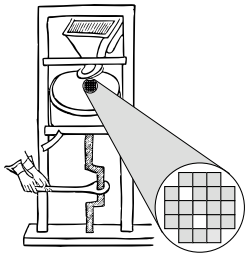
Technische Universität Darmstadt
FG Eingebettete Systeme und
ihre Anwendungen (ESA)

Prof. Dr. Andreas Koch



"Eingebettete Prozessorarchitekturen"
Aufgabe 4: Adaptive Computer
Abgabe bis zum 03.03.2008, 23:59 MET

1 Einleitung

In dieser praktischen Übung befassen Sie sich mit der Programmierung einer einfachen Anwendung auf einem adaptiven Computer. Als Sprachen verwenden Sie dabei C für den Software-Teil (CPU) und die Hardware-Beschreibungssprache Verilog für den RCU-Teil. Im Gegensatz zu den bisherigen Umgebungen haben Sie hier die Möglichkeit, Ihren Entwurf auch auf echter Hardware, nämlich dem am FG ESA entwickelten Xilinx ML310 ACS auszuprobieren.

2 Vorinstalliertes Material

Im Unterverzeichnis `phase4-material` Ihres HOME-Bereiches ist ein Beispielprojekt mit den zur Bearbeitung dieser Aufgabe erforderlichen Dateien bereits vorinstalliert. Falls Sie eine neue Version benötigen, können Sie die entsprechende Archivdatei vom Web-Site der Veranstaltung herunterladen und auspacken. Es handelt sich dabei um eine Muster-Anwendung, deren RCU-Teil `user.v` ein les- und schreibbares Register (von hier vier möglichen) auf der RCU-Adresse 0 implementiert. Die Datei `stimulus.v` enthält die Testdaten für die Simulation der RCU, hier werden Lese- und Schreibzugriffe auf das RCU-Register ausgeführt. In `main.c` finden Sie den dazugehörigen C-Teil, der nun die CPU das Register schreiben und lesen lässt und die dafür benötigte Zeit misst. Diese Dateien sollten Sie als Grundlage Ihres ACS-Programmes verwenden. Eine Nicht-ACS-Musterimplementierung des in dieser Aufgabe zu bearbeitenden Algorithmus in reinem C finden Sie in der Datei `deinterlace.c`. Testdaten liegen in Form eines 256x256 Pixel großen Bildes in 8b Graustufen als Datei `lena256i.pgm` vor. Ebenso gibt es ein Referenzbild `lenaout-reference.pgm`, das die korrekte Bearbeitung des Eingabebildes wiedergibt.

3 Hardware und Arbeitsraum

An jedem der Arbeitsplatzrechner ist ein eigenes ML310 ACS angeschlossen, das jeweils nur von *einem* Benutzer parallel benutzt werden kann. Überprüfen Sie daher bitte mit dem Kommando `who`, ob Sie Ihren ausgewählten Rechner wirklich für sich haben! Da bei Fehlprogrammierungen (und gelegentlich [aber selten] im Regelbetrieb) das ACS den Betrieb einstellt, ist es hilfreich, die Tests mit echter Hardware *vor Ort* im Praktikumsraum des FG ESA zu machen. Sprechen Sie dazu ein Mitglied des FG an, wir schließen Ihnen gerne auf!

(a) Kombinierte Halbbilder

(b) "Verschmiertes" Vollbild

Abbildung 1: Eingabe und Ausgabe des einfachen Deinterlacers

4 Aufgabe

Fernseh- und Videobilder werden traditionell als eine Folge von Halbbildern übertragen, bei denen jedes der Halbbilder abwechselnd die geraden und die ungeraden Bildzeilen enthält (*interlaced scan)*.

Im Gegensatz zu den in Fernsehern üblichen Kathodenstrahlröhren findet die Bildverarbeitung auf Computern sowie die Darstellung auf den meisten modernen Flachbildschirmen im Vollbildverfahren (*progressive scan*) statt. Hier werden also immer vollständige Bilder gehandhabt. Um nun auch Interlaced-Bilder auf solchen Geräten darzustellen, müssen diese *deinterlaced* werden. Hierbei werden die Folgen von Halbbildern geeignet zu einer Folge von Vollbildern kombiniert. Dies kann auf die verschiedensten Weisen geschehen (siehe z.B. Wikipedia unter dem Stichwort Deinterlacing).

Sie sollen einen *sehr* einfachen Deinterlacer implementieren, der ein Eingabebild bestehend aus zwei bereits übereinandergelegten Halbbildern (siehe Abbildung 1.a) von 256 x 256 Pixeln, jeder ein 8b Grauwert mit 0=Schwarz und 255=weiß, in ein höherwertiges Vollbild gleicher Abmessungen umwandelt. Als Verfahren sollen die Pixel jeder Zeile des Ausgabebildes als *Mittelwert* ihres eigentlichen Wertes im Eingabebild und des in der darüberliegenden Zeile des Eingabebildes gelegenen Pixels gebildet werden. Die im ursprünglichen Eingabebild vorhandenen Bildzeilen aus unterschiedlichen Halbbildern werden also vertikal von Zeile zu Zeile über das Bild "verschmiert" und schwächen so die, häufig beim einfachen Übereinanderlegen von Halbbildern auftretende, grobe Zeilenrasterung ab (Bild 1.b). Nochmal zur Klarstellung: Der Pixel aus der Vorgängerzeile kommt aus dem *Original*-Eingabebild, *nicht* aus dem im Entstehen begriffenen verbesserten Vollbild! Dieses Verfahren, das keine sonderlich hochwertigen Vollbilder liefert, wird so auch heute noch in (sehr billigen) Geräten der Unterhaltungselektronik verwendet. Eine C-Beispielrealisierung dieses Algorithmus finden Sie in der Datei `deinterlace.c` der Materialsammlung.

Damit die Hardware-Realisierung für Sie einfacher wird, sollen Sie die echten Bilder (siehe Abbildung 2.a) allerdings um 90 Grad *im Gegenuhrzeigersinn* verdreht bearbeiten (siehe Abbildung 2.b). Ihr Beispieleingabebild `lena256i.pgm` hat bereits dieses Format. Der Grund für diesen scheinbaren Umweg liegt darin, dass Sie nun die ehemaligen Zeilen als horizontal benachbarte, also auch im Speicher direkt aufeinanderfolgende, Werte ansehen können und somit keine ganze Zeilen in der RCU puffern müssen.

Abbildung 2.b zeigt den Zusammenhang zwischen dem um 90 Grad gedrehten Eingabebild (nun mit *Spalten* aus abwechselnden Halbbildern) und dem Ausgabebild (Abbildung 2.c). Hier ist auch die Berechnungsvorschrift für die einzelnen Pixel angegeben. Wie Sie sehen, wird jeder Pixel aus den Originalwerten im Eingabebild von sich selbst und seinem linken Nachbarn gebildet. Die Pixel am linken Rand des Bildes nehmen den neutralen Grauwert 128 für ihren fiktiven linken Nachbarn an.

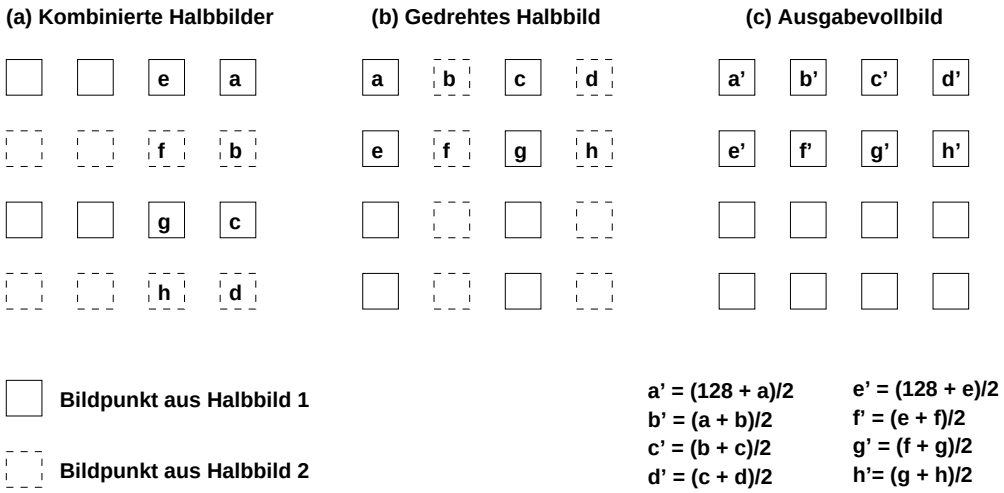


Abbildung 2: Bildformate und Pixelzusammenhänge

5 Werkzeugfluß

Um sich mit der ACS-Umgebung vertraut zu machen, sollten Sie zunächst mit einer sehr einfachen Anwendung experimentieren. Diese realisiert lediglich 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.

5.1 Dateistruktur

Die Materialsammlung für Phase 4 enthält für diese 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 00). 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.

`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.

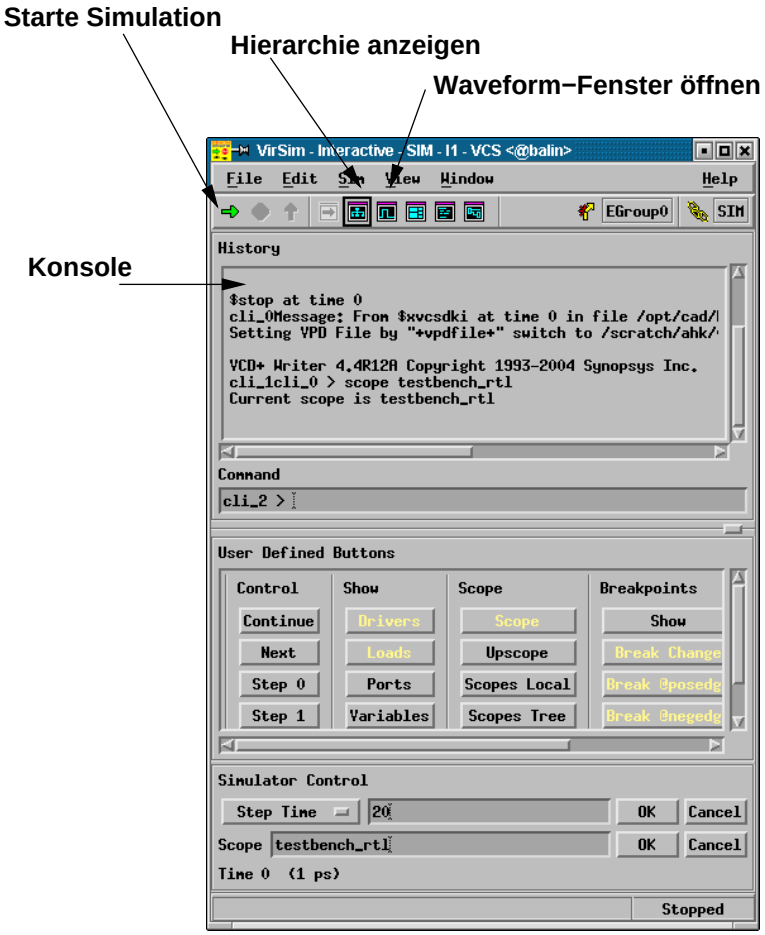


Abbildung 3: Graphische Simulationsumgebung VirSim

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.

5.2 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 Unix-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 3).

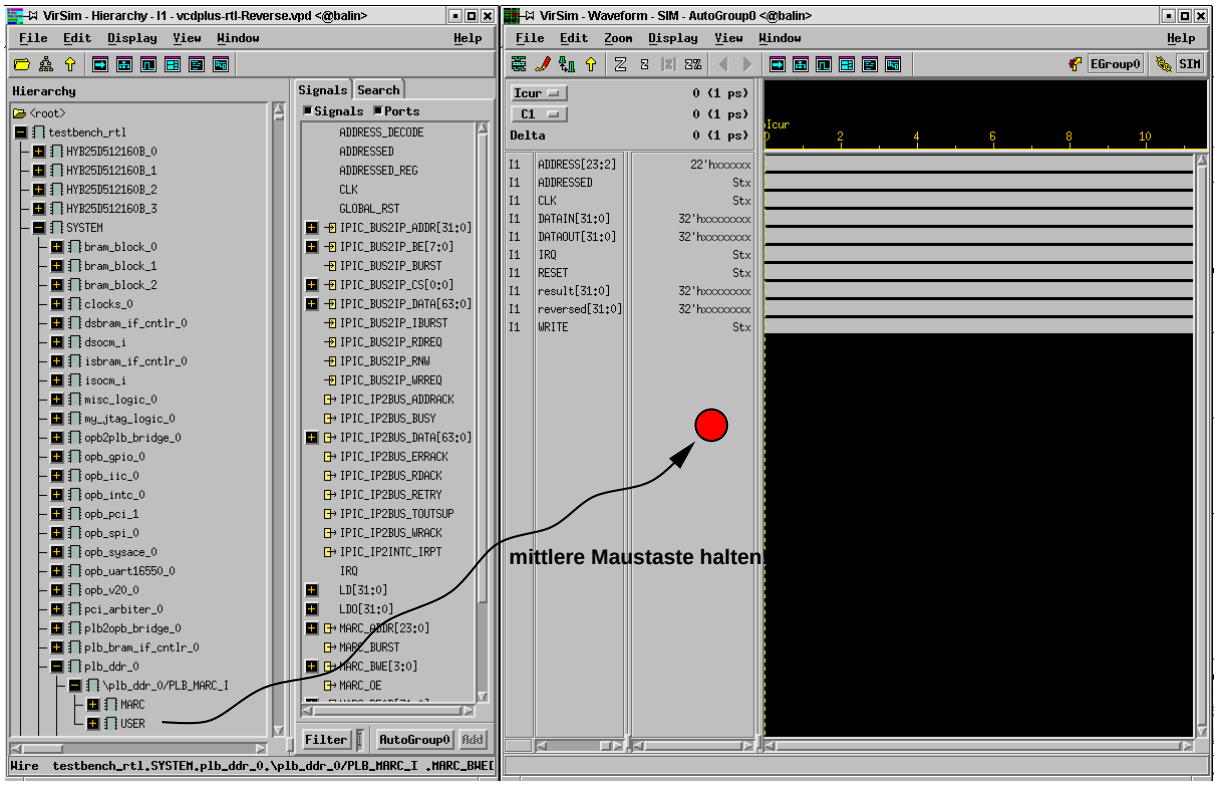


Abbildung 4: Auswahl der Signale zur Waveform-Anzeige

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 3) sowie ein leeres Waveform-Fenster öffnen (ebenfalls in entsprechendes Icon klicken). Dieser Zustand ist in Abbildung 4 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 3) 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

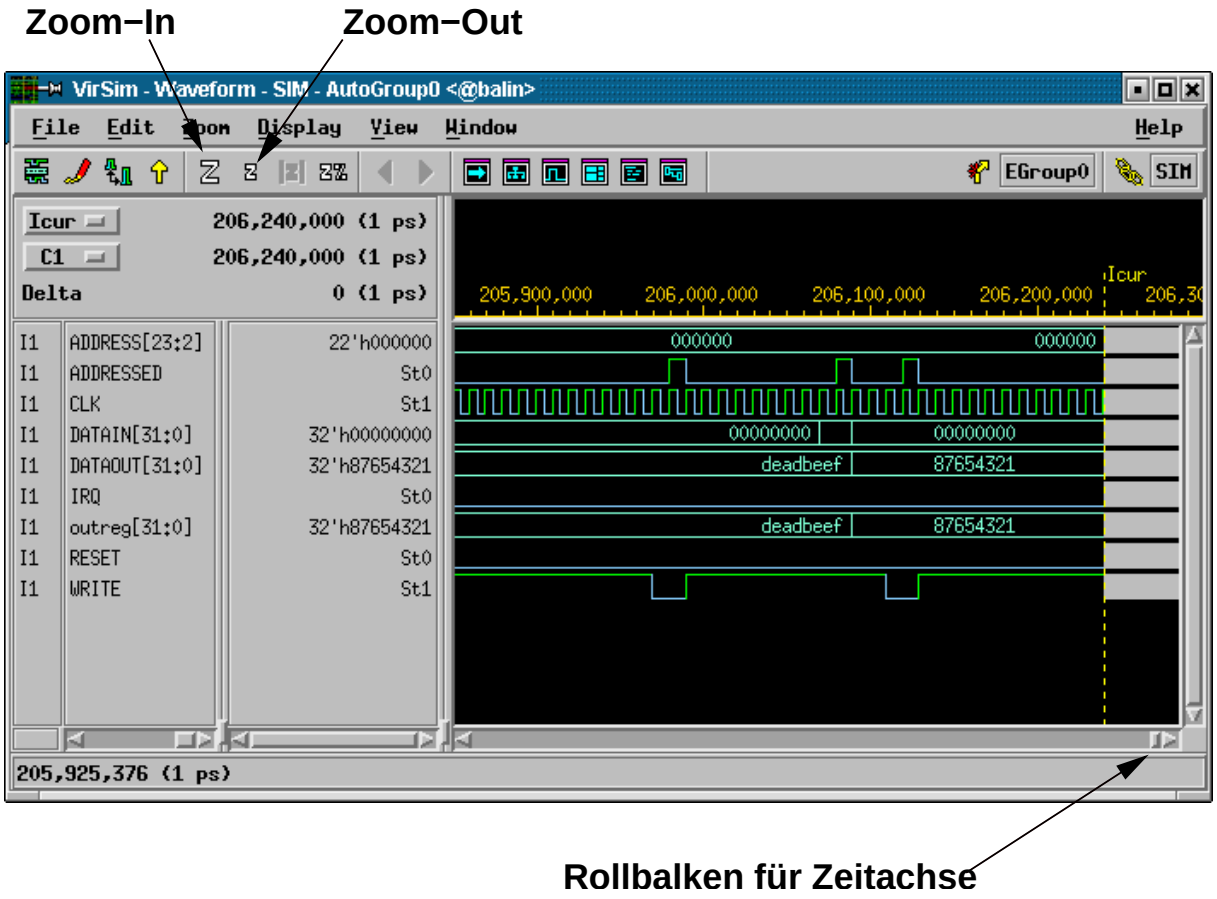


Abbildung 5: Waveform-Darstellung der Simulationsergebnisse

```
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 (Erinnerung TGDI2: Die Zeit *vor* der steigenden Flanke ist die Setup-Zeit, die nach der Flanke die Hold-Zeit. Die Hold-Zeit kann bei uns mit 0 angenommen werden, relevant ist also nur der Wert eines Signals vor der Flanke).

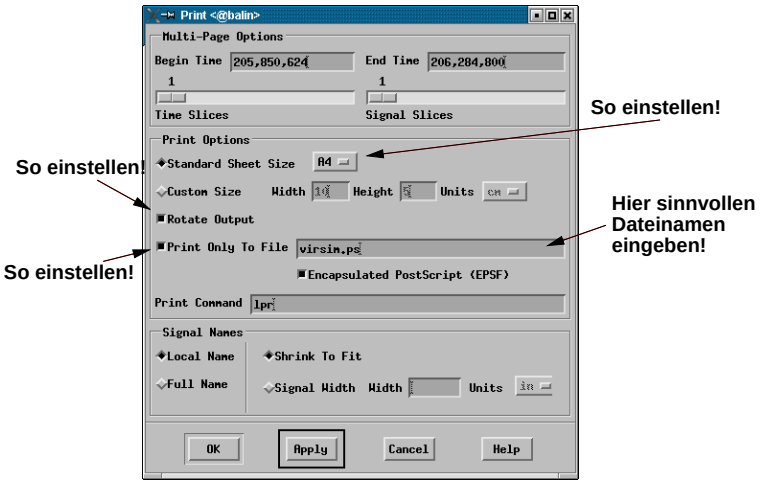


Abbildung 6: Export von Waveforms in EPS-Datei

Um eine solche Waveform-Darstellung aus dem Simulator zu exportieren (z.B. für 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 6 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 Unix-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 7).

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.

5.3 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 Unix-Kommando

```
make bits
```

Dieser Schritt ist sehr rechenaufwendig und schließt beispielsweise neben einer Kompilierung der Verilog-Quelltexte in digitale Schaltungen (sogenannte Logiksynthese) auch Platzierungs- und Verdrahtungsschritte für mehr als 7300 Logikblöcken 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 *Simulation* und nicht durch einfaches Ausprobieren auf der ACS-Hardware geschehen sollte!

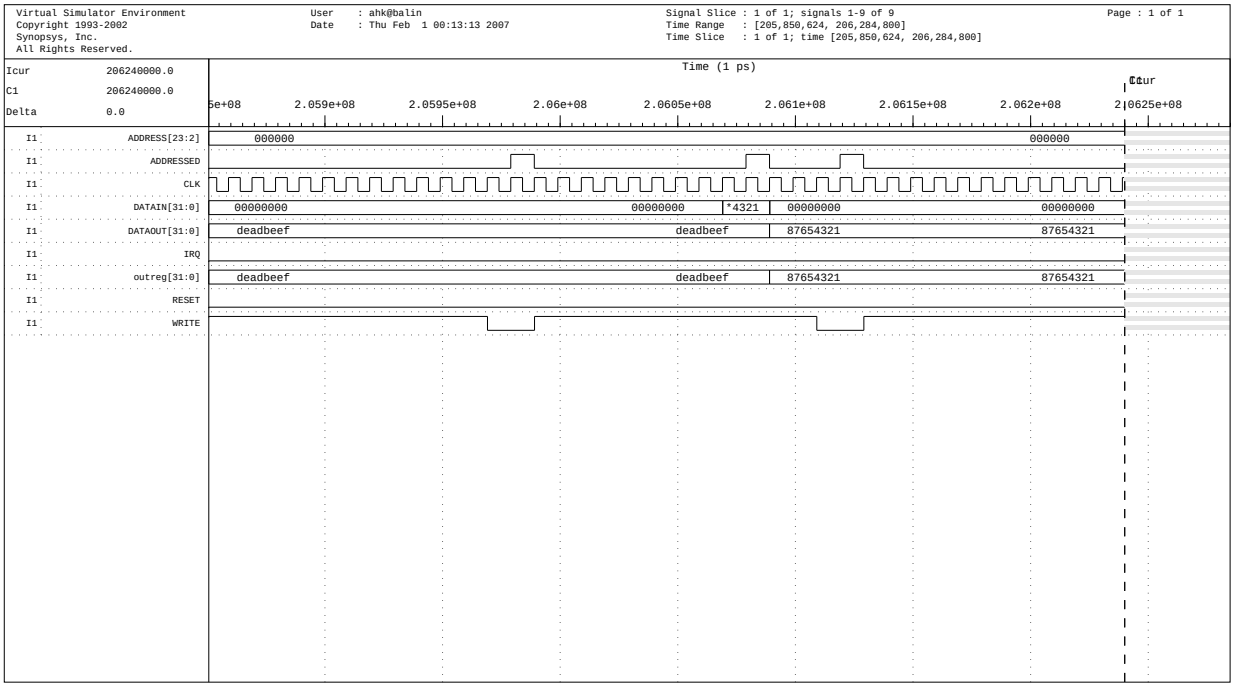


Abbildung 7: Beispiel für in LaTeX eingebundene Waveforms

5.4 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 Unix-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 Unix-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
```

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 geben 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
```

Hier erfolgt hoffentlich die erwartete Ausgabe (0xDEADBEEF), die angibt, das 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 Unix Kommandozeile zurück.

5.5 Hardware-Test des Gesamtsystems

Nachdem jetzt Ihre RCU selbst bereits korrekt in Hardware läuft, ist es nun an der Zeit, auch 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 ML310 ACS übersetzt, sowie ein bootfähiges Linux zusammengestellt. Dieses wird dann auf die Hardware übertragen und gestartet. Nach dem Durchlauf vieler Zwischenmeldungen (bei denen

gelegentliche Pausen von 5-10s **normal** sind) kommt schliesslich ein vertrauter Anmeldeprompt, bei dem Sie sich mit Ihrem normalen Login-Namen (`gruppe01` oder ähnlich) und Password anmelden können. Nach erfolgreicher Anmeldung hier arbeiten Sie nun nicht mehr auf Ihrem Arbeitsplatzrechner, sondern auf dem ML310 ACS!

Dies äußert sich zum einen durch den Inhalt Ihres HOME-Bereichs. Auf dem ACS steht *nicht* mehr Ihr normaler HOME-Bereich aus dem Netz des FG ESA zur Verfügung. Sie befinden sich nun in einem dedizierten Netz, nur bestehend aus Ihrem Arbeitsplatzrechner und dem nebenstehenden ACS. Dabei wird das Verzeichnis

```
/scratch/gruppe01
```

des Arbeitsplatzrechners (wobei `gruppe01` nur ein Beispiel für Ihren Login-Namen ist) als HOME-Bereich für diesen Login-Namen dem ACS zugänglich gemacht. Wenn Sie also eine Datei `foo.txt` *vom* Arbeitsplatzrechner *auf* das ACS übertragen wollen, geben Sie *auf dem Arbeitsplatzrechner* das Kommando

```
cp foo.txt /scratch/gruppe01
```

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

```
cp /scratch/gruppe01/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 (siehe Leitfaden zum Rechnernetz des FG ESA), sollten Sie alle nicht automatisch wiederherstellbaren Dateien regelmäßig in Ihren HOME-Bereich auf dem Arbeitsplatzrechner kopieren.

Aber schauen wir uns an, was durch das Kommando **`make linux`** noch geschehen ist: Auf unserem ACS-HOME-Bereich liegt jetzt auch eine Datei `main`. Es handelt sich dabei um das auf dem Arbeitsplatzrechner übersetzte Software-Programm unserer ACS-Anwendung, das automatisch bereits in den ACS-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 ACS-Anwendung möglicherweise noch benötigt. Da diese Dateien zwischen unterschiedlichen Applikationen variieren können, kann sie der ACS-Werkzeugfluß nicht automatisch vom Arbeitsplatzrechner auf den ACS-HOME-Bereich kopieren. Falls Ihre Anwendung also Eingabebilder etc. benötigt, kopieren Sie diese durch Absetzen geeigneter Kommandos (siehe oben!) *selber* in den HOME-Bereich auf dem ACS.

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 ACS-Anwendung, bestehend aus RCU- und CPU-Teil (unserem übersetzten C-Programm) gestartet werden. Dazu wird einfach der Name des übersetzten Programmes, hier also

```
./main
```

als Unix-Kommando gegeben. 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. 20s ohne weitere Ausgabe hängen sollte, tippen Sie die Tastenfolge Control-A Control-A. Dann sollten Sie wieder einen Unix-Prompt auf dem Arbeitsplatzrechner bekommen. Geben Sie dann einfach das Kommando **`make linux`** erneut. Sollte auch nach dem dritten Versuch der Start des ACS-Linux fehlschlagen, halten Sie am blauen ACS-Gehäuse den Reset-Taster (an der Gehäusefront unter dem größeren Einschaltknopf) ca. 2-3s gedrückt, um einen echten Hardware-Reset des ACS auszulösen. Warten Sie dann, bis sich die LEDs auf der ML310 Leiterplatte beruhigt haben, und probieren Sie

make linux noch einmal. Wenn nun auch noch drei weitere Versuche fehlschlagen, greifen Sie auf die Rückseite des ACS-Gehäuses und unterbrechen mit dem dort gelegenen Netzschalter die Stromzufuhr für ca. 1 Minute, bevor Sie das ACS wieder einschalten. Warten Sie nun ebenfalls auf stetig leuchtende LEDs (bis sich das System grundinitialisiert hat), 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 ACS 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.

6 Untersuchung der reinen Software-Version (2 Punkte)

Nachdem Sie sich nun mit dem ACS-Werkzeugfluß vertraut gemacht haben, gilt es jetzt, die konkrete Aufgabe der Beschleunigung des Primitiv-Deinterlacers aus Abschnitt 4 zu lösen. Sie finden eine reine Software-Version des Algorithmus in der Materialsammlungs-Datei `deinterlace.c`, speziell in der Funktion `deinterlace`.

Übersetzen Sie diese Datei zunächst auf dem Arbeitsplatzrechner selbst mit dem Kommando

```
gcc -o deinterlace deinterlace.c
```

und führen Sie das Kompilat durch das Kommando

```
./deinterlace
```

aus. Das Programm liest ein Eingabebild aus der (fest vorgegebenen) Datei `lena256i.pgm` und schreibt das Ergebnis in die Datei `lenaout.pgm`. Vergleichen Sie das so berechnete Bild mit den Referenzdaten in der Datei `lenaout-reference.pgm` durch das Kommando

```
cmp lenaout.pgm lenaout-reference.pgm
```

Falls alles richtig ist, wird hier *keine* Meldung ausgegeben und der Unix-Shell-Prompt erscheint sofort wieder.

Nun sollen Sie die Ausführungszeit des reinen Software-Programmes auf dem für eingebettete Anwendungen realistischeren PowerPC 405 Prozessor des ACS messen. Erweitern Sie dazu die `deinterlace`-Funktion um geeignete Zeitmessungen (wie im Beispielprogramm `main.c`, dem Skript und der Vorlesung beschrieben). Achten Sie darauf, dass Sie in `deinterlace` *nur* die eigentliche Deinterlace-Berechnung über das ganze Bild messen. Es darf keine dieser Anweisungen in der Zeitmessung fehlen, aber auch keine weiteren (z.B. Ausgaben oder Dateioperationen) hinzukommen!

Da der Werkzeugfluß das Hauptprogramm immer in der Datei `main.c` erwartet, kopieren Sie Ihr modifiziertes `deinterlace.c` nun nach `main.c` (wenn Sie möchten, können Sie das Original `main.c` vorher sichern). Durch das Kommando:

```
make main
```

wird das Programm nun mit Hilfe eines sogenannten Cross-Compilers (Compiler läuft auf anderer Hardware als der, für die Code erzeugt wird) in PowerPC 405 Maschinen-Code übersetzt. Wenn dieser Vorgang fehlerfrei abläuft, kopieren Sie die Eingabe- und Referenzbilder vom Arbeitsplatzrechner auf das angeschlossene ACS und starten jetzt das dortige Betriebssystem durch das Kommando:

```
make linux
```

Loggen Sie sich nun auf dem ACS ein und lassen Sie **./main** laufen. Vergleichen Sie das auf der ACS-CPU berechnete Ausgabebild mit dem Referenzbild. Falls auch hier alles übereinstimmt, notieren Sie die durch Ihr Programm gemessene Ausführungszeit der `deinterlace`-Funktion. Sichern Sie diesen Stand von `main.c`, da er Bestandteil der Abgabe sein wird.

7 Entwicklung der RCU-beschleunigten Deinterlacers (6 Punkte)

Als nächstes gilt es, den Ablauf der Operation durch eine im Slave-Mode betriebene RCU zu beschleunigen. Dazu müssen Sie sich zunächst Gedanken über eine geeignete Architektur machen (Datenformate?, SIMD?, ...). Sie müssen darauf achten, dass Ihre RCU auch mehrere Programmstarts hintereinander (ohne komplettes Neu-Booten des Systems!) korrekt bearbeitet. Tipp: Es bietet sich an, die Werte der vorigen Pixel (des zukünftigen linken Nachbarn für die Mittelwertberechnung) *innerhalb* der RCU zu speichern (und nicht jedesmal neu übergeben, das würde viel zu langsam).

Falls Sie mehrere les- und schreibbare Register in der RCU realisieren möchten, finden Sie Anregungen auf Seite 34 des ACS-Skripts in Zeile 111–118 zum Bereitstellen lesbarer Register. Die Handhabung mehrerer schreibbarer Register wird auf Seite 34–35, Zeile 140–161 vorgeschlagen. Für RCU-interne Register, die nicht von der CPU zugreifbar sein müssen, ist dieser Aufwand nicht erforderlich. Hier reicht es, das Register geeignet mit reg zu deklarieren und in einem always @(posedge CLK ...)-Block zu verwenden.

Konkret sollen Sie bei der Realisierung Ihrer RCU-Architektur wie folgt vorgehen:

1. Passen Sie `user.v` entsprechend Ihrer geplanten RCU-Architektur an.
2. Schreiben Sie Testmuster in `stimulus.v`, anhand derer Sie die korrekte Funktion Ihrer RCU überprüfen können.
3. Erproben Sie Ihre RCU durch RTL-Simulation mit Ihren neu erstellten Testmustern. Dabei können Sie gezielt auch $display-Anweisungen im Verilog verwenden. Aber Vorsicht: Wenn diese nicht geschickt platziert werden, produzieren sie *in jedem* Takt eine Ausgabe. Sie würden also alleine für den Systemstart, noch bevor Ihre Read32/Write32-Anweisungen überhaupt ausgeführt werden, bereits Millionen Zeilen an Ausgabe produzieren. Von der nun stark verlangsamten Simulationsgeschwindigkeit abgesehen sind solche Mengen an Daten natürlich auch nicht mehr hilfreich beim Debugging. Verlassen Sie sich daher besser auf die Waveforms (auch, wenn sie für weniger hardware-erfahrene Entwickler zunächst eine etwas ungewohnte Form des Outputs darstellen).

Wenn hier alles zu Ihrer Zufriedenheit funktioniert, machen Sie einen aussagekräftigen Ausdruck Ihrer Waveforms (in eine Datei, wie oben beschrieben). Dieser wird auch Teil der Abgabe.

7.1 Hardware-Erprobung der RCU

Nachdem nun Ihre RCU in der Simulation korrekt funktioniert, kann Sie jetzt direkt in echter Hardware erprobt werden. Sie sollten den Hardware-Test zunächst via **`make download`** und **`xmd`** auf die RCU selber beschränken und durch geeignete XMD-Kommandos im Dialog mit Ihrer RCU prüfen, ob Daten korrekt entgegengenommen, berechnet und ausgegeben werden.

7.2 Hardware-Erprobung des Gesamtsystems

Jetzt können Sie sich an den Test des Gesamtsystems wagen. Ändern Sie dazu Ihr `main.c` so, dass nun statt der reinen Software-Berechnung Ihre RCU angesprochen wird (Initialisierung, Datentransfers). Starten Sie dann das Gesamtsystem mit **`make linux`**. Die Bilddaten haben Sie bereits in einem vorigen Schritt auf das ACS kopiert, eine eventuell vorhandene Ausgabedatei `lenaout.pgm` sollten Sie aber sicherheitshalber löschen. Starten Sie nun die kombinierte CPU-RCU-Anwendung selbst und überprüfen Sie, ob die neu geschriebenen Ausgabedaten mit den Referenzdaten übereinstimmen. Falls alles geklappt hat, starten Sie Ihr **`./main`**-Programm gleich nocheinmal und überprüfen die neu geschriebene Ausgabedaten (auf Datum/Uhrzeit schauen!) anhand der Referenz erneut.

Wenn auch jetzt noch alles stimmt: Herzlichen Glückwunsch, Ihre erste *eigene* ACS-Anwendungen läuft nun korrekt. Notieren Sie die jetzt gemessenen Rechenzeit und freuen Sie sich über die (hoffentlich erreichte) Beschleunigung gegenüber der reinen PowerPC 405-Software-Lösung.

8 Abgabe

Zusammenfassend geben Sie für diesen Aufgabenteil ab:

1. Den kommentierten Quellcode der reinen Software-Version von `main.c`, angereichert um die Anweisungen zur Zeitmessung.
2. Die Ausführungszeit in Mikrosekunden der eigentlichen Deinterlace-Berechnung der reinen Software-Version auf dem PowerPC 405-Prozessor des ML310 ACS.
3. Die kommentierten Quellcodes Ihrer RCU und des darauf umgestellten C-Programmes (die Dateien `user.v` und `main.c`) sowie die von Ihnen zur Simulation verwendete Testdatei `stimulus.v`.
4. Die Ausführungszeit in Mikrosekunden der eigentlichen Deinterlace-Berechnung der durch Ihre Slave-Mode-RCU beschleunigten (?) Anwendung.
5. Einen aussagekräftigen Auszug aus Ihren Waveforms, der die Funktionsfähigkeit Ihrer RCU zeigt, sowie eine kurze Erläuterung, was man darauf sieht und warum es richtig ist.

Für den Prosa-Teil der beiden Aufgaben dieses Blattes können Sie zusammen eine PDF-Datei abgeben (die am besten auch die Waveforms einbindet), die `.v` und `.c` Dateien hängen Sie aber bitte getrennt an die Abgabe-Mail an!

Ihre Abgaben tätigen Sie fristgerecht in Form einer E-Mail an die Adresse

`epa07@esa.informatik.tu-darmstadt.de`

mit dem Betreff "Gruppe N Aufgabe 4", wobei N Ihre Gruppennummer ist. Diese E-Mail hat als Anhänge die in den Teilaufgaben geforderten Komponenten.

9 Hinweise zum Thema Plagiarismus

Im Rahmen dieser Veranstaltung wird eine vorher festgelegte Arbeitsgruppe bewertet. Fremde Code-Bibliotheken außer den vom Dozenten 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.

Der Fachbereich Informatik misst der Einhaltung der Grundregeln der wissenschaftlichen Ethik großen Wert bei. Mit der Abgabe einer Lösung (Hausaufgabe, Programmierprojekt, etc.) bestätigen Sie, dass Ihre Gruppe die alleinigen Autoren des gesamten Materials sind. Weiterführende Informationen zu diesem Thema finden Sie unter `http://www.informatik.tu-darmstadt.de/Plagiarism`.