georg.von-der-brueggen [☺] tu-dortmund.de
daniel.kuhse [☺] tu-dortmund.de

Hardwarepraktikum
Sommersemester 23

Übungsblatt 1 – 12 Punkte

(Block A – insgesamt 58 Punkte)

Bearbeiten ab Donnerstag, 6. April 2023.
Abgabe bis spätestens Freitag, 14. April 2023, 23:59 Uhr.

In Hardware Praktikum werden Sie digitale Schaltungen implementieren, die Sie in Rechnerstrukturen kennengelernt haben. Dazu zählen Gatter, Multiplexer, Kodierer und andere einfache Komponenten, sowie komplexere Schaltungen wie Addierer, Multiplizierer, Flip-Flops, Speicher und endliche Automaten. Dabei werden komplexeren Schaltungen Schritt für Schritt aus einfacheren Komponenten erstellt. Zum Abschluss des Praktikums werden Sie einen MIPS-Prozessor implementieren und erweitern, wobei wir Ihnen die wesentlichen Entwurfsschritte als Hilfestellung vorgeben werden. Nach dem Praktikum werden Sie in der Lage sein, einen eigenen Prozessor zu entwickeln.

Für die Implementierung aller Schaltungen werden wir die Hardwarebeschreibungssprache VHDL verwenden. VHDL benötigt ähnlich wie Programmiersprachen eine gewisse Einarbeitungszeit. Dabei geben wir uns Mühe, Sie soweit wie möglich zu unterstützen indem wir im Moodle ein Skript und im LS12 YouTube Kanal einen (englischen) VHDL Crash Course [5] zur Verfügung stellen. Außerdem stellen wir für einige Übungsblätter Beispiele mit Erläuterungen zur Verfügung an denen Sie sich orientieren können.

Die für die Bearbeitungen der Übungsblätter benötigten Dateien werden wir Ihnen in einem zip-file zur Verfügung stellen.

Bei Fragen oder Unklarheiten können Sie gerne im Forum ein Thema erstellen, sich an Ihre Übungsleitung wenden oder am Helpdesk teilnehmen.

1.1 VHDL und GHDL + GTKWave (5 Punkte)

VHDL (VHSIC (Very High Speed Integrated Circuit) Hardware Description Language) ist eine Hardwarebeschreibungssprache, mit der unabhängig vom Stand der Technik digitale Systeme auf unterschiedlichen Abstraktionsebenen modelliert und simuliert werden können. Man kann also mit VHDL Modelle von digitalen Schaltungen als Code aufstellen und sie dann ausführen, wobei der Code maschinen- und anwenderlesbar ist.

Um ein digitales System in VHDL zu entwerfen, spezifiziert ein Designer zuerst die benötigten Komponenten als Black Box, indem nur die In- und Outputs angegeben werden. Danach erfolgt eine Implementierung der Funktionalität und die Verbindung der einzelnen Komponenten. Im nächsten Schritt wird ein Simulationsprogramm genutzt, um das Modell auf korrekte Funktionalität zu prüfen. Nach ausgiebigen Tests kann dann der VHDL-code synthetisiert werden. Das heißt, echte Hardware kann direkt aus dem VHDL-Code erstellt werden. In der Hybrid-Variante des HaPras werden wir uns vorerst nicht mit der Synthese beschäftigen. Dafür bräuchte man FPGAs (Field Programmable Gate Array). Stattdessen werden wir mit Simulatoren arbeiten.

GHDL [1] ist ein Open Source Simulator für VHDL, mit dem VHDL Programme auf dem Heim- oder Arbeitsrechner kompiliert und ausführt werden können, ganz ohne zusätzliche Hardware. Damit Sie die Funktionalität der entworfenen Schaltungen selbst nachvollziehen und verifizieren können, wird noch Software benötigt, um die Signale zu plotten. Dafür nutzen wir in diesem Praktikum GTKWave.

GHDL und GTKWave laufen bei uns auf Windows, Linux-Distributionen und OS X ohne größere Schwierigkeiten. Trotzdem kann es vorkommen, dass vereinzelt Probleme bei der Installation auftreten. Daher empfehlen wir sich rechtzeitig mit der Installation zu beschäftigen, so dass potentielle Fragen im HelpDesk oder im Forum gestellt werden können. Falls es unter Windows oder OS X zu Problemen kommt hilft die Installation einer Linux Distribution (z.B. von Ubuntu), direkt oder in einer virtuellen Maschine (VM), oft das Problem zu umgehen.

Informationen über die Installation für verschiedenen Betriebssystem finden sich auf den entsprechenden Websites [2, 4]. Das erste Video im VHDL Crash Course [5] beschäftigt sich mit der Installation unter Windows. (Häufig können Probleme unter Windows – z.B. gcc not found oder file format not recognized – durch das Herunterladen der 32-Bit Version

ghdl-0.37-mingw32-mcode.zip behoben werden.) Unter Linux können GHDL und GTKWave mit folgendem Befehl installiert werden:

```
$ sudo apt-get install ghdl gtkwave
```

Um VHDL-Code mit GHDL zu kompilieren und mit GTKWave anschauen zu können, müssen im entsprechenden Verzeichnis folgende Befehle nacheinander ausgeführt werden ("#"kennzeichnet den Beginn eines Kommentars):

```
$ ghdl -s test_file.vhdl # Syntax-Check
$ ghdl -a test_file.vhdl # Analyse
$ ghdl -e test_file # Build
$ ghdl -r test_file --vcd=testbench.vcd # VCD-Dump
$ gtkwave testbench.vcd # Startet GTKWave
```

Aufgaben:

a. (5 Punkte) Um zu überprüfen ob die Installation funktioniert, führen Sie die fünf obigen Befehle mit der bereitgestellten Datei *test_file* aus und betrachten Sie die Wellenform in GTKWave. Schauen Sie danach in die Dokumentation von GHDL [3] hinein und suchen Sie nach Erläuterungen zu den erwähnten Befehlen zur Simulation von Systemen in VHDL. Schreiben Sie zu jedem der 5 Befehl eine kurze Zusammenfassung.

1.2 Python Skript (0 Punkt)

Für den weiteren Verlauf stellen wir ein Skript *vhdl_script.py* zur Verfügung, welches die Befehle aus der vorherigen Aufgabe für jede angegebene Datei ausführt und anschließend die letzte Datei in GTKWave öffnet. Die letzte Datei entspricht dabei immer der Testbench. Diese beinhaltet den Versuchsaufbau zum Testen der implementierten Schaltung(en).

Um das Skript auszuführen, muss folgender Befehl ausgeführt werden:

```
$ python3 vhdl_script.py gate1.vhdl gate2.vhdl ... gateN.vhdl gate_tb.vhdl
```

Dabei entsprechen $gate1, ..., gateN$ den benutzen Gattern und $gate\_tb$ der Testbench-Datei. Zusätzlich ist zu beachten, dass für die Reihenfolge der Gatter $i < j$ gelten muss, wenn Gatter i Teil von Gatter j ist.

Hinweis: Zur Verwendung des Skripts ist kein Wissen über Python als Programmiersprache nötig. Eine entsprechende Python Installation muss jedoch vorhanden sein. Bei Linux Distributionen ist Python oft bereits mitgeliefert. Details zur Installation von Python finden sich auf der Python Webseite [6].

1.3 Beispielaufgabe und Beispielabgabe (3 Punkte)

Im Laufe des HaPras werden wir kleinere VHDL Beispiele vorbereiten die Sie beim Verständnis unterstützen sollen. Dazu werden wir den entsprechenden VHDL Code kommentieren um neu eingeführte Ideen und Konzepte zu erläutern. Hier geschieht dies in Form einer Beispielabgabe, die außerdem unsere Abgabekonventionen motivieren soll und (am Beispiel) einige wichtige Funktionen von GTK Wave erläutert. Normalerweise werden die VHDL Beispiele jedoch nur aus kommentierten Quellcode Dateien bestehen.

Die Beispielabgabe ist für ein imaginäres Übungsblatt mit den folgenden Aufgaben:

1. Erstellen Sie ein *notgate* in VHDL und testen Sie mit Hilfe einer Testbench, ob Ihre Implementierung korrekt ist.
2. Erstellen Sie ein *andgate* in VHDL und testen Sie mit Hilfe einer Testbench, ob Ihre Implementierung korrekt ist.
3. Erstellen Sie ein *nandgate* in VHDL indem Sie die bereits erstellten Gatter verwenden und testen Sie mit Hilfe einer Testbench, ob Ihre Implementierung korrekt ist.

Laden Sie zunächste die Beispielabgabe aus dem Moodle herunter und entpacken Sie diese in ein neues Verzeichnis. Sie sehen 3 Unterverzeichnisse, eines für jeden der 3 Aufgabenteile mit aussagekräftigem Verzeichnisnamen. Im Verzeichnis für das notgate finden Sie 6 Dateien, von denen bei einer „echten“ Abgabe 3 selbsterstellt wären (*notgate.vhdl*, *notgate_tb.vhdl*, *command.txt*), eine vorgegeben wäre (*vhdl_script.py*) und 2 beim erfolgreichen Durchlaufen des Skriptes automatisch erzeugt werden (*testbench.vcd* und *work-obj93.cf*). Die Datei *notgate.vhdl* ist ausführlich kommentiert und gibt Hinweise auf die Funktion der einzelnen Code-Elemente. Die Datei *notgate_tb.vhdl* ermöglicht die Testung des notgate. Dazu wird zunächst die entsprechende Komponente initialisiert und mit sinnvollen Eingabekombinationen getestet. Starten Sie GTKWave und öffnen Sie die vorgegebene *testbench.vcd* Datei. Links im Fenster SST sehen sie *notgate_tb.vhdl*. Durch klicken sehen Sie darunter die Signale in der Testbench. Wenn Sie auf diese klicken werden die Signale im rechten Teil als Wellen dargestellt. Außerdem können Sie durch klicken auf das + bei *notgate_tb.vhdl* eine Liste der Komponenten öffnen (hier nur eine *not_g*). Durch klicken auf die Komponente können Sie die entsprechenden Signale der Komponente sehen (inklusive der nur intern verwendeten Signale). Durch das Auswählen von *Time* und dann *Zoom* und *Zoom Full* wird die Zeitachse der Darstellung so skaliert, dass die volle Zeitspanne der Evaluation zu sehen ist.

a. (1 Punkt) Welche Signale in der Komponente *not_g* haben immer den gleichen Wert wie das Signal *c*?

Bisher haben wir für die Analyse von *notgate.vhdl* nur *notgate_tb.vhdl* anhand der mit abgegebene Datei *testbench.vcd* betrachtet. Diese mit abzugeben ist zum Beispiel hilfreich um Teilpunkte zu bekommen, wenn nur Teile des Codes funktionieren aber der Code am Ende nicht mehr kompilierbar ist weil sich im letzten Schritt der Bearbeitung noch Fehler eingeschlichen haben. Um die volle Punktzahl zu erreichen genügt dies aber nicht, sondern alle Dateien müssen kompilierbar sein und das richtige Ergebnis liefern.

Die Datei *command.txt* gibt den Befehl für das *vhdl_script.py* an, mit dem *notgate.vhdl* und *notgate_tb.vhdl* nacheinander ausgeführt werden und das Ergebnis dargestellt werden kann. Solch eine Datei mit dem zugehörigen Kompilierbefehlen muss bei jedem Aufgabenteil bei dem einen Implementierung nötig ist mit abgegeben werden um das korrekte Kompilieren aller Dateien zu gewährleisten. Um zu motivieren warum dies sinnvoll ist, löschen Sie die Datei *work-obj93.cf* und führen Sie den folgenden verkürzten Kompilierbefehl aus:

```
$ python3 vhdl_script.py notgate_tb.vhdl
```

b. (1 Punkt) Wie ändert sich das Signal für *c*?

c. (1 Punkt) Was ist der Grund für diese Änderung?

In diesem simplen Beispiel sind solche Fehler natürlich leicht zu korrigieren. Im Laufe des HaPras wird es aber Situationen geben wo 10 oder mehr Dateien kompiliert werden und es nicht offensichtlich ist welche Datei welche andere als Voraussetzung hat. Auch die ausgegebene Warnung kann gewollt oder zumindest unproblematisch sein. Der entsprechenden Kompilierbefehl stellt also sicher, dass das Richtige bewertet wird. Das für jede Teilaufgabe ein eigenes Verzeichnis verwendet wird in dem sich alle benötigten Dateien befinden stellt außerdem sicher, dass stets die richtige Version der Komponente verwendet wird.

In den Dateien *nandgate.vhdl* und *nandgate_tb.vhdl* im entsprechenden Unterverzeichnis finden Sie noch weitere Hinweise, die für den Einstieg in VHDL hilfreich sein können.

1.4 Erste Schritte in VHDL (4 Punkte)

Grundwissen über die Modellierung von Schaltungen in VHDL wird im weiteren Verlauf des Praktikums nötig sein.

a. (0 Punkte) Arbeiten Sie im VHDL-Skript das erste Kapitel durch.

b. (4 Punkte) Fertigen Sie eine Skizze der Schaltung an, die im unten stehenden VHDL-Code implementiert ist. Verwendet Sie die IEC 60617-12 Norm für die Zeichnung von Gattern. Beschriften Sie Ihre Skizze mit den Bezeichnungen a-g und geben Sie die implementierte boolesche Funktion an.

```
entity unknown is
  port(
    a : in std_logic;
    b : in std_logic;
    c : in std_logic;
    d : in std_logic;
    e : out std_logic
  );
end unknown;

architecture behavior of unknown is
  signal f : std_logic;
  signal g : std_logic;
begin
  f <= a or b;
  g <= c or d;
  e <= f and g;
end unknown;
```

Hinweis: Halten Sie sich bei der Abgabe im Moodle bitte an die folgenden Konventionen:
(1) Geben Sie den schriftlichen Teil der Abgaben bitte in einer zusammenhängenden pdf-Datei ab.
(2) Verwenden Sie zur Abgabe von Quellcode Containerdateien (zip). Am besten eine Datei mit mehreren Unterverzeichnissen. Verwendet dabei die Bezeichnung *Gruppe"Nummer"_Blatt"Nummer".zip* (wahlweise mit Namenszusatz), z.B. für Gruppe 153, Blatt 2: *Gruppe153_Blatt2.zip* oder *Gruppe153_Blatt2.zip_(Ada_Lovelace_und_Konrad_Zuse).zip*
(3) Quellcode-Dateien müssen **kommentiert** und **kompilierbar** sein. Bitte kommentieren Sie den Code angemessen und ausreichend. Anderenfalls wird diese (Teil-) Aufgabe unter Umständen nicht bewertet.
(4) Wenn bei einer Aufgabe eine Quellcode-Datei (vhdl-Dateien, Scripte, etc) erstellt oder verändert werden soll, geben Sie bitte die entsprechende Datei zusammen mit allen anderen für die Ausführung benötigten Dateien ab (selbst wenn diese Dateien vorgeben sind und nicht verändert wurden oder in einer anderen (Teil-) Aufgabe erstellt wurden). Die abgegebene zip-Datei sollte außerdem das vorgegebene Python Skript sowie eine Datei *command.txt* enthalten die den Befehl angibt mit dem die Abgabe mit Hilfe des vorgegebenen Skriptes ausgeführt werden kann. Wenn z.B. eine Test-Datei für eine vorgegebene Datei geschrieben werden soll, geben Sie also bitte sowohl die Testdatei als auch die vorgegebene Datei ab sowie das Skript und die Datei *command.txt* für die entsprechende Aufgabe. VCD-Dateien und die *work_obj93.cf* Datei sollten ebenfalls mit abgegeben werden. Eine Beispielabgabe finden Sie bei den Dateien die für Übungsblatt 1 zur Verfügung gestellt werden.

Literatur

[1] GHDL GitHub Repository.
https://github.com/ghdl/ghdl

[2] GHDL Installation Instructions.
http://ghdl.free.fr/site/pmwiki.php?n=Main.Installation

[3] GHDL User Guide.
http://ghdl.free.fr/site/pmwiki.php?n=Main.UserGuide

[4] GTKWave Website.
http://gtkwave.sourceforge.net/

[5] VHDL Crash Course.
https://www.youtube.com/playlist?list=PL7E8QcsvNJ_NzcbZi_-idur-tEh8j5aBQ

[6] Python Website.
https://www.python.org/