Coding Guidelines

In dem vorliegenden Dokument „Coding Guidelines“ finden sich Richtlinien, Hinweise und Empfehlungen zur Projektorganisation und zur Gestaltung von Schaltungsentwürfen mit VHDL. Viele dieser Hinweise sind nicht auf die Hardwarebeschreibungssprache VHDL beschränkt. Die meisten Hinweise sind generell für formale Sprachen (wie C, C++, Phyton, Matlab, Fortran, Java) anwendbar.

Die Hinweise gliedern sich in die Projektorganisation, die Gestaltung des Sourcecodes und Regeln, welche speziell die Synthese betreffen. Die Sinnhaftigkeit bzw. Notwendigkeit derartiger Regeln erschließt sich leider erst ab einer gewissen Größe der zu bearbeitenden Aufgabe (im Labor ab Versuch 4). Es ist aber von Vorteil, die gegebenen Hinweise schon bei kleineren Aufgaben zu verwenden.

Das Erstellen und Einhalten von Coding Guidelines erfolgt zumeist für ein Projekt, eine Abteilung oder auch eine ganze Firma.

1. Projektorganisation

Die zu entwickelnde Schaltung sollte in mehrere Module aufgeteilt werden. Für jedes Modul sollte ein eigenes Directory angelegt werden. Eine typische Untergrenze für ein eigenes Modul ist im Versuch 4 und im Versuch 6 zu sehen. Das Directory sollte den Modulnamen tragen. Folgende Subdirectories sind empfehlenswert:

- src: enthält die VHDL Modelle (entity und architecture)
- cores: enthält alle Dateien, die vom Xilinx core generator erzeugt werden
- testbench: enthält die testbench (es), mit denen das Modul oder die Untermodule getestet werden
- testdata: Directory, welches die Referenzdaten (Eingangswerte und korrekte Ausgabedaten) für den Test enthält
- work: die work library

2. Gestaltung des Sourcecodes

Für die Gestaltung eines Sourcecodes sind die Ziele:

- maximale Lesbarkeit
- gute Wartbarkeit
- gute Erweiterungsmöglichkeit

Diese Ziele gelten auch schon im Laborbetrieb. Mehrere Studenten arbeiten am gleichen Sourcecode. Die Verständigung innerhalb der Gruppe muss gegeben sein. Vorhandene Lösungen (VHDL Sourcecodes) müssen erweitert werden.

2.1 Namensgebung

In einem VHDL Code benötigen viele Konstrukte einen Namen (identifier), beispielsweise Signale, Variablen und Konstanten sowie Entities, Funktionen und Prozeduren. Weiterhin sind Namen für Instanziierung erforderlich. Auch für Prozesse und Schleifen ist es möglich Namen bzw. Labels zu vergeben.

Der wichtigste Hinweis ist, für die genannten Objekte und Strukturen **aussagekräftige** Namen zu verwenden. Dies führt im Allgemeinen zu etwas längeren Namen (>= 3 Zeichen) und erfordert etwas mehr Schreibarbeit, erhöht aber die Lesbarkeit sehr. Beispielsweise für ein Signal, welches eine Adresse für ein RAM oder ROM darstellt ist es besser einen identifier wie „addr „oder „address“ zu verwenden statt „a“.

Für extrem häufig verwendete „Standardsignale“ wie Takt und Rücksetzsignal sollten stets die gleichen Namen verwendet werden. Konkreter Vorschlag ist die Verwendung von „clk“ für den Takt und „rst“ für das Rücksetzsignal.

Die Möglichkeit Labels für Prozesse zu verwenden sollte mit aussagekräftigen Namen genutzt werden. Diese Labels verwendet auch der Simulator. Ohne Labels werden Zeilennummern verwendet die bei längeren Codes mit mehreren Prozessen schwer zuzuordnen sind und sich mit Erweiterungen am Code auch noch ständig ändern.

VHDL unterscheidet nicht zwischen Klein- und Großschreibung. Es gibt somit beispielsweise keinen Unterschied zwischen den Bezeichnern „min_wert“ und „Min_WERT“. Daher sollten (fast) alle identifier sowie die VHDL-keywords grundsätzlich kleingeschrieben werden.

Eine Ausnahme bilden Konstanten und generische Konstanten (**constant** bzw. **generic**). Hier sollte für die Bezeichner die Großschreibung verwendet werden. Die Hinweise bzgl. Groß-/Kleinschreibung erhöhen die Lesbarkeit und das Verständnis des Codes.

Bei Signalen, die an Ports von Komponenten angeschlossen werden, ist zu prüfen ob nicht der gleiche oder ein ähnlicher Bezeichner verwendet werden kann. Ziel ist es, die Signalnamen über die Hierarchiegrenzen hinweg zu erhalten. Auch dieser Hinweis dient wieder der erhöhten Lesbarkeit und somit Verständlichkeit des Codes.

2.2 Kommentare

Ein wichtiges Mittel zur Verbesserung des Verständnisses bei Sourcecodes ist die Verwendung von Kommentaren. Folgende Hinweise gelten:

- Ein paar generelle Kommentarzeilen sollten den vorliegenden VHDL Code beschreiben
  - Zweck des Moduls
  - Warum wird es benötigt
  - Wie funktioniert es
  - Was sind die Annahmen an die Umgebung
- Verfasser, Datum und Revisionstand sollten vermerkt sein.
- Alle Signal der Portliste der entity sollten mit einem Kommentar versehen sein.
- Alle im VHDL Code deklarierten Signale, Variablen und Konstanten sollten kommentiert werden. Dies gilt ebenso für die Deklaration von Komponenten. Gegebenenfalls genügt es, eine zusammengehörige Gruppe von Signalen (z.B. alle Eingangssignale für ein RAM) mit einem gemeinsamen Kommentar zu versehen.
  Übrigens: Die Verwendung aussagekräftiger Bezeichner reduziert den Bedarf an Kommentaren.
- Je nach Größe sollte auch ein process mit einem Kommentar versehen sein, der den Zweck des vorliegenden Prozesses beschreibt.
- Das „Auskommentieren“ von Deklarationen und Statements sollte unterlassen werden, da es beim Lesen nur Verwirrung stiftet. Derartige Zeilen sollten gelöscht werden, es sei denn, Sie wollen diese Zeilen noch mit einem Betreuer diskutieren

Die Sinnhaftigkeit dieser Hinweise erschließt sich leider erst ab einer gewissen Codegröße (d.h. ab Versuch 5). Im Labor können Sie bei den ersten 4 Versuchen gerne auf entsprechende Kommentare verzichten.

2.3 Layout des Codes

Das Layout eines Codes in formaler Sprache ist für die Lesbarkeit des Codes von entscheidender Bedeutung. Folgende Hinweise gelten:

- Eine separate Zeile für jede Deklaration eines Typs, eines Signals und einer Variablen
- Pro Zeile nur ein Statement
- Für den Kontrollfluß (if-Konstrukt, case-Konstrukt) sowie jedes „begin“ verbessern Einrückungen die Lesbarkeit. Es werden 3 Leerzeichen empfohlen.

- Keine Tabulatoren für Einrückungen verwenden. Bei Wechsel des Editors oder beim Ausdrucken funktioniert in der Regel etwas mit den Einrückungen per Tabulator nicht.
- Die Zeilenlänge (Anzahl der Spalten) sollte nicht zu groß sein; Empfehlung: max. 132 Spalten.
- Bei Instanziierungen sollte die "named association" verwendet werden (bessere Verständlichkeit, geringere Fehleranfälligkeit). Für jede Assoziierung (a => b,) sollte eine separate Zeile verwendet werden.

2.4 Einige allgemeine Codierhinweise

Die folgenden Hinweise sind ebenfalls hilfreich:

- Statt fest codierter Werte (als Konstante) bietet es sich an, das **constant**-Konstrukt zu verwenden. Also statt

  if addr = 10 **then** ..

  sollte eine Konstante z.B. LAST_ADDR mit dem Wert 10 definiert werden. Das Codefragment lautet dann

  if addr = LAST_ADDR **then** ..

  Damit ist der Code besser änderbar und erweiterbar, insbesondere wenn diese Konstante öfters verwendet wird.

- Multi-bit Objekte (std_logic_vector, signed, unsigned) sollten in der Regel die Richtung **downto** haben und als rechten Index die 0 verwenden (siehe Skript).
- Keinen Bereich eines Signals (einer Variablen) spezifizieren, wenn der gesamte Vektor angesprochen wird. Die Angabe eines Bereiches führt stets zu der Annahme, dass das Objekt noch mehr Elemente als angegeben enthält.
- Bei größeren Projekten (größer als die Versuche im Labor) sollten Parameterwerte und Funktionen in einem (oder mehreren) separaten package(s) gehalten werden.
- Sensitivitylisten sollten nicht mehr Signale enthalten als für die Funktionalität erforderlich.
- Variablen sollten nur für (kombinatorische) Zwischenwerte und nicht für gespeicherte Werte verwendet werden.

3. Syntheserelevante Hinweise

Die folgenden Hinweise erleichtern das Erstellen von synthetisierbarem Code und verbessern die Übereinstimmung zwischen synthetisierter Netzliste und VHDL Code.

- Nur Verwendung der packages std_logic_1164 und numeric_std der IEEE library (siehe Skript); Grund: Portierbarkeit
- Keine Initialisierungswerte für Signale und Variablen verwenden, Grund: Übereinstimmung von Code und synthetisierter Netzliste

- Bei getakteten Prozessen die Templates des Skripts verwenden; Grund: Lesbarkeit, Gewährleistung der Synthetisierbarkeit
- Template für kombinatorische Prozesse des Skriptes einhalten; Grund: Vermeidung von Latches und Übereinstimmung von Code und synthetisierter Netzliste
- Bei Verwendung von geschachtelten if-Konstrukten sollte die alternative Verwendung eines Case- Konstruktes geprüft werden (siehe Skript); Grund: Verzögerungszeiten
- Keine intern erzeugten Resets
- Keine gated Clocks
- Keine gemischten Flanken

Die letzten drei Hinweise sind generelle Hinweise (siehe auch Skript) für die Erstellung von Hardware. Natürlich werden bei großen Projekten auch ggf. mehrere Takte und intern erzeugte Resets notwendig sein. Dies sollte aber in separaten Modulen erfolgen und mit allergrößter Vorsicht und Sorgfalt implementiert werden.