Informationstechnik

Fakultät Technik

Facharbeit T2000

Entwicklung wiederverwendbarer Klassen zur Temperatursteuerung und zum Nachrichtenaustausch zwischen Klassen für eine Klassenbibliothek

am 07. September 2021 vorgelegtes Forschungsprojekt

von Patrick Scheich

im Fachbereich Informatik

Bearbeitungsdauer: 1. Dezember 2021 bis 31. Juli 2022

Matrikelnummer: 8060451

Prüfer: Prof. Dr. Mario Babilon

Betreuer: Thanh Ngo

Firma: Murrelektronik GmbH

Eigenständigkeitserklärung

Ich versichere, dass ich diese Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Oppenweiler

Name, Vorname

Ort, den 1. Juli 2022

Kurzzusammenfassung

Die Aufgabenstellung- Gegenstand der Arbeit

Erste Schwierigkeiten

Ziele der Arbeit

Vorgehensweise

Inhaltsverzeichnis

Abbildungsverzeichnis

[1] Analog- Digital- Wandler, engl. analog-digital-converter, elektronisches Bauteil zum quantisieren analoger in digitale Werte

Listings

Abkürzungsverzeichnis

ADC Analog- Digital- Wandler, engl. analog-digital-converter, elektronisches Bauteil zum quantisieren analoger in digitale Werte V

OSR Überabtast- verhältnis, engl. oversampling ratio, Verhältnis von Modulatorfrequenz ind Ausgangsdatenrate 5

SPS Abtastungen in einer Sekunde, engl. samples per second, beschreibt die Datenrate eines AD- Wandlers 6

PGA Verstärker mit programmierbarer Verstärkung, engl. Programmable Gain Amplifier, Operationsverstärker, der als nicht invertierter Verstärker beschaltet ist und einen programmierbaren Analogmultiplexer verbaut hat 6

BOCS Burn-Out-Stromquellen, engl. Burn Out Current Source, Stromquellen für den Sensor Test 6

SPI engl. Serial Peripheral Interface, Bus-System „lockeren“ Standards für synchrone serielle Datenbusse (Synchronous Serial Ports) nach dem Master-Slave-Prinzip 6

LSB Bit mit der geringstejn Bitwertigkeit, engl. Least Significant Bit . . . 8

RTD Messung der Temperatur mit einem Widerstand, engl. Resistance Temperature Detector 13

CPU zentrale Recheneinheit, engl. Central Processing Unit, eines Rechners oder Mikrocontrollers. Umgangssprachlich oft als Prozessor bezeichnet. . . . 14

DMA Direkter Speicherzugriff, engl. Direct Memory Access, seperater Speicherbus mit Controller um Daten ohne Umwege über die CPU direkt in den Arbeitsspeicher schreiben zu können. 21

MCU Mikrocontroller, engl. Micor Controlling Unit, Einplatinenchips, die gleichzeitig einen Prozessor und Peripheriefunktionen enthalten . . . 22

1 Einleitung

„Zeit ist Geld "- schon im Jahr 1748 trifft Benjamin Franklin in seinem Buch „Ratschläge für junge Kaufleute" (nach *Zeit ist Geld* [5]) ein Schlüsselprinzip der modernen Wirtschaft. Franklin erläutert dort die unabdingbar Verzahnung von Zeit und Geld als Unternehmer. Das hat bis heute noch Gültigkeit: Da in der Regel keine leistungsorientierte Vergütung erfolgt, sondern nach Stunden abregerechnet wird, zeigt sich direkt am Mitarbeiter, wie eng Zeit und Geld zusammenhängen.

Effizientes und kostenorientiertes Arbeiten rückt demnach weitaus mehr in den Vordergrund. Das hat natürlich auch seine Gründe: Zum einen erhöht der bewusste Umgang mit Zeit den Umsatz und damit die Gewinnspanne eines Unternehmens, zum anderen fördert es aber auch die Wettbewerbsfähigkeit mit anderen Unternehmen sowie die Konkurrenzfähigkeit des eigenen Landes mit dem Ausland.

Es gilt also, mit ausgeklügelten Techniken Zeitaufwand für Planung, Entwicklung, Umsetzung und Produktion zu minimiern. Gewissermaßen liegt es also in der Verantwortung jeder Abteilung, an Optimierungsprozessen zu arbeiten.

Um den Zeitaufwand für die Softwareentwicklung neuer Produkte zu minimieren, sucht deshalb auch die Entwicklung des Unternehmens nach neuen Konzepten und Wegen, ihre Prozesse zu verkürzen. Dieses Ziel kann erreicht werden, wenn bei der Software-Architektur das Prinzip der Wiederverwendbarkeit stärker betont wird. Dazu müssen Gemeinsamkeiten im Anwendungsbereich der verschiedenen Busmodule identifiziert und Strukturen der Software verbessert werden.

Eine weitere Methode zur Kostenminimierung ist das Ersetzen von Softwareentwicklern durch Programmierer. Dazu muss die Softwareentwicklung allerdings stark genug abstrahiert und vereinfacht werden, um gleichbleibende Qualität zu gewährleisten. Für beide Aspekte gibt es eine abdeckende Lösung:

Um nämlich all das umzusetzen, soll ein umfassendes, firmeninternes Framework[2] entwickelt werden, in dem einzelne Funktionalitäten auf jeweils eine Komponente

[2] Rahmenwerk für Struktur und Architektur von zu entwickelnder Software, bestehend aus Standardmodulen- und Bibliotheken mit genau definierten und dokumentierten Schnittstellen (siehe IT-Glossar [4])

heruntergebrochen werden sollen. Die Benutzerschnittstellen diese Komponenten sollen auf wesentliche Funktionen reduziert werden, um die Nutzung des Frameworks einfach und intuitiv zu gestalten, dabei aber sämtliche für die Module relevanten Funktionen zur Verfügung zustellen und für den späteren Programmierer irrelevante Operationen im Hintergrund durchzuführen.

Folgende Ausarbeitung beschäftigt sich mit dem Aspekt der Wiederverwendbarkeit von Software und gibt einen Ausblick auf deren automatisch Generierung. Im speziellen soll dabei auf den Anwendungsfall, für den die Software vorerst konzipiert wurde, eingegangen werden.

Kern der Arbeit wird demnach das Konzipieren und Entwickeln der Klassen für die analoge Messung der Temperatur mithilfe eines AD- Wandlers, die Auswertung der Daten, Diagnosefunktionen[3] im System sowie das Senden von Fehlercodes im Hinblick der Wiederverwendbarkeit sein. Dabei soll die Wiederverwendbarkeit der Temperaturmessung in den Fokus gestellt werden. Um dies zu präsentieren, wird gezeigt, wie die Klasse ohne große Änderungen zur Spannungsmessungen genutzt werden kann.

Die aus den Diagnosen folgenden Ergebnisse müssen in Errorcodes übersetzt werden. Diese sollen gemäß eigener Definitionen vereinheitlicht werden, um das Mappen (Umformen) auf protokollspezifische Error Codes zu gewährleisten. Im Umfang der Arbeit wird das Mappen auf das IOLink Protokoll gezeigt. Es soll aber auch einen Ausblick auf das Mappen in andere Protokollcodes gegeben werden.

Die Software wird anschließend auf ein analog- Busmodul des Produktkatalogs geflasht werden. Dieses wird mit zu den Messungen passenden Sensoren ausgerüstet werden und soll sämtliche Diagnosen durchführen. Bei erfolgreichem Durchlaufen aller Tests wird die Software auf ebendiesen Modulen laufen und zum Verkauf freigegeben werden.

[3] Mit Diagnosefunktionen ist die Überprüfung der Funktionalität des Systems durch Überwachung bestimmter Werte auf das Überschreiten von Schwelllwerten sowie das Prüfen von Hardwarekomponenten auf Funktionstüchtigkeit gemeint

2 Mocha Analog

3 Der Analog- Digital- Wandler

Im ersten Teil der Arbeit geht es um die digitale Messung analoger Größen des Moduls. Das Framework wurde am Beispiel der Temperaturmessung des neuen Moduls entwickelt, demnach soll dieser Anwendungsfall hier genauer beschrieben werden.

Das Messen analoger Größen ist einer der häufigsten Anwendungsfälle eingebetteter Systeme. Die meisten Größen unseres Umfeldes sind analoger Natur: Stimmen, Bilder, Druck und eben Temperatur. Die Verarbeitung analoger Signale ist schwierig, kostspielig und störanfällig. Deswegen werden Analog- Digital Wandler eingesetzt, um robuste und performante Anwendungen zu gewährleisten.

Zunächst sei hier gesagt, dass mithilfe eines ADC nicht nur die Temperatur gemessen werden kann. Aufgabe des Bauteils ist es, eine analoge Spannung in einen digitalen Wert zu formen. Dazu wird über eine über ein auf externe Reize reagierendes Bauteil abfallende Spannung an die Eingänge des Bauteils geschalten. Diese Spannung kann aber auch von einem beliebigen Sensor stammen. Abhängig davon, ob dieser Sensor oder externe Bauteil auf Temperatur oder eben andere Reize reagiert, kann der ADC verschiedene Größen messen.

Grob gesagt misst ein ADC die anliegenede Messspannung im Verhältnis zu einer anliegenden Referenzsspannung. Das Bauteil bildet danach das Verhältnis von anliegender Messspannung zu Referenzsspannung. Daraus wird ein einheitenloser Wert gebildet (siehe auch Gleichung 3.1, die Einheit kürzt sich).

Das Verhältnis diesen Wertes zum maximal messbaren Wert des Bauteils entspricht dem Verhältnis der beiden vorhin genannten Spannungen (siehe Gleichung 3.1, dort entspricht „Digitaler Wert“dem digitalisierten, analogen Wert und „Auflösung des Bauteils“dem maximal messbarem Wert).

$$\frac{Messpannung}{Referenzspannung} = \frac{DigitalerWert}{AufloesungdesBauteils} \tag{3.1}$$

Aufgabe eines Software Entwicklers ist es nun, diesen Wert mit der größten Spannung, die über den Sensor abfällt, zu verrechnen und aus dieser neuen Spannung eine geeignete

Größe zu berechnen. Im Anwendungsfall dieser Arbeit die Temperatur.

Im Späteren wird gezeigt, wie die aktuelle Temperaturmessung durch austauschen von Submodulen zur Messung einer anderen analogen Größe- nämlich der Betriebsspannung- genutzt werden kann.

3.1 Hardware

3.1.1 Der externe AD- Wandler: ADS124S08

Auf der Platine des Analog- Moduls ist ein serieller 24- Bit Delta- Sigma Wandler verbaut (wie im Datenblatt von Texas Instruments und Incorporated [2, S. 1] beschrieben). Dank der rausch- sowie driftarmen Architektur eignet sich der **ADS124S08** bessonders für Niederspannungssensoren wie einen Temperatursensor.(Texas Instruments und Incorporated [2, S. 29] beschrieben)
Hauptanwendungsbereich dieses Wandlers ist die Prozessor- als auch industrielle Steuerung.

Der Hauptvorteil von Delta-Sigma-ADCs gegenüber ADCs mit Nyquist-Rate besteht darin, dass sie keine hochpräzisen Analogkomponenten benötigen. Diese Anforderung wird gelockert, da eine hohe Auflösung durch Überabtastung erreicht wird. (nach Kasem Khalil [1, S. 6])

Diese ADC- Topologie biete die höchste Auflösung für Signale niederer Frequenzen. Da er Signale meist mit einer wesentlich höheren Frequenz (Modulatorfrequenz) abtastet, als das Nyquist- Kriterium zum vermeiden von Aliasing verlangt, besitzt er zeitgleich einen digitalen Filter am Wandlerausgang, der das Ausgangsrauschen des Signals verringert. (nach Rich Miron [9], Sigma-Delta-Wandler)
Das Verhältnis zwischen Modulatorfrequenz und Ausgangsdatenrate wird als Überabtastverhältnis (OSR[4]) bezeichnet. (siehe Texas Instruments und Incorporated [2, S. 24])

Durch Verringern der Ausgangsdatenrate und damit erhöhen des OSR kann das Rauschverhalten des ADC optimiert werden, da dann mehr Abtastwerte des internen Modulators

[4] Überabtast- verhältnis, engl. oversampling ratio, Verhältnis von Modulatorfrequenz ind Ausgangsdatenrate

gemittelt werden. (nach Texas Instruments und Incorporated [2, S. 24] Deswegen ist die **Ausgangsdatenrate** ein konfigurierbarer Wert des in diesem Framwork.
Der ADS124S08 bietet Ausgangsdatenraten im Bereich von 2.5 SPS[5] bis 4000 SPS.

Zusätzlich besitzt der AD- Wandler nach dem Eingangsmultiplexer einen rauscharmen, programmierbaren Verstärker (PGA[6]). Dadurch wird ein externer Verstärker, der zum messen kleiner Signale notwendig wäre, unnötig. Die PGA- Verstärkung ist in Binärschritten von 1 bis 128 programmierbar. Der PGA überwacht das Wandlungsergebnisses über Ausgangsspannungsmonitore. (Nach Texas Instruments und Incorporated [2, S. 29])
In diesem Framework muss der Verstärkungsfaktor des **PGA** angegeben werden.

Um eine mögliche Sensorfehlfunktion zu erkennen, bietet der ADS124S08 wählbare Stromquellen, die als Burn-Out-Stromquellen (BOCS[7]) fungieren. Wenn aktiviert, liefert eine dieser Stromquellen Strom an den ausgewählten positiven Analogeingang und die andere Stromquelle nimmt Strom vom ausgewählten negativen Analogeingang ab. Ein Offener Stromkreis führt zur Vollskalenanzeige. Messwerte mit vollem Skalenwert können auch bedeuten, dass der Sensor überlastet ist oder die Referenzspannung fehlt. Ein Messwert nahe Null kann auf einen kurzgeschlossenen Sensor hinweisen.
Da die Messwerte des ADCs verfälscht werden können, wenn die Burnout- Stromquellen aktiviert sind, sollten diese nur zu Beginn eines Burnouttests aktiviert werden und nach dem Durchführen des Tests direkt wieder deaktiviert werden. (nach Texas Instruments und Incorporated [2, S. 52])
In diesem Framework wird in regelmäßigen Abständen ein **Burnouttest** durchgeführt.

Der AD- Wandler verfügt zudem über eine serielle SPI[8]- Schnittstelle, über welche die digitalisierten Werte eingelesen werden und der ADC konfiguriert werden kann. Die gelesenen Daten werden zusätzlich mit einem CRC Code kodiert, um eine höhere Datenintegrität zu gewährleisten. In diesem Framework wird der AD- Wandler über sein **SPI- Interface** angesprochen.

[5] Abtastungen in einer Sekunde, engl. samples per second, beschreibt die Datenrate eines AD- Wandlers
[6] Verstärker mit programmierbarer Verstärkung, engl. Programmable Gain Amplifier, Operationsverstärker, der als nicht invertierter Verstärker beschaltet ist und einen programmierbaren Analogmultiplexer verbaut hat
[7] Burn-Out-Stromquellen, engl. Burn Out Current Source, Stromquellen für den Sensor Test
[8] engl. Serial Peripheral Interface, Bus-System „lockeren“ Standards für synchrone serielle Datenbusse (Synchronous Serial Ports) nach dem Master-Slave-Prinzip

Drei- Leiter Messung

Temperaturfühler haben in der Regel lange, mit dem Messgerät verbundene, Anschlusskabel. Diese sollen die Spannung zum Sensor und wieder zurück ans Messgerät leiten. Sie dienen also nur zur Spannungsleitung.

Da in der Praxis ein Temperaturfühler meist ein temperaturempfindlicher Widerstand ist (wie in Abbildung 3.1 dargestellt), besteht die Temperaturmessung also aus dem berechnen der über den Widerstand abfallenden Spannung in eine Temperatur. Haben die Anschlussbkabel zwar in der Theorie keinen eigenen Widerstand, so lässt sich in der Praxis dennoch ein kleiner Widerstand messen. Dieser Widerstand nennt sich Leitungswiderstand.

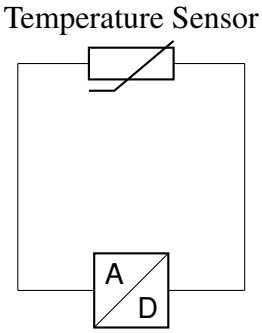

Abbildung 3.1: Schematiche Darstellung eines Temperaturfühlers an einem AD- Wandler

Da nun die „Messspannung“nun nicht mehr nur über den Temperaturfühler, sondern auch über die Anschlusskabel abfällt, kann das Messergebnis verfälscht werden: Durch diese leichte Erhöhung des gesamten Widerstands wird die Software als Beispiel statt tatsächlichen 80 Grad Celsius 85 Grad messen.

Um diese Problematik zu lösen, wird die Drei- Leiter Messung benutzt. Dabei wird ein dritter Leiter an Messwiderstand und Messgerät angeschlossen. Mithilfe dieses zusätzlichen Leiters wird ein zweiter Messkreis erstellt. (siehe Abbildung 3.2) Dieser Messkreis dient zur Ermittlung des Leitungswiderstands. Wichtig dabei ist allerdings, dass der dritte Leiter unbedingt die gleichen elektrischen Charakteristika aufweißt als die beiden anderen Leiter.
Damit kann über den dritten Leitungswiderstand ein Korrekturfaktor ermessen werden, mit dem ein korrekter Messwert berechnet werden kann. (nach Interview: *Hardware im*

Projekt [8])

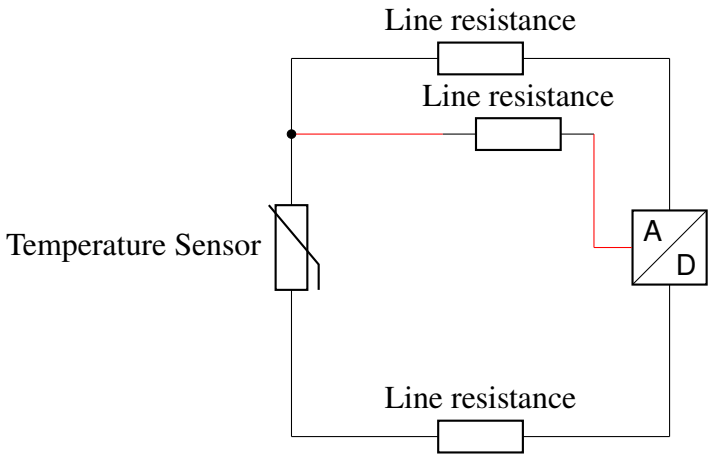

Abbildung 3.2: Schematiche Darstellung der Drei- Leiter Messung mit eingezeichneten Leitungswiderständen

3.1.2 Der interne AD- Wandler des STM32G4

Der in den Microcontrollern der STM32G4- Serie intern verbaute ADC basiert auf dem Verfahren der sukzessiven Approximation. (siehe STMicroelectronics [3, S. 3]) Bei dieser Topologie wird die anliegende Eingangsspannung mit Bruchteilen der Referenzsspannung verglichen.

Das Vorgehen ist iterativ: Es wird ein Wort der halben Referenzspannung erzeugt. Danach wird entschieden, ob die Eingangsspannug größer oder kleiner als das „Referenzwort“ist. Ist es zum Beispiel kleiner, wird das Referenzwort auf ein Viertel der Referenzsspannung gesetzt. Ist es wieder kleiner, wird das Referenzswort auf ein Achtel gesetzt. Dabei der Nenner immer die nächst Höhere ZweierPotenz (2^n). Bis zur maximalen Auflösung (Dieser ADC hat die Auflösung 12 Bit, in diesem Fall also 2^{16}).
Der Zäler des Bruchs beschreibt dabei immer das Mittel der beiden vorigen Werte: Is die anliegende Eingangsspannung zum Beispiel kleiner als die Halbe Referenzspannung aber Größer als ein Viertel, so ist das nächste „Referenzwort“drei Achtel. Der AD- Wandler erreicht dabei eine Genauigkeit von 0,5 LSB[9].(siehe STMicroelectronics [3, S. 3]).

[9] Bit mit der geringstejn Bitwertigkeit, engl. Least Significant Bit

Auch der intern im STM32G4 verbaute AD- Wandler kann mit verschiedenen Parametern wie die Wandlungszeit in Takten konfiguriert werden.
Zwar ist diese in diesem Framework ebenfalls konfigurierbar, im Kontext dieses Projektes ist im Allgemeinen aber nur der Eingangskanal des ADC relevant. Davon hat dieser 42 (siehe STMicroelectronics [3, S. 2]). Im Anwendungsfall des Analog- Hubs wurden davon zwei benutzt: Ein Kanal, der mit einem internen Temperatursensor in der CPU verbunden ist und die interne Betriebstemperatur misst und einer zum Überwachen der Versorgungsspannung des Moduls.

3.2 Software

In dem Framework soll der AD- Wandler als einzelnes Objekt dargestellt werden. Zusätzlich soll ein einheitliches Interface zur Verfügung gestellt werden, über das der ADC angesteuert werden kann. Um das zu Erreichen, müssen die Kernprozesse zum Messen von Spannungen mithilfe eines ADC identifiziert werden. Diese sind in Abbildung 3.3 zu sehen. Einen Überblick der gesamten Klassenstruktur ist in Unterabschnitt C.3 zu finden.

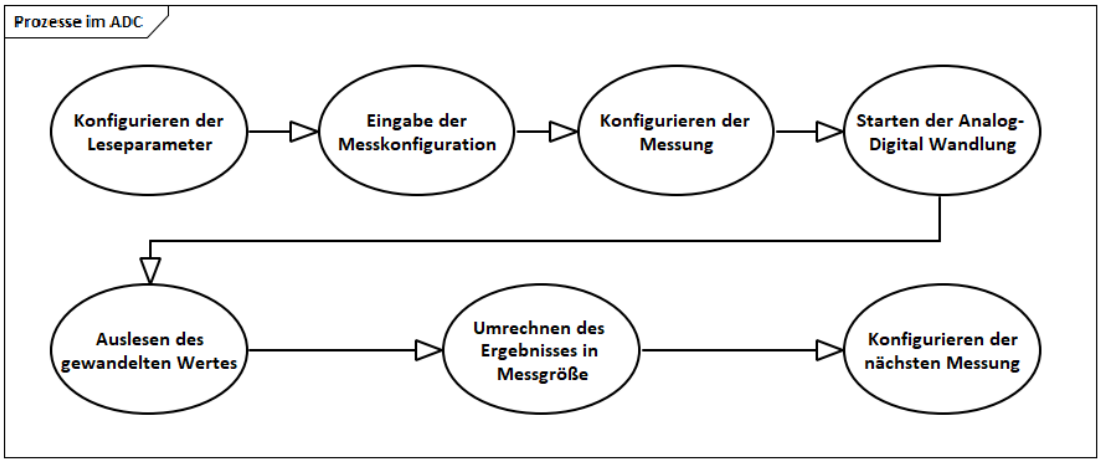

Abbildung 3.3: Ablaufende Prozesse in einem AD- Wandler

Ziel des Frameworks ist allerdings, die Nutzung des Objektes einfach und intuitiv zu gestalten. Demnach sollen nur die Prozesse dem Nutzer offenbart werden, die individuelle Konfiguration benötigen. Technische Prozesse sollen weitesgehend verborgen bleiben. Demnach sind nur folgende Aufgaben sichtbar:

- Konfigurieren der wichtigsten Daten wie Datenrate
- Übergeben der Messkonfiguration
- Starten des AD- Wandlers
- Abrufen der gemessenen Daten.

Starten der einzelnen Messungen oder das Umrechnen, Verarbeiten und Testen von Daten geschieht autonom und wird bei Fehlern durch Diagnosewarnungen an andere interne Komponenten im Framework weitergegeben. Der Nutzer des Frameworks muss sich keine Gedanken machen, wann Messungen gestartet werden und wann die Ergebnisse verfügbar sind. Die Daten werden beim internen als auch beim externen ADC unterschiedlich im Hintergrund aktualisiert (Dazu mehr in den jeweiligen Kapiteln Unterunterabschnitt 3.2.1 und Unterunterabschnitt 3.2.2).

Die letztendliche Umrechung soll in Datenumrechnungsklassen stattfinden. Diese werden in Form eines sogenannten Converters dem ADC übergeben. Wird also zu einem späteren Moment vom Master das Signal gegeben, dass über den Eingang eine andere Größe gemessen wird, kann der Converter ausgetauscht und die Konfiguration geändert werden, ohne einen neuen ADC zu erstellen oder neue Firmware[10] zu schreiben.

Ein Problem bei der Ansteuerung zwei stark unterschiedlicher ADC ist die Messkonfiguration. Der in dem Modul verbaute *STM32G473* besitzt einen integrierten AD-Wandler, der intern die von STM bereitgestellte *HAL- Bibliothek* benutzt. Da dieser nur für das Messen der internen Spannungsversorgung und der CPU- Temperatur benutzt werden sollte, muss hier kein großer Wert auf die Konfiguration der Messungen gelegt werden. Es reicht, lediglich den zu messenden Kanal und den „Converter“(Dazu mehr in Unterunterabschnitt 3.2.3) für die Umrechnung der gemessenen Spannung in eine reale Messgröße anzugeben.

Der extern auf dem Modul verbaute ADC hingegen ist für weit vielseitigere Anwendungsbereiche ausgelegt. Auch muss er weitaus genauere Ergebnisse liefern, da er die

[10] Firmware beschreibt Software, die in elektronische Geräte fest implementiert ist. Sie ist im Gegensatz zu herkömmlicher Software mit der Hardware unzertrennlich verankert. Die Firmware wird auf einem Flash-Speicher, (z.B. einen EEPROM) gespeichert und ist vom Benutzer nicht ohne Weiteres auswechselbar. (Nach it-service.network [7]

Hauptaufgabe des Produktes übernimmt: Das Messen der an den Ports angelegten Spannungen. Dafür sind demnach auch mehr Parameter für die Messkonfiguration notwendig: Das wären zum einen zwei an dem ADC verbaute Eingänge, über die die Spannung gemessen werden soll. Die über diese beiden Bezugspunkte abfallende Spannung wird anschließend in die Messgröße umgerechnet.
Zusätlich kann an dem Bauteil mehrere Referenzspannungen angelegt werden. Somit kann eine von mehreren Referenzsspannungen ausgewählt werden. Auch muss die Konvertierungszeit, im Datenblatt von Texas Instruments auch als Datentrate beschrieben, für verschiedene Messarten eingestellt werden. In Unterunterabschnitt 3.2.1 wird genauer auf die zu beachtenden Messarten eingegangen. Zuletzt muss auch das sogenante *Gain*, der Verstärkungsfaktor des im ADS124S08 verbaute Operationsverstärkers eingestellt werden. Dieser ist notwendig, da das Produkt im Betrieb in der Regel sehr kleine Spannungen messen muss.

Da der externe ADC allerdings zusätzlich zu den oben genannten Eigenschaften auch die Konfigurationsparameter des internen ADC benötigt, kann man diese Konfigurationsstruktur weiterverwenden. Da im ADC Interface ein gemeinsamer Datentyp als Parameter für die Konfigurationsmethode benötigt wird, kann man die gemeinsame Konfigurationsstruktur als eine Art *Interface* nutzen. Somit kann die Struktur des externen ADC, aber auch gegebenenfalls später folgende Strukturen, unter diesem Interface als einheitlicher Datentyp zusammengefasst werden. (Siehe in Listing 3 Unterabschnitt D.1)

Beachtet man alle oben genannten Kriterien, dürfte das Interface letztendlich wie in Abbildung 3.4 aussehen. Es beinhaltet einen Timer, da der ADC gemäß des *Time Pattern* agieren soll. Das heißt, dass nicht eine Messung gestartet wird und anschließend auf ein Ergebnis gewartet, sondern im Hintergrund aktuelle Daten zur Verfügung stehen. Diese können dann mit der Methode *getData* angefordert werden. Eine genaue Erläuterung dieses Pattern ist beim externen ADC zu finden.

Das Attribut *amoutChannels* dient nur zum Speichern der Länge des Messvektors, damit soll etwas Performanz gespart werden, da diese an vielen Stellen der Klasse benötigt wird. Im *data* Attribut wird das aktuelle Messergebnis zwischengespeichert. Das Attribut *sampleTime* beinhaltet die konfigurierte Wandlungszeit und in *MEF_TimerMessage* wird die Nachricht für das Aufsetzen eines neuen Timers abgespeichert. Da der Timer nicht

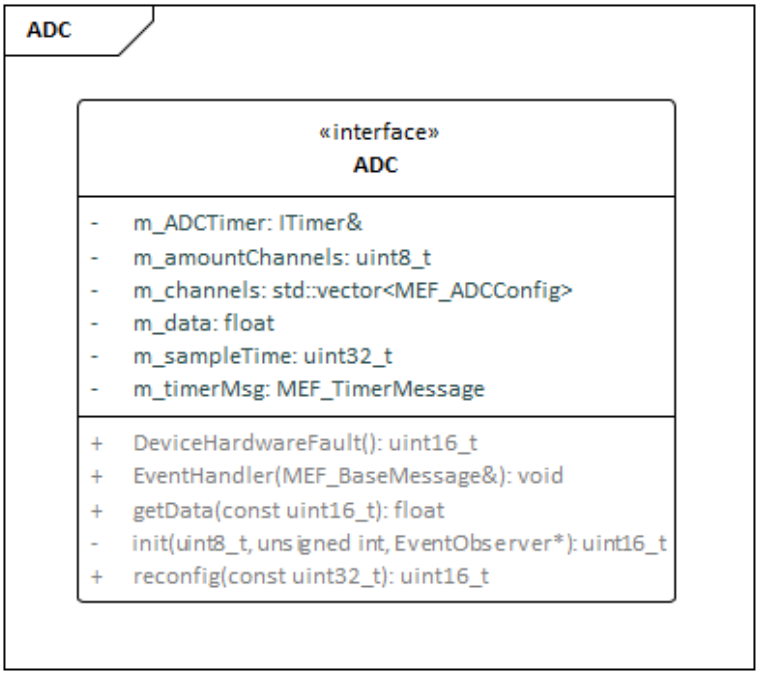

Abbildung 3.4: Klassenstruktur des ADC Interfaces

Umfang dieser Arbeit ist, wird er in Unterunterabschnitt 3.2.1 bei der Implentierung des Timepatterns nur kurz umschnitten.

Die Methode *DeviceHardwareFault* ermittelt die korrekte Arbeitsweise des ADCs und wird in der gleichnamigen Diagnose ausgewertet (Siehe Abschnitt 4- Diagnosen). Der Eventhandler wird wegen des eventgesteuerten Timers benötigt, um nach Ablauf dessen und Auswerten der Nachricht das Messergebnis korrekt auszulesen und umzurechnen. Deutlich wird das in Unterunterabschnitt 3.2.1 bei Erläuterung der Implementierung des Time Patterns. Die Methoden *init* und *reconfig* dienen zur initialisierung und gegebenenfalls umkonfigurierung der ADC- Parameter.

3.2.1 Der externe AD- Wandler

Der externe ADC kapselt sich in drei Klassen, von denen eine abstrakt ist und zwei konkret. Eine der beiden konkreten Klassen fungiert dabei nur als Treiber des externen AD- Wandlers. Einschränkungen des im Folgenden dargestellten Treiber- Konzepts wird

in Unterunterabschnitt 3.2.4-Vergleich und Analyse einer zweiten Lösung aufgezeigt und diskutiert.

Um ein Objekt für den externen ADC zu erzeugen, muss man einen Sensorobjekt erstellen. Dieser Sensor verbildlicht das Messen der Größe. Somit muss der Nutzer nicht wissen, wie analoge Größen gemessen werden, sondern lediglich dass der Sensor mit gewissen Messparametern ein Ergebnis produziert, dass weiterverwendet werden kann. Ein Sensor misst eine Größe- also wird auch ein Sensor erzeugt.

Intern startet der Sensor (In diesem Beispiel ist es der RTD[11]- Sensor zum Messen einer Temperatur) den ADC und konfiguriert diesen. Die Messwertbereitstellung geschieht über das Timepattern, welches anhand des Programmcodes in dem Kapitel RTDSensor-Klasse genauer beschrieben wird.

Im Folgenden werden der Aufbau und die Funktion der einzelnen Klassen genauer beschrieben:

ExternalADC- Klasse

Der externe ADC dient zur Kapselung der Funktionalität, die alle externen ADCs zusammen haben: Die Kommunikation zwischen Wandler und der Firmware, die auf der CPU abläuft und technische Konstanten, die für das Setzen und auslesen einzelner Bits in den Registern ADCs benötigt werden (Jedes Register beinhaltet 8 Bits). EIn Überblick dieser Funktionen ist in Abbildung 3.5 zu sehen.

Im Konstruktor wird der Klasse ein Objekt des Typs „Connector“übergeben. Der Connector ist dabei kein spezifisches Objekt, sondern ein Interface für beliebige Kommunikationsprotokolle. In diesem Framework stehen dazu der *SPI- Bus* und der IOLink-Stack für die Kommunikation zur Verfügung. Der IOLink- Stack wird allerdings nur zur Kommunikation des Moduls mit dem Master genutzt. Intern können keine Bauteile mit dem IOLink Stack angesprochen werden. Auf der Hardware wird das SPI- Protokoll zur Bauteiladressierung genutzt. Dementsprechend wird in der Hauptapplikation des Projekts dem ADC ein Connector des Typs SPI übergeben.
Zusätzlich zu diesem Kommunikationsmodul wird eine sogenannte „Device ID“benötigt.

[11] Messung der Temperatur mit einem Widerstand, engl. Resistance Temperature Detector

Abbildung 3.5: Klassenstruktur des abstrakten Klasse des externen ADC

Das ist eine einmalig vergebene Zahl, mit der ein Bauteil direkt identifiziert werden kann.

Der Konnektor wird in den Methoden *sendCommand* und *receiveData* benötigt. Diese sollen die Kommunikation eines externen AD- Wandlers mit der CPU[12] kapseln. Dabei wird in den Methoden jeweils eine vom Framework definierte Nachricht (*MEF_SPIMessage*) gebaut. In diese wird die Device ID hinzugefügt. Zusätzlich wird bei der Methode sendCommand der Befehl mit seiner Speichergröße (Command- Parameter) oder bei der Methode receiveData Speicherplatz für die gelesenen Daten zugewiesen. Bei dem Empfangen der Daten muss zuätzlich die Antwort des AD- Wandlers umgeformt werden. Dieser sendet den digitalen Wert in einem Byte- Array. Mit dem Algorithmus aus Listing 4 wird das Byte- Array in eine *uint32_t* Variable konvertiert.

[12] zentrale Recheneinheit, engl. Central Processing Unit, eines Rechners oder Mikrocontrollers. Umgangssprachlich oft als Prozessor bezeichnet.

Weiterhin instanziiert die Klasse ein Timerobjekt des Typs ITimer. Dieses wird für das bereits erwähnte Time- Pattern in der Klase RTD- Sensor benötigt. Da in der Regel allerdings alle externen AD- Wandler dieses Pattern nutzen werden, kann das Timerobjekt bereits hier gespeichert werden. Implementiert werden kann das Pattern allerdings noch nicht, da dazu ebenfalls Zugriff auf die spezifischen Messkonfigurationen benötigt wird. Da nicht jeder externe ADC zwingend das gleiche Konfigurationsobjekt nutzen wird, steht dieses nur in der Klasse RTDSensor zur Verfügung.

Auch ITimer ist ein Interface. Timer sind in der Regel auf der CPU verbaut. Daher können diese beim Wechsel auf eine neue CPU anders implementiert werden müssen. Im aktuellen Projekt stehen die Timer des Prozessors STM32G473 sowie Softwaretimer zur Verfügung. Softwaretimer sind dabei lediglich schnelle ein schneller Timer, auf den sich mehere Objekte einschreiben können. Diese werden nur alle n Takte benachricht (Wenn ein Timer zum Beispiel jede Sekunde abläuft, aber ein Objekt nur alle fünf Sekunden benachrichtigt werden muss, wird dieses nur jedes fünfte Mal benachrichtigt).

ADS124S08- Klasse

Die ADS124S08- Klasse ist zwischen dem externen ADC und dem RTDSensor platziert und stellt sämtliche Hardwarefunktionen zur Verfügung, die für die Benutzung des AD-Wandlers benötigt werden. Zusätzlich enthält dieses Klasse die aktuelle Konfiguration des Wandlers. Die Komplette Klassenstruktur ist in Abbildung 3.6 zu sehen.
Im Folgenden werden nur die für die Messung eines Wertes notwendigsten Werte eingegangen. Andere Methoden wurden im Nachgang von einem Hardwareentwickler des Unternehmens hinzugefügt.

Im Allgemeinen sind die Methoden zum Setzen und Lesen von Konfigurationsparametern sehr ähnlich: Zuerst wird eine Registeradresse benötigt. Diese bestimmt das Register, in dem der Konfigurationswert gesetzt wird.

Da meist nie das ganze Register neu gesetzt wird, sondern nur einzelne Bits darin geändert werden, muss zuerst der aktuelle Wert des Registers ausgelesen werden. Der gelesene Wert wird mit einer Bitmaske bitweise verundet, um einzelne Bits zu extrahiern. Diese können danach mit einem bitweise Oder gesetzt werden.
Ist somit der neue Registerwert gebaut, kann dieser erneut in das Register geschrieben

werden.

Abbildung 3.6: Klassenstruktur des ADC- Treibers

Um das ganze nochmals zu verdeutlichen, wird der Code am Beispiel der Methode *pgaGain(ADS124S08PGAGain gain)* durchgesprochen. Der Code für diese Methode ist in Listing 1 zu sehen.

Bei dem Parameter gain handelt es sich dabei um ein im Framework deklariertes Enum.

Der intern verbaute Operationsverstärker ist zwar konfiguerierbar, lässt allerdings nur die Werte 1, 2, 4, 8, 16, 32, 64 und 128 zu. Deshalb wurde für den Nutzer die Eingabe eines Verstärkungsfaktors durch dieses Enum auf ebendiese Werte beschränkt.

```
1 uint32_t ADS124S08::pgaGain(ADS124S08PGAGain gain) {
2   uint8_t pgaRegTemp = m_pgaReg;
3   pgaRegTemp &= ~(BIT_2_MASK | BIT_1_MASK | BIT_0_MASK);
4   pgaRegTemp |= (uint8_t)gain;
5
6   uint32_t error = writeReg((uint8_t)ADS124S08RegAddr::PGA_ADDR,
      pgaRegTemp);
7
8   if(!error) {
9     m_pgaReg = pgaRegTemp;
10  }
11
12  return error;
13 }
```

Listing 1: Neukonfigurieren des intern verbauten Operationsverstärkers

Da Treiber nicht nur die Hardwarefunktionen für die Bedienung des ADC breitstellt, sondern auch diesen auf Hardwareebene repräsentiert, müssen die jeweiligen Registerwerte erst ausgelesen werden. Stattdessen wird der Inhalt der Register in verschiedenen Attributen abgespeichert. Damit kann die Zeit, die zum auslesend er Daten über SPI benötigt wird, gespart wird.

Die oben genannten Verstärkundsfaktoren sind von null aufsteigend numerisch kodiert. Die Kodierung für den Verstärkungsfaktor 64 ist beispielsweise 110. Da es acht verschiedene gibt, werden drei Bit für diese Kodierung benötigt. Diese drei Bit befinden sich in den untersten drei des Registers. Um diese zu Maskieren, werden in Codezeile drei die Bitmasken der Bits eins, zwei und drei bitweise verodert: Das Ergebnis ist **00000111**. Wird diese Maske mithilfe der Tilde bitweise invertiert, ergibt sich **11111000**. Eine bitweise Verundung mit dem Registerwert setzt somit alle Stellen, die den Verstärkungsfaktor definieren, auf Null. Das Ergebnis dieser Operation ist **xxxxx000**.

Wird nun der ganzzahlige Wert des mitgegebenen Verstärkungsparameter (der durch einfaches Casten aus dem Enum erhalten werden kann) mit dem neuen bitweise Verundet,

erhält man den neuen Wert des Registers. Bei einem Verstärkungsfaktor von 64 wäre das **xxxxx110**.

Dieser Wert kann nun in das Register geschrieben werden (siehe Codezeile 6). Ist der resultierende Fehlercode *MEF_ERROR_NO_MALFUNCTION*, intern kodiert als 0x0000, wurde der Wert erfolgreich in das Register geschrieben. Das Klassenattribut für dieses Register kann nun durch den neuen Wert ersetzt werden (Siehe Codezeile 9).

RTDSensor- Klasse

Die Hauptaufgabe der RTD- Sensors ist das Managen der Messungen, die über den ADS124S08 durchgeführt werden. Namensgebend für die Klasse ist der gewünschte Messvorgang: Das Messen der Temperatur über RTD- Sensoren. Eine Übersicht der Klasse ist in Abbildung 3.7 zu finden.

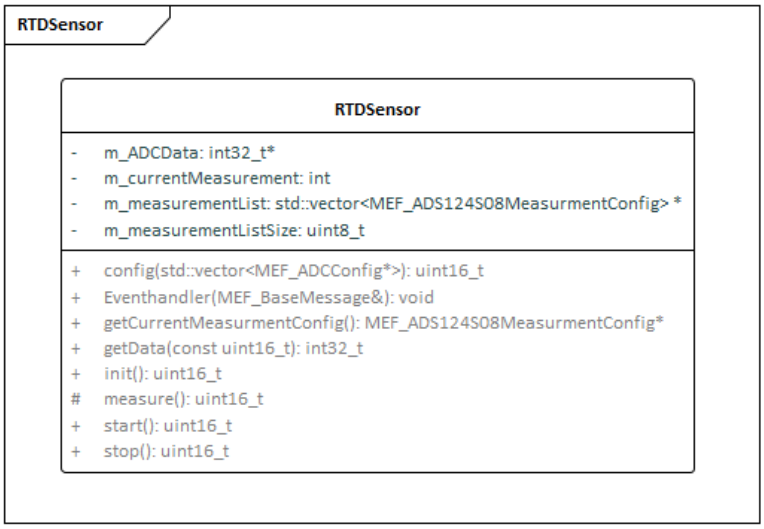

Abbildung 3.7: Klassenstruktur des RTD- Sensors

Die Klasse bietet die Möglichkeit, die Messungen zu starten oder zu stoppen. Ist der Messvorgang erstmal gestartet, werden im Hintergrund die Daten nach erfolgreicher Umwandlung des Bauteils aktualisiert. Wird in der Software also ein Wert gebraucht, kann dieser direkt aus der Klasse extrahiert werden.
Zusätzlich kann der RTD- Sensor initialisiert werden, indem eine Testmessung stattfindet. Wichtig jedoch sind die *config*- Methoden. Ihnen wird ein Vektor mit Messkonfigurati-

onsstrukturen (wie in Unterabschnitt 3.2 bereits besprochen) übergeben. Diese Methode ersetzt die alte Reihenfolge der Messungen mit der ihr Übergebenen. Das ganze kann dynamisch zur Laufzeit geschehen. So kann auf Nachrichten des IOLink Masters reagiert werden und Messungen entfernt, geändert oder hinzugefügt werden.

Kern der Klasse ist der Eventhandler. In ihm wird das Time Pattern implementiert. Dadurch wird erreicht, dass nur Rechenzeit benötigt wird, um ein Messergebnis zu bearbeiten und eine neue Messung zu starten. Ein Überblick über die Abläufe des Eventhandlers liefert Abbildung 3.8.

Der erste dort abgebildetete Schritt, das Initialisieren des AD- Wandlers, führt der Nutzer noch selbst durch. Der folgende Teil wird läuft im Hintergrund ab.

Zuerst wird der Timer gestoppt. Danach wird dieser mit einer neuen Zeit initialisiert. Diese ergibt sich aus der eingestellten Wandlungszeit aus der Messkonfiguration (siehe Texas Instruments und Incorporated [2, S. 42]).
Anschließend wird dem Timer der Eventhandler selbst mitgegeben. Das ist nur das erste mal nach dem Starten der Firmware notwendig.

Ist der Timer soweit konfiguriert, kann dem AD- Wandler der Befehl zum starten der neuen Wandlung gegeben werden. Dazu wird eine weitere Methode aufgerufen: In ihr werden all die Attribute aus der aktuellen Messkonfiguration in die Register des ADCs geschrieben.
Anschließend wird der STOP- Befehl gesendet, um den Wandler zu pausieren und folgend wieder mit dem START- Befehl zu aktivieren. Nun wird die Wandlung mit der neuen Konfiguration durchgeführt.

Nach Aufruf dieser Methode bleibt im Eventhandler nur nach das Starten des oben konfiguerierten Timers. Dieser sollte erst nach dem Starten der Messung aktiviert werden, da damit Sichergestellt ist, dass er nach Beenden der Wandlung ein Interrupt startet.

Danach beendet sich der Eventhandler im ersten Durchlauf und überlässt die Rechenzeit der CPU anderen Teilen der Software. Erst beim Ablaufend der Timer wird der Eventhandler erneut aufgerufen.

Da nun bereits eine Messung stattgefunden hat, ist der erste Schritt nicht mehr das neu

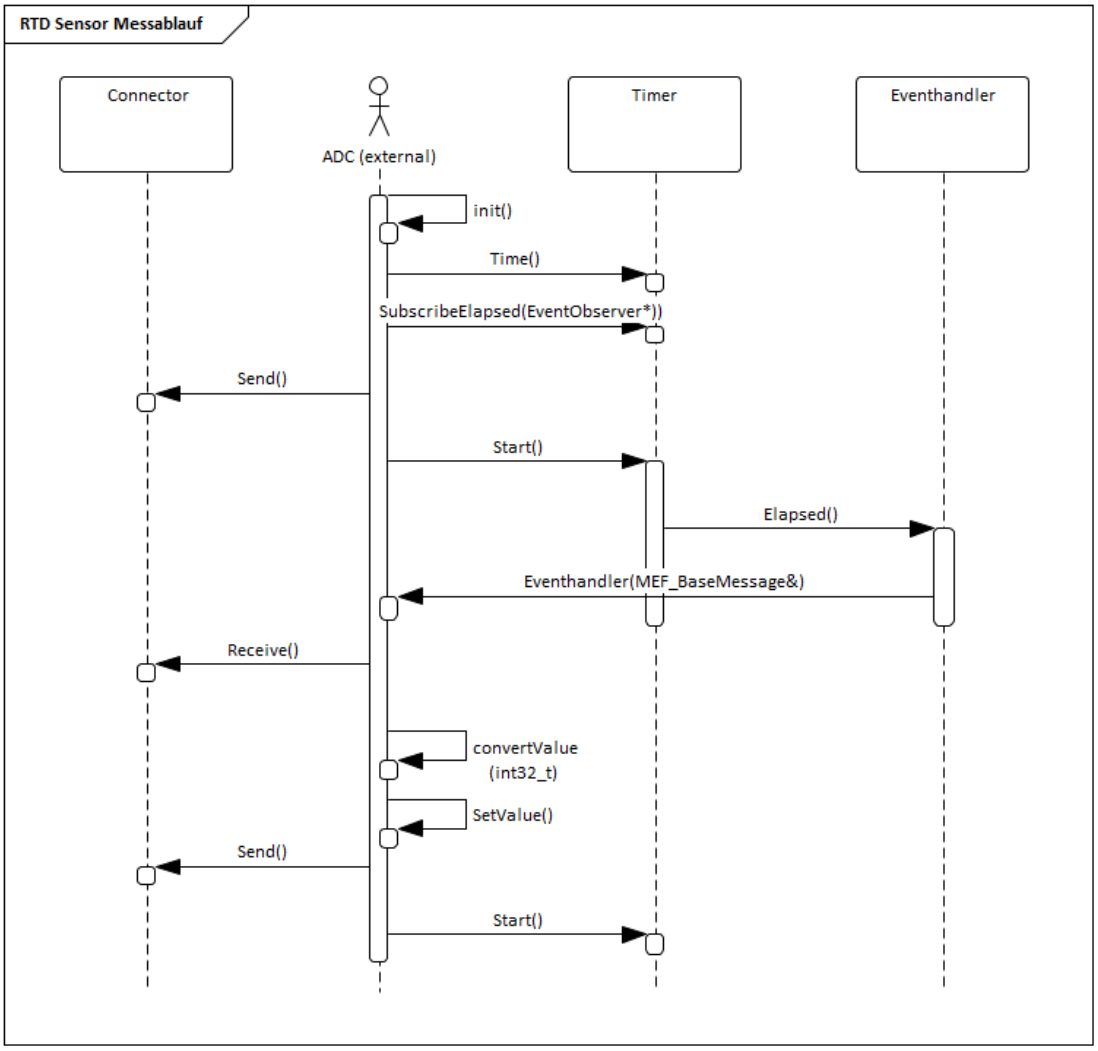

Abbildung 3.8: Ablaufdiagramm über das bereitstellen neuer Daten

konfigurieren der des Timers. An dieser Stelle wird das aktuell am ADC zwischengespeicherte Ergebnis abgerufen. Das passiert mit der ReceiveMethode, bei der zusätzlich über SPI nicht nur der READ- Befehl gesendet wird, sondern auch ein Array mit dem benötigten Speicherplatz, um das Messergebnis zwischen zu speichern. Um der hohen Auflösung des Bauteils gerecht zu werden, wird dazu ein Array aus drei vorzeichenlosen Acht- Bit- Integer- Werten benutzt.

Wie bereits erwähnt, wird dieser Wert direkt in ein 32- Bit Integer Wert gewandlet (siehe Listing 4 in Unterabschnitt D.2) und kann direkt weiterverarbeitet werden. Das

geschieht in den sogenannten Datenumrechnungsklassen in Unterunterabschnitt 3.2.3. Der Eventhandler nutzt nochmals die Konfigurationsstruktur der vorherigen Messung. Die dortige Umrechnungsklasse beinhaltet eine *convert(uint32_t)* Methode, die den Messwert in eine reale Messgröße umrechnet.

Ist dieser Wert nun umgerechnet, kann dieser in dem Array an seiner jeweiligen Position abgespeichert werden. Über die Methode getData kann dieser Wert nun abgerufen und außerhalb der Klasse weiterverwendet werden.

Nun werden die oben erläuterten Schritte erneut durchgeführt und somit die nächste Messung gestartet.

3.2.2 Der interne AD- Wandler

Der interne AD- Wandler ist direkt auf dem CPU verbaut. Im Umfang des Porojektes dient er zur internen Versorgungsspannung- sowie Referenzspannungsüberwachung. Das Messen der intern vorliegenden Referenzspannung dient dem genaueren Messen mit dem externen AD- Wandler: Wie so viele andere Größen unterliegt auch die Referenzspannung Schwankungen über der Zeit. Um somit den aktuellen Referenzwert möglichst genau zu haben, wird für die Referenz anstelle der idealerweise volriegenden Spannung, die nur auf ein Zentel Milivolt genau spezifiziert ist, ein deutlich genauerer Wert verwendet, der die aktuelle Abweichung retuschiert.

Er kommunizert direkt mit der CPU. Daher kann er diese auch über einen direkten Speicherbus: Den DMA[13]- Bus. Dieser sichert eine höhere Ausführungsgeschwindigkeit laufender Programme.

Zum einen muss der Prozessor nicht erst Daten in die eigenen internen Register einlesen, um sie direkt wieder in den Arbeitsspeicher zu schreiben, sondern die Daten über eine spezielle Datenleitung zwischen der Hardware senden. beispielsweise benötigt ein Speichertransfer über den Prozessor in der Regel ungefähr 40 Takte. Über DMA kann das innerhalb von vier Takten geschehen. (nach Elektronik Kompendium [6])

[13] Direkter Speicherzugriff, engl. Direct Memory Access, seperater Speicherbus mit Controller um Daten ohne Umwege über die CPU direkt in den Arbeitsspeicher schreiben zu können.

Zum anderen kann speziell im Anwendungsfall der Analog- Digital Umwandlung Zeit gespart werden. Der interne AD- Wandler kann mit den verschiedenen auszuführenden Messungen programmiert werden und in den sogenannten *Continious Mode* gesetzt werden.
Das heißt, er führt die einzelnen Messungen der Reihe nach immer wieder erneut durch, bis die Stromversorgung versiegt. Die Ergebnisse der Wandlung müssen auch nicht erst am Bauteil abgeholt werden, sondern stehen direkt im Arbeitsspeicher. So muss sich die Software nur einmal initial um den Wandler kümmmern und anschließend nur noch wenn die Messungen geändert werden sollen.

STM liefert zu den hauseigenen MCU[14]s eine Softwarebibliothek um diese anzusteuern: Die HAL- Bibliothek. Aktiviert man in dieser Bibliothek den internen AD- Wandler der MCU, stellt die HAL- Bibliothek ein *Handle*[15] zur Verfügung, über das der ADC konfiguriert und angesprochen werden kann.

Wird der Kanal gewechselt, muss dieser über das Handle umgeschrieben werden. Dafür ist die private Methode *configChannel* in Abbildung 3.9 zuständig.
Dort wird ein neues *ADC_ChannelConfTypeDef* - struct erstellt, dass unter anderem ein Attribut für den Kanal oder auch die Konvertierungszeit hat. Mit der von HAL zur Verfügung gestellte Funktion *HAL_ADC_ConfigChannel(adc_handle, &sConfig_struct)* kann die Konfiguration in das Handle geschrieben werden und somit einen neuen Kanal messen.

Dabei muss beim Messen mehrerer Kanäle allerdings nicht regelmäßig der Kanal geändert werden. Beim Verwenden von DMA sollte im Anwendungsfall mehrerer Kanäle der sogenannte *Multimode* akitiviert werden. Mithilfe verschiedener Rängen kann in den Konfigurationen sichergestellt werden, dass die Kanäle nicht überschireben werden. Ein Beispiel des Messens mehrerer Kanäle ist in Listing 5 in Unterabschnitt D.3 dargestellt.

Da bei der Initialisierung der Treiber der HAL- Bibliothek das ADC- Handle mit einem DMA Handle verknüpft wird [16], muss das DMA- handle nicht weiter zur Konfiguration an die ADC- Klasse weitergeleitet werden- alles relevante wird über das ADC- Handle

[14] Mikrocontroller, engl. Micor Controlling Unit, Einplatinenchips, die gleichzeitig einen Prozessor und Peripheriefunktionen enthalten
[15] Referenzwert einer Ressource, das durch ein bestimmtes System verwaltet wird
[16] Die Funktion lautet __HAL_LINKDMA(adcHandle, DMA_Handle, hdma_adc1)

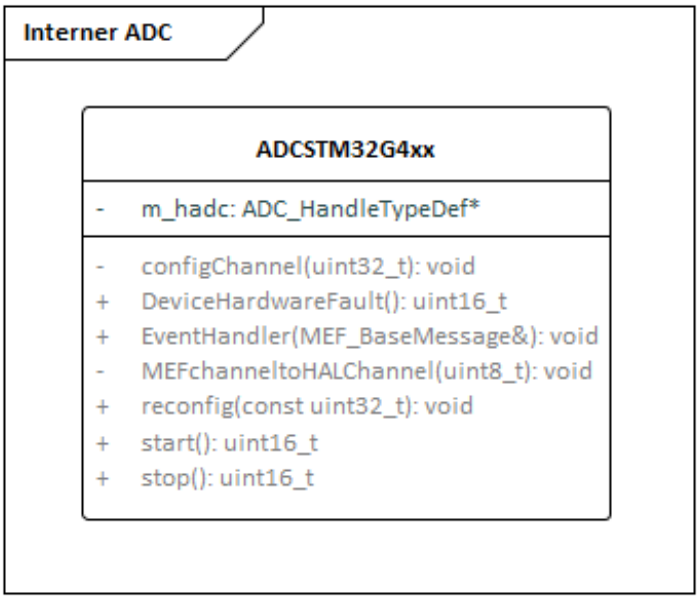

Abbildung 3.9: Klassendiagramm des internen ADC

konfiguriert unnd verknüpft.

Des weiteren beinhaltet die Klasse einen Methode *MEFchanneltoHALChannel*, welche Frameworkinterne Definitionen der ADC- Kanäle auf die HAL- internen Definitionen abbildet. Damit kann das Framework eigene Definitionen verwenden, die nicht so lang wie die der HAL- Bibliothek sind und anstatt den Namen der HAL- Bibliothek den des Frameworks enthalten.

Die weiteren Methoden der Klasse sind bereits aus obigen Erläuterungen bekannt und werden deswegen an dieser Stelle nicht mehr aufgeführt.

3.2.3 Datenumrechnungsklassen

Da in diesem Framework Wiederverwendbarkeit an wichtigster Stelle steht, müssen einzelne Bearbeitungsschritte austauschbar gestaltet werden. In diesem Zug wurde die Umrechnung modular für jede Messung konzipiert. Das bedeutet, dass sie auf Konfigurationsebene für jede Messung angegeben werden muss.

Dabei hat es nicht gereicht, eine Umrechnung als Enum- Value anzugeben, der intern in

der Klasse die Umrechnungsformel angibt.

Deswegen wurde diese Formeln in jeweils einer eigenen Klasse gekapselt. Das bringt den Vorteil, dass sie nicht nur an anderen Stellen der Firmware wiederverwendbar zur Verfügung stehen und der Nutzer diese gegebenenfalls in eigenen Applikationen benutzen kann, sondern auch dass im Notfall neue Umrechnungen geschrieben werden können, ohne Änderungen am ADC vornehmen zu müssen.
Dies entspricht ganz dem Prinzip der Softwareentwicklung **Seperation of Concerns**: Das Umrechnen eines Wertes hat nichts mit dem Umwandeln eines analogen Wertes in einen digitalen Wert zu tun- es ist lediglich im Use Case der Anwendung notwendig, den Wert aufzubereiten.

Der Grundaufbau dieser Umrechnungsbausteine ist ein Konstruktor, über den alle zur Umrechnung wichtigen Konstanten übergeben werden und die Methode zum umrechnen des Messertes (siehe Abbildung 3.10).

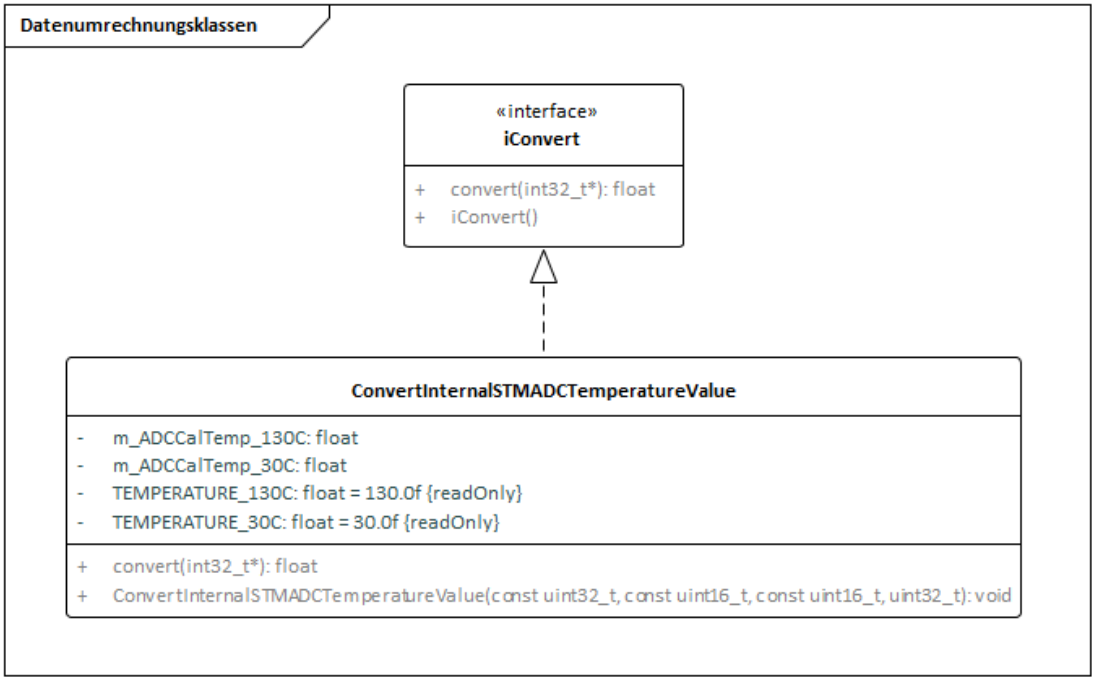

Abbildung 3.10: Beispielhafte Struktur einer Umrechnungsklasse mit dem Interface

Konstanten sollten sich innerhalb der gleichen Messkonfiguration nicht ändern. Dementsprechend bietet es sich an, diese über den Konstruktor an das Objekt zu übergeben.

Desweiteren hat es den Vorteil, dass der Konstruktor als einzige „Methode “nicht vom Interface erbt und somit auch nicht die gleiche Parameterstruktur im Funktionskopf beinhalten muss. Somit kann jeder Konvertierungsklasse ein individueller Satz an Parametern übergeben werden.
In Abbildung 3.10 wird beispielsweise die Referenzspannung des internen Ad- Wandlers als auch die bei der Produktion getesteten Kalibrierungswerte übergeben. Im Konstruktor werden daraufhin die beiden Konstanten m_ADCCalTemp_30 und m_ADCCalTemp_130 aus diesen Werten berechnet. Mit diesen beiden Konstanten ergibt sich darauf die Formel aus Listing 2.

```
1 (static_cast<float>(*data) - m_ADCCalTemp_30C)/(m_ADCCalTemp_130C -
    m_ADCCalTemp_130C) * (TEMPERATURE_130C - TEMPERATURE_30C) +
    TEMPERATURE_30C;
```

Listing 2: Formel zur Berechnung der Betriebstemperatur

Der Umrechnungsmethode wird ein Pointer des Typs uint32_t übergeben. Damit stellt das Interface sicher, dass Klassen, die zwei Werte zur Umrechnung benötigen, diese auch im Funktionskopf in Form eines Array annehmen können.

In Abbildung 3.10 wird das nicht benötigt. Allerdings kann es im Fall der Drei- Leiter Messung (siehe Unterunterabschnitt 3.1.1) ist dies wichtig.

Eine Anforderung an das Projekt war, zur Laufzeit dynamisch auf Änderungen zu reagieren. Auch das ist gewährleistet. Ändert sich zum Beispiel am externen AD- Wandler ein angeschlossener Sensor von beispielsweise einem PT100 (Platindraht mit Basiswiderstand von 100 Ohm) zu einem NI200 (Nickeldraht mit Basiswiderstand von 200 Ohm), so kann darauf reagiert werden, indem in der jeweiligen Messkonfigurationsstruktur zur Laufzeit das Datenumrechnungsobjekt mit dem des neuen Sensors ausgetauscht wird.

Weitere Umrechnungsklassen, die für die Anforderung des Projektes geschrieben wurden, umfassen das Konvertieren des Bitwertes in die Betriebsspannung des Moduls oder in die intern vorliegende Referenzspannung. Beide Umrechnungen funktionieren weitesgehend analog zu der vorgestellten Formel.

Weiter sind aber auch Umrechnungen für den externen AD- Wandler benötigt. Dieser muss den gemessenen Wert in einen Widerstand umrechnen. Der aktuelle Widerstand

ist abhängig von der Temperatur und repräsentiert somit ebendiese. Dieser Widerstand muss daraufhin auf den des Basissensors (PT100) herunter gerechnet werden und kann daraufhin erst in eine Temperatur gefomrt werden.
Da im Lastenheft des Produktes mehrere Sensoren aufgelistet sind, mit denen das Modul funktionieren soll, ist diese Klasse stark von dem ausgewählten Sensor abhängig. Ebenso ist die Klasse aber auch abhängig vom eingestellten Verstärkungsfaktor des integrierten Operationsverstärkers.
Weitere Parameter, die der Klasse im Konstruktor übergeben werden müssen ist die Art der Verdrahtung des Sensors.

In der Convert- Methode wird daraufhin folgend vom ersten gemessenen Widerstand (dem Gesamtwiderstand der Messchaltung) zweimal der zweite Widerstand (Leitungswiderstand, zweimal für Leitung vor- und nach dem Sensor) abgezogen. Daraus wird der Widerstand berechnet und abhängig vom Sensortyp die Temperatur berechnet. Die Formeln dazu sind den Datenblättern entnommen. Der Programmcode für diese Umrechnung ist zu umfassend, um in dieser Ausarbeitung als Listing aufgeführt zu werden.

3.2.4 Vergleich und Analyse einer weiteren Lösung

Während der Theoriephase des vierten Semesters wurde die Programmstruktur des AD- Wandlers von einem weiteren Softwareentwickler des Unternehmens während der Applikationsentwicklung für ein zweites Produkt überarbeitet. Im Zuge dessen wurde eine Analyse beider Problemlösungen durchgeführt, um aus beiden das Beste für das Projekt zu extrahieren und eine Lösung beziehungsweise eine Mischform beider Entwürfe für das Framework zu finden oder Schwächen beider Lösungen aufzudecken.

4 Diagnosen

5 Errorcodes

6 Fazit

6.1 Bewertung

6.2 Ausblick: Codegenerierung

Literatur

[1] Kasem Khalil, „Analog to Digital Converter Architecture,“ Jg. 2015, 2015.

[2] Texas Instruments und Incorporated, „ADS124S0x Low-Power, Low-Noise, Highly Integrated, 6- and 12-Channel, 4-kSPS, 24-Bit, Delta-Sigma ADC with PGA and Voltage Reference datasheet (Rev. C): Datenblatt,“ *Datenblatt*, Jg. 2016, 2016. Adresse: https://www.ti.com/lit/ds/symlink/ads124s08.pdf?ts=1655085524410 (besucht am 2017).

[3] STMicroelectronics, Hrsg., *AN5346: STM32G4 ADC use tips and recommendations*, 2019. Adresse: https://www.st.com/resource/en/application_note/an5346-stm32g4-adc-use-tips-and-recommendations-stmicroelectronics.pdf.

[4] IT-Glossar, *Framework*, 2021. Adresse: https://www.modernizing-applications.de/it-glossar/framework-definition/.

[5] *Zeit ist Geld*, 2022. Adresse: https://www.wortbedeutung.info/Zeit_ist_Geld/.

[6] Elektronik Kompendium, Hrsg., *DMA - Direct Memory Access*.

[7] it-service.network, it-service.network, Hrsg. Adresse: https://it-service.network/it-lexikon/firmware.

[8] *Hardware im Projekt*, Murrelektronik GmbH, 12. 01. 2022.

[9] Rich Miron, *Die passende ADC-Topologie für jede Anwendung finden: Bei Analog/Digital-Wandlern den Überblick behalten*, all-electronics, Hrsg., 1. Aug. 2018. Adresse: https://www.all-electronics.de/elektronik-entwicklung/adc-topologien-111.html.

A Werkzeug- Liste

- Software
  - Visual Studio (Entwicklungsumgebung):
    https://visualstudio.microsoft.com/de/vs/
  - CubeMonitor (Software- Controlling):
    https://www.st.com/en/development-tools/stm32cubemonitor.html
  - PCANVIEW (CAN- Überwachung):
    https://www.peak-system.com/PCAN-View.242.0.html
- Hardware
  - Peak System CAN Dongle (Dongle USB to CAN):
    https://www.peak-system.com/PCAN-USB.199.0.html
  - STM32 G473 (Geplanter MCU):
    https://www.st.com/en/microcontrollers-microprocessors/stm32g473qe.html
  - STM32 F412ZG (benutzter MCU):
    https://www.st.com/en/microcontrollers-microprocessors/stm32f412zg.html
  - SN65HVD235D (CAN Transceiver):
    https://www.ti.com/product/SN65HVD235

B Tabellarische Aufführung der Tätigkeiten

TIMELINE 1: *Auflistung der Tätigkeiten*

1 - Lötpraktikum und Einstieg zu den Produkten der Murrelektronik mit anschließendem Praktikum

2 - Einführung in C++ und objektorientierter Progammierung mit klassen und Vererbung

3 - Parallelisierung mit Threads und Datenaustausch von Prozessen mittels fortgeschrittener Datenstrukturen und Eventhandling mit Übungsprojekt

4 - Einführung in UML Diagramme für Software- Entwicklung und Enterprise Architekt als Werkzeug

5 - Einführungen in Software- Kapselung und Bibliotheken (Statisch sowie Dynamisch) und deren Bau in C++

6 - Planung und Bau einer Bibliothek zum Parsen sowie Schreiben eines XML- Dokuments

7 - Einführung in TCP/ IP und Websockets zum anpingen von Servern sowie zum Datenaustausch mit aschließendem Projekt

8 - Flashen von Software über das CAN- Protokoll sowie UI- Design mit WPF und C# als Praxisprojekt der Praxisarbeit 1

9 - Bau einer Bibliothek mit Klassen zur Kommunikation über das SPI- Protokoll und zur Kommunikation zu einem EEPROM mittels SPI

10 - Projekt zum Thema Wasserstoff als führende Antriebskraft

C Klassendiagramme

C.1 Überblick ADC

C.2 Überblick Diagnosen

C.3 Überblick ADC

D Code Beispiele

D.1 Konfigurationsstruktur der ADCs

```
1  struct MEF_ADCConfig
2  {
3    uint16_t channelID;
4
5    iConvert* converter;
6  };
7
8
9  struct MEF_ADS124S08Config : MEF_ADCConfig
10 {
11   ADS124S08Pin Ain_plus;
12
13   ADS124S08Pin Ain_minus;
14
15   ADS124S08RefSelection ref;
16
17   ADS124S08Datarate datarate = ADS124S08Datarate::DR_400SPS;
18
19   ADS124S08PGAGain gain;
20 };
```

Listing 3: Konfigurationsstruktur der ADCs mit Interface(MEF_ADCConfig) und der Erweiterung für den externen ADC (MEF_ADS124S08Config)

D.2 Umwandlungsalgorithmus eines Bytearrays mit 4 Gliedern in einen Integer- Wert

```
1 // Convert 3 bytes of uint8_t array to int32_t, data starts at index
      1
2 for(uint8_t index = 3; index > 0; index--)
3   {
4     data = data | ((int32_t)(rxData[4 - index])) << index * 8;
5   }
6 data = data >> 8;
```

Listing 4: Konvertierungsalgorithmus

D.3 Konfigurieren eines Multimodes zum Messen mehrerer Kanäle über DMA ohne CPU- Zeit für das Konfigurieren der nächsten Kanäle zu verschwenden

```
1 multimode.Mode = ADC_MODE_INDEPENDENT;
2
3 if (HAL_ADCEx_MultiModeConfigChannel(&hadc1, &multimode) != HAL_OK)
4 {
5     Error_Handler();
6 }
7
8 /** Configure Regular Channel 1
9 */
10 sConfig.Channel = ADC_CHANNEL_1;
11 sConfig.Rank = ADC_REGULAR_RANK_1;
12 sConfig.SamplingTime = ADC_SAMPLETIME_640CYCLES_5;
13 sConfig.SingleDiff = ADC_SINGLE_ENDED;
14 sConfig.OffsetNumber = ADC_OFFSET_NONE;
15 sConfig.Offset = 0;
16 if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
17 {
18     Error_Handler();
19 }
20
```

```
21 /** Configure Regular Channel 2
22 */
23 sConfig.Channel = ADC_CHANNEL_TEMPSENSOR_ADC1;
24 sConfig.Rank = ADC_REGULAR_RANK_2;
25 if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
26 {
27     Error_Handler();
28 }
29
30 /** Configure Regular Channel 3
31 */
32 sConfig.Channel = ADC_CHANNEL_VREFINT;
33 sConfig.Rank = ADC_REGULAR_RANK_3;
34 if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
35 {
36     Error_Handler();
37 }
```

Listing 5: Konfigurieren des Multimodes des internen ADC mit verschiedenen Rängen