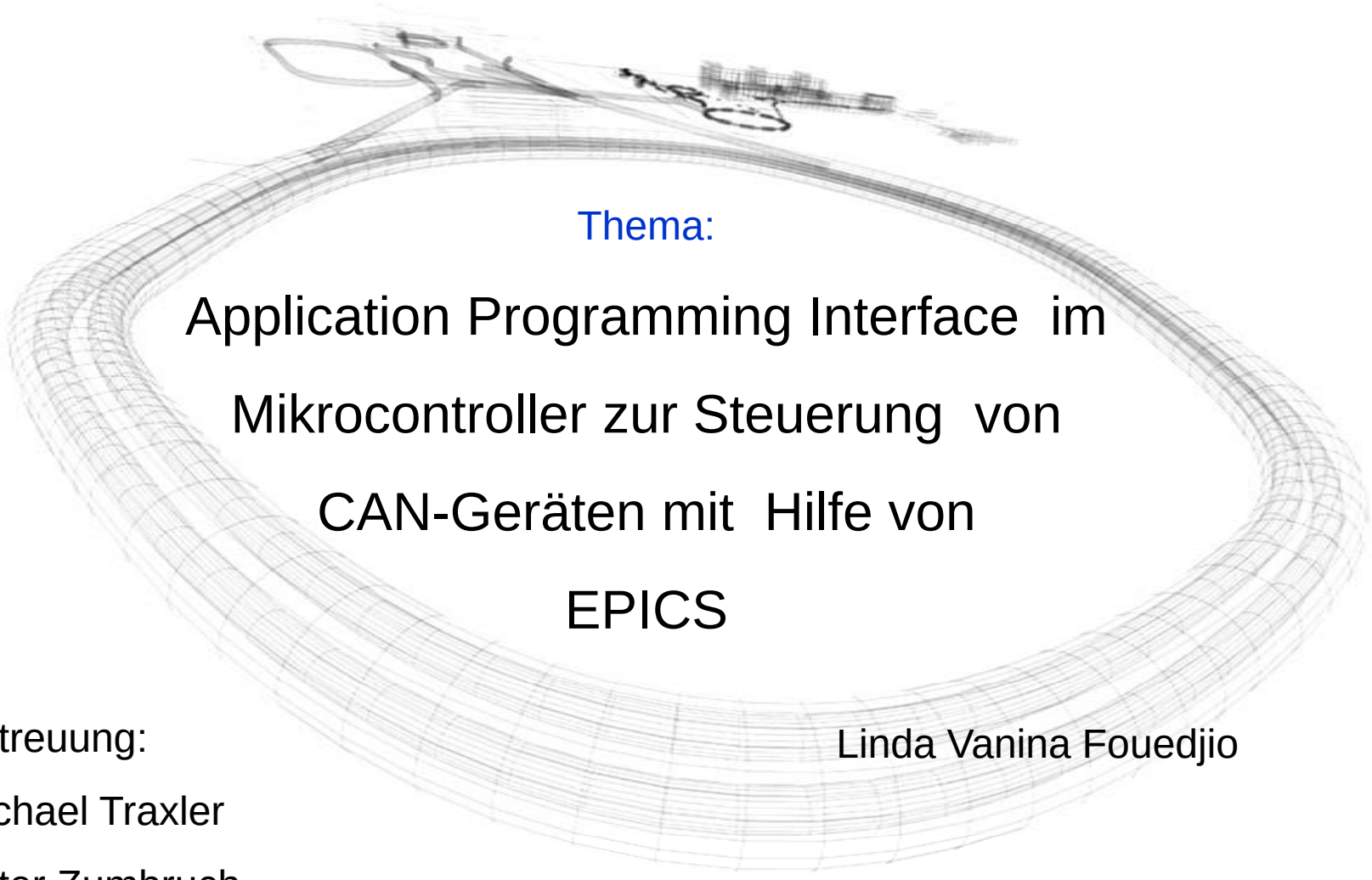
h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Arbeitsvortrag



Thema:

Application Programming Interface im Mikrocontroller zur Steuerung von CAN-Geräten mit Hilfe von EPICS

Linda Vanina Fouedjio

Betreuung:

Michael Traxler

Peter Zumbruch

GSI

Inhaltsverzeichnis

1. Problemstellung

- Beim Betrieb im HADES entsteht stark ionisierende Strahlung
  - Zutritt verboten
  - Kein Zugang auf Detektorkomponente und andere Geräte

- Für die Überwachung
  - Fernsteuerung
    - Spannungen, Ströme, Temperaturen auf Detektor Komponente
    - Regelung und Steuerung der Geräte
  - Viele kontrollierte Geräte haben ein CAN-Interface

2. Ziel der Arbeit

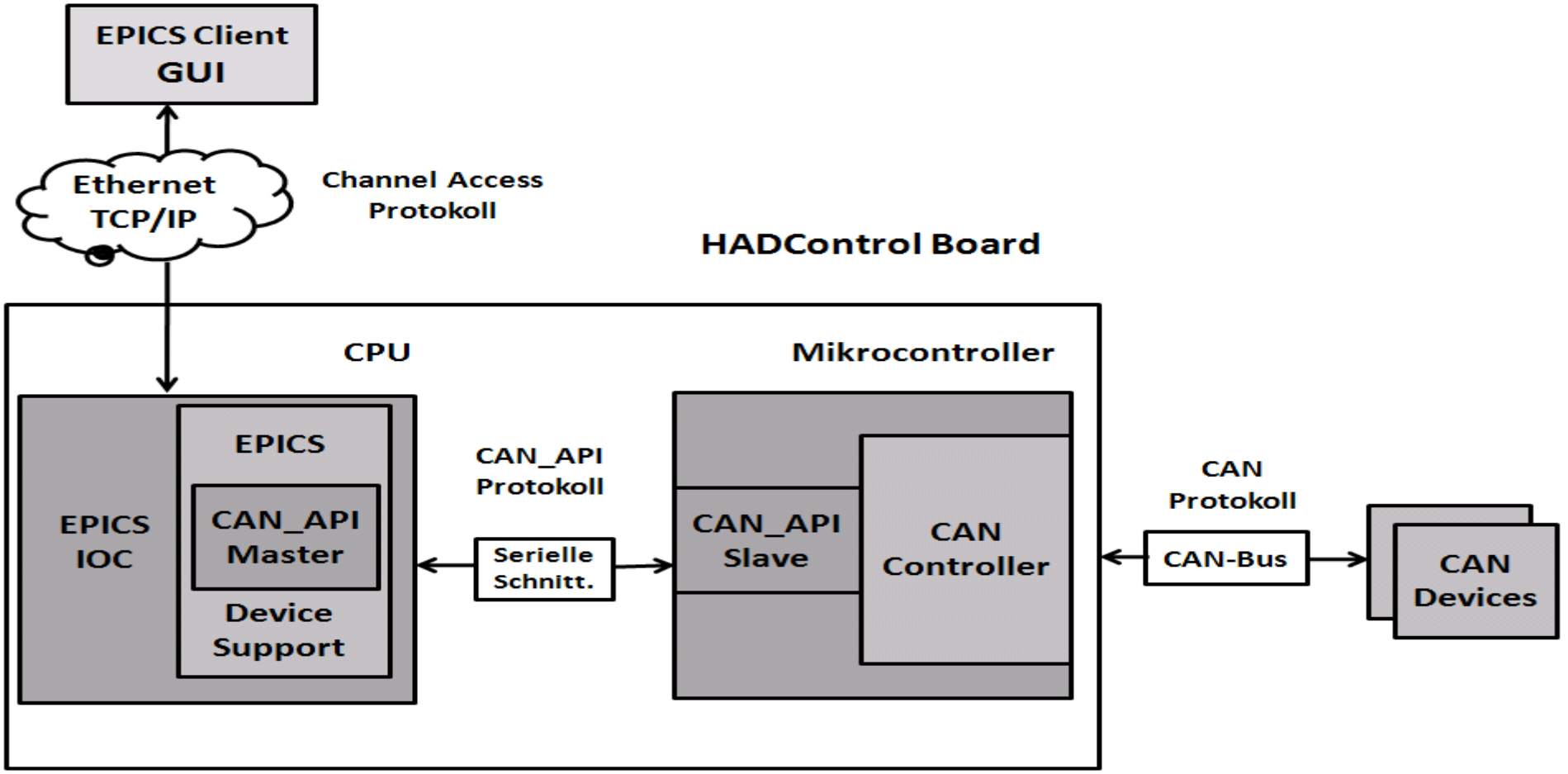
- Ersetzen des auf VME basierten Systems zur Steuerung der CAN-Geräte durch das HADControl Board
  - VME basierte System ist alt, zentral, überlastet und nicht erweiterbar
  - HADControl ist verteilbar, flexibel, leicht erweiterbar
  - HADControl hat CAN und TCP/IP-Schnittstelle

- Implementierung eines **Application Programming Interface** (API) im HADControl Board

- Nutzung des APIs mittels EPICS

3. Lösungskonzept

Überblick über das Gesamtsystem



3. Lösungskonzept

Lösungsansätze

- Erstellung eines Kommunikationsprotokolls
- Implementierung des APIs im Mikrocontroller
- Konfiguration eines EPICS Device Supports

3. Lösungsweg

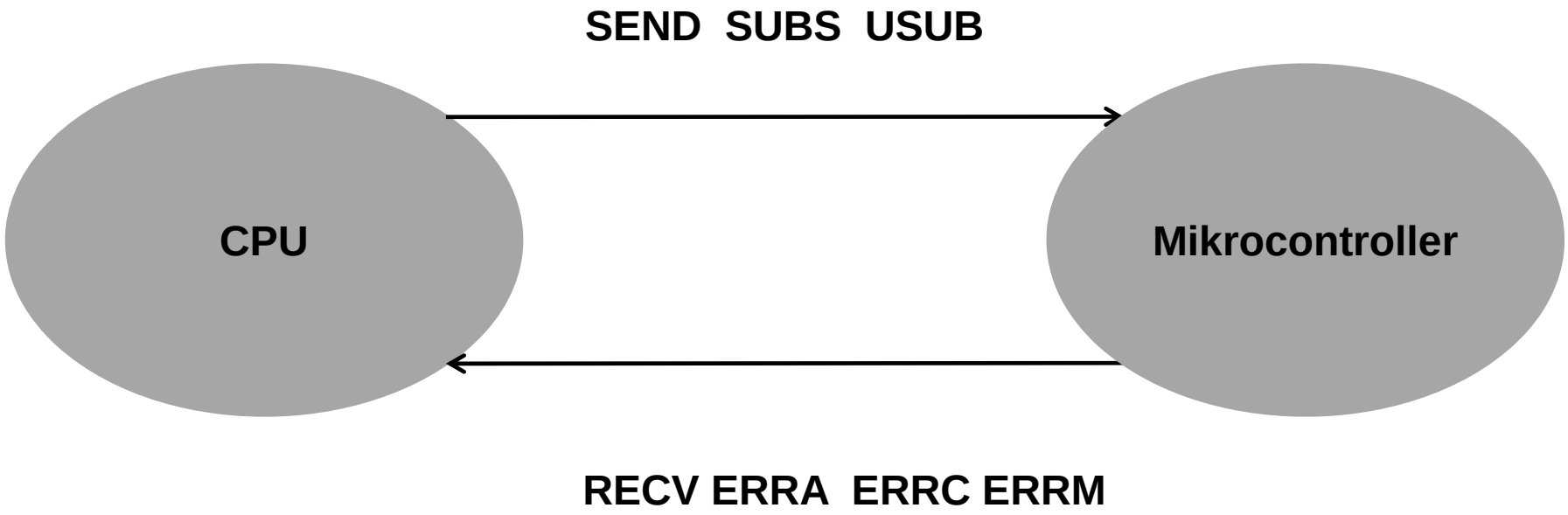
Kleine Einführung in CAN (Control Area Network)

- Identifier(ID) kennzeichnet den Inhalt der Nachricht
- **Remote Transmision Request** (RTR)
- Maximal 8 Byte Datenfeld

4. Lösungsweg

Beschreibung des CAN_API-Protokolls

- Befehls- und Antwortübersicht



4. Lösungsweg

Beschreibung des CAN_API-Protokolls

- Bedeutung der Kommandos

SEND

Kommando für das Lesen und Schreiben auf den CAN-Bus

SUBS

Registrierung für das Lesen bestimmter Identifier

USUB

Abmeldung für das Lesen bestimmter Identifier

4. Lösungsweg

Beschreibung des CAN_API-Protokolls

- Bedeutung der Antworten

RECV

Erfolgreiche Kommunikation

ERRA

Syntaxfehler des APIs

ERRC

Fehler während der CAN-Kommunikation

ERRM

Interner Fehler in CAN-Controller des Mikrocontroller

4. Lösungsweg

Beschreibung des CAN_API-Protokolls

- Struktur für Kommandos

  “Commando-Name Message-ID Mask-ID RTR Length “

  “Commando-Name Message-ID Mask-ID RTR Length Data“

  “Commando-Name Message-ID Mask-ID“

Beispiel:

“SEND 304 0 1 8“ Lesen: Anzahl Umdrehung des Lüfters pro Sekunde

“SEND 84 0 0 1 1“ Schreiben: Ausschalten des Crates

“SUBS 10 FF7“

- Struktur der Daten

  “Commando-Response Mob-Number Message-ID Length DATA“

  “Commando-Response commando will carried out“

  “Commando-Response Error-Number Error-Description”

Beispiel:

“RECV 0 304 8 83 b8 31 32 31 ff ff ff “

“ERRA 6 Data is out of Range“

4. Lösungsweg

Beschreibung des CAN_API-Protokolls

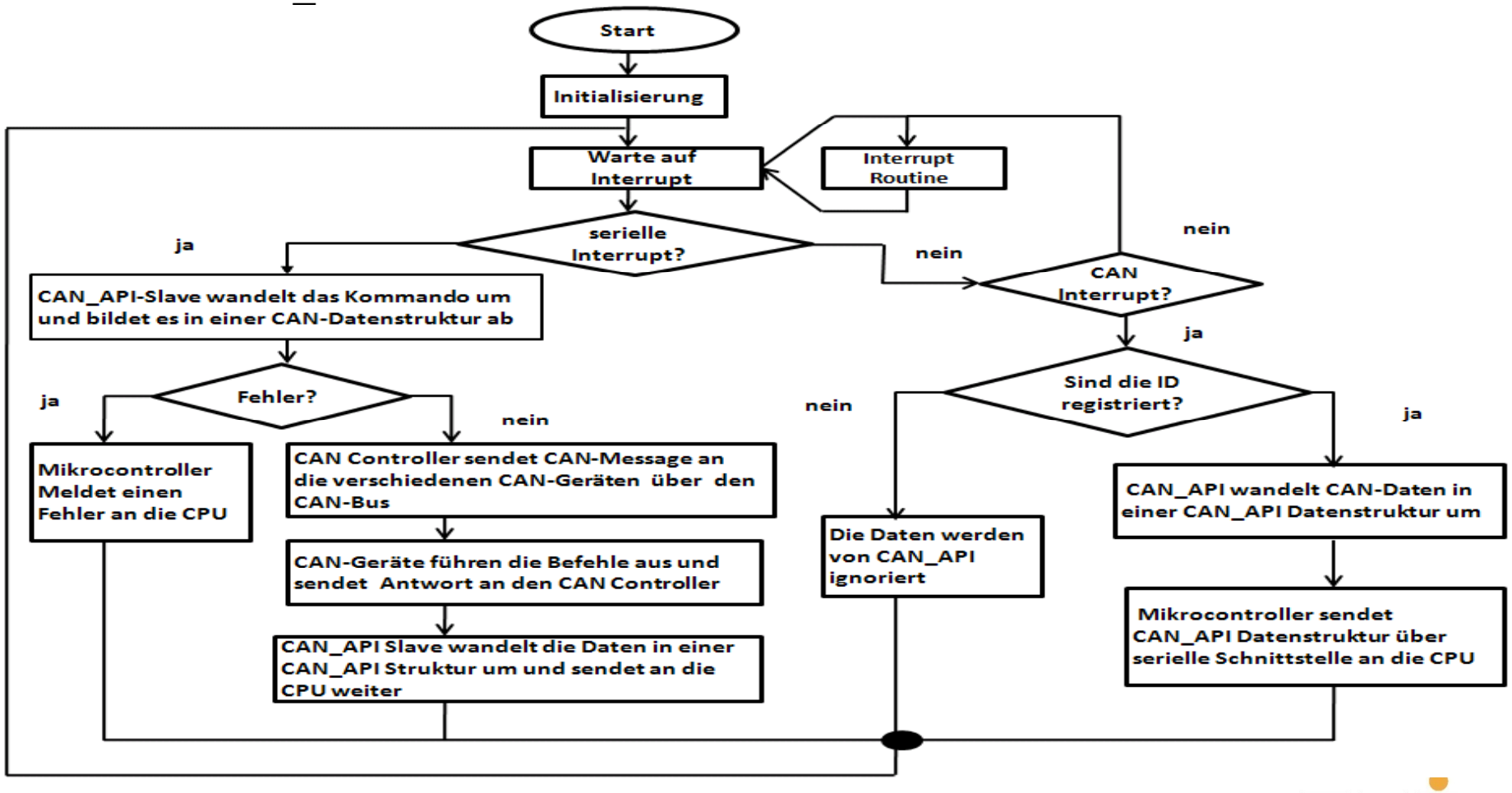
- Zusammenfassung

Parametername	**Bedeutung**	**Erlaubte Werte**
Command-Name	Name des zu sendenden Kommandos	SEND/SUBS/USUB
Command-Response	Antwort auf Kommando	RECV/ERRA/ERRC/ERRM
DATA	Datenbyte	0x0 …FF
Error-Code	Nummer von Fehler	0…9
Error-Description	Beschreibung von Fehler	0x41…0x5A und 0x61..0x71
Length	Anzahl der Datenbyte	0…8
Message-ID	CAN Message ID	0x0 …FFFFFFF7
MOB-Number	Message Object Block	0...14
Mask-ID	CAN Message ID-Mask	0x0 …FFFFFFF7
RTR	Remote Transmission Request	0/1

4. Lösungsweg

CAN_API Implementierung

- CAN_API -Slave



4. Lösungsweg

HADControl mit EPICS

IOC

Input Output Controler

Process Variable

ist ein Datensatz mit einem Namen, der über diesen angesprochen wird

Record

ist die Definition der Process Variable und seiner Attribute

Database

ist eine Sammlung von Records

STREAM Protocol

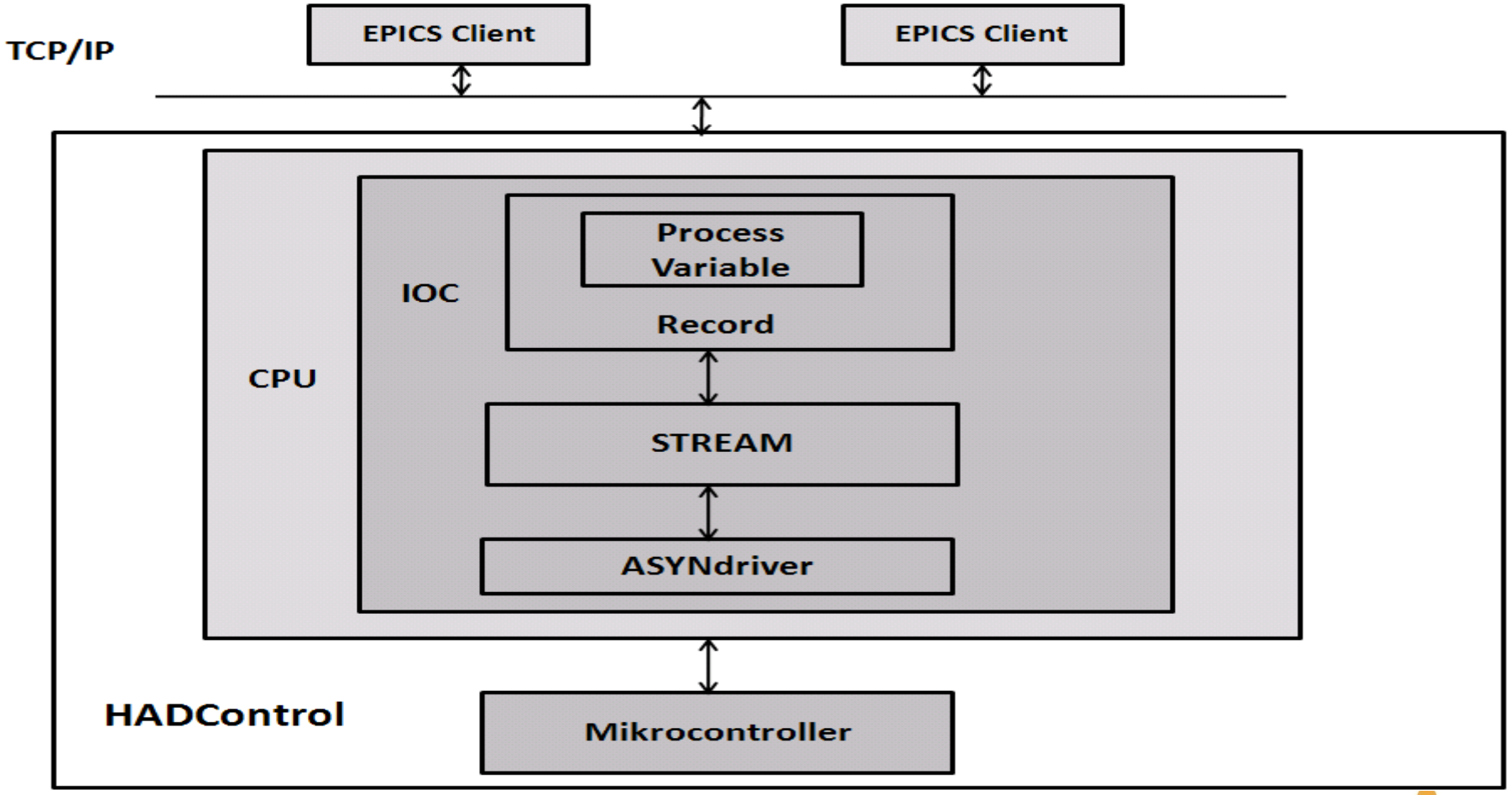
EPICS Device Support für Devices mit Byte Stream

ASYNdriver

asynchrone Kommunikation

4. Lösungsweg

Überblick über das System



4. Lösungsweg

- Stream Protocol File

HadConCAN.proto

```
Terminator = LF;
LockTimeout = 4000;
ReplyTimeout = 500;
WriteTimeout = 3000;
ExtraInput = Ignore;

#After processing finishes, the record contains the fan speed Value.
get8{
#Example of command
#field(INP,“@HadConCAN.proto get8(304,0,1,8,recordName,error)HadConCAN”)
         #MessageID = %\${1}x;   #Mask = %\${2}x;              #RTR = %\${3}x;
         #Length = %\${4}x;      #recordNamePrefix = %\${5}x;  #errorChannelPrefix = %\${6}x;

          out “SEND \$1 \$2 \$3 \$4”;

          in  “RECV %*x %*x %*x %(VAL)x %(\${5}_2.VAL %(\${5}_3.VAL %(\${5}_4.VAL %(\${5}_5.VAL
                                        %(\${5}_6.VAL %(\${5}_7.VAL %(\${5}_8.VAL”;

          @mismatch{err_6};
}

#this error is called within other protocols
err_6{
          in “ERR*1c %(\${6}:errorNr.VAL)i %(\%{6}:error.VAL)39c”;
}
```

4. Lösungsweg

- Record

HadConCAN.db

```
record(ai, “fanSpeed_1”){

        field(DTYP, “stream”)
        field(INP,“@HadConCAN.proto get8(304,0,1,8,fanSpeed,error) HadConCAN”)

}
```

```
record(ai ,”fanSpeed_2”){
}
. . .
record(ai ,”fanSpeed_8”){
}
```

4. Lösungsweg

- Startup Script für IOC

```
#Definition of ASYN port and its attributes connecting to a serial device
drvAsynSerialPortConfigure("HadConCAN","/dev/ttyS1",0,0,0)
asynSetOption("HadConCAN",0,"baud","38400")
asynSetOption("HadConCAN",0,"bits","8")
asynSetOption("HadConCAN",0,"parity","none")
asynSetOption("HadConCAN",0,"stop","1")
asynSetOption("HadConCAN",0,"clocal","Y")
asynSetOption("HadConCAN",0,"crtscts","N")


## Load record instances
dbLoadRecords("db/HadConCAN.db","HOSTNAME=${HOSTNAME}")
```

5. Zusammenfassung / Ausblick

- Erfolgreiche Implementierung des APIs
- Nutzung des implementierten APIs mittels EPICS
- Test des gesamten Systems im Labor

❖ Das API wird bei HADES eingesetzt werden

Ich bedanke mich für Ihre Aufmerksamkeit