Die seriellen Datenübertragung zwischen µC und PC

Der Datentransfer zwischen PC und µC erfolgt binär und nach einem festen Protokoll:

Jede Übertragung, gleichgültig in welcher Richtung, beginnt mit einem 3-Byte langen Header bestehend aus:
- dem konstanten Startbyte,
- dem Frametyp (die Angabe des angeforderten "Dienstes")
- der Anzahl der nachfolgenden Datenbytes.
- Es folgt mindestens 1 Datenbyte.

Startbyte, Frametyp, Anzahl Datenbytes, DatenByte0, [DatenByte1 ....]

Folgende Frametyps sind definiert (als Konstanten in global.h / global_h.py) :

- TWOWIRE   : TWI-Read/TWI-Write
- TWISPEED : TWI-SCL-Takt verändert
- ONEWIRE   : DS18B20 Temperatursensoren an PD.2 ... PD.7 auslesen
- ADCREAD   : ADC an PC.0 ... PC.3 auslesen
- INTMODE   : Interrupt an Portpins aktivieren
- LOGMODE   : als TWI-Slave empfangene Daten werden weitergereicht
- PORTMODE : Pin Mode setzen (freie Pins an PortB, PortC und PortD)
- PORTREAD : Pin Status auslesen (DDR-, PORT- und PIN-Register)
- PORTWRITE : Pin Status setzen (freie Pins an PortB, PortC und PortD)
- PORTTOG   : Pin Status togglen (freie Pins an PortB, PortC und PortD)

1. TWI-Übertragung an einen TWI-Slave, der am TWI-Bus des µC angeschlossen ist

Die Adressen der TWI-Slaves werden links ausgerichtet erwartet,
die Adressen zum TWI-Write sind immer geradzahlig [0, 2, 4 .. 252, 254].
Für ein TWI-Read wird das RW-Bit (Bit.0) gesetzt (= ungradzahlige Adresse).

TWI-Write (der PC sendet an einen am µC angeschlossenen TWI-Slave)
STARTBYTE, TWOWIRE, Anzahl Bytes, TWI-Adresse, (Register_Adr,) Byte0, Byte1, ...

0xA5, 0xE1, 0x02, 0x40, 0x10

(Sendet an den TWI-Slave mit der Adresse 0x40 das Datenbyte 0x10)

Rückmeldung: keine - ausser bei einem Fehler:
STARTBYTE, TWI_ERR, 0x01, TWI-Adresse (bei deren Adressierung ein Fehler aufgetrat).

0xA5, 0x5A, 0x01, 0x40

(Fehler bei Write an TWI-Adresse 0x40)

Ob die Datenbytes bei einem TWI-Write Adressen (Register) oder Daten darstellen, das ist abhängig vom TWI-Slave.
Ein Portexpander PCF8574 kennt keine interne Register, ein RTC PCF8583 hat diverse, jeweils mit einem einzigen Byte zu adressierende Register, ein Eeprom benötigt zur Adressierung des Speichers 2 Byte.

TWI-Read (der PC liest Daten von einem am µC angeschlossenen TWI-Slave)
Bei einem TWI-Read muss das R/W-Bit in der TWI-Adresse gesetzt werden !
Startbyte, TWOWIRE, Anzahl Bytes, (TWI-Adresse | 0x01), Anzahl zu lesender Bytes

0xA5, 0xE1, 0x02, 0x41, 0x01

(Sendet ein Read an den TWI-Slave mit der Adresse 0x41 und fordert 1 Byte an)

Der µC liefert auf ein TWI-Read die Daten aus:
STARTBYTE, TWOWIRE, Anzahl Bytes, sendende TWI_Adresse, DatenByte0, [DatenByte1 ....]

0xA5, 0xE1, 0x02, 0x41, 0xFF

(Der Slave 0x41 gibt den Wert 0xFF zurück.)

Bei einem TWI-Fehler antwortet der µC:
STARTBYTE, TWI_ERR, 0x01, TWI-Adresse (bei deren Adressierung ein Fehler aufgetrat).

0xA5, 0x5A, 0x01, 0x41

(Fehler bei Lesen von TWI-Adresse 0x41)

Vor einem TWI-Read muss häufig das Register/der Adressbereich auf dem TWI-Slave ausgewählt werden, vom dem aus gelesen werden soll.
Das TWI-Read unterstützt daher die Option, vor dem Byte, das die Anzahl der zu lesenden Datenbytes festlegt, ein oder zwei Bytes als Registeradresse zu übertragen.
Diese Adressangabe werden zuerst per TWI-Write an den Slave gesendet, mit einem Repeated Start folgt dann der lesende Zugriff auf die angeforderten Daten.
STARTBYTE, TWOWIRE, Anzahl Bytes, (TWI-Adresse | 0x01), Register_Adr, Anzahl Bytes

0xA5, 0xE1, 0x03, 0xA1, 0x02, 0x03

(Sendet zuerst an den TWI-Slave 0xA0 das Byte 0x02 und liest anschließend von der TWI-Adresse 0xA1 3 Byte aus.)
STARTBYTE, TWOWIRE, Anzahl Bytes, (TWI-Adresse | 0x01), Adr_high, Adr_low, Anzahl Bytes

0xA5, 0xE1, 0x04, 0xA3, 0x00, 0x80, 0x10

(Sendet zuerst an den TWI-Slave 0xA2 die Adressbytes 0x00 und 0x80 und liest anschließend von der TWI-Adresse 0xA3 16 Byte aus.)

TWI-Takt
Der TWI-Takt ist nach einem Reset auf 102.400 KHz (TWBR = 0x0A) voreingestellt.
Er kann bei Bedarf reduziert werden auf ca. 50KHz, 25KHz oder 12KHz.
Startbyte, TWISPEED, Anzahl Bytes, [100, 50, 25, 12]

0xA5, 0xEA, 0x01, 0x0C

(Der TWI-Takt wird auf 12KHz eingestellt.)

Der µC liefert den aktuellen Wert des Registers TWBR zurück:
STARTBYTE, TWISPEED, Anzahl Bytes, TWBR-Register

0xA5, 0xEA, 0x01, 0x91

Die SCL-Frequenz kann berechnet werden mit der Formel:
SCL frequency = 3.686.400 / (16 + 2 * (TWBR))

Beschränkungen in der Datensatzlänge:
Die Übertragung der Daten erfolgt über 2 Ringpuffer der Länge 256 Bytes, von denen jedoch 3 Byte für den Header benötigt werden. Empfehlung: Datenlänge auf 128 Byte beschränken !

Einschränkungen bei TWI-Read
Während einer aktiven Übertragung können bereits neue Jobs im Ringpuffer aufgenommen werden, der Empfang erfolgt Interrupt-gesteuert im Hintergrund.
Neue Jobs werden aber erst dann abgearbeitet, wenn alle älteren abgeschlossen sind.

Dadurch ergibt sich ein potentielles Problem:
Werden die neuen Jobs auf Dauer schneller gesendet, als sie abgearbeitet werden können, dann wird der Empfangs-Ringpuffer früher oder später von hinten her "überfahren".

Kritisch werden können Read-Aktionen, wenn mehr Daten empfangen als gesendet werden:
Etwa, wenn 36 Jobs zum Lesen von Eeprompages je 128 Byte gesendet werden
(36 * 7 Byte = 252 Byte), dann müssen 36 * (128 + 4) Byte = 4.752 Byte zurückgesendet werden. Wenn hier weitere Jobs gesendet werden, dann droht Ärger.

Um dieses Problem zu vermeiden, muss (bei einer größerer Anzahl umfangreicher Read-Aktionen) nach jedem TWI-Read bis zum Eintreffen der Daten gewartet werden, erst dann darf der nächste Job zum Lesen abgeschickt werden.

Das TWI-Write ist unkritisch, da der TWI-Bus ca. 10x schneller ist als die Serielle Schnittstelle (bei der Einstellung 9600 Baud).

2. Temperatursensoren DS18B20 auslesen (angeschlossen an PortD des µC)

Dienst DS18B20 auslesen
Die Sensoren müssen an PD.2 ... PD.6 angeschlossen sein. Die PinMaske legt fest, an welchen Pins Sensoren ausgelesen werden sollen.
Die Temperatur wird in 1/10° zurückgeliefert.
STARTBYTE, ONEWIRE, Anzahl Bytes, PinMaske

0xA5, 0xE2, 0x01, 0x0C

(es werden die Sensoren an PD.2 und PD.3 ausgelesen.

Die Rückgabe der Messwerte erfolgt automatisch (nach ca. 800ms):
STARTBYTE, ONEWIRE, Anzahl Bytes, PinMaske, t1_high, t1_low, t2_high, t2_low, ...

0xA5, 0xE2, 0x05, 0x0C, 0x00, 0xFF, 0x01, 0x04

(Sensor an PD.2: 25.5°, Sensor an PD.3: 26.0°)

Wird an einem abgefragten Pin kein Sensor gefunden, dann wird der Wert -999 zurückgegeben.

Werden in der PinMaske Pins definiert, an denen keine Sensoren angeschlossen werden können, dann wird eine Fehlermeldung zurückgegeben:
STARTBYTE, OWI_ERR, Anzahl Bytes, PinMaske

0xA5, 0x6A, 0x01, 0x03

(An den Pins PD.0 und PD.1 kann kein Sensor angeschlossen werden -> RxD, TxD)

Hinweis:
Der µC sendet das Ergebnis der Temperaturmessung nach ca. 800ms an den PC.
Wird innerhalb dieser Zeitspanne ein neues Kommando zur Messung gegeben, dann wird für die erste Messung kein Messwert ausgeliefert.
Das bedeutet, dass als Messintervall eine Zeit von mindestens 800ms gewählt sein sollte !

3. ADC auslesen (PortC)

ADC auslesen:
Die freien, auslesbaren Pins sind PC.0 .. PC.3.
STARTBYTE, ADCREAD, Anzahl Bytes, Pin-Nr. [0, 1, 2, 3]

0xA5, 0xE3, 0x01, 0x03

(Den ADC an PC.3 auslesen)

Rückgabe:
STARTBYTE, ADCREAD, Anzahl Bytes, Pin_Nr, adc_high, adc_low

0xA5, 0xE3, 0x03, 0x03, 0x01, 0x02

(Messwert an PC.3 = 258)

Wird ein Pin mit einer PinNr. > 3 adressiert, dann wird eine Fehlermeldung gesendet.
STARTBYTE, ADC_ERR, Anzahl Bytes, adressierter Pin

0xA5, 0xE3, 0x01, 0x04

(es wurde PC.4 adressiert, an diesem Pin wird aber SDA bedient)

Der ADC-Eingang sollte vor der Messung als Eingang konfiguriert und der Pullup-Widerstand sollte deaktiviert sein.
Arbeitet der ADC-Pin als Ausgang, dann wird als Messwert 0 oder 1023 geliefert.
Auch ein gesetzter Pullup wird voraussichtlich zum Messwert 1023 führen.

4a. Interrupt-Eingang einrichten (INT0 / INT1)

PD.2 / PD.3 für Interrupt-Eingang einrichten:
Als Interrupt-Pins sind INT0 und INT1 (PD.2, PD.3) gewählt.
STARTBYTE, INTMODE, Anzahl Bytes, Pin, Sense
Pin = [0x00 = INT0, 0x01 = INT1]
Sense = [0x00 = LOWLEVEL, 0x01 = ANYCHANGE, 0x02 = FALLINGEDGE, 0x03 = RISINGEDGE]
Wenn für Sense einen Wert > RISINGEDGE eingetragen wird, dann wird der Interrupt deaktiviert.

0xA5, 0xE4, 0x02, 0x01, 0x02

(INT1 wird auf FallingEdge eingestellt)

Ein ausgelöster Interrupt meldet sich dann mit:
STARTBYTE, INTMODE, Anzahl Bytes, INTx, Sense, PinStatus von INTx

0xA5, 0xE4, 0x03, 0x01, 0x02, 0x00

(INT1 hat ausgelöst, Sense ist FALLINGEDGE, der Pin-Status ist low)

Die Interrupt-Routine auf dem µC löscht die Interrupt-Einstellung nach jedem Interrupt!
Die Routine, die am PC den Interrupt bearbeitet, muss daher ggf. den Interrupt neu setzen!
Noch einmal: Jeder Interrupt wird nur 1 einziges Mal ausgelöst, die Interrupt-Routinen für INT0 und INT1 löschen das jeweilige Interrupt_Enable Flag !
Ein Interrupt wird deaktiviert, indem als Sense 0xFF gesendet wird (oder ein sonstiger Wert > 0x03).

0xA5, 0xE4, 0x02, 0x00, 0xFF

(Deaktiviert den Interrupt INT0.)

4b. Interrupt-Eingang einrichten (PCINT)

Beliebige verfügbare Pins für einen PinChange Interrupt einrichten:
Als Interrupt-Pins sind alle verfügbaren Pins an allen Ports einsetzbar.
Es wird grundsätzlich auf einen PinChange (Flankenwechsel) reagiert.
STARTBYTE, INTMODE, Anzahl Bytes, Port, Pinmaske
Port = [0x10 = PortB, 0x20 = PortC, 0x40 = PortD]
Pinmaske = für jeden Pin des Ports, der einen Interrupt auslösen soll, muss das zugehörige Bit gesetzt werden.
Eine Pinmaske mit dem Wert 0 löscht den Interrupt.

0xA5, 0xE4, 0x02, 0x40, 0x0C

(PD.2 und PD.3 werden auf PinChange eingestellt)

Ein ausgelöster Interrupt meldet sich dann mit:
STARTBYTE, INTMODE, Anzahl Bytes, INTx, Pinmaske, Pinstate

0xA5, 0xE4, 0x02, 0x40, 0x0C, 0x04

(PinChange-Interrupt an PortD, PD.2 = high, PD.3 = low)

Im Gegensatz zu den INT0/INT1-Interrupts werden die PinChange-Interrupts nicht automatisch gelöscht !
Sie eignen sich daher besonders für nicht prellende, elektronische Schalter (z.B. Bewegungsmelder).

PinChange Interrupts an einem Port werden gelöscht durch die Pinmaske 0:
STARTBYTE, INTMODE, Anzahl Bytes, Port, Pinmaske

0xA5, 0xE4, 0x02, 0x40, 0x00

Löscht den PinChange-Interrupt an allen Pins von PortD

5. Daten via TWI als TWI-Slave empfangen

Der µC arbeitet im Hintergrund als TWI-Slave.
Ein externer Master kann an die Adresse des Slaves (default: 0x04) Daten senden - z.B. zum Debuggen oder für eine Datenübergabe an den PC).
Diese Option wird benutzt um in einer Umgebung, in der die Serielle Schnittstelle belegt, aber ein TWI-Bus vorhanden ist.
Hier werden Debugging bzw. Protokolldaten vom Master des TWI-Bus an den Slave gesendet.
Der wiederrum sendet die Daten über seine Serielle Schnittstelle an den PC weiter.

Der Einsatz des "Rasp"Mega in diesem Slave-Modus ist nicht kompatibel mit dem TWI-Modus aus 1.), da keine Multi-Master-Fähigkeit implementiert ist.

Das empfangen Frame (auf der PC-Seite) sieht so aus:
STARTBYTE, LOGMODE, Anzahl Bytes, Byte0, Byte1 .....

0xA5, 0xE5, 0x05, 0x45, 0x52, 0x52, 0x4F, 0x52

(Der Slave hat die Zeichenfolge "ERROR" empfangen und an den PC weitergereicht.)

Beschränkung:
Der Slave arbeitet mit einem Ringpuffer der Länge 256 Byte. Da noch etwas Verwaltungs-overhead erforderlich ist, dürfen nie mehr als 250 Bytes übertragen werden !

Während der Weiterleitung der Daten an den PC ist der TWI-Slave inaktiv und nimmt keine neuen Daten an!

6. I/O-Port-Pins des µC konfigurieren, setzen und auslesen

Die Ports werden über die Nummer 0 ... 3 angesprochen (PortA = 0, PortB = 1, PortC = 2, PortD = 3), die Pins über die Nummern 0 ... 7.
PortA ist auf dem ATmega88 nicht vorhanden.

PortMode (Pins eines Ports als Ausgang oder Eingang setzen):
STARTBYTE, PORTMODE, Anzahl Bytes, Port, PinMaske, IOMaske

0xA5, 0xE6, 0x01, 0x07, 0x01

(An PortB wird PB.0 als Ausgang und PB.1/PB.2 als Eingang gesetzt)

Rückgabe: keine

PortRead (den aktuellen Status aller Pins eines Ports auslesen)
STARTBYTE, PORTREAD, Anzahl Bytes, Port

0xA5, 0xE7, 0x01, 0x01

(Liest den PortB aus)

Rückgabe:
STARTBYTE, PORTREAD, Anzahl Bytes, Port, DDR-Register, PORT-Register, PIN-Register

0xA5, 0xE7, 0x04, 0x01, 0x07, 0x01, 0x01

(An PortB sind:
DDR -Register : PB.0, PB.1 und PB.2 als Ausgang gesetzt, der Rest als Eingang
PORT-Register : PB.0 ist High, PB.1 ... PB.5 sind low
PIN -Register : PB.0 ist High, PB.1 ... PB.5 sind Low)

PortWrite (AusgangsPins auf high oder low setzen):
STARTBYTE, PORTWRITE, Anzahl Bytes, Port, PinMaske, PinStatus

0xA5, 0xE8, 0x03, 0x01, 0x07, 0x01

(Setzt an PortB PB.0 auf High und PB.1 und PB.2 auf Low)

Rückgabe: keine

PortToggle (einzelne Pins an einem Port umschalten)
STARTBYTE, PORTTOG, Anzahl Bytes, Port, Pinmaske

0xA5, 0xE9, 0x02, 0x01, 0x03

(An PortB werden die Pins PB.0 und PB.1 umgeschaltet)

Rückgabe: keine

Hinweis:
Programmtests habe mit HTerm ausgeführt, die Eingabe erfolgte mit den Einstellungen bei Input Control -> Send on enter: None (es darf kein Return an die Daten angehängt werden !)

16.08.2013

Michael S.