Daten via Serielle Schnittstelle empfangen aus dem Powerdown-Modus heraus

Im Powerdown-Modus nimmt ein Controller wie der ATtiny2313 nur eine sehr geringe Leistung auf. Das ist schön.

Weniger schön ist, dass er in diesem Modus nicht so ohne weiteres über die Serielle Schnittstelle ansprechbar ist. Denn für die asynchrone Datenübertragung muss das Timing genau stimmen, so dass ein externer Quarz erforderlich ist.
Der benötigt einige Zeit, um einzuschwingen - zu viel Zeit, um die eingehenden Bytes mitzulesen.

Darum habe ich den Powerdown-Modus bislang nicht genutzt, wenn auch die Serielle Schnittstelle benötigt wurde.

Nun hatte ich kürzlich einen Hinweis gelesen, dass man mit einem Trick die oben angeführte Klippen umschiffen kann:
Angeblich sollte man durch das Senden von 0xFF (oder 0x00 ?) den Controller aus dem Sleep wecken und anschließend seriell mit ihm kommunizieren können.

Das wollte ich mir mal genauer ansehen.

Aber bei den ersten Tests arbeitete der Controller absolut nicht so wie erwartet.

Darum habe ich einen Logicanalyser angeschlossen und im Programmcode einige Spione eingebaut, die mir den aktuellen Zustand des Controllers verraten:

1.)
Bevor der Controller in den Sleep-Modus fällt, zieht er PB0 auf low.
Die 2. Aktion nach dem Erwachen aus dem Sleep ist, den PB0 wieder auf high zu ziehen.

2.)
Um zu sehen, ob der Controller arbeitet, ist der Timer0 für den CTC-Mode konfiguriert, das Compare-Register ist mit dem Wert 184 belegt.
Bei einem CPU-Takt von 3.686400 Hz sollte damit an PB2 (OC0A) ein Takt von ca. 10 kHz anliegen (3.686.400 / (184 * 2)).

Das ist der "Puls" des Controllers.

3.)
Der RX-PIN (PD0) und der TX-PIN (PD1) werden ebenfalls mitgeschnitten.
Am RX-PIN sieht man die eingehenden Daten (die vom PC aus an den Controller gesendet werden), am TX-PIN das, was der Controller an den PC zurücksendet (die Konstante 0x5A).

4.)
Der RX-PIN ist gleichzeitig mit dem PCINT1 (PB1) verbunden.
Der PCINT1 erkennt die fallende Flanke am RX-PIN und beendet den Sleep-Modus.

Der relevante Programmausschnitt um das "sleep" herum sieht so aus:

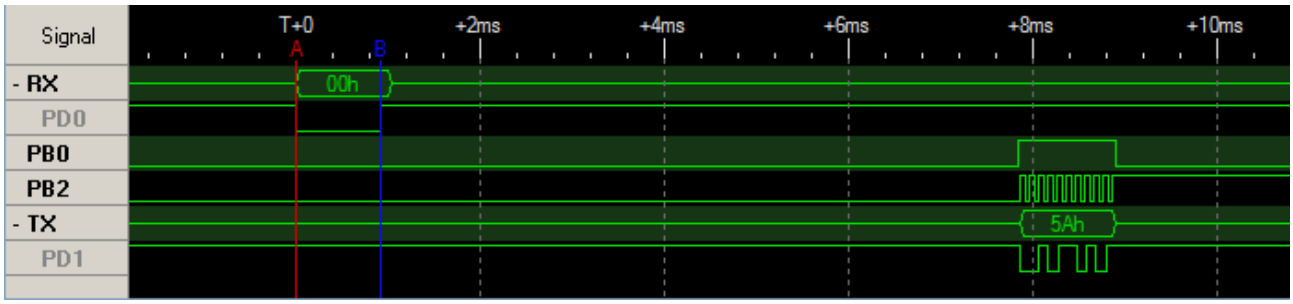
```
MCUCR  = (1<<SE | 1<<SM1 | 1<<SM0); GIMSK  = (1<<PCIE);
PCMSK  = (1<<PCINT1);
PORTB &= (~(1<<PB0));

asm volatile("sleep");

GIMSK  = 0;
PORTB |= (1<<PB0);
put_c(0x5a);
```

Unmittelbar vor dem Sleep wird PB0 auf low gesetzt.

Vom PC aus wird nun ein 0x00 gesendet (siehe Zeile RX).

Interessant ist nun, wann PB0 auf high geht. Und wann an PB2 das 10kHz Signal anliegend.

Nun die folgende Abbildung zeigt es:



Nach der fallenden Flanke an RX tut sich erst einmal gar nichts.

Nach etwa 8ms geht PB0 auf high, gleichzeitig arbeitet der Timer0 und es wird ein 0x5A gesendet. Nachdem das Flag TXC in UCSRA gesetzt ist (alle Datenbytes also versandt sind) wird wieder in den Sleep-Modus zurückgefallen.

Was heißt das nun praktisch ?

1.)
Man kann den Controller mit einem (beliebigen) Byte aus dem Sleep wecken, wartet eine definierte Zeit (im obigen Beispiel vielleicht 10ms) und nimmt dann die Verbindung auf.

2.)
Man weckt den Controller mit einem Byte, wartet auf seine Rückmeldung und schickt dann die Daten. Das ist der sichere Weg.

In beiden Fällen muss der Controller nach dem Aufwecken natürlich noch eine hinreichend lange Zeit auf eingehende Bytes warten (was im Beispiel oben nicht tut).

In den Fuses kann man für die Taktquelle unterschiedliche "Einschwingzeiten" auswählen. Allerdings konnte ich keine Veränderung im Startverhalten des ATtiny2313 erkennen, das Timing schien unabhängig von den gewählten Einstellungen zu sein.

23.01.2016

mfg

Michael S.