Drucksensoren	21	19	Anschlüsse der PGSI-SI2 (Vorderseite) . . . . .	35
7	Beispiel für lineare Transfer-Funktion (Druck) des KP256 . . . . .	21	20	Anschlüsse der PGSI-SI2 (Rückseite) . . . . .	36
8	Struktur der SPI-Kommandos . . . . .	23	21	Kabelverbindung zwischen PGSISI2 und Platine mit Sensor-Sockel . . . . .	37
9	Struktur der SPI-Antworten für KP256 und KP256 . . . . .	24	22	Funktionseinheiten des XC164-16-Mikrcontrollers . . . . .	39
10	Struktur der SPI-Antworten für KP253 . . . . .	24	23	Komponenten des UWLinks . . . . .	40
11	Struktur der SPI-Antworten für Kommando „Identifier“ . . . . .	24	24	UWLink mit Daughterboard und digitalem Drucksensor . . . . .	40
12	Master-Slave-Kommunikation mit vier Leitungen bei SPI . . . . .	26	25	Funktionseinheiten der XC88xCLM-Mikrcontroller . . . . .	42
13	SPI-Timing des digitalen Drucksensors KP256	28	26	Abbildung der Mikrocontroller-Module in Software . . . . .	46

27	Projektstruktur für 8-Bit-Mikrocontroller . . . . .	51	44	Bereich für Sensordaten nach gestarteter Messung . . . . .	93
28	XC88xCLM SSC Control Register . . . . .	56	45	Kontextmenü der Graphen . . . . .	94
29	Projektstruktur für 16-Bit-Mikrocontroller . . . . .	61	46	Log-Bereich der digitalen Sensoren . . . . .	94
30	EEPROM-Aufbau der Drucksensoren . . . . .	64	47	Buttons im Log-Bereich	95
31	Write- und Erase-Pattern für EEPROM-Programmierung . . . . .	65			
32	Master-Slave-Prinzip zwischen GUI und Firmware . . . . .	67			
33	Die Projektstruktur der GUI . . . . .	70			
34	GUI-Prototyp mit Windows Forms-Designer . . . . .	73			
35	Vollständiges Klassendiagramm der Geschäftslogik . . . . .	76			
36	Klassendiagramm der untersten Ebene . . . . .	76			
37	Klassendiagramm der mittleren Ebene . . . . .	79			
38	Klassendiagramm der obersten Ebene . . . . .	80			
39	Das Benutzerhandbuch mit FrameMaker 8 . . . . .	86			
40	Kontrollelemente der GUI nach Auswahl eines Sensors . . . . .	90			
41	Menüleiste in der GUI	90			
42	EEPROM-Programmierung über das Fenster EEPROM Map	91			
43	Programmer und Sensor Area in der Seitenleiste . . . . .	92			

## Tabellenverzeichnis

1	Merkmale der digitalen Sensoren KP253, KP254 und KP256 . . . . .	19	15	Beispiel für GUI-Firmware-Kommunikation mit vereinbartem Protokoll . . . . .	69
2	Funktionen der Sensor-Pins . . . . .	20	16	Abhängigkeiten der Software-Module der GUI . . . . .	72
3	Transfer-Funktionen (Druck) für KP253, KP254 und KP256 . . . . .	22	17	Lines of Code Firmware 8-Bit-Mikrocontroller	88
4	Transfer-Funktionen (Temperatur) für KP253, KP254 und KP256 . . . . .	22	18	Lines of Code Firmware 16-Bit-Mikrocontroller . . . . .	88
5	SPI-Kommandos für KP253, KP254 und KP256 . . . . .	23	19	Lines of Code GUI . . . . .	89
6	Bitfeld „Diagnosis“ . . . . .	25			
7	Antwortverhalten der digitalen Drucksensoren KP253, KP254 und KP256 . . . . .	25			
8	Merkmale der analogen Sensoren KP234, KP235 und KP236 . . . . .	30			
9	Funktionen der Sensor-Pins . . . . .	31			
10	Transfer-Funktionen (Druck) für KP234, KP235 und KP236 . . . . .	34			
11	Versionsübersicht der Keil-Toolchains . . . . .	48			
12	Bitfelder im 8-Bit-Register SSC_CONL_P	57			
13	Nachrichtenformat GUI zu Firmware . . . . .	67			
14	Nachrichtenformat Firmware zu GUI . . . . .	68			

## Listingsverzeichnis

1	Doxygen-Kommentar der Datei Main.c (UWLink) . . . . .	49	6	Methode SSC_ SetSSC_CONH_P in der Datei SSC.c . . . . .	58
2	Pseudocode für Firmware-Architektur .	50	7	Methode SSC_SendDate in der Datei SSC.c . . .	58
3	CmdDefines.h . . . . .	53	8	Beispiel für Device- Function . . . . .	78
4	Main.c . . . . . . .	53	9	Das Enum Sensor- Command . . . . .	80
5	Enumeration SSC_Baud_Rate in Datei SSC.h . . . . .	56	10	XML-Datei für Sen- sorfamile KP25x . . . .	82
			11	Conditional Compila- tion Symbols in C# . .	83

# 1 Einführung

Halbleiterprodukte sind aus unserer modernen Welt nicht mehr wegzudenken. Computer und Co. bilden die Grundlage unseres täglichen Lebens. Sie sorgen unter anderem für den reibungslosen Betrieb von Fertigungsanlagen, ermöglichen schnelle Kommunikation und sichern Leben. Für viele dieser Aufgaben muss der Computer jedoch zuerst seine Umwelt wahrnehmen. Dies geschieht mit Hilfe von weiteren elektronischen Bauteilen: den Sensoren. Es gibt sie in verschiedenster Ausführung zur Rezeption von Temperatur, Druck, Beschleunigung und weiteren physikalischen oder chemischen Eigenschaften.

Die Infineon Technologies AG, ein Entwickler und Hersteller von Halbleiterprodukten, produziert und kalibriert in seinen Werken in Regensburg, Villach und Singapur barometrische Luftdruck-Sensoren. Diese Sensortypen werden zum Beispiel im Automobilbereich eingesetzt. Sie werden dort im Motorraum verbaut und liefern Druckdaten für die Motorsteuerung von Kraftfahrzeugen. Mit Hilfe dieser Druckdaten ist es der Motorsteuerung möglich ein optimales Verhältnis von Kraftstoff und Luft zu mischen. Dies sorgt für einen geringen Kraftstoffverbrauch bei bestmöglicher Leistung. Im Regelbetrieb gelangen die Druckdaten des Sensors direkt zur Motorsteuerung.

Um nun aber Kunden der Infineon Technologies AG, darunter Automobilzulieferer wie die Continental AG, bei der Entwicklung ihrer Bauteile zu unterstützen, sollen die Druckdaten protokolliert und grafisch angezeigt werden. Zudem soll der Infineon Technologies AG die Möglichkeit vorbehalten bleiben das Electrically Erasable Programmable Read-Only Memory (EEPROM) des Drucksensors auszulesen und zu beschreiben. Die Werte in diesem nichtflüchtigen Speicher sorgen für eine möglichst genaue Umwandlung des physikalischen Drucks in ein elektrisches Signal. Diese Werte sind also ein entscheidender Faktor für die Messgenauigkeit des Sensors und werden bei der Produktion kalibriert. Die grafische Darstellung der Sensormessdaten sowie Auslesen und Beschreiben des EEPROMs sollen durch ein einziges Evaluierungswerkzeug bewerkstelligt werden.

## 1.1 Übersicht des Evaluierungswerkzeuges

Die Messdaten des Drucksensors lassen sich am besten an einem PC anzeigen und auswerten. Da der Sensor aber keinen PC-tauglichen Anschluss besitzt,

ist eine Schnittstelle zwischen beiden nötig. Diese stellt das Programmiergerät (mit integriertem Mikrocontroller) dar. Das **Evaluierungswerkzeug** besteht demzufolge **aus** einem **Sensor**, einem **Programmiergerät** und einer **grafischen Benutzeroberfläche** (GUI) auf Seiten des PCs (siehe Abbildung 1).

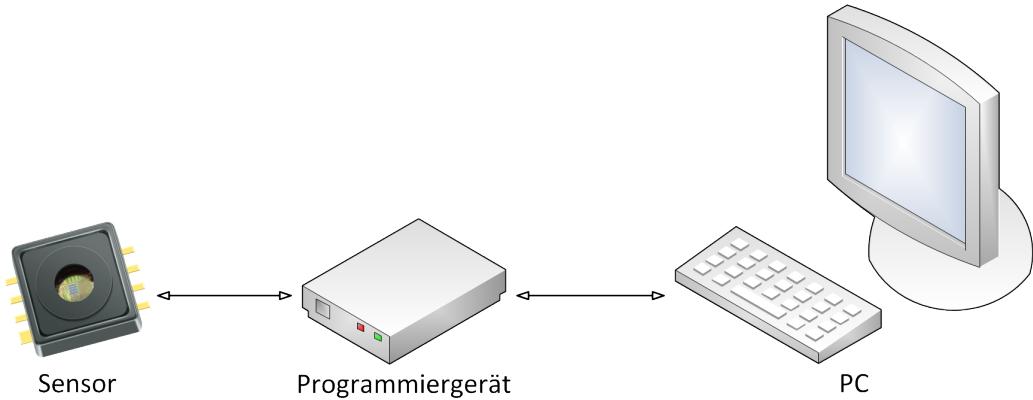


Abbildung 1: Datenfluss zwischen den Komponenten des Evaluierungswerkzeugs

Die folgenden drei Abschnitte liefern eine kurze Beschreibung der Komponenten des Evaluierungswerkzeugs.

### 1.1.1 Sensor

Der Drucksensor ist ein elektronisches Bauteil auf Basis von Halbleitertechnik. Er wandelt den physikalischen Druck<sup>1</sup> in ein zum Druck proportionales elektrisches Signal um. Sensoren arbeiten allgemein nach dem Ausschlagprinzip. D.h. die auf eine Fläche wirkende Kraft wird zuerst in ein elektrisches Signal umgewandelt. Anhand dieses präzisen Signals können nun selbst kleinste „Ausschläge“ (also Druckänderungen) wahrgenommen werden.

Das Messen von Druck erfolgt immer gegenüber eines Referenzdrucks. Die Sensoren des Evaluierungswerkzeugs verwenden als Referenzdruck „null“ (Pascal bzw. Bar) und liefern demzufolge den Absolutdruck. Man spricht deshalb von absoluten Drucksensoren ([4, S. 158-168]).

<sup>1</sup>„Druck ist als Kraft durch Fläche definiert.“ [4, S. 153]

Das Evaluierungswerkzeug unterstützt **digitale** wie **analoge** Drucksensoren. Beide Sensortypen haben jeweils acht Pins (siehe Abbildung 2), unterscheiden sich jedoch im internen Aufbau.

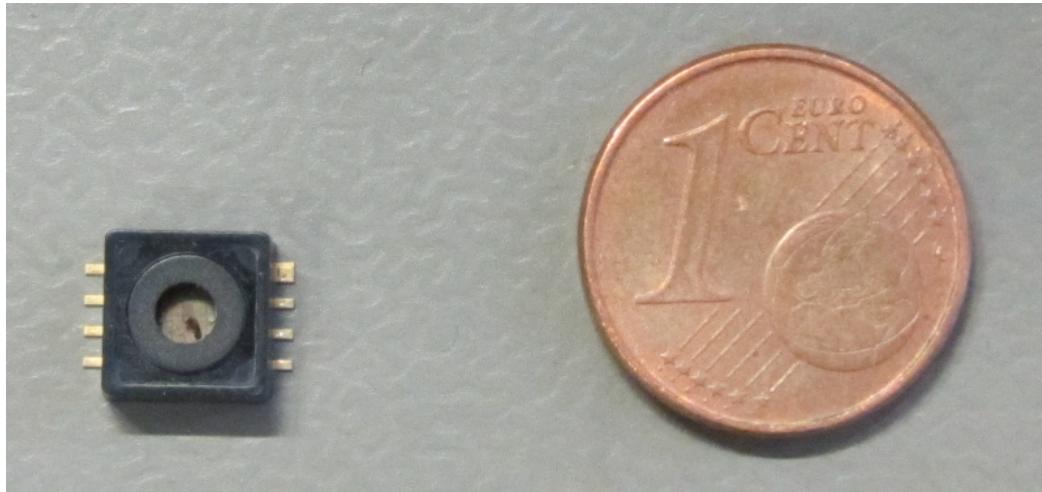


Abbildung 2: Größenvergleich zwischen digitalem Drucksensor mit acht Pins und 1-Cent-Münze

**Digitale Sensoren** Die digitalen Sensoren bestehen im Allgemeinen aus folgenden Komponenten:

- Einem Spannungsregler: Dieser sorgt für eine Stabilisierung der Versorgungsspannung und beugt somit Spannungsschwankungen vor. Schwankungen hätten Einfluss auf die Messgenauigkeit des Sensors und müssen somit verhindert werden.
- Den Sensorzellen: Diese arbeiten bei den vorliegenden Sensoren nach dem kapazitiven Prinzip. D.h. der physikalische Druck verformt eine Membran. Diese Verformung bewirkt, dass sich die Kapazität eines angeschlossenen Kondensators ändert und diese Änderung wird gemessen.
- Einem Analog-Digital-Konverter (ADC): Er konvertiert das analoge Signal in ein digitales. Die vorliegenden digitalen Sensoren nutzen dazu das Sigma-Delta-Verfahren.
- Digitale Signalprozessoren (DSP): Tiefpassfilter sorgen hierbei unter anderem dafür, dass Störungen aus dem digitalen Signal gefiltert werden.

- Einem EEPROM: Es speichert die Werte der Sensor-Kalibrierung. Die Werte werden verwendet, um den Druck möglichst genau in ein elektrisches Signal umzuwandeln.
- Einem Serial Peripheral Interface (SPI): Der gemessene und in ein digitales Signal umgewandelte Druck wird über ein sogenanntes SPI ausgegeben. Bei SPI handelt es sich um ein serielles Bussystem. Siehe Kapitel 2.1.1 auf Seite 26 für eine genauere Beschreibung von SPI.

**Analoge Sensoren** Die analogen Sensoren wandeln das durch die Sensorzellen gewonnene analoge Signal nicht ein digitales um und benötigen deshalb weder einen ADC- noch einen DSP-Baustein und auch kein SPI. Sie geben das analoge Signal direkt an einem ihrer acht Sensor-Pins aus.

Das ausgegebene Signal liegt nun entweder analog in Volt oder digital in Bit vor. Im späteren Verarbeitungsprozess muss das Signal noch in eine für Menschen verständliche Größe (Pascal oder Bar) umgewandelt werden. Dies geschieht anhand einer Transfer-Funktion, auch Kennlinie genannt. Jeder konkrete Sensor besitzt eine eigene lineare Transfer-Funktion (siehe Abbildung 3).

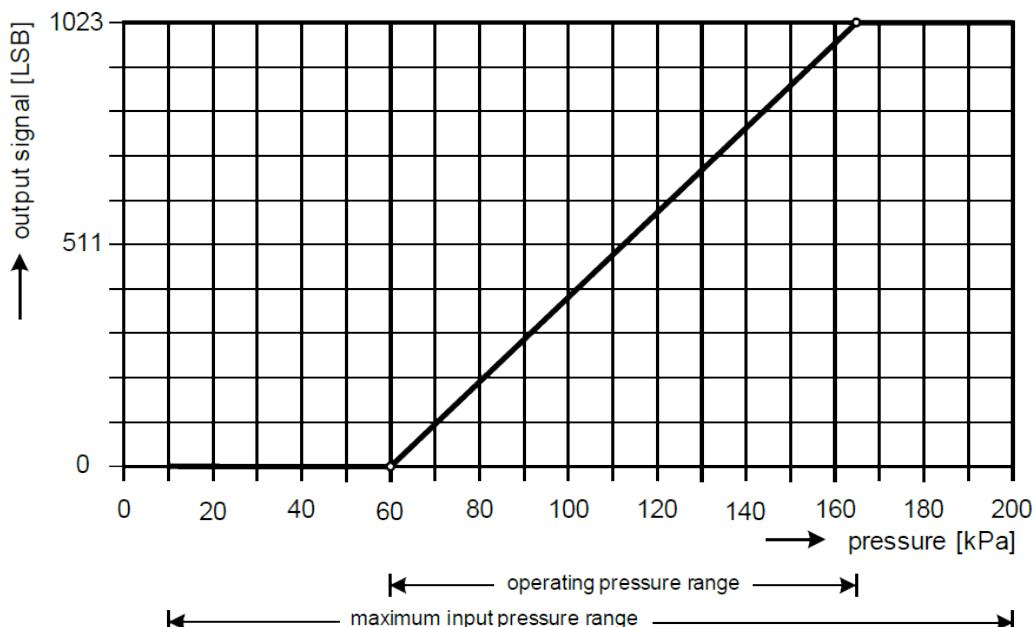


Abbildung 3: Transfer-Funktion eines digitalen Drucksensors [13, S. 9]

Neben der Bauart (digital oder analog) unterscheiden sich die unterstützten Drucksensoren auch im Messbereich des Druckwerts. Je nach Sensor liegt der Bereich zwischen minimal 40 und maximal 165 Kilopascal.

### 1.1.2 Programmiergerät

Das Programmiergerät ist die Schnittstelle zwischen Drucksensor und PC. Diese ist notwendig, da wie in Kapitel 1.1.1 gesehen die Drucksensoren ausschließlich mit acht Pins zur „Ein- und Ausgabe“ ausgestattet sind. Sensor und PC lassen sich so nicht auf direktem Wege verbinden. Die übliche Schnittstelle zwischen beiden Komponenten bildet nun ein Mikrocontroller. Der Mikrocontroller besitzt neben klassischen Befehls- und Speicherkomponenten wie CPU, RAM und ROM auch spezielle periphere Systemkomponenten. Sie haben die Aufgabe Informationen mit extern angeschlossenen Systemen (z.B. Sensoren) auszutauschen und passend umzuwandeln, sodass sie unter anderem von der CPU „verstanden“ werden. Es existieren periphere Systemkomponenten, um Funktionen wie SPI oder andere (serielle) Schnittstellen anzusprechen. Der Mikrocontroller wird dazu mit allen benötigten Hardware-Anschlüssen (z.B. USB-Buchse) auf einer Leiterplatine (PCB, englisch für „Printed Circuit Board“) untergebracht. Die Mikrocontroller-Pins, die für eine bestimmte Funktionalität (z.B. USB) zuständig sind, werden letztendlich mit dem entsprechenden Hardware-Anschluss verbunden.

Beim Evaluierungswerkzeug kommuniziert der Drucksensor direkt mit dem Mikrocontroller. Bei den digitalen Sensoren wird das SPI verwendet. Bei den analogen Sensoren wird der Druckwert über einen ADC des Mikrocontrollers eingelesen. Beide Funktionen werden über Leiterbahnen bzw. Kabel zum Sensor geführt. Als Schnittstelle zum PC wird auf Grund der sehr großen Verbreitung USB gewählt. Die verbauten Mikrocontroller unterstützen als serielle Schnittstellen jedoch nur CAN, RS232C und SPI. Aus diesem Grund wird zwischen PC und Mikrocontroller noch ein integrierter Schaltkreis (IC) der englischen Firma Future Technology Devices International Ltd. (FTDI) geschaltet. Dieser IC übernimmt die „Übersetzung“ von USB in RS232 und zurück.

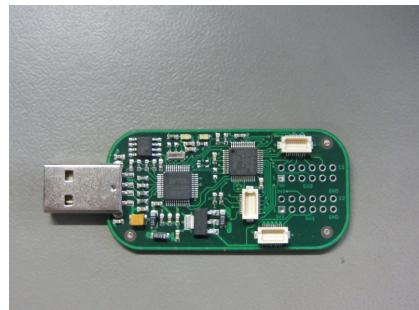
Alle Bauteile, darunter Hardware-Anschlüsse (USB und RS232), Mikrocontroller und USB-RS232-Übersetzer, sind auf einem PCB befestigt und in einem Gehäuse untergebracht. Im Rahmen dieses Projekts spricht man nun vom Programmiergerät, da es mit diesem nicht nur möglich ist Druckwerte

auszulesen, sondern auch das EEPROM des angeschlossenen Drucksensors zu programmieren.

Das Evaluierungswerkzeug unterstützt zwei verschiedene Programmiergeräte. Das vollwertige Programmiergerät Programmer System Infineon Sensor Interface 2 (PGSISI2) (siehe Abbildung 4a). Es unterstützt digitale wie analoge Sensoren und ist in der Lage das EEPROM der Sensoren zu programmieren. Daneben gibt es eine abgespeckte Version, Universal Wireless Link (UWLink) genannt (siehe Abbildung 4b). Dieses Programmiergerät unterstützt ausschließlich digitale Drucksensoren und bietet nur einen USB-Anschluss als Verbindung zum PC. Zudem ist es mit dem UWLink nicht möglich das EEPROM eines Sensors zu programmieren. Die PGSISI2 ist für die interne Nutzung bei der Infineon Technologies AG gedacht und ist mit einem 16-Bit-Mikrocontroller von Infineon ausgestattet. Der UWLink dagegen bietet für Kunden eine preisgünstige Schnittstelle zwischen Sensor und PC. Auf dem UWLink ist daher nur ein 8-Bit-Mikrocontroller, ebenfalls von Infineon, verbaut.



(a) PGSISI2



(b) UWLink

Abbildung 4: Die unterstützten Programmiergeräte des Evaluierungswerkzeugs

### 1.1.3 PC (Grafische Benutzeroberfläche)

Die PC-Software mit grafischer Benutzeroberfläche wird verwendet, um die Funktionalitäten des Sensors zu visualisieren. Dazu wird der PC, auf dem die GUI betrieben wird, via USB mit dem Programmiergerät verbunden. Das Programmiergerät ist wiederum mit dem Drucksensor verbunden.

Die GUI erlaubt es mehrere Sensoren gleichzeitig zu betreiben. Jedoch benötigt

tigt dann jeder Drucksensor ein eigenes Programmiergerät. Nach Auswahl des zu verwendenden Programmiergeräts und Sensors konfiguriert die PC-Software das Programmiergerät entsprechend den Vorgaben des Sensors (z.B. SPI-Parameter). Nun können Messdaten des Sensors am PC empfangen werden. Die Daten werden in einem 2-dimensionalen Graphen angezeigt und in einer Liste protokolliert. Mit Hilfe der GUI ist es auch möglich das EEPROM des Sensors neu zu programmieren.

Die GUI ist in der Programmiersprache C# geschrieben und greift auf Bestandteile des .NET-Frameworks 2.0 zurück. Das objektorientierte Programmierparadigma von C# eignet sich sehr gut, um wiederverwendbare GUI-Elemente („Objekte“) zu erstellen. Die automatische Speicherverwaltung in C# sorgt für eine einfachere Programmierung als bei Sprachen ohne automatisierte Speicherverwaltung wie z.B. C/C++. Zudem wird die Entwicklung einer GUI durch die angeschlossene Klassenbibliothek .NET-Framework erheblich beschleunigt. Sie bietet einen umfassenden Zugriff auf vordefinierte Klassen für die verschiedensten Bereiche der Programmierung, darunter GUI-Programmierung, Datenbankzugriff, Ein-/Ausgabe und vieles mehr. Das ältere .NET-Framework in Version 2.0<sup>2</sup> wird verwendet, damit einzelne Komponenten der GUI in anderen - bereits bestehenden - Projekten der Infineon Technologies AG wiederverwendet werden können.

Die GUI wird ausschließlich für Microsoft-Betriebssysteme ab Windows 2000 entwickelt und setzt das Microsoft .NET-Framework in Version 2.0 voraus. Um größtmögliche Kompatibilität bei Kunden herzustellen wird die GUI als 32-Bit-Anwendung zur Verfügung gestellt.

## 1.2 Der Projektumfang

Das Projekt zeigt die Entwicklung eines Evaluierungswerkzeuges für barometrische Luftdruck-Sensoren, bestehend aus Sensor, Programmiergerät und GUI. Es umfasst die Programmierung der Mikrocontroller, die in den Programmiergeräten verbaut sind. Dabei handelt es sich um einen 16- und einen 8-Bit-Mikrocontroller von Infineon. Die Firmwares werden in der Programmiersprache C geschrieben. Neben den Firmwares wird noch die grafische Benutzeroberfläche implementiert. Dies geschieht auf Basis von C# und Microsofts .NET-Framework.

---

<sup>2</sup>Microsoft stellt bereits das .NET-Framework in Version 4.0 bereit.

Hilfen bei der Programmierung der Mikrocontroller-Firmware bietet [18]. Auch wenn es primär für den 8-Bit-Mikroprozessor Zilog Z80 verfasst wurde, ist es eine große Hilfe beim Umgang mit Mikroprozessoren-/controllern. Es vermittelt grundlegende Konzepte der Programmierung, dem Hardweraufbau sowie der Funktionsweise von Mikroprozessoren und natürlich auch speziellen Themen zum Z80. Seit 2009 wird das Buch vom Autor kostenlos im Internet zur Verfügung gestellt: [19]. Als gutes Nachschlagewerk für alle Aspekte der Programmiersprache C bietet sich das Standardwerk [14] der C-Erfinder Kernighan und Ritchie an. Jedoch sei erwähnt, dass die C-Compiler für die konkreten Mikrocontroller oftmals nur eine Untermenge des ANSI-C99-Standards akzeptieren, siehe [1]. Den umfangreichen C#-Standard (4.0) erläutern die Sprachschöpfer von C# in ihrem eigenen Buch [3]. Einen praktischeren Einstieg dagegen findet die deutsche Alternative [15]. Viele praktische Beispiele erleichtern hier den Einstieg. Zudem widmen sich eigene Kapitel der Programmierung von GUIs mit Hilfe der klassischen Windows Forms und auch der neueren Windows Presentation Foundation. Unerlässlich für die Entwicklung mit dem .NET-Framework ist Microsofts Developer Network Library [17]. Die Library enthält eine umfassende API-Dokumentation aller verschiedener Versionen des .NET-Frameworks.

**Kapitel 2** stellt die konkret verwendeten Komponenten des Evaluierungsgerätes vor. Darunter Sensoren, Programmiergeräte und den PC. Wie die Hardware-Komponenten in Software abgebildet und auch angesprochen werden wird in **Kapitel 3** beschrieben. Dazu zählt die Mikrocontroller-Firmware, die grafische Benutzeroberfläche und ein Protokoll, welches beiden „Welten“ verbindet. Die grafische Benutzeroberfläche, als Ergebnis des Entwicklungsprozesses, wird in **Kapitel 4** gezeigt. Es werden die Merkmale vorgestellt, die für Kunden gedacht sind und auch jene die der Infineon Technologies AG vorbehalten bleiben. **Kapitel 5** fasst die wichtigsten Punkte dieses Projekts nochmal zusammen.

## 2 Die Komponenten des Evaluierungswerkzeuges

Wie in Kapitel 1.1 gesehen, besteht das Evaluierungswerkzeug aus einem absoluten Drucksensor, einem Programmiergerät und einer GUI am PC. Das Evaluierungswerkzeug unterstützt digitale wie analoge Drucksensoren mit unterschiedlichen Messbereichen. Bei den Programmiergeräten werden Infineons PGSISI2 und UWLink unterstützt. Die GUI läuft auf allen Rechnern die mit Microsofts Betriebssystem Windows 2000 oder höher ausgestattet sind. Alle konkret unterstützten Komponenten werden in den folgenden Abschnitten beschrieben.

### 2.1 Unterstützte Sensoren

Das Evaluierungswerkzeug unterstützt digitale und analoge barometrische Drucksensoren. Digitale Drucksensoren geben das Drucksignale über ein SPI (siehe Kapitel 2.1.1, Seite 26) aus und besitzen zudem zusätzlich noch einen integrierten Temperatursensor. Analoge Drucksensoren dagegen legen das Drucksignal als Spannungswert an einem ihrer Sensor-Pins an.

#### 2.1.1 Digitale Sensoren

Die unterstützten digitalen Drucksensoren umfassen die Baureihen KP253, KP254 und KP256. Bei allen Baureihen handelt es sich um barometrische Luftdruck-Sensoren, die nach dem kapazitivem Prinzip arbeiten.

Die Sensoren wandeln den physikalischen Druck in einen digitalen 10-Bit-Wert um (12 Bit beim KP253) und senden diesen über SPI (siehe Kapitel 2.1.1, Seite 26). Neben der Druckmessung unterstützen die digitalen Sensoren auch noch die Messung der Temperatur mittels eines integrierten Temperatursensors. Je nach erhaltenem SPI-Kommando schickt der Sensor entweder den digitalen Druckwert oder den Temperaturwert zurück.

Um den hohen Sicherheitsanforderungen im Automobilbereich gerecht zu werden, besitzen die digitalen Sensoren einen speziellen Diagnosemodus. Dieser kann mittels eines SPI-Kommandos angestoßen werden und testet dann

die Sensorzellen und die Signalpfade des Sensors. Das Ergebnis wird dann wieder über SPI übertragen.

Alle digitalen Drucksensoren verfügen über folgende charakteristischen Merkmale (siehe Tabelle 1).

Merkmal	KP253	KP254	KP256
Messbereich in kPa	60 - 165	10 - 115	60 - 165
Messgenauigkeit in kPa	$\pm 1,0$	$\pm 1,5$	$\pm 1,0$
Integrierter Temperatursensor	Ja	Ja	Ja
Auflösung in Bit (Druck und Temperatur)	12	10	10
Selbstdiagnosemodus	Ja	Ja	Ja
Anzahl der Pins	8	8	8

Tabelle 1: Merkmale der digitalen Sensoren KP253, KP254 und KP256

Die digitalen Sensoren KP253, KP254 und KP256 nutzen die selbe Pin-Konfiguration und den identischen internen Aufbau.

**Pin-Konfiguration** Abbildung 5 zeigt die Pin-Konfiguration der digitalen Drucksensoren KP253, KP254 und KP256.

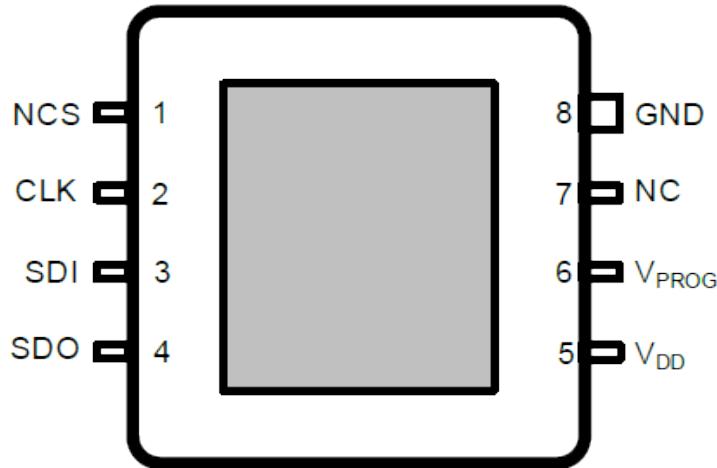


Abbildung 5: Pin-Konfiguration der digitalen Drucksensoren KP253, KP254 und KP256 [13, S. 8]

Tabelle 2 zeigt die Funktionen der einzelnen Sensor-Pins.

<b>Pin</b>	<b>Name</b>	<b>Funktion</b>	<b>Bemerkung</b>
1	NCS	Not-Chip-Select (active-low)	Kommunikation aktiviert, wenn NCS auf low.
2	CLK	Serial Clock	Externer Taktgeber für serielle Kommunikation.
3	SDI	Serial Data In	Serieller Eingang (d.h. für Mikrocontroller).
4	SDO	Serial Data Out	Serieller Ausgang (Ausgabe von Druck etc.).
5	VDD	Supply Voltage	Versorgungsspannung.
6	VPROG	Programming Voltage	Nur für EEPROM-Programmierung nötig.
7	Not Used	Pin is not bonded	Pin nicht in Verwendung.
8	GND	Ground	Masse.

Tabelle 2: Funktionen der Sensor-Pins

**Interner Aufbau** Abbildung 6 auf Seite 21 zeigt den internen Aufbau der digitalen Drucksensoren KP253, KP254 und KP256.

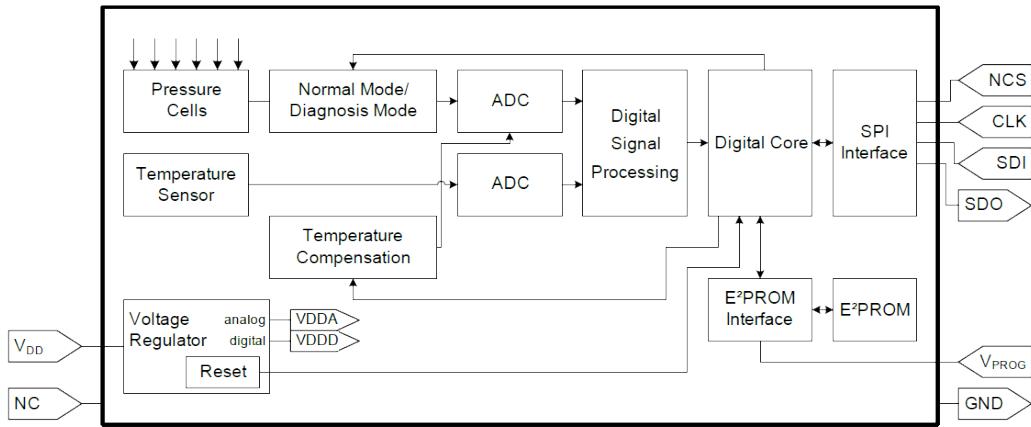


Abbildung 6: Blockdiagramm zum internen Aufbau der digitalen Drucksensoren [13, S. 9]

**Transfer-Funktion für Druck und Temperatur** Die digitalen KP25x-Sensoren sind vollständig kalibriert. Sie besitzen eine lineare Transfer-Funktion für Druck und Temperatur. Sie bildet den Druck bzw. die Temperatur auf das digitale Ausgangssignal ab (siehe Abbildung 7).

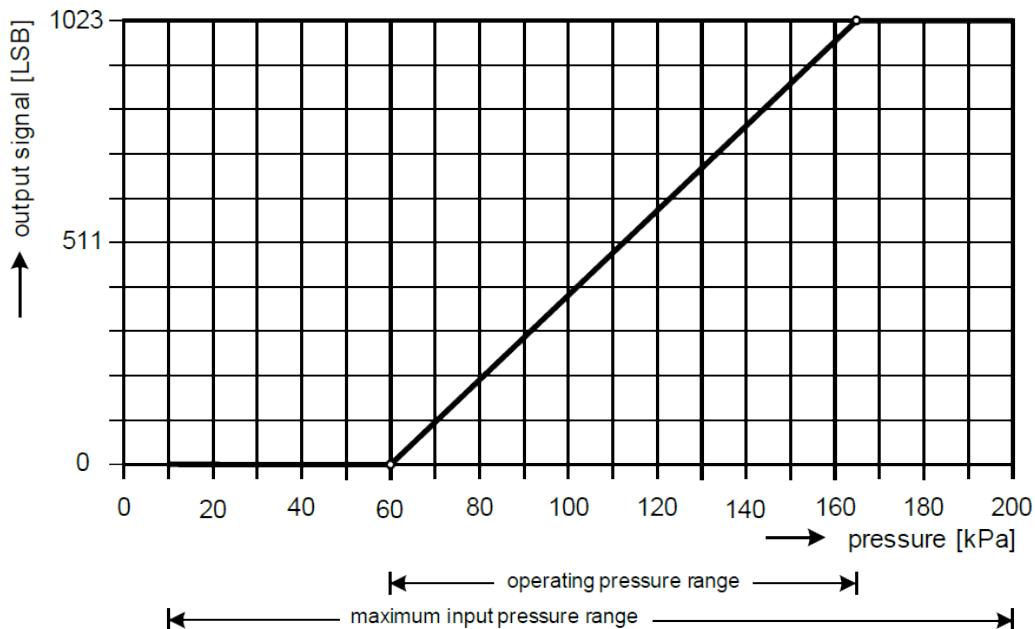


Abbildung 7: Beispiel für lineare Transfer-Funktion (Druck) des KP256 [13, S. 9]

Die Transfer-Funktion stellt sich wie folgt dar:

$$f(x) = m \times x + t, \quad \begin{aligned} m &: \text{Steigung in } \frac{\text{Bit}}{\text{kPa}} \\ x &: \text{Druck in kPa} \\ t &: \text{y-Achsenabschnitt} \end{aligned}$$

$f(x)$  liefert dann das Ausgangssignal in Bit.

Tabelle 3 liefert eine Übersicht der Transfer-Funktionen (Druck) für die Sensoren KP253, KP254 und KP256.

Sensor	Druck (kPa)		Ausgabe (Bit)		Steigung	y-Achsenabschnitt
	Min.	Max.	Min.	Max.		
KP253	60	165	0	4095	68,25	-2340
KP254	40	115	0	1023	13,64	-545,6
KP256	60	165	0	1023	9,74	-584,6

Tabelle 3: Transfer-Funktionen (Druck) für KP253, KP254 und KP256

Die Transfer-Funktion, welche die Temperatur auf das digitale Ausgangssignal abbildet, lässt sich folgendermaßen darstellen:

$$f(x) = m \times x + t, \quad \begin{aligned} m &: \text{Steigung in } \frac{\text{Bit}}{\text{°C}} \\ x &: \text{Temperatur in } \text{°C} \\ t &: \text{y-Achsenabschnitt} \end{aligned}$$

Die Transfer-Funktionen für die Temperatur der Sensoren KP253, KP254 und KP256 sind in Tabelle 4 zu sehen.

Sensor	Temp. °C		Ausgabe (Bit)		Steigung	y-Achsenabschnitt
	Min.	Max.	Min.	Max.		
KP253	-40	160	0	4095	20,48	819,2
KP254	-40	160	0	1023	5,12	204,6
KP256	-40	160	0	1023	5,12	204,6

Tabelle 4: Transfer-Funktionen (Temperatur) für KP253, KP254 und KP256

**Serial Peripheral Interface** Die digitalen Drucksensoren nutzen für die Kommunikation ein SPI mit 16 Bit breiten Worten. Die Sensoren unterstützen folgende Kommandos:

- Identifier: Die digitalen Sensoren KP253, KP254 und KP256 antworten alle mit der selben Kennung. Die Kennung enthält unter anderem den Hersteller (Infineon) und Angaben zum verwendeten Silizium des Sensors.
- Pressure: Der Druckwert wird zurückgegeben.
- Temperature: Die Temperatur wird zurückgegeben.
- Power-Down-Mode: Versetzt den Sensor in einen energiesparenden Zustand, in dem keine Messungen durchgeführt werden.
- Diagnosis: Löst einen Selbstdiagnose-Modus aus. Dabei werden Druckzellen und Signalpfade auf Funktionsfähigkeit geprüft.
- Go-Into-Test-Mode: Muss das allererste Kommando nach einem Power-Up sein, zusammen mit einer hohen Spannung (> 10 Volt) am VDD-Pin. Nur in diesem Modus kann das EEPROM programmiert werden.

Diese Kommandos werden als 16-Bit-Wort an den Sensor geschickt (siehe Abbildung 8).

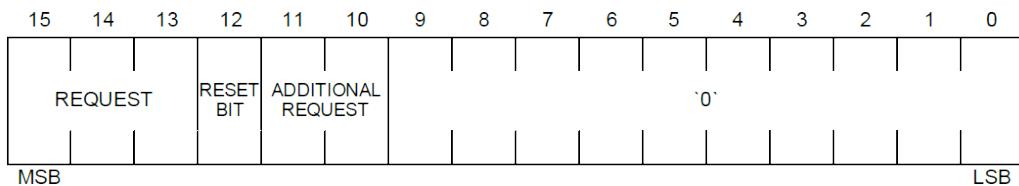


Abbildung 8: Struktur der SPI-Kommandos [13, S. 12]

Kommando	Request (Bit 15, 14 und 13)
Identifier	111
Pressure	001
Temperature	010
Power-Down-Mode	101
Diagnosis	100
Go-Into-Test-Mode	000

Tabelle 5: SPI-Kommandos für KP253, KP254 und KP256

Das Reset-Bit setzt vom Sensor erkannte Fehler zurück. Diese Fehler werden bei einer SPI-Antwort im Bitfeld „Diagnosis“ angezeigt (siehe Abbildung 9).

Der Sensor antwortet mit einem 16-Bit-Wort (siehe Abbildung 9).

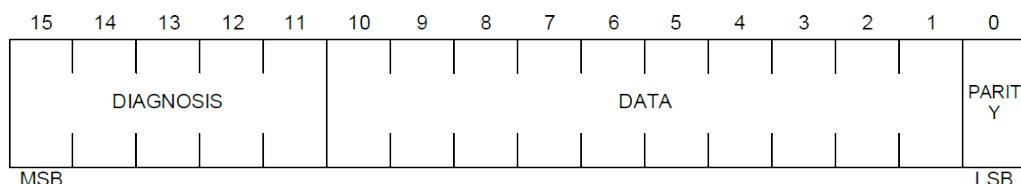


Abbildung 9: Struktur der SPI-Antworten für KP256 und KP256 [13, S. 13]

Der KP253-Sensor hat ein 12 Bit breites Datenfeld (siehe Abbildung 10).

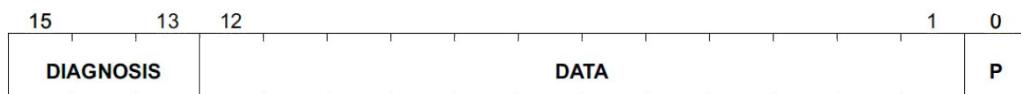


Abbildung 10: Struktur der SPI-Antworten für KP253 [11, S. 4]

Die Antwort zum Kommando Identifier unterscheidet sich von den anderen Antworten. KP253, KP254 und KP256 nutzen hierbei die selbe Struktur (siehe Abbildung 11)

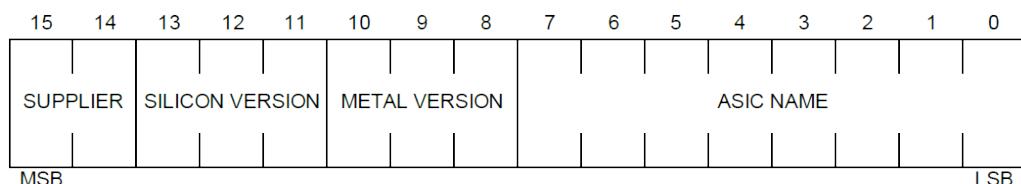


Abbildung 11: Struktur der SPI-Antworten für Kommando „Identifier“ [13, S. 13]

Die digitalen Drucksensoren erkennen automatisch folgende Fehler und zeigen dies im Bitfeld „Diagnosis“ der SPI-Antwort an (siehe Tabelle 6).

Fehler	Prio- rität	Code KP254/KP256	Code KP253
EEPROM: FEC Error	1	1 0 0 0 0	1 0 0
Acquisition Chain Failure: Diag1	2	0 1 0 0 0	0 1 0
Sensor Cell Failure: Diag2	3	0 0 1 0 0	0 0 1
Pressure out of Range: High	4	0 0 0 1 0	-
Pressure out of Range: Low	5	0 0 0 0 1	-
No Error	-	0 1 0 1 0	0 0 0

Tabelle 6: Bitfeld „Diagnosis“

Bei allen Antworten, außer Identifier, wird ungerade Parität verwendet. D.h. Das Paritäts-Bit wird so gesetzt, dass die Anzahl der Einsen ungerade ist. Auf das Kommando Identifier antwortet der Sensor mit gerader Parität.

Alle digitalen Drucksensoren zeigen folgendes Antwortverhalten: Wird ein Kommando (n) an den Sensor geschickt, erhält man die Antwort zum vorhergehenden Kommando (n - 1) zurück. Allgemein: n liefert n - 1 zurück. Die allererste Antwort des Sensors, nachdem dieser mit Spannung versorgt wird, ist der Identifier des Sensors, egal welches Kommando geschickt wird. Tabelle 7 verdeutlicht dieses Verhalten.

Sequenz	Kommando	Antwort
0	Stromversorgung für Sensor aktiviert	
1	Pressure	Identifier
2	Pressure	Pressure (Sequenz-Nr. 1)
3	Temperature	Pressure (Sequenz-Nr. 2)
4	Temperature	Temperature (Sequenz-Nr. 3)
5	Pressure	Temperature (Sequenz-Nr. 4)
...	...	...

Tabelle 7: Antwortverhalten der digitalen Drucksensoren KP253, KP254 und KP256

**Exkurs: SPI** Beim Serial Peripheral Interface (SPI) handelt es sich um ein serielles Bussystem. Es wird für eine synchrone Kommunikation in digitalen Schaltungen verwendet. Das SPI wurde von Motorola entwickelt und ist heute in fast allen Mikrocontroller vertreten. SPI arbeitet nach dem Master-Slave-Prinzip. D.h. der Master steuert die Kommunikation (agiert) und der Slave antwortet nur (reagiert). Das System wird mit einem Master und (theoretisch) beliebig vielen Slaves betrieben. Dazu werden insgesamt vier Leitungen verwendet; zwei Steuer- und zwei Datenleitungen:

- Serial Clock (SCK oder nur CLK)
- Chip Select (CS bzw. NCS für „Not Chip Select“, da diese Leitung active-low ist; d.h. Slave wird ausgewählt, wenn diese Leitung logisch „0“ ist)
- Serial Data Out (SDO)
- Serial Data In (SDI)

Für die beiden Datenleitungen SDO und SDI sind noch andere Begriffe gebräuchlich:

- MOSI (Master Out/Slave In)
- MISO (Master In/Slave Out)

Durch diese Bezeichnungen wird die Richtung des Datenflusses eindeutiger gekennzeichnet.

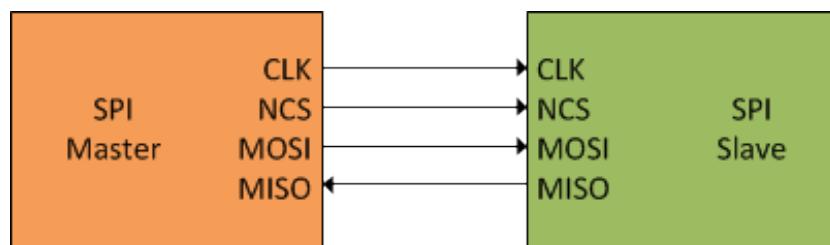


Abbildung 12: Master-Slave-Kommunikation mit vier Leitungen bei SPI

Für die Kommunikation wählt der Master den Slave, mit dem er kommunizieren möchte, mittels der NCS-Leitung aus. Dann legt er das Taktsignal an

der CLK-Leitung an und kann nun mittels SDO Daten an den Slave schicken bzw. mittels SDI Daten empfangen. Mit jedem Taktimpuls an der CLK-Leitung wird ein Bit übertragen. Die Datenlänge einer gesamten Übertragung richtet sich nach der Registerstruktur des Masters. Sie beträgt üblicherweise 8, 16 oder 32 Bit.

Bei der digitalen Kommunikation mit SPI gibt es drei wichtige Parameter:

- Reihenfolge der Bit-Übertragung (Most Significant Bit oder Lowest Significant Bit zuerst)
- Clock-Polarität (CPOL)
  - 0: Takt ist in Ruhe LOW, ein Wechsel auf HIGH zählt als steigende Taktflanke
  - 1: Takt ist invertiert: in Ruhe HIGH, ein Wechsel auf LOW zählt als steigende Taktflanke
- Clock-Phase (CPHA)
  - 0: Daten werden bei steigender Taktflanke (abhängig von CPOL) eingelesen, bei fallender ausgegeben
  - 1: Daten werden bei fallender Taktflanke eingelesen, bei steigender ausgegeben<sup>3</sup>

Diese Parameter gibt der Slave vor. Sie sind dort in der Regel unveränderlich „in Hardware gegossen“. Daraus folgt, dass die Parameter am Master eingestellt werden müssen. Frei wählbar dagegen ist am Master die Taktfrequenz. Sie richtet sich nach dem langsamstem Slave und legt die Datenrate der Übertragung fest. In der Regel wird die Taktfrequenz der Übertragung durch die Angabe einer Baudrate bestimmt.

Abbildung 13 zeigt die SPI-Parameter des KP256:

---

<sup>3</sup>Übersicht zur Einstellung von CPOL und CPHA entnommen aus: [http://www.mikrocontroller.net/articles/Serial\\_Peripheral\\_Interface](http://www.mikrocontroller.net/articles/Serial_Peripheral_Interface)

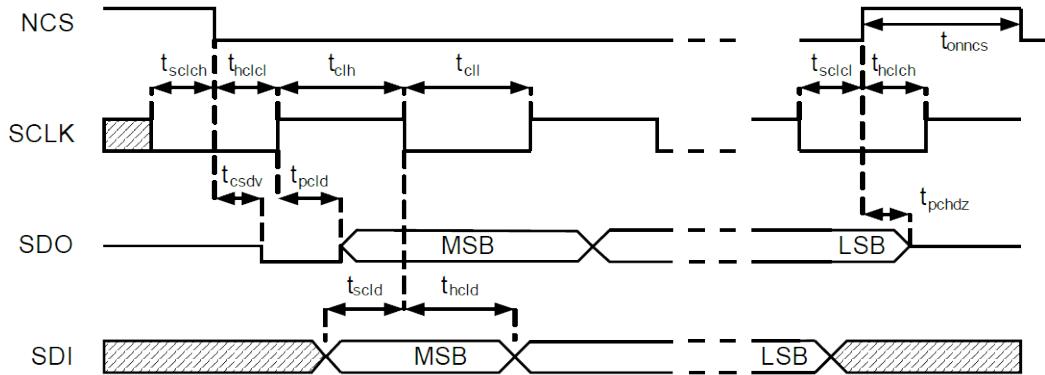


Abbildung 13: SPI-Timing des digitalen Drucksensors KP256 [13, S. 12]

Aus dem Timing-Diagramm lässt sich folgendes ablesen:

- NCS muss während der Übertragung LOW sein.
- SCLK hat als Clock-Polarität 0.
- Der Slave legt bei fallender Taktflanke das Bit an die SDO-Leitung an. Bei der nächsten steigenden Flanke kann es von dort vom Master eingelesen werden (entspricht CPHA 0).
- Der Master legt bei steigender Taktflanke das Bit an die SDI-Leitung an. Bei der nächsten fallenden Flanke kann es von dort vom Slave eingelesen werden.
- Das Most Significant Bit wird zuerst übertragen.

Alternative serielle Interfaces zu SPI sind RS232 und Inter-IC-Interface ( $I^2C$ ).

Dieser Exkurs über das SPI basiert auf [2, S. 208-209].

**Beispielverkabelung** Abbildung 14 illustriert exemplarisch die Verkabelung zwischen einem digitalen Drucksensor und einem Mikrocontroller:

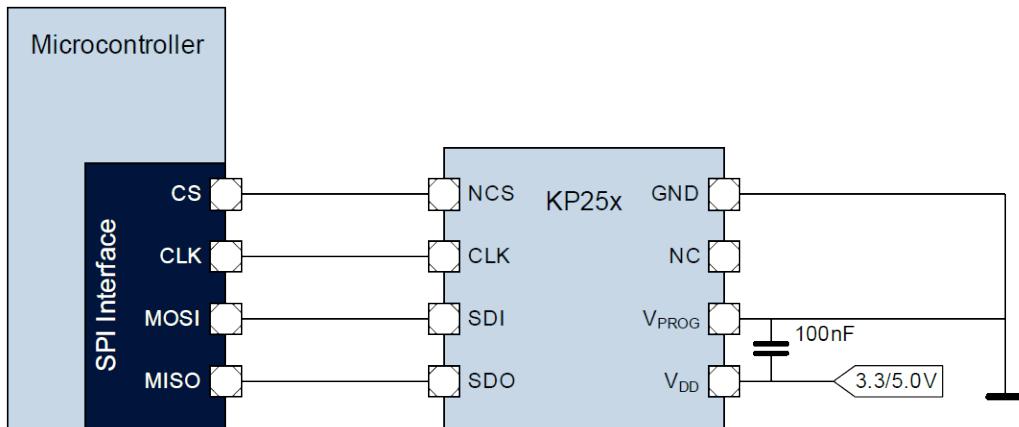


Abbildung 14: Beispielverkabelung zwischen digitalem Sensor und Mikrocontroller [13, S. 20]

**Hinweis** Alle Angaben zu den digitalen Drucksensoren KP254 und KP256 wurden den jeweiligen Datenblättern entnommen ([12] und [13]). Angaben zum KP253 entstammen der Ergänzung zum KP256-Datenblatt ([11]).

### 2.1.2 Analoge Sensoren

Die unterstützten analogen barometrischen Luftdruck-Sensoren umfassen die Baureihen KP234, KP235 und KP236. Alle Drucksensoren arbeiten nach dem kapazitivem Prinzip.

Die Sensoren wandeln den physikalischen Druck in ein analoges Ausgangssignal zwischen minimal 0 Volt und maximal 5 Volt um.

Alle analogen Drucksensoren verfügen über folgende charakteristischen Merkmale:

Merkmal	KP234	KP235	KP236
Messbereich in kPa	15 - 115	40 - 115	40 - 115
Messgenauigkeit in kPa	$\pm 1,5$	$\pm 1,2$	$\pm 1,0$
Integrierter Temperatursensor	Nein	Nein	Nein
Anzahl der Pins	8	8	8

Tabelle 8: Merkmale der analogen Sensoren KP234, KP235 und KP236

**Pin-Konfiguration** Abbildung 15 zeigt die Pin-Konfiguration der analogen Drucksensoren KP234, KP235 und KP236.

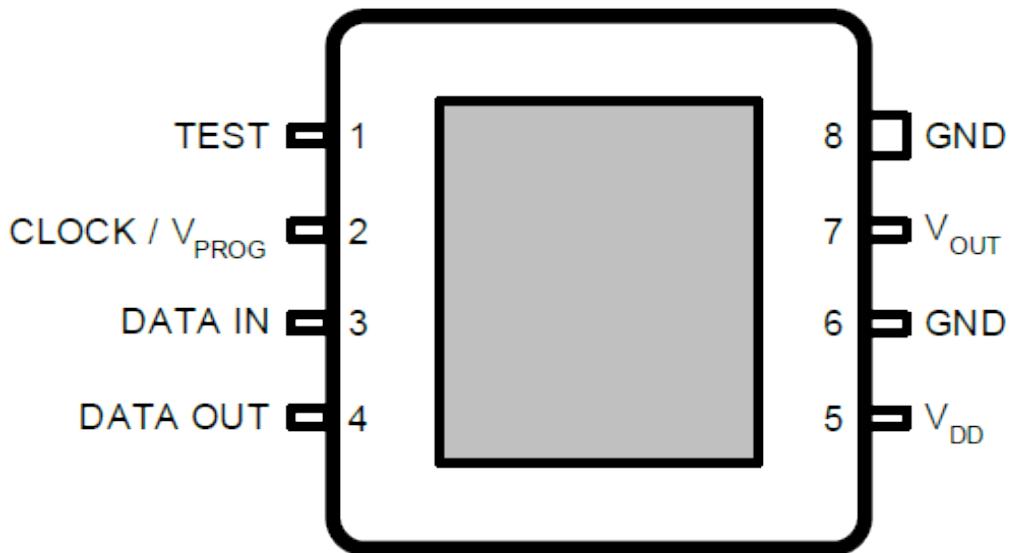


Abbildung 15: Pin-Konfiguration der analogen Drucksensoren KP234, KP235 und KP236 [9, S. 10]

Tabelle 9 zeigt die Funktionen der einzelnen Sensor-Pins.

Pin	Name	Funktion	Bemerkung
1	TEST	Test Pin	Nur für Kalibrierung genutzt.
2	CLK/VPROG	Clock/Programming Pulse	Spannung für EEPROM-Programmierung.
3	DATA IN	Serial Data In	Dateneingang EEPROM-Programmierung.
4	DATA OUT	Serial Data Out	Datenausgang EEPROM-Programmierung.
5	VDD	Supply Voltage	Versorgungsspannung.
6	GND	Circuit Ground Potential	Masse.
7	VOUT	Analog Pressure Signal	Analoges Drucksignal in Volt.
8	GND	Circuit Ground Potential	Masse.

Tabelle 9: Funktionen der Sensor-Pins

**Interner Aufbau** Abbildung 16 zeigt den internen Aufbau der analogen Drucksensoren KP234, KP235 und KP236.

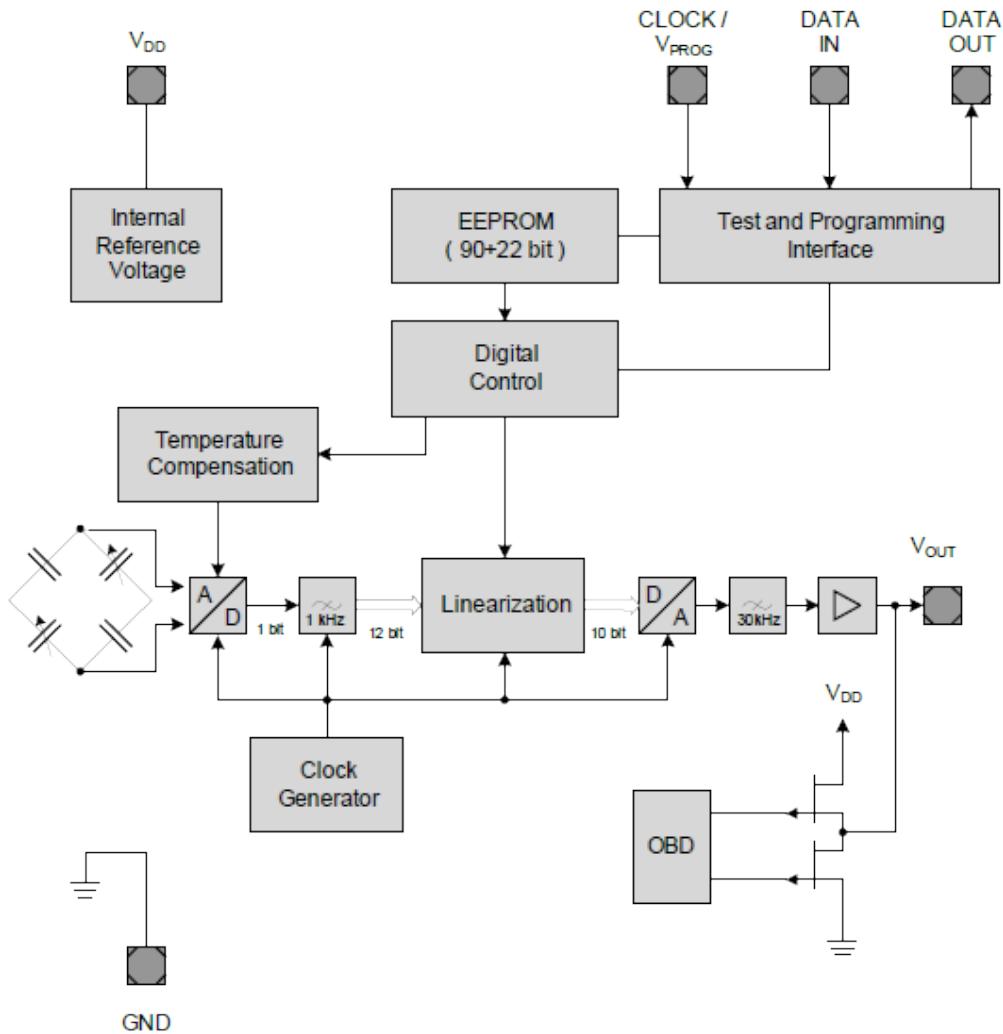


Abbildung 16: Blockdiagramm zum internen Aufbau der analogen Drucksensoren [9, S. 11]

**Transfer-Funktion für den Druck** Die analogen KP23x-Sensoren sind vollständig kalibriert. Sie besitzen eine lineare Transfer-Funktion für den Druck. Sie bildet den Druck auf das analoge Ausgangssignal ab (siehe Abbildung 17).

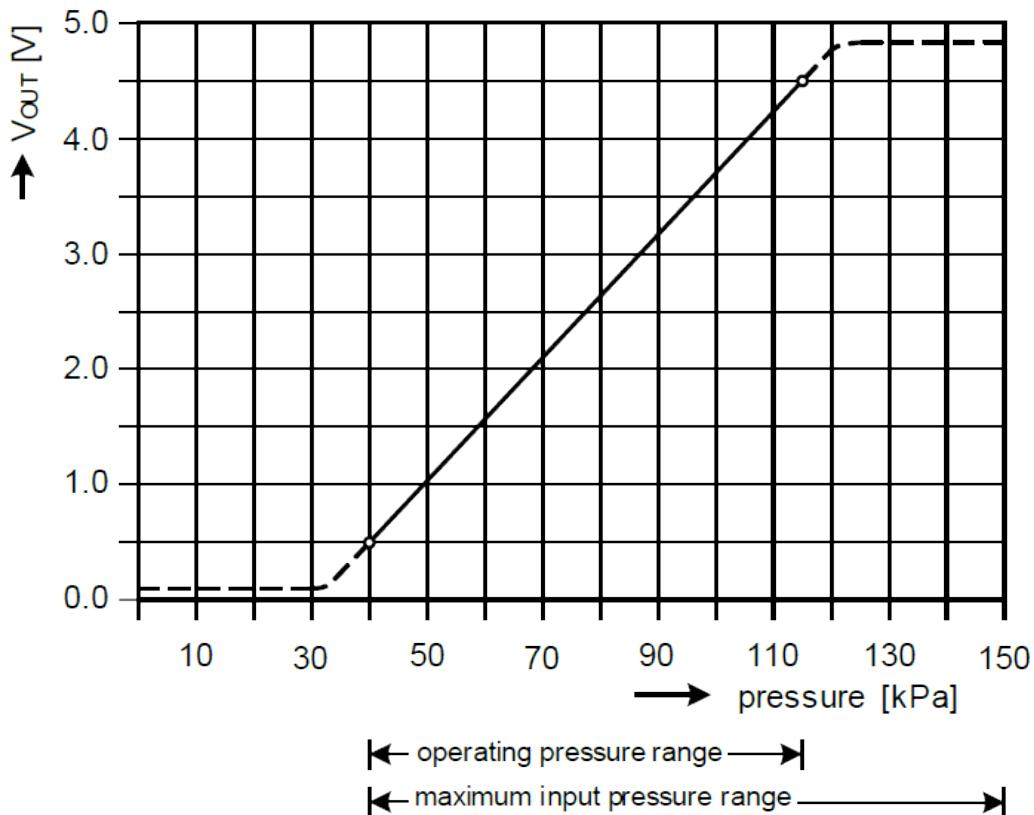


Abbildung 17: Beispiel für lineare Transfer-Funktion (Druck) des KP236 [9, S. 12]

Neben Steigung und y-Achsenabschnitt ist das Ausgangssignal abhängig von der Versorgungsspannung:

$$f(x) = V_{DD} \times (m \times x + t), \quad \begin{aligned} V_{DD} &: \text{Versorgungsspannung in Volt} \\ m &: \text{Steigung in } \frac{1}{kPa} \\ x &: \text{Druck in kPa} \\ t &: \text{y-Achsenabschnitt} \end{aligned}$$

$f(x)$  liefert dann das Ausgangssignal in Volt.

Tabelle 10 liefert eine Übersicht der Transfer-Funktionen (Druck) für die Sensoren KP234, KP235 und KP236.

Sensor	Druck (kPa)	Ausgabe (Volt)	Steigung	y-Achsenabschnitt
	Min.	Max.	Min.	Max.
KP234	15	115	0,2	4,7
KP235	40	115	0,5	4,5
KP236	40	115	0,5	4,5
			0,009	-0,095
			0,01067	-0,32667
			0,01067	-0,32667

Tabelle 10: Transfer-Funktionen (Druck) für KP234, KP235 und KP236

**Beispielverkabelung** Abbildung 18 illustriert exemplarisch die Verkabelung zwischen einem analogen Drucksensor und einem Mikrocontroller.

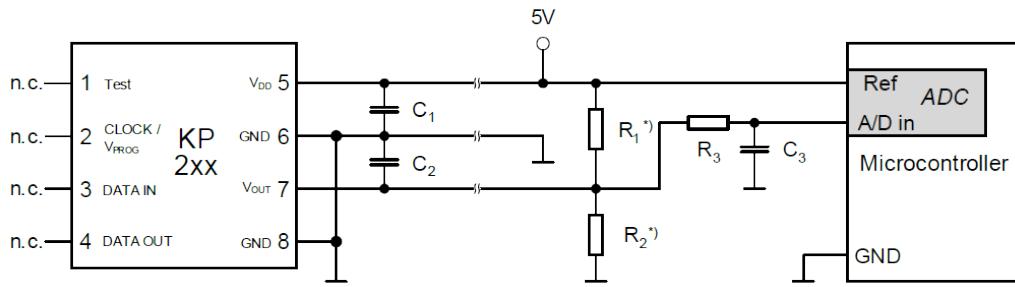


Abbildung 18: Beispielverkabelung zwischen analogem Sensor und Mikrocontroller [9, S. 17]

**Hinweis** Alle Angaben zu den analogen Drucksensoren KP234, KP235 und KP236 wurden den jeweiligen Datenblättern entnommen ( [7], [8] und [9]).

## 2.2 Unterstützte Programmiergeräte

Das Evaluierungswerkzeug unterstützt zwei verschiedene Programmiergeräte: PGSISI2 und UWLink. Sie bilden die Hardware-Schnittstelle zwischen Sensor und PC. Die PGSISI2 beherbergt einen 16-Bit-Mikrocontroller, kann digitale wie analoge Drucksensoren ansteuern und deren EEPROM programmieren. Der UWLink besitzt einen 8-Bit-Mikrocontroller und kann ausschließlich digitale Drucksensoren ansprechen. Das EEPROM kann mit dem UWLink nicht programmiert werden.

### 2.2.1 PGSISI2

Die PGSISI2 ist eine Leiterplatine. Auf der Platine sind verschiedene integrierte Schaltkreise untergebracht, darunter ein 16-Bit-Mikrocontroller und ein IC für USB/RS232-Übersetzung der Firma FTDI. Neben den Schaltkreisen bietet die PGSISI2 die seriellen Anschlüsse DE-9 (9-polige D-Sub-Buchse für RS232) und USB für die Verbindung mit einem PC auf einer Seite. Auf der anderen Seite bietet eine 25-polige D-Sub-Buchse (DB-25) Anschluss für eine Platine mit Sensor-Sockel. Die Leiterplatine mit ihren Hardware-Anschlüssen ist in einem Metallgehäuse untergebracht (siehe Abbildung 19 und 20).



Abbildung 19: Anschlüsse der PGSISI2 (Vorderseite)



Abbildung 20: Anschlüsse der PGSISI2 (Rückseite)

Die PGSISI2 wird über ein Kabel mit der Platine verbunden, auf welcher der Sensor in einem speziellen Sockel sitzt (siehe Abbildung 21). Dazu verbindet das Kabel die 25-polige D-Sub-Buchse der PGSISI2 mit der 15-poligen D-Sub-Buchse der Sensor-Platine. Über diese Verbindung werden die acht Sensor-Pins an die entsprechenden Mikrocontroller-Pins geführt.



Abbildung 21: Kabelverbindung zwischen PGSISI2 und Platine mit Sensor-Sockel

Die Versorgungsspannung für die PGSISI2 wird durch ein externes Netzteil bereitgestellt. Die Verarbeitung von Daten auf der PGSISI2 wird durch den 16-Bit-Mikrocontroller Infineon XC164CS-16F erledigt. Die Kommunikation zum Sensor erfolgt je nach Sensor entweder analog oder digital mittels SPI. Daten zum und vom PC werden über die USB-Schnittstelle gesendet. Dabei wandelt der IC von FTDI vollkommen transparent USB in RS232 um (und umgekehrt). Dies ist notwendig, da der Mikrocontroller nicht über die Funktion verfügt mittels USB zu kommunizieren. Im Folgenden alle weiteren wichtigen Eigenschaften des Mikrocontrollers.

**Mikrocontroller: XC164CS-16F** Der XC164CS-16F ist ein 16-Bit-Mikrocontroller mit fünfstufiger CPU-Pipeline. Er bietet:

- C166V2-Kern
  - 25 ns Instruction Cycle Time bei 40 MHz CPU-Takt
  - 1 CPU-Zyklus für Multiplikation, 18 bis 21 CPU-Zyklen für Division
  - 16 MB absolut adressierbarer Speicher für Code und Daten
- Speicherorganisation nach Von-Neumann-Architektur
  - 8 KB start-up ROM
  - 6 KB RAM
  - 128 KB Program Flash ROM,
- Capture/Compare-Einheiten
  - 2 16-kanalige General-Purpose-Capture/Compare-Einheiten (12 Eingabe/Ausgabe-Pins)
  - Capture/Compare-Einheit für flexible Pulswelten-Modulation
- 5 programmierbare 16-Bit-Timer
- 75 Interrupt-Typen (darunter Timer, UART, SSC und weitere) in 16 Prioritätslevel
- 14-kanaligen AD-Konverter (8- oder 10-Bit-Auflösung)
- 2 Universal Asynchronous Receiver Transmitter (UART) für RS232-Kommunikation
- TwinCAN
- 2 Synchronous-Serial-Channel-Einheiten (SSC) zur Kommunikation via SPI
- 7 Ports und insgesamt 100 Pins zur Ein-/Ausgabe, darunter jene für die Funktionen UART, SPI etc.
- Unterstützung für On-Chip-Debugging

Alle Funktionseinheiten des XC164-16-Mikrocontrollers sind in Abbildung 22 zu sehen.

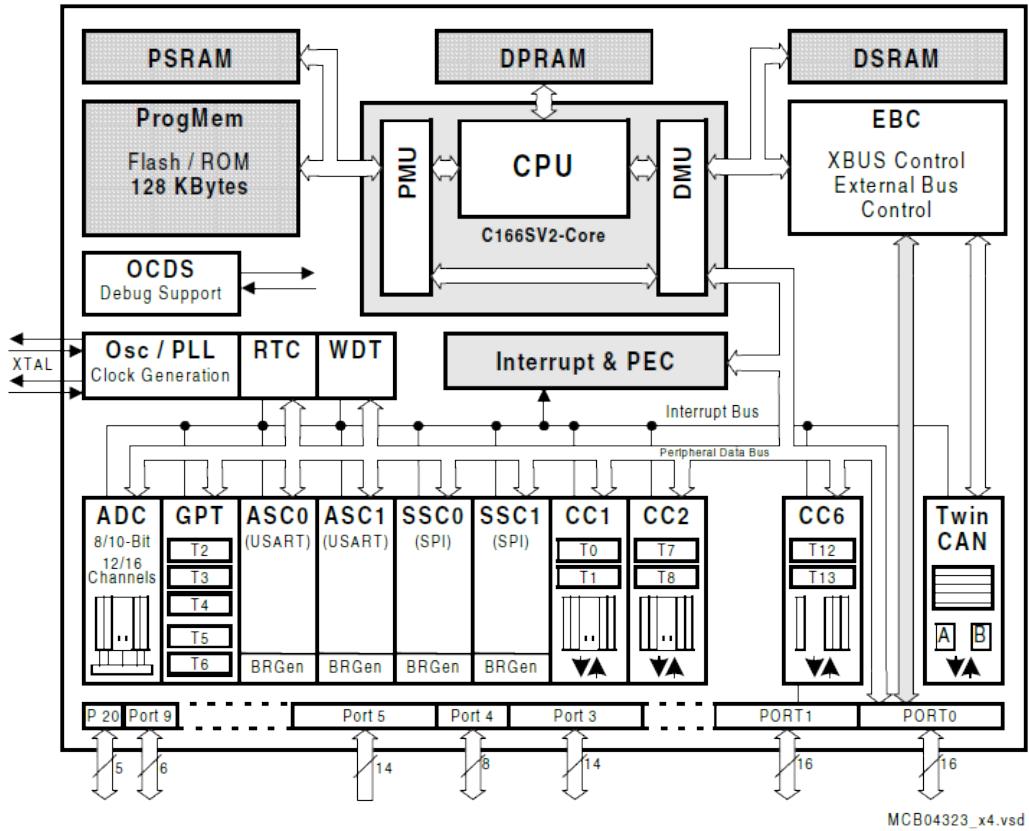


Abbildung 22: Funktionseinheiten des XC164-16-Mikrocontrollers [5, S. 24]

**Hinweis PGSISI2** Die Daten zum Infineon XC164CS-16F entstammen dem Benutzerhandbuch für die XC164-16-Mikrocontroller ([5]).

### 2.2.2 UWLink

Der UWLink ist eine Leiterplatine, Mainboard genannt, mit USB-Anschluss. Die wichtigsten Komponenten und Konnektoren des UWLinks werden in Abbildung 23 veranschaulicht.

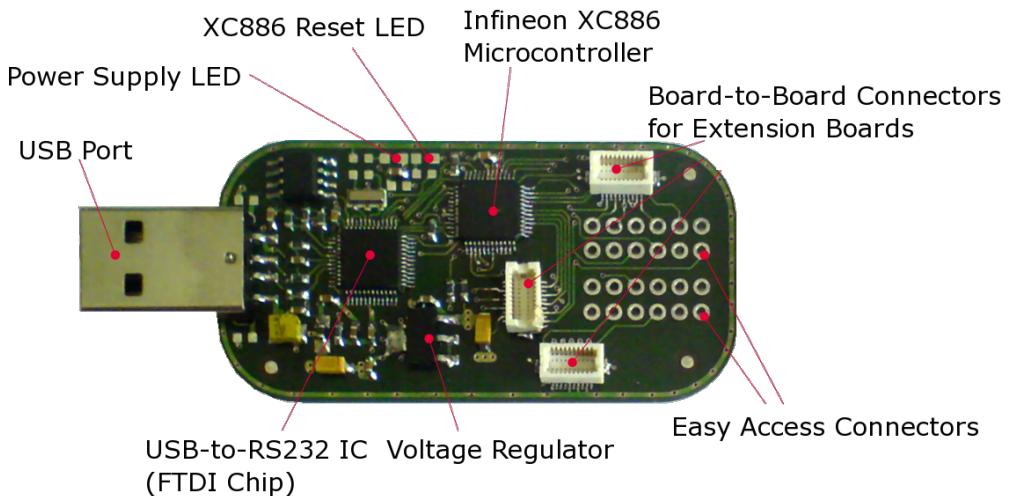


Abbildung 23: Komponenten des UWLinks [10, S. 7]

Die digitalen Drucksensoren werden über eine, Daughterboard genannte, Platine (allgemein: Extension Board) mit dem UWLink verbunden. Das Daughterboard und die Konnektoren des UWlinks verbinden die acht Sensor-Pins mit den entsprechenden Mikrocontroller-Pins (siehe Abbildung 24).

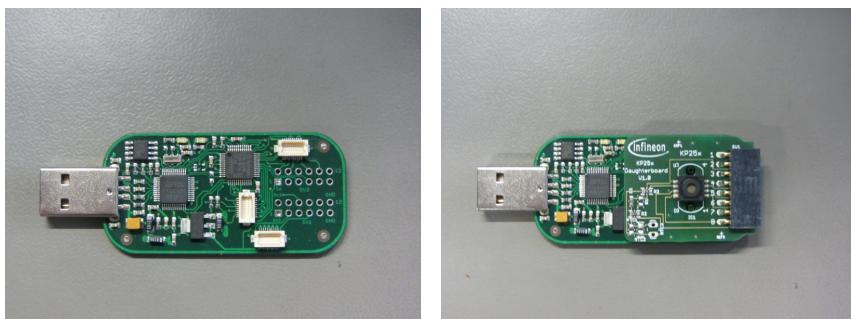


Abbildung 24: UWLink mit Daughterboard und digitalem Drucksensor

Die Versorgungsspannung für den UWLink wird durch den USB-Port des Host-PCs geliefert. Die Verarbeitung von Daten auf dem UWLink erfolgt durch den 8-Bit-Mikrocontroller Infineon XC886CLM-8FF. Die Kommunikation zum Sensor erfolgt über vier Leitungen mittels SPI. Daten zum und vom PC werden über die USB-Schnittstelle gesendet. Dabei wandelt der IC

von FTDI vollkommen transparent USB in RS232 um (und umgekehrt). Dies ist notwendig, da der Mikrocontroller nicht über die Funktion verfügt mittels USB zu kommunizieren. Alle anderen wichtigen Eigenschaften des Mikrocontroller werden im folgenden beschrieben.

**Mikrocontroller: XC886CLM-8FF** Der XC886CLM-8FF ist ein 8-Bit-Mikrocontroller und kompatibel zu Intels Standard-Mikrocontroller 8051. Der Mikrocontroller bietet:

- XC800-Kern
  - mit zwei Takten pro Maschinenzyklus (→ Speicherzugriff ohne Wartezustand)
- Speicherorganisation nach Harvard-Architektur
  - 12 KB Boot-ROM
  - 256 Byte on-chip RAM
  - 1536 Byte on-chip XRAM
  - 32 KB on-chip Program Flash ROM
- Capture/Compare-Einheit mit zwei unabhängigen Timern zur Pulsweiten-Modulation
- 3 programmierbare 16-Bit-Timer
- 15 Interrupt-Typen (darunter Timer, UART, SSC und weitere) in vier Prioritätslevel
- 8-kanaligen 10-Bit-AD-Konverter
- 2 UARts für RS232-Kommunikation
- MultCAN
- SSC zur Kommunikation via SPI
- 6 Ports und insgesamt 48 Pins für Ein-/Ausgabe, darunter jene für die Funktionen UART, SPI etc.
- Unterstützung für On-Chip-Debugging

Alle Funktionseinheiten der XC88xCLM-Familie sind in Abbildung 25 zu sehen.

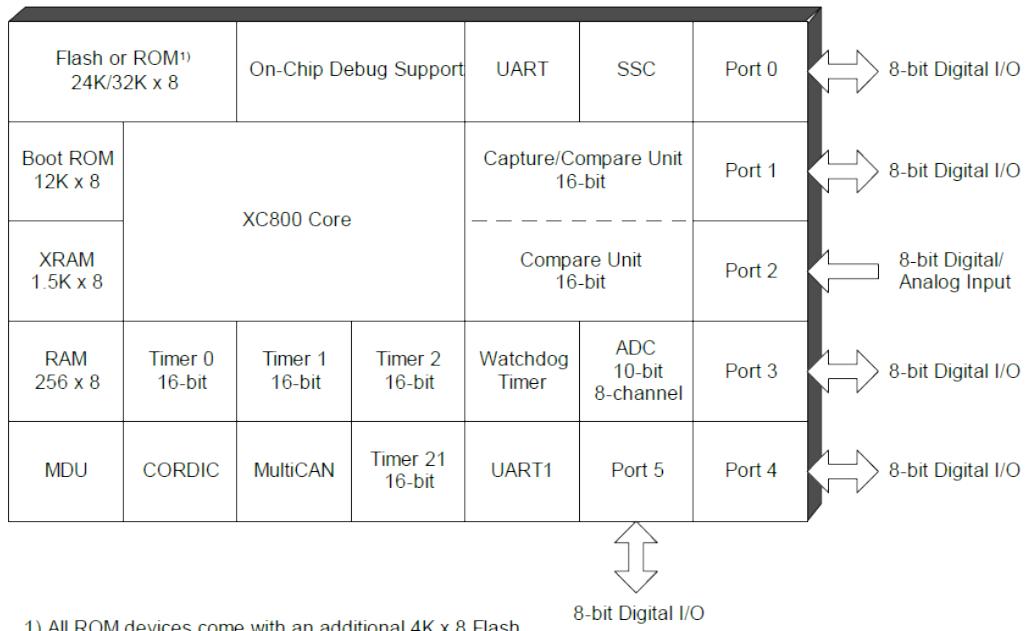


Abbildung 25: Funktionseinheiten der XC88xCLM-Mikrocontroller [6, S. 8]

**Hinweis UWLink** Die Daten zum Infineon XC886CLM-8FF entstammen dem Datenblatt für die 8-Bit-Mikrocontroller-Familien XC886/888CLM ([6]).

## 2.3 PC (Grafische Benutzeroberfläche)

An den PC, auf dem die GUI betrieben wird, werden bestimmte Mindestanforderungen an Hard- und Software gestellt.

### 2.3.1 Hardware

Folgende Hardware wird vorausgesetzt:

- 32-Bit-Prozessor mit 900 MHz oder mehr, um ausreichend schnelle Bearbeitung aller anfallender (Sensor-) Daten zu gewährleisten.
- 30 MB RAM, damit in Echtzeit die letzten 200 Sensordaten in einem Graphen angezeigt werden und die letzten 10.000 Sensordaten in einer Liste protokolliert werden können.
- 5 MB Festplattenspeicherplatz für Installationsdateien.
- Mehr Festplattenspeicherplatz, wenn Messungen laufen, um die Sensordaten direkt auf Festplatte zu speichern zur Vorbeugung von Hard- und Software-Ausfällen.

### 2.3.2 Software

Die GUI ist in der Programmiersprache C# geschrieben und greift auf Bestandteile des .NET-Frameworks 2.0 zurück. Folgende Mindestanforderungen werden an die Softwareumgebung gestellt:

- Microsoft Windows 2000
- .NET Framework 2.0

### 3 Der Softwareentwicklungsprozess

Nach Auswahl der Hardware-Komponenten für das Evaluierungswerkzeug müssen diese mittels Software programmiert werden, sodass eine reibungslose Kommunikation untereinander möglich ist. Zu den Komponenten zählen:

- Digitale Drucksensoren der Baureihen KP253, KP254 und KP256 sowie analoge Drucksensoren der Baureihen KP234, KP235 und KP236,
- PGSISI2 und UWLink als Programmiergeräte und
- ein PC mit GUI, in der unter anderem die Messdaten angezeigt werden und das EEPROM der Sensoren programmiert werden kann.

Die Schnittstelle zum Sensor ist durch seine Bauart vorgegeben. Entweder digital via SPI oder analog durch explizites Ansprechen der Sensor-Pins.

Der Mikrocontroller der PGSISI2 muss so programmiert werden, dass er zum Einen mit dem integrierten SSC-Modul über SPI Druckwerte von digitalen Sensoren auslesen kann. Mittels SPI ist es auch möglich das EEPROM der digitalen Sensoren zu programmieren. Zusätzlich muss für die EEPROM-Programmierung am VPROG-Pin eine vordefinierte Programmierspannung angelegt werden. Um zum Anderen Druckwerte von analogen Sensoren einzulesen muss der Mikrocontroller einen integrierten ADC nutzen. Dieser muss den VOUT-Pin des analogen Drucksensors auslesen. Zum Programmieren des EEPROMs analoger Sensoren werden die Pins DATA IN, DATA OUT und CLK/VPROG benutzt (vergleiche Abbildung 15). Der Mikrocontroller des UWLinks unterstützt nur digitale Sensoren und verwendet dazu ausschließlich sein integriertes SSC-Modul.

Die Kommunikation mit dem Sensor — über den Mikrocontroller des Programmiergeräts — wird letztendlich durch die GUI initiiert. Dazu muss sie in der Lage sein den Mikrocontroller des Programmiergeräts entsprechend den Vorgaben des Sensors zu konfigurieren. Konfiguriert werden muss unter anderem das SSC des Mikrocontrollers, welches die SPI-Kommunikation handhabt. Jeder Sensor kann prinzipiell unterschiedliche Parameter für die SPI-Kommunikation verwenden. Die vorliegenden digitalen Sensoren KP253, KP254 und KP256 nutzen jedoch alle identische Parameter. Damit die GUI überhaupt „Anweisungen“ an den Mikrocontroller senden kann, ist eine gemeinsame Sprache notwendig: ein einfaches Protokoll. Dies ermöglicht der

GUI Befehle direkt an den Mikrocontroller zu senden (z.B. „SSC konfigurieren“) oder auch Daten über den Mikrocontroller an den Sensor zu übertragen (z.B. SPI-Kommando zum Druckauslesen). Und desweiteren auch Daten vom Mikrocontroller oder dem Sensor zu empfangen.

Die Anforderungen an die Mikrocontroller-Firmwares und an die GUI gehen von den verwendeten Drucksensoren aus. Mikrocontroller-Firmwares und GUI müssen so gestaltet sein, dass sie letztendlich mit dem Sensor kommunizieren können. Aus diesem Grund müssen zuallererst die Datenblätter aller Sensoren gelesen werden. Daraus wird ersichtlich welche Mikrocontroller-Module für die Kommunikation genutzt werden können — wie bereits erwähnt ist das z.B. das SSC-Modul. Der nächste wichtige Baustein sind die Datenblätter und Benutzerhandbücher der Mikrocontroller. Sie erläutern den Umgang mit den Mikrocontrollern. Die GUI unterliegt den geringsten Einschränkungen, da sie auf einer universalen Datenverarbeitungsanlage, dem PC, läuft.

Englisch wurde als Sprache für den gesamten Entwicklungsprozess, bestehend aus Firmwares, GUI und (API-) Dokumentationen, gewählt. Dies erleichtert einem global operierenden Unternehmen wie der Infineon Technologies AG die Wiederverwendbarkeit von Code und anderen Komponenten.

Die Entwicklung erfolgt auf einem PC mit Intel-Core-Duo-Prozessor und 4 GB Arbeitsspeicher. Als Betriebssystem steht Microsoft Windows 7 in der 32-Bit-Ausführung zur Verfügung.

Alle Dateien dieses Projekts werden unter Versionsverwaltung gestellt. Dazu wird die zentrale Source-Code-Management-Software Subversion (SVN) verwendet. Als SVN-Client wird TortoiseSVN genutzt.

### 3.1 Die Mikrocontroller-Firmware

Es soll eine Firmware für den 16-Bit-Mikrocontroller XC164CS-16F und den 8-Bit-Mikrocontroller XC886CLM-8FF jeweils in der Programmiersprache C entwickelt werden. Sie soll in diesem Projekt die Kommunikation zwischen einem Drucksensor und einem PC ermöglichen. Dazu werden einzelne Mikrocontroller-Module (u.a. UART für RS232 und SSC für SPI) verwendet.

Es empfiehlt sich aber die Firmware so zu gestalten, dass sie bzw. Teile daraus

auch in Szenarien mit ähnlichem Aufbau wiederverwendet werden können. Der Aufbau sollte also dem Muster Sensor  $\longleftrightarrow$  Mikrocontroller-Schnittstelle  $\longleftrightarrow$  PC entsprechen. Die Wiederverwendbarkeit von Code — bzw. Wissen — kann die Kosten und die Entwicklungszeit von zukünftigen Projekten erheblich senken. Solch eine Wiederverwendbarkeit kann durch eine Modularisierung der vorhandenen Hardware-Module des Mikrocontrollers erreicht werden. Alle notwendigen Hardware-Module werden unabhängig voneinander implementiert. In C geschieht dies durch die Deklaration in einer Header-Datei mit der Endung .h und der (konkreten) Definition in einer Datei mit der Endung .c.

Als Beispiel: Für das Hardware-Modul SSC, das die SPI-Kommunikation mit einem digitalen Sensor übernimmt, wird die Header-Datei *SSC.h* angelegt. In ihr werden alle Methoden deklariert, die für die SPI-Kommunikation notwendig sind, darunter „senden“, „empfangen“ und „SPI-Parameter konfigurieren“. Die Methodendefinitionen sind dann in der c-Datei zu finden. Eine solche Modularisierung ermöglicht nun das Hardware-SSC-Modul des Mikrocontrollers in anderen Projekten wiederzuverwenden. Dazu muss einfach die entsprechende *h*- und *c*-Datei in das Projekt kopiert werden bzw. etwas eleganter: Referenzen auf beide Dateien in das Projekt einfügen. Die Abbildung der Hardware-Module des Mikrocontrollers in *h*- und *c*-Dateien wird in Grafik 26 nochmal verdeutlicht.

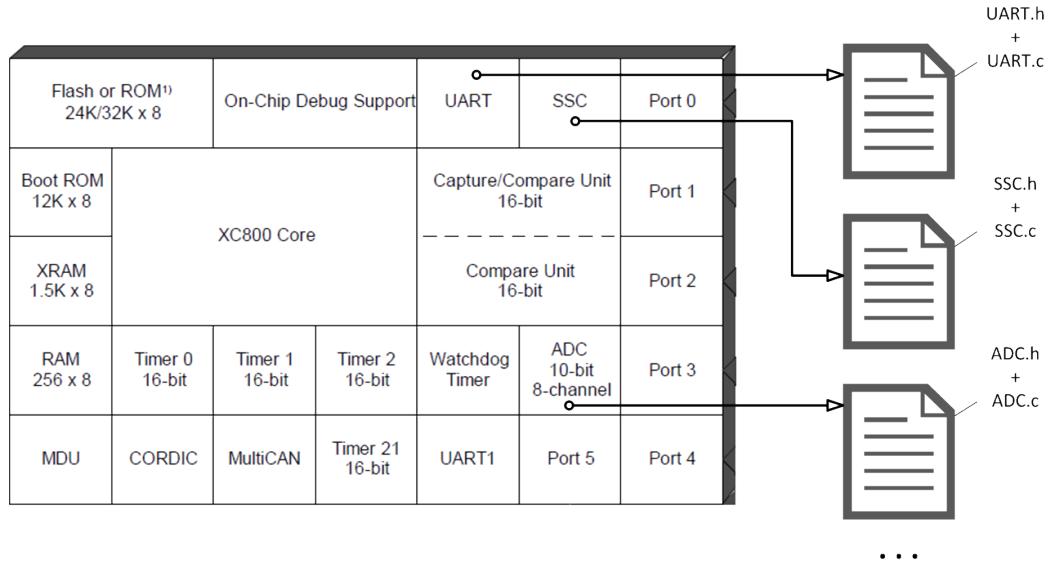


Abbildung 26: Abbildung der Mikrocontroller-Module in Software

Bei der Programmierung von Mikrocontrollern muss die zu Grunde liegende Hardware-Architektur stärker berücksichtigt werden, als bei der Anwendungsprogrammierung mit modernen Hochsprachen. Darunter unter anderem:

- **Registersatz:** Welche Register besitzt der Mikrocontroller? Und welche Funktion haben die einzelnen Register? Mikrocontroller-Komponenten wie das SSC (mit Clock-Polarität usw.) werden z.B. über einzelne Register konfiguriert.
- **Flags:** Innerhalb eines Registers hat jedes Bit eine eigene Bedeutung. Diese wird dem Datenblatt des Mikrocontrollers entnommen.
- **Speichermodell:** Handelt es sich um Harvard- oder Von-Neumann-Architektur? Müssen Speicheradressen vorher in Register geladen werden (RISC oder CISC)?
- **Adressierungsarten:** Absolute oder indirekte (Basis und Offset) Addressierung?
- **Stackfunktionen:** Wie werden Aufrufe von Unterprogrammen gehandhabt?
- **Zeitverhalten:** Wie viele Takte benötigt ein Befehl?
- **Interruptsystem:** Welche Interrupts gibt es? Wie sind Interrupts handzuhaben?

Die Programmierung der Mikrocontroller ist in Assembler möglich. Jedoch gestaltet sich dies oftmals langwierig, fehleranfällig und schlecht wartbar. Aus diesem Grund erfolgt die Programmierung der Firmwares in C. Durch das Zusammenspiel von Compiler, Assembler und Linker entsteht dabei der ausführbare Maschinencode. Dadurch wird die Entwicklung stark vereinfacht, da Aspekte wie Speichermodell und Stackfunktionen nicht mehr separat betrachtet werden müssen, sondern beim Kompiliervorgang automatisch berücksichtigt werden.

Um die Entwicklung komfortabler und schneller zu machen, wird auf die integrierte Entwicklungsumgebung (IDE) Keil<sup>4</sup> µVision in Version 4 zurückgegriffen. Die IDE ist speziell für die Entwicklung von eingebetteten Systemen gedacht. Sie bietet eine einheitliche Oberfläche für Verwaltung, Bau,

---

<sup>4</sup>Seit 2005 zum englischen Prozessor- und Halbleiterhersteller ARM Limited gehörend.

Quelltextbearbeitung und Debugging von eingebetteten Systemen. Über separat installierbare Toolchains werden Compiler, Assembler und Linker für verschiedene Mikrocontroller nachgerüstet. Für den 16-Bit-Mikrocontroller wird die C166-Toolchain verwendet. Der 8-Bit-Mikrocontroller benötigt die C51-Toolchain. Tabelle 11 gibt einen genaueren Überblick zu den verwendeten Versionen. Zudem stellen die Toolchains gültige Start-Konfigurationen für die Mikrocontroller bereit, sodass diese sofort verwendet werden können. Diese Start-Konfigurationen enthalten Assembler-Befehle zur Konfiguration von Taktfrequenz, Speicher (Startadressen der einzelnen Datensegmente) und Stack (Adresse des Stackpointers).

Werkzeug	C166		C51	
	Executable	Version	Executable	Version
C Compiler	C166.exe	7.04	C51.exe	9.02
Assembler	A166.exe	5.33	A51.exe	8.02a
Linker	L166.exe	5.25	BL51.exe	6.22
Librarian	LIB166.exe	4.26	LIB51.exe	4.29
Hex Converter	OH166.exe	4.7a	OH51.exe	2.6
CPU DLL	S166.dll	3.77.0.0	S8051.dll	3.79.0.1

Tabelle 11: Versionsübersicht der Keil-Toolchains

Neben der IDE µVision 4 wird für die Programmierung der Mikrocontroller die Software Infineon Device Access Server (DAS) benötigt. Es stellt API-Methoden unter anderem zum Debuggen, Testen und Programmieren (das Schreiben der Firmware in den Flash bzw. ROM-Speicher) bereit und wird von µVision, für den Benutzer nicht sichtbar, benutzt. Infineon DAS sorgt für einen einheitlichen Zugriff auf Infineon-Geräte (Mikrocontroller usw.). Debuggen und das Schreiben der Firmware in den Mikrocontroller-Speicher sind so einfach durch einen Knopfdruck in µVision verfügbar.

Die Mikrocontroller-Firmwares werden so generisch wie möglich entwickelt, damit sie auch in anderen Projekten verwendet werden können. Die einzelnen Methoden der Firmware stellen somit ein Application Programming Interface (API) dar. Für diese API-Dokumentation wird Doxygen verwendet, da C keine Möglichkeit der Inline-Dokumentation bietet. Durch spezielles Markup können hierbei Dateien, Variablen, Makros und Methoden dokumentiert werden. Dieses Markup wird extrahiert und in verschiedene Ausgabeformate, darunter HTML und L<sup>A</sup>T<sub>E</sub>X, übersetzt. Listing 1 zeigt die Kommentierung

der Datei *Main.c* für den 8-Bit-Mikrocontroller des UWLinks.

```
1  /**
2  * \file Main.c
3  *
4  * \brief This project is intended to configure and use
5  *        UWLink's microcontroller XC886CLM.
6  *
7  * So far only the GPIO, SSC and UART modules are configured
8  * and used.
9  *
10 * Data flow:
11 * -# PC sends data/commands to microcontroller using the USB
12 *    port.
13 * -# Microcontroller receives this data by using its UART
14 *    module.
15 * -# A SSC command is funneled to the connected device (
16 *    sensor) by using the SSC module.
17 * -# Device (sensor) data comes back to microcontroller via
18 *    the SSC module.
19 * -# Microcontroller sends device (sensor) data back to PC
20 *    by using the UART module.
21 * -# PC receives data via the USB port.
22 * .
23 * UWLink's microcontroller (XC886CLM) uses as byte ordering
24 * big endian,
```

\* contrary to other Intel processors, which uses little  
\* endian!

\*  
\* Therefore all used methods rely on the big endian byte  
\* order!

```
21 * \ingroup Main
22 * @{
23 * /
24 ...
```

Listing 1: Doxygen-Kommentar der Datei *Main.c* (UWLink)

Bevor nun die Implementierung der einzelnen Mikrocontroller-Module beginnt muss noch die Architektur geklärt werden, in welche die Module eingebettet werden. Die gesamte Architektur für beide Mikrocontroller ergibt sich aus der Rolle der Programmiergeräte innerhalb des Evaluierungswerkzeuges. Das Programmiergerät mit dem jeweiligen Mikrocontroller ist die Schnittstelle zwischen Sensor und PC. Der PC konfiguriert das Programmiergerät so, dass er mit diesem den Sensor ansprechen kann. Das Programmiergerät agiert ähnlich einem Slave innerhalb des Master-Slave-Prinzips. Er wartet

auf Anweisungen und reagiert nur. Beim Einschalten oder nach einem Reset konfiguriert er seine Module mit bekannten Standardeinstellungen und stellt dann seine Dienste (Module) zur Verfügung. Der Pseudocode in Listing 2 verdeutlicht die Funktionsweise der Firmware.

```
1 void main()
2 {
3     // Initialize hardware modules.
4     ...
5     while(TRUE)
6     {
7         // Receive commands from PC.
8         ...
9         switch(command)
10        {
11             // Execute command.
12             case CMD_GET_FIRMWARE_VERSION:
13                 ...
14                 break;
15             case CMD_CONFIG_HARDWARE_SSC:
16                 ...
17                 break;
18         }
19         // Send data back to PC.
20         ...
21     }
22 }
```

Listing 2: Pseudocode für Firmware-Architektur

Nun kann die konkrete Implementierung der Firmwares beginnen. Zuerst wird der 8-Bit-Mikrocontroller des UWLinks programmiert. Dieser benötigt insgesamt nur zwei Module, um digitale Drucksensoren anzusteuern und mit dem PC zu kommunizieren. Der geringere Funktionsumfang hat den psychologischen Vorteil, dass schnell Fortschritte bei der Entwicklung der Firmware zu sehen sind und Best-Practices gesammelt werden können für die umfangreichere Firmware des 16-Bit-Mikrocontrollers der PGSISI2. Dieser benötigt mehr Module, da mit ihm digitale wie analoge Sensoren angesteuert werden sollen und deren EEPROM programmiert werden soll.

### 3.1.1 Firmware für XC886CLM-8FF (UWLink)

Durch Anlegen eines neuen µVision-Projekts mit der „CPU“ XC886CLM-8FF als Vorgabe wird eine gültige Start-Konfiguration für den Mikrocontroller erzeugt und in das Projekt eingebunden. Nun können weiterer Quelltext-Dateien angelegt und in einer frei wählbaren Ordnerstruktur abgelegt werden. Für den 8-Bit-Mikrocontroller wird folgende Projektstruktur verwendet:

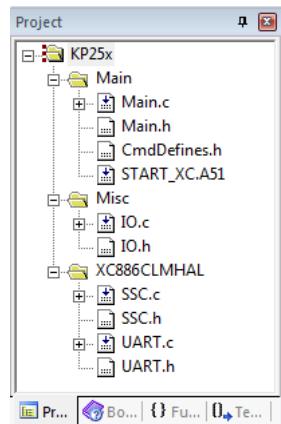


Abbildung 27: Projektstruktur für 8-Bit-Mikrocontroller

- **KP25x**

- **Main**

- \* *Main.c*: Struktur gemäß Listing 2.
    - \* *Main.h*: Enthält unter anderem Makros (z.B. `#define FALSE 0`), die in allen anderen Quelldateien verwendet werden und Adressen von speziellen Mikrocontroller-Registern.
    - \* *CmdDefines.h*: Enthält alle Befehle die der PC an die Firmware senden kann als Makros, z.B. `#define CMD_GET_FIRMWARE_VERSION 0x04`.
    - \* *START\_XC.A51*: Von µVision bereitgestellte Start-Konfiguration für den Mikrocontroller.

- **Misc**

- \* *IO.c/h*: Definitionen und Deklarationen für die Konfiguration der Ein-/Ausgabe-Ports des Mikrocontrollers.  
Ein Pin der Ports wird beispielsweise benötigt um bei der SPI-Kommunikation den NCS-Pin des Sensors zu triggern.

### – XC886CLMHAL

- \* *SSC.c/.h*: Alle notwendigen Definition/Deklarationen für das SSC-Modul zur Kommunikation mit dem Sensor.
- \* *UART.c/.h*: Alle notwendigen Definitionen/Deklarationen für das UART-Modul zur Kommunikation mit dem PC.

Durch einbinden der Dateien *Main.c*, *Main.h* und *START\_XC.A51* kann das Projekt bereits gebaut werden und zum Testen auf den Mikrocontroller übertragen werden. Von hier an kann nun jedes einzelne Hardware-Modul des Mikrocontrollers in kleinen Schritten implementiert werden.

Für die Implementierung der einzelnen Module bietet Infineon mit dem Digital Application virtual Engineer (DAVE) Unterstützung an. Bei DAVE handelt es sich um einen Code-Generator für C. Über eine grafische Oberfläche kann der Mikrocontroller gewählt werden und dann die zu verwendenden Hardware-Module. Daraufhin wird der C-Code für das entsprechende Modul generiert. Beim SSC-Modul sind das z.B. Methoden zum Senden und Empfangen. Die Unterstützung für verschiedene Mikrocontroller kann bei DAVE durch DIP-Dateien (DAVE Integration Package) nachgerüstet werden. Der von DAVE generierte Code ist funktionsfähig, jedoch sollte er zur besseren Lesbarkeit von Hand überarbeitet werden.

Beim 8-Bit-Mikrocontroller des UWLinks werden die Module IO, SSC und UART implementiert. Am Beispiel des SSC-Moduls wird das Vorgehen exemplarisch aufgezeigt.

**SSC-Modul beim XC886CLM-8FF** Die GUI am PC soll die Möglichkeit haben das SSC nach den Vorgaben der digitalen Drucksensoren zu konfigurieren (Clock-Polarität usw.) und dann Kommandos an den Sensor zu schicken. Dazu schickt die GUI gemäß dem Protokoll aus Kapitel 3.2 die Anfrage an die Firmware, welche diese abarbeitet.

Die möglichen Anfragen der GUI werden in der Header-Datei *CmdDefines.h* definiert (siehe Listing 3).

```

1 ...
2 /**
3 * \brief Configure the "High-Speed Synchronous Serial
4 * Interface (SSC)".
5 *
6 * Example byte sequence to configure the SSC module for
7 * KP256 sensor:
8 * 19 11 01 01 01 01 01 01 00 00 FF 00 FF 01 FF 07 FF 00 0B
9 */
10 #define CMD_CONFIG_HARDWARE_SSC 0x19
11 /**
12 * \brief Send data via SCC to the connected device (sensor)
13 * and return device's response.
14 *
15 * Example byte sequence to acquire pressure from KP256
16 * sensor:
17 * 1A 02 20 00
18 */
19 #define CMD_RXTX_HARDWARE_SSC 0x1A
20 ...

```

Listing 3: CmdDefines.h

In der Main-Methode in der Datei *Main.c* werden die Anfragen der GUI durch das UART-Modul eingelesen und an die entsprechenden Methoden des SSC-Moduls weitergereicht (siehe Listing 4).

```

1 ...
2 void main()
3 {
4     // Variables needed by the protocol. Used to store a
5     // receiving frame and a sending frame.
6     // Receiving frame:
7     ubyte RxCommand;
8     ubyte RxPayloadLength;
9     ubyte RxPayload[MAIN_MAX_RX_PAYLOAD];
10    // Sending frame ("RxCommand" is just sent back to PC):
11    ubyte TxOK;
12    ubyte TxPayloadLength;
13    ubyte TxPayload[MAIN_MAX_TX_PAYLOAD];
14
15    // Index variables for using in loops.
16    ubyte i;
17
18    // Initialize hardware modules.
19    ...

```

```
20 while (TRUE)
21 {
22     // Read in the parts of the agreed protocol.
23     RxCommand = UART_GetData();
24     RxPayloadLength = UART_GetData();
25
26     if (RxPayloadLength > MAIN_MAX_RX_PAYLOAD)
27         RxPayloadLength = MAIN_MAX_RX_PAYLOAD;
28
29     for(i = 0; i < RxPayloadLength; i++)
30         RxPayload[i] = UART_GetData();
31     ...
32
33     // Process the requested command.
34     switch(RxCommand)
35     {
36         ...
37         case CMD_CONFIG_HARDWARE_SSC:
38             if (RxPayloadLength == 17)
39             {
40                 SSC_Init_Parms *SSC_InitParams = (SSC_Init_Parms
41                     *) (RxPayload);
42                 SSC_Init(SSC_InitParams);
43
44                 TxOK = TRUE;
45                 TxPayloadLength = 0;
46             }
47             break;
48         ...
49     }
50
51     // Send data back to PC.
52     if (!TxOK)
53         TxPayloadLength = 0;
54
55     UART_SendData(RxCommand);
56     UART_SendData(TxOK);
57     UART_SendData(TxPayloadLength);
58     for(i = 0; i < TxPayloadLength; i++)
59         UART_SendData(TxPayload[i]);
60 }
```

Listing 4: Main.c

Das SSC-Module bietet folgende Methoden an:

- **void SSC\_Init(SSC\_Init\_Parms const \* const Params):**  
Setzt die beiden 8-Bit-Register SSC\_CONH\_P und SSC\_CONL\_P. In diesen Kontroll-Registern werden Parameter für die SPI-Kommunikation festgelegt, die der Mikrocontroller anwendet. Siehe Abbildung 28 im Anschluss an diese Auflistung.
- **ubyte SSC\_GetByte():**  
Ein Byte vom SPI-Slave empfangen.
- **void SSC\_SendByte(ubyte OneByte):**  
Ein Byte an den SPI-Slave senden.
- **uword SSC\_SendData(uword Data):**  
Ein 16-Bit-SPI-Kommando an den angeschlossenen Drucksensor senden und Antwort zurückgeben. Diese Methode wird von der GUI-Anfrage CMD\_RXTX\_HARDWARE\_SSC genutzt. Der 8-Bit-Mikrocontroller kann auf Grund der 8 Bit breiten Register immer nur 8 Bit via SPI senden/empfangen. 16 Bit werden durch zweimaliges Aufrufen von SSC\_SendByte(ubyte OneByte) und SSC\_GetByte() realisiert.
- **void SSC\_SetBaudRate(SSC\_Baud\_Rate BaudRate):**  
Die Baudrate für die SPI-Kommunikation wird über zwei zusätzliche 8-Bit-Register gesteuert.
- **void SSC\_SetSSC\_CONH\_P(ubyte TEN, ubyte REN, ubyte PEN, ubyte BEN, ubyte AREN, ubyte MS):**  
Hilfsmethode für SSC\_Init(...) zum Setzen von SSC\_CONH\_P.
- **void SSC\_SetSSC\_CONL\_P(SSC\_Data\_Width BM, SSC\_Heading HB, SSC\_Clock\_Phase PH, SSC\_Clock\_Polarity P0, ubyte LB):**  
Hilfsmethode für SSC\_Init(...) zum Setzen von SSC\_CONL\_P.

Um die Lesbarkeit des Quelltextes zu erhöhen und die Programmierung durch vordefinierte Wert zu erleichtern — d.h. der Programmierer muss sich keine Gedanken machen, welche Wertebereiche für einen Parameter nun zugelassen sind — verwenden einige Methoden (z.B. `void SSC_SetBaudRate(SSC_Baud_Rate BaudRate)`) eigens definierte Enumerations, anstatt vordefinierter Datentypen für Parameter. Das 2 Byte große Enum `SSC_Baud_Rate` in der Datei `SSC.h` sieht beispielsweise wie folgt aus:

```

1 ...
2 typedef enum
3 {
4     Baud12Mbits      = 0x0000,
5     Baud6Mbits       = 0x0001,
6     Baud1Dot3Mbits   = 0x0008,
7     ...
8 } SSC_Baud_Rate;
9 ...

```

Listing 5: Enumeration `SSC_Baud_Rate` in Datei `SSC.h`

Eine detaillierte Beschreibung für die Funktionen der einzelnen Bitfelder der Register `SSC_CONH_P` und `SSC_CONL_P` liefert [6, S. 318-319]. Eine kurze Übersicht bietet Abbildung 28 und die dazugehörige Tabelle 12.

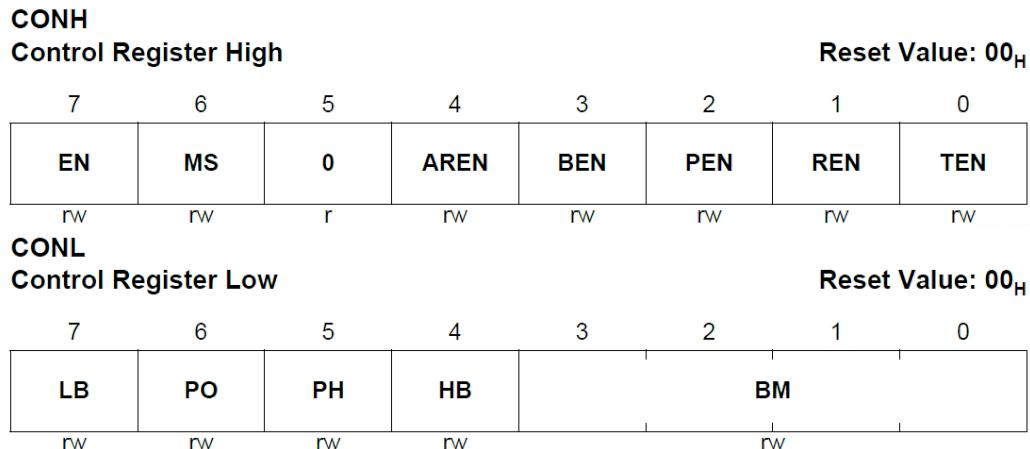


Abbildung 28: XC88xCLM SSC Control Register [6, S. 318-319]

Field	Description
BM	Data Width Selection 0000 Reserved. Do not use this combination. 0001 - 0111 Transfer Data Width is 2...8 bits ( $<BM>+1$ ) Note: BM[3] is fixed to 0.
HB	Heading Control 0 Transmit/Receive LSB First 1 Transmit/Receive MSB First
PH	Clock Phase Control 0 Shift transmit data on the leading clock edge, latch on trailing edge 1 Latch receive data on leading clock edge, shift on trailing edge
PO	Clock Polarity Control 0 Idle clock line is low, leading clock edge is lowto-high transition 1 Idle clock line is high, leading clock edge is highto-low transition
LB	Loop Back Control 0 Normal output 1 Receive input is connected with transmit output (half-duplex mode)

Tabelle 12: Bitfelder im 8-Bit-Register SSC\_CONL\_P [6, S. 318]

Alle zwölf Bitfelder der Register SSC\_CONH\_P und SSC\_CONL\_P sowie die SSC-Baudrate können von der GUI aus gesetzt werden, indem der Anfrage `CMD_CONFIG_HARDWARE_SSC` insgesamt 17 Byte an Payload mitgegeben werden. Erhält die Firmware die 17 Byte von der GUI, können diese einfach in die Struktur `SSC_Init_Params` gecastet werden (Listing 4, Zeile 40) und dann mit der Methode `SSC_Init(...)` in den Registern aus Abbildung 28 platziert werden. Dazu wird auf die Hilfsmethoden `SSC_SetSSC_CONH_P(...)` und `SSC_SetSSC_CONL_P(...)` zurückgegriffen (siehe Listing 6).

```

1 void SSC_SetSSC_CONH_P(ubyte TEN, ubyte REN, ubyte PEN, ubyte
2   BEN, ubyte AREN, ubyte MS)
3 {
4   SSC_CONH_P &= SSC_CLEAR_SSC_CONH_P_MASK;
5
6   SSC_CONH_P |= TEN;
7   SSC_CONH_P |= (REN << 1);
8   SSC_CONH_P |= (PEN << 2);
9   SSC_CONH_P |= (BEN << 3);
10  SSC_CONH_P |= (AREN << 4);
11  SSC_CONH_P |= (MS << 6);
12 }
```

Listing 6: Methode SSC\_SetSSC\_CONH\_P in der Datei SSC.c

Ein weiteres Beispiel für eine Methoden-Implementierung innerhalb des SSC-Moduls liefert Listing 7.

```

1 uword SSC_SendData(uword Data)
2 {
3   // Split parameter "Data" into high and low byte because
4   // microcontroller's SSC only allows eight bit (= one byte
5   // ) per SSC transmission.
6   ubyte HighByte = (ubyte)(Data >> 8);
7   ubyte LowByte = (ubyte)Data;
8
9   uword ReceivedData = 0;
10
11  IO_ResetPin(NCS);           // Select connected device (sensor)
12    and start transmission by bringing "Not Chip Select" (
13    NCS) to "low".
14
15  // Send high byte first (according to device's data sheet).
16  SSC_SendByte(HighByte);
17  // Get high byte.
18  ReceivedData = SSC_GetByte();
19  ReceivedData = ReceivedData << 8;
20
21  // Send low byte.
22  SSC_SendByte(LowByte);
23  // Get low byte.
24  ReceivedData |= SSC_GetByte();
25
26  IO_SetPin(NCS);           // Unselect connected device (
27    sensor) and stop transmission by bringing "NCS" to "high
28    ".
```

```
24     return ReceivedData;
25 }
```

Listing 7: Methode SSC\_SendData in der Datei SSC.c

Die Tests für das SSC-Modul werden direkt nach der Implementierung, händisch unter Zuhilfenahme eines Oszilloskops, durchgeführt. Das Oszilloskop ist notwendig, um das korrekte Zeitverhalten der Spannungen auf den SPI-Leitungen sicherzustellen. Dazu wird in der Main-Methode der Datei *Main.c* der Testcode eingefügt. Dieser besteht aus dem Setzen und Zurücklesen des SSC-Kontroll-Registers und dem wiederholten Senden von SPI-Kommandos. Gleichzeitig werden die Signale der NCS-, CLK-, SDO- und SDI-Leitung mit dem Oszilloskop abgegriffen und verglichen, ob das Zeitverhalten den Einstellungen im SSC-Kontroll-Register entsprechen.

Die Module IO und UART werden ähnlich dem SSC-Module implementiert:

1. Erweitern von *CmdDefines.h* um Befehle, die von der GUI aufgerufen werden dürfen.
2. Hinzufügen von Behandlungsroutine in *Main.c* für neu eingefügte Befehle (*case*-Zweig).
3. Implementieren von Methoden für das jeweilige Modul in den Quelltext-Dateien *Modulename.c/h*.
4. Testen der Implementierung.

Die fertige Firmware wird vor der Auslieferung auf den 8-Bit-Mikrocontroller gespielt und bleibt für den Anwender unveränderbar.

### 3.1.2 Firmware für XC164CS-16F (PGSISI2)

Die Firmware für den 16-Bit-Mikrocontroller der PGSISI2 soll digitale und analoge Drucksensoren unterstützen. Sie basiert auf einem bereits vorhandenen Projekt der PGSISI2, das zur Ansteuerung von Linear-Hall-Sensoren (u.a. für Positionsbestimmung) dient. Es wird daher das bestehende µVision-Projekt verwendet und um die Module erweitert, die notwendig sind um digitale und analoge Drucksensoren anzusprechen und deren EEPROM zu programmieren. Module die ergänzt werden müssen sind:

- SSC: Zur Ansteuerung der digitalen Drucksensoren. Der 16-Bit-Mikrocontroller verfügt über zwei Hardware-SSC-Module, SSC0 und SSC1. Beide funktionieren identisch, verwenden jedoch unabhängige Register. Gemeinsam verwendete Strukturen können in eine Header-Datei `SSC.h` ausgelagert werden und dann von beiden Modulen gleichermaßen verwendet werden. So z.B. das Enum `tSSC_Baud_Rate` für alle unterstützten Baudraten.
- SoftwareSpi: Dabei handelt es sich um kein Hardware-Modul. Dieses wird verwendet, um die Funktionsweise eines Hardware-SSC-Moduls zu emulieren und das EEPROM von analogen Sensoren zu programmieren. Dazu werden drei allgemeine Mikrocontroller-Pins als CLK-, MOSI- und MISO-Leitung verwendet und mit den Sensor-Pins CLOCK/V-PROG, DATA IN und DATA OUT (Abbildung 15, Seite 30) verbunden. Über diese Leitungen werden die Daten an das EEPROM gesendet bzw. empfangen. Die digitalen Sensoren verwenden dazu die normale SPI-Kommunikation, indem sie ein spezielles Write-Kommando an den Sensor schicken.

Die folgende Abbildung 29 zeigt die vollständige Projektstruktur für den 16-Bit-Mikrocontroller:

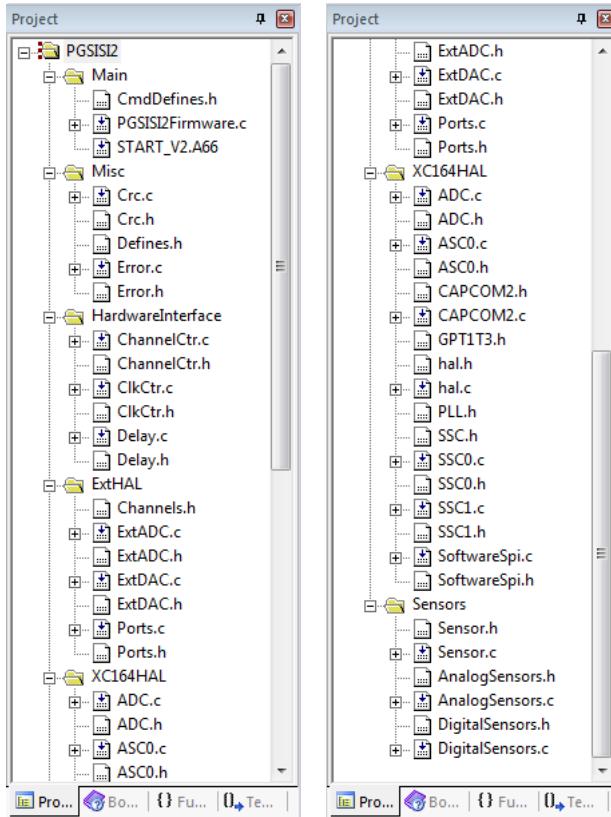


Abbildung 29: Projektstruktur für 16-Bit-Mikrocontroller

- **PGSISI2**

- **Main**

- \* *CmdDefines.h*: Enthält genau die gleichen Befehle wie beim 8-Bit-Mikrocontroller und noch weitere. Dadurch wird erreicht, dass sich PGSISI2 und UWLink vollkommen gleich ansprechen lassen. Darüber hinaus muss auch der Payload (gemäß Protokoll aus Kapitel 3.2) für die einzelnen Befehle gleich sein. Dies ist aber auch sichergestellt. So bekommt z.B. der Befehl **CMD\_CONFIG\_HARDWARE\_SSC** immer 17 Byte an Payload mit, egal ob er an die PGSISI2 oder den UWLink gesendet wird. Dieses Verhalten verringert die Komplexität der GUI erheblich.
    - \* *PGSISI2firmware.c*: Entspricht der *Main.c* für den 8-Bit-Mikrocontroller und folgt auch der Struktur nach Listing 2.

- \* *START\_V2.A66*: Von µVision bereitgestellte Start-Konfiguration für den 16-Bit-Mikrocontroller.
- ...
- **XC164HAL**
  - \* *ADC.c/h*: Makros und Methoden für Verwendung des ADCs.
  - \* *ASC0.c/h*: Das UART-Modul wird im Handbuch des 16-Bit-Mikrocontrollers als Asynchronous Serial Channel (ASC) bezeichnet.
  - \* ...
- **Sensors**
  - \* *Sensors.h/c*: Methoden die von analogen und digitalen Sensoren verwendet werden können. Darunter Methoden um den Sensor mit Spannung zu versorgen und die notwendige Programmierspannung für die EEPROM-Programmierung anzulegen. Da digitale und analoge Sensoren den gleichen Sensor-Pin (Nummer fünf) für die Spannungsversorgung nutzen muss keine Unterscheidung an dieser Stelle gemacht werden. Der VPROG-Pin unterscheidet sich jedoch bei beiden: Pin Nummer sechs bei digitalen, Pin Nummer zwei bei analogen Sensoren. D.h. die Methode für die Programmierspannung benötigt einen Parameter, mit dem der Sensor-Pin ausgewählt werden kann. Bzw. der Parameter bestimmt eigentlich, welcher Mikrocontroller-Pin verwendet wird. Der Mikrocontroller-Pin ist mit dem D-Sub-Stecker der PGSISI2 verbunden und über ein Kabel wird der Kontakt zum Sensor-Pin hergestellt.
  - \* *AnalogSensors.h/c*: Enthält Methoden um den Sensor in den Testmodus zu schicken. Dieser muss aktiviert sein, damit das EEPROM programmiert werden kann. Der Testmodus wird durch ein spezielles Kommando aktiviert, das mit dem Software-SPI gesendet wird. Dieses Kommando wird vom Sensor nur akzeptiert, wenn es das erste Kommando ist, nachdem der Sensor mit Spannung versorgt wird. Zudem muss es innerhalb eines bestimmten Zeitrahmen gesendet werden. Das Aktivieren des Testmodus' erfordert, dass der Sensor beliebig mit Spannung versorgt werden kann. Dies kann z.B. mit dem 8-Bit-Mikrocontroller des UWLinks nicht bewerkstelligt werden, da dort die Spannungsversorgung über den USB-Port des

Host-PCs erfolgt. D.h. sobald der UWLink am USB-Port angeschlossen ist, wird auch der Sensor mit Strom versorgt. Der Testmodus wird verlassen, wenn der Sensor aus- und wieder eingeschaltet wird (Spannung weg und an).

- \* *DigitalSensors.h/c)*: Ähnlich wie *AnalogSensors.h/c*.

Die Architektur der Firmware ist so generisch gestaltet, dass für ein konkretes Projekt nun nur die Ordner **Main** und **Sensor** ausgetauscht bzw. angepasst werden müssen. Dies geschieht in Abhängigkeit der zu verwendenden Sensoren.

Die Implementierung aller notwendigen Module erfolgt auf ähnliche Weise wie beim 8-Bit-Mikrocontroller (beschrieben in Kapitel 3.1.1). Im Folgenden noch eine Erläuterung zur EEPROM-Programmierung der Drucksensoren.

**EEPROM-Programmierung der Drucksensoren** Alle Details zur EEPROM-Programmierung der digitalen und analogen Drucksensoren wird in vertraulichen Dokumenten beschrieben, welche nur für den internen Gebrauch bei der Infineon Technologies AG gedacht sind. Eine genaue Erläuterung aller Vorgänge kann daher nicht gemacht werden, aber ein kurzer Abriss ist möglich.

Das EEPROM der Drucksensoren ist allgemein aus  $i$  Zeilen und  $j$  Spalten aufgebaut. Jedes Element ( $i, j$ ) entspricht einem Bit. Das EEPROM enthält die Koeffizienten der Sensor-Kalibrierung. Diese Koeffizienten werden für die genaue Umwandlung des physikalischen Drucks in ein elektrisches Signal verwendet. Die Zeilen des EEPROMs sind in einzelne, unterschiedlich lange Bitfelder unterteilt. Jedes dieser Bitfelder enthält einen Koeffizienten. Darüber hinaus gibt es in jeder Zeile noch ein 1 Bit breites Feld, welches als Paritäts-Bit für diese Zeile gilt. Eine zusätzliche Zeile gibt es, welche nur aus 1-Bit-Feldern besteht. Sie stellen die Spaltenparität dar. Abbildung 30 verdeutlicht den Aufbau des EEPROMs der Drucksensoren.

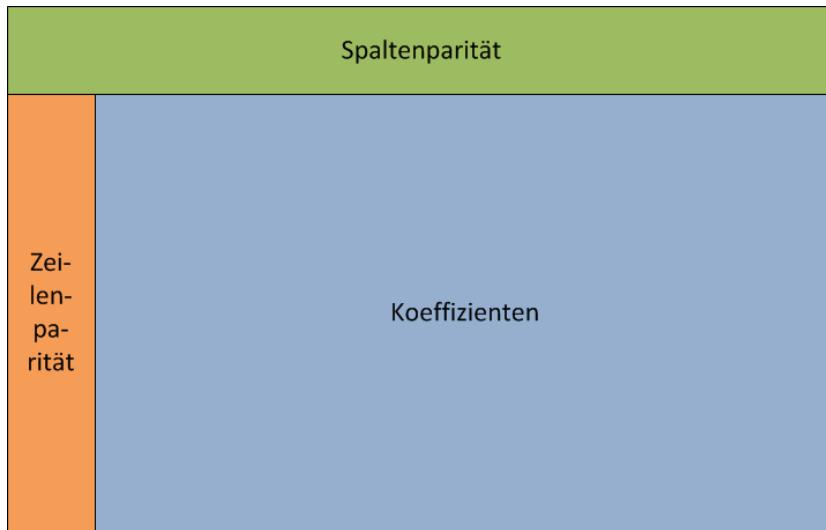


Abbildung 30: EEPROM-Aufbau der Drucksensoren

Das EEPROM ist ein nichtflüchtiger Speicher, sodass dessen Inhalt auch erhalten bleibt, wenn es nicht mit Spannung versorgt wird. Es kann zeilenweise angesprochen werden — lesend oder schreibend. Dabei hat man nie- mals direkten Zugriff auf die EEPROM-Zellen. Der Inhalt wird beim Starten des Sensors in sogenannte Schattenregister kopiert. Für jede EEPROM-Zeile steht ein eigenes Register zur Verfügung. Dieses kann ausgelesen oder beschrieben werden. Wird am VPROG-Pin des Sensors die Programmier- spannung angelegt, so wird der aktuelle Inhalt dieser Schattenregister in die EEPROM-Zellen transferiert.

Das Schreiben des EEPROMs erfolgt in zwei Durchläufen. Im Ersten werden die Bit-Änderungen „0“ zu „1“ programmiert (Modus Write). Im zweiten Durchlauf die Bit-Änderungen „1“ zu „0“ (Modus Erase). Der entsprechende Modus wird durch setzen eines Bitfeldes innerhalb eines Schattenregisters gewählt. Beim Modus Write wird den Zellen Spannung zugeführt, bei Erase wird den Zellen Spannung entzogen. Im jeweiligen Durchlauf steht in den Schattenregistern nicht der neue EEPROM-Inhalt, sondern ein Write- bzw. Erase-Pattern.

Im Modus Write bedeutet eine 1 Spannung zuzuführen, eine 0 hat keine Auswirkung. Hierbei ist es entscheidend, dass wirklich nur den Zellen Spannung zugeführt ist, die von 0 auf 1 programmiert werden. Wird einer 1 nochmals Spannung zugeführt, kommt es zu Überspannungsschäden und das EEPROM

ist defekt. Im Modus Erase bedeutet eine 1 Spannung zu erhalten, bei einer 0 wird der Zelle Spannung entzogen. Abbildung 31 zeigt das Write- und Erase-Pattern, um aus einem alten EEPROM-Inhalt einen neuen Inhalt zu erzeugen.

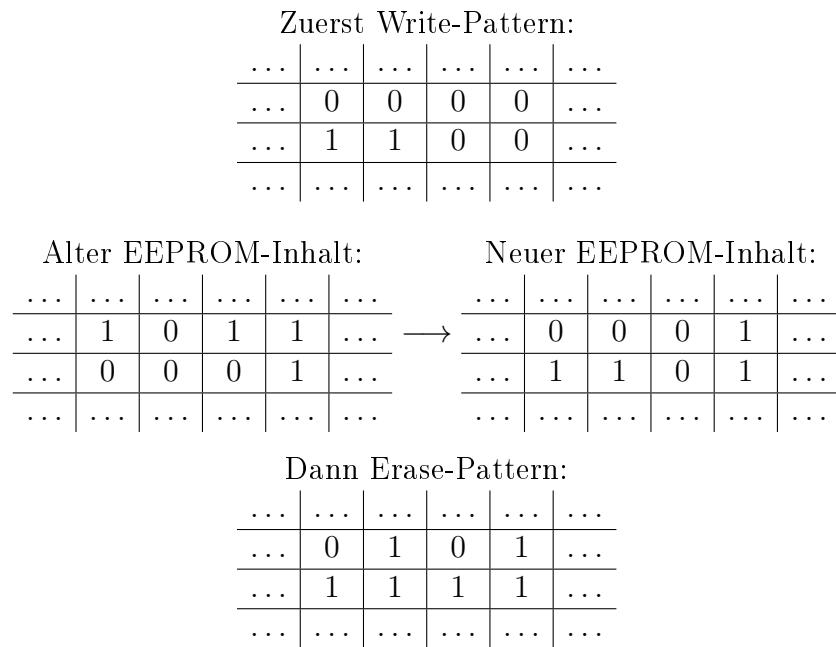


Abbildung 31: Write- und Erase-Pattern für EEPROM-Programmierung

Für die Programmierung des EEPROMs sind folgende Schritte einzuhalten:

1. Den Sensor in den Testmodus versetzen.
2. Im Modus Write „0“ zu „1“ setzen.
3. Programmierspannung anlegen.
4. Im Modus Erase „1“ zu „0“ setzen.
5. Programmierspannung anlegen.
6. Testmodus verlassen.

Hinweise:

- Im Evaluierungswerkzeug kümmert sich die GUI um die Berechnung der Write- und Erase-Patterns.
- Die Write- und Erase-Patterns werden zeilenweise über normale SPI-Kommandos in die Schattenregister geschrieben.
- Die Paritäts-Bits für Zeilen und Spalten müssen selbst ermittelt werden.

Mit der EEPROM-Programmierung schließt auch der hardwarenahe Teil der Implementierung. Wie beim 8-Bit-Mikrocontroller wird die fertige Firmware werkseitig auf den 16-Bit-Mikrocontroller gespielt. Im Gegensatz zum 8-Bit-Mikrocontroller kann die GUI aber auch im Nachhinein eine neue Firmware aufspielen. Dazu überträgt die GUI die kompilierte Firmware an das Programmiergerät, wenn es sich mit diesem verbindet.

Die Firmwares für 8- und 16-Bit-Mikrocontroller sind nun abgeschlossen und können mit Hilfe eines einfachen Protokolls von der GUI aus angesprochen werden.

### 3.2 Die Schnittstelle zwischen Firmware und GUI: Das Protokoll

Die GUI soll mit dem Sensor kommunizieren. Da dies nicht auf direktem Wege möglich ist, wird als Brücke zwischen beiden das Programmiergerät mit Mikrocontroller verwendet. Die Firmware des Mikrocontrollers empfängt die Daten vom PC über sein UART-Modul (der IC von FTDI nimmt zuvor eine transparente USB-RS232-Übersetzung vor), verarbeitet diese und schickt eine Antwort zurück an die GUI. Bei der Verarbeitung der Daten kann der Mikrocontroller mit dem Sensor kommunizieren oder nicht. Empfängt die Firmware z.B. den Befehl `CMD_CONFIG_HARDWARE_SSC` von der GUI betrifft das nur die SSC-Konfiguration auf dem Mikrocontroller und damit nicht den Sensor. Bei `CMD_RXTX_HARDWARE_SSC` dagegen wird die Firmware angewiesen die übergebenen Daten weiter an den Sensor zu schicken und die Antwort zurückzuliefern. GUI und Programmiergerät arbeiten dabei nach dem Master-Slave-Prinzip. D.h. die GUI ist der Master und initiiert die Kommunikation, das Programmiergerät reagiert nur und wird niemals selbstständig tätig.

Damit sich beide Kommunikationspartner verstehen und Nachrichten austauschen können, ist ein Protokoll — also Regeln und Formate für Nachrichten — notwendig.

Die Regeln für die Kommunikation ergeben sich aus dem Master-Slave-Prinzip und bestimmen den Aufbau der Mikrocontroller-Firmware (siehe Listing 2 auf Seite 50). Diese wartet auf Daten von der GUI, verarbeitet diese und schickt wieder Daten zurück. Dies erfolgt streng sequentiell. Erst nachdem eine Anfrage der GUI vollständig abgearbeitet ist, kann eine neue entgegengenommen werden. Das bedeutet für die GUI, dass sie erst eine erneute Anfrage senden darf, nachdem die aktuelle durch die Firmware beantwortet wurde. Dies verdeutlicht nochmal Abbildung 32.

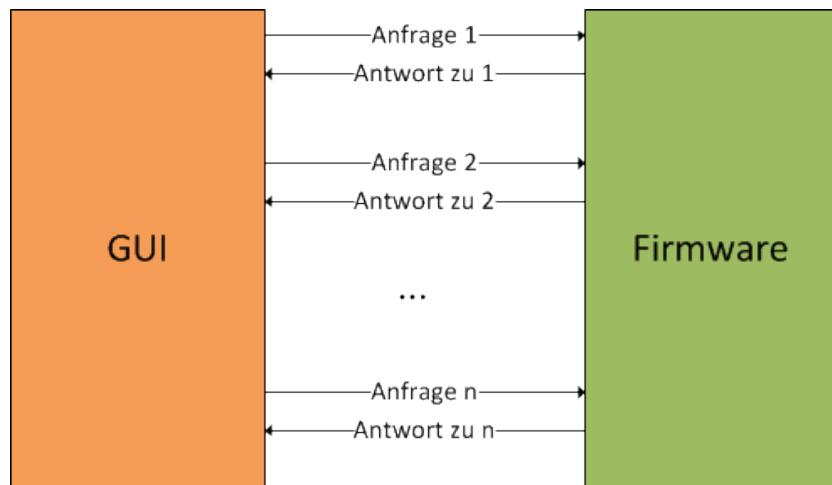


Abbildung 32: Master-Slave-Prinzip zwischen GUI und Firmware

Die GUI soll einzelne Funktionalitäten des Mikrocontrollers - bzw. der Firmware, die diesen steuert - ansprechen. Einige Funktionalitäten benötigen Parameter andere nicht. Daraus ergibt sich das Format der Nachrichten von GUI zu Firmware:

<code>&lt;Command&gt;</code>	<code>&lt;Payload Length&gt;</code>	<code>&lt;Payload&gt;</code>
1 Byte	1 Byte	„Payload Length“ Byte

Tabelle 13: Nachrichtenformat GUI zu Firmware

- Das erste Byte (`<Command>`) bestimmt die Funktionalität, die am

Mikrocontroller verwendet werden soll. Ein Byte ergibt 256 ( $2^8$ ) verschiedene Funktionalitäten die ausgewählt werden können. Das ist für dieses Projekt und auch zukünftige ausreichend. Die PGSI2 stellt im Moment 32 Commands zur Verfügung, der UWLink fünf Commands. Die Commands/Funktionalitäten welche die Firmware anbieten möchte, werden in der Datei *CmdDefines.h* veröffentlicht (Listing 3, Seite 53).

- Das zweite Byte (*<Payload Length>*) bestimmt, ob der Command noch weitere Parameter benötigt. Die Firmware gibt vor, ob ein Command Parameter benötigt oder nicht. Wird einem Command eine ungültige Anzahl an Parametern mitgegeben verarbeitet die Firmware die Anfrage nicht, sondern schickt sie zurück an die GUI gemäß dem Protokoll aus Tabelle 14. Es können maximal 256 Byte an Payload einem Command mitgegeben werden. Im Moment werden maximal 17 Byte benötigt (Konfiguration des SCC mittels `CMD_CONFIG_HARDWARE_SSC`).
- Die nachfolgenden Bytes *<Payload>* stellen die Parameter für den angeforderten Command dar.

Nachdem die Mikrocontroller-Firmware die GUI-Anfrage bearbeitet hat, wird die Antwort an die GUI geschickt. Dafür wird folgendes Nachrichtenformat verwendet:

<i>&lt;Command&gt;</i>	<i>&lt;OK&gt;</i>	<i>&lt;Payload Length&gt;</i>	<i>&lt;Payload&gt;</i>
1 Byte	1 Byte	1 Byte	„Payload Length“ Byte

Tabelle 14: Nachrichtenformat Firmware zu GUI

- Das erste Byte (*<Command>*) entspricht dem empfangenen Command-Byte und wird unverändert wieder zurückgeschickt.
- Das zweite Byte (*<OK>*) zeigt, ob die Anfrage durch die Firmware korrekt verarbeitet wurde. Ist es „0“ konnte die Anfrage nicht verarbeitet werden. D.h. das z.B. ein ungültiges Kommando geschickt wurde oder eine ungültige Anzahl an Parametern für ein Kommando. Ist das OK-Byte „0“ wird auch keine Payload von der Firmware gesendet.
- Das dritte Byte (*<Payload Length>*) zeigt der GUI an wie viel Nutzdaten von der Firmware zurückgesendet werden.

- Die folgenden Bytes (<Payload>) stellen die angefragten Nutzdaten dar.

Die Firmware ist so robust gegenüber fehlerhaften Anfragen (ungültiges Kommando bzw. falsche Anzahl an Parametern).

Im Folgenden nun ein Beispiel für die Kommunikation zwischen GUI und Firmware — sowie Sensor. Das SSC des Mikrocontrollers wurde zuvor mittels `CMD_CONFIG_HARDWARE_SSC` konfiguriert und ist nun in der Lage mit einem digitalen Drucksensor zu kommunizieren. Dazu wird der Befehl `CMD_RXTX_HARDWARE_SSC` ( $1A_{16}$ ) und 2 Byte an Payload ( $20\ 00_{16}$ ) gesendet. Die 2 Byte entsprechen dem SPI-Kommando der digitalen Drucksensoren, um den Druckwert auszulesen (siehe Tabelle 5 auf Seite 23). Die Zahlendarstellung ist hexadezimal.

Anfrage (GUI): **1A 02 20 00**

Antwort (Firmware): **1A 01 02 51 6F**

Tabelle 15: Beispiel für GUI-Firmware-Kommunikation mit vereinbartem Protokoll

Dabei braucht die GUI nicht darauf zu achten, mit welchem Programmiergerät es kommuniziert, da beide gleich angesprochen werden können. Dies ist möglich da die Hardware-Module, die 16- und 8-Bit-Mikrocontroller gleichermaßen verwenden — darunter z.B. SSC — ein identisches Verhalten aufweisen. So verwendet z.B. der 16-Bit-Mikrocontroller der PGSISI2 für die Konfiguration des SSCs ein 16 Bit breites Register. Der 8-Bit-Mikrocontroller des UWLinks dagegen zwei jeweils 8 Bit breite Register. Das Kommando zum Konfigurieren des SSC-Moduls bekommt dann immer 17 Byte an Payload mit, egal ob es sich um die PGSISI2 oder den UWLink handelt.

### 3.3 Die grafische Benutzeroberfläche

Die GUI des Evaluierungswerkzeuges visualisiert die Funktionalitäten des Drucksensors. So sollen die letzten Druckwerte (bei den digitalen Drucksensoren auch die Temperaturwerte) in einem 2-dimensionalem Graphen angezeigt und zusätzlich in einer Liste protokolliert werden. Es soll auch die

Möglichkeit bestehen alle Druckwerte einer Messung als Textdatei zu speichern. Zudem benötigt die interne Version der GUI ein Fenster in dem die Programmierung des EEPROMs möglich ist. Dieses darf aber nur in der intern verwendeten Version angezeigt werden und auch nur in Verbindung mit der PGSISI2, da nur dieses Programmiergerät zur EEPROM-Programmierung der Sensoren genutzt werden kann. Desweiteren benötigt die GUI ein Installationsprogramm, damit die fertige Applikation für die Benutzer einfach durch einen Doppelklick zu installieren ist. Ein Benutzerhandbuch in englischer Sprache rundet die GUI-Entwicklung ab.

### 3.3.1 GUI

Die Entwicklung der GUI erfolgt in C# und mittels Microsofts .NET- "Framework 2.0. Unterstützung liefert die IDE Microsoft Visual Studio 2008. Neben den IDE-typischen Merkmalen wie integrierte Projektstrukturierung, Editor, Compiler und Debugger bietet Visual Studio 2008 den Windows Forms-Designer. Dieser WYSIWYG-GUI-Builders<sup>5</sup> erleichtert und beschleunigt die Erzeugung von grafischen Oberflächen enorm.

Der GUI-Code basiert auf einem Template, dass hausintern für Sensorprojekte, ähnlich wie diesem, entwickelt wurde. Das Template nutzt für die grafischen Elemente die Windows-Forms-API des .NET-Frameworks. Durch seine Struktur sorgt das Template für eine Trennung von grafischer Oberfläche und Geschäftslogik (siehe Abbildung 33).

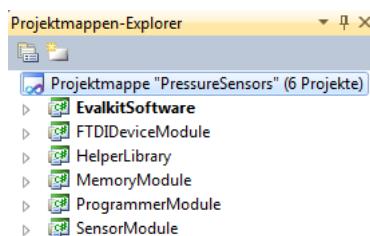


Abbildung 33: Die Projektstruktur der GUI

---

<sup>5</sup>Das Akronym WYSIWYG steht für „What You See Is What You Get“.

- **EvalkitSoftware:** Enthält die eigentliche GUI. Zusätzlich stellt das Template bereits einen „ApplicationManager“ bereit. Dieser sorgt dafür, dass angeschlossene Programmiergeräte — bzw. eigentlich deren USB-RS232-Übersetzer von FTDI — automatisch erkannt und in einer Liste angezeigt werden, von wo sie aus vom Benutzer ausgewählt werden können.
- **FTDIDeviceModule:** FTDI stellt für ihren USB-RS232-Übersetzer eine Dynamic Link Library (DLL) in C++ zur Verfügung. Sie bietet unter anderem Methoden zum Senden von Daten an den FTDI-Chip bereit und auch zum Empfangen. Die GUI kommuniziert praktisch nur direkt mit dem FTDI-Chip. Der Chip gibt das Signal nach der Protokollübersetzung an den Mikrocontroller weiter. Alle Komponenten hinter dem FTDI-Chip — d.h. Mikrocontroller und Sensor — sind der GUI für die Kommunikation egal. Um die C++-DLL in diesem C#-Projekt verwenden zu können, wird sie in diesem Software-Modul in C#-Code gewrapped.
- **HelperLibrary:** Stellt zahlreiche Hilfsmethoden in verschiedenen Klassen bereit. Darunter einen Parser der die Datei *CmdDefines.h* der Mikrocontroller einliest und die Byte-Sequenzen der verfügbaren Befehle extrahiert. Des Weiteren Methoden zur Konvertierung von ganzzahligen Datentypen (Int32 etc.) in Byte-Arrays und umgekehrt. Diese sind hilfreich, da vom FTDI-Chip rohe Byte-Sequenzen empfangen werden. Außerdem bietet die HelperLibrary ein Singleton-Logging-Objekt, das die Logging-Ausgaben in ein eigenes Fenster schreibt.
- **MemoryModule:** Stellt ein EEPROM-Objekt aus vielen Zeilen dar. Die Zeilen sind nochmals in unterschiedlich lange Bitfelder unterteilt. Das EEPROM-Objekt enthält dann die Koeffizienten der Sensor-Kalibrierung.
- **ProgrammerModule:** Eine abstrakte Repräsentation des Programmiergerätes. Enthält unter anderem eine Klasse „PGSISI2“ und eine Klasse „UWLink“.
- **SensorModule:** Enthält die unterstützten Sensoren als einfach zu handhabende Objekte. So kann beispielsweise der aktuelle Druckwert über eine Methode „GetPressure()“ eingelesen werden.

Die folgende Tabelle 16 beschreibt die Abhängigkeiten der Software-Module untereinander. Hieraus wird auch ersichtlich, dass der GUI-Code im Modul

EvalkitSoftware von allen anderen Modulen abhängig ist, die anderen Module aber keine Abhängigkeiten zum GUI-Code besitzen. D.h. die grafische Oberfläche kann leicht ausgetauscht werden.

Modul	Hängt ab von...
EvalkitSoftware	FTDIDeviceModule, HelperLibrary, MemoryModule, ProgrammerModule, SensorModule
FTDIDeviceModule	HelperLibrary
HelperLibrary	-
MemoryModule	HelperLibrary
ProgrammerModule	FTDIDeviceModule, HelperLibrary
SensorModule	HelperLibrary, MemoryModule, ProgrammerModule

Tabelle 16: Abhängigkeiten der Software-Module der GUI

Das Modul **EvalkitSoftware** enthält den GUI-Code, **SensorModule** die sensorspezifischen Implementierungen. Beide müssen im Entwicklungsprozess mit Hilfe von Klassen und weiteren Typen (Enums etc.) implementiert werden. Die anderen, bereits implementierten Module unterstützen die Entwicklung. Sie werden nur bei Bedarf um weitere nützliche Methoden ergänzt.

**Die GUI** Die Entwicklung der grafischen Oberfläche beginnt mit einem Prototyp auf Papier, erst dann wird die Architektur für den „dahinterliegenden“ Nicht-GUI-Code entworfen. Durch diese Reihenfolge werden bereits beim Entwurf des GUI-Prototyps einige Anforderungen an den Aufbau des Nicht-GUI-Codes deutlich: Welche Objekte muss es geben? Welche Methoden müssen diese anbieten? Dieses Erkenntnisfließen dann im Anschluss in den Entwurf der gesamten Softwarearchitektur ein. Der GUI-Prototyp wird dann im Windows Forms-Designer der Visual-Studio-IDE nachgebaut (siehe Abbildung 34).

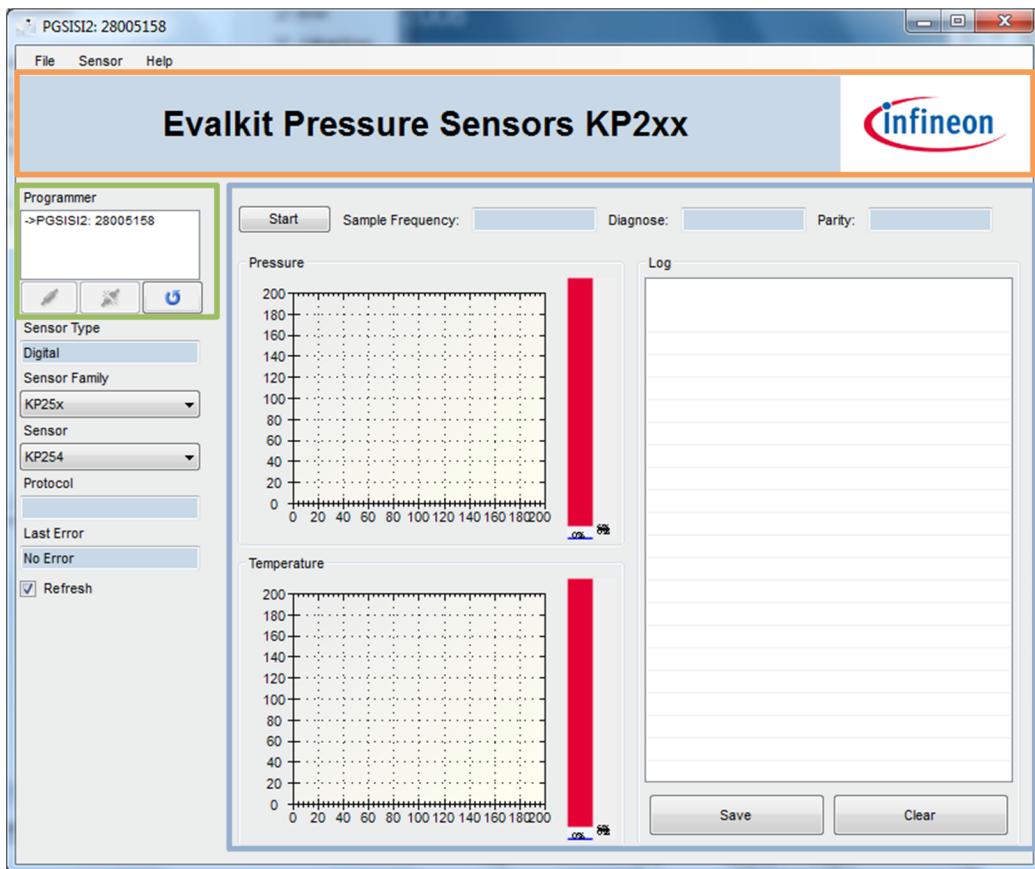


Abbildung 34: GUI-Prototyp mit Windows Forms-Designer

Der Prototyp berücksichtigt das bereits existierende Layout des Templates das in ein Fenster eingebettet ist. Das Layout ist horizontal (orange Box) und vertikal (grüne Box) zweigeteilt. Die orange und grün markierten Bereiche sind durch das Template vorgegeben. Die restlichen Bereiche wurden erweitert. Der blau markierte Bereich ist durch ein C#-Interface vorgegeben (**IMainPanel**). So gibt das Interface vor, dass dieser Bereich (vom C#-Typ **UserControl**) eine Methode anbieten muss, die in einem eigenen Thread Sensormessdaten einliest (beim Drücken des Start-Buttons). Dieser Bereich benötigt dann letztendlich ein konkretes Sensorobjekt aus dem **SensorModule**. Ein eigener Thread ist notwendig, damit während des Einlesens die restliche GUI unverzögert auf Eingaben durch den Benutzer reagieren kann. Die Konstruktion mit dem Interface **IMainPanel** erlaubt eine Wiederverwendbarkeit des Template-Codes. Das umliegende Fenster enthält eine Referenz auf dieses Interface und stellt damit sicher, dass es einen Bereich innerhalb

des Fensters gibt, indem Sensordaten in einem eigenen Thread eingelesen werden können. Dieses Konstrukt kann damit auch in zukünftigen Projekten mit Sensoren eingesetzt werden, wobei jeder Messbereich individuell gestaltet werden kann.

Zu den einzelnen GUI-Elementen:

- Im linken Bereich wird ein Programmiergerät, welches am PC angeschlossen ist, ausgewählt und mit der GUI „verbunden“. Darunter wird mit Hilfe eines Drop-Down-Menüs der zu verwendende Sensor ausgewählt.
- Der rechte Bereich ist den Sensordaten vorbehalten. Über den Button „Start“ lassen sich Messungen starten und stoppen. Es gibt einen 2-dimensionalen Graphen für die letzten 200 Druckwerte. Daneben gibt es einen Ausgabebalken. Dieser ist der rohe Druckwert — bei den digitalen Drucksensoren sind das die 10/13 Daten-Bits einer SPI-Antwort, bei analogen Sensoren die Spannung in Volt am VOUT-Pin — vor Umrechnung in kPa mit Hilfe der Transfer-Funktion. Im Log-Bereich werden die letzten 10.000 Werte protokolliert.  
Die Werte 200 im Graphen und 10.000 im Log-Bereich haben sich im Laufe der Entwicklung ergeben. Sie sind ein Kompromiss aus Nutzeranforderungen und Ressourcenverbrauch. Mit diesen Werten verbraucht die fertige Applikation konstant 30 Megabyte an RAM. Alle Daten einer Messungen werden gleichzeitig gepuffert auf die Festplatte geschrieben. Durch den Button „Save“ können diese an eine beliebige Stelle kopiert werden. Nicht kopierte Messreihen werden wieder von der Festplatte gelöscht.
- Über die Menüleiste am oberen Rand des Fensters lässt sich die Anwendung beenden („File“) und erhält Zugriff auf das EEPROM des Sensors („Sensor“).

Die GUI stellt nur die Funktionen dar, die vom Programmiergerät unterstützt werden. In Verbindung mit dem UWLink wird beispielsweise der Punkt „Sensor“ deaktiviert. Der GUI-Prototyp ist auf digitale Sensoren zugeschnitten. Es wird auch noch ein Prototyp für die analogen Sensoren entworfen. Hierbei muss nur der „blaue“ Bereich (Abbildung 34) angepasst werden. Dieser Bereich implementiert das Interface `IMainPanel`. Der Messbereich für die analogen Sensoren ist also einfach eine neue Klasse. Bei ihr fällt der 2-dimensionale

Graph zur Anzeige der Temperatur weg. Ansonsten ist sie identisch zum Prototyp der digitalen Sensoren.

Nach der Fertigstellung der GUI-Prototypen wird die Architektur hinter der GUI entworfen.

**Der Nicht-GUI-Code — Die Geschäftslogik** Die Geschäftslogik verteilt sich auf drei logischen Ebenen:

- **Die unterste Ebene** bildet die Anbindung an die Mikrocontroller-Firmware. Mit ihr ist es möglich die angebotenen Funktionalitäten der Firmware aufzurufen.
- **Die mittlere Ebene** bietet Sende- und Empfangsrahmen zur Kommunikation mittels SPI. Die Kommunikation mit digitalen Sensoren vertraut vollständig auf SPI, analoge Sensoren dagegen benötigen nur ein emuliertes SPI zur EEPROM-Programmierung.
- **Die oberste Ebene** bilden die unterstützten Sensoren. Dort ist dann über einfache Methoden wie z.B. `ActivateTestMode` und `GetPressure` eine Interaktion mit dem Sensor möglich, ohne sich über die Kommunikationsdetails der darunterliegenden Schichten Gedanken machen zu müssen.

Von unten nach oben hin werden immer mehr die konkreten Hardware-Details verborgen. Abbildung 35 bietet eine Übersicht über die Verteilung der Klassen und Enums auf die einzelnen Ebenen.

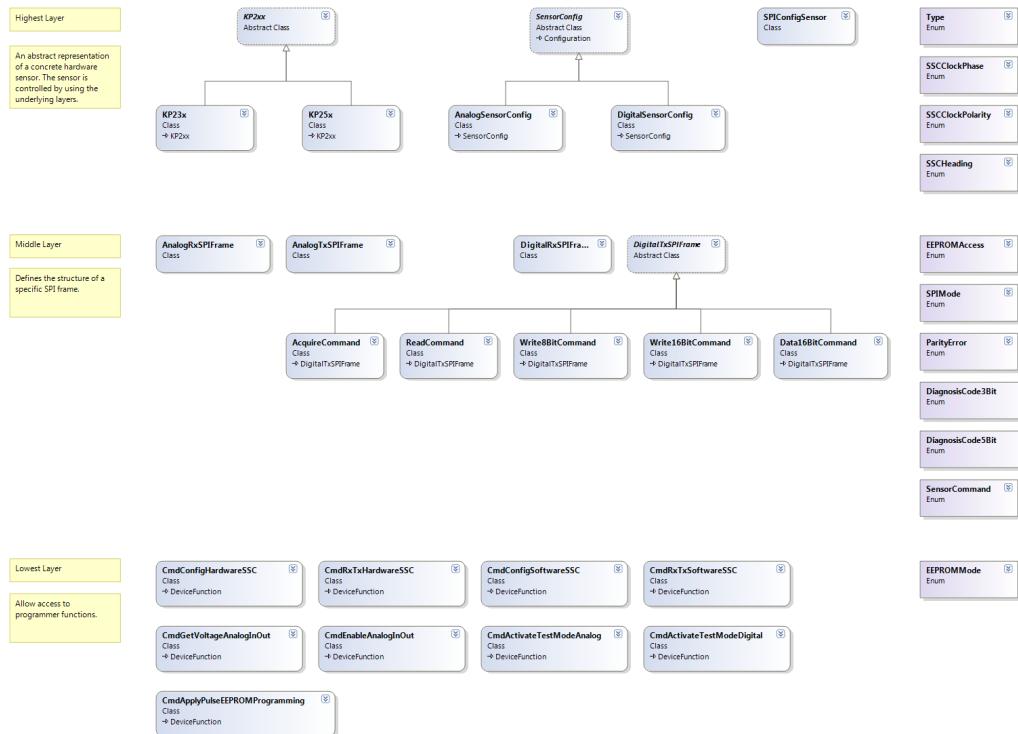


Abbildung 35: Vollständiges Klassendiagramm der Geschäftslogik

Im Folgenden nun die Ebenen genauer erklärt.

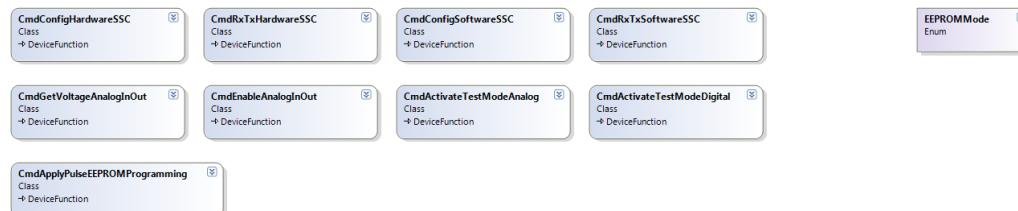


Abbildung 36: Klassendiagramm der untersten Ebene

**Die unterste Ebene** Mit Hilfe der untersten Ebene gelingt der Zugriff auf die Mikrocontroller-Funktionen, die in der Datei *CmdDefines.h* definiert sind.

Die eigentliche Schnittstelle zur Firmware bietet das Programmer-Objekt aus dem Modul **ProgrammerModule**. Es enthält ein FTDI-Objekt aus dem

Modul **FTDIDeviceModule** mit welchem Daten zum FTDI-Chip gesendet und Daten empfangen werden können. Das Programmer-Objekt bietet eine Methode, an welche Parameter vom Typ **DeviceFunction** übergeben werden.

Diese DeviceFunctions repräsentieren die von der Firmware angebotenen Befehle in der Datei *CmdDefines.h*. Jede konkrete **DeviceFunction** erbt von der gemeinsamen Basisklasse ein Byte-Array für zu sendende Daten (Output) und ein Byte-Array für zu empfangende Daten (Input). Jede einzelne **DeviceFunction** platziert nun im Output-Array die Anfrage an die Firmware gemäß des vereinbarten Protokolls (Kapitel 3.2). Im Input-Array wird dann die Antwort der Firmware abgelegt.

Die Byte-Sequenz für ein Firmware-Kommando wird nicht hart in den Quelltext der **DeviceFunction** codiert. In der **DeviceFunction** wird nur der Name des Kommandos angegeben (Listing 8, Zeile 24). Ein Parser liest beim Start der fertigen Anwendung die entsprechende *CmdDefines.h* ein und erzeugt daraus eine Liste aus **StringValuePairs**. Beim Senden wird der Name (String) des Kommandos durch seine Byte-Sequenz (Value) ersetzt. Dies hat zum Einen den Vorteil das der C#-Quelltext durch den Namen des Kommandos lesbar bleibt. Zum Anderen kann auch im Nachhinein die Mikrocontroller-Firmware die Byte-Sequenz für ein Kommando ändern ohne das am C#-Code etwas geändert werden muss.

Die unterste Ebene umfasst alle Kommandos der Firmware, die später von der obersten Schicht gebraucht werden. Listing 8 zeigt exemplarisch die **DeviceFunction CmdRxTxHardwareSSC**, die genutzt wird, um ein 16-Bit-SPI-Kommando an einen digitalen Drucksensor zu senden. Der Senderrahmen **digitalTxSPIFrame** enthält das eigentliche SPI-Kommando des Sensors (Tabelle 5). Das verwendete Programmer-Objekt (**PGSISI2** oder **UWLink**) übergibt der **DeviceFunction** auch die Byte-Reihenfolge die benutzt wird. Der UWLink nutzt die Big-Endian-Reihenfolge, die PGSISI2 dagegen Little-Endian. Diese Parametrisierung stellt sicher, dass das 16-Bit-SPI-Kommando von der Firmware richtig verarbeitet wird.

```
1 using HelperLibrary;
2 using ProgrammerModule;
3 using SensorModule.SPI;
4
5 namespace SensorModule.ProgrammerFunctions
6 {
7     /// <summary>
8     /// Programmer function to send a sensor command to the
9     /// sensor using
10    /// programmer's microcontroller SSC module and get
11    /// sensor's answer as response.
12    /// </summary>
13    public class CmdRxTxHardwareSSC : DeviceFunction
14    {
15        ...
16
17        #region Constructors
18
19        /// <summary>
20        /// Constructor, which builds the byte sequence, that
21        /// will be sent to the microcontroller.
22        /// </summary>
23        /// <param name="digitalTxSPIFrame">The SPI frame,
24        /// which should be sent to the used sensor.</param>
25        /// <param name="byteOrderingProgrammer">The byte
26        /// ordering of the used programmer/microcontroller.</
27        /// param>
28        public CmdRxTxHardwareSSC(DigitalTxSPIFrame
29            digitalTxSPIFrame, ByteConversions.eByteOrdering
30            byteOrderingProgrammer)
31        {
32            mCommandName = "CMD_RXTX_HARDWARE_SSC";
33
34            // Build array of bytes, which will be sent to
35            // the microcontroller.
36            DataOutAdd(ByteConversions.ToByteArray(
37                digitalTxSPIFrame.SensorCommand,
38                byteOrderingProgrammer));
39        }
40
41        #endregion
42    }
43}
```

Listing 8: Beispiel für DeviceFunction

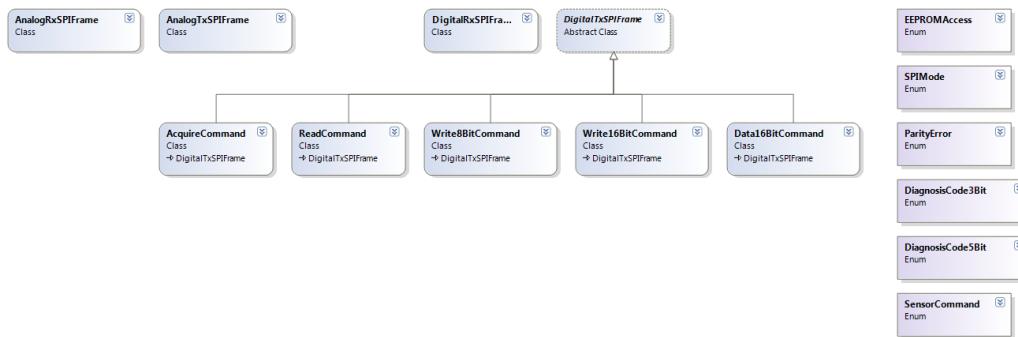


Abbildung 37: Klassendiagramm der mittleren Ebene

**Die mittlere Ebene** Die mittlere Ebene bietet Rahmen für die SPI-Kommunikation an, die mit Hilfe der untersten Ebene verschickt werden.

Es gibt Rahmen für das Senden und für das Empfangen von Daten. Die digitalen Drucksensoren (KP25x) der obersten Schicht nutzen für die gesamte Kommunikation `DigitalTxSPIFrame` zum Senden und `DigitalRxSPIFrame` zum Empfangen. Analoge Sensoren (KP23x) nutzen die Rahmen `AnalogTxSPIFrame` und `AnalogRxSPIFrame` nur für die EEPROM-Programmierung. Zum Auslesen des aktuellen Druckwerts am Sensor-Pin VOUT, mit Hilfe des ADCs des Mikrocontrollers, wird direkt die `DeviceFunction CmdGetVoltage AnalogInOut` verwendet.

Beim Senden nutzt der digitale Drucksensor der obersten Schicht einen `DigitalTxSPIFrame`. Die abgeleiteten Klassen bestimmen das konkrete 16-Bit-SPI-Kommando nach Tabelle 5 auf Seite 23. Die abgeleiteten Klassen enthalten das konkrete 16-Bit-SPI-Kommando als Variable vom Typ `SensorCommand`. Das 2 Byte (entspricht 16 Bit) große Enum `SensorCommand` enthält alle verfügbaren SPI-Kommandos der digitalen Sensoren (siehe Listing 9).

```

1  namespace SensorModule.SPI
2  {
3      /// <summary>
4      /// All available commands, which are understood by a
5      /// sensor of series KP25x.
6      ///
7      /// Each command is of size 16 bits (= 2 bytes).
8      ///
9      /// Note:
10     /// Not every sensor supports all available sensor
11     /// commands!
12     /// </summary>
13     public enum SensorCommand : ushort
14     {
15         /// <summary>
16         /// Command to acquire the pressure.
17         /// </summary>
18         AcquirePressure = 0x2000,
19         ...
20     }
21 }
```

Listing 9: Das Enum SensorCommand

Beim Empfangen der SPI-Antwort wird ein `DigitalRxSPIFrame` genutzt. Dieser Rahmen extrahiert aus der Antwort die einzelnen Bitfelder nach Abbildung 9, Seite 24. Die einzelnen Bitfelder können danach als simple Variablen dieser Klasse angesprochen werden.

Analoge Drucksensoren der oberen Schicht nutzen die SPI-Rahmen `AnalogTxSPIFrame` und `AnalogRxSPIFrame` nach dem selben Prinzip wie die digitalen Drucksensoren; jedoch nur zur EEPROM-Programmierung.

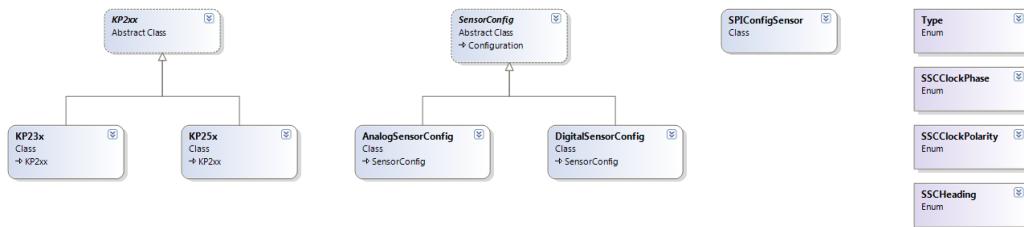


Abbildung 38: Klassendiagramm der obersten Ebene

**Die oberste Ebene** Die oberste Ebene deckt die konkreten Sensoren ab.

Die digitalen Drucksensoren KP253, KP254 und KP256 werden in der Klasse KP25x vereint, die analogen Drucksensoren KP234, KP235 und KP236 in der Klasse KP23x. Dies ist möglich, da sich die digitalen Drucksensoren identisch verhalten und sich nur an Hand der Transfer-Funktion unterscheiden. Beim KP253 ist zusätzlich noch die veränderte SPI-Antwortstruktur zu beachten (vergleiche Abbildungen 9 und 10 auf Seite 24). Die analogen Sensoren unterscheiden sich nur in den Transfer-Funktionen.

Funktionalitäten wie Druckmessen, die den Klassen KP25x und KP23x gemein sind, werden in eine abstrakte Basisklasse KP2xx abstrahiert und dann in den konkreten Klassen KP25x und KP23x implementiert. Die abstrakte Basisklasse KP2xx stellt sicher, dass es Methoden wie

- `ActivateTestMode` zum Aktivieren des Testmodus',
- `BurnEEPROM` zum Programmieren des EEPROMs,
- `GetPressure` zum Messen Drucks

sicher gibt.

Da sich die verwendeten Sensoren (KP234, KP235 usw.) eigentlich nur in der Transfer-Funktion unterscheiden, wird dieser Parameter in eine eigene Konfigurationsdatei ausgelagert und nicht im Quelltext hinterlegt. Dadurch können ganz einfach neue Typen innerhalb einer Familie hinzugefügt werden, ohne die Anwendung neu kompilieren zu müssen. Dies macht die Anwendung zukunftssicher. So besitzt jede Sensor-Familie (KP25x und KP23x) eine eigene XML-Datei. Innerhalb dieser XML-Datei gibt es für jeden konkreten Typ (KP234, KP235 usw.) eine eigene Section mit spezifischen Parametern. Dazu zählen bei den digitalen Drucksensoren auch SPI-Parameter wie Clock-Polarität und Clock-Phase. Diese Parameter werden verwendet, um mit Hilfe der DeviceFunction `CmdConfigHardwareSSC` das SSC des Mikrocontrollers einzustellen. Die SPI-Parameter sind zwar bei allen bisher verwendeten Sensoren identisch, jedoch könnten zukünftige Typen dieser Sensor-Familie andere Parameter verwenden. Listing 10 zeigt beispielhaft den Aufbau der XML-Datei.

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <ArrayOfDigitalSensorConfig ...>
3   <DigitalSensorConfig>
4     <SensorName>KP253</SensorName>
5     ...
6     <SPIConfig>
7       <ClockPolarity>IdleClockLow</ClockPolarity>
8       <ClockPhase>ShiftDataOnLeadingEdge</ClockPhase>
9       <Heading>MSBFIRST</Heading>
10      </SPIConfig>
11      <PressureMinimum>60</PressureMinimum>
12      <PressureMaximum>165</PressureMaximum>
13      <PressureOutputVoltageMinimum>0</
14        PressureOutputVoltageMinimum>
15      <PressureOutputVoltageMaximum>4095</
16        PressureOutputVoltageMaximum>
17      <TemperatureMinimum>-40</TemperatureMinimum>
18      <TemperatureMaximum>160</TemperatureMaximum>
19      <TemperatureOutputVoltageMinimum>0</
20        TemperatureOutputVoltageMinimum>
21      <TemperatureOutputVoltageMaximum>4095</
22        TemperatureOutputVoltageMaximum>
23    </DigitalSensorConfig>
24    <DigitalSensorConfig>
25      <SensorName>KP254</SensorName>
...
</DigitalSensorConfig>
...
</ArrayOfDigitalSensorConfig>
```

Listing 10: XML-Datei für Sensorfamilie KP25x

In der GUI (Abbildung 34) verbindet sich der Benutzer mit dem Programmiergerät und kann dann im Anschluss über das Drop-Down-Menü „Sensor Family“ die Sensorfamilie wählen. Daraufhin wird die XML-Datei der ausgewählten Sensorfamilie deserialisiert und alle darin enthaltenen konkreten Typen im darunterliegenden Drop-Down-Menü „Sensor“ angezeigt. Dadurch kann man schnell und einfach neue Sensortypen für eine Sensorfamilie nachrüsten, indem man neue Sections in der XML-Datei einfügt. Dies Prinzip wird sowohl bei den digitalen als auch den analogen Drucksensoren angewandt.

Die Enums auf dieser Ebene liefern vordefinierte Werte (z.B. „IdleClockLow“ für 0<sub>10</sub>, Listing 10) für die SPI-Parameter Clock-Polarität, Clock-Phase und Reihenfolge der Bits (MSB oder LSB zuerst). Diese Vorgaben werden durch die Mikrocontroller-Firmware gemacht und genau diese Enums sind auch in

der Firmware vorhanden.

Mit dem Voranschreiten der Geschäftslogik wird auch Schritt für Schritt die GUI weiterentwickelt und mit Funktionalitäten gefüllt. So kann beispielsweise der Graph aktualisiert werden, wenn bei den Sensoren die Methode `GetPressure` implementiert ist. Die fertige GUI — das für den Benutzer offensichtliche Produkt des Entwicklungsprozesses — wird in Kapitel 4 vor gestellt.

Abschließend noch drei Anmerkungen zum Entwicklungsprozess.

1. Der Quelltext (Klassen, Variablen, Methoden etc.), welcher im Rahmen des Kapitels 3.3.1 entstand, ist durchweg mit XML-Kommentaren als API-Dokumentation versehen. Dies widerspricht zwar in Teilen den Empfehlungen aus [16], aber erleichtert Personen die zukünftig mit dem Code in Kontakt kommen den Einstieg. Die umfangreiche API-Dokumentation und der beschreibende Code führen im Gegenzug dazu, dass kaum Zeilen- und Blockkommentare verwendet werden.
2. Aus der selben Code-Basis werden zwei Versionen generiert. Eine Version, für den hausinternen Gebrauch mit zusätzlicher Unterstützung zur EEPROM-Programmierung. Die andere Version für Kunden, die nur für Druck- und Temperaturmessungen gedacht ist. Dies ist über C#s Conditional Compilation Symbols<sup>6</sup> möglich. Über die Direktiven `#if`, `#elif`, `#else`, und `#endif` lassen sich die Code-Stellen markieren, die nicht für Kunden gedacht sind (siehe Listing 11).

```
1 namespace Namespace
2 {
3     public class Class
4     {
5         public void ConfidentialMethod()
6         {
7             #if SymbolIsDefined
8                 // Confidential Code
9                 ...
10            ...
11        #endif
12    }
13    ...
14 }
```

<sup>6</sup>Siehe MSDN Library unter: <http://msdn.microsoft.com/en-us/library/aa691095%28v=vs.71%29.aspx>

```
15 }  
16 }
```

Listing 11: Conditional Compilation Symbols in C#

Dadurch ist sichergestellt, dass die markierten Code-Abschnitte bereits vom Compiler bzw. Präprozessor während der lexikalischen Analyse verworfen werden — sofern das Conditional Compilation Symbol nicht definiert ist — und auch nicht in der ausführbaren Datei enthalten sind. Diese Code-Abschnitte können deshalb auch nicht mehr mit Hilfe eines Decompilers hergestellt werden.

3. Unit-Tests für Klassen der Geschäftslogik werden erst im Nachhinein geschrieben. Aus Zeitgründen werden aber nur wenige Klassen durch Tests abgedeckt.

### 3.3.2 Installationsprogramm

Das Installationsprogramm hat die Aufgabe alle notwendigen Dateien auf das Zielsystem zu kopieren und auch eine Deinstallationsroutine anzubieten. Die zu installierenden Dateien umfassen:

- Das zu einer EXE kompilierte Modul **EvalkitSoftware**.
- Die anderen Software-Module als DLLs.
- Die von FTDI bereitgestellte DLL, welche durch das Modul **FTDIDeviceModule** in C#-Code gewrapped ist.
- Die Datei *app.config*. Diese XML-Datei wird standardmäßig von Visual Studio bei GUI-Projekten (hier **EvalkitSoftware**) erzeugt. Über den grafischen Dialog *Settings.settings* in den Projekt-Properties der IDE werden **StringValuePairs** angelegt, die in der ganzen Anwendung verwendet werden können. Diese Datei wird bei jedem Programmstart neu eingelesen. Mit dieser Datei kann der Pfad für die temporären Sensormessdaten variabel gehalten werden.
- Die XML-Konfigurations-Dateien für die Sensorfamilien KP23x und KP25x.
- Die Datei *CmdDefines.h*.

- Das Benutzerhandbuch aus Kapitel 3.3.3.

Das Installationsprogramm wird mit dem Windows Installer XML (WiX) Toolset erstellt. WiX ist eine Programmsammlung von Microsoft, die aus einer XML-Beschreibung ein fertiges Installationspaket (mit der Endung `.msi`) erstellt. WiX steht unter der freien Common Public License und wird unter <http://wix.codeplex.com/> zur Verfügung gestellt. CodePlex ist eine von Microsoft betriebene Plattform für Open-Source-Software-Projekte.

WiX erstellt das Installationspaket an Hand von zwei XML-Konfigurations-Dateien:

- `Files.wxs`: Enthält alle zu kopierenden Dateien (siehe oben).
- `Product.wxs`: Enthält zum Einen eine Beschreibung der Software (z.B. Name, Version etc.), die von Windows unter „Systemsteuerung → Software“ angezeigt wird. Zum Anderen Anweisungen, die vor und nach der Installation der eigentlichen Dateien ausgeführt werden. Diese Anweisungen geben z.B. den Standardpfad für die Installation vor (**`C:\Program Files\`**). Mit Hilfe einer Anweisung die nach der Installation ausgeführt wird, ist es möglich, die Datei `app.config` zu editieren. In ihr wird der Pfad für die temporäre Speicherung von Sensormessdaten geändert. Während der Entwicklung mit Visual Studio zeigt der Pfad in das aktuelle Arbeitsverzeichnis, das Schreibrechte für normale Benutzer gewährt. Nach der Installation wäre dies aber standardmäßig **`C:\Program Files\`**. Dort bestehen in der Regel für normale Benutzer keine Schreibrechte. Microsoft Windows stellt aber mit der Umgebungsvariable `%ALLUSERSPROFILE%` einen Ordner zur Verfügung in dem alle Benutzer Schreibrechte besitzen. Dieser Ordner kann daher für temporäre Dateien einer Anwendung benutzt werden. Die Umgebungsvariable `%ALLUSERSPROFILE%` zeigt
  - bis Windows XP auf **`C:\Documents and Settings\All Users\`**,
  - ab Windows Vista auf **`C:\ProgramData\`**.

WiX erstellt die Datei `setup.exe` und das Installationspaket mit der Endung `.msi`. `setup.exe` ruft das eigentliche Installationspaket auf. In ihm sind alle spezifizierten Dateien eingepackt. Während der Installation werden sie ausgepackt und an die entsprechende Stelle kopiert. WiX erstellt auch einen Eintrag im Windows-Startmenü.

Über „Systemsteuerung → Software“ kann die Anwendung wieder deinstalliert werden. Die automatisch von WiX bereitgestellte Deinstallationsroutine macht alle Kopiervorgänge der Installation rückgängig.

### 3.3.3 Benutzerhandbuch

Um Benutzer mit der GUI und der Funktionsweise des Evaluierungsgeräts vertraut zu machen, wird ein Handbuch in englischer Sprache erstellt. Infineon setzt dabei auf Adobe FrameMaker 8. Bei FrameMaker handelt es sich um einen WYSIWYG-Editor zu Erstellung von Dokumenten (siehe Abbildung 39).

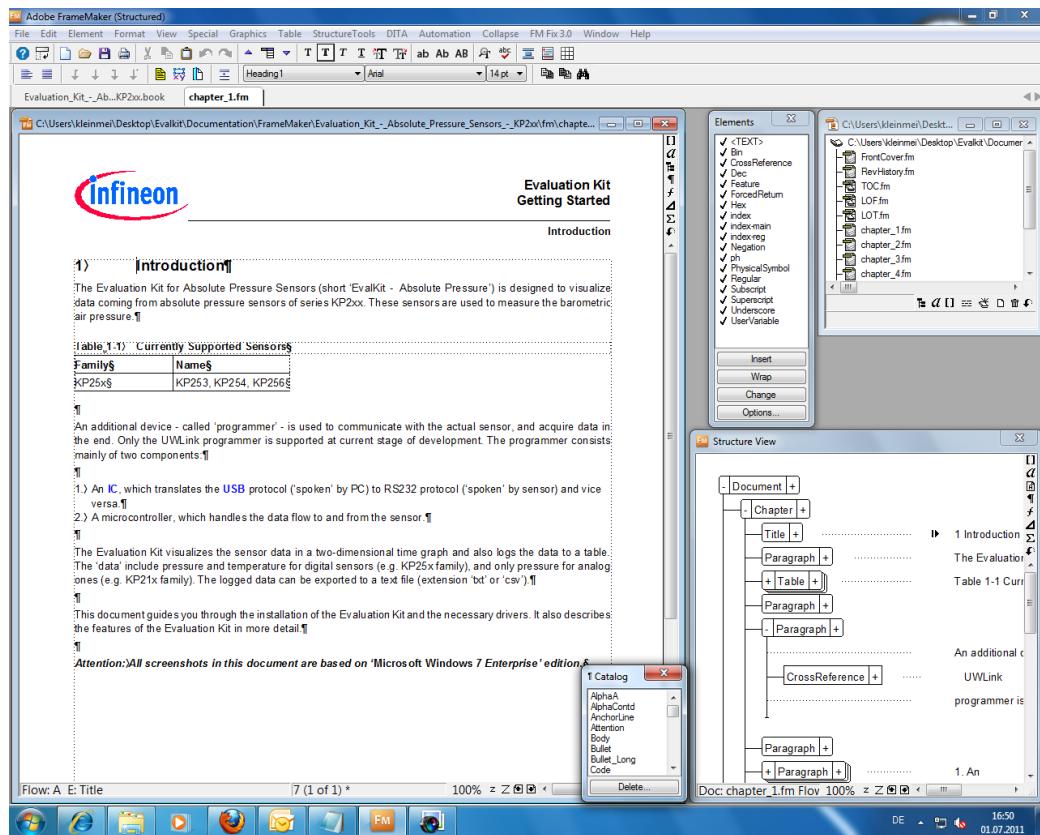


Abbildung 39: Das Benutzerhandbuch mit FrameMaker 8

Mit FrameMaker wird sowohl die interne Version, als auch die für Kunden gedachte Version der GUI beschrieben. Zuerst wird die Dokumentation ge-

schrieben, welche alle Merkmal des Evaluierungswerkzeuges beschreibt. Über die Funktion „Conditional Text“ wird dann der Text ausgeblendet, der für die Kundenversion nicht gebraucht wird.

## 4 Ergebnis: Die grafische Benutzeroberfläche

Der Entwicklungsprozess lieferte nach 98 Tagen insgesamt 3.551 Zeilen C-Code<sup>7</sup> für die beiden Mikrocontroller-Firmwares, davon 1.737 Zeilen für den 16-Bit-Mikrocontroller sowie 1.814 Zeilen für den 8-Bit-Mikrocontroller und 16.355 Zeilen C#-Code<sup>8</sup> für die GUI. Gezählt wurden dabei Leerzeilen, Kommentare und der reine Code (Anweisungen). Darin nicht enthalten sind die bereits vorhanden Software-Module der Firmwares und auch der GUI. Insgesamt umfasst das Projekt 7.477 Zeilen Firmware-Code (16- und 8-Bit-Mikrocontroller) und 24.190 Zeilen GUI-Code. Eine genaue Übersicht über die gesamte Codebasis liefern die Tabellen 17, 18 und 19.

Sprache	Dateien	Leer- zeilen	Kommentar- zeilen (API-Doku)	Code (Anweisungen)
C	4	99	292	233
C/C++ Header	5	84	514	592
Summe	9	183	806	825

Tabelle 17: Lines of Code Firmware 8-Bit-Mikrocontroller

Sprache	Dateien	Leer- zeilen	Kommentar- zeilen (API-Doku)	Code (Anweisungen)
C	19	227	1099	1679
C/C++ Header	25	103	1322	1223
DOS Batch	1	0	0	10
Summe	45	330	2421	2912

Tabelle 18: Lines of Code Firmware 16-Bit-Mikrocontroller

<sup>7</sup>Ermittlung mit CLOC Version 1.55: <http://cloc.sourceforge.net/>

<sup>8</sup>Ermittlung mit C# and VB.NET Line Count Utility Version 1: <http://richnewman.wordpress.com/2007/07/01/c-and-vbnet-line-count-utility/>

Modul	Zeilen	Leer-zeilen	Windows Forms-Designer	Kommentar-zeilen (API-Doku)
EvalkitSoftware	11404	664	4387	2004
FTDIDeviceModule	2032	57	0	960
HelperLibrary	2210	143	0	791
MemoryModule	1107	54	0	406
ProgrammerModule	2486	135	0	1181
SensorModule	4951	485	0	2447
Summe	24190	1538	4387	7789

Tabelle 19: Lines of Code GUI

Obwohl das Evaluierungswerkzeug softwareseitig aus Firmware und GUI besteht, ist für den Benutzer das offensichtliche Endprodukt die GUI. Alle wichtigen Merkmale dieser werden im Laufe dieses Kapitels vorgestellt.

## 4.1 Die drei Kontrollelemente der GUI

Die GUI besteht aus den drei Kontrollelementen:

- Menüleiste
- Seitenleiste
- Bereich für Sensordaten

Siehe Abbildung 40 auf Seite 90.

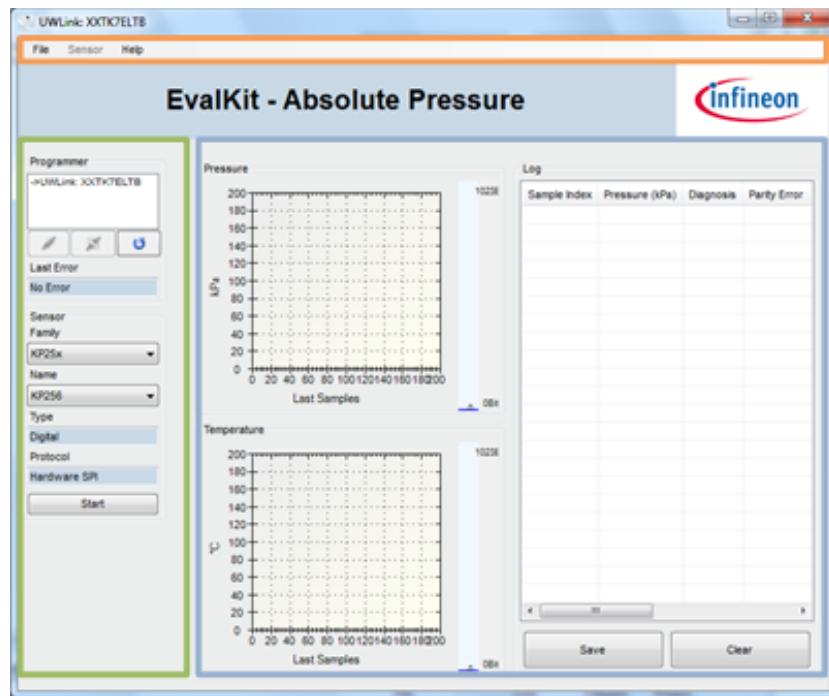


Abbildung 40: Kontrollelemente der GUI nach Auswahl eines Sensors

#### 4.1.1 Menüleiste

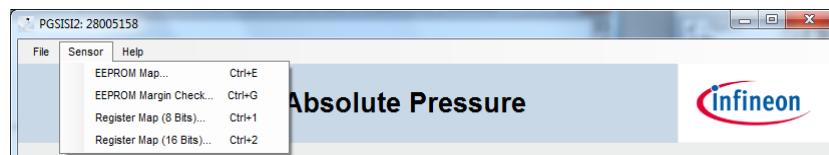


Abbildung 41: Menüleiste in der GUI

Über die Punkte „File“ „Sensor“ und „Help“ in der **Menüleiste** kann das aktuelle Fenster geschlossen werden (jeder Sensor bekommt sein eigenes Fenster), das EEPROM des Sensors betrachtet sowie programmiert werden (siehe Abbildung 42) und das Benutzerhandbuch geöffnet werden.

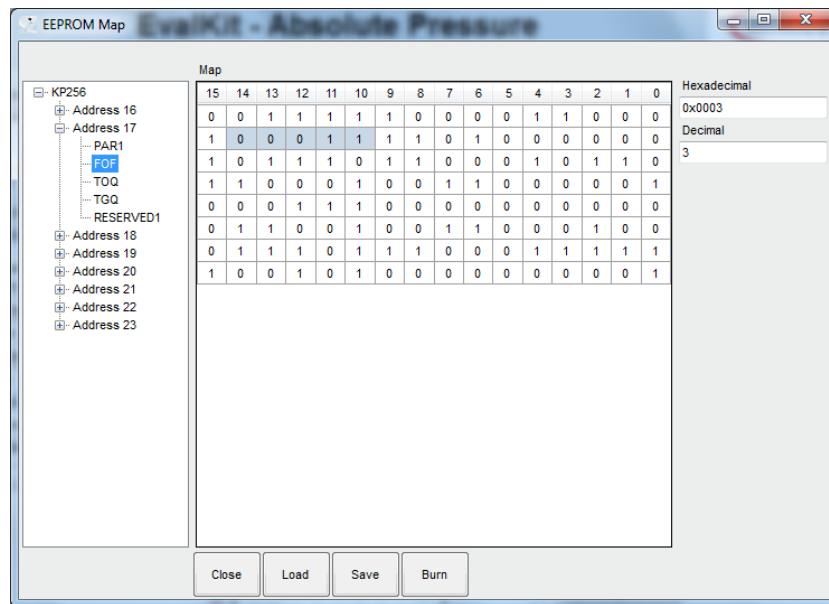


Abbildung 42: EEPROM-Programmierung über das Fenster EEPROM Map

Im linken Teilfenster lassen sich einzelnen Bitfelder innerhalb einer Adresse auswählen. Über die Textboxen „Hexadecimal“ oder „Decimal“ lassen sich die Werte eines Bitfeldes ändern. Die Paritäts-Bits werden dabei automatisch neu berechnet. Über die Buttons am unteren Rand kann die aktuell angezeigte Konfiguration als XML-Datei gespeichert werden, eine Speicherkonfiguration geladen werden und das EEPROM programmiert werden.

#### 4.1.2 Seitenleiste

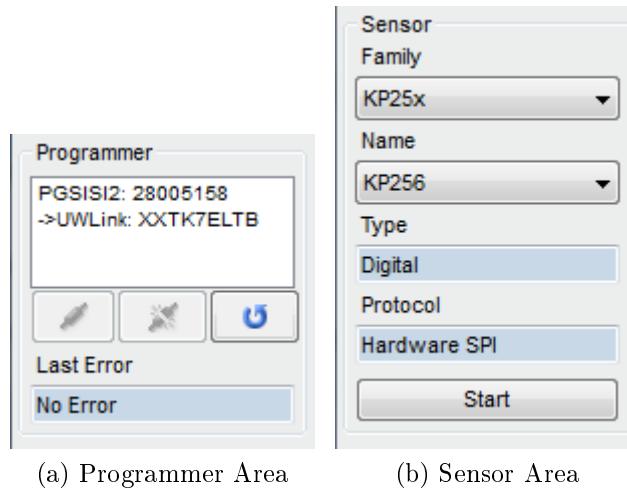


Abbildung 43: Programmer und Sensor Area in der Seitenleiste

Über die **Seitenleiste** verbindet man die GUI zuerst mit einem Programmiergerät. Dann wird über das Drop-Down-Menü „Family“ die Sensorfamilie gewählt. Im darunterliegenden Drop-Down-Menü „Name“ wird der konkrete Sensor innerhalb dieser Familie gewählt.

#### 4.1.3 Bereich für Sensordaten

Nach den Einstellungen in der Seitenleiste baut sich der **Bereich für Sensordaten** auf. Mit dem Start-/Stop-Button können Messungen gestartet, gestoppt und wieder fortgesetzt werden (siehe Abbildung 44).

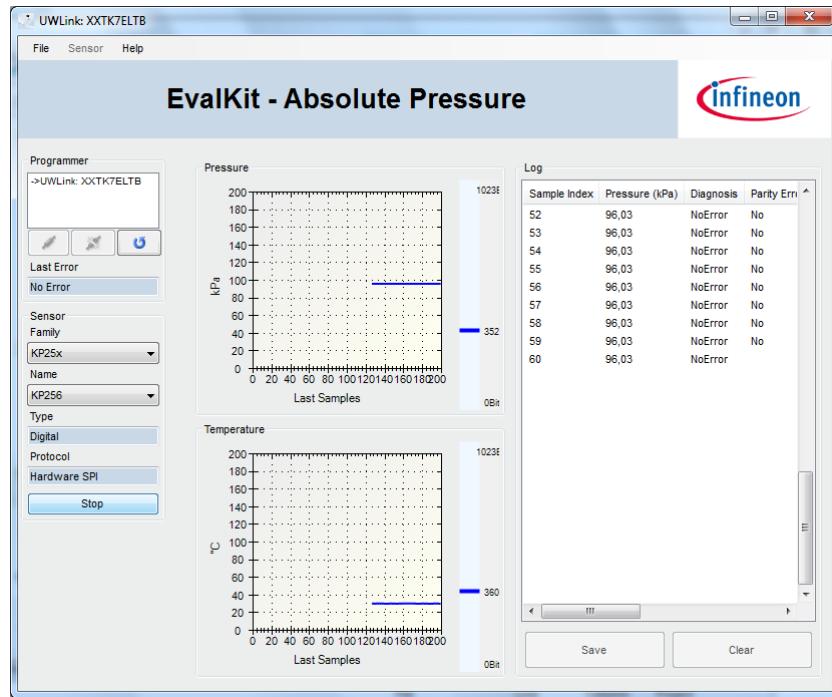


Abbildung 44: Bereich für Sensordaten nach gestarteter Messung

Der Bereich für die Sensordaten ist logisch zweigeteilt.

**Graphen** Im linken Bereich befinden sich die Graphen. Sie zeigen die letzten 200 Druckwerte — bei den digitalen Sensoren auch die letzten 200 Temperaturwerte — an. Über das Kontextmenü der Graphen kann die Skalierung der Y-Achse verändert werden. Die Skalierung kann entweder frei gewählt werden oder automatisch auf den Bereich um den Mittelwert der letzten Messungen skaliert werden (siehe Abbildung 45). Im Balken rechts neben dem Graphen wird der Druckwert vor Umrechnung an Hand der Transfer-Funktion angezeigt. Bei digitalen Drucksensoren ist dieser Wert in Bit, bei analogen in Volt.

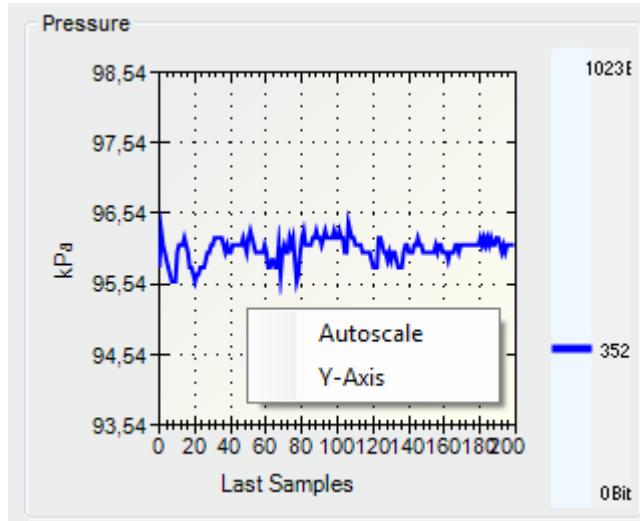


Abbildung 45: Kontextmenü der Graphen

**Log-Bereich** Die Tabelle im Log-Bereich listet die letzten 10.000 Messdaten. Abbildung 46 zeigt das Tabellenformat für die digitalen Sensoren. Bei den analogen Sensoren gibt es nur die Spalten „Sample Index“ und „Pressure (kPa)“.

Log									
Sample Index	Pressure (kPa)	Diagnosis	Parity Error	Response	Temperature (°C)	Diagnosis	Parity Error	Response	
0	95,72	NoError	No	0x52b8	30,38	NoError	No	0x52d1	
1	95,82	NoError	No	0x52bb	30,58	NoError	No	0x52d2	
2	95,92	NoError	No	0x52bd	30,38	NoError	No	0x52d1	
3	95,82	NoError	No	0x52bb	30,38	NoError	No	0x52d1	
4	95,82	NoError	No	0x52bb	30,38	NoError	No	0x52d1	

Abbildung 46: Log-Bereich der digitalen Sensoren

Über den Button „Save“ kann die gesamte Messreihe als Textdatei (mit der Endung .txt oder .csv) abgespeichert werden. In die Textdatei werden alle Spalten der Tabelle geschrieben und mit einem frei wählbaren Trennzeichen geteilt. „Clear“ setzt den gesamten Bereich für die Sensordaten zurück und schafft so Platz für eine neue Messreihe. Siehe Abbildung 47.

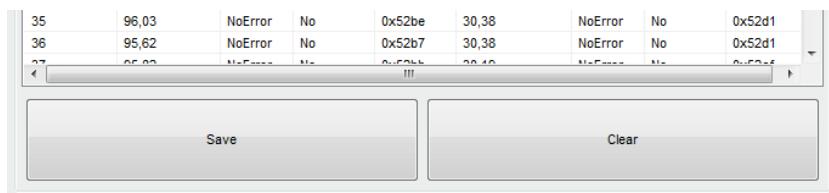


Abbildung 47: Buttons im Log-Bereich

Einen umfassenden Einblick über alle Details der GUI bietet das Benutzerhandbuch.

## 5 Zusammenfassung und Ausblick

Das folgende Kapitel fasst nun alle wichtigen Punkte dieses Projekts nochmals zusammen, gibt eine Übersicht über verbleibende Fragen und bietet einen Ausblick auf die zukünftige Entwicklung des Evaluierungswerkzeuges.

**Rückblick** In den vergangenen Kapiteln wurden die Komponenten des Evaluierungswerkzeuges detailliert beschrieben. Darunter die digitalen Drucksensoren KP253, KP254 sowie KP256, die zusätzlich über einen Temperatursensor verfügen und über SPI angesprochen werden. Daneben die analogen Drucksensoren KP234, KP235 und KP236, die ihren gemessenen Druckwert an einem einzigen Sensor-Pin ausgeben, welcher über einen ADC eines Mikrocontrollers eingelesen wird. Zusätzlich besteht das Evaluierungswerkzeug aus den Programmiergeräten PGSISI2 (mit dem 16-Bit-Mikrocontroller XC164CS-16F) und UWLink (mit dem 8-Bit-Mikrocontroller XC886CLM-8FF). Das Programmiergerät bildet die Schnittstelle zwischen dem Sensor und der GUI, welche auf dem PC läuft. Sie zeigt die Druckwerte an und komplettiert damit das Evaluierungswerkzeug.

Im weiteren Verlauf wurde der Softwareentwicklungsprozess erläutert. Dabei wurde die Firmware für den 16-Bit-Mikrocontroller der PGSISI2 in C programmiert, wobei auf bereits vorhandenem Code zurückgegriffen werden konnte. Mit der PGSISI2 ist es möglich mit den digitalen Drucksensoren mittels SPI zu kommunizieren. So können gemessene Druck- und Temperaturwerte eingelesen werden und das EEPROM des Sensors programmiert werden. Bei den analogen Sensoren wird der Druck über den ADC des Mikrocontrollers eingelesen. Die Programmierung des EEPROMs der analogen Sensoren ist auch möglich. Dazu werden wie bei SPI vier Leitungen verwendet — wobei eine Leitung zum Anlegen der Programmierspannung dient und nicht als NCS verwendet wird — und das Verhalten eines SPIs emuliert. Die Firmware für den 8-Bit-Mikrocontroller des UWLinks unterstützt ausschließlich die digitalen Drucksensoren ohne EEPROM-Programmierung. Die logische Verbindung zwischen Firmware und der GUI am PC bildet ein einfaches Protokoll. GUI und Firmware interagieren dazu nach dem Master-Slave-Prinzip. Die GUI sendet dazu, als Master, Kommandos an die Firmware nach dem Muster < Command >< Payload Length >< Payload >. Die Firmware, als Slave, reagiert nur auf GUI-Anfragen, bearbeitet diese und schickt dann eine Antwort nach dem Muster < Command >< OK >< Payload Length >< Payload > zurück. Die GUI wurde in C# geschrieben und nutzt Microsofts

.NET-Framework 2.0. Sie basiert auf einem Template, das bei der Infineon Technologies AG für ähnliche Sensorprojekte eingesetzt wird. Das Template besteht aus sechs Softwaremodulen und bietet u.a. bereits Klassen zum Zugriff auf das Programmiergerät. Zwei dieser Module wurden implementiert. Ein Modul das den, mit dem Windows Forms-Designer gestalteten, GUI-Code aufnimmt. Das andere Module widmet sich dem Nicht-GUI-Code — der Geschäftslogik. Die Geschäftslogik besteht aus drei logischen Ebenen. Die unterste Ebene bietet Zugriff auf die Funktionalitäten der Firmware. Die Ebene darüber bietet Rahmen für die Kommunikation mit der Firmware. Diese Rahmen geben den Bit-Streams der darunterliegenden Ebene Bedeutung, indem Bitfelder unterschiedlicher Länge extrahiert werden. Die oberste Ebene bilden dann die Klassen für die konkreten Sensoren. Ein Installer, der die fertige Applikation via Doppelklick installierbar macht und ein Benutzerhandbuch haben den Entwicklungsprozess abgeschlossen.

Das folgende Kapitel bot einen Einblick in das Ergebnis, welches für Anwender die größte Bedeutung hat: die grafische Benutzeroberfläche. Sie liegt in zwei Versionen vor. Eine Version mit allen Merkmalen, für den internen Gebrauch bei der Infineon Technologies AG und einer Version mit reduzierten Funktionsumfang für Kunden. Beide Versionen nutzen die selbe Code-Basis und haben demzufolge auch das gleiche Erscheinungsbild. Insgesamt verfügt die GUI über drei logische Bereiche: Menüleiste, Sensorleiste und einen Bereich für Sensordaten.

**Das Ergebnis** Alle diese Punkte zeigten die Entwicklung eines Evaluierungswerkzeuges für barometrische Luftdruck-Sensoren. Ziel dieses Projekts war aber nicht nur eine funktionierende Anwendung zu kreieren, sondern die beiden Software-Bausteine Firmware und GUI so modular und generisch zu gestalten, dass sie auch in anderen Projekten problemlos eingesetzt werden können. Bei der Firmware in der Programmiersprache C ist dies gelungen, indem die einzelne Hardware-Module des Mikrocontrollers jeweils in eigenen Dateien (mit den Endungen .c und .h) implementiert wurden. Diese Dateien können nun bei Bedarf in anderen Projekten verwendet werden. Die GUI in C# war bereits durch das vorgegebene Template stark modularisiert und trennte GUI-Code von Geschäftslogik. Die eigentliche GUI muss für jedes Projekt individuell und damit neu gestaltet werden. Unterstützung geben dabei die Module, die im Template vorhanden sind. Diese wurden im Laufe des Projekts punktuell erweitert, um zukünftig noch komfortabler arbeiten zu können. Besonders die statischen Methoden zur Konvertierung von Byte-

Arrays in C#-Werttypen und umgekehrt wurden ergänzt. Darüber hinaus auch die Klassen zur Ansteuerung der Programmiergeräte.

**Relevanz** Dieses Projekt brachte keine neuen wissenschaftlichen Erkenntnisse hervor — dies war aber auch kein Ziel. Jedoch wurden selbige genutzt, um eine modulare Software mit einer zukunftsfähigen Architektur zu entwickeln. Das entstandene Evaluierungswerkzeug, in Verbindung mit dem UW-Link als Programmiergerät, ist für Kunden nun eine kostengünstige Unterstützung für die Entwicklung von Anwendungen auf Basis von barometrischen Luftdruck-Sensoren. Die Infineon Technologies AG kann nun mit Hilfe des PGSISI2-Programmiergeräts das Verhalten ihrer eigenen Luftdruck-Sensoren genauer prüfen und überwachen.

**Verbleibende Fragen** Ein Aspekt wurde bei der Entwicklung des Evaluierungswerkzeugs jedoch vernachlässigt: automatisiertes Testen auf Seiten der Firmware und der GUI. Beides ist notwendig, um Code-Refactoring zu erleichtern bzw. erst zu ermöglichen. Die Firmware wurde von Hand, mit der Unterstützung eines Oszilloskops, Modul für Modul getestet, jedoch bleibt die Frage eines einheitlichen und vor allem automatisierten Testverfahrens. Mit CUnit<sup>9</sup> existiert ein Unit-Testing-Framework für C. Daneben gibt es Tessy<sup>10</sup>, ein Testwerkzeug für Unit-, Modul-, und Integrationstest speziell für eingebettete Software. Wie jede neue Software erfordern auch diese beiden Einarbeitungszeit und damit auch Kosten. Ob der Bedarf sinnvoll und zielführend ist, muss im Unternehmen geklärt werden, möglicherweise im Rahmen einer weiteren Abschlussarbeit. Auf Seiten der GUI wurde mit Unit-Tests für die Geschäftslogik begonnen, jedoch nicht abgeschlossen. Um eine umfassende Testabdeckung zu erreichen, müssten auch einige Teile der bereits bestehenden Module geändert werden. Für die Automatisierung von UI-Tests ist ab dem .NET-Framework 3.0 der Namespace System.Windows.Automation<sup>11</sup> vorgesehen. Da die GUI aber auf dem .NET-Framework 2.0 aufsetzt, muss eine andere Lösung gesucht werden. Bzw. im Allgemeinen Kosten und Nutzen abgewägt werden.

---

<sup>9</sup>Online unter: <http://cunit.sourceforge.net/>

<sup>10</sup>Online unter: <http://www.hitech.com/index.php?id=172&L=2>

<sup>11</sup>Online unter: <http://msdn.microsoft.com/de-de/library/system.windows.automation%28v=vs.85%29.aspx>

**Ausblick** Software muss sich immer weiterentwickeln, so auch das Evaluierungswerkzeug. In naher Zukunft soll es auch mit dem UWLink möglich sein analoge Sensoren anzusprechen. Auf Grund der soliden Softwarearchitektur muss dazu nur die Firmware um ein ADC-Modul erweitert werden. Beim GUI-Code sind dagegen nur kleinste Änderungen notwendig.

## Glossar

- ADC** Englisches Akronym für **Analog Digital Converter**. Ein elektronisches Bauteil, das ein analoges Eingangssignal in digitale Daten umwandelt.
- API** Englisches Akronym für **Application Programming Interface**. Die Software-Schnittstelle, die ein Programm für andere Programme zur Verfügung stellt (z.B. Methoden).
- ASC** Englisches Akronym für **Asynchronous Serial Channel**. **ASC** wird als Synonym für **UART** verwendet.
- CAN** Englisches Akronym für **Controller Area Network**. Ein asynchrones und serielles Bussystem, das speziell für die Verwendung im Automobilbereich entwickelt wurde.
- DLL** Englisches Akronym für **Dynamic Link Library**. Eine Software-Bibliothek, die Programmcode und Daten enthält, die von anderen Programmen genutzt werden können.
- DSP** Englisches Akronym für **Digital Signal Processor**. Ein Prozessor, der für die Verarbeitung von speziellen digitalen Signalen (z.B. Audiosignale) optimiert ist.

- EEPROM** Englisches Akronym für **Electrically Erasable Programmable Read-Only Memory**. Ein nichtflüchtiger, elektronischer Speicher.
- FTDI** Englisches Akronym für **Future Technology Devices International**. Ein englischer Entwickler und Hersteller von **ICs**, um eine Protokollübersetzung nach **USB** vorzunehmen.
- GUI** Englisches Akronym für **Graphical User Interface**. Eine Oberfläche am PC, bestehend aus grafischen Elementen, die mit Tastatur **und** Maus gesteuert wird - im Gegensatz zu einem Command-Line Interface, welches nur Tastatureingaben akzeptiert.
- I<sup>2</sup>C** Englische Kurzform für **Inter-IC-Interface**. Ein synchrones und serielles Zweidraht- bzw. Dreidraht-Bussystem.
- IC** Englisches Akronym für **Integrated Circuit**. Eine elektronische Schaltung, dessen Bestandteile vollständig auf einem einzelnen (Halbleiter-) Substrat aufgebracht sind im Gegensatz wie bei einem **PCB**.
- PCB** Englisches Akronym für **Printed Circuit Board**. Eine Platine aus isolierendem Material, das mit Leiterbahnen durchzogen ist und als Träger für elektronische Bauteile dient.

**PGSISI2** Englisches Akronym für **Programmer System Infineon Sensor Interface 2**. Ein **PCB** mit aufgebrachtem Mikrocontroller und den seriellen Schnittstellen **RS232C** und **USB**. Wird als Schnittstelle zwischen einem Sensor und einem Computer verwendet.

**RS232C** Eine serielle Schnittstelle mit synchroner und asynchroner Übertragungsmöglichkeit zur Verbindung von Computern mit externen Geräten. Ist heute weitgehend durch **USB** verdrängt worden.

**SPI** Englisches Akronym für **Serial Peripheral Interface**. Ein synchrones und serielles Bussystem, das vier Leitungen (zwei Steuer- und zwei Datenleitungen) für die Kommunikation verwendet: CLK, NCS, SDO und SDI (Serial Clock, Not Chip Select, Serial Data Out und Serial Data In).

**SSC** Englisches Akronym für **Synchronous Serial Channel**. Ein elektronisches Bauteil, das die serielle Datenübertragung

gemäß **SPI**-Spezifikation handhabt und üblicherweise direkt in einem Mikrocontroller integriert ist. **SSC** und **SPI** werden oftmals synonym verwendet.

**UART** Englisches Akronym für **Universal Asynchronous Receiver Transmitter**. Ein elektronisches Bauteil, das zum bitseriellen Senden und Empfangen von digitalen Daten genutzt wird. Zumeist nutzen **UARTs** für die Kommunikation den **RS232C**-Standard. **UARTs** sind heute üblicherweise direkt in Mikrocontrollern integriert.

**USB** Englisches Akronym für **Universal Serial Bus**. Ein serieller Bussystem mit synchroner und asynchroner Betriebsart, um Computer mit externen Geräten zu verbinden.

**UWLink** Englische Abkürzung für **Universal Wireless Link**. Ein **PCB** mit aufgebrachtem Mikrocontroller und einer **USB**-Schnittstellen. Wird als Schnittstelle zwischen einem Sensor und einem Computer verwendet.

## Literatur

- [1] ARM LIMITED: *ISO/ANSI Compliance*. Website, 2012. – Online unter <http://www.keil.com/product/isoansi.asp>, besucht am: 9. Februar 2012
- [2] BEIERLEIN, Thomas ; HAGENBRUCH, Olaf: *Taschenbuch Mikroprozessortechnik*. 4. Auflage. München : Hanser, 2011. – ISBN 978-3-4464-2331-2
- [3] GOLDE, Peter ; HEJLSBERG, Anders ; TORGERSEN, Mads ; WILTAMUTH, Scott: *The C# Programming Language*. 4th Edition. Boston, Massachusetts : Addison-Wesley, 2010. – ISBN 978-03-2174-176-9
- [4] HOFFMANN, Jörg: *Taschenbuch der Messtechnik*. 6. Auflage. München : Hanser, 2011. – ISBN 978-3-4464-2391-6
- [5] INFINEON TECHNOLOGIES AG: *XC164-16 - User Manual*. Revision 2.1, 2004
- [6] INFINEON TECHNOLOGIES AG: *XC886/888CLM - Data Sheet*. Revision 1.2, 2009
- [7] INFINEON TECHNOLOGIES AG: *KP234 - Data Sheet*. Revision 1.0, 2010
- [8] INFINEON TECHNOLOGIES AG: *KP235 - Data Sheet*. Revision 1.0, 2010
- [9] INFINEON TECHNOLOGIES AG: *KP236 - Data Sheet*. Revision 1.0, 2010
- [10] INFINEON TECHNOLOGIES AG: *UWLink Mainboard User Guide*. Revision 1.0, 2010
- [11] INFINEON TECHNOLOGIES AG: *KP253 - 12-Bit Data Format Description*, 2011
- [12] INFINEON TECHNOLOGIES AG: *KP254 - Data Sheet*. Revision 0.6, 2011
- [13] INFINEON TECHNOLOGIES AG: *KP256 - Data Sheet*. Revision 0.6, 2011
- [14] KERNIGHAN, Brian W. ; RITCHIE, Dennis: *The C Programming Language*. 2nd Edition. Upper Saddle River, New Jersey : Prentice Hall, 1988. – ISBN 978-01-3110-362-7
- [15] LOUIS, Dirk ; STRASSER, Shinja ; KANSY, Thorsten: *Microsoft Visual C# 2010 - Das Entwicklerbuch*. 1. Auflage. Unterschleißheim : Microsoft Press, 2010. – ISBN 978-38-6645-529-0
- [16] MARTIN, Robert C.: *Clean Code - Refactoring, Patterns, Tests und Techniken für sauberen Code*. 1. Auflage. Bonn : mitp, 2009. – ISBN 978-38-2665-548-7

- [17] MICROSOFT: *MSDN Library - .NET Development*. Website, 2012. – Online unter <http://msdn.microsoft.com/en-us/library/ff361664%28v=vs.110%29.aspx>, besucht am: 29. Januar 2012
- [18] ZAKS, Rodnay: *Programming the Z80*. 3rd Edition. Berkeley, California : Sybex Inc., 1981. – ISBN 978-08-9588-069-7
- [19] ZAKS, Rodnay: *Programming the Z80*. Website, 2009. – Online unter <http://www.z80.info/zaks.html>, besucht am: 29. Januar 2012