Elektronikpraktikum
SS 2010 9.Serie 06.2010
D. Krambrich, W. Lauth, U. Schäfer, S. Tapprogge
Mi. 30.06.10 13:00-16:00 Uhr, oder Do. 01.07.10 13:00-16:00 Uhr
Ort: Gebäude 02-412 (PC-Pool) 3. Stock, Raum 423

Programmierbare Logikbausteine

I. Ziel der Versuche.

Verständnis von programmierbaren Logikbausteinen, Schaltungsbeschreibung in Hardware-Beschreibungssprachen.

II. Vorkenntnisse.

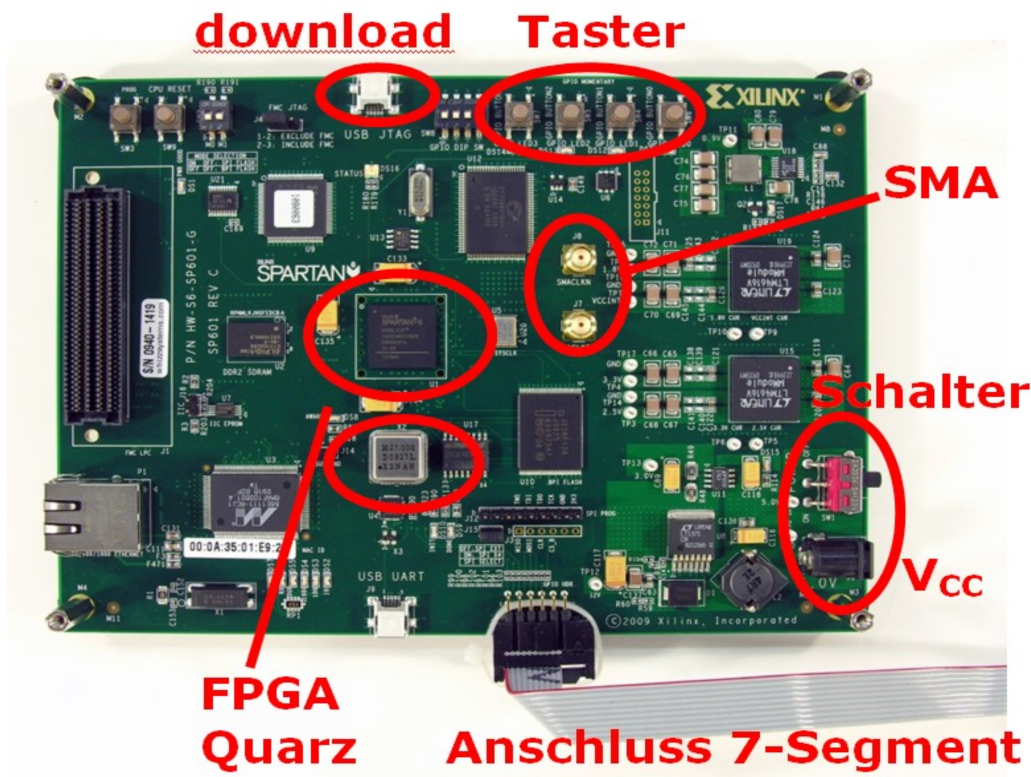
Grundbausteine der digitalen Schaltungstechnik, algorithmische Beschreibung mit Hilfe von Programmiersprachen.

III. Vorbereitung.

Informieren Sie sich über die Hardwarebeschreibungssprache VHDL und arbeiten Sie sich in das vorgegebene Beispiel ein. Entwickeln Sie Verständnis für die konzeptionellen Unterschiede der Hardwarebeschreibung in VHDL zur algorithmischen Beschreibung mit Hilfe üblicher Programmiersprachen.

Programmierbare Logikbausteine und Hardwarebeschreibungssprachen

In der heutigen Praktikumsserie sollen einfache Schaltungen mit Hilfe der Sprache VHDL beschrieben, synthetisiert und implementiert werden. Ein Prototypenmodul steht für die Überprüfung der Funktionalität der Schaltung auf einem FPGA zur Verfügung.



Empfehlung

Nutzen Sie die Installation des "XILINX ISE" Softwarepakets im PC-Pool des Instituts für Physik zur Einarbeitung in die benötigten Werkzeuge für die Simulation und Synthese digitaler Schaltungen (Startknopf → alle Programme → Physik-Pool).

Verwenden Sie die reichlich im Web vorhandenen Informationen zur Vorbereitung. Generell empfiehlt es sich, englischsprachige Suchbegriffe zu verwenden, da Primärliteratur ausschließlich in englischer Sprache vorhanden ist. Einige wichtige (Such-) Begriffe sind kursiv gedruckt.

Verwendeter Baustein: Xilinx XC6S16LX

Aufgaben

Es steht Ihnen kommentierter Beispielcode für das Funktionsmodell eines digitalen time-to-digital converter (TDC) mit 8 bit Auflösung zur Verfügung. Er verfügt über je einen Start- und Stop-Eingang. Simulieren Sie das VHDL-Design unter Verwendung der Xilinx Werkzeuge (*behavioural simulation*). Synthetisieren und laden Sie das Design (ISE / Impact / Ladekabel). Überprüfen Sie die Funktion in der Schaltung. Steuern Sie den Takteingang mit dem Pulsgenerator bei einer Frequenz von etwa 10 Hz an. Steuern Sie Start und Stop per Tastendruck. Überprüfen und korrigieren Sie den Siebensegmentdecoder. Überprüfen Sie das Verhalten bei Meßbereichsüberschreitung. Modifizieren Sie den Beispielcode so, dass der TDC die Messung der Breite eines Doppelpulses erlaubt, wie es zum Beispiel zur Müon-Lebensdauermessung in einem Kerneinfang-Experiment benötigt würde: Erster Puls startet den TDC, ein eventuell folgender zweiter Puls stoppt den TDC und bereitet ihn für das nächste Ereignis vor. Bei Fehlen eines zweiten Pulses erfolgt das Zurücksetzen am Meßbereichsende. Die Anzeige des Meßwertes erfolgt über die Siebensegmentanzeige. Simulieren sie das Design. Überprüfen Sie die Funktion mit Hilfe von Doppelpulsen aus dem Pulsgenerators. Als Zeitbasis für den Wandler dient jetzt der Takt aus dem Quarzoszillator (27 MHz). Wie verhält sich der Wandler bei Doppelpulsen, deren Abstand den Meßbereich überschreitet? Wie kann sichergestellt werden, dass die führende Flanke (*leading edge*) die Zeitmessung bestimmt? Wie hoch ist dann der Fehler der Zeitmessung?

Kurzeinführung in VHDL

Die Sprache VHDL und ihre Verwendung werden hier nicht generell beschrieben. Aus dem gesamten VHDL-Sprachumfang werden lediglich einige wenige Konstrukte erläutert, die für die Versuchsdurchführung relevant sind. Diese Erläuterungen beziehen sich auf das Codebeispiel ….. und sollten unbedingt parallel zum Beispiel gelesen werden. Die verschiedenen Komponenten und Konstrukte werden hier erläutert in der Reihenfolge ihres Auftretens im Beispiel. Schlüsselworte sind teilweise im Dokument farbig hervorgehoben. Eine allgemeine Beschreibung der Sprache VHDL findet sich in vielen im Web verfügbaren Dokumenten. Suchbegriffe: *VHDL, reference, manual*.

VHDL ist eine Sprache zur textbasierten Beschreibung digitaler elektronischer Schaltungen. Ursprünglich wurde die Sprache zur Schaltungssimulation entwickelt, erst später auch für die *Synthese* und *Implementation* von Schaltungen verwendet. Synthese bezeichnet dabei die Umsetzung von der textuellen Ebene auf eine Netzliste von Makrofunktionen und Gattern, Implementation die weitere Umsetzung bis zum Konfigurationsdatenstrom für einen programmierbaren Logikbaustein (*CPLD*, *FPGA*). Die Historie der Sprache bedingt gelegentliche Inkompatibilitäten von Simulations- und Implementationsbeschreibung. Die strikte Beschränkung auf die hier beschriebenen Konstrukte und Regeln sollte solche Probleme jedoch ausschließen.

VHDL ähnelt nur teilweise einer Programmiersprache. Zwar kennt es auch Variablen, mit denen

man in annähernd gewohnter Weise umgehen kann. Unerlässlich ist jedoch die Verwendung von Signalen, die das Signalverhalten auf elektrischen Leitungen modellieren. Einem Signal kann generell nur an einer einzigen Stelle im VHDL-Code ein Wert zugewiesen werden. Dieser Wert kann nicht an einer anderen Stelle überschrieben werden, was aus der Interpretation als Leitung sofort ersichtlich ist: Die Zuweisung unterschiedlicher elektrischer Potentiale an zwei Punkten einer Leitung würde einen Kurzschluss darstellen und ist weder sinnvoll noch zulässig. Ein- und Ausgänge einer Schaltung sind immer elektrische Leitungen und können daher ausschließlich über Signale beschrieben werden, Variablen können nur zur Beschreibung interner Funktionen verwendet werden. Im Rahmen dieses Versuchs werden Variablen nicht benötigt. Im Folgenden wird daher nur die Verwendung von Signalen erläutert.

Ein- und Ausgänge einer Schaltung werden in einer *port list* aufgeführt mit Namen und Typ. Es ist zu beachten, dass es in VHDL *reserved words* gibt. Sie dürfen nicht als Signalnamen verwendet werden. Man sollte zudem niemals ein Signal mit demselben Namen belegen wie z.B. einen Typ oder ein *label*.

In der *port list* wird angegeben, ob die Datenrichtung **in** oder **out** ist. Bidirektionale Leitungen werden hier nicht verwendet. In einem Port, der nach außen geführt ist, sind nur zwei Signaltypen zulässig: **std_logic** ist ein einzelnes elektrisches Signal, also eine einzige Leitung, **std_logic_vector** beschreibt einen Datenbus, bestehend aus einzelnen std_logic Signalen: **datab(7 downto 0)** ist ein 8 Bit breiter Bus. Busse sollten immer in der beschriebenen Weise mit **downto** definiert werden.

Alle verwendeten Signale, die nicht in der port list enthalten sind müssen vor dem **begin** statement deklariert werden, unter Angabe von Namen und Typ. Zudem kann bei der *declaration* ein Initialwert angegeben werden, eine Konstante, abgetrennt durch **:=** . Ein **std_logic** Signal kann nur die Werte **'1'**, **'0'**, oder **'Z'** (in einfachen Anführungszeichen) annehmen, oder – in der Simulation – unbekannt sein. **'Z'** ist ein hochohmiger Leitungszustand, der generell nicht im Schaltungsinneren verwendet werden sollte. Ausgänge hingegen müssen gelegentlich auf hochohmigen Zustand geschaltet werden um das gewünschte Schaltungsverhalten zu erreichen. **std_logic_vector** Konstanten können entweder binär **"11101010"** oder hexadezimal **x"EA"** (in doppelten Anführungszeichen) geschrieben werden.

Initialisierungen sind generell nur für eine Simulation von Bedeutung. Bei der Beschreibung von programmierbaren Logikbausteinen empfiehlt sich eine Initialisierung auf **'0'**, da sie der Simulation genüge tut und im Allgemeinen bei der Implementation keine Probleme hervorruft.

Es können zahlreiche weitere Signaltypen genutzt und auch vom Nutzer selbst definiert werden. Der Typ **integer** beschreibt positive ganze Zahlen. Eine integer Konstante ist eine Dezimalzahl (ohne Anführungszeichen). Für die Verwendung von integer Variablen als Laufindex in einer **loop** ist keine vorheriger Deklaration erforderlich.

Nach dem **begin** Statement finden sich Signalzuweisungen, die *concurrent,* also parallel durchgeführt werden: es gibt keine Sequenz im Code, alle Zuweisungen finden gleichzeitig statt. Die Reihenfolge der Zeilen ist somit belanglos. Zuweisungen von Signalen erfolgen mittels **<=**. Zeilen werden mit Semikolon abgeschlossen. In diesem Versuch kann außer einer einfachen Zuweisung eine bedingte Zuweisung verwendet werden(**a<=b when k='1' else c;**), dieses Konstrukt entspricht einem Multiplexer mit zwei Eingängen b und c, dem Ausgang a und dem Schalteingang k. Der **else** -Zweig ist immer erforderlich. An logischen Operationen stehen z.B. **not**, **and** und **or** zur Verfügung. Auf Vektoren (Bussen) und integer Zahlen sind arithmetische Verknüpfung (+,-) sowie Vergleiche möglich (>, <, >=, <=, =, /=). Damit ist jedwede kombinatorische Logik beschreibbar. Da **std_logic_vector** zusammengesetzt ist aus einzelnen Bits des Typs **std_logic** können einige weitere Operationen auf Vektoren nützlich sein: Ein einzelnes Bit kann aus einem Bus extrahiert werden: **bitsix<=datab(6);**. Der in Klammern stehende Index ist vom Typ **integer**. Zwei Vektoren können zusammengefügt werden (*concatenation*):

widebus<=narrowA & narrowB;. Ebenso kann ein Vektor verlängert werden durch *concatenation* mit einem **std_logic**. Der Zielvektor muss natürlich zu gleicher Länge definiert sein wie die Summe seiner Komponenten.

Da alle im *concurrent* Code abgeleiteten Logikfunktionen gleichzeitig durchgeführt werden muss das zugrundeliegende Logikelement (zum Beispiel ein **or** Gatter) für jedes Vorkommen separat instantiiert (oder abgeleitet / *inferred*) werden. Eine *generate loop* erleichtert die mehrfache Verwendung gleicher Komponenten:

```
label_gen: for i in 0 to 7 generate
  datab(i)<=k;
end generate;
```

Dieses Konstrukt existiert nur in *concurrent* Code. Ein eindeutiges label, abgetrennt mit Doppelpunkt, ist erforderlich.

Im weiteren Verlauf des Beispielcodes ist ein Prozess definiert. Befehlszeilen in Prozessen werden sequentiell abgearbeitet, also in der Reihenfolge ihres Auftretens im Code. Dies ist für die Zuweisung von Variablen von Bedeutung (hier nicht verwendet). Es bedeutet jedoch keine zeitliche sondern eine logische Abfolge. Insofern ist dieser Begriff in VHDL anders belegt als in der diskreten Digitalelektronik. Dort beschreibt er eine zeitliche Sequenz, also zum Beispiel Schaltungen mit Flip-Flops. Allerdings verwenden wir im Rahmen dieses Praktikumsversuches generell Prozesse in Abhängigkeit eines zentralen Taktsignales und wir behelfen wir uns hier mit dem Begriff des synchronen Codes. In diesem Falle wird ein **process(clock)** immer begleitet von einem statement **if rising_edge(clock)**. Innerhalb dieses Konstrukts wird eine Signalzuweisung immer nur zum Zeitpunkt einer positiven Taktflanke auf dem signal **clock** durchgeführt. Der Logikzustand bleibt bis zum nachfolgenden Takt erhalten. Wir haben hiermit also ein Flip-Flop generiert. Ein Prozess kann von mehreren Signalen abhängig sein. In diesem Falle müssen sie alle als Argument (*sensitivity list*) angeführt werden. Die Vollständigkeit der sensitivity list ist für eine korrekte Simulation unabdingbar.

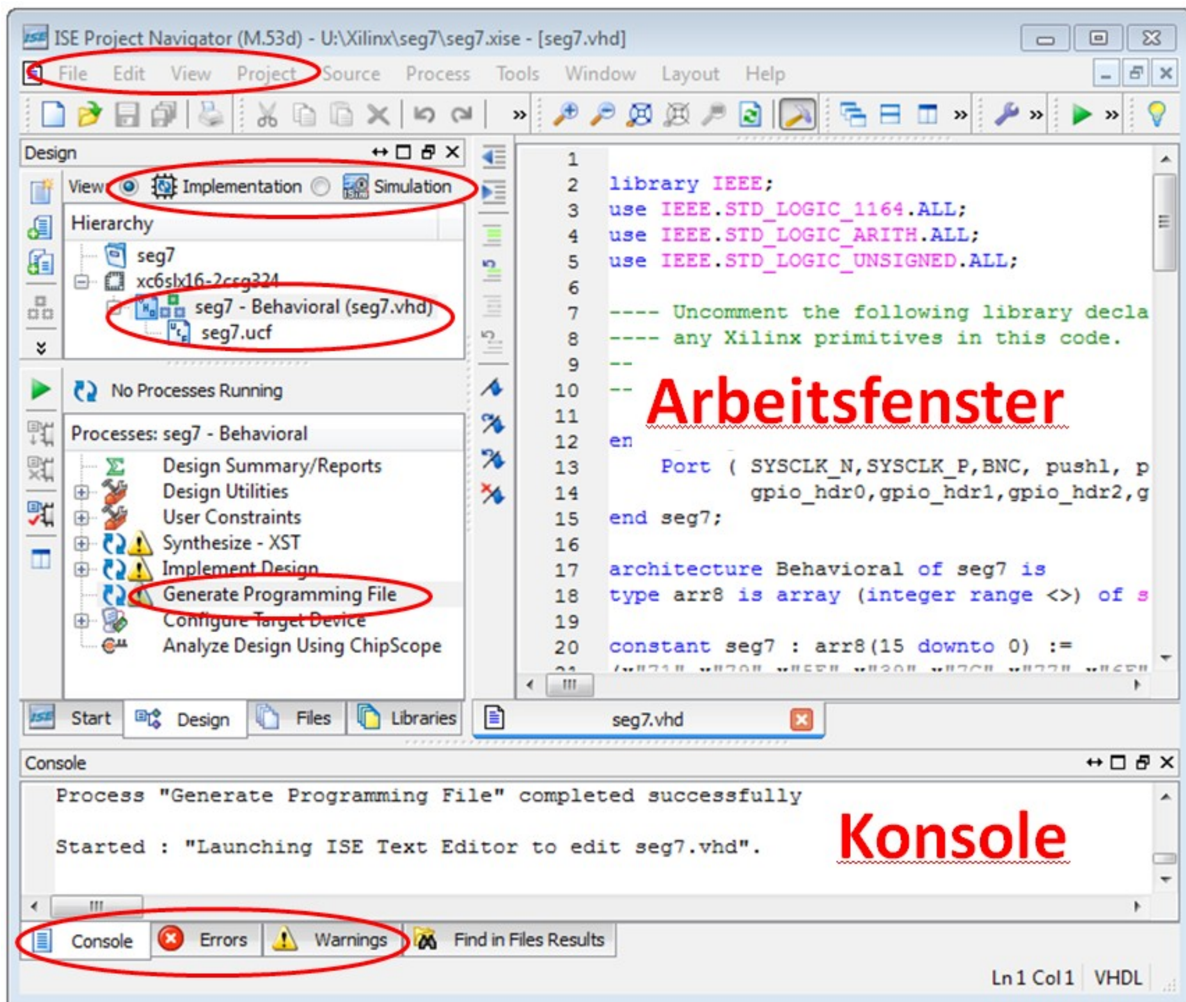
Bedingte Zuweisungen sind im Prozess möglich mit **if**… **then**… **else** … **end if**, Schleifen können mit *loop* Konstrukten gebildet werden (Beispiel: **for i in 0 to 7 loop datab(i)<=k; end loop;**). Es ist zu beachten, dass in *concurrent* Code und in Prozessen ähnliche Konstruktionen möglich sind, die Detailunterschiede sind dennoch erheblich und in jedem VHDL reference manual nachzulesen.

Die obenstehenden Konstrukte sind teilweise im Codebeispiel enthalten und mit Kommentaren (**--**) versehen. Die Struktur von VHDL Code im Allgemeinen wird nicht erläutert, bei Modifikationen sollte darauf geachtet werde, dass die vorgegebene Struktur nicht beschädigt wird und die Reihenfolge der Statements (bis auf die Zuweisungen im *concurrent* Code) nicht vertauscht wird.

Das Beispiel ist simulierbar, synthetisierbar und implementierbar mit Hilfe des Xilinx ISE Pakets.

Kurzanleitung Xilinx Design Software

Das Xilinx Design Paket steht zur Verfügung im PC-Pool.



Im "Project Navigator" finden sich auf der rechten Seite bzw. unten zwei Textbereiche, das Arbeitsfenster, in dem zum Beispiel der VHDL-Editor geöffnet wird, und darunter die Konsole, auf der Textinformationen zum design flow angezeigt werden wie Fehlermeldungen und Informationen. Auf der linken Seite sind 5 Bereiche farbig markiert: Oben finden sich Kontrollelemente zum Öffnen und Speichern von Dateien und Projekten. Die zweite Markierung zeigt den Bereich, in dem die Ansicht des Navigators umgeschaltet wird zwischen drei Betriebsarten, von denen hier nur zwei verwendet werden, Implementation und Behavioural Simulation. Die dritte markiert die Auswahl der zu bearbeitenden Quelle. Dies ist bei der Implementation eines Designs im Allgemeinen das VHDL-File. Einfach-Klick aktiviert die Quelle, Doppelklick öffnet sie im Editor. Die vierte Markierung zeigt im design flow auf die Funktion Implement design. Doppelklick führt diesen Schritt aus. Die fünfte Markierung zeigt auf Reiter, mit deren Hilfe man entweder die gesamte Konsol-Information auswählen kann, oder aber einen Filter aktivieren, der zum Beispiel nur Fehler zur Anzeige bringt.

Damit dürfte der Gang einer Implementation klar werden: gewünschtes Projekt laden (im Allgemeinen das Beispielprojekt), "Implementation" auswählen, "implement design" doppelklicken. Bei Fehlern öffnet sich der Texteditor durch doppelklicken der Fehlermeldung und man kann die Fehler korrigieren und nach Speicherung erneut implementieren. Bei fehlerfreiem code läuft die Implementation durch bis zur Erzeugung eines .bit Files im temporären Projektverzeichnis, der dann in den FPGA geladen werden kann.

Anmerkung: neben dem VHDL file wird für die Implementation noch ein *constraints file* benötigt. Es beschreibt die benutzten Pins des Bausteins und es teilt dem design tool eventuelle Erfordernisse an Durchlaufverzögerung und Taktgeschwindigkeit mit. Es ist in diesem Praktikumsversuch fest vorgegeben und sollte nicht bearbeitet werden.