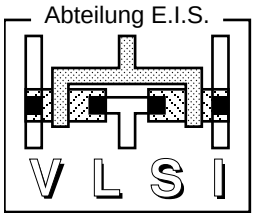
Technische Universität Braunschweig
Abteilung Entwurf integrierter
Schaltungen (E.I.S.)

Prof. Dr. Ulrich Golze
Hagen Gädke-Lütjens



Version 1.0 vom 06.09.2010

Leitfaden zum Praktikum *Adaptive Rechensysteme* WS 2010/11

Herzlich willkommen beim Praktikum über adaptive Rechensysteme. Dieser Leitfaden soll Ihnen den Einstieg in die Benutzung der Entwurfsumgebung und die Bearbeitung der eigentlichen Aufgaben erleichtern.

Der Leitfaden besteht aus zwei Teilen: Der erste beschreibt den Praktikumsablauf mit seinen einzelnen Phasen und Teilaufgaben, der zweite führt in den verwendeten Entwurfsfluss ein (der Ihnen zumindest bis hin zur RTL-Simulation aus den Übungen des letzten Semesters bekannt sein sollte).

Bitte lassen Sie uns wissen, wo Probleme auftreten, damit wir für nachfolgende Jahrgänge Abhilfe schaffen können.

Den Leitfaden finden Sie auch auf der Praktikumswebseite:
http://www.eis.cs.tu-bs.de/gaedke/acs-prak/

1 Praktikumsablauf

1.1 Erstes Treffen und Allgemeines

25.10.2009, 9:45 Uhr - 10:30 Uhr

Am Montag, dem 25.10.2010 um 9:45 Uhr findet im Raum M305 (Informatikzentrum) ein erstes gemeinsames Treffen aller Praktikumsteilnehmer statt. **Für diese Veranstaltung besteht Anwesenheitspflicht.**

Bei dem Treffen geht es vor allem um organisatorische Details wie Gruppeneinteilung, Übungs- und Kolloquiumstermine; außerdem wird der Praktikumsablauf vorgestellt. Abschließend erhalten Sie pro Gruppe je einen Schlüssel für den Praktikumsraum.

Allgemeine Hinweise für's Praktikum:

Der C-Anteil am Praktikum ist äußerst gering. Falls Sie noch gar keinen Kontakt mit der Progammiersprache C hatten, überfliegen Sie eine der zahllosen Einführungen, z. B.
`http://www.uni-giessen.de/hrz/software/programmiersprachen/C/c_teil1.html`, und zwar noch diese Woche, also *vor* Phase 1. Falls Sie trotzdem auf eine bestimmte Frage partout keine Lösung finden, wenden Sie sich vertrauensvoll an Ihren Hiwi.

Wiederholen Sie zu Beginn den Stoff über adaptive Rechner aus dem Skriptkapitel 14. Schauen Sie sich insbesondere auch die dort abgedruckten Quelltexte an (beides ebenfalls *diese Woche, vor Phase 1*). Sie werden beim Verilog-Code für den Master-Mode einige syntaktische Abweichungen zwischen Vorlesungsskript und Praktikumsquelltext finden (z. B. englische statt deutsche Kommentare, ein always-Block pro reg statt ein always-Block insgesamt); die Semantiken sind aber identisch.

Das Praktikum beginnt mit dem ersten Tag der Vorlesungszeit und umfasst maximal 14 Wochen. Nach Ende der Vorlesungszeit können keine Nacharbeiten mehr geleistet werden, d. h. wenn Sie dann mit den Aufgaben nicht fertig sind, kann Ihnen keine erfolgreiche Praktikumsteilnahme bescheinigt werden.

Das Praktikum besteht aus 11 Phasen. Am Ende jeder Phase geben Sie Ihre Lösung beim Hiwi ab. Lesen Sie vor jeder Abgabe die Aufgabenstellung der jeweiligen Phase nocheinmal durch, um auszuschließen, dass Teile in Ihrer Abgabe fehlen. Vom Hiwi werden nur vollständige und korrekte Abgaben anerkannt.

Nach Abschluss jeder der Phasen 2, 3, 6, 7, 10 erfolgt ein Kolloquium beim Assistenten. Die Kolloquiumstermine werden beim ersten Treffen festgelegt. Im Kolloquium müssen Sie (d. h. jedes Gruppenmitglied) in der Lage sein, alle Abgaben (inkl. des kompletten C-Programms und der in Verilog programmierten Hardwarebeschreibung) zu erläutern und Fragen dazu korrekt zu beantworten. Erfüllen Sie diese Leistung nicht, wird Ihnen keine erfolgreiche Praktikumsteilnahme bescheinigt.

Um das Durchsehen Ihrer Quelltext-Abgaben zu erleichtern, halten Sie sich bitte stets an folgende Vereinbarungen (bei Nicht-Einhaltung wird Ihre Abgabe nicht akzeptiert):

- Keine Tabulatoren verwenden.
- Sprechende Namen für Bezeichner.
- Kommentare dort einsetzen, wo man nicht auf Anhieb sieht, was passiert.
- Keine veralteten Kommentare.
- Konsistente Einrückungen.

Beim Hardware-Entwurf erleichtern Ihnen folgende Regeln, die Sie bitte ebenfalls befolgen (Testrahmen sind ausgenommen), die Arbeit:

- Bevor Sie ein Register (reg ...) definieren, machen Sie sich Gedanken darüber, was die Logiksynthese daraus erzeugen soll (FF oder C/L?).
- Verwenden Sie beim Schreiben an ein FF-Register nur die nicht-blockende Zuweisung (<=)!
- Verwenden Sie beim Schreiben an ein C/L-Register nur die blockende Zuweisung (=)!
- Platzieren Sie die nicht-blockende Zuweisung immer in (always)-Blöcken.
- Verwenden Sie nur positiv flankengetriggerte Flip-Flops (@posedge). Triggern Sie dabei nur auf Flanken von CLK und RESET. (Dadurch können Probleme beim Timing vermieden werden, und Ihr Design lässt sich leichter debuggen.)
- *Alle* Register müssen nach einem Reset auf definierte Werte gesetzt werden (if (RESET) begin ...).
- Ein Register darf nur in exakt einem always-Block Ziel einer Zuweisung sein.
- In jedem always-Block darf max. 1 FF-Register Ziel einer Zuweisung sein.
- Auf ein Register dürfen Sie nur dann an unterschiedlichen Stellen innerhalb eines always-Blockes nicht-blockend schreiben, wenn in der Simulation niemals zwei oder mehr dieser Schreibzugriffe gleichzeitig (zur selben Taktflanke) ausgeführt werden.
- Interne Tristate-Buffer (also das explizite Setzen eines Signals bzw. Registers auf den Wert z) sind verboten.
- Verilog **`register`**-Arrays dürfen nicht verwendet werden. Wenn Sie partout in Ihrem Entwurf größere Zwischenspeicher brauchen, halten Sie bitte mit dem betreuenden Assistenten Rücksprache.

Hinweise zum Thema Plagiarismus:

Fremde Code-Bibliotheken außer den vom Assistenten zur Verfügung gestellten dürfen Sie *nicht* verwenden. Zusammenarbeit über Gruppengrenzen hinweg ist in Form der Diskussion von Lösungs*ideen* erlaubt. Es dürfen aber *keine* Artefakte wie Programm-Code, Dokumentationsteile (Text, Zeichnungen, Messergebnisse) oder ähnliches ausgetauscht werden. Mit der Abgabe einer Lösung bestätigen Sie, dass Ihre Gruppe der alleinige Autor des gesamten Materials ist.

1.2 Phase 1: Einführung und Experimente (1 Woche)
25.10.-29.10.2010

In dieser Woche geht es darum, alle für's Praktikum nötigen Features der Entwurfsumgebung kennen zu lernen und mit dem ACE-M3 zu experimentieren. Bevor Sie beginnen, müssen Sie unbedingt das Vorlesungskapitel 14 (insbesondere 14.2, 14.3 und 14.4) durchgearbeitet haben!

1. Legen Sie mit **`mkslave`** ein neues Slave-Mode-Projekt an, und erproben Sie die verschiedenen in der Werkzeugeinführung (Kapitel 2) erklärten Arbeitsschritte. Sie sollten also die Funktionsfähigkeit der Anwendung sowohl in Simulation (RTL und Post-Layout) als auch in realer Hardware auf dem ACE-M3 erproben. Letzteres soll nicht nur durch Ausführen von **`./main`**, sondern auch interaktiv mit dem Werkzeug **`xmd`** erfolgen.
2. Machen Sie analoge Experimente mit einer durch **`mkmaster`** angelegten Master-Mode-Anwendung. Verzichten Sie hier aber auf den interaktiven Test mit **`xmd`**, sondern nehmen die reale Erprobung nur durch Starten von **`./main`** vor.
3. Erweitern Sie Ihre in 1. angelegte Slave-Mode-Anwendung auf die Spiegelung von 32-Bit-Worten, ähnlich zu Abschnitt 14.4.2.2 des CuSE1-Skriptes. Nehmen Sie die gleichen Simulationen und Tests vor. Um in **`main.c`** die Binärdaten ein- und auszulesen, verwenden Sie die Methoden **`fread`** und **`fwrite`**, ähnlich wie in **`/cad/tools/acs-prak/ACS-PRAK/TestData/brighten.c`**. Mit **`fread`** und **`fwrite`** lesen und schreiben Sie Daten binär, d. h. Sie müssen Ihre Memfile-Daten mit dem Tool **`mem2bin`** (siehe Abschnitt 2.4) in Binärdaten umwandeln.

4. Erweitern Sie auch Ihre in 2. angelegte Master-Mode-Anwendung auf die Spiegelung von 32-Bit-Worten. Erstellen Sie Testdaten mit einem Texteditor im `.mem` Format (siehe Abschnitt 2.4), die Sie dann während der Simulation in den simulierten Speicher einlesen. Schreiben Sie die Ausgabedaten mit `WriteMemFile` in eine Datei. Achten Sie darauf, an welche Zieladresse Sie die ReadMem-Daten schreiben und von welcher Adresse Sie sie in der Simulation lesen!

Abgaben: Führen Sie Ihrem Hiwi die laufenden Programme aus 3. und 4. auf dem ACE-M3 vor!

1.3 Phase 2: Messungen (2 Wochen)
01.11.-12.11.2010

Hier werden Sie die Slave-Mode Anwendung reverse, die Sie in der letzten Phase erstellt haben, um Messpunkte erweitern. Ziel ist es zu bestimmen, wie effizient der Datentransfer im Slave-Mode zwischen CPU und RC erfolgt. Dazu werden die maximalen und minimalen Zeiten zwischen zwei CPU-Zugriffen in der Hardware gemessen. Der Software-Teil muss diese Ergebnisse auslesen und dem Benutzer ausgeben. Gehen Sie dabei wie folgt vor:

1. Sie müssen den Hardware-Teil um zwei durch die Software lesbare Register erweitern. Im einen steht die minimale, im anderen die maximale Zeit (in Takten) zwischen zwei Zugriffen.
2. Die Zeit zwischen zwei Zugriffen von der Software auf die RC muss durch einen Hardware-Zähler in Takten gemessen werden.
3. Nach einem Zugriff müssen die minimalen und maximalen Werte mit dem gerade gestoppten Wert des Zählers aktualisiert werden.
4. An Zugriffen sollen Sie in drei Schleifen mit jeweils 10 Iterationen in der Software (d. h. in `main.c`, wobei Sie solche Schleifen zum Verifizieren der korrekten Funktionalität Ihrer Hardware bitte auch schon in `stimulus.v` erstellen) folgende Muster realisieren: Nur aufeinanderfolgende Lese-Operationen, nur aufeinanderfolgende Schreib-Operationen, abwechselnd je eine Lese- und eine Schreib-Operation.
5. Entwickeln Sie Hardware und Software so, dass Sie alle drei Messungen einerseits getrennt für jedes Zugriffsmuster, aber andererseits innerhalb *eines* Programmdurchlaufs durchführen können (das fertige Programm soll also nur einmal gestartet werden und dabei alle relevanten Ergebnisse ermitteln).
6. Simulieren Sie Ihren Entwurf auf RT-Ebene.
7. Passen Sie den Software-Teil so an, dass die gemessenen Werte von der RC zurückgelesen und dem Benutzer ausgegeben werden (C-Funktion `printf`, getrennt für jedes Zugriffsmuster).
8. **Achtung:** Ziel der Zeitmessung ist es, die reine Latenz der HW/SW-Kommunikation herauszufinden; verfälschen Sie also die Messwerte nicht mit davon unabhängigen SW-Latenzen (z. B. printf-Aufrufe oder Array-Zugriffe).
9. Compilieren Sie die gesamte Anwendung und erproben Sie `./main`.

Beachten Sie bei der Realisierung der Messungen folgende Details:

- Ihre Schaltung kann von der CPU über Einzelwort-Transfers wie auch über sogenannte Burst-Transfers angesprochen werden. In beiden Fällen beginnt ein neuer Transfer immer mit einer steigenden Flanke des ADDRESSED-Signals. Bei einem Burst-Transfer bleibt ADDRESSED über mehrere aufeinanderfolgende Takte gesetzt. (Gegebenenfalls wechselt währenddessen der Wert auf dem ADDRESS-Bus.) Für Ihre Zeitmessungen soll ein Burst-Transfer gleich welcher Länge aber nur als *ein* Zugriff gewertet werden.
- Sie sollen also bei deaktiviertem RESET-Signal die Zeit zwischen zwei aufeinanderfolgenden steigenden Flanken des ADDRESSED-Signals messen.

- Beachten Sie, dass eine Flankensteuerung (@posedge) nur für das Clock- und das Reset-Signal erlaubt ist.

Abgaben:

- Das erweiterte HDL-Modell `user.v`.
- Der angepasste Testrahmen `stimulus.v`.
- Das C-Programm `main.c`.
- Eine Textdatei, die eine Erläuterung Ihrer Messmethodik sowie Ihre Messergebnisse enthält.

An dieser Stelle nochmal kurz die wichtigsten Hinweise zur Abgabe.

Beachten Sie stets die Programmierregeln in Abschnitt 1.1. Dies hilft Ihnen bei der Fehlervermeidung und erleichtert uns das Verständnis Ihres Codes.

Zur Abgabe führen Sie das funktionierende Programm auf dem ACE-M3 sowie die funktionierende RTL-Simulation (je nach Phase kann eins der beiden wegfallen, dies wird dann in der jeweiligen Phase beschrieben) Ihrem Hiwi vor und zeigen ihm alle Dateien, die Sie abgeben werden. Waveforms werden als PDF-Dateien (erstellt mit Print... in VirSim wie in Abschnitt 2.6 beschrieben) dargestellt. Texte bitte als reine Text-Dateien (.txt). Nach positiver Begutachtung zippen Sie die Dateien als .tar.gz (keine anderen Archivformate) und schicken Ihrem Hiwi dieses Archiv per Email, mit dem Subject `Praktikum Gruppe N Phase M`, wobei `N` die Gruppennummer und `M` die Nummer der Phase ist.

Nach dieser Phase erfolgt ein Kolloquium.

1.4 Phase 3: Bildbearbeitung (1 Woche)
15.11.-19.11.2010

Als Kernaufgabe in diesem Praktikum werden wir uns mit einem einfachen Problem aus der Bildbearbeitung befassen. Es geht darum, den Kontrast in Graustufenbildern zu erhöhen. Solche Graustufenbilder werden auf dem Rechner als zweidimensionales Feld von Zahlen dargestellt, bei dem jede Zahl die Helligkeit des entsprechenden Bildpunktes angibt. In unserem Beispiel sind diese Werte 8 Bit breit, der Wert 0 entspricht dabei vollständiger Schwärze, der Wert 255 dem hellsten Weiß. Aus Vereinfachungsgründen gehen wir davon aus, dass alle Bilder 256 Bildpunkte breit und 256 Bildpunkte hoch sind, also insgesamt 65536 Bildpunkte enthalten.

Ein einfaches Beispielprogramm, das Ihnen den Umgang mit solchen Bildern näherbringen soll, finden Sie in der Datei `/cad/tools/acs-prak/ACS-PRAK/TestData/brighten.c`. Diese Anwendung hellt ein gegebenes Bild auf, indem auf alle Grauwerte der Wert 100 aufaddiert wird. Zur Erprobung kopieren Sie die Datei in eines ihrer Arbeitsverzeichnisse und übersetzen es mit dem Kommando `make brighten` (in einem Verzeichnis ohne Makefile). Ein Beispielbild `lena256.pgm` liegt ebenfalls im o. a. Verzeichnis. Durch das Kommando `xv lena256.pgm` können Sie es sich anzeigen lassen.
Mit der Anweisung `./brighten lena256.pgm lena256b.pgm` wird in der Datei `lena256b.pgm` eine hellere Version des Bildes erzeugt. Betrachten Sie auch dieses mit `xv`, und beurteilen Sie das Ergebnis der Aufhellung.

Aber zurück zu unserer Aufgabe: Es gibt eine Vielzahl von Algorithmen, die verwaschene Bilder aufbereiten können. Wir schauen uns hier den einfachsten an: Die Aufspreizung des Kontrasts, der folgende Idee zu Grunde liegt.

- Der dunkelste Punkt des Eingabebilds wird immer auf den Grauwert 0 (=schwarz) im Ausgabebild abgebildet (unabhängig von seinem Ursprungswert).
- Der hellste Punkt des Eingabebilds wird immer auf den Grauwert 255 (=weiß) im Ausgabebild abgebildet (auch hier unabhängig von seinem Ursprungswert).

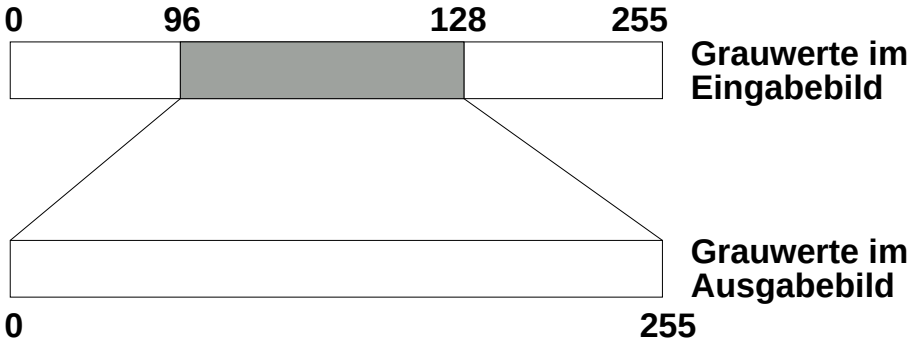


Abbildung 1.1: Idee der Kontrastspreizung

- Alle anderen Grauwerte des Eingabebilds zwischen diesen Minima und Maxima werden auf den Bereich zwischen den 0 und 255 in das Ausgabebild abgebildet. Genauer: es handelt sich (bis auf Rundungsfehler) um eine affine Abbildung des Urbilds $[min, max]$ in das Bild $[0, 255]$. Zur Erinnerung: jede affine, einstellige Funktion lässt sich schreiben als $f(x) = m * x + c$, mit Konstanten m und c.

Als Ergebnis erhält man also aus einem Eingabebild, das den vollen Wertebereich $0 \ldots 255$ nicht vollständig ausnutzt, ein Ausgabebild, das den ganzen Wertebereich verwendet und so einen verbesserten Kontrast hat.

Im Beispiel aus Abbildung 1.1 wird also der minimale Grauwert 96 des Eingangsbildes auf den dunkelsten Wert 0 des Ausgangsbildes abgebildet. Der hellste Grauwert 128 des Eingangsbildes wird im Ausgangsbild zu 255 (weiß). Die $128 - 96 + 1 = 33$ möglichen unterschiedlichen Grauwerte des Urbilds werden nun so gespreizt, dass Sie das volle Intervall von 0 bis 255 gleichmäßig ausfüllen. Auf diese Weise bekommen wir zwar nicht mehr unterschiedliche Graustufen ins Ausgangsbild, aber sie liegen weiter auseinander und sind somit besser voneinander zu unterscheiden (höherer Kontrast).

In dieser Phase des Praktikums sollen Sie auf der Basis eines Slave-Mode-Projektes ein reines Software-Programm für das ACE-M3 schreiben, das die oben beschriebene Kontrastspreizung realisiert. (Einen Bitstrom für das FPGA müssen Sie natürlich trotzdem generieren.) Sie können dabei die Ein- und Ausgabeoperationen aus **`brighten.c`** übernehmen. Verwenden Sie **`xv`**, um die Ergebnisse Ihrer Transformation auch graphisch betrachten zu können.

Wichtig:

- Verwenden Sie ausschließlich Integer-Arithmetik (keine floats), damit später die HW-Realisierung leichter fällt!
- Der dunkelste Grauwert muss auf 0, der hellste muss auf 255 abgebildet werden, und die Abbildung muss affin sein. Stellen Sie dies unbedingt sicher!
- Testen Sie Ihren Algorithmus an mehreren verschiedenen Bildern, auch an ganz extremen, die z. B. nur zwei verschiedene Grauwerte besitzen.
- Eigene PGM-Bilder können Sie erzeugen, indem Sie ein Graustufenbild der richtigen Größe mit Photoshop als RAW speichern und anschließend mit einem Text-Editor die 3 Zeilen im Header einfügen, die Sie auch in z. B. lena.pgm finden.

Sobald Ihr Programm zuverlässig funktioniert, rufen Sie sich die Architektur des ACE-M3 ins Gedächtnis, und überlegen Sie, welche Teile ihres Programms wie in Hardware ausgelagert werden sollten. Tun Sie dies einmal unter der Voraussetzung, dass es sich um eine Slave-Mode-Anwendung handeln soll, und einmal für eine Master-Mode-Anwendung. Halten Sie die Ergebnisse Ihrer Überlegungen schriftlich fest. Wichtige Punkte sind beispielsweise:

- Die Bitbreiten der verarbeiteten Daten.

- Die Hardware-Implementierung verschiedener Operatoren. So kann eine Multiplikation mit einer Zweierpotenz in Hardware einfach durch eine Links-Schiebeoperation realisiert werden.
- Ist Parallelverarbeitung möglich? Berücksichtigen Sie: selbst im Master-Mode können pro Takt maximal 32 Bit an Daten aus dem Speicher gelesen wie auch geschrieben werden!

Abgaben:

- `main.c`
- Eine Beschreibung Ihres Kontrastspreizungs-Algorithmus.
- Ergebnisse der Zeitmessung. Relevant ist die Zeit für die komplette Verarbeitung der Bilddaten, nicht aber das Laden des Originalbildes in den Hauptspeicher sowie das Schreiben des Ergebnisbildes vom Hauptspeicher auf die Festplatte. Achten Sie darauf, dass Sie keine für die Berechnung unwichtigen Dinge wie printfs mitmessen!
- Eine Diskussion der von Ihnen geplanten Hardware-Architektur, insbesondere: welche Daten wollen Sie im Slave- bzw. Master-Mode wann von wo nach wo bewegen, und wann finden wo welche Berechnungen statt? Ihre Vorstellungen werden wir im Kolloquium besprechen.
- (Hier natürlich keine Simulationen.)

Nach dieser Phase erfolgt ein Kolloquium.

1.5 Phase 4: IP-Blöcke (1 Woche)
22.11.-26.11.2010

Für die Hardware-Realisierung Ihres Algorithmus werden Sie einen Dividierer mit variablen Operanden benötigen. Dieser wird nicht automatisch bei der HDL-Synthese erzeugt und ist für dieses Praktikum auch im Entwurf zu aufwendig. Wie in der Praxis üblich, werden Sie also ein schon bestehendes Hardware-Modul, auch *IP-Block* genannt, in Ihren Entwurf einbinden.

In dieser Phase wird Ihnen ein Dividierer-Modul in der von Ihnen gewünschten Größe als Netzliste zur Verfügung gestellt werden (vgl. Übungen letztes Semester). Sie sollen es so in einen Verilog-Testrahmen einbinden, dass zwei variable Operandenregister dividiert werden und Quotient und Rest in Ergebnisregistern abgelegt werden.

Der Dividierer hat unabhängig von den Breiten oder den Datentypen (vorzeichenbehaftet oder -los) seiner Operanden folgende Schnittstelle:

dividend : *N*-Bit Eingang für den Dividenden.

divisor : *M*-Bit Eingang für den Divisor.

quot : *N*-Bit Ausgang für den Quotienten.

remd : *M*-Bit Ausgang für den Rest der Division.

clk : Takteingang.

ce : Bei einer '1' an diesem Eingang wird der Takt aktiviert, d.h. der Dividierer arbeitet.

rfd : Unbenutzt.

Die Schaltung ist derart gepipelined, dass zu jeder Taktflanke (vorausgesetzt, ce ist 1) ein neuer Satz Operanden an den Eingängen akzeptiert wird. Nach einer bestimmten Anzahl von Takten (Latenzzeit), die von den von Ihnen gewählten Parametern (*N* und *M*) abhängt, taucht das entsprechende Ergebnis dann an den beiden Ausgängen auf. Es ist also wichtig, die Ausgänge zum richtigen Zeitpunkt auszuwerten.

Beispiel: Der in Abb. 1.2 untersuchte Dividierer z. B. hat eine Latenz von 19 Takten. Das heißt, dass nach Anlegen von Eingangswerten zur Taktflanke 0 das Ergebnis nach der 19. Taktflanke an den Ausgängen zur Verfügung steht und mit der 20. Taktflanke in Register eingelesen werden kann. Hier werden hintereinander (aber pipeline-parallel) die Divisionen $1234/100$, $1234/50$ und $1234/25$ ausgeführt (alle Werte dezimal). In Slave-Mode-Zugriffen wird zunächst der Dividend (1234) und dann die Divisoren (100,50,25) von der Software auf die RC übertragen. Nach dem Übernehmen jedes Divisors können also jeweils 20 Takte später die Werte für die Quotienten (12,24,49) und die Reste (34,34,9) aus den Ausgängen `quot` und `remd` in Ihre entsprechenden Register übernommen werden. *Achtung:* das abgebildete Beispiel bestückt den Dividierer nur alle 2 Takte mit neuen Argumenten. Sie sollen jedoch *jeden* Takt einen neuen Satz Argumente an den Dividierer anlegen.

Gehen Sie zur Bearbeitung dieser Phase wie folgt vor:

1. Legen Sie ein neues Slave-Mode-Projekt für diese Phase an.
2. Lassen Sie sich von Ihrem Hiwi den gewünschten Dividierer mittels des Werkzeugs CoreGen erzeugen. Dabei werden Sie zwei Dateien erhalten `divider.edn` und `divider.v`. Letztere enthält die Moduldeklaration der Dividiererzelle; hier können Sie die Schnittstelle im Detail sehen. Kopieren Sie beide Dateien in das Projektverzeichnis.
3. Öffnen Sie `user.tcl` mit einem Texteditor und entkommentieren Sie die Platzhalter-Zeile `add_file` für Ihren Dividierer (führendes `#` löschen). Ändern Sie den Namen der Verilog-Datei ggf. auf den Dateinamen des tatsächlich von Ihnen verwendeten IP-Cores und speichern Sie `user.tcl` ab.
4. Instanzieren Sie Ihre Dividiererzelle in `user.v`.

Abbildung 1.2: Pipelined Division von 1234/100, 1234/25 und 1234/25

5. Erweitern Sie `user.v` um die Realisierung der schreibbaren Register für Dividenden und Divisoren und die lesbaren Register für Quotienten und Reste. Ziel ist es, dass Ihre Schaltung ähnlich zu Abb. 1.2 drei Divisionen gepipelined ausführen kann (d.h. zu jedem Takt werden neue Argumente an die Dividierer-Eingänge angelegt!). Außerdem müssen Sie die Adressdekodierung ergänzen.

6. Fügen Sie nun die Steuerung hinzu. Diese muss kontrollieren, *wann* genau die Ergebnisregister ihre Werte von den Dividiererausgängen übernehmen. Hier müssen Sie die Latenz beachten: Latchen Sie die Ausgangswerte zu früh, ist die Berechnung noch nicht abgeschlossen. Sind sie zu spät, überschreiben die durch die Pipeline nachrückenden Werte das gewünschte Ergebnis.

7. Zur Software-Schnittstelle: Sie können selbst bestimmen, wie genau und in welcher Reihenfolge Sie die Operanden zur HW transferieren sowie die Ergebnisse nach der Berechnung von der HW abholen. Wahrscheinlich müssen Sie zwischen dem Schreiben und Lesen in Software etwas Zeit vergehen lassen (die Rechenzeit des Dividierers). Hierzu reicht beispielsweise ein einfaches
`printf("Waiting ...\n");` aus, oder besser: eine for-Schleife (mit einer passenden Anzahl Iterationen), die nichts tut.

8. Simulieren Sie Ihr Verilog-Modell auf RT-Ebene.

9. Schreiben Sie einen kleinen Software-Testrahmen in `main.c`, in dem drei Argumenten-Paare im Slave-Mode zur Division an die RC übertragen werden. Die drei Quotienten und Reste sollen dann aus der Hardware ausgelesen und dem Benutzer angezeigt werden.

10. Führen Sie eine Post-Layout-Simulation mit `make laysim` durch. Tip: oft reicht es, mit der Layout-Simulation die korrekte Funktionalität nachzuweisen. Hierfür brauchen Sie keine grafische Oberfläche; es genügen geschickte Textausgaben per `$display` in `stimulus.v`. Eine Layout-Simulation ohne grafische Ausgabe führen Sie mit `make laysim_no_gui` durch.

11. Testen Sie Ihr Programm auf dem ACE-M3.

Abgaben:

- user.v
- stimulus.v
- main.c
- Kommentierte Waveforms (als PDF) der RTL-Simulation. Die Waveforms sollen mindestens zwei Situationen darstellen: das Anlegen der Argumente an den Dividierer sowie das Auslesen von Quotienten und Resten. *Kommentiert* bedeutet, dass Sie die relevanten Stellen im Waveform bitte nachträglich kenntlich machen. Da es nicht ganz einfach ist, ein PDF mit Annotationen zu versehen, dürfen Sie immer, wenn von kommentierten Waveforms die Rede ist, einen Screenshot machen und die Kommentare (farbige Umrandungen, Linien, Text, ...) mit einer Bildbearbeitungssoftware (z. B. Photoshop) einfügen. Speichern Sie die Grafik dann wieder als PDF ab.

1.6 Phase 5: Slave-Mode-Version (RTL) (1 Woche)
29.11.-03.12.2010

Nun realisieren Sie Ihre in Phase 3 geplante Hardware-Architektur als Slave-Mode-Variante und verwenden dazu den in der vorigen Phase erprobten IP-Core. In der fertigen Anwendung soll die CPU (hier simuliert durch `stimulus.v`) die zu verarbeitenden Daten an die RC übertragen, diese führt die Berechnung aus, und die CPU holt die Ergebnisse ab. *Achten Sie auf eine möglichst gute Parallelisierung der Anwendung!*

Simulieren Sie Ihren Entwurf auf RT-Ebene.

Hinweise:

- Falls Sie mehrere Schleifen in eine Slave-Mode RC auslagern wollen, packen Sie alle Funktionen in *einen* Hardware-Block, und wählen Sie mittels eines von der Software beschreibbaren Registers aus, welche Operation aktuell ausgeführt wird.
- Sie müssen außer **`user.v`** auch **`stimulus.v`** entsprechend anpassen: hier müssen die Bilddaten aus einer .mem-Datei in den simulierten Speicher geschrieben werden und nach erfolgter Berechnung aus dem Speicher in die Ergebnisdatei. Analog zu den in Abschnitt 2.4 beschriebenen Tools bin2mem und mem2bin können Sie auch pgm2mem und mem2pgm verwenden. Letztere lassen bei der Umwandlung ins Mem-Format den PGM-Header weg, bzw. fügen bei der Rückumwandlung den PGM-Header wieder ein.
- Achten Sie stets auf die Adressbereiche, in denen die Daten im simulierten Speicher liegen.
- Wir empfehlen Ihnen *dringend*, sich an die hier vorgeschlagene Lösung mittels Dividierer zu halten. Abweichungen dürfen nur nach Rücksprache und Genehmigung durch den betreuenden Assistenten erfolgen. Gleiches gilt für die restlichen Praktikumsphasen.

Abgaben:

- **`user.v`**
- **`stimulus.v`**
- Eine Beschreibung, wie Ihre Hardware funktioniert; insbesondere im Hinblick auf Parallelisierung der Berechnung.
- Kommentierte Waveforms der RTL-Simulation.
- Zum Testen verwendete .mem-Daten (Input- und zugehörige Output-Dateien).

1.7 Phase 6: Slave-Mode-Version (ACE-M3) (1 Woche)
06.12.-10.12.2010

Führen Sie eine Layout-Simulation durch (ohne GUI genügt) und erproben Sie Ihren Entwurf auf dem ACE-M3. Dazu müssen Sie natürlich **`main.c`** anpassen. Messen Sie (im C-Programm) die reine Ausführungszeit der Kontrastspreizung (also nicht das Lesen bzw. Schreiben der Bilddaten mit fread bzw. fwrite; für die Berechnungen irrelevante Befehle (z. B. printf's) sollen ebenfalls nicht mitgemessen werden).

Abgaben:

- **`user.v`**
- **`main.c`**
- Eine Beschreibung, wie Ihre kombinierte HW-SW-Lösung funktioniert.
- Ergebnisse der Zeitmessung.
- Zum Testen verwendete Bilddaten (Input- und zugehörige Output-Bilder als PGM).
- Den zum Testen benutzten FPGA-Bitstream **`Simple/system-xc2vp30.bit`**.

Nach dieser Phase erfolgt ein Kolloquium.

1.8 Phase 7: Flusskontrolle im Master-Mode (RTL) (2 Wochen)
13.12.2010-17.12.2010 und 03.01.2011-07.01.2011

Ziel dieser Phase ist es, einen längeren Quelldatenstrom aus dem Hauptspeicher durch eine Pipeline mit gegebener Latenz zu schleusen und an einen anderen Bereich im Hauptspeicher zu schreiben. Da es auch bei der gestreamten Kommunikation mit Speichern bei längeren Datensätzen zu Unterbrechungen kommt (z. B. weil im Speicher ein neuer Bereich adressiert werden muss, oder weil andere Teilnehmer den Bus zum Speicher blockieren), ist es zwingend erforderlich, dass wir uns mit einer Stream-Flusskontrolle beschäftigen.

Erzeugen Sie sich zunächst ein Standard-Master-Mode-Projekt. Kopieren Sie dann
`/cad/tools/acs-prak/ACS-PRAK/TestData/pipeline.v` in Ihr Projekt, und fügen Sie es ähnlich wie den Dividierer in `user.tcl` ein. `pipeline.v` ist ein Platzhalter für die Berechnungspipeline, die Sie in den späteren Phasen zur Kontrastspreizung einfügen werden, und hat folgende I/O-Ports:

- CLK
- RESET
- CE: Ein Clock-Enable-Signal. Nur zu den positiven Taktflanken, zu denen `CE = 1` ist, schaltet die Pipeline. Ansonsten bleiben ihre Register im Inneren unverändert, und sie nimmt keine neuen Lesendaten auf. (Der Dividierer verhält sich ebenso.)
- DATA_IN: Ein 32-bittiger Inputdatenbus.
- DATA_OUT: Ein 32-bittiger Outputdatenbus. Hier werden exakt die Daten, die über DATA_IN gelesen wurden, wieder ausgegeben, allerdings erst nach LATENCY Takten, wobei LATENCY ein Modulparameter ist.

Instanzieren Sie das Modul pipeline in `user.v`; setzen Sie LATENCY auf den Wert 10.

Machen Sie sich als nächstes detailliert mit der Flusskontrolle vertraut (siehe auch CuSE1-Skript, Bild 14.20 in Abschnitt 14.4.3.2). Im Folgenden steht [0] steht für den Lesestrom, [1] für den Schreibstrom. Die Flusskontroll sorgt dafür, dass bei einem Read-Stall (STREAM_STALL(0)=1) der Schreibstrom angehalten wird (STREAM_ENABLE(1)=0), dass umgekehrt bei einem Write-Stall (STREAM_STALL(1)=1) der Lesestrom angehalten wird (STREAM_ENABLE(0)=0), und dass bei einem Stall keine bereits gelesenen Daten verloren gehen, d.h. diese werden dann von der Flusskontrolle gepuffert.

Sie finden in `user.v` eine Instanz des Moduls flowcontrol. An dem Dateneingangsport ist momentan STREAM_READ angeschlossen, d. h. die vom Lesestrom gelesenen Daten werden sofort an die Flusskontrolle zum Schreiben weitergereicht. Sie müssen hier anstatt STREAM_READ den Datenoutput der pipeline anschließen. Die Daten vom Lesestrom sollen der Input für die pipeline sein.

Auch die bisher mit der flowcontrol verbundenen Flusskontrollsignale STREAM_ENABLE müssen in analoger Form aufgetrennt und Ihre eigene Schaltung eingefügt werden. Tip: eine Möglichkeit ist, nicht STREAM_ENABLE an die flowcontrol anzuschließen, sondern neue Signale STREAM_ENABLE_FC, und dann die Ausgangssignale STREAM_ENABLE unter Verwendung von STREAM_ENABLE_FC und anderen Bedingungen zu berechnen.

Das zu verwendende Protokoll für die Kontrollsignale sieht wie folgt aus.

STREAM_ENABLE : Dieser Eingang dient zur Steuerung des Datenstroms. Beim Lesestrom bedeutet eine '1' auf diesem Port, dass Ihre Schaltung mit der *übernächsten* positiven Taktflanke ein Eingangsdatum von STREAM_READ in ein Register übernehmen möchte. Beim Ausgangsstrom bedeutet die '1', dass Ihre Schaltung gültige Daten schreiben möchte, die mit der nächsten positiven Taktflanke in den Ausgangsstrom übernommen werden. Eine '0' zeigt entsprechend an, dass zur übernächsten bzw. nächsten positiven Taktflanke keine neuen Daten gelesen bzw. geschrieben werden sollen.

STREAM_STALL : Eine '1' auf diesem Ausgang zeigt an, dass die Benutzerschaltung ggf. zwar Daten lesen bzw. schreiben möchte (STREAM_ENABLE=1), aber der Strom leider unterbrochen ist. Beim Lesestrom bedeutet dies, dass Daten nur zu solchen positiven Taktflanken von Ihrer Schaltung übernommen werden dürfen, wenn zur *vorherigen* positiven Taktflanke STREAM_STALL=0 war. Im anderen Fall

muss Ihre Schaltung warten. Beim Schreibstrom wird nur bei einer positiven Taktflanke das Datum tatsächlich übernommen, wenn zur selben Taktflanke STREAM_STALL=0 ist. Wenn das Signal '1' ist, muss Ihre Schaltung das Ausgangsdatum solange stabil an den Schreibstrom anlegen, bis die Übernahme tatsächlich erfolgt ist. Anderenfalls geht das Datum einfach verloren.

Zusammengefasst kann man sagen, dass Ihre Schaltung zur nächsten positiven Taktflanke ein Datum von STREAM_READ lesen muss, wenn *vor* der gerade vergangenen Taktflanke STREAM_ENABLE(0)=1 und STREAM_STALL(0)=0 waren.

Ein Datum wird hingegen genau dann zur nächsten positiven Taktflanke geschrieben, wenn seit der gerade vergangenen positiven Taktflanke **`STREAM_ENABLE[1]=1`** und **`STREAM_STALL[1]=0`** sind.

Noch einige Hinweise zum Umgang mit den Streams:

- Sie dürfen den Schreibstrom erst starten, wenn tatsächlich Ergebnisse am Ausgang der Pipeline vorliegen.
- Sie dürfen den Schreibstrom erst anhalten, wenn tatsächlich alle Daten aus der Pipeline erfolgreich geschrieben wurden.
- Sie müssen die Pipeline anhalten (ce=0), wenn der Schreibstrom abreißt. (Der Lesestrom wird von der Flowcontrol automatisch angehalten.)
- Sie müssen die Pipeline und den Schreibstrom anhalten, wenn der Lesestrom abreißt. (Der Schreibstrom wird von der Flowcontrol automatisch angehalten.)

Die Systemsimulation provoziert solche Stromunterbrechungen künstlich. Sie können also das Verhalten Ihrer Anwendung schon zur Simulationszeit untersuchen. Dazu müssen Sie natürlich einen Datenstrom durch Ihre Schaltung schicken, der lang genug ist. Benutzen Sie also schon für die RTL-Simulation einen Datensatz, der genauso viele Bilddaten umfasst wie ein in Phase 3 betrachtetes PGM-Bild. Damit das Debugging der Flusskontrolle leichter fällt, empfehlen wir Ihnen, den Datensatz **`/cad/tools/acs-prak/ACS-PRAK/TestData/numbers.mem`** zu verwenden. Solche Test-Datensätze können Sie sich auch leicht unter Zuhilfenahme eines einfachen C-Programms (z. B. **`creatememe`**, Quellcode: **`/cad/tools/acs-prak/ACS-PRAK/Src/createmem.c`**), selbst erstellen. Sie können Unterschiede zwischen der Eingabe-Mem-Datei und der Ausgabe-Mem-Datei sehr leicht feststellen: ersetzen Sie in der Ausgabe-Mem-Datei die Adresse in der ersten Zeilen durch die Adresse, die in der Eingabe-Mem-Datei steht, und führen Sie anschließend das Kommando **`cmp <file1> <file2>`** aus. Falls keine Ausgabe erscheint, sind die Dateien identisch. Ansonsten wird die Zeile des ersten gefundenen Unterschieds angezeigt.

Achtung: Diese Phase ist, zusammen mit Phase 8) erfahrungsgemäß die schwierigste im ganzen Praktikum. Wenn Sie nach einer Woche merken, dass Sie nicht voran kommen oder sich im Kreis drehen, sprechen Sie *rechtzeitig* (und nicht erst kurz vor Abgabetermin) die Hiwis oder ggf. den Assistenten an.

Abgaben:

- **`user.v`**
- **`stimulus.v`**
- Eine textuelle Beschreibung Ihrer Änderungen an der Flusskontrolle.
- Eine textuelle Beschreibung Ihrer Änderungen an **`user.v`**.
- Vier kommentierte RTL-Waveforms, die den Beginn und das Ende eines Read- sowie eines Write-Stalls erklären.
- Zum Testen verwendete .mem-Daten (Input- und zugehörige Output-Dateien).

Nach dieser Phase erfolgt ein Kolloquium.

1.9 Phase 8: Flusskontrolle im Master-Mode (ACE-M3) (1 Woche)
10.01.2011-14.01.2011

Testen Sie nun Ihr Design auf dem ACE-M3! Verwenden Sie zum Test ein PGM-Bild; Ihre Hardware muss dieses Bild ohne eine einzige Veränderung im Zielspeicher ablegen.

Falls Ihnen beim Starten des Linux auf dem ACE-M3 der Kernel abstürzt, liegt das daran, dass Logiken im gerouteten Design einen zu langen kritischen Pfad haben.

Um dies zu beheben, sollten Sie komplizierte Logiken in `user.v` vereinfachen (z. B. durch Vorziehung einer Berechnung in den vorherigen Takt, oder durch Pipelining), und die Korrektheit des entstandenen Designs per RTL-Simulation nachweisen. Vor allem die Logik zur Berechnung von `STREAM_ENABLE` muss möglichst einfach sein.

Abgaben:

- `user.v`
- `stimulus.v`
- `main.c`
- Eine textuelle Beschreibung Ihrer Änderungen gegenüber Phase 7, falls Änderungen nötig waren.
- Zum Testen verwendete Bilddaten (Input- und zugehörige Output-Bilder als PGM).
- Den zum Testen benutzten FPGA-Bitstream `Simple/system-xc2vp30.bit`.

1.10 Phase 9: Master-Mode-Version (RTL) (1 Woche)
17.01.2011-21.01.2011 - ***entfällt für 3SWS-Teilnehmer***

Entwickeln Sie nun auf Basis Ihrer Lösung aus Phase 8 eine Konstrastspreizungslösung für den Master-Mode. Re-usen Sie auch hier den in Phase 4 getesteten IP-Core.

Verifizieren Sie Ihren Entwurf per RTL-Simulation.

Abgaben:

- `user.v`
- `stimulus.v`
- Eine textuelle Beschreibung Ihres Designs, auch im Hinblick auf die im Kolloquium zu Phase 3 besprochenen Optimierungen durch Parallelisierung.
- Zum Testen verwendete .mem-Daten (Input- und zugehörige Output-Dateien).

1.11 Phase 10: Master-Mode-Version (ACE-M3) (1 Woche)
24.01.2011-28.01.2011 - ***entfällt für 3SWS-Teilnehmer***

Testen Sie nun Ihr Design aus Phase 9 auf dem ACE-M3, und führen Sie eine Zeitmessung analog zu Phase 6 durch.

Abgaben:

- `user.v`
- `stimulus.v`
- `main.c`

- Eine Beschreibung der Änderungen gegenüber Phase 9, falls Änderungen nötig waren.
- Zum Testen verwendete Bilddaten (Input- und zugehörige Output-Bilder als PGM).
- Ergebnisse der Zeitmessung (analog zu Phase 6).
- Den zum Testen benutzten FPGA-Bitstream `Simple/system-xc2vp30.bit`.

Nach dieser Phase erfolgt ein Kolloquium.

1.12 Phase 11: Endabgabe (1 Woche)

31.01.2011-04.02.2011 - *entfällt für 3SWS-Teilnehmer*

Hier sollen Sie den Praktikumsverlauf insgesamt (alle Phasen) in einem PDF-Dokument zusammenfassen. Bei dieser Abgabe ist mit einem Gesamtumfang von ca. 15-20 Seiten zu rechnen, bei einfachem Zeilenabstand, 12 Punkt, ohne redundante Beschreibungen. Ihr Dokument soll folgendes enthalten:

- Aufgabenstellungen.
- Überlegungen zu Ihren Lösungen.
- Beschreibung der jeweiligen Lösungen.
- Kommentierte Waveforms.
- Messergebnisse - einerseits die aus Phase 2, andererseits die der Kontrastspreizung, letztere auch im Vergleich Software/Slave/Master, idealerweise tabellarisch und als Diagramm.
- Deuten Sie die Messergebnisse, d. h. bringen Sie Argumente, warum die Ergebnisse so ausfallen, wie sie ausfallen.

Abgaben:

- Das Endabgabe-PDF-Dokument.

2 Entwicklungsumgebung

Einen Einstieg in unsere Entwicklungstools für adaptive Rechner hatten Sie bereits in der zweiten Hälfte des vorigen Semesters, bis hin zur RTL-Simulation von Slave-Mode-Projekten. Hier stellen wir Ihnen nun auch alle übrigen Features unserer Tools vor, damit auch dem Entwurf von Master-Mode-Projekten, der Synthese und dem Test auf dem ACE-M3 nichts mehr im Wege steht.

2.1 Starten der VM

Der Praktikumsumgebung liegt ein Makefile-basierter Flow unter Linux zugrunde. Als erstes muss auf Ihrem Übungsrechner daher Linux gestartet werden:

Start | Alle Programme | VMware | VMware Player

Das zu ladende Linux-Image befindet sich auf der lokalen Scratch-Partition:

d:/scratch/ACS/Ubuntu/Ubuntu.vmx

Die Login-Daten erfahren Sie von Ihrem Hiwi. (In den meisten Fällen arbeiten Sie auf ihrem bekannten Account aus dem letzten Semester weiter.)

2.2 VM-Konsole und Home-Bereiche

Sobald Sie den Linux-Desktop vor sich sehen, minimieren Sie das VMware-Tools-Fenster (nicht schließen) und starten eine Konsole (Doppelklick auf „Terminal"). Falls Ihnen die Unix-typischen Konsolenbefehle noch fremd sind, hier eine kurze Übersicht über die für Sie wichtigsten Befehle:

ls	Listet den Inhalt des aktuellen Verzeichnisses auf.
cd <Verzeichnis>	Wechselt in das Verzeichnis <Verzeichnis>. Mit **cd ..** wechseln Sie in das übergeordnete Verzeichnis. Mit **cd $HOME** wechseln Sie in Ihr VM-Home-Verzeichnis.
rm <Datei>	Löscht <Datei>.
rm -r <Datei/Verzeichnis>	Löscht die angegebene Datei bzw. das angegebene Verzeichnis.
nedit <Datei> &	Öffnet <Datei> in einem Texteditor.
evince <Datei> &	Öffnet <Datei> in einem PDF-Viewer.
tar czvf <neue Datei>.tar.gz <Datei/Verzeichnis>	Komprimiert <Datei/Verzeichnis> in <neue Datei>.tar.gz. (<Datei/Verzeichnis> bleibt dabei unverändert.)
tar xzvf <Datei>.tar.gz	Entpackt <Datei>.tar.gz ins aktuelle Verzeichnis. (<Datei>.tar.gz bleibt dabei unverändert.)
du -s -h <Verzeichnis>	Zeigt, an wieviel Speicherplatz <Verzeichnis> inkl. aller Unterverzeichnisse benötigt.
df -h <Verzeichnis>	Zeigt an, wieviel Speicherplatz in <Verzeichnis> noch frei ist.

Sie befinden sich aktuell in Ihrem VM-Home-Verzeichnis. Beachten Sie: dieses Verzeichnis ist zwar gut geeignet für temporäre Arbeiten (schnelle Daten-/Dateizugriffe), aber:

- es ist nur auf dem Praktikumsrechner sichtbar, auf dem Sie gerade arbeiten,
- es unterliegt nicht der institutsinternen Datensicherung, und
- sein Inhalt verschwindet bei jedem Auswecheln des VM-Images.

Wenn Sie das VM-Home-Verzeichnis wegen der Geschwindigkeitsvorteile zum Arbeiten benutzen möchten, achten Sie bitte darauf, alle Ihre Daten vor dem Ausloggen in Ihr EIS-Home-Verzeichnis (eis-home, identisch mit dem unter Windows gemounteten Laufwerk U:) zu verschieben, damit

- Ihre Daten gesichtert werden, sowie damit
- Ihre Daten vor den Einblicken anderer geschützt bleiben.

Falls Ihnen im Verlauf des Praktikums der Speicherplatz in Ihrem EIS-Home ausgeht, gehen Sie wie folgt vor (bitte die Reihenfolge einhalten):

1. Löschen Sie alte, unnötige Dateien.
2. Falls Sie die generierten Projektdateien aus älteren Projekten nicht mehr brauchen, führen Sie dort jeweils ein `make clean` aus.
3. Komprimieren Sie Verzeichnisse, die Sie länger nicht benötigen (s. o.).
4. Kontaktieren Sie Hagen Gädke-Lütjens.

2.3 Anlegen von neuen Projekten

Zum schnellen Start in die Arbeit können Sie auf bereits lauffähige Musteranwendungen zurückgreifen. Durch ein einzelnes Kommando wird ein Unterverzeichnis angelegt und mit allen nötigen Dateien versehen. Sie müssen dann lediglich Ihre Änderungen an den passenden Stellen einbauen.

Das zu verwendende Kommando unterscheidet sich nach dem Typ der zu erstellenden Anwendung: Für die Slave-Mode Betriebsart verwenden Sie das Ihnen bereits bekannte Kommando `mkslave`, für Master-Mode das Kommando `mkmaster`. In beiden Fällen folgt dem Kommando der von Ihnen gewünschte Name für das anzulegende Projekt.

Beispiel: Mit dem Kommando `mkslave simsel` wird im aktuellen Verzeichnis ein Unterverzeichnis namens `simsel` angelegt. In diesem befinden sich alle für eine Slave-Mode-Anwendung nötigen Dateien. Die Beispielanwendung realisiert ein einzelnes 32b-Register auf der RCU, das durch die CPU geschrieben und wieder ausgelesen werden kann. Nach einem Reset des Systems hat das Register den schon bekannten, leicht wiedererkennbaren Wert 0xDEADBEEF. Andere Teile des RCU-Speicherbereiches, die nicht dieses Register enthalten, liefern beim Lesen den Wert 0xC0FFEE11. Sie kann durch entsprechende Ergänzung leicht an die tatsächlichen Erfordernisse Ihres Entwurfs angepasst werden. Dazu bearbeiten Sie lediglich drei im folgenden Abschnitt beschriebene Dateien. Die Master-Mode Beispielanwendung kopiert einen Speicherbereich durch die RCU auf einen anderen, die CPU ist also beim eigentlichen Kopiervorgang nicht involviert.

2.4 Dateistruktur

Beim Anlegen eines Slave-Mode-Projektes erhalten Sie für Ihre ersten Experimente im wesentlichen drei interessante Dateien:

`user.v` ist die Beschreibung der Slave-Mode-RCU in Verilog. Hier erkennt man die typische Slave-Mode-Schnittstelle sowie die eigentliche Anwendung im Rumpf des Moduls. Die Beispielanwendung erlaubt den Zugriff auf bis zu vier unterschiedliche Register (Dekodierung der letzten beiden Bits der

Wortadresse ADDRESS, also die Möglichkeiten 00, 01, 10 und 11). Davon ist momentan nur die Teiladresse 00 belegt (hier wird das Register outreg an die CPU ausgegeben). In den drei anderen Fällen wird die gut erkennbare Konstante 0xC0FFEE11 ausgegeben. Das Register outreg wird im always-Block auf 0xDEADBEEF zurückgesetzt. Bei Schreibzugriffen auf die Register-Adresse 00 übernimmt es den von der CPU an den Eingabe-Bus DATAIN angelegten Wert. Die Master-Mode-Version enthält zusätzlich noch die Schnittstelle für die MARC-Streams (siehe Vorlesung, Kapitel 14.4.3.1).

`main.c` ist der Software-Teil der Anwendung, der auf der PowerPC-CPU des ML310 ACS ausgeführt wird. Nach den üblichen Vorbereitungen (Initialisierung, bestimmen der Basisadresse des RCU-Bereiches) wird der 32b-Wert an der RCU-Wortadresse 0 (rcu ist als Zeiger auf unsigned long, also auf 32b Werte deklariert) gelesen und ausgegeben. Dies führt also zu einem Zugriff auf das RCU-Register outreg. Danach wird ein Schreibzugriff auf die gleiche Adresse vorgenommen, gefolgt von einem Auslesen des neuen Wertes.

`stimulus.v` Zum Testen der RCU in der Simulation müssen die Zugriffe der CPU auf Adressen im RCU-Speicherbereich nachgeahmt werden. Dazu können in dieser Datei vordefinierte Verilog-Funktionen aufgerufen werden. Solche Funktionen stehen für das Starten und Herunterfahren der Simulationsumgebung ebenso bereit, wie für das Nachahmen von Lese- (via Read32) und Schreibzugriffen (via Write32). Beide Funktionen akzeptieren als ersten Parameter die *CPU*-Adresse für den Zugriff. Zum Test der RCU muss hier als Basisadresse des RCU-Adressbereiches die Konstante SLAVE_BASE angegeben werden. Dazu relativ kann nun die Adresse *innerhalb* des *RCU*-Adressraums angegeben werden. Die Funktion Write32 erwartet als zweiten Parameter den zu schreibenden 32b-Wert, die Funktion Read32 liest einen 32b-Wert von der RCU in ein als zweiten Parameter übergebenes Register (32b breit, im Beispiel heisst das Register data). Die Systemfunktion $display arbeitet ähnlich wie printf in C, indem sie einen Wert entsprechend der Formatangabe (hier: %h steht für hexadezimale Darstellung) als Text auf der Simulatorkonsole ausgibt.

Zur Simulation von Master-Mode-Anwendungen können in **`stimulus.v`** zwei weitere Kommandos zum Umgang mit dem simulierten Speicher verwendet werden. Mit dem Kommando

```
ReadMemFile("infile.mem")
```

wird der Inhalt der Datei **`infile.mem`** zur Simulationszeit in den Speicher geschrieben. Die Eingabedatei (hier **`infile.mem`**) hat folgendes Format: Die Kopfzeile enthält die Adresse des ersten Bytes und die Anzahl der folgenden 32b Worte. Nun folgen die vorher angegebene Anzahl von 32b Worten, eines pro Zeile. Dann ist die Datei zu Ende, oder es folgt eine weitere Kopfzeile. Alle Zahlen werden hexadezimal dargestellt. Eine Beispieldatei **`infile.mem`** könnte wie folgt aussehen:

```
1000 3
12345678
87654321
deadbeef
2000 2
10101010
01010101
```

Nach **`ReadMemFile("infile.mem")`** würde auf Adresse 4096 (dezimal) das Wort 0x12345678 beginnen, auf Adresse 4100 das Wort 0x87654321, auf Adresse 4104 das Wort 0xDEADBEEF. Der zweite Block weist Adresse 8192 das Wort 0x10101010 und Adresse 8196 das Wort 0x01010101 zu. Man beachte hier, dass alle Adressen als Byte-Adressen angegeben sind und ein 32b Wort vier Bytes an Speicherplatz benötigt.

Um einen Speicherauszug des simulierten Speichers in eine Datei zu schreiben kann das Kommando **`WriteMemFile("outfile.mem", 32'h1000, 3)`** verwendet werden. Mit den hier gezeigten Parametern werden drei 32b Worte beginnend bei Byte-Adresse 4096 (dezimal) in die Datei **`outfile.mem`** geschrieben. An das vorige Beispiel anschließend hätte diese dann den folgenden Inhalt:

```
1000 3
12345678
87654321
deadbeef
```

Um bestehende Dateien nach und von diesem Format zu wandeln stehen zwei Hilfsprogramme bereit. Bei Eingabe von `bin2mem <lena256.pgm >lena256.mem` auf Unix Kommandoebene wird die Graustufenbilddatei `lena256.pgm` als hexadezimaler Speicherauszug in die Datei `lena256.mem` geschrieben. Wichtig: Die Kopfzeile (Startadresse und Anzahl von 32b Worten) fehlt noch und muss *manuell* mit einem Texteditor in der Datei `lena256.mem` nachgetragen werden. Der umgekehrte Schritt ist mit `mem2bin <lena256contrast.mem >lena256contrast.pgm` möglich. Hier sind keine manuellen Schritte mehr nötig. `lena256contrast.pgm` enthält genau die Daten aus `lena256contrast.mem`, die Kopfzeile wurde automatisch entfernt. `mem2bin` ist auf die Bearbeitung von Eingabedateien beschränkt, die nur einen Speicherbereich enthalten.

2.5 Simulation auf Registertransferebene (RTL)

Nach dem Sichten der Eingabedateien können Sie die Funktion der Slave-Mode-RCU im Verilog-Simulator erproben. Dabei werden die in der Stimulus-Datei beschriebenen Zugriffe auf die RCU ausgeführt und währenddessen verschiedene Signale der RCU aufgezeichnet. Diese Signale können bei Ende der Simulation graphisch in Form von Signalverlaufsdiagrammen (*waveforms*) dargestellt werden. Gegenüber den ebenfalls möglichen Textausgaben auf der Simulatorkonsole haben die Waveforms der Vorteil, dass hier leichter zeitliche Zusammenhänge zwischen parallelen Signalverläufen erkannt werden können.

Das Linux-Kommando

```
make rtlsim
```

übersetzt die Verilog-Beschreibungen für die Simulation. Sollten hierbei keine Fehler aufgetreten sein (diese würden den Vorgang abbrechen und müssten erst in den Quelldateien behoben werden), wird das Visualisierungswerkzeug für die Signaldiagramme gestartet (siehe Abbildung 2.1).

Man könnte die Simulation schon jetzt starten, würde dann aber nur die Textausgaben durch die $display-Aufrufe im Verilog auf der Simulatorkonsole sehen. In der Regel ist man aber eher an Waveforms interessiert. Dazu muss dem Simulator *vor* der Simulation mitgeteilt werden, welche Signale tatsächlich aufgezeichnet werden sollen. Der Einfachheit halber werden wir den Simulator anweisen, *alle* Signale unserer im Verilog-Modul user definierten Slave-Mode RCU anzuzeigen. Dies schließt sowohl die Schnittstelle zum Restsystem als auch eventuelle interne Register ein. Um diese Auswahl vorzunehmen, lassen wir uns zunächst die komplette Schaltungshierarchie (Klick auf entsprechendes Icon aus Abbildung 2.1) sowie ein leeres Waveform-Fenster öffnen (ebenfalls in entsprechendes Icon klicken). Dieser Zustand ist in Abbildung 2.2 gezeigt.

Im Hierarchy-Abschnitt des Hierarchie-Fensters öffnet man jetzt Hierarchie-Stufen durch Klicken auf das +-Icon, bis das Modul USER im Pfad

```
testbench_rtl.SYSTEM.plb_ddr_0.\plb_ddr_0/PLB_MARC_I.USER
```

sichtbar wird. Mit gehaltener **mittlerer** Maustaste ziehen wir nun das Modul USER auf den leeren grauen Bereich im linken Teil des Waveform-Fensters. Der Cursor ändert bei einem akzeptablen Abwurfpunkt die Form und wird grün. Nach dem Loslassen der mittleren Maustaste tauchen nun die Signalnamen im Waveform-Fenster auf. Die Waveforms selber sind grau (Verilog-Wert X = undefiniert).

Nun kann die Simulation durch Klicken des Rechtspfeil-Icons (gezeigt in Abbildung 2.1) gestartet werden. Da auch die vergleichsweise schnelle RTL-Simulation hier ein komplettes System inklusive Speicher-Controller und verschiedener On-Chip-Busse zu bearbeiten hat, dauert es einige Minuten, bis sie durchgelaufen ist. Der Abschluss der Simulation wird durch eine Konsolen-Meldung ähnlich zu

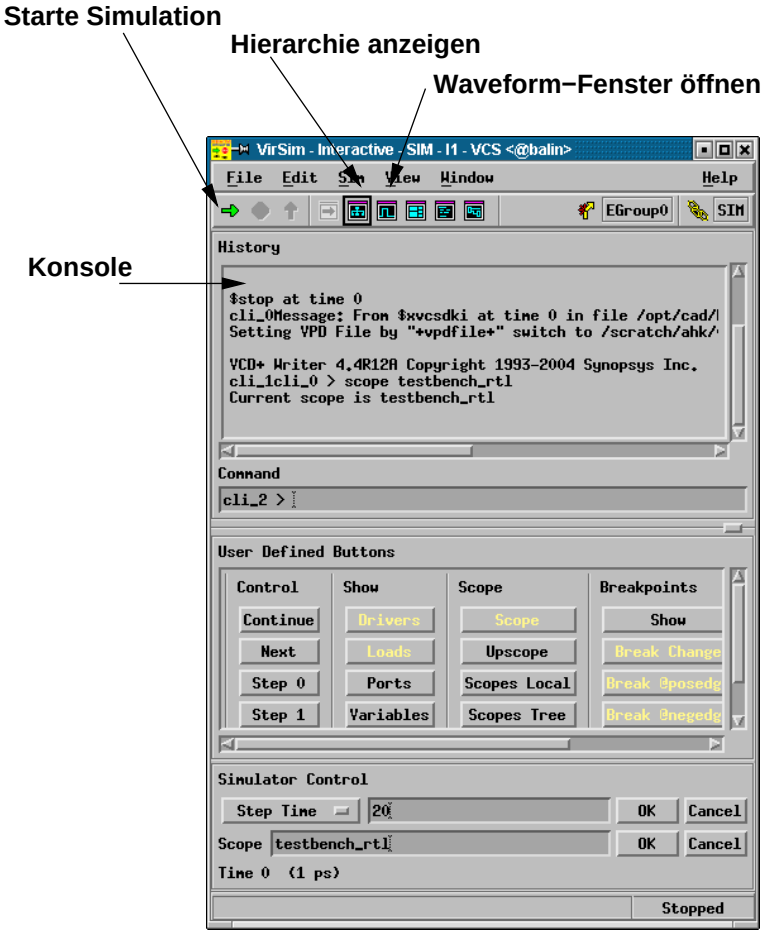


Abbildung 2.1: Graphische Simulationsumgebung VirSim

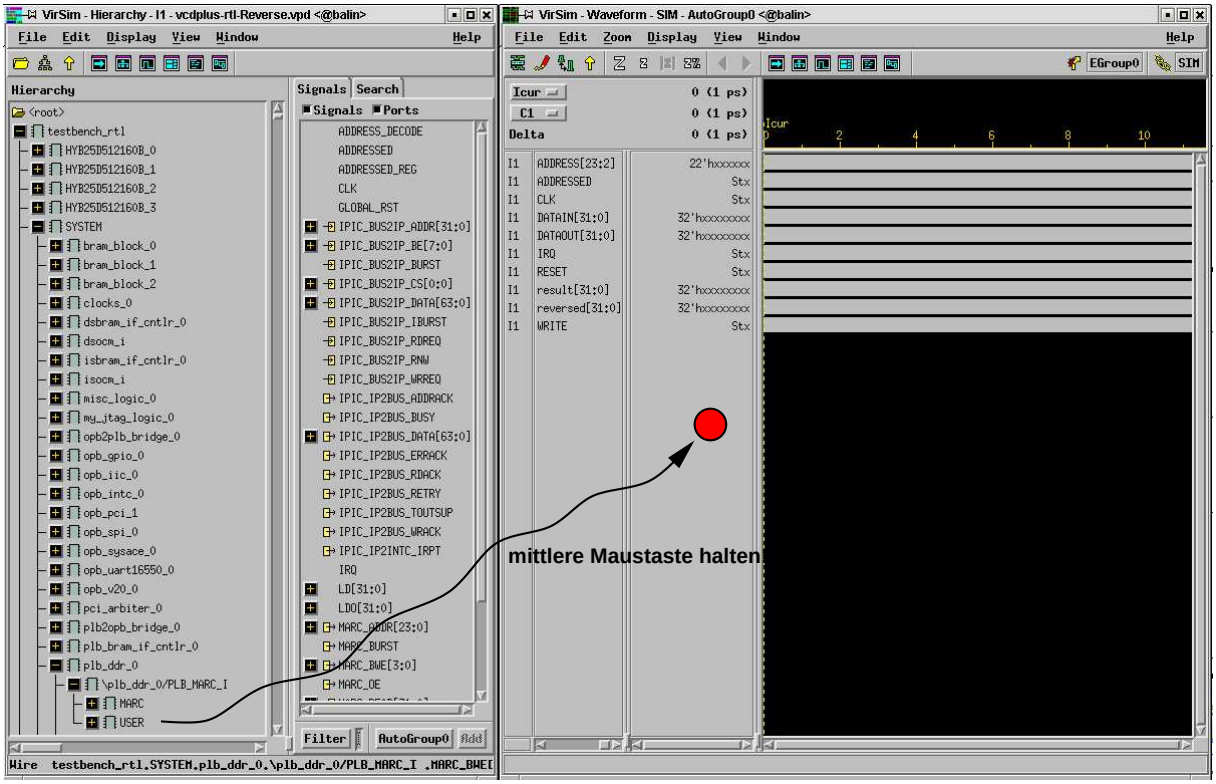


Abbildung 2.2: Auswahl der Signale zur Waveform-Anzeige

```
Register 0: deadbeef

Register 0: 87654321

$finish at simulation time            206240000
           V C S   S i m u l a t i o n   R e p o r t
Time: 206240000 ps
CPU Time:     39.160 seconds;       Data structure size: 520.2Mb
Thu Feb  1 00:02:13 2007
<######## Simulator is Terminated ########>
```

angezeigt. Sie entdecken hier auch die beiden Meldungen, die Ihre $display-Aufrufe in **stimulus.v** produziert haben (ggf. im Konsolenbereich einige Zeilen nach oben scrollen). Hier wird zuerst der korrekte Wert beim Lesen des Initialzustands (0xDEADBEEF) von outreg ausgegeben. Darauf folgt dann die Ausgabe des neuen Wertes (0x87654321) von outreg nach dem Schreibzugriff.

Scrollen Sie nun *horizontal* mit dem Rollbalken für die Zeitachse ans rechte Ende des Waveform-Fensters und verkleinern Sie den Maßstab solange durch Klicken auf das kleine Zoom-Out-Icon (kleines z-Symbol) der Menüleiste, bis Sie für das Taktsignal CLK eine ganze Reihen von Taktflanken erkennen können. Machen Sie nun die drei Zugriffe (Lesen, Schreiben, Lesen) ausfindig. Achten Sie jeweils auf die Korrelation zwischen den Signalen CLK, ADDRESSED und WRITE: Wenn bei einer *steigenden* Flanke von CLK das Signal ADDRESSED auf 1 liegt, liegt ein Zugriff der CPU (hier vertreten durch die Stimulus-Datei) auf die RCU vor. Wenn zu dieser steigenden CLK-Flanke nun WRITE den Werte 1 hat, liegt ein Schreibzugriff vor, sonst ein Lesezugriff. Hinweis: Die Formulierung “zu dieser steigenden Flanke” bedeutet, dass das betreffende Signal unmittelbar *vor* der steigenden Flanke den entsprechenden Wert hat.

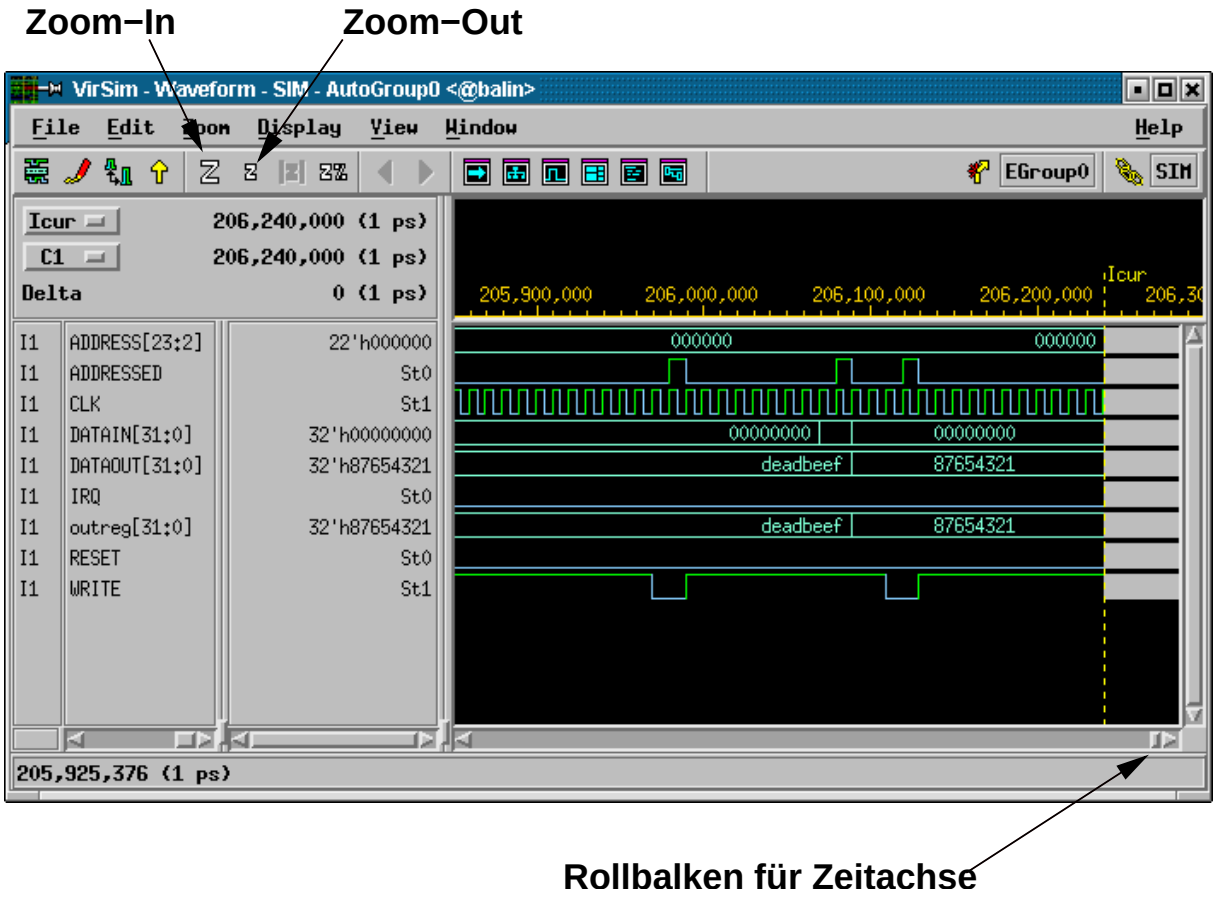


Abbildung 2.3: Waveform-Darstellung der Simulationsergebnisse

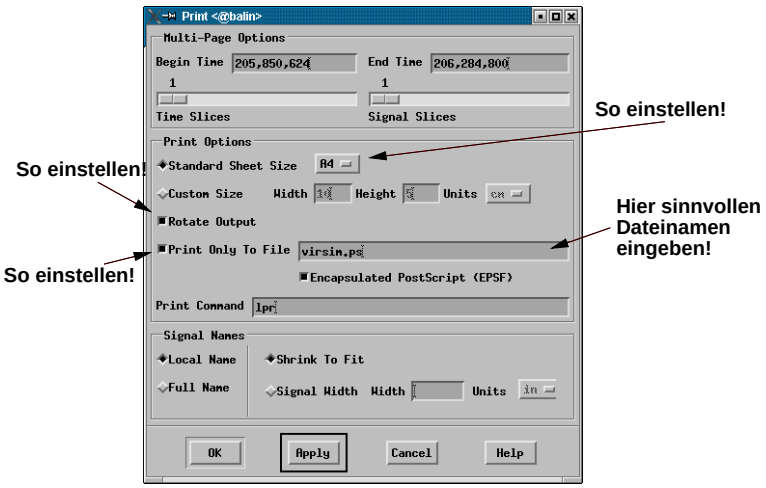


Abbildung 2.4: Export von Waveforms in EPS-Datei

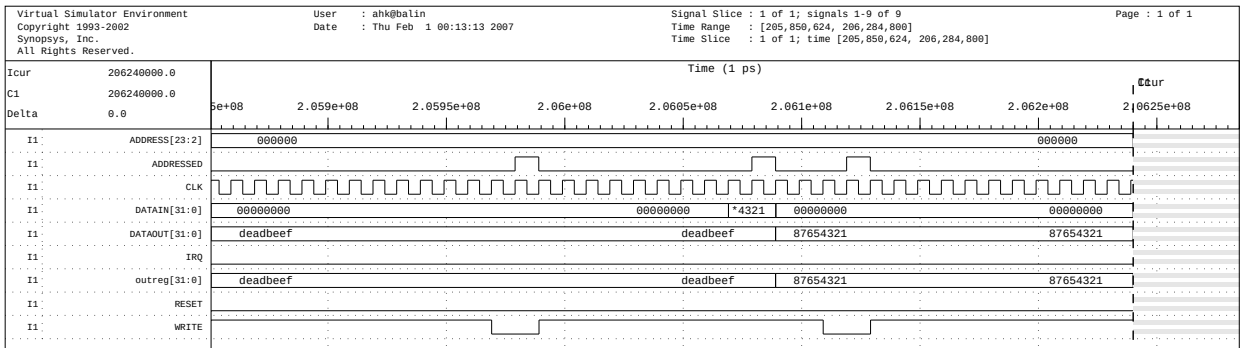


Abbildung 2.5: Beispiel für in LaTeX eingebundene Waveforms

2.6 Exportieren von Waveforms

Um eine solche Waveform-Darstellung aus dem Simulator zu exportieren (z. B. zur Einbindung in eigene Dokumentation) wählen Sie aus dem Menü File des Waveform-Fensters den Punkt Print... aus und füllen den nun angezeigten Druck-Dialog wie in Abbildung 2.4 aus. Nach dem Klicken von Apply wird die Ausgabe dann in die angegebene Datei geschrieben. Die Grafik in dieser ist allerdings häufig ungeschickt auf der Druckseite angeordnet. Durch das Linux-Kommando

```
ps2epsi virsim.ps
```

(geeignet angepasst für den von Ihnen in den Druckdialog eingetragenen Ausgabedateinamen) wird eine neue Datei mit der Endung `.epsi`, hier also `virsim.epsi` erzeugt. Diese kann dann leicht, beispielsweise in LaTeX, eingebunden werden (Abbildung 2.5).

Eine Umwandlung nach PDF ist ebenso möglich:

```
ps2pdf virsim.ps
```

erzeugt die PDF-Datei `virsim.pdf`, die Sie z. B. mit

```
evince virsim.pdf&
```

betrachten können.

Beenden können Sie den Simulator durch den Menüpunkt File, Exit.

Falls Sie eigene Experimente mit dem bisherigen Werkzeugfluss machen möchten: Führen Sie beispielsweise die Lese- und Schreibzugriffe nicht Wortadresse 0 der RCU (wie in der ursprünglichen Stimulus-Datei), sondern auf Wortadresse 1 aus. Überprüfen Sie das von Ihnen jetzt erwartete Verhalten mit einer geeigneten RTL-Simulation.

2.7 Erzeugen der RCU-Hardware

Nachdem die Simulation nun den prinzipiellen Nachweis der Funktion der RCU geliefert hat, kann die Verilog-Beschreibung jetzt auf die echte FPGA-Hardware des ML310 ACS abgebildet werden. Dies geschieht durch das Linux-Kommando

```
make bits
```

Dieser Schritt ist sehr rechenaufwendig und schließt beispielsweise neben einer Compilierung der Verilog-Quelltexte in digitale Schaltungen (sogenannte Logiksynthese) auch Platzierungs- und Verdrahtungsschritte für mehr als 7300 Logikblöcke und mehr als 47000 Zwei-Terminal-Nets ein. Die Laufzeit dieses Vorgangs

liegt je nach Variation der Schaltung bei ca. 35-40 Minuten. An dieser Stelle wird offensichtlich, dass der Test einer in Entwicklung befindlichen Schaltung weitgehend durch *RTL-Simulation* und nicht durch einfaches Ausprobieren auf der Hardware geschehen sollte!

2.8 Post-Place-&-Route-Simulation

Selbst wenn eine RTL-Simulation korrekte Ergebnisse liefert und bei der Synthese keine Fehler aufgetreten sind, kann es passieren, dass die Schaltung auf der Zielhardware nicht fehlerfrei läuft. Der erste Schritt in solch einem Fall ist, die Log-Dateien der Entwicklungswerkzeuge nach Hinweisen über zu lange kritische Pfade zu durchsuchen (vgl. Abschnitt 2.11). Eine weitere sehr wichtige Methode zur Lokalisierung von Fehlerursachen ist die Post-Place-&-Route-Simulation (auch Post-Layout-Simulation oder hier abkürzend Laysim) genannt, die Sie in Ihren Projekten mit dem Kommando

```
make laysim
```

einsetzen können. Im Unterschied zur RTL-Simulation werden hier auch Gatter- und Leitungsverzögerungen mitsimuliert. (Das bedeutet natürlich, dass vorher eine Synthese sowie Platzierung und Verdrahtung stattgefunden haben muss.) Eine Laysim ist deutlich zeitaufwändiger als eine RTL-Simulation, daher sollte sie nur durchgeführt werden, wenn auf RTL-Ebene alle Fehler ausgeräumt sind. Andersherum ist eine Laysim aber nötig, um ein Design zu verifizieren, *bevor* es auf der Zielhardware getestet wird.

Eine Schwierigkeit, die bei der Laysim auftritt, ist die Umbenennung von Signalen durch die vorangegangene Synthese. Im Basis-Slave-Mode-Projekt z. B. wurde der Adress-Bus **ADDRESS** umbenannt in

```
\plb_ddr_0/plb_ddr_0/plb_bus2ip_addr [31:0].
```

Dem Basis-Projekt liegt deswegen eine Einstellungs-Datei bei, die das neue, unverständlichere Signal automatisch auf den altbekannten Signalnamen **ADDRESS** abbildet. Solche symbolischen Links auf Signale oder eine Menge von Signalen können Sie auch selbst mit dem Bus-Builder in VCS erzeugen (siehe Abb. 2.6).

2.9 Hardware-Test der Slave-Mode-RCU

Bei Slave-Mode-RCUs kann die eigentliche RCU nach erfolgreicher Simulation auch gezielt in echter Hardware erprobt werden, selbst wenn der Software-Teil der Anbindung noch nicht bereitsteht. Dieser Vorgang kann durch das Linux-Kommando

```
make download
```

gestartet werden. Hier sollten am Ende Meldungen ähnlich zu

```
INFO:iMPACT:579 - '2': Completed downloading bit file to device.
INFO:iMPACT:580 - '2': Checking done pin ....done.
'2': Programmed successfully.
```

ausgegeben werden. Der Bitstrom, der das FPGA mit dem Gesamtsystem einschliesslich Ihrer RCU definiert, ist nun korrekt in den FPGA-Baustein übertragen worden und Sie können jetzt durch interaktive Kommandos Zugriffe auf die Hardware durchführen. Dazu geben Sie ein weiteres Linux-Kommando ein:

```
xmd
```

Nun wird als Prompt **XMD%** angezeigt. Hier geben Sie nun folgendes Kommando ein, um die Kommunikation mit der Hardware aufzubauen:

```
connect ppc hw
```

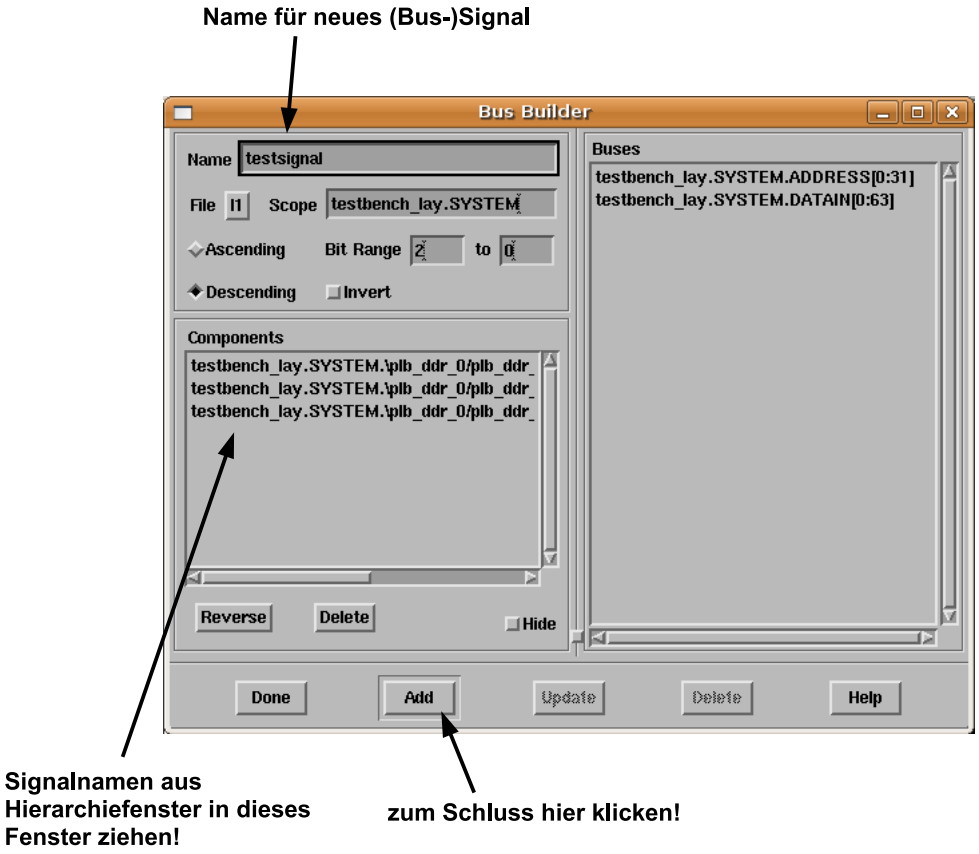


Abbildung 2.6: Zusammenfassung von Signalen zu Bussen mit dem Bus-Builder

Einige Zwischenmeldungen sollten hier mit den Zeilen

```
Connected to "ppc" target. id = 0
Starting GDB server for "ppc" target (id = 0) at TCP port no 1234
```

enden und der Prompt wieder erscheinen. Lösen Sie jetzt einen systemweiten Reset aus, um die gesamte ACS-Hardware in einen definierten Anfangszustand zu bringen. Hierzu verwenden Sie das Kommando

```
rst
```

Die Basisadresse der RCU auf dieser XMD-Sicht ist 0x10000000. Diese Adresse ist nun eine *Byte-Adresse*, keine Wortadresse (wie noch im Verilog) mehr. Das heisst also, dass Sie um z. B. auf das zweite 32b Register der RCU zuzugreifen, die Adresse 0x10000004 ansprechen würden. Dies würde zu einem Wert des ADDRESS-Eingangs der RCU von 0x000001 führen (dort RCU-relative Adresse, Adressbasis 0x10000000 im CPU-Speicherbereich entfällt, Zugriff auf Byte 4 (CPU) entspricht Zugriff auf Wort 1 (RCU), da ein Wort vier Bytes hat).

Zum Auslesen des Registers auf Wortadresse 0 (dort liegt ja outreg) geben Sie jetzt den Befehl

```
mrd 0x10000000
```

ein. Hier erfolgt hoffentlich die erwartete Ausgabe (0xDEADBEEF), die angibt, dass das Register auf RCU-Adresse 0 in der Tat korrekt initialisiert worden ist.

Jetzt schreiben wir den Wert 0x87654321 in das Register mit dem Kommando

```
mwr 0x10000000 0x87654321
```

Zur Überprüfung lesen wir nun den Wert erneut aus:

```
mrd 0x10000000
```

Wenn unsere RCU auch in Hardware richtig arbeitet, wird nun der neu geschriebene Wert (0x87654321) auch wieder ausgelesen. Um das Verhalten zu testen, dass von den Wortadressen 1, 2 und 3 immer der in `user.v` vorgegebene Wert geliefert wird, machen wir die Probe auf's Exempel und lesen von den entsprechenden Byte-Adressen im CPU-Adressraum:

```
mrd 0x10000004
mrd 0x10000008
mrd 0x1000000C
```

In allen drei Fällen wird 0xC0FFEE11 geliefert, unser Marker für ein noch nicht benutztes RCU-Register. Da wir nur auf die letzten beiden Bits der Wortadresse achten, wiederholt sich diese Registeranordnung alle vier Worte. Auf der RCU-Wortadresse 4 findet sich also wieder `outreg` (die beiden untersten Bits der Wortadresse sind 00, genau hier liegt outreg). Das Kommando

```
mrd 0x10000010
```

beweist diese These, es wird wieder der derzeit aktuelle Wert von outreg (0x87654321) geliefert. Beenden Sie die Sitzung nun durch das Kommando

```
exit
```

und kehren wieder zur Linux-Kommandozeile zurück.

2.10 Hardware-Test des Gesamtsystems

Nachdem nun Ihre RCU korrekt in Hardware läuft, ist es an der Zeit, das Gesamtsystem zu überprüfen. Durch das Kommando

```
make linux
```

wird auch der Software-Teil der Anwendung (aus der Datei `main.c`) für den PowerPC-405-Prozessor auf dem FPGA des ACE-M3 übersetzt, sowie ein bootfähiges Linux zusammengestellt. Dieses wird dann auf die Hardware übertragen und gestartet. Nach dem Durchlauf vieler Zwischenmeldungen (bei denen zwei Pausen von etwa 90s **normal** sind [1]) kommt schliesslich ein vertrauter Anmeldeprompt, bei dem Sie sich mit Ihrem normalen Login-Namen (`gruppe5` oder ähnlich) und Password anmelden können. Nach erfolgreicher Anmeldung arbeiten Sie nun nicht mehr auf Ihrem Arbeitsplatzrechner, sondern auf der ACE-M3!

Dies äußert sich zum einen durch den Inhalt Ihres HOME-Bereichs. Ihr Home auf der ACE-M3 entspricht einem lokalen Scratch-Verzeichnis ihres VM-Linux, für Gruppe 5 z. B. wäre dies

```
/scratch/gruppe5
```

Wenn Sie also eine Datei `foo.txt` *vom* VM-Linux *auf* das ACE-M3 übertragen wollen, geben Sie *in dem VM-Linux* das Kommando

```
cp foo.txt /scratch/gruppe5
```

ein. Die Datei `foo.txt` taucht damit in Ihrem HOME-Bereich auf dem ACE-M3 auf. In umgekehrter Richtung, also *vom* ACE-M3 *zum* VM-Linux lautet das Kommando für eine Datei `bar.txt` auf der VM analog

```
cp /scratch/gruppe5/bar.txt .
```

Die Datei `bar.txt` taucht nun im aktuellen Verzeichnis auf dem Arbeitsplatzrechner auf. Da der `/scratch`-Bereich nicht Bestandteil der Datensicherung ist und auch bei Platzknappheit jederzeit gelöscht werden kann, sollten Sie alle nicht automatisch wiederherstellbaren Dateien regelmäßig in Ihren HOME-Bereich (VM auf dem Arbeitsplatzrechner) kopieren.

Aber schauen wir uns an, was durch das Kommando `make linux` noch geschehen ist: Auf unserem ACE-M3-HOME-Bereich liegt jetzt auch eine Datei `main`. Es handelt sich dabei um das auf dem VM-Linux übersetzte Software-Programm unserer ACE-M3-Anwendung, das automatisch bereits in den ACE-M3-HOME-Bereich kopiert wurde. Eventuelle Dateien mit Namen beginnend mit `vcdplus-`... können Sie ignorieren. Es handelt sich dabei um Zwischendateien einer früheren Simulation auf dem Arbeitsplatzrechner, die dieser aus Platz- und Rechenzeitgründen auch auf Ihrem `/scratch`-Unterverzeichnis abgelegt hat. Sie sind für die Arbeit auf dem ACS aber bedeutungslos.

Nicht bedeutungslos sind allerdings eventuelle Eingabedateien, die Ihre spezielle ACE-M3-Anwendung möglicherweise noch benötigt. Da diese Dateien zwischen unterschiedlichen Applikationen variieren können, kann sie der Werkzeugfluss nicht automatisch vom VM-Linux auf den ACE-M3-HOME-Bereich kopieren. Falls Ihre Anwendung also Eingabebilder o. Ä. benötigt, kopieren Sie diese durch Absetzen geeigneter Kommandos (siehe oben!) *selber* in den HOME-Bereich auf dem ACE-M3.

Wenn alle benötigten Dateien vorhanden sind (im Fall unserer einfachen Beispielanwendung hier mit dem les- und schreibbaren Register sind keine weiteren erforderlich), kann nun die vollständige ACE-M3-Anwendung, bestehend aus RCU- und CPU-Teil (unserem übersetzten C-Programm) gestartet werden. Dazu wird einfach der Name des übersetzten Programms, hier also

```
./main
```

[1] Dies haben wir Xilinx und auch VMware zu verdanken: da der USB-Treiber von Xilinx unter Linux nicht verlässlich läuft, muss ein Parallel-Programmier-Kabel für das ML310 verwendet werden. VMware ist aktuell leider nicht in der Lage, den schnelleren ECP-Modus der parallelen Schnittstelle zu emulieren, daher kommt hier der extrem langsame Bidirektional-Modus zum Einsatz.

als Linux-Kommando eingegeben. Das nun ablaufende Programm initialisiert die RCU, stellt die Kommunikation her und führt die Lese- und Schreibzugriffe aus. Die printf-Funktionen geben die Registerwerte vor (0xDEADBEEF) und nach dem Schreiben des neuen Wertes (0x87654321) aus.

Tipp: Falls **`make linux`** länger als ca. 3 Min. ohne weitere Ausgabe hängen sollte, tippen Sie die Tastenfolge Control-**`A`** Control-**`A`**. Dann sollten Sie wieder einen Linux-Prompt auf dem Arbeitsplatzrechner bekommen. Geben Sie dann einfach das Kommando **`make linux`** erneut ein. Sollte auch nach dem dritten Versuch der Start des ACE-M3-Linux fehlschlagen, schalten Sie das ACE-M3 aus und wieder ein. Warten Sie etwa 5 s, und versuchen dann wieder **`make linux`**. Wenn alle diese Versuche fehlschlagen, kontaktieren Sie bitte Ihren Betreuer für diese Lehrveranstaltung.

Falls bis hierher alles geklappt hat, haben Sie gerade erfolgreich alle Schritte von der Formulierung getrennter RCU- und CPU-Teile bis hin zur Ausführung der vollständigen Anwendung auf einem modernen adaptiven Rechner absolviert. Nun ist es Zeit für Ihre eigenen Entwicklungen! Dazu modifizieren Sie die **`main.c`**, **`user.v`** und **`stimulus.v`**-Dateien der Materialsammlung entsprechend den aktuellen Anforderungen.

2.11 Details zur Entwicklungsumgebung

Abbildung 2.7 stellt den detaillierten Compile-Flow für den Hardware-Teil Ihrer Programme dar und kann zu Debugging-Zwecken hilfreich sein.

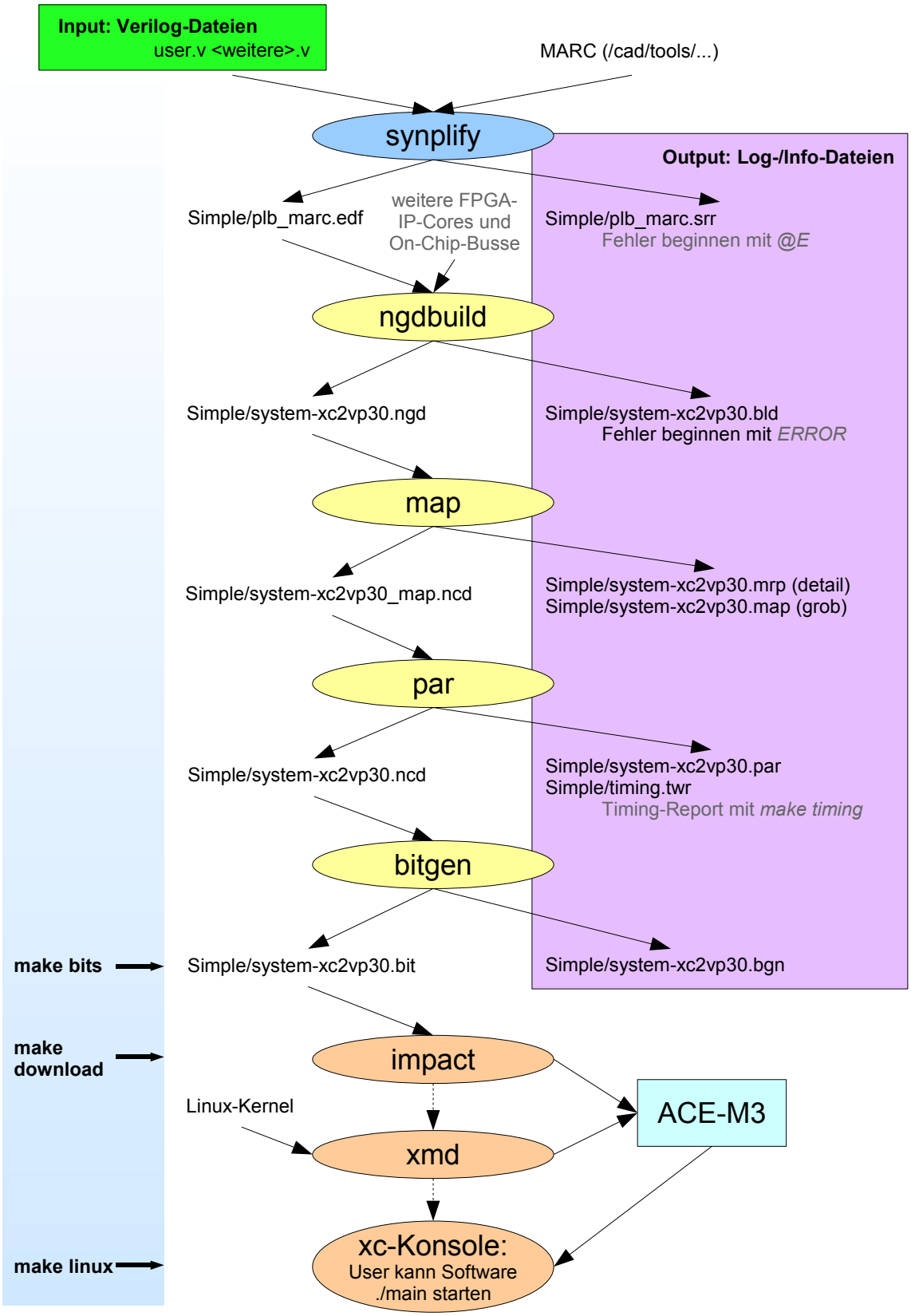


Abbildung 2.7: Ablauf der Hardware-Erzeugung in der verwendeten Entwicklungsumgebung