



Verifikation der Software von Sensor Lösungen bei der ams AG

Stephan Puri-Jobi

29.06.2016

Confidential © ams AG 2016

About ams

Shaping the world with sensor solutions

Focus

Designing and manufacturing advanced analog sensor solutions

Markets: communications, consumer, industrial, automotive and medical

Solutions: intelligent light sensors, RFID/NFC, chemical sensors, active-noise-cancelling ICs, position sensors, ultra-low power management, and more

Standard products & custom solutions (ASICs & foundry services)

People:

More than 2,100 employees in 20 countries

18 design centers

20 sales offices

30+ channel partners

Manufacturing

IDM with 30+ years of experience

Advanced processes: CMOS, HV-CMOS and SiGe, 3D TSV

Certified for automotive & medical production

Full service foundry including packaging and testing options

8 inch wafer fab in Austria (180k wspa)

Test facility in Calamba, Philippines

Strong relationships with global manufacturing partners

Financials

Revenues 2015 EUR623m/\$691m
(2014: EUR 464m/\$614m)

Revenues Q4 2015 EUR147m/\$161m
(Q4 2014: EUR139m/\$141m)

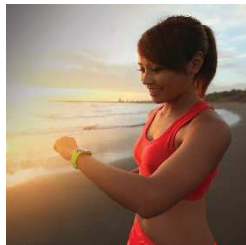
The world of sensors

Smart Phones & Tablet



- Ambient light, color & proximity sensors
- Gesture recognition
- NFC-based contactless payment solutions
- Environmental sensors
- Active Noise Cancellation
- Spectral sensors

Wearables



- Biosensors, heart rate monitoring
- Power management
- NFC-based contactless payment solutions
- Active Noise Cancellation
- Environmental sensors

Smart Home & Buildings



- Gas sensors
- Temperature sensors
- Smart light sensors
- Humidity sensors
- Pressure sensors
- Flow sensors
- Lightning sensors



Automotive



- Position sensors
- Sensors for advanced driver assistance
- Air quality sensors
- Hydrogen sensors

Industrial



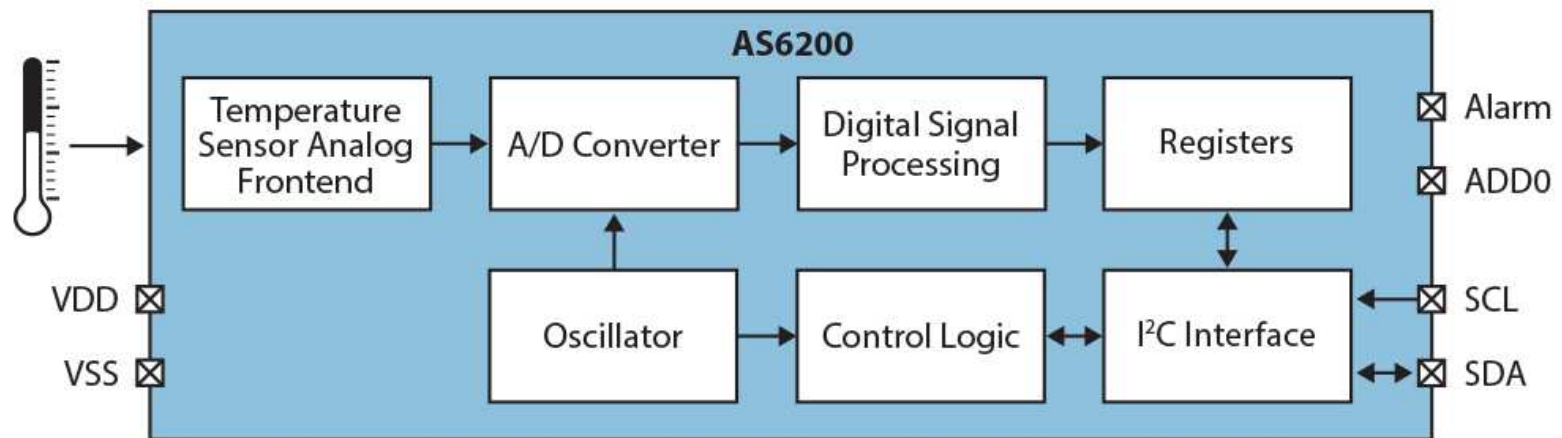
- Position sensors
- CMOS sensors for machine vision and drones
- Industrial/building automation
- Flow sensors (Heat, Water, Gas Metering)
- NFC Sensor Tags

Medical

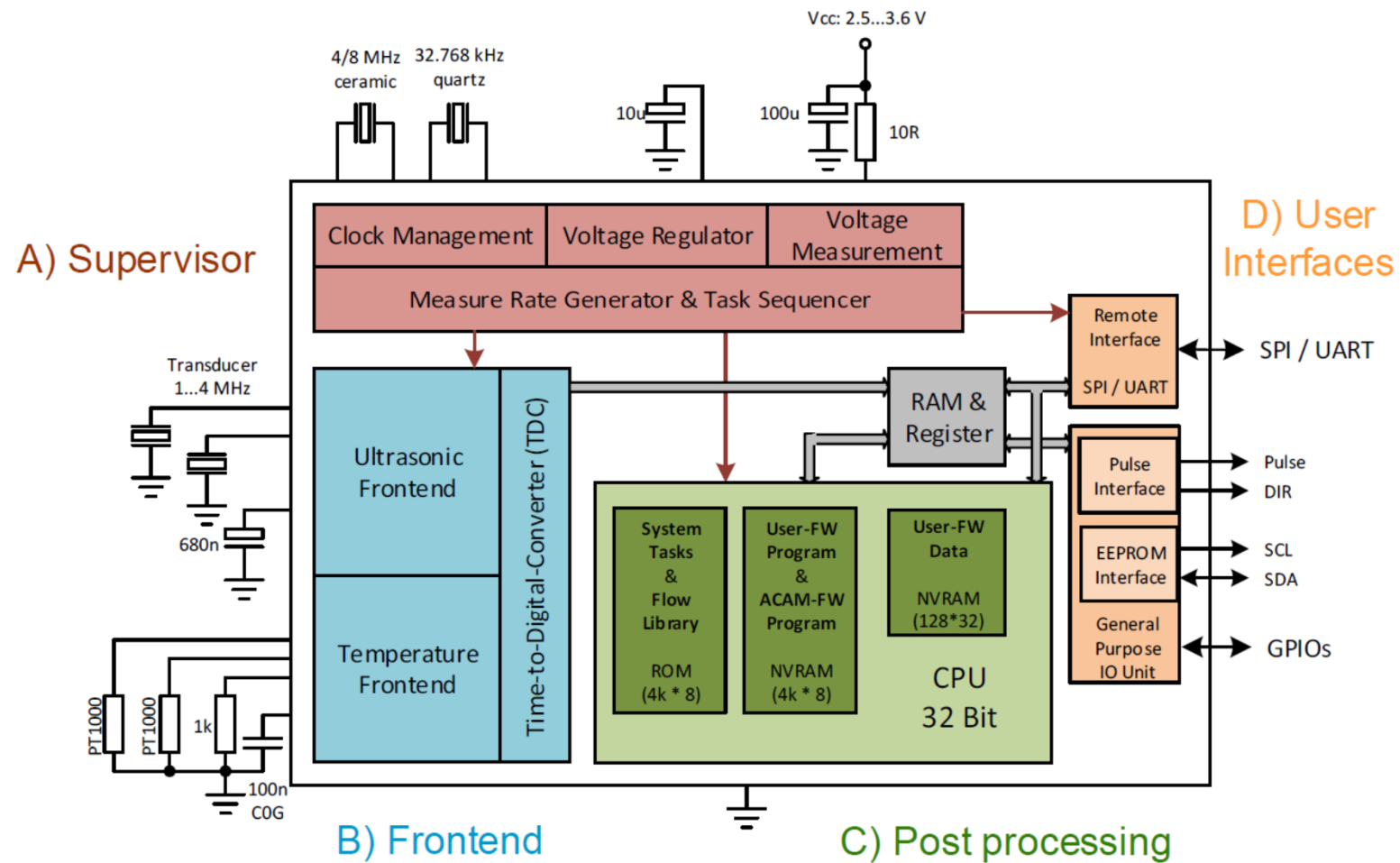


- Image Sensors for:
- Computed tomography
- Digital x-ray
- Ultrasound
- Surgical robots
- CMOS image sensors for endoscopy (miniature cameras)

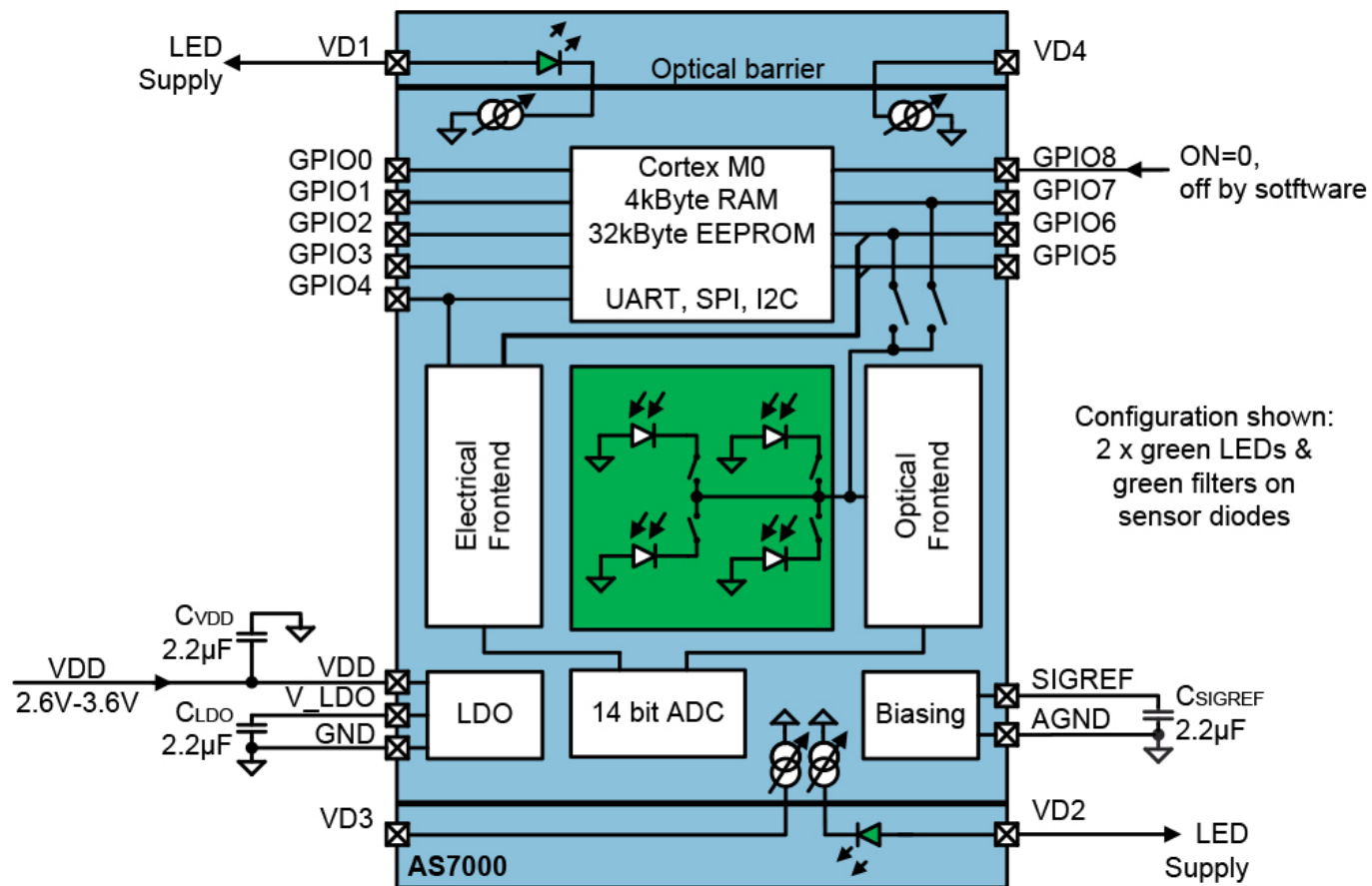
Temperatur Sensor



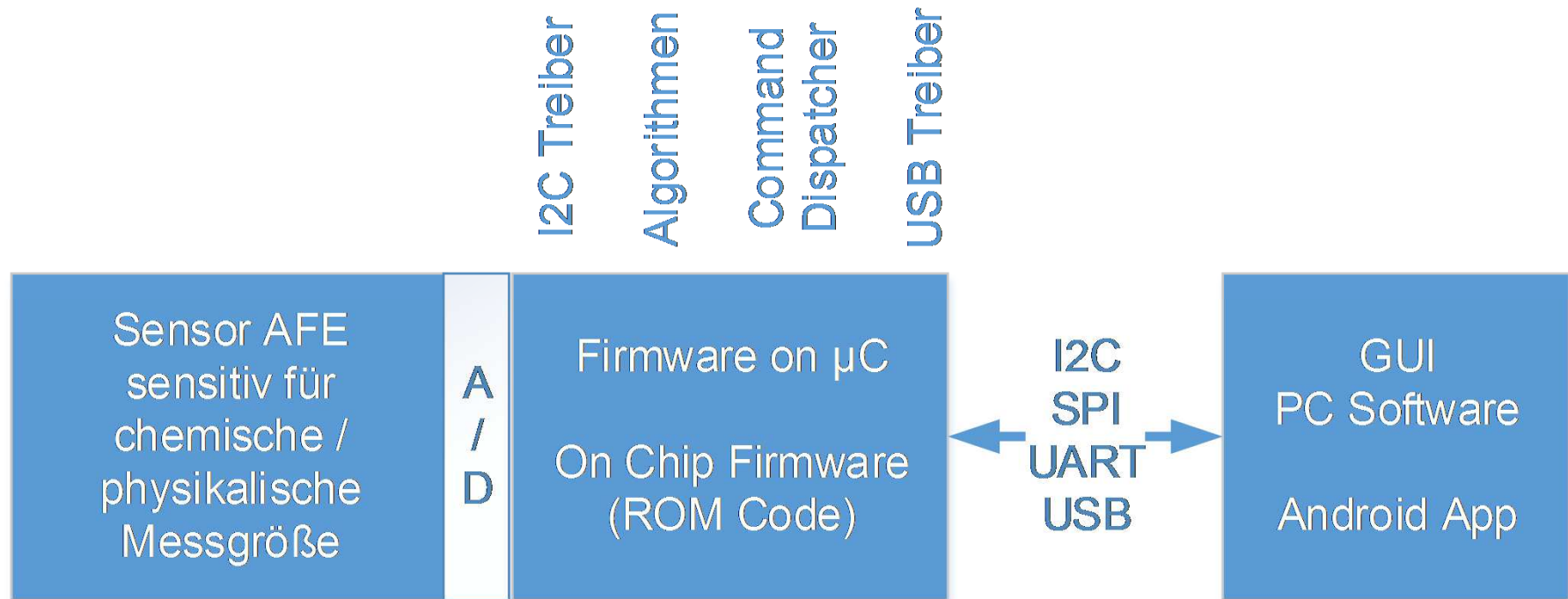
Durchfluss Sensor Lösung



Biosensor Lösung



Abstrahierte Sensor Lösung System Überblick



Eine abstrahierte Sensor Lösung aus Software Sicht

Unterscheidung der SW in

- **HW nahen / HW abhängigen Code (HW Treiber)**
 - Registerzugriff auf Sensor Hardware
- **Zeitkritischen Code**
 - Steuerung von Heizungselementen
 - Kommunikationsprotokolle
- **Zielplattform abhängigen / optimierten Code**
 - Algorithmen mit spezieller Mathematik-Bibliothek
- **Völlig unabhängigen Code**
 - SW die am PC läuft

Aber... wofür überhaupt so viel Software?

Komplexe Informationen aus einfachen Messungen

- **Kombination mehrerer Messwerte**
- **Kombination Messwerte mehrerer Sensoren**
- **Ableitung der Messwerte über die Zeit**
- **Filterung**
- **Mustererkennung**

Erkennung von Fehlmessungen

- **Mittelwertberechnung**
- **Plausibilitätscheck**

Zu Demonstrationszwecken

- **GUI zur optischen Aufbereitung der Messwerte**
- **Integration in den Linux Kernel für Android Applikationen**

Und was kann da alles schief gehen?

Potentielle Fehler / Error in mission-critical Software

Reine SW Probleme

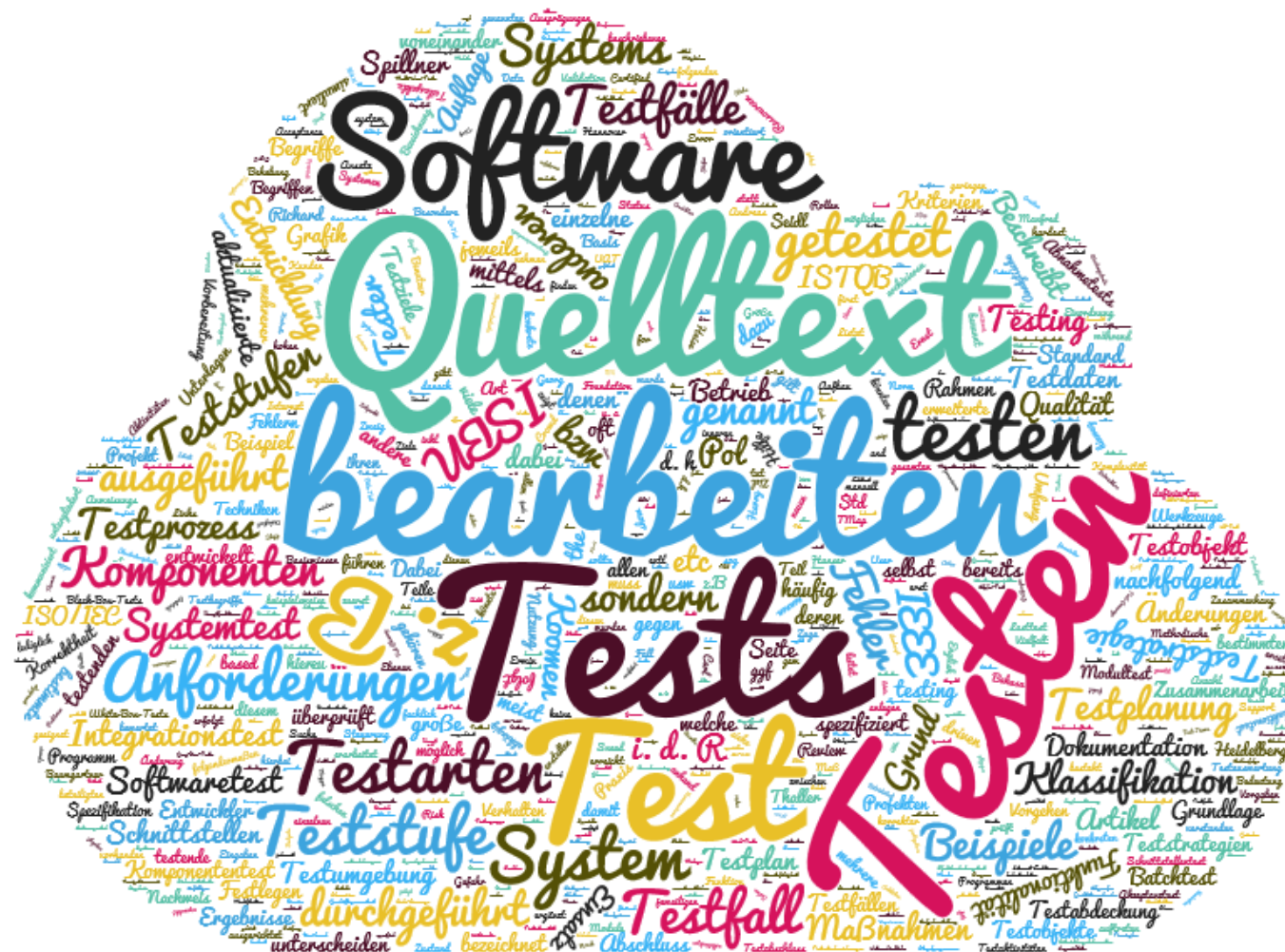
- Rundungsfehler, signed / unsigned Zuweisungen
- Timeouts
- Interrupt Überlast
- RAM Speicherzugriffsfehler / Pufferüberläufe, Stack / Heap Verwendung
- Ausführungszeit von ISRs / Algorithmen

Probleme von Embedded Systemen (die nicht unbedingt SW Fehler sind, aber beim SW Design berücksichtigt werden müssen)

- Leistungsaufnahme (wenn alle LEDs auf einmal eingeschaltet werden)
- Rückkopplung (PWM stört Sensor / Kommunikationskanal)

Softwaretest

Was ist das richtige für mein Projekt / Produkt?



Verifikation der Software bei der ams AG

System Tests oder Komponenten Tests, oder doch beides?

Akzeptanz Tests um zu zeigen dass das vom Kunden gewünschte Verhalten existiert

System Tests damit man sieht dass wirklich das gesamte System funktioniert, besonders auch im negativ Fall. Iteratives herunter brechen in autonome, definierte Sub-Systeme (Wiederverwendbarkeit)

Wenn hier schon alles funktionieren würde bräuchte man keine Komponenten Tests

Wenn Akzeptanz Tests fehlschlagen sieht man nicht wo das Problem liegt, nur dass man eines hat

Gute Tests sollten bereits sehr genau eingrenzen wo das Problem liegt damit man sich Zeit beim Debuggen spart

Komponenten Tests als Sicherheit für den Entwickler, und damit man im Fehlerfall schneller reagieren kann

Anforderungen

Was ist das gewünschte Verhalten?

Was wird gebraucht? Was soll gebaut werden? Und wann sind wir damit fertig?

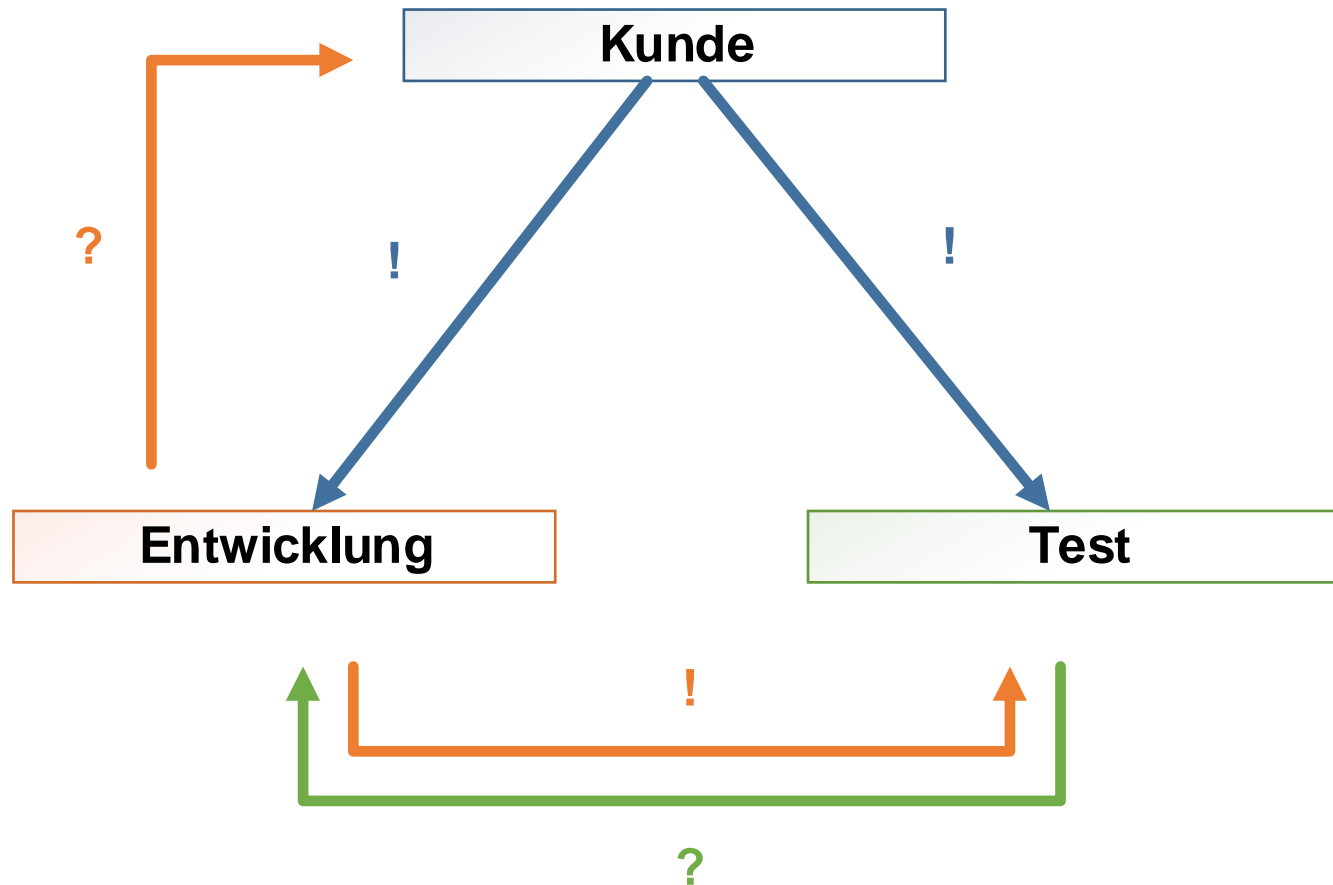
Oft schwierig, besonders bei „internem Auftrag“

Anforderungen müssen vom Kunden kommen, Akzeptanz Tests helfen diese auf ein vernünftiges Detaillevel zu bringen.

Für NFC Produkte gibt es vom NFC Forum eine Menge spezifizierter Tests die erfüllt werden müssen.

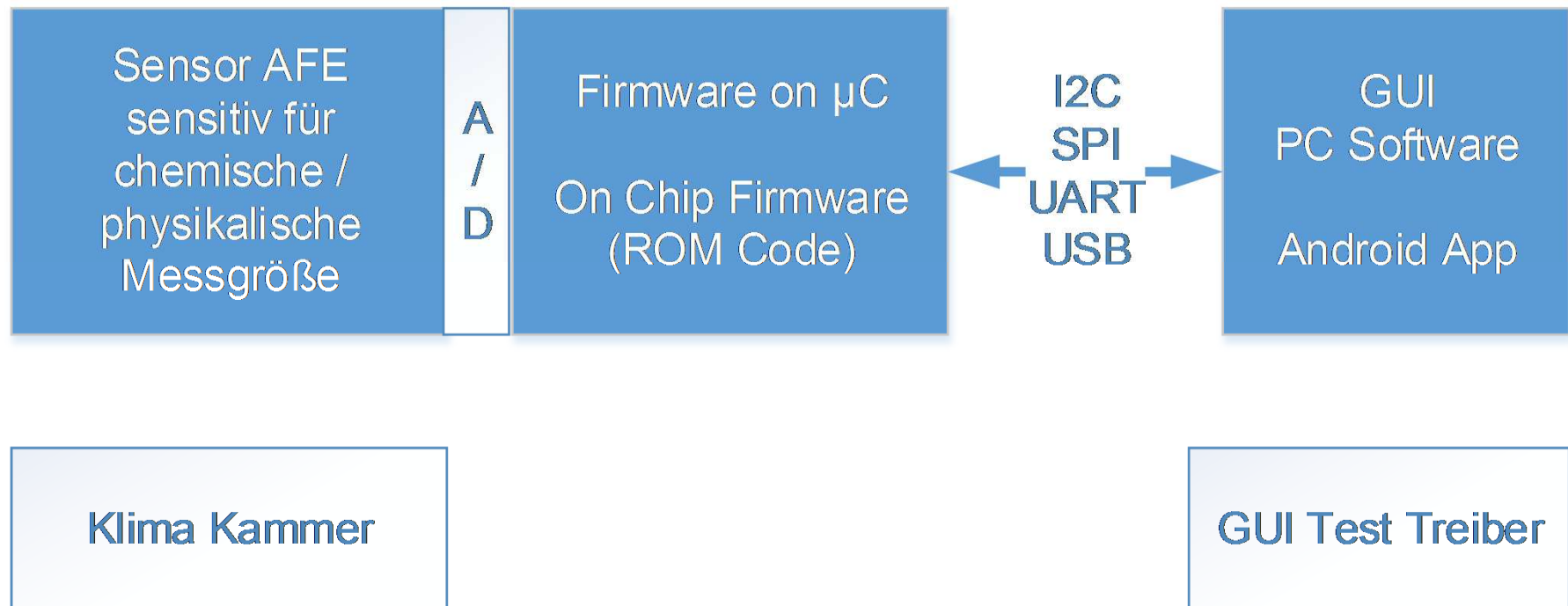
ISO26262 für uns (noch) keine Anforderung

Verfeinerung der Anforderungen



Akzeptanz Tests

Nur hier steckt die ganze Wahrheit



Akzeptanz Tests über das gesamte System von der chemischen / physikalischen Stimulanz des Sensors bis zur graphischen Ausgabe

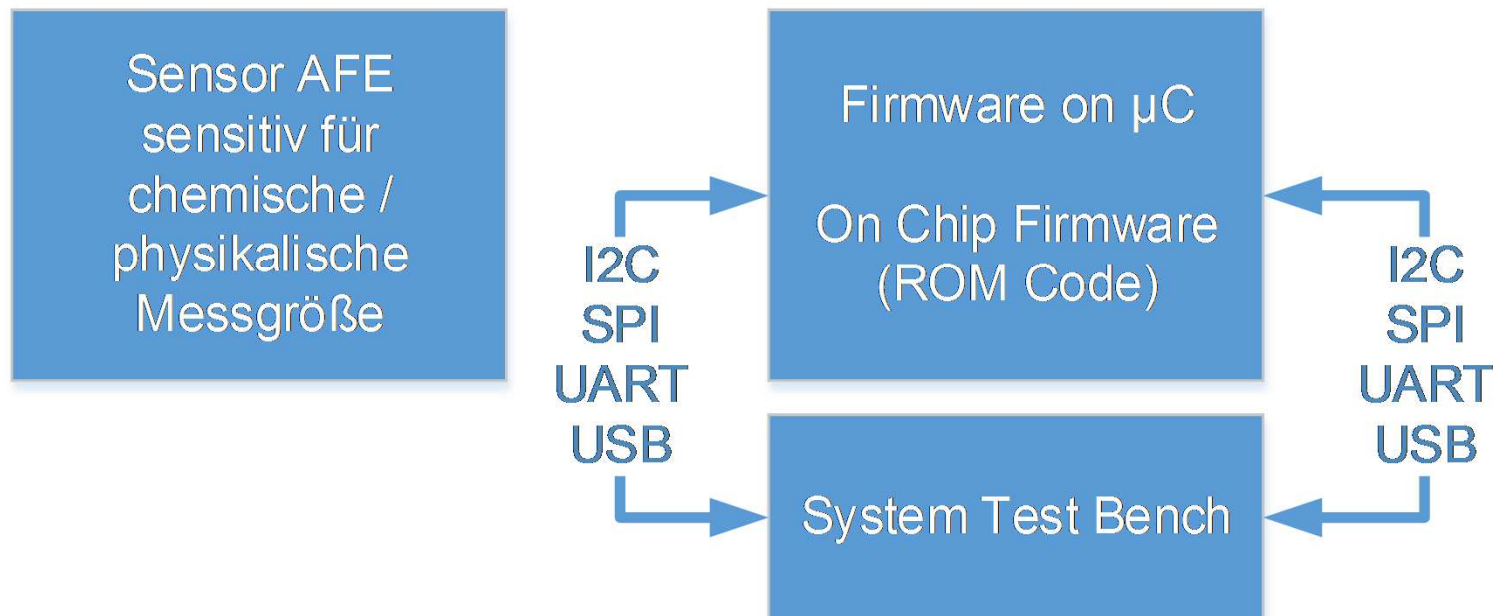
Unser Ansatz

Divide and Conquer

Off Target Komponenten Tests
(Verifikation der Funktionalität
von Algorithmen)

On Target Komponenten Tests
(Verifikation des richtigen
Verhaltens auf der Zielplattform)

HW Evaluation



Komponenten Tests

Hilfreich für:

- Entwickler zum Verifizieren des Algorithmus
- Funktionaler Check von Funktionen

Möglich:

- Check auf Buffer overflow, ja für einen Buffer in der Funktion, nein für systemweite Buffer

Unzureichend für:

- Race Conditions
- Zeitabhängiges Verhalten
- Fehlerweitergabe durch verschiedene SW Schichten

Oftmals gar nicht möglich am finalen Produkt:

- Keine Möglichkeit mehr die SW updaten (ROM)
- Keinen Zugriff auf SW (kein JTAG)

System Test

Verifiziert das Verhalten gegen die Anforderungen

SW läuft auf tatsächlicher Zielplattform

Können sehr komplex werden

Gutes System Verständnis wichtig um wirklich das zu testen was man testen möchte

Gut dokumentierte Anforderungen wichtig damit man weiß wie sich das System dann überhaupt verhalten soll, besonders im Fehlerfall oft unzureichend definiert

Zeigt auch gleich Benutzbarkeit des System weil es auf verschiedenste Arten „benutzt“ wird

Modulare System Tests

Design von System Tests für

- unterschiedliche HW Anbindungen (I2C, SPI, UART)
- unterschiedliche Angriffspunkte (Mock für Sub-Systeme)
- Unterschiedliche Entwicklungszustände des Sensors (Steuerung der Klima Kammer vs. direkter Register zugriff um Wert zu simulieren)

Test Logik getrennt von Daten Ein- / Ausgabe zum SUT

Vom Umgang mit System Test Umgebungen

Qualitätsanforderung für Test Umgebung sollte sehr hoch sein.

Wenn man der Test Umgebung nicht vertrauen kann wird sie nicht verwendet werden bzw. man wiegt sich in falscher Sicherheit.

Tests für die Testfälle um „false negative“ zu vermeiden.

Tests für das Test Environment um „false positive“ zu vermeiden. „Bemerkt“ der Test das mögliche Fehlverhalten überhaupt?

Herausforderungen mit System Tests

Timing:

- Unmöglich die ganze System Test Umgebung im Griff zu haben (Windows), besonders für genaue Messungen (Ausführungszeit)
- Gewährleisten dass viele Daten mit einer hohen Datenrate übertragen werden (UART)
- Gewährleisten dass „sofort“ und nicht nur „so schnell wie möglich“ geantwortet wird

Das Problem wird immer schlimmer je weiter man vom SUT weg ist (direkt der Host Controller der das steuert oder ein virtueller PC) -> dezidierte HW die vorprogrammiert wird und dann im Test autonom arbeitet

Herausforderungen mit System Tests

Testorakel:

- Es gibt oft kein hartes Pass / Fail Kriterium
- Tests dürfen ein gewisse Anzahl fehlschlagen (Stress Tests NFC)
- Ein Systemverhalten ändert sich über Zeit (zu früh -> +/- OK -> zu spät)
- Algorithmus nähert sich an (Toleranz 10% / 5% / 3%)

Man weiß erst nach einer Reihe von Tests und deren Nachbearbeitung dass die Software den Qualitätsanforderungen entspricht

Herausforderungen mit System Tests

Multi Threading:

- Korrektes Synchronisieren der Threads
- Verhindern von Deadlocks in Testfall bei einem Fail
- Assertion Handling im Thread

Test Automatisierung



Jenkins

- Möglichkeit zur Aktualisierung der SW am SUT notwendig
- Möglichkeit zum HW Reset des SUT nach einem Fail



Vielen Dank

**Bitte besuchen Sie unsere Webseite
www.ams.com**