CLAR Beispiele Raspberry | C - Linux - Arduino - Raspberry

C-Programme für die GPIO-Schnittstelle vom Raspberry Pi B+ mit RASPIAN Debian Wheezy.

Als Entwicklungrechner wurde ein Linux PC verwendet. Betriebssystem:

```
Linux pc780mint 3.11-2-686-pae #1 SMP Debian 3.11.8-1 (2013-11-13) i686 GNU/Linux
```

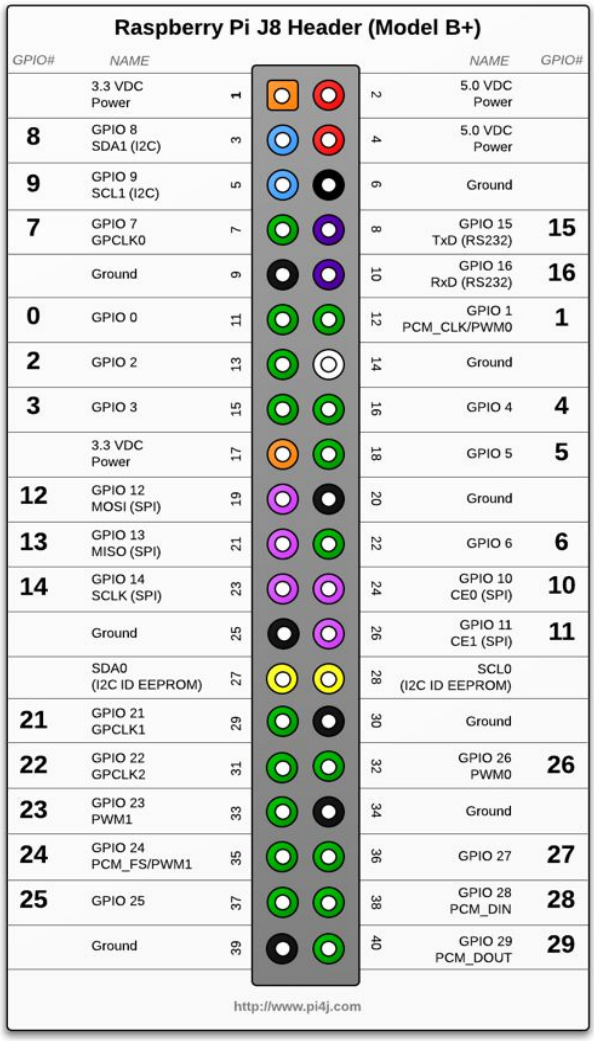
Die beschriebenen Grundeinstellungen erlauben es die C-Programme auf PC und Pi zu compilieren.

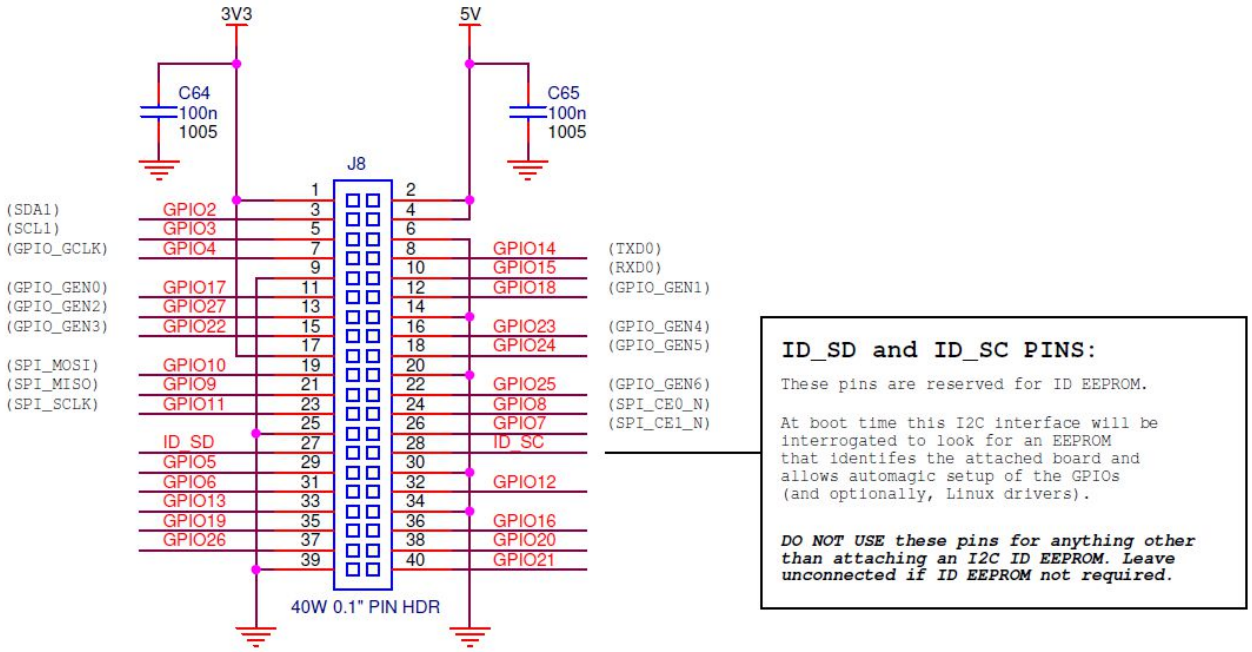
Zum Ansteuern der GPIO wird die Bibliothek WiringPi verwendet.
Für die Tests am PC wird eine Simulations-Library mit einer Kopie der WiringPi-Header verwendet.

Inhaltsverzeichnis

Hardware:

Stecker J8





Pinnummern mit wiringPi

Model B+ is 100% compatible with the existing Model B, Rev 2:

Bezeichnung der Pins:

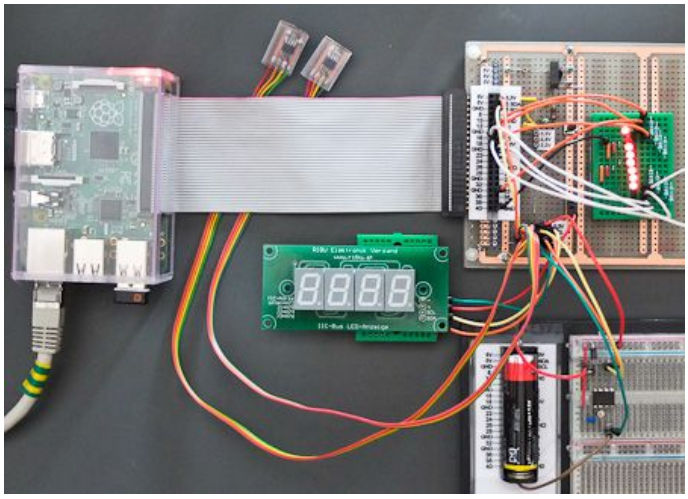
```
gpio readall
```

Es werden die BCM Broadcom GPIO-Pinnummern verwenden:

```
wiringPiSetupGpio()
```

```
gpio readall
+-----+-----+---------+------+---+--B Plus--+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 |     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |     |     |
 |   2 |   8 |   SDA.1 |   IN | 1 |  3 || 4  |   |      | 5V      |     |     |
 |   3 |   9 |   SCL.1 |   IN | 1 |  5 || 6  |   |      | 0v      |     |     |
 |   4 |   7 | GPIO. 7 |   IN | 1 |  7 || 8  | 1 | ALT0 | TxD     | 15  | 14  |
 |     |     |      0v |      |   |  9 || 10 | 1 | ALT0 | RxD     | 16  | 15  |
 |  17 |   0 | GPIO. 0 |  OUT | 0 | 11 || 12 | 0 | IN   | GPIO. 1 | 1   | 18  |
 |  27 |   2 | GPIO. 2 |   IN | 0 | 13 || 14 |   |      | 0v      |     |     |
 |  22 |   3 | GPIO. 3 |   IN | 0 | 15 || 16 | 0 | IN   | GPIO. 4 | 4   | 23  |
 |     |     |    3.3v |      |   | 17 || 18 | 0 | IN   | GPIO. 5 | 5   | 24  |
 |  10 |  12 |    MOSI |   IN | 0 | 19 || 20 |   |      | 0v      |     |     |
 |   9 |  13 |    MISO |   IN | 0 | 21 || 22 | 0 | IN   | GPIO. 6 | 6   | 25  |
 |  11 |  14 |    SCLK |   IN | 0 | 23 || 24 | 1 | IN   | CE0     | 10  | 8   |
 |     |     |      0v |      |   | 25 || 26 | 1 | IN   | CE1     | 11  | 7   |
 |   0 |  30 |   SDA.0 |   IN | 1 | 27 || 28 | 1 | IN   | SCL.0   | 31  | 1   |
 |   5 |  21 | GPIO.21 |   IN | 1 | 29 || 30 |   |      | 0v      |     |     |
 |   6 |  22 | GPIO.22 |   IN | 1 | 31 || 32 | 0 | IN   | GPIO.26 | 26  | 12  |
 |  13 |  23 | GPIO.23 |   IN | 0 | 33 || 34 |   |      | 0v      |     |     |
 |  19 |  24 | GPIO.24 |   IN | 0 | 35 || 36 | 0 | IN   | GPIO.27 | 27  | 16  |
 |  26 |  25 | GPIO.25 |   IN | 0 | 37 || 38 | 0 | IN   | GPIO.28 | 28  | 20  |
 |     |     |      0v |      |   | 39 || 40 | 0 | IN   | GPIO.29 | 29  | 21  |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+--B Plus--+---+------+---------+-----+-----+
```

GPIO Beispiele



Alle Pins werden immer mit der physikalischen Pinnummer des J8 Steckers bezeichnet.
Diese Nummern werden für die Bibliothek WiringPi automatisch BCM_GPIO Nummern umgerechnet.

io00 LED schalten

Ordner: c/pi/io_00
LED ein- ausschalten. Physikalischer Anschluss J8, Pin 11.

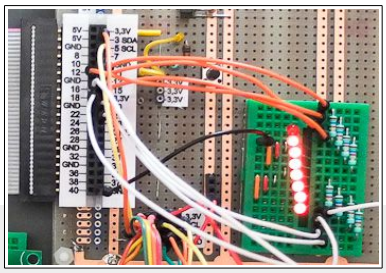
Bildschirmausgabe mit Library iocon in blau. Siehe c/libs/iocon
Aufrufe von Bibliotheksfunktionen aus WiringPi sind rot.

```
int main(void)
{ clrScr();                          // Bildschirm löschen
  printBlock(Info , Blockfarbe1);  // Textblock anzeigen


  // Interface aktivieren ---------------------------------
  // BCM_GPIO Nummern verwenden
  if (wiringPiSetupGpio() == -1 )
  { printf("wiringPiSetupGpio() fehlgeschlagen!\n");
    return 1;
  }

  WiringInfo();

  // Pinnummer zu J8-Pin 11 berechnen -->  BCM_GPIO 17
  int8_t Pin_11= physPinToGpio(11);

  printf("%s",Blockfarbe2); clrEol();  // farbige Zeile
  printf("LED Anschluss j8_11 -> BCM:%i\n", Pin_11);
  printf("%s",BlockNormal);          // normale Textausgabe
  printf("\n");

  pinMode( Pin_11 , OUTPUT);         // Pin als Ausgabepin verwenden

  savePos();                         // Cursorposition speichern
  uint16_t c=taste_nul, value=LOW;

  while (c!=taste_esc)
  { if (value==LOW) value=HIGH;
    else value=LOW;
    restPos();                       // restore Cursor

    digitalWrite( Pin_11 , value);  // Led schalten

    printf("LED --> %i\n\n", value); fflush(stdout);
    c=getTaste(" - Ein/Aus mit Taste\n - Ende ESC\n ");
  }

 digitalWrite( Pin_11 , LOW);
 printf("\n\n");
 return 0;
} // ------------------------------------------------------------------------
```

```
pi@pi7: ~/c/pi
GPIO-Beispiel io00:

 C Library wiringPi und Simulation wiringPi_sim
 makefile: am Pi --> wiringPi
           am PC --> wiringPi_sim

 - Boardinfos anzeigen
 - Pins: Broadcom-Nummern für Anschluss j8 anzeigen
 - LED auf Pin j8-11 ein/ausschalten

WiringInfo:
model=3, rev=3, mem=512, maker=2, overVolted=0
.....................................................
wiringPiSetupGpio():
 | j8| physikalischer Anschluss
 |BCM| Broadcom Nummern

|BCM| J8  |BCM|
| -1| 1-2 |-1 |
|  2| 3-4 |-1 |
|  3| 5-6 |-1 |
|  4| 7-8 |14 |
| -1| 9-10|15 |
| 17|11-12|18 |
| 27|13-14|-1 |
| 22|15-16|23 |
| -1|17-18|24 |
| 10|19-20|-1 |
|  9|21-22|25 |
| 11|23-24|8  |
| -1|25-26|7  |
|  0|27-28|1  |
|  5|29-30|-1 |
|  6|31-32|12 |
| 13|33-34|-1 |
| 19|35-36|16 |
| 26|37-38|20 |
| -1|39-40|21 |

Weiter mit Taste
```

```
pi@pi7: ~/c/pi
LED Anschluss j8_11 -> BCM:17

LED --> 1

 - Ein/Aus mit Taste
 - Ende ESC
```

io01 LEDs mit Byte schalten



Ordner: c/pi/io_01
8 LED's mit einem Byte steuern.

Aufrufe von Bibliotheksfunktionen aus WiringPi sind rot.
Aufrufe der Speicherverwaltung vars in blau. Siehe c/libs/vars

Pin-Struktur definieren:

```
// Pin Eigenschaften festlegen
typedef struct
{ int8_t BitNr;   // Bitnummer frei festgelegt
  int8_t J8Nr;    // physikalischer Anschluss J8
  int8_t BcmNr;   // zugeordnete BCM_GPIO Nummern,brechnet
  int8_t PinMode; // Input, Output
  int8_t PullUp;  // Pull-Up Widerstand
} tPin;
```

Pin anlegen:

```
void setPinMode(int8_t J8Nr, int8_t BitNr, int8_t PinMode, int8_t PullUp)
{
  int8_t BcmNr=physPinToGpio(J8Nr);
  if (BcmNr<0)
   { printf("Fehler: Pinnummer %i falsch",J8Nr); exit(EXIT_FAILURE);}

  tPin *p=newStruct(sizeof(tPin)); // neue Strukturvariable anfordern
  printf("J8 %i --> BitNr %i\n",J8Nr, BitNr);

  p->BitNr   =BitNr;
  p->J8Nr    =J8Nr;
  p->BcmNr   =BcmNr;
  p->PinMode =PinMode;
  p->PullUp  =PullUp;

  pinMode        ( BcmNr, PinMode);
  pullUpDnControl( BcmNr, PullUp);
}
```

```
pi@pi7: ~/c/pi
 8 LEDs mit Byte ansteuern
J8 11 --> BitNr 0
J8 12 --> BitNr 1
J8 13 --> BitNr 2
J8 15 --> BitNr 3
J8 16 --> BitNr 4
J8 18 --> BitNr 5
J8 19 --> BitNr 6
J8 21 --> BitNr 7
Var Info: 0x1b21008
Nr |  Adresse   |Group/Sub| Name          | Typ       | Wert
000|  0x1b21008 |  -1/  0 | -             | 4 Struct | 0x1b21038
001|  0x1b21048 |  -1/  0 | -             | 4 Struct | 0x1b21078
002|  0x1b21088 |  -1/  0 | -             | 4 Struct | 0x1b210b8
003|  0x1b210c8 |  -1/  0 | -             | 4 Struct | 0x1b210f8
004|  0x1b21108 |  -1/  0 | -             | 4 Struct | 0x1b21138
005|  0x1b21148 |  -1/  0 | -             | 4 Struct | 0x1b21178
006|  0x1b21188 |  -1/  0 | -             | 4 Struct | 0x1b211b8
007|  0x1b211c8 |  -1/  0 | -             | 4 Struct | 0x1b211f8
---|  0x1b58c |         | &VarNull
Filter: Group=-1, SubGrp=0, chkgroup=true, chksubgr=false
```

GPIO und Pins Initialisieren: Reihenfolge der Bit-Nummern beliebig.

```
void  InitGPIO()
{ // Bei Abbruch ordentlich abschalten
  signal(SIGINT,  ExitSignal); // Strg-C
  signal(SIGTERM, ExitSignal); // kill

  if (wiringPiSetupGpio() == -1 ) { printf("wiringPiSetup() fehlgeschlagen!\n"); exit(EXIT_FAILURE); }

  VarSetGroups(-1,0); VarClrGroups(); // Pinvariablen in die Speichergruppe -1 legen
  setPinMode(11, 0, OUTPUT, PUD_OFF); // J8-Nr=11, Bit-Nr=0
  setPinMode(12, 1, OUTPUT, PUD_OFF); // J8-Nr=12, Bit-Nr=1
  ...
  setPinMode(21, 7, OUTPUT, PUD_OFF); // J8-Nr=21, Bit-Nr=2
} //---------------------------------------------------------------------------
```

Pins nach Byte Bits ein- oder ausschalten:

```
void setPins(uint16_t Bits)
{ tPin *p;
  for( p=nxtStruct(true); p!=NULL; p=nxtStruct(false))
  { uint16_t maske=1 << p->BitNr ;
    if ((maske & Bits)==0)  digitalWrite( p->BcmNr , LOW );
    else                    digitalWrite( p->BcmNr , HIGH);
  };
} //---------------------------------------------------------------------------
```

Schleife über die Variablen der aktuellen Speichergruppe -1:

```
for( p=nxtStruct(true); p!=NULL; p=nxtStruct(false))
  // p=nxtStruct(true)  liefert die 1. Variable oder NULL der aktuellen Speichergruppe
  // p=nxtStruct(false) liefert die folgende Variable oder NULL der aktuellen Speichergruppe
```

Pin mit Bit-Muster schalten:

```
setPins(0b11111010);  // Pins mit Bit-Nr 0 und 2 ausschalten und  alle anderen Pins einschalten
```

Pins sicher abschalten mit Signal-Handler: auch bei kill oder Strg-C

```
void ExitSignal(int Signal)
{ // Signal Handler
  printf("Alle LEDs abschalten\n");
  setPins( 0b00000000 );
  exit(EXIT_SUCCESS);
} //---------------------------------------------------------------------------
```

Speicherinformation der Library varlist:

Speichergruppen-Nummern: Zahl aus int16_t . Negative Zahlen sind sicher frei.

```
VarPrint();  // Speicherbelegung anzeigen
```

In diesem Beispiel wurden Pointervariablen ohne Name (>) und freier Struktur (Wert ?) verwendet:

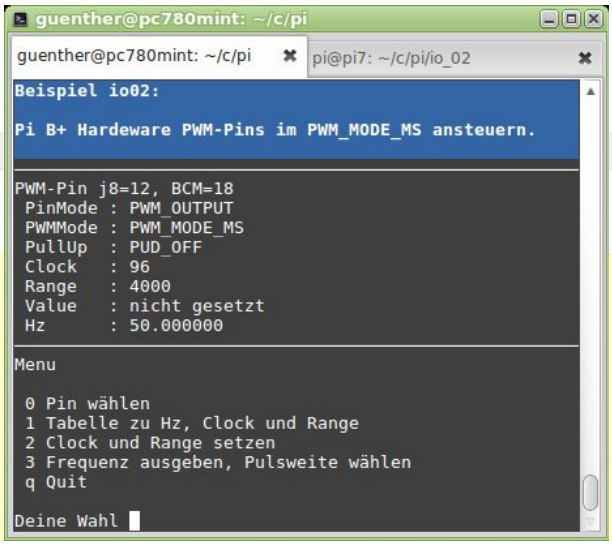
```
Nr |  Adresse   |Group/Sub| Name          | Typ       | Wert
000|  0x1b21008 |  -1/  0 | -             | 4 Struct | 0x1b21038
001|  0x1b21048 |  -1/  0 | -             | 4 Struct | 0x1b21078
...
---|  0x1b58c |         | &VarNull
Filter: Group=-1, SubGrp=0, chkgroup=true, chksubgr=false
```

```
VarClrGroups(); // Alle Variablen im aktuellen Filter freigeben
```

Der von der Struktur belegte Speicher wird freigegeben. Die Pointer bleiben gültig und zeigen danach auf VarNull.

io02 Hardware PWM

Ordner: c/pi/io_02
Hardware Puls-Weiten-Modulation im PWM_MODE_MS



Die Frequenz und Pulsweite wird mit den Parametern
Clock und Range nach folgender Formel festgelegt:

```
double getHz(uint16_t Clock, uint16_t Range)
{ return 19.2e6 / (double)Clock / (double) Range; }
```

Mit der Menü-Option '1 Tabelle zu Hz, Clock und Range' können
mögliche Clock/Range Kombinationen berechnet werden:

```
-------------------------------------------------------------------------------
Es werden nun Clock-Werte zu Hz und Range berechnet.
Range beschreibt die Teilung des Intervalls.

Formel --> double Hz = 19.2e6 / (double)Clock / (double)Range

Frequenz Hz 1- 65535 ? 50
Maximum Range 2-4095 ? 4095
Minimum Range 2-4095 ? 3990
-------------------------------------------------------------------------------
Hz 50.125313 --> Clock   96, Range 3990
Hz 50.112754 --> Clock   96, Range 3991
...
Hz 50.427849 --> Clock   93, Range 4094
Hz 50.415534 --> Clock   93, Range 4095
```

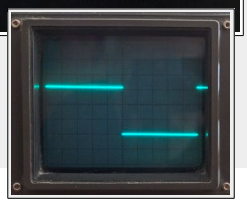
In der letzten Zeile wird die optimale Kombination für Clock/Range angezeigt:

```
Hz 50.000000 --> Clock   96, Range 4000
```

PWM-Mode aktivieren

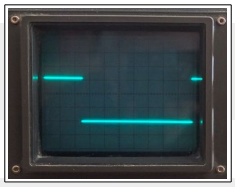

```
void setPWMPinMode(tPWMPin *PWMPin, int8_t PWMMode, int8_t PullUp)
{
  int8_t BcmNr=physPinToGpio(PWMPin->J8Nr);

  PWMPin->BcmNr   =BcmNr;
  PWMPin->PinMode =PWM_OUTPUT;
  PWMPin->PWMMode =PWMMode;
  PWMPin->PullUp  =PullUp;
  PWMPin->Clock   =96;        //50 Hz
  PWMPin->Range   =4000;
  PWMPin->Value   =0xffff;

  pinMode(PWMPin->BcmNr, PWMPin->PinMode);
  pwmSetMode(PWMPin->PWMMode);
  setPWMClockRange(PWMPin, PWMPin->Clock, PWMPin->Range);

  printf("J8 %i --> BCM %i\n",PWMPin->J8Nr, PWMPin->BcmNr);
}
```

Clock und Range setzen:

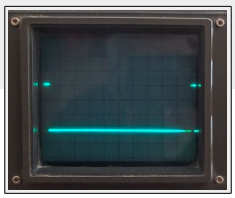


```
void setPWMClockRange(tPWMPin *PWMPin, uint16_t Clock, uint16_t Range)
{ // Wiringpi setzt beide PWM-Pins
  PWMPin->Clock=Clock;  pwmSetClock(PWMPin->Clock); // max 0xfff
  PWMPin->Range=Range;  pwmSetRange(PWMPin->Range);
}
```

Pulsweite setzen: 0 -> immer LOW, 100 --> immer HIGH



```
void setPulsWeite(tPWMPin *PWMPin, int16_t Prozent)
{ double r= (double) PWMPin->Range * (double) Prozent / 100.0;
  PWMPin->Value=(uint16_t) r;
  pwmWrite(PWMPin->BcmNr, PWMPin->Value);
}
```



Servo steuern:

I2C Beispiele

i2c00 Temperatursensor MCP9801

IC2 Temperatursensor MCP9801 abfragen:

```
pi@pi7: ~/c/pi
i2c00:

 I2C Temperatursensor MCP9801 abfragen
 J8-Pins: 3 ist SDA1, 5 ist SCL1
 Kernelmodul wird automatisch geladen:
  gpio load i2c (defaultt 100Kb/sec)

Kernmodul geladen: gpio load i2c 2>/dev/null
I2C 0x4f: using /dev/i2c-1

Temperatursensor MPC9801
 I2C-Id    : 0x4f
 Auflösung: 12 Bits
 Handle    : 3

0x4f: t=27.2°C  (27.1875)
```

Beschreibung MPC9801:

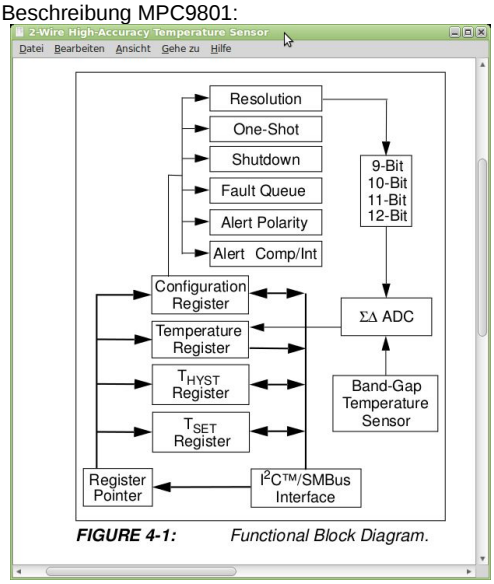


FIGURE 4-1: *Functional Block Diagram.*

Pins:

1 SDA Bidirectional Serial Data
2 SCLK Serial Clock Input
3 ALERT
4 GND
5 A2 Address Select Pin (bit 2)
6 A1 Address Select Pin (bit 1)
7 A0 Address Select Pin (bit 0)
8 VDD Power Supply Input

RegPointer: bit 7-3 0, bit 2-0 , Register Pointer

bit 7	6	5	4	3	2	1	bit 0
0	0	0	0	0	0	Pointer	

Pointer

```
00 = Temperature Register (TempReg)
01 = Configuration Register (ConfigReg)
10 = Temperature Hysteresis Register (HystReg)
11 = Temperature Limit-set Register (LimitReg)
```

TempReg: 2 Byte, Temperature Register

bit 15	14	13	12	11	10	9	8	bit 7	6	5	4	3	2	1	bit 0
Sign	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	0	0	0	0

Temp= code x 2^n mit n = -1 bei 9-bit, -2 bei 10-bit ,-3 bei 11-bit und -4 für 12-bit

Beispiele: TempReg = TempReg >> 4, t = TempReg*0.625

```
+125°C          0111 1101 0000 uuuu          0x7D0    2000      t= 2000 * 0.0625      =  125°C
+25.4375°C      0001 1001 0111 uuuu          0x197     407      t=  407 * 0.0625      =  25.4375°C
−25.4375°C      1110 0110 1001 uuuu          0xE69    -407      t= -404 * 0.0625      = -25.4375°C
−55°C           1100 1001 0000 uuuu          0xC90    -880      t= -808 * 0.0625      = -55 °C
+23.625°C       0001 0111 1010 uuuu          0x17A     378      t=  378 * 0.0625      =  23.625 °C
```

ConfigReg: 1 Byte, Configuration Register

bit 7	6	5	4	3	2	1	bit 0
One-Shot	Resolution		Fault Queue		Alert Polarity	COMP/INT	Shutdown

bit 7:

```
 0 = Disabled ( Default )
```

bit 5-6: Resolution

```
00 =  9 bit  (Default)
01 = 10 bit
10 = 11 bit
11 = 12 bit
```

bit 3-4:

```
00 =  1 ( Default )
```

bit 2:

```
 0 = Activ-Low ( Default )
```

bit 1: COMP/INT

```
 1 = Interrupt Mode
 0 = Comparator Mode ( Default )
```

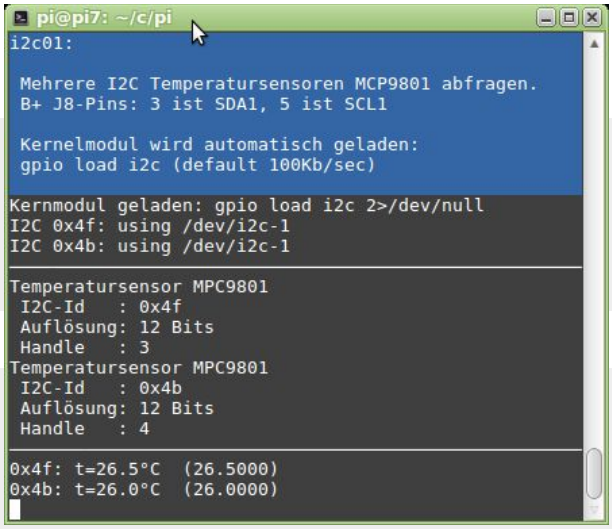
bit 0: SHUTDOWN bit

```
 1 = Interrupt Mode
 0 = Comparator Mode ( Default )
```

Beispiel: 12 Bit Auflösung

```
wiringPiI2CWriteReg8 (fd, 01 ,  96) ;
```

i2c01 Mehrere Temperatursensoren



Siehe auch i2c00.

Variable Daten für MPCF9801:

```
// Variable Daten für MPC9801
typedef struct MPC9801_t
{ uint16_t i2cId;    // I2C Device Id
  int      fd;       // -1 oder Handle

  // Auflösung ------------------------------------
  uint8_t ResConf;   // Auflösung 9,10,11 oder 12 Bit
  float   faktor;    // Umrechnungsfaktor in Grad

} tMPC9801;
```

Konstante Daten für MPC9801

```
// Konstanten für MPC9801 -------------------------------
static tMPC9801_const ICc=
{ .Name      = "MPC9801",  // IC Bezeichnung

  // Register-Adressen
  .TempReg   = 0b00000000, // Temperatur
  .ConfigReg = 0b00000001, // Configuration
  .HystReg   = 0b00000010, // Hysterese
  .LimitReg  = 0b00000011  // Limit
 }; // -------------------------------------------------
```

Inititialisieren:

```
void initMPC9801(tMPC9801 *IC,  uint16_t i2cId, uint8_t Aufloesung)
{
  IC->i2cId=i2cId;

  IC->fd=initI2C(i2cId);  // Filehandle oder Abbruch

  switch (Aufloesung)
  { case  0: IC->ResConf=0b00000000; break; // 9  Bit
    case  1: IC->ResConf=0b00100000; break; // 10 Bit
    case  2: IC->ResConf=0b01000000; break; // 11 Bit
    default: IC->ResConf=0b01100000;        // 12 Bit
  }
  IC->faktor=0.0625;

  // Chip konfigurieren
  if (-1==wiringPiI2CWriteReg8(IC->fd, ICc.ConfigReg, IC->ResConf ) )  { printf("Fehler in initMPC9801()\n"); exit(EXIT_FAILURE); }

} // ---------------------------------------------------------------------------
```

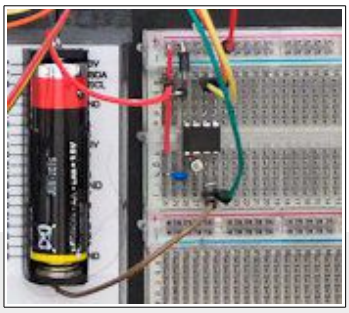
Temperatur lesen. Datenbytes werden in der Reihenfolge BigEndian geliefert.
BigToSysEndian(data) ordnet die Bytes passend zur Systemreihenfolge (Library utils).

```
void printTempMPC9801(tMPC9801 *IC)
{
  uint16_t data=wiringPiI2CReadReg16(IC->fd, ICc.TempReg);

  float t= IC->faktor * (float) (BigToSysEndian(data) >> 4);

  printf("0x%x: t=%.1f°C  (%.4f)\n", IC->i2cId, t, t);
} // ---------------------------------------------------------------------------
```

Main:

```
enum { DEVID1=0x4f, DEVID2=0x4b };

int main(void)
{ clrScr(); printBlock( tmpStrF("%s",Info), FarbeWhiteBlue );

  VarSetGroups(-2,0); VarClrGroups();
  initMPC9801( newStruct( sizeof(tMPC9801)), DEVID1, -1);
  initMPC9801( newStruct( sizeof(tMPC9801)), DEVID2, -1);

  printLine(-1);
  for(tMPC9801 *p=nxtStruct(true); p!=NULL; p=nxtStruct(false))  printInfoMPC9801(p);
  printLine(-1);

  savePos();

  while (1)
  {
    for(tMPC9801 *p=nxtStruct(true); p!=NULL; p=nxtStruct(false))  printTempMPC9801(p);

    fflush(stdout); delay(1000); restPos();
  }

  printf("\n");
  return EXIT_SUCCESS;
}
```

i2c02 Uhr PCF8583



I2c Echtzeituhr PCF8583 abfragen/setzen.
Die Zeitquellen Pi Systemzeit, Uhr und Timeserver synchronisieren.

Diese Funktionen wurden in Programm clock8583 verwendet.
Bitte dort nachschauen!

Variable Daten für PCF8583 in pcf8583.h:

```
// Variable Daten für PCF8583
typedef struct PCF8583_t
{ uint16_t i2cId;       // I2C Device Id
  int      fd;          // -1 oder Handle

  uint8_t CtrlByte;     // Control Register Wert
  //                       --76543210
  #define C0_TimerFlag     0b00000001
  #define C1_AlarmFlag     0b00000010
  #define C2_AlarmEnable   0b00000100
  #define C3_MaskFlag      0b00001000  // 1: Bits für Jahr und Wochentag ausblenden

  #define C45_Mode32KHZ    0b00000000
  #define C45_Mode50HZ     0b00010000
  #define C45_ModeCount    0b00100000
  #define C45_ModeTest     0b00110000

  #define C6_CountFlag     0b01000000
  #define C7_StopCountFlag 0b10000000

  tTime   Time; // struct tm aus time.h
} tPCF8583;
```

Fixe Daten für alle PCF8583 pcf8583.h:

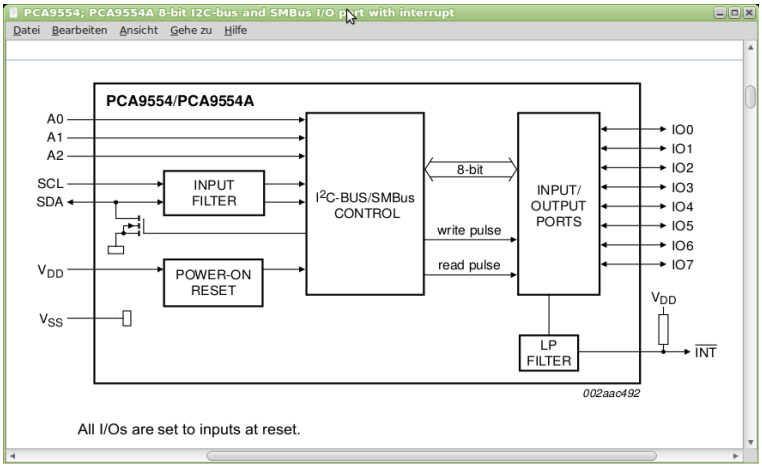
```
static tPCF8583_const ICc=
{ .Name      = "PCF8583"  // IC Bezeichnung

  // Register-Adressen
  #define CtrlReg        0x00  //
  #define Sec100Reg      0x01  //
  #define SecReg         0x02  //
  #define MinReg         0x03  //
  #define HourReg        0x04  //
  #define YearDateReg    0x05  //
  #define wDayMonthReg   0x06  //
  // ------------------------
  #define TimerReg       0x07
  #define AlarmCtrlReg   0x08
  //      ...
  #define AlarmTimerReg  0x0F
  // ------------------------
  //                           freies RAM
  #define YearByteL      0x10  // Jahr Low  Byte
  #define YearByteH      0x11  // Jahr High Byte
  //      RAM bis        0xFF
 };
```

Das aktuelle Jahr wird im freien RAM als uint16_t unter YearByteL gespeichert. Power-on: YearByteL/ YearByteH = 0.
Im Register YearDateReg ist (Jahr % m) = 0, 1, 2, 3 gespeichert. 0 ist das Schaltjahr.

Für die Verarbeitung von Datum/Zeit werden die vorhanden Linux Funktionen verwendet.
Für diese Funktionen steht im Modul datetime.h ein Interface bereit.

i2c03 Expander PCA9554

fehlt noch

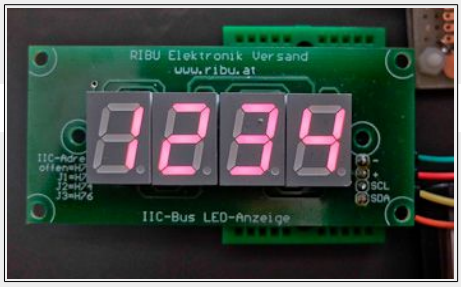


i2c04 7-Segmentanzeige

7-Segmentanzeige mit Chip SAA1064 ansteuern.

Variable Daten für SAA1064:

```
// Variable Daten für SAA1064
typedef struct SAA1064_t
{ uint16_t i2cId;     // I2C Device Id
  int      fd;        // -1 oder Handle
  uint16_t CtrlByte;  // Ansteuerungsmodus

  // Bits für CtrlByte
  # define C0_Dynamic       0b00000001 // 1: dynamic Mode, 0: static Mode,
  # define C1_notBlanked13  0b00000010 // 1: Digit 1+3 not blanked, 0: Digit 1+3 blanked
  # define C2_notBlanked24  0b00000100 // 1: Digit 2+4 not blanked, 0: Digit 1+3 blanked
  # define C3_Testmode      0b00001000 // 1: all on, Testmodus
  # define C4_3mA           0b00010000 // 1:  3mA pro Segment
  # define C5_6mA           0b00100000 // 1:  6mA pro Segment
  # define C6_12mA          0b01000000 // 1: 12mA pro Segment

} tSAA1064;
```



Fixe Daten für SAA1064:

```
static tSAA1064_const ICc=
{ .Name      = "SAA1064",  // IC Bezeichnung

  // Register-Adressen , Auto-Increment
  .CtrlReg   = 0b00000000, // Controlregister
  .Digit1Reg = 0b00000001, // Digit 1
  .Digit2Reg = 0b00000010, // Digit 2
  .Digit3Reg = 0b00000011, // Digit 3
  .Digit4Reg = 0b00000100, // Digit 4
 };
// ------------------------------------------------

// LED's einer 7-Segmentanzeige
#define a 0b00000001
#define b 0b00000010
#define c 0b00000100
#define d 0b00001000
#define e 0b00010000
#define f 0b00100000
#define g 0b01000000
#define punkt 0b10000000
#define minus 0b01000000

uint8_t LED[]=
{ a|b|c|d|e|f  ,   // 0
    b|c        ,   // 1
  a|b|  d|e  |g,   // 2
  a|b|c|d|    g,   // 3
    b|c|    f|g,   // 4
  a|  c|d  |f|g,   // 5
  a|  c|d|e|f|g,   // 6
  a|b|c        ,   // 7
  a|b|c|d|e|f|g,   // 8
  a|b|c|d|  f|g,   // 9
  a|b|c|  e|f|g,   // A
      c|d|e|f|g,   // b
  a|    d|e|f  ,   // c
    b|c|d|e|  g,   // d
  a|    d|e|f|g,   // e
  a|      e|f|g    // f
};
```

LES's einer 7-Segmentanzeige schalten:

```
void writeLEDsSAA1064(tSAA1064 *p, uint8_t n, uint8_t LEDs)
{ // 7-Segmentanzeige Nummer n schreiben. n=1-4
  if (-1==wiringPiI2CWriteReg8(p->fd, ICc.CtrlReg+n, LEDs) )
  { printf("Fehler in writeDigit_SAA1064()\n"); exit(EXIT_FAILURE); }
}
```

Anzeige löschen:

```
void clrSAA1064(tSAA1064 *p)
{ for(uint8_t i=4; 0<i; i--)  writeLEDsSAA1064(p, i, 0);
}
```

Beispiel uint16_t anzeigen. Führende Nullen unterdrücken:

```
void writeUInt16_SAA1064(tSAA1064 *p, uint16_t wert)
{
  if (wert>9999) { overflowSAA1064(p); return; }

  for(uint8_t i=4; 0<i; i--)
  { uint16_t Digit = wert % 10;
    if ((i==4) || (Digit!=0)) writeLEDsSAA1064(p, i, LED[Digit]);
    else                      writeLEDsSAA1064(p, i, 0);
    wert=wert / 10;
  }
}
```

i2c05 LCD-Display HD47780

LCD-Display im 4-Bitmodus mit PCA9554 über I2C ansteuern.

Anschlüsse:

```
////////////////////////////////////////////////////////////////////////////
// Ansteuerung des LCDs über I2C mit Chip PCA9554
//
//   PCA9554 | LCDs                   | LTN211R
//   Bit-Nr  | Funktion               | Pin-Nr 1-14
//   --------+------------------------+-------------
//           | -                      | 1 GND
//   0 NC    | -                      | 2 +5V
//   1 NC    | -                      | 3 Helligkeit, 0-5V, hell-dunkel
//   2       | RS: 1=Data, 0=Command  | 4 RS Register Select
//           | RW: 0=read, 1=write    | 5 GND, für nur lesen
//   3       | E : 0-1-0 Pulse        | 6 E  Datenübernahme bei 1-0 Flanke
//           | -                      | 7-10 GND, Daten NC
//   4       | Datenbit 4             | 11 Daten
//   5       | Datenbit 5             | 12 Daten
//   6       | Datenbit 6             | 13 Daten
//   7       | Datenbit 7             | 14 Daten
```

Enable Puls 0-1-0 für die Datenübernahme:

```
////////////////////////////////////////////////////////////////////////////
// Erzeugt einen Enable-Puls 0-1-0
static void enableLCD()
{ writePCA9554(IOPort, IOPort->OutBits | (uint8_t) LCD_EN );    // Enable auf 1 setzen
  delayMicroseconds( LCD_ENABLE_US );                           // kurze Pause
  writePCA9554(IOPort, IOPort->OutBits & ~((uint8_t) LCD_EN) ); // Enable auf 0 setzen
}
```

4-Bits ins LCD schreiben:

```
////////////////////////////////////////////////////////////////////////////
// Bits 7,6,5,4 nach LCD schreiben. 4-Bitmodus
static void write4LCD( uint8_t data, uint8_t RS )
{ // RS = 0     : Befehl schreiben
  // RS = LCD_RS: Daten schreiben
  IOPort->OutBits = (IOPort->OutBits & ~((uint8_t) (LCD_PINS) )) | ((data & LCD_DB) | RS);
  enableLCD();
}
```

Befehlsbyte schreiben:

```
////////////////////////////////////////////////////////////////////////////
// Befehlsbyte an das LCD schreiben
void cmdLCD( uint8_t data )
{
  write4LCD( data    , 0);             // zuerst die oberen 4 Bits
  delayMicroseconds( LCD_ENABLE_US );  // kurze Pause
  write4LCD( data<<4 , 0);             // die unteren 4 Bits senden
}
```

Datenbyte schreiben:

```
////////////////////////////////////////////////////////////////////////////
// Datenbyte an das LCD schreiben
void printLCD( uint8_t data )
{
  write4LCD( data    , LCD_RS);          // zuerst die oberen 4 Bits
  delayMicroseconds( LCD_WRITEDATA_US ); // kurze Pause
  write4LCD( data<<4 , LCD_RS);          // die unteren 4 Bits senden
}
```

```
pi@pi7: ~/c/pi
i2c05:

 LCD HD47780 mit I2C 8-Bit I/O Port PCA9554A ansteuern.
 Testmodus im makefile mit -DTest einschalten

 B+ J8-Pins: 3 ist SDA1, 5 ist SCL1

 Kernelmodul wird automatisch geladen:
 gpio load i2c (default 100Kb/sec)

Testmodus: aus!
 Flag 'Test' im makefile setzen: CFLAGS = ... -DTest  # Test ein

PCA9554 initialisieren
.................................................
Kernlmodul geladen: gpio load i2c 2>/dev/null
I2C 0x3c: using /dev/i2c-1
I2C 8-Bit IO Port PCA9554
 I2C-Id      : 0x3c
 Handle      : 3
 Input Pins  : 0b 0000 0011
 Invertiert  : 0b 0000 0000
.................................................
LCD initialisieren

Zeichen ausgeben      :  printLCD('x')
Goto Zeile 2, Spalte 1:  gotoYXLCD( 2, 1 )
String ausgeben       :  printStrLCD("Hello World!")
gotoYXLCD( 1, 5 )
```

GNU General Public License

```
/*
 *
 * Copyright 2016 Günther Schardinger <v.schardinger@gmx.net>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
 * MA 02110-1301, USA.
*/
```