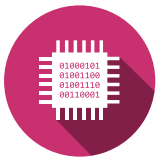


Serieller Empfänger

Labor Digitales Design

Inhalt

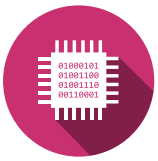
1 Ziele

Dieses Labor zielt darauf ab, die Konstruktion synchroner digitaler Schaltungen unter Verwendung einer 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.



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.

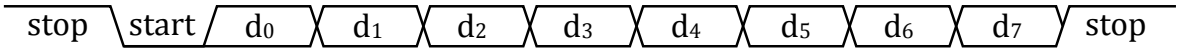


Abbildung 1 - Serielle Übermittlung

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.

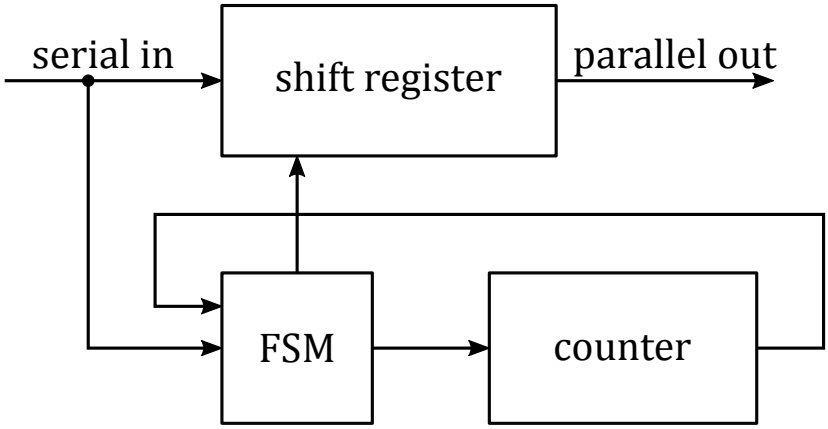
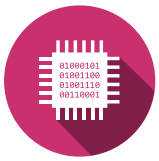


Abbildung 2 - Blockschaltung des seriellen Empfängers

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`** wird eine FSM implementiert, um den Empfang serieller Daten zu steuern. Die FSM verwaltet das Timing der Abtastung eingehender Bits und stellt sicher, dass jedes Bit zur richtigen Zeit relativ zur Baudrate des eingehenden Datenstroms gelesen wird.

3.1 Elemente der gegebenen FSM

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

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“.

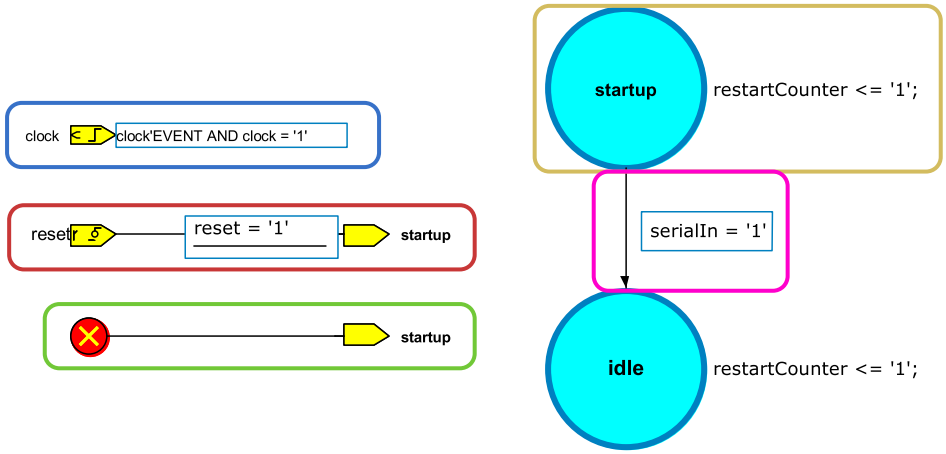


Abbildung 3 - Elemente der gegebenen FSM

3.2 Zustände und Übergänge

Die FSM wird mehrere Zustände haben, die jeweils eine spezifische Phase des Empfangsprozesses repräsentieren. Die Übergänge zwischen diesen Zuständen werden durch Ereignisse ausgelöst, die in den Pfeilen definiert sind. Wenn der Pfeil keine Bedingung hat, bedeutet dies, dass der Übergang immer aktiv ist und im nächsten Taktzyklus in jedem Fall zum nächsten Zustand wechselt.

Oben im Editor befinden sich mehrere Symbolleisten mit allen erforderlichen Funktionen zum Bearbeiten der FSM. Die erste Symbolleiste Abbildung 4, die es ermöglicht, neue Zustände (blauer Kreis) und Übergänge (schwarzer Pfeil) zu erstellen.

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.

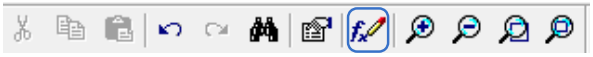


Abbildung 5 - Toolbar zum Öffnen des Expression-Builders

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.

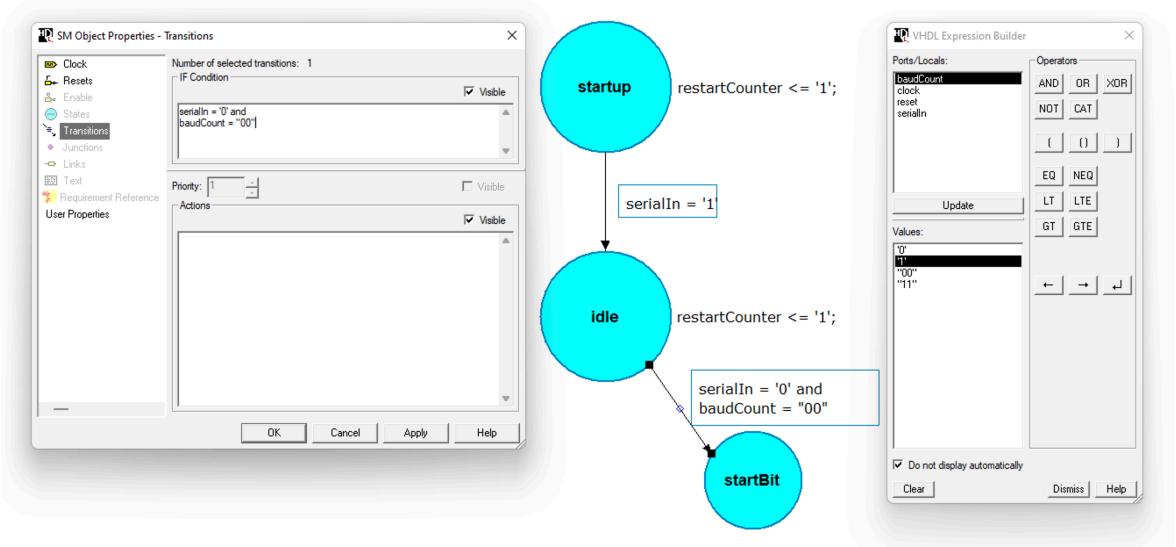


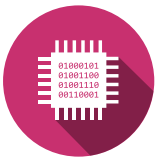
Abbildung 6 - Code-Editor für eine Zustandsaktion oder Übergangsbedingung

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

1. Zunächst einmal ist kein Vergleich mit zwei Gleichheitszeichen **`==`** wie in anderen Programmiersprachen erforderlich, sondern nur ein **`=`**.
2. Der zweite Unterschied besteht darin, dass der Vergleichswert für Ein-Bit-Signale in einfache Anführungszeichen **`'`** und für Mehr-Bit-Signale in doppelte Anführungszeichen **`"`** eingeschlossen ist.
3. Mit booleschen Operatoren wie **`and`**, **`or`** und **`not`** können Sie mehrere Bedingungen miteinander 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.

Starten Sie eine Simulation des **`COM_test/serialPortReceiver_tb`** mit der Datei **`$SIMULATION_DIR/COM.do`**

- Bestimmen Sie die Anzahl der Taktzyklen, die der Testbench benötigt, um ein einzelnes Bit zu senden, indem Sie das Signal **`serialOut`** analysieren.
- Bestimmen Sie die Anzahl der Bits, die für den Zähler **`COM/baudrateCounter`** erforderlich sind.

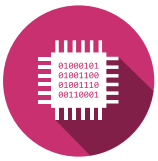
4.2 Entwicklung

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

Implementieren Sie das Schieberegister **`COM/shiftRegister`**. Die 8 Bits serieller Daten pro Frame werden im parallelen Format ausgegeben. Bei jedem Impuls des Signals **`shiftEn`** wird das nächste Bit in das Schieberegister verschoben.

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.
- Die Daten müssen so nah wie möglich an der Mitte der Bitperiode abgetastet werden.



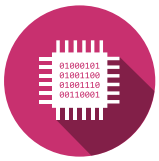
Implementieren Sie die endliche Zustandsmaschine **`COM/receiverController`**. Sie muss den Beginn und das Ende eines Pakets erkennen und die Signale **`restartCounter`** sowie **`shiftEn`** entsprechend generieren.

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`**.

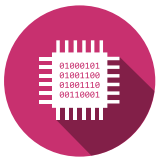
- Validieren Sie die gesendeten und decodierten Daten.
- Berechnen Sie die Baudrate der seriellen Übertragung.



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.
- ☐ 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, 1, 1, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 8

***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