HOCHSCHULE KONSTANZ TECHNIK, WIRTSCHAFT UND GESTALTUNG
UNIVERSITY OF APPLIED SCIENCES

# Teamprojekt Modulare Plattform für intelligente IoT-Sensoren

**Konstanz den, 25. Februar 2017**


| | |
|---|---|
| **Team:** | Fabian Gendusa & Thomas Gnädig |
| **Zeitraum:** | WS 16/17 |
| Fachbetreuer: | I. Schoppa |
| Fakultät: | Informatik |
| Studiengang: | Angewandte Informatik |

# 1

# Das Projekt

## 1.1 Projekt Beschreibung

Im Rahmen des Studiums ist ein mit 12 ECTS gewichtetes Teamprojekt vorgesehen. Das Projekt IoT Sensor kombiniert drei selbst erstellte Platinen zu einem funktionierenden System.

**Das $\mu$ - Controller-Board**

Auf der Hauptplatine wird ein Mikroprozessor der MSP430FR Familie und drei PMOD Schnittstellen angebracht. Über die PMOD Stecker soll mit den anderen Boards kommuniziert werden.

**USB-UART-Interface**

Die Kommunikationsplatine wird über USB mit dem PC verbunden. Diese wandelt das differenzielle USB-Signal, mithilfe des FT231X, in ein serielles UART-Signal um. Das umgewandelte Signal wird dann über einen PMOD Stecker via UART an das Main-Board weitergeleitet. Auf der Platine befindet sich auch eine galvanische Trennung, welche den Stromkreis des Computers von dem des Mikrocontrollers trennt. Hierfür ist der ADUM1401 verantwortlich, welcher das Signal induktiv in den anderen Stromkreis überträgt.

**7-Segmentanzeige**

Die Anzeige Platine stellt eine 7 Segmentanzeige zur Verfügung, welche die Darstellung von Messwerten übernehmen könnte. Hierzu dient der AS1108 welcher die Übersetzung einer Zahl auf die Sieben Segmentanzeige übernimmt. Die Zahlen werden über SPI vom Mainboard erhalten.

Mit der Sensorplatine (nicht teil des Projektes) wird ebenfalls über SPI kommuniziert und

liefert Messwerte an das Main-Board. Diese kann bei Bedarf bereits Fertig gekauft werden.

## 1.2 Arbeitsschritte

### 1.2.1 Bauteilsuche

Nachdem die Vorgabe vollständig ist, kann mit der Bauteilsuche begonnen werden. Einige Teile waren auch bereits vorgegeben. Kriterien beim auswählen der Bauteile waren:

- Funktion, - Angaben müssen erfüllt werden
- Verfügbarkeit - Teile müssen einzeln bestellbar und auch in Zukunft noch Angeboten werden
- Preis

### 1.2.2 Platinen Design

Nachdem alle Teile gefunden sind und auch alle Datenblätter vorliegen, kann mit dem Design der Platine begonnen werden. hierzu wurde das Tool Pulsonix verwendet. Nachdem der Schaltplan entworfen war, konnte das Platinen-Layout erstellt werden. Die größte Hürde hierbei war das Konfigutieren des korrekten Rasters für Bauteile und Vias. Zu beachten war auch die Positionierung der Bauteile. Manche Bauteile mussten möglichst nah an anderen Bauteilen angebracht werden. So beim FT231X, wo ein differentielles Signal anliegt. Auch mussten an den PMOD Steckern zwei Kondensatoren angebracht werden um Störsignale welche beispielsweise beim Einstecken entstehen auf die Masse abzuleiten.

### 1.2.3 Hardware bestücken

Die Hardware wurde bei `www.digikey.com` bestellt und wurde innerhalb kurzer Zeit geliefert. Um die Platinen zu bestellen, mussten zunächst Gerberfiles erstellt werden, welche dann an Q-print electronic GmbH geschickt wurden. Zum Bestücken der Platine war Teamarbeit gefragt, während einer das Bauteil möglichst exakt auf den kleinen Pads positioniert muss der Andere das erste Beinchen fest löten. Die wohl größte Herausforderung hierbei war das befestigen des FT321X auf der Kommunikationsplatine. Nachdem diese gemeistert war, stellten die anderen Bauteile keine größeren Probleme mehr dar. Als Änderung für neue Platinen wäre bei der Anzeigenplatine ein Größerer $RSET$ Widerstand einzubauen, da der

eingebaute 10 $k\Omega$ etwas zu klein dimensioniert ist. Wenigstens 11,35 $k\Omega$ siehe Datenblatt AS1108 Seite 9.

### 1.2.4 Softwareentwicklung

Es musste nicht nur ein Treiber für den MSP geschrieben werden um über UART und SPI mit unseren Boards zu kommunizieren, sondern auch ein Treiber, welcher die Kommunikation zwischen PC und Kommunikationsplatine übernimmt. Die genauere beschreibung folgt im nächsten Kapitel.

# 2

# Entwickelte-Software

## 2.1 Python

Für die Programmierung, um auf das Kommunikationsboard zugreifen zu können, haben wir uns für Python entschieden, da es hier relativ einfache und schnelle wege gibt um c librarys einzubinden und zu benutzen. Wir greifen hier auf die vom Hersteller bereitgestellte library "ftd2xxßurück.

### 2.1.1 MSP430connect.py

Dieses Script dient zur Übertragung eines durch CCS erstellten Programms mithilfe des USB-UART-Konverters auf den Microcontroller. Es kann mit dem Parameter -p [Dateipfad] ausgeführt werden, wobei dann das Programm heruntergeladen wird. Wird es ohne Angabe von Parametern gestartet öffnet sich ein textuelles Benutzerinterface. Mit dem Parameter h kann eine kleine Hilfe zum Programm angezeigt werden. Wie die Das Programm zum herunterladen auf den Microkontroller mit CCS erstellt werden muss, wird auch in dieser beschrieben. Das Script verwendet unseren MSP430.py Treiber, welcher einfache Methoden zum öffnen eines Devices, lesen und schreiben sowie zurücksetzen dieses Devices anbietet. Diese Methoden können für weitere Programme genutzt werden.

### 2.1.2 Übersicht Dateien

| Dateiname | Beschreibung |
|---|---|
| FT_declarations.py | Enthält Deklarationen für den Wrapper des FT2XX |
| FT_Device.py | Wrapper-Klasse für die Kommunikation mit dem FT2XX. |
| MSP430.py | Klasse, zur Kommunikation mit dem Bootstraploader. Bietet Funktionen zum Schreiben, Lesen der seriellen Schnittstelle. Weiter auch die Funktion, um ein Programm auf den Microkontroller zu laden. |
| ProgrammContainer.py | Hilfsklasse zum Parsen und Speichern des Programms, das auf den Mikrokontroller zu laden ist. |

## 2.2 C-Code

### 2.2.1 Einstiegs Codes

Da wir uns das erste mal mit CCS und dem MSP430FR befassen, haben wir uns zunächst ein paar simple Anwendungen überlegt um ein Gefühl für die Umgebung zu bekommen. Hierzu haben wir vor allem die auf dem Experimentierboard angebrachten LEDs als Ausgabe verwendet um möglichst schnell und einfach ein Feedback zu erhalten. Eine einfache Anwendung war das erstellen eines Lauflichts, welches über Tastendruck zusätzlich beeinflusst werden konnte (CD\Code\runled.c). Auch den auf dem Board eingebauten Wärmesensor haben wir ausgelesen und dessen wert direkt an die LEDs angelegt (CD\Code\adc.c). Dass sich die Zahl beim auflegen des Fingers auf den Sensor verändert hat war ein kleines aber feines Erfolgserlebnis.

### 2.2.2 UART

Für die Kommunikation von UART haben wir einen kleinen Treiber geschrieben, welcher zum initialisieren von maximal 2 UART Schnittstellen verwendet werden kann, eine Funktion um eine Zeichenkette zu lesen oder schreiben wird auch bereit gestellt (CD\Code\UART\uart.c\h).

### 2.2.3 SPI

Auch für die beiden SPI Schnittstellen wurde ein Treiber geschrieben, dieser ist ebenfalls auf der CD vorhanden(CD\Code\SPI\spi.c\h). Ein Beispielcode zum ansprechen der Siebensegmentanzeige befindet sich ebenfalls in diesem Ordner, hier ist es uns bis heute leider nicht gelungen,

den Treiberbaustein korrekt zu konfigurieren, die Funktionalität der SPI Schnittstelle ist aber bereits geprüft und funktioniert wie in diesem Beispielcode beschrieben.

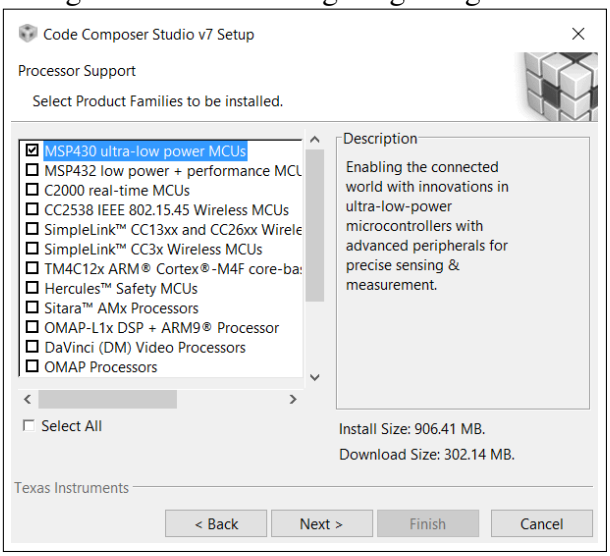
# 3

# Hilfsmittel/Arbeitsumgebung

Zum Erstellen des Schaltplans und der Gerberfiles haben wir das auf den Laborrechnern bereitgestellte Tool Pulsonix verwendet.
Die Python Programmierung haben wir im Notepad++ durchgeführt.
Um den MSP430FR zu beschreiben griffen wir auf das vom Hersteller bereitgestellte Tool CCS zurück.

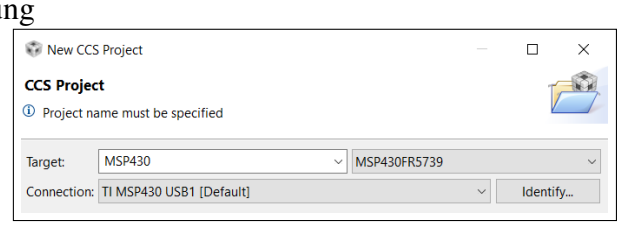
### 3.0.1 Installation und anwendung des CCS

Die aktuellste Version des CCS kann unter folgender URL bei ti direkt heruntergeladen werden `http://processors.wiki.ti.com/index.php/Download_CCS`.
Nach dem Ausführen der css_setup.exe sollte im Schritt "processor support"wenigstens die Familie MSP430 wie in folgendem Screenshot gezeigt ausgewählt werden.




Nach der Installation des Programms kann ein Workspace angelegt oder ein bereits

bestehender genutzt werden. Beim Erstellen eines neuen Projektes sollte darauf geachtet werden, dass auch der richtige Microcontroller ausgewählt ist. Folgender Screenshot zeigt die korrekte Einstellung




Durch diese Auswahl werden bereits die richtigen Bibliotheken eingebunden. Hier kann auch ein kleines Beispielprogramm erstellt werden, welches eine LED blinken lässt. Die weitere Programmierung ist wie in herkömmlichen Entwicklungsumgebungen. Eine nette und hilfreiche Erweiterung ist der Debugmodus, welcher uns schon oft weiter geholfen hat. Hier kann das Programm auf dem MSP430 direkt laufengelassen, angehalten und fortgesetzt werden.
Ein bereits fertiger Workspace-Ordner mit allen hier aufgelisteten Programme ist auch auf der CD zu finden.

### 3.0.2 Erstellen des Download-Files

Ein Programm, das mit dem MSP430Connect Skript auf den Mikrokontroller geladen werden soll muss in einem bestimmten Dateiformat vorliegen. Zur Erstellung dieser Datei müssen im Code Composer Studio (CCS) zuerst einige Einstellungen vorgenommen werden.
1. unter ’Build’ Configuration auf Release stellen
2. unter ’MSP430 Hex Utility’ haken bei Enable MSP430 Hex Utility aktivieren
3. unter ’Output Format Options’ das Output format auf Output TI_TXT hex Format setzen
4. Build Release ausführen, ’projektname’.txt liegt jetzt im workspace
Diese ’projektname’.txt muss dann dem MSP430Connect Skript als Parameter übergeben werden.