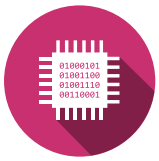


Serieller Empfänger

Labor Digitales Design

Inhalt

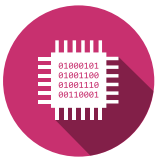
1 Ziele

Dieses Labor zielt darauf ab, die Konstruktion synchroner digitaler Schaltungen unter Verwendung einer Finite State Machine (FSM) zu üben. Es konzentriert sich auf die Implementierung eines seriellen RS232-Empfängers, einschliesslich:

- Verständnis der Prinzipien der asynchronen seriellen Kommunikation.
- Entwurf und Implementierung eines Schieberegisters und eines Zählers.
- Erstellung einer FSM im HDL Designer zur Steuerung des Empfangsprozesses.

Durch dieses Labor werden die Studierenden praktische Erfahrungen in der Kombination von sequentieller Logik (FSM, Zähler) mit der Datenverarbeitung (Schieberegister) sammeln, um ein komplettes Kommunikationsmodul zu realisieren.

Die Funktionalität des Empfängers wird durch das Dekodieren einer Nachricht überprüft, die vom Xilinx PicoBlaze μProzessor gesendet wurde, der in vorherigen Laboren entwickelt wurde.



Das vorherige Labor **`$LABO_DIR/CNT`** ist eine Voraussetzung. Stellen Sie sicher, dass Sie es abgeschlossen, vollständig getestet und in Ihr Projekt importiert haben.

2 Serieller Empfänger

2.1 Serielle Übermittlung

Ein serieller Empfänger ist eine digitale Schaltung, die einen eingehenden asynchronen seriellen Datenstrom in parallele Daten umwandelt, die für eine weitere Verarbeitung geeignet sind. Die Abbildung 1 zeigt das Timing der seriellen Übertragung eines Datenworts, bei der die Daten bitweise übertragen werden, beginnend mit einem Startbit, gefolgt von Datenbits (Least Significant Bit (LSB) niederwertiges Bit zuerst) und endend mit einem Stoppbit.

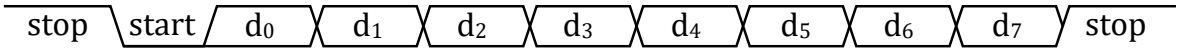


Abbildung 1 - Serielle Übermittlung

2.2 Schaltung

Der Kern des Designs in Abbildung 2 besteht aus:

- Einem Schieberegister, um die eingehenden Bits zu sammeln und in ein paralleles Format umzuwandeln.
- Einem Zähler, um die Systemuhr zu teilen und sich mit der Baudrate der eingehenden Daten zu synchronisieren.
- Einer Zustandsmaschine (FSM), um den Empfangsprozess zu steuern, das Startbit zu erkennen, jedes Datenbit zur richtigen Zeit zu speichern und das Stoppbit zu validieren.

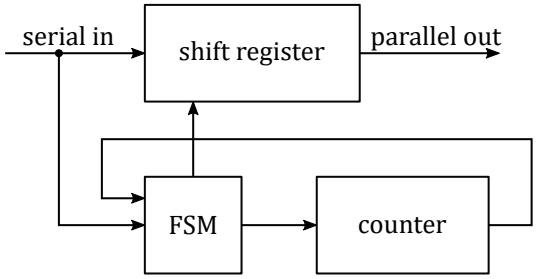
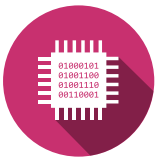


Abbildung 2 - Blockschaltung des seriellen Empfängers

Die FSM überwacht die serielle Eingangsleitung, erkennt den Übergang, der den Beginn eines Frames anzeigt (start bit), und löst regelmässige Abtastimpulse mithilfe der Zählers aus. Jedes abgetastete Bit wird in das Register verschoben, bis das vollständige Byte empfangen wurde. Die Ausgabe wird dann in paralleler Form für nachfolgende Stufen des Systems bereitgestellt.

Untersuchen Sie die vorhandenen Elemente und Signale im Block **`COM/serialPortReceiver`**.



3 FSM

Im Block **`COM/receiverController`** implementiert eine FSM den Empfang serieller Daten. Die FSM verwaltet das Timing der Abtastung der eingehenden Bits und stellt sicher, dass jedes Bit zur richtigen Zeit in Bezug auf die Baudrate des eingehenden Datenstroms gelesen wird.

3.1 Elemente der gegebenen FSM

Mehrere Elemente sind bereits verfügbar, sie sind farblich codiert in Abbildung 3:

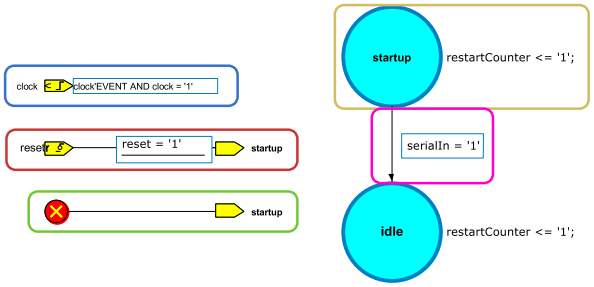


Abbildung 3 - Elemente der gegebenen FSM

1. Definition des Taktsignals und seiner Auslösebedingung. In diesem Fall tritt das Taktereignis am steigenden Rand des Signals **`clock`** auf.
2. Definition des Resets und wie die Flip-Flops zurückgesetzt werden und der Anfangszustand nach einem Reset. In diesem Fall das Signal **`reset`** aktiv hoch, mit dem Anfangszustand **`startup`**
3. Der Wiederherstellungszustand, der verwendet wird, wenn keine andere gültige Zustandszuweisung vorhanden ist.
4. Zustände der FSM. Jeder Zustand benötigt einen eindeutigen Namen. Der Code auf der rechten Seite wird ausgeführt, während Sie sich in diesem Zustand befinden (*Code ist optional*). In diesem Fall:

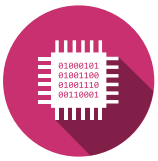
```
restartCounter <= '1';
--   ^          ^   ^ ^
--   |          |   | +-- Erforderliches Endsymbol einer Anweisung
--   |          |   +---- Zuweisungswert für das Signal
--   |          +-------- Zuweisungsoperator (Achtung <= nicht =)
--   +------------------- Signal, dem ein Wert zugewiesen werden soll
```

Listing 1 - Code, der im Zustand **`startup`** ausgeführt wird

5. Übergangsbedingungen, um Zustände miteinander zu verbinden (*Bedingungen sind optional*). In diesem Fall:

```
serialIn = '1'
--  ^    ^  ^  ^
--  |    |  |  +-- Kein Endsymbol erforderlich, da dies eine Bedingung ist
--  |    |  +----- Bedingung für den Übergang
--  |    +-------- Vergleichsoperator (Achtung = nicht ==)
--  +------------- Linke Seite der Bedingung ist in diesem Fall das Signal `serialIn`
```

Listing 2 - Übergangsbedingung zwischen den Zuständen **`startup`** und **`idle`**



Es kann übersetzt werden mit „wenn das Signal **`serialIn`** gleich **`'1'`** ist, dann wechselt der Zustand“.

3.2 Zustände und Übergänge

Die FSM hat mehrere Zustände, die jeweils eine bestimmte Phase des Empfangsprozesses darstellen. Die Übergänge zwischen diesen Zuständen werden durch Ereignisse ausgelöst, die in den Pfeilen definiert sind. Wenn der Pfeil keinen Auslöser hat, bedeutet dies, dass der Übergang immer aktiv ist und im nächsten Taktzyklus immer in den nächsten Zustand wechselt.

Oben im Editor befinden sich mehrere Symbolleisten mit allen erforderlichen Funktionen zum Bearbeiten der FSM. Die erste Symbolleiste Abbildung 4 ermöglicht das Erstellen neuer Zustände (blauer Kreis) und Übergänge (schwarzer Pfeil).

Abbildung 4 - Toolbar zum Erstellen neuer Zustände oder Übergänge in einer FSM

3.3 Aktionscode, Übergangsbedingungen und Expression-Builder

Für alle Aktionen oder Übergangsbedingungen müssen Sie Very Highspeed Integrated Circuit Hardware Description Language (VHDL)-Code schreiben. Für dieses Labor müssen wir nur einfache Bedingungen für die Übergänge schreiben - siehe Listing 2 - oder einfache Signalzuweisungen für die Zustände - siehe Listing 1 -.

Die zweite Symbolleiste enthält die Schaltfläche zum öffnen des Expression-Builder Abbildung 5. Damit können Sie neue Ausdrücke für die Bedingungen der Übergänge und den Code innerhalb der Zustände erstellen.

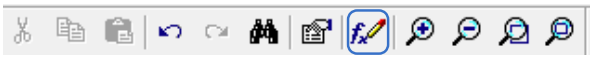


Abbildung 5 - Toolbar zum Öffnen des Expression-Builders

Um eine Zustandsaktion oder eine Übergangsbedingung zu bearbeiten, können Sie im Editor auf den Zustand oder die Transition doppelklicken. Dadurch wird ein Code-Editor geöffnet, in dem Sie den VHDL-Code für die Aktion oder Bedingung schreiben/bearbeiten können.

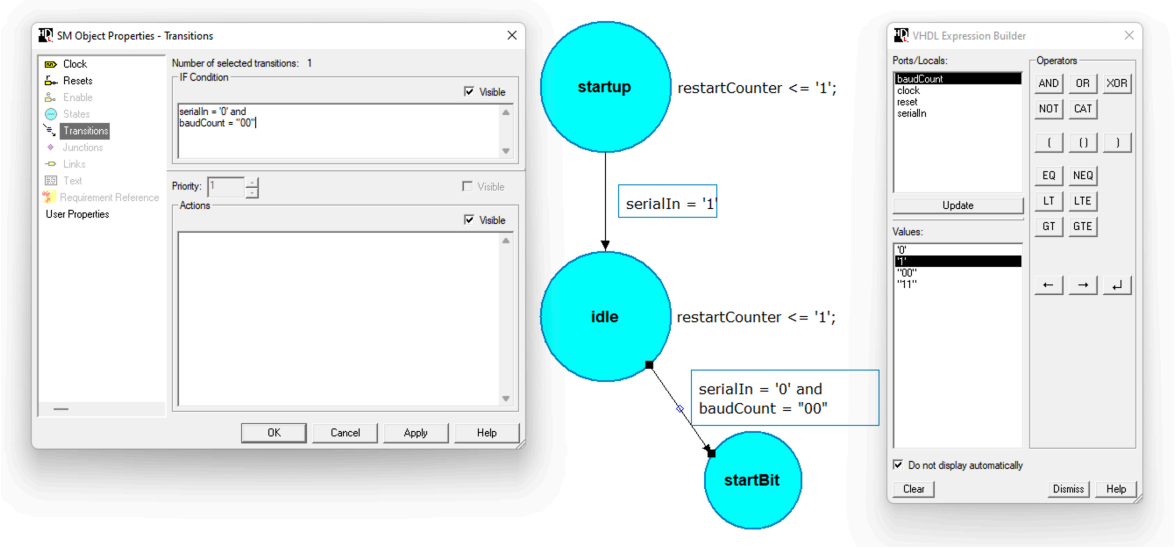


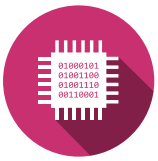
Abbildung 6 - Code-Editor für eine Zustandsaktion oder Übergangsbedingung

Ein Vergleich mit einem Ein-Bit-Signal wie **`serialIn`** oder einem Mehr-Bit-Signal wie **`baudCount`** ist etwas anders.

1. Vergleiche werden nicht mit **`==`** wie in anderen Programmiersprachen durchgeführt, sondern nur mit einem **`=`**.
2. Der Vergleichswert für Ein-Bit-Signale in einfache Anführungszeichen **`'`** und für Mehr-Bit-Signale in doppelte Anführungszeichen **`"`** eingeschlossen ist.
3. Sie können mehrere Bedingungen mit booleschen Operatoren wie **`and`**, **`or`** und **`not`** kombinieren.

Zum Beispiel, um zu überprüfen, ob der serielle Eingang **`serialIn`** niedrig ist und der Zähler **`baudCount`** Null ist, können Sie schreiben:

```
  serialIn = '0' and baudCount = "00"
--  \_____________/  ^  \_____________/
--         |         |          +-- Vergleich eines Mehrbit-Signals
--         |         +------------- boolescher Operator
--         +----------------------- Vergleich eines Ein-Bit-Signals
```



4 Realisierung

Die drei Hauptkomponenten des seriellen Portempfängers, die implementiert werden sollen, sind:

- **`COM/shiftRegister`** zum Sammeln der eingehenden Bits und Umwandlung in ein paralleles Format.
- **`COM/baudrateCounter`** zum Teilen der Systemuhr und Synchronisierung mit der Baudrate der eingehenden Daten.
- **`COM/receiverController`** zur Steuerung des Empfangsprozesses, Erkennung des Startbits, Abtastung jedes Datenbits zur richtigen Zeit und Validierung des Stoppbits.

4.1 Analyse

Um die serielle Datenübertragung präzise zu steuern, ist das Timing der empfangenen Bits entscheidend.

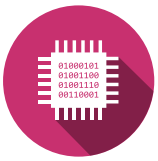
Das Senden am seriellen Port erfolgt durch den in den vorherigen Laboren entwickelten μProzessor Picoblaze.

Der verwendete Code ist der lineare, ohne Schleifen, wie unter Listing 3 dargestellt:

```
LOAD        s3, FF              ; load stop bit
OUTPUT      s3                  ; output stop bit
LOAD        s3, s3              ; no operation
LOAD        s3, s3              ; no operation
LOAD        s3, s3              ; no operation
LOAD        s3, s3              ; no operation
LOAD        s0, 00              ; load start bit
OUTPUT      s0                  ; output start bit
INPUT       s1                  ; load word to send
OUTPUT      s1                  ; output word, LSB is considered
SR0         s1                  ; shift word, bit 1 -> LSB
OUTPUT      s1                  ; output bit 1
SR0         s1                  ; bit 2 -> LSB
OUTPUT      s1                  ; output bit 2
SR0         s1                  ; bit 3 -> LSB
OUTPUT      s1                  ; output bit 3
SR0         s1                  ; bit 4 -> LSB
OUTPUT      s1                  ; output bit 4
SR0         s1                  ; bit 5 -> LSB
OUTPUT      s1                  ; output bit 5
SR0         s1                  ; bit 6 -> LSB
OUTPUT      s1                  ; output bit 6
SR0         s1                  ; bit 7 -> LSB
OUTPUT      s1                  ; output bit 7
LOAD        s3, s3              ; no operation
OUTPUT      s3                  ; output stop bit
```

Listing 3 - Linearer Algorithmus

Indem Sie den gegebenen Code Listing 3 lesen und sich den Block **`COM_test/serialPortReceiver_tb`** ansehen, bestimmen Sie, welcher serielle Code vom Prozessor gesendet wird.



Bestimmen Sie die Anzahl der für das Senden eines Bits erforderlichen Taktzyklen.

Beziehen Sie sich dazu auf den Code Listing 3, die im vorherigen Labor **`CNT`** durchgeführte Baudratenberechnung sowie das **`serialOut`**-Signal des **`COM_test/serialPortReceiver_tb`**-Schaltkreises, wenn es mit der Datei **`$SIMULATION_DIR/COM.do`** simuliert wird.

4.2 Entwicklung

Mit allen Informationen aus dem vorherigen Abschnitt können Sie nun die drei Hauptkomponenten des seriellen Portempfängers implementieren.

4.2.1 Schieberegister

Daten werden seriell, ein Bit nach dem anderen, empfangen und müssen für die weitere Verarbeitung in einem parallelen Format gesammelt und gespeichert werden. Ein Schieberegister ist hierfür ideal, da es jedes eingehende Bit auf einem Steuersignal verschieben und das vollständige Byte parallel ausgeben kann.

Implementieren Sie das Schieberegister **`COM/shiftRegister`**. Jedes Mal, wenn die Eingabe verschoben werden muss, wird das Signal **`shiftEn`** auf **`1`** gesetzt.

4.2.2 Baudrate-Zähler

Um die eingehenden seriellen Daten genau abtasten zu können, wird ein Zähler benötigt, der die Systemuhr auf die Baudrate der eingehenden Daten teilt. Dieser Block erzeugt einen Zählwert, der verwendet werden kann, um zu wissen, wann die eingehenden Bits abgetastet werden sollen, wo wir uns im empfangenen Rahmen befinden und/oder wann für den nächsten Rahmen zurückgesetzt werden soll…
Beim Abtasten eines Bits muss dies so nah wie möglich an der Mitte der Bitperiode erfolgen.

- Bestimmen Sie die Anzahl der für den Zähler **`COM/baudrateCounter`** erforderlichen Bits, abhängig davon, wie Sie ihn verwenden werden.
- Geben Sie die erforderliche Anzahl von Bits im System ein. Doppelklicken Sie dazu im **`COM/serialPortReceiver`** auf die Konstantendefinition **`baudCounterBitNb`** oben links im Schaltplan und bearbeiten Sie sie im neu geöffneten Fenster unter **`User Declarations`**, wie unter Abbildung 7 gezeigt, und klicken Sie dann auf **`Apply`**:

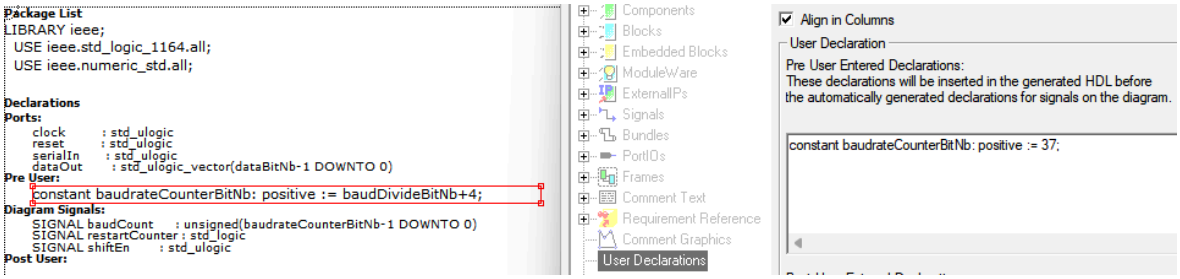


Abbildung 7 - Anzahl der Bits des Systemzählers ändern

Implementieren Sie den Zähler **`COM/baudrateCounter`**.

- Vergessen Sie nicht, die Anzahl der Bits des Signals **`baudCount`** auf dem **`COM/serialPortReceiver`** zu validieren/ändern.
- Der Zähler muss eine synchrone Rücksetzung mit dem Signal **`restartCounter`** unterstützen.

4.2.3 Empfängersteuerung (FSM)

Die endliche Zustandsmaschine (FSM) ist das Gehirn des seriellen Portempfängers. Sie überwacht die serielle Eingangsleitung, erkennt den Übergang, der den Beginn eines Frames anzeigt (Startbit), und löst regelmässige Abtastimpulse basierend auf dem Zählerwert aus. Jedes abgetastete Bit wird in das Register verschoben, bis das vollständige Byte empfangen wurde. Die Ausgabe wird dann in paralleler Form für nachfolgende Stufen des Systems bereitgestellt.

Implementieren Sie die endliche Zustandsmaschine **`COM/receiverController`**. Sie muss den Beginn und das Ende eines Pakets erkennen und die Signale **`restartCounter`** sowie **`shiftEn`** entsprechend generieren.

4.3 Simulation

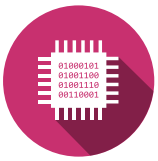
Nach der Implementierung simulieren Sie die drei Komponenten mit der Testbench **`COM_test/serialPortReceiver_tb`**. Der μProzessor Xilinx Picoblaze, der in den vorherigen Laboren entwickelt wurde, wird verwendet, um die seriellen Daten an den **`COM/serialPortReceiver`** zu senden.

Simulieren Sie den Testbench **`COM_test/serialPortReceiver_tb`** mit der Simulationsdatei **`$SIMULATION_DIR/COM.do`**.

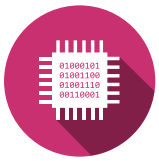
Validieren Sie, dass die verschiedenen gesendeten Bytes korrekt auf dem Signal **`dataOut`** decodiert werden.

4.4 Systemanalyse

Jetzt, da der serielle Portempfänger implementiert und simuliert ist, beantworten Sie die folgenden Fragen:



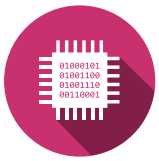
1. Wie lange dauert es, bis ein Bit vom μProzessor Picoblaze in Bezug auf die Zeit (d.h. Übertragungsbitrate) gesendet wird?
2. Wie lange dauert es, bis zwei Daten empfangen werden?
3. Warum werden die Datenbits so nah wie möglich an der Mitte der Bitperiode abgetastet?
4. Was passiert, wenn die Baudrate des Senders nicht mit der Baudrate des Empfängers übereinstimmt?



5 Checkout

Dies ist das Ende des Labors, Sie haben erfolgreich ein System mit mehreren verschiedenen Blöcken entwofren. Bevor Sie das Labor verlassen, stellen Sie sicher, dass Sie die folgenden Aufgaben erledigt haben:

- ☐ Theorie
  - ☐ Sie verstehen wie man im HDL-Designer Zustandsmaschinen FSM erstellt.
- ☐ Schaltungsentwurf
  - ☐ Die drei Blöcke **`COM/shiftRegister`**, **`COM/baudrateCounter`** und **`COM/receiverController`** wurden erstellt und getestet.
- ☐ Simulationen
  - ☐ Die gesendeten Daten des **`COM/nanoprocecssor`** werden vom **`COM/serialPortRecevier`** korrekt gelesen und parallelisiert.
  - ☐ Die einzelnen Bits werden so nahe an der Mitte gelesen wie möglich.
- ☐ Analyse
  - ☐ Sie haben die Analysefragen zum System beantwortet.
- ☐ Dokumentation und Projektdateien
  - ☐ Stellen Sie sicher, dass alle Schritte (Entwurf, Simulationen, Vergleich) gut in Ihrem Laborbericht dokumentiert sind.
  - ☐ Speichern Sie das Projekt auf einem USB-Stick oder im gemeinsamen Netzwerkordner (**`\\filer01.hevs.ch`**).
  - ☐ Teilen Sie die Dateien mit Ihrem Laborpartner, um die Kontinuität der Arbeit sicherzustellen.



Glossar

***FSM* – Finite State Machine** 1, 1, 1, 2, 2, 3, 3, 3, 3, 4, 4, 4, 8, 10

***LSB* – Least Significant Bit** 2

PicoBlaze: PicoBlaze is a small, 8-bit microcontroller designed by Xilinx for use in FPGAs. It is often used in educational settings to teach basic microcontroller concepts. 1

***VHDL* – Very Highspeed Integrated Circuit Hardware Description Language** 4, 4