Technische Universität Graz
Fakultät für Informatik und Biomedizinische Technik

Institut für Angewandte Informationsverarbeitung
und Kommunikationstechnologie

Rechnerorganisation Konstruktionsübungen 2016

LV Nr. 705.037
Version vom 20. Mai 2016
Karl C. Posch

Latitude: *55.382469* | ***Longitude:*** *14.054739*

Inhaltsverzeichnis

Einleitung

Dieses Dokument dient als Basis für die Aufgaben, welche du im Rahmen der Lehrveranstaltung „KU Rechnerorganisation“ im Sommersemester 2016 machen sollst. Die Aufgaben beziehen sich auf die in der Vorlesung Rechnerorganisation besprochenen Themen. Ich rate also dringend an, Vorlesung und Konstruktionsübungen als ein zusammen gehörendes Ganzes zu betrachten. Wenn du die Themen der Vorlesung wirklich verstehen willst, dann musst du die Übungen machen. Und wenn du die Übungen erfolgreich machen möchtest, dann musst du dich mit dem Vorlesungsstoff auseinander setzen. Erst durch diese gemeinsame Sichtweise bekommst du ein vollständiges Bild vom Thema Rechnerorganisation.

Ich wünsche dir viel Spaß beim Arbeiten.

Leistungsdokumentation

Du bist auf dem Weg, eine Ingenieurin oder ein Ingenieur werden zu wollen. Ein professioneller Zugang zu Ingenieursarbeit beinhaltet auch die Dokumentation der Arbeit. Dazu gibt es folgende Werkzeuge:

Das Ingenieurstagebuch: In einem Ingenieurstagebuch dokumentierst du für dich selbst deine Arbeit am Projekt. Deine Ideen, deine Versuche, Skizzen, Termine, Treffen mit Gruppenmitgliedern und vieles mehr. Du dokumentierst dabei vor allem auch, wann du wie viele Stunden gearbeitet hast. Du möchtest am Ende ja schließlich wissen, wie viel Zeit du für die Lehrveranstaltung aufgewendet hast. Ich werde dich, nachdem du die Note bekommen hast, ebenfalls danach fragen. Denn ich möchte auch wissen, wie viel Zeit Studierende für die Lehrveranstaltung aufwenden. Erst dadurch kann ich meine anfängliche Schätzung mit wirklichen Zeiten vergleichen und daraus lernen. Der geplante Aufwand für Vorlesung und Übungen zusammen für den durchschnittlichen Studierenden ist mit 4,5 ECTS-Punkten vorgegeben. Dies entspricht einem durchschnittlichen Zeitaufwand von etwa 120 Arbeitsstunden. Ob diese Planung stimmt, kann ich nur mit Hilfe der Zeitaufzeichnungen einer größeren Menge von Studierenden im Nachhinein ermitteln. Und mit diesem Ergebnis kann ich das Übungsausmaß für das nachfolgende Jahr kalibrieren.

Die Deliverables: Das sind alle Dokumente, welche du dem Auftraggeber, in unserem Fall hier also dem Lehrer und dem/der Tutorin rechtzeitig liefern (= deliver) musst.

Die Präsentation: Du wirst im Rahmen von zwei Abgabegesprächen Gelegenheit haben, deine Arbeit zu präsentieren und zu verteidigen. Damit kannst du dokumentieren, mit welcher Qualität du die Arbeit gemacht hast.

Termine: Apropos „rechtzeitig liefern“. Ein wesentliches Element von professioneller Qualität einer Arbeit besteht schließlich im Einhalten von Terminen. Ich dränge dich also als Auftraggeber dazu, die Abgabefristen einzuhalten. Der jeweils angegebene Abgabetermin ist der spätestens mögliche. Du darfst auch früher abgeben. Ein guter Rat besteht darin, dass du dir jeweils einen eigenen internen Termin setzt, welcher ein gutes Stück vor dem von mir vorgegebenen Termin liegt. Damit vermeidest du am ehesten extern verursachten Stress. Du hast für alle Aufgaben zusammen 3 mal 24 Stunden Überziehungsfrist, welche sich nicht auf die Punkte und somit auf die Note auswirken. Spare dir diese Überziehungsfrist so lange es geht auf. Du solltest diese Reserve nur in unvorhergesehenen Notfällen „anzapfen“. Jede darüber hinaus gehende Überziehung schlägt sich mit 10 Punkten Abzug (von den erreichten Gesamtpunkten für die gesamte Übung) pro 24 Stunden nieder. Sollten

Gruppenmitglieder ihr Überziehungskonto bei den ersten Abgaben verschieden stark belastet haben, so zählt für die Gruppe der ungünstigste Einzelfall. Mehr zum Thema „Zeitplanung“ gibt es auf Seite 21 dieses Dokuments unter dem Titel „Huch, es geht sich mit der Zeit nicht aus“.

Anmeldefrist zur Übung

Melde dich bis spätestens 11. März 2016 an einer der 5 Übungsgruppen im TUGRAZonline an. Solltest du zu diesem Zeitpunkt noch keinen TUGRAZonline-Zugang besitzen, weil du soeben erst im 1. Semester inskribiert hast, dann schicke vor dem 11. März 2016 eine Mail an Karl.Posch@iaik.tugraz.at.

Wenn du nicht angemeldet bist, dann kannst du keine positive Note bekommen. Eine Note bekommst du, sobald du zumindest eine Abgabe gemacht hast.

Teilnahmepflicht am Tutorium 0

Die Teilnahme am Tutorium 0 ist verpflichtend. Wenn du dieses versäumst, dann kannst du keine Note kriegen. Das Tutorium 0 findet je nach Übungsgruppe an folgenden Terminen statt:

- 8. März 2016, 8:00-9:00 Gruppe Hadzic, IAIK-Seminarraum
- 7. März 2016, 9:00-10:00 Gruppe Hölbling, IAIK-Seminarraum
- 9. März 2016, 9:00-10:00 Gruppe Schwarzl, IAIK-Seminarraum
- 4. März 2016, 9:00-10:00 Gruppe Ulbel, IAIK-Seminarraum
- 7. März 2016, 8:00-9:00 Gruppe Weinrauch, IAIK-Seminarraum

Der IAIK-Seminarraum befindet sich in der Inffeldgasse 16a im Erdgeschoß.

Abgabetermine

Aufgabe 0:	Montag,	21. März 2016, 14:00
Aufgabe 1:	Montag,	21. März 2016, 14:00
Aufgabe 2:	Montag,	2. Mai 2016, 14:00
Gruppenmeldung:	Montag,	23. Mai 2016, 14:00
Aufgabe 3:	Montag,	23. Mai 2016, 14:00
Aufgabe 4:	Freitag,	10. Juni 2016, 14:00

Abgabe der Deliverables über das „Student Tick System“ (STicS) unter https://stics.iaik.tugraz.at. Vergiss nicht, dich rechtzeitig vor der ersten Abgabe bei diesem System zu registrieren. Die Möglichkeit zur Registrierung besteht ab dem 14. März 2016. Verwende bei dieser Registrierung als E-Mail deine TU-Graz-E-Mail-Adresse. In besonderen Fällen darfst du deine Abgabe auch per E-Mail an ro-abgabesystem@iaik.tugraz.at machen.

Gruppenbildung

Die vierte Aufgabe dieser Übung machst du in einer 3-er-Gruppe. Suche dir im Laufe der ersten Wochen des Semesters drei GruppenpartnerInnen. Die Gruppenmeldung erfolgt über das „Student Tick System“ (STicS) unter https://stics.iaik.tugraz.at. Termin für diese Meldung ist der 23. Mai 2016, 14:00.

Aufgabe 0:

Vorstellung bei der/dem TutorIn; Auseinandersetzung mit den Prinzipien guter wissenschaftlicher Praxis

Besuche das Tutorium 0. Lege fest, ob du das Tutorium weiterhin besuchen willst/kannst oder nicht und teile dies dem/der TutorIn mit. Im Falle der Entscheidung, das Tutorium zu besuchen, machst du deine beiden Abgabegespräche mit dem/der TutorIn. Ansonsten mit dem Lehrer selbst.

Verstehe die Begriffe „Plagiat“ und „Täuschungsversuch“.

Termin des Tutoriums 0

8. März 2016, 8:00-9:00	Gruppe Hadzic,	IAIK-Seminarraum
7. März 2016, 9:00-10:00	Gruppe Hölbling,	IAIK-Seminarraum
9. März 2016, 9:00-10:00	Gruppe Schwarzl,	IAIK-Seminarraum
4. März 2016, 9:00-10:00	Gruppe Ulbel,	IAIK-Seminarraum
7. März 2016, 8:00-9:00	Gruppe Weinrauch,	IAIK-Seminarraum

Spätester Abgabetermin

- Montag, 21. März 2016, 14:00.
- Abgabe über das „Student Tick System“ (STicS) unter https://stics.iaik.tugraz.at. Vergiss nicht, dich rechtzeitig vor der ersten Abgabe bei diesem System zu registrieren. Die Möglichkeit zur Registrierung beim „Student Tick System“ besteht ab dem 14. März 2016. Verwende bei dieser Registrierung als E-Mail deine TU-Graz-E-Mail-Adresse. In besonderen Fällen darfst du deine Abgabe auch per E-Mail mit dem Betreff „[RO] Abgabe 1“ an ro-abgabesystem@iaik.tugraz.at machen.

Vorgangsweise

- Setze dich mit den Begriffen „Plagiat“ und „Täuschungsversuch“ auseinander. Studiere dazu die Materialien, welche zum Beispiel unter http://www.plagiarism.org zu finden sind.
- Studiere das Dokument “Richtlinie zur Sicherung guter wissenschaftlicher Praxis” https://tu4u.tugraz.at/fileadmin/public/Studierende_und_Bedienstete/Richtlinien_und_Verordnungen_der_TU_Graz/Gute_wissenschaftliche_Praxis_Richtlinie.pdf
- Schreibe einen Text, in welchem du erklärst, dass du (1) die in den obigen Punkten beschriebene Arbeit gemacht hast, dass du (2) verstanden hast, was ein Plagiat bzw. ein Täuschungsversuch ist,

dass du (3) die Konsequenzen eines Fehlverhaltens in dieser Lehrveranstaltung verstanden hast, und dass du (4) kein Plagiat abgeben wirst. Verwende in diesem Text Sätze mit dem Wort „ich“ als Subjekt. Setze unter deinen Text deinen Namen. Das Dateiformat dieses Textes ist PDF. Der Name der PDF-Datei entspricht deiner Matrikelnummer gefolgt von der Dateierweiterung „.pdf“;

Wozu Aufgabe 0?

Jede/r angemeldete StudentIn sollte explizit die Entscheidung treffen, ob sie/er die Übung innerhalb einer Übungsgruppe mit Betreuung eines/einer TutorIn machen möchte oder ob dies nicht möglich ist. Ein typischer Grund für eine Nicht-Teilnahme wäre eine berufsbedingte Verhinderung.

Die Entscheidung zur Teilnahme sollte zu Beginn der Übung getroffen werden und muss dem Lehrer und der/dem TutorIn mitgeteilt werden. Damit ist es für den Lehrer und für die TutorInnen möglich, sich bestmöglich auf die Betreuung derjenigen Studierenden zu konzentrieren, welche am Tutorium teilnehmen.

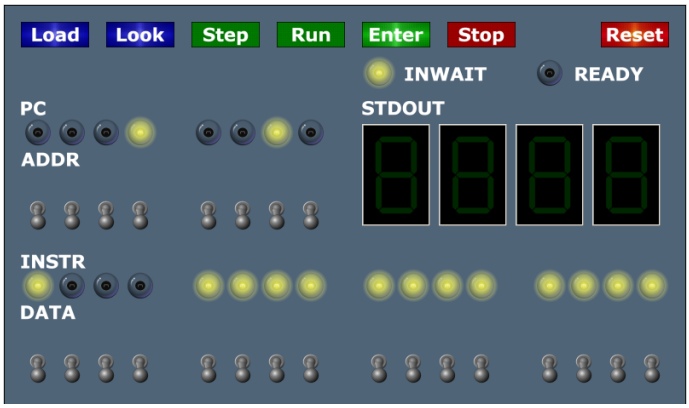
Ein lediglich sporadischer Besuch der Tutorien ist unerwünscht.

Solltest du – in seltenen Fällen – trotz Teilnahmeentscheidung an einem Tutorium nicht teilnehmen können, dann teilst du dies dem/der TutorIn vorher per Mail mit. Solltest du dich vom Tutorium abmelden wollen, dann kannst du dies ebenfalls per einfacher Mail an den/die TutorIn sowie an den Lehrer (Karl.Posch@iaik.tugraz.at) machen.

Der wesentliche Grund für diese Vorgangsweise liegt im Ansinnen, Studierende zu einem professionellen Verhalten zu drängen. Professionalität drückt sich dabei folgendermaßen aus: Du lernst, Entscheidungen über deine nähere Zukunft zu treffen und dich dann an diese Entscheidungen zu halten. Du lernst zudem, im Falle von Verhinderung im Vorhinein geeignete Aktionen zu setzen.

Durch die explizite Beschäftigung mit dem Themen „Plagiat“ und „Täuschungsversuch“ lernst du die Grundregeln der wissenschaftlichen Praxis kennen.

Aufgabe 1:



Assemblerprogramm mit Visual X-TOY

(Einzelarbeit)

Entwickle ein Assemblerprogramm für TOY, welches Werte vom Standard-Input einliest und jeden dieser Werte im Hauptspeicher in einer Liste bestehend aus den bereits eingelesenen Werten der Größe nach „einsortiert“. Sobald der Wert 0 vom Standard-Input gelesen wird, beendet das Programm seine Arbeit. Wenn der Wert „minus 1“ (0xFFFF) gelesen wird, soll das Programm eine Null (0x0000) gefolgt von den bisher eingegebenen Werten in aufsteigender Reihenfolge sortiert am Standard-Output ausgeben. Die zu sortierenden Werte kommen aus dem Intervall (0x0001, 0x7FFF). Assembliere das TOY-Assemblerprogramm und teste das resultierende TOY-Maschinenprogramm mit Visual X-TOY.

Beginne deine Überlegungen zum Sortieren indem du zunächst einen C-Code entwickelst. Ausgehend von diesem C-Quellcode sollst du iterativ funktionsidente Veränderungen an diesem C-Code vornehmen. Du beginnst mit dem Ersetzen von Schleifen durch Goto-Anweisungen; sodann sollst du die Array-Zugriffe so modifizieren, dass die in der TOY-Assemblersprache vorhandenen Möglichkeiten im C-Code direkt ersichtlich werden. Das Resultat dieser Modifikationen soll ein funktionierender C-Code sein, der Zeile für Zeile in TOY-Assemblerbefehle übersetzt werden kann.

Beispiel:

Eingabe:

0005
0006
FFFF
0007
0003
FFFF
0004
0002
0001
FFFF
0000

Die dazu richtige Ausgabe:

0000
0005
0006
0000
0003
0005
0006
0007
0000
0001
0002
0003
0004
0005
0006
0007

Restriktionen: Die maximale Anzahl der zu sortierenden Werte ist 20.

Abgabe

Abzugeben sind folgende Dateien, welche zusammen in eine ZIP-Datei gepackt werden. Der Name der ZIP-Datei entspricht deiner Matrikelnummer gefolgt von der Dateierweiterung „.zip“; also zum Beispiel so: „1234567.zip“. Folgende Dateien sollen in der ZIP-Datei enthalten sein:

- Der C-Code:
  Dateiname ist Matrikelnummer-dot-c; also z.B. „1234567_1.c“
- Der modifizierte, funktionsidente C-Code (Schleifen in Goto-Anweisungen umgeformt):
  Dateiname ist Matrikelnummer-dot-c; also z.B. „1234567_2.c“
- Der modifizierte, funktionsidente C-Code (Array-Zugriffe aufgelöst):
  Dateiname ist Matrikelnummer-dot-c; also z.B. „1234567_3.c“
- Der modifizierte, funktionsidente C-Code (Verwendung von TOY-CPU-Registernamen):
  Dateiname ist Matrikelnummer-dot-c; also z.B. „1234567_4.c“
- Der Assemblercode:
  Dateiname ist Matrikelnummer-dot-ams; also z.B. „1234567.asm“
- Der Maschinencode:
  Dateiname ist Matrikelnummer-dot-toy; also z.B. „1234567.toy“

Spätester Abgabetermin

Montag, 21. März 2016, 14:00.

Abgabe über das „Student Tick System“ (STicS) unter https://stics.iaik.tugraz.at. Vergiss nicht, dich rechtzeitig vor der ersten Abgabe bei diesem System zu registrieren. Die Möglichkeit zur Registrierung besteht ab dem 14. März 2016. Verwende bei dieser Registrierung als E-Mail deine TU-Graz-E-Mail-Adresse. In besonderen Fällen darfst du deine Abgabe auch per E-Mail mit dem Betreff „[RO] Abgabe 1“ an ro-abgabesystem@iaik.tugraz.at machen.

Vorgangsweise

- Lege ein Ingenieurstagebuch an und trage dort immer Notizen zu deinen Arbeiten ein. Samt Datum und aufgewendeter Zeit. Dieses Ingenieurstagebuch musst du zu den Abgabegesprächen mitbringen.
- Lerne, mit Visual X-TOY umzugehen. Studiere die Beispiele, welche mit TOY mitgeliefert wurden. Du findest Visual X-TOY hier: http://introcs.cs.princeton.edu/xtoy/
- Lerne den TOY-Assembler kennen. Es gibt 2 Varianten: das Python-Programm „atoyasm.py“ (https://bigfiles.iaik.tugraz.at/get/3052ac06fd05b30bcb4c995c6cd21a6a) und „toyasm“ (basierend auf dem C-Quellcode „toyasm.c“, https://seafile.iaik.tugraz.at/f/76c5da794b/).
- Probiere kleine Beispiele zuerst.
- Vermutlich sind folgende Studienmaterialien hilfreich: https://seafile.iaik.tugraz.at/f/46e1468ef6/. Darin findest du eine detaillierte Beschreibung des TOY-Computers.
- Ein heißer Tipp ist auch der Text „From C to TOY Machine Code: A Workout in 22 Sections“ (https://seafile.iaik.tugraz.at/f/ce6ddd6e1e/)
- Eventuell gefällt dir auch folgendes Dokument (vor allem ab Kapitel 3: „Die Zerlegung einer For-Schleife“): https://seafile.iaik.tugraz.at/f/e873638306/.
- Beginne mit einem Modell des zu entwickelnden Programms in der Sprache C. Wandle den C-Code dann sukzessive so um, dass der TOY-Assemblercode sichtbar wird. Assembliere den TOY-Assemblercode mit *toyasm* oder *atoyasm.py.*
- Beginne dabei mit einem Teil des Codes. Das könnte zum Beispiel das Lesen der Eingabe sein.
- Teste den Maschinen-Code auf Visual X-TOY.

Wozu Aufgabe 1?

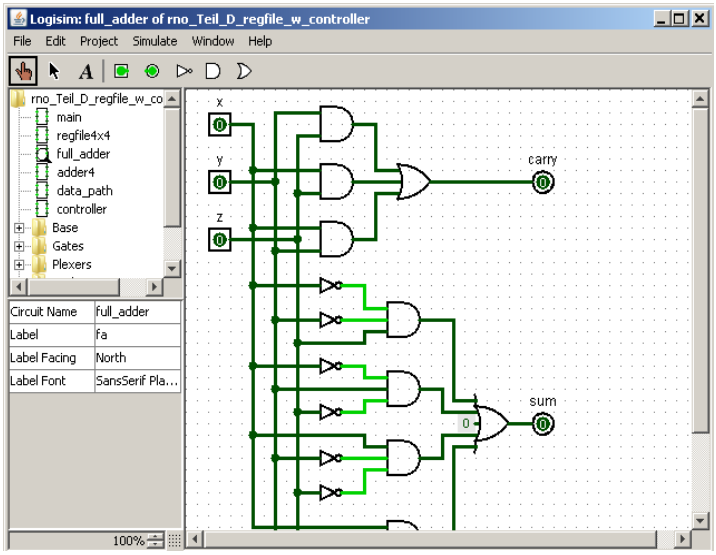
Der TOY-Computer wird in der Lehrveranstaltung als durchgehendes Beispiel verwendet, um über die Organisation von Rechenmaschinen, also von Computern zu sprechen. TOY ist sehr einfach und ein guter Startpunkt ist die Beschäftigung mit dem TOY-Simulator *Visual X-TOY*. Mit dem Wissen, welches du in der ersten Aufgabe kriegst, versetzt du dich in die Lage, viel besser den nachfolgenden Themen folgen zu können. Wir werden in der Vorlesung den TOY-Computer und dabei speziell die TOY-CPU bis zur Logik-Ebene hin entwerfen.

In diesem Beispiel beschäftigst du dich auch mit der Übersetzung zwischen C-Code und Assemblercode. Damit lernst du einige typischen Schritte dieser Übersetzung kennen; diese übernimmt ja zumeist der Compiler. Du wirst das in dieser Aufgabe erworbene Wissen bei der Aufgabe 2 und 3 dieser Übung brauchen. Auch dort beschäftigen wir uns mit dem gleichen Problem.

Aufgabe 2:

Logiksimulation mit Logisim

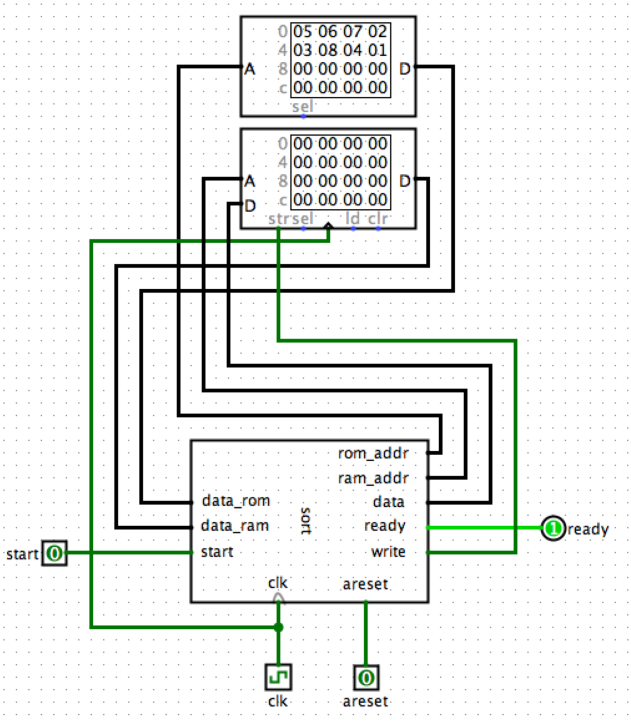
(Einzelarbeit)



Entwickle einen synchronen Automaten, welcher 16-Bit-Werte aus einem Read-Only-Memory (ROM) liest und diese Werte in sortierter Reihenfolge in einem Random-Access-Memory (RAM) ablegt. Zu Beginn besitzen alle Speicherstellen im RAM den Wert 0. Der Automat besitzt einen Start-Knopf *start*. Sobald dieser Startknopf gedrückt wird, beginnt der Automat mit dem Einlesen und Sortieren. Der Automat besitzt zudem einen Kontrollausgang *ready*. Dieser Kontrollausgang hat nach Einschalten des Automaten den Wert 1. Damit meldet der Automat, dass er „bereit“ ist.

Nach Drücken des Startknopfes geht der Wert von *ready* auf 0. Sobald der Automat aus dem ROM den Wert 0 einliest, geht das Signal *ready* wiederum auf 1.

Der Automat muss aus einem Datenpfad und einer dazu passenden Kontrolllogik bestehen.



So soll das Top-Level-Modell aussehen:

Du siehst hier die Schaltung „sort“ mit dem Kontrolleingang *start* und dem Kontrollausgang *ready*. Darüber befinden sich das ROM, aus welchem die zu sortierenden Werte gelesen werden, und das RAM, in welchem die bereits eingelesenen Werte in sortierter Reihenfolge liegen. Zusätzlich gibt es noch den obligaten Takteingang *clk*. Zum Rücksetzen der gesamten Schaltung dient das asynchrone Rücksetzsignal *areset*.

Verwende für deine Entwicklung den Logiksimulator Logisim.

Restriktionen:

- Verwende ein ROM mit 16 Werten zu je 16 Bit.
- Verwende ein RAM mit 16 Werten zu je 16 Bit.

Abzugebende Datei

- Dein Logisim-Modell: Dateiname ist Matrikelnummer-dot-circ, also z.B. „1234567.circ“.

Spätester Abgabetermin

- Montag, 2. Mai 2016, 14:00.
- Abgabe über das „Student Tick System“ (STicS) unter https://stics.iaik.tugraz.at. In besonderen Fällen darfst du deine Abgabe auch per E-Mail mit dem Betreff „[RO] Abgabe 2“ an ro-abgabesystem@iaik.tugraz.at machen.

Vorgangsweise

- Vergiss nicht, Eintragungen in dein Ingenieurstagebuch zu machen. Datum und Zeitaufwand sind wichtig.
- Studiere die Vorlesungsunterlagen, speziell die Texte „Primer on Logic Functions“, „Primer on Finite State Machines“ und „Primer on Algorithmic State Machines“.
- Baue die Schaltungen, welche du dort findest, selbst in Logisim nach.
- Verstehe, was es bedeutet, eine „synchrone Schaltung“ zu bauen.
- Studiere das Tutorium „Euklid“.
- Entwickle deine Lösung in Anlehnung an die Lösung im Tutorium „Euklid“.
- Gehe schrittweise vor. Studiere zuerst den Algorithmus in einem C-Modell oder Python-Modell. In diesem Modell kannst du den Algorithmus in seinen Einzelschritten nachbilden und auf komfortable Weise testen. Aus diesem kannst du dann das ASM-Diagramm entwickeln. Im ASM-Diagramm steckt dann „alles“ drin, was du für eine Implementierung mit Logisim brauchst.

Welche Logisim-Bauteile müssen/dürfen verwendet werden

- Es müssen ganz klar die Bestandteile „Datenpfad“ und „Kontrolllogik“ erkennbar sein.
- Datenpfad: Verwende das Logisim-Bauelement „Register“. Für Dekoder und Multiplexer darfst du ebenfalls die unter „Plexers“ zu findenden Logisim-Bauteile verwenden. Für arithmetische und logische Operationen gibt es Addierer, Subtrahierer, Vergleicher, Shifter etc. Du darfst auch alle Logik-Gatter verwenden.
- Kontrolllogik: Diese soll „ganz streng“ als Moore-Automat oder Mealy-Automat aufgebaut sein. Die Next-State-Logik und die Output-Logik sollst du so wie im Tutorium „Euklid“ gezeigt realisieren: Ausgehend von der Wahrheitstabelle kannst du die Schaltung auf Logikebene

synthetisieren. Der Zustandsspeicher soll mit einzelnen Flipflops aufgebaut werden.

Wozu Aufgabe 2?

Mit dem Simulator Logisim lernst du „digitale Hardware von unten“ her kennen. Ausgehend von einfachen Elementen wie logischen Gattern, Multiplexern und den dazu gehörigen Wahrheitstabellen sollst du größere Schaltungen „zusammen löten“ (so hätte man dies vor vielen Jahren ausgedrückt). Logisim ist vom Prinzip her sehr einfach und betont die strukturelle Bottom-Up-Sichtweise von digitalen Schaltungen. Die Beschäftigung mit Logisim soll dir helfen, die Grundelemente von digitalen Schaltungen kennen zu lernen und diese auch zu verwenden.

Ziel dieser Übung ist, ein Verständnis für den Bau von einfachen synchronen Automaten zu entwickeln. Um Automaten zu entwerfen, muss man das Grundprinzip von synchronen Automaten kennen.

Um einen speziellen Automaten zu entwerfen, beginnt man am besten mit einem Modell in einer Sprache wie z.B. C. Aus dieser entwickelt man dann ein ASM-Diagramm. Im ASM-Diagramm findet man alle notwendigen Details für den Bau des Automaten auf Registertransferebene bzw. Logikebene.

Bei der Umwandlung eines C-Modells in eine Registertransferdarstellung bzw. in ein Logikmodell lernst du Schaltungen kennen, welche Bits speichern können (Flipflop, Register) oder auch logische Schaltungen, welche einfache arithmetische Operationen ausführen können. Dieses Wissen wird dir helfen, die Verwendung der Hardware-Modellierungssprache Verilog besser zu verstehen. Wir werden Verilog ab der 3. Aufgabe einsetzen. Das Werkzeug Logisim solltest du als Lernwerkzeug betrachten. Mit Verilog betreten wir dann die professionelle Welt der Schaltungsentwicklung.

Materialien zu Aufgabe 2

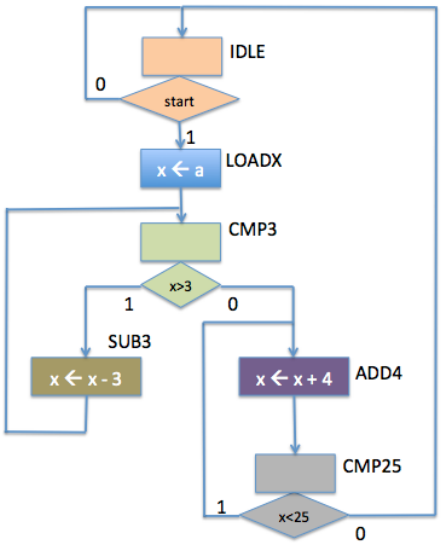
Folgende Materialien brauchst du:

- Die Vorlesungsunterlagen https://teaching.iaik.tugraz.at/ro/start, speziell die Texte „Primer on Logic Functions“, „Primer on Finite State Machines“ und „Primer on Algorithmic State Machines“.
- Das Tutorium zu Aufgabe 2
- Logisim (http://ozark.hendrix.edu/~burch/logisim/)
- Testschaltungen: „test1.circ“, „test2.circ“, „test3.circ“, „test4.circ“.
- Weitere Dateien: „test4.c“, „ram_contents“, und „rom_contents“.
- Vorlage für Lösung zu Aufgabe 2: „1234567.circ“.

Aufgabe 3:

„Funktionale Modellierung“

ISE-Modellierung mit Verilog



(Einzelarbeit)

Entwickle ein funktionales Modell des Sortier-Algorithmus aus Aufgabe 2. Beschreibe dieses funktionale Modell mit einem ASM-Diagramm. Basierend auf diesem ASM-Diagramm sollst du ein ISE-Modell in Verilog bauen. Dieses ISE-Modell sollst du dann in einem Testbett auf Richtigkeit testen.

Abzugebende Dateien

- Das ASM-Diagramm in PDF-Format: Dateiname ist Matrikelnummer, dann ASM, und dann dot-pdf, also z.B. „1234567**ASM**.pdf“.
- Dein Verilog-ISE-Modell: Dateiname ist Matrikelnummer, dann ASM, und dann dot-v, also z.B. „1234567**ASM**.v“. Diese Datei soll auch das auszuführende Programm beinhalten.

Spätester Abgabetermin

- Montag, 23. Mai 2016, 14:00.
- Abgabe über das „Student Tick System“ (STicS) unter https://stics.iaik.tugraz.at. In besonderen Fällen darfst du deine Abgabe auch per E-Mail mit dem Betreff „[RO] Abgabe 3“ an ro-abgabesystem@iaik.tugraz.at machen.

Vorgangsweise

- Vergiss nicht, Eintragungen in dein Ingenieurstagebuch zu machen. Datum und Zeitaufwand sind wichtig.
- Lerne mit Verilog umzugehen.
- Studiere das Konzept „Testbett“.
- Studiere ASM-Modelle und die daraus resultierenden ISE-Modelle in Verilog.
- Entwickle eine ASM für den Sortier-Algorithmus aus Aufgabe 2, modelliere diese ASM als ISE-Modell in Verilog und teste dieses Modell mit zufällig generierten Eingabewerten.

Wozu Aufgabe 3?

Aufgabe 3 gibt dir die Gelegenheit, mit algorithmischen Zustandsmaschinen und mit ISE-Modellen zu experimentieren. Du sollst ganz klar den Unterschied von herkömmlichen Flussdiagrammen zu ASM-Diagrammen verstehen. Du sollst dich auch intensiv mit ISE-Modellierung beschäftigen. Nicht jedes funktionierende Verilog-Programm mit sogar richtigem Output stellt ein ISE-Modell dar. Es gibt eine Reihe von Regeln, wie man mit Verilog ein ISE-Modell aufbaut. Diese sollst du lernen.

Materialien zu Aufgabe 3:

Folgende Materialien brauchst du:

- Die Vorlesungsunterlagen
- Das Tutorium zur Übung 3
- Den Verilog-Simulator Veriwell (http://sourceforge.net/projects/veriwell/). Am besten ist es, wenn du Veriwell auf Linux ausführst. Wenn du noch kein Linux hast, solltest du dir eine virtuelle Maschine mit einem Linux installieren. Verwende dazu diese Installationsanleitung: https://seafile.iaik.tugraz.at/f/9a2ac17e5e/

  (Alternativ kannst du auch den Verilog-Simulator „Verilogger“ bzw. besser dessen Kommandozeilenversion *vlogcmd* verwenden. Diese Version läuft auch unter Windows, hat jedoch den Nachteil einer Beschränkung der Dateigröße. Diese Beschränkung sollte jedoch bei dieser Aufgabe hier nicht schlagend werden, da die Aufgabe nicht sehr groß ist. Ich selbst habe die Beispiele auf Ubuntu gemacht. Veriwell ist zudem auf dem Rechner „pluto.tugraz.at“ installiert. Mit SSH einloggen und mit „veriwell“ aus der Konsole heraus starten.)

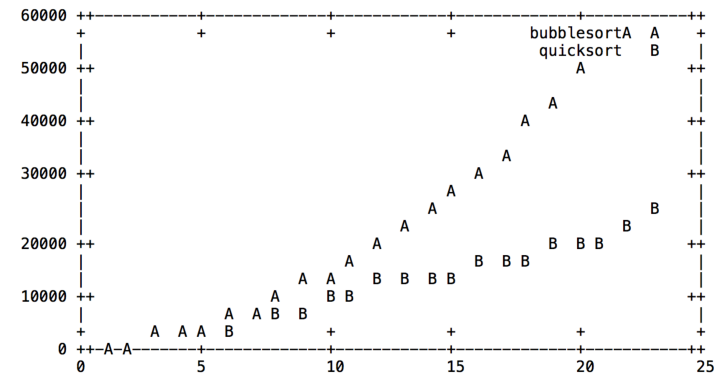
Aufgabe 4:

Vergleich „Quicksort“ und „Bubblesort“

TOY mit Stack

(Gruppenarbeit)

```
Terminal type set to 'aqua'
gnuplot> set term dumb
Terminal type set to 'dumb'
Options are 'feed  size 79, 24'
gnuplot> plot 'quick_vs_bubble' using 1:3 title 'bubblesort',
'quick_vs_bubble' using 1:2 title 'quicksort'
```




```
gnuplot> quit
$
```

- **Organisatorisches zum Beginn:**
  Der innerhalb der Gruppe geplante Prozess zur Lösung der Aufgabe, die geplante Arbeitsaufteilung innerhalb der Gruppe, die Abweichungen zwischen Realität von Plan sowie die erzielten Resultate sind zu dokumentieren und sind Bestandteil der Abgabe. Am besten ist es, wenn ihr die Ergebnisse der einleitenden Diskussion zu organisatorischen Fragen schriftlich dokumentiert und jedes Gruppenmitglied sich per Unterschrift sich zu diesem Dokument bekennt.

- **Und jetzt zum technischen Teil:**
  Entwickle ein Programm für STOY („TOY mit Stack“), welches den Sortieralgorithmus „Quicksort“ ausführt. Es sollen Werte von Standard-Input eingelesen werden und sortiert am Standard-Output ausgegeben werden.

- Teste das Programm sowohl mit dem STOY-Python-Simulator als auch mit dem Verilog-ISE-Modell von STOY.

- Vergleiche die Laufzeiten (gemessen in Taktzyklen) von Quicksort und Bubblesort. Experimentiere dabei mit verschiedenen Eingabelängen und Eingabewerten. Protokolliere die Ergebnisse deiner Experimente.

- Entwickle ein Bash-Script, welches deinen Quicksort-Code mit verschiedenen Eingabelängen und zufälligen Werten auf Richtigkeit testet.

- Bestimme die maximale Länge der Liste der zu sortierenden Zahlen, welche dein Programm zulässt. Verwende dazu zufällige Werte und auch „vorsortierte“ Eingabewerte. Protokolliere die Resultate.

- Optimiere den Code, damit die maximale Länge bei zufällig gewählten Werten maximiert wird. Wie lang ist die maximale Eingabelänge? Wodurch wird diese begrenzt?

- Implementiere eine neuen Adressierungsmodus für STOY und wende diesen auf Load- und Store-Operationen an. Dieser neue Adressierungsmodus soll „Base-plus-Index“ verwenden. Wir nennen die beiden neuen Instruktionen LDX und STX. LDX soll folgende Eigenschaften haben: R[IR[11:8]] ← mem[R[IR[7:4]] + R[IR[3:0]]]. STX macht folgendes: mem[R[IR[7:4]] + R[IR[3:0]]] ← R[IR[11:8]]. Der Befehl LDX soll den ursprünglichen Befehl JR ersetzen. Der Befehl STX soll den ursprünglichen Befehl JL ersetzen.

- Optimiere den Quicksort-Code unter Verwendung von STX und LDX. Protokolliere deine Ergebnisse.

- Fertige einen Bericht[1] an, welche die Ergebnisse der Arbeiten an Aufgabe 4 präsentiert.

Abzugebende Dateien

Abzugeben sind folgende Dateien, welche zusammen in eine ZIP-Datei gepackt werden. Der Name der ZIP-Datei ist „RO_Aufgabe4_<XXX>.zip; für die Gruppe „hadzic01“ soll diese Datei also beispielsweise so heißen: „RO_Aufgabe4_hadzic01.zip“. Die ZIP-Datei soll folgende Dateien enthalten:

- Der kommentierte und optimierte STOY-Assemblercode „Quicksort“. Dateiname ist der Gruppenname, dann die Buchstaben CODE, und dann dot-asm, also z.B. „hadzic01**_STOY**.asm“.

- Das funktionale Verilog-ISE-Modell von STOY, welches du für den Test verwendet hast: Dateiname ist der Gruppenname, dann die Buchstaben ASM, und dann dot-v, also z.B. „hadzic01**_STOY**.v“. Diese Datei soll mit Hilfe einer $include-Anweisung das auszuführende Maschinenprogramm „hadzic01_**STOYcode**.toy“ inkludieren.

- Das Maschinenprogramm „hadzic01_**STOYcode**.toy“.

- Das Bash-Script (mit Kommentaren), welches den optimierten STOY-Code laufend auf STOY mit zufälligen Werten und verschiedenen Eingabelängen auf Richtigkeit testet. Dateiname des Bash-Scripts ist „hadzic01**_STOY**.sh“.

- Der kommentierte und optimierte Assemblercode „Quicksort“ für ATOY (STOY mit den neuen Instruktionen LDX und STX). Dateiname ist der Gruppenname, dann die Buchstaben CODE, und dann dot-asm, also z.B. „hadzic**_ATOY**.asm“.

- Das funktionale Verilog-ISE-Modell von ATOY welches du für den Test verwendet hast: Dateiname ist der Gruppenname, dann die Buchstaben ASM, und dann dot-v, also z.B. „hadzic01**_ATOY**.v“. Diese Datei soll mit Hilfe einer $include-Anweisung das auszuführende Maschinenprogramm „hadzic01_**ATOYcode**.toy“ inkludieren.

- Das Maschinenprogramm „hadzic01_**ATOYcode**.toy“.

[1] Details über die Form dieses Berichtes findest du auf Seite 23.

- Das Bash-Script (mit Kommentaren), welches den optimierten ATOY-Code laufend auf ATOY mit zufälligen Werten und verschiedenen Eingabelängen auf Richtigkeit testet. Dateiname des Bash-Scripts ist „hadzic01_**ATOY**.sh“.
- Eine Dokumentation zu Aufgabe 4. Dateiname ist der Gruppenname, dann „dok“ und schließlich „dot“ und „pdf“, also z.B. „hadzic01_**DOK**.pdf“.

Spätester Abgabetermin

- Freitag, 10. Juni 2016,14:00.

  Abgabe über das „Student Tick System“ (STicS) unter https://stics.iaik.tugraz.at. In besonderen Fällen darfst du deine Abgabe auch per E-Mail mit dem Betreff „[RO] Abgabe 4“ an ro-abgabesytem@iaik.tugraz.at machen.

Vorgangsweise

- Vergiss nicht, Eintragungen in dein persönliches Ingenieurstagebuch zu machen. Datum und Zeitaufwand sind wichtig. Diese Eintragungen sind für die Erstellung des Berichts zu Aufgabe 4 vermutlich sehr nützlich.
- Setze dich rechtzeitig mit der Erstellung des Berichtes auseinander. Als Ausgangspunkt dazu kannst du den Text auf Seite 23 nehmen.
- Studiere die Materialien der Kapitel 6 bis 10 der (neuen, englischen) Vorlesungsmaterialien samt den dazu gehörigen Verilog-Dateien.
- Studiere das ASM-Modell und das ISE-Modell der CPU des STOY-Computers, wie es als Resultat im Kapitel 10.7 des (neuen, englischen) Vorlesungsstoffs vorliegt: „stoy_base_plus_offset.v “. Dies ist der Ausgangspunkt. In diesem Modell findest du bereits die CPU, den Hauptspeicher, den IO-Modul (mit Datenregister DR und Kontrollregister CR).
- Lerne, wie der Stack verwendet wird: Für Unterprogrammaufrufe, für lokale Variablen, für die Übergabe von Parametern. Lerne, wie man einen Stackframe aufsetzt und wieder abbaut.
- Studiere das Beispiel „Fibonacci“ aus dem (neuen, englischen) Vorlesungsstoff. Finde heraus, warum der rekursive Algorithmus schlechter ist als die Schleifen-Variante.
- Die Kommunikation mit dem I/O-Modul soll mittels Polling funktionieren. Wie Polling funktioniert, wird im Kapitel 9 der (neuen, englischen) Vorlesungskapitel 9 besprochen. Der I/O-Modul, welcher sich im Modell „stoy_base_pus_offset.v“ befindet, ist bereits für Polling vorbereitet. Das I/O-Datenregister DR kann über 0xFF angesprochen werden. Das I/O-Kontrollregister CR ist auf Adresse 0xFE „gemappt“.
- Für die Entwicklung des TOY-Maschinenprogramms sollte man am besten von einem C-Modell ausgehen; daraus kann man das TOY-Assemblerprogramm entwickeln. Dieses kann man schließlich mit dem TOY-Assembler in TOY-Maschinensprache umwandeln.

Wozu Aufgabe 4?

Du kannst im Rahmen der Arbeit bei dieser Aufgabe Fertigkeiten in vielfacher Hinsicht entwickeln.

(1) Du lernst bei dieser Aufgabe, wie man Modifikation bei einem bestehenden Modell vornimmt. Dies ist sehr oft üblich. Man kommt als IngenieurIn neu in ein Team und als erstes Projekt muss man sich oft in eine bestehende Umgebung einarbeiten und danach ein paar technische Veränderungen oder eine Weiterentwicklung vornehmen. Dieser Sachverhalt ist in dieser Übung vorgegeben. Man startet also nicht bei „Null“, sondern man macht eine Variante von etwas Bestehendem. Das „Einarbeiten“ in eine neue Umgebung funktioniert am besten, wenn man eine gute Dokumentation des Bestehenden vorfindet. Durch die eigenen Erweiterungen entsteht die „Umgebung“ für die Zukunft. Daraus resultiert auch die Notwendigkeit der Dokumentation der eigenen Arbeiten.

(2) Diese Aufgabe sollst du in einer Gruppe bestehend aus 3 Personen durchführen. Dabei soll es keine Arbeitsteilung geben. Alle sollen alles machen und alles verstehen. Als Vergleich verwende ich hier eine Bergsteigergruppe. Die 3er-Gruppe geht auf einen Berg. Es müssen also alle hinauf. Es gilt nicht, dass nur einer rauf geht und den anderen danach Bilder vom Gipfel zeigt. Und diese dann womöglich auch noch daran glauben, dass sie auch oben waren. Warum also dann eine Gruppe, wenn eh jeder hinauf muss? Ja warum? Wenn eine/r Schwierigkeiten beim Aufstieg hat, dann helfen die anderen. Wenn eine/r Motivationsschwächen hat, dann helfen die anderen. Wenn eine/r sich weh tut, dann helfen die anderen. Und es ist meist lustiger, gemeinsam den Ausblick zu genießen und darüber zu sprechen. Doch kommen wir zurück zur Realität: Als Ingenieur arbeitet man – ich würde sagen – nie alleine. Man arbeitet immer im Kontext einer Gruppe. Und es ist wichtig, zu üben, wie man in einer Gruppe agiert. Es ist wichtig, zu sehen, dass andere anders arbeiten als man es selbst macht. Schnelle kommen drauf, dass es Langsamere gibt und umgekehrt. Verlässlichere kommen drauf, dass es Unzuverlässige gibt – und umgekehrt. Es ist wichtig, früh im Leben damit umgehen zu lernen. Es ist wichtig, zu lernen, Abmachungen mit anderen zu treffen. Aus Gruppenerfolgen und auch aus Gruppenmisserfolgen lernt man und wird Stück für Stück professioneller.

(3) Du lernst das Konzept „Testbett“ und „Device Under Test“ („DUT“) auf mehrfache Hinsicht kennen. Innerhalb des Verilog-Modells gibt es Testbett und DUT. Das Bash-Script ist eine weitere Ebene, auf der Testbett und „DUT“ vorkommen. Ich bin der festen Überzeugung, dass dir diese Sichtweise und die bei dieser Übung erlernten Fähigkeiten beim Testen von Systemen noch oft begegnen werden.

(4) Du lernst im Detail kennen, wie der Stack funktioniert und, vor allem, wie er verwendet wird. Nach erfolgreicher Arbeit an dieser Aufgabe kennst du dich bei Stackpointer, Basepointer, lokalen und globalen Variablen, Parameterübergabe, und Sinn und Unsinn von Rekursion gut aus.

(5) Du lernst, wie man als IngenieurIn sowohl Software als auch Hardware „gleichzeitig“ für ein bestimmtes Ziel optimieren kann. Der Einbau von neuen Instruktionen in STOY zum Zwecke der Optimierung einer Berechnung wird geübt. Diese Betrachtungsweise wird oft „Hardware-Software-Co-Design“ genannt. Du lernst vor allem auch, dass die Arbeitsweise bei Hardware-Entwurf und Software-Entwurf sehr ähnlich sind. Wenn du dies verstanden hast, bist du vielen

IngenieurskollegInnen vermutlich „weit voraus“.

Materialien zu Aufgabe 4:

Folgende Materialien brauchst du:

- Die (neuen, englischen) Vorlesungsunterlagen, hauptsächlich Kapitel 6 bis 10. Du findest diese unter https://teaching.iaik.tugraz.at/ro/start.
- Den Verilog-Simulator Veriwell (http://sourceforge.net/projects/veriwell/). Am besten ist es, wenn du Veriwell auf Linux ausführst. Wenn du noch kein Linux hast, solltest du dir eine virtuelle Maschine mit einem Linux installieren. Verwende dazu diese Installationsanleitung: https://seafile.iaik.tugraz.at/f/9a2ac17e5e/

  (Alternativ kannst du auch den Verilog-Simulator „Verilogger“ bzw. besser dessen Kommandozeilenversion *vlogcmd* verwenden. Diese Version läuft auch unter Windows, hat jedoch den Nachteil einer Beschränkung der Dateigröße. Diese Beschränkung sollte jedoch bei dieser Aufgabe hier nicht schlagend werden, da die Aufgabe nicht sehr groß ist. Ich selbst habe die Beispiele auf Ubuntu gemacht. Veriwell ist zudem auf dem Rechner „pluto.tugraz.at“ installiert. Mit SSH einloggen und mit „veriwell“ aus der Konsole heraus starten.)

Abgabegespräche

Es gibt 2 Abgabegespräche. Eines nach der Abgabe der 2. Aufgabe und ein zweites am Ende der Übung. Du triffst dich dabei zusammen mit deinen Gruppenmitgliedern mit dem/der TutorIn oder mit dem Lehrer. Typischerweise dauert so ein Gespräch eine halbe Stunde. Du machst zusammen mit deinen Gruppenmitgliedern mit dem/der TutorIn oder dem Lehrer einen individuellen Termin für dieses Gespräch aus.

Neben der Leistungsüberprüfung sollte mit der Möglichkeit der Präsentation ein positiver Stimulus geschaffen werden: Die aktive Rolle der Studierenden sollte in den Vordergrund gerückt werden. Bei offensichtlichem Leistungsdefizit sollte dem Studierenden ein „Warnschild“ signalisiert werden.

Die (positive und/oder negative) Kritik zur Präsentation beim ersten Abgabegespräch dient zweierlei:

- Verbesserung bzw. Korrektur der Präsentation mit dem Ziel, bei der zweiten Präsentation am Ende des Kurses besser abzuschneiden. Eine nicht gelungene Präsentation sollte bei vorhandenem Wissen zu keinem Punkteabzug führen.
- Beurteilung des Detailwissens über die Aufgaben. Bewertet wird auch die Existenz von Teillösungen. Dies mündet in der Anzahl der Punkte für die Note.

Beim zweiten Abgabegespräch gibt es folgende Aspekte:

- Die Studierenden haben die Chance, durch eine entsprechende Präsentation ihrer Arbeit aktiv zu bestimmen, welchen Verlauf das Gespräch nimmt.
- Die Möglichkeit zur Präsentation der Arbeit sollte von den Studierenden als Chance begriffen werden, ihren Freiraum zu erhöhen. Eine nicht gelungene Präsentation führt zu keinem Punkteabzug, jedoch zu einer prüfungsähnlichen Atmosphäre, da der/die TutorIn bzw. der Lehrer gezwungen ist, intensiver als Fragesteller aufzutreten.
- Bestandteile des Gesprächs sind typischerweise:
  - Erläuterung der Lösungen
  - Demonstration am Rechner
  - Besprechung aufgetretener Schwierigkeiten
  - Vorgangsweise beim Entwurf
  - Benötigte Arbeitszeit
  - Aufteilung der Arbeit unter den Gruppenmitgliedern
  - Diskussionsphase

Punkte

Wenn Tutorium 0 nicht besucht wurde, gibt es 0 Punkte und keine Note. Wenn Tutorium 0 besucht wurde und ansonsten keine Aufgabe abgegeben wurde, gibt es ebenfalls keine Note. Sobald eine der Aufgaben abgegeben wurde, gibt es eine Note.

Aufgabe 0	keine Punkte
Aufgabe 1	max. 15 Punkte
Aufgabe 2	max. 30 Punkte
Aufgabe 3	max. 15 Punkte
Aufgabe 4	max. 40 Punkte

Notenschlüssel

Zur Erreichung einer positiven Note müssen alle 4 Aufgaben abgegeben werden und bei jeder Aufgabe muss ein „ernsthafter Versuch“ erkennbar sein, die Aufgabe lösen zu wollen. Als „ernsthafter Versuch“ bei den Aufgaben 1 bis 3 zählt $^1/_3$ der maximal zu erreichenden Punkte. Bei Aufgabe 4 müssen für einen „ernsthaften Versuch“ zumindest 20 Punkte erreicht werden.

- 0 – 50 Punkte Nicht genügend
- 51 – 62 Punkte Genügend
- 63 – 75 Punkte Befriedigend
- 76 – 87 Punkte Gut
- 88 – 100 Punkte Sehr gut

Schwindeln, Täuschungsversuch, Plagiat

Wenn ein/e Student/in seine/ihre Arbeit erschwindelt, wird dies mit der Note "U" ("Ungültig/Täuschung") bewertet. Wenn ein Team seine Arbeit erschwindelt, werden alle Teammitglieder mit der Note "U" ("Ungültig/Täuschung") bewertet.

Huch, es geht sich mit der Zeit nicht aus

Ein wesentlicher Aspekt auf dem Weg zum Profi ist der richtige Umgang mit dem Faktor „Zeit“. Jedes Projekt hat einen Liefertermin. Das ist der spätestens mögliche Liefertermin. Wenn du einmal verstanden hat, dass du ja auch früher liefern darfst, dann hast du bereits viel gewonnen. Du darfst also in deinem eigenen Kalender durchaus einen „Privatliefertermin“ eintragen, welcher zum Beispiel eine Woche früher als der vom Lehrer geforderte Termin ist. Die zweite wichtige Komponente ist der „Plan“. Dazu musst du die Arbeit eines Projektes anfänglich abschätzen, einen bestimmten Risikofaktor einberechnen, und dann die geplanten Arbeitszeiten in deinem Kalender festlegen.

Nimm als Beispiel die Arbeit an Aufgabe 1. Du liest die Aufgabenstellung durch und findest heraus, dass du „keine Ahnung“ hast, wie lange die Arbeit daran werden wird. „Keine Ahnung“ bedeutet zweierlei: (1) Du setzt dich rasch – man nennt dies „as soon as possible“ (ASAP) – hin, um eine „Ahnung“ zu kriegen. (2) Je weniger „Ahnung“, desto höher musst du den „Risikofaktor“ setzen. Daraus folgt für mich die wichtigste Grundregel: Bei „wenig bis gar keine Ahnung“ muss ich mich „sofort“ (= ASAP) darum kümmern, wenn es mir wichtig ist, zeitgerecht liefern zu können. Du könntest also zum Beispiel 5 Stunden investieren um eine „Ahnung“ zu bekommen, wie viel Aufwand sich hinter Aufgabe 1 verbirgt.

Nehmen wir ein zweites Beispiel: Du hast schon mal auf Assemblerebene programmiert und hast also bereits „ein bisschen Ahnung“ von Aufgabe 1. Du schätzt, dass es wohl in etwa 3 Tage Arbeit sein wird, doch so richtig sicher bist du dir dabei nicht. Ich würde in einem solchen Fall das Risiko bewerten und statt der 3 Tage vermutlich sicherheitshalber 9 Tage planen. Auf jeden Fall würde ich die 9 Tage Arbeit in meinem Kalender einplanen. Im „Hinterkopf“ kannst du ja behalten, dass – wenn du gut drauf bist – ein Teil dieser 9 Tage nicht gebraucht wird.

Je öfter du das Spiel mit „den geplanten Zeitaufwand im Voraus schätzen“, „den tatsächlichen Zeitaufwand messen“ und schließlich „aus dem Vergleich zwischen Schätzung und Messung lernen“ machst, desto besser lernst du schätzen und planen. Die Fähigkeit zu planen wirst du dein ganzes Leben lang brauchen können.

Die obige Vorgangsweise führt mit großer Wahrscheinlichkeit zur erfolgreichen, rechtzeitigen Fertigstellung von Arbeiten und damit zur fristgerechten Lieferung.

Das Eintreffen unerwarteter Ereignisse kann jedoch auch einen guten Plan ins Wanken bringen. Wenn dies (hoffentlich selten) der Fall sein sollte, dann solltest du folgende Punkte beachten: Ein Agieren vor dem Liefertermin ist viel besser als ein Agieren nach dem verpassten Liefertermin. Ein Brief an den Lehrer vor dem Abgabetermin ist wesentlich effizienter also ein Brief danach. Eventuell ist ein persönliches Vorsprechen sogar noch am besten; damit vermeidest du den Eindruck von „so wichtig ist mir das aber nicht“. Professionell ist, wenn man vorbereitet ist, einen Ersatzliefertermin anzubieten; diesen muss man dann aber auch wirklich einhalten.

Nehmen wir also an, du bist mit dem Projekt zu Aufgabe 1 „on track“, doch zum Abgabetermin fehlt noch „ein bisschen“. Dann kannst du bei dieser Lehrveranstaltung dein Lieferverzögerungskonto (insgesamt maximal 3 Tage für die gesamte KU) anzapfen. Solltest du dies machen, dann solltest du gleichzeitig auch analysieren, wie es dazu kommen konnte. Daraus kannst du für die Zukunft lernen. Vielleicht bist du dem „20-80-Problem“ auf dem Leim gegangen. Es kommt oft vor, dass man für die ersten 80% eines Projektes lediglich 20% der zu Verfügung stehenden Zeit braucht; doch die verbleibenden 20% des Projekts verbrauchen 80% der Zeit. Wie gesagt: Lerne, dieses Problem in den Griff zu kriegen.

OK. Du hast geliefert. Doch kurz danach, jedoch noch vor dem Abgabegespräch, kommst du drauf, dass deine Lieferung fehlerhaft bzw. ungenügend ist. In diesem Fall würde ich „proaktiv“ handeln: Ausbessern bzw. nachbessern und – egal, wie spät – nochmals in verbesserter Form abgeben. Damit hast du klar zum Ausdruck gebracht, dass du „auf Draht“ bist. Mag sein, dass du für die verbesserte, nachträgliche Abgabe keine Extrapunkte kriegst; auf jeden Fall kannst du jedoch für das Abgabegespräch selbst mit deinem „augmentierten Wissen“ „punkten“. Ich kenne keine Situation, in welcher ein solcher „proaktiver“ Ansatz nicht geholfen hätte.

Jetzt hast du das Abgabegespräch mit der Tutorin gemacht und hast unglücklicherweise knapp das Punkteminimum verfehlt. Und dies trotz deiner Extraabgabe nach Abgabefrist. In diesem besonderen Fall solltest du bereit sein, durch geeignete, nachträgliche „Zusatzlieferung“ über das Limit zu kommen. Du könntest der Tutorin zum Beispiel vorschlagen, innerhalb von ein paar Tagen noch „jede Menge auf die Waage zu legen“. Deine Tutorin wird auch in diesem Fall kooperativ sein.

Kommen wir zum unerwünschten Fall: Du bist jung und willst die Grenzen ausloten. Die Abgabefrist verstreicht und du möchtest wissen, was passiert, wenn du einfach „durchtauchst“. In der Schule gab es ja auch immer wieder eine Lösung; egal, wie weit weg du vom Soll warst. Na ja, in diesem Fall haben wir es einfach: Solltest du nie liefern, dann kriegst du keine Note. Solltest du zumindest einmal geliefert haben, dann kriegst du eine negative Note.

Und wie lang soll die Dokumentation sein?

Die (aus Sicht der StudentIn) „blöde“ Antwort ist vermutlich auch die weiseste: So kurz wie möglich, aber nicht kürzer. Meistens drückt diese Frage ein Defizit im Verständnis des Sinns einer Dokumentation aus. Deshalb möchte ich hier etwas breiter ausholen.

Eine Dokumentation ist ein schriftlicher Bericht. Der Zweck eines Berichts ist zu informieren oder zu überzeugen. Als StudentIn möchtest du einerseits über deine Arbeit informieren, andererseits aber auch die LehrerIn von der Qualität deiner Arbeit überzeugen.

Ein Bericht ist ein Aufsatz, welcher eine bestimmte Form hat. Die Form drückt sich in der Verwendung einer typischen Struktur und einer geeigneten Sprache aus. Die sofort sichtbaren Elemente einer typischen Struktur sind Titel der Arbeit, Autorennennung, Kapitelüberschriften und Quellenangabe. Das erste Kapitel heißt immer „Einleitung“ und das letzte Kapitel immer „Zusammenfassung“. Illustrationen werden immer mit einer „Caption“ versehen. Es steht also „Abbildung 3.4“ oder so ähnlich dabei. Im Text geht man dann auf die Illustration ein, indem man etwa schreibt: „Wie in Abbildung 3.4 zu sehen ist, ...“.

Strukturelemente findet man jedoch auch innerhalb eines Kapitels. Schau dir den Text, welchen du soeben liest an. Der Text ist in Absätze gegliedert. Als Zeichen für einen neuen Absatz wurde ein etwas größerer Zeilenabstand gewählt. Jeder Absatz drückt eine „Idee“ aus. Ein Absatz besteht typsicherweise aus mehreren Sätzen. Meistens bezieht sich ein Satz auf seinen Vorgänger. Der Leser sollte in die Lage versetzt werden, beim Lesen ohne Schwierigkeiten einen sinnvollen Zusammenhang zwischen den einzelnen Sätzen zu erkennen. Damit dies de Leser leichter gemacht wird, verwenden wir verschiedene Tricks: Überleitungswörter wie „jedoch“, „außerdem“, „beispielsweise“ usw. sind eine Möglichkeit.

Auch innerhalb eines Satzes gibt es eine fixe Struktur. Die Wörter eines Satzes sind durch die dazwischen liegenden Leerzeichen und sonstigen Satzzeichen erkennbar. Die Reihenfolge der Wörter wird durch die Grammatik bestimmt. Jedes Wort folgt seinen eigenen Rechtschreibregeln. Und jeder Buchstabe eines Wortes ist Bestandteil eines Zeichensatzes („Font“).

Es gibt auch eine Reihe von Regeln, was man im ersten Kapitel, also in der „Einleitung“ eines Berichtes schreibt. Eine typische Reihenfolge könnte so aussehen: Worum geht es in der Arbeit? Warum ist diese Arbeit wichtig/interessant? Welche Voraussetzungen sind notwendig? In welcher Reihenfolge wird die Arbeit präsentiert?

In den weiteren Kapiteln werden dann, wie in der Einleitung festgelegt, die einzelnen Details beschrieben. Das letzte Kapitel fasst die Arbeit zusammen. Es hilft dem Leser nach dem Aufsaugen aller Details „zur Oberfläche“ zurückzukehren. In der Zusammenfassung kann man dem Leser also nochmals das Wichtigste der Arbeit „in Erinnerung“ rufen.

Kommen wir zur Frage der Sprache. Die Sprache sollte informativ und nüchtern sein. Die Zeitform ist (meist) das Präsens. Das wichtigste an der Sprache in der Wissenschaft ist Genauigkeit. Man sagt genau das, was man sagen möchte. Die Sprache sollte auch „klar“ sein. Damit ist gemeint, dass alles Nicht-Notwendige weggelassen wird. Ein weiteres Attribut der Sprache sollte „Aufrichtigkeit“ und „Offenheit“ sein. Die Sprache sollte also nicht „geheimnisvoll“, „zweideutig“ oder „augenzwinkernd“

sein. Schließlich sollte die Sprache „geläufig“ und „vertraut“ sein. Du sollt durchaus einfache Wörter verwenden.

Am Anfang dieses Kapitels stand die Frage nach der Länge der Dokumentation. Wir haben bis jetzt jedoch nur über Struktur und Sprache gesprochen. Mit diesen Mitteln sollst du nun einen Text machen, der „eine Geschichte erzählt“. Die „Geschichte“ besteht aus den Ergebnissen deiner Arbeit im Rahmen der Konstruktionsübung. Dem Leser deiner „Geschichte“ sollte auf möglichst einfache und rasche Art und Weise klar werden, was du gemacht hast und welche Ergebnisse bei deiner Arbeit rausgekommen sind. Ohne Schnörksel. Ohne Rufzeichen. Ohne Abschweifungen. Also „kurz und bündig“.

Möglicherweise bist du nach wie vor der Meinung, dass das Anfertigen einer Dokumentation im Rahmen der Konstruktionsübung „überflüssig“ ist. „Schikane“ sozusagen. Dieser Ansicht halte ich Folgendes entgegen: Ein Profi im technisch-wissenschaftlichen Umfeld ist in der Lage, sich rasch in ein neues Problem einzuarbeiten und geeignete Lösungen vorzuschlagen oder auch anzufertigen. Hier ist also „technisches Know-How“ gefragt. Das übst du im Rahmen jeder Konstruktionsübung im Detail. Jedes Projekt wird immer auch durch eine geeignete Dokumentation begleitet. Spätestens am Ende eines Projektes sollte alles Wesentliche zum Projekt in einem Bericht festgehalten werden. Dieser Bericht dient hauptsächlich dazu, Resultate für die mögliche Weiterarbeit am Projekt zu einem späteren Zeitpunkt festzuhalten. Eine neue Mitarbeiterin – das kannst auch du selbst sein – kommt ins Projekt und „liest sich ein“, um darauf hin „am Projekt weiterzuarbeiten“. Eine professionelle Dokumentation dient also dazu, in einer Gruppe effizient arbeiten zu können.

Eine spezielle Form eines Berichts ist „der Antrag“. In einem Antrag versucht der Schreiber den Leser zu überzeugen, Ressourcen (Zeit, Mitarbeiter, Geld, Geräte) zur Verfügung gestellt zu kriegen. Ziel eines Antrags ist dessen „Genehmigung“. Es geht also um die Kunst des Überzeugens. Auch ein Antrag hält sich an die meisten der oben skizzierten Tipps zu Struktur und Sprache.

Im Rahmen der Konstruktionsübung sollst du also auch die Gelegenheit haben, die Anfertigung einer Dokumentation zu üben. Die Fertigkeit, einen wissenschaftlichen Text zu schreiben, kann man nur dadurch erreichen, indem man es tut. Man muss also schreiben.

Als Abschluss meiner Überlegungen zur Dokumentation möchte ich die Frage „Englisch“ oder „Deutsch“ beleuchten. Die Fachsprache im IT-Bereich ist Englisch. Die meisten Texte sind in Englisch verfasst. Je früher du das Schreiben von Berichten in englischer Sprache übst, umso besser. Solltest du im Rahmen dieser Konstruktionsübung jedoch davor zurückscheuen und den Bericht auf Deutsch verfassen, so ist dies erlaubt. Als Lehrer möchte ich dir jedoch in diesem Falle mitteilen, dass dir mit großer Wahrscheinlichkeit das Erlernen des Verfassens von Berichten in englischer Sprache noch bevorsteht. Je älter du jedoch wirst, umso schwieriger wird es für dich, eine neue Sprache zu erlernen.

ANHANG

Was darf in einem ISE-Verilog-Modell nicht vorkommen?

1 Warum soll ich das lesen?

1.1 Das weiß ich schon

Hardwarebeschreibungssprachen haben wie alle Programmiersprachen eine Syntax. Wenn man diese Syntax nicht einhält, beschwert sich der Compiler beim Übersetzen mit einer Fehlermeldung. Manches Mal auch nur mit einer Warnung, dass etwas nicht korrekt sein könnte. Diese Problematik kennst du.

Ist ein Programm dann einmal aus Sicht des Compilers fehlerfrei, also syntaktisch korrekt, dann kann man es exekutieren oder auch simulieren, wie wir bei Hardwarebeschreibungssprachen sagen. Dann kommt „etwas Richtiges“ oder „etwas Falsches“ heraus. Dies lässt sich durch Analyse des Ergebnisses feststellen. Auch das kennst du.

Es besteht auch die Problematik des Testens: Habe ich mein Programm mit genügend vielen Testfällen untersucht? Denn mit jedem Testfall versuche ich lediglich das Vorhandensein von Fehlern zu testen. Ich kann mit Testen normalerweise nicht die Abwesenheit von Fehlern feststellen. Auch das lernst du in Lehrveranstaltungen, wo Programmieren unterrichtet wird.

Um Programme leichter lesen zu können gibt es meistens auch Vorgaben, wie man Variablen benennen soll, wie man die Einrückungen machen soll, usw. Das nennt man Coding-Style. Auch hier gibt es üblicherweise einigermaßen klare Vorgaben um in Gruppen arbeiten zu können.

1.2 Das weiß ich vielleicht noch nicht

In Rechnerorganisation verwenden wir die Hardwarebeschreibungssprache Verilog, um Modelle von Hardware zu bauen. Du lernst dabei mehrere Modellierungsarten kennen. Die erste nennen wir „funktionale Modellierung mit impliziter Zustandskodierung“. Dann gibt es noch die „gemischte Modellierung“ und die „Strukturmodelle“. Je nach Modellierungsart gibt es hier weitere Einschränkungen, ob ein Modell „richtig“ oder „falsch“ ist. Mit dieser Art von Einschränkungen haben viele Studierende anfänglich Probleme.

Wenn man den Inhalt der Lehrveranstaltung gut versteht, dann hat man eh keine wirklichen Probleme mit diesem Thema. Doch auf dem Weg zum guten Verständnis des Stoffes kann man auch straucheln und der Abgabetermin ist auf einmal da. Dieser Anhang versucht, dir auf eine alternative Art darzulegen, was in deinem Code nicht vorkommen darf. Ich versuche auch, Gründe für dieses Verbot kurz und bündig darzulegen. Die Details dazu findest du „natürlich“ auch in den Vorlesungs- und Übungsunterlagen.

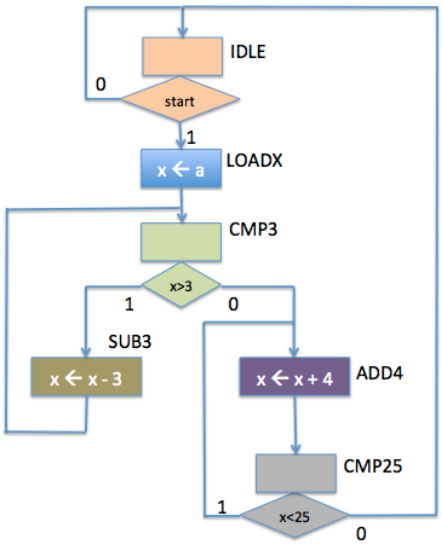
Dieses Dokument dient also dazu, dir auf einfache Art mitzuteilen, ob deine Modelle einige typische Fehler, die immer wieder gemacht werden, enthalten.

Dabei versuche ich im Gegensatz zu den Vorlesungsunterlagen von der „anderen Seite her“ zu argumentieren: Was ist falsch? Und warum ist es falsch?

2 ASMs und implizite Zustandskodierung

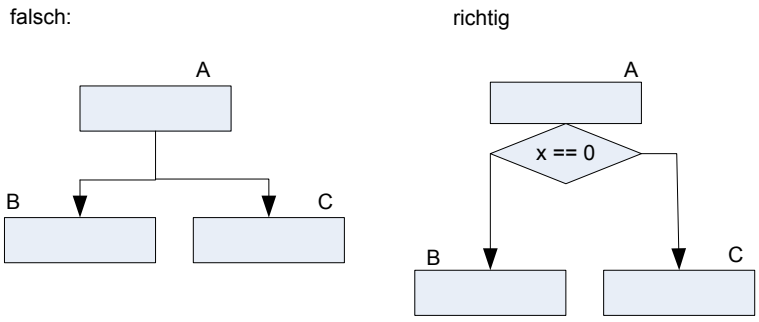
2.1. Maschinen, deren Zustände und Ausgänge, und die Rolle von Registertransfers

Betrachten wir das untenstehende ASM-Diagramm.

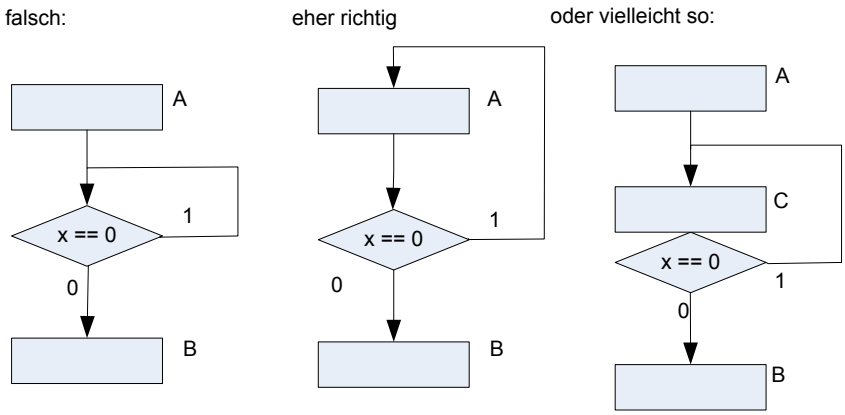


Eine ASM kann durch ein ASM-Diagramm beschrieben werden. Es zeigt die Abfolge von Zuständen. Diese sind durch Rechtecke gekennzeichnet. Zustände besitzen Namen, welche wir außerhalb hinschreiben. So etwa haben wir hier die Zustände IDLE, LOADX, CMP3 usw. Die Abfolge der Zustände wird durch die verbindenden Pfeile zwischen den Rechtecken definiert. Wenn mehrere Nachfolgezustände möglich sind, so verwenden wir eine dazwischen liegende Raute. Diese gehört immer zum ursächlich vorausgehenden Zustand dazu. Wenn die Entscheidung über den Nachfolgezustand eine komplexere Situation darstellt, dann können wir auch mehrere Rauten zusammen als eine Art „Mehrweg-Entscheidung“ betrachten. Im Zustand CMP3 wird etwa durch Betrachtung des Wertes von x entschieden – und das drücken die Rauten aus –, welcher Nachfolgezustand eingenommen werden soll. Der Zustandsübergang wird jeweils durch die aktive Taktflanke bestimmt. In dieser Lehrveranstaltung ist dies immer die positive Flanke.

Nachfolgend ein paar Ausschnitte aus „falschen“ ASMs sowie deren Verbesserung:



Der Fehler ist, dass nach dem Zustand A nicht gleichzeitig der Zustand B und der Zustand C eintreten kann. Es fehlt die Raute, in welcher während des Zustands A entschieden wird, was der Nachfolgezustand sein soll.



Der Fehler im linken Bild ist, dass die Raute alleine ja kein eigener Zustand ist, sondern eventuell zum Zustand A dazu gehört (wie im mittleren Bild gezeigt) oder einen eigenen zusätzlichen Zustand benötigt (wie im rechten Bild gezeigt).

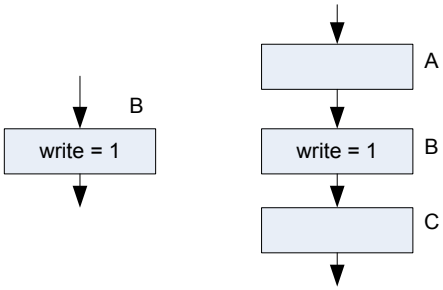
Betrachten wir das fehlerhafte ASM-Diagramm oben links: Wenn im Zustand A der Wert x nicht gleich 0 ist, dann ist der Nachfolgezustand B. Doch was passiert, wenn x gleich 0 ist? Der Pfeil mündet im linken falschen Bild in keinem Nachfolgezustand. Vermutlich hat der Irrende (die Irrende) eine Situation wie im mittleren oder rechten Bild gezeigt gemeint.

Bei allen Bildern oben ist noch ein grundsätzlicher Fehler drin: Was passiert nach den Zuständen B. Ein ASM-Diagramm sollte zu jedem Zustand einen Nachfolgezustand definieren. Eine ASM braucht für jeden Zustand einen Nachfolgezustand.

Was bedeuten jetzt die in die Zustandsrechtecke hinein geschriebenen Texte? Da gibt es genau zwei strikt zu unterscheidende Situationen: Einerseits die Festlegung, welchen Wert die Ausgänge des Automaten während dieses Zustands haben, und andererseits die Veränderung von Registerwerten; die tatsächliche Veränderung findet dann bei Verlassen des Zustands statt.

Festlegung von Ausgängen:

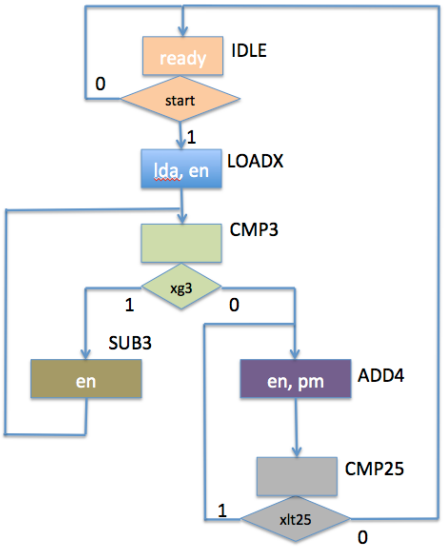
Dies mache wir mit dem „=“-Zeichen. Wenn also ein Ausgangssignal (z.B. „write“) in einem Zustand den Wert 1 annehmen soll, dann schreiben wir das in den Zustand folgendermaßen hinein:



Aufgepasst: Damit ist das Ausgangssignal „write“ lediglich in diesem Zustand (hier Zustand B) auf 1.

Will man in anderen Zuständen für das Ausgangssignal „write“ andere Werte, dann schreibt man die entsprechenden Angaben ebenfalls in die zugehörigen Zustände hinein. Damit die Diagramme jedoch nicht allzu überladen werden, treffen wir folgende Konvention: Wenn das Ausgangssignal den „inaktiven“ Wert 0 haben soll, dann brauchen wir das im ASM-Diagramm nicht explizit hinschreiben. Dies ist in obigem Bild rechts in den Zuständen A und C der Fall. Dort hat das Signal „write“ jeweils den Wert 0.

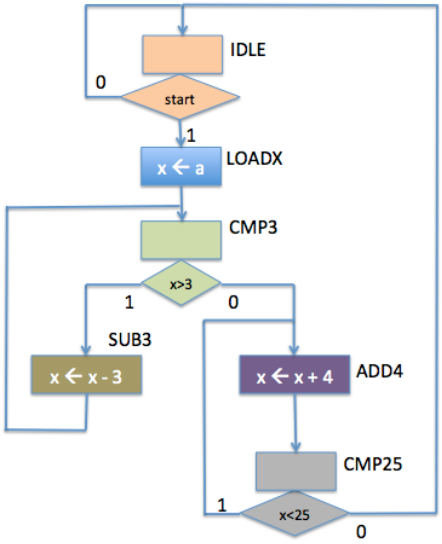
Zusätzlich treffen wir die Konvention, dass wir bei 1-Bit-Signalen mit dem lediglichen Nennen des Ausgangssignals ausdrücken, dass dieses Signal dann 1, also „aktiv“ sein soll. Im folgenden Bild trifft dies auf die Signale „ready“, „lda“, „en“ und „pm“ zu.



Das Ausgangssignal „ready“ ist im obigen ASM-Diagramm also während des Zustands IDLE auf 1, in allen anderen Zuständen hingegen auf 0.

Festlegung von Registertransfers:

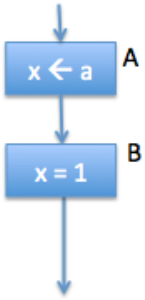
Die zweite Möglichkeit, „etwas“ in das Zustandsrechteck einer ASM hineinzuschreiben, sind Registertransferanweisungen. Diese unterscheiden sich von der vorhin besprochenen Anweisung zur Festlegung des Wertes eines Ausgangs grundsätzlich.



Nehmen wir als Beispiel das oben dargestellten ASM-Diagramm. Du siehst kein „=“-Zeichen! Sondern einen Pfeil, der nach links zeigt. So bestimmt etwa die Registertransferanweisung „x ← a“, dass der Wert des Registers mit dem Namen x bei Verlassen des Zustands LOADX den Wert a annehmen soll. Bei Verlassen des Zustands SUB3 wird der Wert im Register x um 3 vermindert. Wird während eines Zustands einem Register keine Registertransferanweisung „verordnet“, dann behält es den Wert. So verändert sich im obigen Beispiel der Wert von x bei Verlassen des Zustands CMP3 nicht.

Was wird oft falsch gemacht?

Ein typischer Fehler ist, dass man die beiden Fälle verwechselt. Oder gar nicht unterscheiden kann. Oder dass man folgendes macht:



Das ASM-Diagramm oben macht aus Hardware-Sicht keinen Sinn. Denn im Zustand A verwenden wir x als ein Register. Und im Zustand B verwenden wir x wie ein Ausgangssignal. Dies darf bei der Modellierungsart, wie wir sie in dieser Lehrveranstaltung lernen, nicht vorkommen.

2.2 Und nun zum Verilog-Modell

Jedes ASM-Diagramm kann auch in Verilog modelliert werden. Damit meinen wir, dass sich das Verilog-Programm bei der Simulation genau so verhält wie wir es mit dem ASM-Diagramm meinen. Wenn du alles richtig machst, dann gibt es eine Übereinstimmung zwischen

- deiner Vorstellung, wie eine Schaltung „funktionieren“ sollte,
- dem ASM-Diagramm, mit dem du deine Vorstellung grafisch ausdrückst, und

- dem Simulationsergebnis des Verilog-Modells, welches aus dem ASM-Diagramm entsteht.

Das ist wichtig. Doch noch wichtiger ist es, dass die in der Lehrveranstaltung gelernte Modellierungstechnik zu Modellen führt, welche „synthetisierbar“ sind. Damit meint man, dass aus dem Modell zwingend eine funktionierende synchrone Schaltung gebaut werden kann. Und das auf so einfache Weise, dass man Übersetzungsprogramme, sogenannte Synthesizer dafür entwerfen kann.

Registertransferanweisungen versus Definieren von Ausgangswerten

Registertransferanweisungen schreibt man so. Und nur so!:

```
R <= @(posedge clk) value;
```

Wir sehen, dass dem Register R bei der steigenden Taktflanke der Wert „value“ zugewiesen werden soll.

Und die Festlegung von Ausgangswerten von Automaten schreibt man so. Und nur so!:

```
write = 1;
```

Bei der Festlegung der Ausgangswerte kommt die Taktflanke also nicht vor.

Eine Verilog-Variable mit dem Datentyp „reg“ kann sowohl ein Register als auch ein Ausgangssignal sein. Aus der Deklaration kann man also nicht erkennen, worum es sich handelt. Die Festlegung, ob es sich um ein Hardware-Register oder um ein Ausgangssignal eines Automaten handelt, wird durch die Unterscheidung der beiden oben dargestellten Code-Zeilen getroffen.

Folgendes ist also falsch und macht mit der bei uns verwendeten Modellierungsmethode keinen Sinn:

```
//...
@(posedge clk) enter_new_state(`ZUSTAND_A);
R <= @(posedge clk) value;
//...
@(posedge clk) enter_new_state(`ZUSTAND_B);
R = 1;
//...
```

Zuerst wird dem Register R mit einer Registertransferanweisung ein Wert (value) zugewiesen. Und einige Taktperioden später wird R wie ein Ausgangswert behandelt. Das geht nicht.

Hinweise auf ein falsches Verständnis liefern meistens auch Codestücke in folgender Form:

```
//...
@(posedge clk) enter_new_state(`ZUSTAND_A);
R <= value;
//...
```

In den meisten Fällen hat der/die AutorIn dieses Codestücks nicht verstanden, wie man Registertransferanweisungen bei ISE-Modellen richtig modelliert. Es fehlt wohl das „@(posedge clk)“.

Die Verwendung von „wait“:

„wait“ darf in der Always-Schleife, welche das Modell mit impliziter Zustandskodierung darstellt, niemals vorkommen. Verwende „wait“ nur in anderen Code-Teilen wie etwa im Testbett.

Die Verwendung von „initial“:

„initial“-Blöcke dürfen keine Werte initialisieren, welche in der Always-Schleife des Modells mit impliziter Zustandskodierung vorkommen. Die dort vorkommenden Hardware-Register und Ausgangsvariablen müssen in der Always-Schleife mit entsprechenden Registertransferanweisungen initialisiert werden.

3 Beispiele mit Fehlern

In den nachfolgenden Beispielen findest du einige typische Fehler bei der ISE-Modellierung. Ich habe sie durch Kommentare versehen:

```
initial
    begin
        write <= @(posedge clk) 0;      // FALSCH: Synchrones Inizialisieren von
                                        // FALSCH: Hardware-Registerwerten
        i <= @(posedge clk) 0;          // FALSCH: geht bei ISE-Modellierung
                                        // FALSCH: nur in always-Schleife
    end

always
    begin
        @(posedge clk) enter_new_state(`IDLE);
        wait(wrack == 1);               // FALSCH: "wait" darf im ISE-Modell nicht vorkommen

        @(posedge clk) enter_new_state(`WRITE);
        i     <= @(posedge clk) i+1;
        write <= @(posedge clk) 0;
        //...
        @(posedge clk) #1;              // FALSCH: warum ohne enter_new_state?
        //...
    end
```

```
always
    begin
        if (70 > $random(k) % 100)
            begin
                if ((readack == 1)&&(empty==1))
                    begin
                        read <= @(posedge clk) 1;
                        @(posedge clk) enter_new_state(`C_RECEIVE);
                        @(posedge clk);             // FALSCH: darf so hier nicht vorkommen
                        @(posedge clk);             // FALSCH: darf so hier nicht vorkommen
                        dest[i] <=  @(posedge clk) rdata;
                        while ((readack == 1)&&(empty==1))
                            @(posedge clk) enter_new_state(`C_ACK1);
                        @(posedge clk) enter_new_state(`C_ACK0);
                        read <= @(posedge clk) 0;
                        i <=  @(posedge clk) i+1;
                    end
                else
                    @(posedge clk) enter_new_state(`C_WAIT);
            end
        else
            @(posedge clk) enter_new_state (`C_NO_ACCEPT);
    end
```

```
always @(posedge clk) // FALSCH: so darf keine ISE-Always-Schleife anfangen
    begin
        if (full == 0 && write == 0 && wrack == 0)
            begin
                write <= @(posedge clk) 1;
                wdata <= @(posedge clk) wdata + 1;
            end
        else if (wrack==1 && write==1)
            begin
                write <=@(posedge clk) 0;
            end
    end
```

```
always begin
    @(posedge clk)  enter_new_state(`IDLE);
    ready = 1;
    write <=  @(posedge clk) 0;
    if (pb == 1) begin
        @(posedge clk) enter_new_state(`ASK);
        wdata <=  @(posedge clk) $random;

        if (full == 0 & fready == 1) begin
            @(posedge clk) enter_new_state(`SEND);
            while (wrack == 1)
                @(posedge clk) enter_new_state(`WAIT);
            while (wrack == 0) begin
                @(posedge clk) enter_new_state(`TX);
                write <= @(posedge clk) 1;
            end
            write        <=  @(posedge clk) 0;         // FEHLER: zu welchem Zustand
            src[count1]  <=  @(posedge clk) wdata;     // FEHLER: gehören diese 3 Regisgter-
            count1       <=  @(posedge clk) count1 +1; // FEHLER: transferanweisungen

        end // if (full == 0 & fready == 1)
    end // if (pb == 1)
end // always begin
```

```
always
    begin
        @(posedge clk)  enter_new_state(`IDLE);
        ready = 1;
        read <=  @(posedge clk) 0;
        if (start_cons == 1)
            begin
                @(posedge clk) enter_new_state(`ASK);
                if (empty == 0 & f_ready == 1 )
                    begin
                        @(posedge clk) enter_new_state(`RES);
                        while (rdack == 1)
                            @(posedge clk) enter_new_state(`WAIT);
                        while (rdack == 0)
                            begin
                                @(posedge clk) enter_new_state(`RX);
                                rreg <= @(posedge clk) rdata;
                                read <= @(posedge clk) 1;
                            end

                        read <= @(posedge clk) 0; // FALSCH: zu welchem Zustand gehört diese
                                                  // FALSCH: Registertransferanweisung?
                        @(posedge clk) enter_new_state(`TEST);
                        dest    <= @(posedge clk) rreg;
                        count1  <= @(posedge clk) count1 + 1;
                    end // if not empty
            end // if start_cons == true
        end
```

4 Fragen

- Zeige mir in einem deiner Verilog-Programme den Unterschied in der Modellierung des Testbetts und der Modellierung mit impliziter Zustandskodierung.
- Zeige mir die Zeilen in deinem Programm, wo Zustandsübergänge festgelegt werden.
- Was macht der Task „enter_new_state“?
- Übersetze mir folgenden Always-Block, welcher ein Modell mit impliziter Zustandskodierung darstellt, in ein ASM-Diagramm.

5 Zusammenfassung

Dieses Dokument wurde gemacht, um immer wieder vorkommende stereotype Modellierungsfehler vermeiden zu helfen.