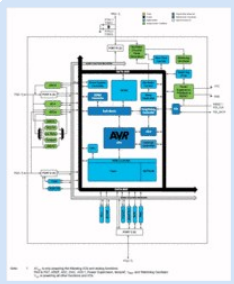
Atmel AVR xMega32-E5 based XStack board

- 8/16-bit, high-performance Atmel AVR RISC CPU
- konfigurierbarer Oscillator: f = 32 kHz …32 MHz
- 141 instructions, Hardware multiplier
- 32x8-bit registers directly connected to the ALU
- Efficient support for 8-, 16-, and 32-bit arithmetic
- Programm + Boot (Flash) 32 kByte + 4 kByte
- SRAM 4 kByte, EEPROM 0,5 kByte
- 2 16bit Timer/Counter, TCC4 und TCC5
- 2 USARTs; 4 GPIO Ports; 12bit ADC; 2 12bit DAC



Atmel AVR xMega32-E5

Laboringenieur
Dipl.-Ing. Th. Wegner
Raum: W1314
Tel.: 498 7267

Schnelleinstieg
Publikationen
Anfahrt
Kontakt
Laborpraktikum
Lehrangebot
Highlights
Projekte

GPIO - General Purpose I/O Ports

3 parallele Eingabe/Ausgabe-Ports

PortA (0x0600): PA0 - PA7; Taste an PA4
PortC (0x0640): PC0 - PC7
PortD (0x0660): PD0 - PD7; PD6 und PD7 für RX und TX von USART0
PortR (0x07E0): PR0, PR1; LED1 und LED2

Input-, Output- und Control-Register der Ports

Data Direction Register: DIR, DIRset, DIRclear, DIRtoggle
Data Output Value Register: OUT, OUTset, OUTclear, OUTtoggle
Data Input Value Register: IN
Interrupt Control Register: INTctrl, INTmask, INTflags
Pin Remap Register: REMAP
Pin n Control Register: PIN0CTRL ... PIN7CTRL

C-Syntax

```
/*
 * Ansteuerung von LED1 an Port R, Pin 0
 * Basis Adresse von PORTR: 0x07E0
 * Offset von Register DIRSET: +0x01
 * Offset von Register OUTCLR: +0x06
 * E/A-Adresse von PORTR.DIRSET: 0x07E1
 * E/A-Adresse von PORTR.OUTCLR: 0x07E6
*/
#include <avr/io.h>    // dort weiter zu iox32e5.h
int main(void)
{
   PORTR.DIRSET = 1; // Pin 0 auf 1 setzen, d.h. Output-Mode
   PORTR.OUTCLR = 1; // Pin 0 auf 0 setzen, LED on, LEDs low activ

/* PORT A im InputMode mit PullUpWiederstand an Pin 4, reagiert auf fallende Flanke */
   PORTA.PIN4CTRL = PORT_OPC_PULLUP_gc | PORT_ISC_RISING_gc;
}
```

Timer/Counter - Counter with Capture/Compare-Channels

2 16bit Timer/Counter

TCC4 (0x0800):
TCC5 (0x0840):

Counter- und Control-Register von Timer 4 und 5

Control Register A - E: CTRLA ... CTRLE
Control Register G clear und set: CTRLGCLR, CTRLGSET
Interrupt Control Register A und B: INTCTRLA, INTCTRLB
Counter Register low und high: CNTL, CNTH
Period Register low und high: PERL, PERH

C-Syntax

```
/*
 * Konfiguration von Timer 5 an Port D
 * Basis Adresse von Timer 5: 0x0840
 * Offset von Register CTRLGCLR: +0x08
 * E/A-Adresse von TCC5.CTRLGCLR: 0x0848
*/
#include <avr/io.h>
int main(void)
{
   TCC5.CTRLGCLR = TC5_DIR_bm; // Pin 0 auf 0 setzen, d.h. Vorwärtszähler
   TCC5.CNT = 0x00;                    // Counter Register 0 setzen
   TCC5.PER = 31250;                   // Period Register mit Konstante laden
}
```

PMIC - Progammable Multilevel Interrupt Controller

PMIC (0x00A0)
3 konfigurierbare Interrupt-Level: low, medium, high
Im Low-Level-Bereich ein Round-Robin-Prioritäts-Schema wählbar
NMI für zeitkritische Funktionen

C-Syntax

```
/*
 * Konfiguration des Interrupt Controllers
 * Basis Adresse des PMIC: 0x00A0
 * Offset von Register CTRL: +0x02
 * E/A-Adresse von PMIC.CTRL: 0x00A2
*/
#include <avr/io.h>
#include <avr/interrupt.h>
/* Interrupt-Service-Routinen für Taste an PortA.4 und Timer/Counter 5 Overflow
 * Port A Interrupt Vector: 0x003C (PORTA_INT_vect)
 * TCC 5 Interrupt Vector: 0x0024 (TCC5_OVF_vect)
*/
ISR(PORTA_INT_vect)   // externer Interrupt von Taste an Pin PA4
{
 counter = 0;                  // counter reset durch Tastendruck
                               // writing a 1 will clear the flag
 PORTA.INTFLAGS = PORT_INT4IF_bm; // Interruptanforderung löschen
}


ISR(TCC5_OVF_vect)     // interner Interrupt von Timer5-Overflow
{
 counter++;                    // Zähler erhöhen
                               // writing a 1 will clear the OVFIF-flag
 TCC5.INTFLAGS = TC5_OVFIF_bm;  // Interruptanforderung löschen

}


int main(void)
{
   cli();                      // disable globaler Interrupt, BCLR SREG.7, Reset-Zustand
                               // Pin PA4 als externe Interruptquelle
  PORTA.INTMASK = PORT_INT4IF_bm;    //  enable PortA.4 Interrupt
  PORTA.INTCTRL = PORT_INTLVL_LO_gc; // Interrupt-Priorität Low Level
                               // Timer Overflow Interrupt enable, high level
  TCC5.INTCTRLA = 0x03;      // OVFINTLVL=3
                               // alle drei Interrupt-Level am Interrupt-Controller aktivieren
   PMIC.CTRL = (PMIC_HILVLEN_bm | PMIC_MEDLVLEN_bm | PMIC_LOLVLEN_bm);
                               //
   sei();                      // enable globaler Interrupt
}
```

USART - universal synchronous and asynchronous serial receiver and transmitter

USART0 (0x09C0) an PORTD
PD6: RXD0
PD7: TXD0
Separate interrupts for Transmit Data Register empty, Receive complete, Transmit complete

Input-, Output- und Control-Register von USART0

Data Register RXB und TXB: DATA
Status Register: STATUS
Control Register A-D: CTRLA ... CTRLD
Baud Rate Control Register A und B: BAUDCTRLA und BAUDCTRLB

C-Syntax

```
/*
 * Konfiguration von USART0 an Port D
 * Basis Adresse von USARTD0: 0x09C0
 * Offset von Register CTRLB: +0x03
 * E/A-Adresse von USARTD0.CTRLB: 0x09C3
*/
#include <avr/io.h>
int main(void)
{
/* Port D konfigurieren, PD7: TX, PD6: RX */
   PORTD.REMAP |= PORT_USART0_bm;          // Port Remap von C nach D
   PORTD.OUTSET = (1 << 7);                // HIGH-Pegel an PD7 (TXD0)
   PORTD.DIRSET = (1 << 7);                //  Output anschalten
/* USARTD0 konfigurieren,  921600 bit/s bei 32MHz CPU-clock */
   USARTD0.BAUDCTRLA = 75;                 // BSEL=75, BSCALE=-6
   USARTD0.BAUDCTRLB = ((-6) << 4) | ((75) >> 8);
   USARTD0.CTRLB = USART_RXEN_bm | USART_TXEN_bm;  // RX/TX enable
volatile uint8_t c;
while  ((USARTD0.STATUS & USART_RXCIF_bm) == 0); // Receive complete interrupt flag
c = USARTD.DATA;                           // Zeichen einlesen
/* The transmit buffer can only be written when the DREIF flag in the STATUS register is set */
while  ((USARTD0.STATUS & USART_DREIF_bm) == 0); // Data Register Empty Flag
USARTD.DATA = 'A';                         // den Buchstaben A ausgeben
}
```