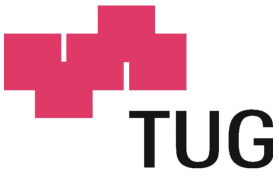
Diplomarbeit

Modellgetriebenes Systemdesign bei der Entwicklung eines Sensors zur Erfassung der solaren Einstrahlung

Florian Köck



Institut für Elektronik

Technische Universität Graz
Infeldgasse 12, 8010 Graz, Austria

Betreuer: Dipl.-Ing. Christian Neureiter
Begutachter: Ass.-Prof. Dipl.-Ing. Dr.techn. Bernd Eichberger

Graz, Mai 2011

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am ....
(Unterschrift)

Kurzfassung

Zielsetzung dieser Diplomarbeit war die Erstellung eines Prototyps, der die solare Einstrahlung misst. Der Prototyp soll an einen Solarregler (Energy Control Unit) angeschlossen werden und die solare Einstrahlung messen. Der Momentanwert der solaren Einstrahlung soll als Parameter in die Regelung einer solarthermischen Anlage einfließen. Bei der Realisierung wurden Methoden des Systems Engineerings und der modellgetriebenen Entwicklung verwendet.

Zu Beginn erfolgt eine kurze Einführung in die Thematik des Systems Engineerings, der modellgetriebenen Entwicklung und die für die Modellierung notwendigen Werkzeuge.
In Folge wurden Modelle verwendet, die angefangen von der Festlegung der Anforderungen bis hin zur Implementierung des Prototyps die Realisierung beschreiben. Es wurden verschieden Umsetzungsmöglichkeiten gegeneinander abgewogen, und eine realisiert.
Abschließende Messungen am Prototyp belegen die Funktionalität des Sensors und der gewählten Methode zur Realisierung.

Schlüsselwörter: Systems Engineering, modellgetriebene Architektur, solare Einstrahlung, Schnittstelle, MSP430

Abstract

The aim of this thesis was to create a prototype to measure the solar radiation. The prototype should be connected to a solar controller (Energy Control Unit) to measure the solar radiation. The current value of the solar radiation should flow in as a parameter in the control of a solar thermal system. For the realization, methods of systems engineering and model driven development were used.

First of all a short introduction to the topics of systems engineering, model driven development and the need of modeling tools is given.
Then models were used to implement the Sensor, from the beginning of the definition of the system requirements to the implementation of the sensor prototype. There were various implementation options weighed, and finally one of them implemented.
Final measurements of the prototype show the functionality of the sensor and the used method for the realization.

Keywords: systems engineering, model driven architecture, solar radiation, interface, MSP430

Danksagung

Diese Diplomarbeit wurde im Studienjahr 2010/11 bei der Firma NTE Systems zusammen mit dem Institut für Elektronik an der Technischen Universität Graz durchgeführt.

Bedanken möchte ich mich bei Herrn Dipl.-Ing. Georg Stasny der die Durchführung dieser Diplomarbeit ermöglichte. Mein spezieller Dank gilt Herrn Dipl.-Ing. Christian Neureiter der die Betreuung der Diplomarbeit übernahm, und mir stets mit gutem Rat zur Seite stand. Auch bei allen Mitarbeitern der Firma NTE Systems bedanke ich mich für die große Hilfsbereitschaft und das angenehme Arbeitsklima.

Ich danke Herrn Ass.Prof. Dipl.-Ing. Dr.techn. Bernd Eichberger am Institut für Elektronik für die Betreuung des Themas und der Hilfe während der Durchführung der Diplomarbeit.

Ein besonderer Dank gilt meinen Eltern, die mir dieses Studium ermöglichten. Ohne die finanzielle und persönliche Unterstützung wäre dieses Studium nicht möglich gewesen.

Für die Unterstützung und die spannende Zeit möchte ich allen FreundInnen und Mitstudierenden danken, die mich während meines Studiums in Graz und Sevilla begleitet haben.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Die Nutzung von alternativen Energieformen nimmt einen immer höheren Stellenwert ein. Photovoltaik- und Solarthermieanlagen sind bereits weit verbreitet und nutzen die Energie der Sonne, um Strom oder Wärme zu erzeugen. Abhängig von der solaren Einstrahlung bringen diese Anlagen einen bestimmten Ertrag an elektrischer bzw. thermischer Leistung.

Um den aktuell möglichen Ertrag der Solaranlage zu messen, werden Sensoren eingesetzt die die aktuelle solare Einstrahlung $[W/m^2]$ messen. Dieser Messwert dient als Referenz für die Anlage. Treten Fehler in der Anlage auf, besteht die Möglichkeit zusammen mit anderen Messwerten den Fehler zu lokalisieren. Gängige Parameter sind dabei Temperatur und Durchfluss. In Abbildung 1.1 ist ein schematischer Aufbaue einer solarthermischen Anlage abgebildet.

Ziel dieser Diplomarbeit ist, einen Prototyp zu entwickeln der den Momentanwert der solaren Einstrahlung misst.

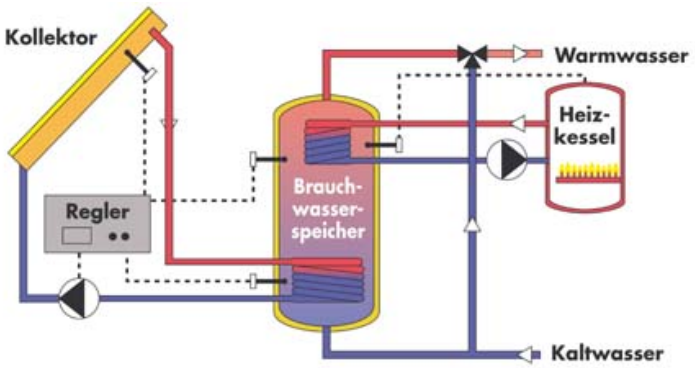


Abb. 1.1: *Schema einer thermischen Solaranlage Quelle:res-solar.de*

Bei bereits bestehenden Anlagen ist oft ein enormer Aufwand zu betreiben um nachträglich Messleitungen zu installieren. Damit sich der Installationsaufwand in Grenzen hält, soll der Sensor an die Messleitung des PT1000 Kollektorfühlers angeschlossen werden. Die vorhandene Temperaturmessung, die mit dem Solarregler der Energy Control Unit ECU stattfindet, muss weiterhin möglich sein.

Zusätzliche soll der Sensor so ausgelegt werden, dass er als Basisstation für die Errichtung eines „Wireless Sensor Network (WSN)“ fungieren kann. Ein WSN besteht aus einem Netz verschiedener Sensoren, die Umgebungsparameter erfassen und diese über Funk an eine Basisstation senden. Die Basisstation muss ständig empfangsbereit sein, was erhöhte Anforderungen an die Energieversorgung stellt.

Bei der Umsetzung wurden Aspekte des Systems Engineerings beachtet und Methoden der modellgetriebenen Entwicklung verwendet. Dazu erfolgt zu Beginn dieser Diplomarbeit eine kurze Übersicht und Einführung in diese Themenbereiche.

1.2 Systems Engineering

Die Hauptaufgaben des Systems Engineering sind:

- das Definieren der Systemanforderungen,
- die Erstellung der Dokumentation in der frühen Entwicklungsphase,
- die Erstellung eines Systemdesigns,
- und die Überprüfung auf deren Einhaltung unter Berücksichtigung des Gesamtproblemes.

Das Systems Engineering beleuchtet unterschiedliche Disziplinen wie Softwareentwicklung, Hardwareentwicklung und Verfahrenstechnik und führt diese zusammen.
Das in [18] beschriebene SIMILAR Modell gibt einen Überblick über den Prozess des Systems Engineerings. In Abbildung 1.2 ist der Prozess grafisch dargestellt. Das SIMILAR Modell wird unterteilt in:

- **S**tate the problem (Anforderungsmodell erstellen)
  Zu Beginn steht die Beschreibung der Anforderungen an das System. Werden diese nicht eindeutig festgelegt, kann dies im Laufe der Entwicklung sehr kostspielig werden und zu Komplikationen in der Umsetzung führen.
- **I**nvestigate alternatives (Alternative Lösungen prüfen)
  Eine oft vernachlässigte Aufgabe ist die Prüfung von alternativen Konzepten bei der Umsetzung. Es ist unerlässlich verschiedene Möglichkeiten der Realisierung gegeneinander abzuwägen. Oft existiert in unseren Köpfen bereits eine Idee zur Lösung der Aufgabenstellung und konzentriert sich nur auf einen Lösungsansatz.
- **M**odel System (Systemmodell erstellen)
  Das Systemmodell stellt neben der Spezifikation auch die Verwaltung des Systems während des gesamten Lebenszykluses dar. Dafür werden Modellierungswerkzeuge wie die *Unified Modeling Language (UML)* verwendet.
- Integrate (System einbetten)
  Um das Interagieren des Systems mit seiner Umgebung zu ermöglichen müssen Systemschnittstellen definiert werden.
- **L**aunch the system (System implementieren)
  Entsprechend den in den vorangegangen Phasen festgelegten Systemanforderungen wird die Implementierung der einzelnen Komponenten vorgenommen.
- **A**ssess performance (Messwerte prüfen)
  Das lauffähige System wird getestet und geprüft ob es den Systemanforderungen entspricht.

- **R**e-evaluate (Projektergebnisse prüfen)
  Diese Aufgabe steht übergreifend über allen anderen Aktivitäten. Erkenntnisse die während des Prozesses gewonnen wurden werden kritisch geprüft und bewertet.

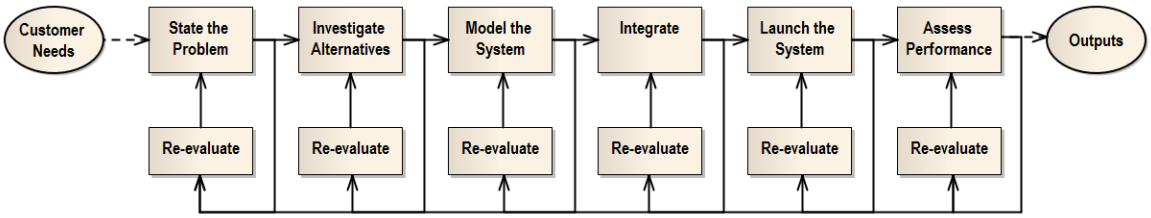


Abb. 1.2: *Similar Prozessmodell Quelle:[1]*

Das Risikomanagement ist ein unerlässlicher Teil des Systems Engineering. Es sorgt dafür, dass potentielle Risiken identifiziert und Maßnahmen ergriffen werden.

1.3 Agile Methoden und Traditionelle Prozesse

Wie in [3] auf Seite 14ff beschrieben ist das primäre Ziel bei agilen Projekten die Erstellung eines funktionierenden Systems, das den Anforderungen der Auftraggeber und Anwender gerecht wird. Das sekundäre Ziel bei einem agilen Designprozess ist die Weiterentwicklung eines Projektes. Dies beinhaltet eine klare Struktur und eine lückenlose Dokumentation, um einem anderen Team das Weiterarbeiten zu ermöglichen.
Als Mehrwert von Agilen Methoden werden gesehen:

- Rapid Learning
  Ein frühes Feedback erlaubt ein dynamisches Planen. Im Gegensatz zur ballistischen Planung wo zu Beginn festgelegt wird was umzusetzen ist, werden bei der dynamischen Planung durch Rücksprache mit dem Auftraggeber bereits in der frühen Umsetzungsphase Unsicherheiten ausgeräumt.

- Early Return on Investment
  Im Gegensatz zum klassischen Wasserfallmodell ist bei einer agilen Vorgehensweise eine Teilfunktionalität bereits sehr früh gegeben. Ebenso können Teile die im Projekt mit hohem Risiko eingeschätzt werden zu Beginn gelöst werden, womit wiederum das Gesamtrisiko des Projekts gesenkt werden kann.

- Satisfied Stakeholders
  Agile Methoden ermöglichen es validierte Funktionalität bereits in einer frühen Phase den Stakeholdern zu demonstrieren. Funktionalität kann, abhängig von den Prozesskriterien, nach verschiedenen Gesichtspunkten implementiert werden (Highest risk first, most critical first, infrastructure first, available information first).

- Improved Control
  Das Identifizieren der Projektziele und das Verfolgen dieser stellt die Herausforderung dar und wird oft unterschätzt. Die projektrelevanten Teile müssen so verfolgt werden, dass sie effektiv verwaltet, angemessene Neuplanungen möglich sind oder Teile gegebenenfalls verworfen werden können. Neuplanungen sind nur dann möglich wenn mehr Informationen zur Verfügung stehen als beim Erstellen des ursprünglichen Planes. Unsicherheiten, verbunden mit Kosten, Zeit, Aufwand und Qualität sollten damit reduziert werden.

- Responsiveness to Change
  In jedem Projekt gibt es Dinge die zu Beginn nicht geplant werden können. Bei der agilen Projektumsetzung werden Fehler in der Planung bis zu einem bestimmten Maße akzeptiert und angepasst. Dies erfolgt durch die Unterteilung des Projektes in verschiedene Zyklen, die in 1.4 beschrieben werden.

- Earlier and Greater Reduction in Project Risk
  Die häufigste Ursache für das Scheitern eines Projektes ist das Ignorieren von Risiken. Um dies zu vermeiden werden früh Machbarkeitsstudien (Proof of Concepts) durchgeführt.

- Efficient Hight Quality Development
  Das Ziel, hohe Qualität mit permanent vorzeigbarem Fortschritt und ausführbaren Teilfunktionalitäten, kann bereits in einer frühen Entwicklungsphase erreicht werden.

1.4 Model Driven Architecture (MDA)

Als modellgetriebene Architektur bezeichnet man einen Entwicklungsansatz, der auf einer strikten Trennung von Funktionalität und Technik beruht. Der Begriff stammt aus der Softwareentwicklung mit dem Ziel, aus Modellen einen automatisch generierten Code zu erzeugen und somit die Qualität und die Wiederverwendbarkeit von Source Code zu erhöhen.
Die Object Management Group (OMG) bezeichnet den Terminus Modell als eine abstrakte Beschreibung oder Spezifikation eines Systems und seines Umfeldes für einen bestimmten Verwendungszweck. Die wichtigsten Merkmale von Modellen sind:

- Ein Modell ist eine Vereinfachung des zu modellierenden Systems
- Ein Modell beschreibt einen bestimmten Zweck oder Absicht
- Modelle legen den Fokus auf spezielle Aspekte einer Zielsetzung

Um dies zu erreichen, werden verschiedene Modelle mit unterschiedlichem Abstraktionsgrad erstellt. Ausgehend von dem in der Hierachie am höchstgelegenen Modell werden untergeordnete Modelle abgeleitet 1.3, idealerweise zu einem gewissen Grad automatisch erstellt. MDA unterteilt das Gesamtmodell in folgende vier Ebenen.[3]

- Computation-Independent Model (CIM)
  Das CIM beschreibt die Zielsetzung des Systems, ohne dabei auf dessen Implementierung einzugehen. Das CIM beschreibt die geforderte Funktionalität ohne einzelne Teile genau zu beschreiben. Wichtige Elemete sind dabei Anwendungsdiagramme (Use Case Diagramme), sie beschreiben das System verbal. Von ihnen werden die Anforderungen abgeleitet und die Systemgrenzen festgelegt. Das CIM erfasst und organisiert die Anforderungen des Systems in Worten und ordnet diese in Epics und Use–Cases.

- Platform-Independent Model (PIM)
  Im PIM werden die für die Realisierung notwendigen systemrelevanten Teile und ihr grundlegendes Verhalten festgelegt, abgeleitet aus den Use Cases des CIM. Es werden ausschließlich fachliche Aspekte betrachtet und ist unabhängig von der Plattform auf der das System integriert wird. Für das Modellieren von Abläufen und Architekturen werden Sequence Diagramme, State Machines und Structure Diagramme verwendet.

- Platform-Specific Model (PSM)
  Das PSM repräsentiert Informationen über die ausgewählte Plattform und das Zielsystem auf dem es realisiert wird. Es ist das Designmodell und ist unerlässlich bei der Umsetzung und Optimierung der geforderten Funktionalitäten. Neben den fachlichen Informationen sind auch Informationen über die eingesetzte Technologie enthalten.

- Platform-Specific Implementation (PSI)
  Die PSI ist kein Modell im Sinne von CIM, PIM und PSM. Sie bezieht sich auf den Code der aus dem PSM generiert wurde.

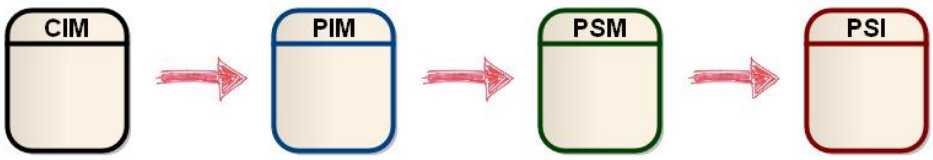


Abb. 1.3: *MDA Modelle Quelle:[3]*

1.5 Unified Modeling Language (UML)

UML ist eine graphische, allgemein verwendbare Modellierungssprache zur Spezifikation, Konstruktion und Dokumentation von Systemen. Sie stellt Diagramme und Notationselemente zur Verfügung, mit deren Hilfe sowohl statische als auch dynamische Aspekte beliebiger Anwendungsgebiete modelliert werden können. Ein Überblick über den Aufbau von UML ist in Abbildung 1.4 zu sehen.

1. Es wird zwischen Struktur- (z.B. Klassen und Komponenten) und Verhaltenselementen (z.B. Aktivitäten und Zustandsautomaten) unterschieden. Im Bereich Sonstiges sind Elemente die sich auf Verhalten und Struktur beziehen. Strukturelemente dienen zum Modellieren von statischen, zeitunabhängigen Elementen des Systems. Verhaltenselemente modellieren die dynamischen Aspekte.

2. Die weitere Unterscheidung erfolgt in Modell und Diagramm. Das Modell ist die vollständige Beschreibung des Systems, das Diagramm die Visualisierung.

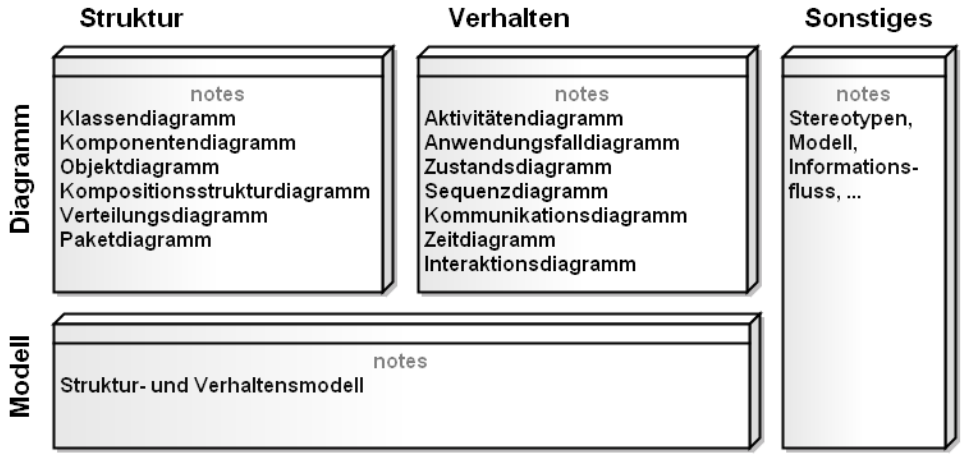


Abb. 1.4: *UML Aufbau Quelle:[18]*

Kapitel 2

Methoden

2.1 Computation–Independent–Model (CIM)

In der Anfangsphase jedes Projektes, welcher Domäne auch immer, ist es erforderlich die Zielsetzung klar zu definieren. Die Anforderungen an das System stellen die sogenannten Stakeholder. Sie sind entscheidend für eine erfolgreiche Umsetzung eines Projektes da sie die nötigen Informationen an das zu entwickelnde System und das Domänen Know How liefern. Aus diesen Informationen werden die technischen Anforderungen abgeleitet. Für die Umsetzung des Sensors wurden folgende Stakeholder definiert:

- Product Owner: Projektinitiator
- User: Betreiber der Solarthermieanlage

Der Projektinitiator legt fest dass dies über die Sensorleitung des Kollektorfühlers stattfinden soll.
Der Benutzer will die aktuelle solare Einstrahlung messen, um sie als Parameter für seine Regelung zu verwenden.

2.1.1 Stakeholder Requirements

Um die technischen Anforderungen des Systems zu erhalten wurde jeder Stakeholder befragt und eine Liste in Tabellenform erstellt 2.1. Die Fragen wurden dabei klar und präzise formuliert und spiegeln die Bedürfnisse der Stakeholder wieder.
Um das System besser analysieren zu können wurden die Stakeholder Requirements in Epics und User Stories eingeteilt, die im Weiteren die System Requirements ergaben.
Der Unterschied zwischen Stakeholder Requierements und System Requirements besteht darin, dass die Stakeholder Requirements die Bedürfnisse der Stakeholder repräsentieren, die System Requierements legen fest wie das System die Stakeholder Requirements erfüllt. [3]

1.	Epic	Messung durchführen	
1.1	User Story	Funktionalität	Ich als Product Owner möchte die solare Einstrahlung messen um eine Referenz zur aktuell möglichen Ernte meiner Solarthermieanlage zu erhalten.
1.2	User Story	Funktionalität	Ich als User möchte dass die Temperaturmessung weiterhin stattfinden kann um die Kollektortemperatur zu messen.
1.3	User-Story	Energieversorgung	Ich als Product Owner möchte den Solarsensor mit den vorhandenen Energiequellen versorgen um keine Versorgungsleitungen auf dem Dach installieren zu müssen.
1.3	User-Story	Energieversorgung	Ich als Product Owner möchte dass die Energieversorgung für mögliche Erweiterungen für ein Wireless Sensor Network gerüstet ist.
2.	Epic	Messwerte übertragen	
2.1	User-Story	Leitungslänge	Ich als Product Owner möchte dass eine Übertragung bis zu 100m sichergestellt ist
2.2	User-Story	Datenübertragung	Ich als Product Owner möchte dass die Messwerte über die vorhandene Zweidrahtleitung des PT1000 übertragen werden um keine zusätzliche Datenleitung installieren zu müssen.
2.3	User-Story	Abtastfrequenz	Ich als User möchte dass jede Minute ein neuer Einstrahlungswert überliefert werden kann um einen aktuellen Referenzwert zu haben.
2.4	User-Story	Temperaturbereich	Ich als Product Owner möchte dass die Funktionalität im Temperaturbereich von $-20 \ldots +80°C$ gewährleistet ist um im Winter und Sommer gültige Ergebnisse zu erhalten.
2.5	User-Story	Codierung	Ich als Product Owner möchte auf derselben Leitung auch andere Daten übermitteln können um für etwaige Erweiterungen gerüstet zu sein.

Tabelle 2.1: *Stakeholder Requirements*

2.1.2 System Requirements

2.1.2.1 Functional Requirements

Um auf eine technische Abstraktionsebene zu gelangen wurden aus den Epics und User-Stories Technical Issues gebildet, die für die Beurteilung und die spätere Umsetzung als relevant erachtet wurden. Die Technical Issues wurden in folgende Teilbereiche unterteilt:

Power Supply

- Temperaturmessung nicht beeinflussen
  Die bestehende Messung wird beibehalten oder erfolgt mit dem Sensor.
- Keine Versorgungsleitung
  Für die Energieversorgung darf keine zusätzliche Versorgungsleitung verwendet werden.
- Versorgungskonzept
  Nach dem 1-Wire Prinzip (eine gemeinsame Energie- und Datenleitung), oder Energy Harvesting über Photovoltaik Zellen.
- Energieversorgung für Wireless Sensor Network (WSN)
  Für das Errichten eines WSN, wie ZigBee oder EnOcean, ist ein erhöhter Energieaufwand notwendig ($\approx 35mA$ bei 3V Versorgungsspannung).

Kommunikation

- Worst Case: Ungeschirmte Zweidrahtleitung
  Für eine Leitungslänge von 100m wird eine Übertragungsfrequenz von 10kHz festgelegt.
- Kommunikationskonzept
  Die Kommunikation erfolgt unidirektional oder bidirektional.

Sensorik

- Detektion der Einstrahlung
  Um eine exakte Referenz für Solarthermieanlagen zu erhalten, muss die Messung thermisch erfolgen. Ungenau aber kostengünstig erfolgt die Messung mit einer Photodiode.

2.1.2.2 Non Functional Requirements

- Montage
  Eine einfache Montage des Sensors am Dach muss möglich sein.

- Temperaturbereich
  Da es am Dach zu hohen Temperaturschwankungen kommt ist die Funktionalität im Temperaturbereich von -20°C bis +80°C zu gewährleisten.

- Vorbereitung für WSN
  Der Sensor soll für eine Erweiterung eines WSN gerüstet sein, das erhöhte Ansprüche an die Energieversorgung stellt. Die Datenkodierung muss erweiterbar sein.

2.1.3 Deployment Model

Um die Interaktion des Sensors mit der Umgebung zu beschreiben wurde das Deployment Model, dargestellt in Abbildung 2.1, erstellt. Es zeigt die Interaktion des Sensors mit der *Energy Control Unit (ECU)*, dem Solarregler. Die Messung der solaren Einstrahlung erfolgt ohne dabei die Temperaturmessung zu beeinflussen. Als Übertragungsmedium dient die PT1000 Sensorleitung.

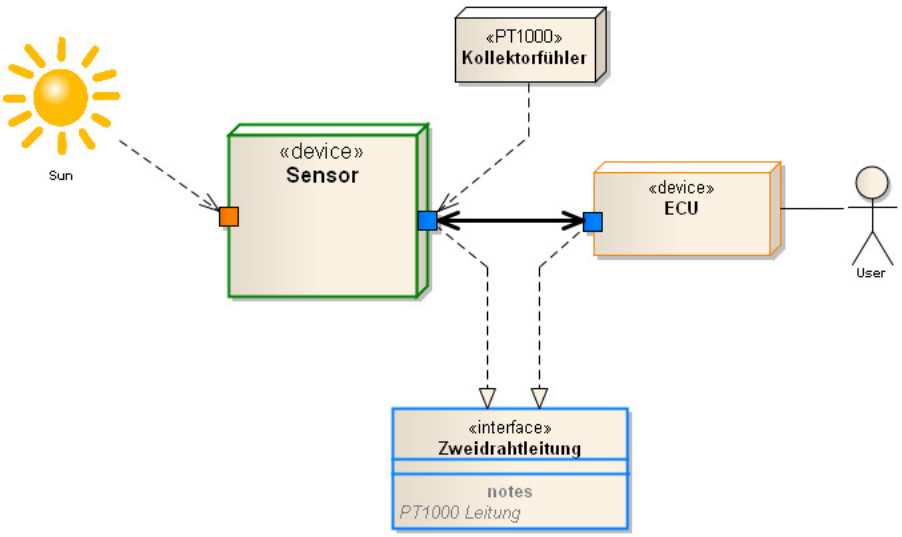


Abb. 2.1: *Deployment Model*

2.1.4 Zusammenfassung der System Requirements

- Erfassung der solaren Einstrahlung über die Kollektorfühler Messleitung
- Die Messung der Temperatur muss weiterhin möglich sein, ohne an Genauigkeit zu verlieren
- Leitungslängen bis 100 Meter müssen möglich sein

- Die Möglichkeit zum Aufbau eines Wireless Sensor Networks am Sensor muss möglich sein
- Die Funktionalität im Temperaturbereich von -20°C bis 80°C ist zu gewährleisten

2.2 Platform–Independent–Model (PIM)

Nachdem die System Requierements festgelegt wurden werden verschiedene Möglichkeiten der Realisierung evaluiert. Der Fokus wurde dabei auf eine einfache und kostengünstige Umsetzung gelegt. Kernelement dafür ist die Energieversorgung des Sensors, die im Weiteren die Art der Datenübertragung und die Temperaturmessung beeinflusst.

2.2.1 Evaluierung verschiedener Varianten

2.2.1.1 Interface

Energy Harvesting Der Sensor wird autark mit Photovoltaikzellen versorgt. Der Temperaturmesskanal bleibt erhalten und die ECU führt weiterhin die Messung durch. Nachteil dieser Methode ist, dass die Energieversorgung von der Sonneneinstrahlung abhängig ist und somit nicht ständig gewährleistet ist. Ein weiterer Nachteil weil mit hohen Kosten verbunden ergibt sich bei Errichtung eines WSN. Als mögliche WSN wurden ZigBee von Texas Instruments und EnOcean der EnOcean Allianz in Erwägung gezogen. Da der Sensor als Basisstation fungiert, muss er ständig empfangsbereit sein um die Daten der Sensorknoten zu empfangen. Aus den Datenblättern geht hervor, dass sowohl ZigBee als auch EnOcean bei 3V Versorgungsspannung einen Strom von $\approx 35mA$ benötigen. Mit der Größe des Photovoltaik-Panels steigen die Kosten.

1-Wire Prinzip Für den Sensor wird ein neuer Kanal auf der ECU integriert. Die Messung der solaren Einstrahlung und die Temperaturmessung werden vom Sensor durchgeführt. Bei 1-Wire erfolgt die Stromversorgung und die Datenübertragung über die selbe Leitung. Vorteil dieser Variante ist die Unabhängigkeit der Stromversorgung von der Einstrahlung. Weiters kann das Interface so dimensioniert werden dass es für die Errichtung eines WSN vorbereitet ist.

- Maxim 1-Wire Lösung:
  Verschiedene 1-Wire Löungen sind in [8] beschrieben. Vorteile dabei sind die für verschiedene Plattformen bereits vorhandene Implementierung und die Möglichkeit der Verwendung eines Interface Chips von Maxim. Nachteil ist, dass diese Chips Single-Source-Devices sind und somit nur von Maxim angeboten werden. Ebenso ist die Bereitstellung der benötigten Leistung ohne zusätzlichen Aufwand nicht möglich.
- Eigene 1-Wire Lösung:
  Bei der Implementierung einer an das 1-Wire Prinzip angelehnten Lösung kann das

Interface den Anforderungen angepasst werden. Die Verwendung von Standard Bauteilen erlaubt eine hohe Flexibilität und man ist nicht von Single-Source-Devices abhängig.

Fazit: Schlußendlich wurde für die „Eigene 1–Wire Lösung“ entschieden. Grund dafür war vor allem die Unabhängigkeit von der Sonneneinstrahlung, die Möglichkeit der Verwendung von Standardbauteilen und die Flexibilität in Bezug auf die Anpassung an die Requirements.

2.2.1.2 Messung der Einstrahlung

Die von der Sonne ausgesendete, kurzwellige Strahlung befindet sich im Bereich von etwa $0,3 \ldots 2,5 \mu m$. Abbildung 2.2 zeigt die Strahlungsintensität eines idealen schwarzen Körpers, der Sonnenstrahlung außerhalb der Erde und die durch Absorption und Streuung gedämpfte, resultierende Strahlung auf der Erde, abhängig von der Wellenlänge.

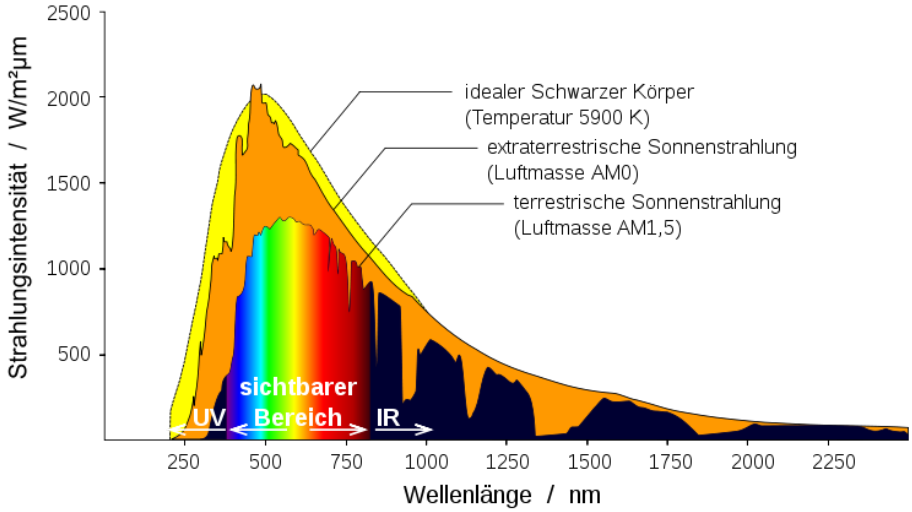


Abb. 2.2: *Strahlungsinstensität der Sonne, Quelle: de.wikipedia*

Um das gesamte Strahlungsspektrum zu erfassen führt man die Messung mit thermischen Detektoren durch. Diese hauptsächlich in der Meteorologie eingesetzten Messgeräte werden Pyranometer genannt. Grundelemente von Pyranometern sind Thermoelemente die zu einer Thermosäule geschaltet sind. Die Detektoroberfläche ist geschwärzt, um über einen weiten Spektralbereich eine konstant hohe Empfindlichkeit zu erreichen. Durch Vergleich der Spannung mit einer nicht bestrahlten Seite wird die Bestrahlungsstärke bestimmt. Thermosäulen sind sehr aufwendig herzustellen und sehr teuer.

Eine weitaus günstigere Methode ist die Erfassung der Einstrahlung mit Photodioden. Der Nachteil ist, dass Photodioden eine von der Wellenlänge abhängige Empfindlichkeit

besitzen und somit keine exakte Messung erlauben. Photodioden erreichen im Einsatz als Referenzquellen in Photovoltaikanlagen eine gute Genauigkeit, da sie auf dem selben Grundprinzip beruhen und abhängig vom Halbleitermaterial eine ähnliche spektrale Empfindlichkeit aufweisen. Dagegen nutzen solarthermische Kollektoren annähernd das gesamte Strahlungsspektrum, abhängig vom Absorber des Sonnenkollektors.

Fazit: Zur Messung der Einstrahlung wurde aus Kostengründen eine Photodiode verwendet. Mit der Messung wird geprüft, ob ein Ertrag mit der Solaranlage möglich ist, oder ob es Probleme in der Anlage gibt.

2.2.2 Proof of Concept Interface

Um in dieser Phase des Designs abschätzen zu können ob das gewählte Konzept auch wirklich realisierbar ist, wurde ein Proof of Concept (POC) des Interfaces erstellt. Sollte der Proof of Concept belegen, dass dieses Konzept nicht umsetzbar ist, halten sich die Kosten in dieser frühen Phase in Grenzen.

2.2.2.1 Interface

Das Interface wurde nach dem Master–Slave Prinzip konzipiert. Der Master stellt die Ressourcen für den Sensor zur Spannungsversorgung und Kommunikation bereit.
Das Prinzipschaltbild ist in Abbildung 2.3 zu sehen. Der Master liefert über die Stromquelle I1 den benötigten Strom für den Sensor. Bei einer Übertragung wird die Stromquelle I1 deaktiviert und die Stromquelle I2 aktiviert. Der Slave überträgt seine Daten durch kurzschließen der Zweidrahtleitung. Während dieser Zeit wird mit dem Kondensator C1 die Stromversorgung des Sensors überbrückt, die Diode D2 verhindert das Entladen des Kondensators. Die Ausgangsspannung wird mit einem Spannungsregler auf die Betriebsspannung geregelt.

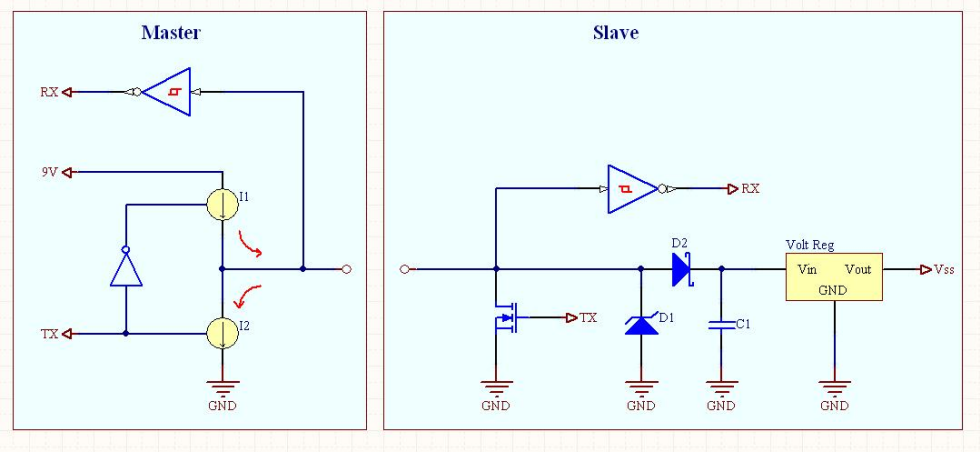


Abb. 2.3: *PIM: POC Interface*

Anhand von Simulationen und/oder Messungen wird entschieden ob auf dem gewählten Ansatz weiterentwickelt wird. Sind die Ergebnisse nicht zufriedenstellend, ist ein Redesign nötig.

2.2.3 Behavioural Model

Nachdem das Gerüst des Interfaces den Anforderungen entsprach, wurden als nächste Schritte das Protokoll, das Timing der Kommunikation und der Programmablauf definiert.

2.2.3.1 Übertragungsfrequenz

Um einen Kompromiss aus Übertragungsgeschwindigkeit und Leitungslänge zu finden wurde eine Übertragungsfrequenz von 10kHz definiert. Der Kondensator C1 in Abbildung 2.3 muss während der Übertragung der neun Byte ($\approx 10ms$) die Versorgungsspannung überbrücken, oder es wird nach jedem Datenbyte eine Verzögerung eingeführt, um den Kondensator wieder nachzuladen.

2.2.3.2 Timing der Kommunikation

Abbildung 2.4 zeigt das Timing der Kommunikation. Der Master (TX_{MASTER}) startet die Kommunikation und fordert Daten vom Slave. Der Slave dekodiert den empfangenen Befehl, führt ihn aus, codiert das Ergebnis und sendet die Daten (TX_{SLAVE})an den Master zurück.
Das Timing wurde so ausgelegt dass bei einer Übertragungsfrequenz von 10kHz alle 100ms eine Kommunikation gestartet werden kann. Die restliche Zeit von 80ms wird zur Ausführung der Befehle und zum Nachladen des Kondensators C1 in Abbildung 2.3 verwendet.

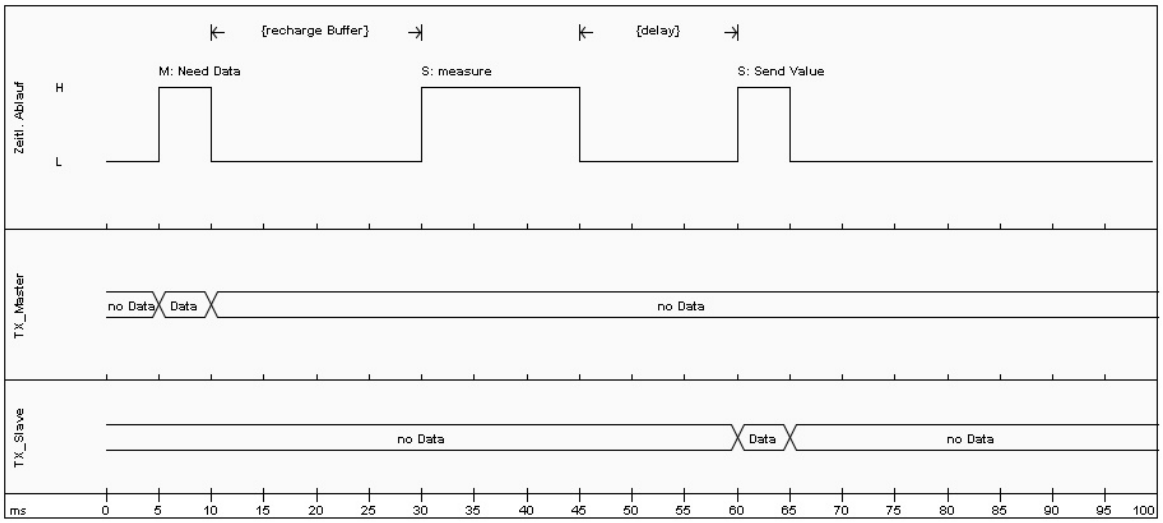


Abb. 2.4: *Timing-Diagramm der Kommunikation*

2.2.3.3 Protokoll

Das Protokoll wurde für einen Slave ausgelegt. In Abbildung 2.5 ist die Struktur dargestellt.

Protokollstruktur

SYNC	COM ID	8 Bit	8 Bit	8 Bit	8 Bit	8 Bit	8 Bit	CRC

Abb. 2.5: *Protokollstruktur*

Jedes Kommandowort besteht aus neun Byte und ist in seiner Struktur immer gleich. Zu Beginn wird ein Synchronisations-Byte gesendet, danach folgt die Kommandoidentifikation, die die Art des Befehls darstellt. Die nächsten sechs Byte sind befehlsabhängig. Am Ende wird immer die Cycle Redundancy Check (CRC) Prüfsumme angehängt.
Die Darstellung von Gleitkommazahlen erfolgt nach dem IEEE 754 Standard mit einfacher Genauigkeit (32Bit).
Zur Bildung der acht Bit CRC Prüfsumme wurde das Generatorpolynom 2.1 des Dallas/-Maxim 1–Wire Bus verwendet. Eine Beschreibung ist in [9] zu finden.

$$x^8 + x^5 + x^4 + 1 \tag{2.1}$$

Das gesamte Protokoll ist in Abbildung 2.6 abgebildet.

Get Channel Value

SYNC	0xF0	CH	NUM	0xFF	0xFF	0xFF	0xFF	CRC

Send Channel Value

SYNC	0xFA	CH	NUM	32Bit Float	CRC

Get Sensor ID

SYNC	0x33	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	CRC

Send Device ID

SYNC	0x44	Device ID	CRC

Abb. 2.6: *Protokoll*

2.2.3.4 State–Machine

Abbildung 2.7 zeigt den Ablauf des Programms. Nach dem Hochfahren wird der Slave (Sensor) initialisiert und geht in einen Stromsparmodus. Wird ein Interrupt ausgelöst, verlässt er den Stromsparmodus und überprüft die empfangenen Daten auf ihre Gültigkeit. Wurden gültige Daten empfangen wird der Befehl ausgeführt, die Daten codiert und an den Master (ECU) gesendet. Anschließend wird in den Stromsparmodus gegangen.
Sind die Daten fehlerhaft, werden die Daten verworfen und es wird sofort in den Stromsparmodus gegangen.

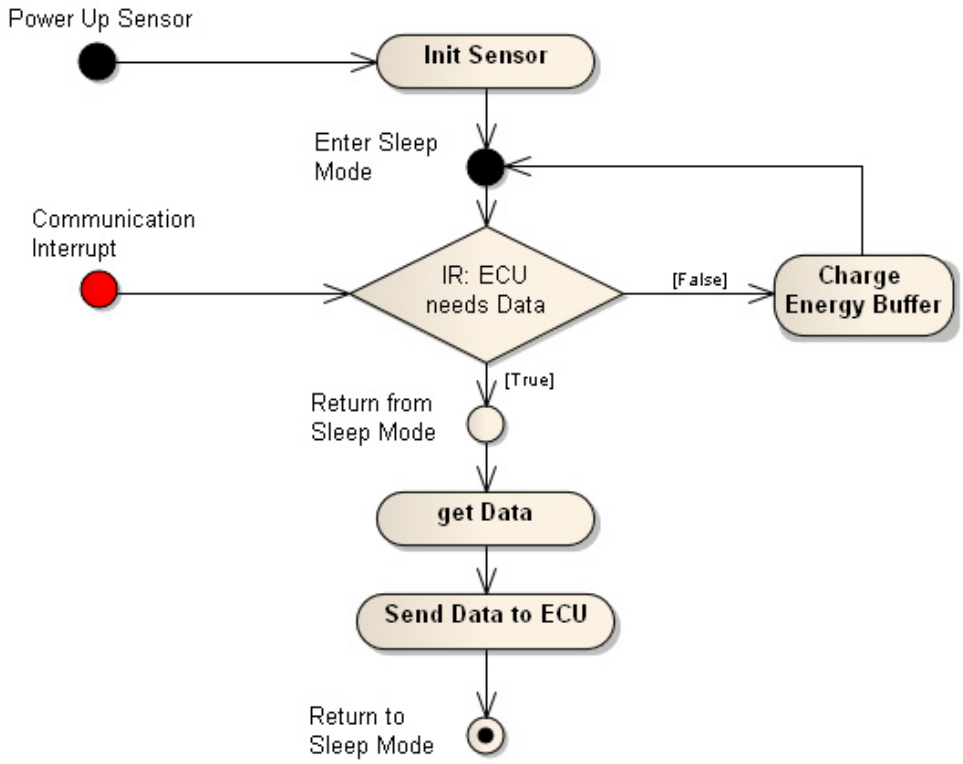


Abb. 2.7: *PIM: State Machine*

2.2.4 Structural Model

Im Structural Model wird die Struktur einzelner Elemente betrachtet. Dabei geht es darum, eine sinnvolle Unterteilung zu treffen um das Gesamtproblem in Teilprobleme zu unterteilen, die für sich implementiert und anschließend zusammengeführt werden.

2.2.4.1 Hardware

Das Modell in Abbildung 2.8 beschreibt die Hardware Struktur des Sensors. Die Kernelemente sind:

- das Interface,
- der Temperaturmesskanal (Intemp),
- der Einstrahlungsmesskanal (Insol),
- und ein Core, der die Messungen und die Kommunikation steuert.

Das Hardware Structural Model gibt einen Überblick über die Teile des Sensors, die für die Funktionalität notwendig sind. Dabei ist noch nicht festgelegt, wie z.B. die Temperaturmessung erfolgt. Es wurde festgelegt dass es einen Temperaturmesskanal am Sensor geben wird.

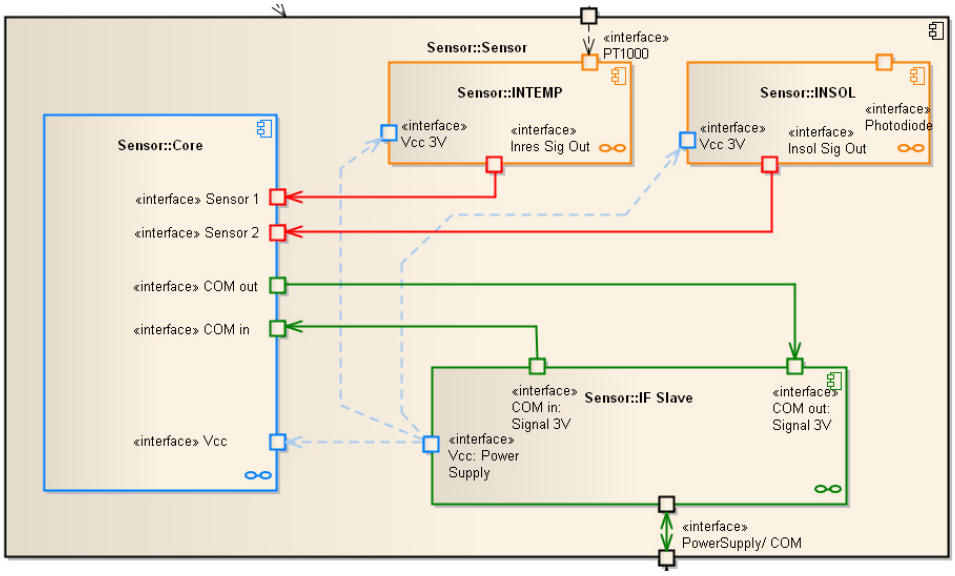


Abb. 2.8: *PIM: Structural Model Hardware*

2.2.4.2 Firmware

Aufbauend auf der Hardware Struktur wurde die Firmware Struktur erstellt. Die Steuerung wurde bereits durch die State Machine definiert. Um die Aktionen der State Machine zu realisieren, wurde die Firmware in Klassen unterteilt die sich wiederum in unterschiedlichen Abstaktionsebenen befinden, siehe Abbildung 2.9.

Die Struktur unterteilt sich in drei Gruppen:

- Der System Manager initialisiert den Sensor, er realisiert den State *„Init Sensor“* der State Machine in Abbildung 2.7.
- Das Epic *„Messwerte übertragen“* wird durch den *Communication Manager* und den *Interface Treiber* realisiert
- Das Epic *„Messung durchführen“* wird von der *Measurment Unit* und dem *Analog-Digital-Umsetzer Treiber* durchgeführt.

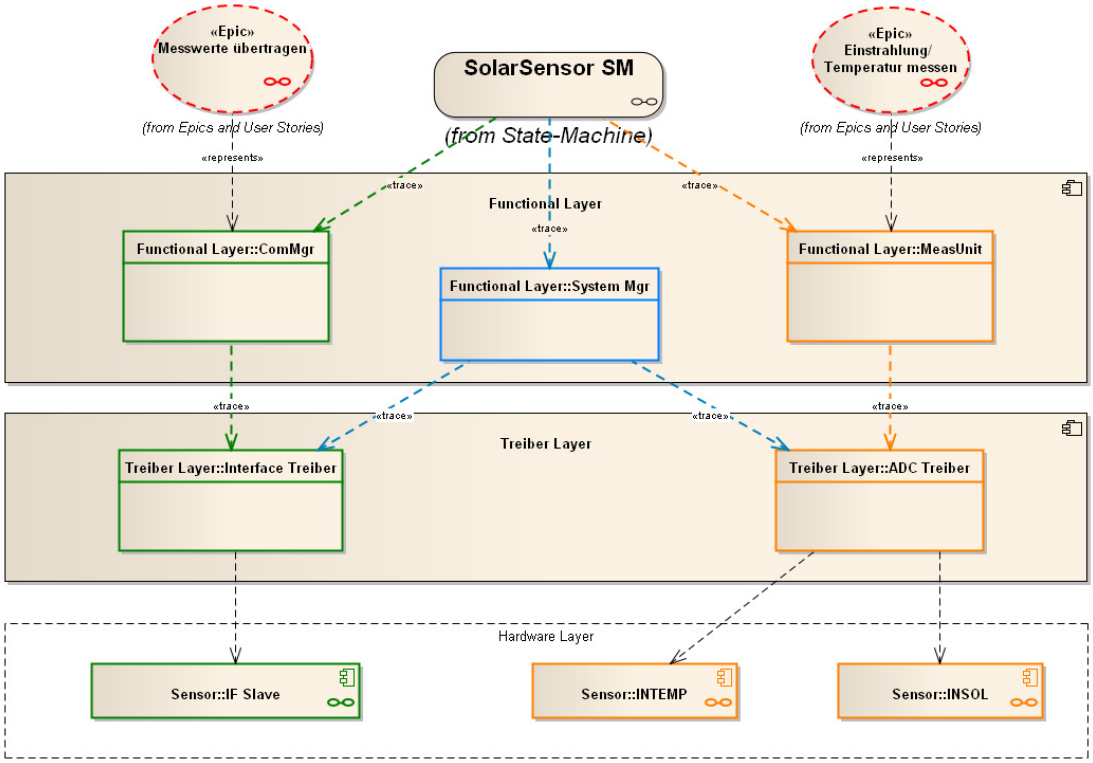


Abb. 2.9: *PIM: Structural Model Firmware*

2.2.5 Zusammenfassung

Durch das PIM wurden die Systemgrenzen definiert und mehrere Konzepte zur Realisierung abgeschätzt. Dabei wurde versucht eine möglichst allgemeine Sicht auf das System zu behalten. Welche Plattform im nächsten Schritt auch ausgewählt wird, eine Konvertierung der Messgrößen muss stattfinden und ein Treiber für die Ansteuerung des Konverters wird notwendig sein. Ob dabei ein Wägeverfahren oder Zählverfahren verwendet wird ist in dieser Phase des Designs nicht relevant. In der nächsten Design-Phase wurde der Sensor auf eine Plattform portiert, wo genau diese Punkte festgelegt wurden.

2.3 Platform–Specific–Model (PSM)

Das PSM beschreibt die Portierung des Systems auf eine Plattform. Dabei werden Berechnungen für das Interface und die Messkanäle durchgeführt und ein entsprechender Controller für den Sensor ausgewählt. Die Firmwarestruktur wird an den Controller angepasst und erweitert.

2.3.1 Hardware

Aufbauend auf dem Hardware Structural Model auf Seite 18 wurden die Messkanäle spezifiziert, ein Controller ausgewählt und das Interface sowie die Spannungsversorgung festgelegt.

2.3.1.1 Dimensionierung der Stromquelle

Für das Master Interface wurden zwei Stromquellen, eine NPN- und eine PNP-Stromquelle, verwendet. Eine genaue Beschreibung zu Stromquellen kann in Kapitel 4.1.1 in [15] nachgelesen werden.

$$\begin{aligned} I_q &\approx \frac{V_{SS}}{R_1 + R_2} \\ I_q R_2 &\approx I_a R_3 + V_{BE} \end{aligned} \qquad \Rightarrow I_a \approx \frac{1}{R_3}\left(\frac{V_{SS} \cdot R_2}{R_1 + R_2} - V_{BE}\right) \tag{2.2}$$

mit $V_{BE} \approx 0,7V$.

In Abbildung 2.10 ist die Schaltung einer PNP-Stromquelle abgebildet. Die Dimensionierung wurde folgendermaßen durchgeführt:

Die Versorgungsspannung wurde auf $V_{SS} = 9V$ festgelegt, der Ausgangsstrom der Stromquelle wurde auf $I_a \approx 50mA$ dimensioniert. Der Spannungsabfall am Widerstand R_2 wurde mit $V_{R2} = 1,5V$ angenommen.

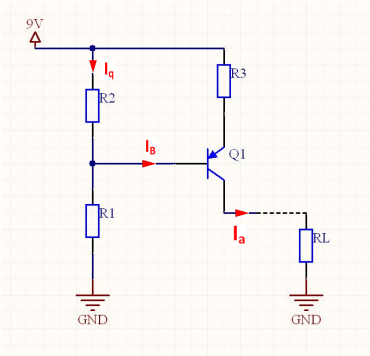


Abb. 2.10: *PNP Stromquelle*

Die Arbeitspunkteinstellung der Stromquelle erfolgt über den Spannungsteiler R_1 und R_2.

$$\frac{V_{R2}}{V_{SS}} = \frac{R_2}{R_1 + R_2} \tag{2.3}$$

Diese Widerstände bestimmen auch den Querstrom. Der Basisstrom des Transistors Q1 kann vernachlässigt werden $I_q >> I_B \approx 0$

$$I_q \approx \frac{V_{SS}}{R_1 + R_2} \tag{2.4}$$

Der Widerstand R_3 wird über die Maschengleichung

$$R_3 \cdot I_a = I_q \cdot R_2 + V_{BE} \tag{2.5}$$

berechnet.

Die Vorgehensweise zur Dimensionierung der NPN Stromquelle erfolgt gleich wie für die PNP Stromquelle. Der Unterschied liegt in der Versorgungsspannung des Spannungsteilers. Die NPN Stromquelle ist nur während einer Übertragung aktiv und wird mit 3,3V angesteuert.

2.3.1.2 Interface Master (ECU)

Abbildung 2.11 zeigt einen Überblick über den Aufbau des Interfaces. Die Versorgungsspannung von 12V wird über einen Spannungsregler auf die Betriebsspannung geregelt, die Stromquellen sind für die Spannungsversorgung und Kommunikation zuständig und werden über TX und 3V3 angesteuert. Zum Empfangen der Daten wird ein Schmitt-Trigger verwendet. Eine Schutzbeschaltung befindet sich am Ausgang des Interfaces.

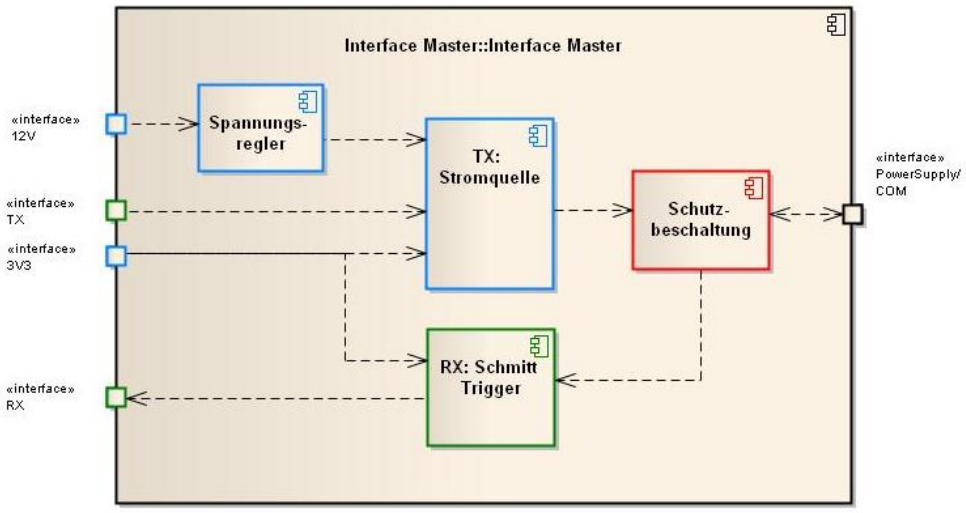


Abb. 2.11: *PSM: Master Interface*

Funktion: Findet keine Übertragung statt ($TX_{ECU} = 0V$) ist die PNP Stromquelle (Q2) in Abbildung 2.12 aktiv und versorgt den Sensor. Der Strom wird durch die gewählten Widerstandswerte auf $\approx 60mA$ begrenzt. Die NPN Stromquelle (Q3) ist deaktiviert. Die Funktionsweise während einer Datenübertragung ist:

TX_{ECU} auf 0V (Low):
0V entsprechen dem selben Zustand wenn keine Übertragung stattfindet. Das Potential der Zweidrahtleitung wird durch die Zenerspannung der Zenerdiode bestimmt.

TX_{ECU} auf 3,3V (High):
Bei einer logischen 1 wird die NPN Stromquelle eingeschaltet, die PNP Stromquelle ausgeschaltet und das Potential der Zweidrahtleitung auf Masse gezogen.

Geschaltet werden die beiden Stromquellen über den Transistor Q1 dessen Basispotential fest auf 3,3V liegt. Die Übertragung erfolgt über TX_{ECU}. Beträgt die Spannung am Pin TX_{ECU} 0V, leitet der Transistor Q1 und die PNP Stromquelle ist aktiv. Liegen 3,3V an TX_{ECU}, wird Q2 hochohmig, über den Spannungsteiler R_5 und R_6 wird der Arbeitspunkt eingestellt und die NPN Stromquelle aktiviert. Das Potential der Zweidrahtleitung wird dann auf Masse gezogen.
Mit den gewählten Widerstandswerten stellt sich ein Kollektorstrom des Transistors Q2 von $\approx 60mA$ ein. Die Zenerdiode D_2 und der Widerstand R_{LOAD} bilden die Last der Stromquelle. Als Ausgangsspannung V_{OUT} stellt sich die Zenerspannung von $\approx 7,5V$ ein.

Die Empfangseinheit (Abbildung 2.13) wurde mit dem Standard HEX Schmitt Trigger SN74HC14 zur Flankenregeneration und einem Spannungsteiler am Eingang realisiert. Der Spannungsteiler (R_1 und R_2) dient zur Anpassung der Pegel der Zweidrahtleitung an die

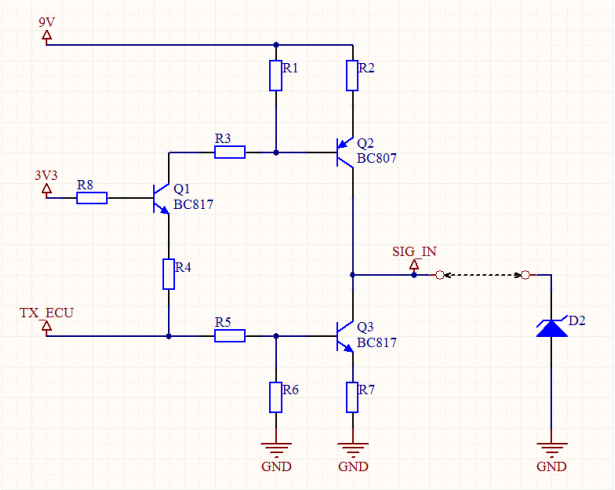


Abb. 2.12: *Sendeeinheit Master*

Eingangspegel des Schmitt Triggers.

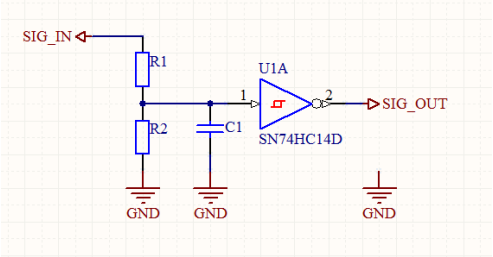


Abb. 2.13: *Empfangseinheit Master*

Schutzbeschaltung: Als Schutz vor ESD Pulsen wurde ein Varistor, eine Ferritperle zur Unterdrückung von hochfrequenten Störungen und Schottky–Dioden (BAR43S) zum Schutz der Transistoren am Ausgang des Interface geschalten.

Realisierung: Der Schaltplan der realisierten Schaltung befindet sich im Anhang auf Seite 68.

2.3.1.3 Interface Slave (Sensor)

Die Betriebsspannung des Sensors wurde auf 3V festgelegt. Einen Überblick liefert die Abbildung 2.14.

Power Supply: Der Slave stellt die Last der Stromquelle in Form der Zenerdiode D2 ($V_Z = 7,5V$) dar. Die Schottky-Diode D1 verhindert das Entladen des Kondensators C1 während einer Datenübertragung.

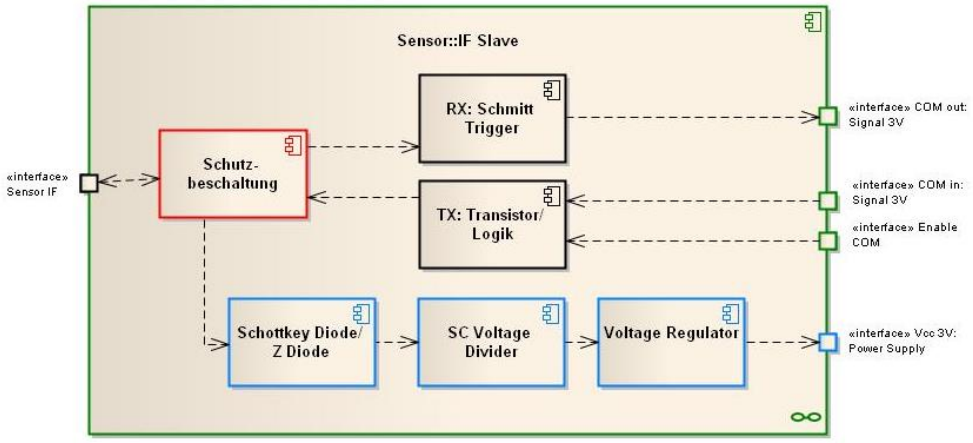


Abb. 2.14: *PSM: Slave Interface*

Für die Spannungsversorgung des Sensors stehen $\approx 7V$ zur Verfügung. Um die Differenz von 4V nicht als Verlust in Wärme umzuwandeln wird ein Switched Capacitor Voltage Converter (MAX660) als Spannungshalbierer eingesetzt.
Ein Low–Dropout Spannungsregler (LP3985) regelt die $\approx 3,5V$ auf die Spannung von $3V$, die für den Sensor als Betriebsspannung dient.

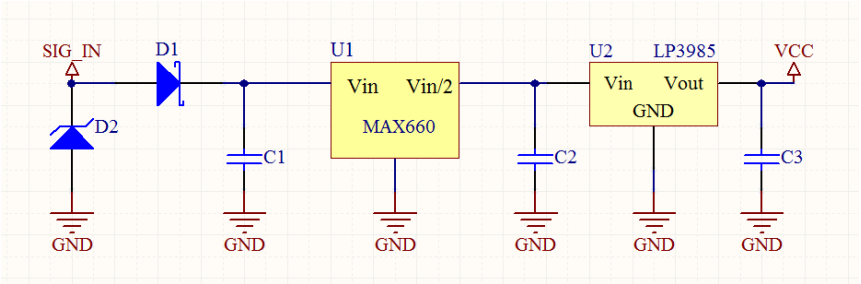


Abb. 2.15: *PSM: Power Supply Slave*

Startup Logik: Damit sich der Sensor beim Hochfahren nicht selbst die Spannungsversorgung nimmt, wurde eine Logikschaltung implementiert die ein explizites Freischalten der Sendeeinheit erfordert. Die verwendeten Logikgatter (SN74HC132) dienen gleichzeitig als Empfangseinheit.

Schutzbeschaltung: Ein Varistor bietet wie beim Master Schutz vor ESD Pulsen, eine Ferritperle Schutz vor hochfrequenten Störungen. Einen zusätzlichen Schutz bietet die Zenerdiode.

2.3.1.4 Messkanäle

Der Kanal Intemp dient zur Temperaturmessung, der Kanal Insol zur Einstrahlungserfassung.

Insol: Die Einstrahlung wird mit einer Photodiode, die in Sperrrichtung betrieben wird, erfasst. Der zur Einstrahlung porportionale Kurzschlussstrom der Photodiode wird mit einem Transimpedanzverstärker (Abbildung 2.17) in eine Spannung umgewandelt. Der Operationsverstärker muss dabei einen geringen Eingangsruhestrom besitzen, um den Kurzschlussstrom detektieren zu können und den Fehler gering zu halten. Dazu eignen sich OPVs mit FET Eingangsstufe.
Die Ausgangsspannung ergibt sich aus

$$V_a = I_K \cdot R_1 \tag{2.6}$$

Als Verstäker wurde der Rail-to-Rail Operationsverstärker TLV277x von Texas Instruments verwendet. Er besitzt einen Eingangsruhestrom (Input Bias Current) von 2pA.
Zur Messung der Einstrahlung wurde die Photo PIN Diode BPW34 verwendet. Sie besitzt einen Kurzschlussstrom von $70\mu A$ bei einer Beleuchtungsstärke von 1klx oder $4,7\mu A$ bei einer Bestrahlungsstärke von $1W/m^2$ und einer Wellenlänge von $\lambda = 960nm$. Bei dieser Wellenlänge besitzt die Photodiode die höchste Empfindlichkeit, ersichtlich in Abbildung 2.16. Diese Photodiode wurde deshalb gewählt, weil sie eine breite spektrale Empfindlichkeit aufweist, kostengünstig und leicht verfügbar ist.

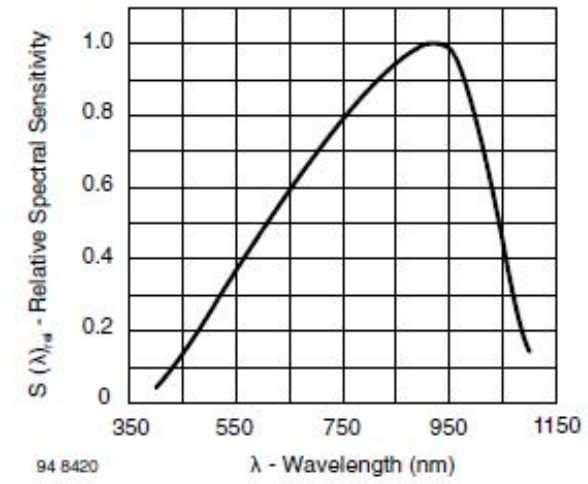


Abb. 2.16: *Spektrale Empfindlichkeit BPW34*

Um die Strahlungsstärke von $0 \ldots 1000W/m^2$ detektieren zu können ist bei einer Betriebsspannung von $3V$ ein Widerstand von 660Ω notwendig. Der in der Messschaltung eingesetzte Widerstandswert wurde durch empirische Messungen mit 1500Ω festgelegt.

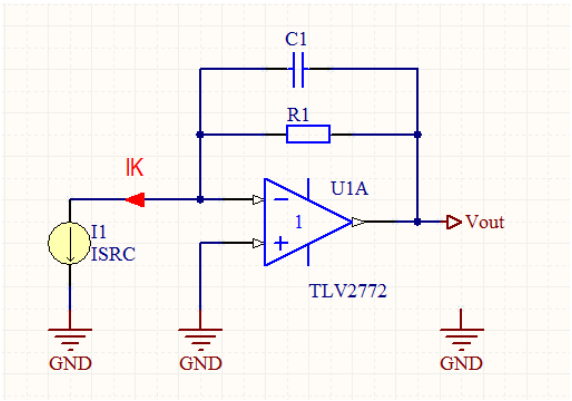


Abb. 2.17: *Transimpedanzverstärker*

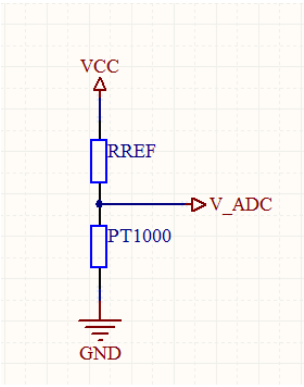


Abb. 2.18: *Ratiometrische Messung*

Intemp: Die Temperatur soll im Bereich von -20°C bis +150°C mit einer Genauigkeit von 0,5K detektiert werden. Dies ist die Anforderung des Temperaturkanals der ECU

Die Messung erfolgt ratiometrisch über das Widerstandsverhältnis von Referenzwiderstand zu PT1000 Temperaturfühler (Abbildung 2.18.

Berechnung: Die Spannungsversorgung beträgt 3V. Zu Beginn wird die notwendige Auflösung des ADC's bestimmt, um eine Genauigkeit von 0,5°C zu erzielen.
Die Berechnung wird um 0°C durchgeführt. Die Ergebnisse für den oberen und unteren Bereich sind in der Tabelle 2.2 aufgelistet. Der Widerstandswert eines PT1000 Fühlers ändert sich mit $\approx 4\Omega/°K$.

$$V_{s|_{0°}} = V_{ref} \cdot \frac{R_s}{R_s + R_{ref}} = 3V \cdot \frac{1000\Omega}{1000\Omega + 1200\Omega} = 1,3636V \tag{2.7}$$

$$V_{s|_{0,5°}} = V_{ref} \cdot \frac{R_s}{R_s + R_{ref}} = 3V \cdot \frac{1002\Omega}{1002\Omega + 1200\Omega} = 1,3664V \tag{2.8}$$

$$\Delta V_{s|_{\Delta\vartheta=0,5°K}} = V_{s|_{0,5°}} - V_{s|_{0,0°}} = 2,7272mV \tag{2.9}$$

ADC–Auflösung: Berechnung des Spannungshubes

$$\Delta V_{hub} = V_{s|_{150,5°}} - V_{s|_{-20°}} = 0,4009V \tag{2.10}$$

Berechnung der Stufenzahl für die geringste Spannungsänderung. Diese befindet sich im oberen Temperaturbereich.

$$n = \frac{\Delta V_{hub}}{\Delta V_s} = \frac{0,4009V}{2,1635mV} = 185,29 \tag{2.11}$$

$Rs@0°C[\Omega]$	$Vs@0°C[V]$	$Vs@0,5°C[V]$	$\Delta Vs[mV]$
1000	1,3636		
1002		1,3664	2,7272
$Rs@-20°C[\Omega]$	$Vs@-20°C[V]$	$Vs@-19,5°C[V]$	$\Delta Vs[mV]$
921,6	1,3032		
923,6		1,3060	2,8281
$Rs@150°C[\Omega]$	$Vs@150°C[V]$	$Vs@150,5°C[V]$	$\Delta Vs[mV]$
1573,25	1,7019		
1575,25		1,7040	2,1635

Tabelle 2.2: *Inres Berechnung*

$$N = ld\left(\frac{\Delta V_{hub}}{\Delta V_s}\right) = ld\left(\frac{0,4009V}{2,1635mV}\right) = 7,53\hat{=}8bit \tag{2.12}$$

Da es keinen ADC für einen Messbereich von 0,4V gibt, und der ADC nicht nur für den Intemp Kanal sondern auch für den Insol Kanal verwendet wird, wird die Berechnung der Stufenzahl auf die 3V Referenzspannung bezogen.
Berechnung der Auflösung

$$N = ld\left(\frac{V_{ref}}{\Delta V_s}\right) = ld\left(\frac{3V}{2,1635mV}\right) = 10,43\hat{=}11bit \tag{2.13}$$

Mit einem 12Bit ADC kann auf

$$\Delta V_s = \frac{V_{ref}}{2^{12}} = \frac{3V}{4096} = 732,42\mu V \tag{2.14}$$

aufgelöst werden, das entspricht einer Genauigkeit von 0,2K.

Ratiometrische Messung: Damit Schwankungen der Versorgungsspannung keine Auswirkungen auf das Messergebnis haben wird die Messung ratiometrisch durchgeführt.

$$N_{ADC} = 2^{12}\frac{V_s}{V_{ref}} \tag{2.15}$$

Durch einsetzen von Gleichung 2.7 in 2.15 erhält man

$$R_s = \frac{N_{ADC} \cdot R_{ref}}{2^{12} - N_{ADC}} \tag{2.16}$$

Realisierung: Der Intemp Kanal besteht aus einem Spannungsteiler der aus einem Referenzwiderstand R_{ref} und dem Pt1000 Fühler gebildet wird. Zusätzlich ist noch ein Tiefpass Filter ($f_g \approx 16Hz$) nachgeschalten. Als Referenzwiderstand wurde ein Widerstand mit einer Genauigkeit von 0,1% gewählt. Als Schutz gegen ESD-Pulse ist ein Varistor (12V) am Eingang des Kanals geschalten, der im Falle einer Überspannung diese gegen Masse ableitet.

2.3.1.5 Controller

Bei der Auswahl des Controllers fiel die Wahl auf die MSP-430 Serie von Texas Instruments. Dabei handelt es sich um eine leistungsfähige Reihe von 16-Bit RISC basierenden Mixed–Signal Prozessoren speziell für Low-Power Anwendungen. Auswahlkriterien der MSP430 Familie waren der integrierte 12Bit A/D-Wandler, der geringe Leistungsverbrauch und die Möglichkkeit ein WSN (ZigBee) mit dieser Controllerfamilie aufzubauen.
Eine einfache In-Circuit-Programmierung und In-Circuit-Debugging Möglichkeit ist mit dem Target Interface MSP-FET430UIF gewährleistet.
Die Abbildung 2.19 zeigt das Blockschaltbild des verwendeten Controllers.

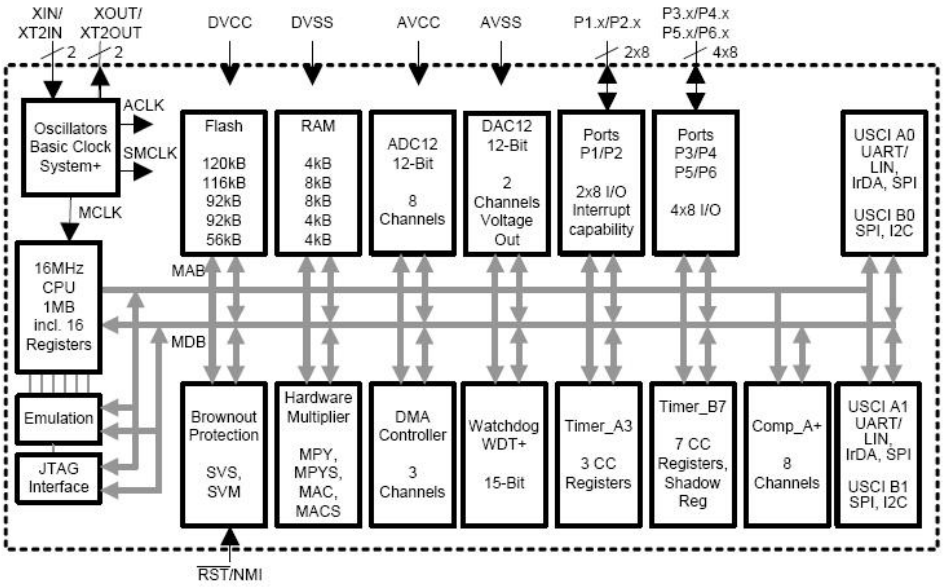


Abb. 2.19: *MSP430 F2618 Blockschaltbild*

Eckdaten des MSP430 F2618

- 8kB RAM
- 116kb Flash Speicher
- 8 Channel, 12-Bit A/D Wandler
- 2 x USCI Module
- 2 x 16–Bit Timer
- Vcc = 1,8 ... 3,6V

Eine genaue Beschreibung des Controllers kann im „Family-User's Guide“ [14], die Technischen Daten im [13] nachgelesen werden. Die Ansteuerung des Master- und Slave–Interfaces

erfolgt mit UART (Universal Asynchronous Receiver Transmitter). Grund dafür ist die gute Verfügbarkeit auf anderen Controllern und die bereits vorhandene Schnittstelle auf der ECU.

Verwendete Peripherie und ihre Konfiguration:

ADC12 - Analog to Digital Converter

- SMCLK/5 = 200kHz
- $V_{REF} = V_{CC}$
- Single-Channel, Single-Conversion
- Sample-and-Hold Source: ADC12SC Bit
- Pulse-Sample-Mode

USCIA0 - Uart Interface

- SMCLK = 1MHz
- Interrupt Mode
- 8Bit Data, LSB First, One Stop Bit
- Oversampling Mode, 9600Baud

WDT+ - Watchdog Timer

- Watchdog Mode
- ACLK/8192 = 0,7Hz
- Reset Function

SVS - Supply Voltage Supervisor

- Voltage Level Detection: 2,2V
- Power On Reset (POR)

Port Mapping: Abbildung 2.20 gibt einen Überblick über die Zuweisung der Messkanäle und des Interfaces auf den Controller. Im ersten Prototypen wurden zwei Kanäle zur Messung der Einstrahlung und ein Kanal zur Messung der Temperatur implementiert. Weiters wurden zwei weitere Pins des ADC´s für debug Zwecke vorgesehen.

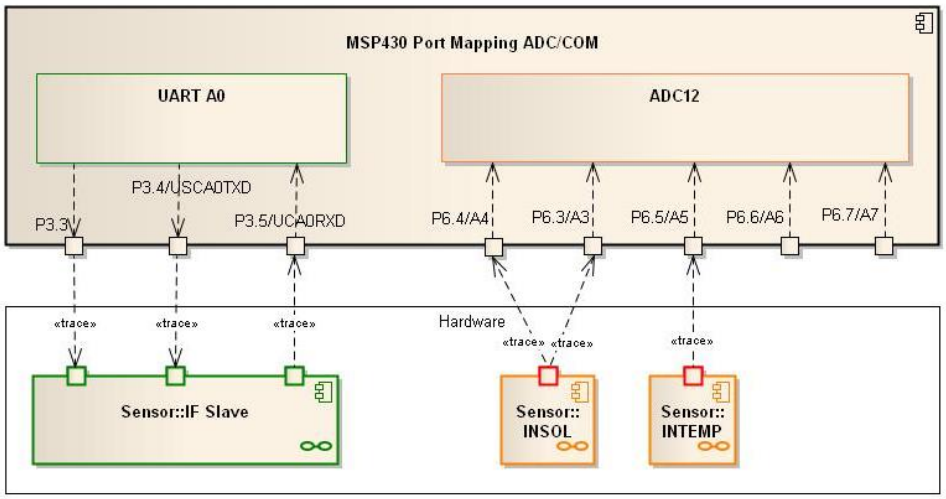


Abb. 2.20: *Sensor Port Mapping*

Clock Verteilung: Das „Basic Clock Module“ des MSP430 besitzt zwei interne und zwei optionale externe Oszillatoren. Für den Sensor werden der „Digital Controlled Oscillator (DCO)“ und der „Very Low Power/Low Frequency Oscillator (LP/LF)“ verwendet. Von diesen Ozillatoren sind drei Clock Signale verfügbar: ACLK, MCLK und SMCLK
In den unterschiedlichen Low Power Modes des Controlles sind nur bestimmte Clocksignale aktiv um die Stromaufnahme zu senken. In der Tabelle 2.3 sind abhängig von den Stromsparmodi die Clocksignale aufgelistet.
In Abbildung 2.21 ist die zugewiesene Clockverteilung auf die Peripherie des Controllers abgebildet. Der DCO wird auf 1MHz konfiguriert, der LP/LF Oszillator besitzt eine feste Frequenz von $\approx 12kHz$. Dadurch, dass keine externen Oszillatoren verwendet wurden und SMCLK für die Taktung von USCIA0 zuständig ist, konnte maximal „Low Power Mode 1 (LPM1)“ verwendet werden.

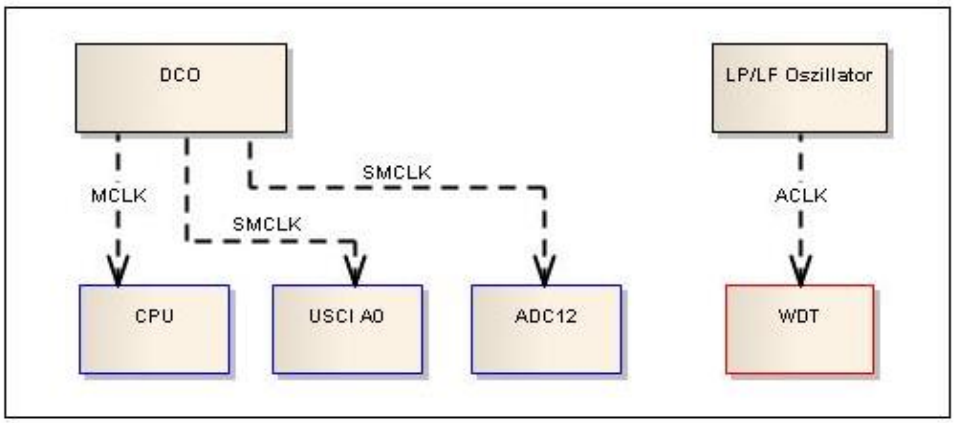


Abb. 2.21: *Sensor Clock Verteilung*

SCG1	SCG0	OSCOFF	CPUOFF	Mode	CPU and Clocks Status
0	0	0	0	Active	CPU is active, all clocks are active
0	0	0	1	LPM0	CPU, MCLK are disabled SMCLK, ACLK are active
0	1	0	1	LPM1	CPU, MCLK are disabled, DCO and DC generator are disabled if the DCO is not used for SMCLK ACLK is active
1	0	0	1	LPM2	CPU, MCLK, SMCLK, DCO are disabled DC generator remains enabled ACLK is active
1	1	0	1	LPM3	CPU, MCLK, SMCLK, DCO are disabled DC generator disabled ACLK is active
1	1	1	1	LPM4	CPU and all clocks disabled

Tabelle 2.3: *MSP430 CPU and Clocks Status*

2.3.1.6 Gehäuse

Als Gehäuse wurde ein Standardgehäuse mit einem transparenten Deckel gewählt.

Kenndaten Spelsberg TK PC 99

- IP66 nach EN 60529
- Temperaturbereich $-35°C \ldots +80°C$
- UV- beständig
- Witterungsbeständig

2.3.2 Firmware

Die im Firmware Structural Model erstellte Struktur auf Seite 19 wurde an die Plattform angepasst und erweitert.
Das Modell verwendet die Struktur des „Cyan Structured Software Model (CSSM)“ das in [2] beschrieben ist. Die Unterteilung des Codes erfolgt in vier Schichten. Ziel der Unterteilung ist die Wartbarkeit und Wiederverwendbarkeit von Embedded Software zu erhöhen. Das Modell unterteilt Code in die Layer

- Application Layer
  Im Application Layer befindet sich die State Machine des Systems.

- Functional Layer
  Dieser Layer nutzt die Schnittstellen die vom Technology Layer bereitgestellt werden.
- Technology Layer
  Der Technology Layer bietet Funktionen die auf einer speziellen Technologie beruhen.
- Resource Layer
  Der Resource Layer stellt ein Application Programming Interface (API) zur Peripherie des Controllers wie Timer und General Purpose Inputs/Outputs dar.

Der Vorteil dieses Modelles liegt in der Entkopplung des Controllers durch den Resource Layer. Soll das System auf einen anderen Controller portiert werden muss nur der Resource Layer an die neue Plattform angepasst werden, die darüber liegenden Layer können wiederverwendet werden.
Um dies zu erreichen dürfen Funktionen einer höheren Ebene nur auf Funktionen der darunter liegenden Ebenen zugreifen, da sonst die Kapselung und somit die Flexibilität verloren gehen. Die Programmierung erfolgte in C++.
In Abbildung 2.22 ist die Unterteilung der Firmware in ihre Ebenen, mit den enthaltenen Klassen, dargestellt.

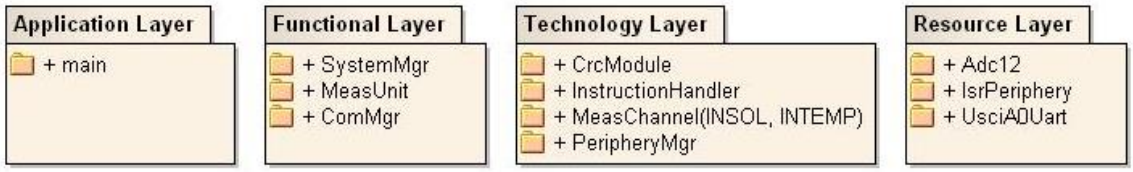


Abb. 2.22: *PSM: Schichtenmodell nach CSSM*

2.3.2.1 Application Layer

Im Application Layer befindet sich die `main()` Routine. Der Ablauf der `main()` Routine wurde durch die *State Machine* auf Seite 17 im PIM definiert. Eine Auflistung der einzelnen Funktionen und Methoden der einzelnen Klassen befindet sich in Kapitel 2.4.2.1.

- main:
  Nach dem Hochfahren werden die Klassen instanziiert und der Sensor wird konfiguriert. Die Funktion der Konvertierung des Datentyps `float` nach dem IEEE 754 Standard befindet sich in der `main` Datei.

2.3.2.2 Functional Layer

- SystemMgr:
  Der System Manager wird beim Hochfahren des Sensors aufgerufen. Hier wird der Sensor nach dem initialisieren gestartet und der Stromsparmodus (Low Power Mode) aktiviert.

- ComMgr:
  Gültige empfangene Datenwörter werden in ihre Einzelteile zerlegt (dekodiert). Die Messwerte werden zu Datenwörtern codiert.

- MeasUnit:
  Abhängig von der Pinbelegung des Sensors werden beim Initialisieren die entsprechenden Messkanäle angelegt. Die Verwaltung der Messungen führt ebenso die Measurement Unit durch

2.3.2.3 Technology Layer

- InstructionHandler
  Der Instruction Handler führt den „Cycle Redundancy Check (CRC)“ und die Überprüfung des *SyncBytes* durch. Daten die übertragen werden wird das *SyncByte* angefügt.

- Crc8
  Hier wird der CRC durchgeführt. Dabei wurde die Berechnung und das Polynom von Dallas 1–Wire übernommen. Die Berechnung des CRC Wertes erfolgt mit einer „Look Up Table“. Eine genaue Beschreibung ist in [9] zu finden.

- PeripheryMgr
  Der Periphery Manager ruft die `init()` Funktionen der Treiber im „Ressources Layer“ auf.

- Measurement Channels
  Die einzelnen Messkanäle unterscheiden sich nur in ihrer Umrechnung. Als Ausgangsbasis dient jedem Kanal das Wandlungsergebnis eines bestimmten ADC Kanals.

  1. Berechnung Insol (Einstrahlungskanal):
     Hier erfolgt die Umrechnung des Wandlungswertes in die Einstrahlung $[W/m^2]$.

     $$V_{IN}[V] = \frac{ADC_{Value} \cdot V_{CC}}{ADC_{FullRange} - 1} \tag{2.17}$$

     $$E[W/m^2] = \frac{ADC_{Value}}{I_K \cdot R_F} \cdot FKT_{Case} \tag{2.18}$$

  2. Berechnung Intemp (Temperaturkanal):
     Hier erfolgt die Umrechnung des Wandlungswertes in die Temperatur $[°C]$.

     $$R_{PT1000}[\Omega] = \frac{R_{Ref} \cdot ADC_{Value}}{ADC_{FullRange} - 1} \tag{2.19}$$

     $$Temp[°C] = \frac{R_{PT1000} - R_0}{coef \cdot R_0} \tag{2.20}$$

3. Berechnung Involt (Spannungskanal):
   Dieser Kanal wurde für debug Zwecke erstellt. Es erfolgt die Umrechnung des Wandlungswertes in die Spannung $[V]$

$$V_{IN}[V] = \frac{ADC_{Value} \cdot V_{CC}}{ADC_{FullRange} - 1} \tag{2.21}$$

2.3.2.4 Ressources Layer

Die gewählten Einstellungen der Peripherie des Controllers sind in Kapitel 2.3.1.5 auf Seite 29 aufgelistet.

- UsciA0Uart
  Die Konfiguration der Uart Schnittstelle und die Initialisierung des Empfangs- und Sendebuffers wird durchgeführt

- Adc12
  Die Konfiguration des Analog Digital Wandlers wird vorgenommen. Da es sich um keine zeitkritsche Anwendung handelt werden mehrere Wandlungen hintereinander durchgeführt und der arithmetische Mittelwert aus den Wandlungsergebnissen gebildet.

- IsrPeriphery
  Die Aktivierung und Konfiguration folgender Peripherie wird durchgführt:

  1. Aktivierung des Watchdog Timer´s
  2. Kalibrierung des Digital Controlled Oszillator (DCO)
  3. Konfiguration der Spannungsüberwachung (SVS)
  4. Der Software Interrupt der Kommunikation wird festgelegt
  5. Verwaltung der Low Power Modes (LPMx)

  Die Interrupt Service Routinen befinden sich ebenfalls hier.

Konstanten und Systemparameter sind in einer externen Datei gespeichert.

2.4 Platform–Specific–Implementation (PSI)

Die PSI beinhaltet demäß der Object Management Group (OMG) den Source Code, der aus dem PSM erzeugt wird. Bei dieser Arbeit dienen die Modelle des Platform–Independent–Model der Dokumentation und geben einen Überblick über die Funktionalität der einzelnen Klassen.
Die Implementierung der Hardware wurde mit Altium Designer Summer 09 durchgeführt. Altium Designer verbindet das Erstellen der Schaltpläne, die Simulation der Schaltung

und das Erstellen des Layouts in einem Programm. Die Möglichkeit beim PCB–Layout eine 3D–Ansicht der Platine zu erstellen vereinfacht die Verbindung der Elektronik mit dem Gehäuse. Auf Seite 63 ist das 3D–Modell der Sensorplatine abgebildet.

2.4.1 Firmware

Die Programmierung der Firmware erfolgte mit *IAR Embedded Workbench*. Zur Modellierung der Klassendiagramme wurde *Enterprise Architect 8.0 (EA)* von *Sparx Systems* verwendet. Mit Hilfe des Codegenerators von EA wurden die Klassen- und Funktionsheader erzeugt. Mit Hilfe von Round Trip Engineerings wurde die Konsistenz zwischen dem implementierten Code im *Embedded Workbench* und den Klassendiagrammen im *Enterprise Architect* hergestellt. In Abbildung 2.23 sind die Ebenen (Layer) und die einzelnen Klassen mit ihren Funktionen, Methoden und Variablen abgebildet.

2.4.2 Klassendiagramm

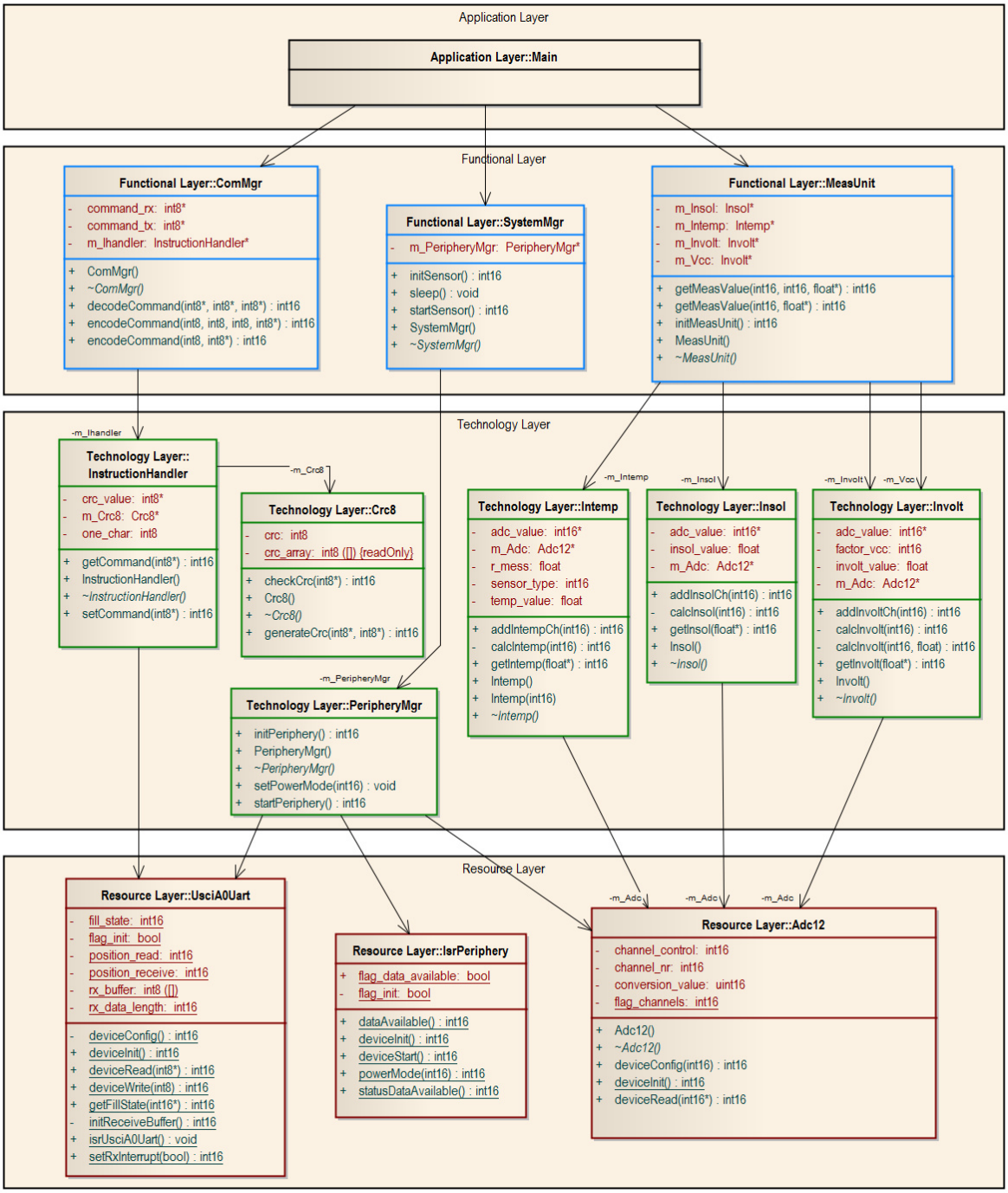


Abb. 2.23: *PSI: FW Klassendiagramm*

2.4.2.1 Firmware Funktionen und Methoden

Resource Layer:

UsciA0–Uart Mode

Method	Notes	Parameters
Static deviceConfig() int16 Private	Configuration USCI A0 - Uart Mode Not in use	
Static deviceInit() int16 Public	Initialize USCI A0 - Uart Mode Set Baudrate, PortPins, Enable Interrupt @param: - @return: OK/Init failed	
Static deviceRead() int16 Public	get one Byte from the RX Buffer @param: Byte Pointer to get and save one Byte @return: OK/No Byte available	int8* [in] _rx_byte
Static deviceWrite() int16 Public	Send one Byte through the UART @param: Byte to send @return: OK	int8 [in] _tx_byte
Static getFillState() int16 Public	Fill state of the receive Buffer @param: int16 Pointer to get and save the Fill state @return: OK/No Byte available	int16* [in] _fill_state
Static initReceiveBuffer() int16 Private	Initialize Receive Buffer Buffer size defined in the config file, Buffer is initialized to 0x00 @param: - @return: OK	
Static setRxInterrupt() int16 Public	Enable/disable Uart Receive IR @param: enable/disable (TRUE/FALSE) receive interrupt @return: OK/Set IR failed	bool [in] _ir_state

Tabelle 2.4: *UsciA0Uart Funktionen*

Analog Digital Converter (Adc12)

Method	Notes	Parameters
Adc12() Public	Constructor @param: - @return: -	
~Adc12() Public	Destructor @param: - @return: -	
deviceConfig() int16 Public	Configuration of the ADC Channels Not all channels are implemented, only requiered channels for the Sensor @param: Pysical ADC Channel of the Controller @return: OK/failed	int16 [in] _channel
Static deviceInit() int16 Public	Init function ADC12 static, configuration of the common Register @param: - @return: OK/failed	
deviceRead() int16 Public	returns an average of ADC conversions @param: 16bit Pointer for the conversion result @return: OK	int16* [in] _value

Tabelle 2.5: *Adc12 Funktionen*

IsrPeriphery

Method	Notes	Parameters
Static deviceInit() int16 Public	Init ISR and Periphery calibration of the DCO, activate LowPower/LowFrequ Oszillator, Enable SVS enable Software Interrupt @param: - @return: OK/Failed	
Static deviceStart() int16 Public	Start the WDT and enables the Communication, General Interrupt enable @param: - @return: OK	
Static powerMode() int16 Public	Low Power Modes The Controller is going in the selected Low Power Mode @param: Low Power Mode, defined in the Config File @return: -	int16 [in] _mode
Static statusDataAvailable() int16 Public	Data available @param: - @return: OK/No data available	

Tabelle 2.6: *IsrPeriphery Funktionen*

Technology Layer:

Cycle Redundancy Check (Crc8)

Method	Notes	Parameters
checkCrc() int16 Public	checks if the CRC sum is right @param: address pointer of the data array to check the CRC @return: OK/ failed	int8* [in] _data_to_check
Crc8() Public	Constructor @param: -	
Crc8() Public	Destructor	
generateCrc() int16 Public	generates 8 Bit CRC check sum @param: Pointer for the generated CRC Value, address pointer of the data to generate the CRC @return: OK/ failed	int8* [in] _crc_value int8* [in] _data_to_check

Tabelle 2.7: *Crc8 Funktionen*

Insol Channel

Method	Notes	Parameters
addInsolCh() int16 Public	Init Insolation Channel @param: Insol channel number (possible channels defined in the config file) @return: OK/ failed	int16 [in] _ch_nr
calcInsol() int16 Private	Insolation Calculation @param: adc_value @return: OK	int16 [in] _adc_value
getInsol() int16 Public	INSOL Function @param: float pointer for the insolation value @return: OK	float* [in] _insol_value
Insol() Public	Constructor @param: -	
˜Insol() Public	Destructor	

Tabelle 2.8: *Insol Funktionen*

Involt Channel

Method	Notes	Parameters
addInvoltCh() int16 Public	Init Voltage Channel @param: Involt channel number (possible channels defined in the config file) @return: OK/ failed	int16 [in] _ch_nr
calcInvolt() int16 Private	Voltage Calculation from adc result VCC is defined in the config file @param: adc_value @return: OK	int16 [in] _adc_value
calcInvolt() int16 Private	Voltage Calculation from internal VCC calculation @param: adc_value, vcc_value @return: OK	int16 [in] _adc_value float [in] _vcc_value
getInvolt() int16 Public	INVOLT Function @param: float pointer for the voltage value @return: OK/ failed	float* [in] _involt_value
Involt() Public	Constructor @param: -	
˜Involt() Public	Destructor	

Tabelle 2.9: *Involt Funktionen*

Intemp Channel

Method	Notes	Parameters
addIntempCh() int16 Public	Init Temperatur Channel @param: Intemp channel number (possible channels defined in the config file) @return: OK/ failed	int16 [in] _ch_nr
calcIntemp() int16 Private	Temperature Calculation PT1000 and KTY depending on sensor_type Variable (default PT1000) @param: adc_value @return: OK	int16 [in] _adc_value
getIntemp() int16 Public	INTEMP Function @param: float pointer for the temperature value @return: OK	float* [in] _temp_value
Intemp() Public	Default Constructor @param: -	
Intemp() Public	Constructor Sensor type, not yet coded @param: sensor type PT1000/KTY	int16 [in] _sensor_type
˜Intemp() Public	Destructor	

Tabelle 2.10: *Intemp Funktionen*

Instruction Handler

Method	Notes	Parameters
getCommand() int16 Public	Get one Command (7 Byte) @param: address pointer of the command word memory to save the command @return: OK/ wrong CRC / No Command available	int8* [in] _get_com_word
InstructionHandler() Public	Constructor	
~InstructionHandler() Public	Destructor	
setCommand() int16 Public	Send one Command (SyncByte + 7 Byte + 1 Byte CRC) @param: address pointer of the data to build one command @return: OK/	int8* [in] _set_com_word

Tabelle 2.11: *Instruction Handler Funktionen*

Periphery Manager

Method	Notes	Parameters
initPeriphery() int16 Public	Init Uart, General Interrupt, Software Interrupt Flag, ADC12, SVS,... Last Instruction is General Interrupt Enable, ready to receive data @param: - @return: OK/ failed	
PeripheryMgr() Public	Constructor @param: -	
PeripheryMgr() Public	Destructor	
setPowerMode() void Public	enter Low Power Modes @param: Power Mode, defined in config file @return: -	int16 [in] _mode
startPeriphery() int16 Public	Function to start the device, all periphery is configured @param: - @return: OK	

Tabelle 2.12: *Periphery Manager Funktionen*

Functional Layer:

System Manager

Method	Notes	Parameters
initSensor() int16 Public	Initialize the Sensor Create instance of PeripheryMgr @param: - @return: OK/error/init failed	
sleep() void Public	Send Sensor to sleep Mode @param: - @return: -	
startSensor() int16 Public	Start the sensor @param: - @return: OK/error/init failed	
SystemMgr() Public	Constructor @param: -	
SystemMgr() Public	Destructor	

Tabelle 2.13: *System Manager Funktionen*

Communication Unit

Method	Notes	Parameters
ComMgr() Public	Constructor @param: -	
ComMgr() Public	Destructor	
decodeCommand() int16 Public	decode one Command @param: pointer of communication_ID, pointer of channel_type, pointer of channel_number @return: OK / Error	int8* [in] _com_id int8* [in] _ch_type int8* [in] _ch_nr
encodeCommand() int16 Public	Send Value Encoder @param: communication_ID, channel_type, channel_number, address pointer of the data @return: OK / Error	int8 [in] _com_id int8 [in] _channel int8 [in] _nr int8* [in] _data
encodeCommand() int16 Public	Send ID Number Encoder @param: communication_ID, address pointer of the device ID @return: OK	int8 [in] _com_id int8* [in] _data

Tabelle 2.14: *Communication Manager Funktionen*

Measurement Unit

Method	Notes	Parameters
getMeasValue() int16 Public	Get one measured value @param: channel type, channel number, float pointer to save the result @return: OK / Error / No_value	int16 [in] _ch_type int16 [in] _ch_nr float* [in] _measurement
getMeasValue() int16 Public	Get one measured value for the internal Vcc channel @param: channel type, float pointer to save the result @return: OK / Error / No_value	int16 [in] _ch_type float* [in] _measurement
initMeasUnit() int16 Public	Init the Measurement Unit @param: -	
MeasUnit() Public	Constructor Creates the Instances of the Measurement Channels The number of the specific channels are saved in the config file @param: -	
MeasUnit() Public	Destructor	

Tabelle 2.15: *Measurement Unit Funktionen*

Kapitel 3

Ergebnisse

Die Funktionalität des Sensors wurde mit verschiedenen Messungen validiert.

3.1 Interface

Die Messungen wurden unter Raumtemperatur durchgeführt. Als WSN Simulation wurde ein 82Ω Widerstand verwendet. Die im folgenden verwendeten Bezeichnungen beziehen sich auf die Schaltpläne die sich im Anhang (S.63ff) befinden.

3.1.1 Power Supply

Stromaufnahme: Die Messungen am Master Interface wurden einmal mit und einmal ohne angeschlossenem Sensor durchgeführt.

Die Ausgangsspannung wurde an der Klemme `SS1`, die Stromaufnahme über `JP0`. Die Messung wurde mit zwei Multimetern durchgeführt.

Master Interface ohne Sensor
$V_{OUT} = 8,96V$ ... Spannung an den Klemmen
$I_0 = 4,88mA$ ... Stromaufnahme des Interfaces ohne Sensor

Master Interface mit Sensor
$V_{OUT} = 7,54V$ ... Spannung an den Klemmen
$I_0 = 57,6mA$ ... Stromaufnahme des Interfaces mit Sensor

Stromaufnahme des Sensors
Die Stromaufnahme wurde über `JP0` mit einem Multimeter gemessen. Zur Simulation eines WSN wurde die Versorgungsspannnung mit einem 82Ω Widerstand belastet, angeschlossen an `VCC1`.

$I_{Sensor} = 1,74mA$ ... Stromaufnahme über `JP0` des Sensors
$I_{Sensor+82\Omega} = 20mA$ ... Stromaufnahme des Sensors über `JP0` mit WSN Widerstand

Spannungsverlauf: Die folgenden Messungen zeigen die Versorgungsspannung des Sensors bei unterschiedlichen Belastungen. Da die Spannungsüberwachung (SVS) des MSP430 aktiviert wurde, darf die Versorgungsspannung 2,2V nicht unterschreiten da der Sensor sonst einen Power On Reset (POR) durchführt.

Messaufbau: Am Master Interface `TX_ECU` wurde ein Funktionsgenerator angeschlossen und ein Signal ($V_{TX\_ECU} = 3,3V$) mit verschiedenen Puls–Pausen–Verhältnissen eingestellt. Die Oszilloskopbilder zeigen den Spannungsverlauf der Zweidrahtleitung `TL` und die Versorgungspannung des Sensors gemessen an `VCC1`.

Abbildung 3.1 zeigt dass die Spannungsversorgung nach $\approx 300ms$ einzubrechen beginnt.

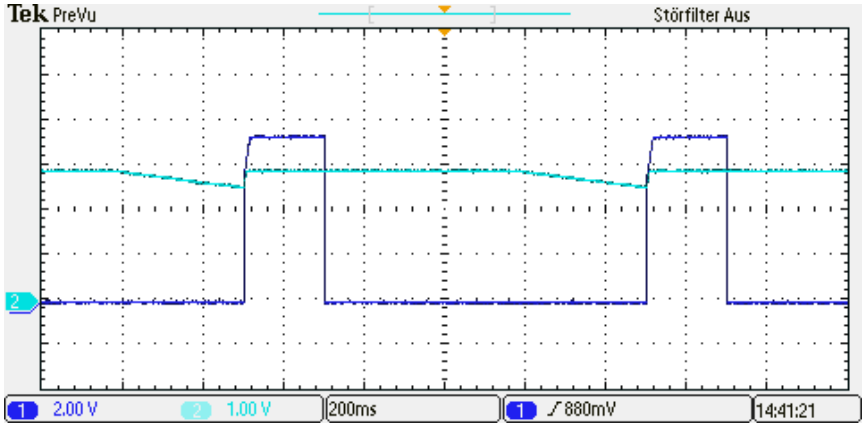


Abb. 3.1: *Last: Sensor*

Einstellungen Frequenzgenerator			
Amplitude [V]	Offset [V]	Frequenz [Hz]	DutyCycle [%]
3,3 V	1,65 V	1 Hz	20%

In Abbildung 3.2 wurde mit der Erhöhung der Belastung die Stromaufnahme durch ein WSN Device simuliert. Die Versorgungsspannung beginnt nach $\approx 20ms$ einzubrechen. Die Übertragung dauert $\approx 10ms$.

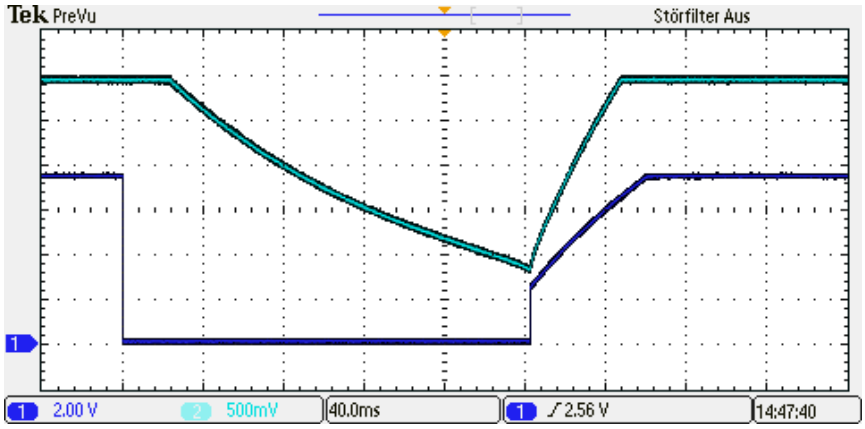


Abb. 3.2: *Last: Sensor mit WSN*

Einstellungen Frequenzgenerator			
Amplitude [V]	Offset [V]	Frequenz [Hz]	DutyCycle [%]
3,3 V	1,65 V	1,5 Hz	50%

In Abbildung 3.3 wurde eine Übertragung von $\approx 15ms$ simuliert

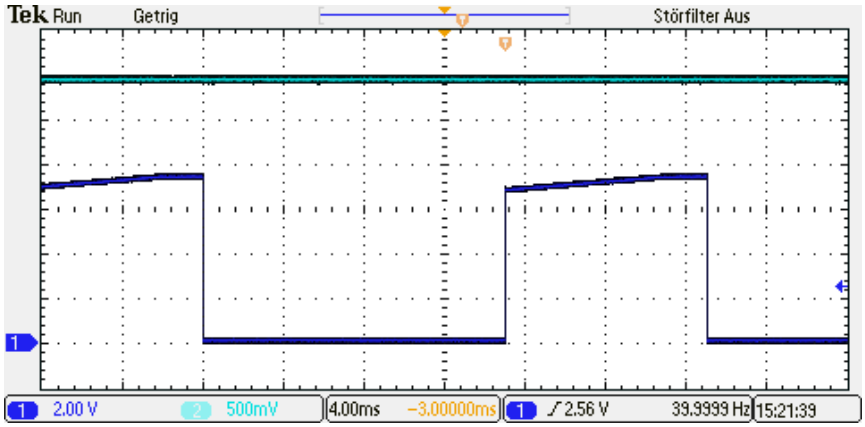


Abb. 3.3: *Last: Sensor mit WSN*

Einstellungen Frequenzgenerator			
Amplitude [V]	Offset [V]	Frequenz [Hz]	DutyCycle [%]
3,3 V	1,65 V	40 Hz	40%

Selbst im Worst Case, wenn über eine Zeit von $\approx 15ms$ die Zweidrahtleitung auf Masse liegt und der Sensor über den Kondensator versorgt wird kommt es zu keinem Einbruch der Versorgungsspannung.

3.1.2 Kommunikation

Diese Messungen untersuchen verschiedene Übertragungsfrequenzen und verschiedene Leitungslängen. Am Master wurden mit einem Funktionsgenerator verschiedene Frequenzen eingestellt und die Spannungsverläufe der Zweidrahtleitung `TL` und `RX` gemessen. Mit den unterschiedlichen Kapazitäten werden die Leitungslängen simuliert. 1nF entsprechen $\approx 100m$ Leitungslänge.

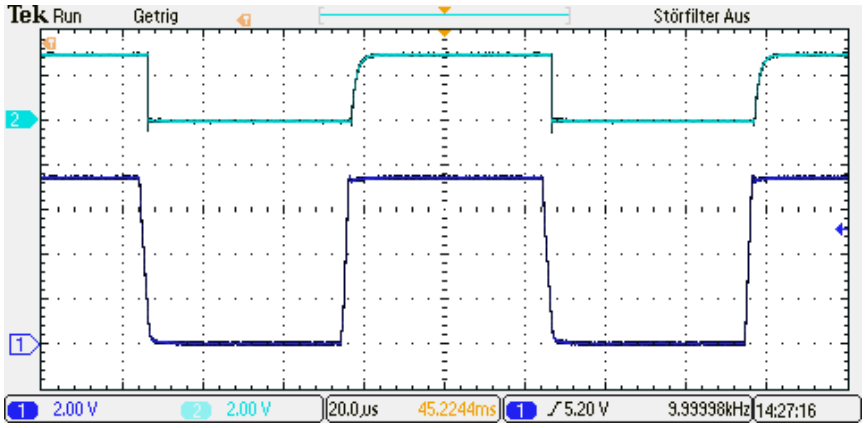


Abb. 3.4: *f... 10kHz, C ... 0nF*

Einstellungen Frequenzgenerator

Amplitude [V]	Offset [V]	Frequenz [Hz]	DutyCycle [%]	Kapazität [nF]
3,3 V	1,65 V	10 kHz	50%	0

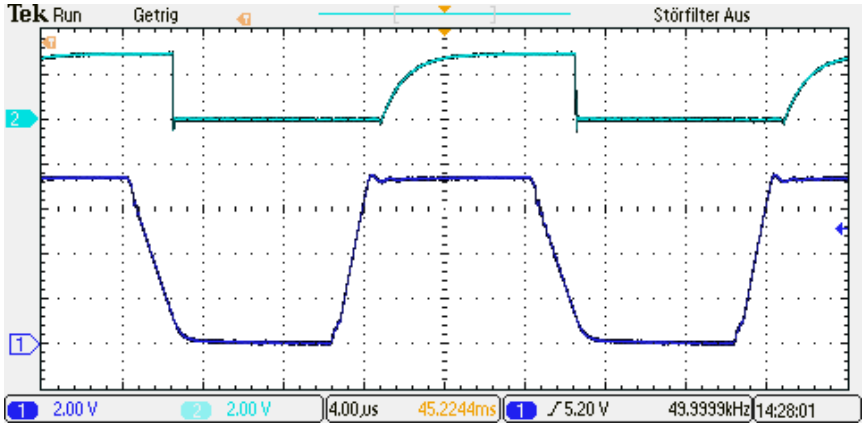


Abb. 3.5: *f... 50kHz, C ... 0nF*

Einstellungen Frequenzgenerator

Amplitude [V]	Offset [V]	Frequenz [Hz]	DutyCycle [%]	Kapazität [nF]
3,3 V	1,65 V	50 kHz	50%	0

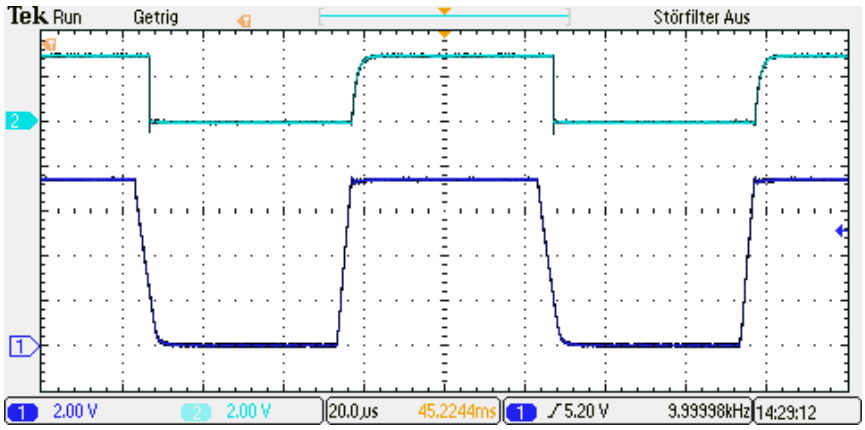


Abb. 3.6: *f... 10kHz, C ... 10nF*

Einstellungen Frequenzgenerator

Amplitude [V]	Offset [V]	Frequenz [Hz]	DutyCycle [%]	Kapazität [nF]
3,3 V	1,65 V	10 kHz	50%	10

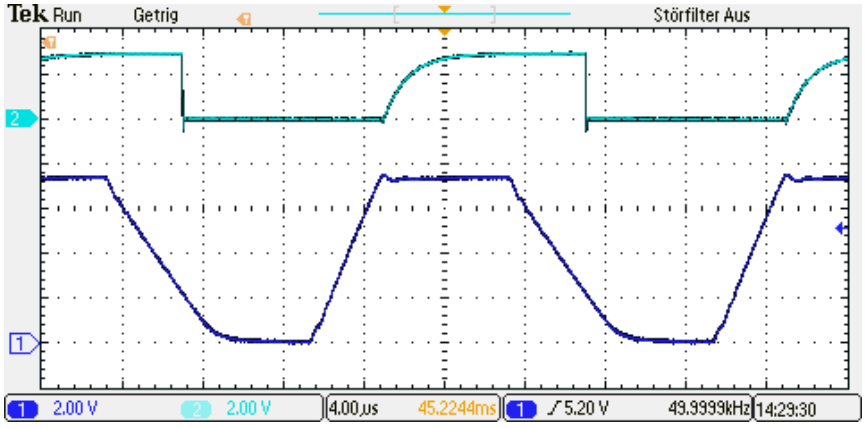


Abb. 3.7: *f... 50kHz, C ... 10nF*

Einstellungen Frequenzgenerator

Amplitude [V]	Offset [V]	Frequenz [Hz]	DutyCycle [%]	Kapazität [nF]
3,3 V	1,65 V	50 kHz	40%	10nF

3.1.3 Verzögerung

Messaufbau: Am Sendepin des Masters `TX_ECU` wurde ein 10kHz Signal angelegt und gemessen. Gleichzeitig wurde am Empfangspin `RX` des Slave gemessen . Die Messung wurde einmal ohne Kondensator und einmal mit Kondensator (100nF) durchgeführt. Die Verzögerung in Abbildung 3.8 entsteht durch die fehlende Anpassung der Threshold Spannungen der Schmitt Trigger an den Spannungsverlauf der Zweidrahtleitung.
In Abbildung 3.9 sieht man eine Vergrößerung aufgrund der höheren kapazitiven Belastung am Ausgang.

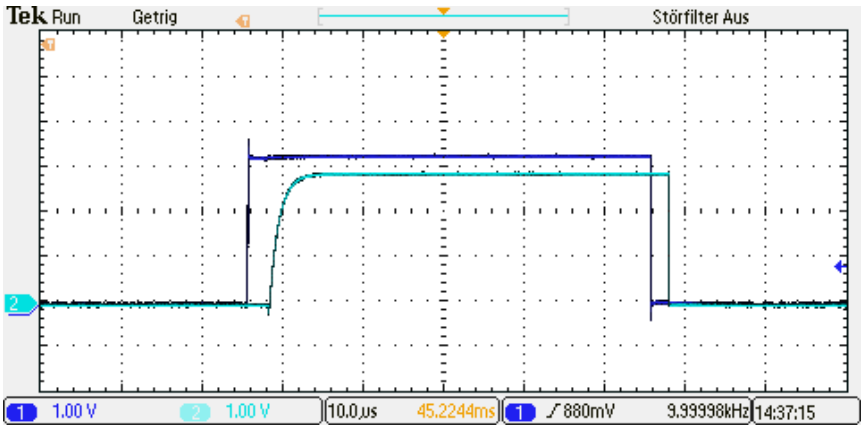


Abb. 3.8: *f... 10kHz, C ... 0nF*

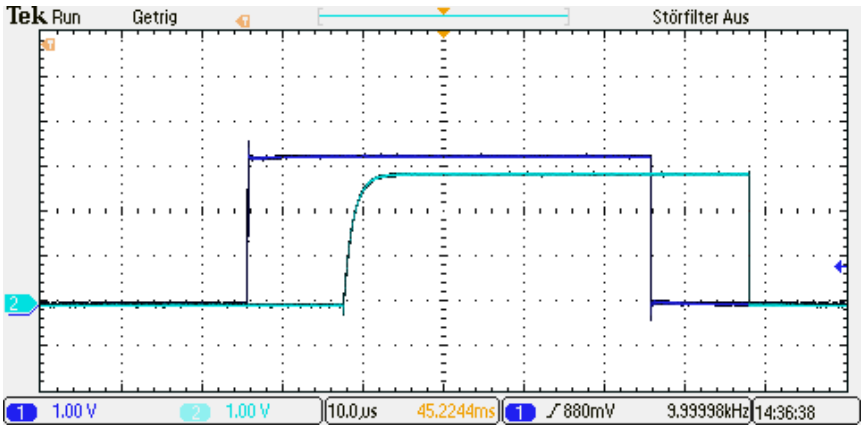


Abb. 3.9: *f... 10kHz, C ... 100nF*

3.1.4 Messung im Betrieb

Diese Messung zeigt die Stromaufnahme des Sensors im Anwendungsfall mit den Kommandos `GetID` und `GetValue`. Dafür wurde eine Konsole mit Microsoft .NET in C# programmiert, die es ermöglicht die einzelnen Befehle zu testen und die Messwerte zu überprüfen. Die Stromaufnahme wurde mit dem Oszilloskop über einen $\approx 1,3\Omega$ Widerstand gemessen.

3.1.4.1 GetID

Der Slave sendet seine Identifikationsdaten an den Master. In Abbildung 3.10 sind die Versorgungsspannung (`VCC`, blau) und der Spannungsverlauf der Zweidrahtleitung (`TL`, türkis) des Sensors dargestellt. In Abbildung 3.11 ist die Stromaufnahme des Sensors, in Abbildung 3.12 die Stromaufnahme des Sensors mit WSN Simulation abgebildet.

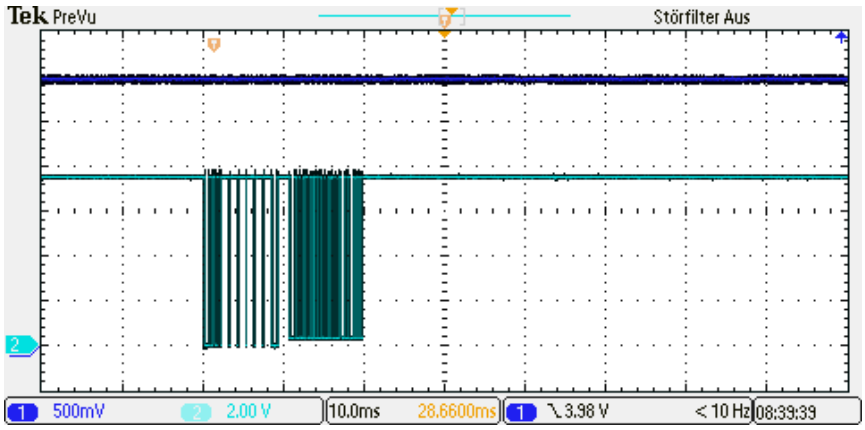


Abb. 3.10: *Spannung GetValue: Interface und PowerSupply*

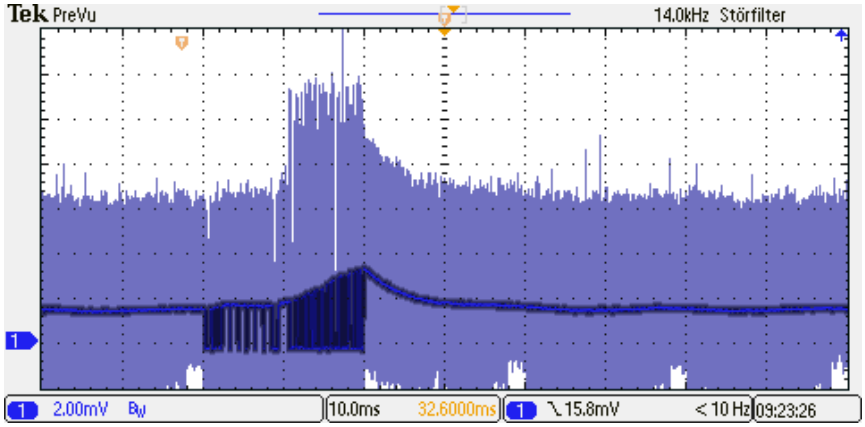


Abb. 3.11: *Stromaufnahme GetID*

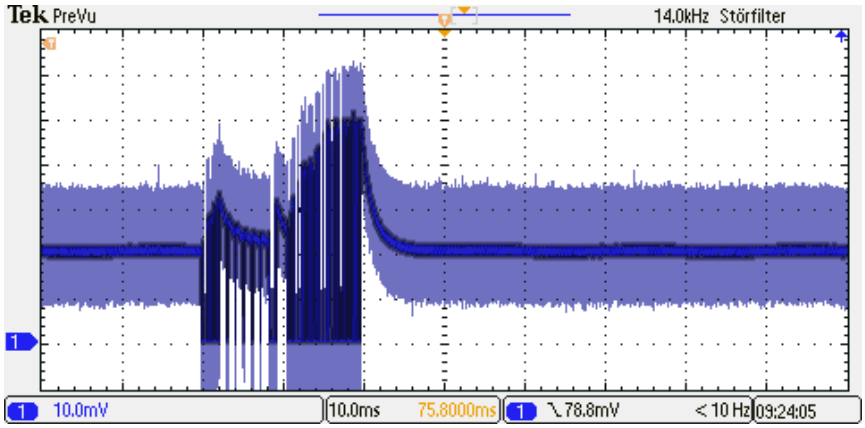


Abb. 3.12: *Stromaufnahme GetValue mit WSN Simulation*

3.1.4.2 GetValue

Der Slave führt eine AD Wandlung durch und sendet das Ergebnis an den Master. In Abbildung 3.13 sind die Versorgungsspannung (`VCC`, blau) und der Spannungsverlauf der Zweidrahtleitung (`TL`, türkis) des Sensors dargestellt. In Abbildung 3.11 ist die Stromaufnahme des Sensors, in Abbildung 3.12 die Stromaufnahme des Sensors mit WSN Simulation abgebildet.
Im Vergleich zum Befehl `GetID` ist eine deutliche Verzögerung zwischen dem Empfangen und Senden der Daten sichtbar. In dieser Zeit findet die Analog–Digital Wandlung statt. Ein vollständiger Kommunikationszyklus ist in $\approx 50ms$ abgeschlossen. Es sind somit maximal 20 Messungen pro Sekunde möglich.

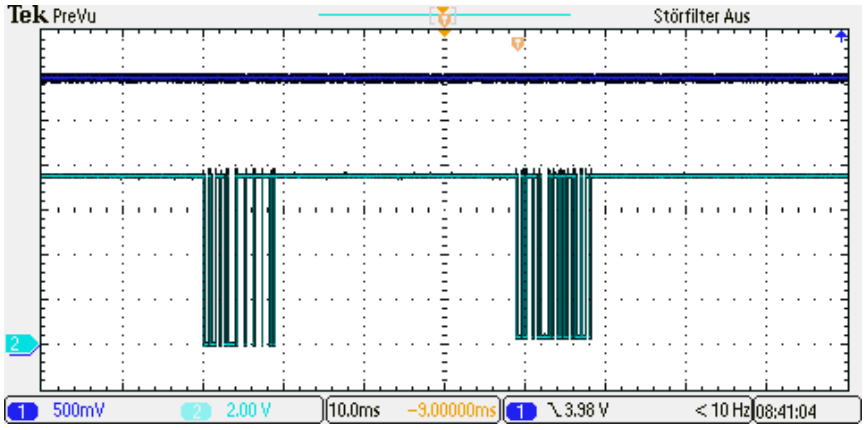


Abb. 3.13: *Spannung GetValue: Interface und PowerSupply*

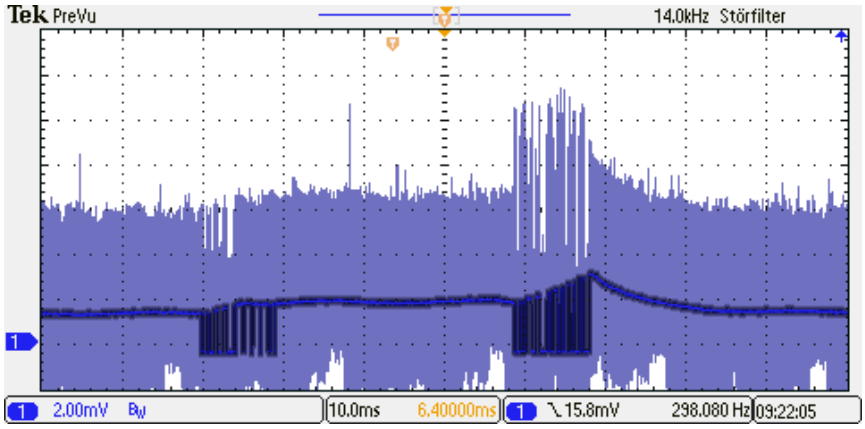


Abb. 3.14: *Stromaufnahme GetValue*

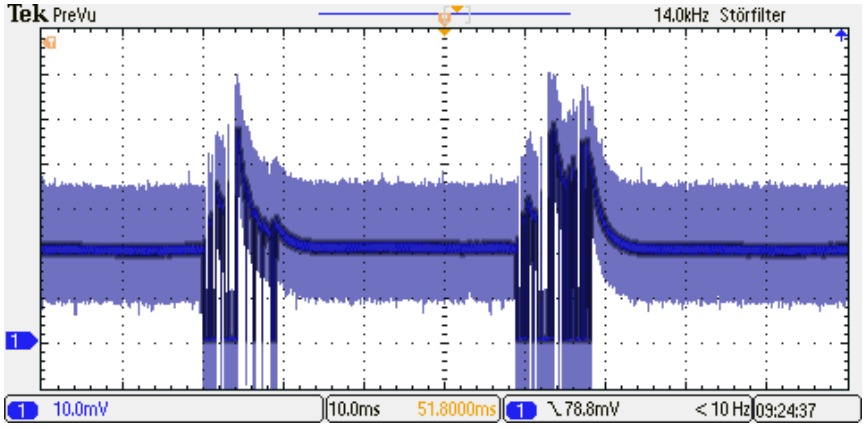


Abb. 3.15: *Stromaufnahme GetValue mit WSN Simulation*

3.2 Kalibrierung der Einstrahlungsmessung

Bei der Kalibrierung der Messschaltung zur Erfassung der Einstrahlung wurden als Referenz Sensoren der Hersteller Technische Alternative TA (GBS01) und Resol (CS10) verwendet. Der Sensor der TA liefert mit der im Datenblatt angeführten Messschaltung eine Spannung von $1,2 \ldots 2V$. Der Sensor von Resol liefert über eine Photodiode (BPW20RF), einen zur Einstrahlung proportionalen Kurzschlußstrom. Um diesen in eine Spannung umzuwandeln wurde die Messschaltung des Insol Kanals verwendet. Die Erfassung der Spannungswerte erfolgt mit dem Datenerfassungsmodul USB-6009 von National Instruments. Für die Datenaufzeichnung wurde in LabView ein Programm erstellt, das die Spannungen aufzeichnet und speichert. Die Daten wurden mithilfe von Microsoft Excel ausgewertet. In Abbildung 3.16 ist der Messaufbau abgebildet.

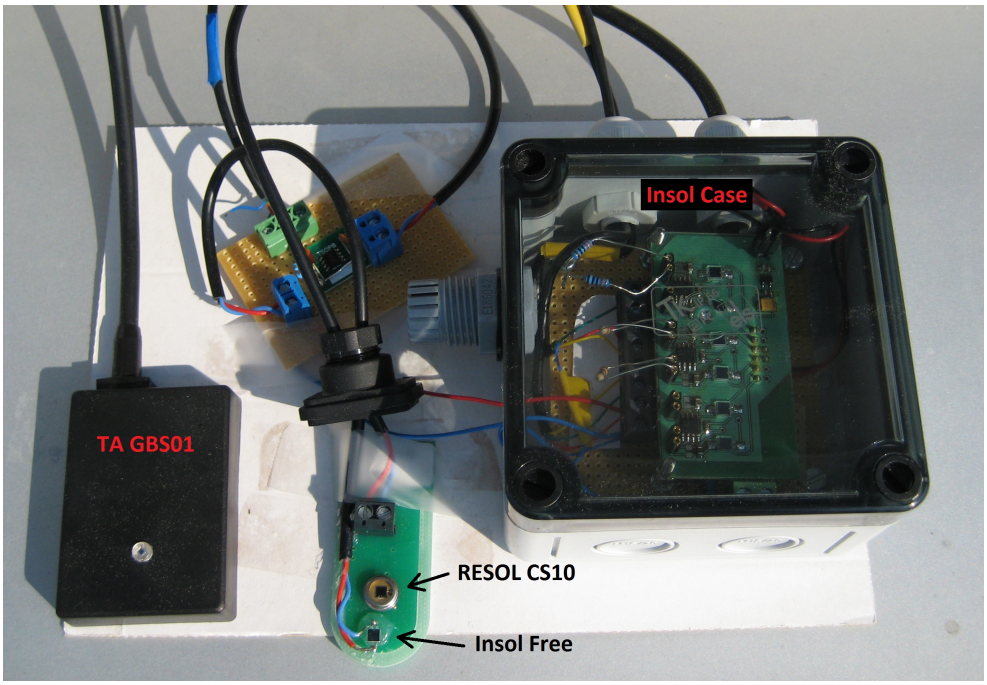


Abb. 3.16: *Messaufbau zur Kalibrierung des Insol Kanals*

Für die Kalibrierung des Insol Kanals wurde je eine Messung mit und ohne Gehäuse durchgeführt. Der Korrekturfaktor des Gehäuses wurde empirisch ermittelt. Der Rückkoppelwiderstand wurde so gewählt dass der Messbereich von 3V ausgenutzt wird.

$$E[W/m^2] = \frac{V_{Insol}}{R_F \cdot I_K} \cdot FKT_{Case}$$

Sensor	R_F	I_K	FKT_{Case}
Resol	$1,5k\Omega$	$2,03\mu A$	1
Insol Free	660Ω	$2,8\mu A$	1
Insol Case	$1,5k\Omega$	$2,8\mu A$	1,52

Tabelle 3.1: *Insol Parameter*

Mit den gewählten Parametern ist der höchstmögliche Einstrahlungswert des Sensors mit $1085W/m^2$ definiert.

Die Diagramme zeigen den Eistrahlungsverlauf über die Tageszeit. Der Insol Kanal wurde so kalibriert, dass dieser im Bereich der beiden anderen Sensoren liegt.

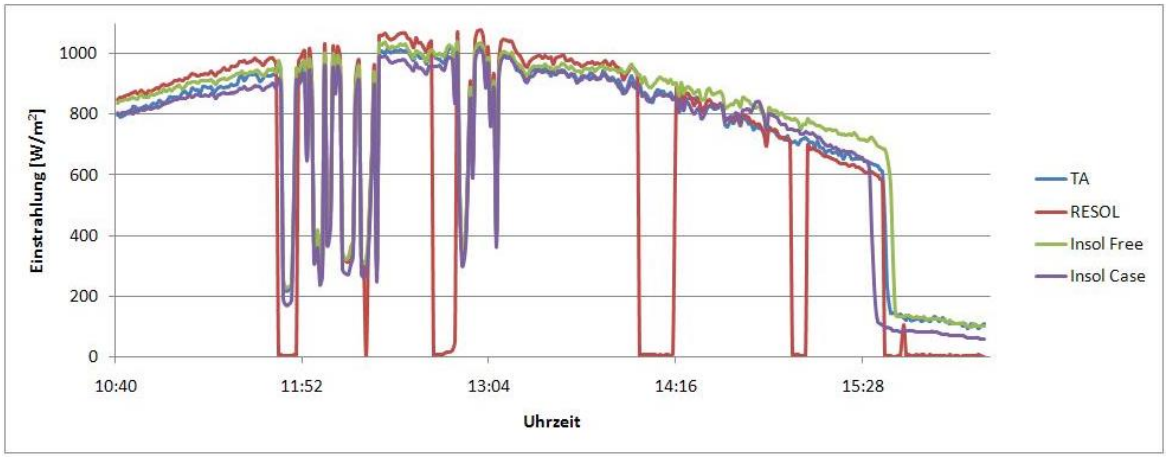


Abb. 3.17: *Einstrahlungsmessung vom 19.04.2011*

Der Verlauf in Abbildung 3.17 zeigt wechselnde Bewölkung um die Mittagszeit. Ab ca. 15:30 wirft das Gebäude seinen Schatten auf die Sensoren. Die zwischenzeitlichen Ausfälle des Resol Sensors sind auf Probleme mit der Auswerteschaltung zurückzuführen.

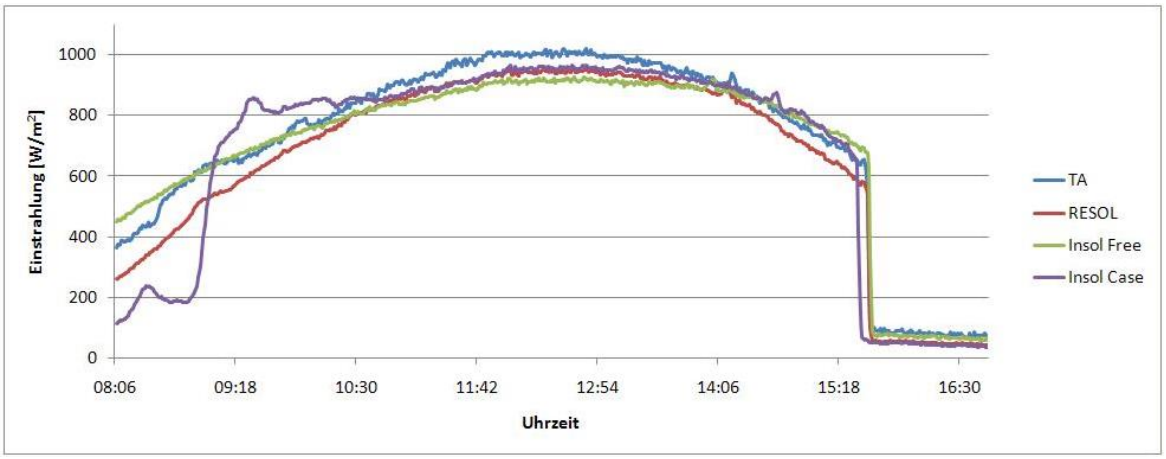


Abb. 3.18: *Einstrahlungsmessung vom 20.04.2011*

Abbildung 3.18 zeigt den typischen Tagesgang bei wolkenlosem Himmel. Alle Sensoren zeigen einen ähnlichen Verlauf. Der Verlauf des Sensors im Kanal *Insol Case* ist auf die Geometrie des Gehäuses zurückzuführen, da derselbe Sensor ohne Gehäuse einen normalen Verlauf zeigt. Der flachere Verlauf des *Insol Free* Kanals ist auf den größeren Öffnungswinkel der verwendeten Photodiode im Vergleich zu den Photodioden von Resol und Technische Alternative zurückzuführen.

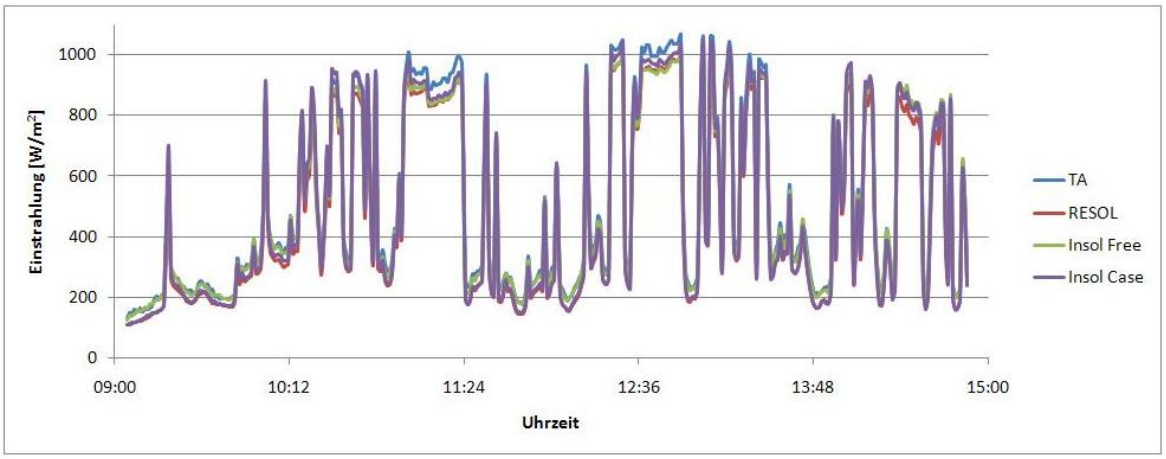


Abb. 3.19: *Einstrahlungsmessung vom 26.04.2011*

Abbildung 3.19 zeigt den Tagesgang bei wechselnder Bewölkung.

3.3 Temperaturmessung

3.3.1 Temperaturberechnung

Die Eigenschaften von Platin Widerstandsthermometern sind in der Norm IEC751 festgelegt. Dabei sind auch zwei Polynome zur Umrechnung des Widerstandswertes für unterschiedliche Temperaturbereiche und die dazugehörigen Koeffizienten definiert. Der erste Temperaturbereich von -200°C bis 0°C wird durch ein Polynom 3. Grades beschrieben, der Temperaturbereich von 0°C bis 850°C durch ein Polynom 2. Grades.

$$R_T = R_0 \left(1 + \alpha \cdot T + \beta \cdot T^2\right) \tag{3.1}$$

Für den hier benötigten Temperaturbereich ist es ausreichend mit der linearen Gleichung

$$R_T = R_0 + \alpha \cdot T \tag{3.2}$$

mit $\alpha = 3,9083 \cdot 10^{-3}/°C$ zu rechnen.

3.3.2 Messung

Der Temperaturmesskanal wurde mit verschiedenen Pt1000 Widerstandswerten belastet, und die Spannung am Ausgang gemessen. Die Widerstandswerte reichen von $R_{PT1000} = 916\Omega$ bis $R_{PT1000} = 1539\Omega$, das entspricht einem Temperaturbereich von -20°C bis ca. 140°C. Das Zuschalten der Widerstände erfolgte über eine Relaiskarte die mit LabView angesteuert wurde. Die Widerstandserhöhung beträgt $\Delta R \approx 40\Omega$. Die Widerstandswerte wurden vorher mit einem Multimeter ermittelt, um den Messfehler gering zu halten. In Tabelle 3.2 sind die Messwerte aufgelistet. In Abbildung 3.20 ist die Temperaturabweichung zur gesetzten Temperatur dargestellt.

$R_{PT1000}[\Omega]$	$U_{calc}[V]$	$T_{calc}[°C]$	$U_{mess}[V]$	$R_{mess}[\Omega]$	$T_{mess}[°C]$	$\Delta[°C]$
916	1,2987	-21,49	1,2976	914,67	-21,83	-0,34
955	1,3295	-11,51	1,3290	954,45	-11,65	-0,14
996	1,3607	-1,02	1,3603	995,49	-1,15	-0,13
1035	1,3893	8,96	1,3892	1034,92	8,94	-0,02
1077	1,4190	19,70	1,4193	1077,43	19,81	0,11
1117	1,4463	29,94	1,4457	1116,10	29,71	-0,23
1157	1,4726	40,17	1,4729	1157,35	40,26	0,09
1197	1,4981	50,41	1,4989	1198,17	50,71	0,30
1237	1,5228	60,64	1,5229	1237,23	60,70	0,06
1278	1,5472	71,13	1,5477	1278,82	71,34	0,21
1318	1,5703	81,37	1,5704	1318,16	81,41	0,04
1357	1,5921	91,34	1,5922	1357,16	91,38	0,04
1398	1,6143	101,83	1,6141	1397,53	101,71	-0,12
1439	1,6358	112,33	1,6357	1438,73	112,26	-0,07
1479	1,6562	122,56	1,6555	1477,67	122,22	-0,34
1519	1,6760	132,79	1,6751	1517,24	132,34	-0,45

Tabelle 3.2: *Messung Intemp Kanal*

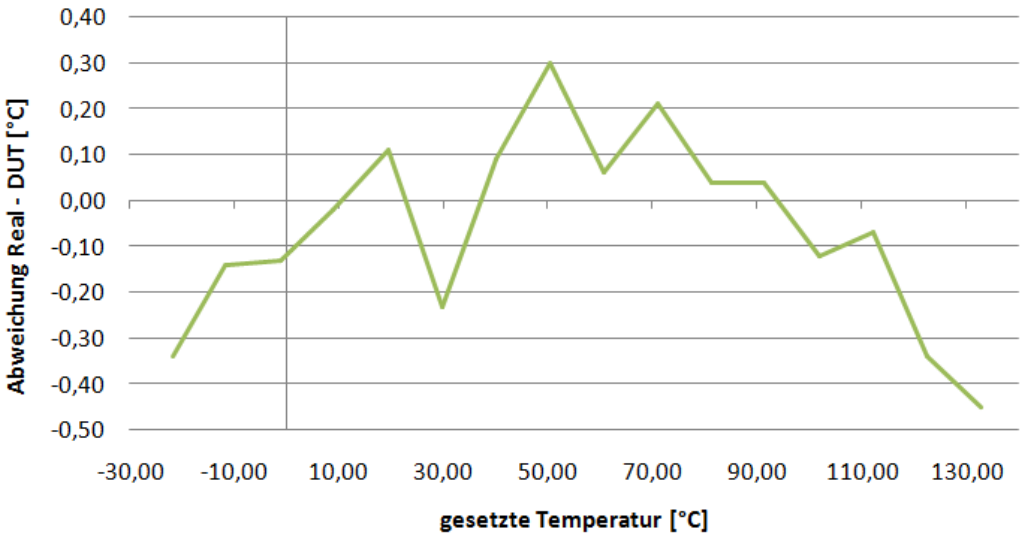


Abb. 3.20: *Absoluter Fehler Intemp Kanal*

Kapitel 4

Ausblick

In dieser Diplomarbeit wurde ein Prototyp zur Messung der solaren Einstrahlung erstellt, mit Methoden des modellgetriebenen Systemdesigns.

Ein großer Vorteil der „Model Driven Architecture (MDA)“ ist die konsistente Dokumentation die im Laufe des Projektes mithilfe der Modelle automatisch generiert wird. Man darf jedoch nicht das eigentliche Ziel aus den Augen verlieren, nämlich die Implementierung des Systems und nicht die Perfektionierung von Modellen. Sie sollen das Essentielle auf den Punkt bringen und zur Unterstützung dienen.

Modellgetrieben beinhaltet auch die automatische Generierung von Sourcecode. Dies beschränkte sich bei dieser Arbeit jedoch auf die Generierung der einzelnen Dateien sowie der Funktionsheader. Genutzt wurde die Möglichkeit des Reverse–Engineerings um die Konsistenz zwischen Modell und implementierten Code herzustellen. Bei der Hardware dienen die erstellten Modelle der Dokumentation und der Übersicht.

Die Ergebnisse die der Sensor liefert, sind wie erwartet sehr ungenau, für diesen Anwendungsfall aber ausreichend. Eine genauere Messung ist mit einer Photodiode nicht möglich, da die Empfindlichkeit stark von der Wellenlänge und vom Diodentyp abhängt. Der Sensor liegt im ersten Vergleich mit den Sensoren der Technischen Alternative und Resol auf gleicher Ebene. Für eine exakte Messung besteht die Möglichkeit die Kalibrierung mit einem Pyranometer oder mit Hilfe eines Sonnensimulators durchzuführen.

Das Interface des entworfenen Prototyps wurde für ein ZigBee WSN konzipiert. Dies bestimmte auch die Größe des Kondensators, der während der Kommunikation die Spannungsversorgung überbrückt. Wird der Sensor nur zur Messung von Temperatur und Einstrahlung benutzt, ist der im Pototyp verwendete Kondensator zur Überbrückung der Spannungsversorgung zu groß dimensioniert. Dasselbe gilt für den verwendeten Controller. Andere Controller der Familie „MSP430 F2xxx“ besitzen weniger Peripherie bei der gleichen Pinkonfiguration bei deutlich niedrigeren Kosten.

Des Weiteren ist das Temperaturverhalten des Sensors zu untersuchen, da er hohen Temperaturschwankungen ausgesetzt ist. Alle verwendeten Bauteile besitzen mindestens einen Temperaturbereich von -20° bis +85°C. Als mögliches Problem könnte sich die Alterung der Kondensatoren mit der verbundenen Abnahmen der Kapazität herausstellen. Um die Sensorplatine vor der direkten Sonneneinstrahlung zu schützen kann der Sensor auf zwei Ebenen aufgeteilt werden, sodass nur mehr die Photodiode selbst der direkten Einstrahlung ausgesetzt ist.

In Abbildung 4.1 ist der Prototyp des Sensors abgebildet.

Abb. 4.1: *Sensor Prototyp*

Literaturverzeichnis

[1] A. T. Bahill and B. Gissing. Re-evaluating systems engineering concepts using systems thinking. *IEEE_J_SMCC*, 28(4):516–527, 1998.

[2] Dr. Sean Cochrane. *A Structured Embedded Software Model for Improved Code Portability and Reuse*. Cyan Technology Ltd.

[3] Bruce Powel Douglass. *Real Time Agility*, volume 1. Addison-Wesley, 2009.

[4] H. Hartl, E. Krasser, G. Winkler, W. Pribyl, and P. Söser. *Elektronische Schaltungstechnik*. Pearson Studium, 2008.

[5] Hans Häckel. *Meteorologie*, volume 6. Verlag Eugen Ulmer, 2008.

[6] IAR Systems. *IAR C/C++ Compiler–Reference Guide*.

[7] Christoph Kecher. *UML 2.0*. Galileo Computing, 2006.

[8] Maxim. *Choosing the Right 1-Wire® Master for Embedded Applications*. Application Note 4206.

[9] Maxim. *Understanding and Using Cyclic Redundancy Checks with Maxim iButton Products*. Application Note 27.

[10] Klaus Schmaranz. *Softwareentwicklung in C++*. Springer Verlag, 2001.

[11] Texas Instruments. *MSP430 Software Coding Techniques*, 2006. SLAA294A (Rev.A).

[12] Texas Instruments. *MSP430 Hardware Tools User's Guide*, 2010. SLAU278F (Rev.F).

[13] Texas Instruments. *MSP430F241x, MSP430F261x Mixed Signal Microcontroller*, 2011. SLAS541G (Rev.G).

[14] Texas Instruments. *MSP430x2xx Family User's Guide*, 2011. SLAU144H (Rev.H).

[15] Ulrich Tietze and Christoph Schenk. *Halbleiter- Schaltungstechnik*, volume 12. Springer, 2002.

[16] Vishay Semiconductors. *Measurement Techniques*. Application Note.

[17] Marian Walter and Stefan Tappertzhofen. *Das MSP430 Mikrocontroller Buch*, volume 1. Elektor Verlag, 2011.

[18] Tim Weilkiens. *Systems Engineering mit SysML/UML*, volume 2. dpunkt.verlag, 2008.

Anhang A

Anhang

A.1 System Modelle

A.1.1 Technical Issues

A.1.1.1 Kommunikation

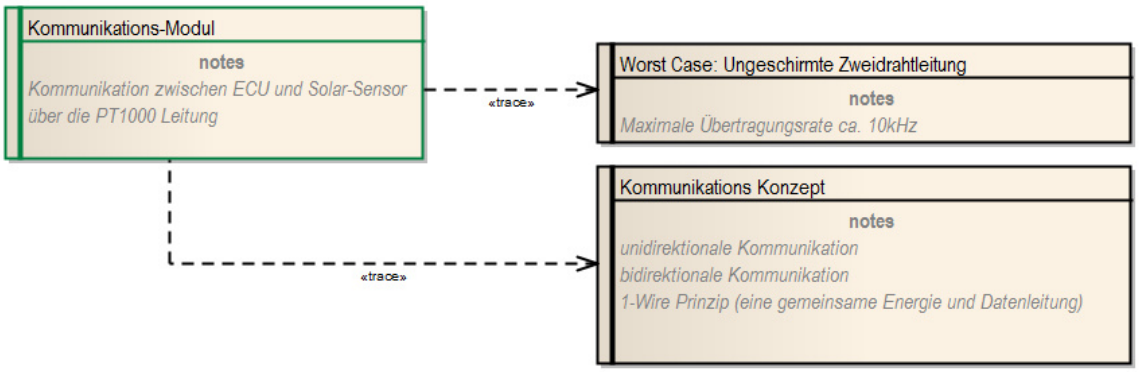


Abb. A.1: *Technical Issue Kommunikations Modul*

A.1.1.2 Sensorik

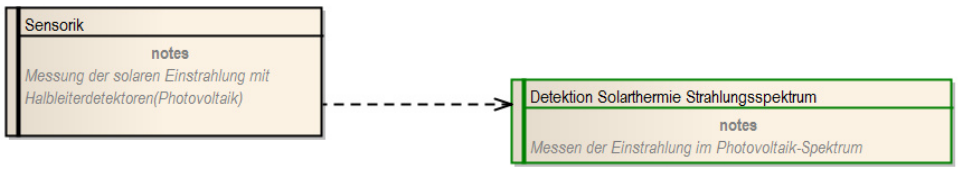


Abb. A.2: *Technical Issue: Sensorik*

A.1.1.3 Power Supply

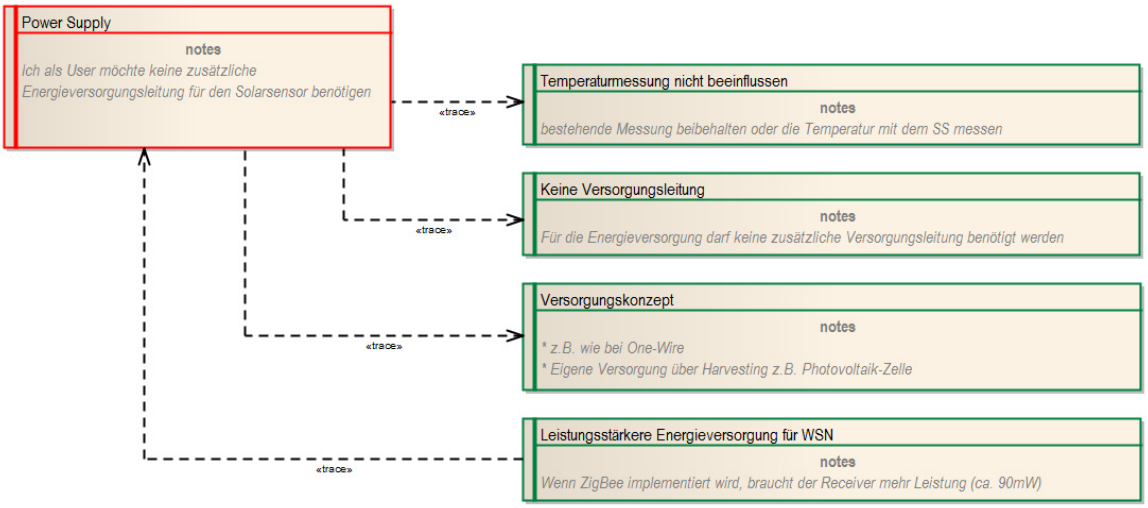


Abb. A.3: *Technical Issue: Power Supply*

A.1.2 Non–Functional Requirements

Abb. A.4: *Non–Functional Requirements*

A.2 Software

IAR Systems	Embedded Workbench 6.0, Kickstart Edition
Altium	Altium Designer Summer 09, Student License
Sparcx Systems	Enterprise Architect 8.0.861
Microsoft	.NET Framework 4.0
National Instruments	Labview 2009, Student Edition

A.3 Messgeräte

Extech Instruments	True RMS Multimeter 430	Multimeter
Hewlett Packard	33120A	Funktionsgenerator
Tektronix	MSO2012 Mixed Signal Oscilloscope	Oszilloskop
TTi	CPX200	Netzgerät

A.4 Schematics

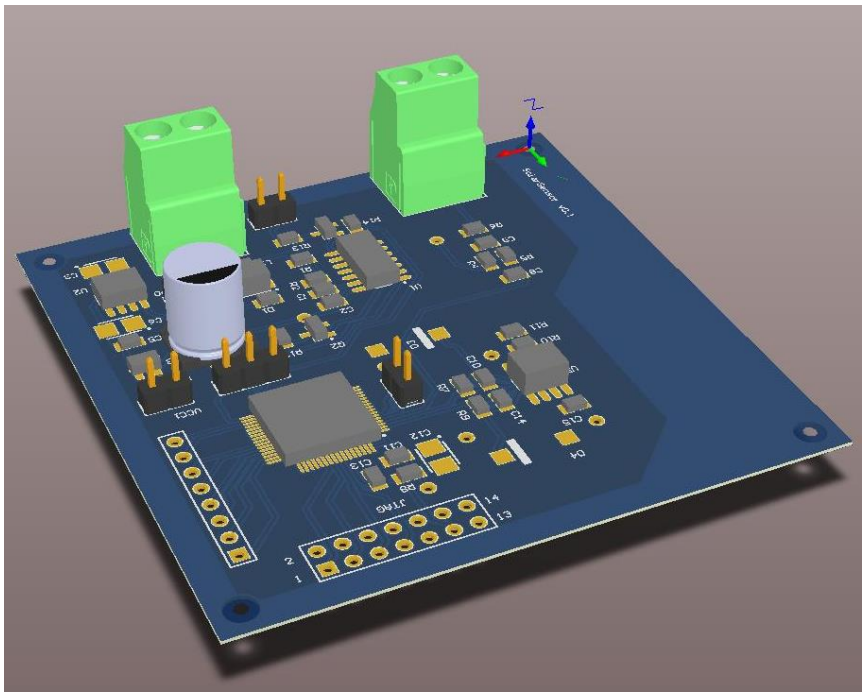
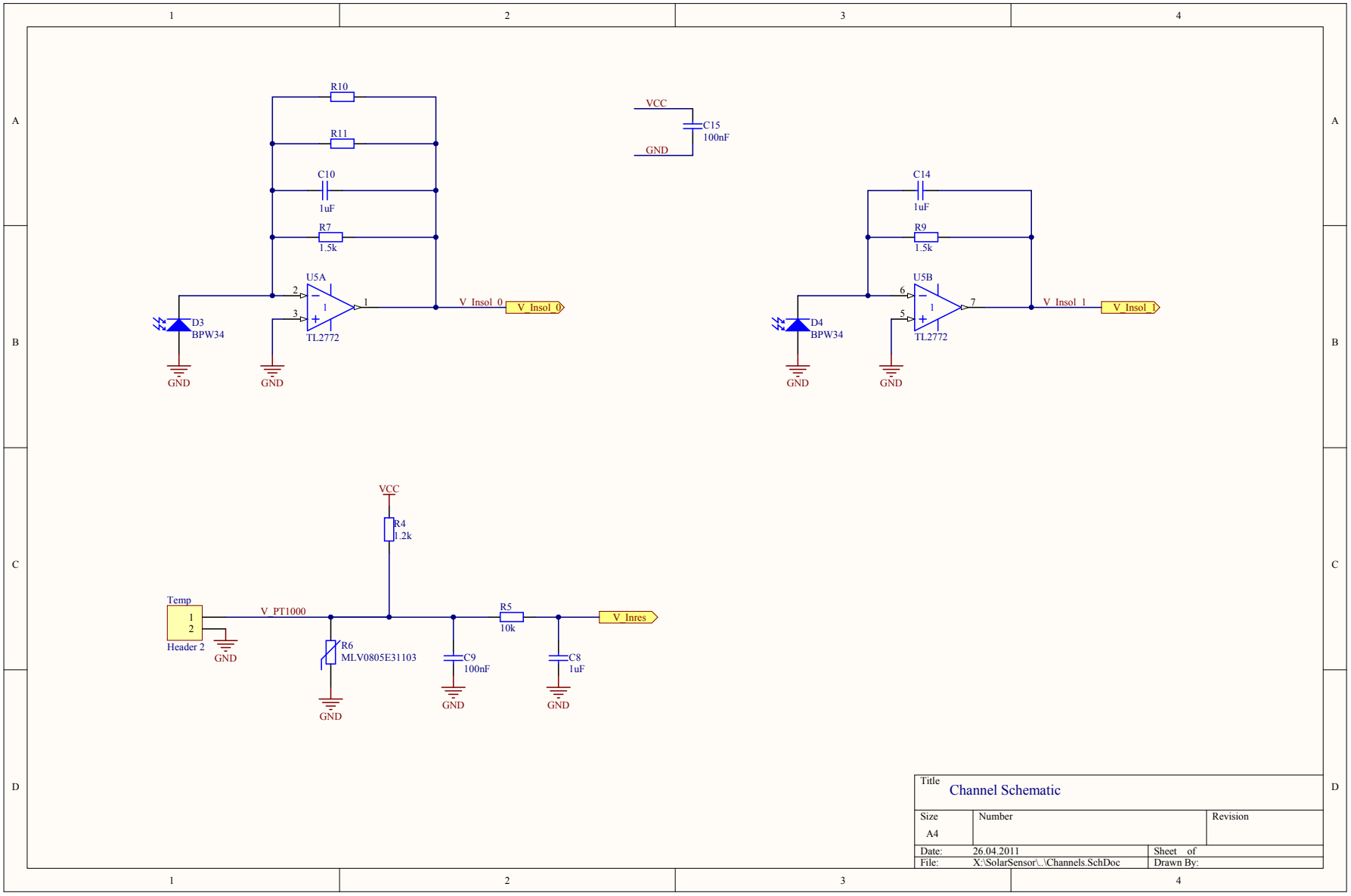
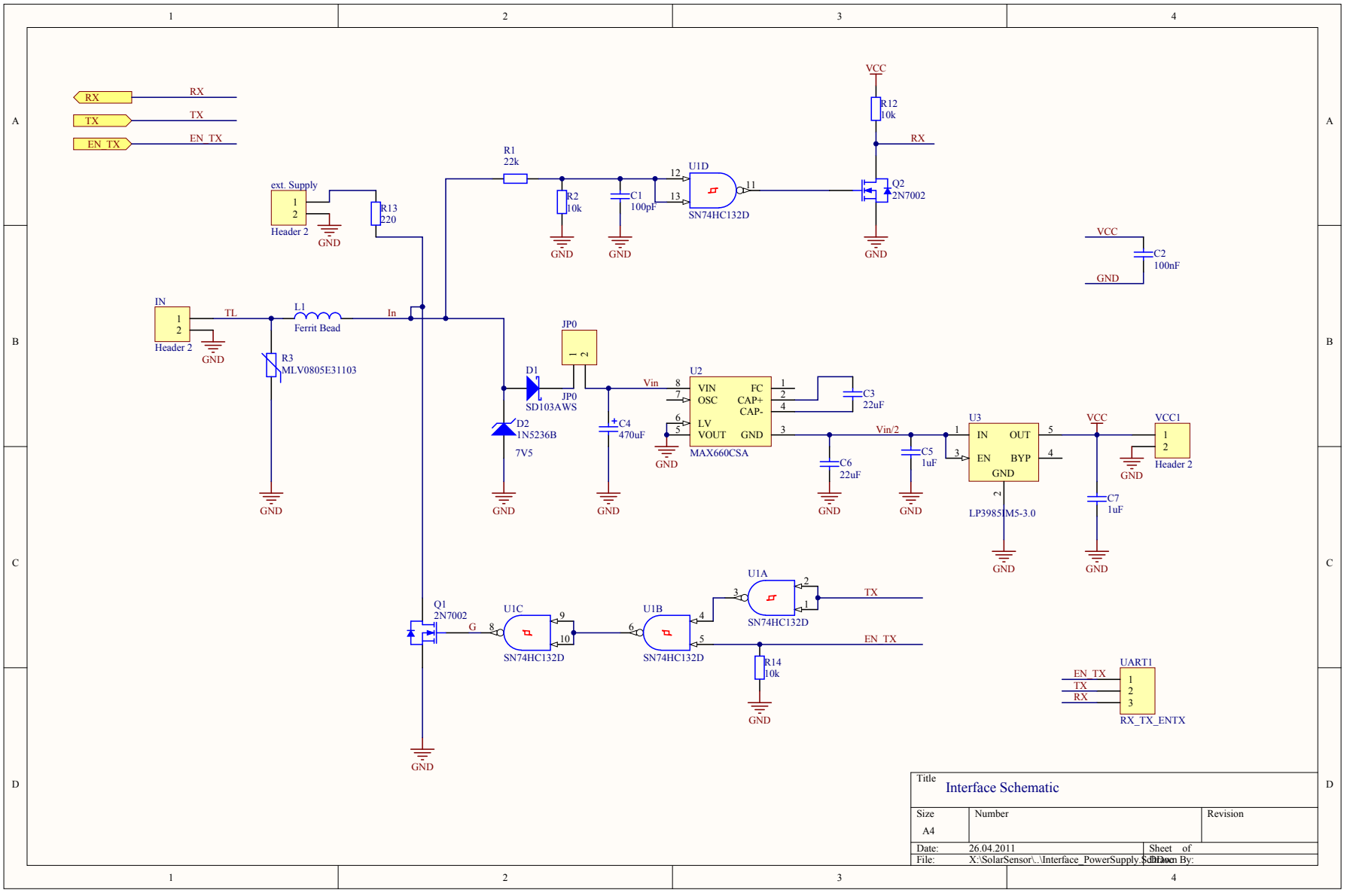
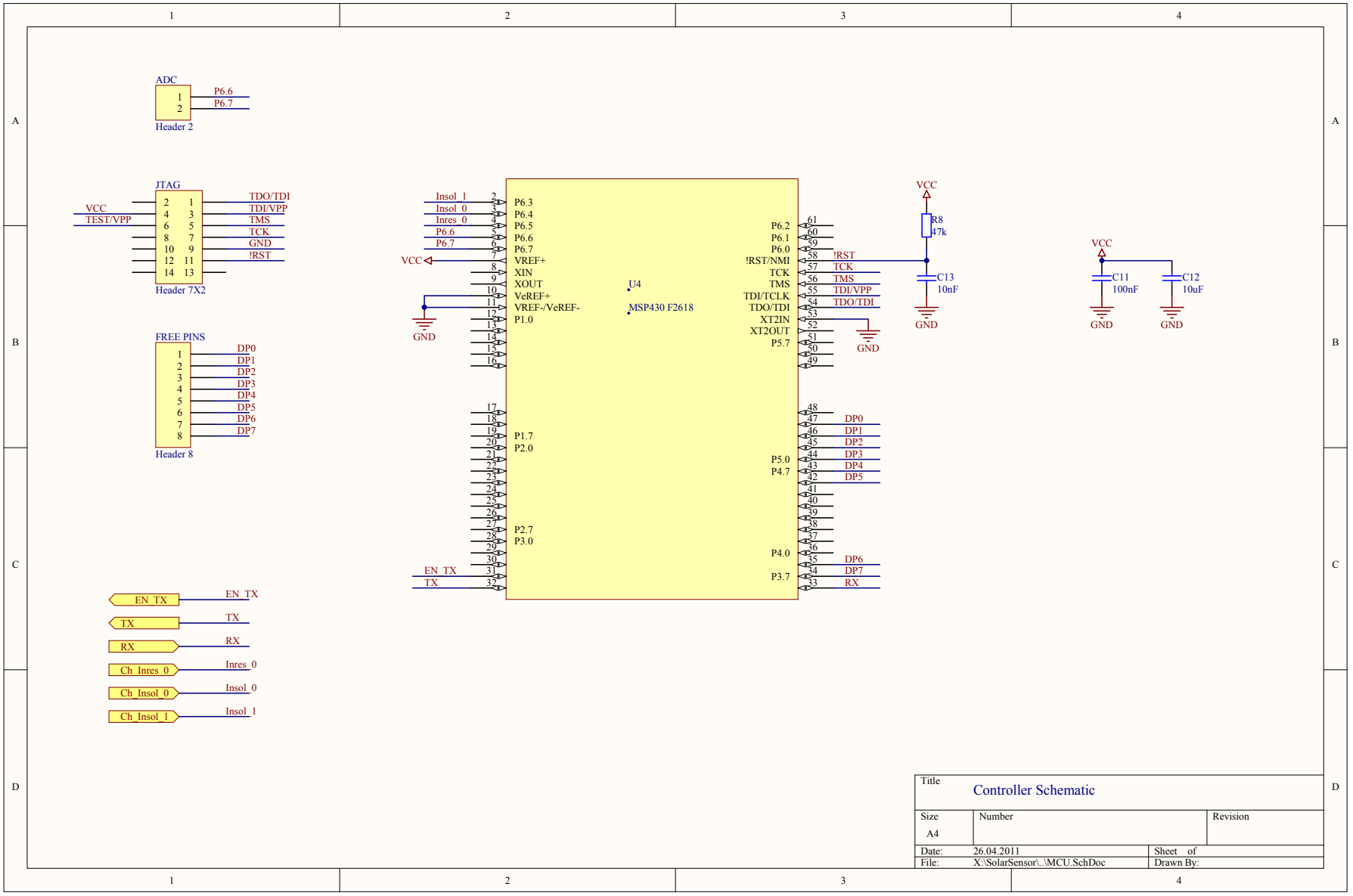


Abb. A.5: *PSI: Altium Designer 3D Modell*


R10
R11
C10
1uF
R7
1.5k
U5A
TL2772
D3
BPW34
V_Insol_0
V_Insol_0
GND
GND
VCC
C15
100nF
GND
C14
1uF
R9
1.5k
U5B
TL2772
D4
BPW34
V_Insol_1
V_Insol_1
GND
GND
VCC
R4
1.2k
Temp
Header 2
V_PT1000
R6
MLV0805E31103
C9
100nF
R5
10k
V_Inres
C8
1uF
GND
GND
GND
GND
Title
Channel Schematic
Size
A4
Number
Revision
Date:
26.04.2011
Sheet of
File:
X:\SolarSensor\..\Channels.SchDoc
Drawn By:


RX
TX
EN_TX
ext. Supply
Header 2
R13
220
R1
22k
R2
10k
C1
100pF
U1D
SN74HC132D
R12
10k
Q2
2N7002
VCC
C2
100nF
IN
Header 2
TL
L1
Ferrit Bead
In
R3
MLV0805E31103
JP0
D1
SD103AWS
D2
1N5236B
7V5
Vin
U2
MAX660CSA
C3
22uF
C4
470uF
C6
22uF
Vin/2
C5
1uF
U3
LP3985IM5-3.0
C7
1uF
VCC1
Header 2
Q1
2N7002
U1C
SN74HC132D
U1B
SN74HC132D
U1A
SN74HC132D
R14
10k
UART1
RX_TX_ENTX
GND
Title
Interface Schematic
Size
A4
Number
Revision
Date:
26.04.2011
Sheet of
File:
X:\SolarSensor\...\Interface_PowerSupply.SchDoc
Drawn By:


ADC
P6.6
P6.7
Header 2
JTAG
VCC
TEST/VPP
TDO/TDI
TDI/VPP
TMS
TCK
GND
!RST
Header 7X2
FREE PINS
DP0
DP1
DP2
DP3
DP4
DP5
DP6
DP7
Header 8
EN_TX
TX
RX
Ch_Inres_0
Ch_Insol_0
Ch_Insol_1
Inres_0
Insol_0
Insol_1
Insol_1
Insol_0
Inres_0
P6.6
P6.7
VCC
GND
P6.3
P6.4
P6.5
P6.6
P6.7
VREF+
XIN
XOUT
VeREF+
VREF-/VeREF-
P1.0
P1.7
P2.0
P2.7
P3.0
EN_TX
TX
U4
MSP430 F2618
P6.2
P6.1
P6.0
!RST/NMI
TCK
TMS
TDI/TCLK
TDO/TDI
XT2IN
XT2OUT
P5.7
P5.0
P4.7
P4.0
P3.7
!RST
TCK
TMS
TDI/VPP
TDO/TDI
DP0
DP1
DP2
DP3
DP4
DP5
DP6
DP7
RX
R8
47k
C13
10nF
C11
100nF
C12
10uF
Title
Controller Schematic
Size
A4
Number
Revision
Date:
26.04.2011
Sheet of
File:
X:\SolarSensor\..\MCU.SchDoc
Drawn By:

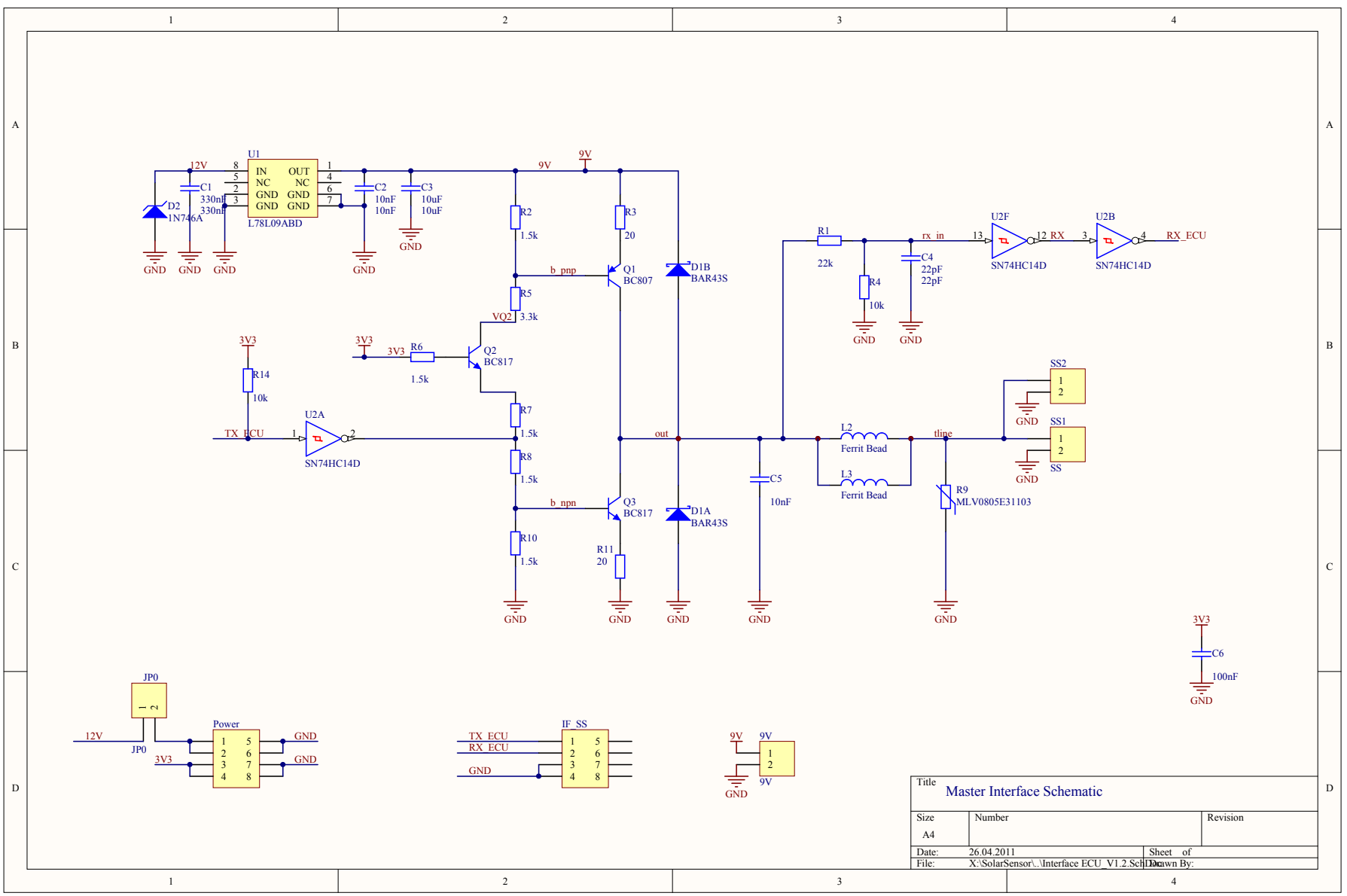
Bill of Materials

Source Data From: SolarSensor.PrjPcb
Project: SolarSensor.PrjPcb
Variant: None

Creation Date: 26.04.2011 13:16:15
Print Date: 40659 40659,58267

Footprint	Comment	LibRef	Designator	Description	Quantity
RESC2012N		Res2	R10, R11	Resistor	2
DIOM4326X24	1N5236B	1N5236B	D2	Half Watt Zener	1
CAPC2012N	1uF	Cap	C5, C7, C8, C10, C14	Capacitor	5
RESC2012N	1.2k	Res2	R4	Resistor	1
RESC2012N	1.5k	Res2	R7, R9	Resistor	2
SOT23_N	2N7002	2N7002	Q1, Q2	N-channel Enhancement Mode Field-effect T	2
RESC2012N	10k	Res2	R2, R5, R12, R14	Resistor	4
CAPC2012N	10nF	Cap	C13	Capacitor	1
TC3528-1411	10uF	Cap	C12	Capacitor	1
RESC2012N	22k	Res2	R1	Resistor	1
TC3528-1411	22uF	Cap	C3, C6	Capacitor	2
RESC2012N	47k	Res2	R8	Resistor	1
CAPC2012N	100nF	Cap	C2, C9, C11, C15	Capacitor	4
CAPC2012N	100pF	Cap	C1	Capacitor	1
RESC2012N	220	Res2	R13	Resistor	1
Elko_8x10	470uF	Cap Pol3	C4	Polarized Capacitor (Surface Mount)	1
BPW34S	BPW34	BPW34S	D3, D4	Silicon PIN Photodiode	2
INDC2012L	Ferrit Bead	Inductor	L1	Inductor	1
MKDSN 1,5/ 2	Header 2	MKDSN 1,5/ 2-5,	IN, Temp		2
HDR1X2	Header 2	Header 2	ADC, ext. Supply, VCC1	Header, 2-Pin	3
HDR2X7	Header 7X2	Header 7X2	JTAG	Header, 7-Pin, Dual row	1
HDR1X8	Header 8	Header 8	FREE PINS	Header, 8-Pin	1
HDR1X2	JP0	Header 2	JP0	Header, 2-Pin	1
MF05A_N	LP3985IM5-3.0	LP3985IM5-3.0	U3	Micropower, 150mA Low-Noise Ultra Low-Dr	1
NSO8_N	MAX660CSA	MAX660CSA	U2	CMOS Monolithic Voltage Converter	1
RESC2012N	MLV0805E31103	Res Varistor	R3, R6	Varistor (Voltage-Sensitive Resistor)	2
TSQFP50P12	MSP430 F2618	MSP430 F2618	U4	MSP430 F2618	1
HDR1X3	RX_TX_ENTX	Header 3	UART1	Header, 3-Pin	1
RESC2012N	SD103AWS	D Schottky	D1	Schottky Diode	1
D014_N	SN74HC132D	SN74HC132D	U1	Quadruple Positive-NAND Gate with Schmitt	1
SOIC127P600	TL2772	TLV2772	U5	Low Voltage Rail To Rail OPAmp	1
					50

Approved	Notes	


U1
L78L09ABD
IN
OUT
NC
NC
GND
GND
GND
GND
12V
C1
330nF
330nF
C2
10nF
10nF
C3
10uF
10uF
D2
1N746A
9V
R2
1.5k
R3
20
Q1
BC807
b_pnp
D1B
BAR43S
R5
VQ2
3.3k
3V3
R6
1.5k
Q2
BC817
R14
10k
U2A
TX_ECU
SN74HC14D
R7
1.5k
R8
1.5k
out
b_npn
Q3
BC817
D1A
BAR43S
R10
1.5k
R11
20
C5
10nF
R1
22k
R4
10k
C4
22pF
22pF
rx_in
U2F
SN74HC14D
U2B
SN74HC14D
RX
RX_ECU
L2
Ferrit Bead
L3
Ferrit Bead
tline
SS2
SS1
SS
R9
MLV0805E31103
C6
100nF
JP0
Power
IF_SS
TX_ECU
RX_ECU
Title
Master Interface Schematic
Size
A4
Number
Revision
Date:
26.04.2011
Sheet of
File:
X:\SolarSensor\..\Interface ECU_V1.2.SchDoc
Drawn By:

Bill of Materials

Source Data From: Interface_ECU.PrjPcb
Project: Interface_ECU.PrjPcb
Variant: None

Creation Date: 26.04.2011 13:18:21
Print Date: 40659 40659,58806

Footprint	Comment	LibRef	Designator	Description	Quantity
HDR1X2	9V	Header 2	9V	Header, 2-Pin	1
CAPC2012N	330nF	Cap	C1	Capacitor	1
CAPC2012N	10nF	Cap	C2, C5	Capacitor	2
CAPC2012N	10uF	Cap	C3	Capacitor	1
CAPC2012N	22pF	Cap	C4	Capacitor	1
CAPC2012N	100nF	Cap	C6	Capacitor	1
SOT23_N	BAR43S	BAR43S	D1	BAR43S Schottky Diodes	1
DO-35	1N746A	1N746A	D2	Half Watt Zener	1
HDR2X4_CEN	IF_SS	Header 4X2A	IF_SS	Header, 4-Pin, Dual row	1
HDR1X2	JP0	Header 2	JP0	Header, 2-Pin	1
INDC2012L	Ferrit Bead	Inductor	L2, L3	Ferrit Beat	2
HDR2X4_CEN	Power	Header 4X2A	Power	Header, 4-Pin, Dual row	1
SOT23_N	BC807	BC807	Q1	PNP General-purpose Transistor	1
SOT23_N	BC817	BC817	Q2, Q3	NPN General-purpose Transistor	2
RESC2012N	22k	Res2	R1	Resistor	1
RESC2012N	1.5k	Res2	R2, R6, R7, R8, R10	Resistor	5
RESC2012N	20	Res2	R3, R11	Resistor	2
RESC2012N	10k	Res2	R4, R14	Resistor	2
RESC2012N	3.3k	Res2	R5	Resistor	1
RESC2012N	MLV0805E31103	Res Varistor	R9	Varistor (Voltage-Sensitive Resistor)	1
HDR1X2	SS	Header 2	SS1	Header, 2-Pin	1
MKDSN 1,5/ 2	MKDSN 1,5/ 2-5,(	MKDSN 1,5/ 2-5,(	SS2		1
SO8_N	L78L09ABD	L78L09ABD	U1	Positive Voltage Regulator	1
D014_N	SN74HC14D	SN74HC14D	U2	Hex Schmitt-Trigger Inverter	1
					33

Approved **Notes**