4. Übung	Fachgebiet Architektur eingebetteter Systeme **Hardwarepraktikum (TI)**	AES Embedded Systems Architecture SoSe 17

Ausgabe: 17.05.2017

Theorie Präsentation: 24.05.2017
Praxis Abgabe: 31.05.2017

Arbeitsziel

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

Arbeitsmaterialien

- Modul `Ampel.vhd`
- Modul `ArmRS232Interface.vhd`
- Testbench `ArmRS232Interface_tb.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. Zur Einführung von FSMs in VHDL wird zuerst eine einfache Beispielanwendung realisiert.

Aufgabe 1 (4 Punkte) Implementierung einer vereinfachten Ampelsteuerung

Der Automatengraph aus Abbildung 1 beschreibt (etwas realitätsfern) die Steuerung einer Ampel. Die

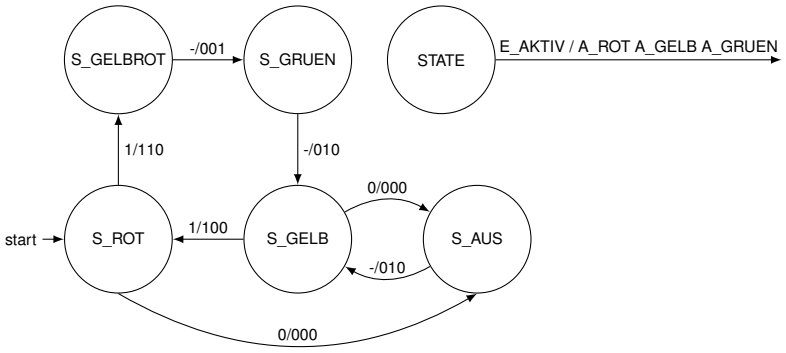

Abbildung 1: Automat zur Steuerung einer Ampel

Steuerung verfügt über drei „Ausgangssignale“, mit denen die Leuchten der Ampel aktiviert werden. Diese Ausgangssignale sind mit `A_ROT`, `A_GELB` bzw. `A_GRUEN` benannt.

Unsere Ampel zeigt im Normalbetrieb 4 verschiedene Leuchtkombinationen: grün, gelb, rot, gelbrot, welche in genau dieser Reihenfolge, gesteuert durch ein externes Taktsignal, durchlaufen werden. Jede Kombinationen ist durch einen Zustand des Automaten repräsentiert: `S_GRUEN`, `S_GELB`, `S_ROT` und `S_GELBROT`, welche im Normalbetrieb sequenziell durchlaufen werden.

Im Notfallbetrieb jedoch soll der Ausgang `A_GELB` „blinken“. Dies wird durch einen permanenten Zustandsübergang zwischen `S_GELB` und dem zusätzlichen Zustand `S_AUS` erreicht. Das Eingangssignal `E_AKTIV` bestimmt, ob die Ampel im Normalbetrieb (`E_AKTIV = 1`) oder Notfallbetrieb (`E_AKTIV = 0`) arbeitet. Unmittelbare Übergänge in den Notfallbetrieb sind aber nur aus den Zuständen `S_ROT` und `S_GELB` möglich. Dass ein Zustandsübergang nicht vom Eingangssignal `E_AKTIV` abhängt, wird durch ein don't care („-“) gekennzeichnet.

Der Automat verfügt über ein synchrones Reset-Signal, welches ihn in den Startzustand `S_ROT` versetzt, unabhängig davon, in welchem Zustand er sich bei Eintreten des Resets befindet.

Die Ausgangssignale („Farben“) werden beim Übergang zwischen den Zuständen in Abhängigkeit von aktuellem Zustand und Eingangssignal `E_AKTIV` gesetzt, beim Übergang von `S_GELBROT` nach `S_GRUEN` wird beispielsweise `A_GRUEN` auf `1` gesetzt.

Vervollständigen Sie das vorgegebene Modul **Ampel.vhd**, welche über die Schnittstelle aus Tabelle 1 verfügt. Realisieren Sie die Ampelsteuerung als Zwei-Prozess-FSM.

Ein Prozess modelliert den Zustandsübergang mit jeder steigenden Taktflanke des Moduleingangssignals `CLK`. Im zweiten (asynchronen) Prozess wird der Nachfolgezustand und der Wert der Ausgangssignale bestimmt. Der Typ zur Modellierung der Zustände sowie zwei Signale für aktuellen und Nachfolgezustand sind bereits vorgegeben. Testen Sie die Funktionalität Ihrer Steuerung mittels einer Testbench. Benennen Sie diese Testbench Ampel_tb.vhd.

HINWEIS

Für das Synthetisieren von FSMs bietet es sich an, in *PlanAhead* die Syntheseeinstellungen anzupassen. Dazu muss zunächst das zu synthetisierende Modul zum Top Module erklärt werden. Über den *Project Manager* haben die Projekteinstellungen (*Project Settings*). Wählen Sie im anschließenden Fenster *Synthesis* aus, um auf die Einstellung für *XST* zugreifen zu können (siehe Abbildung 2).

RST	in	Highaktives Reset-Signal.
CLK	in	Das Taktsignal, mit dessen steigender Flanke Zustandswechsel stattfinden.
E_AKTIV	in	Eingangssignal zur Steuerung von Zustandsübergängen und Ausgängen.
A_ROT	out	Ausgangssignal zur Steuerung des roten Lichts.
A_GELB	out	Ausgangssignal zur Steuerung des gelben Lichts.
A_GRUEN	out	Ausgangssignal zur Steuerung des grünen Lichts.

Tabelle 1: Schnittstelle der Ampelsteuerung

Sie können dort gezielt eine Art der Zustandscodierung für Automaten erzwingen (*fsm_encoding*). Der Hintergrund: bei der abstrakten Beschreibung gültiger Zustände des Automaten in VHDL wird nicht festgelegt, wie deren Codierung aussehen soll. Die fünf Zustände der Ampel könnten beispielsweise durch 5 verschiedene Wertkombinationen von drei Variablen (und damit Flipflops in der synthetisierten Schaltung) realisiert werden, oder z.B. auch durch 5 Variablen, von denen immer genau eine den Wert 1 annimmt (One-Hot-Encoding). Nähere Informationen zu den zur Verfügung stehenden Optionen erhalten Sie in der Programm-Hilfe (Button *Help* unten rechts im *Properties*-Fenster).

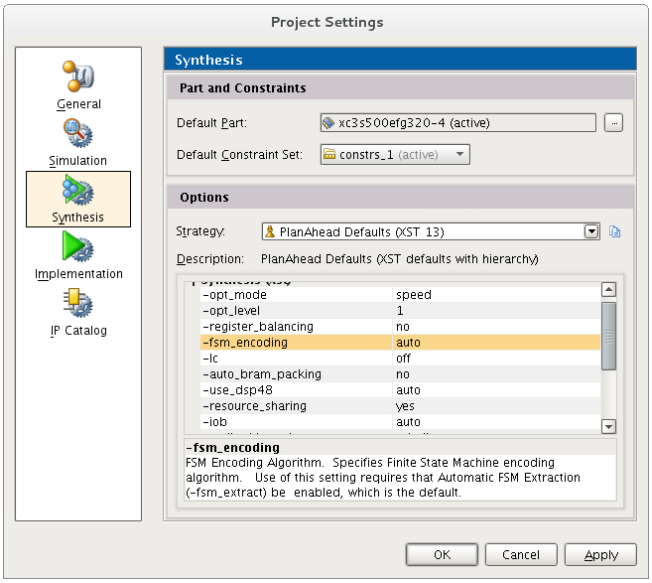

Abbildung 2: PlanAhead Synthese-Einstellungen

In der Vorgabe Ampel.vhd finden Sie etwas Code, der während einer Verhaltenssimulation aktiv ist, aber nicht synthetisiert wird. Er umfasst ein Modell des Automatengraphs in Form einer Adjazenzmatrix und einen Test-Prozess. Innerhalb des Prozesses wird sichergestellt, dass die während einer Simulation auftretenden Zustandsübergänge im Rahmen des Automatenmodells zulässig sind. Teil des Quellcodes ist hier also ein Abschnitt der überprüft, ob zugesicherte Eigenschaften (*Assertions*) der Entwurfsgrundlage (hier: mögliche Transitionen im Automatengraph) eingehalten werden. Es ist gängige Praxis im Hardwareentwurf, Assertions direkt in Module einzubauen und auf diese Weise spätere Tests zu vereinfachen. In VHDL existieren zu diesem Zweck **assert**-Anweisungen, die jedoch nur einfache Bedingungen abtesten und die Verletzung einer Bedingung durch eine Fehlermeldung im Simulator anzeigen können.

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 2-Prozess-FSM wie in Aufgabenteil 1. Zeichnen Sie vor der Umsetzung in VHDL das Zustandsdiagramm.

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.

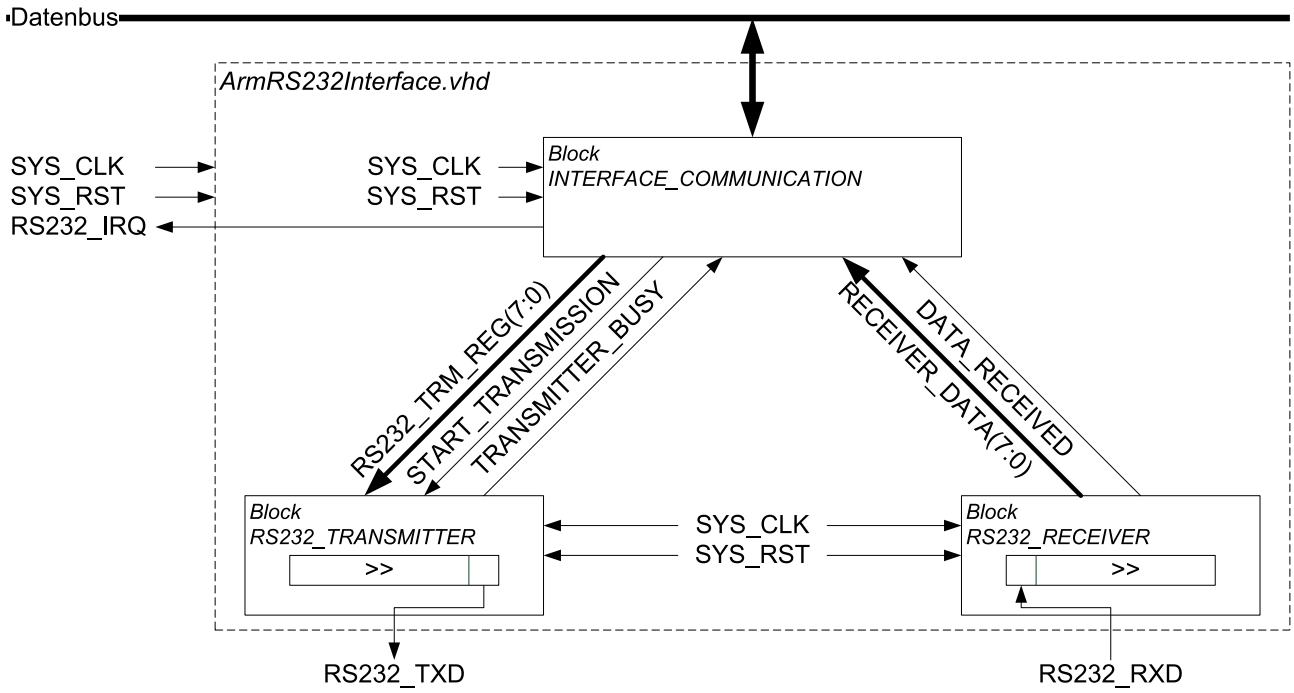

Abbildung 3: Blockschaltbild der RS232 Schnitstelle

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] FLIK, THOMAS: *Mikroprozessortechnik und Rechnerstrukturen*. Springer Berlin, 7., neu bearb. Aufl. Auflage, 2004. ISBN 3-540-22270-7.

[2] REICHARDT, JÜRGEN und BERND SCHWARZ: *VHDL-Synthese: Entwurf digitaler Schaltungen und Systeme*. Oldenbourg, 4., überarbeitete Auflage. Auflage, Oktober 2007. ISBN 3486581929.