Fachhochschule Münster University of Applied Sciences

Labor für Halbleiterbaulelemente und Bussysteme

Übung Bussysteme SS2015
Tutor: Diplom Informatiker Gregor Rebel

Versuch 2

Serielle Schnittstelle USART

In diesem Praktikum wird die Verwendung der USART Schnittstelle behandelt (Universal Synchronous Asynchronous Receiver/Transceiver). Eine vereinfachte Version dieser Schnittstelle ohne synchrone Datenübertragung wird mit UART abgekürzt. Während dieses Praktikums werden Sie lernen, wie man die Schnittstelle verbindet und programmiert.

Die USART Schnittstelle ist ein Stück Hardware, welches parallele Daten in serielle wandelt und wieder zurück. USARTs werden meist in Verbindung mit einem Kommunikationsstandard wie EIA, RS-232, RS-422 oder RS-485 verwendet. Die vielen unterschiedlichen Bezeichnungen zeigen, dass das Datenformat sowie die Übertragungsgeschwindigkeit konfigurierbar sind. Die für die Übertragung notwendigen elektrischen Spannungspegel werden von einem Treiberschaltkreis außerhalb des USART bereitgestellt.

Asynchrone Serielle Datenübertragung

Ein UART überträgt Datenbytes als sequentielle Abfolge von einzelnen Bits. Beim Empfänger setzt ein zweiter UART die einzelnen Bits wieder zu vollständigen Bytes zusammen. Jeder UART besitzt ein Schieberegister, welches die grundlegende Methode zur Wandlung zwischen seriellen und parallelen Daten darstellt. Die serielle Übertragung von digitalen Informationen über eine einzelne Leitung ist kostengünstiger als die parallele Übertragung über mehrere Leiter. Für gewöhnlich generiert oder empfängt ein UART die externen Signale, welche Informationen zwischen Komponenten übertragen, nicht direkt. Separate Schnittstellen werden genutzt um zwischen den logischen Pegeln des UART und den externen Signalen zu konvertieren. Diese Signale können viele Formen haben. Beispiele für Schnittstellen für spannungsbasierte Signalübertragung sind RS-232, RS-422 und RS-485.

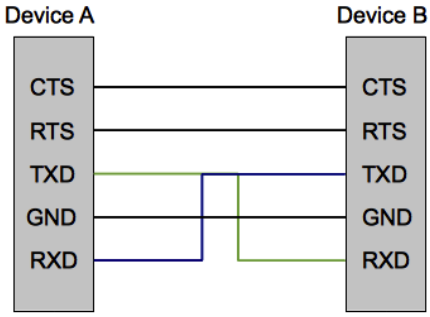


Bild 1: UART Signalschema

Ein UART hat typischerweise folgende Komponenten:

- Einen Taktgenerator
- Ein- und Ausgangsschieberegister
- Sende/Empfangskontrolle
- Lese/Schreibe Kontroll-Logik
- Sende/Empfangs-Buffer (optional)
- Paralleler Daten-Buffer (optional)
- FIFO Buffer-Speicher (optional)

Übertragung von Zeichen:

Start	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7	Parity	Stop

Tabelle 1: Typische Zeichenübertragung

Der Ruhezustand, wenn keine Daten übertragen werden, ist durch eine hohe Spannung gekennzeichnet (Open drain mode). Jedes Zeichen wird durch ein logisch niedriges Start-Bit, eine konfigurierbare Anzahl an Daten-Bits (typischerweise 8), ein optionales Paritäts-Bit sowie ein oder mehrere logisch hohe Stopp-Bits übertragen.

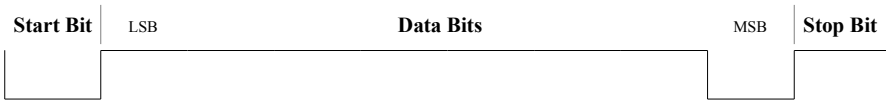


Tabelle 2: Darstellung von Signalen auf dem Oszilloskop.

Das Start-Bit signalisiert dem Empfänger, dass ein neues Zeichen gesendet werden soll. Die nächsten fünf bis acht Bits repräsentieren, je nach gewählter Einstellung, das zu sendende Zeichen. Auf die Daten-Bits kann dann ein Paritäts-Bit folgen. Die folgenden ein bis zwei Bits sind immer logisch hoch (Mark condition) und werden Stopp-Bits genannt. Diese signalisieren dem Empfänger, dass das Zeichen vollständig ist. Da das Start-Bit logisch Null und das Stopp-Bit logisch Eins ist, werden zwischen den übertragenen Zeichen immer mindestens zwei Änderungen des Signals garantiert. Falls die Leitung für länger als eine Übertragungszeit auf logisch Null gehalten wird, so ist dies das Zeichen für eine Pause (break condition), welche vom Empfänger erkannt werden kann.

Aufgabe 1

Eine einfache serielle Ausgabe benötigt lediglich eine TxD-Leitung und kann auch per Software über einen GPIO-Pin realisiert werden. Die Wandlung von parallel zu seriell geschieht dann per Software. Auch das Timing muss von der Software erzeugt werden. Ziel dieser Aufgabe ist die Implementierung einer seriellen 1-Pin Ausgabe in Software.
Nutzen Sie dazu Ihre Kenntnisse über GPIO-Pins aus dem vorigen Versuch.

Parallel Seriell Wandlung in Software

Eine Wandlung von parallel nach seriell kann per Software mit Hilfe einer Bitmaske innerhalb einer for-Schleife durchgeführt werden:

```
void printSerial(ttc_gpio_pin_e PinTxD, unsigned char Byte) {
    unsigned char Bit;
    for (Bit = 0; Bit <= 7; Bit++) {
        unsigned char Mask = 1 << Bit; // Bitmaske erstellen
        if (Byte & Mask) { // bit #Bit is 1
            // Bit 1 ausgeben
        }
        else {
            // Bit 0 ausgeben
        }
       // warten entsprechend Bitrate
    }
}
```

Quelltexte zum Projekt hinzufügen

Die Datei main.c ist schlecht geeignet um Programmcode aufzunehmen. Die Datei wird schnell unübersichtlich. Weil main.c die main() Funktion enthält, kann sie nicht von anderen Quelldateien eingebunden werden. Um C-Programme übersichtlich zu gestalten und die Wiederverwendung von Funktionen zu ermöglichen, können zusätzliche Quelltexte zum Programm hinzugefügt werden. The ToolChain bietet dafür das Skript _/source.pl an. Dieses erlaubt es ein Paar aus Quelltext- und Headerdatei automatisch zu erstellen und zum aktuellen Projekt hinzuzufügen. Das Skript wird dazu in der Shell im Projektordner aufgerufen:

```
_/source.pl
```

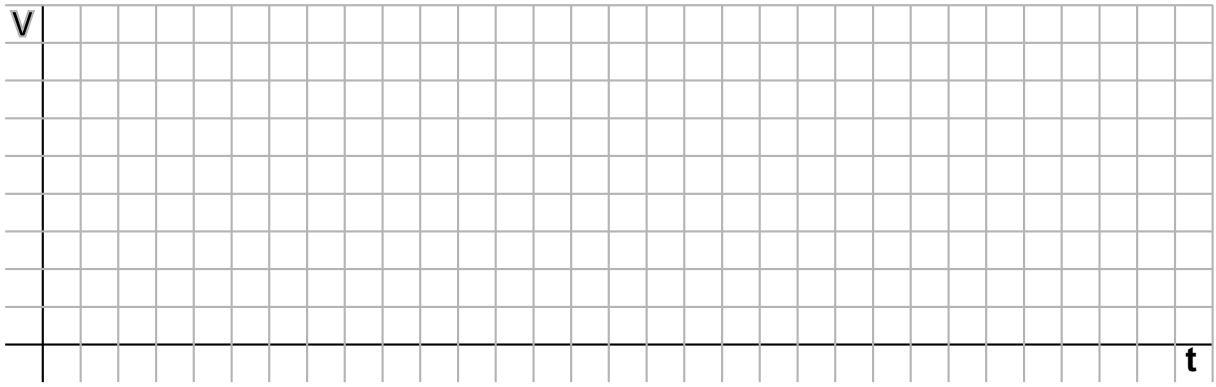
Ohne Argumente gibt das Skript eine Hilfeseite aus. Ein Paar foo.c und foo.h kann z.B. so erzeugt werden:

```
_/source.pl add foo
```

Nach dem Erzeugen wird einfach das Projekt im QtCreator neu übersetzt. Die neuen Dateien tauchen danach im QtCreator auf und können sofort benutzt werden. Die entsprechenden Includes und Objectfiles wurden bereits in main.c und extensions.local/makefile.700_extra_settings eingetragen.

Aufgaben

1. Erstellen Sie ein neues TTC-Projekt namens BS_Gruppe1_Versuch2 bzw. BS_Gruppe2_Versuch2.
2. Ändern Sie die Konfiguration in activate_project.sh
   1. Aktivieren Sie Ihr Prototypboard im Rank 100
   2. Aktivieren Sie Ihren Programmer im Rank 350
   3. Aktivieren Sie 500_ttc_gpio
   4. Deaktivieren Sie 600_example_leds
3. Fügen Sie eine neue Quelltextdatei namens softserial mittels _/source.pl zum Projekt hinzufügen
4. Implementieren Sie printSerial in softserial.c und softserial.h mit einer Datenrate von 2400 Bit/s
   1. Um ttc_gpio in softserial benutzen zu können, müssen Sie ttc-lib/ttc_gpio.h einbinden:
      `#include "ttc-lib/ttc_gpio.h"`
   2. Implementieren Sie auch Start-, Parity- und Stop-Bit
5. Wählen Sie einen GPIO-Pin für TxD und initialisieren Sie den Pin in taskMain()
6. Rufen Sie von taskMain() aus printSerial() auf um endlos die Zeichenfolge 'S' 'O' 'S' gefolgt von 100 ms Pause auszugeben
7. Betrachten Sie die serielle Ausgabe im Oszilloskop
8. Zeichnen Sie die Signalfolge auf



Die UART Schnittstelle

Das Senden und vor allem Empfangen von seriellen Daten per Software benötigt viel Rechenzeit. Per Software ist es kaum möglich mehrere Schnittstellen in hoher Geschwindigkeit zu bedienen. Dafür enthalten Mikrocontroller spezielle Schnittstellenbausteine namens UART bzw. USART. Diese nehmen der CPU einen Großteil der Bitfummelei ab. Die Software braucht dann nur noch komplette Bytes in Ausgaberegister zu schreiben. Während die einzelnen Bits über die Leitung wandern kann die CPU sich um andere Dinge kümmern. Die Software braucht auch nicht ständig auf eingehende Daten zu lauschen. Der Empfangsteil kann automatisch einen Interrupt auslösen und eine vorher bezeichnete Funktion (die sogenannte Interrupt Service Routine) aufrufen.

UART Empfangsteil

Alle Aktivitäten der UART Hardware werden durch ein internes Taktsignal kontrolliert, welches mit einem Vielfachen der Datenrate taktet. Der Empfänger prüft den Zustand des eingehenden Signals mit jedem Takt und wartet auf den Beginn eines Start-Bits. Wenn ein scheinbares Start-Bit mindestens eine halbe Bitperiode lang stabil bleibt, so ist es gültig und signalisiert den Beginn eines neuen Zeichens. Falls es nicht stabil bleibt wird es als falscher Impuls ignoriert. Nach einer weiteren Bitperiode wird der Zustand der Leitung erneut ausgewertet und das Ergebnis in das Schieberegister geschoben. Nachdem die erforderliche Anzahl an Bitperioden für das Zeichen durchlaufen wurde, wird der Inhalt des Schieberegisters dem empfangenden System verfügbar gemacht (In paralleler Weise). Der UART setzt ein Flag, mit welchem er signalisiert, dass neue Daten verfügbar sind. Es ist auch möglich, dass er einen Interrupt generiert und so den Prozessor auffordert die empfangenen Daten abzurufen.

UART Sendeteil

Das Senden von Daten ist einfacher, da das sendende System die Kontrolle über den Prozess hat. Sobald Daten in das Schieberegister abgelegt wurden, generiert der UART ein Start-Bit, schiebt die erforderlichen Bits aus dem Register auf die Leitung und hängt das Paritäts-Bit sowie die Stopp-Bits an. Da das Übertragen eines einzelnen Zeichens, verglichen mit der CPU-Geschwindigkeit, sehr langsam ist, setzt der UART ein Flag, welches anzeigt, dass er beschäftigt ist. Auf diese weise unternimmt das sendende System nicht den Versuch neue Daten in das Schieberegister zu schreiben, bevor das Zeichen komplett übertragen wurde. Dies kann auch per Interrupt geschehen. Im Full-Duplex-Modus müssen Daten gleichzeitig geschrieben und empfangen werden. Daher verfügt ein UART über je ein Schieberegister für das Senden und Empfangen von Zeichen.

High-Level Treiber ttc_usart

Im Mikrocontroller werden die USART-Schnittstellen per memory-mapped IO konfiguriert und betrieben. Die Registerstruktur ist bei jedem Mikrocontrollertyp unterschiedlich. Ein STM32F1xx hat andere USART-Register als ein STM32L1xx. Für die Einstellung der richtigen Bitrate muss der Wert des externen Oszillatorquarzes bekannt und der Clocktree des aktuellen Mikrocontrollers entsprechend konfiguriert werden.
The ToolChain bietet einen High-Level Treiber namens ttc_usart welcher den Anwender alle Einstellung einheitlich und architekturunabhängig zur Verfügung stellt. Die gewünschten Einstellungen werden in einer Struktur vorgenommen. Ein Aufruf von ttc_usart_init() ruft dann den low-level Treiber für das aktivierte Prototypboard auf und konfiguriert die Hardware entsprechend. Auch können komplette Zeichenketten anstatt nur einzelner Bytes gesendet und empfangen werden.

Um ttc_usart zu verwenden muss diese als Erweiterung 500_ttc_usart zunächst aktiviert werden. Nach einer Neuübersetzung des Programmes kann ttc-lib/ttc_usart.h eingebunden werden.

Generelle Verwendung von ttc_-Treibern

Die Verwendung von High-Level Treibern in The ToolChain folgt einem praktisch immer gleichen Schema:

1. **ttc_XXX_config_t* Config = ttc_XXX_get_configuration(LOGICAL_INDEX);**
   LOGICAL_INDEX bezeichnet dabei die logische Nummer der zu verwendenden USART-Schnittstelle dar.
   1 = TTC_USART1, 2 = TTC_USART2, …
   Welcher USART konkret verwendet wird, kommt darauf an, wie TTC_USART1, … aktuell definiert ist. Den aktuellen Wert finden Sie einfach per QtCreator.
2. **Felder in Config anpassen**
   Die wesentlichen Felder konfigurieren BaudRate, (halbe) StopBits, Wortlänge, Paritätsnutzung und Paritätsart.
   Einige dieser Einstellungen sind einzelne Bits im Bitfeld Flags.
   Eine genaue Auflistung aller Konfigurationsfelder finden Sie in der Struktur ttc_usart_config_t.
3. **ttc_XXX_init(LOGICAL_INDEX);**
   Die Schnittstelle wird mit der vorher bestimmten Konfiguration initialisiert. Die Konfiguration darf danach nicht mehr geändert werden. Details hierzu stehen wiederum in der Struktur ttc_usart_config_t.
4. **Andere Funktionen ttc_XXX_*() aufrufen**
   Nach der Initialisierung dürfen alle anderen Funktionen des High-Level Treiber wie z.B. Sendefunktionen aufgerufen werden.
5. **ttc_XXX_deinit(LOGICAL_INDEX);**
   Nach Verwendung kann eine Schnittstelle wieder geschlossen werden. Diese Funktion ist jedoch oft nicht implementiert.

Beispiel ttc_usart Verwendung

```
#include "ttc-lib/ttc_usart.h"

#define USART_INDEX 1 // logical index of usart device to use (1 = TTC_USART1, ...)

ttc_usart_config_t* Config = ttc_usart_get_configuration(USART_INDEX);
Assert(Config, ec_UNKNOWN); // will block if no usart available


Config->BaudRate = 115200;
Config->HalfStopBits = 2;


/* Configure Parity */
Config->WordLength = 9;               // required to use Parity
Config->Flags.Bits.ParityEven     = 0;
Config->Flags.Bits.ControlParity  = 1;
Config->Flags.Bits.ParityOdd      = 0;
Config->Flags.Bits.TransmitParity = 1;


/* Now initialize device */
Error = ttc_usart_init(USART_INDEX);


/* send some constant string */
ttc_usart_send_string_const(USART_INDEX, TextHello, -1);
```

Um Daten vom Mikrocontroller an einen Desktop Rechner zu senden wird oft die RS232-Schnittstelle mit 9-poligen Sub-D Steckern verwendet. Dazu wird entweder eine serielle Schnittstelle am Rechner oder ein USB-RS232 Adapter benötigt. Die ARM-USB-OCD-H und einige andere Programmieradapter haben solch einen Adapter eingebaut.

grTerminal ein serielles Terminalprogramm

Am Desktop Rechner wird noch ein serielles Terminalprogramm benötigt. Eine Möglichkeit ist das quelloffene grTerminal. Dabei handelt es sich um ein mit dem Qt SDK entwickeltes grafisches Terminalprogramm welches neben der seriellen auch die Kommunikation über Ethernet per TCP und UDP sowie Named Pipes anbietet. grTerminal_Example ist vor allem als Startpunkt zur Entwicklung von grafischen Anwendungen zur Steuerung von Mikrocontrollern gedacht.

grTerminal sollte auf den Laborrechnern vorinstalliert sein. Eine Installationsanleitung und genauere Beschreibung findet sich hier: http://hlb-labor.de/projekte/grterminal/

Aufgabe 2

1. Erweitern Sie Ihr Programm um die Initialisierung der ersten USART-Schnittstelle mit 2400 Bit/s, 8 Datenbits, gerader Parität und einem Stoppbit (8e1).
2. Senden Sie abwechselnd SOS über den GPIO-Pin und per ttc_usart_send_string_const()
3. Vergleichen Sie die beiden Signale am Oszilloskop. Was fällt Ihnen auf?
4. Besorgen Sie sich ein Protoboard mit 9-poliger RS232 Buchse und verbinden Sie diese mit dem passenden Stecker an einem ARM-USB-OCD-H Programmer.
5. Starten Sie das Programm grTerminal aus der Shell
    1. Öffnen Sie ein neues Terminal (Strg+n)
    2. Konfigurieren Sie die serielle Verbindung aus 2400 Bit/s, 8e1
    3. Öffnen Sie die Schnittstelle ttyUSB0
    4. Jetzt müßten Sie den SOS Text im Fenster lesen

Aufgabe 3

Die Ausgabe von konstanten Texten ist recht langweilig. Für gewöhnlich geben Mikrocontroller Messwerte als Mischung von konstantem Text und formatierten Variableninhalten aus. Dafür wird ein Stringbuffer zur Vorbereitung der Ausgabe benötigt. Ein Stringbuffer ist einfach ein char-Array welches groß genug ist um auch die längste Fassung einer Textausgabe aufzunehmen. Ähnlich der printf() Funktion kann mit einer entsprechenden snprintf() Funktion ein Formatstring und zugehörige Variablen in einen Stringbuffer geschrieben werden. ttc_usart bietet entsprechend eine spezielle Funktion zur Versendung von nicht-konstanten Strings.

1. Legen Sie eine globale Variable Temp als Array von char der Größe 50 an
2. Schreiben Sie eine for-Schleife welche von 1 bis 100 zählt
    1. Schreiben mit Sie den Schleifenzähler mit folgendem Formatstring in Temp: „Iteration %d\n“
    2. Nutzen Sie dazu ttc_string_snprintf()
    3. Aktivieren Sie ttc_string und binden Sie ../ttc-lib/ttc_string.h ein
3. Senden Sie Temp anstelle von SOS mithilfe einer passenden Funktion aus ttc_usart.