Hes·so VALAIS WALLIS
Haute Ecole d'Ingénierie
Hochschule für Ingenieurwissenschaften

Studiengang Systemtechnik

Vertiefungsrichtung Infotronics

Diplom 2014

Louis Mayencourt

Gasdetektor

Dozent
Joseph Moerschell

Experten
Prof. Hierold | ETH Zürich
Prof. Niebel | FH Jena

Datum der Abgabe des Schlussberichts
31. Juli 2014

Données du travail de diplôme
Daten der Diplomarbeit

☒ FSI ☐ FTV	Année académique / *Studienjahr* **2013/14**	No TD / *Nr. DA* **it/2014/67**
Mandant / *Auftraggeber* ☐ HES—SO Valais ☐ Industrie ☒ Etablissement partenaire *Partnerinstitution*	Etudiant / *Student* **Louis Mayencourt** Professeur / *Dozent* **Joseph Moerschell**	Lieu d'exécution / *Ausführungsort* ☐ HES—SO Valais ☐ Industrie ☒ Etablissement partenaire *Partnerinstitution*
Travail confidentiel / *vertrauliche Arbeit* ☐ oui / ja [1] ☒ non / nein	Experts / *Experte (données complètes)* **Prof. Hierold** | ETH Zürich **Prof. Niebel** | FH Jena	

Titre / *Titel*

Gasdetektor

Description / *Beschreibung*

Man kann Karbon-Nanoröhrchen für eine empfindliche Detektion der Konzentration gewisser Gase einsetzen. Aufgabe dieses Projektes ist es, eine elektronische Baugruppe zu entwickeln und zu erproben, welche die Konzentration von Stickoxyd ermittelt mithilfe von Sensorprototypen, welche an der ETH Zürich hergestellt wurden.

Zur Erprobung der Schaltung wird der Sensor durch eine äquivalente Schaltung mit festen elektronischen Bauteilen simuliert. Neben einer genauen Erfassung des Kanalstroms des Nanoröhrchens besteht die Herausforderung für die Schaltung in der starken Streuung gewisser Kenngrössen der Sensoren von einem Exemplar zum anderen.

Aufgrund des geringen Leistungsbedarfs des Sensors eignet er sich für Überwachungsaufgaben, z. B. der Umwelt, in batteriegetriebenen Geräten.

Objectifs / *Ziele*

Die folgenden Arbeiten sollen durchgeführt werden :

— Entwicklung der Schaltung zur Simulation des Gasdetektors, Herstellung und Erprobung der Funktion
— Erstellen eines Testplans um die Funktion und Leistungsmerkmale der Auswerteschaltung des Gasdetektors zu prüfen.
— Programmierung und Test der Software für die Auswerteschaltung, zum Durchlaufen der Messzyklen, Erfassung der Messdaten und Kommunikation mit einem PC.
— Prüfung der Schaltung gemäss Testplan.
— Diskussion der Ergebnisse und Erstellen eines Pflichtenhefts für eine integrierte Auswerteschatlung.

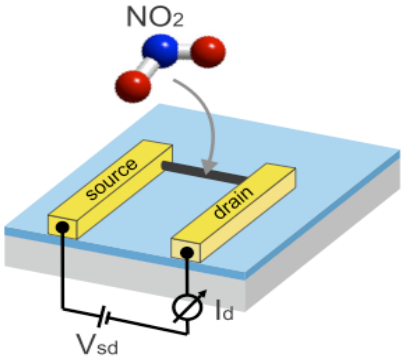
Signature ou visa / *Unterschrift oder Visum* Responsable de l'orientation *Leiter der Vertiefungsrichtung:* ..... [1] Etudiant / Student :	Délais / *Termine* Attribution du thème / *Ausgabe des Auftrags:* 13.05.2014 Remise du rapport / *Abgabe des Schlussberichts:* 31.07.2014 Expositions / *Ausstellungen der Diplomarbeiten:* 27 – 29.08.2014 Défense orale / *Mündliche Verfechtung:* Semaine | Woche 36

[1] *Par sa signature, l'étudiant-e s'engage à respecter strictement la directive DI.1.2.02.07 liée au travail de diplôme.*
Durch seine Unterschrift verpflichtet sich der/die Student/in, sich an die Richtlinie DI.1.2.02.07 der Diplomarbeit zu halten.

Rapport reçu le / *Schlussbericht erhalten am* ..... Visa du secrétariat / *Visum des Sekretariats* .....

Gasdetektor

Diplomand/in Louis Mayencourt



Ziel des Projekts

Das Ziel dieser Arbeit ist einen NO2 Gasdetektor auf der Grundlage eines Sensorprototypen, welche an der ETH Zürich hergestellt wurde erreichen.

Methoden | Experimente | Resultate

Mann kann Karbon-Nanoröhrchen für eine empfindliche Detektion der Konzentration gewisser Gase einsetzen. Die Einführung dieser Nanoröhren in einem Feldeffekttransistor erlaubt ein gassensitiven Element zu erhalten. In Kontakt mit dem Gas wird die Kennlinie des Transistors nach links verschoben. Ein Maß für dieses Kennlinie erlaubt es, die Konzentration von Gas vorhanden beurteilen.
Die Messung ist mit einer bestimmten Messsequenz des Gate des Transistors getan, um den Einfluss von äußeren Parametern, wie Feuchtigkeit zu minimieren. Jedes Kennlinie wird dann gemessen und sind mit drei verschiedenen Methoden analysiert, und das Ergebnis wird an eine Benutzerschnittstelle übertragen.
Eine äquivalente Sensor Schaltung ist entwickelt, um die Funktion der Messschaltung zu testen. Sie wird mit festen Elektronischen Bauteilen realisiert.
Das Ergebnis dieser Arbeit ist ein Funktionel Schaltung mit einer Benutzeroberfläche, die der Transistorcharakteristik korrekt visualisieren kann.

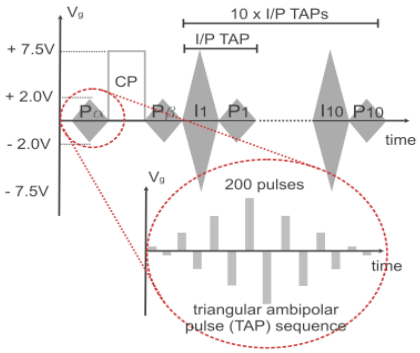
Diplomarbeit
| 2014 |

Studiengang
Systemtechnik

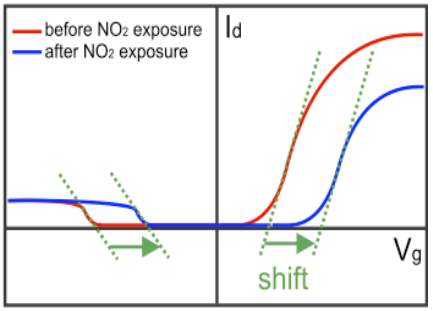
Anwendungsbereich
Infotronic

Verantwortliche/r Dozent/in
Dr Joseph Moerschell
Joseph.Moerschell@hevs.ch

Partner
Ernst-Abbe Fachhochschule Jena



Figur: Messsequenz



Figur: Wirkung von Gas auf der Kurve

INHALTSVERZEICHNIS

1 EINLEITUNG

NO_2 ist ein rötlich-braunes, giftiges Gas, welches vor allem von Motoren und in Kraftwerken produziert wird. Es wird in der Industrie zur Produktion von Salpetersäure HNO3 eingesetzt und ist eine der wichtigsten Schadstoffe der Atmosphäre. Er ist für den sauren regen verantwortlich und kann beim Einatmen zur Entzündung der Atemwege und zu weiteren Gesundheitsschäden führen. Es ist relativ schwierig, einen NO2-Sensor zu implementieren, da die zu erfassenden Menge in der Regel gering (in der Größenordnung von 20 bis 200 ppm) ist.

Mann kann Karbon-Nanoröhrchen für eine empfindliche Detektion der Konzentration gewisser Gase einsetzen. Die Einführung dieser Nanoröhren in einem Feldeffekttransistor erlaubt ein gassensitiven Element zu erhalten. In Kontakt mit dem Gas wird die Kennlinie des Transistors nach links verschoben. Die Größe der Verschiebung ist ein Maß für die Konzentration des Gases.

Dieser Bericht ist eine Fortsetzung des Semesterprojekts. Er beschreibt die verschiedenen Schritte, um ein funktionsfähiges Softwaresystem zu erreichen. Ein erster Analyse Schritt ist, um die Bedürfnisse zu erkennen und das notwendig Wissen erwerben, um das Projekt durchzuführen. Dann kommt ein Entwicklungsphase. In dieser Phase werden die Elemente der Analyse wieder gefasst und für dem Projekt erarbeitet. Schließlich wird eine Reihe von Tests durchgeführt und dokumentiert werden, um den Betrieb des Systems zu überprüfen.

Die entwickelt Teile hier sind das Programm für die Prozessor Karte und die Entwicklung eines fiktiven Sensors, um die Schaltung zu testen. Das Ausführung der Benutzerschnittstelle, und die Kommunikation mit der Karte sind auch in dem Bericht enthalten. Die Voruntersuchung und die Entwicklung von elektronischen Karte sind die Objekte der Arbeit Semester.

1.1 Danksagung

Ich bedanke mich herzlich bei allen Personen, welche dieses Projekt mit ihrem Wissen und ihren Erfahrungen unterstützt haben. Namentlich erwähnt seien hier:

- Dr. Dominique Gabioud, für die Erlaubnis, das Labor scada in dieser Arbeit verwenden
- Herrn Medard Rieder, für die Erlaubnis, das Labor scada und das XF in dieser Arbeit verwenden
- Dr. Niebel, für seine Unterstützung während der Arbeit

Weiter bedanken ich mich bei Herrn Joseph Moerschell für die erhaltene Möglichkeit dieses Projekt im Rahmen des „Projet de diplome“ (PRd) an der Hochschule Hes-so in Sion durchzuführen.

2 MANAGEMENT

Für die Diplomarbeit, sollen die folgenden Arbeiten durchgeführt werden :

- Programmierung und Test der Software für die Auswerteschaltung, zum Durchlaufen der Messzyklen, Erfassung der Messdaten und Kommunikation mit einem PC.
- Entwicklung der Schaltung zur Simulation des Gasdetektors, Herstellung und Erprobung der Funktion.
- Erstellen eines Testplans um die Funktion und Leistungsmerkmale der Auswerteschaltung de Gasdetektors zu prüfen.
- Prüfung der Schaltung gemäss Testplan.
- Diskussion der Ergebnisse und Erstellen eines Pflichtenhefts für eine integrierte Auswerteschaltung.

2.1 Pflichtenheft

Die Spezifikation wird von den Daten der Semesterprojekt definiert.

- Die Entwicklung eines modularen Software, um zukünftige Verbesserungen durchzuführen.
- Die Messparameter müssen leicht zu ändern.
- Die Software-Entwicklung sollte die Implementierung der Schaltung in einem System zu erleichtern.
- Die Benutzeroberfläche muss einfach zu bedienen sein.
- Die Dummy-Sensor muss das Verhalten eines CNFET simulieren.

2.2 Zeitplan

Die Einrichtung eines Zeitplans ist notwendig, um einen guten Überblick über das Projekt zu haben. Es definiert die wichtigen Aufgaben zu erfüllen, und ihre Dauer.

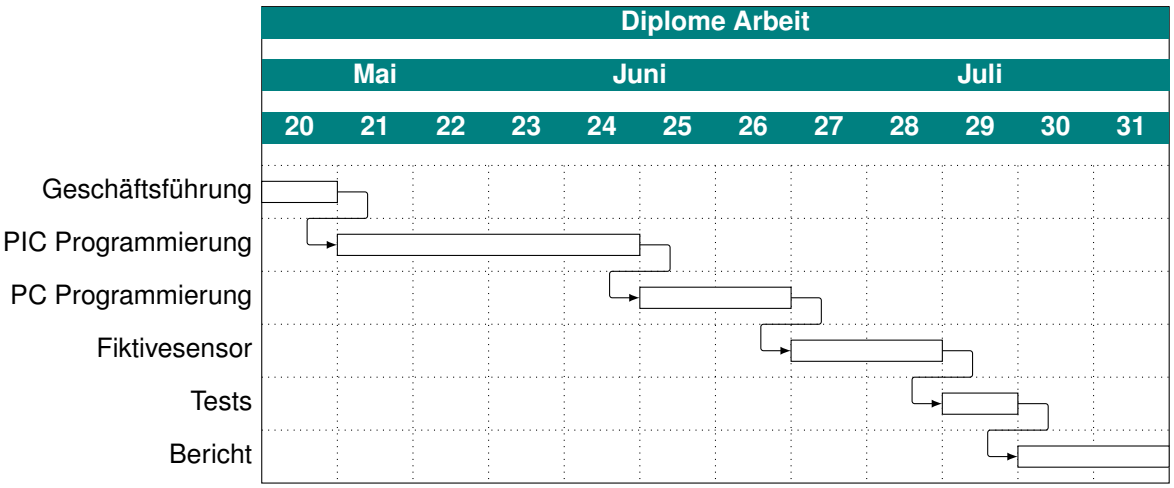


Abbildung 1: Zusammenfassung des Zeitplans

Dieser Zeitplan ist nur eine Zusammenfassung. Der gesamte Zeitplan ist im Anhang (siehe Anhang A) beigefügt.
Die erste Woche ist das Projektmanagement gewidmet. Prozessor-Programmierung wird in den folgenden Wochen durchgeführt. Als nächstes kommt der Programmierung des Benutzerschnittstelle. Der Gesamtbetrieb des Systems wird getestet und der Bericht wird geschrieben.

2.3 Verwendete Software

Alle Software in diesem Projekt verwendet werden, sind frei.

- Die Prozessor Programmierung ist mit MPLAB X IDE durchgeführt. Dieses kostenlos Programm steht vor Web page von Mircochip [1]. Der Compiler XC16 ist auf der Microchip Seite [2].
- Das Computerprogramm ist in Java mit Eclipse implementiert. Diese IDE ist auf ihrer Website zur Verfügung [3].
- Die UML Modellierung ist mit Umlet [4] durchgeführt. Diese kleine Software kann einfach UML-Diagramme zeichnen.
- Die graphischen Teil der Website ist mit den Bibliotheks Highcharts [5] gemacht.
- Der Bericht wird mit LaTeXgeschrieben.

[1] http://www.microchip.com/pagehandler/en-us/family/mplabx/
[2] http://www.microchip.com/pagehandler/en_us/devtools/mplabxc/
[3] http://www.eclipse.org/downloads/
[4] http://www.umlet.com
[5] http://www.highcharts.com

3 ZUSAMMENFASSUNG DER ARBEIT DES SEMESTERS

Der Sensor, in dieser Arbeit verwendet ist ein CNFET (*Carbon Nanotube Field Effect Transistor*). Das NO_2 Gas wird durch den Transistor absorbiert, wodurch sich seine Kennlinie ändern.

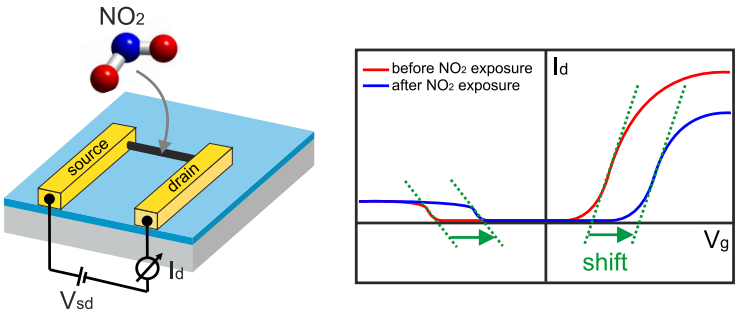


Abbildung 2: Wirkung der NO_2 Gas auf die Kennlinie [4]

Die Absorption des Gases verschiebt die Kennlinie nach rechts. Es ist daher möglich, die Konzentration des Gas aus die analyse des Charakteristik zu beurteilen. Es gibt drei Methoden, um die Konzentration zu bewerten.

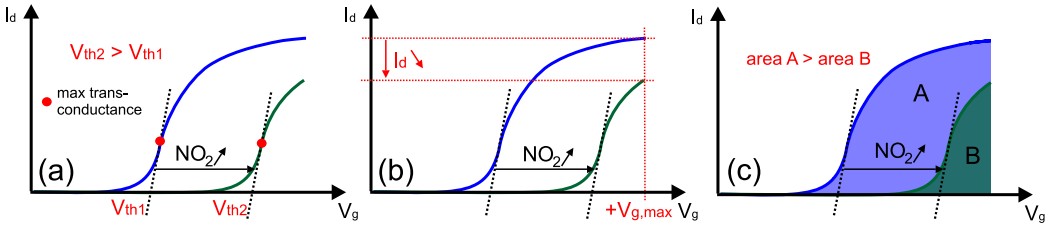


Abbildung 3: Messverfahren [4]

Die erste Methode besteht darin, den Wert von V_{th} extrahieren. Diese Methode aufgerufen ROS1 für den Rest des Dokuments.
Die zweite Methode ist, den Stromwert zu messen, wenn die Gate-Spannung maximal ist. Diese Methode aufgerufen ROS2 für den Rest des Dokuments.
Das letztere Verfahren ist das Integral der charakteristischen zu berechnen. Diese Methode aufgerufen ROS3 für den Rest des Dokuments.
Das stabilere Verfahren ist das letztere. Da es ein Prototyp werden die drei Verfahren implementiert. Es wird möglich sein, um die Entwicklung von drei verschiedenen Verfahren sichtbar zu überwachen.

Der Transistor weist eine Hysterese in der Charakteristik, wenn eine Messung ausgeführt ist. Es einen schlechten Einfluss auf die Messung hat. Um diesen Effekt zu entfernen, wird einen bestimmten Gate Stimulation verwenden sollen. Diese Stimulation wird durch die unten dargestellte Bild.

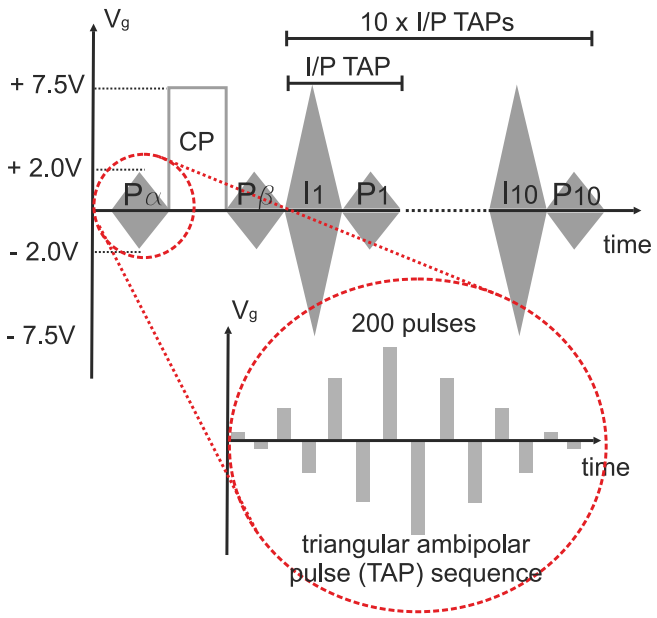


Abbildung 4: Gate Stimulation [4]

Vor jeder Messung (Px) wird eine Initialisierung Sequenz (Ix) ausführen. Mit dieser Methode wird die Kennlinie von jedem Hysterese freigegeben.

Um korrekte Messungen durchzuführen, wurde eine Schaltung mit der folgenden Architektur entwickelt :

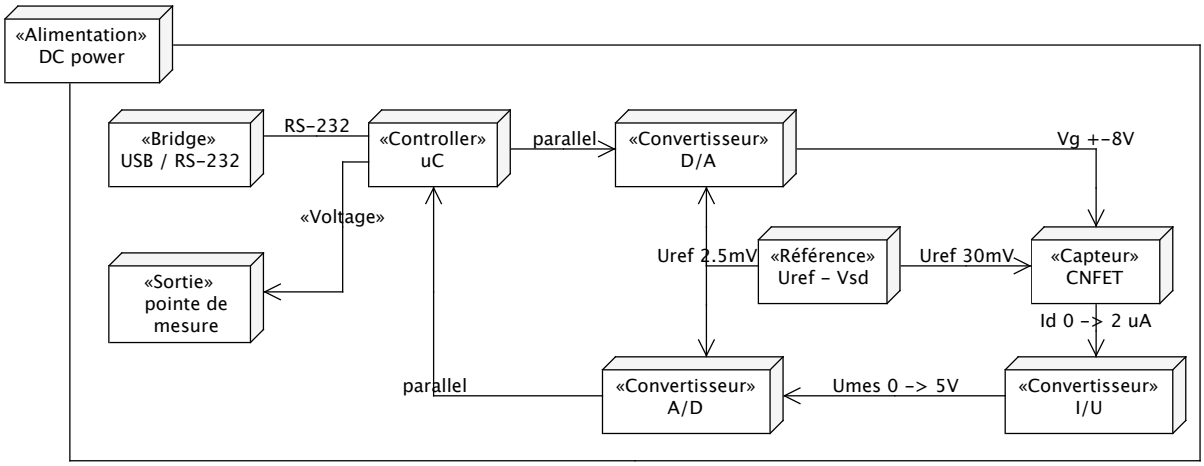


Abbildung 5: Blockschema

Weitere Informationen finden Sie im Bericht der Arbeits des Semesters [3] und in der Diplomarbeit von Herr Moritz P.P. Mattmannn [4].

4 ANALYSE

Vor Beginn der Entwicklung, ist ein erste Schritt von Dokumentation und Analyse notwendig. Dieser Abschnitt enthält die grundlegenden Elemente für das Verständnis notwendig Entwicklung.

4.1 Verteilungen

Das System enthält eine elektronische Schaltung mit einem Computer verbunden. Die Kommunikation zwischen dem Computer und der Schaltung ist mit RS-232 innerhalb USB Frames gemacht. Die Schaltung enthält eine Softwarekomponente um die Messung und die Kommunikation auszuführen. Der Computer enthält auch eine Softwarekomponente um den Beaufsichtigung und die Kommunikation auszuführen.

Abbildung 6: Verteilungen

In Figur 6, das HMI ist eine Applikation in Java oder QT. Die Benutzerschnittstelle und die Kommunikationsschicht sind in einem Programm. Um eine bessere Baukastenprinzip zu haben, ist es besser die Schnittstelle getrennt. Sie wird als eine Web-Seite ausgeführt. Dass hießt, dass mehrere Kunden gleichzeitig verbinden können und sie ist unabhängig von einer Plattform.

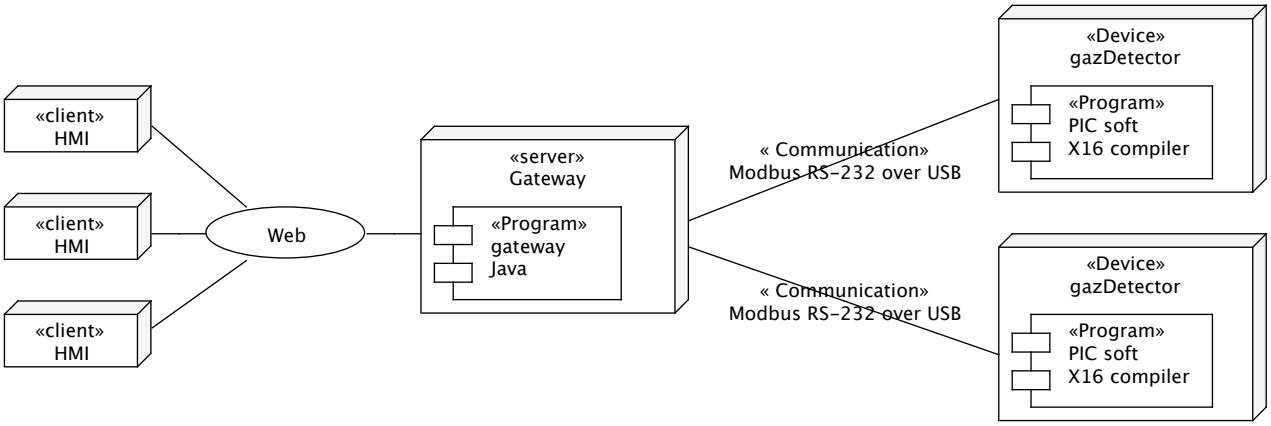


Abbildung 7: Letzte Verteilungen

Die Verwendung eines standardisierten Protocol zwischen dem Server und der Schaltung macht das System unabhängiges. Die Schaltung kann mit einem anderen Schnittstelle, das die Protokoll implementiert verwendet werden. In unserem Fall, die gewählte Protokoll ist Modbus.

4.2 Benutzerschnittstelle

Der Benutzer gelangt zum Sensordaten über eine Webseite. Die Schnittstelle Funktionalität werden über ein Anwendungsfalldiagramm definiert.

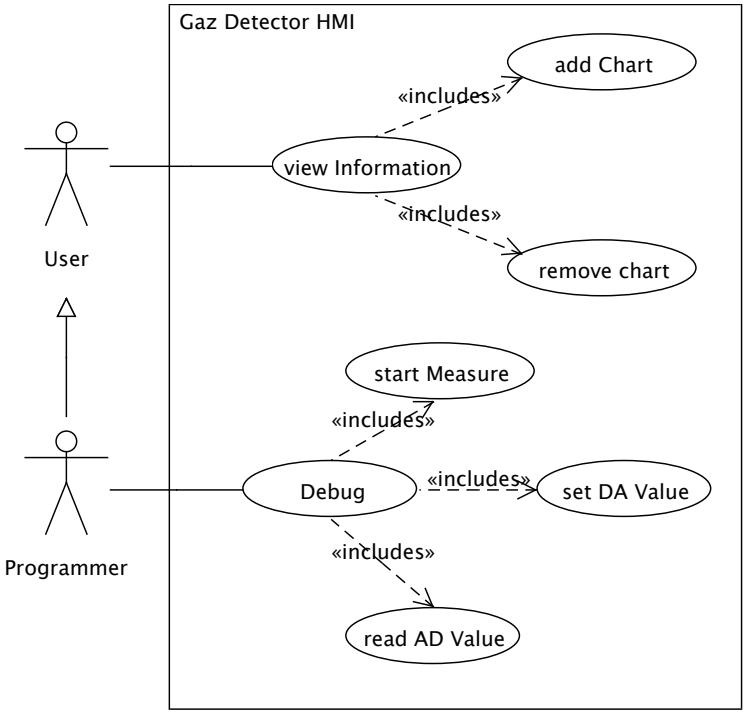


Abbildung 8: Anwendungsfalldiagramm

Ein normaler Benutzer kann nur Graphen hinzufügen oder löschen. Wenn der Benutzer ein neuer Grafik fügt hinzu, kann er die Messverfahren und den Filter wählen.
Ein zweites Fenster ermöglicht den Zugriff auf den Debug-Modus von einem Programmer Benutzer. In diesem Modus, ist es möglich den Sensor zu steuern. Steuerungsmöglichkeiten sind: der Wert des AD-Wandlers, der Wert des DA-Wandlers und der Erzeugung einer Messsequenz.

4.2.1 Kommunikation

Die Kommunikation zwischen der Benutzeroberfläche und dem Server wird mit einem websocket getan. Das verwendete Protokoll ist in der Dokumentation des Super-Labor scada (siehe Anhang B) definiert.

Type	Parameters
SubscribeRequest	id
SubscribeResponse	id, status, value
ValueChangeIndication	id, value
GetRequest	id, serial
GetResponse	id, serial, value, status
SetRequest	id, serial, value
SetResponse	id, serial, status
SignalIndication	id, text, raised
AlarmIndication	id, text, active, raised, acknowledged
AlarmAcknowledge	id
AlarmAcknowledgeIndication	id, acknowledged

Abbildung 9: Zusammenfassung von der WebSocket Protokoll

Diese Nachrichten werden mit dem JSON-Format ausgetauscht. Dies vereinfacht die Entwicklung.

4.2.2 Webseite

Die verwendete Sprache um die Website zu implementieren ist HTML. Es ist eine Tagssprache. Sie gehört zur gleichen Familie wie XML. Für dieses Projekt, nur ein paar Grundelemente verwendet werden :

<titel> Dieser Tag ermöglicht Ihnen das Erstellen einer Titel auf dem Web-Browser.

Beispiel :

```
<title> Gaz Detector HMI </title>
```

** ** Definiert einen Abschnitt in einem HTML-Dokument. Es wird als Behälter für die dynamischen Elemente verwendet.

Beispiel :

```
<div id="graphContainer"> </div>
```

<h1> Dieser Tag ermöglicht Ihnen das Erstellen einer Titel für einen Absatz in der Seite.

	Ergebnis:	Quelle:
Beispiel :	**My Title** the rest of the file	`<h1>My Title</h1>` `the rest of the file`

<tab> Dieser Tag ermöglicht Ihnen das Erstellen eine Tabelle. Die `<tr>` Tag wird verwendet, um Zeilen zu schaffen und `<td>` Tag für Spalten.

Ergebnis:

Beispiel :

cell 1	cell 2
cell 3	

Quelle:

```
<table>
    <tr>
    <td>
        cell 1
    </td>
    <td>
        cell 2
    </tr>

    <tr>
    <td>
         cell 3
    </td>
    </tr>
</table>
```

<fieldset> Dieser Tag ermöglicht Ihnen das Erstellen einer Gruppe. Es wird mit `<legend>` Tag verwendet, um den Gruppennamen anzugeben.

Ergebnis:

Beispiel :

A group
Hello World!

Quelle:

```
<fieldset>
    <legend> A group </legend>
    <input type="submit" value="Hello World!">
</fieldset>
```

<img> Dieser Tag ermöglicht die Einführung der Bilder im Dokument. Es wird mit den folgenden Parametern verwendet :

Name	Beschreibung
src	der Pfad von der Quelldatei
alt	Bildbeschreibung
width	die Breite des Bild
height	Die Höhe des Bildes
align	die Position des Bild

Abbildung 10: Beschreibung des <img> Tag

Ergebnis:

Beispiel :

Hes·so VALAIS WALLIS

Quelle:

```
<img src="img/hes_so.png" alt="hes-so" width="140" height="30" align="
    left">
```

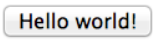
<input> Dieser Tag kann verschiedene Eintrag Art für den Benutzer zu erstellen. Es wird mit den folgenden Parametern verwendet :

Name	Beschreibung
id	Die Kennung, um die Informationen zu abrufen
type	der Eingabetyp (Knopf, Texte, ...)
value	der Text des Objekts
onclick	Funktion, die beim Klicken auf

Abbildung 11: Beschreibung des <input> Tag

Beispiel :

Ergebnis:



Quelle:

```
<input type="submit" value="Hello world!" onclick="buttonClicked()">
```



```
<input type="text" value="Hello world!">
```

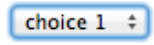
<select> Dieser Tag ermöglicht eine MehrfachauswahlFeld. Es wird mit `<option>` Tag verwendet, um die Auswahl zu erstellen Es wird mit den folgenden Parametern verwendet :

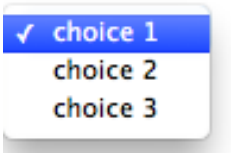
Name	Beschreibung
id	Die Kennung, um die Informationen zu abrufen
name	der Name des Objekts

Abbildung 12: Beschreibung des <img> Tag

Beispiel :

Ergebnis:





Quelle:

```
<select name="My_choice" id="choice">
    <option value="1">choice 1</option>
    <option value="2">choice 2 </option>
    <option value="3">choice 3</option>
</select>
```

<a> Dieser Tag ermöglicht Ihnen URL Link in das Dokument einbetten.

Beispiel :

Ergebnis:

Google

Quelle:

```
<a href="http://www.google.com">Google</a>
```

<script> Dieser Tag ermöglicht Ihnen Javascript-Code in das Dokument einbetten. Javascript wird, um das Verhalten von Website zu implementieren verwendet.

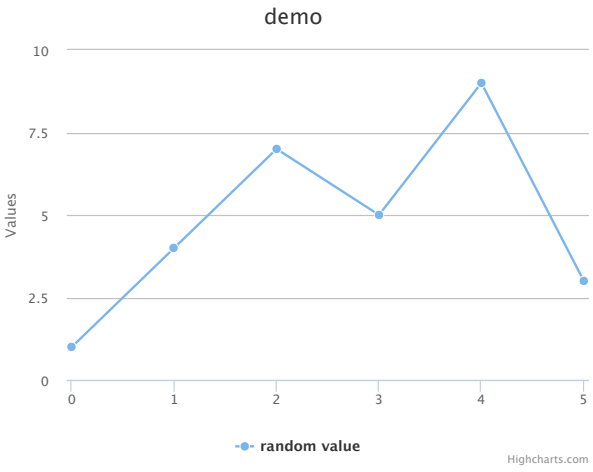
Beispiel :

```
<script language="javascript" type="text/javascript">
```

Grafik Grafiken sind mit der javascript-Bibliothek *Highchart* gemacht. Diese Bibliothek erlaubt es Ihnen, interaktive Grafiken erstellen. Jedes Grafiken wird in einem `<div>` Behälter enthaltenen.

Beispiel :

Ergebnis:



Quelle:

```
new Highcharts.Chart({
        title: {
            text: 'demo'
        },
        series: [{
            name : 'random value',
            data: [
                1,4,7,5,9,3
            ]
        }]
    });
```

4.3 Server

Der Server-Teil wurde nicht während dieses Projektes entwickelt. Es ist aus dem SCADA Super-Lab genommen. Dieses Labor gehört von dem Vorlesung des drittes Jahr in den Electronic Abschnitt. Es erlaubt Ihnen ein System, das die folgende Hierarchie aufweist überwachen.

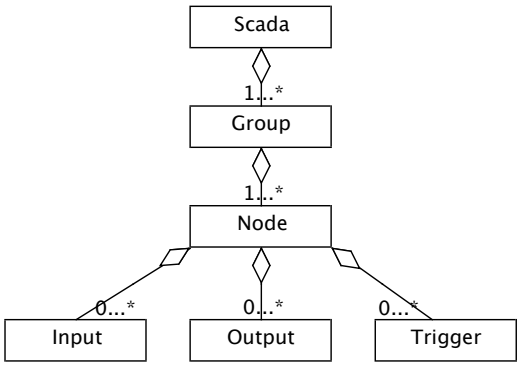


Abbildung 13: Scada Hierarchie

Die höchste Hierarchiestufe ist die Gruppe. Eine Gruppe enthält eine oder mehrere Knoten. Jeder Knoten enthält eine Reihe von Ein-und Ausgangs und Alarme.

Die Serverstruktur kann mit dem folgenden Paketdiagramm zusammengefasst werden :

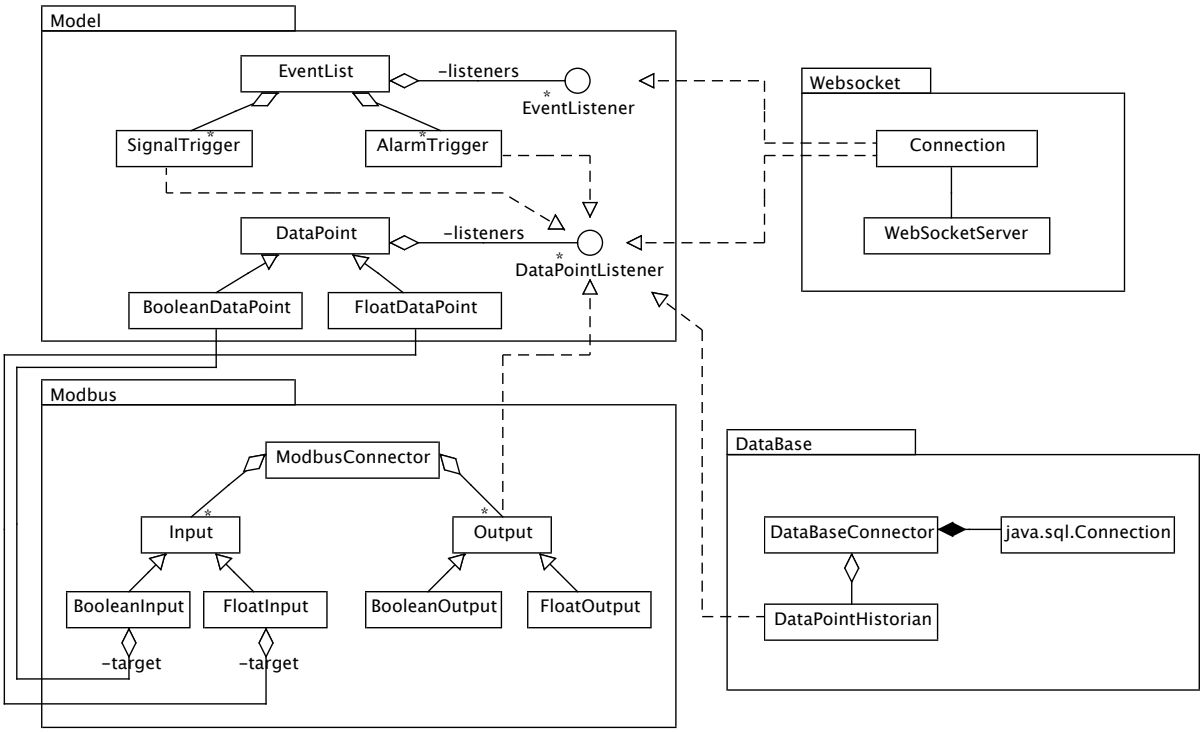


Abbildung 14: Paketdiagramm für PC

Der Server enthält vier verschiedene Paket: Modbus, Server, Database und Model.
Modbus-Paket enthält Klassen für die serielle Kommunikation mit der Karte. Die vier Klassen *BooleanInput*,*FloatInput*, *Booleanoutput* und *Floatoutput* darstellt das Modell der Modbus-Daten.
Der Modell Paket enthält die Server-Datenmodell. Jedes Modbus Datenmodell ist mit einem Datenpunkt verbunden. Diese Datenpunkte implementieren das Beobachter-pattern. Sie können daher die beobachter Objekte von eine Wertänderung informieren. Die Datenpunkte können Alarme oder Signale darstellen. Ein signal oder ein Alarm wird erzeugt, wenn der Wert des Datenpunktes gleich als seine Trigger ist.
Der Database Paket enthält eine Datenbank SQL-Kunden Klasse. Die Datenbank wird verwendet, um die Datenpunkt zu schaffen, und die alten Werte zu speichern. Alten Werte werden von dem Historiker Klasse, die das Beobachter-pattern implementiert eingeführt.
Und schließlich wird der websocket Packet, der die Kommunikation Klassen mit der Benutzerschnittstelle enthält. Jedes Fenster schafft eine neue Verbindung, die das Beobachter-pattern implementiert um die neue Werte zu bekommen.

4.3.1 XML Beschreibung

Das Modell der Server ist am Anfang leer. Beim Start wird es für die Informationen in der Datenbank suchen. Die Datenbank muss eine Beschreibung des Systems, die damit verbunden sind enthalten. Diese Beschreibung wird mit einem XML Dokument realisieren. Eine XSLT-Transformation übersetzt die XML-Beschreibung in SQL-Abfrage, um die Datenbank zu füllen.

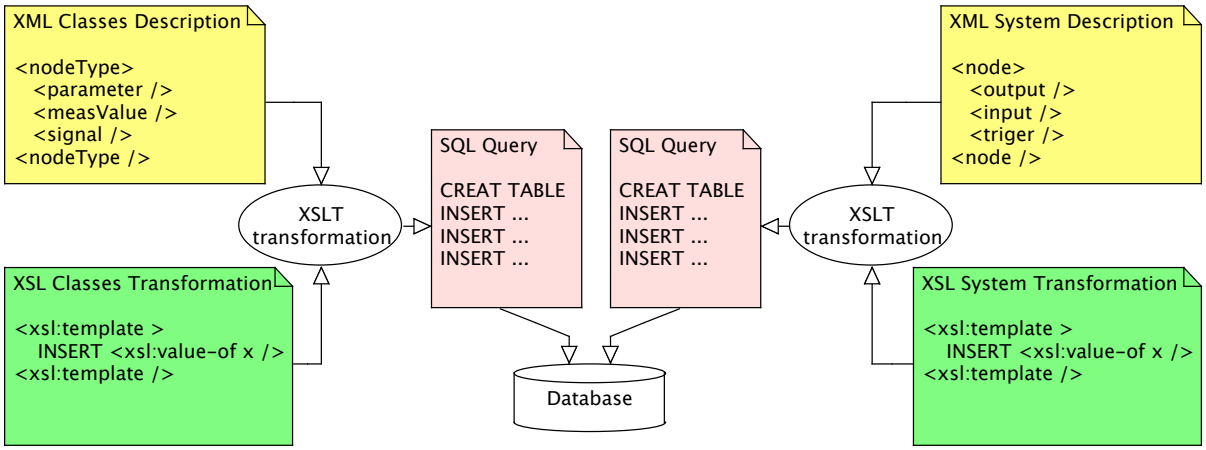


Abbildung 15: XML - SQL Übersetzung

Es gibt zwei verschiedene XML-Dokument. Der *GazDetectorClasses.xml* Datei enthält die allgemeine Beschreibung des angeschlossenen System. Die Einheiten und Art der Ein-und Ausgänge sind dorthin definiert. Jeder Eingang entspricht einem Punkt in der `measValue` xml tag. Jeder Ausgang entspricht einem Punkt in der `parameter` xml tag.
Der *GazDetectorSystem.xml* Datei enthält spezifische Beschreibung des Ein-und Ausgänges. Der Modbus-Adresse und die Zugriffsfunktion sind dorthin definiert. Jeder `parameter` xml tag in der *GazDetectorClasses.xml* Datei wird ein `output` xml tag. Und jeder `measValue` tag wird in ein `input` tag.

4.3.2 Database

Die Datenbank ist in einem MySQL-Server gespeichert. Dieser Server kann lokal oder entfernt sein. Der Anhang C beschreibt, wie eine MySQL-Server auf einem Linux-System installieren. Der Server enthält die folgenden Tabellen :

Name	Beschreibung
Group	enthält die verschiedenen Gruppen
Node	enthält den KnotenName der in den Gruppen enthaltenen
NodeType	enthält die Nodetypen
Field	enthält die verschiedenen Ein-und Ausgänge
Point	enthält die Beschreibung von die verschiedenen Ein-und Ausgänge
Datatype	enthält die Datentypen verwendet
Unit	enthält die Beschreibung der verwendeten Einheiten
Historian	enthält die alten Messwerte

Abbildung 16: Zusammenfassung von der tabellen des Datenbank

Der Anhang B enthält das komplette Beziehungsdiagramm.

4.4 Eingebettet System

Der Sensor ist die Schaltung während der Semester Projekt entwickelt. Seine Architektur kann in der Blockschema 5 zusammengefasst werden.

Um die Softwarearchitektur zu erstellen, wird jeder Blöcke darum den Prozessor in einer Klasse zu erstellen genommen. Diese Klassen werden in der Hardware-Paket enthalten. Dieses Paket ist die Hardware-Abstraktionsschicht.
Ein weiteres Paket wird erstellt, um die Anwendungsschicht enthalten. Es enthält drei Klassen. Eine Kommunikationsklasse für die serielle Kommunikation mit dem Computer. Eine Erfassungsklasse, um Messung zu nehmen und eine Controllerklasse, um alles zu kontrollieren.
Eine letzte Paket enthält das XF (eXecution Framework), dem Betriebssystem der Karte. Die XF erlaubt Zustandsmachine zu implementieren, die die Programmierung vereinfacht. Alle Pakete sind verbunden, um miteinander zu kommunizieren.

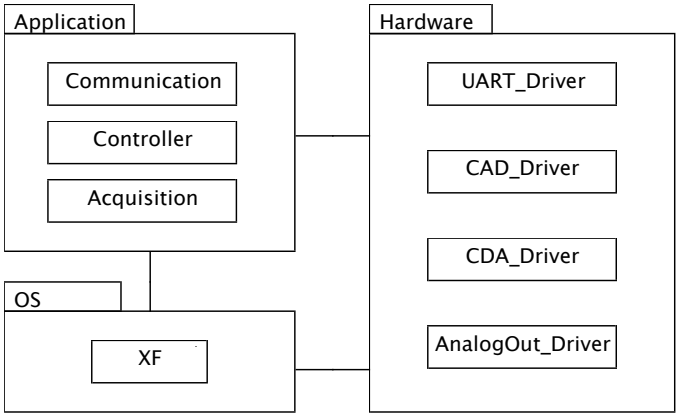


Abbildung 17: Paketdiagramm für den PIC

4.4.1 Controller

Die `Controller` klasse steuert den Gesamtbetrieb der Schaltung. Er empfängt Befehle von der Kommunikation Klasse und aktive die Erfassung Klasse. Es enthält auch die Methoden zur Berechnung der Ausgangswerte.

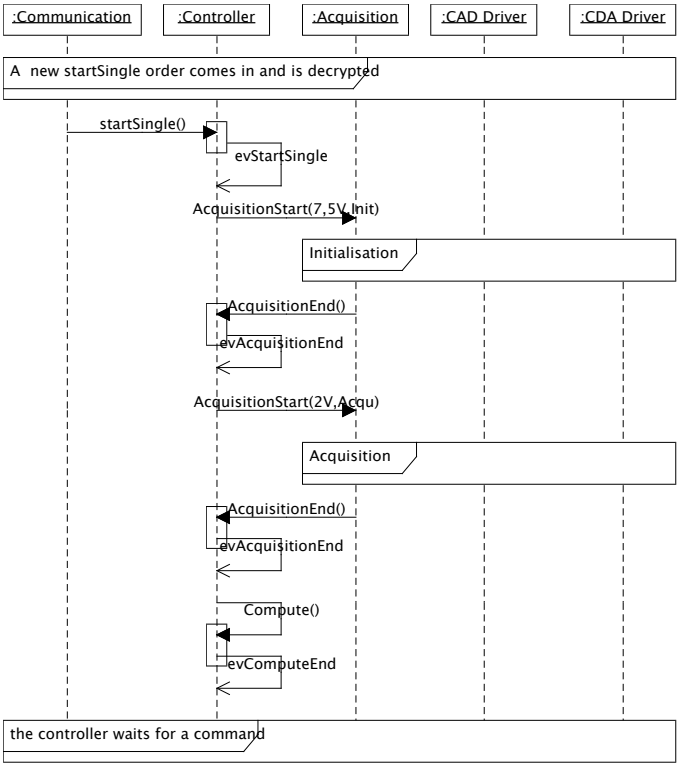


Abbildung 18: Sequence Diagramm der Controller

der Controller enthält zwei verschiedene Betriebsarten. Einen Normalmodus, die kontinuierlich Messungen durchführt. Und eine Debug-Modus, um den Betrieb der Schaltung überprüfen.

4.4.2 Erfassung

der `Erfassung` Klasse erzeugt die Stimulation Reihenfolge des Transistors. Code ist derselbe für die Erfassung oder die Initialisierung. Während der Initialisierung wird die Anzahl von Proben einfach auf 0 gesetzt. Die Zeiten werden mit den Hardware-Timer erzeugt.

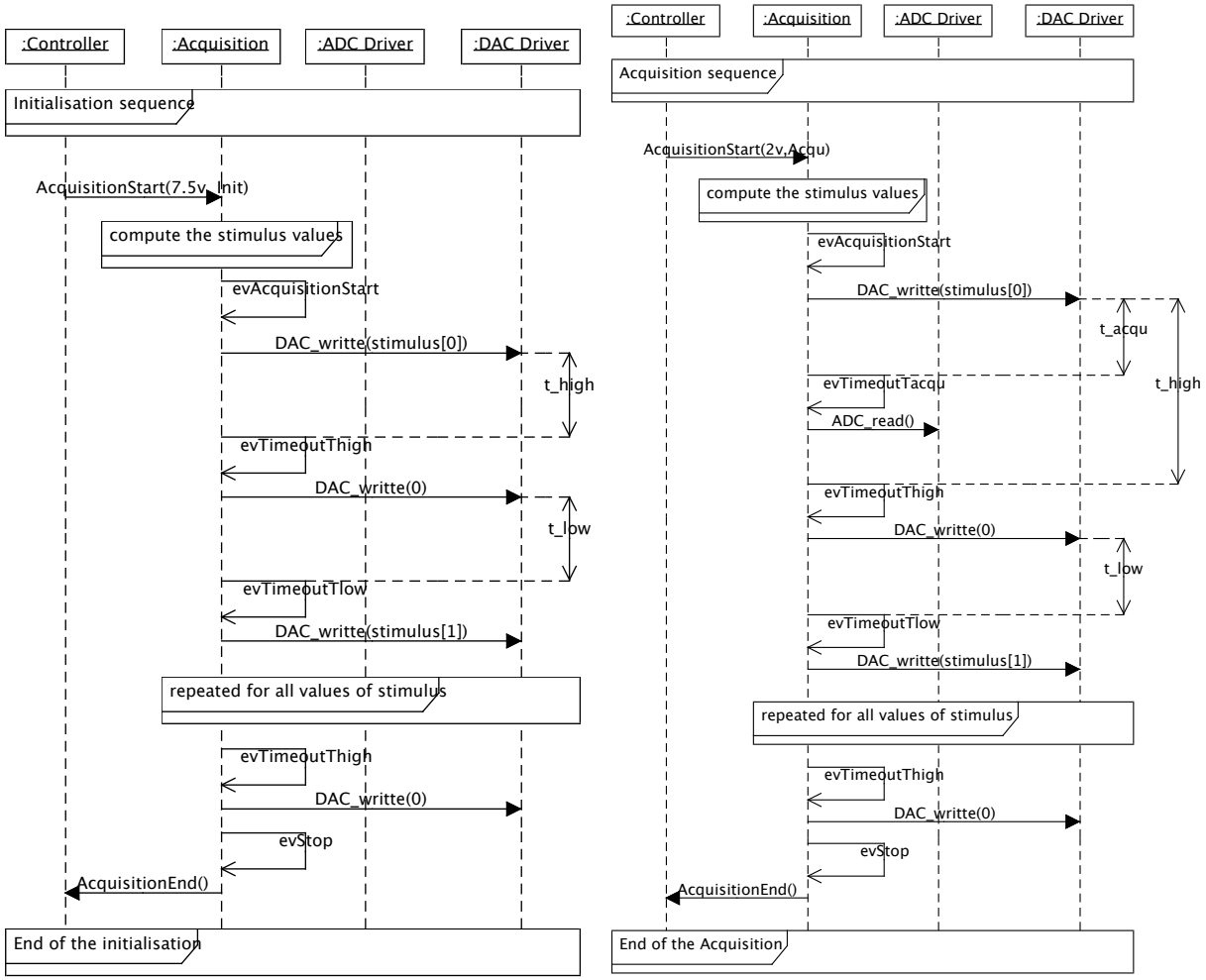


Abbildung 19: Sequence Diagramm der Erfassung

Für jede Sequenz wird die Werte der Gatespannung auf der Grundlage der übergebenen Parameter berechnet. Mit der Geschwindigkeit des Wandlers ist es möglich, mehrere Proben pro Stimulus nehmen. Die Anzahl der Proben kann durch eine Funktion modifiziert werden. Ein weiterer wichtiger Parameter ist die Zeit t_{high}, die auch modifiziert sein können.

4.4.3 Kommunikation

Die Kommunikation zwischen dem Server und die Schaltung wird über die RS-232 realisieren. Um die Verbindung zu vereinfachen, wird die RS-232 in USB Frames eingekapselt. Die usb Verwaltung ist transparent aus der Sicht der Programmierung.
Das verwendete Protokoll ist Modbus. Dies ist ein Kommunikationsprotokoll der Anwendungsebene (Ebene 7)

des OSI-Modells. Modbus definiert die Ebene 7 [7] und 2 [6] des OSI-Modells. Ein Frame besteht normalerweise aus folgenden Elemente :

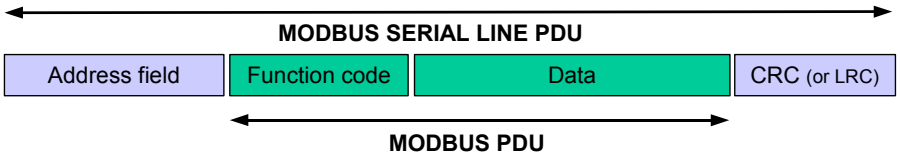


Abbildung 20: Modbus frame [7]

Das Modell der Kommunikation ist der Client-Server Modell. Für die Kommunikation über eine serielle Leitung, gibt es einem Master und mehreren Slaves. Der Master als Client agiert, und Slave als Server. Der Master muss die Sklaven regelmäßig fragen, um diese Daten auf dem neuesten Stand zu sein.

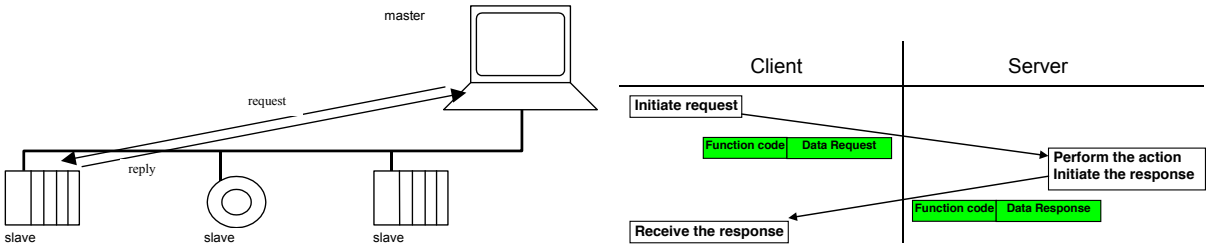


Abbildung 21: Master-Client / Sklaven-Server Model [7]

Die Sklave Adressen werden in einem Byte kodiert. Es ist möglich, 255 Server in derselben Leitung verbunden. Der Funktionscode ist auch eine Zahl zwischen 0 und 255. Für unsere Anwendung wird nur eine geringe Anzahl von Funktions erforderlich. Übertragungssicherheit wird mit einer CRC am Ende des Rahmens gewährleistet

Slave Address	Function Code	Data	CRC
1 byte	1 byte	0 up to 252 byte(s)	2 bytes CRC Low CRC Hi

Abbildung 22: Modbus frame für eine serielle Kommunikation [6]

Modbus bietet zwei Modus der seriellen Übertragungen : RTU und ASCII. RTU-Modus wird verwendet, weil es einen größeren Rate hat. Jedes Charakter enthält ein Startbit, 8 Datenbits und 2 Stopbits.

Start	1	2	3	4	5	6	7	8	Stop	Stop

Abbildung 23: Modbus Timing [6]

der Frame Differenz wird mit Timing durchgeführt. Jeder Charakter auf dem gleichen Frame gehören, müssen in 1,5 Charakter übertragen werden. Jeder Frame wird durch eine Zeit größer als 3,5 Charakter getrennt.

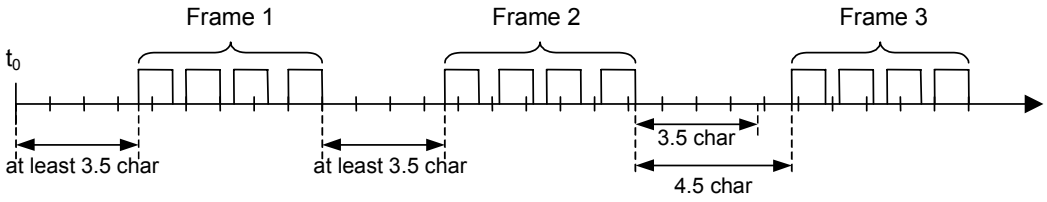


Abbildung 24: Modbus Charakter [6]

Das Modbus Protokoll enthält vier Typen von Daten :

Primary tables	Object type	Type of	Comments
Discretes Input	Single bit	Read-Only	This type of data can be provided by an I/O system.
Coils	Single bit	Read-Write	This type of data can be alterable by an application program.
Input Registers	16-bit word	Read-Only	This type of data can be provided by an I/O system
Holding Registers	16-bit word	Read-Write	This type of data can be alterable by an application program.

Abbildung 25: Modbus-Datentyp

Die `Discretes Input` werden für jeden `BOOLEAN Trigger` verwendet. Die `Coils` werden für jeden `BOOLEAN Output` verwendet. Die `Input Registers` werden für jeden `FLOAT Output` verwendet. Und die `Holding Registers` werden für jeden `FLOAT Input` verwendet.

Jeden Datentyp gehört eine Anzahl von Funktionen. Diese Funktionen werden verwendet, um Daten zu lesen oder zu ändern.

				Function Codes			
				code	*Sub code*	*(hex)*	*Section*
Data Access	**Bit access**	Physical Discrete Inputs	Read Discrete Inputs	**02**		**02**	6.2
		Internal Bits Or Physical coils	Read Coils	**01**		**01**	6.1
			Write Single Coil	**05**		**05**	6.5
			Write Multiple Coils	**15**		**0F**	6.11
	16 bits	Physical Input Registers	Read Input Register	**04**		**04**	6.4
		Internal Registers	Read Holding Registers	**03**		**03**	6.3
			Write Single Register	**06**		**06**	6.6

Abbildung 26: Modbus Funktion

Die Grundfunktionen, die durchgeführt werden sollen sind: 02, 04, 05 und 06.

4.4.4 XF

XF ist eine kleine Bibliothek für die Umsetzung von Zustandsmaschinen. Es wurde von Herrn Medard Rieder für Sommerschule *Snake* des 2. Jahr entwickelt. Diese Hauptfunktionen sind Durchführung von Events und Timer-Software. In diesem Projekt, ist eine kleine Teile hinzugefügt, um Hardware Timer zu verwalten.

4.4.5 uart

Der Prozessor muss Modbus Frame empfangen. Das UART Modul des Prozessor enthält einen Hardware Buffer von 4 Bytes. Ein Frame enthält aus mindestens 8 Byte, ist es notwendig, den Buffer so schnell wie möglich leeren. Dafür, sind die Unterbrechungen, die in jedem empfangenen Charakter auftritt verwendet. Einen Hardware-Timer erzeugt eine Zeitbasis, um des Endes eines Frame zu aufspüren.

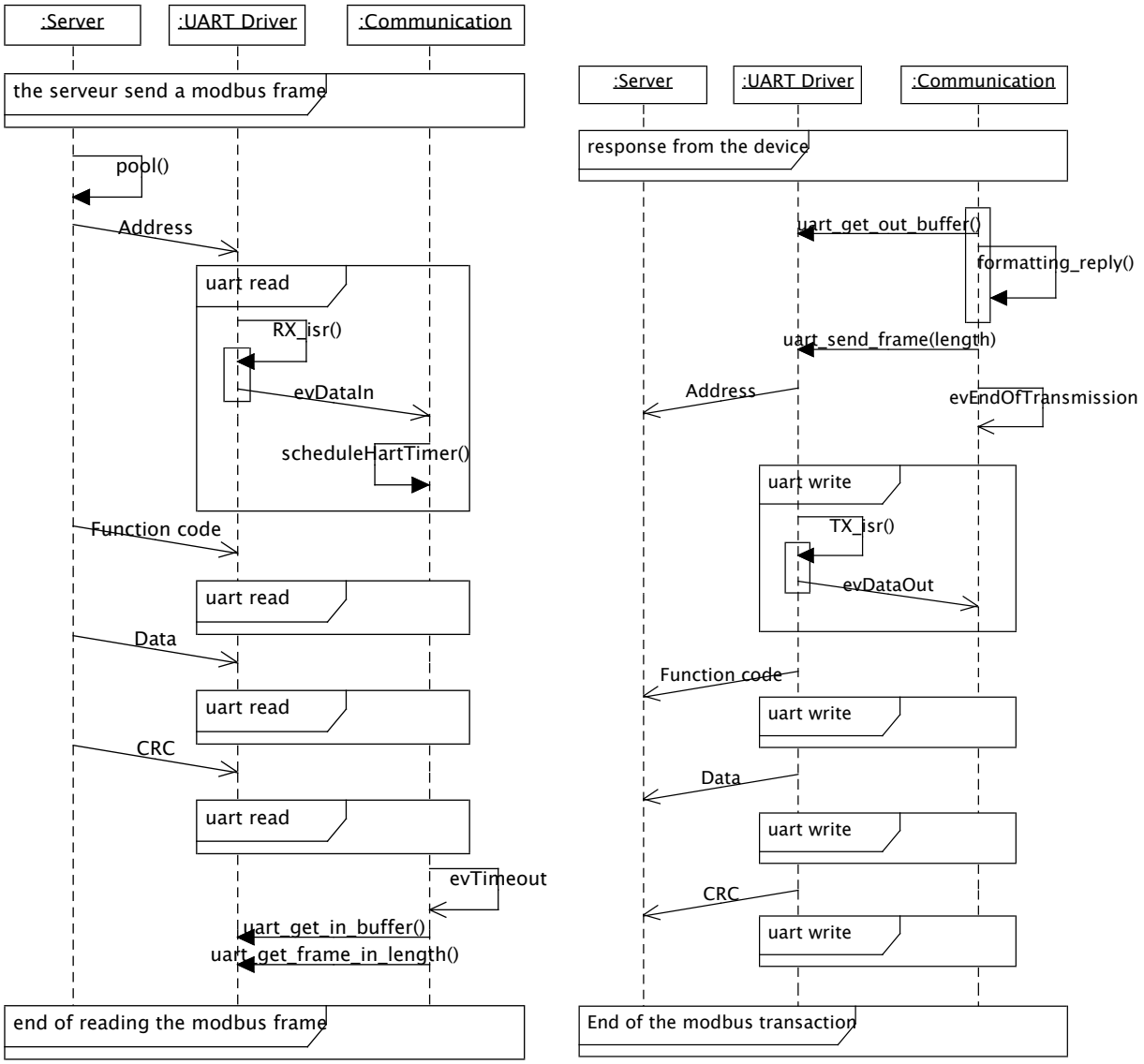


Abbildung 27: Sequence Diagramm der Kommunikation

Die Übertragung folgt dem gleichen Verfahren. Eine Unterbrechung informiert, wenn das nächste Zeichen übertragen werden können.

4.4.6 AD-Wandler

Der Wandler ist mit dem Prozessor mit 16 Datenleitungen und 5 Steuerleitungen angeschlossen. Befehle enthalten den folgenden Signalen :

Name	Beschreibung
$\overline{SHDN}$	ermöglicht die Komponentenaktivierung
$\overline{CS}$	ermöglicht die Komponenten Wählen
$\overline{CONVST}$	ermöglicht eine Umwandlung zu beginnen
$\overline{RD}$	ermöglicht der Ausgangsaktivierung
$\overline{BUSY}$	zeigt den Zustand des Umwandlungs

Abbildung 28: Steuerleitungen des AD-Wandler

Die Signale $\overline{SHDN}$ und $\overline{CS}$ werden verwendet, um den Zustand des Wandlers zu ändern. In unserer Anwendung, ist der Verbrauch nicht ein kritischer Punkt. Die geringer Verbrauch Modi wird nicht verwendet.
Die Andere Signale werden verwendet, um die Umwandlung zu steuern. Sie müssen die folgende Zeit treffen :

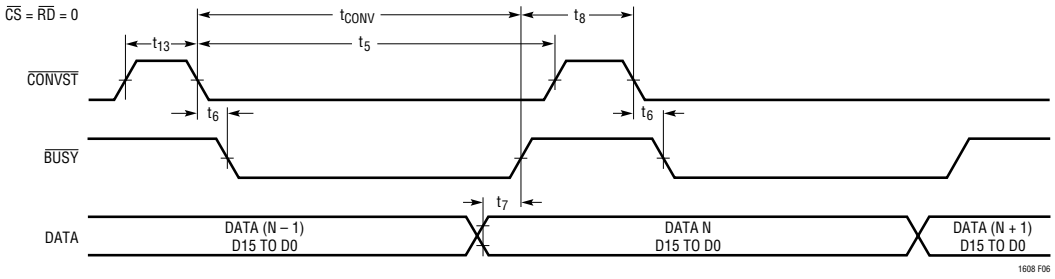


Abbildung 29: AD Timing sequence [1]

4.4.7 DA-Wandler

Der Wandler ist mit dem Prozessor mit 16 Datenleitungen und 1 Steuerleitungen angeschlossen. Der Wandler erzeugt eine neue Ausgangsspannung bei jedem Clock tick

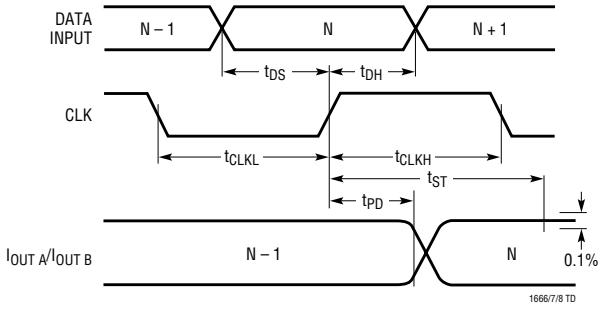


Abbildung 30: DA Timing sequence [2]

4.5 Sensor

Die CNFETs sind empfindlich Transistoren. Da es ein Prototyp, ist es immer noch teuer. Aus diesen und anderen Gründen habe ich keinen Sensor zur Verfügung. Um das Verhalten der Schaltung zu bestätigen, ist es notwendig eine fiktive Sensor zu entwickeln.

Die CNFETs sind Feldeffekttransistoren. Daher ist es möglich, das Verhalten zu simulieren. Er hat uns einem Dokument, das einige Parameter und die Ersatzschaltung des Transistors enthält gegeben.

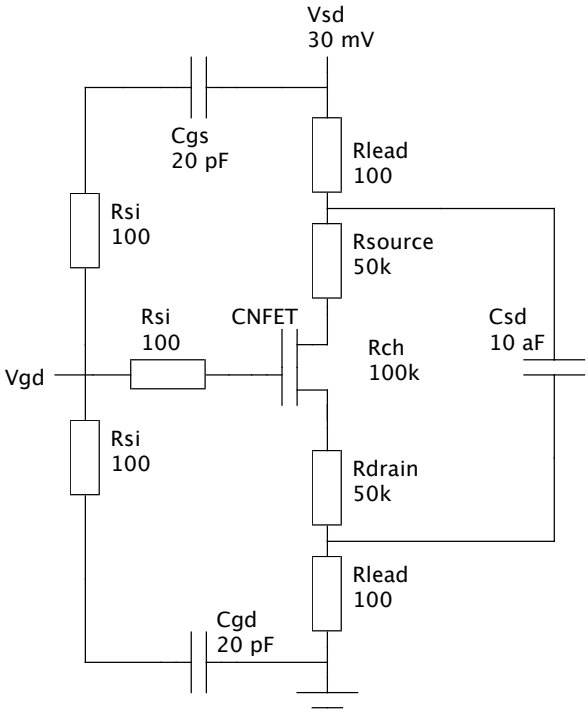


Abbildung 31: Ersatzschaltung

Die Ersatzschaltung enthält einen idealen Transistor. Bei einem guten Modell, müssen wir einen Weg, es so nah wie möglich zu einem idealen Verhalten zu finden.
Ein idealer Transistor enthält einen Widerstandsbereich und einen Stromquelle Bereich. Der Widerstandsbereich ist durch den Widerstand des Transistorkanal definiert. Die Stromquelle Bereich ist, wenn der Kanal schmal ist und kann nicht mehr Strom fahren. Das ideale Modell eines MOSFET wird eine Stromquelle durch eine Spannung gesteuert.

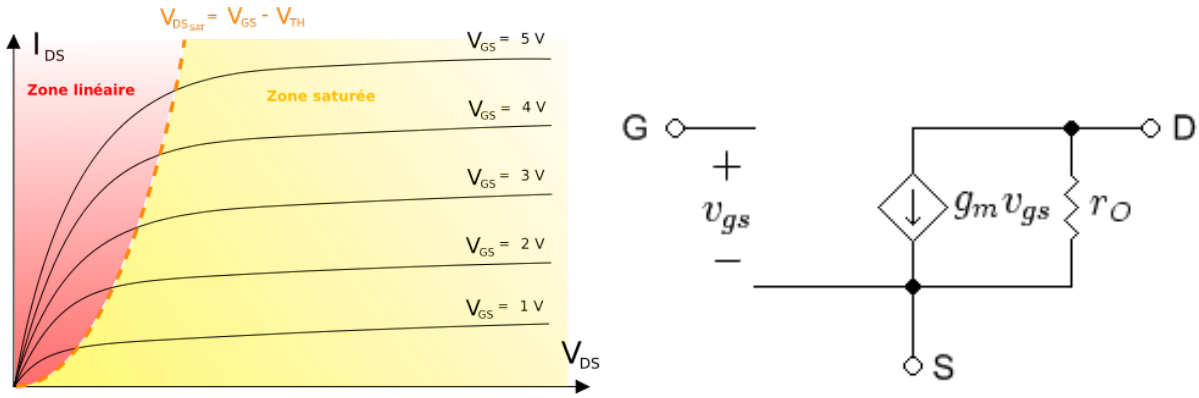


Abbildung 32: Kennlinie und Model eines typischen MOSFET

5 ENTWICKLUNG

In diesem Abschnitt, werden die Elemente im vorigen Abschnitt beschriebenen verwendet, um den Entwurf des Projektes durchzuführen. Das Ziel ist hier, um zu beschreiben die wichtigsten Elemente sorgen für einen Überblick über die Funktionsweise des Systems.

5.1 Benutzerschnittstelle

Die Benutzerschnittstelle ist in HTML realisieren. Das HTML-Dokument enthält eine Javascript Teil für die Interaktion und die Kommunikation. Die HTML-Abschnitt beschreibt das Aussehen und die Struktur der Web-Seite, entgegen die Javascript Teil, die das Verhalten beschreibt.

5.1.1 HTML

Seite für den Standardbenutzer sollen nur erlauben, Grafiken hinzufügen oder löschen. Es gibt dazu ein Knopf, die Graphen hinzufügen können. Wenn die Taste gedrückt wird, wird eine neue Gruppe mit ein Graphik und eine Löschtaste erstellt.

Die HTML-Elemente in Analyse-Abschnitt beschriebenen werden verwendet, um das folgende Ergebnis liefern:

Abbildung 33: HMI Webseite

Die `add` Taste fügt eine Gruppe mit einer Grafik am Ende der Webseite.

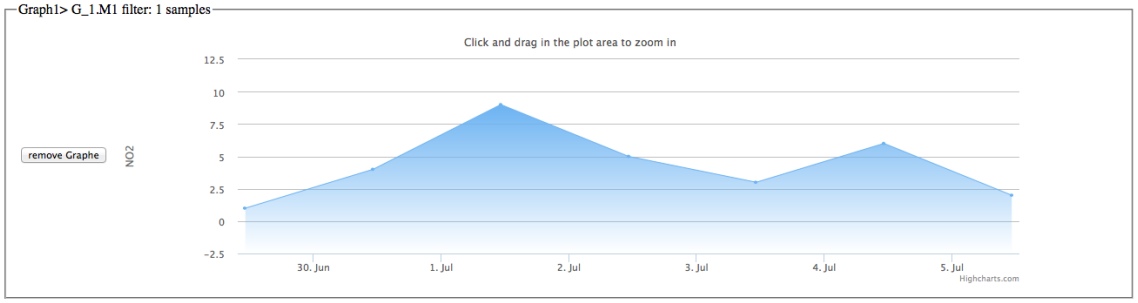


Abbildung 34: HMI Webseite

Die Seite für Programmierer können Befehle senden und die Kennlinie sehen. Sie enthält zusätzlich eine Grafik für die Kennlinie und eine Kontrollgruppe.

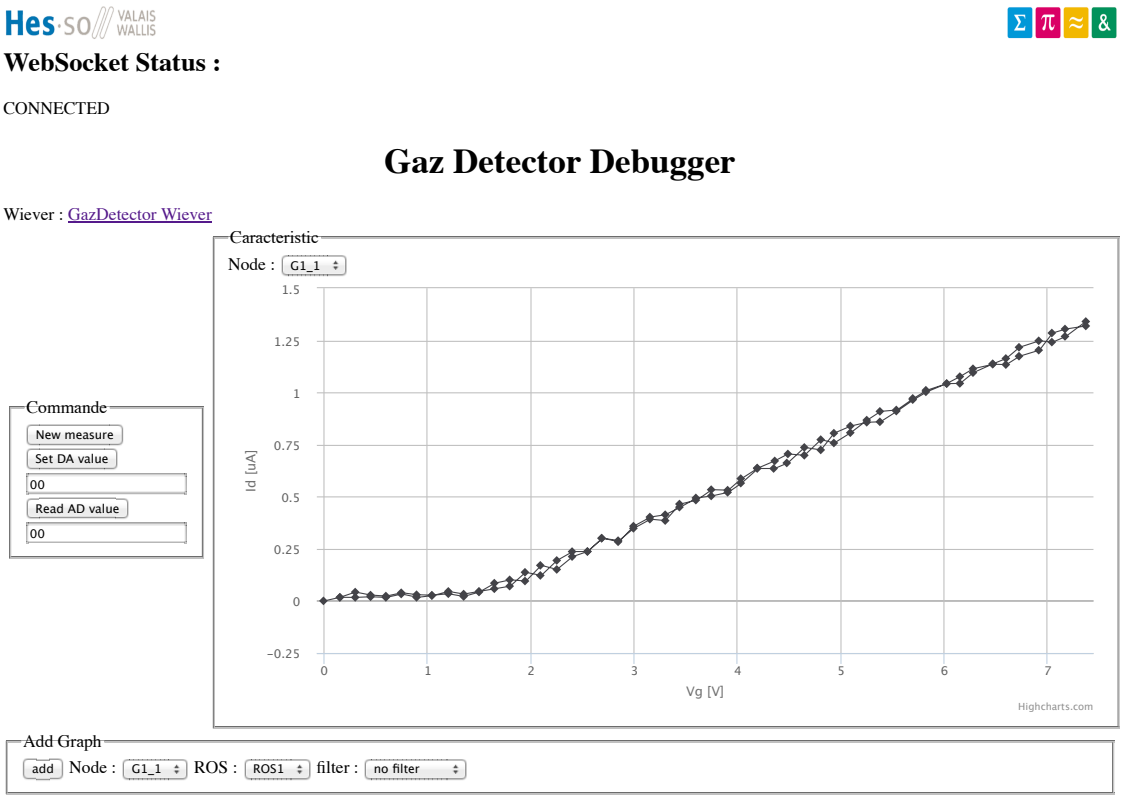


Abbildung 35: HMI Webseite für Debug

Der gesamte HTML-Code ist im Anhang (siehe Anhang D) beigefügt.

5.1.2 JavaScript

Die JavaScript wird verwendet, um ein Verhalten zu Web-Seite geben. Es implementiert zwei Funktionen : die Kommunikation mit dem Server und die Benutzerinteraktion. Dazu wird die Datei in drei Teile getrennt : WebSocket, Protokoll und Benutzerinteraktion.

WebSocket: Die Kommunikation ist mit einem websocket durchgeführt. Für dieses, der Code enthält die folgenden Funktionen:

Name	Beschreibung
webSocketBehavior	diese Funktion ruft andere Funktionen nach einer Kommunikation Veranstaltung
onOpen	diese Funktion auftritt beim Öffnen der socket
onClose	diese Funktion auftritt bei Schließung der socket
onMessage	diese Funktion auftritt an der Rezeption einer Nachricht
onError	diese Funktion auftritt an der Rezeption einem Fehler
doSend	Diese Funktion sendet eine Nachricht angegebenen in Parameter

Abbildung 36: Kommunikationsmethoden

Protokoll: Die verwendete Protokoll zwischen dem Server und dem Web-Seite ist ezHMI. Sie ist in der Dokumentation des Super-Labor scada (siehe Anhang B) definiert. Um dies zu erreichen, werden die folgenden Funktionen implementiert sind :

Name	Beschreibung
subscribeRequest	Diese Funktion sendet eine Registrierungsanfrage
SetRequest	Diese Funktion sendet eine Änderungsanfrage
GetRequest	Diese Funktion sendet eine Wertanfrage
SetTimerInterval	Diese Funktion ermöglicht es, die Geschwindigkeit des Modbus-Umfrage ändern

Abbildung 37: Kommunikationsmethoden

Diese Methoden nutzen die Funktionen der websocket Teil um die Anfragen zu senden. Die `SetTimerInterval` Methode ist nicht Teil der Basis-Protokoll. Sie wird für die schnelle Ablesung der Werte des Merkmals verwendet.

Benutzerinteraktion: Die Interaktion mit dem Benutzer wird mit den HTML-Elemente `<input>` Typ getan. Jedes Element spezifiziert eine JavaScript-Funktion, die aufgerufen wird, wenn es verwendet wird. Funktionen, die aufgerufen werden können, sind:

Name	Beschreibung
addGraphe	Diese Funktion erzeugt dynamisch ein Diagramm und es fügt zum Ende der Seite
removeGraphe	Diese Funktion löscht die Graphen in der gleichen Gruppe wie die Taste drücken enthaltenen
normalMode	Diese Funktion macht die Schaltung im normalen Modus (automatische Messung)
nur für Debug-Seite	
newMeasure	Diese Funktion sendet die Order zum Schaltung, um eine neue Messung durchzuführen
setDA	Diese Funktion ermöglicht es Ihnen, den Ausgangswert des DA-Wandler ändern
readAD	Diese Funktion liest den Eingangswert des DA-Wandler
debugMode	Diese Funktion macht die Schaltung im Debug Modus (manuell Messung)

Abbildung 38: Benutzerinteraktion Methoden

Einige dieser Funktionen verwenden das Protokoll Teil. Die Methoden um Grafiken Hinzufügen oder Löschen benutzen die Highchart Bibliothek.

Wenn eine Grafik hinzugefügt wird, ist es möglich, einen Filter auswählen. Dieser Filter ist vom Typ `low pass moving average filter`. Es wird mit der Gleichung hergestellt :

$$y[i] = \frac{1}{span} \sum_{n=i-span+1}^{i} x[n] \tag{1}$$

5.2 Server

Der Server-Teil wird vom Labor scada genommen. Das Programm ist modifiziert, um dieses Projekt passen. Der Unterschied zwischen diesem Projekt und das Labor ist die verwendete Modbus-Master, die ist ein serieller Master und nicht ein TCP Master. Der Rest des Programms bleibt das selbe.

5.2.1 Java

Der Server benutzt die `modbus4j` Bibliothek für Modbus Protokoll. Diese Bibliothek kann ein Master, der die serieller Kommunikation steuert erstellen. Die Änderung ist, die Modbus-Objekt `TCP-Master` auf `Master-RTU` ändern. Diese Änderung ist in der `connect` Methoden gemacht.

```
1  public boolean connect(String portCom, int baud, int dataBits, int stopBits, int parity){
2
3      // Construct modbus Serial parameters.
4      SerialParameters params = new SerialParameters();
5
6      params.setCommPortId(portCom);
7      params.setBaudRate(baud);
8      params.setDataBits(dataBits);
9      params.setStopBits(stopBits);
10     params.setParity(parity);
11
12     // Create the modbus object
13     modbus = new ModbusFactory().createRtuMaster(params);
14
15     // Connect to the slave.
16     modbus.init();
17
18     // If we arrive here, all is fine.
19     return true;
20 }
```

Eine neue Methode wird auch geschaffen, um die richtige serielle Port beim Starten des Server auswählen. Sie zeigen die verfügbaren Ports auf der Konsole.

```
1      public static String[] listSerialPorts() {
2
3          Enumeration ports = CommPortIdentifier.getPortIdentifiers();
4          ArrayList portList = new ArrayList();
5          String portArray[] = null;
6          while (ports.hasMoreElements()) {
7              CommPortIdentifier port = (CommPortIdentifier) ports.nextElement();
8              if (port.getPortType() == CommPortIdentifier.PORT_SERIAL) {
9                  portList.add(port.getName());
10             }
11         }
12         portArray = (String[]) portList.toArray(new String[0]);
13         return portArray;
14     }
```

5.2.2 XML

Der Server benötigt eine Datenbank, die die Beschreibung des Systems verbunden enthält. Diese Datenbank wird unter Verwendung der durch die Bild 15 beschriebenen Verfahren besiedelten. Um die Datenbank zu füllen, müssen wir die `GazDetectorClasses.xml` und `GazDetectorSystem.xml` Dateien entwerfen.

GazDetectorClasses.xml Diese Datei enthält die allgemeine Beschreibung des Systemprozessors. Das erste Element ist die Beschreibung des Knotens

```
1  <nodeType id="N1" name="GazDetector">
```

Jeder Knoten enthält die Tags `parameter`, `measValue`, `signal` und `alarm`. Der `parameter` Tag setzt die Ausgänge von Systemen.

```
1  <!-- A set point defines a target value for field level control -->
2  <parameter>
3      <point id="P1" name="mode" description="Mode Auto/Manual" datatype="BOOLEAN"/>
4      <point id="P2" name="newMeas" description="Start a new measure" datatype="BOOLEAN"/>
5      <point id="P3" name="DAValue" description="Value of the DAC" datatype="FLOAT"/>
6  </parameter>
```

Der erste Parameter wird verwendet, um die Schaltung in Debug-Modus eingestellt. Parameter `P2` verwendet werden, um neue Sequenzen zu erzeugen, wenn die Schaltung im Debug-Modus ist. Letzteres wird verwendet, um den Wert von DA Wandler ändern.

Das `measValue` Tag beschreibt die Systemeingänge.

```
<!-- A process variable is an analogue measurement result updated periodically -->
<measValue>
    <point id="M1" name="ROS1" description="Voltage measurement" datatype="FLOAT" unit="V"/>
    <point id="M2" name="ROS2" description="Current measurement" datatype="FLOAT" unit="A"/>
    <point id="M3" name="ROS3" description="Surface measurement" datatype="FLOAT" unit="VA"/>
    <point id="M4" name="Char I" description="Current measurement" datatype="FLOAT" unit="A"/>
    <point id="M5" name="Char V" description="Voltage stimulus" datatype="FLOAT" unit="V"/>
    <point id="M6" name="AD Value" description="Current value of the ADC" datatype="FLOAT" unit="V"/>
</measValue>
```

Die MeasValue `M1` bis `M3` stellt unterschiedliche Messsystem (ROS). `M4` und `M5` enthält einen charakteristischen Punkt. Letzteres stellt den aktuellen Wert des AD Wandlers.

Das `signal` Tag wird verwendet, um Ereignisse zu beschreiben.

```
<!-- A signal represents the occurence of an event -->
<signal>
    <point id="S1" name="High NO2 Concentration"  description="GazDetector ({ref}) detect a high
        concentration of NO2." datatype="BOOLEAN" triggerCondition="equal" triggerValue="true"/>
    <point id="S2" name="End of the transmission of the characteristic"  description="Characteristic send
        complete" datatype="BOOLEAN" triggerCondition="equal" triggerValue="true"/>
    <point id="S3" name="End of the measurement of the characteristic"  description="Characteristic measure
        complete" datatype="BOOLEAN" triggerCondition="equal" triggerValue="true"/>
</signal>
```

Ein Signal kann erzeugt werden, wenn ein Gas Grenzwert erreicht ist. Die `S2` und `S3` Signal wirden verwendet, um das Ende eines Kennlinie zu informieren.
Es ist auch möglich Alarmen beschreiben. Aber dieser Parameter ist nicht für dieses Projekt verwendet.

GazDetectorSystem.xml Diese Datei enthält die spezifische Beschreibung des Systemprozessors. Die verwendete Struktur ist die in Fig. 13 beschriebene. Das erste Element ist die Beschreibung des Gruppen.

```
<group id="G1" name="GazDetector Serial">
    <node id="G1_1"  name="GazDetector-1" nodeTypeRef="N1"/>
</group>
```

Jede Groupe enthält die Tags `output`, `input`, `signal` und `trigger`. Das `input` Tag beschreibt die Systemeingänge, die als `measValue` beworben werden.

```
<inputs>
    <!-- Measure Value -->
    <input type="FLOAT" deviceType="modbus" deviceAddress="1" address="100" addressType="4" nodeRef="G1_1"
        pointRef="M1" conversion="1.0*value"/>
    <!-- Measure Value -->
    <input type="FLOAT" deviceType="modbus" deviceAddress="1" address="101" addressType="4" nodeRef="G1_1"
        pointRef="M2" conversion="1.0*value"/>
    <!-- Measure Value -->
    <input type="FLOAT" deviceType="modbus" deviceAddress="1" address="102" addressType="4" nodeRef="G1_1"
        pointRef="M3" conversion="1.0*value"/>
    <!-- Char Value -->
    <input type="FLOAT" deviceType="modbus" deviceAddress="1" address="103" addressType="4" nodeRef="G1_1"
        pointRef="M4" conversion="1.0*value"/>
    <!-- AD Value -->
    <input type="FLOAT" deviceType="modbus" deviceAddress="1" address="104" addressType="4" nodeRef="G1_1"
        pointRef="M5" conversion="1.0*value"/>
</inputs>
```

Alle Einträge sind vom Typ `float`. Die Adresse beginnen um 100 um die Gruppe 1xx zu zeigen. `addressType` definiert das Modbus-Funktion, die `4` ist zum Lesen eines Registers.

Der `output` Tag beschreibt die Ausgänge von Systemen, die als `parameter` beworben werden.

```
<outputs>
    <!-- Auto / Manual -->
    <output type="BOOLEAN" deviceType="modbus" deviceAddress="1" address="110" addressType="5" nodeRef="G1_1"
          pointRef="P1"/>
    <!-- new Measure -->
    <output type="BOOLEAN" deviceType="modbus" deviceAddress="1" address="111" addressType="5" nodeRef="G1_1"
          pointRef="P2"/>
    <!-- DA Value -->
    <output type="FLOAT" deviceType="modubs" deviceAddress="1" address="120" addressType="6" nodeRef="G1_1"
         pointRef="P3"/>
</outputs>
```

Die `boolean` Ausgänge beginnen zu Adresse 110. Das Modbus-Funktion ist `5`, um ein Bit zu schreiben. Die `float` Ausgänge beginnen zu Adresse 120 und sie haben das Modbus-Funktion `6`. Der Wert der DA-Wandler ist Typ `float` und erreichbar mit der Funktion `6`.

Wie bei der vorherigen Tag, werden `alarme` und `signal` unter dem `trigger` Tag aufgeführt.

```
<triggers>
    <trigger type="BOOLEAN" deviceType="modbus" deviceAddress="1" address="130" addressType="2" nodeRef="G1_1
          " pointRef="S1"/>
    <trigger type="BOOLEAN" deviceType="modbus" deviceAddress="1" address="131" addressType="2" nodeRef="G1_1
          " pointRef="S2"/>
    <trigger type="BOOLEAN" deviceType="modbus" deviceAddress="1" address="132" addressType="2" nodeRef="G1_1
          " pointRef="S3"/>
</triggers>
```

Das Typ ist immer `boolean` und das Modbus-Funktion `2`. Der `S1` Signal wird verwendet, um eine Rate von NO2 anzuzeigen. Der `S2` Signal zum Melden des Endes einer Messung verwendet.

5.3 Eingebettet System

Das Erfassungssystem weist zwei Wandler und einen Prozessor. Um Messungen zu machen, enthält der Prozessor einen Code in C. Der Code ist mit UML-Zustandsmaschinen beschrieben. Die XF Bibliothek ermöglicht, diese Zustandsmaschinen implementieren.

5.3.1 XF

Für den XF Bibliotheksarbeit, müsst man einen Timer zuweisen, um die Timer Software zu erzeugen. Timer 1 wird für diese Aufgabe ausgewählt.

```
// initialisation of the TIMER1
T1CON = 0x8010;
PR1 = 0x2710;

// enable the TIMER1 interrupt
IPC0bits.T1IP = 4;
IFS0bits.T1IF = 0;
IEC0bits.T1IE = 1;
```

Die Software Timer Management ist in der Unterbrechung der timer1 getan.

```
// timer ISR
void _ISR _T1Interrupt(void)
{
    XF_decrementAndQueueTimers();
    IFS0bits.T1IF = 0;
}
```

Der Rest des Programms verwendet auch die Hardware-Timer. Um ihre Verwendung zu vereinfachen, ist ihr Management zu XF gewidmet. Drei neue Methoden sind in der Schnittstelle des XF hinzugefügt

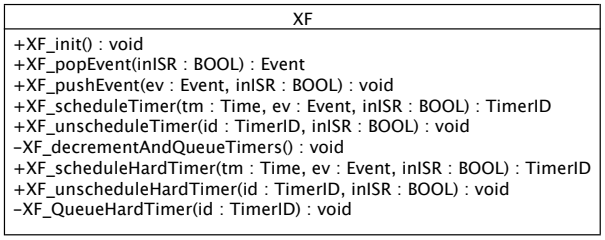


Abbildung 39: XF Klasse Diagramm

Die neuen Funktion sind: `XF_scheduleHardTimer`, `XF_unscheduleHardTimer` und `XF_QueueHardTimer`. Ihre Schnittstelle ist die gleiche wie die Software-Timer, um die Verwendung zu vereinfachen.

XF_scheduleHardTimer(tm: Timer, ev: Event, inISR: BOOL) Diese Funktion erlaubt es, Hard-Timer zuweisen. Sie berechnen den Zählwert für die Erzeugung der Zeitbasis durch `tm` angeben.

```
1  // calcul the value of PR et PS
2  if( tm >= 15000)
3  {
4      PS = 1;
5      PR = (tm/TCY) * 125;
6  }
7  else
8  {
9      PS = 0;
10     if(tm <= TCY)
11         PR = (tm*100)/TCY * 10;
12     else
13         PR = (tm/TCY) * 1000;
14 }
```

Eine Tabelle lässt Sie wissen, wenn ein Timer frei ist, die Ereignis, der ID und ihr Wert.

```
1  if(theXF.hardTimerList[i].free == true)
2  {
3      theXF.hardTimerList[i].ev = ev;
4      theXF.hardTimerList[i].free = false;
5      theXF.hardTimerList[i].id = i+1;
6      hardTimerID = i+1;
7      ...
```

Und gewährt das Wert auf den frei Timer:

```
1  switch(hardTimerID)
2  {
3      case 1: // allocate the TMR2
4          PR2 = PR;
5          TMR2 = 0x0000;
6          T2CONbits.TCKPS = PS;
7          IFS0bits.T2IF = 0;
8          IEC0bits.T2IE = 1;
9          T2CONbits.TON = 1;
10         break;
11
12     case 2: // allocate the TRM3
13         ...
14 }
```

Der XF wurde für einen kleinen Prozessor entwickelt. Die `entercritical` Funktion ist nicht gedacht, um Priorität Unterbrechungen zu haben. Aus diesem Grund müssen alle Unterbrechungen die gleiche Priorität haben. Ist dies nicht der Fall, kann ein Interrupt in einem kritischen Bereich kommen, und der XF kann Ereignisse verloren.

5.3.2 Controller

die `Controller` Klasse verwendet, um die Arbeitsweise der Schaltung wählen. Es enthält zwei verschiedene Modus : Normal oder Debug. Der Normal Modus nimmt kontinuierliche Messungen. Der Debug Modus wird verwendet, um nur eine Messungen zu nehmen. Sein Verhalten wird über die folgende Zustandsmaschinen beschrieben.

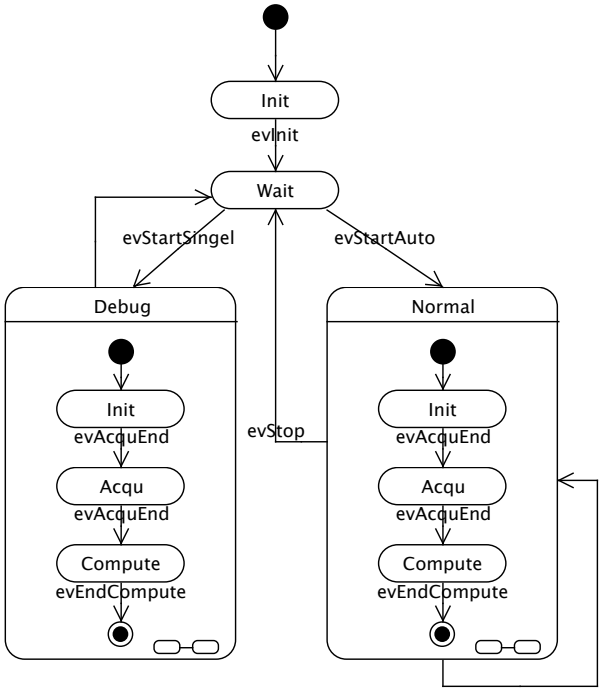


Abbildung 40: Zustandsmaschine des Controller

Nach der Initialisierung, geht der Controller durch den `Wait` Zustand, um in den `Normal` Zustand zu gehen. Ein `evStop` beendet den `Normal` Zustand und der Controller geht in den `Wait` Zustand. Zu diesem Zeitpunkt ist der Controller in den `Debug` Modus.
Jeder der beiden Modus enthält die gleichen internen Zustände : `Init`, `Acqu` und `Compute`. Die ersten beiden benutzen den `Erfassung` Klasse, um die Gate Signal zu erzeugen. Der letzte Zustand ruft die Rechenmethode, die es Ausgangswerte berechnet.

Berechnung von ROS1 ROS1 stellt den Schwellenwert der Kennlinie. Dieser Wert wird durch die Analyse der Ableitung der Funktion und durch Extrapolation.
Eine einfachere Lösung ist in diesem Fall durchgeführt. Wenn der Wert der Proben beginnen zu stark erhöht, wird die Probe vor der Zunahme ausgewählten.

```
1  // compute the ROS1 value
2  for(i=0;i<NBR_SAMPLE_UP;i++)
3  {
4      temp = sample[i];
5      if(sample[i+1]>= temp+ROS1_DELTA &&
6          sample[i+2] >= temp + ROS1_DELTA &&
7           sample[i+3] >= temp + ROS1_DELTA)
8          break;
9  }
10 me->ros1Value = temp;
```

Berechnung von ROS2 ROS2 stellt den höchsten gemessenen Stromwert. Daher ist die Probe mit dem größten Spannung.

```
1 // compute the ROS2 value
2 temp = sample[NBR_SAMPLE_UP];
3 me->ros2Value = temp;
```

Berechnung von ROS3 ROS3 stellt das Integral der charakteristischen. Hierzu wird eine diskrete Integration durchgeführt. Um die Berechnung zu vereinfachen, führt es nur eine Addition aller Proben. Um ein genaues Ergebnis zu haben, sollte die Summe durch eine Integrationskonstante multipliziert.

```
1 // compute the ROS3 value
2 temp = 0;
3 for(i=0;i<NBR_SAMPLE_UP;i++)
4 {
5     temp += sample[i];
6 }
7 coeff = (me->acqu->delta*16.4/65536)*2/65.536;
8 temp = temp*coeff;
9 me->ros3Value = temp;
```

Die Controller kann mit der folgenden Schnittstelle gesteuert werden.

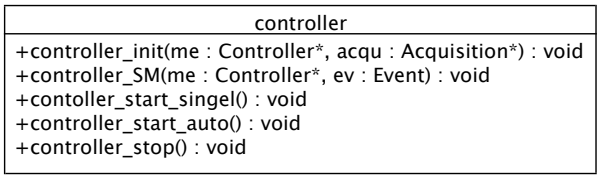


Abbildung 41: Controller Klasse Diagramm

Die Klasse enthält Funktionen zur Zustandsmaschinen, sowie Funktionen, um die Zustände der Maschine ändern.

controller_start_singel() Diese Methode ermöglicht es Ihnen, eine Messung zu starten, sobald die Steuerung gestoppt.

```
1 XF_pushEvent(evContSingel,false);
```

controller_start_auto() Dieses Methode wird zum Normalbetrieb zurückzukehren.

```
1 XF_pushEvent(evContAuto,false);
```

controller_stop() Diese Methode legt den Controller im Debug-Modus.

```
1 XF_pushEvent(evContStop,false);
```

5.3.3 Erfassung

Der `Acquisition` Klasse wird verwendet, um Messsequenzen und Akquisition zu starten. Die Sequenz durchgeführt werden soll, ist in Fig. 4 gezeigt. Um eine gute Zeitbasen zu erzeugen, wird ein Hardware Timer verwendet. Ihre Interrupt wird verwendet, um das Ende jeder Phasen der Konstruktion des Signals signalisieren.

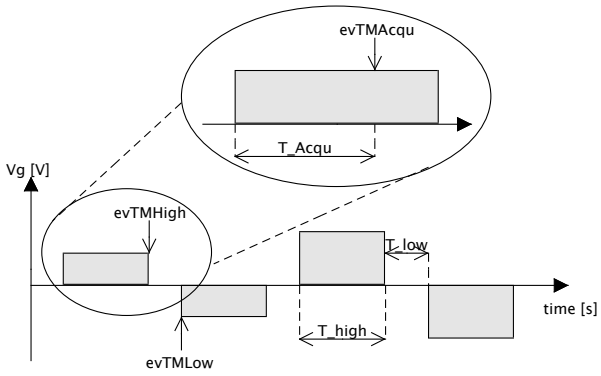


Abbildung 42: Zeitdiagramm der Erfassungs

Die Zustandsmaschine entwickelt, um das Signal zu konstruieren und die Durchführung der Messung ist wie folgt:

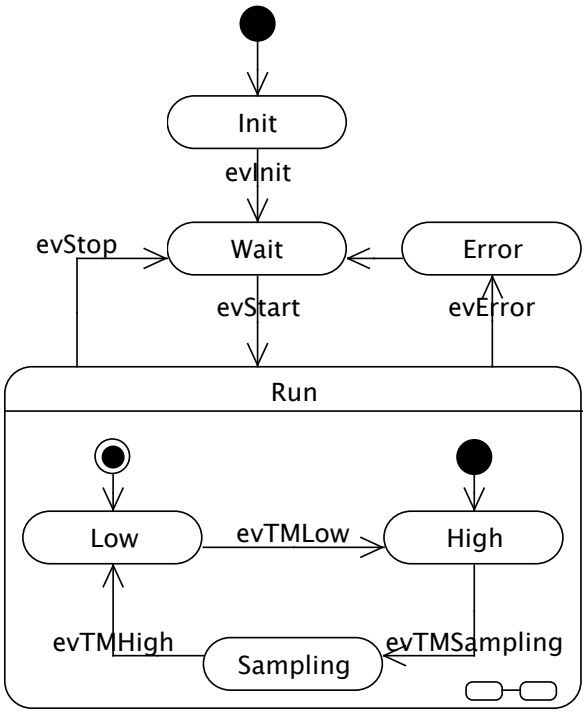


Abbildung 43: Zustandsmaschine des Erfassung

Sie enthält zwei Hauptzustände : `Wait` und `Run`. Die `Run` Zustand wird von drei Zustände zusammensetzt: `Low`, `High` und `Sampling`. Der Übergang zwischen diesen Zuständen erfolgt mit dem Timeout des Hardware-Zeitgeber. Der Fehlerzustand ist erreicht, wenn ein Timeout nicht eingehalten.

High Der `High` Zustand ist zum Erzeugen der Spannungsrampe verantwortlich. Die Rampe mit zwei Flaggen verwaltet : `up_down` und `pos_neg`. Die erste erfaßt, ob es das obere Ende der Rampe erreicht hat.

```
1 // check the top of the rampe
2 if(me->stimulusStep >= NBR_VALUE-1)
3 {
4     me->up_down = !me->up_down;
5     me->stimulusStep=0;
6 }
```

Die zweite wird verwendet, um eine positive Spannung zu erzeugen und um das nächste Mal die gleiche Spannung, aber negativ zu erzeugen.

```
1  // generate the Vg value
2  if(me->up_down == true)
3      dac_write(me->stimulus[me->stimulusStep]);
4  else
5      dac_write(me->stimulus[NBR_VALUE-1 - me->stimulusStep]);
6
7  me->stimulusStep++;
8
9  // toggle the positive negative flag
10 me->pos_neg = !me->pos_neg;
```

Sampling Der `Sampling` Zustand ermöglicht der Probenahme des Kennlinie. Werte werden nur für positive Werte von Spannung entnommen und, wenn es eine Erfassung und nicht eine Initialisierung ist.

```
1  // sampling only for the positive value
2  if(me->pos_neg == true)
3  {
4      if(me->acqu_init == true)
5      {
6          value = 0;
7          for(i=0; i<me->nbrOversample;i++)
8          {
9              value += adc_read();
10         }
11         value = value/me->nbrOversample;
12         me->sample[me->sampleStep] = value & 0x0000ffff;
13         analogOut_write((me->sample[me->sampleStep])>>6);
14         me->sampleStep++;
15     }
16 }
```

Low Dieser Zustand erzeugt der Null zwischen zwei Spannungsimpulse und prüft, ob es am Ende einer Sequenz ist.

```
1  // generate the 0 value for Vg
2  dac_write(STIMULUS_ZERO);
3
4  // check the end condition
5  if(me->up_down == false)
6  {
7      if(me->stimulusStep == NBR_VALUE-2)
8          XF_pushEvent(evAcquEnd,false);
9  }
```

Die Klasse besitzt die folgende Schnittstelle, um Sequenzen zu konfigurieren und auszuführen.

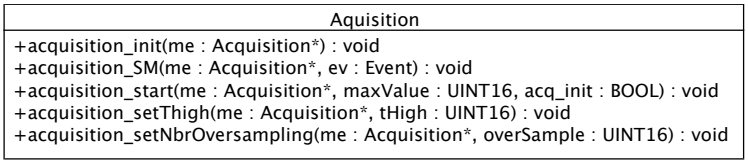


Abbildung 44: Klasse Diagramm des Erfassung

Die Klasse enthält Funktionen für die Zustandsmaschine und Konfigurationsfunktionen.

acquisition_start(me: Acquisition*, maxValue: UINT16, acq_init: BOOL) Diese Funktion wird verwendet, um eine neue Sequenz zu starten. Die Rampenwerte werden auf Basis der `maxValue` Parameter berechnen. Die `ac_init` Flagge ermöglicht die Erfassung.

```
delta = (maxValue-STIMULUS_ZERO)/(NBR_VALUE>>1);

y=0;
for(i=0;i<NBR_VALUE; i+=2)
{
    me->stimulus[i] = STIMULUS_ZERO + y*delta;
    me->stimulus[i+1] = STIMULUS_ZERO - y*delta;
    y++;
}
```

acquisition_setThigh(me: Acquisition*, tHigh: UINT16) Diese Funktion stellt die Zeitimpulse.

```
me->thigh = tHigh;
me->tacqu = tHigh*0.9;
```

acquisition_setNbrOversampling(me: Acquisition*, overSample: UINT16) Diese Funktion stellt die Anzahl der Proben für jeden Impuls genommen.

```
if(overSample > NBR_OVERSAMPLE_MAX)
{
    me->nbrOversample = NBR_OVERSAMPLE_MAX;
}
else
{
    me->nbrOversample = overSample;
}
```

5.3.4 Kommunikation

Die `Communication` Klasse implementiert das Modbus Protokoll. Dazu, nutzt es die `uart_driver` Klasse und eine Hardware-Timer. Ihre Verwendung ist in der folgenden Abbildung dargestellt :

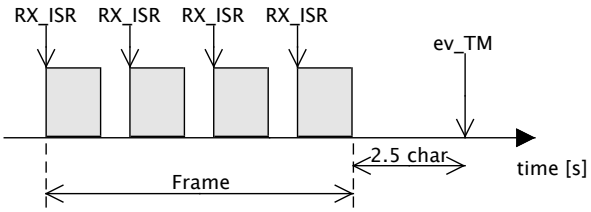


Abbildung 45: Zeitdiagramm der Kommunikation

Eine UART Unterbrechung, die der Hardware Timer startet, auftritt um jeden Empfang eines Charakters. Wenn der Timer-Interrupt auftritt, bedeutet dies das Ende des Rahmens.
Der Zustandsmaschine der `communication` Klasse ist, dass durch das Modbus Dokumentation, mit zwei zusätzlich Zustände vorgeschlagen.

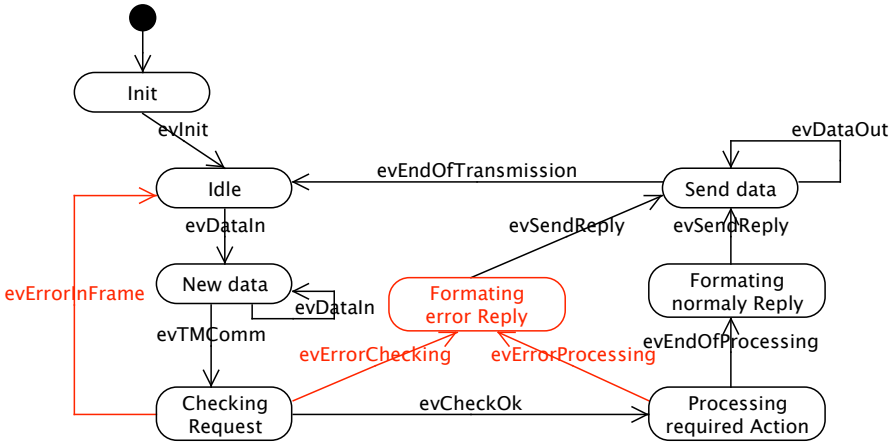


Abbildung 46: Zustandsmaschine des Kommunikation

Eine UART Unterbrechung ändert den Zustand der Maschine von `Idle` nach `New Data`. Die Hardware-Timer wird gestartet und auf null zurückgesetzt für alle Charaktere empfangen. Wenn der Timer-Interrupt auftritt, macht die Maschine die Überprüfung der Frame. Wenn es keine Fehler gibt, wird die Abfrage ausgeführt. Im Fehlerfall wird eine Fehlerantwort gesendet. Nach dem Ausführen der Abfrage wird die Antwort gebildet und gesendet.

Daten werden in Form von Tabellenzeiger verwaltet. Die `communication` Klasse enthält nur die Referenz von der Daten in den anderen Klassen enthalten. Mit diesem Struktur, kann es leicht zu lesen alle Daten.

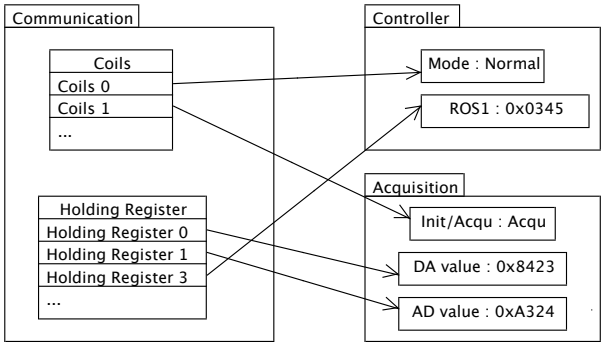


Abbildung 47: Tabellenzeiger des Kommunikation Klasse

New data Dieser Zustand startet den Timer für jeden neuen Charakter.

```
1 XF_unscheduleHardTimer(me->timerID,false);
2 me->timerID = XF_scheduleHardTimer(T_END_OF_FRAME,evCommTM,false);
```

Checking Request Dieser Zustand ermöglicht es Ihnen, die Gültigkeit des empfangenen Rahmens überprüfen. Ein Rahmen ist gültig, wenn die Adresse gültig ist :

```
1 addr = me->frame_ptr[0];
2 // check the address
3 if(addr == MY_ADDRESS)
4 { ...
```

wenn der CRC korrekt ist :

```
// check the CRC
length = uart_get_frame_in_length();
CRCInt = CRC16(me->frame_ptr,length-2);
CRC = ((me->frame_ptr[length-1])<<8);
CRC = CRC + me->frame_ptr[length-2];

if(CRC == CRCInt)
{ ...
```

und schließlich, wenn der Funktionscode unterstützt

```
// check data address and fonction code
valide = false;
fctCode = me->frame_ptr[1];
arg1 = (me->frame_ptr[2]<<8) + me->frame_ptr[3];
arg2 = (me->frame_ptr[4]<<8) + me->frame_ptr[5];

switch(fctCode)
{
    case 0x02: // Read Discretes Input
    ...
```

Wenn einer dieser Parameter nicht korrekt ist, wird eine Ausnahme erzeugt.

Processing required Action Dieser Zustand wird Daten zu suchen oder zu schreiben. Zum Beispiel das Schreiben ein `Coils` :

```
case 0x05: // Write Singel Coil
    me->response_ptr += 3;
    if(arg2 == 0xff00)
        *(me->coils[arg1]) = true;
    else
        *(me->coils[arg1]) = false;

    Communication_single_coil(me);
    break;
```

Einen Funktion wird verwendet, um die Daten vor oder nach der Operation zu aktualisieren. Es gibt eine Funktion für jeden Funktionscode. Diese Funktionen werden nachstehend beschrieben.

Formating normaly reply Dieser Zustand baute die Antwortframe. Eine Antwort besteht aus der Adresse :

```
// Slave Address
me->response_ptr = uart_get_out_buffer();
me->response_ptr[0] = MY_ADDRESS;
```

PDU, die spezifisch für die Funktion ist :

```
// PDU
switch(fctCode)
{
    case 0x02: // Read Discrete Input
        me->response_ptr[1] = me->frame_ptr[1];
        me->response_ptr[2] = 1;
        length = 4;
        me->response_ptr += length;
        break;
        ...
```

Und eine CRC :

```
// CRC
CRC = CRC16(uart_get_out_buffer(),length);
*(me->response_ptr) = CRC & 0x00ff;
me->response_ptr++;
*(me->response_ptr) = (CRC&0xff00)>>8;
```

Send data Sobald eine Antwort erstellt wird, wird es durch den Zustand gesendet.

```
1 uart_send_frame(me->response_ptr - uart_get_out_buffer());
```

Wenn das erste Charakter gesendet wird, wird das Getriebe automatisch von der `uart` Klasse getan.

Die Klasse hat die folgenden Methoden:

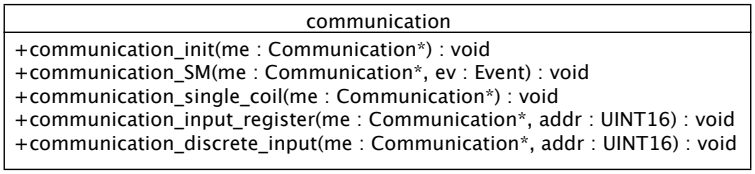


Abbildung 48: Communication Klass Diagramme

communication_single_coil(me: Communication*) Diese Methode ermöglicht es Ihnen, Befehle an den Controller-Klasse basierend auf der Änderung der Flaggen senden. Es kann ein Betriebsartenwechsel werden:

```
1  if(me->mode == DEBUG)
2  {
3      Controller_stop();
4      me->cont->debugMode = true;
5  }
6  else
7  {
8      Controller_start_auto();
9      me->cont->normalMode = true;
10 }
```

Oder eine Anfrage für eine neue Messung :

```
1 if(me->new_meas == true)
2 {
3     Controller_start_single();
4     me->new_meas = false;
5     me->sample_index = 0;
6 }
```

communication_input_register(me: Communication*, addr: UINT16) Diese Funktion setzt den Registrierungswert Tagen vor dem Lesen. Es kann ein Lesen der Wandler sein :

```
1 if(addr == AD_ADDR)
2 {
3     me->cad_value = adc_read();
4 }
```

Oder Lesen der Kennlinie :

```
1  if(me->sample_index <= 49)
2      me->stimulus_value = me->stimulus[me->sample_index*2];
3  else
4      me->stimulus_value = me->stimulus[ NBR_SAMPLE - (me->sample_index-49)*2];
5
6
7  if(me->sample_index < NBR_SAMPLE-1)
8      me->sample_index++;
9  else
10 {
11     me->sample_index = 0;
12     me->end_char = true;
13 }
```

communication_discrete_input(me: Communication*, addr: UINT16) Diese Funktion wird für Alarme verwendet. Es deaktiviert das Flag einmal gelesen wurde.

```
1 // clear the flag after reading
2 *(me->discreteInput[addr]) = false;
```

5.3.5 Prozessor Konfiguration

Die Prozessor Konfiguration wird durch eine Reihe von Register nur zugänglich für die Programmierung durchgeführt. Die Register in Frage sind die Register `CW1` bis `CW4`, die die Konfiguration bits enthält.
Nur wichtige Parameter sind hier beschrieben.

CW1 Der Watchdog-Timer ist deaktiviert. Die Programmierung ist auf der Pins `PGEC2` und `PGED2` getan. JTAG-Port muss, um bestimmte Ein-und Ausgänge Kiefern zu verwenden deaktiviert werden.

```
1 // non watchdog timer - programmation on pin PGEx2 - JTAG off
2 _CONFIG1( FWDTEN_WDT_DIS & ICS_PGx2 & JTAGEN_OFF);
```

CW2 Der interne Oszillator des Prozessors verwendet wird. Der primäre Oszillator ist deaktiviert. Die Pins `OSCO` sind als Ein-und Ausgänge verwendet. Die verwendete Oszillator ist die FRC Oszillator mit PLL.

```
1 // non primari oscillator - OSCIO pin IO fonction - interne FRC osc with PLL
2 _CONFIG2( POSCMD_NONE & OSCIOFCN_ON & FNOSC_FRCPLL);
```

CW3 Die Pins `SOSC` sind als Ein-und Ausgänge verwendet.

```
1 // SOSC in digital mode
2 _CONFIG3( SOSCSEL_OFF);
```

CW4 PLL verwendet unmittelbar das Ausgangssignal des internen Oszillators.

```
1 // pll config
2 _CONFIG4(PLLDIV_NODIV);
```

Oszillator Die Betriebsfrequenz des Prozessors wird nach dem verwendeten Oszillator eingestellt. Mit ohne externe Oszillator an den Mikrocontroller angeschlossen, muß die interne Oszillator verwenden. FRC kann direkt oder mit einer PLL verwendet werden. Der PLL wird verwendet, um mehr Flexibilität bei der Arbeitsfrequenz zu haben. Auswahl der internen Oszillator mit PLL ist dabei unter Verwendung der oben beschriebenen Konfigurationsregister getan.

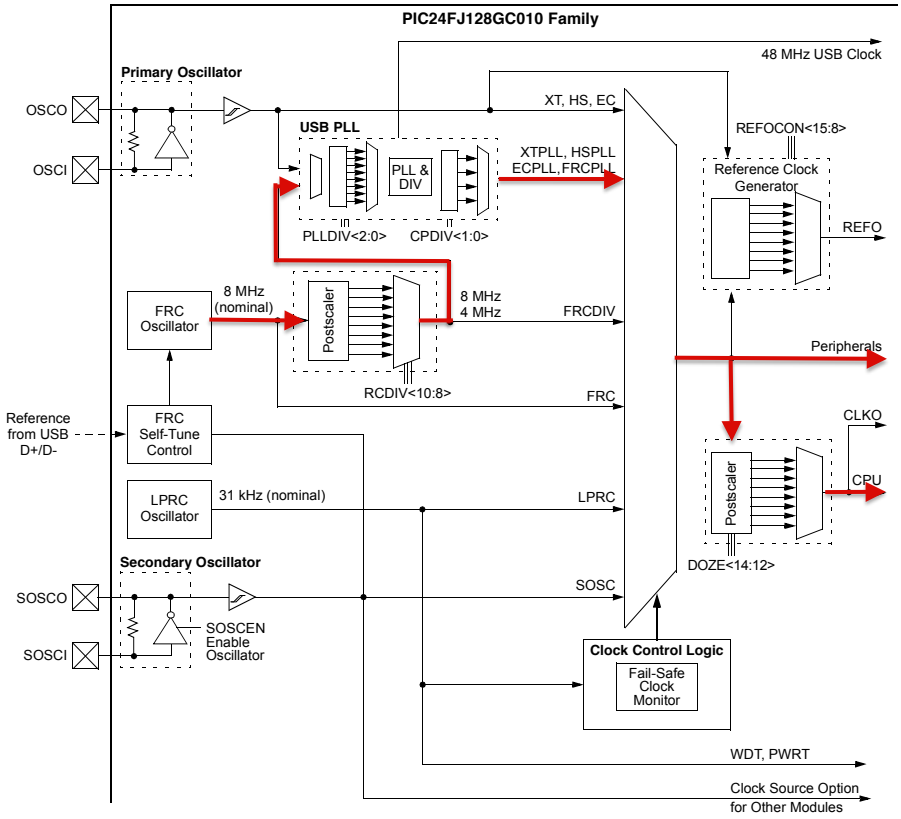


Abbildung 49: Blockschaltbild des Oszillators

Der Frequenzbereich unter Verwendung der PLL ist von 4 bis 32 [MHz]. Die Auswahl der Prozessorgeschwindigkeit ist mit der `CPDIV` Bits von der `CLKDIV` Register getan.

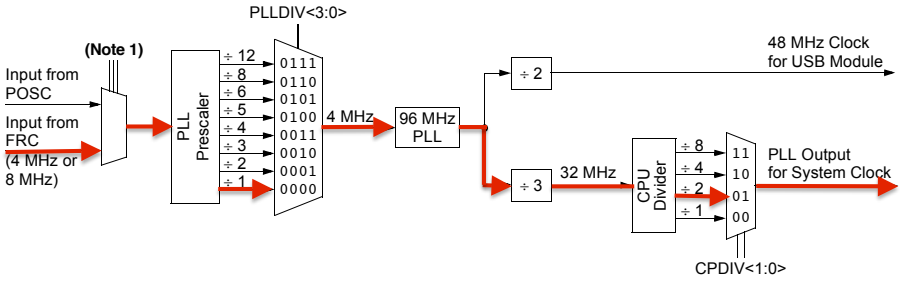


Abbildung 50: Blockschaltbild des Pll

Der Code verwendet viele Timer und Interrupts. Um die Unterbrechung Reaktionszeit zu verringern und die Genauigkeit der Timer zu erhöhen, wird ein Hochfrequenz verwendet. Die gewählte Frequenz ist 8 [MHz]. Dafür brauchen wir einen Wert von 16 MHz Ausgang des Oszillators Block, weil der Prozessor-Zyklus ist $F_{cycl} = F_{osc}/2$. Der Wert des Divisors `CPDIV` ist 2 (01).

```
1 // configuration of the intern oscillator
2 CLKDIVbits.CPDIV = 1;
```

5.3.6 UART Treiber

UART Prozessormodul wird verwendet, um Asynchrone Informationen zu übertragen. Es wird durch die `Communication` Klasse verwendet. Seine Schnittstelle wird durch das folgende Diagramm definiert.

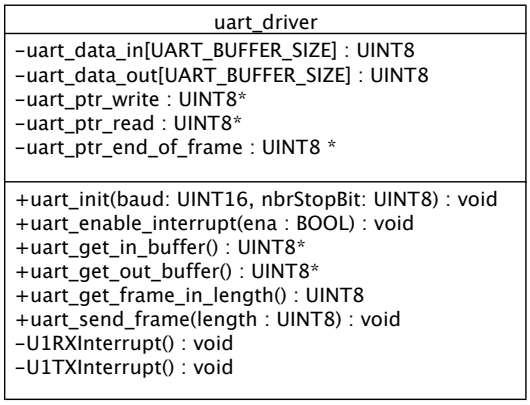


Abbildung 51: UART Klasse Diagramm

uart_init(baud: UINT16, nbrStopBit: UINT8) Diese Funktion initialisiert die UART-Modul. Es wird mit drei Registern konfiguriert : `UxMODE`, `UxSTA` und `UxBRG`.

Das `UxMODE` Register dient zur Konfiguration der Betriebsart des Moduls. Das Modbus-Protokoll definiert zwei Stop-Bits und 8 Datenbits. Die Zahl der Stop-Bits wird als Parameter in der Initialisierung Funktion vergangen.

```
1 // no parity
2 U1MODEbits.PDSEL = 0;
3 // stop bits
4 U1MODEbits.STSEL = nbrStopBit-1;
5 // enable the module
6 U1MODEbits.UARTEN = 1;
```

Das `UxSTA` Register enthält den aktuellen Zustand der UART-Modul. Es definiert auch wenn Unterbrechungen auftreten.

```
1 // enable the transmission
2 U1STAbits.UTXEN = 1;
3 // configure the transmit interrupt
4 U1STAbits.UTXISEL = 2;
```

Das `UxBRG` Register setzt die Übertragungsgeschwindigkeit. Die Geschwindigkeit wird mit der folgenden Formel definieren : $BaudRate = \frac{FCY}{16 \cdot (UxBRG+1)}$

```
1 // set the baudrate
2 U1BRG = (FCY/baud)/16-1;
```

Der UART-Modul hat keine spezifischen Pin. Die Ein-und Ausgänge des Moduls können überall `RPx` Pin gemultiplext werden. Dazu muss die folgende Sequenz ausgeführt werden.

```
1  // Unlock Registers
2  __builtin_write_OSCCONL(OSCCON & 0xbf);
3
4  // Assign U1RX To Pin RP0
5  RPINR18bits.U1RXR = 0;
6
7  // Assign U1TX To Pin RP1
8  RPOR0bits.RP1R = 3;
9
10 // Lock Registers
11 __builtin_write_OSCCONL(OSCCON | 0x40);
```

uart_enable_interrupt(ena: BOOL) Diese Funktion wird verwendet, um die Unterbrechung des UART Module aktivieren.

```
if(ena == true)
{
    // Clear the flag
    IFS0bits.U1RXIF = 0;
    IFS0bits.U1TXIF = 0;

    // enable the interrupt
    IEC0bits.U1RXIE = 1;
    IEC0bits.U1TXIE = 1;
}
else
{
    // disable the interrupt
    IEC0bits.U1RXIE = 0;
    IEC0bits.U1TXIE = 0;
}
```

uart_get_in_buffer() Diese Methode gibt einen Zeiger auf das Buffer von UART Empfang.

```
return uart_data_in;
```

uart_get_out_buffer() Diese Methode gibt einen Zeiger auf das Buffer von UART Übertragung.

```
return uart_data_out;
```

uart_get_frame_in_length() Diese Funktion gibt die Anzahl der Charakter empfangen seit dem letzten Lese.

```
UINT8 length;

length = uart_ptr_read - uart_data_in;
uart_ptr_read = uart_data_in;

return length;
```

uart_send_frame(length: UINT8) Dieser Funktion beginnt die Übertragung der `length` Zeichen Inhalt der Tabelle `uart_data_out`.

```
if(length <= UART_BUFFER_SIZE)
{
    uart_ptr_end_of_frame = uart_data_out+length;
    uart_ptr_write = uart_data_out;

    // Clear the flag
    IFS0bits.U1TXIF = 0;

    // enable the interrupt
    IEC0bits.U1TXIE = 1;

    U1TXREG = *uart_ptr_write;
    uart_ptr_write++;
}
```

U1RXInterrupt() Diese Funktion ist jedes Mal gerufen, wenn ein Charakter wird bekommen. Sie kopiert das empfangene Charakter in der Empfangsbuffer.

```
1  *uart_ptr_read = U1RXREG;
2  if(uart_ptr_read >= (uart_data_in + UART_BUFFER_SIZE) ||
3          uart_ptr_read < uart_data_in)
4      uart_ptr_read = uart_data_in;
5  else
6      uart_ptr_read++;
7
8  XF_pushEvent(evCommDataIn,true);
9  IFS0bits.U1RXIF = 0;
```

U1RXInterrupt() Diese Funktion ist jedes Mal gerufen, wenn ein Charakter wird verschicken. Er überträgt dann die in der `uart_data_out` Buffer enthaltenen Charakter, bis es die durch den `uart_ptr_end_of_frame` Zeiger festgelegte Grenze erreicht.

```
1  if(uart_ptr_write == uart_ptr_end_of_frame)
2  {
3      U1TXREG = *uart_ptr_write;
4      XF_pushEvent(evCommEndOfTransmission,true);
5      uart_ptr_write = uart_data_out;
6      IEC0bits.U1TXIE = 0;
7  }
8  else
9  {
10     U1TXREG = *uart_ptr_write;
11     uart_ptr_write++;
12 }
13
14 XF_pushEvent(evCommDataOut,true);
15 IFS0bits.U1TXIF = 0;
```

5.3.7 CAD Treiber

Der DA-Wandler ist, um verschiedene Spannungspegel an das Gate des CNFET erzeugen verwendet. Es wird durch die `Acquisition` Klasse verwendet. Seine Schnittstelle wird durch das folgende Diagramm definiert.

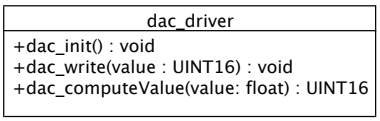


Abbildung 52: ADC Klasse Diagramm

dac_init() Diese Funktion initialisiert die Port von Ein-und Ausgängen zur Konverter-Schnittstelle.

Ausgang : Befehl

Funktion	I/O PORT
CKL	PORTB 12

Ausgang : Signale

Funktion	I/O PORT	Funktion	I/O PORT
DA0	PORTE 0	DA8	PORTG 6
DA1	PORTE 1	DA9	PORTG 7
DA2	PORTE 2	DA10	PORTG 8
DA3	PORTE 3	DA11	PORTG 9
DA4	PORTE 4	DA12	PORTB 5
DA5	PORTE 5	DA13	PORTB 4
DA6	PORTE 6	DA14	PORTB 2
DA7	PORTE 7	DA15	PORTB 3

Abbildung 53: I/O Pins

dac_write(value: UINT16) Diese Funktion erzeugt eine Sequenz für eine neue Konvertierung mit dem `value` Wert.

```
1  // write the value on PORT
2  LATE = 0x00FF & value;
3  LATG = 0x03C0 & (value>>2);
4  DA12 = (0x1000 & value)>>12;
5  DA13 = (0x2000 & value)>>13;
6  DA14 = (0x4000 & value)>>14;
7  DA15 = (0x8000 & value)>>15;
8
9  // set the clock
10 CKL = 0;
11 CKL = 1;
```

dac_computeValue(value: UINT16) Diese Funktion gibt die Nummer, die mit der Funktion `dac_write` übergeben muss, um eines Spannungswert von `value` zu generieren.

```
1  return (value + 8.2) * 65536/16.4;
```

5.3.8 CDA Treiber

Der AD-Wandler wird verwendet, um den Wert der Drain-Strom des Transistors zu messen. Es wird durch die `Acquisition` Klasse verwendet. Seine Schnittstelle wird durch das folgende Diagramm definiert.

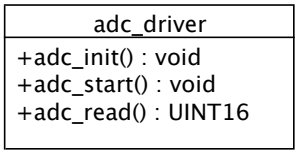


Abbildung 54: AD Klass Diagramm

adc_init() Diese Funktion initialisiert die Port von Ein-und Ausgängen zur Konverter-Schnittstelle.

Eingang : Befehl

Funktion	I/O PORT
BUSY	PORTF 3

Eingang : Signale

Funktion	I/O PORT	Funktion	I/O PORT
AD0	PORTC 12	AD8	PORTC 13
AD1	PORTC 15	AD9	PORTD 6
AD2	PORTD 8	AD10	PORTD 5
AD3	PORTD 9	AD11	PORTD 4
AD4	PORTD 10	AD12	PORTD 3
AD5	PORTD 0	AD13	PORTD 2
AD6	PORTC 13	AD14	PORTD 1
AD7	PORTD 7	AD15	PORTC 14

Ausgang : Befehl

Funktion	I/O PORT	Funktion	I/O PORT
SHDN	PORTF 0	CS	PORTF 1
CONV	PORTF 4	RD	PORTF 5

Abbildung 55: I/O Pins

adc_start() Diese Funktion startet eine neue Umwandlung.

```
1  while(!BUSY);
2
3  // start of a convertion
4  CONV = 0;
```

adc_read() Diese Funktion liest das Ergebnis der Umwandlung auf der verschiedenen Eingangsport.

```
1  UINT16 value;
2
3  // start the conversion
4  adc_start();
5
6  // wait the end of the conversion
7  while(!BUSY);
8
9  // compute the adc value
10 value = AD0 + (AD1<<1) + (AD2<<2) + (AD3<<3);
11 value += (AD4<<4) + (AD5<<5) + (AD6<<6) + (AD7<<7) ;
12 value += (AD8<<8) + (AD9<<9) + (AD10<<10) + (AD11<<11);
13 value += (AD12<<12) + (AD13<<13) + (AD14<<14) + (AD15<<15);
14
15 // restor the cmd value
16 CONV= 1;
17
18 // return the read value
19 return (value-32768);
```

5.3.9 AnalogOut Treiber

Der 10 bits interne DA Wandler des CPU ist für Debug verwendet. Seine Schnittstelle wird durch das folgende Diagramm definiert.

AnalogOut_driver
+AnalogOut_init() : void +AnalogOut_write(value : UINT16) : void

Abbildung 56: AnalogOut Klass Diagramm

AnalogOut_init() Diese Funktion initialisiert den analogen Ausgang und dem Konverter-Modul.

```
1 // configure the PORT in analog mode
2 ANSBbits.ANSB13 = 1;
3
4 // configuration of the DAC module
5 DAC2CON = 0x8082;
```

AnalogOut_write(value: UINT16) Dieses Funktion schreibt in dem Ausgangsregister des Wandlers.

```
1 // write the value on the register
2 DAC2DAT = value & 0x03FF;
```

5.4 Sensor

Die Dummy-Sensor ist mit der Ersatzschaltung gemacht. Es ist entworfen, um die Schaltung zu testen.

Um den Sensor zu realisieren, müssen die Ersatzschaltung zu implementieren. Der Wert der meisten Komponenten bekannt sind. Der unbekannte Partei ist der Transistor in der Mitte des Diagramms. Es Modelle eine perfekte Transistor. Die Herausforderung ist daher, um die perfekte Transistor zu erreichen.

5.4.1 Stromquelle

Ein Transistor durch eine spannungsgesteuerten Stromquelle modelliert werden. Es ist möglich, einen solchen Quelle mit ein Operationsverstärker zu erreichen.

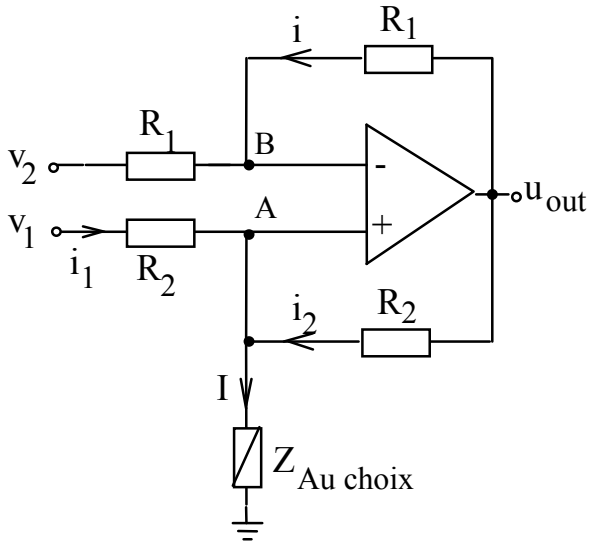


Abbildung 57: Strom - Spannung Wandler

der Ausgang dieser Schaltung ändert sich entsprechend der folgenden Gleichung :

$$I = i_1 + i_2 = \frac{V_2}{R_2} \tag{2}$$

Dieser Schaltung bildet nur die ohmsche Region der Charakteristik. Eine Schwellenspannung kann durch Zugabe einer Zener-Diode mit der Eingangs erhalten werden. Der Sättigungsbereich ist durch Verringern der Lieferung des Operationsverstärkers getan, um die Sättigung erhalten zu werden.

Diese Lösung ist jedoch nicht durchführbar. Es hat einen Eingang für das Gate und einen Ausgang für die Source, aber keine Eingang für den Drain. Es ist nicht möglich, sie in der Äquivalenzschaltung zu integrieren. Allerdings wäre es immer noch möglich sein, die Schaltung zu befestigen, um Testen durchzuführen.

5.4.2 MOSFET

Eine andere Möglichkeit ist es, einen MOSFET-Transistor mit der gleichen Eigenschaften wie CNFET wählen und ihn integriert in der Ersatzschaltung. Die wesentlichen Parameter sind: die Schwellenspannung und die Kanalwiderstand.

Schellenspannung Die Schwellenspannung, die zwischen 2 und 4 V sein muß. Diese Informationen werden direkt in der Tabelle CNFET Charakteristik gefunden und ist auch auf dem Graphen der Kenn gelesen.

Kanalwiderstand Der Widerstandswert des Kanals wird von der charakteristischen abgeleitet. Wenn der Transistor nicht leitet, neigt der Widerstandswert unendlich. Wenn der Transistor leitet, der maximale Strom, der durch den Transistor fließen kann, ist 7 [μA]. Der Wertebereich ist von $R_{ch} = \frac{V_{sd}}{I_d} = \frac{30mV}{7\mu A} = 43[K\Omega]$ bis $\infty[\Omega]$.
Der MOSFET ist nach diesen zwei Parametern und der verfügbare Bestand ausgewählt. Die einzige Komponente füllt am besten diese Daten ist der `BSS138`. Es hat eine Schwellenspannung von 1,2 [V] und eine Kanalwiderstand von 1,4 [Ω].
Um den richtigen Wert des Stroms zu erhalten, werden der Widerstand der Source-und Drain halbiert. Der maximale Strom in der Source-Drain-Zweig wird :

$$i_{Dmax} = \frac{V_{sd}}{R_{source} + R_{Drain}} = \frac{30mV}{25K + 25K} = 600[nA] \tag{3}$$

6 TEST

Viele kleine Tests wurden im Rahmen dieses Projektes durchgeführt. Es ist leider nicht möglich, alle zu dokumentieren. Dieser Abschnitt führt die Tests, die mir erlaubt haben, um die wichtigen Punkte des System zu validieren.

6.1 Eingebettet System

die Messschaltung wurde während der Semester-Projekt entwickelt. Es wurde jedoch nicht getestet. Hierzu wird ein Testprotokoll realisiert. Es die wichtigen Funktionen der Schaltung, die geprüft werden müssen enthält. Einige Hardware brauchen eine Software zu laufen. Dies ist der Grund, dass das Programm zusammen mit der Schaltung getestet wird. Die meisten Messungen mit einem Oszilloskop YOKOGAWA DLM2054 2.5GS/s - 500MHz aufgenommen. Die Protokoll für die Testschaltung festgelegten ist in Anhang G vorhanden.

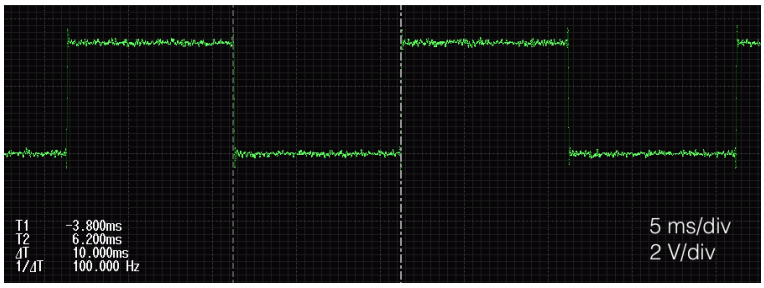
6.1.1 Test des Schaltung

Die Schaltungsteile, die ohne Programmierung getestet werden können, sind hierin beschrieben.
Die ersten Messungen sind auf Spannungsquellen durchgeführt. Sobald ihre Wert sind gültig, werden die Spannungsreferenzen gemessen. Der letzte Schritt ist, zum Programmieren des Prozessors zu prüfen. Die Testen sind im Testprotokoll enthalten.

6.1.2 XF

Der erste geprüft Teil des Codes ist das XF. Es verwendet den Hardware `Timer 1`, um die Software Timer zu erzeugen. Der `Timer 1` ausgebildet ist, um eine Zeitbasis alle 10 [ms] zu generieren. Ein Prozessor Ausgang wird um jedem Timer-Interrupt invertiert.

Ergebnis:



Quelle:

```
void _ISR _T1Interrupt(void)
{
// togel the DEBUG1 pin
    DEBUG1 = !DEBUG1;
    XF_decrementAndQueueTimers();
    IFS0bits.T1IF = 0;
}
```

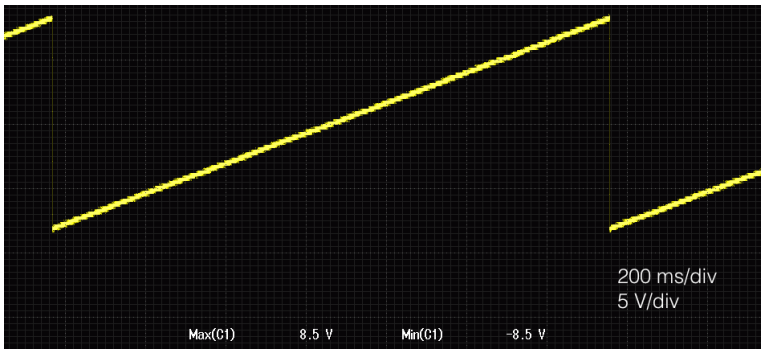
Abbildung 58: XF Timer-Interrupt

Wir sehen, daß die Periode des Zeitbasis 10 [ms] ist. Die Zeitbasis ist daher richtig.
Die Hardware Timer sind in Erfassung Teil getestet.

6.1.3 CDA Treiber

Der DA-Wandler musst eine Rampe von ± 8 [V] erzeugen. Die Erzeugung einer Rampe zeigt, dass alle innere Werte möglich sind.

Ergebnis:



Quelle:

```
i=0;
while(1)
{ // infinity loop

    dac_write(i);
    i++;
}
```

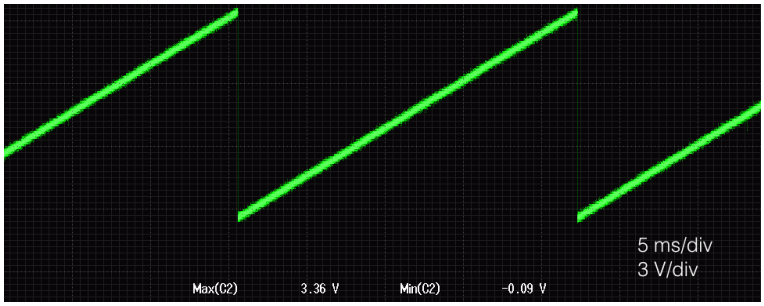
Abbildung 59: CDA Ausgang

Das Bild bestätigt somit die Schnittstelle zwischen dem Prozessor und dem Wandler. Die Messung wird auf dem Testpunkt des Gate erfolgen.

6.1.4 Analog Ausgang

Die Validierung des internen Konverter ist wie der CDA Validierung ausgeführt. Der DA-Wandler musst eine Rampe von $0 \rightarrow 3.3$ [V] erzeugen.

Ergebnis:



Quelle:

```
i=0;
while(1)
{ // infinity loop

    analogOut_write(i);
    i++;
}
```

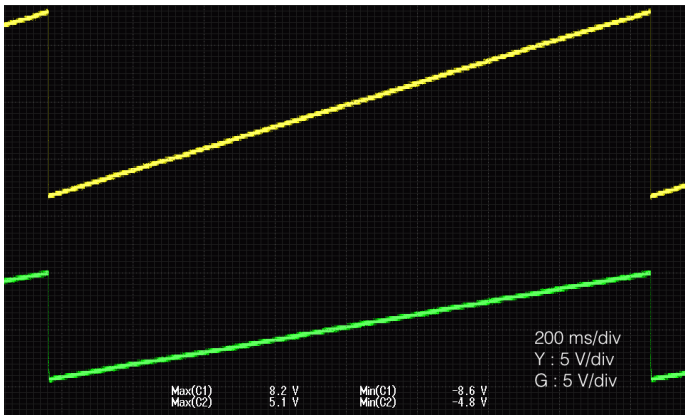
Abbildung 60: Analog Ausgang

Die Messung erfolgt an dem `AnalogOut` Testpunkt geführt. Man kann auch der richtige Betrieb des Wandlers sehen.

6.1.5 CAD Treiber

Die Validieren des AD-Wandler ist in zwei Teilen durchgeführt. Erstens muss die Strom-Spannungs-Wandler validiert. Dafür wird eine Rampe mit dem DA-Wandler erzeugt. Diese Rampe ist mit einem Widerstand zwischen dem Gate-Ausgang und dem Drain-Einlaß angelegt. Die Messung erfolgt an dem `V_out` Testpunkt durchgeführt.

Ergebnis:



Quelle:

```
i=0;
while(1)
{ // infinity loop

    dac_write(i);
    i++;
}
```

Abbildung 61: Rampe Ausgang des iU-Wandler

Die Rampe zeigt, dass der Bereich der Ausgangsspannung korrekt ist. Es ist nun notwendig, das Verhalten von ein Sprungantworten zu überprüfen.

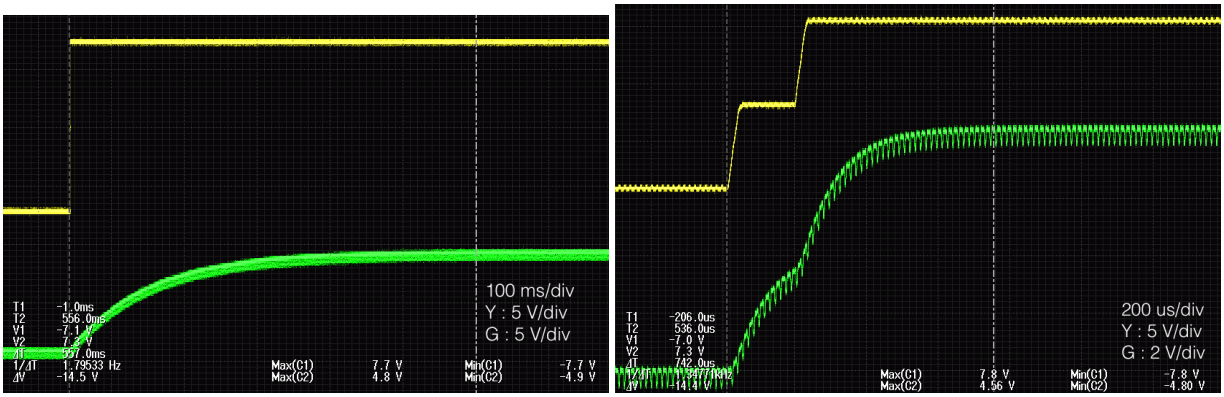


Abbildung 62: Anstiegszeit vor und nach der Änderung der Kapazität

Die erste Antwort ist mit einer Kapazität von 100 [nF]. Die Zeitkonstante ist : $\tau = R \cdot C = 1M \cdot 100n = 100[ms]$. Dieser Wert stabilisiert nach einer Zeit von $5\tau = 500[ms]$. Die Hoch Mindestzeit ist gleich 0.5 [ms]. Die Zeitkonstante musst reduziert werden.
Tests wurden mit einer Kapazität von 100 [pf] durchgeführt wird, um eine Zeitkonstante von 100[ns] zu haben. Jedoch mit dieser Kapazität, gibt es ein Rauschen in der Messung von fast 1V.
Um das Rauschen in der Messung zu verringern, wird eine Kapazität von 470 [nF] montiert. Es besteht eine Schwingung, aber leicht reduziert. Dies erhöht die Zeit konstant. Aber die meisten der in der Dissertation durchgeführten Tests sind in einer Zeit t_high von 5 ms.

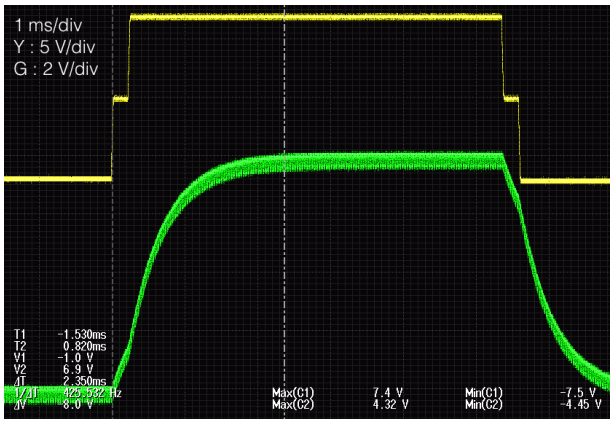
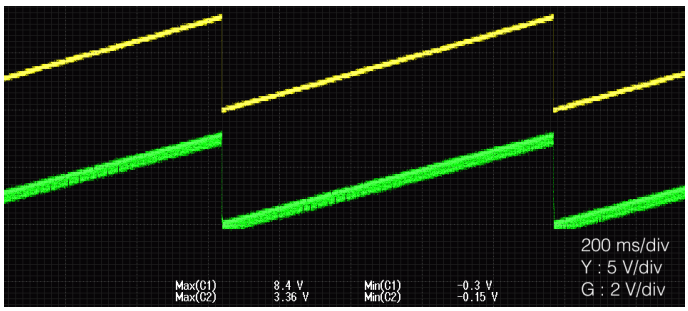


Abbildung 63: Geräuschmessung mit einer Kapazität von 470 [nf]

Der zweite Teil des Tests ist es, die Ausgangswerte des Wandler überprüfen. Um diesen Test durchzuführen, wird eine Rampe mit dem DA-Wandler erzeugt. Für jeden Ausgangswert , wird eine Probe entnommen. Diese Proben werden von den internen DA-Wandler verwendet. Das Ergebnis am `Analog_out` Ausgang, müssen eine Kopie des Gate Ausgang sein.

Ergebnis:



Quelle:

```
i=0;
while(1)
{ // infinity loop

    dac_write(i);
    value = adc_read();
    analogOut_write(value>>6);
    if(i<STIMULUS_ZERO)
        i = STIMULUS_ZERO;
    else
        i++;
}
```

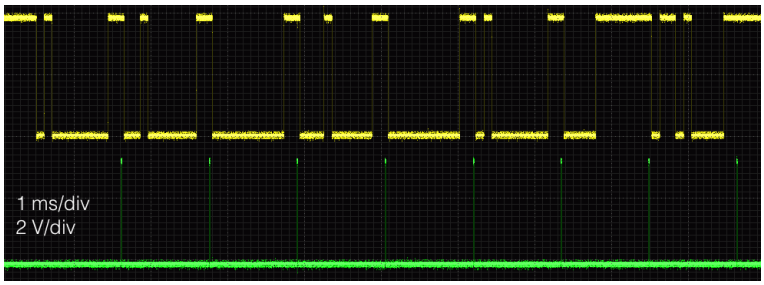
Abbildung 64: Validierung der abgetasteten Werte

Die Grüne Rampe, die die gelbe Kopie ist zeigt das ordnungsgemäße Funktionieren des Konverters.
Dieses Ergebnis wird nicht direkt erhalten gewesen. Ein Unsachgemäße Schweiß der Komponente erstellte Konvertierungsfehler.

6.1.6 UART Treiber

Die UART-Modul wird verwendet, um die Modbus Frame zu lesen. Dazu verwendet er die Unterbrechungen. Zum Testen der Funktion des Moduls wird ein Rahmen übertragen. Ein Ausgang des Prozessors ist um jeder Unterbrechung aktiviert.

Ergebnis:



Quelle:

```
void _ISR _U1RXInterrupt(void)
{
    DEBUG1 = 1;
    IFS0bits.U1RXIF = 0;
    DEBUG1 = 0;
}
```

Abbildung 65: uart Unterbrechung

Stellen wir fest, dass es für jeden neuen Charakter eine Unterbrechung gibt. Der Betrieb des Getriebes wird in den Modbusteil geprüft.

6.1.7 Modbus

Das Modbus Teil muss ein Ende des Frames erkennen. Um dies zu erreichen, wird ein Hardware-Timer verwendet. Die Unterbrechung wird voraussichtlich um ein Zeit von $2.5char = 2.5 \cdot 11(bits) \cdot \frac{1}{9600} = 2.86[ms]$ ankommen.

Ergebnis:

Quelle:

```
// state machine of the communication
case ST_SM_COMM_NEW_DATA:
   if (ev == evCommTM)
   { // end of trame detected
      DEBUG2 = 1;
      me->sm_Communication =
        ST_SM_COMM_CHECKING_REQUEST;
      DEBUG2 = 0;
   }
   break;
```

Abbildung 66: Erfassen des Endes eines Kommunikation

Wie auf dem Bild des Oszilloskops ersichtlich, tritt der Interrupt zum richtigen Zeitpunkt und der Rahmen erfasst wird.

6.1.8 Erfassung

Der erste Schritt ist die Erzeugung der Messsequenz zu testen. Wichtige Parameter sind die Zeit-und Spannungspegel. Um die Initialisierung Sequenzen und Erfassungsequenzen zu unterscheiden, wird der Amplitude des Erwerb reduziert. Die Initialisierungssequenz hat eine Amplitude von 7,5 V, und die Messsequenz von 2V. Eine Frei Prozessorausgang wird verwendet, um zu signalisieren, wenn eine Probe genommen.

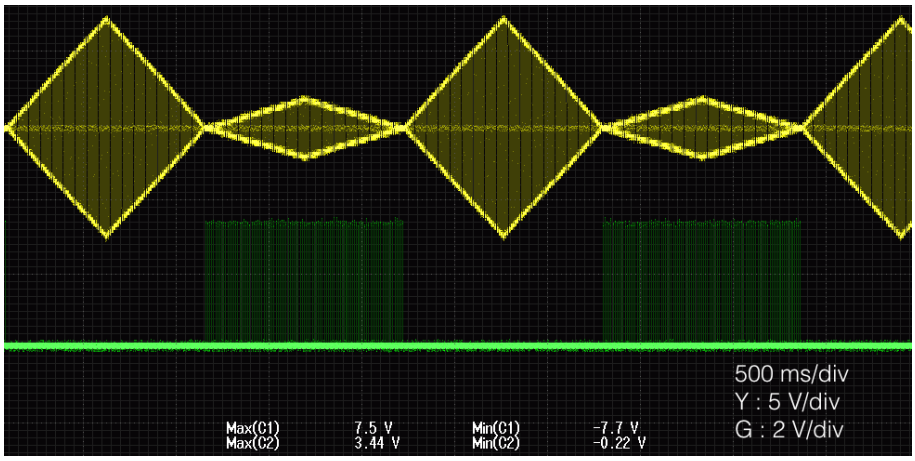


Abbildung 67: Erzeugung der Messsequenz

Die Initialisierungssequenz hat eine Amplitude von 7,5 V, und die Messsequenz von 2V. Die Proben sind nur an der Messsequenz genommen.

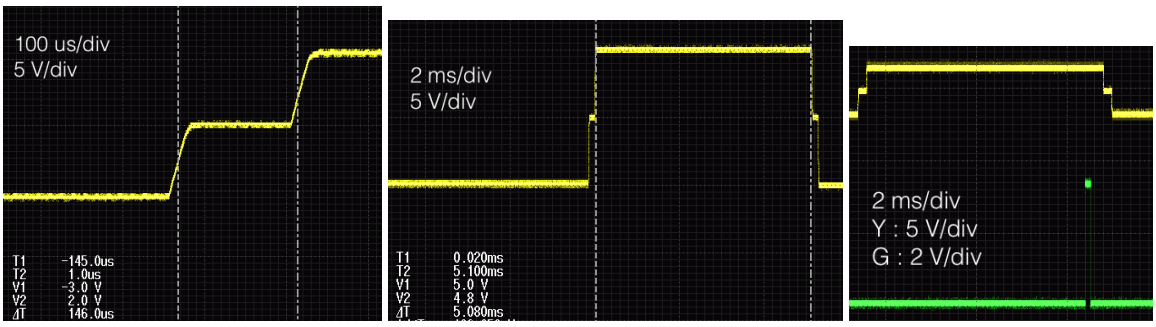


Abbildung 68: Zeit des Sequenz

Die Verwendung von Ereignissen macht die zeit weniger genaue. Wenn die Unterbrechung auftritt, müssen wir uns die Verteilung Zeit des Veranstaltung hinzufügen. Diese Verteilung Zeit beträgt ca. 100 [μs].
Die Zeit von 5 ms ist anständig. Für den Wert von 100 ms, muss der Timer-Wert sehr niedrig für eine gute Zeit eingestellt werden. Der Impuls des repräsentative Probenahme ist gut aufgestellt.

Die ersten Erwerbs Tests wird mit einem Widerstand erreicht. Eine 4 M Widerstand ist zwischen dem Gate und dem Drain verbunden. Um die Ergebnisse zu sehen, wird der interne DA-Wandler verwendet. Die Messung wird einmal mit 2 [V] Amplitude durchgeführt und einmal mit 8 [V].

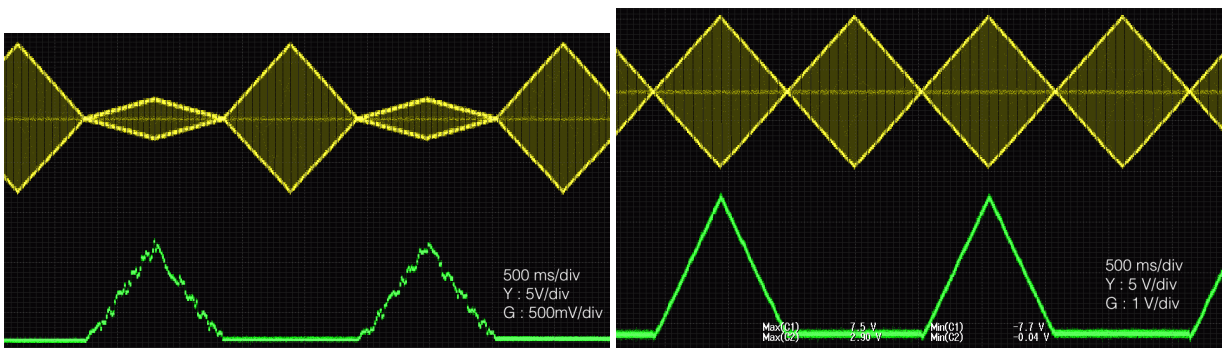


Abbildung 69: Messung eines Widerstands

Die grüne Kurve stellt den Wert der Proben an der inneren DA-Wandler. Wie erwartet werden kann, ist das Ergebnis ein Dreieck. Die lineare Widerstandskennlinie durchläuft einen Spiegeleffekt.

6.2 Benutzerschnittstelle

Der Code ist weitgehend aus dem scada Labor genommen, können wir die Server-Seite als validiert sagen. Der neue Teil ist die Web-Seite und serielle Modbus.

6.2.1 Verbindung zur Datenbank

Wir müssen sicherstellen, dass der Server in die erstellt Datenbank verbinden. Um dies zu tun, wird eine Nachricht auf dem Terminal für jeder neuen erstellt Punkt aus der Datenbank angezeigt.

```
Model > new FLOAT Input Datapoint :G1_1.M1
Model > new FLOAT Input Datapoint :G1_1.M2
Model > new FLOAT Input Datapoint :G1_1.M3
Model > new FLOAT Input Datapoint :G1_1.M4
Model > new FLOAT Input Datapoint :G1_1.M5
Model > new FLOAT Input Datapoint :G1_1.M6
Model > new FLOAT Output Datapoint :G1_1.P3
Model > new BOOLEAN Output Datapoint :G1_1.P1
Model > new BOOLEAN Output Datapoint :G1_1.P2
Model > new BOOLEAN trigger Datapoint :G1_1.S1
Model > new BOOLEAN trigger Datapoint :G1_1.S2
Model > new BOOLEAN trigger Datapoint :G1_1.S3
```

Man kann sehen, dass der Server die Punkte in der Datenbank gefunden hat.

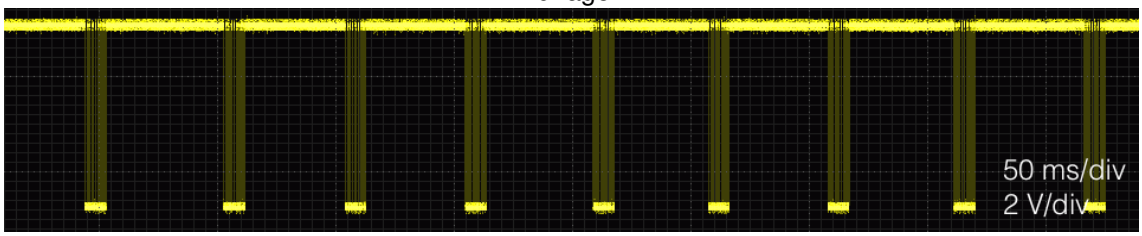
6.2.2 serielle Kommunikation

Der Server kommuniziert mit einer virtuellen seriellen Port. Der Server enthält eine Funktion, um die verfügbaren Ports auf dem Terminal anzuzeigen.

```
PORT 0 : /dev/tty.RN42-20CB-SPP
PORT 1 : /dev/cu.RN42-20CB-SPP
PORT 2 : /dev/tty.Bluetooth-Modem
PORT 3 : /dev/cu.Bluetooth-Modem
PORT 4 : /dev/tty.iPhonedeloulou-Wireless
PORT 5 : /dev/cu.iPhonedeloulou-Wireless
PORT 6 : /dev/tty.Bluetooth-Incoming-Port
PORT 7 : /dev/cu.Bluetooth-Incoming-Port
PORT 8 : /dev/tty.usbserial-A800fvOJ
PORT 9 : /dev/cu.usbserial-A800fvOJ
```

in unserem Fall ist der Port `/dev/tty.usbserial-A800fvOJ` verwendet. Mit der Auswahl der richtigen Anschluss müssen die Daten über die serielle Schnittstelle übertragen werden.

Abfrage :



Abfrage und Antwort :

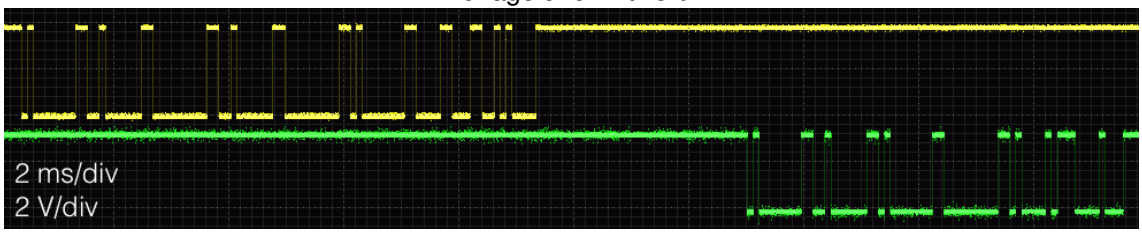


Abbildung 70: serielle Datenübertragung

Beachten Sie, dass die Übertragung korrekt arbeitet. Für jede Antwort ein neuer Wert gespeichert wird. Diese Werte werden auf dem Terminal angezeigt.

```
1 Insert in historian> id =G1_1.M1 value :4.259557
2 Insert in historian> id =G1_1.M6 value :7.678397
3 Insert in historian> id =G1_1.M5 value :1.503984
4 Insert in historian> id =G1_1.M4 value :6.816194
5 Insert in historian> id =G1_1.M3 value :8.71914
6 Insert in historian> id =G1_1.M2 value :6.161373
```

6.2.3 Websocket

Die websocket wird für die Kommunikation mit dem Web-Seite verwendet. Das Kommunikationsprotokoll ist auf dem `ezHMI`[5] basiert. Die Hauptinformation, die gesendet hat sind den Sensorausgabewerten.

Für die Testen, die Inhalte der `pollImpl` Funktion der `Modbus` Klasse wird durch die folgende Zeile ersetzt.

```
1 target.setValue(false);
```

Dies ermöglicht die Prüfung ohne die Modbus Kommunikation.
Der Test ist um die Kommunikation zu prüfen. Für diese, die Webseite, fügt ein Punkt auf dem Graphen für jeden empfangenen Daten.

```
1 Insert in historian> id =G1_1.M1 value :9.918816
2 Insert in historian> id =G1_1.M1 value :5.7589226
3 Insert in historian> id =G1_1.M1 value :6.2748675
4 Insert in historian> id =G1_1.M1 value :4.1889887
5 Insert in historian> id =G1_1.M1 value :0.06352604
6 Insert in historian> id =G1_1.M1 value :6.7691708
7 Insert in historian> id =G1_1.M1 value :4.259557
8 Insert in historian> id =G1_1.M1 value :2.7092385
```

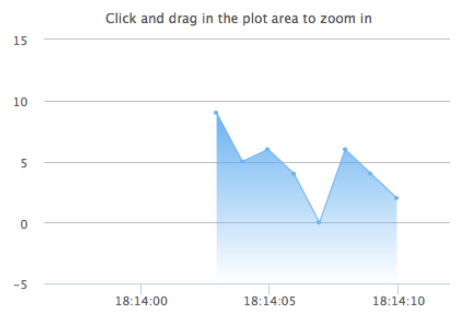


Abbildung 71: Kommunikation Test

Der Graph zeigt, daß die Kommunikation richtig funktioniert.

6.3 Sensor

die Dummy-Sensor verwendet wird, um den Betrieb des Systems zu demonstrieren. Dazu, sind drei verschiedene Testserien geführt : Linear Test, Schwellentest und Test mit dem Transistor. Für jeden dieser Tests wird eine andere Ersatzschaltung des Sensors verwendet. Das Ergebnis wird mit der Benutzerschnittstelle in den Debug-Modus angezeigt.

6.3.1 Linear Test

Der linear Test wird durch ein Widerstand zwischen dem Gate Ausgang und dem Source Eingang durchgeführt. Der Wert des Widerstandes ist so gewählt, um den maximalen Strom im maximalen Gatespannungs erhalten : $R = \frac{U_{Gmax}}{I_{dmax}} = \frac{7.5V}{2\mu A} = 3.75 \Rightarrow 4[M\Omega]$.

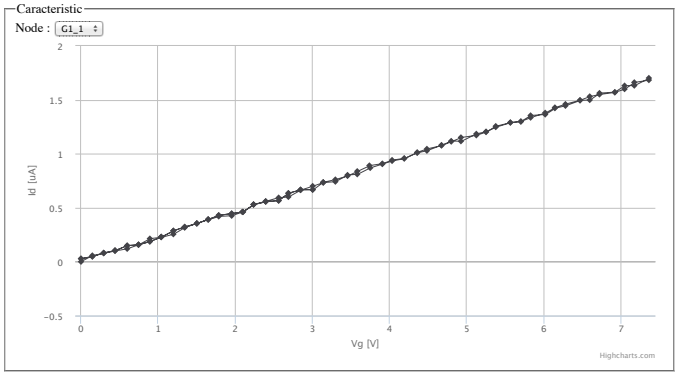


Abbildung 72: Linear Test

Die Ausgangswerte für diese Messung sind: ROS1 = 0.229 [V], ROS2 = 1.7 [μA] und ROS3 = 6.37 [VμA]. ROS1 gibt einen Schwellenwert am Anfang an, was richtig ist. ROS2 zeigt einen maximalen Strom von 1.7 [μA], die auf der Kurve leicht nachprüfbar ist. Schließlich gibt ROS3 einen Wert von 6.37 [VμA], die leicht von einer linearen Kennlinie überprüft werden kann.

$$ROS3 = \int_0^{V_{Gmax}} I_d dV_g \approx \frac{\Delta V_g \cdot \Delta I_d}{2} = \frac{7.5V \cdot 1.7}{2} = 6.375[V\mu A] \tag{4}$$

Wir können also sagen, dass die Berechnung der Ausgangswerte korrekt sind.

6.3.2 Schwellentest

Der Linear Test kann nicht wirklich überprüfen den Schwellenwert ROS1. Um diesen Wert zu bestätigen, muss der Sensor gewechselt werden. Mit dem Zusatz von einer Diode in Serie mit dem Widerstand, wird die Kurve linear erst nach Überschreiten des Wertes der Schwellen des Diode. Um einen hohen Schwellenwert haben, wird eine LED eingesetzt.

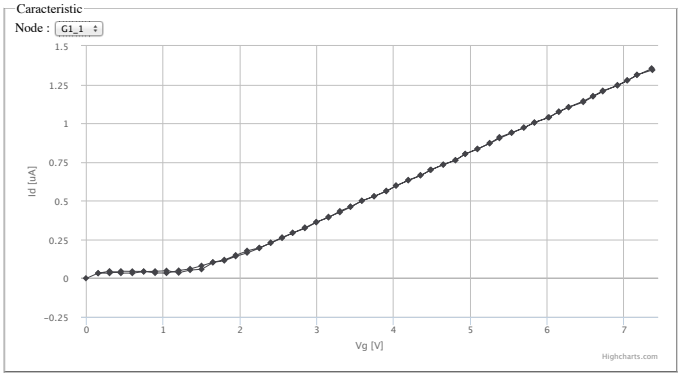


Abbildung 73: Linear Test

Die Ausgangswerte für diese Messung sind: ROS1 = 1.78 [V], ROS2 = 1.34 [μA] und ROS3 = 4.18 [VμA]. Der Schwellenwert ist mit einem Multimeter geprüft. Das Messgerät zeigt 1.799 V, was unser Maß entspricht. Der ROS2 Wert ist wegen der Diode etwas niedriger. Der ROS3 Wert kann auch durch Berechnungen überprüft werden.

$$ROS3 = \int_0^{V_{Gmax}} I_d dV_g \approx \frac{(\Delta V_g - U_{Schwelle}) \cdot \Delta I_d}{2} = \frac{(7.5V - 1.2) \cdot 1.34}{2} = 4.25[V\mu A] \quad (5)$$

Ausgangswerte sind immer noch richtig.

6.3.3 Test mit dem Transistor

Der letzte Test wird mit der Ersatzschaltung des CNFET geführt. Dieser Test prüft das Verhalten der Schaltung mit einem Sensor-Modell, das der Wirklichkeit nahe kommt.

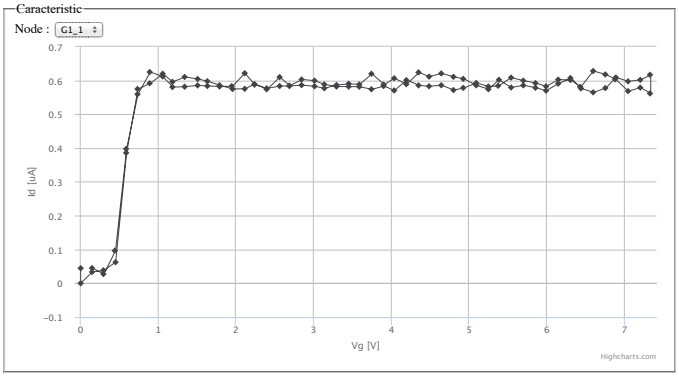


Abbildung 74: Test mit dem Transistor

Die Ausgangswerte für diese Messung sind: ROS1 = 0.45 [V], ROS2 = 0.56 [μA] und ROS3 = 4.09 [VμA]. Die Schwellenspannung ist relativ korrekt. Der maximale Stromwert variiert stark auf die Messung. So war die letzte Probe nicht unbedingt der höchste Wert, aber ist ihr Wert zufriedenstellend. Das Integral kann durch Angleichung geprüft werden.

$$ROS3 = \int_0^{V_{Gmax}} I_d dV_g \approx (\Delta V_g - ROS1) \cdot \Delta ROS2 = (7.5V - 0.45) \cdot 0.56 = 3.92[V\mu A] \quad (6)$$

Der Erwerb ist richtig.

6.4 Diskussion der Ergebnisse

6.4.1 Eingebettet System

Die Schaltung korrekt arbeitet. Die Spannungsreferenzen sind korrekt, und Konvertern sind verwendet, um eine Messcharakteristik zu machen.
Die Spannung am Eingang des AD-Wandlers ist jedoch größer als der Bereich der Wandlungsbereich. Der Eingang arbeitet zwischen 0 $\rightarrow$ 5 V und die Ausgangsspannung des i-U Wandlers ist -5 $\rightarrow$ 5V.
Ein 500 mV Rauschen besteht am Ausgang des i-U Wandlers. Hinzufügen von Kapazität auf der Ausgangsstufe des DA-Wandlers kann dieses Rauschen zu reduzieren. Darüber, die Änderung der Kapazität von Reaktion des iU Wandler könnte die Messung verbessern.

6.4.2 Programmierung

Die Programmierung des Prozessors können die Messungen korrekt durchzuführen. Die Modbus Kommunikation funktioniert mit dem Computer.
Die Zeit von 100 us der Messsequenz hat einige Schwankungen. Die Benutzung des Timer direkt in der Funktion kann die Genauigkeit der Zeitbasis verbessern.

6.4.3 Benutzerschnittstelle

Die Grundfunktionen der Benutzerschnittstelle funktioniert. Grafiken können hinzugefügt oder durch entfernt mit andere Filter und ROS Wert werden. Die Schnittstelle ermöglicht es dem Entwickler, die Schaltung zu stoppen, um die gemessenen Kenn anzuzeigen.
Die Übertragung des kennlinie ist jedoch ein sehr langsamer. Modbus begrenzen die Übertragungsgeschwindigkeit:

$$
\begin{aligned}
t_{char} &= nbr_{bits} \cdot 1/baud = \frac{11}{9600} = 1.14[ms] \\
t_{frame} &= nbr_{char} \cdot t_{char} = 8 \cdot 1.14[ms] = 13.17[ms] \\
t_{transaction} &= t_{request} + t_{compute} + t_{response} = 13.17[ms] + 1[ms] + 13.17[ms] = 27.34[ms] \\
t_{transfert} &= nbr_{point} \cdot t_{transaction} = 200 \cdot 27.34[ms] = 5.46[s]
\end{aligned}
$$

Die Mindestzeit beträgt etwa 5 Sekunden. jedoch, fragt der Server alle Punkte um jeder Umfrage. Wenn es mehr änderbaren Parameter gibt, wird die Transaktion mehr lang sein.

6.4.4 Sensor

Die Messung Tests zeigt die ordnungsgemäße Funktion der Schaltung. Die Messung mit Widerstand gibt eine gute lineare Kennlinie. Die Ausgangswerte sind korrekt berechnet werden. Wir bemerken eine leichte Rauschen in der Messung, aber das scheint nicht gestört. Der Test mit der Diode führt zum gleichen Reflexions.

Der Test mit dem Transistor ist weniger zufriedenstellend. Die Kennlinie enthält viele Messrauschen, die das ROS2 Messergebnis verfälschen. Die Gitterrauschen wird durch den Transistor und der Strom-Spannungswandler verstärkt. Die Kapazität der Ausgangsstufe des DA-Wandlers kann sicherlich diese Ergebnis verbessern.

Wir beachten auch in die Messung, dass die Schwellspannung des MOSFET ist relativ gering. Der gemessene Wert beträgt 0.5 [V], während das Datenblatt sagt 1.2 [V].

7 Zukünftige Entwicklung

7.1 iU Wandler

Der Ausgangs des i→U Wandler ist entworfen worden, um den fließenden Strom des Transistor zu spiegeln. Die Strommessung kann positive oder negative Werte haben. Dies bewirkt auch eine positive und negative Ausgangsspannung. Der Eingang des AD-Wandlers kann eine Spannung von 0 bis 5 V haben. Bei der Erzeugung eines negativen Impulses, wird auch die Spannung am Eingang des AD-Wandlers negativ. Die Eingangsspannung ist nicht mehr innerhalb des Messbereichs. Um dieses Problem zu beheben, muss die Verstärkung des Verstärkers nach dem Wandler ändern.

7.2 HMI

Die xml-Dateien verwendet werden, um einfach das Verhalten der Server ändern. Allerdings muss die Benutzerschnittstelle manuell in dem Fall der Änderung geändert werden. Idealerweise, kann der Benutzerschnittstelle-Code wie die SQL-Abfrage aus XML-Dateien generiert werden. Dazu muss eine XSLT-Transformation durchgeführt werden.

7.3 Modbus

Die Modbusgeschwindigkeit ist abhängig von der RS232. Die Übertragung des Charakteristik kann schneller sein, wenn die Geschwindigkeit von RS232 verändert.

Gelegentlich, tritt ein Übertragungsfehler. Wenn dies geschieht, kann der Server nicht immer die Kommunikation fortzusetzen. Das Problem ist in der Regel durch einen Neustart des Servers gelöst. Eine Verlängerung der Code könnte einfach der Kommunikation wieder starten, wenn ein Kommunikationsfehler tritt. Es ist möglich, dass das Problem der Schweiß der USB-Anschluss ist.

Das Hauptproblem von Modbus ist, dass der Kunde ständig fragen dem Server müsst. Der Server kann nicht einen neuen Wert zu übertragen, wenn sie sich ändert.
Es wäre besser, das Protokoll ändern. Das CAN Protokoll könnte perfekt dieses Problem lösen. Es ermöglicht ein Kommunikationsmodell von Typ Erzeuger - Verbraucher, der besser geeignet um unsere Situation ist. Darüber hinaus ermöglicht die physikalische CAN-Schicht, um mehrere Knoten miteinander zu verbinden.

7.4 Qualität des Messung

Die Spannung, die durch den AD-Wandler gemessene umfasst eine Störung. Es ist eine Schwingung von 66[kHz] und 500 [mV] Amplitude. Diese Störung kommt wahrscheinlich von der Ausgangsstufe des DA-Wandlers, der von der Stufe AD verstärkt wird. Die Ausgangsstufe des DA-Wandlers weist zwei Stelle für den Kondensator. Hinzufügen der Kondensator kann Schwingung zu reduzieren und so verbessern die Qualität der Messung.

7.5 Test mit CNFET

Die bisherigen Tests wurden mit Dummy-Sensor durchgeführt. Das Verhalten der Maßnahme mit einem echten CNFET ist nicht bekannt, obwohl es sich ähnlich wie bei den Tests sein. Es ist daher notwendig, einige tatsächliche Versuche durchzuführen.
In einer ersten Zeit können die Tests mit einer nicht-empfindlichen Sensor durchgeführt werden. Die hierfür verwendeten Sensor ist ein CNFET ohne Schlitz Aussetzen der Nanoröhre an die Umgebungsbereich. Wenn die Tests erfolgreich sind, dann können wir die Schaltung mit einem aktiven Sensor zu testen.

7.6 Zukunft Pflichtenheft

Dieses Projekt ist eine Pilotstudie für eine Schaltung, die in einen komplexeren System integriert werden kann. Die Entwicklung wurde getan, um eine Integration einfach zu sein. Zum Beispiel, die Verwendung von Modbus. Jedoch gibt es einige Verbesserungen, um ein fertiges Produkt zu erhalten.

- Entwicklung einer Mezzanine zum Multiplexen der Eingang des Analog-Digital-Wandler. Prinzip der Switch-Box in der Diplomarbeit [4].
- Entwicklung ein Schaltwandler, um die verschiedener innerer Spannung mit einem einzigen Eingangsleistung zu erstellen.
- Die Ausgangsspannung der internen DA Wandler kann als Ausgangsspannung auf den Wert des Gases verwendet werden.
- Ändern die Hardware, um eine CAN oder RS485 Schnittstelle zu haben.

Diese nicht vollständige Liste ist eine Ergänzung zu den oben genannten Punkten.

8 ABSCHLUSS

Bei diesem Projekt, wird ein NO_2 Sensor realisiert. Die Arbeitsmethode verwendet ist das V Konzept, oder top-down. Nach diesem Prinzip, beginnt der Entwurf mit einem weiten Blick und dann es konzentriert sich auf jedes Element.

Während der Analysephase werden die unterschiedlichen theoretischen Elemente im Projekt verwendet beschrieben. Die Anforderungen und die gesamte Struktur des Systems sind auch in diesem Abschnitt definiert. Während der Entwicklungsphase, werden Sequenzdiagramme in der Analysephase definiert in den Zustandsmaschine übersetzt. Die Umsetzung dieser Zustandsmaschinen werden beschrieben. Änderungen an Scada SuperLab für dieses Projekt, die Überlegungen zur Auslegung eines fiktiven Sensor, um das System zu überprüfen und die Web-Teil sind auch Teil dieses Abschnitts.
Während der Testphase, ist das Funktionieren des Systems beweisen. Zunächst wird der Schaltkreis getestet. Dann kommt die Bestätigung der Programmierung. Schließlich wird eine Reihe von Messungen mit unterschiedlichen fiktiven Sensoren realisiert.

Die Programmierung ermöglicht Messungen mit der Schaltung zu nehmen. Die Architektur der Programmierung kann einfach angepasst werden, wenn Zukünftige Änderungen in der Hardware getan sind. Die bisher durchgeführten Messungen wurden mit einem Dummy-Sensor durchgeführt. Diese fiktive Sensor wurde in diesem Projekt entwickelt, um den Betrieb der Schaltung zu überprüfen. Allerdings ist es nicht ein perfekt model das Verhalten eines CNFET. Dieses Projekt kann mit die angegebenen Schaltungsänderungen im Abschnitt ßukünftige Verbesserungenfortgesetzt werden. Das endgültige Ziel ist Test mit einem CNFET und Gas zu machen.

Mayencourt Louis
4. August 2014

LITERATUR

[1] Linear Technology Corporation. Ltc1608 technical description. 2000.

[2] Linear Technology Corporation. Ltc1668 technical description. 2000.

[3] Mayencourt Louis. Projet de semestre 2014 / détecteur de gaz. *HES-so valais*, 2014.

[4] Moritz Mattmann. *Carbon Nanotube Field Effect Transistors with Al2O3*. PhD thesis, ETH Zurich, 2011.

[5] Medard Rieder Dominique Gabioud Michael Clausen, Patrice Rudaz. ezhmi web socket protocol specification. *Module Systèmes distribués, HES-so valais*, 2014-2015.

[6] Modbus.org. Modbus over serial line specification. *http://www.modbus.org/*, 2006.

[7] Modbus.org. Modbus application protocol specification. *http://www.modbus.org/*, 2012.

9 ANHANG

A ZEITPLAN

B SCADA SUPER-LAB DOKUMENTATION

C INSTALLATION EIN MYSQL-SERVER AUF LINUX

D WEBSEITE CODE

E XML DATEI

F EINGEBETTET SYSTEM CODE

G TEST PROTOKOLL

Ernst-Abbe-Fachhochschule Jena
Hochschule für angewandte Wissenschaften

Hes·so VALAIS WALLIS

Anhang A

Zeitplan

travail de diplôme

10 juin 2014

http://

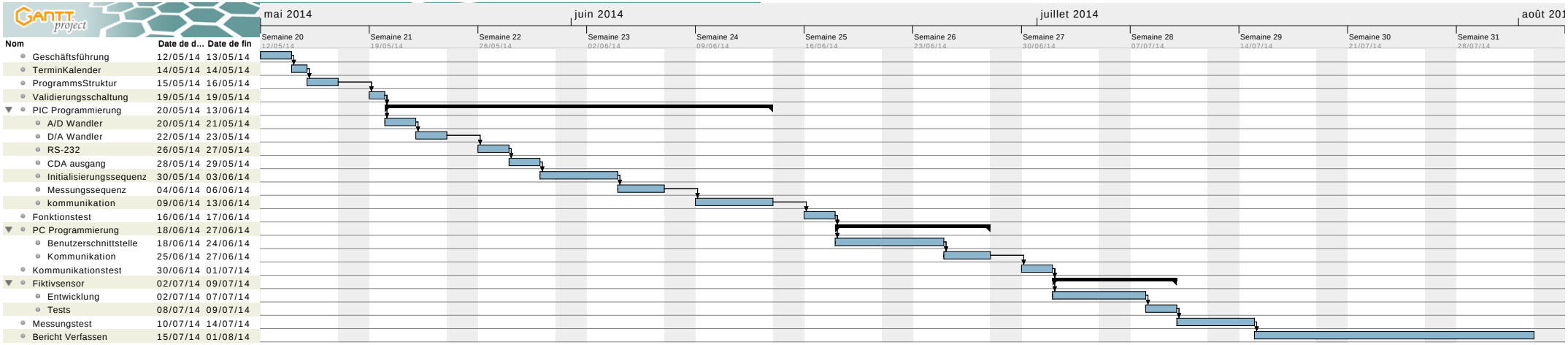
Chef de projet	Mayencourt Louis
Dates de début/fin du projet	12 mai 2014 - 2 août 2014
Avancée	0%
Tâches	22
Ressources	0

réalisation de mon diplôme en Allemagne

Tâches

Nom	Date de début	Date de fin
Geschäftsführung	12/05/14	13/05/14
TerminKalender	14/05/14	14/05/14
ProgrammsStruktur	15/05/14	16/05/14
Validierungsschaltung	19/05/14	19/05/14
PIC Programmierung	20/05/14	13/06/14
A/D Wandler	20/05/14	21/05/14
D/A Wandler	22/05/14	23/05/14
RS-232	26/05/14	27/05/14
CDA ausgang	28/05/14	29/05/14
Initialisierungssequenz	30/05/14	03/06/14
Messungssequenz	04/06/14	06/06/14
kommunikation	09/06/14	13/06/14
Fonktionstest	16/06/14	17/06/14
PC Programmierung	18/06/14	27/06/14
Benutzerschnittstelle	18/06/14	24/06/14
Kommunikation	25/06/14	27/06/14
Kommunikationstest	30/06/14	01/07/14
Fiktivsensor	02/07/14	09/07/14
Entwicklung	02/07/14	07/07/14
Tests	08/07/14	09/07/14
Messungstest	10/07/14	14/07/14
Bericht Verfassen	15/07/14	01/08/14

Diagramme de Gantt



Nom	Date de d...	Date de fin
Geschäftsführung	12/05/14	13/05/14
TerminKalender	14/05/14	14/05/14
ProgrammsStruktur	15/05/14	16/05/14
Validierungsschaltung	19/05/14	19/05/14
PIC Programmierung	20/05/14	13/06/14
A/D Wandler	20/05/14	21/05/14
D/A Wandler	22/05/14	23/05/14
RS-232	26/05/14	27/05/14
CDA ausgang	28/05/14	29/05/14
Initialisierungssequenz	30/05/14	03/06/14
Messungssequenz	04/06/14	06/06/14
kommunikation	09/06/14	13/06/14
Fonktionstest	16/06/14	17/06/14
PC Programmierung	18/06/14	27/06/14
Benutzerschnittstelle	18/06/14	24/06/14
Kommunikation	25/06/14	27/06/14
Kommunikationstest	30/06/14	01/07/14
Fiktivsensor	02/07/14	09/07/14
Entwicklung	02/07/14	07/07/14
Tests	08/07/14	09/07/14
Messungstest	10/07/14	14/07/14
Bericht Verfassen	15/07/14	01/08/14

Ernst-Abbe-Fachhochschule Jena
Hochschule für angewandte Wissenschaften

Hes·so VALAIS WALLIS

Anhang B

Scada superlab Dokumentation

Hes·so VALAIS WALLIS
School of Engineering

SDi / SuperLab

SCADA Server

Dominique Gabioud
Medard Rieder
Michael Clausen
Patrice Rudaz

Table des matières

Introduction

Le SuperLab SCADA illustre l'ensemble des connaissances acquises dans le module SDi, à l'exception de la sécurité, dans un seul travail final.

La Figure 1 ci-dessous donne un aperçu de l'environnement physique de ce laboratoire.

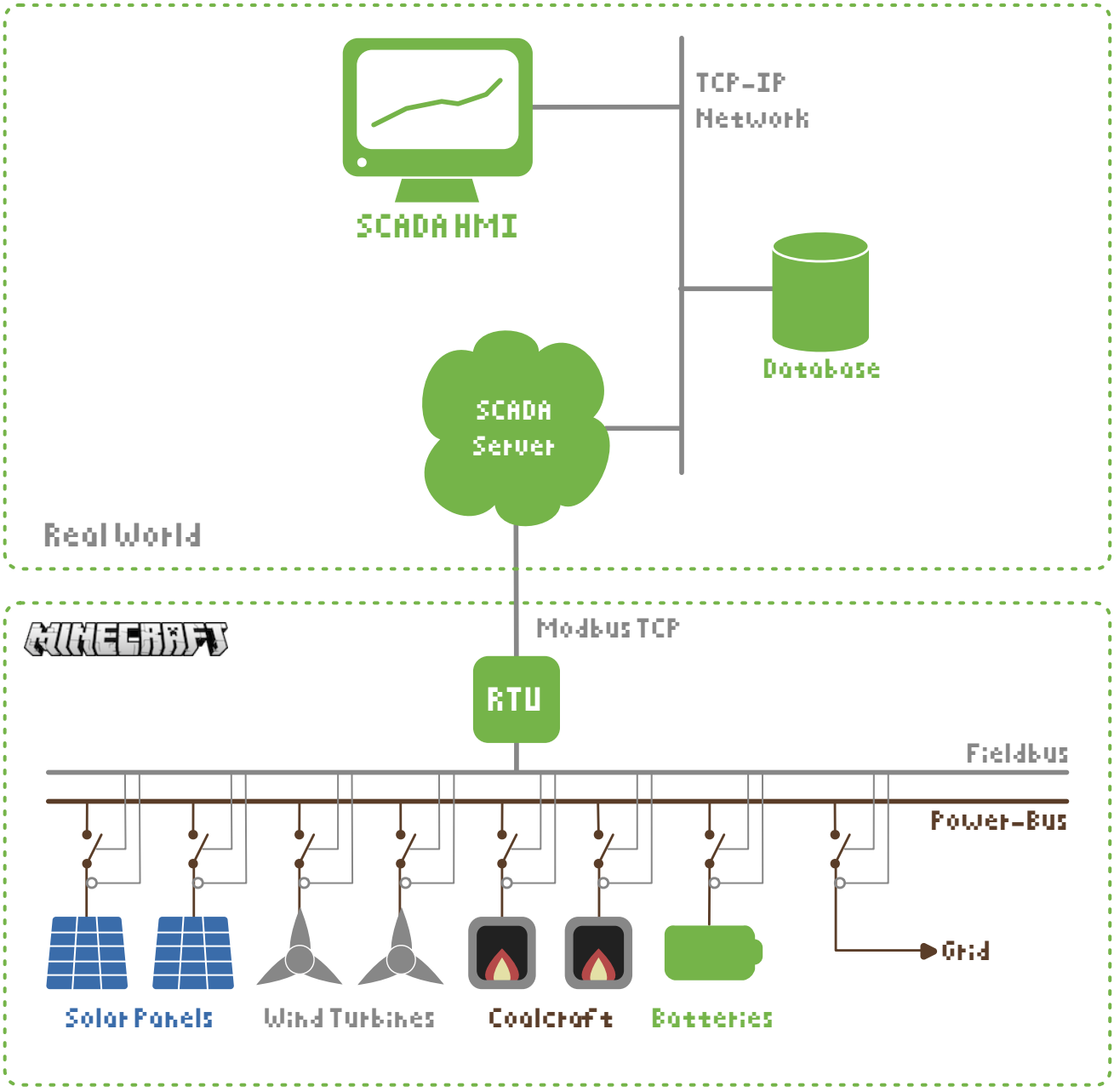


Figure 1: Vue générale de l'environnement physique du SuperLab

Durant ce SuperLab, vous allez travailler intensivement avec cet environnement et votre tâche sera de mettre en œuvre le serveur SCADA représenté au centre de la Figure 1. L'objectif de ce laboratoire est de réviser et d'utiliser les technologies que vous avez apprises dans les cours du module SDi.

Vous rencontrerez les technologies suivantes :

- Réseaux industriels
- Développement de protocoles
- Base de données & XML
- Systèmes d'information distribués

Pour mettre en place un système tel que celui présenté à la Figure 1, trois étapes principales sont nécessaires :

- Déploiement du processus physique (dans le cas de l'exemple : panneaux photovoltaïques, lignes électriques, etc.).
- Intégration système : sélection des composants (automates, serveur SCADA) et programmation / configuration par rapport aux exigences du processus physiques.
- Développement de composants élémentaires (également fabrication dans le cas de composants matériels), à disposition des intégrateurs système (dans l'ordre chronologique, cette étape est la première).

Dans ce superlab :

- Le processus physique est simulé. Il s'agit d'un monde MINECRAFT avec un mode "Energie" qui se veut au plus proche de la réalité. Ce mode a été développé principalement par Charles Papon et Michael Clausen et il vous est fourni (pour l'installer, suivez les indications qui se trouvent sur Cyberlearn). Merci Charles et Michael !
- En ce qui concerne l'intégration système :
  - la régulation locale est effectuée,
  - une passerelle (Remote Terminal Unit, RTU), qui permet de lire ou écrire toutes les données pertinentes, est disponible (lecture, écriture),
  - une interface utilisateur graphique appropriée est configurée
- Le composant serveur SCADA n'est pas (est partiellement) disponible.

L'objectif de votre travail dans ce superlab est le développement d'un composant serveur SCADA générique.

Notez que vous n'avez rien à implémenter dans la couche terrain et dans la couche client SCADA. Ces couches-là ont déjà été implémentées et testées entièrement. Elles vous seront fournies. Néanmoins, vous devez en comprendre exactement le fonctionnement afin de concevoir des interfaces correctes entre la couche "SCADA Server" et les couches en-dessous et en-dessus.

La couche client SCADA est une application basée web qui tourne dans n'importe quel navigateur internet.

Esquisse de l'architecture

La Figure 2 présente une esquisse de l'architecture du système :

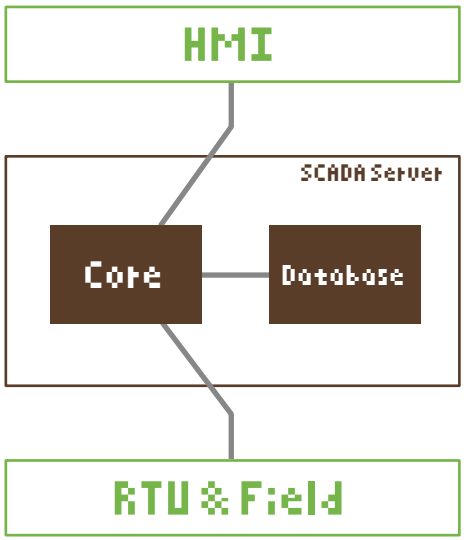


Figure 2: Esquisse de l'architecture

- Le RTU est un serveur qui met à disposition du serveur SCADA (agissant dans ce cas comme client !) des registres accessibles en lecture ou/et en écriture.
- Le bloc HMI voit le système comme un ensemble de composants (instances) dont les types (classes, par exemple « éolienne » ou « batterie ») sont prédéfinis.
- La base de données stocke :
  - la définition des classes de composants pour le bloc HMI,
  - la définition des instances de composants pour le bloc HMI,
  - les paramètres pour l'accès en lecture et en écriture aux registres du RTU,
  - la relation entre les registres du RTU et les composants du HMI,
  - les valeurs historiques.

Définition du système basée XML

Principe

La vision suivante a guidé la conception de l'environnement SCADA :

> **Le travail de l'intégrateur système doit se limiter à élaborer un fichier de configuration.**

Concrètement, deux fichiers de configuration au format XML sont utilisés :

- `scadaClasses.xml` Ce fichier définit les classes de composants pour le bloc HMI. Lorsqu'une nouvelle classe est rajoutée, un composant équivalent doit être créé dans le bloc HMI.
- `scadaSystem.xml` Ce fichier définit les instances de composants pour le bloc HMI et leurs relations avec des registres de RTU.

La Figure 3 présente la fonction des fichiers de configuration XML :

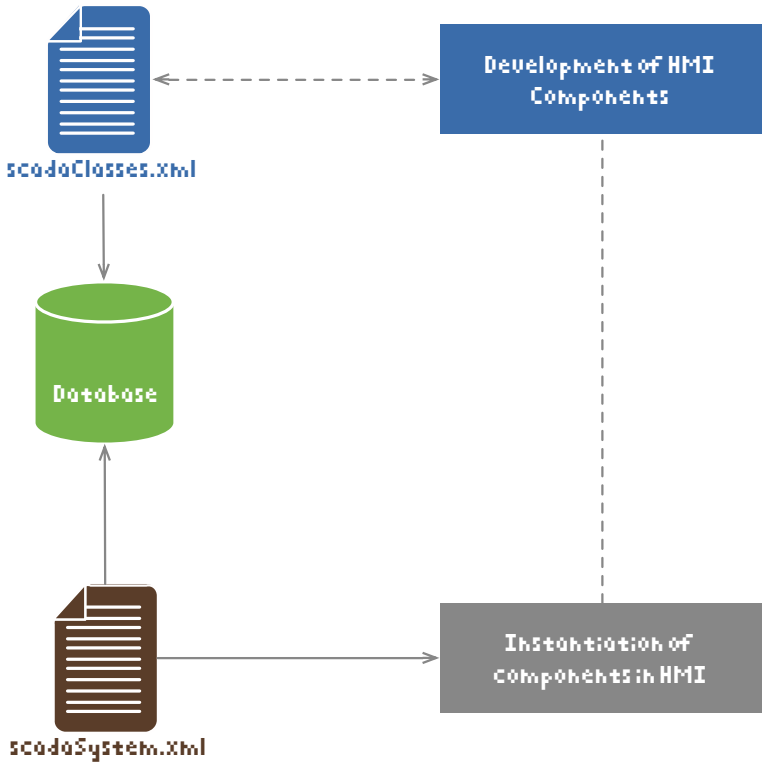


Figure 3: Fonction des fichiers de configuration XML

La structure de la base de données est fixe. En plus du contenu des fichiers XML, elle stocke également toutes les valeurs historiques.

Définition des classes de composants `scadaClasses.xml`

Un type de composant HMI est modélisé par un élément XML `<nodeType>`.

Un `<nodeType>` dispose d'un modèle de données standard dont les éléments (`<point>`) sont classés en quatre catégories (voir Tableau 1). Un `<point>` correspond à une grandeur lue ou écrite sur le RTU.

Catégorie de `<points>`	Définition	Traitement
`<parameter>`	Un `<point>` de type `<parameter>` est un paramètre de configuration ou une consigne pour un régulateur local.	Le HMI permet de mettre à jour les paramètres. Le serveur SCADA met à jour le RTU et logue le changement dans la base de données.
`<measValue>`	Un `<point>` de type `<measValue>` représente une valeur analogique mesurée.	Le serveur SCADA met à jour le HMI avec les nouvelles valeurs et stocke toutes les valeurs dans la base de données en y rajoutant un *time stamp*.
`<signal>`	Un `<point>` de type `<signal>` représente l'occurrence d'un événement.	Le serveur SCADA transmet au HMI l'occurrence des événements et stocke tous les événements dans la base de données en y rajoutant un *time stamp*.
`<alarm>`	Une alarme `<alarm>` est un type particulier de `<signal>` qui nécessite une quittance d'un opérateur.	Le serveur SCADA transmet au HMI l'occurrence des alarmes et stocke toutes les alarmes dans la base de données en y rajoutant un *time stamp*. Le HMI permet de quittancer les alarmes. Les quittances sont stockées dans la base de données.

Tableau 1: Catégorie de **`<points>`**

Définition d'un système SCADA `scadaSystem.xml`

Le fichier `scadaSystem.xml` contient :

- la définition des instances des composants HMI, et
- le lien entre chaque `<point>` d'un composant HMI et un registre du RTU.

Les classes de composants HMI étant appelés `<nodeType>`, les instances de ces composants sont naturellement des `<node>`. Un exemple de définition d'un `<node>` avec la référence à son type est présenté ci-dessous :

```
<node id="G1_1"  name="Solar-A"  nodeTypeRef="N1"/>
```

Figure 4: Définition d'un élément `<node>` se référant à un `<nodeType>` spécifique

Dès lors, un `<point>` particulier est identifié par :

- l'identifiant d'un`<node>` (par exemple « `G1_1` »), et
- l'identifiant d'un `<point>` dans la définition du `<nodeType>`.

Pour accéder à des registres de RTU, il s'agit de définir :

- un protocole (`modbus` ou autre), et
- des modalités d'accès aux registres spécifiques au protocole utilisé,

Les registres sont classés en 3 catégories :

- `<inputs>` : Les registres de cette catégories doivent être lus et leurs contenus stockés dans la base de données et envoyés vers les HMIs connectés.
- `<outputs>` : Lorsqu'un HMI demande l'écriture du `<point>` associé à un `<output>`, le serveur SCADA écrit le registre et logue l'opération dans la base de données.
- `<triggers>` : Les `<triggers>` se distinguent des `<inputs>` par le fait que seuls des changements de valeurs entre deux lectures successives .

Le traitement sur les registres effectué par le serveur SCADA est illustré par la Figure 5.

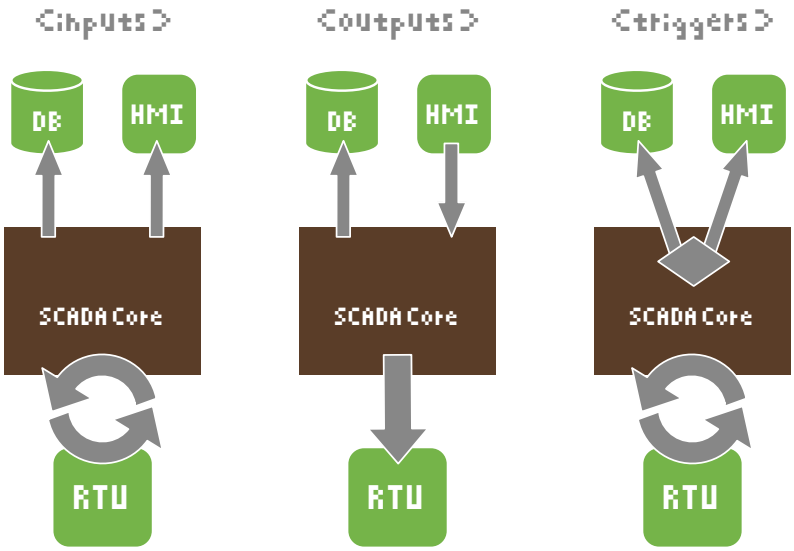


Figure 5: Traitement des registres RTU par le serveur SCADA

Base de données

La base de données, dont la structure est indépendante du système SCADA, contient les tables présentées dans la Figure 6.

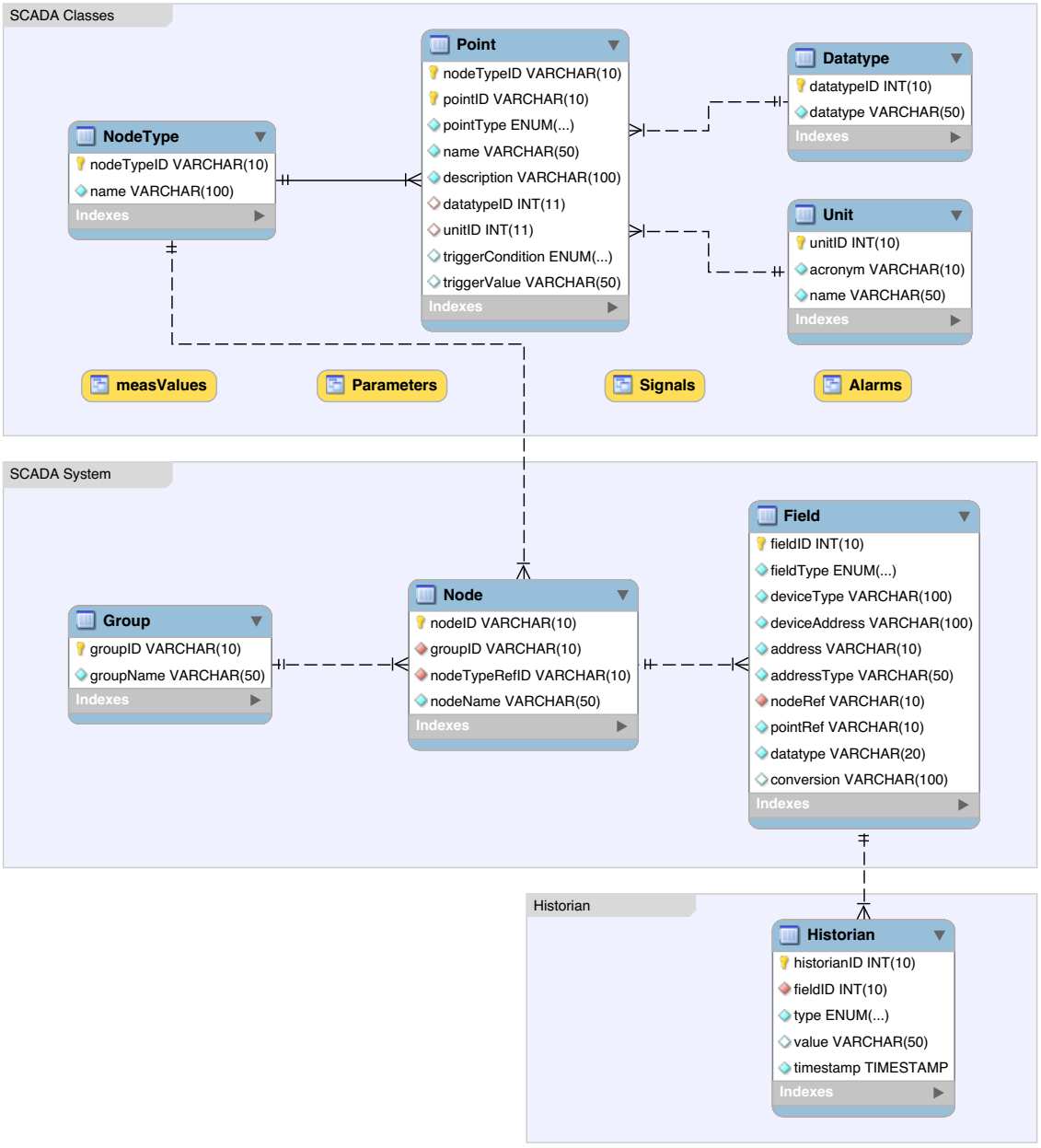


Figure 6: Schéma relationnel de la base de données

La Figure 6 regroupe les tables dans trois groupes :

- **SCADA Classes** : Stockage de la structure des composants HMI (`<nodeType>`). Des vues (*views*) particulières permettent d'obtenir la liste des points par catégorie (Parameters, Meas values, Signals, Alarms).
- **SCADA System :** Stockage de la topologie d'un système SCADA particulier : instances de composants HMI, lien entre points et registres.
- **Historian** : L'unique table de ce groupe stocke toutes les valeurs instantanées mémorisées.

Le contenu des fichiers `scadaClasses.xml` et `scadaSystem.xml` sert à remplir les tables des groupes « SCADA Classes », respectivement « SCADA System », selon le principe illustré par la Figure 7.

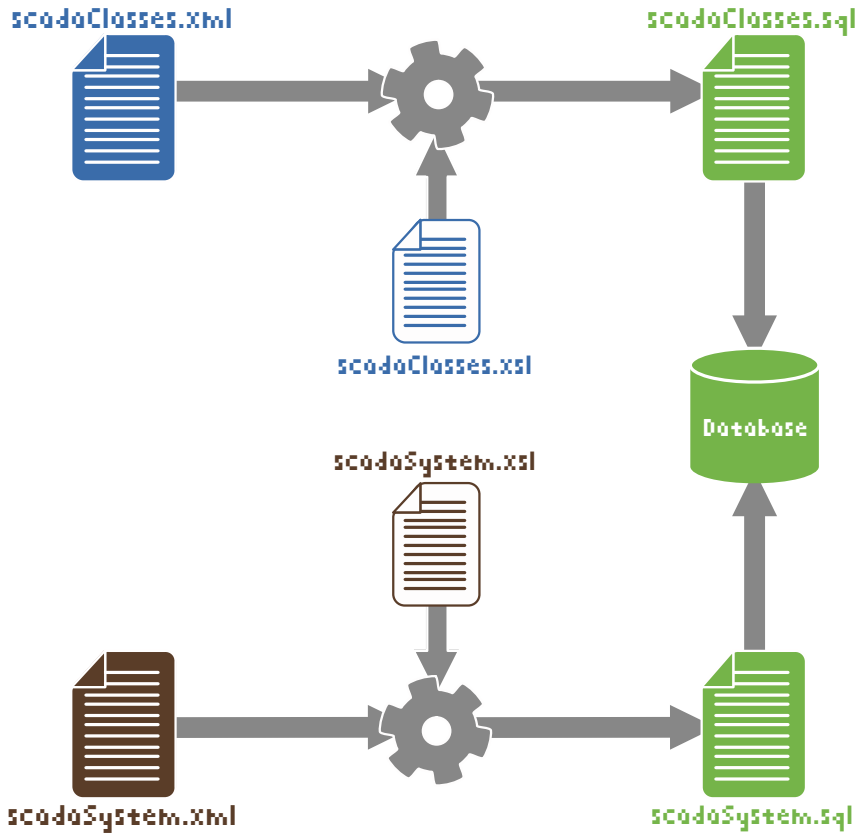


Figure 7: De XML à SQL

Tâche 1 : Base de données

Création de la base de données et insertion des données statiques.

1.a *Appliquez la transformation `scadaClasses.xsl` au fichier `scadaClasses.xml` pour obtenir le fichier de sortie `scadaClasses.sql`.*

1.b *Exécutez le script `scadaClasses.sql`.*
Relevez le schéma relationnel obtenu et le contenu des tables.

1.c *Définissez les commandes SQL pour remplir les tables du groupe SCADA System. Développez ensuite la transformation XSLT `scadaSystem.xsl` qui génère la suite de commandes SQL dans le script `scadaSystem.sql`.*

1.d *Exécutez le script `scadaSystem.sql`. Relevez le contenu des tables mises à jour.*

Serveur SCADA

Principe

La Figure 8 présente une vue logique du fonctionnement du serveur SCADA.

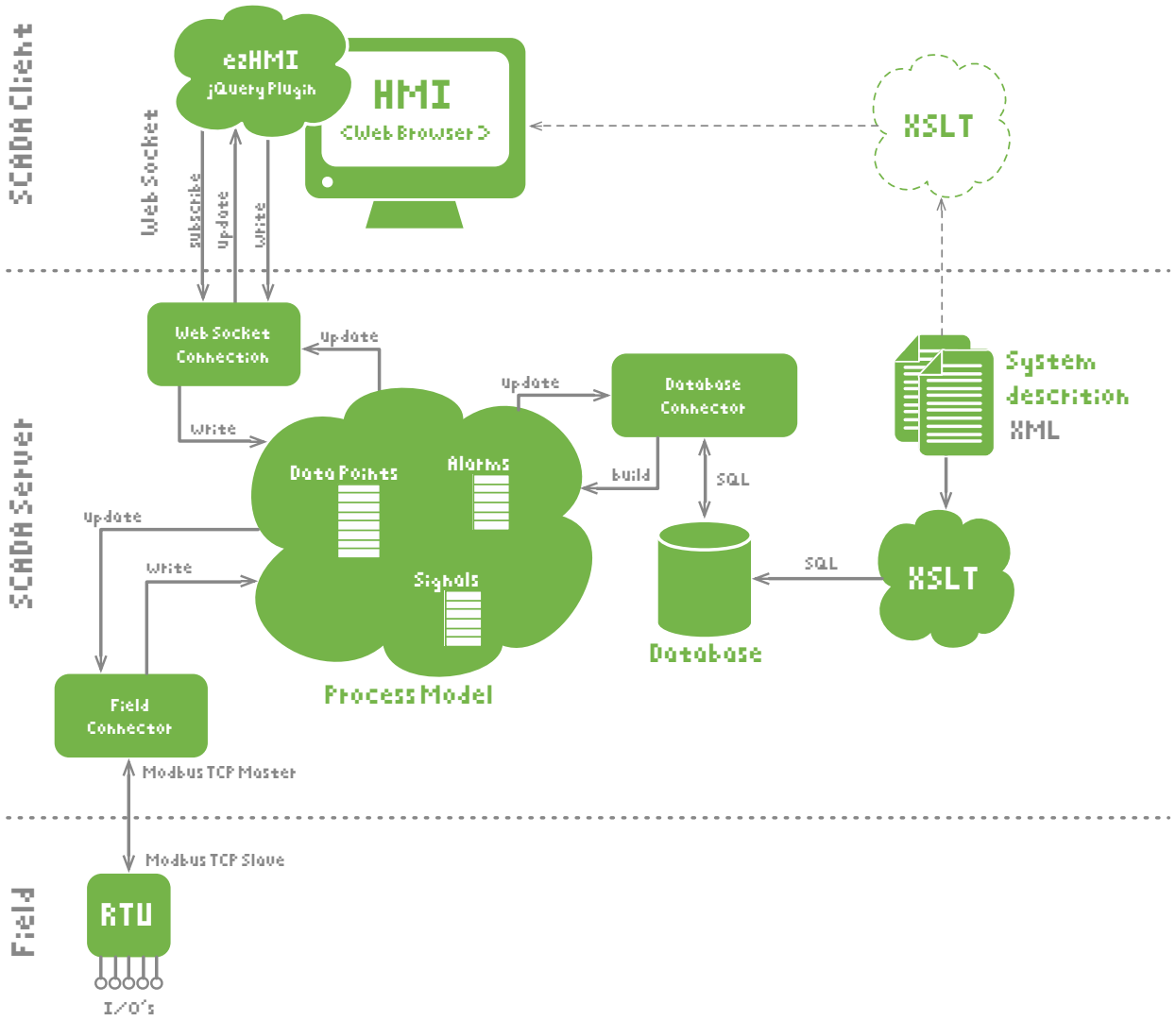


Figure 8: Vue logique du serveur SCADA

Fonctionnement

Cette section décrit en termes généraux le fonctionnement du serveur SCADA. Toutes les déclarations se rapportent à la Figure 8.

« Process Model »

L'élément central est le modèle du processus (*Process Model*). Il contient essentiellement trois listes: une liste de données, une liste de signaux et une liste d'alarmes. D'autres éléments peuvent également souscrire un abonnement dans ce modèle. Ainsi toutes les modifications faites dans le modèle pourront être notifiées (*update*). En outre, les données dans le modèle peuvent être modifiées (*write*).

« Database Connector »

Au démarrage du serveur SCADA, le modèle de processus est généré (*build*) à l'aide du Connecteur de Base de données (*Database Connector*) qui trouvera les informations nécessaires dans la base de données.

Au démarrage le modèle de processus ne contient aucune donnée en terme de valeurs mesurées, de signaux ou d'alarmes. Les données sont enregistrées uniquement pendant le fonctionnement du serveur. On notera encore que seul un changement dans les données est "logué" dans la base de données (*update*). Le modèle ne contient donc que les données actuelles, pas de données historiques. Le fait d'enregistrer toute nouvelle donnée est appelé fonction "Historian".

« Field Connector »

Le "*Field Connector*" est connecté au modèle de processus dans lequel il a d'ailleurs également souscrit un abonnement (*subscribe*). Il peut changer les données dans le modèle (*write*) en communiquant les changements via le protocole ModBus TCP. Il peut également modifier les valeurs dans la couche de terrain, si les modifications désirées lui sont transmises par le modèle (*update*).

« WebSocket Connector »

Des changements de données dans la couche terrain peuvent être demandés par les clients SCADA. Ces changements montrent non seulement ce qui se passe dans la couche terrain, mais peuvent également influencer les éléments du terrain.

La liaison entre les clients SCADA et le serveur est assurée par le "*WebSocket Connector*". C'est au travers de ce connecteur, qui s'est également abonné au modèle, que communiquent les clients SCADA. Ainsi, tout changement de données dans le modèle sera notifié au connecteur et tous les clients SCADA seront automatiquement mis à jour (*update*). Si un client souhaite faire des modifications, le connecteur transmettra les nouvelles valeurs aux données concernées du modèle (*write*).

De plus amples informations sur chacun des éléments ci-dessus seront donnés dans la suite de ce document. Mais pour l'instant, nous allons décrire la structure générale de la couche "serveur SCADA" (*SCADA Server*).

Structure générale de la couche "SCADA Server"

Le serveur est divisé en quatre parties différentes, chacune ayant un objectif spécifique. La figure ci-dessous illustre ces quatre parties.

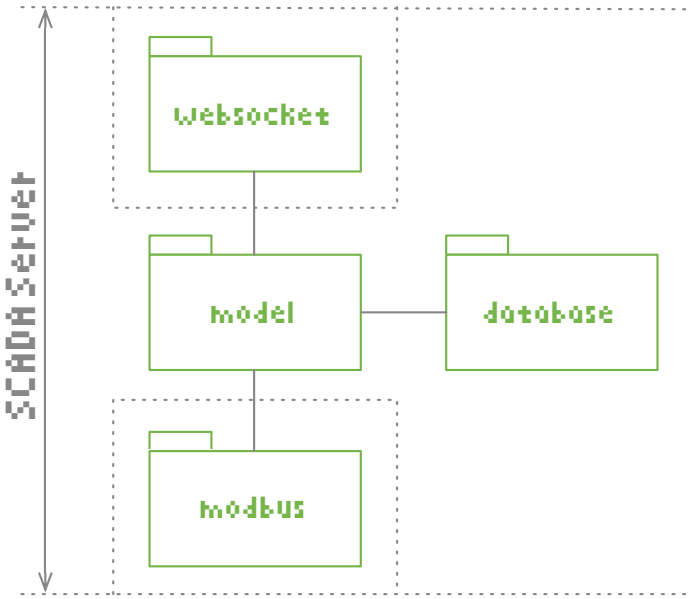


Figure 9: Architecture générale de la couche "SCADA Server"

- **WebSocket**: Cette partie fournit l'interface pour les clients SCADA. Vous devez la concevoir et l'implémenter entièrement.
- **Model:** Cette partie décrit le modèle du processus « SCADA Server » et contient tous les éléments requis pour son fonctionnement. Elle sera entièrement fournie.
- **ModBus:** Ce paquet présente l'interface vers la couche terrain. Il sera entièrement fourni également.
- **Database:** C'est l'interface vers la base de données. Elle contient tous les éléments qui sont nécessaires à la génération du modèle de processus ainsi que l' « historian ». On vous donne sa structure et vous devrez l'implémenter.

Le paquet "Model"

Le "*Model Package*" est responsable de fournir les fonctionnalités et les types de données pour le modèle de processus (*Process Model*). Il contient également la fonctionnalité qui permet aux éléments extérieurs de s'abonner pour être informés des changements qui surviennent à l'intérieur du "*Process Model*". La Figure 10 montre un aperçu des classes de ce package.

On y trouve deux classes importantes:

- `DataPoint`
- `Eventlist`

Dans le "*Process Model*" deux types de points de données (`DataPoint`) existent : les points de données binaires (`BooleanDataPoint`) et les points de données analogiques (`FloatDataPoint`). Chaque point de données contient une liste d'abonnés

(`DataPointListener`), qui permet d'en informer toute modification de valeur. Cette interface doit être implémentée pour toutes les parties concernées.

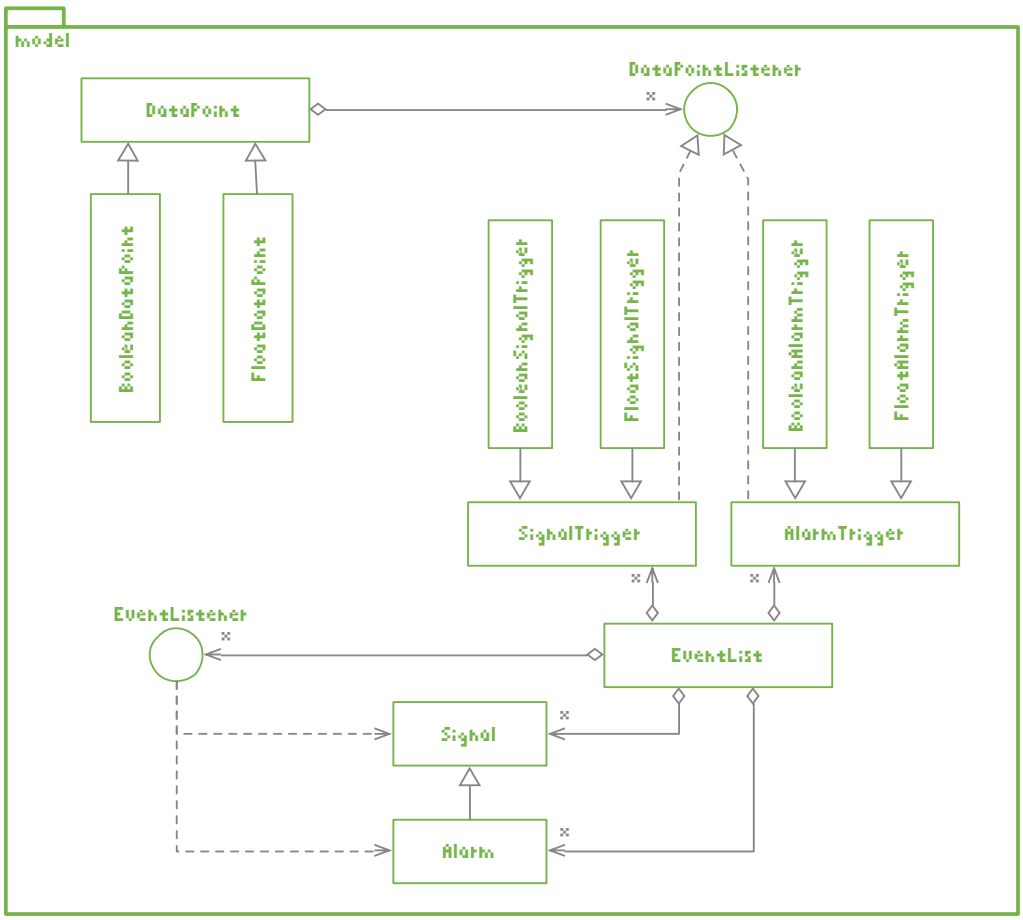


Figure 10: Vue générale du "*Model*"

La liste d'événements (`EventList`) contient essentiellement deux catégories d'éléments:

- Triggers
- Evénements

Les "Triggers" (`SignalTrigger`, `AlarmTrigger`) s'abonnent aux points de données. Ainsi, un signal ou une alarme va être émis à chaque fois qu'un point de données va franchir une valeur de seuil.

Les signaux (`Signal`) et alarmes (`Alarm`) sont également insérés dans la liste des événements.

En outre, la liste des événements offre toujours une interface qui permet d'être informé des nouveaux événements. Vous allez devoir implémenter cette interface (`EventListener`).

Au démarrage du serveur SCADA, tous les points de données sont extraits de la base de données en utilisant le connecteur de base de données. Ceci est également le cas pour l'ensemble des "triggers". Toutes les informations nécessaires sont stockées dans la base de données et sont lues en utilisant le connecteur de base de données.

Ce paquet a déjà été entièrement implémenté. Le code approprié et la documentation correspondante (diagrammes de classes, etc.) sont à disposition sur Cyberlearn.

Le paquet « Database »

Le paquet "*Database*" établit la connexion entre le modèle de processus et la base de données. Dans la Figure 11, ce service se nomme "*Database Connector*". Comme on peut le voir, il existe trois principales actions :

- Le modèle de processus écrit des données historiques dans la base de données. (*Data Historian Function*)
- Le modèle de processus est construit au moyen des informations contenues dans la base de données (*Builder Function*)
- Des événements et des alertes ainsi que le moment de leur arrivée sont enregistrés dans la base de données (*Event Historian Function*).

La figure ci-dessous montre le paquet *Database*.

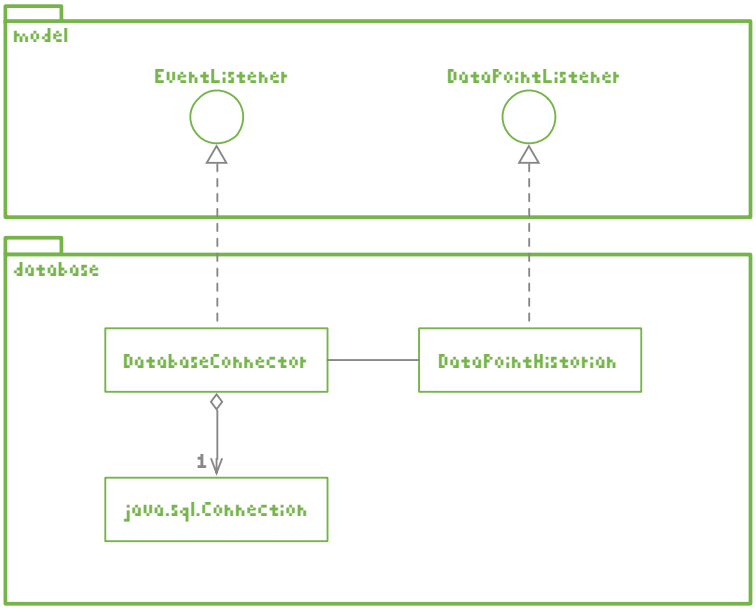


Figure 11: Architecture du paquet "*Database*"

Le Connecteur de base de données (`DatabaseConnector`) a les tâches suivantes:

- Il crée la connexion à la base de données.
- Au démarrage du serveur SCADA, il crée le modèle de processus, l'ensemble des points de données ainsi que la liste des événements. Dans cette dernière, il enregistre déjà tous les triggers d'événements et d'alertes (*Builder Function*).
- Il crée aussi un observateur d'événements qui sera averti de tous les événements Ainsi, il doit aussi les enregistrer dans la base de données (*Event Historian Function*).

L'"Historian" (`DataPointHistorian`) est inscrit comme observateur des points de données (`DataPointListener`). Il sera par conséquent averti de tout changement des points de données, changement qu'il enregistrera avec son *time stamp* dans la base de données (D*ata Historian Function*).

Ce paquetage est partiellement mis à votre disposition. Vous trouvez le code squelette (code des classes sans implémentation) ainsi que la documentation correspondante (diagrammes de classes etc.) sur Cyberlearn.

Tâche 2 : Paquet "Database"

Dans cette tâche vous devez complétez le code des classes du paquet Database en vous basant sur les diagramees de classe et le squelette qui sont mis à votre disposition.

Les étapes sont les suivantes:

2.a *Développez, testez et documentez le code Java de la classe* `DatabaseConnector`.

2.b *Développez, testez et documentez le code Java de la classe* `DataPointHistorian`.

Le paquet "ModBus"

ModBus TCP est le standard de communication utilisé pour les échanges entre la couche terrain et la couche serveur SCADA.

L'interface entre la couche serveur SCADA et la couche de terrain est fournie par le paquet ModBus. La Figure 12 en est une illustration:

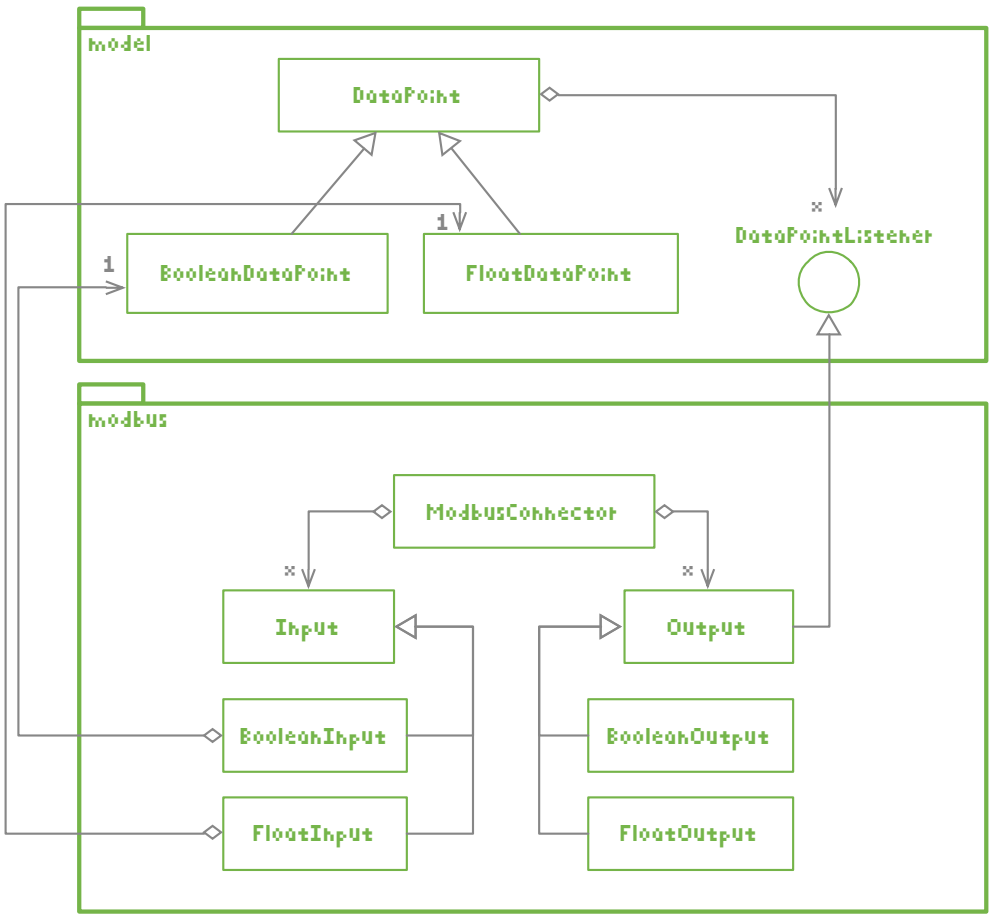


Figure 12: Aperçu du paquet ModBus

La communication ModBus TCP est basée sur le principe question-réponse. En outre, toutes les informations échangées par ModBus TCP sont validées.

Tous les appareils de la couche terrain (Field) ont des entrées/sorties digitales ou analogiques qui sont toutes listées dans la RTU.

Le connecteur ModBus (`ModBusConnector`) fournit les mécanismes de communications avec la RTU. Il a également une liste contenant toutes les entrées digitales ou analogiques (`BooleanInput`, `FloatInput`) et toutes les sorties (`BooleanOutput`, `FloatOutput`). Celles-ci représentent les entrées et sorties physiques de la couche terrain.

Les entrées sont directement liées aux points de données du modèle (`BooleanDataPoint`, `FloatDataPoint`) et, ainsi, tout changement d'état pourra être reflété dans le Process Model.

Les sorties s'inscrivent dans le *Process Model* (`DataPointListener`). En conséquences, chaque changement d'état sera notifié et les sorties de la couche terrain pourront être directement synchronisées.

Ce package a déjà été entièrement implémenté. Le code approprié et la documentation correspondante (diagrammes de classes, etc.) sont à disposition sur Cyberlearn.

Le paquet "WebSocket"

Comme vous pouvez le voir sur les figures 1 & 2, les couches *SCADA Client* et *SCADA Server* communiquent entre-elles via TCP/IP à l'aide de Web Sockets.

La couche client est une Interface Homme Machine (HMI). C'est une page web écrite en HTML5 et utilisant CSS3 et jQuery. Elle peut être déployée sur un serveur web ou stockée dans un fichier local.

La conception de la fenêtre du HMI SCADA est assez simple. Elle suit le principe présenté dans la Figure 13.

L'HMI SCADA offre trois fonctions:

- Affichage de la couche terrain
- Contrôle de la couche terrain
- Affichage des événements et des alarmes de la couche *Process Model*.

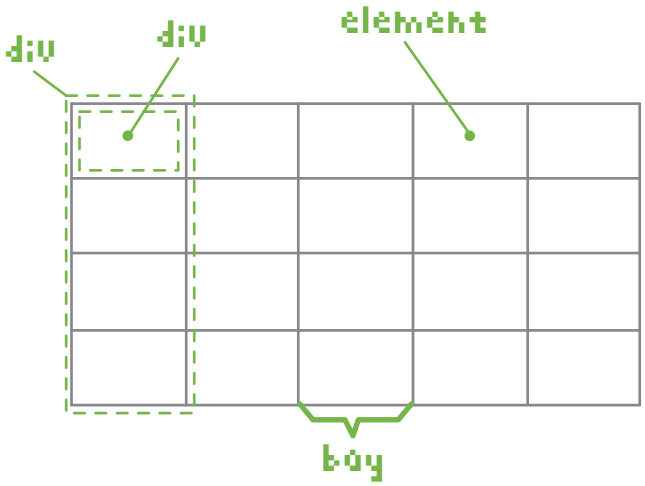


Figure 13: Organisation de la fenêtre du navigateur

L'affichage et le contrôle d'éléments de la couche terrain nécessitent une communication bidirectionnelle entre les couches serveur et client SCADA. Cette communication est assurée par les Web Sockets. Deux des grands avantages de cette technologie sont la simplicité de mise en œuvre et le fait qu'elle est supportée par les navigateurs internet actuels. Un protocole spécifique de communication, appelé EzHMI, a été développé pour la communication entre ces deux couches.

L'interface entre les couches serveur et client SCADA est assurée par le *WebSocket Package* de la couche serveur SCADA. La Figure 14 en montre une structure possible:

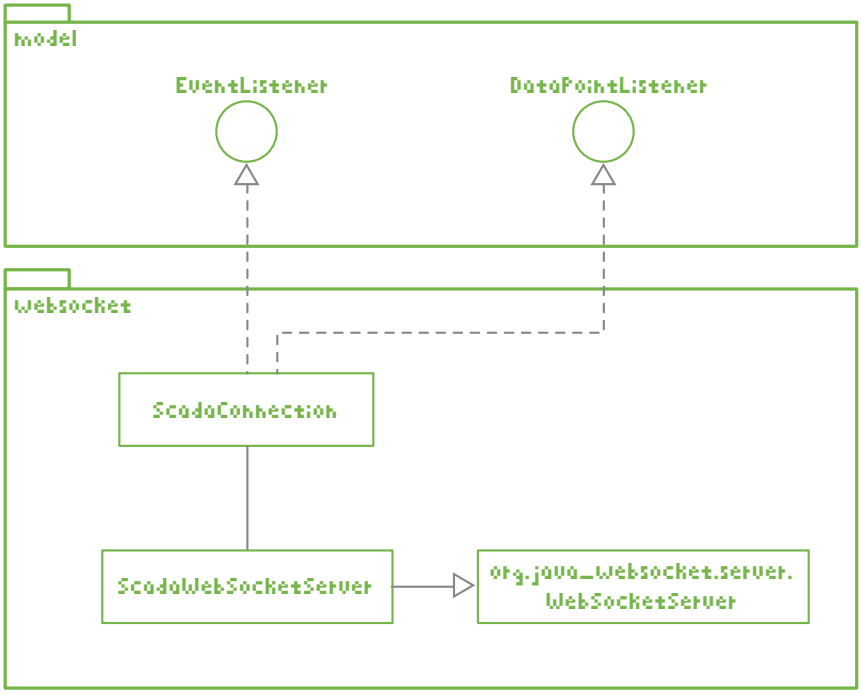


Figure 14: Structure possible pour le paquet "Web Socket"

Le Web Socket Server (`ScadaWebSocketServer`) est une implémentation server standard qui peut accepter des demandes de connexion émanant de clients SCADA. Si une telle connexion est acceptée, les scenarii suivants sont possibles:

- **Scenario I:** Les valeurs de tous les points de données et des événements (modèle complet) sont transmises à l'HMI SCADA afin d'afficher l'état actuel de la couche terrain.
- **Scenario II:** Le WebSocket Connector de la Figure 8 (`ScadaConnection`) s'inscrit dans les points de données du Process Model (`DataPointListener`) et sera ainsi notifié lors de tout changement de données. Pour cela on utilise les web Sockets et le protocole EzHMI.
- **Scenario III:** Le *WebSocket Connector* de la Figure 8 (`ScadaConnection`) s'inscrit dans la liste des événements du *Process Model* (`EventListener`) et recevra ainsi tous les événements et les alarmes. On utilise également les web Sockets et le protocole EzHMI.
- **Scenario IV:** 'HMI SCADA change l'état des sorties de la couche terrain. Ces demandes de modification sont transmises via le protocole EzHMI par le WebSocket Connector (`ScadaConnection`) et transmises par ce dernier aux points de données du *Process Model*.

- **Scenario V:** L'HMI SCADA quittance une alarme. cette demande de quittance est transmise via le protocole EzHMI par le WebSocket Connector (`ScadaConnection`) et transmise par ce dernier à la liste des alarmes du *Process Model*.

Vous ne recevrez pas d'autres informations pour ce paquet que vous allez devoir implémenter entièrement.

Quelques informations supplémentaires, en particulier la définition complète du protocole EzHMI, sont à disposition sur Cyberlearn.

Tâche 3 : Paquet « WebSocket »

Développez le contenu du paquet WebSocket complètement en suivant les étapes présentées ci-dessous:

3.a *Proposez pour chaque scénario un diagramme de séquences pertinent.*

3.b *Elaborez un diagramme de classe complet pour le paquet WebSocket.*

3.b *Développez en Java l'entier du contenu du paquet WebSocket.*

3.c *Testez les scenarii présentés ci-dessus et établissez un rapport de test.*

Planification

Agenda

Pour ce SuperLab, on vous propose la planification suivante :

	Lundi		Mardi	Mercredi
Semaine 1	Introduction	Installation & travail (XSLT)	Travail	Travail
Semaine 2	Travail	Travail	Travail	Travail
Semaine 3	Vacances de Pâques			
Semaine 4	Travail	Fin du travail	Présentation Groupes 1 à 4	Présentation Groupes 5 à 8

Tableau 2: Planification du SuperLab

Il y aura donc 9 blocs de 2 heures de travail disponibles.

Vous avez 10 minutes à disposition pour la présentation finale. 5 minutes sont réservées aux questions des experts.

Tâche 4 : Présentation

Présentez votre solution et expliquez ce que vous avez fait, pourquoi vous l'avez fait et comment vous l'avez fait.

4.a *Une explication de votre solution.*

4.b *Une démonstration qui prouve que le système fonctionne.*

4.c *Dites ce que vous n'avez pas pu faire et pourquoi*

ezHMI Web Socket Protocol
Specification

Introduction

The communication between the SCADA server and all HTML5-based ***ezHMI*** panels is done using the web socket standard. Whereas the SCADA-Server acts as the server and all ***ezHMI*** Panels act as clients. This implies that the server must serve multiple clients at the same time.

And not only that, the server has to ensure that all the clients are in sync. For example if a user acknowledges an alarm on one ***ezHMI*** panel, the change should be reflected on all panels.

The Web socket messages are exchanged using the JSON data format. This simplifies the development on both sides: the client and the server side.

```
{
  "type": "SubscribeRequest",
  "id": "Batt.A.charge"
}
```

Table of content

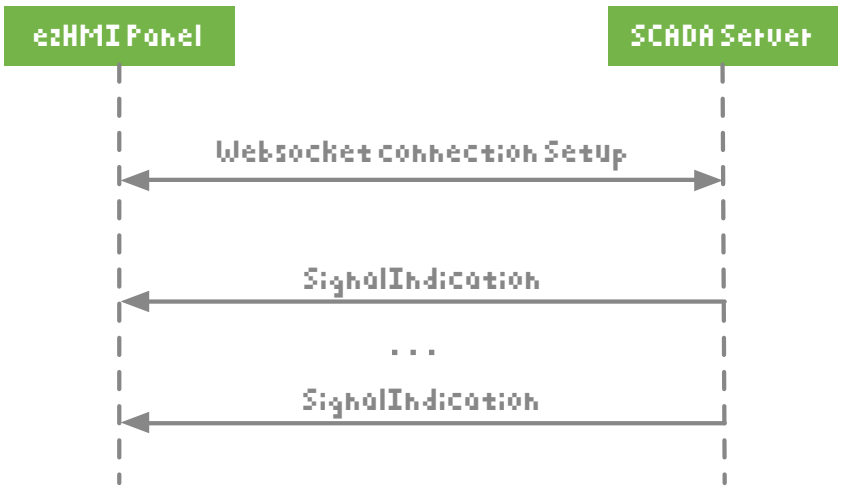
Initial Communication Sequence

The sequence of communication between the ezHMI Panel and the SCADA server is completely asynchronous. There are no constraints in which order messages have to be send nor do request require the peer to hold back other messages before responding to the request.

Despite this, there are some still some things to consider in the initial communication sequence. This chapter will list the three types of activities that either the SCADA Server or the ezHMI Panel have to start after a successful connection setup.

Initial SignalIndications

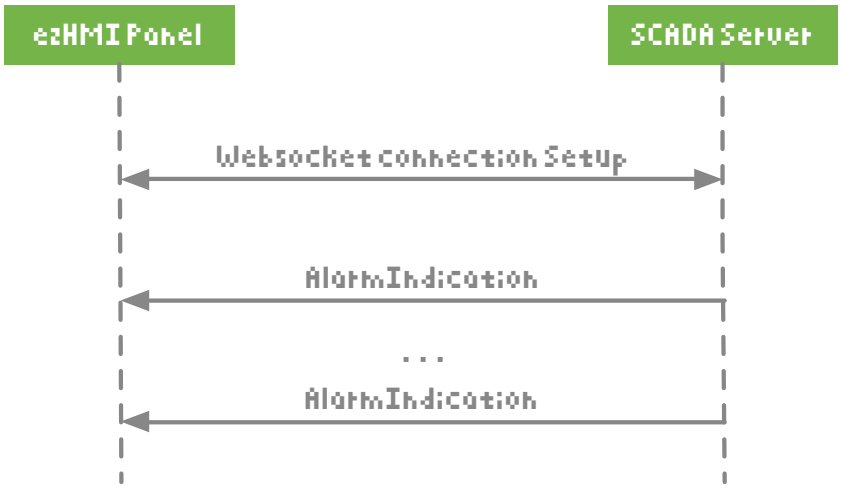
Once the Web Socket handshake is done, the SCADA Server should send all present signals to the client using the ***SignalIndication*** message.



Note that for simplicity, each signal is sent using its own Web Socket message.

Initial AlarmIndications

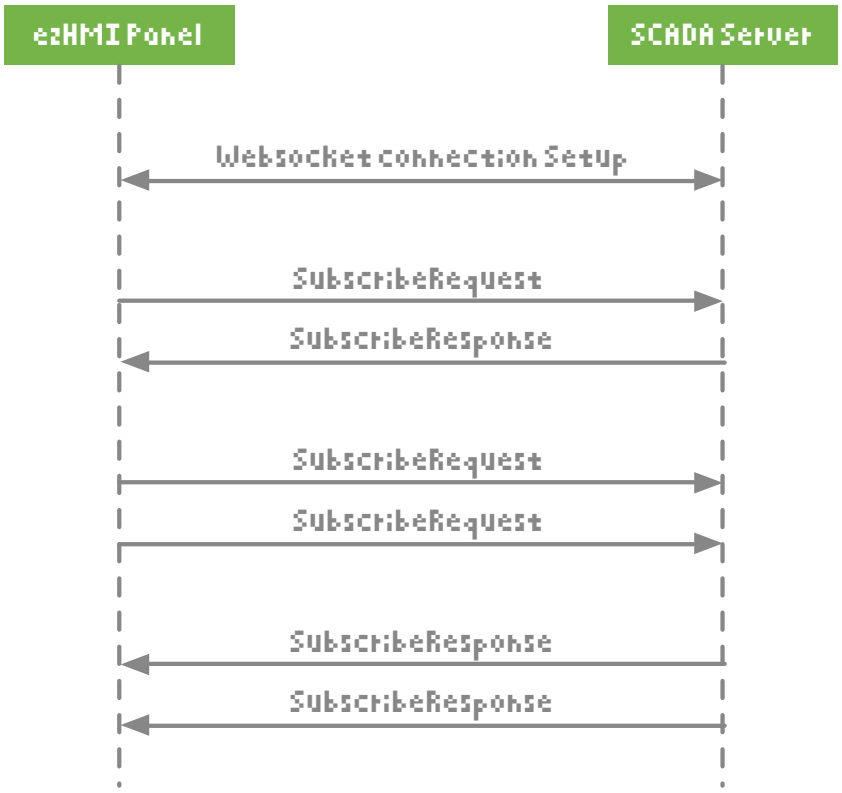
The same applies to Alarms. After the Web Socket handshake, all present Alarms should be sent to the ezHMI Panel using the ***AlarmIndication*** message.



Analogue to the Signals, each Alarm is sent using its own Web Socket message.

Subscriptions

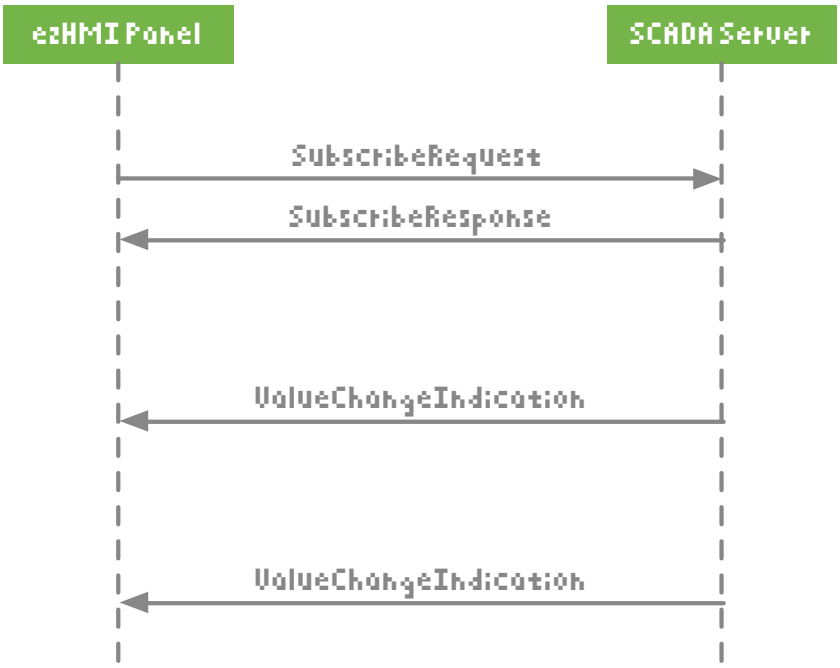
Once the connection setup is finished, the ezHMI Panel can send request to subscribe to changes of a given data point. The ***SubscribeRequest*** message will be used for that task.



As you see, the client does not need to wait for a response. The Panel can immediately send another request. This must not even be a SubscribeRequest as shown in the sequence diagram above; it can be any other message.

Messages

Subscribing to data point changes



The ezHMI Panel can send at any time a ***SubscribeRequest*** to the SCADA Server in order to subscribe to value changes of a given data point. The SCADA Server has to respond with an ***SubscribeResponse*** message and from this moment on, each change of the given data point triggers a ***ValueChangeIndication*** message from the SCADA Server to the ezHMI Panel.

SubscribeRequest

This message is send from the ezHMI Panel to the SCADA Server in order to subscribe to the changes of a given data point.

Field	Description/Value
type	**"SubscribeRequest"**
id	ID of the data point to subscribe to.

Example:

```
{
  "type": "SubscribeRequest",
  "id": "Batt.A.charge"
}
```

SubscribeResponse

Send from the SCADA Server to the ezHMI Panel in response to a SubscribeRequest message.

Field	Description/Value
type	**"SubscribeResponse"**
id	ID of the data point of the subscription.
status	Status of the operation: **"OK"** if the ezHMI Panel could be subscribed to the given data point, a descriptive error string otherwise.
value	The actual value of the data point. Valid formats are: - Integer numbers like "42". - Floating point numbers like "1.23" - Booleans like "true" or "false"

Example:

```
{
  "type": "SubscribeResponse",
  "id": "Batt.A.charge",
  "status": "OK",
  "value": "0.75"
}
```

ValueChangeIndication

Send from the SCADA Server to the ezHMI Panel each time a data point to which the ezHMI Panel is subscibed has changed.

Field	Description/Value
type	**"ValueChangeIndication"**
id	ID of the data point that has changed its value.
value	The actual value of the data point. Valid formats are: - Integer numbers like "42". - Floating point numbers like "1.23" - Booleans like "true" or "false"

Example:

```
{
  "type": "ValueChangeIndication",
  "id": "Batt.A.charge",
  "value": "0.4"
}
```

Reading value of a data point

The ezHMI Panel might need to read the value of a certain data point only once. So it would not make sense to subscribe to changes on that data point. The ezHMI Panel can request the actual value of the data point using the ***GetRequest*** message and the SCADA Server has to respond with an ***GetResponse*** containing the actual value of the data point.

GetRequest

This message is send from the ezHMI Panel to the SCADA Server in order to get the actual value of a given data point.

Field	Description/Value
type	**"GetRequest"**
id	ID of the data point to get the actual value from.
serial	Request serial number, used in the response to identify the request.

Example:

```
{
  "type": "GetRequest",
  "id": "Batt.A.charge",
  "serial": "384673"
}
```

GetResponse

This message is send from the SCADA Server to the Panel in response to a ***GetRequest***.

Field	Description/Value
type	**"GetResponse"**
id	ID of the data point.
serial	Serial number of the request.
value	The actual value of the data point. Valid formats are: - Integer numbers like "42". - Floating point numbers like "1.23" - Booleans like "true" or "false"
status	Status of the operation: **"OK"** if the value of the data point could be read, a descriptive error string otherwise.

Example:

```
{
  "type": "GetResponse",
  "id": "Batt.A.charge",
  "serial": "384673",
  "value": "0.367",
  "status": "OK"
}
```

Writing a value to a data point

In order to change a output data point, an ezHMI Panel will send the ***SetRequest*** to the SCADA Server. The SCADA Server will try to change the output value of the data point and will report status back to the Panel using the ***SetResponse*** message.

SetRequest

This message is send from the ezHMI Panel to the SCADA Server in order to change the actual value of a given output data point.

Field	Description/Value
type	**"SetRequest"**
id	ID of the data point to change the value.
value	The value to change the data point to. Valid formats are: - Integer numbers like "42". - Floating point numbers like "1.23" - Booleans like "true" or "false"
serial	Request serial number, used in the response to identify the request.

Example:

```
{
  "type": "GetRequest",
  "id": "Batt.A.control",
  "value": "true",
  "serial": "384673"
}
```

SetResponse

This message is send from the SCADA Server to the Panel in response to a ***SetRequest***.

Field	Description/Value
type	**"SetResponse"**
id	ID of the data point.
serial	Serial number of the request.
status	Status of the operation: **"OK"** if the value of the data point could be set, a descriptive error string otherwise.

Example:

```
{
  "type": "SetResponse",
  "id": "Batt.A.control",
  "serial": 384673,
  "status": "OK"
}
```

Signals

Signals are uncritical events. The SCADA Server has to send these signals to all ezHMI Panels when they arise on the Server and additionally it has to send all existing to a Panel at connection setup. Signals are not acknowledged by the ezHMI Panel.

SignalIndication

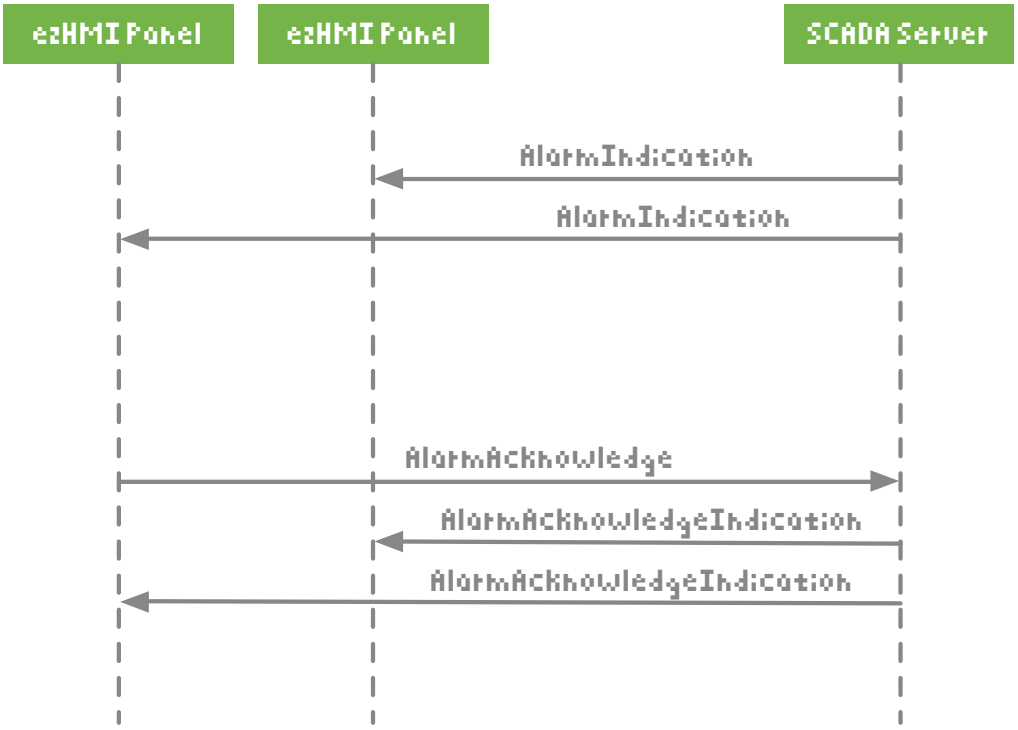
Send from the SCADA Server to the ezHMI Panel each time a new signal is raised and just after connection setup.

Field	Description/Value
type	**"SignalIndication"**
id	ID of the signal (Each signal has a unique ID).
text	Text describing the signal.
raised	Timestamp when the signal was raised in milliseconds since epoch.

Example:

```
{
  "type": "SignalIndication",
  "id": "134",
  “text”: “No beer left in cave!”,
  “raised”: 1395313823375
}
```

Alarms



Alarms are similar to Signals, but they are critital and they have to be acknowledged. The SCADA Server has to send these alarms to all ezHMI Panels when they arise on the Server and additionally it has to send all existing to a Panel at connection setup. The message type is ***AlarmIndication***.

If the operator acknowledges an alarm on an ezHMI Panel, the Panel sends an ***AlarmAcknowledge*** message to the server. The server has to send an ***AlarmAcknoledgementIndication*** to all Panels in order they update their alarm list.

Alarmindication

Send from the SCADA Server to the ezHMI Panel each time a new alarm is raised and just after connection setup.

Field	Description/Value
type	**"AlarmIndication"**
id	ID of the alarm (Each alarm has a unique ID).
text	Text describing the signal.
active	"true" if the alarm is not already acknowledged, "false" otherwise.
raised	Timestamp when the alarm was raised in milliseconds since epoch.
acknowledged	Timestamp when the alarm was acknowledged in milliseconds since epoch.

Example:

```
{
  "type": "AlarmIndication",
  "id": "1",
  "text": "Wine is out too!",
  "active": "true",
  "raised": 1395315823374,
  "acknowledged": 0
}
```

AlarmAcknowledge

Send from a ezHMI Panel to the SCADA Server in order to acknowledge an alarm.

Field	Description/Value
type	**"AlarmAcknowledge"**
id	ID of the alarm (Each alarm has a unique ID).

Example:

```
{
  "type": "AlarmAcknowledge",
  "id": "1"
}
```

AlarmAcknowledgeIndication

Send from the SCADA Server to each ezHMI Panel when an alarm has been acknowledged.

Field	Description/Value
type	**"AlarmAcknowledgeIndication"**
id	ID of the alarm (Each alarm has a unique ID).
acknowledged	Timestamp when the alarm was acknowledged in milliseconds since epoch.

Example:

```
{
  "type": "AlarmAcknowledgeIndication",
  "id": "1",
  "acknowledged": 1395315823374
}
```

Ernst-Abbe-Fachhochschule Jena
Hochschule für angewandte Wissenschaften

Hes·so VALAIS WALLIS

Anhang C

Installation ein MySQL-Server auf Linux

Diplomarbeit
Gasdetektor
Installation ein MySQL-Server auf Linux

Mayencourt Louis

10. Juli 2014

1 Einleitung

Dieses Dokument beschreibt die Schritte durchführen, um einen MySql Linux-Server zu installieren. Dieser Server wird zum Speichern von Datenbank erforderlich, um die SCADA-Server laufen verwendet.

2 Serveur MySql

Der erste Schritt ist, um MySQL-Server zu installieren. Dazu führen Sie die folgende Befehlszeile :

```
1 sudo apt-get install mysql-server mysql-client
```

Während der Installation werden Sie aufgefordert, ein Administratorkennwort zu bringen. Der Name des Administrator-Konto ist root. Um das Passwort nicht zu vergessen, wird dieselbe Benutzernamen gewählt : root.

3 Erreichbarkeit

Um den Zugriff auf die Datenbank von der Außenseite zu haben, ist es notwendig, den Server zu konfigurieren.
Das erste, was zu tun ist, die lokale Hören löschen mit die folgende Befehlszeile :

```
1 sudo nano /etc/mysql/my.cnf
```

und kommentieren Sie die Zeile :

```
Bind-address = 127.0.0.1 wird
#bind-address = 127.0.0.1
```

Danach wieder starten Sie den Server :

```
sudo /etc/init.d/mysql restart
```

Erstellen Sie nun einen Benutzer, und ordnen Sie eine Datenbank :

```
CREATE USER 'prd'@'%' IDENTIFIED BY '***';
GRANT USAGE ON * . * TO 'prd'@'%' IDENTIFIED BY '***' WITH
        MAX_QUERIES_PER_HOUR 0
MAX_CONNECTIONS_PER_HOUR 0 MAX_UPDATES_PER_HOUR 0
        MAX_USER_CONNECTIONS 0 ;
CREATE DATABASE IF NOT EXISTS 'GazDetector' ;
GRANT ALL PRIVILEGES ON 'GazDetector' . * TO 'prd'@'%';
```

Der `prd` Benutzer hat nun vollen Zugriff auf `GazDetector` Datenbank.

Der letzte Schritt ist, um den Anschluss 3306 für externe Verbindungen öffnen :

```
sudo /Sbin/iptables -A INPUT -i eth0 -p tcp -destination-port 3306
        -j ACCEPT
```

4 Serveur phpMyAdmin

PhpMyAdmin ist eine grafische Oberfläche für die Verwaltung von Datenbanken. Die Installation wird mit der folgenden Kommandozeile :

```
sudo apt-get install phpmyadmin
```

In die Installation Unterstützung, ist es wichtig, den Apache-Server wählen. Die automatische Konfiguration der Datenbank mit `dbconfig-common` akzeptieren.

Man muss dann konfigurieren Sie den Apache-Server mit phpMyAdmin arbeiten :

```
sudo nano /etc/apache2/apache2.conf
```

hinzufügen am Ende der Datei die folgende Zeile :

```
Include /etc/phpmyadmin/apache.conf
```

Drücken Sie STRG + X und Y zu verlassen und speichern. Starten Sie den Server :

```
1 sudo /etc/init.d/apache2 restart
```

5 Überprüfung der Installation

Durch einen Webbrowser, sollten Sie in der Lage, die folgende Adresse zugreifen : `localhost/phpmyadmin`.

Ernst-Abbe-Fachhochschule Jena
Hochschule für angewandte Wissenschaften

Hes·so VALAIS WALLIS

Anhang D

Webseite Code

Σ π ≈ &

Valais excellence CERTIFIED

```
<!DOCTYPE html>
<!--meta charset="utf-8" /-->
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

<title> Gaz Detector HMI </title>

    <!-- 1. Add these JavaScript inclusions in the head of your page -->
    <script type="text/javascript" src="http://code.jquery.com/
        jquery-1.10.1.js"></script>
    <script type="text/javascript" src="http://code.highcharts.com/
        highcharts.js"></script>

    <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.8.2/
        jquery.min.js"></script>
    <script src="http://code.highcharts.com/highcharts.js"></script>


<!-- ____________________________________________________________ -->
 <html>
<body>

<img src="img/hes_so.png" alt="hes-so" width="140" height="30" align="left">
<img src="img/hes_logo.png" alt="hes-logo" width="100" height="30" align=
    "right"><br>


<h2>WebSocket Status :</h2>  <p id="output"></p>

<h1 style="text-align:center;">Gaz Detector Wiever</h1>

        Debugger :
        <a href="HMI_Debug.html">GazDetector Debugger</a>

    <fieldset>
    <legend>Add Graph</legend>
    <input type="hidden" value="0" id="grapheValue" />
    <input type="submit" name="addGraphe" value="add" onclick="addGraphe()" />
    Node :
    <select name="node" id="node">
        <option value="G1_1">G1_1</option>
    </select>
    ROS :
    <select name="ROS" id="ROS">
        <option value="M1">ROS1</option>
        <option value="M2">ROS2</option>
        <option value="M3">ROS3</option>
    </select>
    filter :
    <select name="filter" id="filter">
        <option value="1">no filter</option>
        <option value="2">2 samples</option>
        <option value="5">5 samples</option>
        <option value="10">10 samples</option>
        <option value="20">20 samples</option>
        <option value="40">40 samples</option>
    </select>
    </fieldset>

    <!-- 3. Add the container -->
    <div id="graphContainer"> </div>
```



```
</body>
</html>

<!-- ____________________________________________________________ -->
<script language="javascript" type="text/javascript">
    var wsUri = "ws://localhost:8888";
    var output;
    var RequestSerial = 0;
    // the tab for the subscribe chart
    var chartList = [];

/*****************************************************************************/
/* WebSocket                                                                 */
/*****************************************************************************/

    /**
     * Initialisation function
     *
     * @param       -
     * @return      -
     */
    function init()
    {
        output = document.getElementById("output");
        webSocketBehavior();
    }

    /**
     * function for the behavior of the websocket
     *
     * @param       -
     * @return      -
     */
    function webSocketBehavior()
    {
        websocket = new WebSocket(wsUri);
        websocket.onopen = function(evt)
        {
            onOpen(evt)
        };
        websocket.onclose = function(evt)
        {
            onClose(evt)
        };
        websocket.onmessage = function(evt)
        {
            onMessage(evt)
        };
        websocket.onerror = function(evt)
        {
            onError(evt)
        };
    }

    /**
     * this function append when the websocket open
     *
     * @param       evt : the data coming from the websocket
     * @return      -
```

```
 */
function onOpen(evt)
{
    writeToScreen("CONNECTED");
    addGraphe();
    normalMode();
}

/**
 * this function append when the websocket close
 *
 * @param       evt : the data coming from the websocket
 * @return      -
 */
function onClose(evt)
{
    writeToScreen("DISCONNECTED");
}

/**
 * this function append when the websocket receive a message
 *
 * @param       evt : the data coming from the websocket
 * @return      -
 */
function onMessage(evt)
{
    var request = JSON.parse(evt.data);
    if(request["type"] == "ValueChangeIndication")
    {
        // uptade all graph in the list
        for (var i=0; i<chartList.length; i++) {
            if(chartList[i][2] == request["id"])
            {
                // apply the filter
                var newpoint = 0;

                if(chartList[i][4].length >= chartList[i][3])
                {
                    (chartList[i][4]).shift();
                }

                chartList[i][4].push(parseInt(request["value"]));

                for(var y=0; y<chartList[i][4].length;y++)
                {
                    newpoint += chartList[i][4][y];
                }
                newpoint = newpoint/chartList[i][4].length;

                // apply the ROS value conversion
                switch(chartList[i][2])
                {
                    case 'G1_1.M1' :
                        newpoint = newpoint*16.4/65536 - 8.2;
                        break;

                    case 'G1_1.M2' :
                        newpoint = newpoint*2/65536;
                        break;
```



```
                    case 'G1_1.M3' :
                        newpoint = newpoint/1000;
                        break;

                    default:break;
                }

                chartList[i][0].series[0].addPoint([request["timestamps"],
                    newpoint]);
            }
        }
    }
    else if(request["type"] == "SignalIndication")
    {
        if (request["text"] == "Enter in Normal Mode")
        {
        }
    }
}

/**
 * this function append when the error append whit the websocket
 *
 * @param       evt : the data coming from the websocket
 * @return      -
 */
function onError(evt)
{
    writeToScreen('<span style="color: red;">ERROR:</span> ' + evt.data);
}

/**
 * send a message with the websocket
 *
 * @param       message : the message to send
 * @return      -
 */
function doSend(message)
{
    websocket.send(message);
}

/*****************************************************************************/
/* Protocol                                                                  */
/*****************************************************************************/

    /**
     * send a set request to the server
     *
     * @param       id : the id of the desired subscribe value
     * @param       value : the value to set
     * @param       serial : the serial
     * @return      -
     */
    function SetRequest(id,value,serial) {
        var msg = {
            type: "SetRequest",
            id: id,
            value: value,
```

```
            serial: serial
        };

        websocket.send(JSON.stringify(msg));
    }

    /**
     * send a subscribe request to the server
     *
     * @param       id : the id of the desired subscribe value
     * @return      -
     */
    function subscribeRequest(id) {
        var msg = {
            type: "SubscribeRequest",
            id: id
        };
        switch(id)
        {

        }
        websocket.send(JSON.stringify(msg));
    }

/*****************************************************************************/
/* User Interaction                                                          */
/*****************************************************************************/

    /**
     * set the processor in normal mode
     *
     * @param       -
     * @return      -
     */
    function normalMode(){
        // get the selected node and value
        var nodeSelector = document.getElementById("node");
        var nodeValue = nodeSelector.options[nodeSelector.selectedIndex].value
            ;
        SetRequest(nodeValue+".P1","false",(String) (RequestSerial));
        RequestSerial = RequestSerial +1;
    }

    /**
     * display a message on the web page
     *
     * @param       message : the message to display
     * @return      -
     */
    function writeToScreen(message)
    {
        var p = document.getElementById("output");
        p.innerHTML = message;
    }

    /**
     * add a new graph in the web page
     *
     * @param       -
     * @return      -
```



```
     */
    function addGraphe() {
        // get the container for the graph and a unique ID
        var container = document.getElementById("graphContainer");
        var newdiv = document.createElement('div');
        var numi = document.getElementById('grapheValue');
        var num = (document.getElementById('grapheValue').value -1)+ 2;

        numi.value = num;
        var divIdName = 'graphDiv'+num;
        var divChartName = 'graphContainer'+num;

        // get the selected node, ROS and filter value
        var nodeSelector = document.getElementById("node" );
        var nodeValue = nodeSelector.options[nodeSelector.selectedIndex].value
            ;

        var ROSSelector = document.getElementById("ROS" );
        var ROSValue = ROSSelector.options[ROSSelector.selectedIndex].value;

        var filterSelector = document.getElementById("filter" );
        var filterValue = filterSelector.options[filterSelector.selectedIndex]
            .value;

        // creat the new div for the graph
        newdiv.setAttribute('id',divIdName);
        newdiv.innerHTML = '<fieldset><legend>Graph'+num+'> '+nodeValue+'.'+
            ROSValue+' filter: '+filterValue+' samples </legend><table><tr><td>
            <input type="submit" name="removeGraphe"" value="remove Graphe"
            onclick="removeGraphe('+divIdName+','+num+')"></td><td><div id="'+
            divChartName+'" style="width:80%; height:300px;"></div></td></tr></
            table></fieldset>';
        container.appendChild(newdiv);

        creatChart(divChartName,'Graphe' + num, nodeValue+'.'+ROSValue,
            filterValue);

        // subscribe the server for the value update notification
        subscribeRequest(nodeValue+'.'+ROSValue);
    }

    /**
     * remove a graph on the web page
     *
     * @param       evt : the data coming from the websocket
     * @return      -
     */
    function removeGraphe(divId,num) {
        var container = document.getElementById('graphContainer');
        deleteChart(num);
        container.removeChild(divId);
    }

    /**
     * delete the datamodel of a graph
     *
     * @param       num : the id of the chart in the data list
     * @return      -
     */
    function deleteChart(num){
```

```
    var text = 'Graphe' + num;
    for(var i=0; i<chartList.length;i++)
    {
        if(chartList[i][1] == text)
        {
            chartList.splice(i,1);
        }
    }
}

/**
 * create the datamodel of a graph
 *
 * @param       divID : the id of the container of the graph
 * @param       num : the id of the chart in the data list
 * @param       ros : the type of the y value to plot
 * @param       filter : the filter value for the graph
 * @return      -
 */
function creatChart(divId,num,ros,filter){
    var yaxis;

    switch(ros)
    {
        case 'G1_1.M1' :
            yaxis = "Vth [V]";
            break;

        case 'G1_1.M2' :
            yaxis = "Id [µA]";
            break;

        case 'G1_1.M3' :
            yaxis = "∫Id dVg [VµA]";
            break;

        default:break;
    }

    chartList.push([new Highcharts.Chart({
        chart: {
            renderTo: divId,
            defaultSeriesType: 'spline',
            zoomType: 'x',
        },
        title: {
            text: ''
        },
        subtitle: {
            text: document.ontouchstart === undefined ?
                'Click and drag in the plot area to zoom in' :
                'Pinch the chart to zoom in'
        },
        xAxis: {
            type: 'datetime',
            tickPixelInterval: 150,
            maxZoom: 20 * 1000
        },
        yAxis: {
            minPadding: 0.2,
```



```
            maxPadding: 0.2,
            title: {
                text: yaxis,
                margin: 80
            }
        },
         plotOptions: {
            area: {
                fillColor: {
                    linearGradient: { x1: 0, y1: 0, x2: 0, y2: 1},
                    stops: [
                        [0, Highcharts.getOptions().colors[0]],
                        [1, Highcharts.Color(Highcharts.getOptions().
                            colors[0]).setOpacity(0).get('rgba')]
                    ]
                },
                marker: {
                    radius: 2
                },
                lineWidth: 1,
                states: {
                    hover: {
                        lineWidth: 1
                    }
                },
                threshold: null
            }
        },
        legend: {
            enabled: false
        },
        series: [{
            type: 'area',
            name: 'NO2 data value',
            pointStart: (new Date()).getTime(),
            data: []
        }]
    }),num ,ros ,filter, new Array()] );
}

    window.addEventListener("load", init, false);
</script>
```

```
<!DOCTYPE html>
<!--meta charset="utf-8" /-->
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

<title> Gaz Detector HMI </title>

    <!-- 1. Add these JavaScript inclusions in the head of your page -->
    <script type="text/javascript" src="http://code.jquery.com/
        jquery-1.10.1.js"></script>
    <script type="text/javascript" src="http://code.highcharts.com/
        highcharts.js"></script>

    <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.8.2/
        jquery.min.js"></script>
    <script src="http://code.highcharts.com/highcharts.js"></script>

<!-- ____________________________________________________________ -->

 <html>
<body style="background:url('theme/monde.jpeg') no-repeat;">

<img src="img/hes_so.png" alt="hes-so" width="140" height="30" align="left">
<img src="img/hes_logo.png" alt="hes-logo" width="100" height="30" align=
    "right"><br>


<h2>WebSocket Status :</h2>  <p id="output"></p>

<h1 style="text-align:center;">Gaz Detector Debugger</h1>
Wiever :
        <a href="HMI.html">GazDetector Wiever</a>
<center>
    <table>
    <tr>
        <td>
        <fieldset>
            <legend> Commande </legend>
            <input type="submit" value="New measure" onclick="newMeasure()"><
                br>
            <input type="submit" value="Set DA value" onclick="setDA()">
            <input type="text" id="DAValue" name="DAValue" value="00"><br>
            <input type="submit" value="Read AD value" onclick="readAD()">
            <input type="text" id="ADValue" name="ADValue" value="00"><br>
        </fieldset>
        </td>

        <td>
        <fieldset>
            <legend> Caracteristic </legend>
                Node :
                <select name="node" id="nodeCar">
                    <option value="G1_1">G1_1</option>
                </select>
            <div id="caraContainer" style="width:800px; height:400px;"> </div>
        </fieldset>
    </tr>
    </table>
</center>

    <fieldset>
```



```
        <legend>Add Graph</legend>
        <input type="hidden" value="0" id="grapheValue" />
        <input type="submit" name="addGraphe" value="add" onclick="addGraphe()" />
        Node :
        <select name="node" id="nodeGraph">
            <option value="G1_1">G1_1</option>
        </select>
        ROS :
        <select name="ROS" id="ROS">
            <option value="M1">ROS1</option>
            <option value="M2">ROS2</option>
            <option value="M3">ROS3</option>
        </select>
        filter :
        <select name="filter" id="filter">
            <option value="1">no filter</option>
            <option value="2">2 samples</option>
            <option value="5">5 samples</option>
            <option value="10">10 samples</option>
            <option value="20">20 samples</option>
            <option value="40">40 samples</option>
        </select>
        </fieldset>

        <!-- 3. Add the container -->
        <div id="graphContainer"> </div>

</body>
</html>

<!-- ____________________________________________________________ -->
<script language="javascript" type="text/javascript">
    var wsUri = "ws://localhost:8888";
    var output;
    var RequestSerial = 0;
    // the tab for the subscribe chart
    var chartList = [];
    var chartChar;
    var currentSerie=0;

/*****************************************************************************/
/* WebSocket                                                                 */
/*****************************************************************************/
    /**
     * Initialisation function
     *
     * @param       -
     * @return      -
     */
    function init()
    {
        output = document.getElementById("output");
        var container = document.getElementById("caraContainer");
        creatChartChar(container);
        webSocketBehavior();
    }

    /**
     * function for the behavior of the websocket
     *
```

```
 * @param       -
 * @return      -
 */
function webSocketBehavior()
{
    websocket = new WebSocket(wsUri);
    websocket.onopen = function(evt)
    {
        onOpen(evt)
    };
    websocket.onclose = function(evt)
    {
        onClose(evt)
    };
    websocket.onmessage = function(evt)
    {
        onMessage(evt)
    };
    websocket.onerror = function(evt)
    {
        onError(evt)
    };
}

/**
 * this function append when the websocket open
 *
 * @param       evt : the data coming from the websocket
 * @return      -
 */
function onOpen(evt)
{
    writeToScreen("CONNECTED");

    debugMode();
}

/**
 * this function append when the websocket close
 *
 * @param       evt : the data coming from the websocket
 * @return      -
 */
function onClose(evt)
{
    writeToScreen("DISCONNECTED");
}

/**
 * this function append when the websocket receive a message
 *
 * @param       evt : the data coming from the websocket
 * @return      -
 */
function onMessage(evt)
{
    var request = JSON.parse(evt.data);
    if(request["type"] == "ValueChangeIndication")
    {
        if( request["id"] == "G1_1.M4" )
```



```
        {
            var y = (request["value"]/65536);
            y = y*2/65536;
            var x = parseInt(request["value"]&0x0000ffff);
            x = x*16.4/65536 - 8.2;
            chartChar.series[currentSerie].addPoint([x,y]);
        }
        else
        {
            // uptade all graph in the list
            for (var i=0; i<chartList.length; i++) {
                if(chartList[i][2] == request["id"])
                {   // apply the filter
                // apply the filter
                var newpoint = 0;

                if(chartList[i][4].length >= chartList[i][3])
                {
                    (chartList[i][4]).shift();
                }

                chartList[i][4].push(parseInt(request["value"]));

                for(var y=0; y<chartList[i][4].length;y++)
                {
                    newpoint += chartList[i][4][y];
                }
                newpoint = newpoint/chartList[i][4].length;

                // apply the ROS value conversion
                switch(chartList[i][2])
                {
                    case 'G1_1.M1' :
                        newpoint = newpoint*16.4/65536 - 8.2;
                        break;

                    case 'G1_1.M2' :
                        newpoint = newpoint*2/65536;
                        break;

                    case 'G1_1.M3' :
                        newpoint = newpoint/1000;
                        break;

                    default:break;
                }

                chartList[i][0].series[0].addPoint([request["timestamps"],
                    newpoint]);
                }
            }
        }
    }
    else if(request["type"] == "GetResponse")
    {
        var adContainer = document.getElementById("ADValue");
        adContainer.value = request["value"];
    }
    else if(request["type"] == "SignalIndication")
    {
```

```
            if(request["text"] == "Characteristic send complete")
            {
                //currentSerie++;
                setTimeInterval((String)(1),(String)(RequestSerial));
            }
            else if (request["text"] == "Characteristic measure complete")
            {
                chartChar.series[0].remove();
                chartChar.addSeries({data: []});
            }
            else if (request["text"] == "Enter in Debug Mode")
            {
            }
        }
    }

    /**
     * this function append when the error append whit the websocket
     *
     * @param       evt : the data coming from the websocket
     * @return      -
     */
    function onError(evt)
    {
        writeToScreen('<span style="color: red;">ERROR:</span> ' + evt.data);
    }

    /**
     * send a message with the websocket
     *
     * @param       message : the message to send
     * @return      -
     */
    function doSend(message)
    {
        websocket.send(message);
    }

/*****************************************************************************/
/* Protocol                                                                  */
/*****************************************************************************/

    /**
     * send a get request to the server
     *
     * @param       id : the id of the desired subscribe value
     * @param       serial : the serial
     * @return      -
     */
    function getRequest(id,serial) {
        var msg = {
            type: "GetRequest",
            id: id,
            serial: serial
        };
        RequestSerial = RequestSerial +1;
        websocket.send(JSON.stringify(msg));
    }

        /**
```



```
     * send a set request to the server
     *
     * @param       id : the id of the desired subscribe value
     * @param       value : the value to set
     * @param       serial : the serial
     * @return      -
     */
    function setRequest(id,value,serial) {
        var msg = {
            type: "SetRequest",
            id: id,
            value: value,
            serial: serial
        };
        RequestSerial = RequestSerial +1;
        websocket.send(JSON.stringify(msg));
    }

    /**
     * send a subscribe request to the server
     *
     * @param       id : the id of the desired subscribe value
     * @return      -
     */
    function subscribeRequest(id) {
        var msg = {
            type: "SubscribeRequest",
            id: id
        };
        RequestSerial = RequestSerial +1;
        websocket.send(JSON.stringify(msg));
    }

        /**
     * send a set Time Interval request to the server
     *
     * @param       id : the id of the desired subscribe value
     * @param       value : the value of polling
     * @param       serial : the serial
     * @return      -
     */
    function setTimeInterval(value,serial) {
        var msg = {
            type: "SetTimeInterval",
            value: value,
            serial: serial
        };
        RequestSerial = RequestSerial +1;
        websocket.send(JSON.stringify(msg));
    }

/*****************************************************************************/
/* User Interaction                                                          */
/*****************************************************************************/
    /**
     * display a message on the web page
     *
     * @param       message : the message to display
     * @return      -
     */
```

```
function writeToScreen(message)
{
    var p = document.getElementById("output");
    p.innerHTML = message;
}

/**
 * set the processor in debug mode
 *
 * @param       -
 * @return      -
 */
function debugMode(){
    // get the selected node and value
    var nodeSelector = document.getElementById("nodeCar" );
    var nodeValue = nodeSelector.options[nodeSelector.selectedIndex].value
        ;
    setRequest(nodeValue+".P1","true",(String) (RequestSerial));
}

/**
 * append when the user push the newMeasure() button
 *
 * @param       -
 * @return      -
 */
function newMeasure(){

    // get the selected node and value
    var nodeSelector = document.getElementById("nodeCar" );
    var nodeValue = nodeSelector.options[nodeSelector.selectedIndex].value
        ;
    setRequest(nodeValue+".P2","true",(String) (RequestSerial));
    subscribeRequest('G1_1.M4');
    subscribeRequest('G1_1.M5');
    setTimeInterval((String)(0.2),(String)(RequestSerial));
}

/**
 * append when the user push the setDA button
 *
 * @param       -
 * @return      -
 */
function setDA() {

    // get the selected node and value
    var value = document.getElementById('DAValue').value;
    value = ((Math.round(value) + 8.2) * 65536)/16.4;
    value = Math.round(value);
    var nodeSelector = document.getElementById("nodeCar" );
    var nodeValue = nodeSelector.options[nodeSelector.selectedIndex].value
        ;
    setRequest(nodeValue+".P3",(String) (value),(String) (RequestSerial));
}

/**
 * append when the user push the readAD button
 *
 * @param       -
```



```
 * @return      -
 */
function readAD() {

    // get the selected node and value
    var nodeSelector = document.getElementById("nodeCar" );
    var nodeValue = nodeSelector.options[nodeSelector.selectedIndex].value
        ;
    getRequest(nodeValue+".M5",(String)(RequestSerial));
}


/**
 * add a new graph in the web page
 *
 * @param       -
 * @return      -
 */
function addGraphe() {
    // get the container for the graph and a unique ID
    var container = document.getElementById("graphContainer");
    var newdiv = document.createElement('div');
    var numi = document.getElementById('grapheValue');
    var num = (document.getElementById('grapheValue').value -1)+ 2;

    numi.value = num;
    var divIdName = 'graphDiv'+num;
    var divChartName = 'graphContainer'+num;

    // get the selected node, ROS and filter value
    var nodeSelector = document.getElementById("nodeGraph" );
    var nodeValue = nodeSelector.options[nodeSelector.selectedIndex].value
        ;

    var ROSSelector = document.getElementById("ROS" );
    var ROSValue = ROSSelector.options[ROSSelector.selectedIndex].value;

    var filterSelector = document.getElementById("filter" );
    var filterValue = filterSelector.options[filterSelector.selectedIndex]
        .value;

    // creat the new div for the graph
    newdiv.setAttribute('id',divIdName);
    newdiv.innerHTML = '<fieldset><legend>Graph'+num+'> '+nodeValue+'.'+
        ROSValue+' filter: '+filterValue+' samples </legend><table><tr><td>
        <input type="submit" name="removeGraphe"" value="remove Graphe"
        onclick="removeGraphe('+divIdName+','+num+')"></td><td><div id="'+
        divChartName+'" style="width:80%; height:300px;"></div></td></tr></
        table></fieldset>';
    container.appendChild(newdiv);

    creatChart(divChartName,'Graphe' + num, nodeValue+'.'+ROSValue,
        filterValue);

    // subscribe the server for the value update notification
    subscribeRequest(nodeValue+'.'+ROSValue);
}

/**
 * remove a graph on the web page
```

```
 *
 * @param       evt : the data coming from the websocket
 * @return      -
 */
function removeGraphe(divId,num) {
    var container = document.getElementById('graphContainer');
    deleteChart(num);
    container.removeChild(divId);
}

/**
 * delete the datamodel of a graph
 *
 * @param       num : the id of the chart in the data list
 * @return      -
 */
function deleteChart(num){
    var text = 'Graphe' + num;
    for(var i=0; i<chartList.length;i++)
    {
        if(chartList[i][1] == text)
        {
            chartList.splice(i,1);
        }
    }
}

/**
 * create the datamodel of a graph
 *
 * @param       divID : the id of the container of the graph
 * @param       num : the id of the chart in the data list
 * @param       ros : the type of the y value to plot
 * @param       filter : the filter value for the graph
 * @return      -
 */
function creatChart(divId,num,ros,filter){
    var yaxis;

    switch(ros)
    {
        case 'G1_1.M1' :
            yaxis = "Vth [V]";
            break;

        case 'G1_1.M2' :
            yaxis = "Id [µA]";
            break;

        case 'G1_1.M3' :
            yaxis = "∫Id dVg [VµA]";
            break;

        default:break;
    }

    chartList.push([new Highcharts.Chart({
        chart: {
            renderTo: divId,
            defaultSeriesType: 'spline',
```



```
            zoomType: 'x',
        },
        title: {
            text: ''
        },
        subtitle: {
            text: document.ontouchstart === undefined ?
                'Click and drag in the plot area to zoom in' :
                'Pinch the chart to zoom in'
        },
        xAxis: {
            type: 'datetime',
            tickPixelInterval: 150,
            maxZoom: 20 * 1000
        },
        yAxis: {
            minPadding: 0.2,
            maxPadding: 0.2,
            title: {
                text: yaxis,
                margin: 80
            }
        },
         plotOptions: {
            area: {
                fillColor: {
                    linearGradient: { x1: 0, y1: 0, x2: 0, y2: 1},
                    stops: [
                        [0, Highcharts.getOptions().colors[0]],
                        [1, Highcharts.Color(Highcharts.getOptions().
                            colors[0]).setOpacity(0).get('rgba')]
                    ]
                },
                marker: {
                    radius: 2
                },
                lineWidth: 1,
                states: {
                    hover: {
                        lineWidth: 1
                    }
                },
                threshold: null
            }
        },
        legend: {
            enabled: false
        },
        series: [{
            type: 'area',
            name: 'NO2 data value',
            pointStart: (new Date()).getTime(),
            data: []
        }]
    }),num ,ros ,filter, new Array()] );
}

/**
 * create the characteristic  graph
 *
```

```
         * @param       divID : the id of the container of the graph
         * @return      -
         */
        function creatChartChar(divId){
            chartChar = new Highcharts.Chart({
                chart: {
                    type: 'scatter',
                    defaultSeriesType: 'area',
                    renderTo: divId
                    //defaultSeriesType: 'spline',
                },
                title: {
                    text: ''
                },
                xAxis: {
                    title: {
                        text: 'Vg [V]'
                    },
                    gridLineWidth: 1,
                },
                yAxis: {
                    title: {
                        text: 'Id [uA]'
                    },
                    plotLines: [{
                        value: 0,
                        width: 1,
                        color: '#808080'
                    }]
                },
                plotOptions: {
                    series: {
                        lineWidth: 1,
                    }
                },
                legend: {
                    enabled: false
                },
                series: [{
                    name: 'char',
                    pointStart: 0,
                    data: []
                }]
            });
        }

        window.addEventListener("load", init, false);
    </script>
```

Ernst-Abbe-Fachhochschule Jena
Hochschule für angewandte Wissenschaften

Hes·so VALAIS WALLIS

Anhang E

XML Datei

```xml
<?xml version="1.0" encoding="UTF-8"?>
<scadaClasses xmlns="http://vlesdi.hevs.ch/scada-classes"
              targetNamespace="http://vlesdi.hevs.ch/scada-classes">

    <!-- Classes of nodes are defined here -->
    <nodeTypes>

        <nodeType id="N1" name="GazDetector">

            <!-- A set point defines a target value for field level control --
                >
            <parameter>
                <point id="P1" name="mode" description="Mode Auto/Manual"
                    datatype="BOOLEAN"/>
                <point id="P2" name="newMeas" description="Start a new
                    measure" datatype="BOOLEAN"/>
                <point id="P3" name="DAValue" description="Value of the DAC"
                    datatype="FLOAT"/>
            </parameter>

            <!-- A process variable is an analogue measurement result updated
                periodically -->
            <measValue>
                <point id="M1" name="ROS1" description="Voltage measurement"
                    datatype="FLOAT" unit="V"/>
                <point id="M2" name="ROS2" description="Current measurement"
                    datatype="FLOAT" unit="A"/>
                <point id="M3" name="ROS3" description="Surface measurement"
                    datatype="FLOAT" unit="VA"/>
                <point id="M4" name="Char I" description="Current measurement"
                    datatype="FLOAT" unit="A"/>
                <point id="M5" name="Char V" description="Voltage stimulus"
                    datatype="FLOAT" unit="V"/>
                <point id="M6" name="AD Value" description="Current value of
                    the ADC" datatype="FLOAT" unit="V"/>
            </measValue>

          <!-- A signal represents the occurence of an event -->
          <signal>
            <point id="S1" name="High NO2 Concentration"  description=
                "GazDetector ({ref}) detect a high concentration of NO2."
                datatype="BOOLEAN" triggerCondition="equal"
                triggerValue="true"/>
            <point id="S2" name="End of the transmission of the
                characteristic"  description="Characteristic send complete"
                datatype="BOOLEAN" triggerCondition="equal"
                triggerValue="true"/>
            <point id="S3" name="End of the measurement of the characteristic"
                description="Characteristic measure complete"
                datatype="BOOLEAN" triggerCondition="equal"
                triggerValue="true"/>
          </signal>

          <!-- An alarm is a special type of event requiring acknowledgement
              from an operator -->
          <alarm>
          </alarm>

        </nodeType>
```



```xml
    </nodeTypes>
</scadaClasses>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<scadaSystem    xmlns:xi="http://www.w3.org/TR/xinclude/"
                xmlns="http://vlesdi.hevs.ch/scada-system"
                targetNamespace="http://vlesdi.hevs.ch/scada-system">

    <!-- Include the model of SCADA classes -->
    <xi:include href="GazDetector.xml" xmlns:sc="http://vlesdi.hevs.ch/scada-
        classes"/>

    <!-- The instances of nodes for a given process are defined here -->
    <scada>

        <group id="G1" name="GazDetector Serial">
            <node id="G1_1"  name="GazDetector-1" nodeTypeRef="N1"/>
        </group>

        <field>
            <outputs>
                <!-- Auto / Manual -->
                <output type="BOOLEAN" deviceType="modbus" deviceAddress="1"
                    address="110" addressType="5" nodeRef="G1_1" pointRef="P1"/
                    >
                <!-- new Measure -->
                <output type="BOOLEAN" deviceType="modbus" deviceAddress="1"
                    address="111" addressType="5" nodeRef="G1_1" pointRef="P2"/
                    >
                <!-- DA Value -->
                <output type="FLOAT" deviceType="modubs" deviceAddress="1"
                    address="120" addressType="6" nodeRef="G1_1" pointRef="P3"/
                    >
            </outputs>

            <inputs>
                <!-- Measure Value -->
                <input type="FLOAT" deviceType="modbus" deviceAddress="1"
                    address="100" addressType="4" nodeRef="G1_1" pointRef="M1"
                    conversion="1.0*value"/>
                <!-- Measure Value -->
                <input type="FLOAT" deviceType="modbus" deviceAddress="1"
                    address="101" addressType="4" nodeRef="G1_1" pointRef="M2"
                    conversion="1.0*value"/>
                <!-- Measure Value -->
                <input type="FLOAT" deviceType="modbus" deviceAddress="1"
                    address="102" addressType="4" nodeRef="G1_1" pointRef="M3"
                    conversion="1.0*value"/>
                <!-- Char I Value -->
                <input type="FLOAT" deviceType="modbus" deviceAddress="1"
                    address="103" addressType="4" nodeRef="G1_1" pointRef="M4"
                    conversion="1.0*value"/>
                <!-- Char U Value -->
                <input type="FLOAT" deviceType="modbus" deviceAddress="1"
                    address="104" addressType="4" nodeRef="G1_1" pointRef="M5"
                    conversion="1.0*value"/>
                <!-- AD Value -->
                <input type="FLOAT" deviceType="modbus" deviceAddress="1"
                    address="105" addressType="4" nodeRef="G1_1" pointRef="M6"
                    conversion="1.0*value"/>
            </inputs>

            <triggers>
```



```xml
                <trigger type="BOOLEAN" deviceType="modbus" deviceAddress="1"
                    address="130" addressType="2" nodeRef="G1_1" pointRef="S1"/
                    >
                <trigger type="BOOLEAN" deviceType="modbus" deviceAddress="1"
                    address="131" addressType="2" nodeRef="G1_1" pointRef="S2"/
                    >
                <trigger type="BOOLEAN" deviceType="modbus" deviceAddress="1"
                    address="132" addressType="2" nodeRef="G1_1" pointRef="S3"/
                    >
            </triggers>

        </field>
    </scada>
</scadaSystem>
```

Ernst-Abbe-Fachhochschule Jena
Hochschule für angewandte Wissenschaften

Hes·so VALAIS WALLIS

Anhang F

Eingebettet System Code

```c
/*
 * File:   acquisition.h
 * Author: louismayencourt
 *
 * Created on 23 mai 2014, 10:59
 */

#ifndef ACQUISITION_H
#define ACQUISITION_H

#include "hevstypes.h"
#include "xf.h"
#include "adc_driver.h"
#include "dac_driver.h"

#ifdef  __cplusplus
extern "C" {
#endif

#define T_ACQU_HIGH 5000 // us
#define T_ACQU_LOW 5 // us
#define T_ACQU_TM 100

#define NBR_OVERSAMPLE_MAX 10
#define NBR_PULSE 200
#define NBR_VALUE 100
#define NBR_SAMPLE 100
#define NBR_SAMPLE_UP 50

#define STIMULUS_INIT_HIGH 62738
#define STIMULUS_INIT_LOW 2797
#define STIMULUS_MEASURE_HIGH  62738//40760
#define STIMULUS_MEASURE_LOW 24775
#define STIMULUS_ZERO 32768

    typedef enum SM_ACQUISITION
    {
        ST_SM_ACQU_INIT,
        ST_SM_ACQU_WAIT,
        ST_SM_ACQU_HIGH,
        ST_SM_ACQU_SAMPLING,
        ST_SM_ACQU_LOW,
        ST_SM_ACQU_ERROR
    }SM_ACQUISITION;

    typedef struct Acquisition
    {
        SM_ACQUISITION sm_Acquisition, oldSm_Acquisition;
        TimerID timerID;
        TimerID sampleTimerID;
        TimerID watchdogID;

        UINT16 thigh;
        UINT16 tacqu;

        UINT16 nbrOversample;
        UINT16 sample[NBR_SAMPLE];

        UINT16 delta;
        UINT16 stimulus[NBR_VALUE];
```



```c
        UINT16 stimulusStep;
        UINT16 sampleStep;
        BOOL up_down;
        BOOL pos_neg;

        BOOL acqu_init;
    }Acquisition;

    void Acquisition_init(Acquisition* me);
    void Acquisition_SM(Acquisition* me,Event ev);
    void Acquisition_start(Acquisition* me,UINT16 maxValue,BOOL acqu_init);
    void Acquisition_setTHigh(Acquisition* me,UINT16 tHigh);
    void Acquisition_setNbrOversample(Acquisition* me,UINT16 overSample);

#ifdef  __cplusplus
}
#endif

#endif  /* ACQUISITION_H */
```

```c
/*****************************************************************************/
/* FILE           : acquisition.c                                            */
/* AUTHOR           : Mayencourt Louis                                       */
/*---------------------------------------------------------------------------*/
/* COMMENTS     : acquisition file for the gazDetector                       */
/*---------------------------------------------------------------------------*/
/* REVISION     : 1.0  - tested with XC16    compiler                       */
/*****************************************************************************/
#include "acquisition.h"
#include "adc_driver.h"
#include "controller.h"
#include "analogOut_driver.h"

/*****************************************************************************/
/* FUNCTION     : acquisition_init                                           */
/* INPUT            : me, the acquisition machine                            */
/* OUTPUT           : -                                                      */
/*---------------------------------------------------------------------------*/
/* COMMENTS     : initialisation function                                    */
/*****************************************************************************/
void Acquisition_init(Acquisition* me)
{
    int i;

    me->sm_Acquisition = ST_SM_ACQU_INIT;

    me->oldSm_Acquisition = me->sm_Acquisition;
    me->timerID = NULLTIMER;
    me->sampleTimerID = NULLTIMER;

    Acquisition_setTHigh(me,T_ACQU_HIGH);
    Acquisition_setNbrOversample(me,4);

    for(i=0;i<NBR_VALUE; i++)
    {
        me->stimulus[i] = STIMULUS_ZERO;
    }

    me->stimulusStep = 0;
    me->sampleStep = 0;
    me->pos_neg = false;
    me->up_down = true;
    me->acqu_init = false;

    XF_pushEvent(evInit,false);
}
/*****************************************************************************/
/* FUNCTION     : acquisition_start                                          */
/* INPUT            : me, the acquisition machine                            */
/*                    maxValue, the max value of the stimulus                */
/*                    Uout = Value* 16,4/2^16 - 8.2                          */
/*                    acqu_init, acquisition sequence = true                 */
/*                               initialisation sequence = false             */
/* OUTPUT           : -                                                      */
/*---------------------------------------------------------------------------*/
/* COMMENTS     : set function                                               */
/*****************************************************************************/
void Acquisition_start(Acquisition* me,UINT16 maxValue,BOOL acqu_init)
{
    UINT16 i;
```



```c
    UINT16 y;

    me->delta = (maxValue-STIMULUS_ZERO)/(NBR_VALUE>>1);

    y=0;
    for(i=0;i<NBR_VALUE; i+=2)
    {
        me->stimulus[i] = STIMULUS_ZERO + y*(me->delta);
        me->stimulus[i+1] = STIMULUS_ZERO - y*(me->delta);
        y++;
    }

    me->up_down = true;
    me->pos_neg = false;
    me->stimulusStep = 0;
    me->sampleStep = 0;
    me->acqu_init = acqu_init;

    XF_pushEvent(evAcquStart,false);
}
/*****************************************************************************/
/* FUNCTION     : acquisition_setTHigh                                       */
/* INPUT            : me, the acquisition machine                            */
/*                    tHigh, the new value of tHigh in us                    */
/* OUTPUT           : -                                                      */
/*---------------------------------------------------------------------------*/
/* COMMENTS     : set function                                               */
/*****************************************************************************/
void Acquisition_setTHigh(Acquisition* me,UINT16 tHigh)
{
    me->thigh = tHigh;
    me->tacqu = tHigh*0.9;
}
/*****************************************************************************/
/* FUNCTION     : acquisition_setNbrOversample                               */
/* INPUT            : me, the acquisition machine                            */
/*                    overSample, the new value of overSample                */
/* OUTPUT           : -                                                      */
/*---------------------------------------------------------------------------*/
/* COMMENTS     : set function                                               */
/*****************************************************************************/
void Acquisition_setNbrOversample(Acquisition* me,UINT16 overSample)
{
    if(overSample > NBR_OVERSAMPLE_MAX)
    {
        me->nbrOversample = NBR_OVERSAMPLE_MAX;
    }
    else
    {
        me->nbrOversample = overSample;
    }
}
/*****************************************************************************/
/* FUNCTION     : Acquisition_sm                                             */
/* INPUT            : me, the acquisition / ev, the current event            */
/* OUTPUT           : -                                                      */
/*---------------------------------------------------------------------------*/
/* COMMENTS     : State machine function                                     */
/*****************************************************************************/
void Acquisition_SM(Acquisition* me, Event ev)
```

```
{
    int i,y;
    me->oldSm_Acquisition = me->sm_Acquisition;

    //this is the sm_SWITCH_ control switch

    switch (me->sm_Acquisition)
    {
        case ST_SM_ACQU_INIT:
            if (ev == evInit)
            {
                me->sm_Acquisition = ST_SM_ACQU_WAIT;
            }
            break;
        case ST_SM_ACQU_WAIT:
            if (ev == evAcquStart)
            {
               me->sm_Acquisition = ST_SM_ACQU_HIGH;
            }
            break;
        case ST_SM_ACQU_HIGH:
            if (ev == evAcquTMSampling)
            {
                me->sm_Acquisition = ST_SM_ACQU_SAMPLING;
            }
            if (ev == evAcquTM)
            {
                me->sm_Acquisition = ST_SM_ACQU_ERROR;
            }
        case ST_SM_ACQU_SAMPLING:
            if (ev == evAcquTMHigh)
            {
                me->sm_Acquisition = ST_SM_ACQU_LOW;
            }
            if (ev == evAcquTM)
            {
                me->sm_Acquisition = ST_SM_ACQU_ERROR;
            }
            break;
        case ST_SM_ACQU_LOW:
            if (ev == evAcquTMLow)
            {
                me->sm_Acquisition = ST_SM_ACQU_HIGH;
            }
            if (ev == evAcquStop)
            {
                me->sm_Acquisition = ST_SM_ACQU_WAIT;
            }
            if (ev == evAcquEnd)
            {
                me->sm_Acquisition = ST_SM_ACQU_WAIT;
            }
            if (ev == evAcquTM)
            {
                me->sm_Acquisition = ST_SM_ACQU_ERROR;
            }
            break;
        case ST_SM_ACQU_ERROR:
            if (ev == evAcquErrorAck)
            {
```



```
                me->sm_Acquisition = ST_SM_ACQU_WAIT;
            }
            break;

    }

    //this is the sm _SWITCH_ action switch

    switch (me->sm_Acquisition)
    {
        case ST_SM_ACQU_WAIT:
            if (me->oldSm_Acquisition == ST_SM_ACQU_INIT)
            {
                XF_pushEvent(evAcquDefault,false);
            }
            if (me->oldSm_Acquisition == ST_SM_ACQU_LOW)
            {
                // unschedul the hard timer
                XF_unscheduleHardTimer(me->timerID,false);

                // generate a 0 value for Vg
                dac_write(STIMULUS_ZERO);

                // restart the sampling condition
                me->stimulusStep = 0;
                me->sampleStep = 0;
                me->up_down = true;
                me->pos_neg = false;
            }
            if (me->oldSm_Acquisition == ST_SM_ACQU_ERROR)
            {
                // restart the sampling condition
                me->stimulusStep = 0;
                me->sampleStep = 0;
                me->up_down = true;
                me->pos_neg = false;
                XF_pushEvent(evAcquStart,false);
            }
            break;
        case ST_SM_ACQU_HIGH:
            if (me->oldSm_Acquisition == ST_SM_ACQU_WAIT ||
                    me->oldSm_Acquisition == ST_SM_ACQU_LOW)
            {
                // restart the watchdog timer
                XF_unscheduleTimer(me->watchdogID,false);
                me->watchdogID = XF_scheduleTimer(T_ACQU_TM,evAcquTM,false);

                // check the top of the rampe
                if(me->stimulusStep >= NBR_VALUE-1)
                {
                    me->up_down = !me->up_down;
                    me->stimulusStep=0;
                }

                // generate the Vg value
                if(me->up_down == true)
                    dac_write(me->stimulus[me->stimulusStep]);
                else
                    dac_write(me->stimulus[NBR_VALUE-1 - me->stimulusStep]);
```

```
            me->stimulusStep++;

            // toggle the positive negative flag
            me->pos_neg = !me->pos_neg;

            // schedule the hard timer
            me->timerID = XF_scheduleHardTimer(me->thigh,evAcquTMHigh,
                false);
            me->sampleTimerID = XF_scheduleHardTimer(me->tacqu,
                evAcquTMSampling,false);
        }
        break;
    case ST_SM_ACQU_SAMPLING:
        if (me->oldSm_Acquisition == ST_SM_ACQU_HIGH)
        {
            UINT32 value;
            // restart the watchdog timer
            XF_unscheduleTimer(me->watchdogID,false);
            me->watchdogID = XF_scheduleTimer(T_ACQU_TM,evAcquTM,false);

            // sampling only for the positive value
            if(me->pos_neg == true)
            {
                if(me->acqu_init == true)
                {
                    value = 0;
                    for(i=0; i<me->nbrOversample;i++)
                    {
                        value += adc_read();
                        //me->sample[me->sampleStep] += adc_read();
                    }
                    value = value/me->nbrOversample;
                    me->sample[me->sampleStep] = value & 0x0000ffff;
                   // me->sample[me->sampleStep] = (me->sample[me-
                       >sampleStep])/me->nbrOversample;
                    analogOut_write((me->sample[me->sampleStep])>>6);
                    me->sampleStep++;
                }
            }

        }
        break;
    case ST_SM_ACQU_LOW:
        if (me->oldSm_Acquisition == ST_SM_ACQU_SAMPLING)
        {
            // restart the watchdog timer
            XF_unscheduleTimer(me->watchdogID,false);
            me->watchdogID = XF_scheduleTimer(T_ACQU_TM,evAcquTM,false);

            // generate the 0 value for Vg
            dac_write(STIMULUS_ZERO);

            // check the end condition
            if(me->up_down == false)
            {
                if(me->stimulusStep == NBR_VALUE-2)
                    XF_pushEvent(evAcquEnd,false);
            }

            // schedule the hard timer
```



```
                me->timerID = XF_scheduleHardTimer(T_ACQU_LOW,evAcquTMLow,
                    false);
            }
            break;
        case ST_SM_ACQU_ERROR:
            if (me->oldSm_Acquisition == ST_SM_ACQU_HIGH ||
                    me->oldSm_Acquisition == ST_SM_ACQU_SAMPLING ||
                    me->oldSm_Acquisition == ST_SM_ACQU_LOW )
            {
                // restart the acquisition
                XF_pushEvent(evAcquErrorAck,false);

            }
            break;
    }
}
```

```c
/*
 * File:   adc_driver.h
 * Author: louismayencourt
 *
 * Created on 19 mai 2014, 15:34
 */

#ifndef ADC_DRIVER_H
#define ADC_DRIVER_H

#include <p24FJ128GC006.h>
#include "hevstypes.h"

#ifdef  __cplusplus
extern "C" {
#endif

#define SHDN LATFbits.LATF0
#define CS LATFbits.LATF1
#define BUSY PORTFbits.RF3
#define CONV LATFbits.LATF4
#define RD LATFbits.LATF5

#define AD0 PORTCbits.RC12
#define AD1 PORTCbits.RC15
#define AD2 PORTDbits.RD8
#define AD3 PORTDbits.RD9
#define AD4 PORTDbits.RD10
#define AD5 PORTDbits.RD11
#define AD6 PORTDbits.RD0
#define AD7 PORTCbits.RC13
#define AD8 PORTDbits.RD7
#define AD9 PORTDbits.RD6
#define AD10 PORTDbits.RD5
#define AD11 PORTDbits.RD4
#define AD12 PORTDbits.RD3
#define AD13 PORTDbits.RD2
#define AD14 PORTDbits.RD1
#define AD15 PORTCbits.RC14

//prototypes of adc - interface
    void adc_init();
    void adc_start();
    UINT16 adc_read();


#ifdef  __cplusplus
}
#endif

#endif  /* ADC_DRIVER_H */
```

```
/*****************************************************************************/
/* FILE           : adc_driver.c                                             */
/* AUTHOR           : Mayencourt Louis                                       */
/*---------------------------------------------------------------------------*/
/* COMMENTS     : adc file for the gazDetector                               */
/*---------------------------------------------------------------------------*/
/* REVISION     : 1.0  - tested with XC16    compiler                        */
/*****************************************************************************/
#include "adc_driver.h"
#include "xf.h"

/*****************************************************************************/
/* FUNCTION     : adc_init                                                   */
/* INPUT            : -                                                      */
/* OUTPUT           : -                                                      */
/*---------------------------------------------------------------------------*/
/* COMMENTS     : initialisation function                                    */
/*****************************************************************************/
void adc_init()
{
    // configure the PORT in Digital
    ANSF = 0x0000;
    ANSC = 0x0000;
    ANSD = 0x0000;

    // configure the PORT direction
    TRISFbits.TRISF0 = 0;
    TRISFbits.TRISF1 = 0;
    TRISFbits.TRISF3 = 1;
    TRISFbits.TRISF4 = 0;
    TRISFbits.TRISF5 = 0;

    TRISC = 0xFFFF;
    TRISD = 0xFFFF;

    // initial value
    SHDN = 1;
    CS = 0;
    RD = 0;
    CONV = 1;
}
/*****************************************************************************/
/* FUNCTION     : adc_start                                                  */
/* INPUT            : -                                                      */
/* OUTPUT           : -                                                      */
/*---------------------------------------------------------------------------*/
///* COMMENTS     : start function                                           */
/*****************************************************************************/
void adc_start()
{
    while(!BUSY);

    // start of a convertion
    CONV = 0;

}
/*****************************************************************************/
/* FUNCTION     : adc_read                                                   */
/* INPUT            : -                                                      */
/* OUTPUT           : the value of the current                               */
```



```
/*---------------------------------------------------------------------------*/
/* COMMENTS     : read function                                              */
/*****************************************************************************/
UINT16 adc_read()
{
    UINT16 value;

    // start the conversion
    adc_start();

    // wait the end of the conversion
    while(!BUSY);

    // compute the adc value
    value = AD0 + (AD1<<1) + (AD2<<2) + (AD3<<3);
    value += (AD4<<4) + (AD5<<5) + (AD6<<6) + (AD7<<7) ;
    value += (AD8<<8) + (AD9<<9) + (AD10<<10) + (AD11<<11);
    value += (AD12<<12) + (AD13<<13) + (AD14<<14) + (AD15<<15);

    // restor the cmd value
    CONV= 1;

    // return the read value
    return (value-32768);

}
```

```c
/*
 * File:   analogOut_driver.h
 * Author: louismayencourt
 *
 * Created on 20 mai 2014, 13:39
 */

#ifndef ANALOGOUT_DRIVER_H
#define ANALOGOUT_DRIVER_H

#include <p24FJ128GC006.h>
#include "hevstypes.h"

#ifdef  __cplusplus
extern "C" {
#endif

    //prototypes of AnalogOut - interface
    void analogOut_init(void);
    void analogOut_write(UINT16 value);


#ifdef  __cplusplus
}
#endif

#endif  /* ANALOGOUT_DRIVER_H */
```

```c
/*****************************************************************************/
/* FILE           : analogOut_driver.c                                       */
/* AUTHOR           : Mayencourt Louis                                       */
/*---------------------------------------------------------------------------*/
/* COMMENTS     : dac file for the gazDetector                               */
/*---------------------------------------------------------------------------*/
/* REVISION     : 1.0  - tested with XC16    compiler                        */
/*****************************************************************************/
#include "analogOut_driver.h"

/*****************************************************************************/
/* FUNCTION     : analogOut_init                                             */
/* INPUT            : -                                                      */
/* OUTPUT           : -                                                      */
/*---------------------------------------------------------------------------*/
/* COMMENTS     : initialisation function                                    */
/*****************************************************************************/
void analogOut_init()
{
    // configure the PORT in analog mode
    ANSBbits.ANSB13 = 1;

    // configuration of the DAC module
    DAC2CON = 0x8082;
}
/*****************************************************************************/
/* FUNCTION     : analogOut_write                                            */
/* INPUT            : the value to write (10 bits)                           */
/* OUTPUT           : -                                                      */
/*---------------------------------------------------------------------------*/
/* COMMENTS     : write function                                             */
/*****************************************************************************/
void analogOut_write(UINT16 value)
{
    // write the value on the register
    DAC2DAT = value & 0x03FF;
}
```

```c
/*
 * File:   communication.h
 * Author: louismayencourt
 *
 * Created on 20 mai 2014, 13:52
 */

#ifndef COMMUNICATION_H
#define COMMUNICATION_H

#include <p24FJ128GC006.h>
#include "hevstypes.h"
#include "xf.h"
//#include "uart_driver.h"
#include "crc.h"
#include "controller.h"
#include "dac_driver.h"
#include "adc_driver.h"

#ifdef  __cplusplus
extern "C" {
#endif

#define T_END_OF_FRAME 3000 // us = 2.5 char (rounded)
#define FRAME_MAX_SIZE 259 // 256(PDU) + 1(ADDR) + 2(CRC)
#define RESPONSE_MAX_SIZE 10

#define T_COMM_TM 100

#define MY_ADDRESS 1

#define NBR_SUPPORTED_FCT_CODE 4

#define NBR_DISCRETE_INPUT 5
#define NBR_COILS 2
#define NBR_INPUT_REG 6
#define NBR_HOLDING_REG 1

#define MODE_RUN 1
#define MODE_STOP 0
#define NORMAL 0
#define DEBUG 1

#define CHAR_ADDR 3
#define AD_ADDR 4

    typedef enum SM_COMMUNICATION
    {
        ST_SM_COMM_INIT,
        ST_SM_COMM_IDLE,
        ST_SM_COMM_NEW_DATA,
        ST_SM_COMM_CHECKING_REQUEST,
        ST_SM_COMM_PROCESSING_ACTION,
        ST_SM_COMM_FORMATTING_REPLY,
        ST_SM_COMM_FORMATTING_ERROR,
        ST_SM_COMM_SEND_DATA,
        ST_SM_COMM_NO_POLL
    }SM_COMMUNICATION;

    typedef struct Communication
```



```c
    {
        SM_COMMUNICATION sm_Communication, oldSm_Communication;
        TimerID timerID;
        TimerID watchdog;

        Controller* cont;

        // communication temp variable
        UINT8* frame_ptr;

        UINT8* response_ptr;

        UINT8* send_ptr;

        UINT8 exeptionCode;

        // index for reading the caracteristic
        UINT16 sample_index;
        UINT16* sample;
        UINT16* stimulus;

        // cmd temp variable
        BOOL mode;
        BOOL new_meas;
        BOOL reset_index;

        // data temp variable
        UINT16 cda_value;
        UINT16 cad_value;
        UINT16 sample_value;
        UINT16 stimulus_value;

        // signal temp variable
        BOOL end_char;

        // datastructure;
        BOOL* discreteInput[NBR_DISCRETE_INPUT];
        BOOL* coils[NBR_COILS];
        UINT16* inputRegister[NBR_INPUT_REG];
        UINT16* holdingRegister[NBR_HOLDING_REG];

    }Communication;

    void Communication_init(Communication* me,Controller* cont);
    void Communication_SM(Communication* me,Event ev);
    void Communication_single_coil(Communication* me);
    void Communication_input_register(Communication* me,UINT16 addr);
    void Communication_discrete_input(Communication* me,UINT16 addr);

#ifdef  __cplusplus
}
#endif

#endif  /* COMMUNICATION_H */
```

```c
/*****************************************************************************/
/* FILE           : communication.c                                          */
/* AUTHOR           : Mayencourt Louis                                       */
/*---------------------------------------------------------------------------*/
/* COMMENTS     : communication file for the gazDetecto                      */
/*---------------------------------------------------------------------------*/
/* REVISION     : 1.0  - tested with XC16    compiler                       */
/*****************************************************************************/
#include "communication.h"
#include "uart_driver.h"

/*****************************************************************************/
/* FUNCTION     : Communication_init                                         */
/* INPUT            : me, the communication machine                          */
/*                    cont, the Controller                                   */
/* OUTPUT           : -                                                      */
/*---------------------------------------------------------------------------*/
/* COMMENTS     : initialisation function                                    */
/*****************************************************************************/
void Communication_init(Communication* me,Controller* cont)
{
    me->sm_Communication = ST_SM_COMM_INIT;

    me->oldSm_Communication = me->sm_Communication;
    me->timerID = NULLTIMER;

    me->cont = cont;

    me->frame_ptr = uart_get_in_buffer();
    me->response_ptr = uart_get_out_buffer();

    me->exeptionCode = 0;

    me->sample_index = 0;
    me->sample = cont->sample;
    me->stimulus = cont->acqu->stimulus;

    me->end_char = false;

    me->inputRegister[0] = &(cont->ros1Value);
    me->inputRegister[1] = &(cont->ros2Value);
    me->inputRegister[2] = &(cont->ros3Value);
    me->inputRegister[3] = &(me->sample_value);
    me->inputRegister[4] = &(me->stimulus_value);
    me->inputRegister[5] = &(me->cad_value);

    me->coils[0] = &(me->mode);
    me->coils[1] = &(me->new_meas);

    me->discreteInput[0] = &(cont->high_NO2);
    me->discreteInput[1] = &(me->end_char);
    me->discreteInput[2] = &(cont->end_measure);
    me->discreteInput[3] = &(cont->normalMode);
    me->discreteInput[4] = &(cont->debugMode);


    me->holdingRegister[0] = &(me->cda_value);

    XF_pushEvent(evInit,false);
}
```



```c
/*****************************************************************************/
/* FUNCTION     : Communication_single_coil                                  */
/* INPUT            : me, the communication machine                          */
/* OUTPUT           : -                                                      */
/*---------------------------------------------------------------------------*/
/* COMMENTS     : send the cmd signal to the crontroller                     */
/*****************************************************************************/
void Communication_single_coil(Communication* me)
{
    if(me->mode == DEBUG)
    {
        Controller_stop();
        me->cont->debugMode = true;
    }
    else
    {
        Controller_start_auto();
        me->cont->normalMode = true;
    }

    if(me->new_meas == true)
    {
        Controller_start_single();
        me->new_meas = false;
        me->sample_index = 0;
    }
}
/*****************************************************************************/
/* FUNCTION     : Communication_discrete_input                               */
/* INPUT            : me, the communication machine                          */
/* OUTPUT           : -                                                      */
/*---------------------------------------------------------------------------*/
/* COMMENTS     : update the input value                                     */
/*****************************************************************************/
void Communication_discrete_input(Communication* me,UINT16 addr)
{
    // clear the flag after reading
    *(me->discreteInput[addr]) = false;
}
/*****************************************************************************/
/* FUNCTION     : Communication_input_register                               */
/* INPUT            : me, the communication machine                          */
/*                    addr, the address of the read input                    */
/* OUTPUT           : -                                                      */
/*---------------------------------------------------------------------------*/
/* COMMENTS     : update the input value                                     */
/*****************************************************************************/
void Communication_input_register(Communication* me,UINT16 addr)
{
    if(addr == CHAR_ADDR && me->cont->busy == false)
    {
        me->sample_value = me->sample[me->sample_index];

        if(me->sample_index <= 49)
            me->stimulus_value = me->stimulus[me->sample_index*2];
        else
            me->stimulus_value = me->stimulus[ NBR_SAMPLE - (me->sample_index-
                49)*2];
```

```
            if(me->sample_index < NBR_SAMPLE-1)
                me->sample_index++;
            else
            {
                me->sample_index = 0;
                me->end_char = true;
            }
        }

        if(addr == AD_ADDR)
        {
            me->cad_value = adc_read();
        }

    }
    /*****************************************************************************/
    /* FUNCTION     : Communication_sm                                           */
    /* INPUT            : me, the Communication / ev, the current event          */
    /* OUTPUT           : -                                                      */
    /*---------------------------------------------------------------------------*/
    /* COMMENTS     : State machine function                                     */
    /*****************************************************************************/
    void Communication_SM(Communication* me, Event ev)
    {
        me->oldSm_Communication = me->sm_Communication;

        //this is the sm_SWITCH_ control switch
        switch (me->sm_Communication)
        {
            case ST_SM_COMM_INIT:
                if (ev == evInit)
                {
                    me->sm_Communication = ST_SM_COMM_IDLE;
                }
                break;
            case ST_SM_COMM_IDLE:
                DEBUG2 = 0;
                if (ev == evCommTMWatchdog)
                {
                   me->sm_Communication = ST_SM_COMM_NO_POLL;
                }
                if(ev == evCommDataIn)
                {
                    me->sm_Communication = ST_SM_COMM_NEW_DATA;
                }
                break;
            case ST_SM_COMM_NEW_DATA:
                if (ev == evCommDataIn)
                {
                    me->sm_Communication = ST_SM_COMM_NEW_DATA;
                }
                if (ev == evCommTM)
                {
                    DEBUG2 = 1;
                    me->sm_Communication = ST_SM_COMM_CHECKING_REQUEST;
                }
                break;
            case ST_SM_COMM_CHECKING_REQUEST:
                if (ev == evCommCheckOk)
                {
```



```
                    me->sm_Communication = ST_SM_COMM_PROCESSING_ACTION;
                }
                if (ev == evCommErrorChecking)
                {
                    me->sm_Communication = ST_SM_COMM_IDLE;
                }
                if (ev == evCommErrorProcessing)
                {
                    me->sm_Communication = ST_SM_COMM_FORMATTING_ERROR;
                }
                break;
            case ST_SM_COMM_PROCESSING_ACTION:
                if (ev == evCommEndProcessing)
                {
                    me->sm_Communication = ST_SM_COMM_FORMATTING_REPLY;
                }
                if (ev == evCommErrorProcessing)
                {
                    me->sm_Communication = ST_SM_COMM_FORMATTING_ERROR;
                }
                break;
            case ST_SM_COMM_FORMATTING_REPLY:
                if (ev == evCommSendReply)
                {
                    me->sm_Communication = ST_SM_COMM_SEND_DATA;
                }
                break;
            case ST_SM_COMM_FORMATTING_ERROR:
                if (ev == evCommSendReply)
                {
                    me->sm_Communication = ST_SM_COMM_SEND_DATA;
                }
            case ST_SM_COMM_SEND_DATA:
                if (ev == evCommDataOut)
                {
                    me->sm_Communication = ST_SM_COMM_SEND_DATA;
                }
                if (ev == evCommEndOfTransmission)
                {
                    me->sm_Communication = ST_SM_COMM_IDLE;
                }
                break;
            case ST_SM_COMM_NO_POLL:
                if (ev == evCommDefault)
                {
                    me->sm_Communication = ST_SM_COMM_IDLE;
                }
                break;

        }

        //this is the sm _SWITCH_ action switch

        switch (me->sm_Communication)
        {
            case ST_SM_COMM_IDLE:
                if (me->oldSm_Communication == ST_SM_COMM_INIT)
                {
                    uart_enable_interrupt(true);
                    me->watchdog = XF_scheduleTimer(T_COMM_TM,evCommTMWatchdog,
```

```
                false);
        }
        if (me->oldSm_Communication == ST_SM_COMM_SEND_DATA)
        {
            me->watchdog = XF_scheduleTimer(T_COMM_TM,evCommTMWatchdog,
                false);
        }
        if (me->oldSm_Communication == ST_SM_COMM_CHECKING_REQUEST)
        {
            me->watchdog = XF_scheduleTimer(T_COMM_TM,evCommTMWatchdog,
                false);
        }
        break;
    case ST_SM_COMM_NEW_DATA:
        if (me->oldSm_Communication == ST_SM_COMM_IDLE)
        {
            XF_unscheduleTimer(me->watchdog,false);
            me->timerID = XF_scheduleHardTimer(T_END_OF_FRAME,evCommTM,
                false);
        }
        if (me->oldSm_Communication == ST_SM_COMM_NEW_DATA &&
                ev == evCommDataIn)
        {
            XF_unscheduleHardTimer(me->timerID,false);
            me->timerID = XF_scheduleHardTimer(T_END_OF_FRAME,evCommTM,
                false);
        }
        break;
    case ST_SM_COMM_CHECKING_REQUEST:
        if (me->oldSm_Communication == ST_SM_COMM_NEW_DATA)
        {
            UINT8 addr;
            UINT16 CRC;
            UINT16 CRCInt;
            UINT8 length;
            UINT8 fctCode;
            BOOL valide;
            UINT8 i;
            UINT16 arg1;
            UINT16 arg2;

            addr = me->frame_ptr[0];

            // check the address
            if(addr == MY_ADDRESS)
            {
                // check the CRC
                length = uart_get_frame_in_length();
                CRCInt = CRC16(me->frame_ptr,length-2);
                CRC = ((me->frame_ptr[length-1])<<8);
                CRC = CRC + me->frame_ptr[length-2];

                if(CRC == CRCInt)
                {
                    // check data address and fonction code
                    valide = false;
                    fctCode = me->frame_ptr[1];
                    arg1 = (me->frame_ptr[2]<<8) + me->frame_ptr[3];
                    arg2 = (me->frame_ptr[4]<<8) + me->frame_ptr[5];
```



```
                    switch(fctCode)
                    {
                        case 0x02: // Read Discretes Input
                            if(arg2 >= 0x0001 && arg2 <=NBR_DISCRETE_INPUT
                                )
                            {
                                if(arg1 + arg2 <= NBR_DISCRETE_INPUT)
                                    valide = true;
                                else
                                    me->exeptionCode = 2;
                            }
                            else
                                me->exeptionCode = 3;

                            break;

                        case 0x04: // Read Input register
                            if(arg2 >= 0x0001 && arg2 <=NBR_INPUT_REG)
                            {
                                if(arg1 + arg2 <= NBR_INPUT_REG)
                                    valide = true;
                                else
                                    me->exeptionCode = 2;
                            }
                            else
                                me->exeptionCode = 3;

                            break;

                        case 0x05: // Write Singel Coil
                            if(arg2 == 0x0000 || arg2 == 0xff00)
                            {
                                if(arg1 <= NBR_COILS)
                                    valide = true;
                                else
                                    me->exeptionCode = 2;
                            }
                            else
                                me->exeptionCode = 3;

                            break;

                        case 0x06: // Write Single Register
                            if(arg2 >= 0x0000 && arg2 <= 0xffff)
                            {
                                if(arg1 <= NBR_HOLDING_REG)
                                    valide = true;
                                else
                                    me->exeptionCode = 2;
                            }
                            else
                                me->exeptionCode = 3;

                            break;

                        default: // fct code non valide
                            me->exeptionCode = 1;
                            break;
                    }
```

```
                    if(valide == true)
                        XF_pushEvent(evCommCheckOk,false);
                    else
                        XF_pushEvent(evCommErrorProcessing,false);
                }
                else
                    XF_pushEvent(evCommErrorChecking,false);
            }
            else
                XF_pushEvent(evCommErrorChecking,false);
        }
        break;
    case ST_SM_COMM_PROCESSING_ACTION:
        if (me->oldSm_Communication == ST_SM_COMM_CHECKING_REQUEST)
        {
            UINT16 fctCode;
            UINT16 arg1;
            UINT16 arg2;
            UINT16 i;
            UINT16 value;

            DEBUG2= 1;
            fctCode = me->frame_ptr[1];
            arg1 = (me->frame_ptr[2]<<8) + me->frame_ptr[3];
            arg2 = (me->frame_ptr[4]<<8) + me->frame_ptr[5];

            me->response_ptr = uart_get_out_buffer();
            switch(fctCode)
            {
                case 0x02: // Read Discrete Input
                    // write directly in the reponse frame
                    me->response_ptr += 3;
                    value = *(me->discreteInput[arg1]);
                    *(me->response_ptr) = value;

                    Communication_discrete_input(me,arg1);
                    break;

                case 0x04: // Read Input register
                    // write directly in the reponse frame
                    me->response_ptr += 3;

                    for(i=arg1; i<(arg2+arg1); i++){
                        Communication_input_register(me,i);

                        value = *(me->inputRegister[i]);
                        *(me->response_ptr) = value>>8;
                        me->response_ptr++;
                        *(me->response_ptr) = value&0x00ff;
                        me->response_ptr++;
                    }

                    break;

                case 0x05: // Write Singel Coil
                    me->response_ptr += 3;
                    if(arg2 == 0xff00)
                        *(me->coils[arg1]) = true;
                    else
                        *(me->coils[arg1]) = false;
```



```
                    Communication_single_coil(me);
                    break;

                case 0x06: // Write Single Register
                    *(me->holdingRegister[arg1]) = arg2;

                    dac_write(me->cda_value);
                    break;

                default: // fct code non valide
                    break;
            }
            DEBUG2 = 0;
            XF_pushEvent(evCommEndProcessing,false);
        }
        break;
    case ST_SM_COMM_FORMATTING_REPLY:
        if (me->oldSm_Communication == ST_SM_COMM_PROCESSING_ACTION)
        {
            UINT16 fctCode;
            UINT16 arg1;
            UINT16 arg2;
            UINT16 i;
            UINT16 CRC;
            UINT8 length;

            fctCode = me->frame_ptr[1];
            arg1 = (me->frame_ptr[2]<<8) + me->frame_ptr[3];
            arg2 = (me->frame_ptr[4]<<8) + me->frame_ptr[5];

            // Slave Address
            me->response_ptr = uart_get_out_buffer();
            me->response_ptr[0] = MY_ADDRESS;

            // PDU
            switch(fctCode)
            {
                case 0x02: // Read Discrete Input
                    me->response_ptr[1] = me->frame_ptr[1];
                    me->response_ptr[2] = 1;
                    length = 4;
                    me->response_ptr += length;
                    break;

                case 0x04: // Read Input register
                    me->response_ptr[1] = me->frame_ptr[1];
                    me->response_ptr[2] = arg2<<1;
                    length = arg2*2 + 3;
                    me->response_ptr += length;
                    break;

                case 0x05: // Write Singel Coil
                    me->response_ptr = uart_get_out_buffer();
                    length = 6;
                    for(i=0; i<length; i++){
                        *(me->response_ptr) = me->frame_ptr[i];
                        me->response_ptr++;
                    }
                    break;
```

```
                case 0x06: // Write Single Register
                    length = 6;
                    for(i=0; i<length; i++){
                        *(me->response_ptr) = me->frame_ptr[i];
                        me->response_ptr++;
                    }
                    break;

                default: // fct code non valide
                    break;
            }

            // CRC
            CRC = CRC16(uart_get_out_buffer(),length);
            *(me->response_ptr) = CRC & 0x00ff;
            me->response_ptr++;
            *(me->response_ptr) = (CRC&0xff00)>>8;

            XF_pushEvent(evCommSendReply,false);
        }
        break;
    case ST_SM_COMM_FORMATTING_ERROR:
        if (me->oldSm_Communication == ST_SM_COMM_CHECKING_REQUEST)
        {
            UINT16 CRC;
            UINT8 length;

            // Slave Address
            me->response_ptr[0] = MY_ADDRESS;

            // PDU
            me->response_ptr[1] = 0x80 + me->frame_ptr[1];
            me->response_ptr[2] = me->exeptionCode;

            // CRC
            CRC = CRC16(uart_get_out_buffer(),length);
            me->response_ptr[3] = CRC & 0x00ff;
            me->response_ptr[4] = (CRC&0xff00)>>8;

            XF_pushEvent(evCommSendReply,false);
        }
        if (me->oldSm_Communication == ST_SM_COMM_PROCESSING_ACTION)
        {
            XF_pushEvent(evCommDefault,false);
        }
        break;
    case ST_SM_COMM_SEND_DATA:
        if (me->oldSm_Communication == ST_SM_COMM_FORMATTING_REPLY ||
                me->oldSm_Communication == ST_SM_COMM_FORMATTING_ERROR)
        {

            uart_send_frame(me->response_ptr - uart_get_out_buffer());

        }
        if (me->oldSm_Communication == ST_SM_COMM_SEND_DATA &&
                ev == evCommDataOut)
        {
            XF_pushEvent(evCommEndOfTransmission,false);
        }
```



```
            break;
        case ST_SM_COMM_NO_POLL:
            if (me->oldSm_Communication == ST_SM_COMM_IDLE)
            {
                // dummy read
                uart_get_frame_in_length();

                XF_pushEvent(evCommDefault,false);
            }
            break;
    }
}
```

```c
//*********************************************************************
// PIC18LF6680 Standard configuration Bits Settings for LABYGAME
//*********************************************************************

#include <p24FJ128GC006.h>

// section 34 of the pdf
// non watchdog timer - programmation on pin PGx2 - JTAG off
_CONFIG1( FWDTEN_WDT_DIS & ICS_PGx2 & JTAGEN_OFF);
// non primari oscillator - OSCIO pin IO fonction - interne FRC osc
_CONFIG2( POSCMD_NONE & OSCIOFCN_ON & FNOSC_FRCPLL);
// SOSC in digital mode
_CONFIG3( SOSCSEL_OFF);
// pll config
_CONFIG4(PLLDIV_NODIV);
```

```
/*
 * File:   controller.h
 * Author: louismayencourt
 *
 * Created on 23 mai 2014, 15:13
 */

#ifndef CONTROLLER_H
#define CONTROLLER_H

#include "hevstypes.h"
#include "xf.h"
#include "acquisition.h"

#ifdef  __cplusplus
extern "C" {
#endif

#define NBR_VALUE 100
#define ROS1_DELTA 1200

    typedef enum SM_CONTROLLER
    {
        ST_SM_CONT_INIT,
        ST_SM_CONT_WAIT,
        ST_SM_CONT_SINGLE_INIT,
        ST_SM_CONT_SINGLE_ACQU,
        ST_SM_CONT_SINGLE_COMPUT,
        ST_SM_CONT_AUTO_INIT,
        ST_SM_CONT_AUTO_ACQU,
        ST_SM_CONT_AUTO_COMPUT
    }SM_CONTROLLER;

    typedef struct Controller
    {
        SM_CONTROLLER sm_Controller, oldSm_Controller;
        TimerID timerID;

        Acquisition* acqu;

        UINT16 ros1Value;
        UINT16 ros2Value;
        UINT16 ros3Value;
        UINT16* sample;

        BOOL high_NO2;
        BOOL end_measure;
        BOOL busy;
        BOOL normalMode;
        BOOL debugMode;

    }Controller;

    void Controller_init(Controller* me,Acquisition* acqu);
    void Controller_SM(Controller*me,Event ev);
    void Controller_start_single(void);
    void Controller_start_auto(void);
    void Controller_start_init(void);
    void Controller_stop(void);
```



```
#ifdef  __cplusplus
}
#endif

#endif  /* CONTROLLER_H */
```

```c
/*****************************************************************************/
/* FILE           : controller.c                                             */
/* AUTHOR           : Mayencourt Louis                                       */
/*---------------------------------------------------------------------------*/
/* COMMENTS     : controller file for the gazDetector                        */
/*---------------------------------------------------------------------------*/
/* REVISION     : 1.0  - tested with XC16    compiler                       */
/*****************************************************************************/
#include "controller.h"
#include "analogOut_driver.h"

/*****************************************************************************/
/* FUNCTION     : Controller_init                                            */
/* INPUT            : me, the controller machine                             */
/* OUTPUT           : -                                                      */
/*---------------------------------------------------------------------------*/
/* COMMENTS     : initialisation function                                    */
/*****************************************************************************/
void Controller_init(Controller* me,Acquisition* acqu)
{
    me->sm_Controller = ST_SM_CONT_INIT;

    me->oldSm_Controller = me->sm_Controller;
    me->timerID = NULLTIMER;

    me->acqu = acqu;

    me->sample = acqu->sample;

    me->busy = false;
    me->high_NO2 = false;
    me->end_measure = false;
    me->normalMode = true;
    me->debugMode = false;

    XF_pushEvent(evInit,false);
}
/
    ****************************************************************************
    ****/
/* FUNCTION     : Controller_start_init
    */
/* INPUT            : -
    */
/* OUTPUT           : -
    */
/
    *---------------------------------------------------------------------------
    ---*/
/* COMMENTS     : Interface fonction for the control of a single
    initialisation*/
/
    ****************************************************************************
    ****/
void Controller_start_init(void)
{}
/*****************************************************************************/
/* FUNCTION     : Controller_start_single                                    */
/* INPUT            : -                                                      */
/* OUTPUT           : -                                                      */
```



```c
/*---------------------------------------------------------------------------*/
/* COMMENTS     : Interface fonction for the control of a single measure  */
/*****************************************************************************/
void Controller_start_single(void)
{
    XF_pushEvent(evContSingel,false);
}
/*****************************************************************************/
/* FUNCTION     : Controller_start_auto                                      */
/* INPUT            : -                                                      */
/* OUTPUT           : -                                                      */
/*---------------------------------------------------------------------------*/
/* COMMENTS     : Interface fonction for the control of a auto measure    */
/*****************************************************************************/
void Controller_start_auto(void)
{
    XF_pushEvent(evContAuto,false);
}
/*****************************************************************************/
/* FUNCTION     : Controller_stop                                            */
/* INPUT            : -                                                      */
/* OUTPUT           : -                                                      */
/*---------------------------------------------------------------------------*/
/* COMMENTS     : Interface fonction for the control of a stop measure    */
/*****************************************************************************/
void Controller_stop(void)
{
    XF_pushEvent(evContStop,false);
}
/*****************************************************************************/
/* FUNCTION     : Controller_compute                                         */
/* INPUT            : -                                                      */
/* OUTPUT           : -                                                      */
/*---------------------------------------------------------------------------*/
/* COMMENTS     : compute the output value with the sample value          */
/*****************************************************************************/
void Controller_compute(Controller* me)
{
    UINT32 temp;
    float coeff;

    UINT16* sample;
    UINT8 i;

    sample = me->acqu->sample;

    // compute the ROS1 value
    for(i=0;i<NBR_SAMPLE_UP;i++)
    {
        temp = sample[i];
        if(sample[i+1]>= temp+ROS1_DELTA &&
            sample[i+2] >= temp + ROS1_DELTA &&
             sample[i+3] >= temp + ROS1_DELTA)
            break;
    }
    me->ros1Value = me->acqu->stimulus[i*2];

    // compute the ROS2 value
    temp = sample[NBR_SAMPLE_UP];
    me->ros2Value = temp;
```

```
        temp = 0;
        // compute the ROS3 value
        for(i=0;i<NBR_SAMPLE_UP;i++)
        {
            temp += sample[i];
        }
        coeff = (me->acqu->delta*16.4/65536)*2/65.536;
        temp = temp*coeff;
        me->ros3Value = temp;

        me->end_measure = true;
        me->busy = false;


    }
    /*****************************************************************************/
    /* FUNCTION     : Controller_sm                                              */
    /* INPUT            : me, the controller / ev, the current event             */
    /* OUTPUT           : -                                                      */
    /*---------------------------------------------------------------------------*/
    /* COMMENTS     : State machine function                                     */
    /*****************************************************************************/
    void Controller_SM(Controller* me, Event ev)
    {
        me->oldSm_Controller = me->sm_Controller;

        //this is the sm_SWITCH_ control switch
        switch (me->sm_Controller)
        {
            case ST_SM_CONT_INIT:
                if (ev == evInit)
                {
                    me->sm_Controller = ST_SM_CONT_WAIT;
                }
                break;
            case ST_SM_CONT_WAIT:
                if (ev == evContSingel)
                {
                   me->sm_Controller = ST_SM_CONT_SINGLE_INIT;
                }
                if (ev == evContAuto)
                {
                   me->sm_Controller = ST_SM_CONT_AUTO_INIT;
                }
                break;
            case ST_SM_CONT_SINGLE_INIT:
                if (ev == evAcquEnd)
                {
                    me->sm_Controller = ST_SM_CONT_SINGLE_ACQU;
                }
                break;
            case ST_SM_CONT_SINGLE_ACQU:
                if (ev == evAcquEnd) {
                    me->sm_Controller = ST_SM_CONT_SINGLE_COMPUT;
                }
                break;
            case ST_SM_CONT_SINGLE_COMPUT:
                if (ev == evContEndComput) {
                    me->sm_Controller = ST_SM_CONT_WAIT;
```



```
                }
                break;
            case ST_SM_CONT_AUTO_INIT:
                if (ev == evAcquEnd)
                {
                    me->sm_Controller = ST_SM_CONT_AUTO_ACQU;
                }
                if (ev == evContStop)
                {
                    me->sm_Controller = ST_SM_CONT_WAIT;
                }
                break;
            case ST_SM_CONT_AUTO_ACQU:
                if (ev == evAcquEnd)
                {
                    me->sm_Controller = ST_SM_CONT_AUTO_COMPUT;
                }
                if (ev == evContStop)
                {
                    me->sm_Controller = ST_SM_CONT_WAIT;
                }
                break;
            case ST_SM_CONT_AUTO_COMPUT:
                if (ev == evContEndComput) {
                    me->sm_Controller = ST_SM_CONT_AUTO_INIT;
                }
                if (ev == evContStop)
                {
                    me->sm_Controller = ST_SM_CONT_WAIT;
                }
                break;
        }

        //this is the sm _SWITCH_ action switch

        switch (me->sm_Controller)
        {
            case ST_SM_CONT_WAIT:
                if (me->oldSm_Controller == ST_SM_CONT_INIT)
                {
                    XF_pushEvent(evContAuto,false);
                }
                if (me->oldSm_Controller == ST_SM_CONT_SINGLE_COMPUT)
                {
                }
                if (me->oldSm_Controller == ST_SM_CONT_AUTO_INIT)
                {
                }
                if (me->oldSm_Controller == ST_SM_CONT_AUTO_ACQU)
                {
                }
                if (me->oldSm_Controller == ST_SM_CONT_AUTO_COMPUT)
                {
                }
                break;
            case ST_SM_CONT_SINGLE_INIT:
                if (me->oldSm_Controller == ST_SM_CONT_WAIT)
                {
                    me->busy = true;
                    Acquisition_start(me->acqu,STIMULUS_INIT_HIGH,false);
```

```
                }
                break;
            case ST_SM_CONT_SINGLE_ACQU:
                if (me->oldSm_Controller == ST_SM_CONT_SINGLE_INIT)
                {
                    Acquisition_start(me->acqu,STIMULUS_MEASURE_HIGH,true);
                }
                break;
            case ST_SM_CONT_SINGLE_COMPUT:
                if (me->oldSm_Controller == ST_SM_CONT_SINGLE_ACQU)
                {
                    Controller_compute(me);
                    XF_pushEvent(evContEndComput,false);
                }
                break;
            case ST_SM_CONT_AUTO_INIT:
                if (me->oldSm_Controller == ST_SM_CONT_WAIT ||
                        me->oldSm_Controller == ST_SM_CONT_AUTO_COMPUT)
                {
                    me->busy = true;
                    Acquisition_start(me->acqu,STIMULUS_INIT_HIGH,false);
                }
                break;
            case ST_SM_CONT_AUTO_ACQU:
                if (me->oldSm_Controller == ST_SM_CONT_AUTO_INIT)
                {
                    Acquisition_start(me->acqu,STIMULUS_MEASURE_HIGH,true);
                }
                break;
            case ST_SM_CONT_AUTO_COMPUT:
                if (me->oldSm_Controller == ST_SM_CONT_AUTO_ACQU)
                {
                    Controller_compute(me);
                    XF_pushEvent(evContEndComput,false);
                }
                break;
        }
    }
```

```c
#include "hevstypes.h"

UINT16 CRC16( UINT8 * puchMsg, UINT16 usDataLen );
```

```
#include <stdint.h>
#include "crc.h"

const UINT8 auchCRCHi[] = {
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00,
    0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
    0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81,
    0x40, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
    0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00,
    0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
    0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80,
    0x41, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
    0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00,
    0xC1, 0x81,
0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
    0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81,
    0x40, 0x01,
0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
    0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00,
    0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
    0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80,
    0x41, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
    0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00,
    0xC1, 0x81,
0x40
};
const UINT8 auchCRCLo[] = {
0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06, 0x07, 0xC7, 0x05,
    0xC5, 0xC4,
0x04, 0xCC, 0x0C, 0x0D, 0xCD, 0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B,
    0xC9, 0x09,
0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A, 0x1E, 0xDE, 0xDF,
    0x1F, 0xDD,
0x1D, 0x1C, 0xDC, 0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12,
    0x13, 0xD3,
0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3, 0xF2, 0x32, 0x36,
    0xF6, 0xF7,
0x37, 0xF5, 0x35, 0x34, 0xF4, 0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE,
    0xFA, 0x3A,
0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29, 0xEB, 0x2B, 0x2A,
    0xEA, 0xEE,
0x2E, 0x2F, 0xEF, 0x2D, 0xED, 0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7,
    0xE6, 0x26,
0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60, 0x61, 0xA1, 0x63,
    0xA3, 0xA2,
0x62, 0x66, 0xA6, 0xA7, 0x67, 0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D,
```



```
    0xAF, 0x6F,
0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68, 0x78, 0xB8, 0xB9,
    0x79, 0xBB,
0x7B, 0x7A, 0xBA, 0xBE, 0x7E, 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74,
    0x75, 0xB5,
0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71, 0x70, 0xB0, 0x50,
    0x90, 0x91,
0x51, 0x93, 0x53, 0x52, 0x92, 0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54,
    0x9C, 0x5C,
0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B, 0x99, 0x59, 0x58,
    0x98, 0x88,
0x48, 0x49, 0x89, 0x4B, 0x8B, 0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D,
    0x4C, 0x8C,
0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42, 0x43, 0x83, 0x41,
    0x81, 0x80,
0x40
};

UINT16 CRC16( UINT8 * puchMsg, UINT16 usDataLen )
{
    UINT8 uchCRCHi = 0xFF;  /* high byte of CRC initialized */
    UINT8 uchCRCLo = 0xFF;  /* low byte of CRC initialized */
    UINT8 uIndex;                   /* will index into CRC lookup table */
    while (usDataLen--) /* pass through message buffer */
    {
        uIndex = uchCRCLo ^ *puchMsg++ ; /* calculate the CRC */
        uchCRCLo = uchCRCHi ^ auchCRCHi[uIndex] ;
        uchCRCHi = auchCRCLo[uIndex] ;
    }
    return (uchCRCHi << 8 | uchCRCLo) ;
}
```

```c
/*
 * File:   dac_driver.h
 * Author: louismayencourt
 *
 * Created on 19 mai 2014, 16:05
 */

#ifndef DAC_DRIVER_H
#define DAC_DRIVER_H

#include <p24FJ128GC006.h>
#include "hevstypes.h"

#ifdef  __cplusplus
extern "C" {
#endif

#define CKL LATBbits.LATB12

#define DA0 LATEbits.LATE0
#define DA1 LATEbits.LATE1
#define DA2 LATEbits.LATE2
#define DA3 LATEbits.LATE3
#define DA4 LATEbits.LATE4
#define DA5 LATEbits.LATE5
#define DA6 LATEbits.LATE6
#define DA7 LATEbits.LATE7
#define DA8 LATGbits.LATG6
#define DA9 LATGbits.LATG7
#define DA10 LATGbits.LATG8
#define DA11 LATGbits.LATG9
#define DA12 LATBbits.LATB5
#define DA13 LATBbits.LATB4
#define DA14 LATBbits.LATB2
#define DA15 LATBbits.LATB3

    //prototypes of dac - interface
    void dac_init(void);
    void dac_write(UINT16 value);
    UINT16 dac_computeValue(float value);


#ifdef  __cplusplus
}
#endif

#endif  /* DAC_DRIVER_H */
```

```c
/*****************************************************************************/
/* FILE           : dac_driver.c                                             */
/* AUTHOR           : Mayencourt Louis                                       */
/*---------------------------------------------------------------------------*/
/* COMMENTS     : dac file for the gazDetector                               */
/*---------------------------------------------------------------------------*/
/* REVISION     : 1.0  - tested with XC16    compiler                        */
/*****************************************************************************/
#include "dac_driver.h"

/*****************************************************************************/
/* FUNCTION     : dac_init                                                   */
/* INPUT            : -                                                      */
/* OUTPUT           : -                                                      */
/*---------------------------------------------------------------------------*/
/* COMMENTS     : initialisation function                                    */
/*****************************************************************************/
void dac_init()
{
    // configure the PORT in Digital
    ANSBbits.ANSB12 = 0;

    ANSBbits.ANSB2 = 0;
    ANSBbits.ANSB3 = 0;
    ANSBbits.ANSB4 = 0;
    ANSBbits.ANSB5 = 0;
    ANSE = 0x0000;
    ANSG = 0x0000;

    // configure the PORT direction
    TRISBbits.TRISB12 = 0;

    TRISBbits.TRISB2 = 0;
    TRISBbits.TRISB3 = 0;
    TRISBbits.TRISB4 = 0;
    TRISBbits.TRISB5 = 0;
    TRISE = 0x0000;
    TRISG = 0x0000;

    // initial value
    CKL = 1;

    dac_write(0x0000);
}
/*****************************************************************************/
/* FUNCTION     : dac_write                                                  */
/* INPUT            : the value to write                                     */
/* OUTPUT           : -                                                      */
/*---------------------------------------------------------------------------*/
/* COMMENTS     : write function                                             */
/*****************************************************************************/
void dac_write(UINT16 value)
{
    // write the value on PORT
    LATE = 0x00FF & value;
    LATG = 0x03C0 & (value>>2);
    DA12 = (0x1000 & value)>>12;
    DA13 = (0x2000 & value)>>13;
    DA14 = (0x4000 & value)>>14;
    DA15 = (0x8000 & value)>>15;
```



```c
    // set the clock
    CKL = 0;
    CKL = 1;
}
/
    ****************************************************************************
    **/
/* FUNCTION     : dac_computeValue
    */
/* INPUT            : the voltage value in v
    */
/* OUTPUT           : the translate voltage value in number
    */
/*---------------------------------------------------------------------------
    */
/* COMMENTS     : transform a voltage value to the corespondante number
    value*/
/
    ****************************************************************************
    **/
UINT16 dac_computeValue(float value)
{
    return (value + 8.2) * 65536/16.4;
}
```

```c
/*
 * File:   defines.h
 * Author: louismayencourt
 *
 * Created on 19 mai 2014, 16:05
 */
#ifndef DEFDEF
#define DEFDEF

#include <stdint.h>

typedef  unsigned char BOOL;
#ifndef false
#define false 0
#endif
#ifndef true
#define true  1
#endif

enum
{
    evInit = 10

};


//.........................

#define evNoEvent 0
#define evDefault 1

#define evInit   10
#define evGameInit 11
#define evTM1000 12

#define evContDefault 20
#define evContSingel 21
#define evContStop 22
#define evContAuto 23
#define evContEndComput 24

#define evAcquDefault 30
#define evAcquStart 31
#define evAcquStop 32
#define evAcquTMSampling 33
#define evAcquTMHigh 34
#define evAcquTMLow 35
#define evAcquEnd 36
#define evAcquTM 37
#define evAcquErrorAck 38

#define evCommDefault 40
#define evCommDataIn 41
#define evCommDataOut 42
#define evCommCheckOk 43
#define evCommEndProcessing 44
#define evCommSendReply 45
#define evCommEndOfTransmission 46
#define evCommTM 47
#define evCommErrorProcessing 48
```



```c
#define evCommErrorChecking 49
#define evCommTMWatchdog 50

#define TM1000   1000
#define TM2000   2000
#define TMLONG   2000

#define SOFTTIMERLIMIT 20
#define FCY 8000000
#define TCY 125

#define DEBUG1 LATBbits.LATB14
#define DEBUG2 LATBbits.LATB15

#define PIC

#endif
```

```
/
    ******************************************************************************
    *****/
/* FILENAME : hevstypes.h
        */
/
    *-----------------------------------------------------------------------------
    ----*/
/* GOAL     : Defines the standards types names used in HEVs
        */
/
    ******************************************************************************
    *****/
#ifndef   HEVSTYPES_H
#define   HEVSTYPES_H

#define  UINT8  unsigned char       // 8 bits unsigned type
#define  INT8   signed char         // 8 bits signed type
#define  UINT16 unsigned int        // 16 bits unsigned type
#define  INT16  signed int          // 16 bits signed type
#define  UINT32 unsigned long       // 32 bits unsigned type
#define  INT32  signed long         // 32 bits signed type

#endif
```

```c
/*
 * File:   main.c
 * Author: louismayencourt
 *
 * Created on 16 mai 2014, 16:52
 */

#include "xc.h"
#include "hevstypes.h"
#include "defines.h"
#include "adc_driver.h"
#include "dac_driver.h"
#include "analogOut_driver.h"
#include "xf.h"
#include "controller.h"
#include "acquisition.h"
#include "communication.h"

void init(void);

struct Controller cont;
struct Acquisition acqu;
struct Communication comm;

/*****************************************************************************/
/* FUNCTION     : main                                                       */
/* INPUT            : -                                                      */
/* OUTPUT           : -                                                      */
/*---------------------------------------------------------------------------*/
/* COMMENTS     : Called at system power on                                  */
/*****************************************************************************/
int main(void)
{
    init();

    // new object
    Controller_init(&cont,&acqu);
    Acquisition_init(&acqu);
    Communication_init(&comm,&cont);

    // init of the state machine
    XF_pushEvent(evInit, false);

    Event ev;
    while(1)
    { // infinity loop
        // pop and distribut the event
        ev = XF_popEvent(false);
        Controller_SM(&cont,ev);
        Acquisition_SM(&acqu,ev);
        Communication_SM(&comm,ev);
    }

    return 0;
}

/*****************************************************************************/
/* FUNCTION     : init                                                       */
/* INPUT            : -                                                      */
/* OUTPUT           : -                                                      */
```



```c
/*---------------------------------------------------------------------------*/
/* COMMENTS     : initialisation function                                    */
/*****************************************************************************/
void init(void)
{
    // configuration of the interne oscillator
    CLKDIVbits.CPDIV = 1;

    // initialisation XF
    XF_init();

    // initialisation DAC
    dac_init();

    // initialisation ADC
    adc_init();

    // initialisation AnalogOut
    analogOut_init();

    // initialisation UART
    uart_init(9600,2);

    // initialiation for the debug pin
    TRISBbits.TRISB14 = 0;
    TRISBbits.TRISB15 = 0;
    ANSBbits.ANSB14 = 0;
    ANSBbits.ANSB15 = 0;
}
```

```c
/*
 * File:   uart_driver.h
 * Author: louismayencourt
 *
 * Created on 19 mai 2014, 16:18
 */

#ifndef UART_DRIVER_H
#define UART_DRIVER_H

#include <p24FJ128GC006.h>
#include "hevstypes.h"
#include "defines.h"

#ifdef  __cplusplus
extern "C" {
#endif

#define UART_BUFFER_SIZE 50

    // data buffer
    UINT8 uart_data_in[UART_BUFFER_SIZE];
    UINT8 uart_data_out[UART_BUFFER_SIZE];
    UINT8* uart_ptr_write;
    UINT8* uart_ptr_read;
    UINT8* uart_ptr_end_of_frame;

    //prototypes of uart - interface
    void uart_init(UINT16 baud,UINT8 nbrStopBit);
    void uart_enable_interrupt(BOOL ena);
    UINT8 uart_get_frame_in_length(void);
    UINT8* uart_get_in_buffer(void);
    void uart_send_frame(UINT8 length);
    UINT8* uart_get_out_buffer(void);

#ifdef  __cplusplus
}
#endif

#endif  /* UART_DRIVER_H */
```

```c
/*****************************************************************************/
/* FILE           : uart_driver.c                                            */
/* AUTHOR         : Mayencourt Louis                                         */
/*---------------------------------------------------------------------------*/
/* COMMENTS     : uart file for the gazDetector                              */
/*---------------------------------------------------------------------------*/
/* REVISION     : 1.0  - tested with XC16    compiler                        */
/*****************************************************************************/
#include "uart_driver.h"
#include "xf.h"

/
    ****************************************************************************
    *****/
/* FUNCTION     : uart_init
    */
/* INPUT            : baud := the baud rate of the uart communication
    */
/*                    nbrStopBit := the number of stop bit in the
    communication */
/* OUTPUT           : -
    */
/
    *---------------------------------------------------------------------------
    ---*/
/* COMMENTS     : initialisation function
    */
/
    ****************************************************************************
    *****/
void uart_init(UINT16 baud,UINT8 nbrStopBit)
{
    // configure the data in buffer
    uart_ptr_write = uart_data_in;
    uart_ptr_read = uart_data_in;
    uart_ptr_end_of_frame = uart_data_in;

    // configure the operation mode
    U1MODEbits.PDSEL = 0;
    U1MODEbits.STSEL = nbrStopBit-1;
    U1BRG = ((FCY/baud)/16-1);
    U1MODEbits.UARTEN = 1;
    // enable the transmission
    U1STAbits.UTXEN = 1;
    // configure the interrupt
    U1STAbits.UTXISEL1 = 1;
    U1STAbits.UTXISEL0 = 0;

    // configure the PORT in Digital
    ANSBbits.ANSB0 = 0;
    ANSBbits.ANSB1 = 0;

    // Unlock Registers
    __builtin_write_OSCCONL(OSCCON & 0xbf);

    // Assign U1RX To Pin RP0
    RPINR18bits.U1RXR = 0;

    // Assign U1TX To Pin RP1
```



```c
    RPOR0bits.RP1R = 3;

    // Lock Registers
    __builtin_write_OSCCONL(OSCCON | 0x40);

    // Configure the interrupt priority
    IPC2bits.U1RXIP = 4;
    IPC3bits.U1TXIP = 4;

    // Clear the flag
    IFS0bits.U1RXIF = 0;
    IFS0bits.U1TXIF = 0;

    // disable the interrupt
    uart_enable_interrupt(true);

}
/*****************************************************************************/
/* FUNCTION     : uart_enable_interrupt                                      */
/* INPUT            : enable ?                                               */
/* OUTPUT           : -                                                      */
/*---------------------------------------------------------------------------*/
/* COMMENTS     : interrupt enable function                                  */
/*****************************************************************************/
void uart_enable_interrupt(BOOL ena)
{
    if(ena == true)
    {
        // Clear the flag
        IFS0bits.U1RXIF = 0;
        IFS0bits.U1TXIF = 0;

        // enable the interrupt
        IEC0bits.U1RXIE = 1;
        IEC0bits.U1TXIE = 1;
    }
    else
    {
        // disable the interrupt
        IEC0bits.U1RXIE = 0;
        IEC0bits.U1TXIE = 0;
    }
}
/*****************************************************************************/
/* FUNCTION     : uart_send_frame                                            */
/* INPUT            : the number of byte to s                                */
/* OUTPUT           : -                                                      */
/*---------------------------------------------------------------------------*/
/* COMMENTS     : send  function                                             */
/*****************************************************************************/
void uart_send_frame(UINT8 length)
{
    if(length <= UART_BUFFER_SIZE)
    {
        uart_ptr_end_of_frame = uart_data_out+length;
        uart_ptr_write = uart_data_out;

        // Clear the flag
        IFS0bits.U1TXIF = 0;
```

```
            // enable the interrupt
            IEC0bits.U1TXIE = 1;

            U1TXREG = *uart_ptr_write;
            uart_ptr_write++;
        }
    }

    /
        ***************************************************************************
        **/
    /* FUNCTION     : uart_get_frame_in_length
        */
    /* INPUT            : -
        */
    /* OUTPUT           : the number of byte received on the serial
        communication*/
    /*-------------------------------------------------------------------------
        */
    /* COMMENTS     : read length function
        */
    /
        ***************************************************************************
        **/
    UINT8 uart_get_frame_in_length(void)
    {
        UINT8 length;

        length = uart_ptr_read - uart_data_in;
        uart_ptr_read = uart_data_in;

        return length;
    }

    /*************************************************************************/
    /* FUNCTION     : uart_get_in_buffer                                     */
    /* INPUT            : -                                                  */
    /* OUTPUT           : the pointer on the input buffer                    */
    /*-----------------------------------------------------------------------*/
    /* COMMENTS     : get buffer in function                                 */
    /*************************************************************************/
    UINT8* uart_get_in_buffer(void)
    {
        return uart_data_in;
    }

    /*************************************************************************/
    /* FUNCTION     : uart_get_out_buffer                                    */
    /* INPUT            : -                                                  */
    /* OUTPUT           : the pointer on the output buffer                   */
    /*-----------------------------------------------------------------------*/
    /* COMMENTS     : get buffer out function                                */
    /*************************************************************************/
    UINT8* uart_get_out_buffer(void)
    {
        return uart_data_out;
    }

    /*************************************************************************/
    /* FUNCTION     : _U1RXInterrupt                                         */
```



```
    /* INPUT            :                                                   */
    /* OUTPUT           :                                                   */
    /*-----------------------------------------------------------------------*/
    /* COMMENTS     : uart reception ISR                                     */
    /*************************************************************************/
    void _ISR _U1RXInterrupt(void)
    {
        DEBUG1 = 1;
        *uart_ptr_read = U1RXREG;
        if(uart_ptr_read >= (uart_data_in + UART_BUFFER_SIZE) ||
                uart_ptr_read < uart_data_in)
            uart_ptr_read = uart_data_in;
        else
            uart_ptr_read++;

        XF_pushEvent(evCommDataIn,true);
        IFS0bits.U1RXIF = 0;
        DEBUG1 = 0;
    }
    /*************************************************************************/
    /* FUNCTION     : _U1TXInterrupt                                         */
    /* INPUT            :                                                   */
    /* OUTPUT           :                                                   */
    /*-----------------------------------------------------------------------*/
    /* COMMENTS     : uart transmit ISR                                      */
    /*************************************************************************/
    void _ISR _U1TXInterrupt(void)
    {
        if(uart_ptr_write == uart_ptr_end_of_frame)
        {
            U1TXREG = *uart_ptr_write;
            XF_pushEvent(evCommEndOfTransmission,true);
            uart_ptr_write = uart_data_out;
            IEC0bits.U1TXIE = 0;
        }
        else
        {
            U1TXREG = *uart_ptr_write;
            uart_ptr_write++;
        }

        XF_pushEvent(evCommDataOut,true);
        IFS0bits.U1TXIF = 0;
    }
```

```
/*
 *  xf.h
 *  FEMTO-XF
 *
 *  Created by Medard Rieder on 26.08.11.
 *  Copyright 2011 JFAM. All rights reserved.
 *
 *  Modified by Louis Mayencourt on 26.05.14
 *
 */

#ifndef XFDEF
#define XFDEF

#include <p24FJ128GC006.h>
#include "hevstypes.h"
#include "defines.h"


#ifdef __cplusplus
extern "C"
{
#endif

typedef  unsigned char Event;
typedef  unsigned int Time;
typedef  unsigned char TimerID;

typedef enum IRFLAG
{
    IFTMR1
} IRFLAG;

typedef struct Timer
{
    Time tm;
    Event ev;
        TimerID id;
} Timer;

typedef struct HardTimer
{
    Event ev;
        BOOL free;
        TimerID id;
} HardTimer;

#define MAXTIMER 16
#define MAXEVENT 32
#define NULLEVENT 0
#define NULLTIMER 9999
#define NULLID 0
#define MAXHARDTIMER 4
#define TICKINTERVAL 10

typedef struct XF
{
    //this will be the attributes of the femto-xf
    Timer timerList[MAXTIMER];
    Event eventQueue[MAXEVENT];
```



```
        HardTimer hardTimerList[MAXHARDTIMER];
} XF;


//prototypes of xf - interface

void XF_init();
Event XF_popEvent(BOOL inISR);
void XF_pushEvent(Event ev, BOOL inISR);
TimerID XF_scheduleTimer(Time tm, Event ev, BOOL inISR);
TimerID XF_scheduleHardTimer(Time tm,Event ev,BOOL inISR);
void XF_unscheduleTimer(TimerID id, BOOL inISR);
void XF_unscheduleHardTimer(TimerID id, BOOL inISR);
void XF_delay_us(int tm);
//void interrupt XF_ISR(void);
void XF_ISR(void);
inline void LEAVECRITICAL(BOOL inISR);
inline void ENTERCRITICAL(BOOL inISR);

#ifdef __cplusplus
}
#endif

#endif
```

```c
/*
 *  xf.c
 *  FEMTO-XF
 *
 *  Created by Medard Rieder on 26.08.11.
 *  Copyright 2011 JFAM. All rights reserved.
 *
 */

#include "xf.h"

XF theXF;
int current_cpu_ipl;

inline void ENTERCRITICAL(BOOL inISR)
{
    if (inISR == false)
    {
        // disable interrputs
        SET_AND_SAVE_CPU_IPL(current_cpu_ipl, 7);
    }
    else
    {
        //do nothing
    }
}

inline void LEAVECRITICAL(BOOL inISR)
{
    if (inISR == false)
    {
        // enable interrupts
        RESTORE_CPU_IPL(current_cpu_ipl);
    }
    else
    {
        //do nothing
    }
}


void XF_init()
{
    // initialisation of the event queue
    int i;
    for (i=0; i<MAXEVENT; i++)
    {
        theXF.eventQueue[i] = NULLEVENT;
    }

    // initialisation of the timer queue
    for (i=0; i<MAXTIMER; i++)
    {
        theXF.timerList[i].tm = NULLTIMER;
        theXF.timerList[i].ev = NULLEVENT;
        theXF.timerList[i].id = NULLID;
    }

    // initialisation of the hard timer list
    for (i=0; i<MAXHARDTIMER; i++)
```



```c
    {
        theXF.hardTimerList[i].free = true;
    }

    // initialisation of the TIMER1
    T1CON = 0x8010;
    PR1 = 0x2710;

    // enable the TIMER1 interrupt
    IPC0bits.T1IP = 4;
    IFS0bits.T1IF = 0;
    IEC0bits.T1IE = 1;

    // initialisation of TIMER2,3,4,5
    T2CON = 0x0000;
    PR2 = 0x0000;

    T3CON = 0x0000;
    PR3 = 0x0000;

    T4CON = 0x0000;
    PR4 = 0x0000;

    T5CON = 0x0000;
    PR5 = 0x0000;

    // configure the timer2,3,4,5 interrupt
    IPC1bits.T2IP = 4;
    IPC2bits.T3IP = 4;
    IPC6bits.T4IP = 4;
    IPC7bits.T5IP = 4;
}

Event XF_popEvent(BOOL inISR)
{
    Event ev;
    int i;
    ev = NULLEVENT;
    ENTERCRITICAL(inISR);
    if (theXF.eventQueue[0] != NULLEVENT)
    {
        ev = theXF.eventQueue[0];
    }
    for (i=0; i<MAXEVENT-1 && theXF.eventQueue[i]!= NULLEVENT; i++)
    {
        theXF.eventQueue[i] = theXF.eventQueue[i+1];
    }
    theXF.eventQueue[i-1] = NULLEVENT;
    LEAVECRITICAL(inISR);
    return ev;
}

void XF_pushEvent(Event ev, BOOL inISR)
{
    int done;
    int i;
    done = 0;
    ENTERCRITICAL(inISR);
    for (i=0; i<MAXEVENT; i++)
    {
```

```
            if (theXF.eventQueue[i] == NULLEVENT)
            {
                theXF.eventQueue[i] = ev;
                done = 1;
                break;
            }
        }
        //here you could use done to react
        //if eventqueue is full (done == 0)
        LEAVECRITICAL(inISR);
    }

    void XF_delay_us(int tm)
    {
        int i=0;
        int seq = 0;

        seq = (tm*100)/TCY * 10;
        for(i=0; i<seq;i++)
            Nop();
    }

    TimerID XF_scheduleTimer(Time tm, Event ev, BOOL inISR)
    {
        static unsigned char TID = 0;
        int done;
        int i;
        done = 0;

        ENTERCRITICAL(inISR);

        if(tm%TICKINTERVAL != 0)
            tm -= tm%TICKINTERVAL;

        for (i=0; i<MAXTIMER; i++)
        {
            if (theXF.timerList[i].id == NULLID)
            {
                theXF.timerList[i].tm = tm;
                theXF.timerList[i].ev = ev;
                TID++;
                if (TID >= 255)
                {
                    TID = 1;
                }
                theXF.timerList[i].id = TID;
                done = TID;
                break;
            }
        }

        //here you could use done to react
        //if timerlist is full (done == 0)
      //  assert( done != 0 );

        LEAVECRITICAL(inISR);
        return done;
    }

    TimerID XF_scheduleHardTimer(Time tm, Event ev, BOOL inISR)
```



```
    {
        int i;
        int hardTimerID;
        int PR;
        int PS;

        hardTimerID = 0;

        ENTERCRITICAL(inISR);

        for(i=0; i<MAXHARDTIMER; i++)
        {
            if(theXF.hardTimerList[i].free == true)
            {
                theXF.hardTimerList[i].ev = ev;
                theXF.hardTimerList[i].free = false;
                theXF.hardTimerList[i].id = i+1;
                hardTimerID = i+1;

                // calcul the value of PR et PS
                if( tm >= 15000)
                {
                    PS = 1;
                    PR = (tm/TCY) * 125;
                }
                else
                {
                    PS = 0;
                    if(tm <= TCY)
                        PR = (tm*100)/TCY * 10;
                    else
                        PR = (tm/TCY) * 1000;
                }

                break;
            }
        }

        switch(hardTimerID)
        {
            case 1: // allocate the TRM2
                PR2 = PR;
                TMR2 = 0x0000;
                T2CONbits.TCKPS = PS;
                IFS0bits.T2IF = 0;
                IEC0bits.T2IE = 1;
                T2CONbits.TON = 1;
                break;

            case 2: // allocate the TRM3
                PR3 = PR;
                TMR3 = 0x0000;
                T3CONbits.TCKPS = PS;
                IFS0bits.T3IF = 0;
                IEC0bits.T3IE = 1;
                T3CONbits.TON = 1;
                break;

            case 3: // allocate the TRM4
                PR4 = PR;
```

```
                TMR4 = 0x0000;
                T4CONbits.TCKPS = PS;
                IFS1bits.T4IF = 0;
                IEC1bits.T4IE = 1;
                T4CONbits.TON = 1;
                break;

            case 4: // allocate the TRM5
                PR5 = PR;
                TMR5 = 0x0000;
                T5CONbits.TCKPS = PS;
                IFS1bits.T5IF = 0;
                IEC1bits.T5IE = 1;
                T5CONbits.TON = 1;
                break;

            default:
                break;
        }

        LEAVECRITICAL(inISR);

        return hardTimerID;
    }

    void XF_unscheduleTimer(TimerID id, BOOL inISR)
    {
        int done;
        int i;
        done = 0;

        ENTERCRITICAL(inISR);
        for (i=0; i<MAXTIMER; i++)
        {
            if (theXF.timerList[i].id == id)
            {
                theXF.timerList[i].tm = NULLTIMER;
                theXF.timerList[i].ev = NULLEVENT;
                theXF.timerList[i].id = NULLID;
                done = 1;
                break;
            }
        }
        //here you could use done to react
        //if timerID was not found (done == 0)
        LEAVECRITICAL(inISR);
    }

    void XF_unscheduleHardTimer(TimerID id, BOOL inISR)
    {
        int done;
        int i;
        done = 0;

        ENTERCRITICAL(inISR);

        for (i=0; i<MAXHARDTIMER; i++)
        {
            if (theXF.hardTimerList[i].id == id)
            {
```



```
                theXF.hardTimerList[i].free = true;
                theXF.hardTimerList[i].ev = NULLEVENT;
                theXF.hardTimerList[i].id = NULLID;

                switch(id)
                {
                    case 1:
                        T2CONbits.TON = 0;
                        IFS0bits.T2IF = 0;
                        IEC0bits.T2IE = 0;
                        TMR2 = 0x0000;
                        break;

                    case 2:
                        T3CONbits.TON = 0;
                        IFS0bits.T3IF = 0;
                        IEC0bits.T3IE = 0;
                        TMR3 = 0x0000;
                        break;

                    case 3:
                        T4CONbits.TON = 0;
                        IFS1bits.T4IF = 0;
                        IEC1bits.T4IE = 0;
                        TMR4 = 0x0000;
                        break;

                    case 4:
                        T5CONbits.TON = 0;
                        IFS1bits.T5IF = 0;
                        IEC1bits.T5IE = 0;
                        TMR5 = 0x0000;
                        break;

                    default:
                        break;
                }

                done = 1;
                break;
            }
        }
        //here you could use done to react
        //if timerID was not found (done == 0)
        LEAVECRITICAL(inISR);
    }

    void XF_decrementAndQueueTimers()
    {
        int i;
        for (i=0; i<MAXTIMER; i++)
        {
            if (theXF.timerList[i].id != NULLID)
            {
                theXF.timerList[i].tm-=TICKINTERVAL;
                if (theXF.timerList[i].tm ==0)
                {
                    XF_pushEvent(theXF.timerList[i].ev, true);
                    XF_unscheduleTimer(theXF.timerList[i].id, true);
                }
```

```
            }
        }

        //here you could use done to react
        //if timerID was not found (done == 0)
    }

    void XF_QueueHardTimer(TimerID id)
    {
        int i;
        for(i=0; i<MAXHARDTIMER; i++)
        {
            if(theXF.hardTimerList[i].id == id)
            {
                XF_pushEvent(theXF.hardTimerList[i].ev,true);
                XF_unscheduleHardTimer(theXF.hardTimerList[i].id,true);
                break;
            }
        }

        //here you could use done to react
        //if timerID was not found (done == 0)
    }

    // timer ISR
    void _ISR _T1Interrupt(void)
    {
        XF_decrementAndQueueTimers();
        IFS0bits.T1IF = 0;
    }

    void _ISR _T2Interrupt(void)
    {
        XF_QueueHardTimer(1);
        IFS0bits.T2IF = 0;
    }

    void _ISR _T3Interrupt(void)
    {
        XF_QueueHardTimer(2);
        IFS0bits.T3IF = 0;
    }

    void _ISR _T4Interrupt(void)
    {
        XF_QueueHardTimer(3);
        IFS1bits.T4IF = 0;
    }

    void _ISR _T5Interrupt(void)
    {
        XF_QueueHardTimer(4);
        IFS1bits.T5IF = 0;
    }
```

Ernst-Abbe-Fachhochschule Jena
Hochschule für angewandte Wissenschaften

Hes·so VALAIS WALLIS

Anhang G

Test Protokoll

Protocole de tests

Projet :	Prd Détecteur de Gaz
Nom :	Mayencourt
Prénom :	Louis
date :	11 juillet 2014

Alimentation

statut	mesure	remarques
✓	+5[V]	OK
✓	-5[V]	OK
✓	+3.3[V]	OK
✓	+12[V]	consomation 20 [mA]
✓	-12[V]	consamation 30 [mA]

Références

statut	mesure	remarques
✓	+1.25[V]	OK
✓	+2.5[V]	OK
✓	+30[mV]	OK

Processeur

statut	mesure	remarques
✓	programmation	connecteur pickit pin 6 sur la flêche

CDA

statut	mesure	remarques
✓	sortie -8.2 → 8.2 [V]	OK

CAD

statut	mesure	remarques
✓	entrée 0 → 5 [V]	OK
✓	bon comportement impulsionnelle	changement de la Capacité à 470 [pF]
✓	changement des valeurs de sortie	OK

FTDI

statut	mesure	remarques
✓	transmission avec le PC	OK

Programme

statut	mesure	remarques
✓	interface CAD	OK
✓	interface CDA	tension de sortie : +/-
✓	interface RS-232	OK
✓	sortie analogique	tension de sortie : 0 -> 3.3 [V]