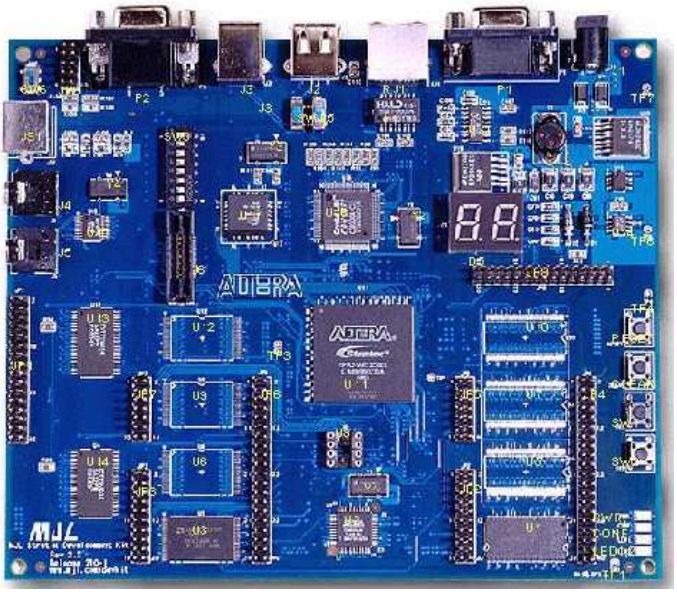
Skriptum zur Laborübung



D i g i t a l D e s i g n

LVA-Nr.: 182.045

A. Steininger, T. Handl

Technische Universität Wien
Institut für Technische Informatik (182/2)
1040 Treitlstr. 3, 2. Stk.

Stand: 11. September 2007

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

LU-SKRIPTUM — Kapitel 1

Aufgabenstellung

Inhalt

1.1 Einleitung

Dieses Dokument enthält alle zur Durchführung der Laborübung *Digitales Design* notwendigen Informationen und ist wie folgt strukturiert. Kapitel eins enthält neben der Vorstellung des Übungs-Designs und Hinweisen zur Übungsdurchführung die Angaben der vier Übungsbeispiele. In Kapitel zwei wird das verwendete FPGA-Board vorgestellt, eine Einführung in die Bedienung des Logikanalysators findet sich in Kapitel drei. Der Design-Flow wird in Kapitel vier erklärt, sowie die verwendete Software in Kapitel fünf. Erläuterungen zur Möglichkeit des Remote Learning befinden sich in Kapitel sechs. Abschließend sind in Anhang A Hinweise zum Testen zusammengefaßt. Ein Arbeitsplatz ist in Abbildung 1.1 dargestellt.

Abbildung 1.1: Ein Arbeitsplatz

1.2 Die Verzeichnisstruktur

Ihr Home-Verzeichnis trägt die Bezeichnung `dide_nn`, wobei `nn` durch Ihre Gruppennummer ersetzt wird. Darunter befinden sich, wie Abbildung 1.2 zeigt, die vier Unterverzeichnisse `bsp_1`, `bsp_2`, `bsp_3` und `bsp_4`, also jeweils ein Verzeichnis pro Aufgabenstellung.

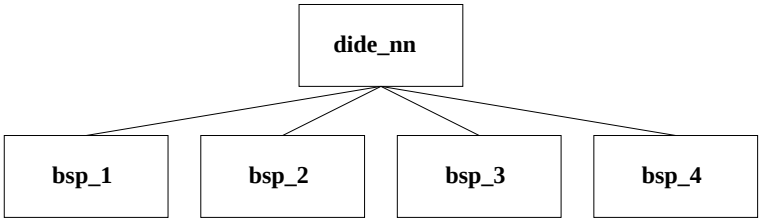


Abbildung 1.2: Das Home-Verzeichnis

Die für die Lösung der jeweiligen Aufgaben benötigten Files finden Sie in diesen Unterverzeichnissen Ihres Home-Verzeichnis. Abbildung 1.3 zeigt die Verzeichnisstruktur einer einzelnen Aufgabenstellung. Vier derartige Verzeichnis-Bäume befinden sich also im Home-Verzeichnis jeder Übungsgruppe.

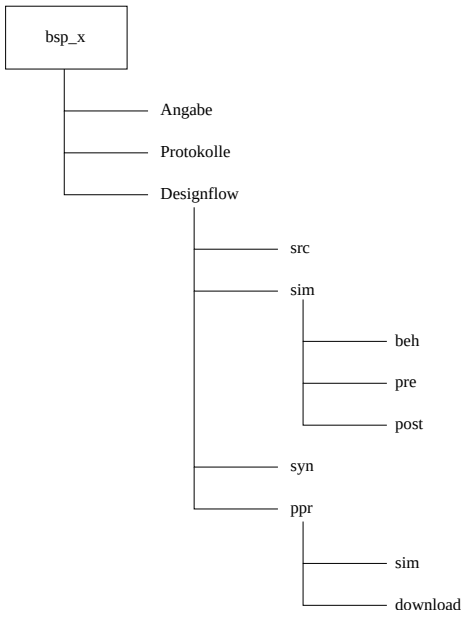


Abbildung 1.3: Die Verzeichnisstruktur

Alle von uns zur Verfügung gestellten Dateien sowie die meisten Verzeichnisse sind für Schreibzugriffe gesperrt. Dies soll verhindern, daß versehentlich Source-Files überschrieben werden. Um die Dateien zwecks Lösung der Aufgaben zu bearbeiten,

müssen sie zuerst vom Verzeichnis `Angabe` in das Verzeichnis `src` kopiert werden. In diesem Verzeichnis wird die Aufgabe dann bearbeitet.

Achtung, Ausnahmen: Für das erste Beispiel müssen Sie nur die `.sof`-Datei in das Verzeichnis `dide_nn/bsp_1/Designflow/ppr/download/` kopieren; bei den übrigen drei Beispielen muß zusätzlich zur oben beschriebenen Vorgangsweise das File `vga_pll.tcl` in das Verzeichnis `dide_nn/bsp_x/Designflow/ppr/download/` kopiert werden. Dafür existiert hier a priori kein `.sof`-File, da dies von Ihnen zu erstellen ist (mehr dazu später).

Das zweite Verzeichnis, bei dem Sie Schreibberechtigung besitzen, ist das Verzeichnis `protokolle`. Wie der Name schon sagt, enthält dieses Verzeichnis ein Latex-Gerüst für das Protokoll. Hier sollte dieses auch abgelegt werden (als PDF). Die folgenden Abschnitte beschreiben die Unterverzeichnisse der einzelnen Übungsaufgaben.

Angabe: In diesem Verzeichnis befinden sich die Source-Files des Beispiels sowie - falls notwendig - eine Datei mit Namen `dide_nn_m.txt`. Die Buchstaben *nn* bezeichnen dabei Ihre Gruppennummer, die Ziffer *m* gibt die Nummer des betroffenen Beispiels an.

Protokolle: Hier befindet sich ein Latex-Gerüst, welches als Hilfe bei der Abfassung der Protokolle gedacht ist.

Designflow: Dieses Verzeichnis enthält alle Unterverzeichnisse, welche notwendig sind, um den Design Flow zu durchlaufen.

src: Der Source-Code des Projekts wird im Verzeichnis `src` abgelegt. Dabei gilt die folgende Konvention: jedes Modul besteht aus den Dateien `<module_name>_ent.vhd`, `<module_name>_arc.vhd` und gegebenenfalls `<module_name>_pak.vhd`. Die Testbenches tragen die Namen `<module_name>_beh_tb.vhd`, `<module_name>_pre_tb.vhd` und `<module_name>_pos_tb.vhd`. Dabei bezeichnet `<module_name>` den Namen des Moduls. Die folgende Auflistung beschreibt die einzelnen Dateien. Für weitere Informationen zur Funktion dieser Dateien in einem VHDL-Design sei auf [Ste07] und [IEE02] verwiesen.

<module_name>_pak.vhd Das Package.

<module_name>_ent.vhd Die Entity.

<module_name>_arc.vhd Die Architecture.

<module_name>_beh_tb.vhd Die Testbench für die Verhaltenssimulation.

<module_name>_pre_tb.vhd Die Testbench für die Pre-Layout–Simulation.

<module_name>_pos_tb.vhd Die Testbench für die Post-Layout–Simulation.

Zusätzlich können noch andere, automatisch generierte Dateien abgelegt werden, z.B. Speicher.

Das hier verwendete VGA-Design (siehe später) besteht aus den folgenden vier Modulen; Abbildung 1.4 zeigt die Pinbelegung.

vga_driver Hier werden die Synchronisationssignale erzeugt.

vga_control Dieses Modul generiert die Farbdaten für die einzelnen Pixel. Die Ausgänge `d_toggle` und `d_toggle_counter` sind nur beim zweiten Beispiel vorhanden.

board_driver Die übrigen für das FPGA-Board notwendigen Signale werden hier implementiert.

vga Dieses Modul faßt die obigen drei Module zum VGA-Design zusammen.

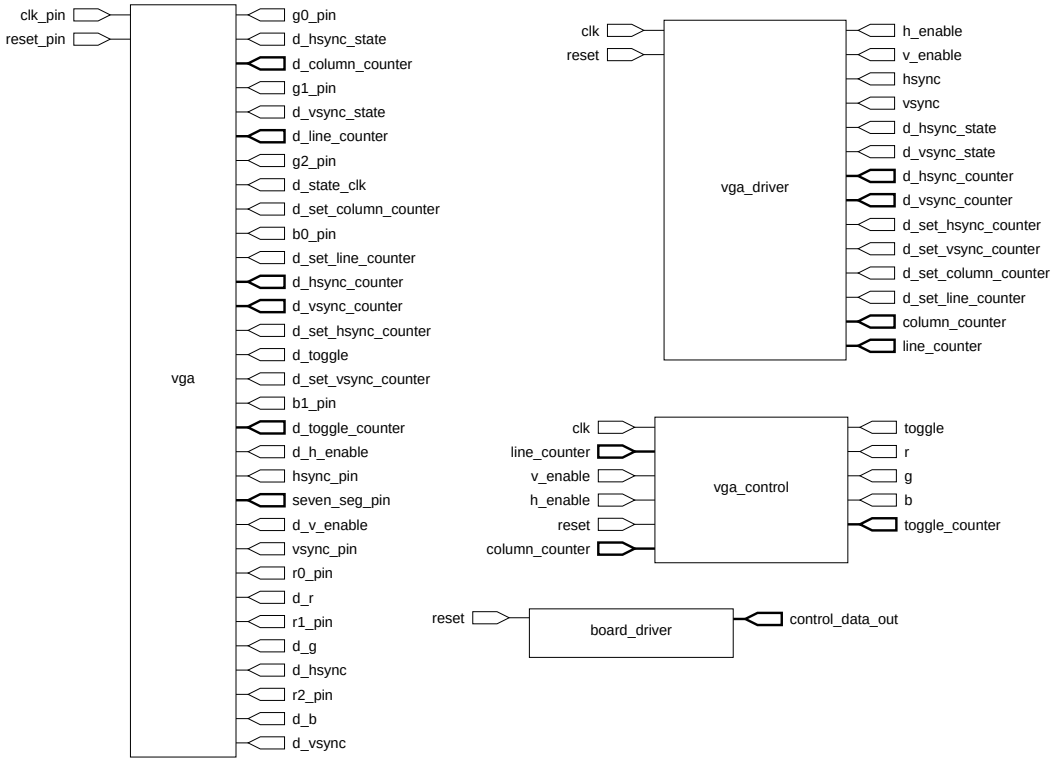


Abbildung 1.4: Die Pinbelegung der Module

sim: Im Verzeichnis `sim` werden die Simulationen ausgeführt. Zu diesem Zweck existiert für jede der drei Simulationen ein eigenes Unterverzeichnis:

beh: Verzeichnis für die Verhaltenssimulation (Behavioral Simulation) (siehe Kapitel 5.1).

pre: Verzeichnis für die Pre-Layout–Simulation (siehe Kapitel 5.3).

post: Verzeichnis für die Post-Layout–Simulation (siehe Kapitel 5.5).

syn: Im Verzeichnis `syn` letztendlich wird die Synthese des Designs durchgeführt, worauf in Kapitel 5.2 eingegangen wird.

ppr: Für das Place & Route steht das Verzeichnis `ppr` zur Verfügung; mehr dazu findet sich in Kapitel 5.4. Es enthält die beiden Unterverzeichnisse `sim` und `download` für Place & Route mit (`download`) und ohne PLL (`sim`) (Phase Locked Loop, siehe Kapitel 5.6).

1.3 Das VGA-Design

1.3.1 Die Struktur

Ziel dieser Laborübung ist das Erstellen, Testen und Debuggen eines VHDL-Designs sowie Übung im Umgang mit den dazu notwendigen Hardware- und Software-Tools. Dies soll am Beispiel der VGA-Schnittstelle (Video Graphics Array) umgesetzt werden; das FPGA-Board fungiert quasi als Grafikkarte. Die Verbindung zum Monitor erfolgt über einen 15-poligen Sub-D-Stecker (alternativ wäre eine BNC-Verbindung mit 5 Leitungen möglich). Die Pinbelegung kann in Kapitel 2.6 nachgeschlagen werden. Man sieht, daß sich die VGA-Schnittstelle im Wesentlichen aus den drei Farbsignalen *RED*, *GREEN* und *BLUE* sowie den Synchronisationssignalen *HSYNC* und *VSYNC* zusammensetzt. Die Farbsignale sind eigentlich analoger Natur, werden aber im Rahmen dieser Übung digital angesteuert.

Mit drei Farbkanälen können maximal acht Farben erzeugt werden. Wie Tabelle 2.5 in Kapitel 2.6 jedoch zeigt, verfügt das FPGA-Board über jeweils drei Signale für die Farben Rot und Grün sowie zwei Signale für die Farbe Blau. So können theoretisch 256 Farben dargestellt werden. Um den Blick für das Wesentliche nicht zu verlieren, werden in dieser Übung nur die drei Basis-Farbkanäle verwendet, das heißt, daß mehrfache Ausgänge für ein Farbsignal im Design zusammengeschlossen werden.

Bei der Ansteuerung eines VGA-Schirms können zwei Teilaufgaben identifiziert werden: das Timing für den Monitor sowie die Erzeugung der Farbwerte für die einzelnen Pixel. Auf das Timing wird in Kapitel 1.3.2 eingegangen. Diese Funktion wird im Modul *VGA_Driver* realisiert. Die Farbinformation wird im Modul *VGA_Control* implementiert.

Abbildung 1.5 zeigt das Blockschaltbild des Designs. Wie bereits erwähnt besteht es aus den Modulen *VGA_Driver*, *VGA_Control*, *Board_Driver* und einer PLL (siehe Kapitel 5.6). Der *Board_Driver* steuert zusätzliche Komponenten des FPGA-Boards an; so gibt er etwa die jeweilige Gruppennummer auf der Siebensegmentanzeige aus. Abhängig von der Aufgabenstellung kann der Speicher `vrom` vorhanden sein, muß aber nicht.

Da es für jene Studenten, welche das Distance-Learning-Angebot annehmen, schwer werden könnte, einen Reset-Taster zu drücken, wurde hier ein anderer Weg beschritten, um das Design in einen definierten Ausgangszustand zu versetzen. Im Modul `vga` ist die `purge_unit` (*Power Up Reset GEnerator*) implementiert. Diese

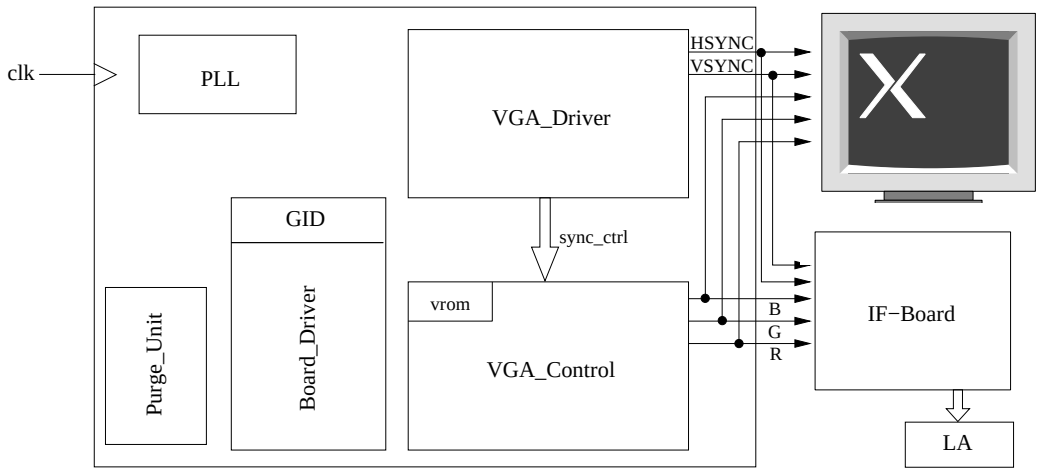


Abbildung 1.5: Das Design

Einheit ist prinzipiell ein Zähler, welcher für eine gewisse Zeit nach dem Download die Reset-Leitung aktiviert und so für einen stabilen, definierten Zustand der Schaltung sorgt.

Wie sich in Kapitel 4 und 5 zeigen wird, ist für den Design Flow die hierarchische Abhängigkeit der einzelnen Module untereinander von äußerster Wichtigkeit. Diese Abhängigkeit wird in Abbildung 1.6 dargestellt. Dazu sind die folgenden Bemerkungen notwendig:

- Die Modulhierarchie wird durch die vertikale Anordnung dargestellt; die horizontale Anordnung spiegelt weder Hierarchie noch Abhängigkeit wieder.
- Die vier Module *VGA*, *Board_Driver*, *VGA_Driver* und *VGA_Control* bestehen jeweils aus einer *Entity* und einer *Architecture*. Die Entity steht in der Hierarchie immer über der Architecture!
- Das Modul *VROM* wird nur für das erste Beispiel benötigt.
- Man beachte die Aufteilung in Download- und Simulationsbereiche. Der Grund für diese Aufteilung wird in Kapitel 5 erläutert.

1.3.2 Das VGA-Protokoll

Das Prinzip

Die Standard-VGA-Auflösung beträgt 640x480 Pixel, deren Farbe vom Zustand der Signale *RED, GREEN* und *BLUE* abhängt. Die Frequenz, mit welcher die einzelnen Pixel neu gezeichnet werden, ist auf 25,175 MHz festgelegt.

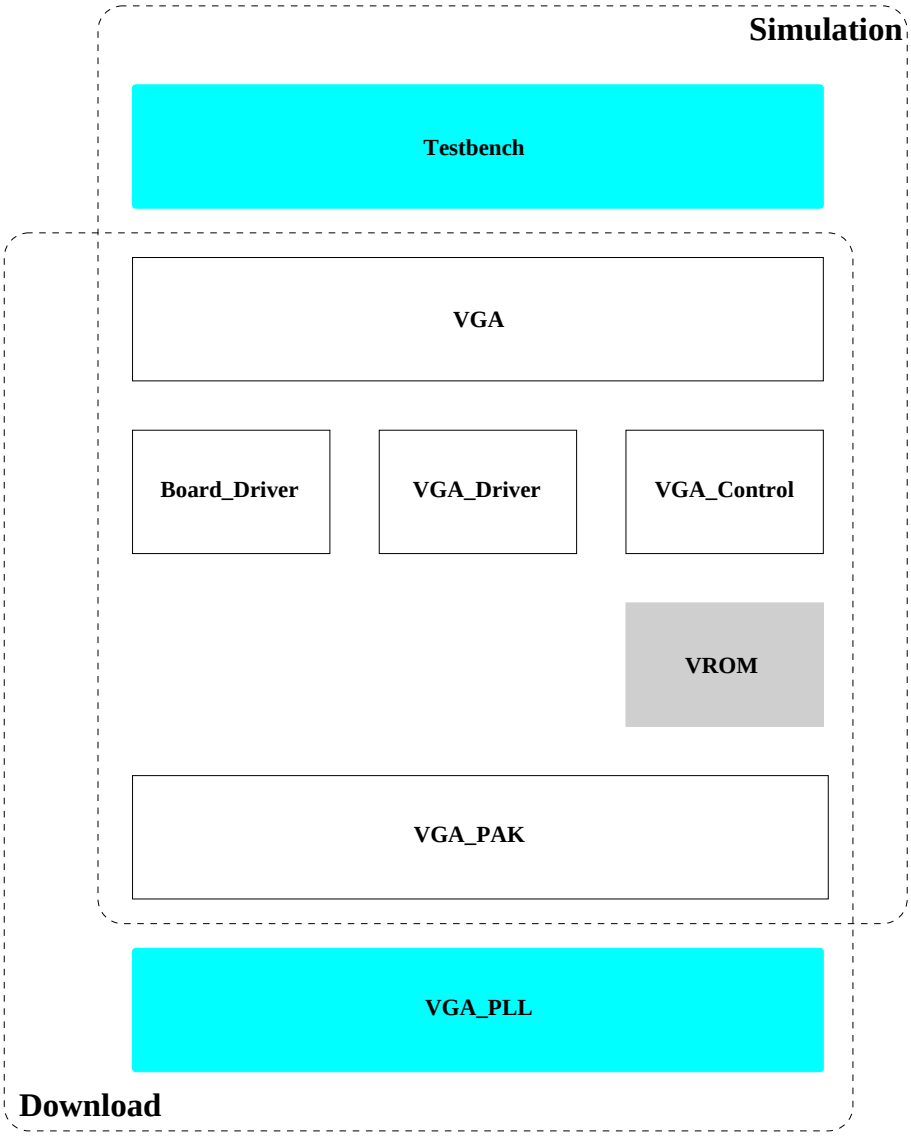


Abbildung 1.6: Die Hierarchie des Designs

Der Refresh des Bildschirms erfolgt nach einem festgelegten Schema, und zwar zeilenweise beginnend in der linken oberen Bildschirmecke. Das Weiterschalten zur nächsten Zeile erfolgt durch einen Puls auf der Leitung *HSYNC*. Ist die letzte (unterste) Zeile des Bildschirms erreicht, so bewirkt ein Puls auf der Leitung *VSYNC* den Start des nächsten Durchgangs in der linken oberen Bildschirmecke (siehe Abbildung 1.7). Den Zeitraum, in welchem der Elektronenstrahl vom Zeilenende zum Zeilenanfang bzw. vom "Bildschirmende" zu dessen Ursprung bewegt wird, nennt man *Austastlücke*. Man beachte, daß *HSYNC*-Signale auch in der Austastlücke gesendet werden.

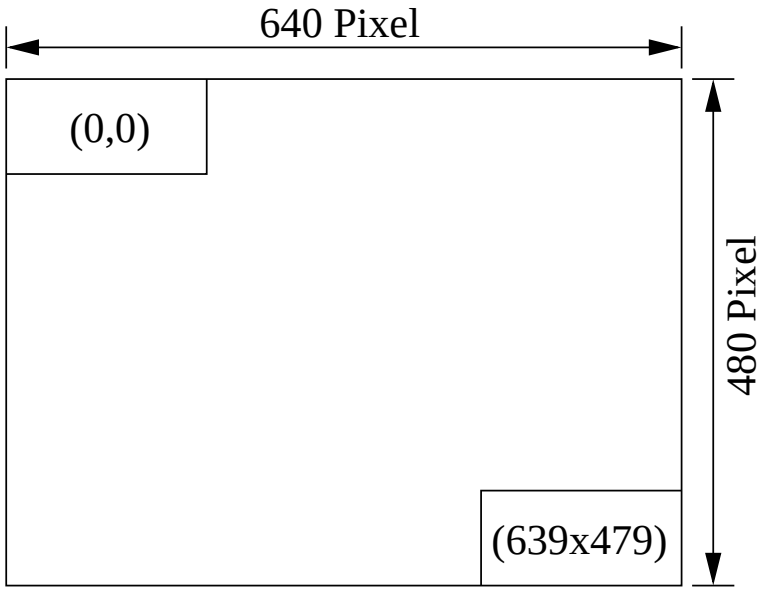


Abbildung 1.7: Der VGA-Bildschirm

Das Timing

Damit der Monitor vernünftig arbeitet, muß er die Daten zu fix vorgegebenen Zeitpunkten erhalten. Die Pulse auf den Leitungen *HSYNC* und *VSYNC* müssen zu genau spezifizierten Zeitpunkten erfolgen, um den Monitor zu synchronisieren, während er die Farbdaten erhält. Die Abbildungen 1.8 und 1.9 zeigen das Timing des horizontalen respektive vertikalen Synchronisationssignals. Die Tabellen 1.1 und 1.2 fassen die entsprechenden Parameter zusammen.

$$\begin{aligned} T_{pixel} &= \frac{1}{f_{CLK}} = 39.7\ ns \\ T_{row} &= A = B + C + D + E = 31.77\mu s \\ T_{screen} &= O = P + Q + R + S = 16.6\ ms \end{aligned}$$

Parameter	A	B	C	D	E
Zeit	$31.77\mu s$	$3.77\mu s$	$1.89\mu s$	$25.17\mu s$	$0.94\mu s$

Tabelle 1.1: Parameter der horizontalen Synchronisation

Die Bedeutung der Symbole ist in Tabelle 1.3 aufgelistet.

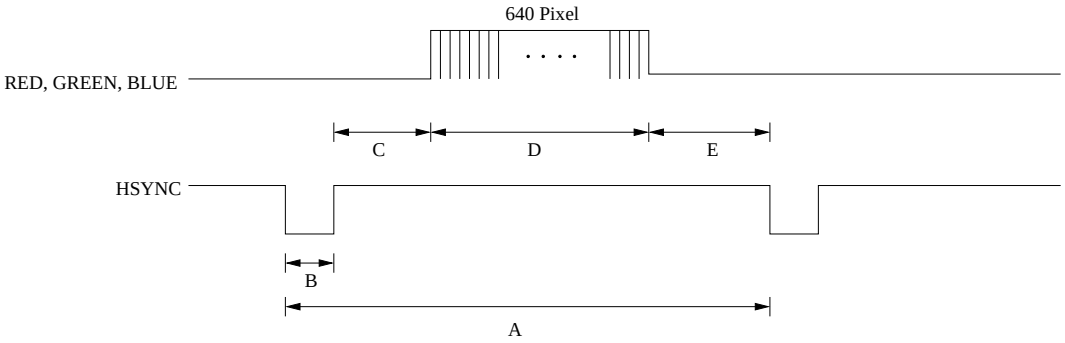


Abbildung 1.8: Horizontale Synchronisation

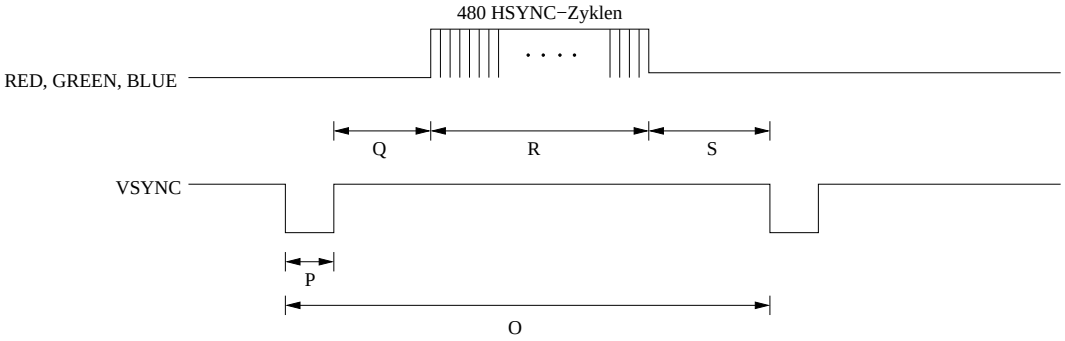


Abbildung 1.9: Vertikale Synchronisation

Parameter	O	P	Q	R	S
Zeit	$16.6ms$	$64\mu s$	$1.02ms$	$15.25ms$	$0.35ms$

Tabelle 1.2: Parameter der vertikalen Synchronisation

1.3.3 Finite State Machines

Die folgenden Absätze beschreiben die oben genannten Module in Form endlicher, deterministischer Automaten (*FSM, Finite State Machine*, [HU02]). Abbildung 1.10 erläutert vorab die Beschriftung der FSMs. Oberhalb eines Transitionspfeiles werden links die Eingangssignale und - getrennt durch einen Schrägstrich - rechts die Ausgangssignale aufgeführt. In der FSM selbst werden dann an Stelle der Signalnamen die aktuellen Werte eingetragen, sofern diese relevant für die Transition sind. Anderenfalls wird an Stelle des Wertes das Zeichen `x` eingetragen.

Wie dem vorigen Abschnitt zu entnehmen ist, ist für die Ansteuerung eines VGA-Schrims sowohl die exakte horizontale als auch vertikale Synchronisation zu gewährleisten. Die zulässige Toleranz kann jedoch von der Fähigkeit des Monitors sich aufzusynchronisieren abhängen.

Symbol	**Bedeutung**
T_{pixel}	Zeit für das Update eines Pixel
f_{CLK}	25.175 MHz
T_{row}	Zeit für das Update einer Zeile
T_{screen}	Zeit für das Update des Bildschirms
C, E	Guard Bands
Q, S	Guard Bands
B	Hsync-Puls
P	Vsync-Puls

Tabelle 1.3: Symbole zu obigen Formeln

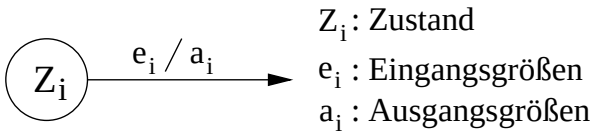


Abbildung 1.10: Symbolik der Zustandsgraphen

Abbildung 1.11 zeigt die FSM zur horizontalen Synchronisation, jene für die vertikale Synchronisation ist in Abbildung 1.13 dargestellt. Die Namen der Zustände entsprechen dabei den Bezeichnungen der jeweiligen Intervalle in Kapitel 1.3.2.

Aus Gründen der Übersichtlichkeit nicht eingezeichnet sind die Reset-Übergänge. Jeder Zustand der Hsync-FSM verfügt über einen Übergang der Form $0x/1010$ in den Reset-Zustand. Der korrespondierende Übergang der Vsync-FSM hat die Form $0x/0010$. Die Übergangsbedingungen für die übrigen Transitionen der beiden FSMs sind in Tabelle 1.12 und 1.14 angegeben.

Zu den Namen der Zustände: sie wurden bewußt so gewählt, daß sie den Intervallen aus den Abbildungen 1.11 und 1.13 sowie den zugehörigen Tabellen 1.12 und 1.14 entsprechen. Ein Beispiel: befindet sich die Hsync-FSM im Zustand *D_STATE*, so bedeutet dies, daß dieser Zustand für die Dauer des oben angeführten Intervalls *D* beibehalten wird. Die *Pre_*-Zustände werden benötigt, um einzelne Signale ausschließlich für einen Taktzyklus zu (de)aktivieren (siehe die Tabellen 1.12 und 1.14).

Wie bereits erwähnt, variiert die Fähigkeit von Bildschirmen, sich auf das VGA-Signal aufzusynchronisieren. Die Dauer, für welche die State Machines in den einzelnen Zuständen verweilen, kann Tabelle 1.4 entnommen werden.

Parameter	PRE_B	B	C	PRE_D	D	E
Zeit	$37ns$	$3.3\mu s$	$1.6\mu s$	$37ns$	$23.4\mu s$	$883ns$
Parameter	PRE_P	P	Q	PRE_R	R	S
Zeit	$37ns$	$29.4\mu s$	$942.6\mu s$	$37ns$	$14.13ms$	$206.2\mu s$

Tabelle 1.4: Tatsächliches Timing des VGA-Designs

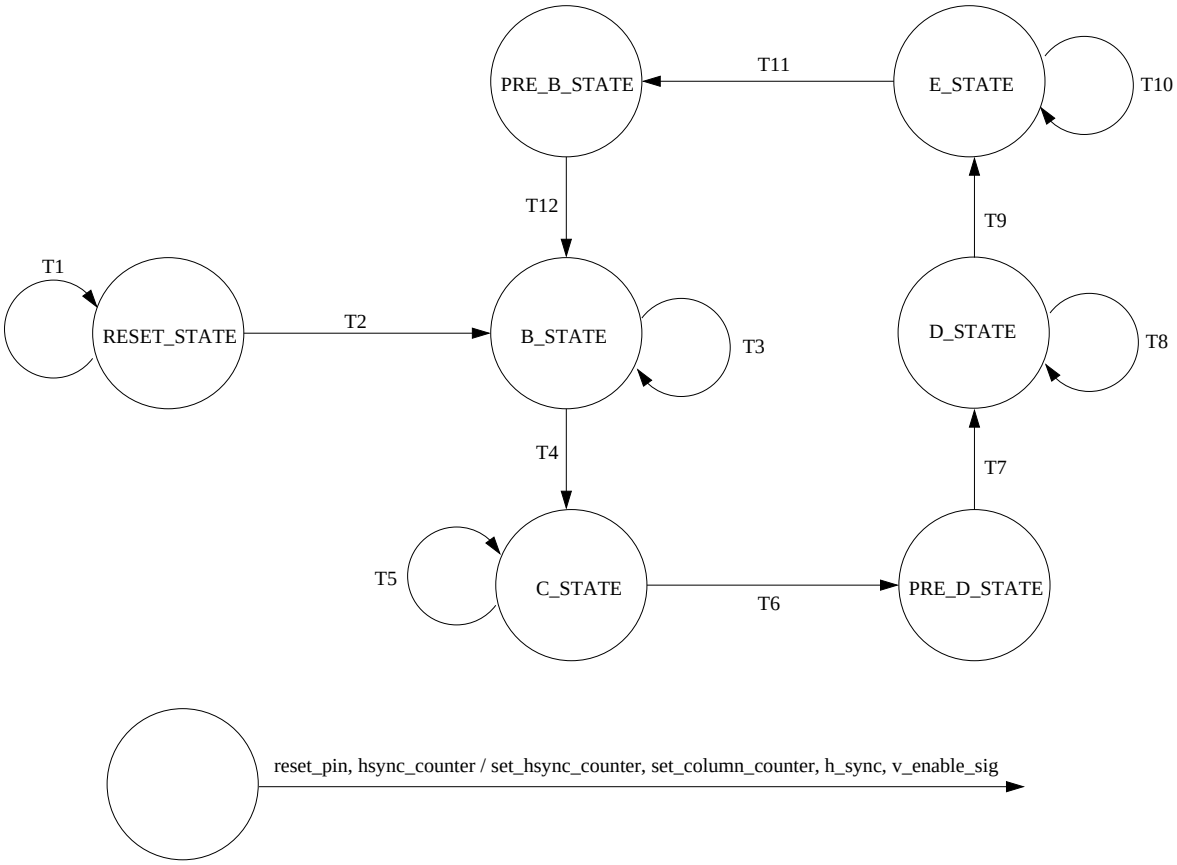


Abbildung 1.11: Die Hsync-FSM

Transition	reset	hsync_counter
T1	0	x
T2	1	x
T3	1	$\neq$ time_b
T4	1	time_b
T5	1	$\neq$ time_bc
T6	1	time_bc
T7	1	x
T8	1	$\neq$ time_bcd
T9	1	time_bcd
T10	1	$\neq$ time_a
T11	1	time_a
T12	1	x

(a) Übergangsbedingungen

State	set_hsync_counter	set_column_counter	hor_sync	v_enable
RESET_STATE	1	0	1	0
B_STATE	0	0	0	0
C_STATE	0	0	1	0
PRE_D_STATE	0	1	1	1
D_STATE	0	0	1	1
E_STATE	0	0	1	0
PRE_B_STATE	1	0	0	0

(b) Output

Abbildung 1.12: Die HSYNC-FSM

Bei der Synthese bleiben die Namen der Zustände natürlich nicht in textueller Form erhalten, sie werden codiert. Dieser Umstand führt dazu, daß bei der Pre- und Post-Layout-Simulation nicht die Namen der States, sondern deren Codes angezeigt werden.

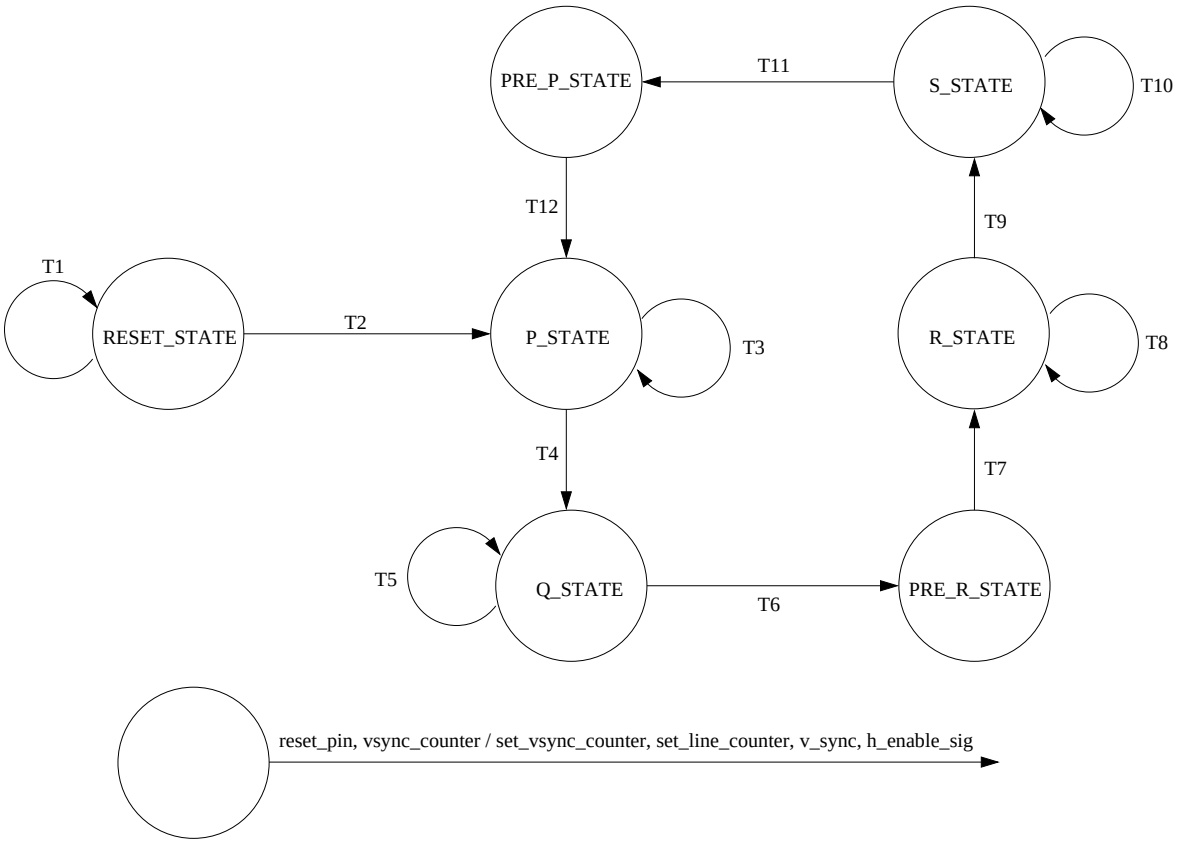


Abbildung 1.13: Die Vsync-FSM

Transition	reset	vsync_counter
T1	0	x
T2	1	x
T3	1	$\neq$ time_p
T4	1	time_p
T5	1	$\neq$ time_pq
T6	1	time_pq
T7	1	x
T8	1	$\neq$ time_pqr
T9	1	time_pqr
T10	1	$\neq$ time_o
T11	1	time_o
T12	1	x

(a) Übergangsbedingungen

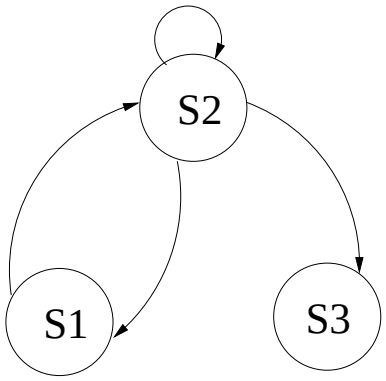
State	set_vsync_counter	set_line_counter	vert_sync	h_enable
RESET_STATE	1	0	1	0
P_STATE	0	0	0	0
Q_STATE	0	0	1	0
PRE_R_STATE	0	1	1	1
R_STATE	0	0	1	1
S_STATE	0	0	1	0
PRE_P_STATE	1	0	0	0

(b) Output

Abbildung 1.14: Die VSYNC-FSM

Für die Codierung stehen mehrere Möglichkeiten zur Verfügung. Die gängigsten Methoden werden am Beispiel der FSM aus Abbildung 1.15(a) erklärt, Tabelle 1.15(b) faßt die Codierung zusammen.

Beim *One-Hot-Encoding* wird für jeden Zustand ein eigenes Register generiert. Diese Codierung erzeugt sehr schnelle FSMs, ist jedoch mit vermehrtem Platzbedarf



(a) Die FSM

State	One-Hot	Binary	Gray
S1	001	00	00
S2	010	01	01
S3	100	10	11

(b) Codierungen

Abbildung 1.15: Die Beispiel-FSM und deren übliche Codierungen

zu bezahlen. Man sieht schon bei diesem kleinen Beispiel, daß *Gray-* oder *Binary-Encodings* deutlich platzsparender sind, in den meisten Fällen sind sie jedoch auch langsamer.

In dieser Übung, mit dem verwendeten Tool und seinen Einstellungen wird *One-Hot-Encoding* verwendet. Diese Tatsache wird spätestens dann interessant, wenn man am Logikanalysator die korrekten Zustandsnamen anzeigen will (siehe dazu 3.5), wie dies für Beispiel 1 notwendig ist.

Betrachtet man die Typdeklaration in Programmfragment 1.3.1 sieht man, daß die Zuordnung eines Codewortes zu einem State in Zusammenhang mit der Reihenfolge der States bei deren Deklaration steht. So wird etwa dem ersten State, S1, der Code "001" zugeordnet.

Programm 1.3.1 Typdeklaration

```
type demo_state_type is (S1, S2, S3);
```

1.3.4 Das Pinout

Die Pinbelegung des Designs ist in Tabelle 1.5 wiedergegeben. Sie stellt den Zusammenhang zwischen Signal, FPGA-Pin und letztendlich den Leitungen des Logikanalysators her.

Es mag auffallen, daß nicht alle Bits aller Counter zum Logikanalysator geführt wurden. Dies liegt daran, daß die im Rahmen dieser Übung geforderte Genauigkeit mit den verwendeten Bits erreicht werden kann. Andererseits kann am LSB einfach erkannt werden, ob der entsprechende Counter arbeitet oder nicht.

	POD 1		POD 2		POD 3		POD 4		POD 5	
Pin	**FPGA**	**Signal**	**FPGA**	**Signal**	**FPGA**	**Signal**	**FPGA**	**Signal**	**FPGA**	**Signal**
0			M19	d_vsync_state(3)	M9	d_line_counter(3)	J21	d_h_enable	H4	d_hsync_counter(0)
1			M18	d_vsync_state(4)	M8	d_line_counter(4)	H18	d_v_enable		
2			M7	d_vsync_state(5)	M6	d_line_counter(5)	H3	d_toggle		
3			M4	d_vsync_state(6)	M5	d_line_counter(6)	G26	d_toggle_counter(0)	G25	d_hsync_counter(7)
4					L24	d_line_counter(7)	G24	d_toggle_counter(15)		
5	Y5	d_hsync_state(0)			L25	d_line_counter(8)	G23	d_toggle_counter(16)	G22	d_hsync_counter(8)
6	F19	d_hsync_state(1)			L23	d_col_counter(0)	G21	d_toggle_counter(17)	G18	d_hsync_counter(9)
7	F17	d_hsync_state(2)			L22	d_col_counter(1)	G20	d_toggle_counter(18)	G9	d_vsync_counter(0)
8			L7	d_hsync	L21	d_col_counter(2)	G5	d_toggle_counter(19)	G6	d_vsync_counter(7)
9	Y2	d_hsync_state(3)	L5	d_vsync	L20	d_col_counter(3)	G3	d_toggle_counter(20)	G4	d_vsync_counter(8)
10	F10	d_hsync_state(4)	L3	d_r	L6	d_col_counter(4)	G1	d_toggle_counter(21)	G2	d_vsync_counter(9)
11	F9	d_hsync_state(5)	K24	d_g	L4	d_col_counter(5)	F25	d_toggle_counter(22)	F26	d_set_hsync_cnt
12	F6	d_hsync_state(6)	K20	d_b	L2	d_col_counter(6)	F23	d_toggle_counter(23)	F24	d_set_vsync_cnt
13	F5	d_vsync_state(0)	K6	d_line_counter(0)	K23	d_col_counter(7)	T19	d_toggle_counter(24)	F21	d_set_line_cnt
14	F4	d_vsync_state(1)	K4	d_line_counter(1)	K19	d_col_counter(8)	K3	d_state_clk	Y23	d_set_col_cnt
15	F3	d_vsync_state(2)	J22	d_line_counter(2)	K5	d_col_counter(9)				

Tabelle 1.5: Die Pinbelegung

1.4 Hinweise zur Übungsdurchführung

Bitte beachten Sie die folgenden Punkte, welche für alle vier Beispiele gelten!

- Für jedes Beispiel ist ein Protokoll pro Gruppe anzufertigen. Dazu ist das zur Verfügung gestellte Latex-Template zu verwenden.
- Die Punkte, die im Protokoll enthalten sein müssen, finden Sie im Skriptum bei den jeweiligen Aufgabenstellungen.
- Wenn bei einer Aufgabe Messungen durchzuführen sind, so sind auch die Meßwerte in textueller Form in das Protokoll aufzunehmen.
- Im Verzeichnis `Protokolle` der Beispiele 1, 2, und 4 befindet sich jeweils ein Textfile mit der Bezeichnung

  `protocol_n.txt`, wobei n der Nummer des Beispiels entspricht. In diese Files müssen Sie jene Werte eintragen, welche Sie bei dem jeweiligen Beispiel gemessen oder errechnet haben oder aber (bei Beispiel 4) die Zeilennummer der gefundenen Fehler. Die Files sind vorformatiert, Sie müssen lediglich die Werte eintragen: ein Wert pro Zeile, das Label am Beginn der Zeile bezeichnet den entsprechenden Wert. Dieser wird nach dem Label zwischen den beiden Underscores eingetragen. Verwenden Sie bitte als Dezimaltrennzeichen den Beistrich und achten Sie auf die angegebenen Einheiten! Bedenken Sie bitte, daß diese Files automatisch bewertet werden. Falsches Eintragen oder eine falsche Formatierung könnte zu unerwünschten Ergebnissen führen.
- Erzeugung von Screenshots: mit der Tastenkombination `Alt-Druck` kopieren Sie ein Abbild des aktuellen Fensters in die Zwischenablage, mit `Strg-Druck` den ganzen Bildschirm; den Inhalt der Zwischenablage können Sie mit einem beliebigen Graphikprogramm in ein File transformieren.
- Ist ein Beispiel gelöst, so ist vom Übungsaufbau mit der Webcam ein Bild zu machen, welches (nach Möglichkeit und Aufgabenstellung) die geforderte Lösung zeigt. Dieses Bild ist in das Protokoll einzugliedern.
- Betreiben Sie die für den Design-Flow benötigten Programme im Full-Screen-Modus. Sie erleichtern sich damit die Bearbeitung der Beispiele.
- Grundsätzlich gilt ein Beispiel erst dann als gelöst, wenn beim Durchlaufen des Design-Flow keine Warnungen mehr auftreten. Zu dieser Regel gibt es jedoch eine Ausnahme: wenn etwa auf dem Bildschirm die Farbe Grün nicht angezeigt wird, so erscheinen dem Synthese-Tool und / oder dem Place & Route-Tool die Signale `g0_out`, `g1_out`, `g2_out` und `d_g` als *Stuck-At Zero*. Es ist anschaulich klar, daß die dadurch erzeugten Warnungen im Rahmen des hier verwendeten Designs nicht eliminiert werden können. Selbiges gilt beispielsweise auch, wenn das / die höchstwertige(n) Bit(s) des Schwellwerts eines Counters auf Null gesetzt sind, dieser also nicht in seiner vollen Breite verwendet wird, oder bei

konstanter Ansteuerung der Siebensegment-Anzeige. Das Synthese-Tool *Synplify Pro* würde in diesem Fall eine Warnung der Form *Removing sequential element ...* äußern.

Diese Warnungen sind kein Grund zur Sorge, man sollte sich jedoch immer den Ursprung derartiger Meldungen bewußt machen!

- Insbesondere für Beispiel vier, aber auch für die Beispiele zwei und drei sei auf die Debug-Hilfestellung in Anhang A hingewiesen.

- Vermerken Sie etwaige Anomalien oder potentiel nicht funktionierende Hardware nach Rücksprache mit einem Tutor im Protokoll. Notieren Sie dabei bitte auch die Nummer des Arbeitsplatzes (auf der Interface-Platine zu finden); Sie erleichtern uns damit die Fehlersuche und bieten uns die Chance, schnellstmöglich wieder ein funtionierendes Environment zur Verfügung zu stellen.

- Beziehen Sie auch das Vorlesungs-Skriptum in Ihre Vorbereitung auf die Übungstests ein.

- Wie schon im Merkblatt erwähnt, ist die Übungsleitung unter der Email-Adresse *dide@ecs.tuwien.ac.at* zu erreichen. Anders adressierte Mails werden bei Fehlen einer guten Begründung ignoriert.

- Noch ein Wort zum Abschlußtest (siehe Merkblatt): bitte beachten Sie, daß Sie sich für diesen Test einzeln und nicht als Gruppe anmelden müssen! Sie werden beim Absolvieren dieses Tests ein der LU-Umbegung entsprechendes Environment vorfinden. Ihre Aufgabe wird es sein, ein fehlerhaftes Design zu debuggen und zum Laufen zu bringen. Bitte beachten Sie, daß Sie beim Abschlußtest weder Projekte anlegen noch die Pinzuweisung durchführen müssen (siehe später). Diese Dinge werden für Sie vorbereitet.

1.5 Aufgabe 1 - Logikanalysator

1.5.1 Zielsetzung

Anhand dieser ersten Aufgabe sollen Sie sich mit dem verwendeten FPGA-Board sowie insbesondere mit dem Logikanalysator vertraut machen. Der Logikanalysator wird detailliert in Kapitel 3 erklärt; eine Beschreibung des Boards findet sich in Kapitel 2. Als zusätzliche Hilfe können Sie die Software des Logikanalysators von der LVA-Homepage downloaden, um sich vorab mit der Bedienung dieses Geräts vertraut zu machen.

Das Lösen dieser Aufgabe sollte Sie in die Lage versetzen, die folgenden Punkte zu beherrschen:

- Evaluation-Board: Inbetriebnahme, Download und Kenntnis der Schnittstellen
- Logikanalysator: State- und Timinganalyse, Setup, Triggerung, Signaldarstellung
- VGA: Verständnis von Spezifikation und Protokoll

1.5.2 Vorbereitung

Machen Sie sich anhand der Kurzanleitung (und eventuell auch der Online-Hilfe der Demo-Software, zu finden unter [did]) mit dem Konzept und der Bedienung des Logikanalysators vertraut. Verschaffen Sie sich weiters einen Überblick über das FPGA-Board (wie funktionieren Inbetriebnahme & Download, welche Schnittstellen gibt es). Studieren Sie die VGA-Spezifikation und versuchen Sie, das horizontale und vertikale Timing zu verstehen. Überlegen Sie, wie Sie an Ihre Meßaufgabe herangehen werden: Welche Signale werden Sie aufzeichnen? Wie werden Sie den Trigger und die Zeitbasis wählen? (Hinweis: Könnten die beiden Busse `d_column_counter` und `d_line_counter` die Messungen vereinfachen?)

1.5.3 Aufgabenstellung

Für diese Aufgabe steht Ihnen ein Design zur Verfügung, welches zwei rechteckige Objekte auf dem Bildschirm auf monochromem Hintergrund ausgibt. Eines davon wird algorithmisch erzeugt, das zweite wird aus einem ROM ausgelesen. Vergessen Sie nicht, die Meßwerte in das File `protocol.txt` im Verzeichnis `Protokolle` einzutragen.

1. Messen Sie mit dem Logikanalysator die horizontale Synchronisationsfrequenz f_{hsync} und die vertikale Synchronisationsfrequenz f_{vsync}. Benutzen Sie für f_{hsync} den *Timing-Mode* und für f_{vsync} den *State-Mode* des Logikanalysators.

2. Stellen Sie auf dem Logikanalysator dar, welche Farbe der Pixel mit den Koordinaten (x,y) besitzt, d.h. bestimmen Sie die Pegel der Farbwerte Rot, Grün und Blau für diese Koordinaten. Die Koordinaten für diese Aufgabe finden Sie im Verzeichnis für das erste Beispiel im Unterverzeichnis `Angabe` im File `dide_nn_1.txt`. Die Buchstaben *nn* bezeichnen dabei Ihre Gruppennummer. Beachten Sie die Lage des Koordinatensystems gemäß Kapitel 1.3.2. Benutzen Sie für diese Aufgabe den *State-Mode* des Logikanalysators.

3. Bestimmen Sie die Farbe des Hintergrundes (mit dem Logikanalysator!). Unter Hintergrund wird dabei jener Bereich des Bildschirms verstanden, welcher den größten Teil des Bildschirms einnimmt, sich farblich eindeutig von den sonst dargestellten Objekten unterscheidet und nur mit einer einzigen Farbe versehen ist. Es genügt dabei einen einzigen (beliebigen) Pixel des Hintergrundes zu messen. Bestimmen Sie außerdem die x-Koordinate der linken Kante des linken Objekts.

   Bei dieser Aufgabe liegt es an Ihnen, den Logikanalysator in einem geeigneten Modus zu betreiben. Begründen Sie, warum Sie einen bestimmten Modus gewählt haben.

4. Stellen Sie auf dem Logikanalysator die Zustandsabfolge der Hsync-FSM dar. Wählen Sie auch hier einen geeigneten Meß-Modus und begründen Sie Ihre Wahl.

1.5.4 Protokoll

Das Protokoll muß die im Folgenden aufgelisteten Punkte beinhalten:

- Triggerbedingung des Logikanalysators mit Begründung sowie Screen-Shot des Trigger-Menüs, jeweils für alle Aufgaben.
- Screenshot der relevanten, am Logikanalysator aufgezeichneten Waveforms. Jeweils für alle Aufgaben.
- Interpretation der aufgezeichneten Daten für alle Aufgaben.

1.6 Aufgabe 2 - Design-Flow

1.6.1 Zielsetzung

Diese Aufgabe soll Ihnen, ausgehend von einem in VHDL beschriebenen Design, den Design-Flow eines FPGA näherbringen.

Das Lösen dieser Aufgabe sollte Sie in die Lage versetzen, die folgenden Punkte zu beherrschen:

- Design-Entry mittels Texteditor (z.B. XEmacs)
- Kenntnis des formalen Aufbaus eines VHDL-Designs
- Simulationen mit *ModelSim SE*
- Synthese mit *Synplify Pro*
- Place&Route mit *Quartus II*

1.6.2 Vorbereitung

Machen Sie sich vertraut mit dem Design-Flow einer digitalen Schaltung im allgemeinen und dem eines FPGA im besonderen: Versuchen Sie, den theoretischen Hintergrund mit der praktischen Vorgangsweise zu korrelieren. Gehen Sie die Schritte in Gedanken durch und überlegen Sie, welche Inputs jeweils benötigt werden bzw. welche Outputs produziert werden. Machen Sie sich anhand der Anleitung in Kapitel 5 mit dem Umgang mit der verwendeten Software vertraut. Überlegen Sie, welche Maßnahmen Sie treffen müssen, um die gestellte Aufgabe (siehe nachfolgender Abschnitt) zu lösen.

1.6.3 Aufgabenstellung

Das Design für dieses Beispiel erzeugt am Bildschirm ein blinkendes rechteckiges Objekt. Ihre Aufgabe besteht nun darin, die Frequenz des Blinkens zu verändern. Unter Blinkfrequenz wird die Anzahl der Zustandsänderungen verstanden.

Dazu müssen Sie im Source-Code eine geeignete Änderung vornehmen und anschließend mit dem modifizierten Code den gesamten Design-Flow (inklusive aller Simulationen!) durchlaufen.

Die geforderte Periodendauer für diese Aufgabe finden Sie im Verzeichnis für das zweite Beispiel im Unterverzeichnis `Angabe` im File `dide_nn_2.txt`. Die Buchstaben *nn* bezeichnen dabei Ihre Gruppennummer.

Hinweis: Um Signale wie etwa `toggle` bei der Simulation sinnvoll darstellen zu können, muß das Verhalten des Designs möglicherweise über einen Zeitraum von mehreren hundert Millisekunden simuliert werden. Solche Simulationen können oft einige Stunden dauern. Man kann sich helfen, indem man (für diese Aufgabe) den Counter `toggle_counter` um z.B. den Faktor 100 skaliert (nachdem Sie - entsprechend Ihrer Angabe - die Periodendauer richtig eingestellt haben) und die Meßwerte nach der Simulation auf die tatsächliche Größenordnung zurückführt.

Führen Sie also die Verhaltenssimulation mit einem derart modifizierten Counter durch und berechnen Sie die Dauer des Blinkintervalls. Sobald Sie sicher sind, die korrekte Frequenz eingestellt zu haben, können Sie im Design Flow fortfahren, nachdem Sie obige Modifikation rückgängig gemacht haben. Für die Prelayout- und Postlayout-Simulation genügt es danach (für dieses Beispiel), wenn Sie zeigen, daß dieser Counter läuft.

Vergessen Sie nicht, den berechneten Counterwert in das File `protocol.txt` im Verzeichnis `Protokolle` einzutragen.

1.6.4 Protokoll

Das Protokoll muß die im Folgenden aufgelisteten Punkte beinhalten.

- Source-Code mit gekennzeichneten Änderungen.
- Für die Simulation sowie den Logikanalysator unbedingt notwendige Signale: `Horiz_sync`, `Vert_sync`, `r`, `g`, `b`, `toggle_counter`, `toggle`
- Schirmbilder der Simulationen
- Auslastung des FPGA laut *Quartus II*
- Die am Logikanalysator aufgezeichneten Waveforms
- Interpretation der aufgezeichneten Daten

1.7 Aufgabe 3 - VHDL

1.7.1 Zielsetzung

In dieser Aufgabe sollen Sie erste Erfahrungen im Design-Entry mittels VHDL machen und Ihre Kenntnisse über den Design-Flow vertiefen.

Das Lösen dieser Aufgabe sollte Sie in die Lage versetzen, die folgenden Punkte zu beherrschen:

- Formulierung digitaler Grundelemente in VHDL (Zähler, State-Machine, logische Verknüpfungen, etc.)
- Vertiefte Kenntnisse über den Design-Flow, insbesondere Simulation und Debugging

1.7.2 Vorbereitung

Machen Sie sich anhand der Vorlesungsunterlagen bzw. eventuell auch anhand der dort empfohlenen Literatur mit den typischen Konstrukten in VHDL vertraut: Wie formuliert man logische Verknüpfungen, State-Machines, Zähler etc. Versuchen Sie insbesondere auch ein Gefühl für die Parallelität der Abläufe in Hardware zu entwickeln. Auch wenn VHDL sich auf den ersten Blick nicht wesentlich von anderen Programmiersprachen unterscheidet, so muß man sich dennoch immer vor Augen halten, daß hier kein (sequentielles) Programm erzeugt wird, sondern (parallele) Hardware. Bei der Formulierung von VHDL-Code muß man sich angewöhnen, in Zyklen

zu denken. Ein weiterer wesentlicher Unterschied zu einer üblichen Programmiersprache ist der Umstand, daß sich in VHDL wesentlich mehr formulieren (und auch simulieren) läßt, als das Synthesetool später auch wirklich in Hardware umsetzen kann. Sehen Sie sich aus diesem Grund auch - vor der Übungseinheit - den Code des VGA-Designs durch und machen Sie sich mit den verwendeten Konstrukten vertraut.

1.7.3 Aufgabenstellung

Für diese Aufgabe steht Ihnen ein Design zur Verfügung, welches das horizontale und vertikale Timing implementiert, nicht aber die Farbsteuerung. Ihre Aufgabe ist es nun, die Ausgabe eines Streifenmusters zu implementieren.

Beginnen Sie damit, einen neuen Prozeß im Modul `vga_control` zu codieren. Benutzen Sie den Zeilen- oder Spaltenzäler (je nach Angabe, siehe unten), um vorerst schwarz-weiße Streifen der gewünschten Breite zu erzeugen. Färben Sie diese Streifen dann gemäß der Angabe ein. Die für das Protokoll notwendigen Punkte beziehen sich auf das fertige - sprich eingefärbte - Design.

Die geforderten Parameter für diese Aufgabe finden Sie im Verzeichnis für das dritte Beispiel im Unterverzeichnis `Angabe` im File `dide_nn_3.txt`. Die Buchstaben *nn* bezeichnen dabei Ihre Gruppennummer. Diese Parameter umfassen: die Anzahl der Streifen, die Breite jedes Streifens in Pixeln sowie die Reihenfolge der darzustellenden Farben.

1.7.4 Protokoll

Das Protokoll muß die im Folgenden aufgelisteten Punkte beinhalten.

- Source-Code mit gekennzeichneten Änderungen und Kommentaren.
- Für die Simulation sowie den Logikanalysator unbedingt notwendige Signale: `Horiz_sync`, `Vert_sync`, `r`, `g`, `b`, `column_counter`, `line_counter`
- Screenshot des am TFT angezeigten Streifenmusters (mit der Web-Cam)
- Schirmbild der Simulation.
- Auslastung des FPGA's laut *Quartus II*.
- Die am Logikanalysator aufgezeichneten Waveforms.
- Interpretation der aufgezeichneten Daten.

1.8 Aufgabe 4 - Simulation und Test

1.8.1 Zielsetzung

Diese Aufgabe soll Sie mit dem Testen und Debuggen von Hardware vertraut machen. Insbesondere sollen Sie erkennen, wie sich diese Aufgaben des Hardware-Design vom Debuggen und Testen von Software unterscheiden.

Das Lösen dieser Aufgabe sollte Sie in die Lage versetzen, die folgenden Punkte zu beherrschen:

- Hierarchisches Design unter VHDL
- Fehlersuche in einem gegebenen VHDL-Design
- Erstellen einer Testbench für einen Simulator
- Übung im Umgang mit Simulator und Logikanalysator.

1.8.2 Vorbereitung

Bei dieser Übung ist es sehr wichtig, daß Sie im Umgang mit den Werkzeugen bereits geübt sind und sich auf den inhaltlichen Teil der Aufgabenstellung konzentrieren können. Wiederholen Sie daher nochmals den bisher erarbeiteten Stoff und versuchen Sie, verbliebene Probleme im Verständnis und im Umgang mit den Tools zu beseitigen. Machen Sie sich nochmals mit dem Design und der VGA-Spezifikation vertraut und beachten Sie die Hinweise in Anhang A.

1.8.3 Aufgabenstellung

Bei diesem Beispiel haben Sie ein Design vor sich, das ein rechteckiges Objekt auf dem Bildschirm darstellt. Dieses Objekt soll mit einer bestimmten Frequenz die Farbe wechseln. (Diesen Parameter finden Sie wieder im Verzeichnis für das vierte Beispiel im Unterverzeichnis `Angabe` im File `dide_nn_4.txt`. Die Buchstaben *nn* bezeichnen wie immer Ihre Gruppennummer.) **Achtung:** Das Einstellen dieser Parameter stellt nicht die Beseitigung von Fehlern dar!

Nun ist dieses Design fehlerhaft. Ihre Aufgabe besteht darin, diese Fehler zu lokalisieren und zu korrigieren, sodaß Sie schließlich ein funktionierendes Design auf das FPGA herunterladen können und am Bildschirm ein störungsfreies Bild sehen. *Das heißt, daß Sie den ganzen Design-Flow mit allen Simulationen durchlaufen müssen.*

Als Hilfsmittel stehen Ihnen dazu der Simulator und der Logikanalysator zur Verfügung. Die für die Simulationen nötige Testbench wird ebenfalls zur Verfügung gestellt. Auch zumindest eine Messung mit dem Logikanalysator ist verpflichtend vorgesehen, welche jeden der gefundenen Fehler dokumentiert (natürlich mit Ausnahme von Syntaxfehlern).

Die Testbench ist so programmiert, daß die Ausgabe von genau einer Pixelzeile simuliert werden kann. Ihre Aufgabe ist nun auch die Erweiterung dieser Testbench, sodaß die Ausgabe einer ganzen Bildschirmseite simuliert werden kann. Die auf diesem Weg erhaltenen Waveforms müssen nur noch interpretiert werden, um die Fehler zu finden.

Vergessen Sie nicht, die gefundenen Fehler in das File `protocol.txt` im Verzeichnis `Protokolle` einzutragen.

1.8.4 Protokoll

Das Protokoll muß die im Folgenden aufgelisteten Punkte beinhalten.

- Source-Code mit gekennzeichneten Änderungen und Kommentaren
- Source-Code der verwendeten Testbench
- Screen-Shot der Simulation, welche den oder die lokalisierten Fehler zeigt
- Screen-Shot des Logikanalysators, welcher den/die ausgebesserten Fehler zeigt, also das gleiche Szenario aus dem vorigen Punkt, nur nach der Fehlerkorrektur
- Interpretation der aufgezeichneten Daten.
- Auslastung des FPGA's laut *Quartus II*.

LU-SKRIPTUM — Kapitel 2

Das MJL-Stratix-Development-Board

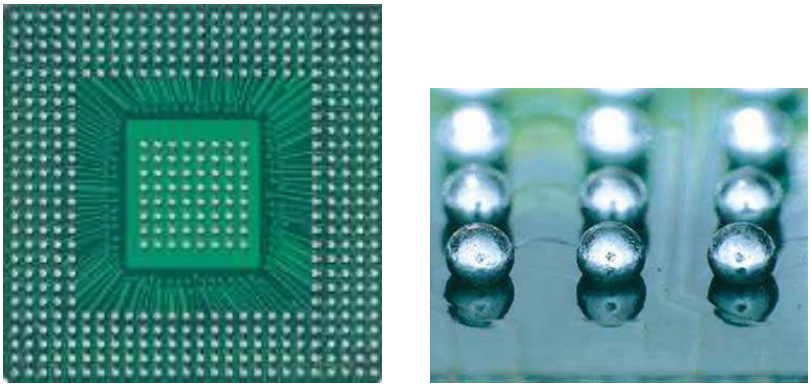
Inhalt

Das MJL-Stratix-Development-Board ist eine Experimentierplatine, welche über ein FPGA der Familie *Stratix EP1S25* verfügt. Abbildung 2.3 zeigt den schematischen Aufbau der Platine und kennzeichnet die wichtigsten Komponenten. Die folgenden Kapitel beschreiben die Funktionalität dieser Bauelemente.

2.1 Das FPGA *EP1S25F672C6*

Technologisch betrachtet basiert dieses FPGA auf rekonfigurierbaren SRAM-Zellen mit programmierbarem Interconnect in Kupfer-Technologie und wird in einem FBGA-Gehäuse (*Fine-Pitch Ball Grid Array*) mit 672 Pins gefertigt.

Es ist aus 25660 *Logic Elements (LEs)* (siehe Abbildung 2.2), mehreren RAM-Blöcken, eingebetteten Multiplizierern sowie 6 PLLs aufgebaut. Jedes LE besteht aus einer *Look-Up Table (LUT)* mit 4 Eingängen, einem programmierbaren Flip-Flop, sowie dedizierten Signalpfaden für Carry- und Kaskadierungsfunktionen.



(a) Das Gehäuse (b) Die Lötkontakte

Abbildung 2.1: Beispiel eines FBGA

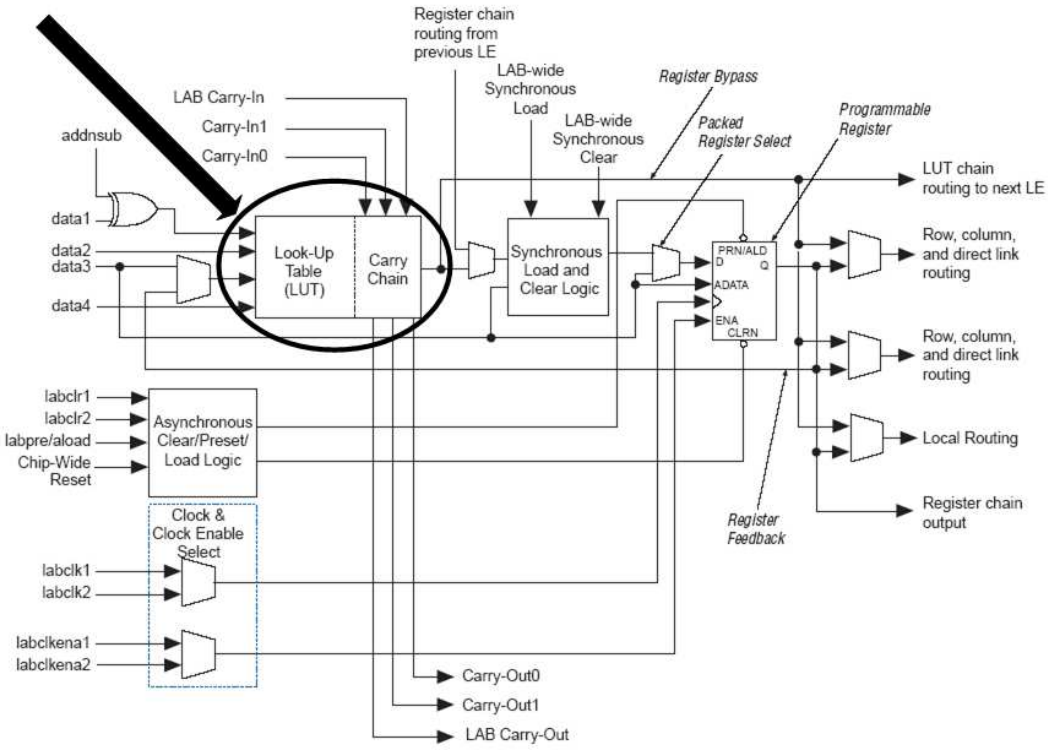


Abbildung 2.2: Aufbau eines Logic Element (LE)

2.2 Der Oszillator

Auf dem Board befindet sich ein Oszillator, welcher ein Rechtecksignal mit 33.33 MHz liefert. Dieses Signal stellt den globalen *Clock* des FPGA dar und wird die-

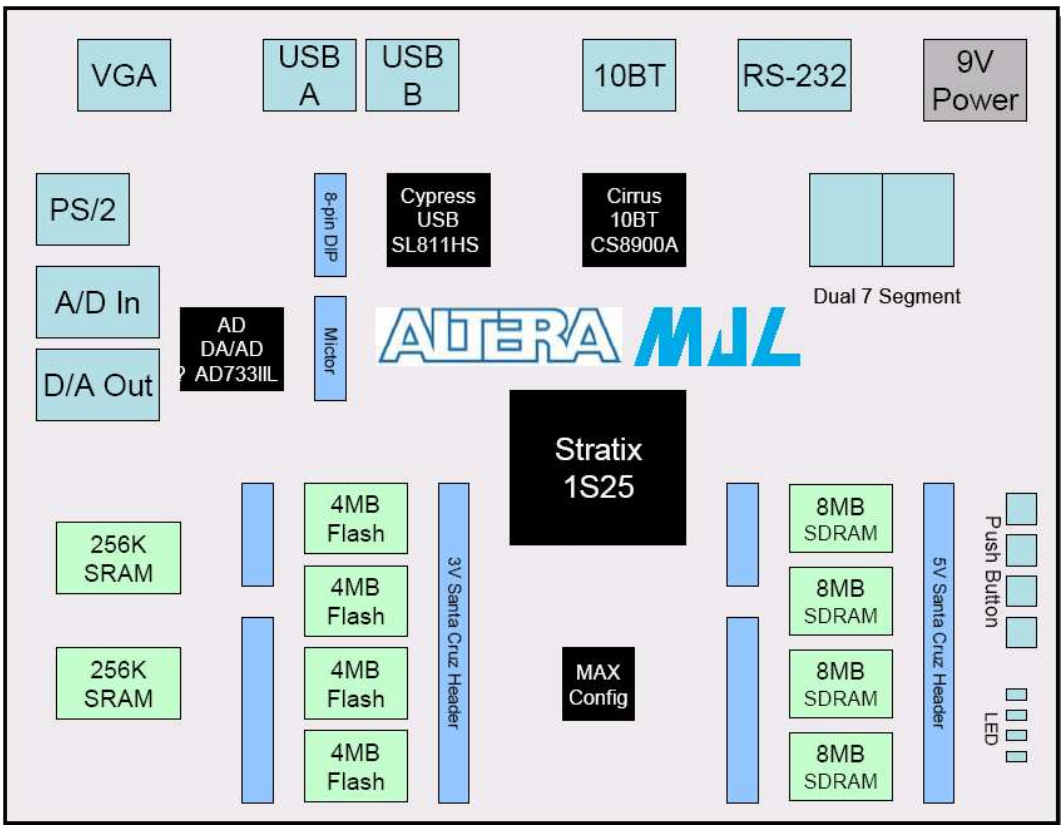


Abbildung 2.3: Das MJL-Stratix-Development-Board

sem über ***Pin N3*** zugeführt. Diese Pin-Zuweisung ist für die Konfiguration und Programmierung des FPGA von größter Bedeutung!

2.3 LEDs und Siebensegmentanzeigen

Auf dem Board stehen zwei benutzerprogrammierbare LEDs zur Verfügung. Jede dieser LEDs verfügt über einen Vorwiderstand mit 330 Ω. Eine LED leuchtet, wenn an dem zugehörigen Anschluß logisch 1 angelegt wird. Die LEDs sind also high-aktiv. Tabelle 2.1 zeigt die Pinbelegung der LEDs.

LED	**Pin**
LED 1	A6
LED 2	A7

Tabelle 2.1: Pinbelegung der LEDs

Zusätzlich stehen zwei Siebensegmentanzeigen zur Verfügung. Jedes Segment ei-

ner dieser Anzeigen kann durch Anlegen von logisch 0 an den zugehörigen I/O-Pin zum Leuchten gebracht werden (low-aktiv). Die Anzeigen sind nicht gemultiplext, können daher also permanent direkt angesteuert werden. Abbildung 2.4 zeigt die Bezeichnungen der einzelnen Segmente, Tabelle 2.2 listet die Pinbelegung auf.

Hinweis: Der Dezimalpunkt ist zwar vorhanden, jedoch nicht mit dem FPGA verbunden.

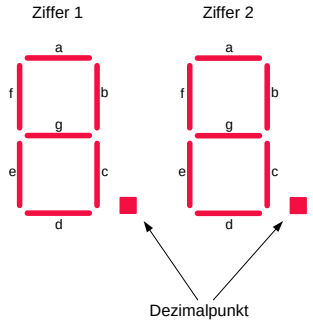


Abbildung 2.4: Die Siebensegmentanzeigen

Segment	Pin für Digit 1	Pin für Digit 2
a	Y11	R8
b	N7	R9
c	N8	R19
d	R4	R20
e	R6	R21
f	AA11	R22
g	T2	R23

Tabelle 2.2: Pinbelegung der Siebensegmentanzeigen

2.4 Taster

Die beiden Taster *SW1* und *SW2* stellen ein low-aktives Signal zur Verfügung. *SW1* ist über ***Pin A3*** mit dem FPGA verbunden, *SW2* über ***Pin A5*** . Beide Taster sind mit einem Pull-Up-Widerstand mit 1 $K\Omega$ ausgestattet.

Zusätzlich verfügt das Board über die dedizierten Taster *RESET* und *CLEAR*. Die Auswirkung des Tasters *RESET* entspricht dem Cold-Start des Boards.

Der Taster *CLEAR* ist mit dem dedizierten Pin ***DEV_CLRn*** (Pin AF17) des FPGA verbunden. Seine Funktionsweise hängt von der Konfiguration des FPGA ab.

Die Taster werden im Rahmen dieser Übung nicht verwendet.

2.5 DIP-Schalter

Mit dem FPGA sind 8 DIP-Schalter verbunden. Ein Input-Pin ist auf logisch 1 gesetzt, wenn der Schalter geöffnet ist, ein geschlossener Schalter setzt den Pin auf logisch 0. Tabelle 2.3 zeigt die Pinbelegung der Schalter.

Schalter	**Pin**
1	E9
2	B9
3	AB9
4	AD9
5	AD17
6	AC17
7	G19
8	E18

Tabelle 2.3: Pinbelegung der DIP-Schalter

2.6 Das VGA-Interface

Das Board verfügt über einen 15-poligen VGA-Anschluß. Über die fünf Signale *RED, GREEN, BLUE, HSYNC* und *VSYNC* kann der Bildschirminhalt gesteuert werden. Tabelle 2.4 zeigt die Pinbelegung des Steckers, Tabelle 2.5 zeigt die Verbindung des Interface mit dem FPGA.

Pin	**Signal**	**Pin**	**Signal**
1	Red	9	Plug
2	Green	10	GND
3	Blue	11	Monitor Sense 0
4	Reserved	12	Monitor Sense 1
5	GND	13	HSYNC
6	Red GND	14	VSYNC
7	Green GND	15	Reserved
8	Blue GND		

Tabelle 2.4: Pinbelegung des VGA-Interface

Wie man in Tabelle 2.5 sieht, verfügt das FPGA-Board über drei Signale für die Farben Rot und Grün sowie zwei Signale für Blau. Über eine Gewichtung mittels dreier bzw. zweier Widerstände wird damit ein DAC realisiert. Auf diese Weise können Farbschattierungen erzeugt werden. Dieser DAC wird in der Laborübung allerdings nicht eingesetzt, verwendet werden ausschließlich die Farben Rot, Grün und Blau.

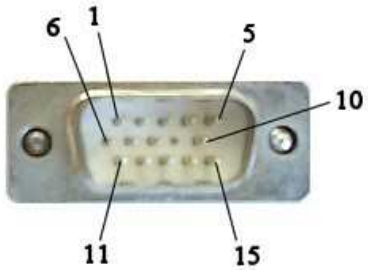


Abbildung 2.5: Ein VGA-Stecker

Dies wird erreicht, indem die drei bzw. zwei hardwaremäßig vorhandenen Signale im Design von ein und demselben Signal angesteuert werden.

Signal	FPGA-Pin	Signal	FPGA-Pin
red0	E22	blue1	T6
green0	E23	red2	T7
blue0	E24	green2	T24
red1	T4	hsync	F1
green1	T5	vsync	F2

Tabelle 2.5: Verbindung FPGA - VGA

Für das Protokoll der VGA-Schnittstelle sei auf Anhang 1.3.2 verwiesen.

2.7 Erweiterungs-Ports

Das FPGA-Board verfügt über mehrere Erweiterungs-Ports, welche mit Signalen der Spannung 3.3V sowie 5V arbeiten. Im Rahmen dieser Übung wird an diesen Ports das in Kapitel 2.8 beschriebene Interface-Board angeschlossen.

2.8 Das Interface-Board

Abbildung 2.6 stellt die Interface-Platine dar. Ihr Zweck besteht darin, das FPGA-Board mit dem Logikanalysator zu verbinden. Die weiteren auf dieser Platine untergebrachten Bauteile werden im Rahmen dieser Übung nicht verwendet. Bei der Bearbeitung der Beispiele kann es passieren, daß einige LEDs (rot, low-aktiv) zu leuchten, blinken, flackern o.Ä. beginnen. Dies ist völlig normal, hat keinen Einfluß auf den Übungsablauf und stellt keinen Grund zur Besorgnis dar.

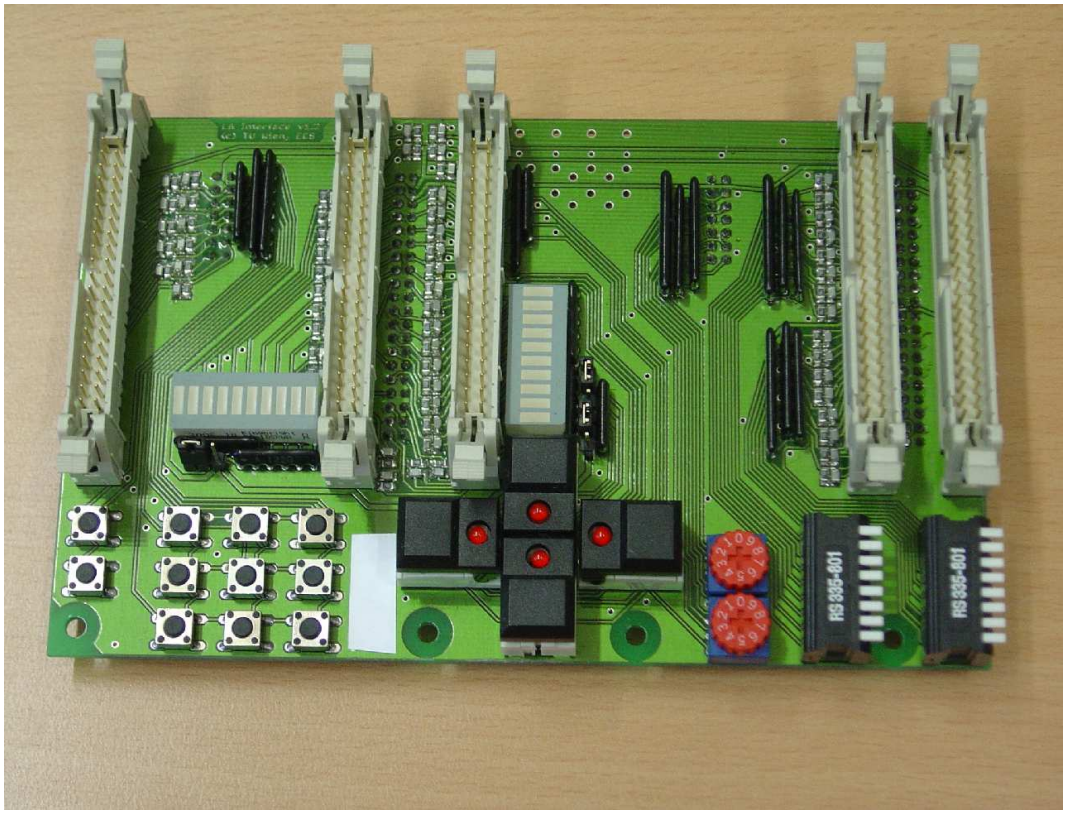


Abbildung 2.6: Das Interface-Board

LU-SKRIPTUM — Kapitel 3

Der Logikanalysator

Inhalt

3.1 Allgemeines

Ein Logikanalysator ist ein Meßgerät zur Darstellung von Signalverläufen. Im Unterschied zum Oszilloskop werden jedoch nur die Signalzustände *HIGH* und *LOW*, abhängig von nur einer Threshold-Spannung, unterschieden. Man kann ihn also als Digital-Oszilloskop mit nur einem Bit Vertikalauflösung betrachten.

Dafür verfügen Logikanalysatoren über wesentlich leistungsfähigere Trigger-Mechanismen sowie über die Fähigkeit zur Darstellung von weitaus mehr Signalverläufen. Die in dieser Laborübung verwendeten Geräte etwa können bis zu 90 Datenkanäle simultan erfassen und eine Auswahl davon darstellen.

Eingangssignale werden durch Vergleich mit einer Threshold-Spannung digitalisiert und in einer Liste gespeichert. Daraus ergibt sich eine Beschränkung des Beobachtungsintervalls durch die Größe des verfügbaren Hauptspeichers und der

Sampling-Rate. Nicht nur, aber auch aus diesem Grund ist es wichtig, Triggerbedingungen möglichst präzise zu formulieren.

Die folgenden Kapitel stellen eine Kurzanleitung zur Bedienung dieser Geräte dar; für weiterführende Informationen sei auf die Online-Hilfe verwiesen. An dieser Stelle sei ausdrücklich darauf hingewiesen, daß die Logikanalysatoren ***ausschließlich*** an die dafür vorgesehenen Ausgänge auf dem dafür vorgesehenen Board angeschlossen werden dürfen. Anderenfalls könnte die Zerstörung des Meßgeräts die Folge sein!

3.2 *Timing Analysis* und *State Analysis*

Ein Logikanalysator bietet üblicherweise zwei Betriebsmodi: den *"Timing Analyzer"* und den *"State Analyzer"*.

Der *"Timing Analyzer"* ist jener Betriebsmodus des Logikanalysators, welcher über die meisten Analogien zum Oszilloskop verfügt. Die horizontale Achse repräsentiert dabei kontinuierlich die Zeit, die vertikale die digitalisierte Spannungsamplitude. Der *"Timing Analyzer"* verfügt über einen internen Takt und mißt somit asynchron.

Der *"State Analyzer"* hingegen erhält seinen Takt vom getesteten System. Er reagiert auf die steigende oder fallende Flanke des eingespeisten Taktes und mißt zu diesem Zeitpunkt die übrigen Signale. Ein Zustand ist in diesem Zusammenhang für eine logische Schaltung ein Abbild der interessierenden Signale zu jenem Zeitpunkt, zu dem sie für die Schaltung gültig sind, also bei den aktiven Taktflanken. Ein *"State Analyzer"* arbeitet also synchron.

Als Faustregel kann man sagen, daß ein *"State Analyzer"* angibt, *was* passiert (die Abfolge der Zustände), und ein *"Timing Analyzer"* angibt, *wann* etwas passiert.

3.3 Grundlegende Bedienungselemente

Bei dem in dieser Laborübung verwendeten Logikanalysator (siehe Abbildung 3.1) handelt es sich prinzipiell um einen Windows-basierten PC mit speziellen Meßeinschüben. Dementsprechend ist das User Interface des Logikanalysators ein Stück Windows-Software, deren grundsätzliche Behandlung / Bedienung als bekannt vorausgesetzt wird. Zusätzlich befinden sich auf der Vorderseite ein Taster zum Einschalten des Geräts, ein Taster zum Starten einer Messung (*Run*), ein Taster zur Durchführung kontinuierlicher Messungen (*Run Rep.*) sowie ein Taster zum Abbruch einer Messung (*Stop*). Der Drehknopf kann – situationsabhängig – für diverse Scroll- oder Zoom-Operationen verwendet werden.

3.4 Der Meßprozeß

Dieses Kapitel beschreibt, wie und in welcher Reihenfolge eine sinnvolle Messung mit einem Logikanalysator durchzuführen ist. Den Überblick zeigt Abbildung 3.2.

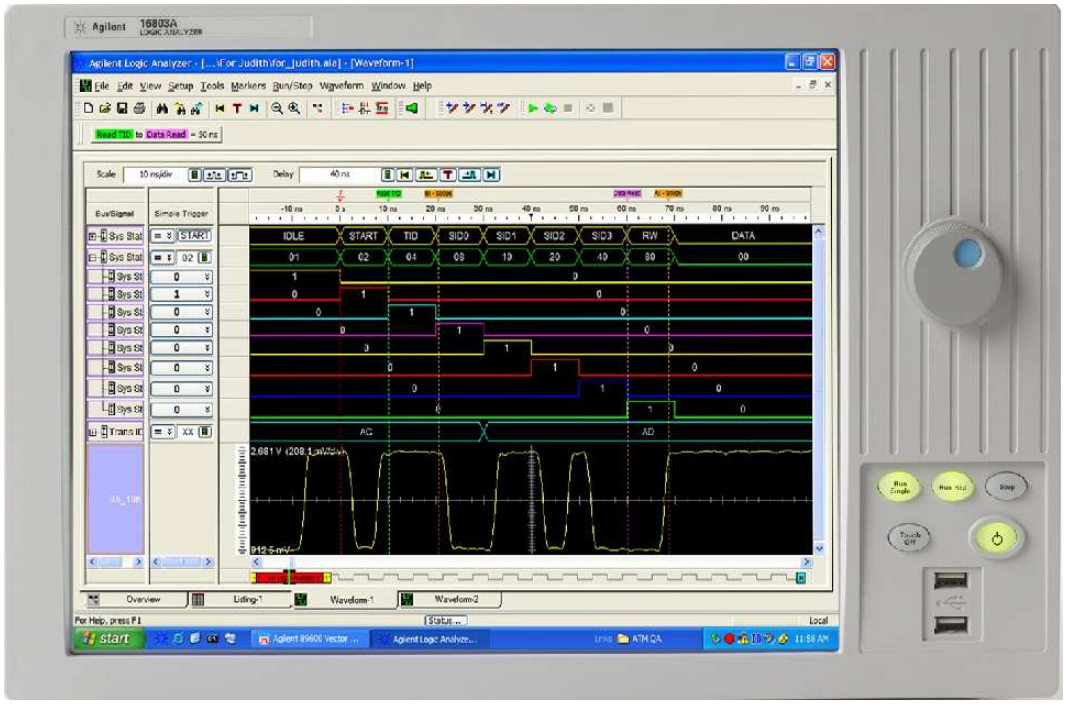


Abbildung 3.1: Das Bedien-Panel des Logikanalysators Agilent 16803

Es sei darauf hingewiesen, daß eine Messung zwar prinzipiell zu jedem Zeitpunkt gestartet werden kann, (aussagekräftige) Ergebnisse wird man jedoch frühestens nach Einstellung der Triggerbedingungen erzielen.

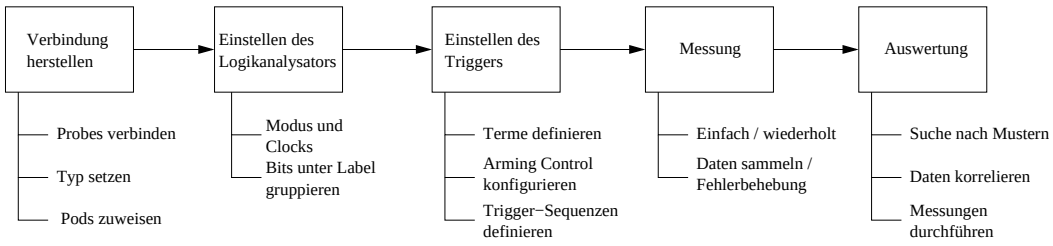


Abbildung 3.2: Der Meßprozeß

3.4.1 Verbindung herstellen

Das getestete System wird mittels Prüfleitungen mit dem Logikanalysator verbunden; dadurch werden die Signale auf die Kanäle des Logikanalysators abgebildet. Jeweils 16 solcher Prüfleitungen sind zu einem sogenannten *Pod* zusammengefaßt. Beim Setup des Logikanalysators ist darauf zu achten, logisch zusammengehörende Leitungen möglichst zu gruppieren.

Physikalisch sollte der Logikanalysator bereits mit dem Interface-Board verbunden sein, da kein Grund zur Trennung dieser Verbindungen besteht. Sollten die Pods nicht angeschlossen sein, wenden Sie sich bitte an die Übungsbetreuung.

Im Rahmen dieser Laborübung werden *Pod 1* bis *Pod 5* verwendet. Prinzipiell erfolgt der Anschluß sämtlicher Leitungen ausschließlich dann, wenn sich alle Geräte im ausgeschalteten Zustand befinden. Dadurch soll die Beschädigung oder Zerstörung dieser Geräte hintangehalten werden. Auch die Masseleitung muß angeschlossen werden! Diese Verbindungen bewerkstelligt die Interface-Platine.

3.4.2 Konfiguration des Logikanalysators

Da die Konfiguration des Logikanalysators eine langwierige und fehlerträchtige Angelegenheit ist, stellen wir Ihnen in dieser Übung ein Konfigurationsfile zur Verfügung. Dieses File finden Sie in Ihrem Home-Verzeichnis unter *agilent/didelu.ala*. TODO
Zum Laden der Konfiguration bedient man sich der Windows-üblichen *File→Open*-Vorgehensweise.

Achtung: die Formulierung von Triggerbedingungen, Wahl des Meßmodus u.Ä. zählt aber weiterhin zu den von Ihnen durchzuführenden Aufgaben!

3.4.3 Einstellen des Triggers

Bevor der Trigger eingestellt wird, muß der Meßmodus gewählt werden. Im Menü *Setup→Timing/State(Sampling)...*, Tab *Sampling* kann zwischen Timing- und State-Mode gewählt werden. Dort können ebenfalls Sampling-Rate und Trigger-Position eingestellt werden. Es sollte allerdings in dieser Übung nicht notwendig sein, derartige Einstellungen vornehmen zu müssen.

Der in dieser Übung verwendete Logikanalysator unterscheidet grundsätzlich zwei Arten von Triggern: *Simple Trigger* und *Advanced Trigger*. Der *Simple Trigger* erlaubt das Triggern auf willkürliche Flanken oder Pegel. Für komplexere Meßaufgaben werden die *Advanced Trigger* eingesetzt (*Setup→Advanced Trigger...*).

Auf der linken Seite dieses Fensters befinden sich die vier Tabs *Edge*, *Bus Pattern*, *Other* und *Advanced*, welche die Triggerbedingungen weiter unterteilen. Zu jeder dieser Kategorien existiert eine Liste von Trigger-Funktionen (blaue Quadrate rechts der Tabs). Aus diesen Trigger-Funktionen kann nun mittels Drag-and-Drop im rechten Teil des Fensters eine – der Aufgabenstellung entsprechende – Trigger-Sequenz zusammengestellt werden. In den einzelnen Stufen der Trigger-Sequenz muß nun noch (meistens mittels Drop-Down-Liste) eingestellt werden, auf welches Signal wie getriggert werden soll und wie bei Eintreten der Triggerbedingung mit der Datenspeicherung verfahren werden soll. In anderen Worten: die Abfolge der Trigger-Funktionen beschreibt die Grobform der Triggerbedingung, die Feineinstellungen werden innerhalb jeder Stufe vorgenommen.

Sie können Triggerbedingungen auch speichern, um sie nicht jedes Mal neu eingeben zu müssen.

3.4.4 Durchführen der Messung

Mit dem Button *Run* oder der Taste *F5* wird eine einzelne Messung gestartet. Durch den Button *Repetitive Run* oder die Tastenkombination *Ctrl-F5* werden wiederholte Messungen veranlaßt; dieses Feature wird aber im Rahmen dieser Laborübung nicht verwendet.

3.4.5 Datenauswertung

Nach Durchführung der Messung werden die aufgezeichneten Daten sowohl als Waveform als auch als Listing dargestellt. Zur Auswahl der Darstellung befinden sich an der Unterkante des Fensters entsprechende Tabs.

Mit einem rechten Mausklick auf ein Signal / einen Bus kann über den Befehl *Base* dessen Darstellungsformat gewählt werden. Um Messungen an den Waveforms vornehmen zu können, benötigt man sogenannte *Marker*, welche mit dem gleichnamigen Menüpunkt generiert und bearbeitet werden können. Um Abstände zwischen zwei Markern bestimmen zu können, wählt man aus dem Menü *Marker* die Funktion *New Time Interval Measurement*. Daraufhin erscheint oberhalb des Waveform-Fensters ein Button, auf welchem das entsprechende Intervall abgelesen werden kann.

3.5 Der Symbolmechanismus

In manchen Anwendungsbereichen, etwa dem Debuggen der Programmabarbeitung von Prozessoren oder der Zustandsfolge von State Machines, ist es angenehm, die Zustände von Bussen mit einfach verständlichen Namen versehen zu können. Dies geschieht mit dem Befehl *Setup→Symbols...*. In der Ihnen zur Verfügung gestellten Konfiguration wurden bereits die Namen der Zustände der Hsync- und Vsync-FSM eingegeben. Dies sollte sowohl die Analyse als auch das Debuggen erleichtern. Nicht benannte Zustände eines Signals oder Busses werden vom Logikanalysator numerisch (je nach eingestellter Basis) angezeigt.

3.6 Screenshots

Die für das Protokoll notwendigen Screenshots können angefertigt werden, indem – bei geeignet gewähltem Bildausschnitt – die Taste *Print* gedrückt wird. Dadurch wird der Screenshot in der Zwischenablage gespeichert. Von dort kann er in Programme wie etwa *Paint* kopiert werden (z.B. Tastenkombination *Ctrl-V*).

LU-SKRIPTUM — Kapitel 4

Design-Flow

Unter *Design-Flow* versteht man jene Abfolge von Aktionen, welche zur professionellen und korrekten Entwicklung eines ASIC (*Application Specific Integrated Circuit*, [Smi97]) notwendig sind. Bild 4.1 zeigt diesen Ablauf mit den zugehörigen Simulationsschritten.

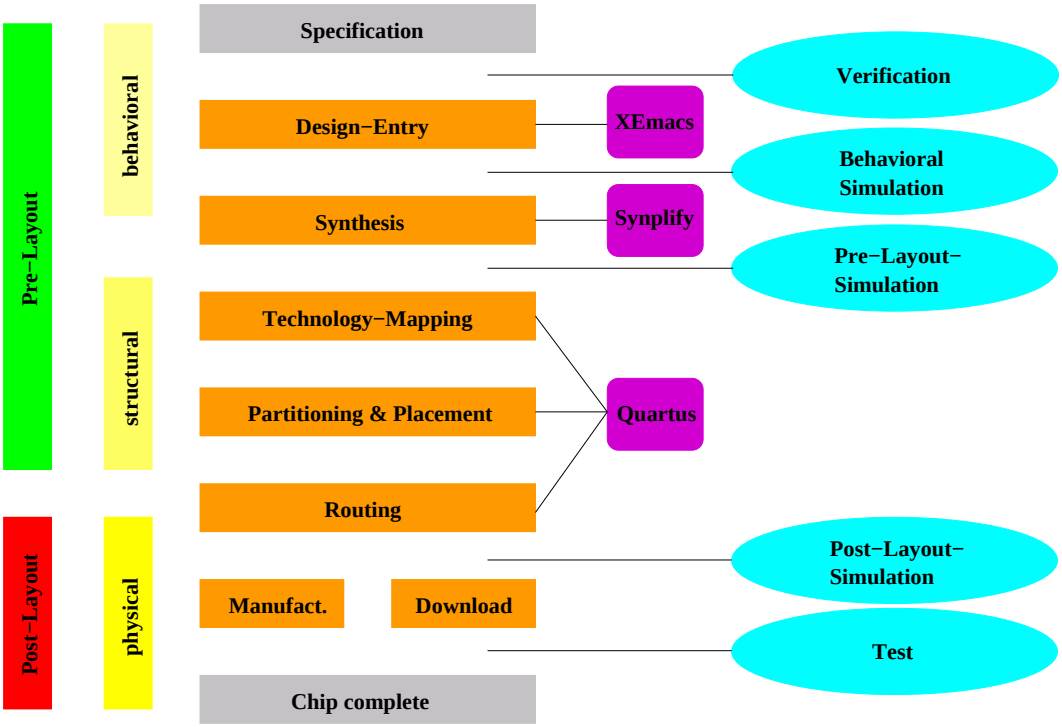


Abbildung 4.1: Design-Flow

Unter ***Spezifikation*** versteht man eine verbale oder formale Beschreibung der Funktionalität des Designs in einer beliebigen, dem Menschen zugänglichen Form.

Im Gegensatz dazu wird beim ***Design-Entry*** die Idee in einer Form beschrieben, die vom Computer erfaßt werden kann, trotzdem aber die menschliche Intuition unterstützt und eine effiziente Darstellung ermöglicht. Grundsätzlich kommen hier einerseits Hardware-Beschreibungssprachen (*HDL, Hardware Description Language*) oder andererseits graphische Eingabeverfahren (*schematic entry*) zum Einsatz.

Es ist wichtig anzumerken, daß man sich hier immer noch auf Verhaltensebene befindet. Deshalb kann mit der anschließend durchzuführenden Simulation (***Behavioral Simulation***) nur das Verhalten des Designs ohne Rücksichtnahme auf technologische oder elektrische Randbedingungen überprüft werden. Auch über das tatsächliche Timing des Designs kann keine Aussage gemacht werden. Man prüft also nur, ob die Spezifikation korrekt umgesetzt wurde. Grundsätzlich muß hier gesagt werden, daß eine nicht erfolgreich durchgeführte Simulation den Rückschritt in eine der davor gelegenen Stufen des Design-Flow zwingend erforderlich macht. In welche Stufe verzweigt wird, hängt vom Ausgang der Simulation im Kontext des Designs ab.

Nach erfolgreicher Simulation kann das Design ***synthetisiert*** werden. An dieser Stelle zeigt sich oftmals, daß viele Konstrukte zwar formulierbar und simulierbar, keineswegs aber synthetisierbar, d.h. in Hardware abbildbar, sind. In diesem Schritt wird eine sogenannte *Netzliste (Netlist)* erzeugt, eine Beschreibung der verwendeten Logikzellen und deren Verbindungen.

Als nächster Schritt wird die sogenannte ***Prelayout-Simulation*** durchgeführt. Hier wird hauptsächlich die korrekte Umsetzung des Designs durch das Synthese-Tool verifiziert. Dadurch, daß das Design nun aus Standard-Gattern, versehen mit einem sogenannten *Unit Delay*, aufgebaut ist, unterscheidet sich das Timing bei dieser Simulation etwas von jenem bei der *Behavioral Simulation*. Beim *Unit Delay* handelt es sich meist um geschätzte Standardwerte, weshalb das hier gefundene Timing keineswegs mit dem Timing des endgültigen ASIC übereinstimmen muß und kann. Man hat nun die Verhaltensebene verlassen und befindet sich auf Strukturebene.

Bis jetzt waren alle Entwicklungsschritte unabhängig von der endgültigen Zielplattform. Durch den Schritt des ***Technology Mapping*** legt man sich auf einen bestimmten Typ eines IC eines bestimmten Herstellers fest. Hierbei werden die bei der Synthese erzeugten Standard-Gatter durch zumeist optimierte Komponenten aus Bibliotheken des IC-Herstellers ersetzt.

Nun wird das Design auf die räumlichen Gegebenheiten des IC abgestimmt. Dieser Vorgang wird als ***Partitioning and Placement*** bezeichnet. Beim *Partitioning* werden große Systeme derart auf mehrere ICs aufgeteilt, daß die einzelnen Blöcke jeweils auf einem gegebenen IC Platz finden. Die exakte Anordnung der Logikzellen auf dem IC wird als *Placement* bezeichnet.

Im nächsten Schritt, dem ***Routing***, werden die Verbindungen zwischen den einzelnen Blöcken und Zellen des Designs hergestellt. Mit diesem Schritt verläßt man die Strukturebene und befindet sich nunmehr auf der physikalischen Ebene.

Die ***Postlayout-Simulation*** stellt die letzte Simulation im Design-Flow dar. Hier wird nun endgültig das reale Timing des Designs inklusive aller elektrischen

und physikalischen Phänomene getestet.

Wurde diese Simulation erfolgreich durchgeführt, kann mit der Herstellung des IC - oder im Fall eines FPGA mit dem ***Download*** des Designs - begonnen werden. Der so gefertigte IC muß nun natürlich noch getestet werden, bevor die Behauptung ***Chip complete*** aufgestellt werden kann.

LU-SKRIPTUM — Kapitel 5

Software

Inhalt

Dieser Abschnitt beschreibt den Design Flow anhand der eingesetzten Tools. Die Kapitel sind in jener Reihenfolge angeordnet, in welcher die einzelnen Schritte durchzuführen sind. Für eine inhaltliche Erklärung des Design Flow sei auf Kapitel 4 sowie auf das Vorlesungsskriptum verwiesen.

Die folgende Liste enthält die verwendeten Tools, deren Einsatzgebiet sowie Referenzen auf weiterführende Literatur.

***ModelSim SE* 6.3** Simulation ([mod], [Men07a], [Men07b])

***Synplify Pro* 8.1** Synthese ([syna], [Synb], [Syn05])

***Quartus II* 6.0** Place & Route ([alta], [Altb])

An dieser Stelle sei nochmals darauf hingewiesen, daß für jedes Beispiel die zu bearbeitenden Source-Files aus dem Verzeichnis `Angabe` in das Verzeichnis `Designflow/src` zu kopieren sind, da Sie nur in Letzterem über Schreibberechtigung verfügen.

Die meisten der verwendeten Programme verfügen über einen Editor. Als Alternative kann auch XEmacs verwendet werden. Dabei ist zu beachten, daß Syntax-Highlighting mit dem Befehl *Options* → *Syntax Highlighting* → *In This Buffer* aktiviert werden muß. Um diese Einstellung nicht bei jedem Start des Editors erneut vornehmen zu müssen, sollte die Konfiguration mit dem Befehl *Options* → *Save Options to Init File* gespeichert werden.

5.1 Behavioral Simulation

Die Verhaltenssimulation verifiziert die Funktionalität des Designs ohne auf Laufzeiten Rücksicht zu nehmen. Sie wird mit dem Programm *ModelSim SE* durchgeführt, welches zu diesem Zweck gestartet werden muß. Prinzipiell werden die Design-Files dabei in eine Bibliothek compiliert, welche üblicherweise den Namen `work` trägt. Dieses Compilat wird anschließend simuliert.

Befehle können sowohl mit der Maus ausgewählt als auch per Command Line eingegeben werden. Bei der Befehlsauswahl per Maus bezeichnet das Symbol → im Weiteren eine Menü-Auswahl, also z.B. *File* → *Open*. Wissen um die grundlegende Bedienung von Windows-Programmen wird vorausgesetzt.

Achten Sie immer darauf, die richtigen Files zu simulieren.

5.1.1 Die Simulationsbibliothek

Im ersten Schritt wird mit dem Mausbefehl *File* → *Change Directory...* in das `sim/beh`-Verzeichnis des aktuellen Projektes gewechselt (übliche Verzeichnisauswahl unter Windows). Danach wird mit dem Befehl *File* → *New* → *Library...* in diesem Verzeichnis eine neue Arbeitsbibliothek erzeugt. Im erscheinenden Fenster muß unter *Create* die Option *a new library and a logical mapping to it* aktiviert werden. In den beiden Textfeldern ist der Name der Bibliothek, `work`, einzutragen, falls dieser nicht schon als Default-Wert vorgegeben wurde.

5.1.2 Simulation

Zum Übersetzen der Design-Files wird der Befehl *Compile* → *Compile...* ausgeführt, woraufhin der Dialog *Compile Source Files* erscheint. In der Drop-Down-Liste *Li-*

brary muß die Bibliothek `work` ausgewählt werden. Nun wechselt man in das `src`-Verzeichnis des Projektes und wählt die zu übersetzenden Dateien aus. Die Source-Files müssen entsprechend ihrer Hierarchie im Design (d. h. das Top-Level zuletzt) compiliert werden. Hier wird auch die Testbench - sofern vorhanden - übersetzt, und zwar als absolut letztes File. Im Falle der Verhaltenssimulation heißt dieses File `vga_beh_tb.vhd`. Sollte das Design über Speicher verfügen, werden diese vor allen anderen Dateien übersetzt. PLLs werden ebenfalls zuallererst compiliert. Es ist zu beachten, daß für die korrekte Simulation von Designs mit Speichern ein `.hex`- oder `.mif`-File zur Beschreibung des Speicherinhaltes im Simulationsverzeichnis vorhanden sein muß.

Für jede Simulation sind alle notwendigen Dateien zu compilieren. Notwendige Dateien für die erste Simulation sind alle Dateien des Designs mit Ausnahme der PLL und dem zugehörigen Top-Level (siehe Kapitel 5.6). Werden danach Änderungen am Source oder an der Testbench vorgenommen, so sind nur die geänderten Dateien sowie alle in der Hierarchie darüber liegenden Files zu compilieren.

Im Zentrum der oberen Werkzeugleiste von *ModelSim SE* befindet sich ein Eingabefeld. Hier muß die Simulationszeit - entsprechend der gewünschten Simulationsdauer der Testbench - eingestellt werden.

Nun kann mit der Simulation begonnen werden. Dazu wird der Befehl *Simulate* → *Start Simulation ...* ausgeführt. Unter *Design* wird in der Bibliothek `work` die *Configuration* der Testbench (für Details siehe [IEE02] oder das Vorlesungsskriptum) ausgewählt.

5.1.3 Visualisierung

Um die Signalverläufe darzustellen, wird mit dem Befehl *View* → *Debug Windows* → *Wave* das entsprechende Fenster geöffnet. Im Fenster *Workspace*, Registerkarte *sim* muß die Architecture der Testbench markiert werden. Dann können im Fenster *Objects* die interessierenden Signale markiert und per Drag & Drop in das *Wave*-Fenster gezogen werden. Ebenso kann im Fenster *Workspace* zur Darstellung interner Signale durch ein hierarchisches Design navigiert werden. Der Befehl *Simulate* → *Run* → *Run nnn* startet nun die eigentliche Simulation. Dabei bedeutet *nnn* die zuvor eingestellte Simulationszeit. Das Ergebnis (die Signalverläufe) kann im Fenster *Wave* betrachtet und analysiert werden. Abbildung 5.1 zeigt einen typischen *ModelSim SE*-Bildschirm.

Der für Messungen benötigte zweite Cursor kann durch *Add* → *Cursor* hinzugefügt werden. Dazu muß das *Wave*-Fenster das aktive Fenster sein.

Um einen besseren Überblick über die Signalverläufe zu bekommen, ist es sinnvoll, das *Waves*-Fenster auf die gesamte Bildschirmfläche zu vergrößern. Dazu klickt man den +-Button rechts oben in diesem Fenster an. Daraufhin wird das Fenster maximiert und der besagte Button ändert sich in ein −. Damit kann das Fenster wieder geschrumpft werden.

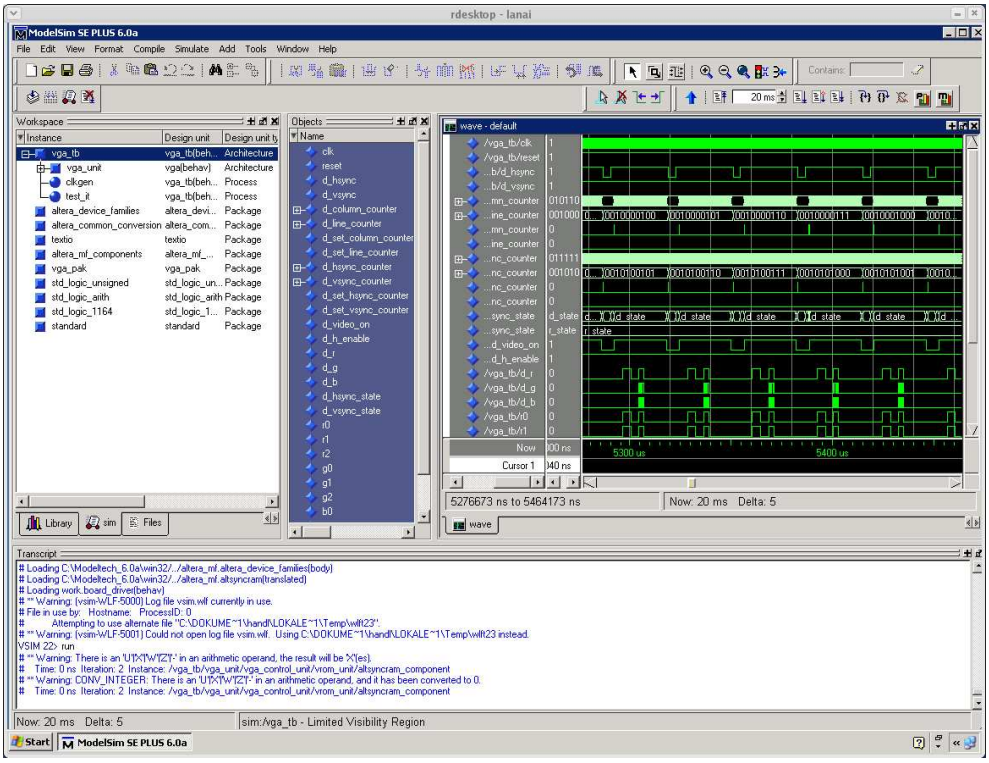


Abbildung 5.1: *ModelSim SE*

Signalnamen werden per Default inklusive ihrer Lage im Design, also mit dem Pfad durch die Design-Hierarchie, der zu ihnen führt, angezeigt. Gelegentlich - vor allem, wenn der Signalname innerhalb des Designs eindeutig ist - verschwenden diese Pfadangaben Platz auf dem Bildschirm. Im Menü *Tools/Options/Wave Preferences...* kann in der Karteikarte *Display* im Eingabefeld *Display Signal Path* die angezeigte Pfad-Tiefe eingestellt werden. Dabei bedeutet der Wert 0, daß der gesamte Pfad angezeigt wird, der Wert 1 unterdrückt jede Pfadangabe.

Inhalte von Speichern können betrachtet werden, indem der Befehl *View* → *Debug Windows* → *Memory* ausgeführt wird. Daraufhin erscheint eine weitere Registerkarte mit der Bezeichnung *Memory*, welche alle im Design vorhandenen Speicher auflistet.

5.2 Synthese

Die Synthese verwandelt den VHDL-Code in eine Netzliste und wird mit dem Programm *Synplify Pro* im Verzeichnis `syn` des Projektverzeichnisses durchgeführt. Zu diesem Zweck muß das Programm gestartet werden.

Nach dem Start zeigt das Programm das zuletzt geöffnete Projekt. Handelt es sich nicht um das zu bearbeitende Projekt, muß es geöffnet oder (beim ersten Mal)

neu erstellt werden.

5.2.1 Ein neues Projekt anlegen

Durch den Befehl *File → Build Project...* wird ein Fenster geöffnet. Im oberen Teil des Fensters navigiert man zuerst zum `src`-Verzeichnis des Projektes und wählt anschließend die VHDL-Dateien des Designs aus. Ein Klick auf den Button *<-Add* fügt die gewählten Dateien dem Projekt hinzu. Sind alle notwendigen Dateien hinzugefügt, wird der Dialog mit dem Button *OK* beendet. Hier dürfen etwaige im Design vorhandene *Speicher nicht vergessen* werden.

Im Projektfenster (links) findet man nun ein Verzeichnis `VHDL`, in dem sich die Source-Files befinden. Die Dateien werden angezeigt, sobald man auf das nebenstehende Plus-Symbol klickt. Nun müssen die Dateien per Drag & Drop in die richtige Reihenfolge entsprechend der Hierarchie im Design gebracht werden. Zuerst wird eventuell generierter Code (Speicher o.Ä., wird nur in Beispiel 1 verwendet, welches Sie aber nicht synthetisieren müssen) gereiht, dann das Package, die Top-Level–Architecture und anschließend die Module (siehe Abbildung 1.4). Bei jedem Modul sowie beim Top-Level muß die Entity vor der Architecture angeführt werden. Hinweis: Die PLL (siehe später) muß nicht mitsynthetisiert werden!

Zuletzt wird das Projekt mit dem Befehl *File → Save As...* gespeichert. Im erscheinenden Dialog navigiert man zuerst in das `syn`-Verzeichnis des aktuellen Designs, trägt im Feld *Dateiname* den Projektnamen **vga** ein und klickt auf den Button *Speichern*. Damit wurde ein neues *Synplify Pro*-Projekt erstellt.

5.2.2 Ein Projekt öffnen

Der Button *Open Project...* öffnet einen Datei-Auswahldialog, in welchem das Projekt gewählt werden kann. Projektdateien haben die Extension *.prj* und befinden sich im `syn`-Verzeichnis des aktuellen Designs.

5.2.3 Device-Optionen und Constraints

Mit dem Button *Impl Options...* können die Optionen für das Projekt festgelegt werden.

Im Register *Device* wird im Feld *Technology* der Typ *Altera STRATIX* ausgewählt. Im Feld *Part* wählt man den Eintrag *EP1S25*. Als *Package* wird *FC672* eingestellt, der Wert *Speed* wird auf *-6* festgelegt.

Das Resultat der Synthese ist eine Verilog-Netzliste. Der Name des erzeugten Files muß im Register *Implementation Results* im Eingabefeld *Result File Name* auf `vga.vqm` geändert werden.

Um das synthetisierte Design auch simulieren zu können, muß im Register *Implementation Results* die Option *Write Mapped VHDL Netlist* aktiviert werden. Im Register *VHDL* muß im Eingabefeld *Top Level Entity:* der Name der jeweiligen Top-Level

Entity eingetragen werden, in unserem Fall also `vga`. Da sonst keine Änderungen in diesem Dialog vorzunehmen sind, kann das Fenster mit dem Button *OK* geschlossen werden.

Am linken Bildschirmrand muß nun das Eingabefeld unterhalb von *Frequency (MHz)* aktiviert und der Wert *25,175* (die VGA-spezifische Frequenz) eingetragen werden. Weiter unten müssen die Optionen *FSM Compiler* (zur Optimierung der State Machines) und *Ressource Sharing* (ebenfalls eine Form der Optimierung) aktiviert werden, falls sie nicht schon aktiviert sind.

5.2.4 Die Synthese

Die Synthese kann nun mit dem Button *Run* gestartet werden. Das *TCL*-Fenster (unten links) zeigt die gerade abgearbeiteten Syntheseschritte und deren Ergebnis (Fehler, Warnungen, Notes). Im fehlerfreien Fall erscheinen nach einiger Zeit im Ergebnisfenster (rechts) die erzeugten Dateien. Im *TCL*-Fenster ist wieder der Prompt (%) zu sehen. Sollten Fehler oder Warnungen aufgetreten sein, sind die betroffenen Dateien im Projektfenster mit einem Rufzeichen markiert. Die Meldungen selbst können im entsprechenden Register an der Fenster-Unterkante gelesen werden. Ein Doppelklick auf eine solche Fehlermeldung oder Warnung öffnet automatisch die Datei, auf welche sich die Meldung bezieht, an der betroffenen Stelle (sofern lokalisierbar). Abbildung 5.2 zeigt einen typischen *Synplify Pro*-Bildschirm.

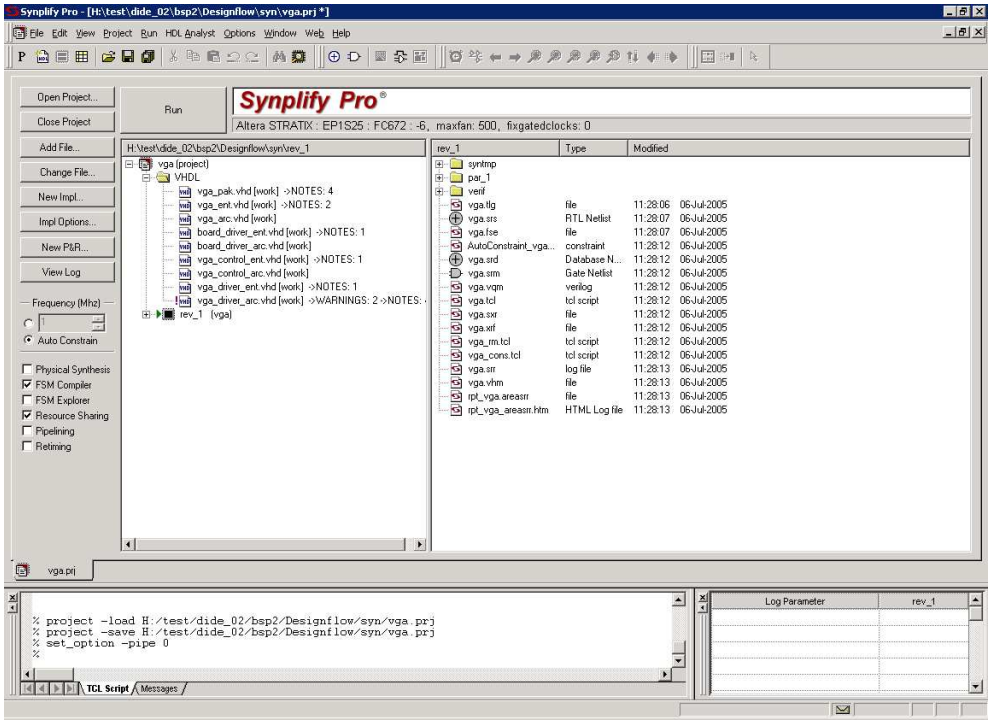


Abbildung 5.2: *Synplify Pro*

Am unteren Bildschrimrand sind die zwei Karteikarten `TCL-Script` und `Messages` sichtbar. Sie enthalten die ihren Namen entsprechenden Meldungen. Die Ihnen zur Verfügung gestellten Designs dürfen - bei richtiger Verwendung des Programms - bei der Synthese weder Warnings noch Errors melden. Wenn Sie Code hinzufügen oder modifizieren, müssen eventuell auftretende Errors auf jeden Fall beseitigt werden, da das Design sonst nicht synthetisierbar ist. Aber auch Warnings müssen auf jeden Fall ernst genommen werden, da sie auf eventuelle Unsauberkeiten oder "Gefahrenquellen" im Code hinweisen, die im Weiteren beim Place & Route zu Fehlern führen könnten. Kurz gesagt müssen für ein funktionierendes Design auch sämtliche aufgetretenen Warnings beseitigt werden.

5.2.5 Hierarchien, RTL- und Technologie–Ansicht

Das Ergebnis der erfolgreichen Synthese kann auf zwei Ebenen betrachtet werden: auf Gatter- oder auf Verhaltens-Ebene.

Das RTL-Fenster wird mit dem Befehl *HDL Analyst* → *RTL* → *Hierarchical View* geöffnet. Ist das Design hierarchisch aufgebaut, kann mit dem Befehl *View* → *Push/Pop Hierarchy* durch das Design navigiert werden. Ist das Hinabsteigen um eine weitere Stufe nicht möglich, ist dies am durchgestrichenen Cursor zu erkennen.

Die Technologie-Ansicht der synthetisierten Schaltung wird mit dem Befehl *HDL Analyst* → *Technology* → *Hierarchical View* dargestellt werden. Auch hier gilt das oben über die Navigation im Design Gesagte.

5.3 Pre-Layout–Simulation

Die Pre-Layout–Simulation wird ebenso wie die Verhaltenssimulation durchgeführt (siehe Kapitel 5.1), jedoch wird an Stelle der VHDL-Sourcen der im vorigen Schritt synthetisierte Code simuliert.

Mit anderen Worten entspricht die Durchführung dieser Simulation dem oben beschriebenen Prozedere mit Ausnahme der Dateiauswahl und des Simulationsverzeichnisses.

Bei dieser Simulation wechselt man in *ModelSim SE* zuerst in das Verzeichnis `sim/pre`. Nach dem Erzeugen der `work`-Bibliothek werden nun mit dem Befehl *Compile* → *Compile...* das synthetisierte File `vga.vhm` im Verzeichnis `syn/rev_1/` des Projektverzeichnis und die Testbench `vga_pre_tb.vhd` übersetzt. Danach kann die Simulation wie gewohnt gestartet werden. Werden Speicher verwendet, muß auch hier ein entsprechendes `.hex`- oder `.mif`-File im Simulationsverzeichnis vorhanden sein.

5.4 Partition, Place & Route

Place & Route findet im Verzeichnis `ppr` im Projektverzeichnis statt und wird vom Programm *Quartus II* (Abbildung 5.3) durchgeführt. Im Rahmen dieser LU ist das Partitionieren des Designs weder notwendig noch vorgesehen. Dieses Kapitel beschreibt das Place & Route der Simulationsvariante des Designs. Für die Erstellung der Downloadvariante ist eine PLL notwendig. Näheres dazu findet sich in Kapitel 5.6.

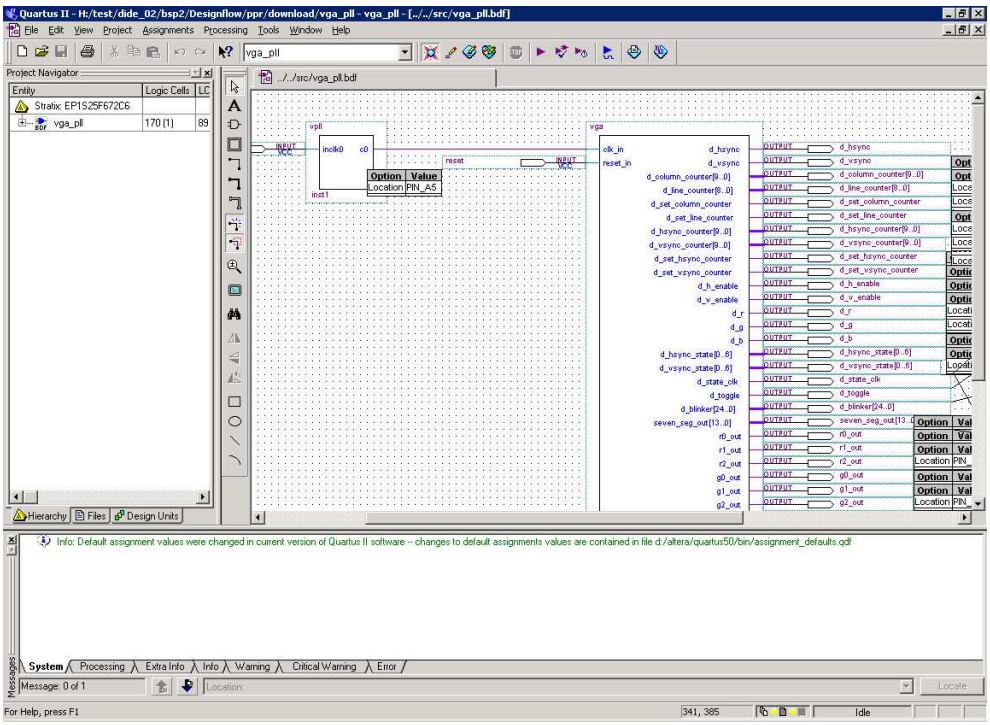


Abbildung 5.3: *Quartus II*

5.4.1 Ein Projekt anlegen

Bei jedem Aufruf von *Quartus II* kann es passieren, daß nach Updates gefragt wird. Diese Frage ist mit *No* zu beantworten.

Beim ersten Aufruf von *Quartus II* muß ein neues Projekt angelegt werden. Dies geschieht durch den Befehl *File* → *New Project Wizard...*. Im obersten Eingabefeld wird das Arbeitsverzeichnis, wie gesagt `ppr/sim` im aktuellen Projektverzeichnis, eingetragen. Im zweiten Eingabefeld wird der Projektname eingetragen, welcher immer mit der Top-Level–Entity ident ist, in unserem Fall also `vga`. Dieser Entity-Name wird auch im dritten Feld eingetragen. Für den nächsten Schritt muß auf den Button *Next* geklickt werden.

Im zweiten Schritt können Dateien zum Projekt hinzugefügt werden. Hier handelt es sich im Speziellen um die bei der Synthese erzeugte Netzliste mit der Extension `.vqm` im Verzeichnis `syn/rev_1` des Designs. Dazu klickt man den Button *Add...*, wählt das File, klickt im Dateiauswahldialog *OK* und weiters *Next*.

Es ist zu beachten, daß etwaige Speicher nicht extra angegeben und mitübersetzt werden müssen; sehr wohl muß aber ein `.hex`-File vorhanden sein.

Im nächsten Schritt wird das FPGA gewählt. In der Drop-Down–Liste *Family* wird die Option *Stratix* eingestellt. Darunter wird die Option *Specific device selected in 'Available devices' list* gewählt. Alle der drei rechten Drop-Down–Listen sind auf den Wert *any* einzustellen. Aus der linken Liste, *Available devices*, wird der Chip mit der Bezeichnung *EP1S25F672C6* ausgewählt.

Ein Klick auf den Button *Next* führt zur Einstellung der Entwicklungswerkzeuge. Im oberen Drop-Down, *EDA Design entry/synthesis* muß der Eintrag *Synplify Pro* ausgewählt werden. In der darunter befindlichen Drop-Down-Liste, *EDA simulation tool*, wird *ModelSim (VHDL)* eingestellt. Wichtig ist, das die Checkboxen links dieser beiden Drop-Downs aktiviert (mit einem Haken versehen) sind. Die dritte Checkbox wird nicht markiert.

Ein Klick auf *Next* zeigt eine Übersicht über die gewählten Projekteinstellungen. Mit dem Button *Finish* wird die Projekterstellung beendet.

5.4.2 Place & Route

Das eigentliche Place & Route wird mit dem Befehl *Processing* → *Start Compilation* gestartet. Im Report-Fenster, welches sich beim Start des Place & Route-Vorgangs öffnet, können diverse Maßzahlen des gerouteten Designs abgelesen werden. Sollten Fehler aufgetreten sein, befinden sich die Meldungen auf der Registerkarte *Processing* an der unteren Fensterkante. Im fehlerfreien Fall können die von *Quartus II* erzeugten Files zur Post-Layout–Simulation herangezogen werden.

5.5 Post-Layout–Simulation

Diese Simulation findet im Verzeichnis `sim/post` des aktuellen Projekts analog zur Pre-Layout–Simulation statt. Das zu simulierende File heißt nun `vga.vho` und befindet sich im Verzeichnis `ppr/sim/simulation/modelsim` unterhalb des aktuellen Projekts. Geladen wird es wie in den Kapiteln 5.1 und 5.3 beschrieben. Ebenso muß die Testbench `vga_pos_tb.vhd` compiliert werden.

Der wichtige Unterschied zu anderen Simulationen besteht darin, daß von *Quartus II* beim Place & Route exakte Timing-Informationen zur Verfügung gestellt wurden. Diese befinden sich in einem File mit der Extension `.sdo` im gleichen Verzeichnis wie das `.vho`-File.

Bevor nun die Simulation gestartet wird, muß der Befehl *Simulate → Start Simulation...* ausgeführt werden. Im daraufhin erscheinenden Dialogfenster wird die Registerkarte *SDF* ausgewählt, der Button *Add...* geklickt und das oben beschriebene `.sdo`-File ausgewählt.

Im Eingabefeld *Apply to Region* muß nach dem Backslash der Instanzname der Top-Level–Entity des *Designs* in der Testbench, nicht der Testbench selbst, angegeben werden. Simuliert werden soll schlußendlich das Design, und dieses wird ja in der Testbench instanziert. Für die Testbench gibt es natürlich keine Informationen aus dem Place & Route. Dieses Fenster wird nun mit dem Button *OK* geschlossen.

In der Registerkarte *Design* muß nach Expandieren der Library `work` die Configurtion der Testbench, `vga_conf_pos` ausgewählt werden. Dieses Fenster wird ebenfalls mit dem Button *OK* geschlossen; die Simulation kann wie gehabt gestartet werden. Zur Auswahl der Signale muß diesmal in der Registerkarte *sim* im Fenster *Workspace* der Eintrag *vga_unit* gewählt werden. Die Signale sind dann wieder im Fenster *Objects* aufgelistet.

Auch bei dieser Simulation muß gegebenenfalls ein `.hex`-File vorhanden sein.

5.6 Über PLLs

Eine PLL (*Phase-Locked Loop*, siehe etwa [TS99] oder [IEE96]) wird benötigt, um einerseits das Design mit einem stabilen Takt zu versorgen (stabiler als der vom Quarz gelieferte Takt) und andererseits, um diesen Takt genau einstellen zu können. Dies ist speziell im Übungsdesign notwendig, da, wie Kapitel 1.3.2 zu entnehmen ist, für das hier verwendete VGA-Design eine Frequenz von 25,175 MHz spezifiziert ist. Der Quarz des FPGA-Boards hingegen liefert 33,33 MHz.

Der Nachteil einer PLL besteht darin, daß sie mit der hier angewandten Design-Methodik sowie den zur Verfügung stehenden Tools nicht simuliert werden kann. Das bedeutet, daß für den Download eine eigene Variante des Designs geroutet werden muß. Das Place & Route findet im Verzeichnis `ppr/download` statt.

Der einzige Unterschied zu der oben beschriebenen Vorgehensweise beim Place & Route besteht nun darin, daß im Blockdiagramm-Editor von *Quartus II* das Design mit einer PLL verbunden wird. Dazu wird, wie gehabt, in *Quartus II* ein neues Projekt angelegt, diesmal aber im Verzeichnis `ppr/download`. Als Projektname und Name der Top-Level-Entity wird `vga_pll` eingegeben.

Beim Hinzufügen der Design-Files müssen das Synthese-Produkt `vga.vqm` und die PLL `vpll.vhd` (Verzeichnis `src`) ausgewählt werden. Danach wählt man im Navigationsfenster (links oben) die Registerkarte *Files* und nach einem Rechtsklick auf die Datei `vga.vqm` die Option *Create Symbol Files for Current File*. Nachdem das Symbol erzeugt wurde, bestätigt man mit *OK* und schließt das Report-Fenster.

Hilfestellung

Die Erzeugung dieses Blockdiagramms nimmt einiges an Zeit in Anspruch. Um Ih-

nen diese Zeit für die eigentliche Bearbeitung der Beispiele zu sparen, stellen wir Ihnen ein solches File (`vga_pll.bdf`) zur Verfügung. Es befindet sich - wie die übrigen Files - im Verzeichnis `Angabe` des jeweiligen Beispiels und muß von dort in das `download`-Verzeichnis dieses Beispiels kopiert werden. Mit dem Befehl *Project / Add/Remove Files in Project...* und dem Button *Add...* im daraufhin erscheinenden Fenster wird dieses File ausgewählt und dem Projekt hinzugefügt. Nun scheint es in der Projektübersicht an der linken Fensterkante in der Karteikarte *Files* auf. Dort klickt man es mit der rechten Maustaste an und wählt *Set as Top-Level Entity* aus. Im Blockdiagramm selbst sind sowohl die PLL als auch sämtliche Pads enthalten, das eigentliche Design fehlt aber noch (siehe "Manuelle Vorgangsweise" weiter unten). Es muß an den dafür vorgesehenen Platz zwischen den Pads eingefügt werden. *Das Design muß sehr sorgfältig plaziert werden, damit auch alle Pads richtig angeschlossen sind!*

Das Blockdiagramm wird nun mit *File → Save As...* im Verzeichnis `ppr/download` unter dem Namen `vga_pll.bdf` gespeichert. Abschließend klickt man im Navigationsfenster mit der rechten Maustaste auf dieses File und wählt *Set as Top-Level Entity* aus. Nun muß ein Mal das Place & Route gestartet und dann die Pinzuweisung durchgeführt werden (siehe Kapitel 5.7). Danach muß das Place & Route nochmals, wie oben beschrieben, gestartet werden und schließlich der Download (Kapitel 5.8) erfolgen.

Manuelle Vorgangsweise

Um das Blockdiagramm manuell zu erzeugen wird unter *File → New* in der Registerkarte *Device Design Files* der Eintrag *Block Diagram / Schematics File* ausgewählt. Mit einem Rechtsklick im daraufhin erscheinenden Zeichenfenster wird über *Insert → Symbol* in der Kategorie *Projekt* der Eintrag *vga* gewählt.

Zum Einfügen der PLL wählt man wieder Rechtsklick, *Insert → Symbol* und klickt dann unterhalb des Inputfeldes *Name* den Button mit den drei Punkten. Im erscheinenden Dateiauswahldialog navigiert man dann in das Verzeichnis `src` des aktuellen Beispiels und wählt dort die Datei `vpll.bsf`.

Praktischerweise ordnet man das PLL-Symbol links von jenem des Designs an. Dann verbindet man den Ausgang `c0` der PLL mit dem Eingang `clk` des VGA-Designs. Dies funktioniert wie in einem üblichen Zeichenprogramm; der Cursor verändert seine Form, sobald er über einen Port bewegt wird. Durch Drücken und gedrückt halten der linken Maustaste können dann Leitungen verlegt werden.

Nun müssen die übrigen Ports an sogenannten *Pads* angeschlossen werden. Dazu wird die rechte Maustaste gedrückt und im Kontextmenü *Insert/Symbol/libraries/primitives/pin/input* beziehungsweise *Insert/Symbol/libraries/primitives/pin/output* ausgewählt. Jeder Eingang und Ausgang muß mit einem derartigen Pin versehen werden.

Jedem Pin muß ein eindeutiger Name gegeben werden. Dazu wird mit der rechten Maustaste auf den jeweiligen Pin geklickt und im Kontextmenü der Punkt *Properties*

ausgewählt. In der daraufhin erscheinenden Maske wird der Name des Pins eingetragen, welcher dem Namen des Pins im Design entsprechen muß. Die Breite von Bussen wird, wie im Block `vga` im Blockdiagramm zu sehen ist, in eckigen Klammern eingefaßt.

Nun wird, wie oben ausgeführt, ein Place & Route-Durchgang gestartet, die Pinbelegung durchgeführt und nochmals geroutet. Dann kann man das Design auf das FPGA downloaden.

5.7 Die Pinbelegung

Für die Download-Version müssen - wie bereits erwähnt - die Pins zugewiesen werden. Auch hier gibt es wieder eine kleine Hilfestellung. Wichtig: Um den Logikanalysator im State-Mode betreiben zu können, muß das Signal `d_state_clk` auf Pin **G3** gelegt werden, da dieses Signal den notwendigen externen Takt für den Logikanalysator bereitstellt!

Hilfestellung

Um diese langwierige - und durchaus fehlerträchtige - Prozedur zu vereinfachen, wird Ihnen eine fertige Pinbelegung in Form eines TCL-Scripts zur Verfügung gestellt. Dieses Skript befindet sich im Angabe-Verzeichnis der einzelnen Beispiele (mit Ausnahme von Beispiel 1 natürlich) und muß in das `download`-Unterverzeichnis des jeweiligen Place & Route-Verzeichnis kopiert werden. Der Name des Skripts lautet `vga_pll.tcl`.

Um diese Pinbelegung auf Ihr Design anzuwenden, müssen Sie in *Quartus II* den Befehl *Tools* → *Tcl Scripts...* ausführen. Im linken Auswahlfenster wählen Sie dieses TCL-File unterhalb von *Project* aus und starten es mit dem Button *Run* rechts oben. Damit wird auch gleichzeitig ein Place & Route-Durchlauf gestartet.

Für die korrekte Funktionsweise dieses Scripts ist es zwingend erforderlich, daß bei der Benennung der Pads (wie in Kapitel 5.6 erklärt) die in Tabelle 5.1 angeführten Namen verwendet werden! Achtung: Nicht in jedem der vier Beispiele müssen auch alle Ports vorhanden sein!

Manuelle Vorgangsweise

Dazu wird der Befehl *Assignments* → *Pins* benutzt, wodurch der *Assignment Editor* geöffnet wird. Sollte die Pin-Liste nicht automatisch erscheinen, wird sie durch die Auswahl *All* → *Locations* → *Pin* in der Drop-Down-Liste rechts des Button *Category:* angezeigt.

Prinzipiell müssen nur jene Signale zugewiesen werden, die man an bestimmte Pins legen möchte. Die übrigen Signale werden von Quartus selbständig zugewiesen. Um den etwaigen Anschluß externer Geräte zu erleichtern (z.B. Logikanalysator), ist es von Vorteil, möglichst alle Pins gemäß einer gewissen Strategie zuzuweisen. *Im*

allgemeinen bedeutet jedoch jede Einschränkung der Freiheiten des Place & Route-Tools potentiell auch einen Performance-Verlust des Chips!

In der Spreadsheet-ähnlichen Tabelle werden nun die Pins des FPGA den Signalen zugewiesen. Vor der ersten Zuweisung ist diese Tabelle fast leer. In den beiden linken Spalten der ersten Zeile befinden sich die Einträge <<*new*>>.

Durch einen Doppelklick auf den Eintrag <<*new*>> in der Spalte *To* öffnet sich eine Drop-Down-Liste. Aus dieser Liste werden die Signale der Top-Level-Entity ausgewählt. Auf die selbe Art und Weise werden die zugehörigen Pins des FPGA in der Spalte *Location* ausgewählt. Jede Zeile enthält also ein Signal sowie den diesem Signal zugewiesenen Pin. *Dieser Vorgang muß für alle zuzuweisenden Pins durchgeführt werden!*

Änderungen an der Pinzuweisung können durch einen Doppelklick auf den entsprechenden Listeneintrag durchgeführt werden. Jedes Signal der Top-Level-Entity kann nur an exakt einen Pin gelegt werden. Bei der Zuweisung von Bussen ist zu beachten, daß jedes einzelne Signal des Busses zugewiesen werden muß, eine globale Zuweisung als Bus ist nicht möglich.

5.8 Download

Für den Download des Designs wird in *Quartus II* mit dem Befehl *Programmer* im Menü *Tools* das Download-Fenster geöffnet. Rechts oben im Abschnitt *Programming Hardware* sollte der *ByteBlasterMV* eingetragen sein. Falls nicht, muß er über den Button *Hardware Setup* ausgewählt werden.

Bei einer erfolgreichen Übersetzung des Designs hat Quartus die Datei `<project_name>.sof` erzeugt (für das erste Beispiel wird diese Datei zur Verfügung gestellt und kann gemäß den weiteren Ausführungen verwendet werden). Diese Datei wird mit dem Button *Add File* ausgewählt. Neben dieser Datei ist die Checkbox *Program/Configure* anzuklicken.

Mit dem Button *Start* wird der Download gestartet. Das Fenster ist für die Dauer der Programmierung deaktiviert. Es wird wieder aktiv, sobald der Download beendet ist.

Pin-Name des VGA-Moduls	Pad-Name im Block-Diagramm
clk_pin	board_clk
reset_pn	reset
d_hsync	d_hsync
d_vsync	d_vsync
d_column_counter	d_column_counter
d_line_counter	d_line_counter
d_set_column_counter	d_set_column_counter
d_set_line_counter	d_set_line_counter
d_hsync_counter	d_hsync_counter
d_vsync_counter	d_vsync_counter
d_set_hsync_counter	d_set_hsync_counter
d_set_vsync_counter	d_set_vsync_counter
d_h_enable	d_h_enable
d_v_enable	d_v_enable
d_r	d_r
d_g	d_g
d_b	d_b
d_hsync_state	d_hsync_state
d_vsync_state	d_vsync_state
d_state_clk	d_state_clk
seven_seg_pin	seven_seg_out
r0_pin	r0_out
r1_pin	r1_out
r2_pin	r2_out
g0_pin	g0_out
g1_pin	g1_out
g2_pin	g2_out
b0_pin	b0_out
b1_pin	b1_out
hsync_pin	h_sync
vsync_pin	v_sync
d_toggle	d_toggle
d_toggle_counter	d_toggle_counter

Tabelle 5.1: Namen der Pads

LU-SKRIPTUM — Kapitel 6

Remote Learning

Inhalt

6.1 Zugang

Wie eine Remote-Verbindung hergestellt werden kann, findet sich auf der Homepage der Laborübung ([did]). Zur Zeit sind 2 Rechner für den Remotebetrieb über VNC zugänglich. Um die Verbindung sicher zu machen, wird der VNC Port über ssh getunnelt. Im folgenden finden sie eine Anleitung für Windows und Linux, die es ihnen ermöglicht von zu Hause aus zu arbeiten.

Bevor sie sich zum ersten mal über VNC verbinden müssen sie ihr Passwort setzen. Dazu verbinden sie sich mit ssh auf einen der beiden Rechner und ändern ihr Passwort mit vncpasswd.

Vor jedem Verbindungsaufbau ersuchen wir sie in ihrem eigenen Interesse, zuerst die Auslastung der einzelnen Rechner zu überprüfen, damit sie maximale Performance erreichen können.

6.2 Windows

Unter Windows benötigen sie einen ssh-Client der tunneling unterstützt sowie einen VNC-Viewer. Wir beschränken uns hier auf PuTTY als ssh-Client und RealVNC als vncviewer. Bevor sie die VNC-Verbindung tunneln, müssen sie den Server starten.

Dazu starten sie PuTTY und geben zuerst geben sie den gewünschten Rechner ein (siehe Abbildung 6.1).

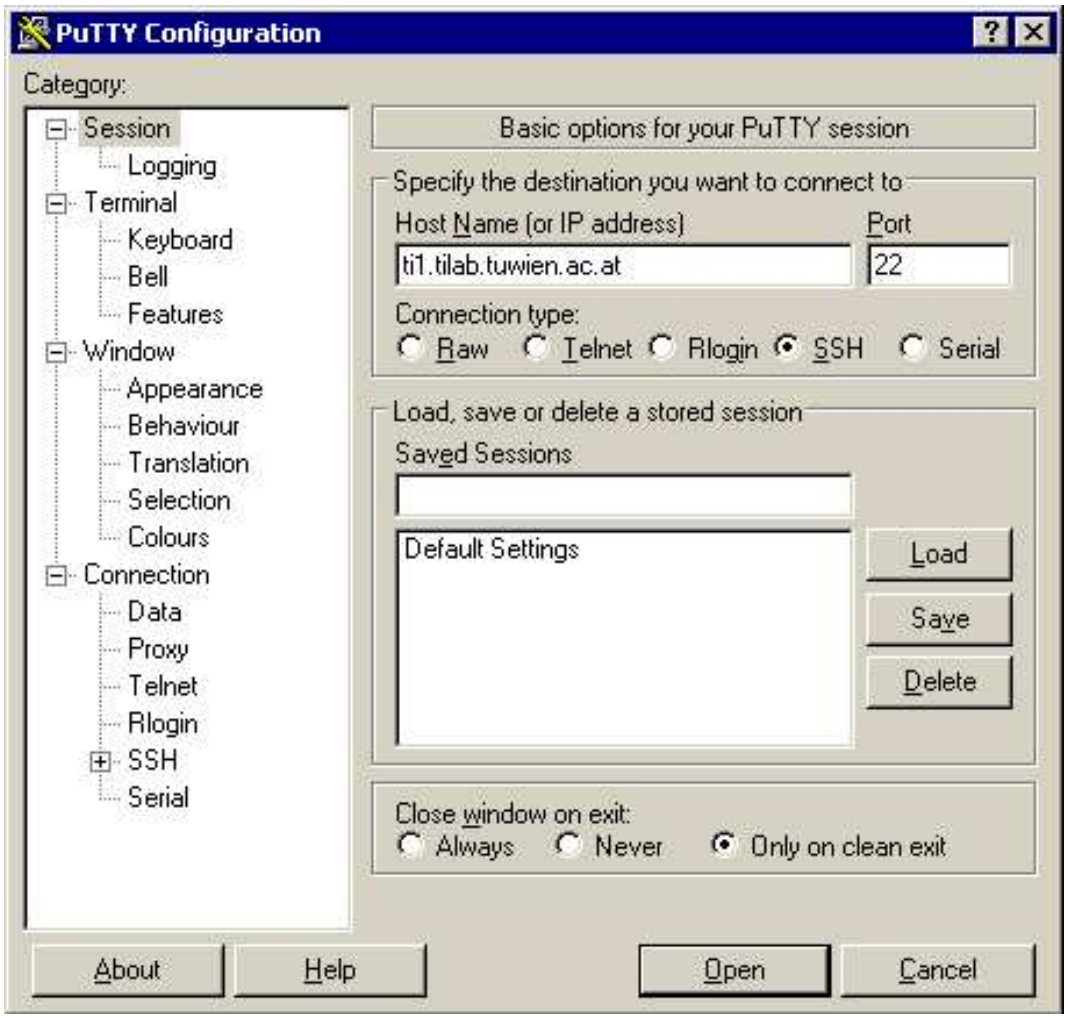


Abbildung 6.1: Starten von PuTTY

Danach tragen sie unter dem Punkt SSH als Remote command **server_start 5901 + Gruppennummer** ein und klicken open (siehe Abbildung 6.2).

Nachdem der Server gestartet wurde, müssen sie PuTTY ein zweitesmal starten. Hier tragen sie wieder unter Host Name den von ihnen gewählten Rechner ein. Danach gehen sie auf den Punkt ssh und aktivieren die Kompression (Enable Compression, siehe Abbildung 6.3).

Zum Abschluss öffnen sie einen Tunnel von ihrem lokalen Port (5901 + Gruppennummer) zum ausgewählten Rechner auf denselben Port (siehe Abbildung 6.4).

Jetzt können sie die Session abspeichern (damit sie die Einstellungen nicht nochmals vornehmen müssen) und klicken danach auf open, starten ihren VNC Client

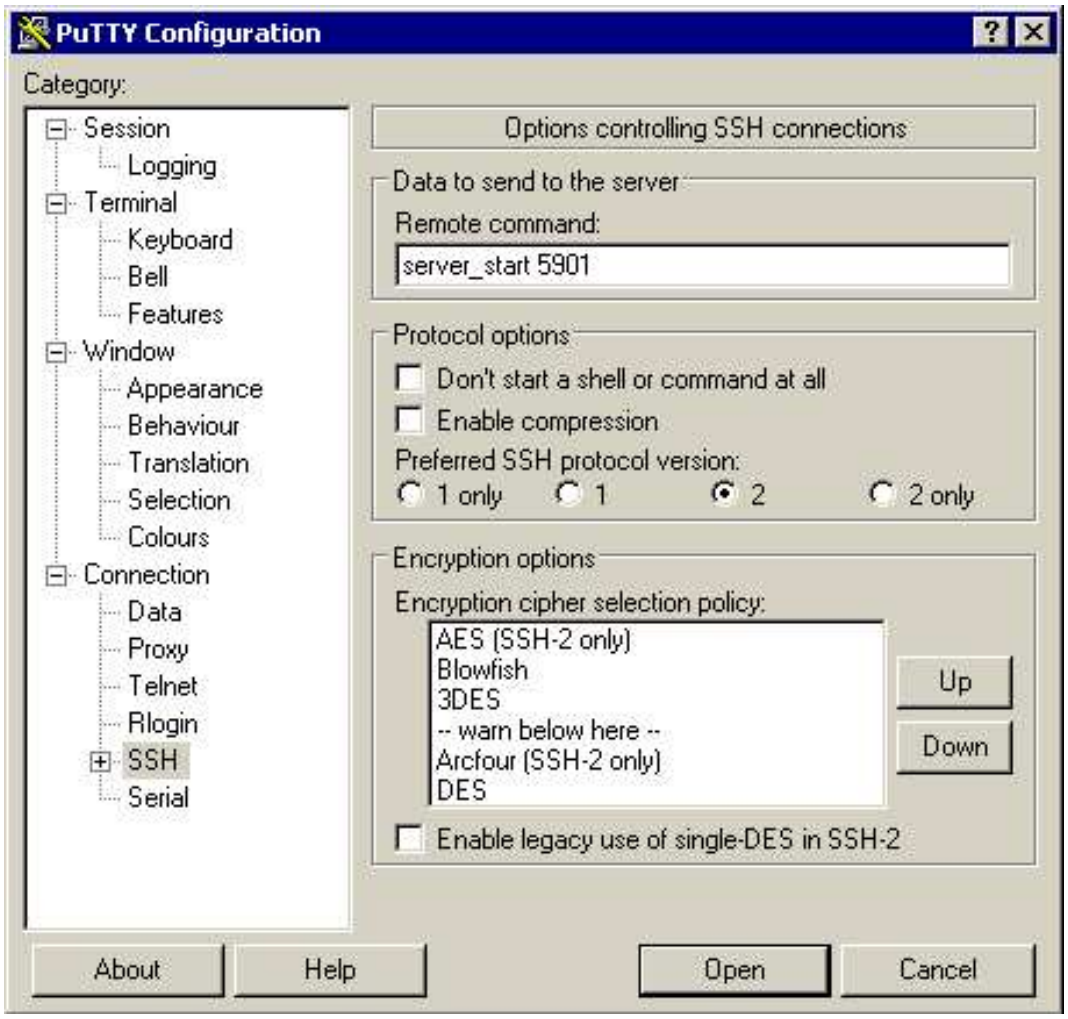


Abbildung 6.2: Starten des VNC-Servers

und geben dort als Server: localhost:5901+Gruppennummer ein.

6.3 Linux

Unter Linux werden lediglich die beiden folgenden Befehle benötigt:

- `ssh -L 59(01+Groupnumber):zielrechner:59(01+Groupnumber) Group@zielrechner \ ‘‘server\_start 59(01+Groupnumber)’’`
- `vncviewer 127.0.0.1:59(01+Groupnumber)`

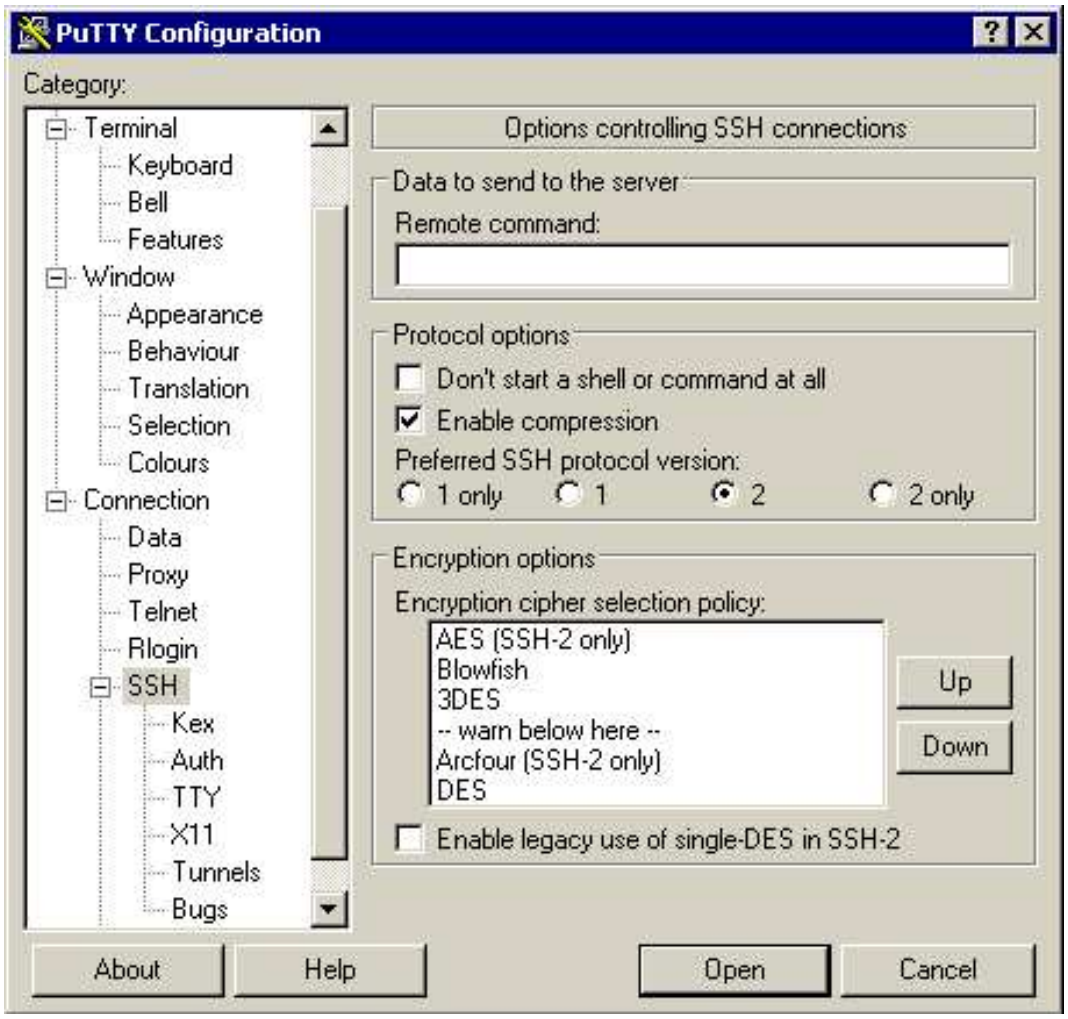


Abbildung 6.3: Aktivieren der Komprimierung

6.4 Der Logikanalysator

Der Logikanalysatorist prinzipiell ein Windows-XP-Rechner; man kann daher bequem mit dem Kommando `rdesktop` eine Verbindung vom Arbeitsplatz zu diesem Gerät herstellen.

Die Logikanalysatoren tragen die Host-Namen *logan0* bis *logan6.* und sind jeweils einem Arbeitsplatz zugeordnet. Diese Zuordnung sieht folgendermaßen aus:

Arbeitsplatz	Logikanalysator

Tabelle 6.1: Zuordnung Arbeitsplatz - Logikanalysator

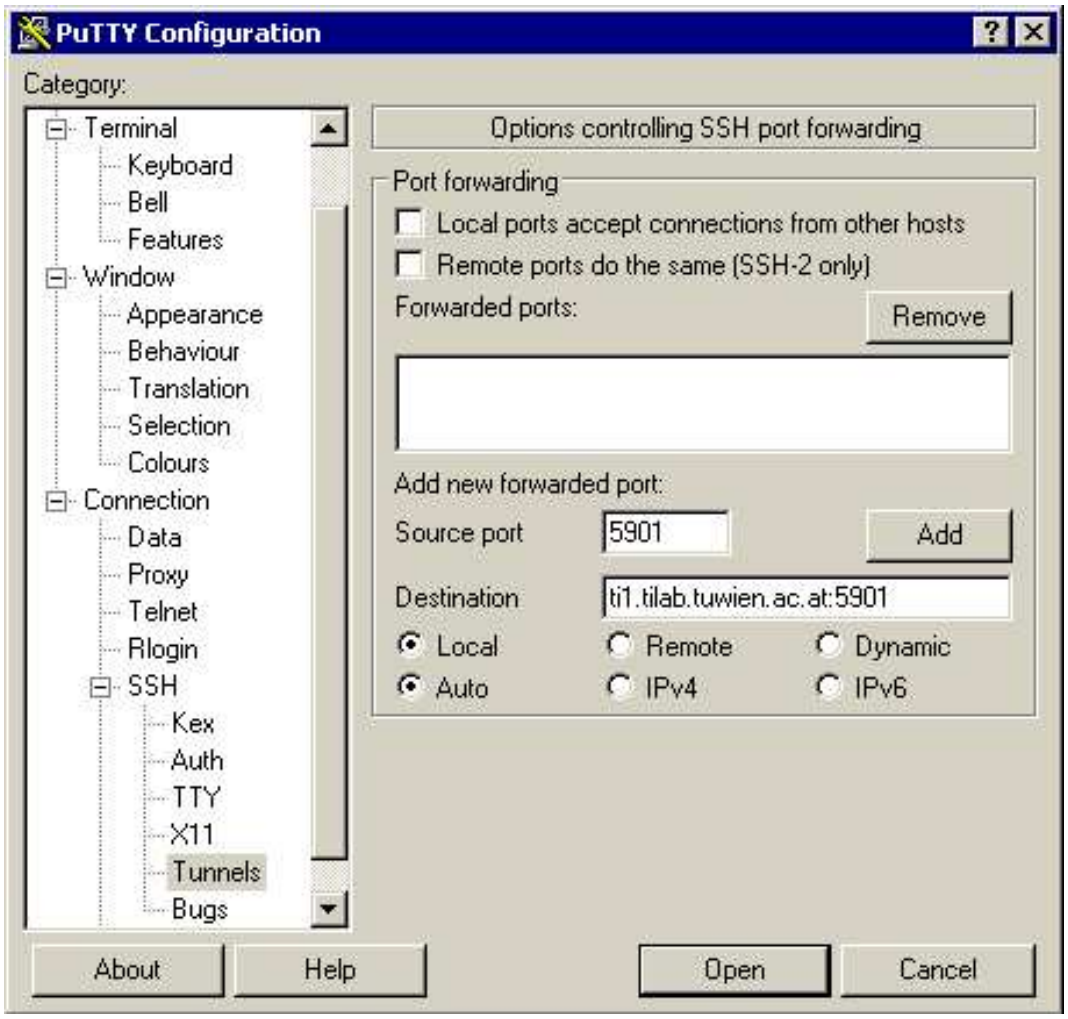


Abbildung 6.4: Öffnen des Tunnels

6.5 Die Kamera

Um die Kamera verwenden zu können, starten Sie das Programm *VRecord*. Die Bildgröße ist standardmäßig auf 320x240 Pixel eingestellt und kann mittels *Optionen* → *Videoformat* geändert werden. Dabei ist zu beachten, daß der Menüeintrag *Optionen* → *Vorschau* aktiviert ist. Farbe und Kontrast können über *Optionen* → *Videoeingenschaften* eingestellt werden. Dieses Fenster ist weitgehend selbsterklärend. Will man etwaige Einstellungen ändern, muß lediglich darauf geachtet werden, daß die Checkbox *Vollautomatisch* deaktiviert ist.

Anhang A

Hinweise zum Testen

Dieses Kapitel beschreibt die empfehlenswerte Vorgehensweise beim Testen und Debuggen des Übungs-Designs und ist besonders für das vierte Beispiel von Bedeutung. Allerdings kann die systematische Fehlersuche auch schon in vorhergehenden Programmierbeispielen hilfreich sein.

Die Art des Fehlers kann entscheiden, welche der im Folgenden angegebenen Schritte anwendbar sind. Es handelt sich also nicht um eine Arbeitsanweisung, welche Punkt für Punkt durchexerziert werden muß, sondern viel mehr um einen Leitfaden, welcher der Situation entsprechend angewandt werden soll.

- Überprüfung aller Anschlüsse, Netzteile, etc.
- Überprüfung, ob der Download erfolgreich war.
- Überprüfung des Schirmbildes (durch Hinsehen):
  - Wird etwas angezeigt?
  - Ist das Bild stabil?
  - Stimmen die Farben?
  - Stimmt gegebenenfalls die Blinkfrequenz (in etwa)?
  - Sind Art und Anzahl der angezeigten Objekte korrekt?
- Aus den Antworten auf die obigen Fragen können erste Schlüsse bezüglich der Fehlerquelle gezogen werden.
- Überprüfung der VGA-Signale (mittels Logikanalysator oder Simulation):
  - Frequenz
  - Tastverhältnis
  - Phasenbeziehung

  - Entsprechen diese Werte der Spezifikation bzw. den in dieser Übung verwendeten Einstellungen?

- Überprüfung der State Machines (mittels Logikanalysator oder Simulation): ist die

  - Abfolge der States
  - Dauer der States
  - Abfolge der Counterwerte

  korrekt?

- Durchforsten des Codes:

  - Überprüfen allfälliger Signalzuordnungen (Namensgebung).
  - Überprüfung auffälliger Werte (Startwerte, Endwerte).

- Überprüfung der Testbenches:

  - Dauer und Polarität des Reset
  - Taktfrequenz
  - Tastverhältnis des Taktes
  - Instanzierung

- Versuchen Sie mit Hilfe der erhaltenen Antworten die Fehlerquelle einzugrenzen oder aber zumindest mögliche Ursachen auszuschließen.

Literaturverzeichnis

Links

[alta] www.altera.com.

[did] http://ti.tuwien.ac.at/ecs/teaching/courses/didelu.

[mod] www.model.com.

[syna] www.synplicity.com.

Referenzen

[Altb] Altera Corporation. *Quartus II Handbook*.

[HU02] J. E. Hopcroft and J. D. Ullman. *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie*. Oldenburg, 2002.

[IEE96] IEEE. *Phase-locked loop techniques. A survey. Guan-Chyun Hsieh and Hung, J.C.*, volume 43 of *Industrial Electronics, IEEE Transactions on*, 1996.

[IEE02] IEEE. *1076 IEEE Standard VHDL Language Reference Manual*, 2002.

[Men07a] Mentor Graphics. *ModelSim SE Tutorial*, 2007.

[Men07b] Mentor Graphics. *ModelSim SE User's Manual*, 2007.

[Smi97] Michael J. S. Smith. *Application-Specific Integrated Circuits*. Addison Wesley, 1997.

[Ste07] A. Steininger. *Digitales Design, Skriptum zur Vorlesung*, Okt. 2007.

[Synb] Synplicity, Inc. *The Synplify Pro Quick Start Guide*.

[Syn05] Synplicity, Inc. *Synplify User Guide and Tutorial*, 2005.

[TS99] U. Tietze and Ch. Schenk. *Halbleiter-Schaltungstechnik*. Springer, 11 edition, 1999.