Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Sommersemester 2003

SEMESTERARBEIT

für

Mikael Feriencik (D-INFK)
Silvan Wegmann (D-INFK)

Betreuer: Herbert Walder
Leitung: Prof. Dr. Lothar Thiele

Ausgabe: 31. März 2003
Abgabe: 4. Juli 2003

HW-Tasks for Reconfigurable OS

Inhaltsverzeichnis

1 Aufgabenstellung

Ziel dieser SA war es, Demonstrationstasks für ein rekonfigurierbares Hardwarebetriebssystem, wie in [1] beschrieben, zu erstellen. Es war für uns sehr schnell klar, dass wir etwas aufbauen wollen, das den RAMDAC und den PS/2 Port des XESS Boards verwendet um so eine interaktive Anwendung zu haben. Wir machten uns also ans Design der Maus- und Bildschirmkomponenten und überlegten uns gleichzeitig, in was für eine Anwendung wir diese Komponenten verpacken wollen.

Im Verlaufe der Zeit mussten wir allerdings feststellen, dass diese Art von Komponenten nicht geeignet sind, um auf dem Hardware OS laufen zu lassen. Userinterface Komponenten, wie wir sie hier vorstellen, müssen permanent vorhanden sein, um einem Benutzer ständig Feedback zu geben. Wenn eine solche Komponente vom OS herausgeladen wird, weiss der Benutzer nicht, ob nun das System in einem Fehlerzustand gelandet ist oder ob noch alles normal abläuft.

In Rücksprache mit Herbert Walder, unserem Betreuer entschieden wir uns, diese Komponenten stattdessen für die Integration in ein MicroBlaze System aufzubereiten, damit sie in anderen Designs verwendet werden können. Dieser Bericht beschreibt die Funktionsweise dieser Komponenten und erklärt, wie sie in einem eigenen Design eingesetzt werden.

2 Datenblatt ps2core

2.1 Merkmale

- ermöglicht bidirektionale Kommunikation mit PS2-Devices (byteweise)
- funktioniert an zwei beliebigen Tristate-Ausgängen des FPGAs (externe Pullups werden benötigt)
- erkennt Übertragungsfehler (Paritätsfehler)
- Timeouts werden nicht erkannt
- mindest-Taktfrequenz: 100 kHz
- benötigt Referenz-Taktsignal (1 MHz)

2.2 Allgemeine Beschreibung

Diese Komponente ist eine VHDL-Implementierung des PS2-Datenübertragungsstandards. Sie ermöglicht die bidirektionale Kommunikation mit jeglichen PS2-Devices (z.B. Tastatur, Maus). Die Kommunikation erfolgt byteweise und kann nach erfolgtem Beginn einer Übertragung (senden oder empfangen) nicht unterbrochen werden, das Ende der Transmission muss abgewartet werden bevor eine neue Transmission (senden oder empfangen) begonnen werden kann. Gesteuert wird die Komponente über 4 Kontrollleitungen (2 Steuerleitungen, 2 Statusleitungen) mittels eines kleinen Protokolls.

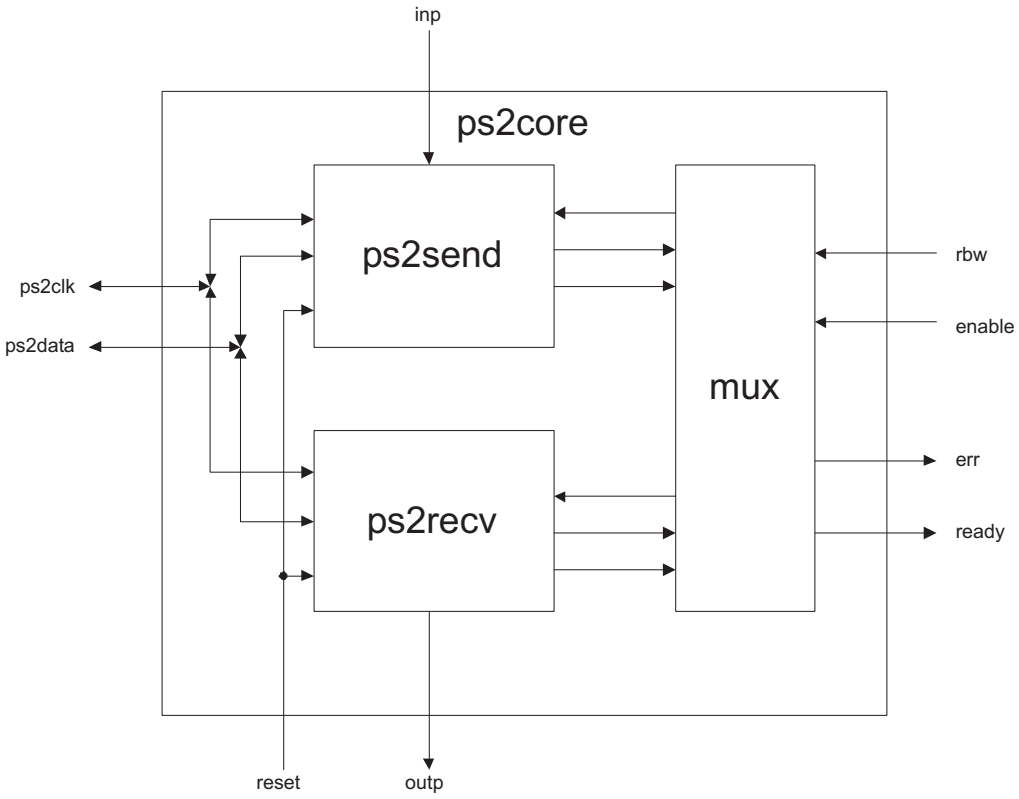


Abbildung 1: Blockdiagramm der ps2core-Komponente

2.3 Blockdiagramm

2.4 Interface

Symbol	Typ	Richtung	Beschreibung
clock	std_logic	I	Taktsignal; > 100 kHz benötigt
ref_clock	std_logic	I	Referenztakt; muss 1 MHz betragen
reset	std_logic	I	Reset Signal
ps2clk	std_logic	I/O	PS2-Taktleitung
ps2data	std_logic	I/O	PS2-Datenleitung
inp	std_logic_vector(7..0)	I	Input; zu sendendes Byte
outp	std_logic_vector(7..0)	O	Output; empfangenes Byte
rbw	std_logic	I	$\overline{\text{R}}$W; ’0’ = empfangen, ’1’ = senden
enable	std_logic	I	Enable-Leitung; ’1’ = enabled
ready	std_logic	O	Bereitschaftsstatus; ’1’ = bereit
err	std_logic	O	Fehlerstatus; ’1’ = Fehler; nur gültig wenn `ready` = ’1’

2.5 Funktionale Beschreibung

Nachfolgend wird das Steuerungsprotokoll der Komponente für das Senden und Empfangen von Daten ausführlich beschrieben.

1. Initialzustand
   `enable` = ’0’
   `rbw` nicht definiert

`ready` = ’1’
`err` nicht definiert

2. Empfangen
   Die Leitung `rbw` wird auf ’0’ gesetzt (empfangen), danach `enable` auf ’1’ gezogen um den Empfangsvorgang zu starten. Nach erfolgtem Start fällt `ready` auf ’0’ und wird wieder ’1’ sobald ein Byte erfolgreich empfangen wurde oder ein Fehler aufgetreten ist (ersichtlich an `err`). Die Werte von `err` und `outp` sind nur in dieser Phase (nach abgeschlossenem Empfangsvorgang) gültig, während des Empfangens können sie ändern.
   Anschliessend muss die Komponente durch setzen von `enable` auf ’0’ wieder in den Initialzustand versetzt werden. Das Timing-Diagramm dazu ist in Abb. 2 zu sehen.

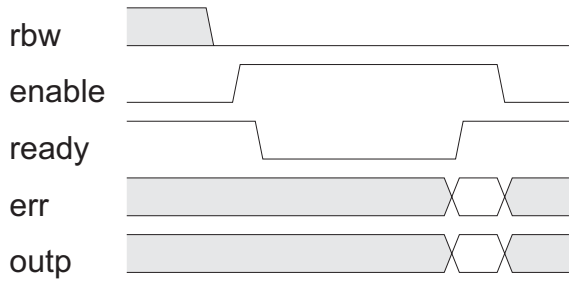


Abbildung 2: Timing-Diagramm für das Empfangen von Daten

3. Senden
   Das Senden von Daten erfolgt analog dem Empfangen. Zuerst wird wieder die Kommunikationsrichtung festgelegt indem `rbw` auf ’1’ gesetzt wird (senden). Nachdem das zu sendende Byte an `inp` angelegt wurde kann der Sendevorgang gestartet werden indem `enable` auf ’1’ gezogen wird. `ready` fällt nun auf ’0’ und bleibt während des Sendevorgangs ’0’. Der Eingang `inp` muss während des gesamten Sendevorgangs konstant gehalten werden! Nach abgeschlossenem Sendevorgang (mit oder ohne Fehler) wird `ready` wieder ’1’ und der Fehlerstatus kann abgelesen werden (`err`). Anschliessend muss die Komponente wieder in den Initialzustand versetzt werden (`enable` = ’0’). Das Timing-Diagramm dazu ist in Abb. 3 zu sehen.

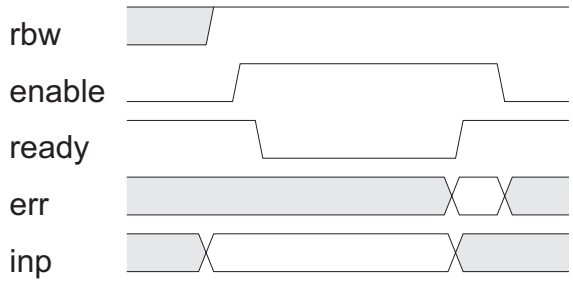


Abbildung 3: Timing-Diagramm für das Senden von Daten

3 Datenblatt ps2mouse

3.1 Merkmale

- eigenständiger PS2-Mousecontroller
- P'n'P: muss nicht konfiguriert oder initialisiert werden
- erkennt ob angeschlossenes PS2-Device eine Maus ist
- ermittelt ständig Mausbewegung und Maustasten-Status
- benützt ps2core für die Kommunikation mit der Maus
- mindest-Taktfrequenz: 100 kHz
- benötigt Referenz-Takt von 1 MHz

3.2 Allgemeine Beschreibung

Die ps2mouse-Komponente initialisiert eine an ihr angeschlossene PS2-Maus und stellt kontinuierlich die von der Maus gelieferten Positions- und Tasteninformationen an ihren Ausgängen zu Verfügung. Die Daten werden in einem Register zwischengespeichert, auf welches mit einem einfachen Protokoll über 2 Kontrolleitungen (1 Steuerleitung, 1 Statusleitung) zugegriffen werden kann.

Zur Kommunikation mit der Maus wird die Komponente ps2core verwendet. Fehler auf der Schicht der Byteübertragung werden abgearbeit und sind von Aussen nicht erkennbar. Die Komponente signalisiert lediglich einen Fehler, wenn das angeschlossene Device keine Maus ist.

3.3 Blockdiagramm

3.4 Interface

Symbol	Typ	Richtung	Beschreibung
clock	std_logic	I	Taktsignal; > 100 kHz benötigt
ref_clock	std_logic	I	Referenztakt; muss 1 MHz betragen
reset	std_logic	I	Reset Signal
ps2clk	std_logic	I/O	PS2-Taktleitung
ps2data	std_logic	I/O	PS2-Datenleitung
err	std_logic	O	Fehlerstatus; = '1' wenn Device keine Maus ist
next_avail	std_logic	O	= '1' wenn neue Daten vorhanden
get_next	std_logic	I	holt neuesten Wert
dx	std_logic_vector(8..0)	O	Bewegung in x-Richtung
x_overflow	std_logic	O	Überlauf in x-Richtung
dy	std_logic_vector(8..0)	O	Bewegung in y-Richtung
y_overflow	std_logic	O	Überlauf in y-Richtung
button_l	std_logic	O	Status der linken Maustaste
button_m	std_logic	O	Status der mittleren Maustaste
button_r	std_logic	O	Status der rechten Maustaste

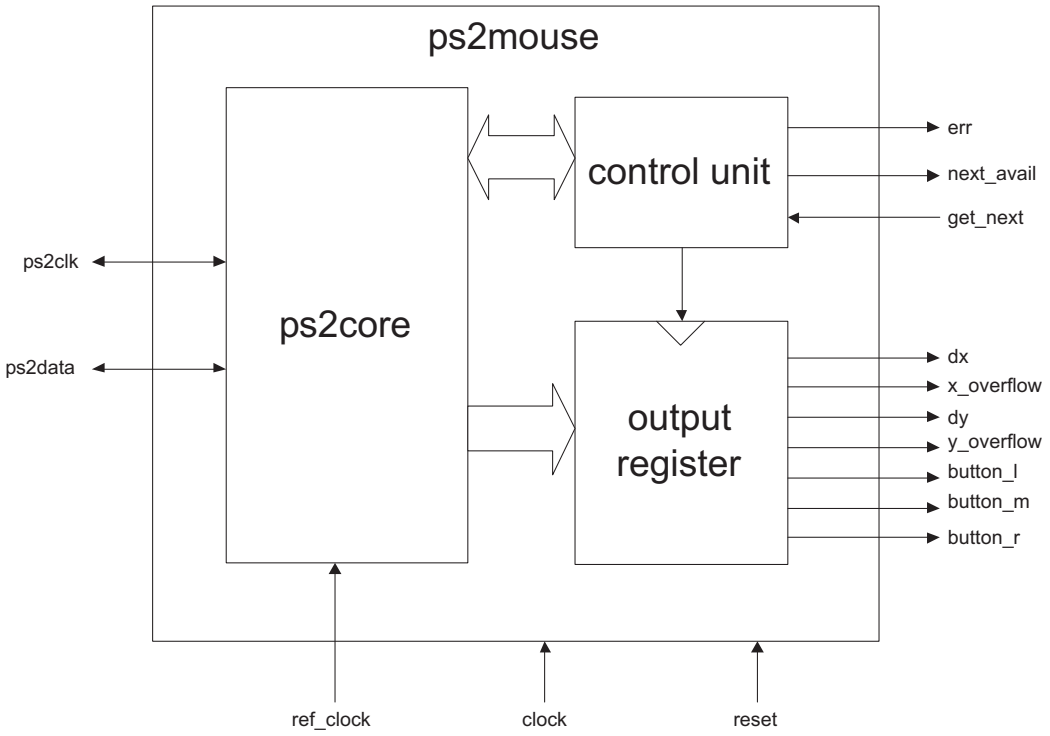


Abbildung 4: Blockdiagramm der ps2mouse-Komponente

3.5 Funktionale Beschreibung

3.5.1 PS2-Maus-Protokoll

Nach erfolgtem Reset oder Power-Up wird das *reset* Kommando an die Maus gesendet. Die Maus antwortet unverzüglich mit einem *ack* auf das Kommando, führt einen *BAT* (Basic Assurance Test) durch und sendet anschliessend das Resultat an den Host (0xAA bei Erfolg, 0xFC sonst). War der *BAT* erfolgreich sendet das Device anschliessend seine ID, im Falle einer Maus 0x00. Die Maus befindet sich nun im *Streaming Mode*, sendet von sich aus jedoch noch keine Positionsinformationen zum Host. Dies wird mit dem Kommando *enable data reporting* (0xF4) aktiviert. Die Maus sendet dann kontinuierlich Pakete mit 3 Byte Länge an den Host. Erfolgt keine Mausaktivität werden keine Pakete geschickt! Im Falle eines Fehlers während der Übertragung sendet der Host das *resend* Kommando (0xFE) an die Maus um das Paket nochmals zu erhalten. Auch wenn der Fehler erst im dritten Byte des Paketes erfolgt werden alle 3 Bytes nochmals gesendet. Wird ein Paket erfolgreich empfangen, wird es in das Output-Register geschrieben und es wird auf das nächste Paket gewartet.

3.5.2 Lesen der Daten

Die von der Maus empfangenen Daten werden in einem Register zwischengespeichert und können von dort abgeholt werden. Sind neue Daten verfügbar, meldet die Komponente dies indem sie die Leitung *next_avail* auf ’1’ setzt. Mit einem Impuls auf der Leitung *get_next* werden die neuen Daten auf den Ausgang gelegt. Das Timing-Diagramm dazu ist in Abb. 6 zu sehen.

Werden neue Daten nicht abgeholt bevor ein neues Paket eintrifft werden die noch nicht abgeholten Daten überschrieben! Das Überschreiben wird nicht angezeigt.

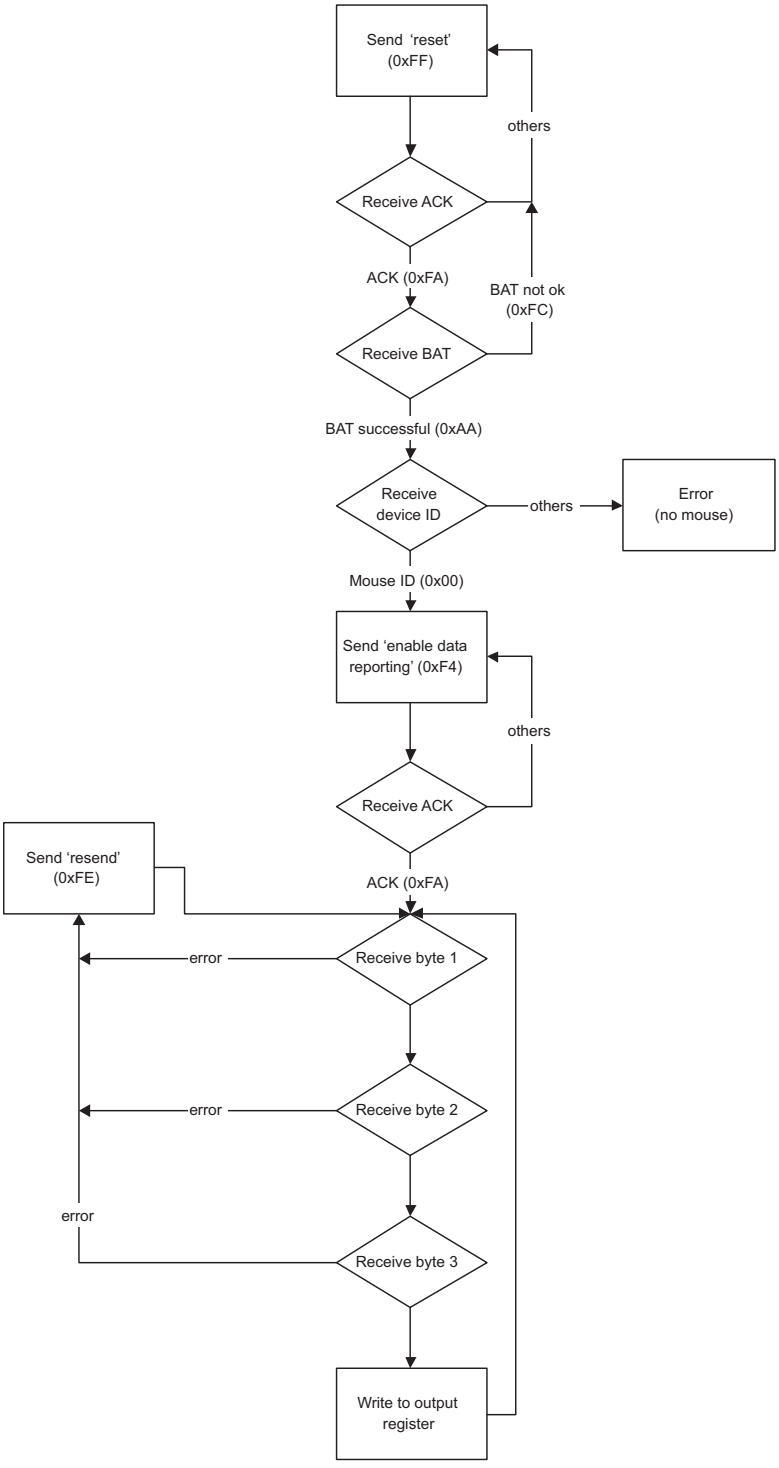


Abbildung 5: Funktionsdiagramm der ps2mouse-Komponente

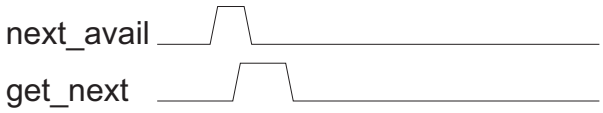
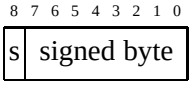


Abbildung 6: Timing-Diagramm für das Lesen der Daten

3.5.3 Datenformate

dx, dy Diese 9-bit Werte bestehen aus einem Sign-Bit s als MSB und einem *signed byte* (2er Komplement). Die Vorzeicheninformation ist also doppelt vorhanden! Das Sign-Bit ist lediglich der Vollständigkeit halber implementiert (es wird von der Maus gesendet) und kann ignoriert werden.



x_overflow, y_overflow signalisieren einen Überlauf des Positionszählers.

button_l, button_m, button_r entsprechen dem Status der jeweiligen Maustasten (links, mitte, rechts), wobei ’1’ eine gedrückte Taste bedeutet.

3.5.4 Bekannte Probleme

- Beim Testen der Komponente trat sporadisch der Fehler auf, dass sich der Mauscursor sprunghaft nach rechts bewegte, egal in welche Richtung man die Maus gerade bewegt. Die Ursache dieses Fehlers konnte nicht gefunden werden. Da wir keine genaue Dokumentation des PS2-Mausprotokolls zur Verfügung hatten, lediglich Third-Party-Dokumente von Personen, die sich ebenfalls mit PS2-Devices beschäftigt haben, kann die Ursache dort zu finden sein. Eine mögliche Erklärung ist, dass nach dem Senden des *resend* Kommandos zuerst ein *ACK* vom Device gesendet wird bevor die Re-Transmission des Datenpaketes erfolgt. Dies ist jedoch reine Spekulation.
- Zum Teil kam es vor, dass eine der FSM in einem Zustand landete, der nicht existierte, und obwohl mit einer *when others*-Anweisung solche Fälle abgedeckt wurden, stoppte sie trotzdem in diesem Zustand. Gelöst wurde dieses Problem, indem bei den Synthetisierungsoptionen die State-Kodierung auf einen anderen Wert als *auto* oder *one-hot* gesetzt wurde.

4 Datenblatt opb_ps2keyboard

4.1 Merkmale

- Arbeitsfrequenz 25 MHz
- Softcore mit OPB-Interface für *Xilinx EDK*
- ermöglicht das Auslesen der Scancodes einer PS2-Tastatur

4.2 Allgemeine Beschreibung

Der Softcore opb_ps2keyboard ist eine Kapselung der ps2core- Komponente in ein *Xilinx OPB Core Template*. Er ermöglicht das Empfangen der Scancodes einer angeschlossenen PS2-Tastatur. Das Senden von Kommandos an das Device ist nicht möglich. Durch die OPB-Schnittstelle lässt er sich einfach im *Xilinx EDK* in ein Design einbinden.

4.3 Blockdiagramm

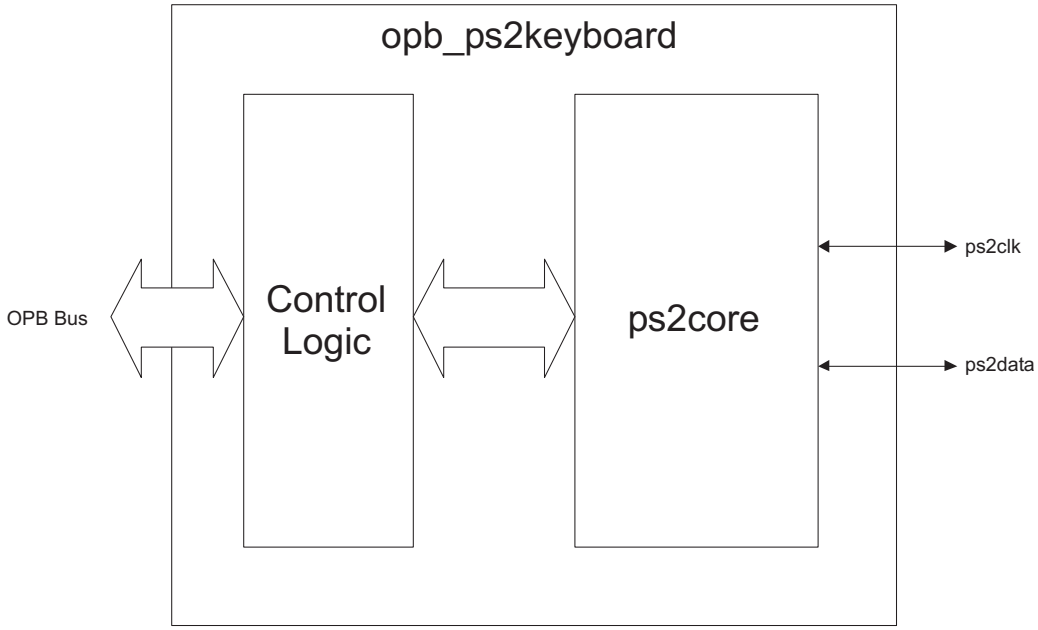


Abbildung 7: Blockdiagramm des opb_ps2keyboard Softcores

4.4 Interface

Zusätzlich zum OPB-Interface besteht das Interface aus den folgenden Leitungen (User I/O):

Symbol	Typ	Richtung	Beschreibung
ps2clk	std_logic	I/O	PS2-Taktleitung
ps2data	std_logic	I/O	PS2-Datenleitung

4.5 Funktionale Beschreibung

Dieser Softcore basiert auf der ps2core Komponente und ist lediglich mit einer kleinen Kontroll-Logik ausgestattet, die den ps2core so ansteuert, dass über den OPB-Bus komfortabel die empfangenen Daten ausgelesen werden können. Für eine ausührlichere Beschreibung siehe2.5.

4.6 Integrieren in ein Xilinx EDK Design

Die Sourcen des Cores sind auf der beiliegenden CD im Verzeichnis `/OPB Cores` gespeichert und können wie in Kapitel 7.1 beschrieben in ein eigenes Design integriert werden. Der Core benöntigt 1 Adresse für die Mausdaten sowie 0xFF Bytes für das Register MIR. Im 'Ports' Tab müssen die Ports `ps2clk` und `ps2data` hinzugefügt werden.

4.7 Programmieren

Das Lesen von Daten erfolgt, indem man auf der Basisadresse einen Lesevorgang initiert. Man erhält einen 32-bit Wert im folgenden Format:

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0	
unused	data	unused	n

Bits 8 bis 15 enthalten das vom Device gesendete Byte, das Bit 0 (`n`) signalisiert, ob die Daten schon mal gelesen ('0') wurden oder neu sind ('1').

4.8 Testumgebung

Um den Softcore ausserhalb einer *Microblaze*-Umgebung testen zu können, wurde eine eigenständige Testumgebung erstellt, die über den *IPIF*-Bus auf den Core zugreift. Die Testumgebung wurde *coretest* genannt, ist selbst ein VHDL-Modul und befindet sich auf der beiliegenden CD.

5 Datenblatt opb_ps2mouse

5.1 Merkmale

- Arbeitsfrequenz 25 MHz
- Softcore mit OPB-Interface für *Xilinx EDK*
- ermöglicht das Ansteuern einer PS2-Maus
- unterstützt Interrupt

5.2 Allgemeine Beschreibung

Der Softcore opb_ps2mouse ist eine Kapselung der ps2mouse- Komponente in ein *Xilinx OPB Core Template*. Er ermöglicht das Auslesen respektive Empfangen der von einer Maus gesendeten Positions- und Maustasteninformation. Das explizite Senden von Kommandos an das Device ist nicht möglich. Durch die OPB-Schnittstelle lässt er sich einfach im *Xilinx EDK* in ein Design einbinden.

5.3 Blockdiagramm

5.4 Interface

Zusätzlich zum OPB-Interface besteht das Interface aus den folgenden Leitungen:

Symbol	Typ	Richtung	Beschreibung
ps2clk	std_logic	I/O	PS2-Taktleitung
ps2data	std_logic	I/O	PS2-Datenleitung
interrupt	std_logic	O	Interrupt

5.5 Funktionale Beschreibung

Dieser Softcore basiert auf der ps2mouse Komponente und ist lediglich mit einer kleinen Kontroll-Logik ausgestattet, die die ps2mouse Komponente so ansteuert, dass über den OPB-Bus komfortabel die empfangenen Daten ausgelesen werden können. Der Core selbst benötigt Prinzipiell nur eine Adresse, das Template jedoch zusätzlich einen Adressraum von 256 Byte (Parametername `c_mir`). Auf der interrupt-Leitung wird signalisiert, ob sich ein neuer Wert im Puffer befindet. Für eine ausührlichere Funktionsbeschreibung siehe 3.5.

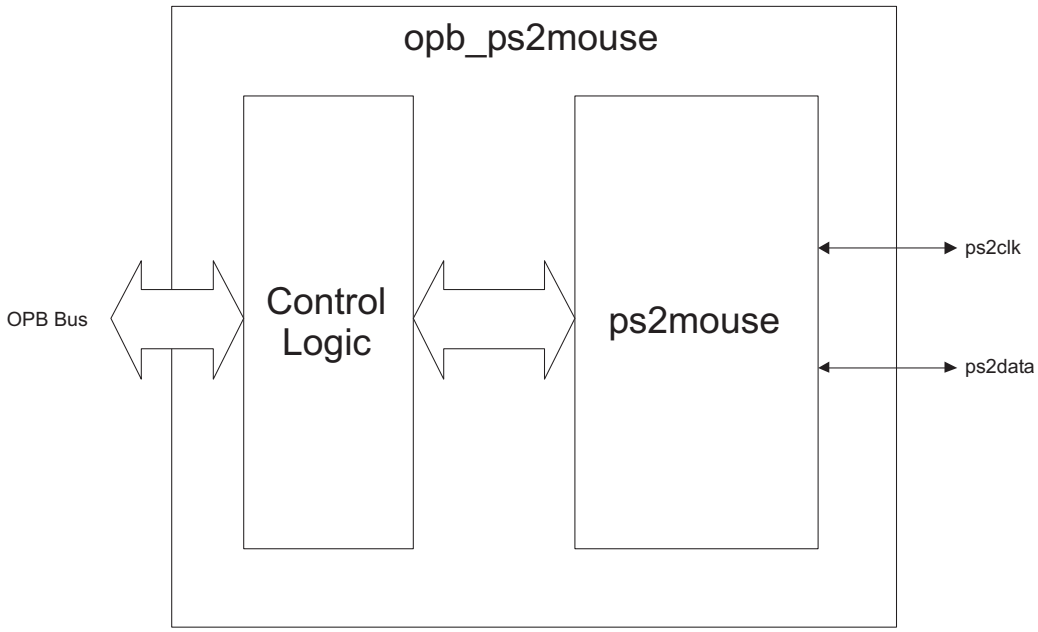


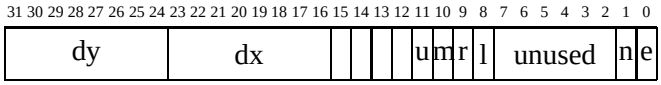
Abbildung 8: Blockdiagramm des opb_ps2mouse Softcores

5.6 Integrieren in ein Xilinx EDK Design

Die Sourcen des Cores sind auf der beiliegenden CD im Verzeichnis `/OPB Cores` gespeichert und können wie in Kapitel 7.1 beschrieben in ein eigenes Design integriert werden. Der Core benöntigt 1 Adresse für die Mausdaten sowie 0xFF Bytes für das Register MIR. Im 'Ports' Tab müssen die Ports `ps2clk` und `ps2data` hinzugefügt werden. Falls ein Interrupt Controller eingesetzt wird (z.B. *opb_intc*), muss auch der interrupt-Ausgang des Cores mit diesem verbunden werden.

5.7 Programmieren

Das Lesen von Daten erfolgt, indem man auf der Basisadresse einen Lesevorgang initiiert. Man erhält einen 32-bit Wert im folgenden Format:



Achtung! Bits 12-15 werden benutzt! Sie sind lediglich aus Layout-technischen Gründen nicht angeschrieben.

0 `error`: entspricht der *err*-Leitung der ps2mouse Komponente. Ist '1' wenn das angeschlossene Device keine Maus ist.

2 `new`: signalisiert ob die Daten bereits gelesen wurden ('0') oder neu sind ('1'). Hardwired mit der interrupt-Leitung.

8 `left_button`: Status der linken Maustaste ('1' = gedrückt)

9 `right_button`: Status der rechten Maustaste ('1' = gedrückt)

10 `middle_button`: Status der mittleren Maustaste ('1' = gedrückt)

11 unbenutzt

12 `x_sign`: Sign-Bit der x-Positionsänderung

13 `y_sign`: Sign-Bit der y-Positionsänderung

14 `x_overflow`: signalisiert Überlauf des x-Positionszählers

15 `y_overflow`: signalisiert Überlauf des y-Positionszählers

16-23 `dx`: Positionsänderung in x-Richtung, Signed Byte (2er Komplement)

24-31 `dy`: Positionsänderung in y-Richtung, Signed Byte (2er Komplement)

6 Datenblatt VGA Core

6.1 Merkmale

- Arbeitsfrequenz 25 MHz
- VGA Timing wird generiert
- Speicherinterface für 300 kByte Bildspeicher
- Bildauflösung von 640×480 Pixel mit 8 bit Farbtiefe
- Textspeicher für 80×30 Zeichen
- integrierter 8×16 VGA Font
- integrierter Mauscursor mit Screenmask und Cursormask
- Register für x- und y-Koordinaten der Maus

6.2 Allgemeine Beschreibung

Der VGA Core ist eine funktionsfähige VHDL-Beschreibung eines VGA Timing Generators, eines Textspeichers, eines Mauscursors und eines Bildspeicherinterfaces. Das Bildspeicherinterface ist für die Onboard SRAM Blöcke des XESS-boards ausgelegt und benutzt 19 Adress- sowie 16 Datenleitungen. Auf diesen RAMs lässt sich ein 640×480 Pixel grosses Hintergrundbild speichern. Für den Mauscursor lässt sich über zwei 9-bit Ports dessen x- und y-Koordinate setzten, der Core legt das Cursorbitmap selbständig an der gegebenen Koordinate über das Hintergrundbild.

6.3 Blockdiagramm

6.4 Interface

Symbol	Typ	Richtung	Beschreibung
clock	std_logic	I	Taktsignal; 25 MHz benötigt
reset	std_logic	I	Reset Signal
pixelclk	std_logic	O	Pixelclock for RAMDAC
hsyncb	std_logic	O	horizontales sync. Signal; VGA kompatibel
vsyncb	std_logic	O	vertikales sync. Signal; VGA kompatibel
blank	std_logic	O	blanking Signal für den Bt481 RAMDAC
index	std_logic_vector(7..0)	O	8 bit Farbwert für den RAMDAC

ram_data	std_logic_vector(15..0)	I/O	Datenleitungen für das externe Video RAM
ram_addr	std_logic_vector(18..0)	O	Adressleitungen für das externe Video RAM
ram_oe	std_logic	O	$\overline{\text{OE}}$ für das externe RAM
ram_cs	std_logic	O	$\overline{\text{CS}}$ für das externe RAM
ram_we	std_logic	O	$\overline{\text{WE}}$ für das externe RAM
mouse_x_pos	std_logic_vector(8..0)	I	X-Koordinate des Mauscursors
mouse_y_pos	std_logic_vector(8..0)	I	Y-Koordinate des Mauscursors
text_we	std_logic	I	synchrones WE für den Textspeicher
text_addr	std_logic_vector(11..0)	I	Adresse für die zu schreibende textspeicher Position
text_di	std_logic_vector(7..0)	I	zu schreibende Textdaten
text_do	std_logic_vector(7..0)	O	Textdaten an der aktuellen Adresse
wrb_ramdac	std_logic	O	$\overline{\text{WR}}$ Signal für den externen RAMDAC
rdb_ramdac	std_logic	O	$\overline{\text{RD}}$ Signal für den externen RAMDAC
data_ramdac	std_logic_vector(7:0)	O	Datenbus für Palettedaten im RAMDAC
rs_ramdac	std_logic_vector(2:0)	O	'Register Select' des RAMDAC

6.5 Funktionale Beschreibung

6.5.1 Video RAM

Das Video RAM Interface erlaubt die Darstellung eines 640×480 Pixel grossen Hintergrundbildes mit einer Farbtiefe von 8 bit. Das Interface ist so konzipiert, dass es direkt an die RAM Blöcke des XESS Boards angehängt werden kann und das gesamte RAM Timing übernimmt. Da das XESS-RAM mit einer Datenbreite von 16 bit ausgelegt ist, enthält der VGA Core eine Logik, die gleich zwei Pixel aus dem RAM liesst und quasi cacht. Auf diese Weise könnte der RAM während jedes zweiten Zyklus andersweitig benutzt werden.

In der aktuellen Implementation ist es nicht möglich, das RAM zu beschreiben, während dieses zur Bilddarstellung benutzt wird. Obwohl man im Prinzip während des horizontalen und des vertikalen Rücklaufs des Elektronenstrahls Zeit hätte, auf das RAM zu schreiben, ist dieses Feature (noch) nicht implementiert worden. Eine Umsetzung hätte zuviel Zeit benötigt und wurde nicht genauer studiert, da das RAM-Interface des XESS-Boards immer auf zwei Pixel gleichzeitig zugreift, was beim Schreiben etwas komplexere Abläufe verursacht.

Um ein Hintergrundbild darzustellen, muss dieses in der aktuellen Version vor der eigentlichen Arbeit in die RAM-Blöcke geladen werden. Dafür wird von XESS eine entsprechende Option in ihrem Downloadtool gsxload angeboten. Die eigenen Bilder müssen zu diesem Zweck in einer 640×480 Pixel grossen BMP Datei gespeichert sein. Mit dem 'Gandalf' Programm (Auf der CD unter `/Software` abgelegt) wird das gewünschte BMP Bild geladen und mit der Option 'convert image' in einer binären Datei gespeichert. Die binäre Datei kann nun mit einem beliebigen bin2hex Konverter in INTEL-HEX umgewandelt werden und diese Daten können direkt an gsxload gefüttert werden.

6.5.2 Textspeicher

Die Darstellung von Text ist auf verschiedene Teilkomponenten des VGA Cores verteilt. Zum einen enthält das System eine ROM Tabelle mit einer vollständigen 256 Zeichen umfassenden ASCII Schriftart mit 8×16 Pixeln pro Zeichen. Andererseits ist ein 80×30 Zeichen grosser Textspeicher implementiert, der die Dualport RAMs des Virtex FPGAs ausnutzt um gleichzeitig Text aus dem Textspeicher darstellen und Zeichen ändern zu können.

Für das Erstellen der Schrift wurde ein Tool entwickelt, das Console-Schriftdateien von UNIX in VHDL Code für eine ROM Beschreibung übersetzt. Das entsprechende Tool 'FontConverter' ist auf der beiliegenden CD im Verzeichnis `/Software` zu finden. Die benötigten Parameter werden bei

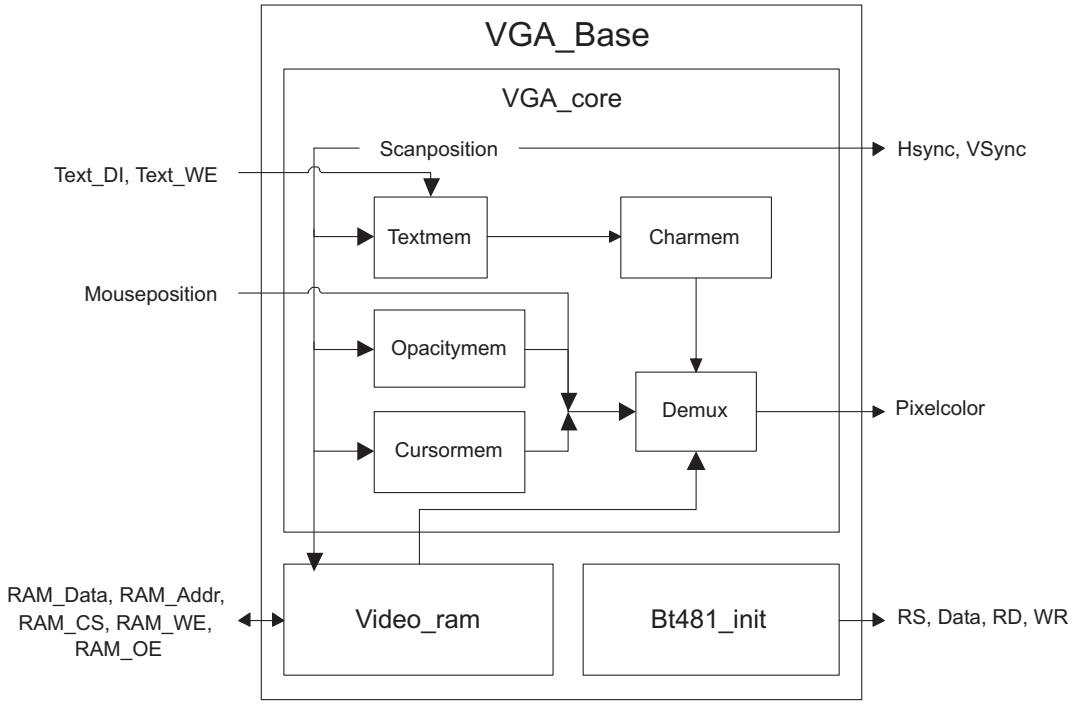


Abbildung 9: Blockdiagramm des VGA Core

einem Aufruf ohne Parameter ausgegeben. Als Inputfile dient z.B. das ebenfalls mitgelieferte Fontfile `vt220l.816`.

6.5.3 VGA Timinggenerator

Der Timinggenerator ist in den VGA_Core Teil der Komponente integriert und erzeugt, basierend auf einem 25 MHz Clock ein akzeptables VGA Timing. Die korrekte Frequenz für VGA wäre 25.175 MHz, das XESS-Board bietet aber nur ganzzahlige Teiler für 100 MHz Ausgangsfrequenz. Die Basis für diesen Code war ein Beispiel von XESS, das angepasst wurde um Bilder mit 640×480 Pixeln und 8 bit Farbtiefe darzustellen.

6.5.4 Bt481_init

Dieser Teil des VGA Cores ist für die Initialisierung des externen RAMDAC zuständig. Es wird eine Standardpalette konfiguriert, die auch bei der Konvertierung von BMP Bildern in HEX Dateien in 'Gandalf' benutzt wird. Für die Einträge in die Palette werden die 8 bit Farbwerte in 3 Teile unterteilt, wie dies in Abb. 10 gezeigt wird. Die einzelnen Farbanteile werden nun wie in Abb. 11 dargestellt, zu 8 bit breiten Werten erweitert um so volle 24 bit Farben zu erhalten.

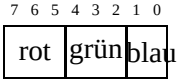


Abbildung 10: Aufbau eines 8 bit Farbwertes

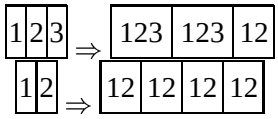


Abbildung 11: Umrechnung der 2 und 3 bit Farbwerte in 8 bit Anteile einer 24 bit RGB Farbe

6.5.5 Mauscursor

Für die Darstellung des Mauscursors enthält die VHDL-Beschreibung des VGA Cores zwei ROM-Tabellen mit der Screenmask und der Cursormask des Mauscursors, der Mauscursor kann also zur Laufzeit nicht mehr geändert werden. Die Cursormask besteht aus 16×16 Einträgen, die die Farbe des Mauscursor angeben (der Mauscursor kann also auch farbig sein). Die Screenmask besteht aus einem ebenfalls 16×16 Einträgen grossen Bitfeld, das angibt ob das entsprechende Pixel durchsichtig sein soll oder nicht. Das System stellt den Mauscursor selbstständig an der durch `MOUSE_X_POS` und `MOUSE_Y_POS` gegebenen Position dar. An diesen Ports muss die Position konstant anliegen, da die Komponente dafür selbst keine Register zur Verfügung stellt.

6.6 Einsatz im Xilinx EDK

Der VGA Core lässt sich sehr leicht in eine MicroBlaze Umgebung einbinden und erlaubt so den softwareseitigen Zugriff auf den Textspeicher und die Mauscursor Position. Die nächsten beiden Kapitel erklären, wie man den Core einbindet und wie er programmiert wird.

6.6.1 Integrieren

Der VGA Core hat keine besonderen Anforderungen ans Integrieren und kann gemäss der Anleitung im Kapitel 'Einbinden eines neuen Cores ins eigene Design' integriert werden. Die Grösse des benöntigten Adressraums beträgt 0xFFF Bytes und im 'Ports' Tab werden alle Ports des VGA Cores als externe Peripherie benötigt. Beim VGA Core sind auch die `c_mir...` Parameter zu setzten. Die Sourcen sind auf der CD unter `/OPB Cores` gespeichert.

6.6.2 Programmieren

Das Setzten der Mausposition erfolgt durch Beschreiben der Adresse 'Basisadresse des VGA Core' + 0x960. Dies ist die erste freie Adresse nach dem Textspeicherbereich und sie wurde deshalb gewählt, um den Zugriff auf den Textspeicher einfach zu halten. Die Mausposition muss im folgenden Format geschrieben werden:

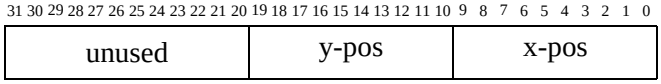


Abbildung 12: Aufbau des Mausposition Registers

Das Beschreiben des Textspeichers verläuft analog zu Videospeicherzugriffen auf dem PC. Soll ein Zeichen an die Textposition (x, y) auf den Bildschirm geschrieben werden (wobei wir x und y mit 0 beginnend zählen und die Position $(0, 0)$ die linke obere Bildschirmecke sei), dann schreiben wir dieses einfach an die Adresse 'Basisadresse des VGA Core'$+80 \times y + x$. Ein Beispiel für die Programmierung kann im Code der MB_VGACore_Mouse Demoapplikation nachgesehen werden.

6.7 Testbench

Um den VGA Core effizient testen zu können, wurde eine einfache Testumgebung entwickelt, die das schnelle Überprüfen der erzeugten Bilder erlaubt. In einer einfachen Loop wird ein einzelnes Bild des VGA Core in eine Datei gespeichert, die dann mit dem Betrachter 'Gandalf' überprüft werden kann. In 'Gandalf' verwendet man die Option 'open RAW image' und wählt das erzeugte Bild (standardmässig unter `testimage.txt` gespeichert).

7 Kleines EDK Tutorial

Der Designflow im EDK (Embedded Development Kit) umfasst sehr viele zum Teil sehr komplexe Schritte, die viele Möglichkeiten für Fehler beeinhalten. Während unserer Arbeit haben wir verschiedene solcher Stolperfallen angetroffen und gelernt diese zu umgehen. In diesem Kapitel wollen wir diese Probleme aufzeigen und verschiedene Schritt für Schritt Anleitungen geben, damit diese Fehler nicht wiederholt werden müssen.

7.1 Einbinden eines neuen Cores ins eigene Design

1. Bei Verwenden eines eigenen Cores, die Sourcen desselben vor dem Start des EDK ins Unterverzeichnis `/pcores` des Projekts kopieren.

2. Unter 'Add/Edit Cores...' → 'Peripherals' den gewünschten Core hinzufügen, einen eindeutigen Adressraum mit angemessener Grösse (häufig reichen hier 0xFF Bytes, im Zweifellsfall konsultiere man aber die Datenblätter des verwendeten Cores) vergeben und im Tab 'Bus Connections' mit dem OPB Bus verbinden.

   Es gibt auch Komponenten, die nicht an den OPB angehängt werden müssen, Details dazu findet man in den Anleitungen zu den Cores.

3. Im Tab 'Ports' müssen die benötigten Ports des Cores hinzugefügt werden und in der Spalte 'Range' mit der korrekten Busbreite ergänzt werden, z.B. `[7..0]` für einen 8 bit breiten Bus. Auch dazu müssen in der Regel Datenblätter konsultiert werden sofern der Name des Ports nicht bereits dessen Funktion vollständig erklärt.

4. Ebenfalls im 'Add/Edit Cores...' Dialog, unter dem Tab 'Parameters' müssen für eigene Cores die beiden Parameter `c_mir_BASEADDR` und `c_mir_HIGHADDR` gesetzt werden. Für diese MIR Register wird ein Adressbereich von 0xFF Byte benötigt, der sonst von keinem anderen Core belegt ist. Werden diese Adressen nicht gesetzt führt dies zu regelmässigen Resets auf dem OPB.

7.2 Schritte zum Konfigurationsbitstream

1. Wenn man bereits zuvor Netzlisten erzeugt oder Sourcen kompiliert hat, räumt man am besten über Tools → Clean → All erstes mal sein ganzes Projekt etwas auf.

2. Nun wird die Netzliste für die Hardwarebeschreibung generiert. Dazu wählt man Tools → Generate Netlist oder drückt den entsprechenden Button in der Toolbar.

3. Als nächstes muss die so generierte Netzliste in ein Xilinx ISE Projekt exportiert werden. Dazu überprüft man das Ausgabeverzeichnis unter Options → Project Options im Tab 'Hierarchy and Flow'. Im unteren Teil des Dialogs wählt man ISE (ProjNav) und setzt das NPL File auf `system.npl` im Unterverzeichnis `/projnav` des Projekts. Dieses Verzeichnis muss bei einem neuen Projekt meist erst erzeugt werden. Jetzt kann über Tools → Export to Projnav die Netzliste exportiert werden.

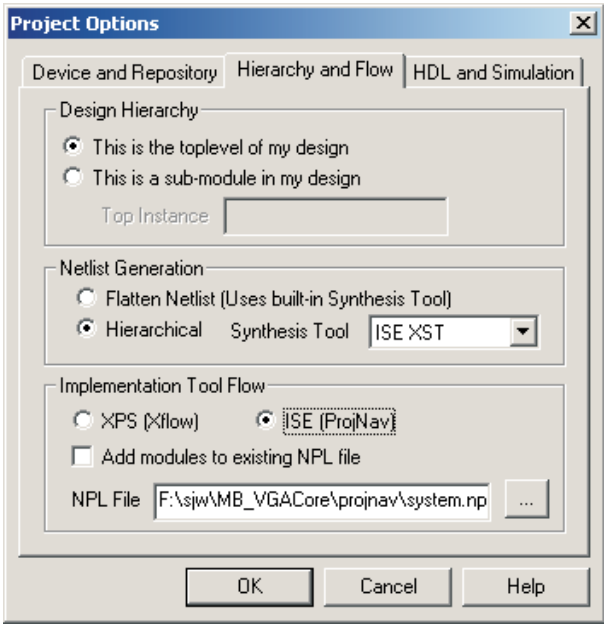


Abbildung 13: Dialog ’Project Options’ für Export von NPL Dateien

4. Die erstellte `system.npl` Datei wird in Xilinx ISE geöffnet und man fügt ein UCF File hinzu oder erstellt ein neues. Jetzt lässt man mit ’Generate Programming File’ im Project Navigator den Bitstream erzeugen.

5. Das generierte Bitstreamfile muss nun wieder ins EDK importiert werden. Dazu wählt man Tools → Import from ProjNav... und wählt dort die entsprechenden Dateien. Einerseits ist dies `system.bit` aus dem Unterverzeichnis `/projnav` und andererseits `bram_init.bmm` aus dem Unterverzeichnis `/implementation`. Diese Einstellungen müssen beim Laden einer neu übersetzten Netzliste wieder vorgenommen werden, da sonst die falschen Dateien verwendet werden.

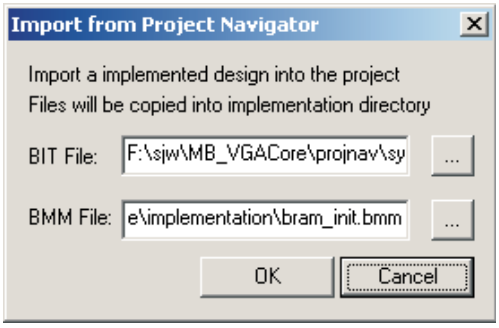


Abbildung 14: Dialog ’Import from Project Navigator’

6. Abschliessend wählt man die Option Tools → Update Bitstream. Dadurch werden die Libraries generiert, die Software kompiliert und der Bitstream auf den neuesten Stand gebracht, sodass die Software auf die Block RAMs konfiguriert wird.

7. Nun kann die Datei `download.bit` im Unterverzeichnis `/implementation` auf den FPGA

heruntergeladen werden.

7.3 Tricks und Fallen beim EDK Komponentendesign

- Für ein einfaches les- und schreibbares Register am OPB hat sich folgendes Schema bewährt:

```
if (Bus2IP_Clk'event and Bus2IP_Clk = '1') then
  if ((Bus2IP_CS = '1') and (Bus2IP_WrCE = '1')) then
    reg <= Bus2IP_Data;
  elsif ((Bus2IP_CS = '1') and (Bus2IP_RdCE = '1')) then
    IP2Bus_Data <= reg;
  end if;
end if;
```

  Dabei entsprechen die `Bus2IP`... und `IP2Bus`... Signale jenen aus den OPB Core Templates.

  Das `Bus2IP_CS` Signal geht auf high sobald der eigene Adressbereich angesprochen wird. Innerhalb dieses Adressbereichs können bestimmte Adressen aber immer noch speziell behandelt werden, dies wird zum Beispiel beim VGA Core gemacht, um die Mausposition zu setzten. Für Details sei auf die Sourcen verwiesen.

- In Schritt 5 zum Herstellen des Konfigurationsbitstreams müssen das BIT File und das BMM bei jedem neuen Importieren wieder explizit ausgewählt werden, da sonst die falschen Dateien benutzt werden.

- Im Unterverzeichnis `/implementation` existieren nach dem ganzen Prozess zwei .BIT Dateien: `download.bit` und `system.bit`. Die Datei `system.bit` enthält nur die Hardwarebeschreibung, ohne Inhalt der RAMs. Wenn diese Datei zur FPGA-Konfiguration verwendet wird, funktioniert die ganze Hardware wie erwartet, aber die Effekte der Software sind nicht sichtbar. Also, immer die richtige Datei verwenden.

- Das EDK warnt einen nicht, wenn man in Schritt 3 von 'Einbinden eines neuen Cores' eine falsche oder gar keine Busbreite angibt. Wenn in der resultierenden Anwendung Busse nur teilweise richtig reagieren, sollte man diese Einstellung nochmal überprüfen.

8 Demoapplikationen

8.1 vga_base_with_mouse

Diese Anwendung demonstriert den Einsatz der Basiskomponenten VGA Core und PS2Mouse im standalone Betrieb. Die beiden Komponenten sind über eine Protokol-FSM für die PS2 Komponente miteinander verbunden, um ständig die aktuelle Mausposition zu berechnen und an den VGA Core weiterzugeben.

Das System hat in dieser Form keinen praktischen Nutzen und dient lediglich der Demonstration der Möglichkeiten aller Komponenten. Es sollte aber sehr einfach sein, Erweiterungen zu schreiben, sodass zum Beispiel bei einem Click das Zeichen an der aktuellen Mausposition geändert wird, um so mehr und mehr Interaktion hineinzubringen.

8.2 MB_VGACore_Mouse

Diese Applikation implementiert im Prinzip die gleiche Funktionalität, wie die obige Anwendung, verwendet dafür aber den MicroBlaze zur Kontrolle sowie den VGA Core den PS2Mouse Core für ihre

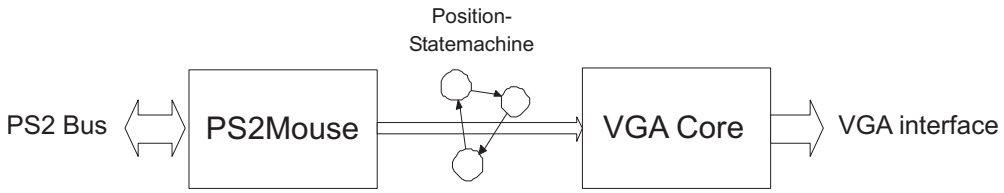


Abbildung 15: Aufbau der ersten Demoapplikation

respektiven Funktionen. Der C Code der Demo zeigt, wie einfach der Zugriff auf den Textspeicher möglich ist, mit wenigen Zeilen Code lässt sich das berühmte 'Hello World' Programm schreiben.

9 Zusammenfassung / Schlussbemerkung

Es ist uns gelungen im Verlaufe dieser Arbeit drei fast vollständig funktionierende Komponenten für den Einsatz in einem EDK Projekt zu entwickeln. Wir haben während unserer Arbeit viele Erfahrungen mit dem Xilinx EDK gesammelt und diese in diesem Bericht zusammengefasst. Wir sind der Meinung, dass es mit der hier gesammelten Erfahrung möglich sein sollte, sehr schnell eigene Designs aufzubauen. Gerade der VGA Core bietet zum Beispiel eine gute Alternative für Debuggingoutput, wenn der Einsatz eines RS232 Anschlusses nicht in Frage kommt oder nicht möglich ist.

In weiterführenden Arbeiten wäre es nun sehr interessant, das Videomemory Interface zu erweitern, um auch schreiben zu können sowie die erwähnten PS2 Probleme zu beheben. Auch das Herunterladen von Hintergrundbildern war sehr mühsam oder gar unmöglich und könnte zum Beispiel durch eine andere Downloadvariante, eventuell sogar zur Laufzeit des Systems, verbessert werden.

An dieser Stelle wollen wir uns noch bei den Leuten bedanken, die uns während der Arbeit unterstützt haben und uns mit wertvollen Hilfen zur Seite standen. An erster Stelle ist hier natürlich unser Betreuer Herbert Walder zu erwähnen, der diese Arbeit überhaupt erst möglich machte und uns jederzeit mit Rat und Tat zur Seite stand. Wir danken aber auch Matthias Dyer, der Herbert während seiner Abwesenheit vertreten hat. Und natürlich danken wir auch Professor Thiele, der Dienstgruppe, sowie unseren Kollegen, die bei anderen Semesterarbeiten zum Teil sehr ähnliche Probleme wie wir hatten.

10 Inhalt der CD

`/Bericht` LaTeX-Quellcode dieses Berichts sowie EPS Dateien für die darin enthaltenen Abbildungen.

`/EDK Projekte` Enthält nur das EDK Projekte MB_VGACore_Mouse, das die Möglichkeiten der EDK Komponenten demonstriert.

`/ISE Projekte` Projekte, die zum Erstellen und Testen der Komponenten verwendet wurden. Aus den Quellen in den Unterverzeichnisen können auch eigene Standaloneanwenungen entwickelt werden, die keinen Microblaze verwenden.

`/OPB Cores` Die Unterverzeichnise in diesem Verzeichnis sind gedacht, um sie direkt ins `/pcores` Verzeichnis eines eigenen Projekts zu kopieren.

`/Praesentation` Powerpointdatei der 15 minütigen Präsentation.

`/Software` PC Tools, die während der Arbeit eingesetzt wurden, um die Arbeit zu erleichtern. Es handelt sich dabei um das Bildkonvertierungsprogramm 'Gandalf', dass aus dem Verezeichnis `/memtool` mit `java image.Gandalf` starten lässt sowie das FontConverter Tool, das mit `java FontConverter` aus dem Verzeichnis `/FontConverter` gestartet wird.

Literatur

[1] Andres Erni, Stephan Reichmuth: *Inter-Task-Communication in Reconfigurable Operating Systems*, Diplomarbeit, ETH Zürich, März 2003

[2] *XSV Board V1.1 Manual*, XESS Corporation, Mai 2001, http://www.xess.com

[3] *MicroBlaze Processor Reference Guide*, Xilinx, EDK (v3.2) Februar 2003, http://www.xilinx.com

[4] Adam Chapweske: *PS/2 Mouse/Keyboard Protocol*, Webseite, http://panda.cs.ndsu.nodak.edu/~achapwes/PICmicro/PS2/ps2.htm

[5] Tomi Enqdahl: *VGA timing information*, Webseite, http://www.epanorama.net/documents/pc/vga_timing.html

Zürich, den 8. Juli 2003

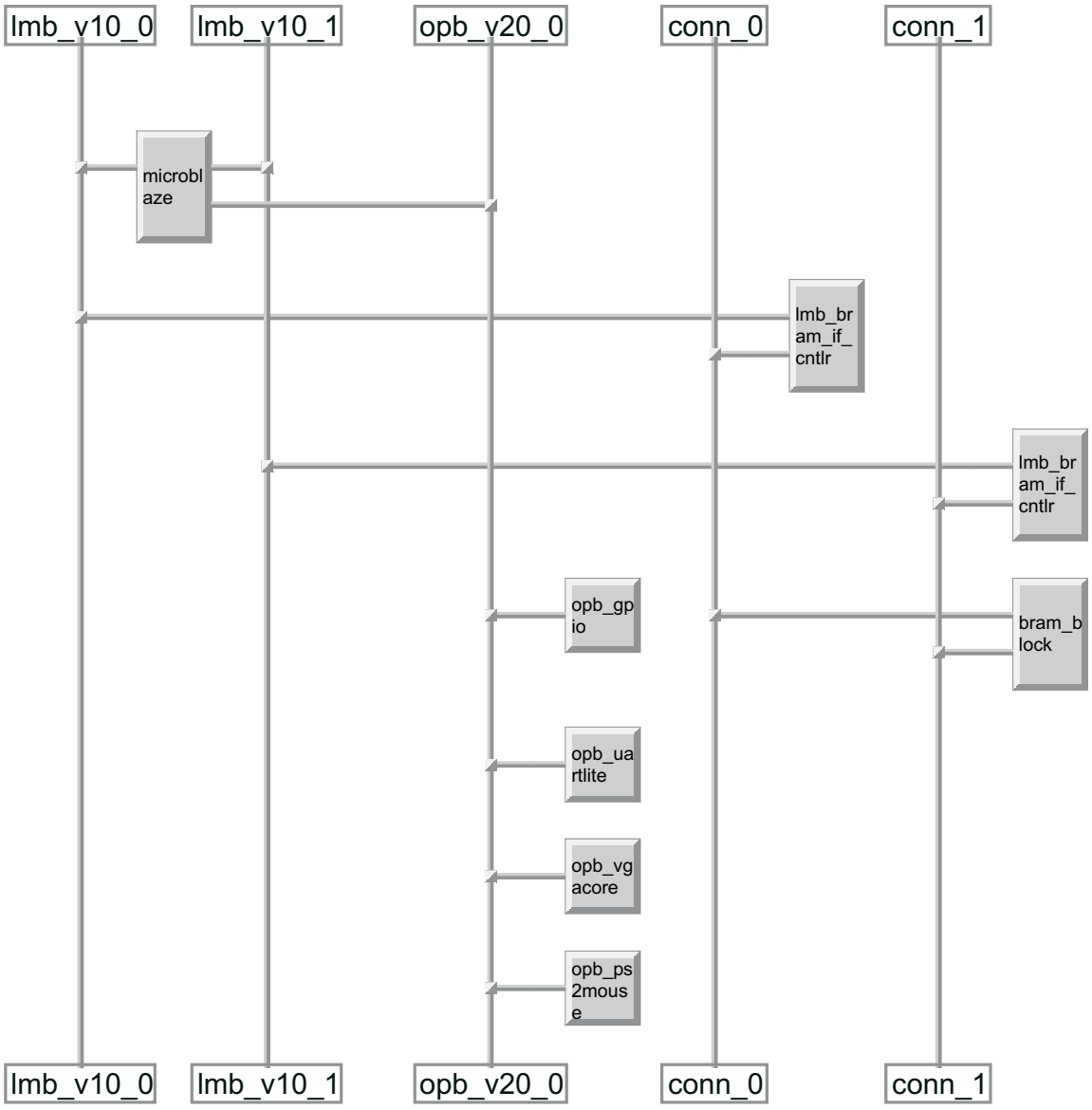


Abbildung 16: Aufbau der zweiten Demoapplikation (Bild aus dem EDK)

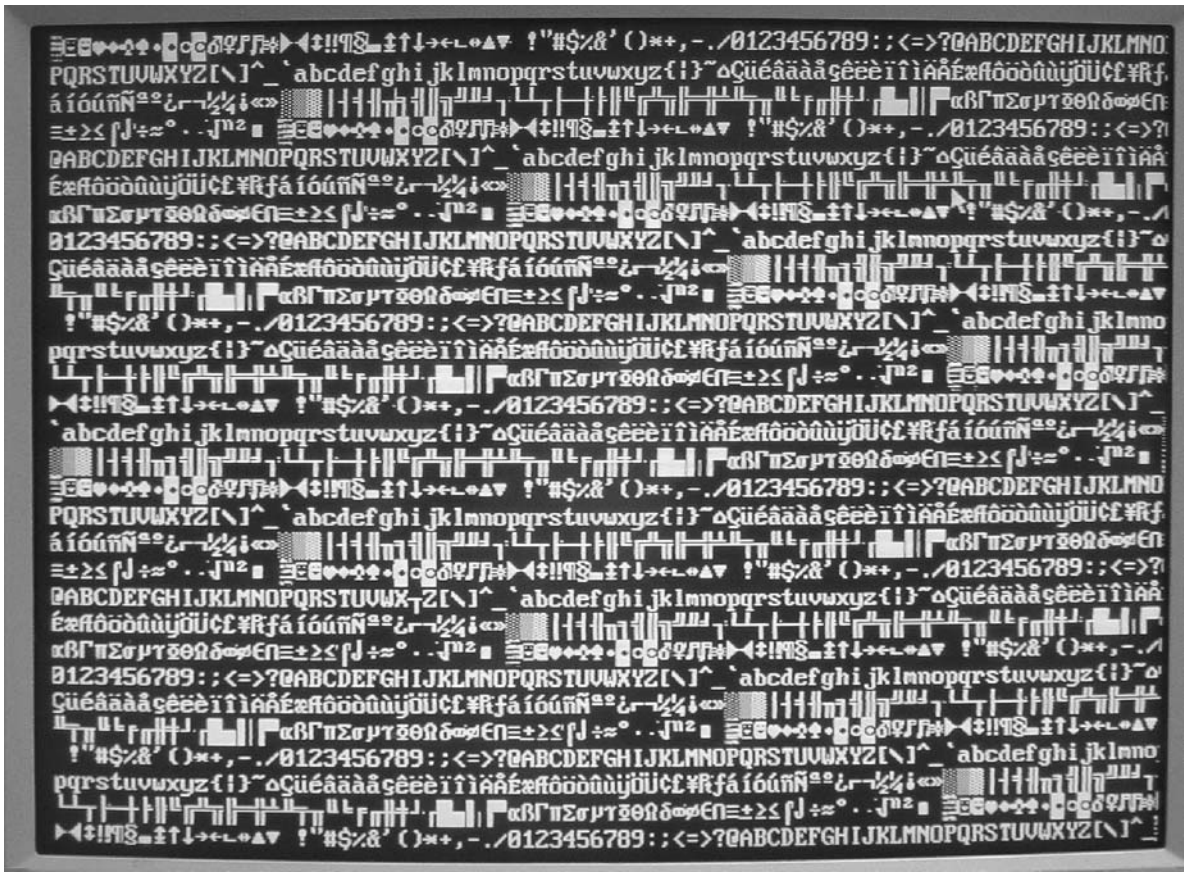


Abbildung 17: Screenshot der Demoapplikation MB_VGACore_Mouse