4. Übung

Fachgebiet Architektur eingebetteter Systeme
Hardwarepraktikum (TI)

AES Embedded Systems Architecture
SoSe 20

Ausgabe: 11.05.2020

Theorie Präsentation: 25./26.05.2020
Praxis Abgabe: 07.06.2020

Arbeitsziel

Verständnis für die Funktionsweise einer einfachen Kommunikationsschnittstelle (RS232), Implementierung eines RS232-Senders als Finite State Machine.

Arbeitsmaterialien

- Modul `PISOShiftReg.vhd`
- Testbench `PISOShiftReg_tb.vhd`
- Modul `ArmRS232Interface.vhd`
- Testbench `ArmRS232Interface_tb.vhd`

Abgabedateien

- Datei `PISOShiftReg.vhd`
- Datei `PISOShiftReg_tb.vhd`
- Datei `ArmRS232Interface.vhd`

Theoretische Vorbetrachtungen

Die folgenden Fragen sind von der eingeteilten Gruppe in einem kurzen Vortrag zu beantworten. Alle Fragen bezüglich der ARM-Architekur beziehen sich immer auf ISA ARMv4:

- Was ist *memory-mapped Input/Output* (speicherbezogene Ein-/Ausgabe), wie können mit diesem Prinzip Ein-/Ausgabegeräte (z.B. eine serielle Schnittstelle) an den ARM-Prozessor angebunden werden und welche Art von Instruktionen eignet sich zum Zugriff auf memory-mapped angebundene Peripherie?
- Wie werden Nutzdaten über eine serielle Leitung gemäß des Standards EIA-232 (bzw. RS232) übertragen? Was wird neben den Nutzdaten noch übertragen (Beschreibung des Datenrahmens) und welche Bedeutung hat die Bitrate/Baudrate für die Signale auf der Leitung?
- Wie kann für die serielle Übertragung ein Schieberegister genutzt werden? Skizzieren Sie eine Lösung auf Register-Transfer-Ebene die zur Übertragung ein Schieberegister verwendet.
- Endliche Zustandsautomaten (Finite State Machines, FSM) können (unter anderem) vom Moore- oder Mealy-Typ sein. Worin unterscheiden sich beide Typen, worin gleichen sie sich?
- Welche Syntaxvarianten (bezogen auf die Zahl der Prozesse) zur Beschreibung endlicher Zustandsautomaten gibt es in VHDL und wie unterscheiden sie sich?

Empfohlene Quellen zur Bearbeitung:

- Mikroprozessortechnik und Rechnerstrukturen, [1, Kapitel 5.2]
- VHDL-Synthese, [2, Kapitel 6]

Aufgabenbeschreibung

In dieser Übung soll ein RS232-Transmitter (der Sender einer seriellen Schnittstelle) in VHDL als FSM implementiert werden.

Aufgabe 1 (4 Punkte) Implementierung eines PISO-Schieberegisters

Ziel dieses Aufgabenblattes ist die Implementierung eines RS232-Transmitters. Bei RS232 handelt es sich um eine serielle Schnittstelle, Daten werden also "Bit für Bit" nacheinander übertragen. Die zu übertragenden Daten werden aber jeweils byteweise (zuzüglich Start- und Stoppbit!) an die Schnittstelle übergeben, müssen also serialisiert werden. Dazu kann ein ein PISO-Schieberegister (Parallel-In, Serial-Out) verwendet werden. Dieses hat einen parallelen Eingang, an dem ein Datum mit einer bestimmten Breite gelesen wird, aber einen 1-bit-breiten, seriellen Ausgang, an dem das Datum dann bitweise ausgegeben wird. In dieser Aufgabe sollen Sie ein solches Schieberegister implementieren.

Das Schieberegister soll taktsynchron ein Eingangsdatum lesen und in einem internen Register speichern. Am Datenausgang des Schieberegisters soll immer das niederwertigste Bit des internen Registers angezeigt werden. Sobald der Schiebevorgang startet, soll das Datum im Register taktweise logisch nach rechts geschoben werden, so dass die einzelnen Bits des Datums nacheinander bis zum höchstwertigsten Bit ausgegeben werden. Nachdem das letzte Bit ausgegeben wurde, soll am Ausgang '0' gehalten werden, bis ein neues Datum gelesen wird und ein neuer Schiebevorgang startet.

Das Schieberegister verfügt nicht nur über einen Clock-Eingang, sondern auch über einen Clock-Enable-Eingang. Nur während das Clock-Enable-Signal aktiv ist, soll das Schieberegister (taktsynchron) arbeiten. Das ist sinnvoll, weil die Baudrate der RS232-Schnittstelle deutlich geringer ist, als die Taktfrequenz des Prozessors. Wäre das Clock-Enable-Signal also permanent aktiv, würde jedes Bit für genau einen Takt angezeigt werden. Setzt man es hingegen nur alle paar Takte auf `'1'`, lässt sich damit die Frequenz des Schieberegisters reduzieren.

Vervollständigen Sie das Modul **PISOShiftReg.vhd**, welches über die Schnittstelle aus Tabelle 1 verfügt. Zudem soll die Breite des Schieberegisters über die *Generic* `WIDTH` angepasst werden können.

Testen Sie ihre Implementierung selbstständig mit einer eigenen Testbench. Verwenden Sie als Grundlage die Testbench **PISOShiftReg_tb.vhd**, in der die Instanziierung des Schieberegisters sowie die Erzeugung der Clock- und Clock-Enable-Signale bereits vorgegeben ist. Ihre Testbench sollte mindestens ein Testdatum in das Schieberegister schreiben, dann den Schiebevorgang starten und das vollständige Datum ausgeben. Die Testbench muss NICHT self-checking sein[1]. Es reicht, wenn Sie die entsprechenden Eingangssignale setzen und die Ausgangs-Signalverläufe in der Simulation manuell überprüfen. Geben Sie ihre Testbench mit ab.

[1] Das heißt, die Testbench muss nicht selbstständig überprüfen, ob die getestete Komponente die korrekte Ausgabe erzeugt

CLK	in	Das Taktsignal.
CLK_EN	in	Das Clock-Enable-Signal. Nur wenn das Signal auf '1' gesetzt ist, soll das Schieberegister arbeiten, ist das Signal auf '0' gesetzt, so wird das Taktsignal CLK ignoriert und das Schieberegister bleibt in seinem aktuellen Zustand.
LOAD	in	Solange das Signal auf '1' gesetzt ist, soll taktsynchron das aktuell an D_IN anliegende Datum gelesen und in einem internen Register gespeichert werden. Sobald das Signal auf '0' gesetzt wird, beginnt das Schieberegister, das zuletzt gelesene Datum taktsynchron nach rechts zu schieben.
D_IN	in	Zu schiebendes Eingangsdatum mit durch WIDTH festgelegter Breite.
D_OUT	out	Datenausgang, zeigt immer das niederwertigste Bit des internen Registers an, in dem das gelesene Datum taktweise nach rechts geschoben wird. Somit werden nacheinander die einzelnen Bits des Datums ausgegeben, vom niederwertigsten zum höchstwertigen Bit. Während LOAD auf '1' gesetzt ist, soll am Datenausgang stets das niederwertigste Bit des zuletzt gelesenen Datums angezeigt werden. Sobald LOAD auf '0' gesetzt wird, beginnt der Schiebevorgang (im ersten Takt mit LOAD = '0' wird also bereits das zweitniedrigste Bit des zuletzt gelesenen Datums angezeigt, weil zu Beginn des Taktes bereits der erste Rechtsshift stattgefunden hat). Sobald das aktuelle Datum vollständig aus dem Register geschoben wurde, soll der Datenausgang '0' anzeigen, bis ein neues Datum gelesen wird.
LAST_BIT	out	Dieses Signal zeigt an, dass das gelesene Datum vollständig aus dem Ausgang geschoben wurde. Es wird in dem Takt '1', in dem das höchstwertige Bit des Datums am Ausgang ausgegeben wird und schaltet erst wieder auf '0', sobald ein neues Datum gelesen wird. Dies wird sich später bei der Implementierung des RS232-Transmitters als nützlich erweisen, um zu erkennen, dass ein zu sendendes Datum vollständig übertragen wurde.

Tabelle 1: Schnittstelle des PISO-Schieberegisters

Aufgabe 2 (8 Punkte) Implementierung eines RS232-Senders

Vervollständigen Sie das Modul **ArmRS232Interface.vhd**.

Das Modul enthält drei VHDL-Funktionsblöcke. Neben der folgenden Beschreibung derselben finden Sie auf der letzten Seite des Aufgabenblattes ein Blockbild, das die Kommunikation zwischen den drei Blöcken visualisiert:

INTERFACE_COMMUNICATION (vorgegeben):

Der Funktionsblock verbindet eine *RS232*-Schnittstelle, bestehend aus Sender (Transmitter) und Empfänger (Receiver), mit dem Datenbus des Prozessors. Insbesondere wird dadurch von den Bussignalen und dem Busprotokoll abstrahiert, sodass die anderen Funktionsblöcke unabhängig von der genauen Arbeitsweise des Datenbus implementiert werden können. Im Block werden 4 Register (eines ist derzeit funktionslos und konstant 0) verwaltet, die durch einen Busmaster am Datenbus angesprochen werden können. Ein Senderegister (RS232_TRM_REG) nimmt das nächste zu sendende Datum entgegen. Dabei löst der Schreibzugriff auf das Register durch einen Busmaster bereits das Senden aus, er hat also einen *Seiteneffekt*. Zwei Statusbits in einem Statusregister (RS232_STAT_REG) zeigen an,

ob aktuell ein Datum gesendet wird (der Sender also belegt ist) und ob ein neues Datum empfangen wurde. Das zuletzt durch den Empfangsblock entgegengenommene Datum wird in einem Empfangsregister (RS232_RCV_REG) am Bus verfügbar gemacht. Ein Lesezugriff auf dieses Register setzt als Seiteneffekt das Statusbit zur Anzeige eines neu empfangenen Datums zurück. Eben dieses Statusbit treibt eine Signalleitung, die mit einem der ARM-Interrupteingänge verbunden werden kann, sodass der Prozessor zügig auf neue Daten reagiert.

RS232_RECEIVER (vorgegeben):

Der Receiver beobachtet permanent eine *RS232*-Empfangsleitung (RS232_RXD). Er nimmt jeweils ein Nutzdatenbyte Bit für Bit entgegen und informiert anschließend die Busschnittstelle (INTERFACE_COMMUNICATION) durch Setzen des Signals DATA_RECEIVED für einen Takt. Das empfangene Datum wird im Register RECEIVER_DATA zur Verfügung gestellt. INTERFACE_COMMUNICATION kopiert das Nutzdatenbyte nach RS232_RCV_REG.

RS232_TRANSMITTER:

Implementieren Sie den Sender selbstständig! Die Busschnittstelle (INTERFACE_COMMUNICATION) startet die Übertragung durch Setzen des Signals START_TRANSMISSION für einen Takt. Der Transmitter übernimmt (kopiert!) daraufhin das niederwertigste Byte des Registers RS232_TRM_REG, setzt das Signal TRANSMITTER_BUSY und sendet die Nutzdaten zusammen mit einem Startbit, einem Stoppbit und ohne Paritätsbit über die RS232-Sendeleitung RS232_TXD. TRANSMITTER_BUSY muss unbedingt in dem Takt gesetzt werden, in dem START_TRANSMISSION = 1 auftritt und wird erst nach Abschluss einer Übertragung zurückgesetzt. Die vorgesehene Baudrate ist in der Konstanten RS232_BAUDRATE in **ArmConfiguration** hinterlegt. Die Dauer eines Symbols, ausgedrückt in Taktperioden des externen Taktsignals (SYS_CLK), kann der Konstanten RS232_DELAY entnommen werden. Zur Realisierung des notwendigen Zeitverhaltens kann (muss aber nicht) die Verzögerungsschaltung (**ArmWaitStateGenAsync.vhd**) vom 0. Aufgabenblatt verwendet werden. Implementieren Sie den Sender als FSM (schauen Sie hierzu in das Theorievideo). Im idealen Fall erhalten Sie zwei FSM-Zustände, es sind aber auch mehr erlaubt. Beachten Sie, dass sich die durch die FSM erzeugten Ausgänge nur taktsynchron ändern sollen. Zeichnen Sie vor der Umsetzung in VHDL das Zustandsdiagramm. Sie können Ihr Schieberegister aus Aufgabe 1 verwenden. Es ist aber auch möglich, die Schieberegister-Funktionalität direkt im Code der FSM-Prozesse umzusetzen, Sie können selbst entscheiden.

Testen Sie Ihre Implementierung mit der Testbench **ArmRS232Interface_tb.vhd**. Sie übernimmt einerseits die Rolle eines Busmasters, der den Transmitter zum Senden von Daten veranlasst, andererseits auch die des Kommunikationspartners an der seriellen Gegenstelle. Die Testbench überprüft, ob die über die Sendeleitung übertragenen Nutzdaten dem Erwartungswert entsprechen. Dazu ermittelt sie den Beginn des Startbits einer Übertragung und tastet die serielle Leitung jeweils in der (zeitlichen) Mitte der Nutzdatenbits ab. Zusätzlich überprüft die Testbench, ob das Timing jeden Bits korrekt ist, es also (mit ca. 5% Toleranz) weder zu kurz noch zu lang auf der Leitung angezeigt wurde. Außerdem werden auch unerwartete Signalflanken (Transitionen) innerhalb des zeitlichen Rahmens eines Nutzdatenbits erkannt. Sie sollten zuerst dafür sorgen, dass ihr Transmitter keine Nutzdatenfehler mehr produziert und anschließend evtl. auftretende Timing- und Transitionsfehler untersuchen.

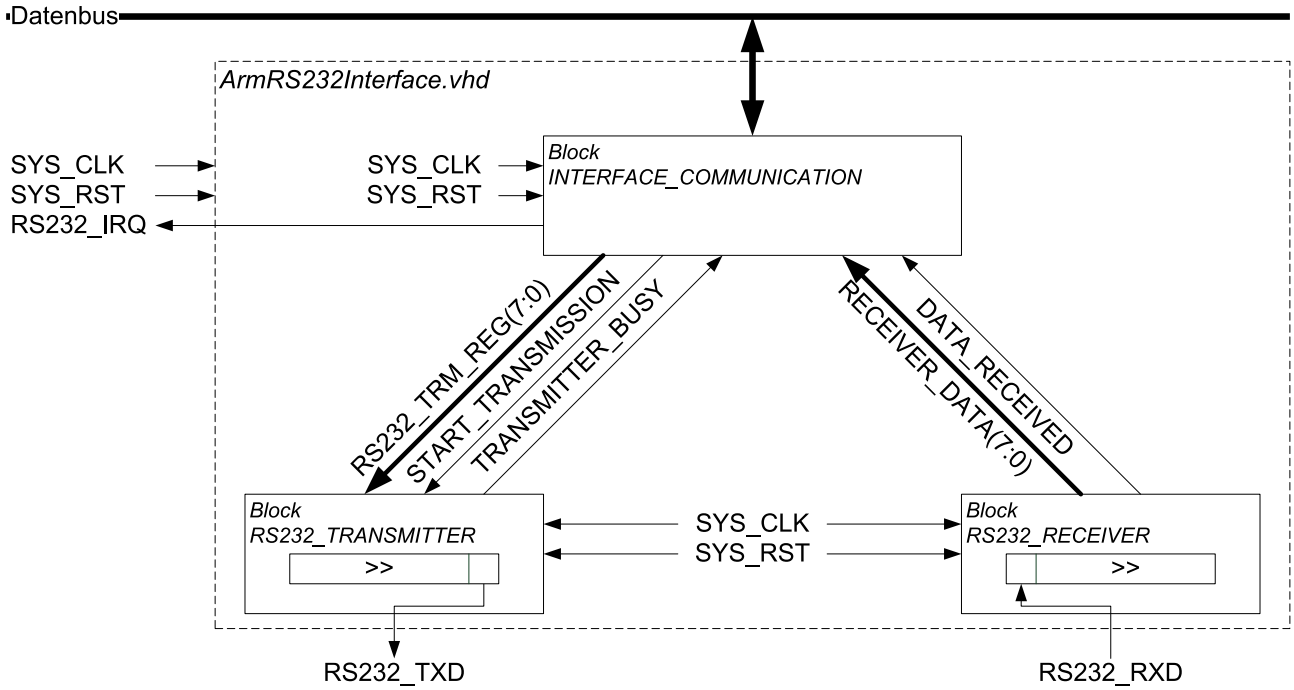

Abbildung 1: Blockschaltbild der RS232 Schnitstelle

HINWEIS

Ihr Sender erzeugt lediglich die logischen Signale auf der Sendeleitung. Eine logische 1 entspricht weiterhin dem Wert `1` von `std_logic`. Die Anpassung an Spannungspegel und negative Logik der physischen Schnittstelle erfolgt außerhalb des FPGAs.

Die Zustände des Automaten können in Form eines VHDL-Typs dargestellt werden. Die konkrete Zustandscodierung wird dann durch das Synthesewerkzeug festgelegt.

Mündliche Rücksprache - 4 Punkte

Literatur

[1] Thomas Flik. *Mikroprozessortechnik und Rechnerstrukturen*. 7., neu bearb. Aufl. Springer Berlin, 2004. ISBN: 3-540-22270-7.

[2] Jürgen Reichardt und Bernd Schwarz. *VHDL-Synthese: Entwurf digitaler Schaltungen und Systeme*. 4., überarbeitete Auflage. Oldenbourg, Okt. 2007. ISBN: 3486581929.