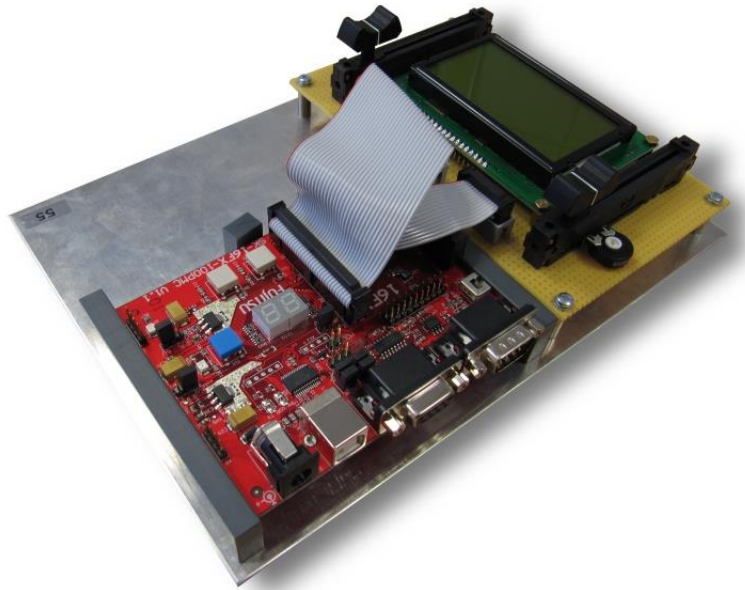


# Programmierpraktikum C und C++

## Embedded Systems - Einführung



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT



ES Real-Time Systems Lab

Prof. Dr. rer. nat. Andy Schürr

Dept. of Electrical Engineering and Information Technology

Dept. of Computer Science (adjunct Professor)

**Anthony Anjorin**

[anthony.anjorin@es.tu-darmstadt.de](mailto:anthony.anjorin@es.tu-darmstadt.de)

[www.es.tu-darmstadt.de](http://www.es.tu-darmstadt.de)

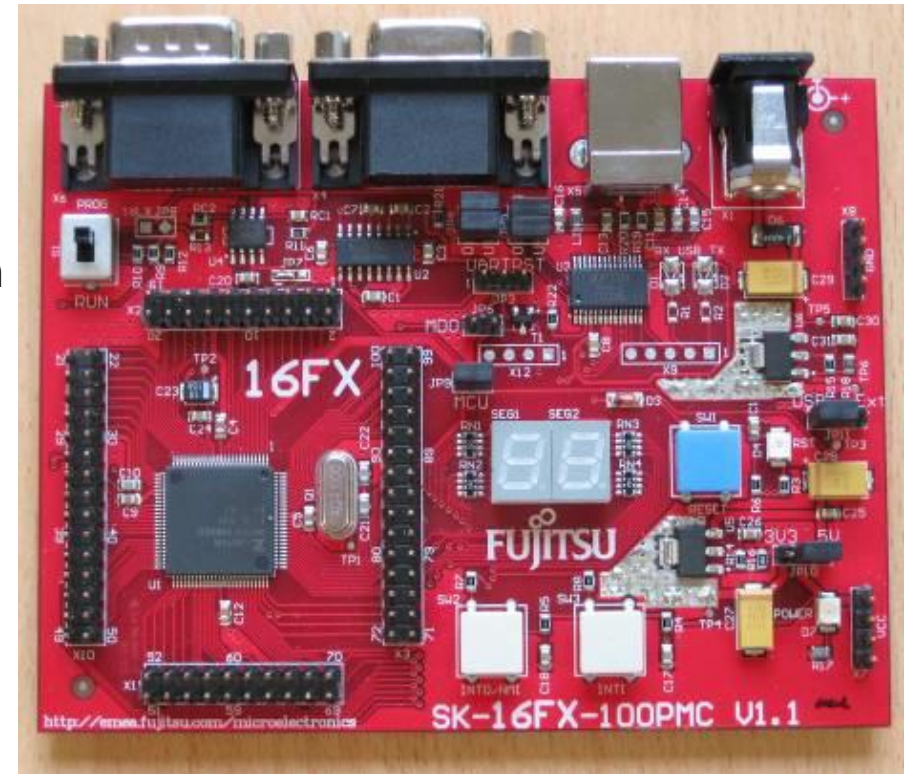
# Entwicklungsboard

## MB96F348HSB Mikrocontroller

- Prozessortaktung: bis 56 MHz
- RAM: 24 KiB
- Flash: 576 KiB
- 82 I/O Pins
- Analog/Digital-Wandler mit 24 Kanälen
- CAN-Controller
- ...

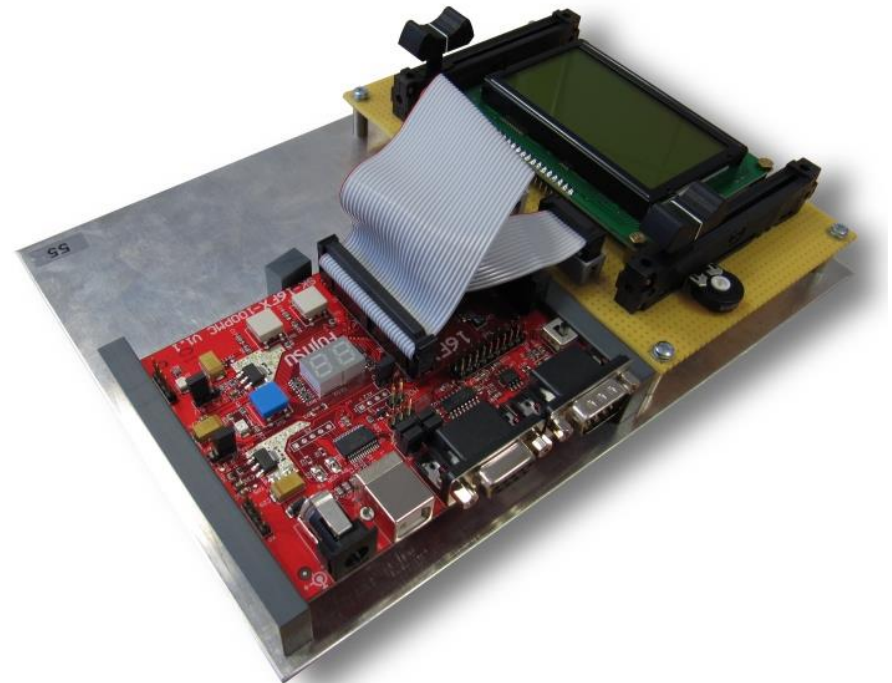
## Starterkit SK-16FX-EUROscope

- Zwei 7-Segment-Anzeigen
- Zwei Buttons
- Stromversorgung über USB (5V)



# Erweiterung am Fachgebiet

- Display
  - AV128641 von Anag Vision
  - Vollgraphisch
  - 128 x 64 Pixel
  - Hintergrundbeleuchtet
- Zwei Schiebepotentiometer



# C-Compiler

- Von Fujitsu Microelectronics Ltd.
- Unterstützt nur ANSI C90, zusätzlich aber einzeilige `//` Kommentare
- Compiler enthält eine interne Funktion namens `__wait_nop()`, die eine CPU-Instruktion zum Warten für einen Taktzyklus („NOP“) auslöst
- Konstanten werden standardmäßig im ROM gespeichert, nicht im RAM (RAM ist wertvoll, da nur 24 KiB zur Verfügung stehen)



## Keine standardisierte „Umgebung“

- Compiler kann nicht wissen, welche Komponenten angeschlossen sind
- Es gibt keine Ausgabe über printf()
- Ansteuerung externer Peripherie muss vom Entwickler selber durchgeführt werden
  - Wird zum Teil unterstützt durch fertige Bibliotheken

Umfangreiche und flexible Hardware → erfordert Konfiguration

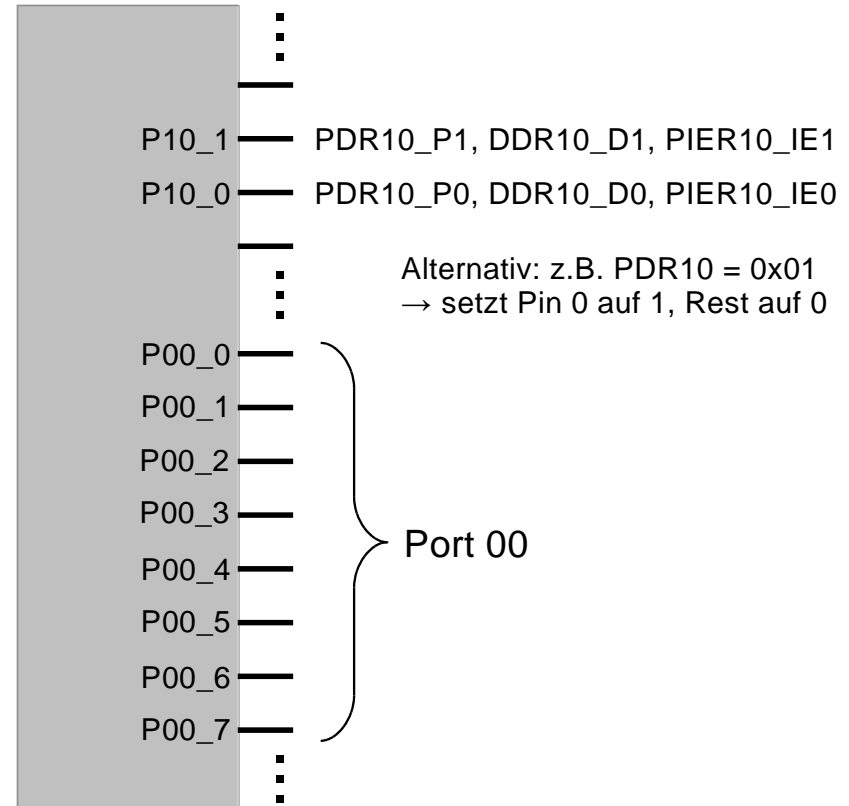
- Realisiert über Register
  - Im Controller integrierte „Variablen“ mit unterschiedlicher Größe
  - Zugriff im Code über fest vorgeschriebene Namen
  - Bedeutung unterschiedlich je nach Register
    - Ganzes oder teil des Registers als Zahlenwert, z.B. Zählerwert
    - Einzelne Bits als „Schalter“ für bestimmte Funktion, z.B. einzelnes Ausgangspin auf High oder Low

Kommunikation mit Außenwelt über

- Einzelne digitale Ein/Ausgänge
- Analoge Eingänge
- Schnittstellen, z.B.
  - USART
  - CAN

## Digitale Ein/Ausgänge:

- Bis zu 8 Pins zusammengefasst zu einem Port
- Jedes Pin hat eigene Konfiguration über mehrere Register, u.a.
  - Port-Data-Register (PDR)
    - Eingang: Abfrage des Zustandes
    - Ausgang: Setzen des Pegels
  - Data-Direction-Register (DDR)
    - Setzen auf Eingang oder Ausgang
    - 0 → Eingang, 1 → Ausgang
  - Port-Input-Enable-Register (PIER)
    - Bei Eingangspin den Eingang aktiv schalten



# Zugriff auf Pins



```
/* Beispiel: Pins als Eingang */
char status;
DDR07_D0 = 0;           // Pin 0 von Port 07 als Input
PIER07_IE0 = 1;         // Pin 0 von Port 07 als Eingang aktiv
status = PDR07_P0;      // Pegel an Pin 0 von Port 07 abfragen
                        // -> Status des linken Tasters
```

```
/* Beispiel: Pins als Ausgang */
DDR00 = 0xff;           // Alle Pins von Port 00 als Output
PDR00 = 0xff;           // Alle Pins von Port 00 auf High-Pegel
                        // -> Rechte 7-Segment-Anzeige komplett aus
PDR00_P7 = 0;          // Pin 7 von Port 00 auf Low-Pegel
                        // -> Punkt der rechten 7-Segment-Anzeige an
```





- 8 Bit oder 10 Bit Genauigkeit (wir verwenden 8 Bit)
- Verschiedene Wandlungsmodi (z.B. mehrere Eingänge sequentiell wandeln)
  - Wir verwenden Stop Mode, ein Kanal wird einmal pro Startsignal gewandelt
  - Start- und Endkanal erhalten bei jeder Wandlung einen identischen Wert

```
/* Beispiel */
unsigned char result;

ADCS_MD    = 3;      // ADC Stop Modus
ADCS_S10    = 1;      // 8 Bit Genauigkeit
ADER0_ADE2 = 1;      // Analoge Eingänge aktivieren: AN2 + AN3
ADER0_ADE3 = 1;      // (ADER0: Eingänge AN0 bis AN7)

ADSR = 0x6C00 + (3 << 5) + 3;    // Start- und End-Kanal 3

ADCS_STRT = 1;                // A/D-Wandler starten
while (ADCS_INT == 0) { }     // Warten bis A/D-Wandlung beendet
result = ADCRL;                // Ergebnis speichern
ADCS_INT = 0;                  // Bit auf 0 für nächste Wandlung
```