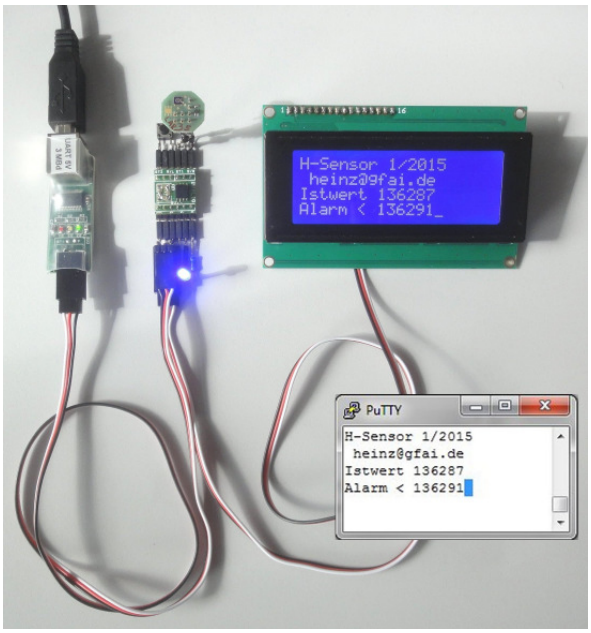
Single-Wire UART (SWART)

Einfacher kann man nicht kommunizieren!

Teil1
Prinzip und Schaltungstechnik

Gerd Heinz, GFal Berlin

Noch vor SPI und I²C stellen Universal Asynchronous Receicer Transmitter (UART) das Fundament aller Kommunikation dieser Erde dar. Sie sind aus dieser Welt nicht mehr wegzudenken. Der UART als dem abgespeckten Kern der "Seriellen Schnittstelle" stehen bewährte Softwaretools zur Verfügung. Die 8-Bit Variante der UART kommt UNIX-, LINUX- wie DOS-artigen Programmen entgegen, Tools wie Hyperterminal, Putty, Brayterm oder Hterm stehen zur Verfügung. Praktisch jede Kommunikation außer UTP baut auf UART und ASCII, man denke an Telnet, Email, SSH, HTML etc. Nicht zuletzt leisten schon kleinste Logikanalysatoren (Saleae, USBee u.a.) eine perfekte ASCII/Hex-Decodierung des seriellen Datenstroms. Zur Verfügung stehende Hilfsmittel für UART-artige Protokolle sind weltweit verbreitet. USB-VCP-Tunnel bilden eine einfache Chance, vom PC aus mit UART-Geräten zu kommunizieren. Insbesondere im Bereich einfacher und lokaler Kommunikationssysteme ist das zeitgleiche Gegensenden (full-duplex) auf den RX- und TX-Leitungen vermeidbar. Kommt man mit dem Prinzip "entweder senden oder empfangen" (simplex) aus, wird der Übergang auf die hier erstmals vorgestellte Eindraht-UART Erleichterung und viel Freude mit sich bringen.



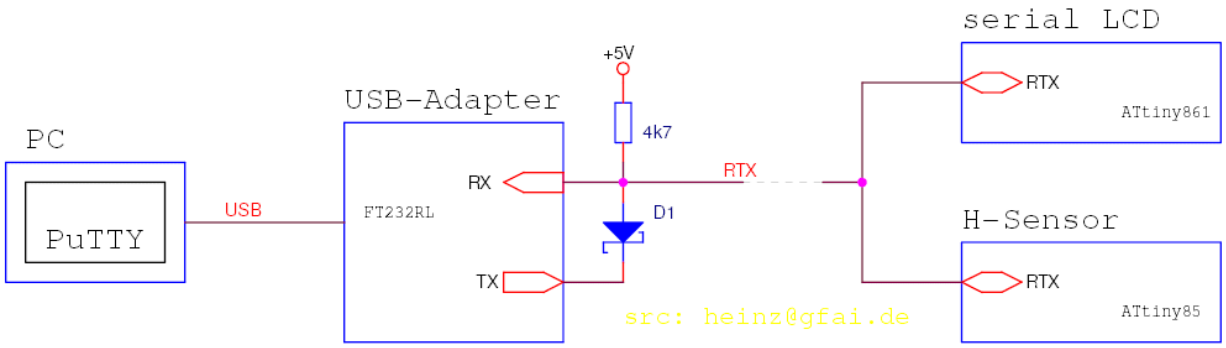


Abb.1: Erster SWART-Bus mit drei Geräten. Putty und LCD zeigen die Meßwerte des H-Sensors an. Putty kann Eingaben an den H-Sensor machen oder Parameter des LCD abfragen. Damit diese drei Geräte kommunizieren können, ist nicht einmal ein Protokoll nötig. Einstellbar sind hier 9600 bis 115200 Baud, 8n1n. Historisch gewachsen befindet sich die Schottky-Diode D1 im USB-Adapter, der Pull-Up Widerstand am H-Sensor. Zur Verdrahtung dienen Servo-Verlängerungskabel.

Vernetzt man nun Sensoren/Aktoren und Geräte mit UARTs, die heute in fast jedem Mikrocontroller angeboten werden, so kommt es vor, daß irgendwann ein Modul statt als UART-Master als UART-Slave genutzt werden soll. Plötzlich stehen sich am Stecker RX und RX sowie TX und TX gegenüber. Die Pins für RX und TX liegen verkehrt herum, ein Hardware-Redesign oder eine Art Null-Modem-Kabel wird erforderlich. Hat man sich darüber einige Male geärgert, sinnt man auf intelligentere Lösungen. Will man die Gefahr von Adaptersteckern zur Umpolung vollkommen vermeiden, empfiehlt sich die Nutzung der bidirektionalen Variante (2-Draht) der RS485. Aber es bleiben zwei Drähte.

Für den Bereich preiswerter, miniaturisierter Sensorik und Aktorik (Servos) im Bereich kurzer Wege (typisch kleiner einem Meter) benötigt man oft einfachste Lösungen zur Kommunikation über nur einen Draht. Aber Lösungen, wie die One-Wire Schnittstelle oder der iButton kommen mit einem exotischen Protokoll daher und können i.a. nur mit spezieller Software und mit speziellen Controllern codiert oder decodiert werden.

Als abgespeckte "Serielle Schnittstelle" hat die UART hingegen nach wie vor größte Popularität im Bereich des Embedded Designs. Sendet man mit zwei Stopbits, empfängt man aber nur mit einem Stopbit, dann genügen sogar interne RC-Oszillatoren im Mikrocontroller, um im vollen Temperaturbereich fehlerfrei kommunizieren zu können. Somit sind auch Applikationen auf den kleinsten Prozessoren (z.B. ATtiny4...10) möglich.

Was also läge näher, als unsere UART von zwei Leitungen auf eine Leitung zu reduzieren? Ein Blick auf Abb.2 zeigt, daß dies unkompliziert möglich ist. Mit zwei kleinen Schottkydioden und einem Widerstand können wir loslegen. Obwohl der low-Pegel durch die Dioden um weniger als 0,3 Volt angehoben wird, funktioniert die Lösung im Experiment tadellos. Verwendet man zum Beispiel den Typ TSS42U im Gehäuse 0603, so fällt bei 1mA weniger als 0,28V über der Diode ab. Bei einer Reverse Voltage von 1V ist die Kapazität mit 10 pF angegeben. Damit wäre die Prinziplösung sogar marktfähig.

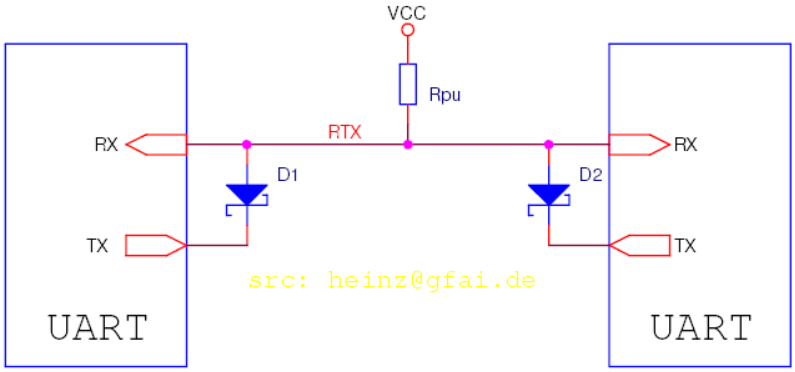


Abb.2: Prinzip einer Single Wire Asynchronous Receiver/Transmitter (SWART) Schnittstelle. Der high-Pegel wird von einem Pull-Up-Widerstand gezogen. TX wird über Schottky-Dioden an RX angeschlossen und zieht nur beim Bitwert "low" nach unten.

Es entsteht eine Kommunikation ("simplex"), die Eigenschaften der UART-Variante der 2-Draht RS485 teilt: Gegensenden ist nicht möglich und beim Senden eines Byte sollte man vorab den eigenen Empfänger ausschalten. Gegenüber der RS485 allerdings ist sie harmlos: Wird versehentlich gegengesendet, werden nur Bits gestört, es brennt nichts durch.

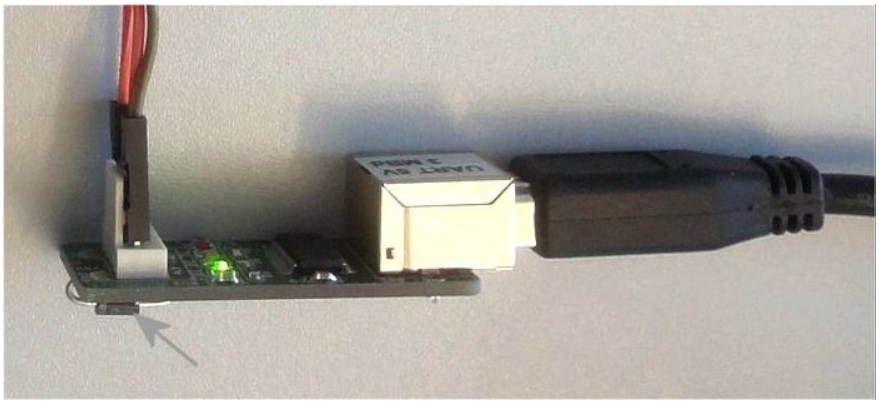


Abb.3: Versuchsaufbau einer SWART als Erweiterung eines USB-VCP Adapters (Basis FTDI FT232RL).

In einem ersten Versuch (Abb.2) wurde ein vierpoliger Stecker eines USB-VCP-Adapters (GND, VDD, TX, RX) mit einer Schottkydiode (Pfeil) zur SWART umfunktioniert. Es kam eine eigentlich zu große Schottkydiode mit hoher Kapazität zum Einsatz. Aber selbst die tut es.

Realisierung mit "weichen" UARTs

Unsere kleinsten Mikrocontroller (z.B. ATtiny) haben meist keine UART als Baugruppe. Hier haben wir die UART per Software nachzubilden. Versuchen wir, die Bedingungen für die Programmierung des UART-Portpins auszuloten, machen wir eine interessante Entdeckung. Das Prinzipbild Abb.1 zeigt schon, daß es nicht nötig ist, den high-Pegel aktiv auszugeben. Den zieht bereits der pull-up Widerstand. Auf das Senden des high-Pegels können wir folglich verzichten. Wird gesendet, besitzt unser Port-Pin nur zwei Zustände. Einerseits haben wir den low-Zustand zu senden, andererseits ist das Senden des high-Zustands überflüssig, stattdessen können wir hier auf Empfang umschalten. Wird nun empfangen, genügt ebenfalls dieser Zustand. Insgesamt kommen wir also mit der Initialisierung zweier Zustände am Portpin aus:

SWART Portpin-Zustände

1. Ausgabe low
2. Empfang

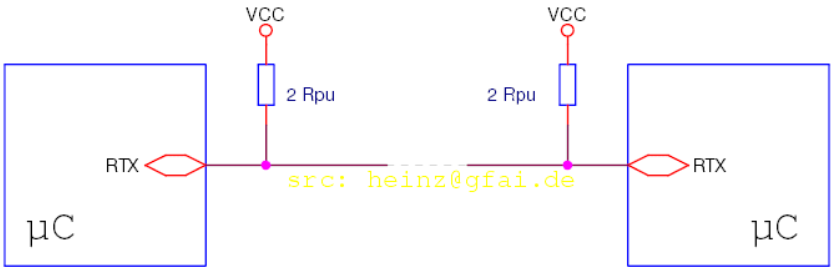


Abb.4: Realisierung einer SWART für Mikrocontroller in Software. Es ist lediglich ein Pullup-Widerstand nötig, der sinnvoll auf zwei Widerstände mit doppeltem Wert aufgeteilt werden kann.

In der Praxis kann die Initialisierung des RTX-Portpins für einen ATtiny85 dann so aussehen (Auszug C-source AVR-Studio4):

```
/* UART Tx/Rx Zustände für Single-Wire UART (Rx und Tx auf einem Pin):
        Uout    Tx sendet low           outport low
        Uout    Tx sendet high          inport mit pullup
        Uin     receive                 inport mit pullup
        --> nur zwei Zustände sind nötig: receive und sendL             */

#define receive { bitclr(Udir, Upin); bitset(Uoport,Upin); } // input pup
#define sendL   { bitset(Udir, Upin); bitclr(Uoport,Upin); } // output low
#define sendH    { receive; }                   // output over pullup
```

"Udir" stellt das Richtungsregister dar, "Uoport" das Ausgaberegister. "Upin" ist die Nummer des Portpins. Wir sehen, daß drei Zustände codiert wurden, der Zustand "sendH" ist dem Zustand "receive" identisch. Die Definition des Zustands "sendH" ist eigentlich überflüssig, er verbessert nur die Lesbarkeit der aufrufenden Routine *fputc()*. Die kann zum Beispiel so aussehen:

```
// Soft_UART_Tx für ATtiny85
void fputc(uint8_t ch) {                // ein UART Zeichen senden
        // non-inverting-UART           // soft-send one character bit by bit
        uint8_t i=0;
        // send startbit low:
        starteBittakt(OneBitDelay);     // start timer with bit clock
        sendL;                          // startbit low
        waitabit();                     // wait for next clock
        // send 8 bits:
        for (i=0; i<8; i++)             // send all bits of "ch"
        {                               // sende jeweils das LSB (rechtes)
                if ((ch & 0x01))        // if right most bit is one
                        { sendH; }      // send one
                else                    // if right most bit is zero
                        { sendL; }      // send zero
                ch = ch >> 1;           // bit shift right (send LSB)
                waitabit();             // wait for next clock
        }
        // sende 2 Stopbits (lieber bissel zuviel, erhöht Fehlertoleranz):
        sendH;                          // send stopbit = high
        waitabit();                     // erstes Stopbit
        waitabit();                     // zweites Stopbit -> interner RC-Osz.
        beendeBittakt();
        // receive;                     // Empfang mit pullup == sendH
}
```

Realisierung mit "harten" UARTs

Größere Prozessoren, man denke an ATmega, aber auch an FT232RL, bieten nun eine UART-Baugruppe in Hardware. Will man leicht erhöhte low-Pegel, wie in der Prinziplösung dargestellt, aus Gründen der Störsicherheit reduzieren, können wir die Schottkydioden durch nMOS Transistoren (n-channel enhancement) ersetzen. Die gibt es preiswert im SOT23-Gehäuse (BSS123, BSS138, BSN20...), siehe Abb.5.

Dabei ergibt sich allerdings ein Problem. Der nMOS-Transistor negiert den TX-Signalpegel. Zur Kompensation haben wir das Signal vorab zu negieren. Eine vorab-Negation des auszugebenden Bytes per Software würde nicht genügen, da durch den Transistor trotzdem Start- und Stopbit negiert werden würden.

Plant man, mehrere Module mit einer SWART auszurüsten, so kann man vorteilhaft auf jeder Platine einen Pullup-Widerstand anordnen, in Abb.5 erscheint er deshalb in zwei Hälften geteilt.

Plant man eine busarige Vernetzung mehrerer SWART-Units, so ist der halbe Pullup je nur an zwei Endpunkten zu bestücken. Oder man bestückt nur einen Pullup irgendwo am Bus. Oder man nutzt die soft-steuerbaren, internen Pullups der Controller.

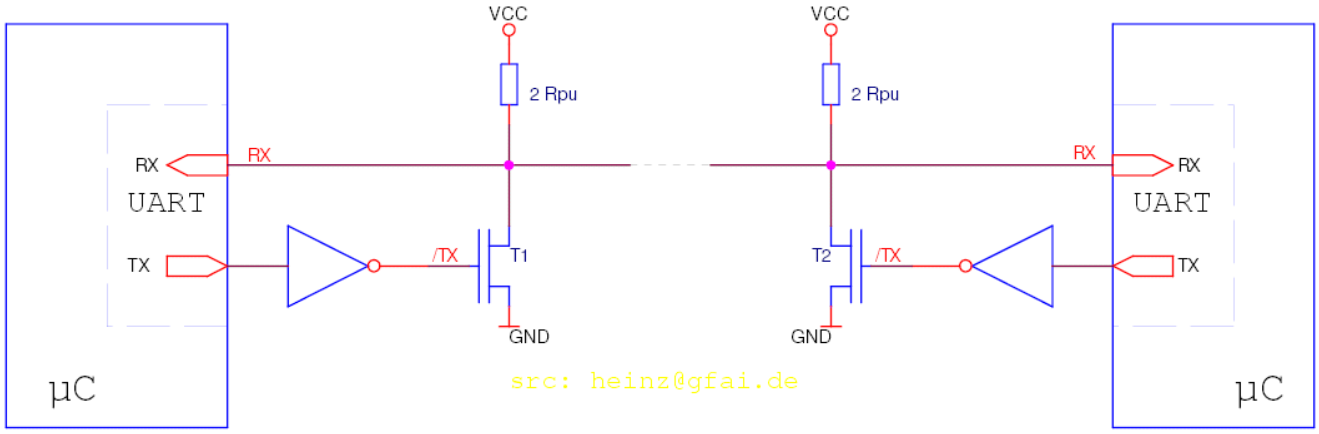


Abb.5: Werden Mikrocontroller mit UARTs in Hardware benutzt, empfehlen sich nMOS-Transistoren zur Kopplung. Vorab ist das Signal zu negieren.

Gegenüber der eleganten, Software-basierten SWART, wie auch gegenüber der Lösung mit Schottkydioden ist diese Lösung recht aufwendig. Um aber einen Zugang zu einem SWART-Modulsystem zu bekommen, wird sie sich am Busmaster (z.B. FTDI, USB-VCP) nicht vermeiden lassen. Bei mir genügt die Lösung mit einer Schottkydiode für Zwecke des Debugging. Man kann hoffen, daß Prozessorhersteller aufspringen werden und ihre UARTs um einen Inverter, ein Registerbit namens "SWART" und einen nMOS-Transistor erweitern werden. Freie, reservierte Register-Bits sind bei Atmel-Prozessoren meist noch vorhanden.

SWART Beispiel

Zur optimalen Leiterkartenentflechtung wie zur Kurzschlußvermeidung bei Verpolung hat sich eine dreipolige Anordnung in der bei Modellbau-Servos üblichen Form im 100 mil Raster (2,54 mm) bewährt:

Anschlüsse SWART (simplex)
Pin3: RTX weiss
Pin2: VDD rot
Pin1: GND schwarz

Abb.6: Für Experimente können Servostecker als SWART-Stecker genutzt werden (hier Bauart Futaba). vlnr: Stecker, Kupplung (Buchse), Layout (Sicht auf Buchse)

Erweitert man diesen Stecker auf die Möglichkeit, auch duplex arbeiten zu können, ergibt sich für den Bus-Master vorteilhaft eine kompatible Anordnung des TX-Sendepins als Pin-Nummer 4.

Anschlüsse Master UART (duplex) incl. SWART

Pin4: TX grün
Pin3: **RX weiss**
Pin2: **VDD rot**
Pin1: **GND schwarz**

Bei Verwendung von TX steckt der Teufel wieder im Detail. Beim zugehörigen Slave UART wären die RX- und TX-Pins zu vertauschen!

Der Anfang mit SWART ist schnell gemacht. Abb.7 zeigt einen Combi-Adapter für SWART oder UART auf Basis eines USB-VCP Wandlers von FTDI. Wird die Schottkydiode D4 bestückt und ist BU2 3-polig, arbeitet die Schaltung als SWART. Wird BU2 vierpolig bestückt und wird D4 weggelassen, arbeitet sie als gewöhnliche UART. TSS42U ist eine Schottkydiode im 0603 Gehäuse mit geringer Kapazität und geringer Durchlaßspannung bei kleinen Strömen.

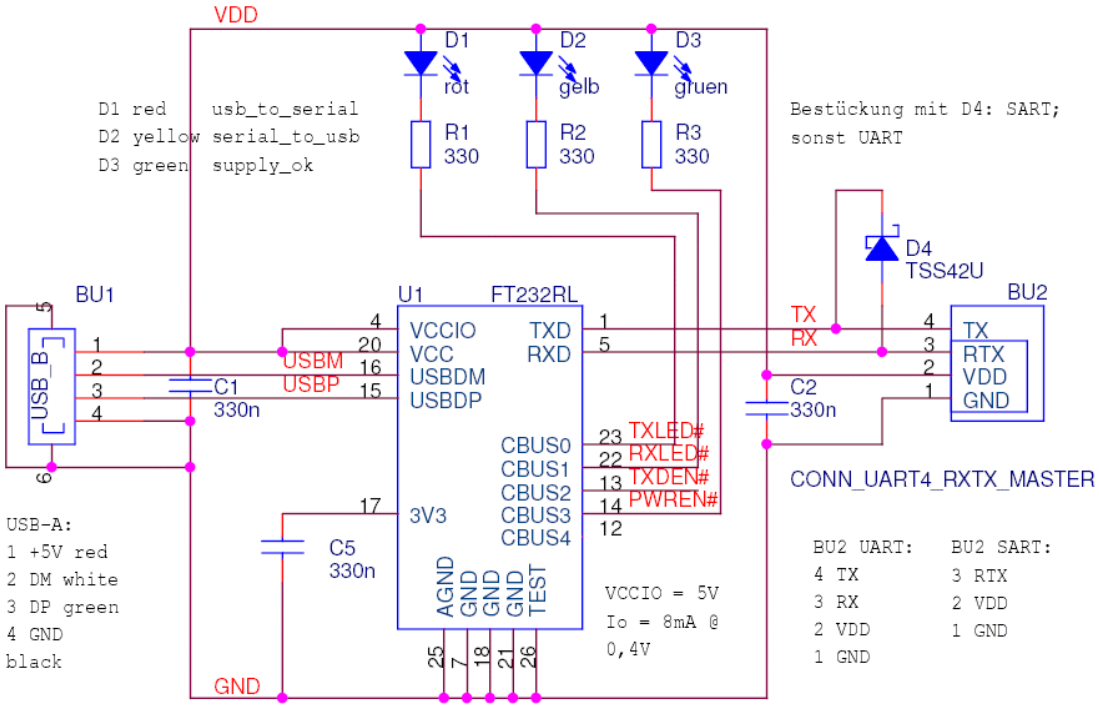


Abb 7: Schaltung eines USB-Combi-Adapters für UART oder SWART.

SWART-Multiplex mit der Betriebsspannung

Eine Lösung, die den iButtons oder dem One-Wire Verfahren ebenbürtig werden kann, entsteht durch Multiplexing mit der Betriebsspannung. Vorausgesetzt, unser Sensor hat einen minimalen Strombedarf, können wir im Protokoll Pausen definieren, in denen niemand zu senden hat. Die Pausen dienen der Hochladung der RTX-Leitung zur Übertragung der Betriebsspannung.

Ein einfachstes Verfahren wäre die Sendung einiger Bytes 0xff nach jedem Nutzbyte. Damit kann bereits eine hinreichende Menge an Energie transferiert werden, die für verschiedene Sensoren mit einer Stromaufnahme im Mikroamperbereich genügt.

Ist mehr Strom erforderlich, ist eine (unsymmetrische) Master-Slave Struktur mit einem pMOS-Transistor (Abb.8) sinnvoll. Dann müssen Pausen definiert werden, um Senden und Gegensenden während des Hochladens zu vermeiden. TX muß per Software blockiert werden, wenn T3 eingeschaltet ist. Der Master unterscheidet sich nun gravierend vom Slave.

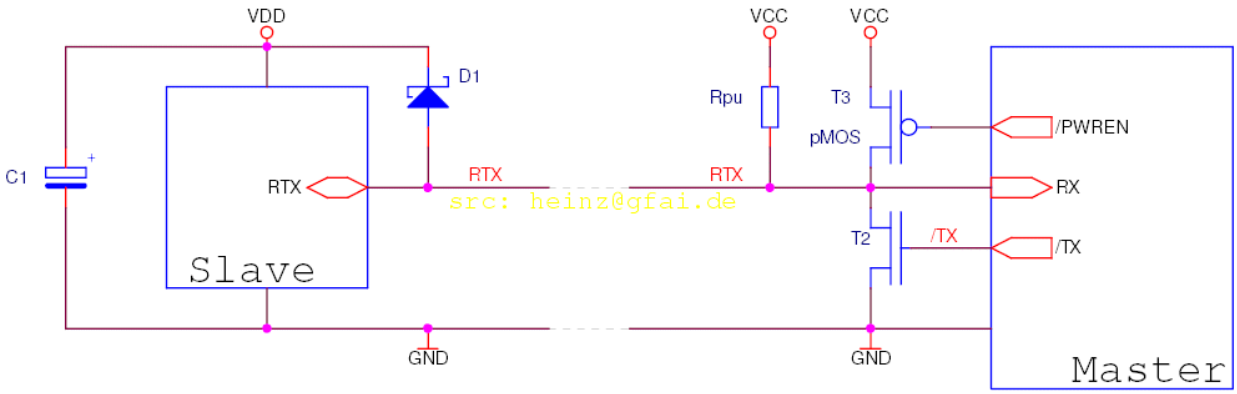


Abb.8: Stromsparende Sensoren lassen sich über Betriebsspannungs-Multiplex mit SWART verbinden. Ein pMOS-Pullup-Transistor T3 kann zusätzlich das Hochladen verstärken. Diode D1 überträgt die Energie zur Betriebsspannung VDD des Slave

Schutzschaltungen an den Pins unserer Controller sind fast generell mit Schutzdioden nach VDD und GND versehen. Die VDD-Schutzdiode könnte in einigen Fällen bereits genügen, eine externe Schottky-Diode D1 entfällt dann.

SWART-Bus

Wie schon bemerkt, können mehrere SWART-Units an eine RTX-Leitung angeschlossen werden. Es entsteht ein relativ langsamer Bus, der mit verfügbaren Terminal-Tools zu debuggen wäre, der aus nur einer Leitung besteht und der für intramaschinelle Kommunikation, z.B. auch in Robotern oder PKW, wunderbar geeignet wäre. Bliebe nur die Protokolldefinition und die Organisation von Postfächern per Software zu tun.

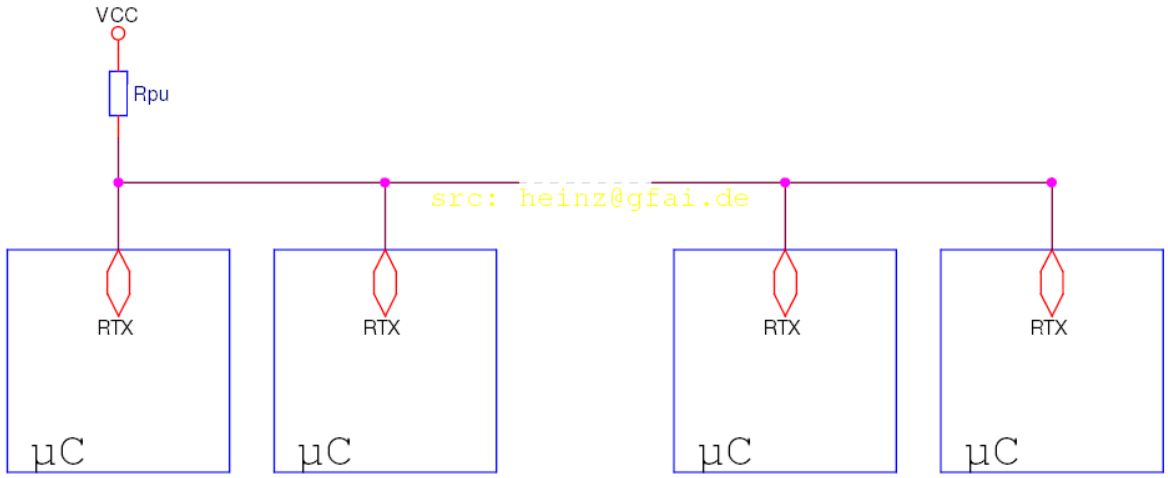


Abb.9: SWART-Bus. Mehrere SWART-Transceiver sind gleichberechtigt auf einer Leitung verbunden.

Als Hilfestellung für das Protokoll: Es sollte und könnte einfach sein. Bleiben wir beim ASCII, können wir ohne Spezialterminal mitlesen. Das Protokoll könnten wir "ASCII-Bus" nennen:

Der Adressraum muß 52 Devices nicht übersteigen (ASCII: a...z und A...Z). Die Adresse besteht aus drei Byte, im ersten steht der Sender, im zweiten der Empfänger, im dritten die Größe des Datenpakets als Zweierpotenz in ASCII "0 ... 9" (2^0 ... 2^9 = 1, 2, 4, 8, 16, 32, 64, 128, 256, 512). So hieße "ky4": Knoten "k" sendet gleich an Knoten "y" genau 2^4 = 16 Byte Daten. Ob die Daten nur ASCII oder auch hex sein dürfen, sei dahingestellt. Hintenan könnte man noch in zwei Byte (entsprechend zwei Nibble) eine ASCII-Checksumme senden in der Form 0...9, a...f. Die ASCII-Steuerzeichen könnten ihrem Namen Ehre machen. RS (request to send (0x1e) könnte vorab auf den Bus gelegt werden und fordert alle anderen auf, zu schweigen. Das Ende jeder Nachricht wird mit EOT (0x04) markiert. Escape (0x1b) wirkt als Bus-Reset, falls mehrere gleichzeitig ein RS sendeten.

Was mich persönlich an einem solchen Protokoll reizt: Es wäre maximal unkompliziert. Man kann sich in Ruhe der eigentlichen Entwicklungsaufgabe widmen und vergeudet nicht Wochen, um den Bus zu verstehen und Bus- Probleme zu fixen.

Zusammenfassung

UART sind weltweit verbreitet. Zur Überwindung von Polaritätsproblemen wird eine simplex Single-Wire UART (SWART) Verbindung vorgestellt, die auf UART aufbaut. Diese ist im Bereich intelligenter Sensorik und Aktorik bei niedrigen

Datenraten und kurzen Entfernungen vorteilhaft anwendbar.
Die SWART-Schnittstelle eignet sich besonders für preissensible, digitale Sensoren und Aktoren mit kleinsten Mikrocontrollern, deren UARTs in Software realisiert sind. Das Sende-Empfangs-Portpin (RTX) kann hierbei mit nur zwei Zuständen programmiert werden (sendLow und receive).
Wird eine SWART auf Mikrocontrollern mit Hardware-UART realisiert, ist entweder eine Schottky-Diode oder ein Inverter und ein nMOS-Transistor erforderlich.
SWART-Multiplex mit der Betriebsspannung ist möglich, erfordert u.U. aber den Einbau zusätzlicher Hochlade-Pausen ins Protokoll.
Werden mehrere SWART-Units an eine RTX-Leitung angeschlossen, entsteht ein SWART-Bus.

Standort des Aufsatzes

Heinz, G.: Single-Wire UART (SWART) - Schaltungstechnik, Prinzip und Ausführungen. Web-Dokument vom 14.12.2014
www.gfai.de/~heinz/publications/papers/2014_swart.pdf

Copyrights

www.gfai.de/~heinz
Mails to heinz@gfai.de

Berlin, den 14.12.2014
editiert: 27.2.2015, 9.4.2015