

# Anleitung zum DE2 Board / NEEK Board

## zum Einsatz im Praktikum Digitale Systeme

von Prof. Dr. U. Sandkühler



# Inhaltsverzeichnis

## **1. Der Aufbau des DE2 Boards**

- 1.1 Die Cyclon II Familie
- 1.2 Taster und Schalter
- 1.3 Rote und Grüne LEDs
- 1.4 7-Segmentanzeigen
- 1.5 Das LCD-Display
- 1.6 Oszillatoren / Taktgeneratoren
- 1.7 Die Expansion Header

## **2. Wichtige Zuordnungen und Einstellungen unter Quartus II**

- 2.1 Device Auswahl
- 2.2 Kapazitive Lasten
- 2.3 Unbenutzte Anschlüsse
- 2.4 Importieren der Pinbelegung (Pin Assignments)
- 2.5 Der Pin Editor
- 2.6 Die Pin Belegung des FPGAs
- 2.7 Die Pin Belegung auf dem DE2-70 Board

## **3. Die Programmierung**

- 3.1 Die JTAG-Schnittstelle
  - 3.1.1 *Der TDI Port*
  - 3.1.2 *Der TDO Port*
  - 3.1.3 *Der TCK Port*
  - 3.1.4 *Der TMS Port*
  - 3.1.5 *Der TRST Port*
- 3.2 Der JTAG Mode
- 3.3 Der AS Mode
- 3.4 Generierung einer POF-Datei

## **4. Dateinamen**

- 4.1 Project Files
- 4.2 Design Files
- 4.3 Ancillary Data Files
- 4.4 Non Editable Ancillary File Types

## **5. Pseudozufallsfolengeneratoren**

## **6. Der PLL-Wizard**

## **7. Das DE2 Control Panel**

- 7.1 Reiter PS2 & 7-SEG
- 7.2 Reiter LED & LCD
- 7.3 Reiter SDRAM / SRAM
- 7.4 Reiter FLASH
- 7.5 Reiter VGA
- 7.6 Reiter TOOLS
- 7.7 Erzeugung von SRAM – Bildern

## **8. Die DE2core Library**

- 8.1 Die Generation unterschiedlicher Taktfrequenzen
  - 8.1.1 *Das VHDL Modul clk\_div*
  - 8.1.2 *Die VHDL Komponente*
- 8.2 Die Sieben-Segment-Ausgabe
  - 8.2.1 *Das VHDL Modul hex7seg*
  - 8.2.2 *Die VHDL Komponente*
- 8.3 AUDIO CODEC
  - 8.3.1 *Das Initialisierungs ROM*
  - 8.3.2 *Das Verilog Programm CLOCK\_500*
  - 8.3.3 *Das VHDL Modul DE2\_I2SOUND*
  - 8.3.4 *Das VHDL Modul AUD\_SP*
  - 8.3.5 *Das VHDL Modul AUD\_PS*
- 8.4 Das LCD Display
  - 8.4.1 *Das VHDL Modul LCD\_Display*
  - 8.4.2 *Das VHDL Modul LCD\_Line*
- 8.5 Einsatz und Arbeitsweise von Character\_ROMs
  - 8.5.1 *Das VHDL Modul CHAR\_ROM*
  - 8.5.2 *Das VHDL Programm*
  - 8.5.3 *Die Struktur der MIF-Datei*
  - 8.5.4 *Das VHDL Modul ASCII\_ROM16*
  - 8.5.5 *Der modifizierte ASCII Code*
  - 8.5.6 *Die Komponente des ASCII\_ROM16 Moduls*
  - 8.5.7 *Spezielle Zeichen Generatoren*
- 8.6 PS2-Tastatur und Keyboard Module
  - 8.6.1 *Die Tastatur*
  - 8.6.2 *Der Scancode*
  - 8.6.3 *Das Übertragungsprotokoll*
  - 8.6.4 *Das VHDL Modul KEYBOARD*
  - 8.6.5 *Das VHDL Modul SCAN2CHAR*

## **8.7 Der Betrieb einer Maus an der PS2 Schnittstelle**

- 8.7.1 Die Betriebsweise der Maus***
- 8.7.2 Das Übertragungsprotokoll***
- 8.7.3 Das VHDL Modul MOUSE***

## **8.8 Der Betrieb von externen VGA Monitoren**

- 8.8.1 Die Synchronisation eines VGA-Monitorsignals***
- 8.8.2 Das VGA\_SYNC Modul im Detail***
- 8.8.3 Monitorsignale in Abhängigkeit von der gewünschten Auflösung***

## **8.9 Externer SRAM Speicher**

- 8.9.1 Die Arbeitsweise des SRAM Speichers***
- 8.9.2 Die VHDL Module SRAM8 und SRAM16***
- 8.9.3 Das VHDL Programm SRAM8***

# **9. Das Touch Panel (TP)**

## **9.1 Die Arbeitsweise des Touch Panels**

- 9.1.1 Die J1 Schnittstelle***
- 9.1.2 Die Kommunikation (LCD)***
- 9.1.3 Die Positionierung (ADC)***

## **9.2 VHDL Module zur grafischen Nutzung des TPs**

- 9.2.1 Das DE2\_TOUCHPANEL Modul***
- 9.2.2 Das TP\_SENSOR Modul***
- 9.2.3 Das TP\_HEXPAD Modul***
- 9.2.4 Das TP\_SYMBOL Modul***
- 9.2.5 Das TP\_RADBUT Modul***
- 9.2.6 Das TP\_DISP\_HEX Modul***
- 9.2.7 Das TP\_DISP\_ASCII Modul***
- 9.2.8 Das POTENTIOMETER Modul***
- 9.2.9 Das TP\_BAR Modul***

## **9.3 Die Zusammenarbeit der Module im Programm TP DESIGN**

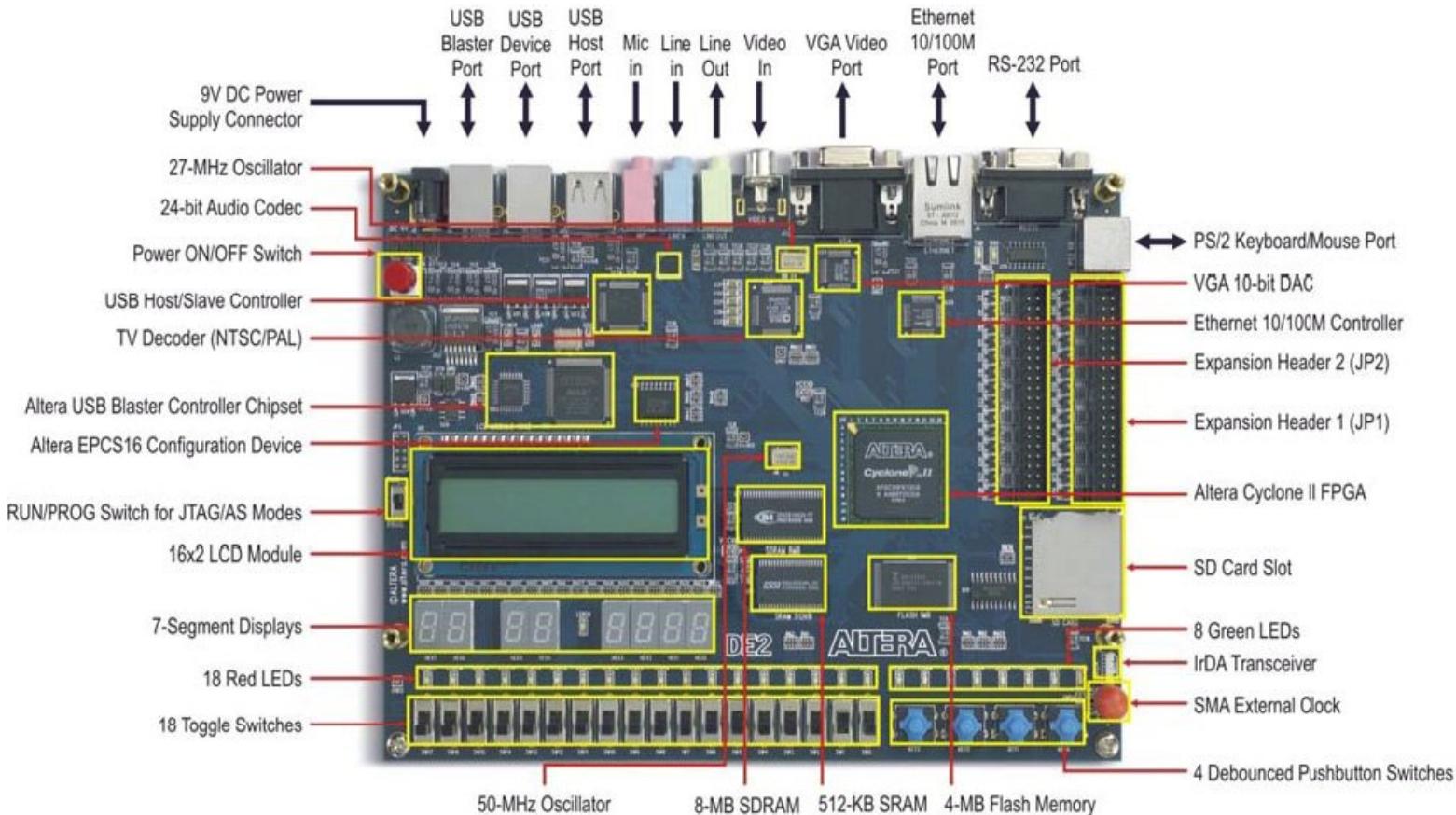
# **10. Das NEEK Entwicklungsboard**

- 10.1 Aufbau und Komponenten**
- 10.2 Das NEEK Board als Ersatz für das DE2 Entwicklungsboard**
- 10.3 Das Modul NEEK TOUCHPANEL**
- 10.4 Die Pin Belegung des NEEK Boards**
- 10.5 Starten eines Projektes von der SD-Karte**

# **11. Literatur**

Im Praktikum Digitale Systeme wird das DE2 Board von Altera mit einem programmierbaren Cyclon II FPGA eingesetzt. Zusätzlichen I/O Komponenten ermöglichen eine komfortabel Ein- und Ausgabe unterschiedlicher Daten und Signale. Entwicklungsboard, Komponenten und fertige VHDL Module sollen im Folgenden vorgestellt werden.

## 1. Der Aufbau des DE2 Boards

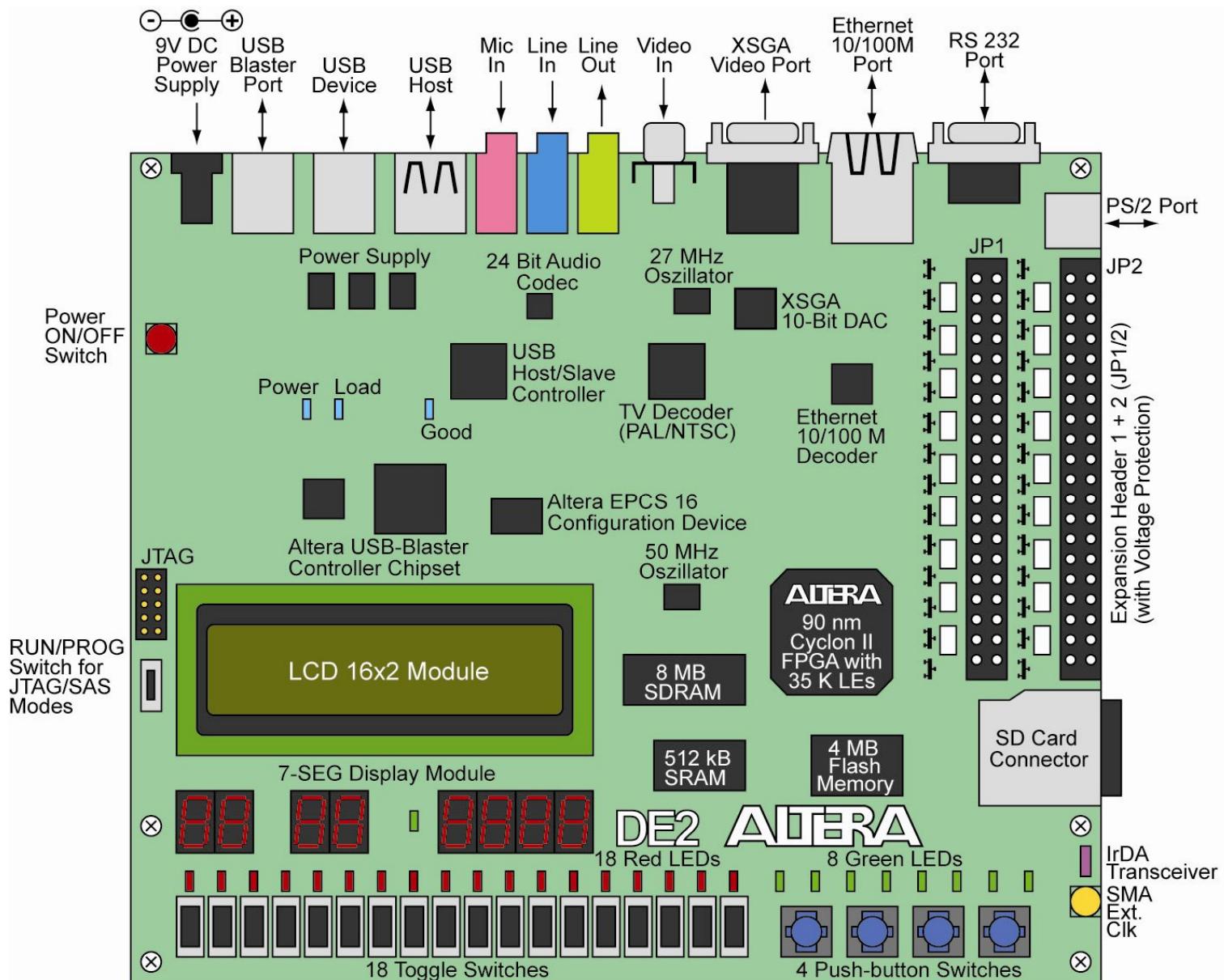


Das oben stehende Bild zeigt das Entwicklungsboard mit allen Bausteinen und Anschlüssen. Eine strukturierte Übersicht bietet die weiter unten gezeigte Grafik. Die wichtigsten und am häufigsten benutzten Hardware-Komponenten werden im Folgenden vorgestellt und beschrieben. Da sie fest mit den Anschlüssen des FPGAs verbunden sind, ist unbedingt auf die richtige Pinbelegung zu achten. Sie wird in Kapitel 2.6 beschrieben.

### 1.1 Die Cyclon II Familie

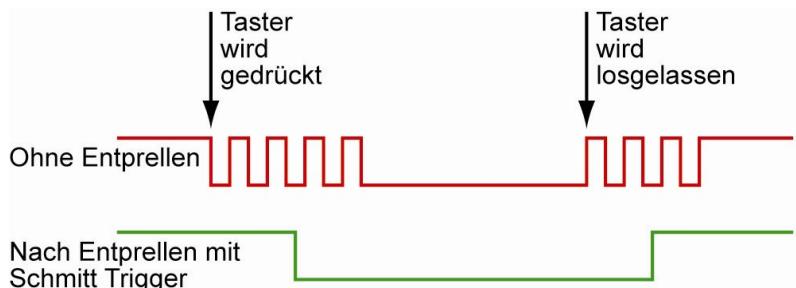
Den Kern des DE2 Boards bildet der programmierbare FPGA Baustein EPC2C35F672C6N. Das Kürzel EPC2C bezeichnet dabei die Bausteinfamilie. Die ungefähre Komplexität in 1000 Logik Elemente wird durch 35 gekennzeichnet. Gehäuse Form und Anschlusszahl spiegelt sich in F672 wieder. Die Geschwindigkeitsklasse 6 ist eine der schnelleren. Weiter Angaben zur Komplexität und Leistungsfähigkeit des hier benutzten Bausteins sowie aller weiteren Vertreter der Cyclon II Familie können der folgenden Tabelle entnommen werden. Das neue weiter entwickelte DE2-70 Board arbeitet mit dem EP2C70 FPGA.

Eigenschaften der Cyclon II Familie von ALTERA							
	EP2C5	EP2C8	EP2C15	EP2C20	EP2C35	EP2C50	EP2C70
<b>Logic Elements (LEs)</b>	4.608	8.256	14.448	18.752	<b>33.216</b>	50.528	68.416
<b>Logic Array locks (LABs)</b>	288	516	903	1172	<b>2076</b>	3158	4276
<b>LAB Zeilen</b>	13	18	26	26	<b>35</b>	43	50
<b>LAB Spalten</b>	24	30	46	46	<b>60</b>	74	86
<b>M4K RAM blocks (4kBits + 512 ParityBits)</b>	26	36	52	52	<b>105</b>	129	250
<b>Total RAM kBits</b>	119,8	165,9	239,6	239,6	<b>483,8</b>	594,4	1.152
<b>Embedded Multipliers</b>	13	18	26	26	<b>35</b>	86	150
<b>Global Clock Networks</b>	8	8	16	16	<b>16</b>	16	16
<b>PLLs</b>	2	2	4	4	<b>4</b>	4	4
<b>Max. user I/O pins</b>	158	182	315	315	<b>475</b>	450	622



## **1.2 Taster und Schalter**

Alle 4 Taster KEY0 – KEY3 (De-bounced Pushbutton Switches) arbeiten **active low** (d.h. im unbetätigten Zustand liefern sie ein logisch 1 Signal) und werden hardware-mäßig über eine Schmitt Trigger Schaltung entprellt (debounced). Der Einsatz eines zusätzlichen „Debounce-Baustein“ in VHDL ist deshalb nicht notwendig!



Um sie unter Quartus II anzusprechen und die Pin-Vergabe über eine Importdatei (\*.csv, siehe Kapitel 2.4) möglichst einfach zu gestalten, sollten diese in VHDL bzw. im Blockfile Editor immer als **KEY[4 downto 0]** bzw. **KEY[0]** etc. definiert werden. Dadurch wird sichergestellt, dass die Anschlüsse „automatisch“ richtig verdrahtet werden. Auf die richtige Namensgebung sollte auch bei allen unten aufgeführten Ein- / Ausgabekomponenten geachtet werden. Andernfalls ist eine manuelle Verdrahtung notwendig.

Alle 18 Schalter SW0 – SW17 (Toggle Switches) liefern in der unteren Stellung logisch 0 und in der oberen Stellung logisch 1. Sie sollten in VHDL oder im Block Design mit **SW[17 downto 0]** bzw. **SW[0]** etc. bezeichnet werden.

## **1.3 Rote und Grüne LEDs**

Alle 18 Red-LEDs und 9 Green-LEDs arbeiten **active high**, d.h., sie leuchten bei einer Ansteuerung mit logisch 1.

Sie sollten in VHDL oder im Block Design mit **LEDR[17 downto 0]** bzw. **LEDG[8 downto 0]** bezeichnet werden bzw. entsprechend der benötigten Anzahl.

## **1.4 7-Segmentanzeigen**

Die acht 7-Segmentanzeigen HEX0 bis HEX7 arbeiten **active low**, d.h., sie leuchten bei einer Ansteuerung mit logisch 0.

Sie sollten in VHDL mit **HEX0[0 to 6]** bis **HEX6[0 to 6]** angesprochen werden. Für Ihre Ansteuerung kann die VHDL-Komponente / der Baustein **hex7seg** eingesetzt werden, der einen Hexadezimalwert in die passende Ansteuerung der Anzeige umsetzt. Diesen und andere Bausteine finden Sie in der **DE2core Bibliothek**. Achten Sie dabei auf die Indizierungsrichtung (**to** statt **downto**). Die Ansteuerung der Dezimalpunkte ist hier nicht möglich im Gegensatz zum weiterentwickelten DE2-70 Board.

## **1.5 Das LCD-Display**

Zur Ausgabe von bis zu 2 x 16 ASCII Zeichen oder einfachen grafischen Darstellungen steht ein zweizeiliges LCD-Display zur Verfügung. Die Ansteuerung dieses Displays erfordert eine spezielle Logik (Controller), die im VHDL Programm **LCD\_Display.vhd** realisiert wird. Es kann individuellen Bedürfnissen angepasst werden. Das Programm sollte jeweils so editiert werden, dass alle 32 HEX / ASCII Zeichen verteilt auf zwei Zeilen richtig angesteuert werden. Ein Beispiel findet sich im Modul **LCD\_Display.vhd** in Kapitel 8.4. D.h. auch nicht benötigte Stellen müssen zumindest als Leerzeichen dargestellt werden.

## 1.6 Oszillatoren / Taktgeneratoren

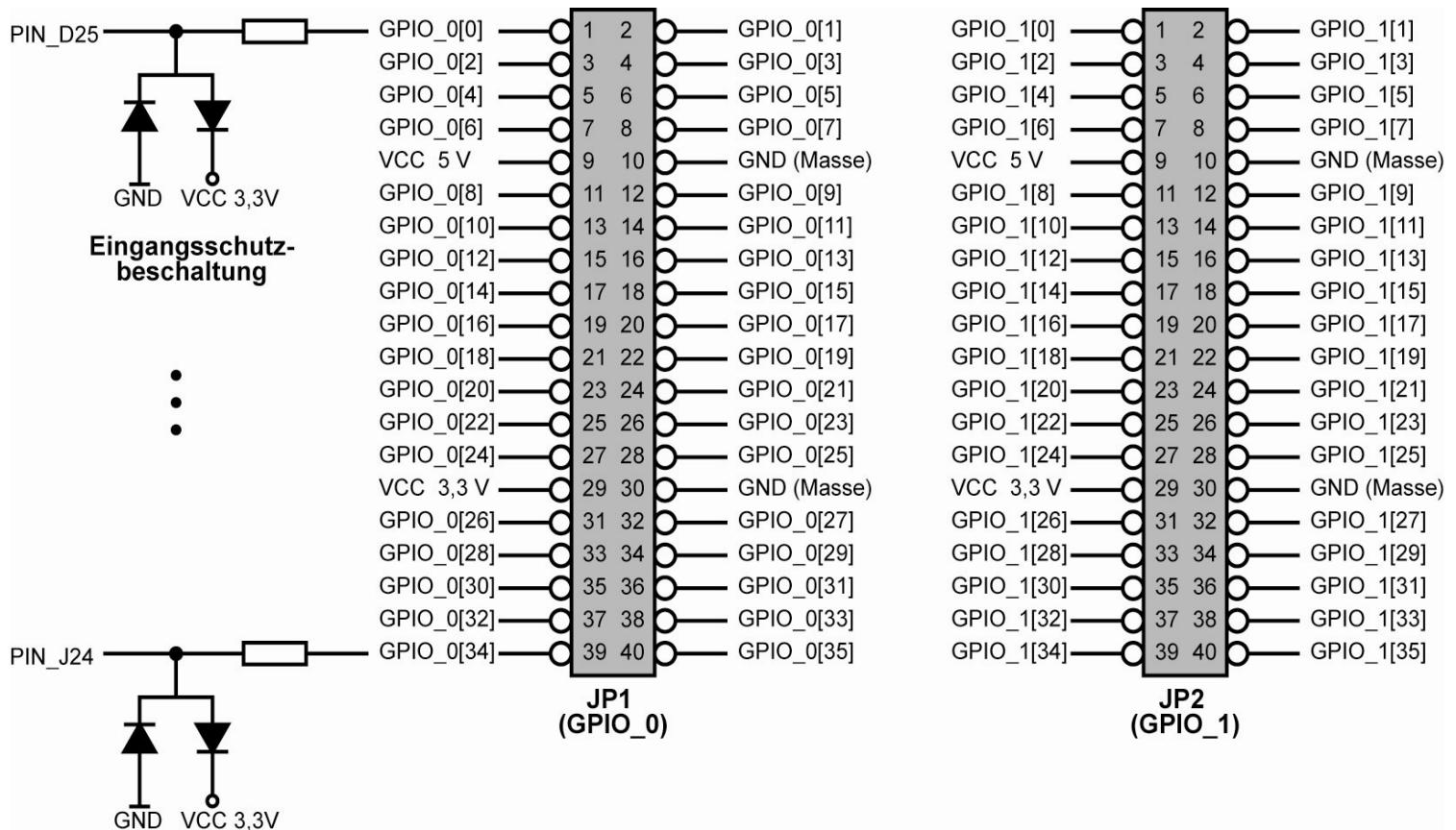
### 1.6 Oszillatoren / Taktgeneratoren

Das DE2-Board enthält 2 quarzstabilisierte Oszillatoren mit 27 MHz und 50 MHz Taktfrequenz. Sie können benutzt werden, um den benötigten Takt einer Schaltung zu generieren. Für die richtige Verdrahtung sollten sie dementsprechend als `CLOCK_27` bzw. `CLOCK_50` bezeichnet werden.

Die Komponente `clk_div` aus der DE2\_core Bibliothek kann dabei sehr einfach benutzt werden, um daraus kleinere Taktraten zu erzeugen wie z.B. 1 MHz ... 1 Hz bei Anschluss des 50 MHz Taktes (siehe Kapitel 8.1).

## 1.7 Die Expansion Header

Ein direkter Zugang zu den Anschlüsse des Cyclon II Bausteins besteht über zwei 40-polige Steckerleisten (Expansion Header im IDE-Anschlussformat). Über die Anschlussbezeichnungen `GPIO_0[35 DOWNTO 0]` und `GPIO_1[35 DOWNTO 0]` stehen 2 x 36 Anschlüsse zur Verfügung. Jeder der Eingänge ist mit einer Schutzschaltung aus einem Widerstand und antiparalle geschalteten Dioden zur Ableitung von Überspannungen versehen. Die verbleibenden Anschlüsse liefern Spannungen von 5 bzw. 3,3 Volt, oder sie liegen an Masse Potential.



## 2. Wichtige Zuordnungen und Einstellungen unter Quartus II

### 2.1 Device Auswahl

Auf dem DE2 Board wird der Cyclon II Baustein **EP2C35F672C6** eingesetzt. Er ist unter **Assignments → Devices** auszuwählen, wenn dieses nicht schon bei der Definition des Projektes erfolgt ist.

### 2.2 Kapazitive Lasten

Zur Vermeidung des oder der Warnings des Quartus II Compilers:

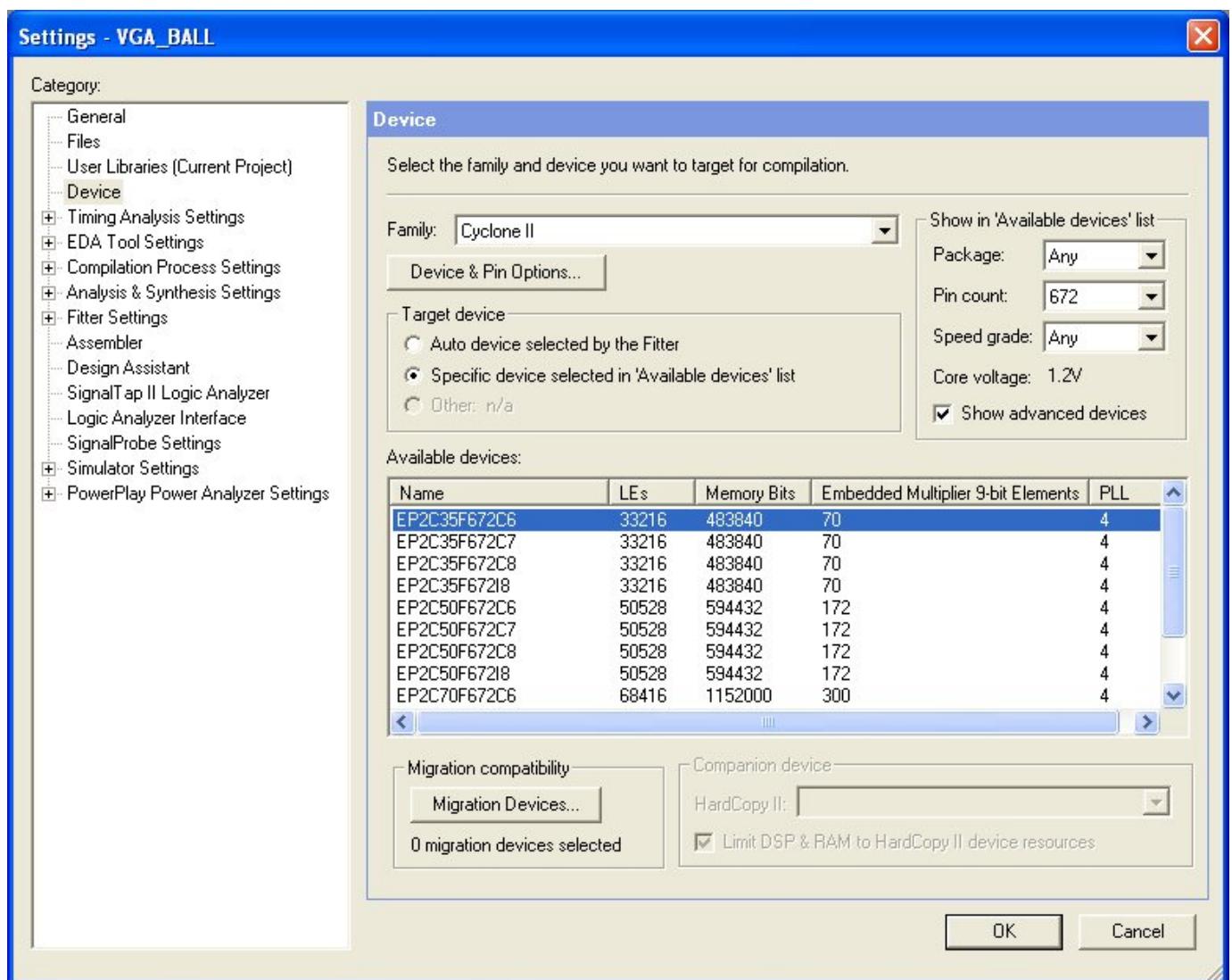
**Warning:** Found 26 output pins without output pin load capacitance assignment

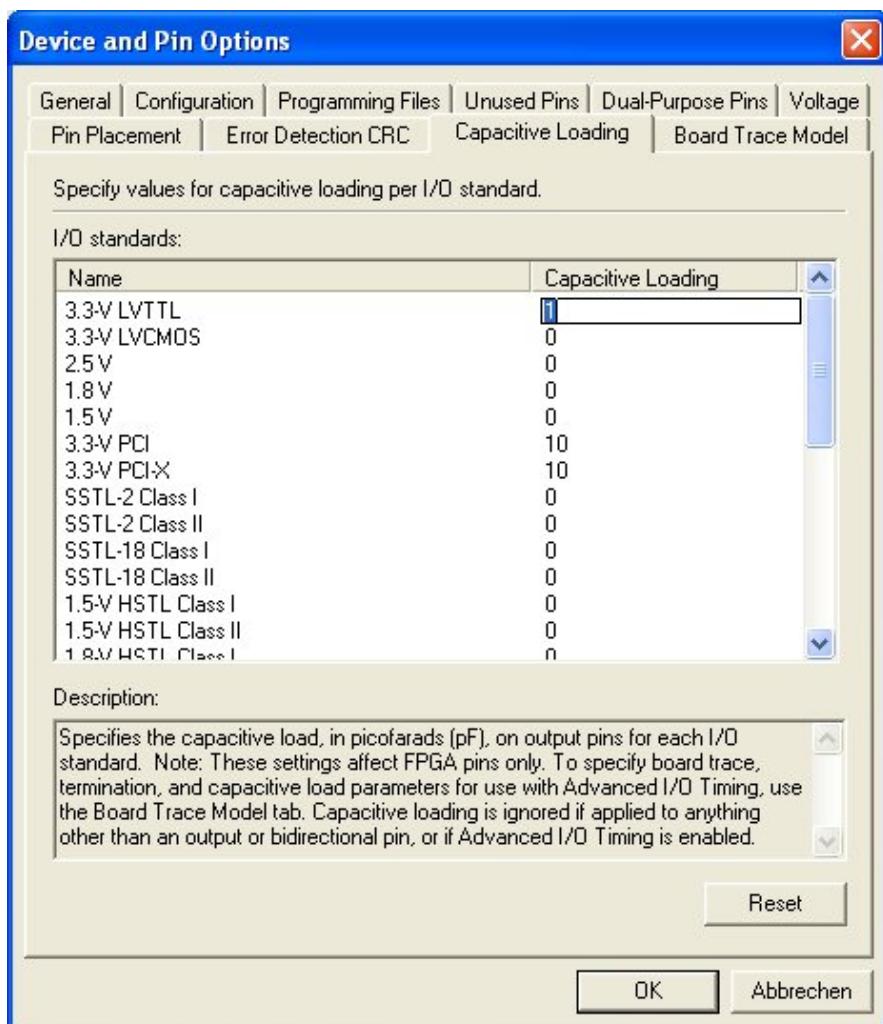
**Info:** Pin "xyz" has no specified output pin load capacitance – assuming default load capacitance of 0 pF for timing analysis.

ist die unten gezeigte Einstellung vorzunehmen:

**Assignments → Devices →**

**Devices and Pin Options / Capacitive Loading: 3.3-V LVTTL = 1**



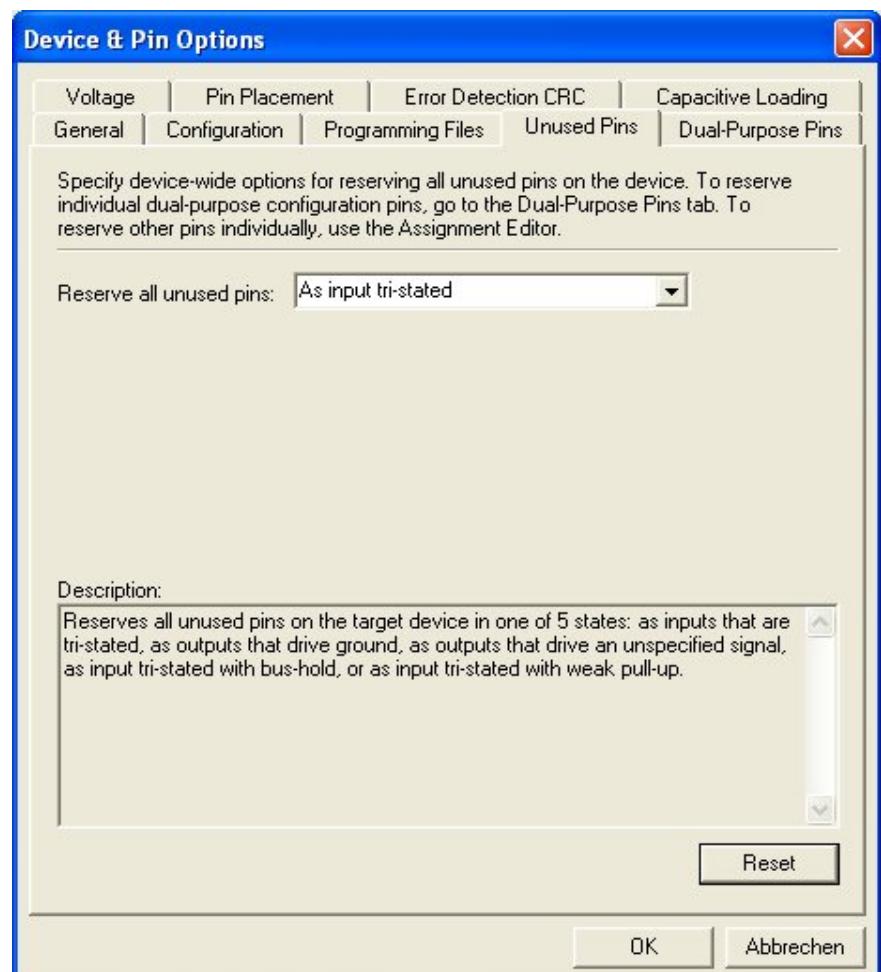


## 2.3 Unbenutzte Anschlüsse

### **WICHTIG !**

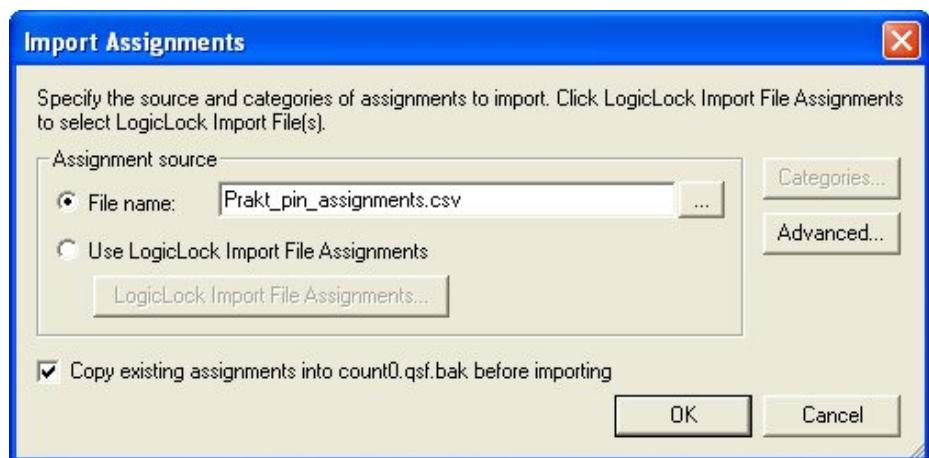
Damit keine Bausteine auf dem Board durch hohe Ströme aus unbenutzten Pins beschädigt werden und um die Leistungsaufnahme des Cyclon II Bausteins zu reduzieren, sollte **unbedingt** auf die folgende Einstellung für unbenutzte Anschlüsse (Unused Pins) geachtet werden (siehe rechts):

**Assignments → Devices → Devices and Pin Options / Unused Pins: As input tri-stated**



## **2.4 Importieren der Pinbelegung (Pin Assignments)**

Nach dem erfolgreichen Compilieren eines Designs müssen alle benutzten, externen Komponenten auf dem DE2 Board mit den entsprechenden Anschlüssen (Pins) des Cyclon II Bausteins verbunden werden. Dies kann durch den Import der Excel-kompatiblen Datei **Prakt\_pin\_assignments.csv** aus der DE2core Bibliothek erfolgen, ohne für jedes Design eine manuelle „Verdrahtung“ vornehmen zu müssen.



Diese Datei ist eine verkürzte Version der kompletten Anschlussbelegung ohne Speicheranschlüsse und GPIOs, im Gegensatz zur Datei **DE2\_pin\_assignments.csv**, die alle Anschlüsse des DE2 Boards enthält. Letztere wird in Kapitel 2.7 vorgestellt. Diese Dateien sind auch unter Excel editierbar. Ebenso kann auf die Anschlussbelegungen aus vergleichbaren Projekten zurückgegriffen werden, indem die dort erstellte Datei \*.qsf in das neue Projekt importiert wird. Die bisherige (alte) Pinbelegung des neuen Projektes wird dabei in einer Datei \*.qsf.bak abgelegt.

Innerhalb eines Quartus II Projektes werden alle Pin-Belegungen und Zuordnungen (Assignments) in \*.qsf Files abgelegt. Diese Files können importiert und exportiert werden.

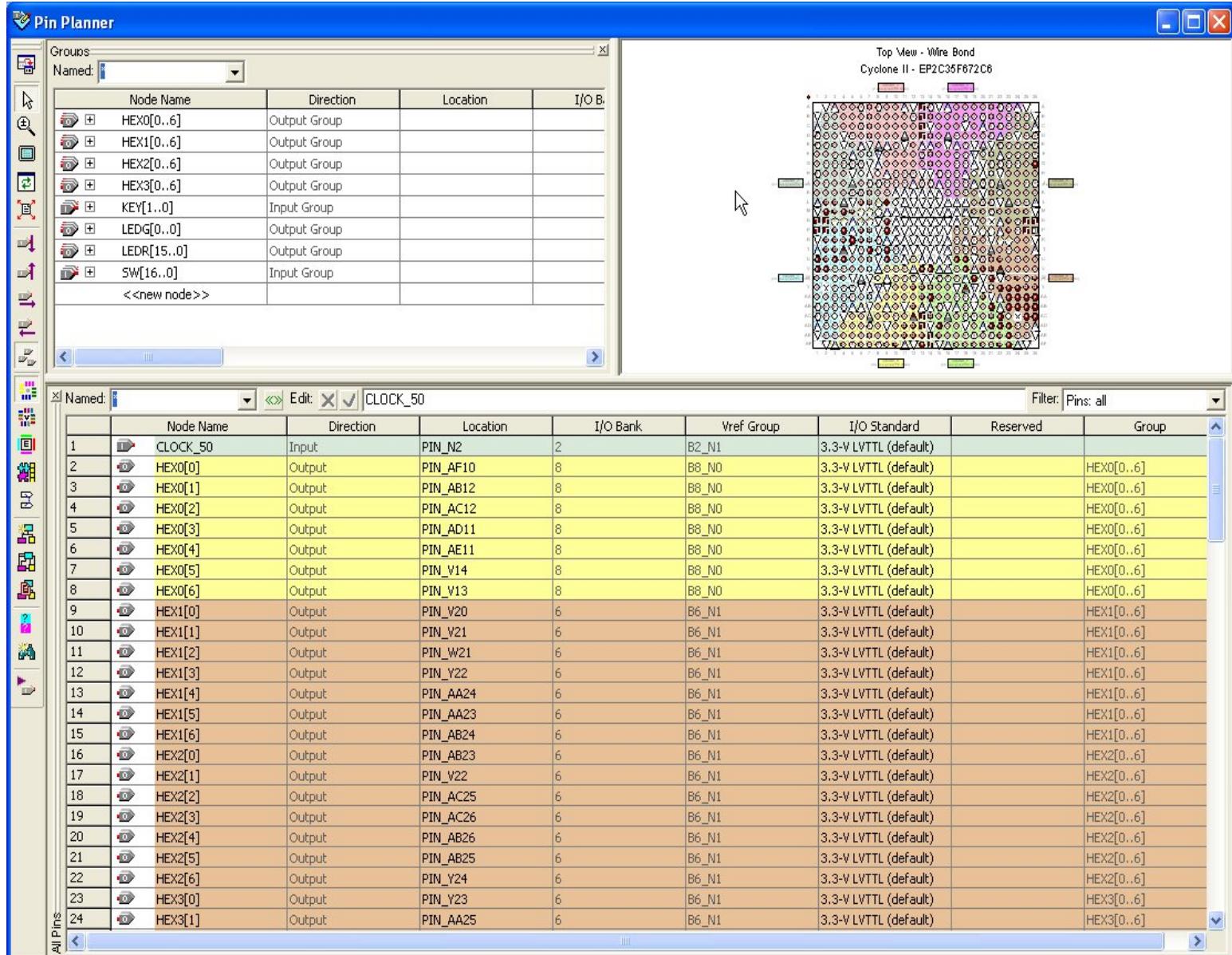
qsf-Dateien enthalten die Pinbelegungen eines Projekts, sowie weitere Einstellungen (Settings). Alle nicht benötigten Anschlüsse sollten, wie schon oben gesagt, danach im Pin-Editor manuell gelöscht werden, um unnötige Warnings zu vermeiden. Der Import wird durch den Aufruf **Assignments → Import Assignments ... ermöglicht.**

## **2.5 Der Pin Editor**

Beim Aufruf des Pin Editors (**Assignments → Pins**) werden alle importierten Anschlüsse detailliert aufgelistet. Betrachten Sie die grafische Darstellung der Pinbelegung für das Ball Grid Array, die beim Aufruf des Pin Editors für jeden programmierbaren Baustein dargestellt wird! Es werden die folgenden Parameter tabellarisch aufgelistet:

- Der vom Anwender definierte Port Name (Node Name),
- die Signalrichtung des Ports (Direction),
- die genaue Pin Bezeichnung (Location),
- die Gruppierung des Anschlusses innerhalb des FPGAs (I/O Bank, VREF Group),
- der eingestellte Spannungsspeigel am Ausgang (I/O Standard),
- der maximal zur Verfügung stehende Ausgangsstrom (Current Strength).

Diese Werte lassen sich gegebenenfalls in Quartus II modifizieren.



## 2.6 Die Pin Belegung des FPGAs

Die Pinbelegung für alle Anschlüsse des Cyclon Bausteins EP2C35F672C, wie sie in der Datei **DE2\_pin\_assignments.csv** von Altera für das DE2 Board vorgegeben werden, zeigt das folgende Listing. Die Datei kann in der DE2core Library gefunden werden; genauso wie die Datei **Prakt\_pin\_assignment.csv**, die sich auf solche Anschlüsse beschränkt, die im Praktikum ganz oder teilweise benutzt werden. Sie bildet quasi eine Teilmenge der ersten Datei und ist deshalb etwas übersichtlicher. Nach dem Importieren der Anschlüsse, indem man eine der beiden Dateien aufruft (siehe Abschnitt 2.4), sollten die nicht benutzten Anschlüsse im PIN-Editor gelöscht werden, um die Anzahl der gemeldeten Warnungen zu reduzieren. Ein Listing aller Anschlüsse (Pinbelegung) des Cyclon II FPGAs auf dem DE2 Board ist auf den folgenden Seiten aufgeführt.

```

# Quartus II Version 5.1 Internal Build 160 09/19/2005 TO Full Version,
# File: D:\de2_pins\de2_pins.csv,
# Generated on: Wed Sep 28 09:40:34 2005,

# Note: The column header names should not be changed if you wish to import
this .csv file into the Quartus II software.,
To, Location

SW[0],PIN_N25 | FL_ADDR[7],PIN_AA16 | HEX5[1],PIN_P6 | LCD_DATA[2],PIN_H1
SW[1],PIN_N26 | FL_ADDR[8],PIN_AD17 | HEX5[2],PIN_P7 | LCD_DATA[3],PIN_H2
SW[2],PIN_P25 | FL_ADDR[9],PIN_AC17 | HEX5[3],PIN_T9 | LCD_DATA[4],PIN_J4
SW[3],PIN_AE14 | FL_ADDR[10],PIN_AE17 | HEX5[4],PIN_R5 | LCD_DATA[5],PIN_J3
SW[4],PIN_AF14 | FL_ADDR[11],PIN_AF17 | HEX5[5],PIN_R4 | LCD_DATA[6],PIN_H4
SW[5],PIN_AD13 | FL_ADDR[12],PIN_W16 | HEX5[6],PIN_R3 | LCD_DATA[7],PIN_H3
SW[6],PIN_AC13 | FL_ADDR[13],PIN_W15 | HEX6[0],PIN_R2 | LCD_ON,PIN_L4
SW[7],PIN_C13 | FL_ADDR[14],PIN_AC16 | HEX6[1],PIN_P4 | LCD_BLON,PIN_K2
SW[8],PIN_B13 | FL_ADDR[15],PIN_AD16 | HEX6[2],PIN_P3 | SRAM_ADDR[0],PIN_AE4
SW[9],PIN_A13 | FL_ADDR[16],PIN_AE16 | HEX6[3],PIN_M2 | SRAM_ADDR[1],PIN_AF4
SW[10],PIN_N1 | FL_ADDR[17],PIN_AC15 | HEX6[4],PIN_M3 | SRAM_ADDR[2],PIN_AC5
SW[11],PIN_P1 | FL_ADDR[18],PIN_AB15 | HEX6[5],PIN_M5 | SRAM_ADDR[3],PIN_AC6
SW[12],PIN_P2 | FL_ADDR[19],PIN_AA15 | HEX6[6],PIN_M4 | SRAM_ADDR[4],PIN_AD4
SW[13],PIN_T7 | FL_ADDR[20],PIN_Y15 | HEX7[0],PIN_L3 | SRAM_ADDR[5],PIN_AD5
SW[14],PIN_U3 | FL_ADDR[21],PIN_Y14 | HEX7[1],PIN_L2 | SRAM_ADDR[6],PIN_AE5
SW[15],PIN_U4 | FL_CE_N,PIN_V17 | HEX7[2],PIN_L9 | SRAM_ADDR[7],PIN_AF5
SW[16],PIN_V1 | FL_OE_N,PIN_W17 | HEX7[3],PIN_L6 | SRAM_ADDR[8],PIN_AD6
SW[17],PIN_V2 | FL_DQ[0],PIN_AD19 | HEX7[4],PIN_L7 | SRAM_ADDR[9],PIN_AD7
DRAM_ADDR[0],PIN_T6 | FL_DQ[1],PIN_AC19 | HEX7[5],PIN_P9 | SRAM_ADDR[10],PIN_V10
DRAM_ADDR[1],PIN_V4 | FL_DQ[2],PIN_AF20 | HEX7[6],PIN_N9 | SRAM_ADDR[11],PIN_V9
DRAM_ADDR[2],PIN_V3 | FL_DQ[3],PIN_AE20 | KEY[0],PIN_G26 | SRAM_ADDR[12],PIN_AC7
DRAM_ADDR[3],PIN_W2 | FL_DQ[4],PIN_AB20 | KEY[1],PIN_N23 | SRAM_ADDR[13],PIN_W8
DRAM_ADDR[4],PIN_W1 | FL_DQ[5],PIN_AC20 | KEY[2],PIN_P23 | SRAM_ADDR[14],PIN_W10
DRAM_ADDR[5],PIN_U6 | FL_DQ[6],PIN_AF21 | KEY[3],PIN_W26 | SRAM_ADDR[15],PIN_Y10
DRAM_ADDR[6],PIN_U7 | FL_DQ[7],PIN_AE21 | LEDR[0],PIN_AE23 | SRAM_ADDR[16],PIN_AB8
DRAM_ADDR[7],PIN_U5 | FL_RST_N,PIN_AA18 | LEDR[1],PIN_AF23 | SRAM_ADDR[17],PIN_AC8
DRAM_ADDR[8],PIN_W4 | FL_WE_N,PIN_AA17 | LEDR[2],PIN_AB21 | SRAM_DQ[0],PIN_AD8
DRAM_ADDR[9],PIN_W3 | HEX0[0],PIN_AF10 | LEDR[3],PIN_AC22 | SRAM_DQ[1],PIN_AE6
DRAM_ADDR[10],PIN_Y1 | HEX0[1],PIN_AB12 | LEDR[4],PIN_AD22 | SRAM_DQ[2],PIN_AF6
DRAM_ADDR[11],PIN_V5 | HEX0[2],PIN_AC12 | LEDR[5],PIN_AD23 | SRAM_DQ[3],PIN_AA9
DRAM_BA_0,PIN_AE2 | HEX0[3],PIN_AD11 | LEDR[6],PIN_AD21 | SRAM_DQ[4],PIN_AA10
DRAM_BA_1,PIN_AE3 | HEX0[4],PIN_AE11 | LEDR[7],PIN_AC21 | SRAM_DQ[5],PIN_AB10
DRAM_CAS_N,PIN_AB3 | HEX0[5],PIN_V14 | LEDR[8],PIN_AA14 | SRAM_DQ[6],PIN_AA11
DRAM_CKE,PIN_AA6 | HEX0[6],PIN_V13 | LEDR[9],PIN_Y13 | SRAM_DQ[7],PIN_Y11
DRAM_CLK,PIN_AA7 | HEX1[0],PIN_V20 | LEDR[10],PIN_AA13 | SRAM_DQ[8],PIN_AE7
DRAM_CS_N,PIN_AC3 | HEX1[1],PIN_V21 | LEDR[11],PIN_AC14 | SRAM_DQ[9],PIN_AF7
DRAM_DQ[0],PIN_V6 | HEX1[2],PIN_W21 | LEDR[12],PIN_AD15 | SRAM_DQ[10],PIN_AE8
DRAM_DQ[1],PIN_AA2 | HEX1[3],PIN_Y22 | LEDR[13],PIN_AE15 | SRAM_DQ[11],PIN_AF8
DRAM_DQ[2],PIN_AA1 | HEX1[4],PIN_AA24 | LEDR[14],PIN_AF13 | SRAM_DQ[12],PIN_W11
DRAM_DQ[3],PIN_Y3 | HEX1[5],PIN_AA23 | LEDR[15],PIN_AE13 | SRAM_DQ[13],PIN_W12
DRAM_DQ[4],PIN_Y4 | HEX1[6],PIN_AB24 | LEDR[16],PIN_AE12 | SRAM_DQ[14],PIN_AC9
DRAM_DQ[5],PIN_R8 | HEX2[0],PIN_AB23 | LEDR[17],PIN_AD12 | SRAM_DQ[15],PIN_AC10
DRAM_DQ[6],PIN_T8 | HEX2[1],PIN_V22 | LEDG[0],PIN_AE22 | SRAM_WE_N,PIN_AE10
DRAM_DQ[7],PIN_V7 | HEX2[2],PIN_AC25 | LEDG[1],PIN_AF22 | SRAM_OE_N,PIN_AD10
DRAM_DQ[8],PIN_W6 | HEX2[3],PIN_AC26 | LEDG[2],PIN_W19 | SRAM_UB_N,PIN_AF9
DRAM_DQ[9],PIN_AB2 | HEX2[4],PIN_AB26 | LEDG[3],PIN_V18 | SRAM_LB_N,PIN_AE9
DRAM_DQ[10],PIN_AB1 | HEX2[5],PIN_AB25 | LEDG[4],PIN_U18 | SRAM_CE_N,PIN_AC11
DRAM_DQ[11],PIN_AA4 | HEX2[6],PIN_Y24 | LEDG[5],PIN_U17 | OTG_ADDR[0],PIN_K7
DRAM_DQ[12],PIN_AA3 | HEX3[0],PIN_Y23 | LEDG[6],PIN_AA20 | OTG_ADDR[1],PIN_F2
DRAM_DQ[13],PIN_AC2 | HEX3[1],PIN_AA25 | LEDG[7],PIN_Y18 | OTG_CS_N,PIN_F1
DRAM_DQ[14],PIN_AC1 | HEX3[2],PIN_AA26 | LEDG[8],PIN_Y12 | OTG_RD_N,PIN_G2
DRAM_DQ[15],PIN_AA5 | HEX3[3],PIN_Y26 | CLOCK_27,PIN_D13 | OTG_WR_N,PIN_G1
DRAM_LDQM,PIN_AD2 | HEX3[4],PIN_Y25 | CLOCK_50,PIN_N2 | OTG_RST_N,PIN_G5
DRAM_UDQM,PIN_Y5 | HEX3[5],PIN_U22 | EXT_CLOCK,PIN_P26 | OTG_DATA[0],PIN_F4
DRAM_RAS_N,PIN_AB4 | HEX3[6],PIN_W24 | PS2_CLK,PIN_D26 | OTG_DATA[1],PIN_D2
DRAM_WE_N,PIN_AD3 | HEX4[0],PIN_U9 | PS2_DAT,PIN_C24 | OTG_DATA[2],PIN_D1
FL_ADDR[0],PIN_AC18 | HEX4[1],PIN_U1 | UART_RXD,PIN_C25 | OTG_DATA[3],PIN_F7
FL_ADDR[1],PIN_AB18 | HEX4[2],PIN_U2 | UART_TXD,PIN_B25 | OTG_DATA[4],PIN_J5
FL_ADDR[2],PIN_AE19 | HEX4[3],PIN_T4 | LCD_RW,PIN_K4 | OTG_DATA[5],PIN_J8
FL_ADDR[3],PIN_AF19 | HEX4[4],PIN_R7 | LCD_EN,PIN_K3 | OTG_DATA[6],PIN_J7
FL_ADDR[4],PIN_AE18 | HEX4[5],PIN_R6 | LCD_RS,PIN_K1 | OTG_DATA[7],PIN_H6
FL_ADDR[5],PIN_AF18 | HEX4[6],PIN_T3 | LCD_DATA[0],PIN_J1 | OTG_DATA[8],PIN_E2
FL_ADDR[6],PIN_Y16 | HEX5[0],PIN_T2 | LCD_DATA[1],PIN_J2 | OTG_DATA[9],PIN_E1

```

OTG_DATA[10], PIN_K6	VGA_B[5], PIN_J11	ENET_CLK, PIN_B24	GPIO_0[31], PIN_J26
OTG_DATA[11], PIN_K5	VGA_B[6], PIN_C11	ENET_CMD, PIN_A21	GPIO_0[32], PIN_L23
OTG_DATA[12], PIN_G4	VGA_B[7], PIN_B11	ENET_CS_N, PIN_A23	GPIO_0[33], PIN_L24
OTG_DATA[13], PIN_G3	VGA_B[8], PIN_C12	ENET_INT, PIN_B21	GPIO_0[34], PIN_L25
OTG_DATA[14], PIN_J6	VGA_B[9], PIN_B12	ENET_RD_N, PIN_A22	GPIO_0[35], PIN_L19
OTG_DATA[15], PIN_K8	VGA_CLK, PIN_B8	ENET_WR_N, PIN_B22	GPIO_1[0], PIN_K25
OTG_INTO, PIN_B3	VGA_BLANK, PIN_D6	ENET_RST_N, PIN_B23	GPIO_1[1], PIN_K26
OTG_INT1, PIN_C3	VGA_HS, PIN_A7	IRDA_TXD, PIN_AE24	GPIO_1[2], PIN_M22
OTG_DACK0_N, PIN_C2	VGA_VS, PIN_D8	IRDA_RXD, PIN_AE25	GPIO_1[3], PIN_M23
OTG_DACK1_N, PIN_B2	VGA_SYNC, PIN_B7	SD_DAT, PIN_AD24	GPIO_1[4], PIN_M19
OTG_DREQ0, PIN_F6	I2C_SCLK, PIN_A6	SD_DAT3, PIN_AC23	GPIO_1[5], PIN_M20
OTG_DREQ1, PIN_E5	I2C_SDAT, PIN_B6	SD_CMD, PIN_Y21	GPIO_1[6], PIN_N20
OTG_FSPEED, PIN_F3	TD_DATA[0], PIN_J9	SD_CLK, PIN_AD25	GPIO_1[7], PIN_M21
OTG_LSPEED, PIN_G6	TD_DATA[1], PIN_E8	GPIO_0[0], PIN_D25	GPIO_1[8], PIN_M24
TDI, PIN_B14	TD_DATA[2], PIN_H8	GPIO_0[1], PIN_J22	GPIO_1[9], PIN_M25
TCS, PIN_A14	TD_DATA[3], PIN_H10	GPIO_0[2], PIN_E26	GPIO_1[10], PIN_N24
TCK, PIN_D14	TD_DATA[4], PIN_G9	GPIO_0[3], PIN_E25	GPIO_1[11], PIN_P24
TDO, PIN_F14	TD_DATA[5], PIN_F9	GPIO_0[4], PIN_F24	GPIO_1[12], PIN_R25
TD_RESET, PIN_C4	TD_DATA[6], PIN_D7	GPIO_0[5], PIN_F23	GPIO_1[13], PIN_R24
VGA_R[0], PIN_C8	TD_DATA[7], PIN_C7	GPIO_0[6], PIN_J21	GPIO_1[14], PIN_R20
VGA_R[1], PIN_F10	TD_HS, PIN_D5	GPIO_0[7], PIN_J20	GPIO_1[15], PIN_T22
VGA_R[2], PIN_G10	TD_VS, PIN_K9	GPIO_0[8], PIN_F25	GPIO_1[16], PIN_T23
VGA_R[3], PIN_D9	AUD_ADCLRCK, PIN_C5	GPIO_0[9], PIN_F26	GPIO_1[17], PIN_T24
VGA_R[4], PIN_C9	AUD_ADCDAT, PIN_B5	GPIO_0[10], PIN_N18	GPIO_1[18], PIN_T25
VGA_R[5], PIN_A8	AUD_DACLRCK, PIN_C6	GPIO_0[11], PIN_P18	GPIO_1[19], PIN_T18
VGA_R[6], PIN_H11	AUD_DACDAT, PIN_A4	GPIO_0[12], PIN_G23	GPIO_1[20], PIN_T21
VGA_R[7], PIN_H12	AUD_XCK, PIN_A5	GPIO_0[13], PIN_G24	GPIO_1[21], PIN_T20
VGA_R[8], PIN_F11	AUD_BCLK, PIN_B4	GPIO_0[14], PIN_K22	GPIO_1[22], PIN_U26
VGA_R[9], PIN_E10	ENET_DATA[0], PIN_D17	GPIO_0[15], PIN_G25	GPIO_1[23], PIN_U25
VGA_G[0], PIN_B9	ENET_DATA[1], PIN_C17	GPIO_0[16], PIN_H23	GPIO_1[24], PIN_U23
VGA_G[1], PIN_A9	ENET_DATA[2], PIN_B18	GPIO_0[17], PIN_H24	GPIO_1[25], PIN_U24
VGA_G[2], PIN_C10	ENET_DATA[3], PIN_A18	GPIO_0[18], PIN_J23	GPIO_1[26], PIN_R19
VGA_G[3], PIN_D10	ENET_DATA[4], PIN_B17	GPIO_0[19], PIN_J24	GPIO_1[27], PIN_T19
VGA_G[4], PIN_B10	ENET_DATA[5], PIN_A17	GPIO_0[20], PIN_H25	GPIO_1[28], PIN_U20
VGA_G[5], PIN_A10	ENET_DATA[6], PIN_B16	GPIO_0[21], PIN_H26	GPIO_1[29], PIN_U21
VGA_G[6], PIN_G11	ENET_DATA[7], PIN_B15	GPIO_0[22], PIN_H19	GPIO_1[30], PIN_V26
VGA_G[7], PIN_D11	ENET_DATA[8], PIN_B20	GPIO_0[23], PIN_K18	GPIO_1[31], PIN_V25
VGA_G[8], PIN_E12	ENET_DATA[9], PIN_A20	GPIO_0[24], PIN_K19	GPIO_1[32], PIN_V24
VGA_G[9], PIN_D12	ENET_DATA[10], PIN_C19	GPIO_0[25], PIN_K21	GPIO_1[33], PIN_V23
VGA_B[0], PIN_J13	ENET_DATA[11], PIN_D19	GPIO_0[26], PIN_K23	GPIO_1[34], PIN_W25
VGA_B[1], PIN_J14	ENET_DATA[12], PIN_B19	GPIO_0[27], PIN_K24	GPIO_1[35], PIN_W23
VGA_B[2], PIN_F12	ENET_DATA[13], PIN_A19	GPIO_0[28], PIN_L21	
VGA_B[3], PIN_G12	ENET_DATA[14], PIN_E18	GPIO_0[29], PIN_L20	
VGA_B[4], PIN_J10	ENET_DATA[15], PIN_D18	GPIO_0[30], PIN_J25	

## 2.7 Die Pin Belegung auf dem DE2-70 Board

Für das neue DE2-70 Entwicklungsboard von Altera mit dem komplexeren FPGA Baustein EPC2C70F896 ist die Pinbelegung für alle Anschlüsse in der Datei **DE2\_70\_pin\_assignments.csv** abgelegt. Die Namen für die festverdrahteten I/O Komponenten wurden entsprechend dem alten DE2 Board gewählt, um eine weitgehende Kompatibilität zu gewährleisten. Allerdings gibt es einige **Abweichungen**. So sind hier zusätzlich die Dezimalpunkte der Sieben-Segment-Anzeigen ansprechbar. Die Speicherbausteine (SRAM, DRAM, Flash) haben u.a. größere Adress- und Datenbusse. Die General Purpose Interfaces (GPIO\_0, GPIO\_1) können nur eingeschränkt als Typ `inout` definiert werden, so dass beim Anschluss des Touch Panels das DE2\_TOUCHPANEL Modul modifiziert werden muss, indem `GPIO_0(2 downto 0)` als `out` und `GPIO_0(35 downto 3)` als `in` deklariert werden. An dem PS2-Anschluss können über ein Y-Kabel eine Tastatur und eine Maus gleichzeitig angeschlossen und betrieben werden.

```

# Copyright (C) 1991-2009 Altera Corporation

# Quartus II Version 8.1
# File: DE2_70_pin_assignments.csv
# Generated on: 27.02.2009

# Note: The column header names should not be changed if you wish to import
this .csv file into the Quartus II software.

```

To,Location		
ENET_CLK, PIN_D27	HEX0[2], PIN_AH9	HEX7[4], PIN_H2
CLOCK_28, PIN_E16	HEX0[3], PIN_AD10	HEX7[5], PIN_H3
CLOCK_50, PIN_AD15	HEX0[4], PIN_AF10	HEX7[6], PIN_G1
CLK_50_2, PIN_D16	HEX0[5], PIN_AD11	HEX7_DP, PIN_G2
CLK_50_3, PIN_R28	HEX0[6], PIN_AD12	LCD_RW, PIN_F3
CLK_50_4, PIN_R3	HEX0_DP, PIN_AF12	LCD_EN, PIN_E2
SW[0], PIN_AA23	HEX1[0], PIN_AG13	LCD_RS, PIN_F2
SW[1], PIN_AB26	HEX1[1], PIN_AE16	LCD_DATA[0], PIN_E1
SW[2], PIN_AB25	HEX1[2], PIN_AF16	LCD_DATA[1], PIN_E3
SW[3], PIN_AC27	HEX1[3], PIN_AG16	LCD_DATA[2], PIN_D2
SW[4], PIN_AC26	HEX1[4], PIN_AE17	LCD_DATA[3], PIN_D3
SW[5], PIN_AC24	HEX1[5], PIN_AF17	LCD_DATA[4], PIN_C1
SW[6], PIN_AC23	HEX1[6], PIN_AD17	LCD_DATA[5], PIN_C2
SW[7], PIN_AD25	HEX1_DP, PIN_AC17	LCD_DATA[6], PIN_C3
SW[8], PIN_AD24	HEX2[0], PIN_AE7	LCD_DATA[7], PIN_B2
SW[9], PIN_AE27	HEX2[1], PIN_AF7	LCD_ON, PIN_F1
SW[10], PIN_W5	HEX2[2], PIN_AH5	LCD_BLON, PIN_G3
SW[11], PIN_V10	HEX2[3], PIN_AG4	VGA_B[0], PIN_B16
SW[12], PIN_U9	HEX2[4], PIN_AB18	VGA_B[1], PIN_C16
SW[13], PIN_T9	HEX2[5], PIN_AB19	VGA_B[2], PIN_A17
SW[14], PIN_L5	HEX2[6], PIN_AE19	VGA_B[3], PIN_B17
SW[15], PIN_L4	HEX2_DP, PIN_AC19	VGA_B[4], PIN_C18
SW[16], PIN_L7	HEX3[0], PIN_P6	VGA_B[5], PIN_B18
SW[17], PIN_L8	HEX3[1], PIN_P4	VGA_B[6], PIN_B19
KEY[0], PIN_T29	HEX3[2], PIN_N10	VGA_B[7], PIN_A19
KEY[1], PIN_T28	HEX3[3], PIN_N7	VGA_B[8], PIN_C19
KEY[2], PIN_U30	HEX3[4], PIN_M8	VGA_B[9], PIN_D19
KEY[3], PIN_U29	HEX3[5], PIN_M7	VGA_BLANK, PIN_C15
LEDR[0], PIN_AJ6	HEX3[6], PIN_M6	VGA_CLK, PIN_D24
LEDR[1], PIN_AK5	HEX3_DP, PIN_M4	VGA_G[0], PIN_A10
LEDR[2], PIN_AJ5	HEX4[0], PIN_P1	VGA_G[1], PIN_B11
LEDR[3], PIN_AJ4	HEX4[1], PIN_P2	VGA_G[2], PIN_A11
LEDR[4], PIN_AK3	HEX4[2], PIN_P3	VGA_G[3], PIN_C12
LEDR[5], PIN_AH4	HEX4[3], PIN_N2	VGA_G[4], PIN_B12
LEDR[6], PIN_AJ3	HEX4[4], PIN_N3	VGA_G[5], PIN_A12
LEDR[7], PIN_AJ2	HEX4[5], PIN_M1	VGA_G[6], PIN_C13
LEDR[8], PIN_AH3	HEX4[6], PIN_M2	VGA_G[7], PIN_B13
LEDR[9], PIN_AD14	HEX4_DP, PIN_L6	VGA_G[8], PIN_B14
LEDR[10], PIN_AC13	HEX5[0], PIN_M3	VGA_G[9], PIN_A14
LEDR[11], PIN_AB13	HEX5[1], PIN_L1	VGA_HS, PIN_J19
LEDR[12], PIN_AC12	HEX5[2], PIN_L2	VGA_R[0], PIN_D23
LEDR[13], PIN_AB12	HEX5[3], PIN_L3	VGA_R[1], PIN_E23
LEDR[14], PIN_AC11	HEX5[4], PIN_K1	VGA_R[2], PIN_E22
LEDR[15], PIN_AD9	HEX5[5], PIN_K4	VGA_R[3], PIN_D22
LEDR[16], PIN_AD8	HEX5[6], PIN_K5	VGA_R[4], PIN_H21
LEDR[17], PIN_AJ7	HEX5_DP, PIN_K6	VGA_R[5], PIN_G21
LEDG[0], PIN_W27	HEX6[0], PIN_H6	VGA_R[6], PIN_H20
LEDG[1], PIN_W25	HEX6[1], PIN_H4	VGA_R[7], PIN_F20
LEDG[2], PIN_W23	HEX6[2], PIN_H7	VGA_R[8], PIN_E20
LEDG[3], PIN_Y27	HEX6[3], PIN_H8	VGA_R[9], PIN_G20
LEDG[4], PIN_Y24	HEX6[4], PIN_G4	VGA_SYNC, PIN_B15
LEDG[5], PIN_Y23	HEX6[5], PIN_F4	VGA_VS, PIN_H19
LEDG[6], PIN_AA27	HEX6[6], PIN_E4	PS2_CLK, PIN_F24
LEDG[7], PIN_AA24	HEX6_DP, PIN_K2	PS2_DAT, PIN_E24
LEDG[8], PIN_AC14	HEX7[0], PIN_K3	PS2_MSCLK, PIN_D26
HEX0[0], PIN_AE8	HEX7[1], PIN_J1	PS2_MSDAT, PIN_D25
HEX0[1], PIN_AF9	HEX7[2], PIN_J2	UART_CTS, PIN_G22
	HEX7[3], PIN_H1	UART RTS, PIN_F23

UART_RXD, PIN_D21	SRAM_DQ[5], PIN_AH12	DRAM1_A[3], PIN_U6
UART_TXD, PIN_E21	SRAM_DQ[6], PIN_AJ12	DRAM1_A[4], PIN_U7
AUD_ADCDAT, PIN_E19	SRAM_DQ[7], PIN_AH16	DRAM1_A[5], PIN_V7
AUD_ADCLRCK, PIN_F19	SRAM_DQ[8], PIN_AK17	DRAM1_A[6], PIN_V8
AUD_BCLK, PIN_E17	SRAM_DQ[9], PIN_AJ17	DRAM1_A[7], PIN_W4
AUD_DACDAT, PIN_F18	SRAM_GW_N, PIN_AG18	DRAM1_A[8], PIN_W7
AUD_DACLRCK, PIN_G18	SRAM_OE_N, PIN_AD18	DRAM1_A[9], PIN_W8
AUD_XCK, PIN_D17	SRAM_WE_N, PIN_AF18	DRAM1_BA[0], PIN_T7
SRAM_A[0], PIN_AG8	DRAM_DQ[0], PIN_AC1	DRAM1_BA[1], PIN_T8
SRAM_A[1], PIN_AF8	DRAM_DQ[1], PIN_AC2	DRAM1_CAS_N, PIN_N8
SRAM_A[10], PIN_AF14	DRAM_DQ[10], PIN_AF2	DRAM1_CKE, PIN_L10
SRAM_A[11], PIN_AG14	DRAM_DQ[11], PIN_AF3	DRAM1_CLK, PIN_G5
SRAM_A[12], PIN_AE15	DRAM_DQ[12], PIN_AG2	DRAM1_CS_N, PIN_P9
SRAM_A[13], PIN_AF15	DRAM_DQ[13], PIN_AG3	DRAM1_LDQM0, PIN_M10
SRAM_A[14], PIN_AC16	DRAM_DQ[14], PIN_AH1	DRAM1_RAS_N, PIN_N9
SRAM_A[15], PIN_AF20	DRAM_DQ[15], PIN_AH2	DRAM1_UDQM1, PIN_U8
SRAM_A[16], PIN_AG20	DRAM_DQ[16], PIN_U1	DRAM1_WE_N, PIN_M9
SRAM_A[17], PIN_AE11	DRAM_DQ[17], PIN_U2	ENET_CMD, PIN_B27
SRAM_A[18], PIN_AF11	DRAM_DQ[18], PIN_U3	ENET_CS_N, PIN_C28
SRAM_A[2], PIN_AH7	DRAM_DQ[19], PIN_V2	ENET_D[0], PIN_A23
SRAM_A[3], PIN_AG7	DRAM_DQ[2], PIN_AC3	ENET_D[1], PIN_C22
SRAM_A[4], PIN_AG6	DRAM_DQ[20], PIN_V3	ENET_D[10], PIN_B25
SRAM_A[5], PIN_AG5	DRAM_DQ[21], PIN_W1	ENET_D[11], PIN_A25
SRAM_A[6], PIN_AE12	DRAM_DQ[22], PIN_W2	ENET_D[12], PIN_C24
SRAM_A[7], PIN_AG12	DRAM_DQ[23], PIN_W3	ENET_D[13], PIN_B24
SRAM_A[8], PIN_AD13	DRAM_DQ[24], PIN_Y1	ENET_D[14], PIN_A24
SRAM_A[9], PIN_AE13	DRAM_DQ[25], PIN_Y2	ENET_D[15], PIN_B23
SRAM_ADSC_N, PIN_AG17	DRAM_DQ[26], PIN_Y3	ENET_D[2], PIN_B22
SRAM_ADSP_N, PIN_AC18	DRAM_DQ[27], PIN_AA1	ENET_D[3], PIN_A22
SRAM_ADV_N, PIN_AD16	DRAM_DQ[28], PIN_AA2	ENET_D[4], PIN_B21
SRAM_BE_N[0], PIN_AC21	DRAM_DQ[29], PIN_AA3	ENET_D[5], PIN_A21
SRAM_BE_N[1], PIN_AC20	DRAM_DQ[3], PIN_AD1	ENET_D[6], PIN_B20
SRAM_BE_N[2], PIN_AD20	DRAM_DQ[30], PIN_AB1	ENET_D[7], PIN_A20
SRAM_BE_N[3], PIN_AH20	DRAM_DQ[31], PIN_AB2	ENET_D[8], PIN_B26
SRAM_CE1_N, PIN_AH19	DRAM_DQ[4], PIN_AD2	ENET_D[9], PIN_A26
SRAM_CE2, PIN_AG19	DRAM_DQ[5], PIN_AD3	ENET_INT, PIN_C27
SRAM_CE3_N, PIN_AD22	DRAM_DQ[6], PIN_AE1	ENET_IOR_N, PIN_A28
SRAM_CLK, PIN_AD7	DRAM_DQ[7], PIN_AE2	ENET_IOW_N, PIN_B28
SRAM_DPA[0], PIN_AK9	DRAM_DQ[8], PIN_AE3	ENET_RESET_N, PIN_B29
SRAM_DPA[1], PIN_AJ23	DRAM_DQ[9], PIN_AF1	EXT_CLOCK, PIN_R29
SRAM_DPA[2], PIN_AK20	DRAMO_A[0], PIN_AA4	FLASH_A[0], PIN_AF24
SRAM_DPA[3], PIN_AJ9	DRAMO_A[1], PIN_AA5	FLASH_A[1], PIN_AG24
SRAM_DQ[0], PIN_AH10	DRAMO_A[10], PIN_Y8	FLASH_A[10], PIN_AH26
SRAM_DQ[1], PIN_AJ10	DRAMO_A[11], PIN_AE4	FLASH_A[11], PIN_AJ26
SRAM_DQ[10], PIN_AH17	DRAMO_A[12], PIN_AF4	FLASH_A[12], PIN_AK26
SRAM_DQ[11], PIN_AJ18	DRAMO_A[2], PIN_AA6	FLASH_A[13], PIN_AJ25
SRAM_DQ[12], PIN_AH18	DRAMO_A[3], PIN_AB5	FLASH_A[14], PIN_AK25
SRAM_DQ[13], PIN_AK19	DRAMO_A[4], PIN_AB7	FLASH_A[15], PIN_AH24
SRAM_DQ[14], PIN_AJ19	DRAMO_A[5], PIN_AC4	FLASH_A[16], PIN_AG25
SRAM_DQ[15], PIN_AK23	DRAMO_A[6], PIN_AC5	FLASH_A[17], PIN_AF21
SRAM_DQ[16], PIN_AJ20	DRAMO_A[7], PIN_AC6	FLASH_A[18], PIN_AD21
SRAM_DQ[17], PIN_AK21	DRAMO_A[8], PIN_AD4	FLASH_A[19], PIN_AK28
SRAM_DQ[18], PIN_AJ21	DRAMO_A[9], PIN_AC7	FLASH_A[2], PIN_AE23
SRAM_DQ[19], PIN_AK22	DRAMO_BA[0], PIN_AA9	FLASH_A[20], PIN_AJ28
SRAM_DQ[2], PIN_AK10	DRAMO_BA[1], PIN_AA10	FLASH_A[21], PIN_AE20
SRAM_DQ[20], PIN_AJ22	DRAMO_CAS_N, PIN_W10	FLASH_A[3], PIN_AG23
SRAM_DQ[21], PIN_AH15	DRAMO_CKE, PIN_AA8	FLASH_A[4], PIN_AF23
SRAM_DQ[22], PIN_AJ15	DRAMO_CLK, PIN_AD6	FLASH_A[5], PIN_AG22
SRAM_DQ[23], PIN_AJ16	DRAMO_CS_N, PIN_Y10	FLASH_A[6], PIN_AH22
SRAM_DQ[24], PIN_AK14	DRAMO_LDQM0, PIN_V9	FLASH_A[7], PIN_AF22
SRAM_DQ[25], PIN_AJ14	DRAMO_RAS_N, PIN_Y9	FLASH_A[8], PIN_AH27
SRAM_DQ[26], PIN_AJ13	DRAMO_UDQM1, PIN_AB6	FLASH_A[9], PIN_AJ27
SRAM_DQ[27], PIN_AH13	DRAMO_WE_N, PIN_W9	FLASH_BYTEN, PIN_Y29
SRAM_DQ[28], PIN_AK12	DRAM1_A[0], PIN_T5	FLASH_CE_N, PIN_AG28
SRAM_DQ[29], PIN_AK7	DRAM1_A[1], PIN_T6	FLASH_DQ[0], PIN_AF29
SRAM_DQ[3], PIN_AJ11	DRAM1_A[10], PIN_T4	FLASH_DQ[1], PIN_AE28
SRAM_DQ[30], PIN_AJ8	DRAM1_A[11], PIN_Y4	FLASH_DQ[10], PIN_AD29
SRAM_DQ[31], PIN_AK8	DRAM1_A[12], PIN_Y7	FLASH_DQ[11], PIN_AC28
SRAM_DQ[4], PIN_AK11	DRAM1_A[2], PIN_U4	FLASH_DQ[12], PIN_AC30

FLASH_DQ[13],PIN_AB30	GPIO_0[34],PIN_N21	OTG_D[13],PIN_E8
FLASH_DQ[14],PIN_AA30	GPIO_0[35],PIN_N24	OTG_D[14],PIN_D9
FLASH_DQ15_AM1,PIN_AE24	GPIO_1[0],PIN_AH14	OTG_D[15],PIN_G10
FLASH_DQ[2],PIN_AE30	GPIO_1[1],PIN_G27	OTG_D[2],PIN_G11
FLASH_DQ[3],PIN_AD30	GPIO_1[2],PIN_AG15	OTG_D[3],PIN_F11
FLASH_DQ[4],PIN_AC29	GPIO_1[3],PIN_G28	OTG_D[4],PIN_J12
FLASH_DQ[5],PIN_AB29	GPIO_1[4],PIN_H27	OTG_D[5],PIN_H12
FLASH_DQ[6],PIN_AA29	GPIO_1[5],PIN_L24	OTG_D[6],PIN_H13
FLASH_DQ[7],PIN_Y28	GPIO_1[6],PIN_H28	OTG_D[7],PIN_G13
FLASH_DQ[8],PIN_AF30	GPIO_1[7],PIN_L25	OTG_D[8],PIN_D4
FLASH_DQ[9],PIN_AE29	GPIO_1[8],PIN_K27	OTG_D[9],PIN_D5
FLASH_OE_N,PIN_AG29	GPIO_1[9],PIN_L28	OTG_DACK0_N,PIN_D12
FLASH_RST_N,PIN_AH28	GPIO_1[10],PIN_K28	OTG_DACK1_N,PIN_E12
FLASH_RY_N,PIN_AH30	GPIO_1[11],PIN_L27	OTG_DREQ0,PIN_G12
FLASH_WE_N,PIN_AJ29	GPIO_1[12],PIN_K29	OTG_DREQ1,PIN_F12
FLASH_WP_N,PIN_AH29	GPIO_1[13],PIN_M25	OTG_FSPEED,PIN_F7
GPIO_0[0],PIN_T25	GPIO_1[14],PIN_K30	OTG_INT0,PIN_F13
GPIO_0[1],PIN_C30	GPIO_1[15],PIN_M24	OTG_INT1,PIN_J13
GPIO_0[2],PIN_T24	GPIO_1[16],PIN_AF27	OTG_LSPEED,PIN_F8
GPIO_0[3],PIN_C29	GPIO_1[17],PIN_L29	OTG_OE_N,PIN_D10
GPIO_0[4],PIN_E28	GPIO_1[18],PIN_AF28	OTG_RESET_N,PIN_H14
GPIO_0[5],PIN_D29	GPIO_1[19],PIN_L30	OTG_WE_N,PIN_E11
GPIO_0[6],PIN_E27	GPIO_1[20],PIN_P26	SD_CLK,PIN_T26
GPIO_0[7],PIN_D28	GPIO_1[21],PIN_P28	SD_CMD,PIN_W28
GPIO_0[8],PIN_E29	GPIO_1[22],PIN_P25	SD_DAT,PIN_W29
GPIO_0[9],PIN_G25	GPIO_1[23],PIN_P27	SD_DAT3,PIN_Y30
GPIO_0[10],PIN_E30	GPIO_1[24],PIN_M29	TD1_CLK27,PIN_G15
GPIO_0[11],PIN_G26	GPIO_1[25],PIN_R26	TD1_D[0],PIN_A6
GPIO_0[12],PIN_F29	GPIO_1[26],PIN_M30	TD1_D[1],PIN_B6
GPIO_0[13],PIN_G29	GPIO_1[27],PIN_R27	TD1_D[2],PIN_A5
GPIO_0[14],PIN_F30	GPIO_1[28],PIN_P24	TD1_D[3],PIN_B5
GPIO_0[15],PIN_G30	GPIO_1[29],PIN_N28	TD1_D[4],PIN_B4
GPIO_0[16],PIN_H23	GPIO_1[30],PIN_P23	TD1_D[5],PIN_C4
GPIO_0[17],PIN_H29	GPIO_1[31],PIN_N29	TD1_D[6],PIN_A3
GPIO_0[18],PIN_G24	GPIO_1[32],PIN_R23	TD1_D[7],PIN_B3
GPIO_0[19],PIN_H30	GPIO_1[33],PIN_P29	TD1_HS,PIN_E13
GPIO_0[20],PIN_J29	GPIO_1[34],PIN_R22	TD1_RESET_N,PIN_D14
GPIO_0[21],PIN_H25	GPIO_1[35],PIN_P30	TD1_VS,PIN_E14
GPIO_0[22],PIN_J30	I2C_SCLK,PIN_J18	TD2_CLK27,PIN_H15
GPIO_0[23],PIN_H24	I2C_SDAT,PIN_H18	TD2_D[0],PIN_C10
GPIO_0[24],PIN_J25	IRDA_RXD,PIN_W22	TD2_D[1],PIN_A9
GPIO_0[25],PIN_K24	IRDA_TXD,PIN_W21	TD2_D[2],PIN_B9
GPIO_0[26],PIN_J24	OTG_A[0],PIN_E9	TD2_D[3],PIN_C9
GPIO_0[27],PIN_K25	OTG_A[1],PIN_D8	TD2_D[4],PIN_A8
GPIO_0[28],PIN_L22	OTG_CS_N,PIN_E10	TD2_D[5],PIN_B8
GPIO_0[29],PIN_M21	OTG_D[0],PIN_H10	TD2_D[6],PIN_A7
GPIO_0[30],PIN_L21	OTG_D[1],PIN_G9	TD2_D[7],PIN_B7
GPIO_0[31],PIN_M22	OTG_D[10],PIN_D6	TD2_HS,PIN_E15
GPIO_0[32],PIN_N22	OTG_D[11],PIN_E7	TD2_RESET_N,PIN_B10
GPIO_0[33],PIN_N25	OTG_D[12],PIN_D7	TD2_VS,PIN_D15

### 3. Die Programmierung

#### 3.1 Die JTAG-Schnittstelle

Für die Programmierung der Bausteine auf dem UP Board steht eine spezielle, international ge normte Schnittstelle zur Verfügung. Diese JTAG-Schnittstelle (Joint European Test Action Group) wurde ursprünglich zum Testen elektronischer Bausteine entwickelt. Sie ist heutzutage in allen FPGAs enthalten und wird zu deren Programmierung genutzt, ohne dass der Baustein dafür aus der Schaltung entfernt werden muss. Dabei ist die Schnittstelle kaskadierbar wie bei einer Daisy-Chain, so dass mehrere Bausteine über eine einzige JTAG-Schnittstelle gleichzeitig programmiert werden können.

Die JTAG-Schnittstelle besteht aus 4 (5) Leitungen, deren Aufgaben im Folgenden kurz beschrieben werden soll.

##### 3.1.1 Der TDI Port

Über den Test Data Input Port werden alle JTAG-Steuerbefehle und die Programmierung (\*.sof, \*.pof Dateien) an die programmierbaren Bausteine (CPLD, FPGA) übertragen. Der TDI-Anschluss wird zu weiteren Bausteinen seriell durchgeschleift, um diese gleichzeitig programmieren zu können. Die Datenübernahme erfolgt jeweils mit der ansteigenden Flanke des Taktsignals (TCK).

##### 3.1.2 Der TDO Port

Der Test Data Output Port bildet den seriellen Ausgang für die Programmierdaten, die so über den nächsten Eingang TDI zu einem weiteren Baustein gelangen, so dass auch nachfolgende Bausteine programmiert werden können.

##### 3.1.3 Der TCK Port

Der Test Clock Eingang überträgt das Taktsignal für die Programmierdaten. Es werden beide Taktflanken zur IC-internen Steuerung verwendet.

##### 3.1.4 Der TMS Port

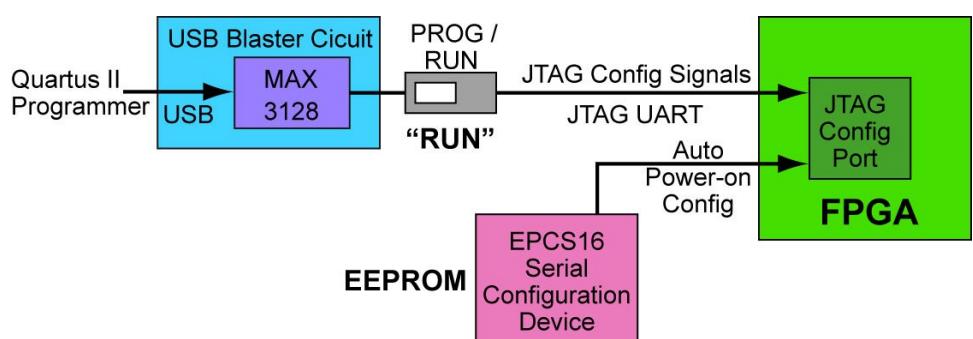
Über den Test Mode Select Eingang wird der Test Access Port Controller (TAP) gesteuert, der auf dem programmierbaren Baustein mit integriert ist. Die Steuerung selbst ist recht kompliziert und soll hier nicht weiter beschrieben werden. Es werden dazu beide Taktflanken ausgenutzt.

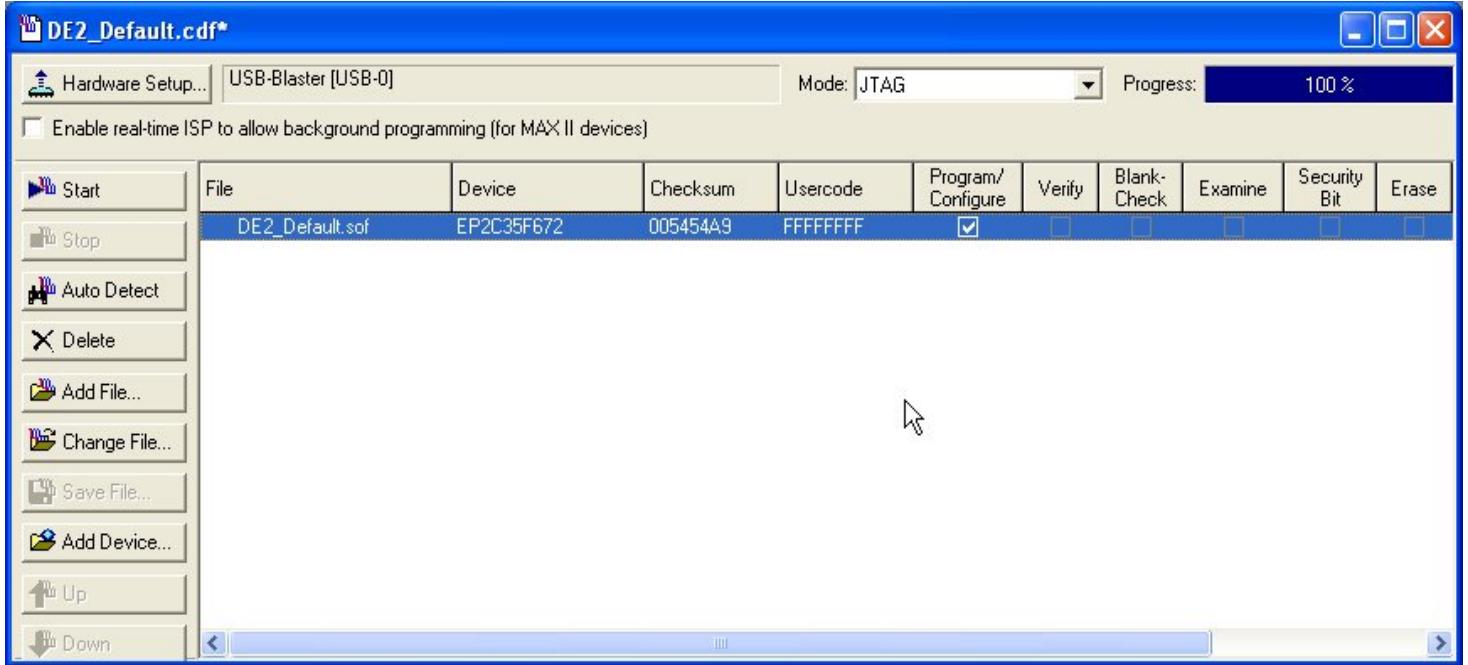
##### 3.1.5 Der TRST Port

Der Test Reset Port stellt eine optionale Reset-Leitung zur Rücksetzung der internen JTAG-Schaltkreise (TAP-Controller, Instruction-Register) zur Verfügung. Für die Programmierung spielt die TRST-Leitung keine Rolle und bleibt unbenutzt.

#### 3.2 Der JTAG Mode

Da mit Verlust der Versorgungsspannung die Programmierung des FPGA Bausteins verloren geht, kann das FPGA-Design auch in einem speziellen, externen EEPROM (EPCS16) abgespeichert werden.

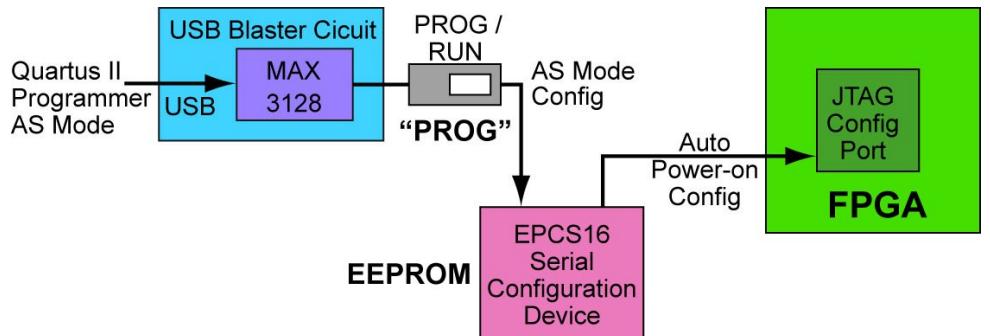




Beim Einschalten des DE2-Boards (Power Up) wird das dort abgelegte Programm dann als erstes seriell in das FPGA geladen. So wird standardmäßig die Datei **DE2\_Default.sof** beim Einschalten aus dem EEPROM ausgelesen. Sie enthält einige Test-Routinen für die Ein- und Ausgänge des Boards.

Im normalen Betrieb (JTAG Mode) kann aber auch eine beliebige andere Datei vom Typ \*.sof direkt in den Cyclon II Baustein geladen werden. SOF steht dabei für **SRAM Object File**. Dazu muss der Schalter an der linken Seite des Boards auf **RUN** (Normalstellung) stehen, so dass der JTAG Config Port direkt angesprochen wird. Das EEPROM Register wird dabei überbrückt, wie im Bild gezeigt. Zur Programmierung des FPGA Bausteins ist nun der Programmer aufzurufen, wie er oben gezeigt ist. Es ist darauf zu achten, dass der Mode JTAG ausgewählt ist und das Fenster **Program/Configure** markiert ist.

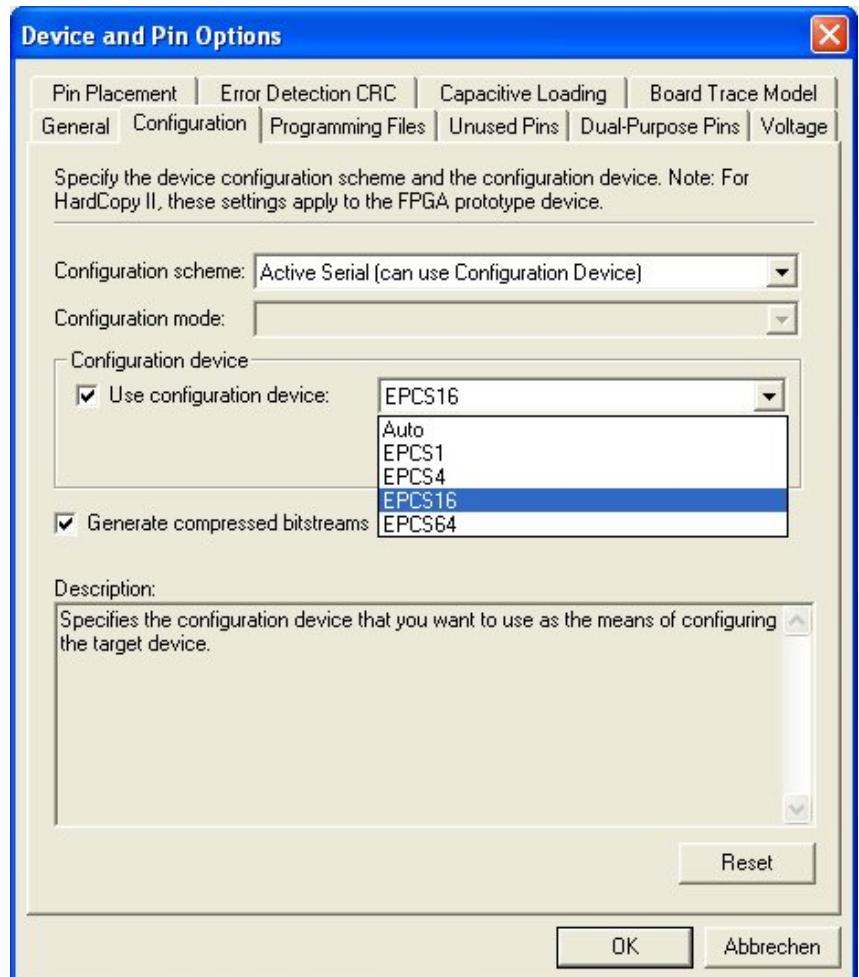
### 3.3 Der AS Mode



Beim Active Serial Programming (AS) muss der **Prog./Run Schalter** auf Prog. stehen, damit die speziell erstellte Programmdatei in das Konfigurations EEPROM EPICS16 geschrieben wird, anstatt das FPGA direkt zu programmieren. Dazu wird eine \*.pof Datei des Designs benötigt. POF steht dabei für **Programmer Object File**. Diese Datei kann gleichzeitig mit der SOF-Datei erstellt werden, wenn dazu die folgende Einstellung vorgenommen wird: **Assignments → Device → Device and Pin Options ...** Es öffnet sich das nebenstehende Fenster, in dem **Active Serial** als Configuration Scheme und **EPICS16** als EEPROM auszuwählen ist. Die nachfolgende Compilation liefert dann die passende POF-Datei. Anschließend kann der Programmer aufgerufen werden. Darin ist auf die unten gezeigte Einstellung zu achten. Der Mode muss auf **Active Serial Programming** stehen. Das Kästchen **Program/Configure** ist zu markieren. Die unten gezeigte Information:

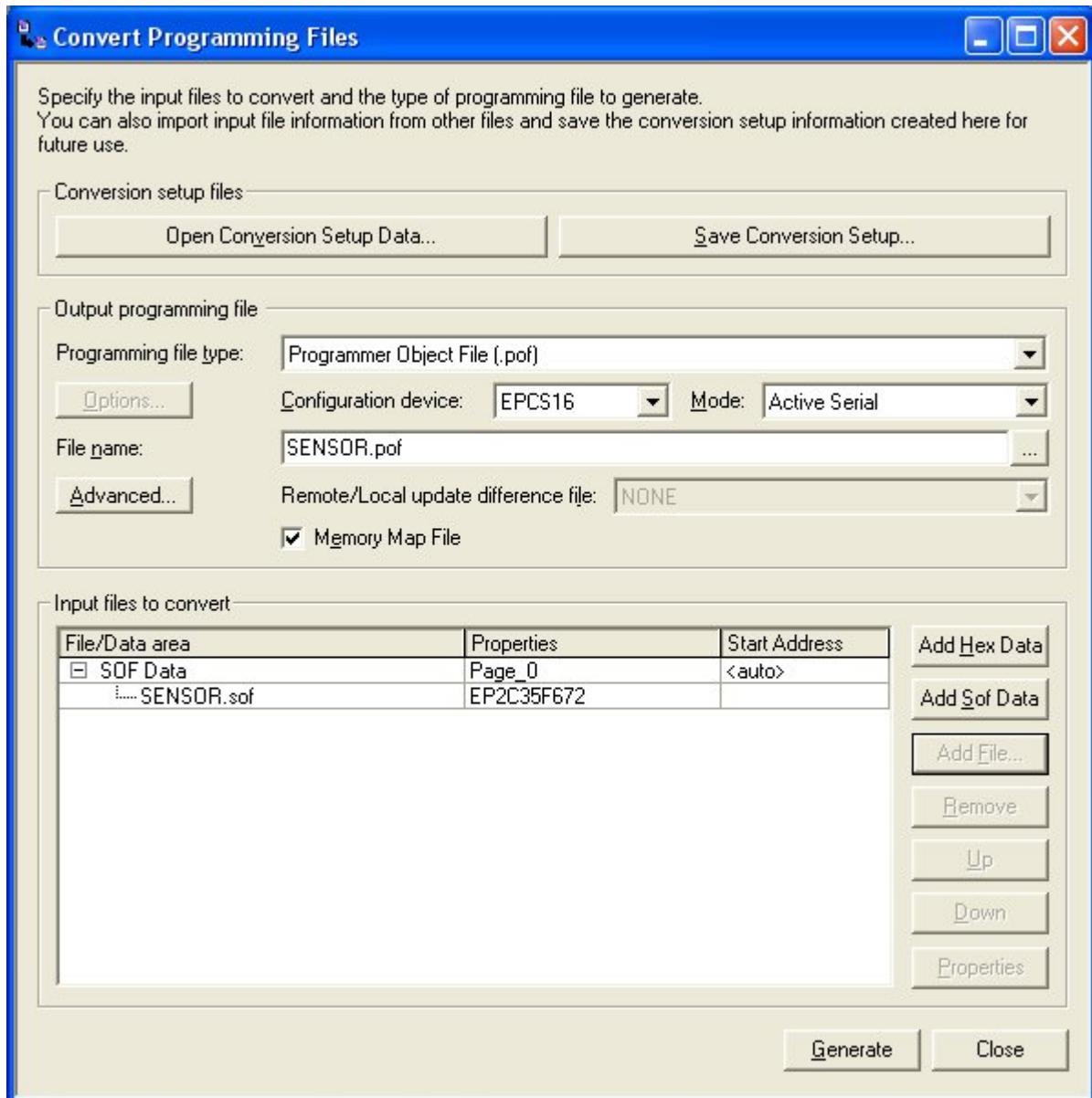
*„Some devices in current device list cannot be added to selected programming mode Active Serial Programming. Do you want to clear all devices in current device list and switch to selected mode?“*

kann mit „Ja“ quittiert werden. Bevor Sie die EEPROM Programmierung starten, stellen Sie den **Schiebeschalter** auf Prog. Die Programmierung selbst dauert deutlich länger als die JTAG Programmierung von SRAM-Zellen (etwa Faktor 100). Dazu trägt auch das vorausgehende Löschen des Speichers bei.

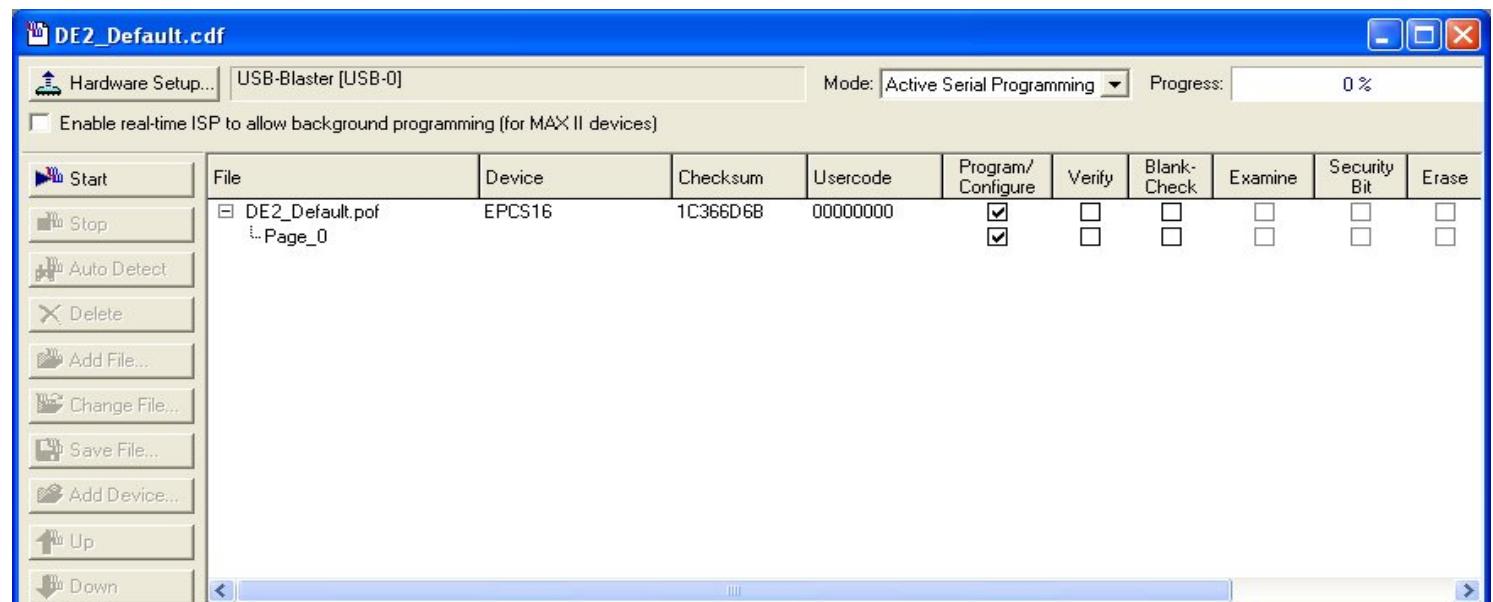


### 3.3 Generierung einer POF-Datei

Die POF-Datei für die Programmierung des EEPROMS kann aber auch nachträglich über den Aufruf **File → Convert Programming Files ...** kreiert werden, indem in Abhängigkeit des Dateinamens die folgenden Angaben in dem sich öffnenden Fenster gemacht werden. Voraussetzung ist, dass eine entsprechende SOF-Datei bereits existiert.



Bei der Programmierung mit Hilfe des Programmers ist darauf zu achten, dass der Mode **Active Serial Programming** eingestellt ist. Verfolgen Sie die lange Programmierdauer es EEPROMS!



## 4. Dateinamen

Quartus II benutzt und erzeugt bei der Compilierung eines Designs unterschiedlichste Dateien. Für die Sicherung des Designs sind nur die Projekt- und Design Files von Bedeutung (Die wichtigsten sind kursiv gekennzeichnet). Einen begrenzten Überblick gibt die folgende Aufstellung:

### 4.1 Project Files

<i>Quartus II Project File</i>	*.qpf
<i>Quartus II Settings File</i>	*.qsf
<i>Quartus II Workspace File</i>	*.qws
<i>Quartus II Default Setting File</i>	*.qdf

### 4.2 Design Files

Altera Design File	*.adf
<i>Block Design File</i>	*.bdf
EDIF Input File	*.edf
Graphic Design File	*.gdf
OrCAD Schematic File	*.sch
State Machine File	*.smf
Text Design File	*.tdf
<i>Verilog Design File</i>	*.v
<i>VHDL Design File</i>	*.vhd
Waveform Design File	*.wdf
Xilinx Netlist File	*.xnf

### 4.3 Ancillary Data Files

Assignment and Config. File	*.acf
Assignment and Config. Output	*.aco
<i>Block Symbol File</i>	*.bsf
Command File	*.cmd
EDIF Command File	*.edc
Fit File	*.fit
Intel Hexadecimal Format File	*.hex
History File	*.hst
Include File	*.inc
JTAG Chain File	*.jcf
Library Mapping File	*.lmf

### Log File

<i>Memory Initialization File</i>	*.mif
Memory Initialization Output File	*.mio
Message Text File	*.mtf
Programmer Log File	*.plf
Report File	*.rpt
Simulator Channel File	*.scf
Standard Delay Format	*.sdf
Standard Delay Format Output File	*.sdo
Symbol File	*.sym
Table File	*.tbl
Tabular Text File	*.ttf
Text Design Export File	*.tdx
Text Design Output File	*.tdo
Timing Analyzer Output File	*.tao
Vector File	*.vec
Verilog Quartus Mapping File	*.vqm
VHDL Memory Model Output File	*.vmo

### 4.4 Non Editable Ancillary File Types

Compiler Netlist File	*.cnf
Hierarchy Interconnect File	*.hif
JEDEC File	*.jed
Node Database File	*.ndb
Programmer Object File	*.pof
Raw Binary File	*.rbf
Serial Bitstream File	*.sbf
Simulator Initialization File	*.sif
Simulator Netlist File	*.snf
SRAM Object File	*.sof

## 5. Pseudozufallsfolgengeneratoren

Bei vielen Anwendungen wird eine Folge von Zufallswerten benötigt. Diese lassen sich sehr einfach mit einem rückgekoppelten Schieberegister erzeugen. Die folgende Tabelle definiert dazu die rückzukoppelnden Register für jede Registerlänge von 3 bis 168. Die Nummerierung beginnt dabei mit dem Register 0. Die Verknüpfung der angegebenen Speicherelemente kann mit XOR oder XNOR Gattern erfolgen. Es ist darauf zu achten, dass bei einem Set / Reset niemals alle Register gleichzeitig auf 0 (XOR) bzw. auf 1 (XNOR) gesetzt werden. Das Schieberegister muss für die angegebene Tabelle **linksschiebend** (vom LSB zum MSB) aufgebaut werden. Es werden maximal 4 rückzukoppelnde Register benötigt.

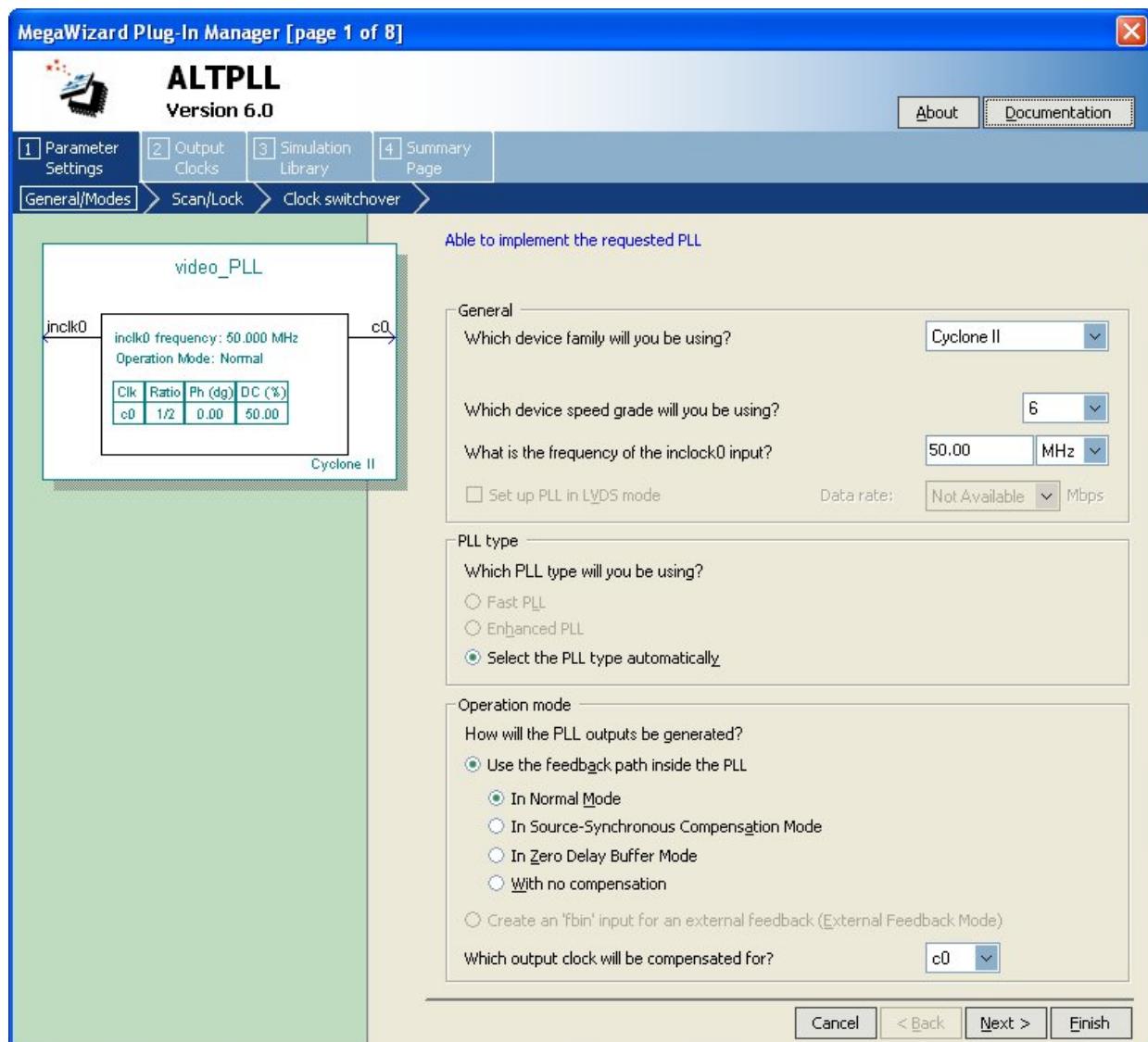
Rückzukoppelnde Elemente eines Schieberegisters							
Anzahl	Rückkopplung	Anzahl	Rückkopplung	Anzahl	Rückkopplung	Anzahl	Rückkopplung
3	2, 1	45	44, 43, 42, 41	87	86, 73	129	128, 123
4	3, 2	46	45, 44, 25, 24	88	87, 86, 16, 15	130	129, 126
5	4, 2	47	46, 41	89	88, 50	131	130, 129, 83, 82
6	5, 4	48	47, 46, 20, 19	90	89, 88, 71, 70	132	131, 102
7	6, 5	49	48, 39	91	90, 89, 7, 6	133	132, 131, 81, 80
8	7, 5, 4, 3	50	49, 48, 23, 22	92	91, 90, 79, 78	134	133, 76
9	8, 4	51	50, 49, 35, 34	93	92, 90	135	134, 123
10	9, 6	52	51, 48	94	93, 72	136	135, 134, 10, 9
11	10, 8	53	52, 51, 37, 36	95	94, 83	137	136, 115
12	11, 5, 3, 0	54	53, 52, 17, 16	96	95, 93, 48, 46	138	137, 136, 130, 129
13	12, 3, 2, 1	55	54, 30	97	96, 90	139	138, 135, 133, 130
14	13, 4, 2, 0	56	5, 54, 34, 33	98	97, 86	140	139, 110
15	14, 13	57	56, 49	99	98, 96, 53, 51	141	140, 139, 109, 108
16	15, 14, 12, 3	58	57, 38	100	99, 63	142	141, 120
17	16, 13	59	58, 57, 37, 36	101	100, 99, 94, 93	143	142, 141, 122, 121
18	17, 10	60	59, 58	102	101, 100, 35, 34	144	143, 142, 74, 73
19	18, 5, 1, 0	61	60, 59, 45, 44	103	102, 93	145	144, 92
20	19, 16	62	61, 60, 5, 4	104	103, 102, 93, 92	146	145, 144, 86, 85
21	20, 18	63	62, 61	105	104, 88	147	146, 145, 109, 108
22	21, 20	64	63, 62, 60, 59	106	105, 90	148	147, 120
23	22, 17	65	64, 46	107	106, 104, 43, 41	149	148, 147, 39, 38
24	23, 22, 21, 16	66	65, 64, 56, 55	108	107, 76	150	149, 96
25	24, 21	67	66, 65, 57, 56	109	108, 107, 102, 101	151	150, 147
26	25, 5, 1, 0	68	67, 58	110	109, 108, 97, 96	152	151, 150, 86, 85
27	26, 4, 1, 0	69	68, 66, 41, 39	111	110, 100	153	152, 151
28	27, 24	70	69, 68, 54, 53	112	111, 109, 68, 66	154	132, 151, 26, 24
29	28, 26	71	70, 64	113	112, 103	155	154, 153, 123, 122
30	29, 5, 3, 0	72	71, 65, 24, 18	114	113, 112, 32, 31	156	155, 154, 40, 39
31	30, 27	73	72, 47	115	114, 113, 100, 99	157	156, 155, 130, 129
32	31, 21, 1, 0	74	73, 72, 58, 57	116	115, 114, 45, 44	158	157, 156, 131, 130
33	32, 19	75	74, 73, 64, 63	117	116, 114, 98, 96	159	158, 127
34	33, 26, 1, 0	76	75, 74, 40, 39	118	117, 84	160	159, 158, 141, 140
35	34, 32	77	76, 75, 46, 45	119	118, 110	161	160, 142
36	35, 24	78	77, 76, 58, 57	120	119, 112, 8, 1	162	161, 160, 74, 73
37	36, 4, 3, 2, 1, 0	79	78, 69	121	120, 102	163	162, 161, 103, 102
38	37, 5, 4, 0	80	79, 78, 42, 41	122	121, 120, 62, 61	164	163, 162, 150, 149
39	38, 34	81	80, 76	123	122, 120	165	164, 163, 134, 133
40	39, 37, 20, 18	82	81, 78, 46, 43	124	123, 86	166	165, 164, 127, 126
41	40, 37	83	82, 81, 37, 36	125	124, 123, 17, 16	167	166, 160
42	41, 40, 19, 18	84	83, 70	126	125, 124, 89, 88	168	167, 165, 152, 150
43	42, 41, 37, 36	85	84, 83, 57, 56	127	126, 125		
44	43, 42, 17, 16	86	85, 84, 73, 72	128	127, 125, 100, 98		

## 6. Der PLL-Wizard

Der PLL-Wizard (MegaWizard Plug-In Manager) kann benutzt werden, um aus dem vorhandenen Systemtakt (inclk0 input mit 50 MHz oder 27 MHz) einen anderen Takt zu erzeugen, wie er z.B. für die Ansteuerung eines externen VGA Monitors benötigt wird. Beide Takte müssen dabei in einem **rationalen Verhältnis** zu einander stehen. Es sind allerdings nicht alle Faktoren / Divisoren (Clock multiplication factor / Clock division factor) realisierbar, was beim Einsatz des PLL-Wizards entsprechend signalisiert wird. Es können bis zu drei verschiedene **Ausgangstaktsignale** (c0, c1, c2) definiert werden.

Darüber hinaus können alle Einstellungen, bis auf die neue Taktfrequenz selber, so vorgenommen werden, wie in den unten gezeigten screen shots zu sehen. Der hier benutzte Cyclon II Baustein besitzt einen „**Speed Grade**“ von 6. Ein kleinerer Wert verweist hier auf eine höhere Geschwindigkeit.

Auf dem Baustein selbst stehen bis zu 4 PLL Module zur Verfügung. Für jedes externe Taktignal wie dem Systemtakt CLOCK\_50 steht jedoch nur ein PLL Modul zur Verfügung. Daraus können wie erwähnt bis zu drei Taktsignale (c0, c1, c2) erzeugt werden. Die Takte können sich in der Ausgangsfrequenz (clock frequency), der relativen Phase (Clock phase shift) und des Tastverhältnisses (Clock duty cycle (%)) unterscheiden.



MegaWizard Plug-In Manager [page 2 of 8]

## ALTPPLL Version 6.0

General/Modes > Scan/Lock > Clock switchover >

Able to implement the requested PLL

Cyclone II

**Dynamic configuration**

- Create optional inputs for dynamic reconfiguration

**Optional inputs**

- Create an 'pllena' input to selectively enable the PLL
- Create an 'areset' input to asynchronously reset the PLL
- Create an 'pfdena' input to selectively enable the phase/freq. detector

**Lock output**

- Create 'locked' output
- Enable self-reset on loss of lock
- Hold 'locked' output low for  cycles after the PLL initializes

**Advanced PLL Parameters**

Using these parameters is recommended for advanced users only

- Create output file(s) using the 'Advanced' PLL parameters
  - Configurations with output clock(s) that use cascade counters are not supported

Cancel < Back Next > Finish

MegaWizard Plug-In Manager [page 3 of 8]

## ALTPPLL Version 6.0

General/Modes > Scan/Lock > Clock switchover >

Able to implement the requested PLL

Cyclone II

**Clock switchover**

- Create an 'inclk1' input for a second input clock
 

What is the frequency of the inclk1 input?  MHz
- Create a 'clkswitch' input to manually select between the input clocks  
(The clkswitch input will behave as an input clock selection control input)
- Allow PLL to automatically control the switching between input clocks  
(The clkswitch input will behave as a manual override control input)

**Input clock switch**

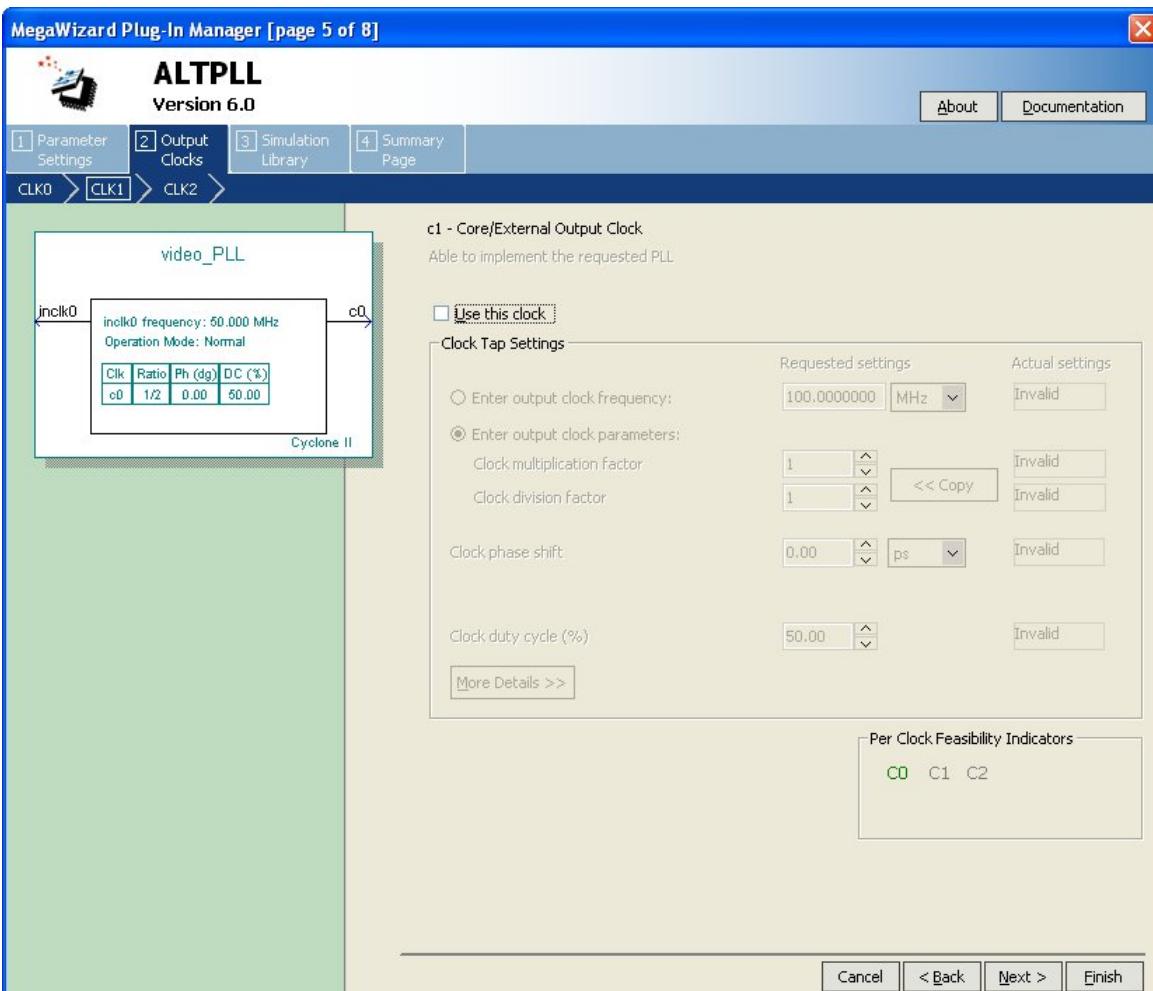
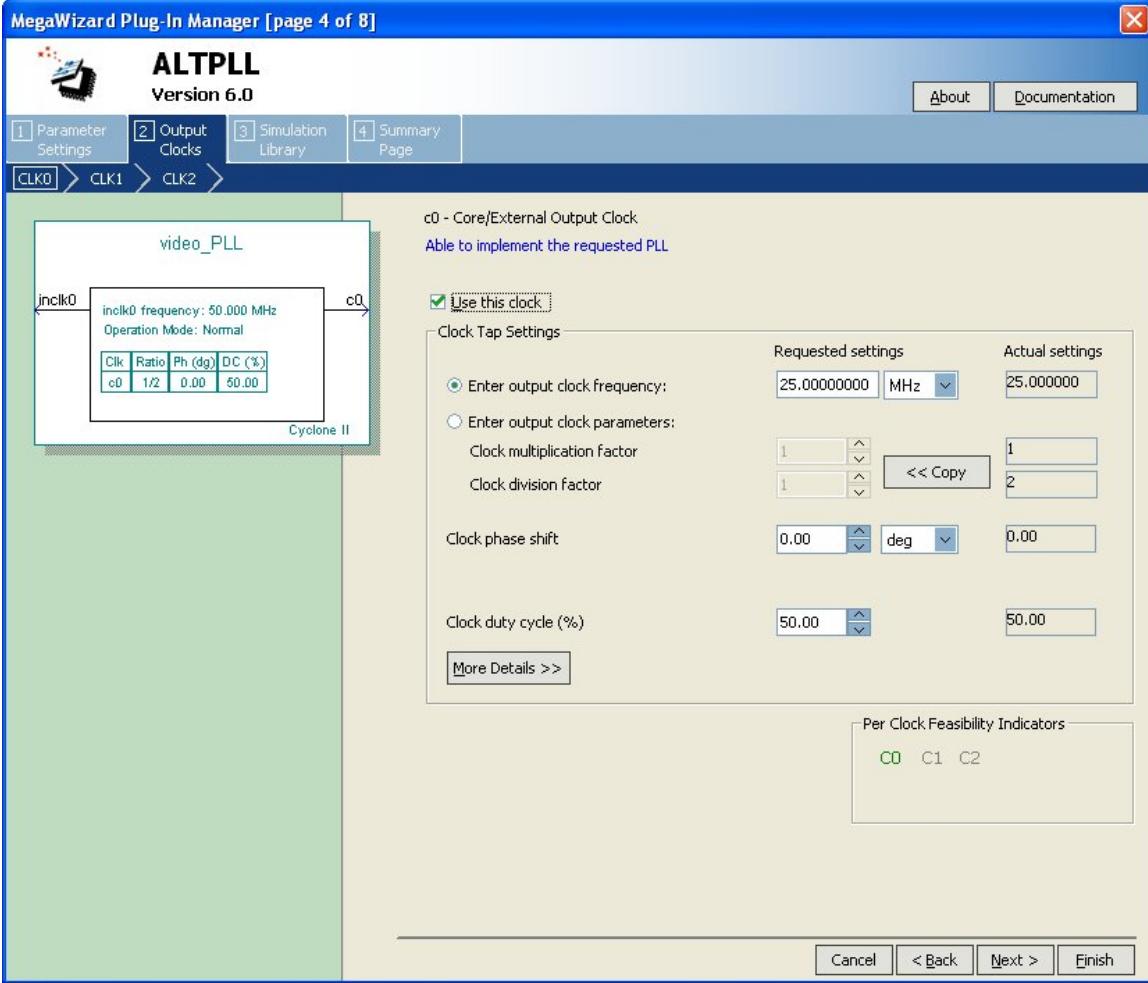
- Perform input clock switch when the input clock goes bad
- Create a 'gclkswitch' input to dynamically toggle between input clocks

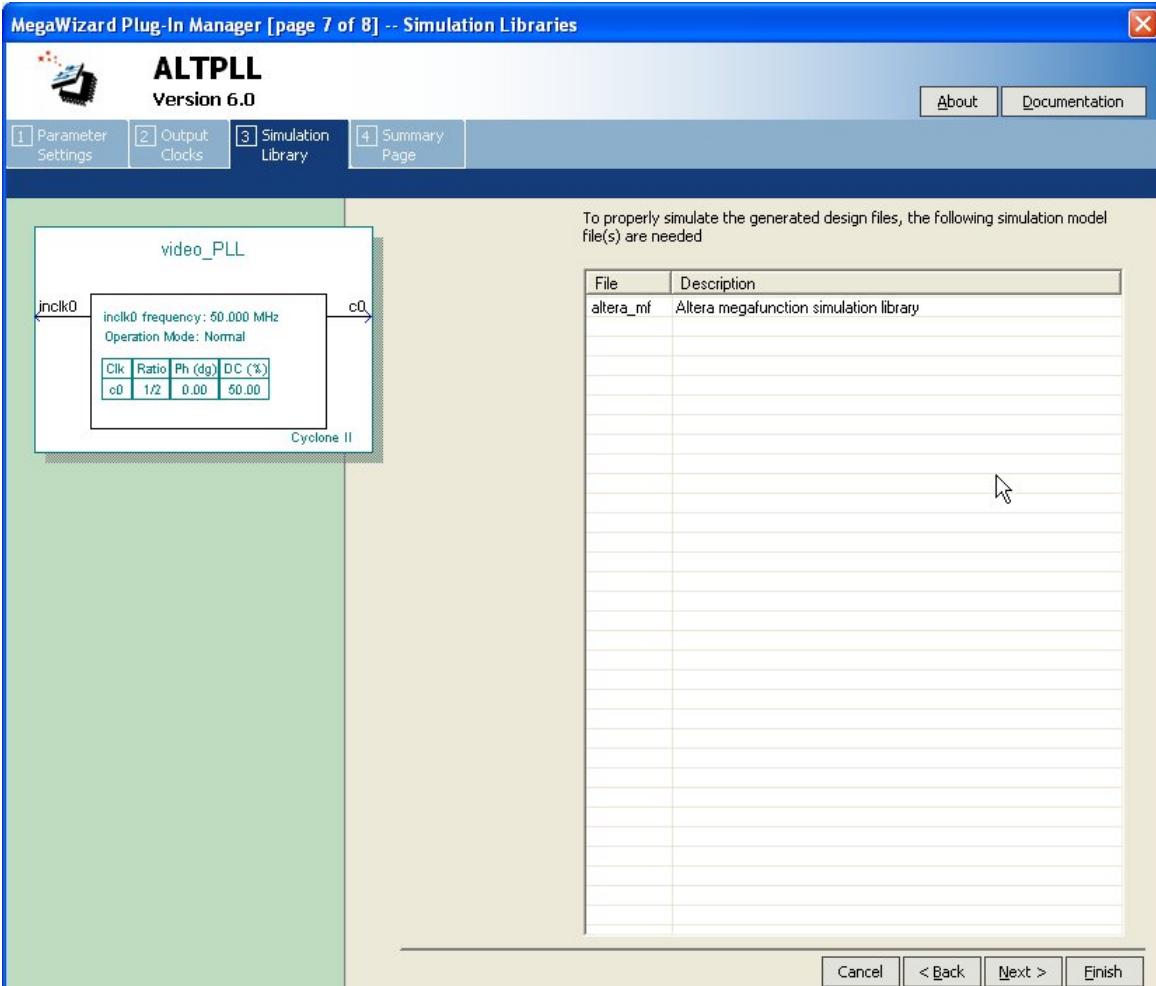
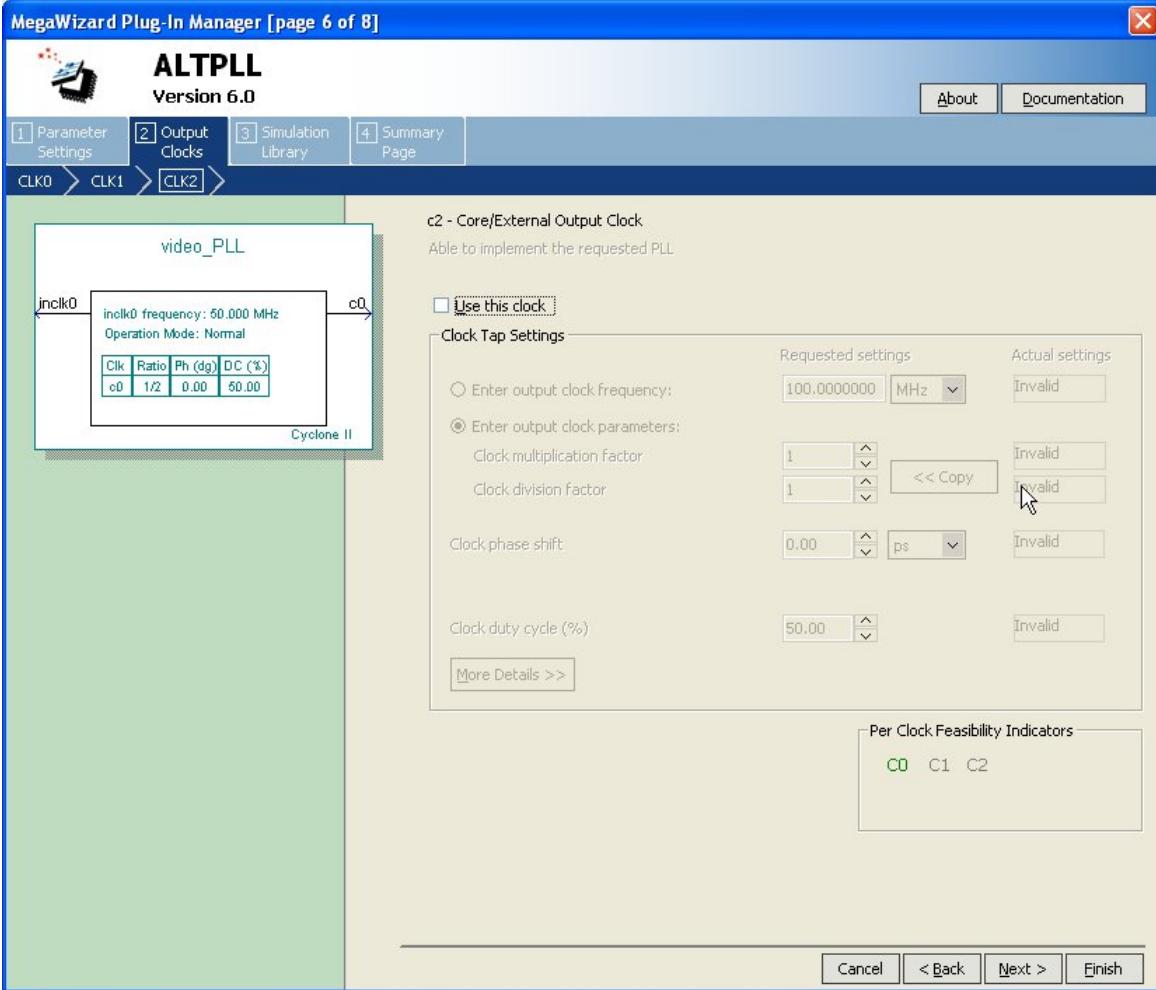
Perform the input clock switchover after  input clock cycles

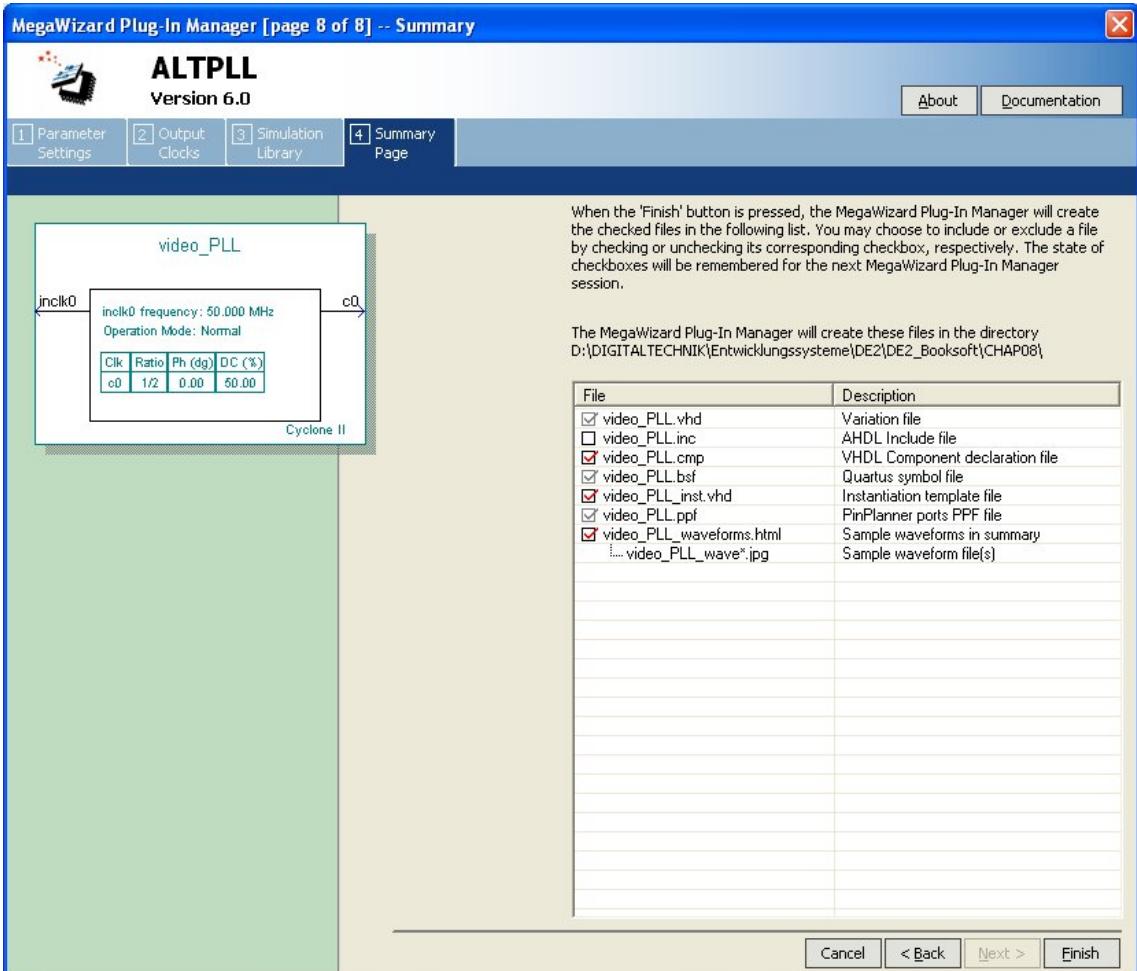
- Create an 'activeclock' output to indicate the input clock being used  
(0 inclk0 is being used/ 1 inclk1 is being used)
- Create a 'clkloss' output (indicates that input clock switchover is initiated)
- Create a 'clkbad' output for each input clock  
(0 input clock is toggling/ 1 input clock is not toggling)

Anleitung:

Cancel < Back Next > Finish







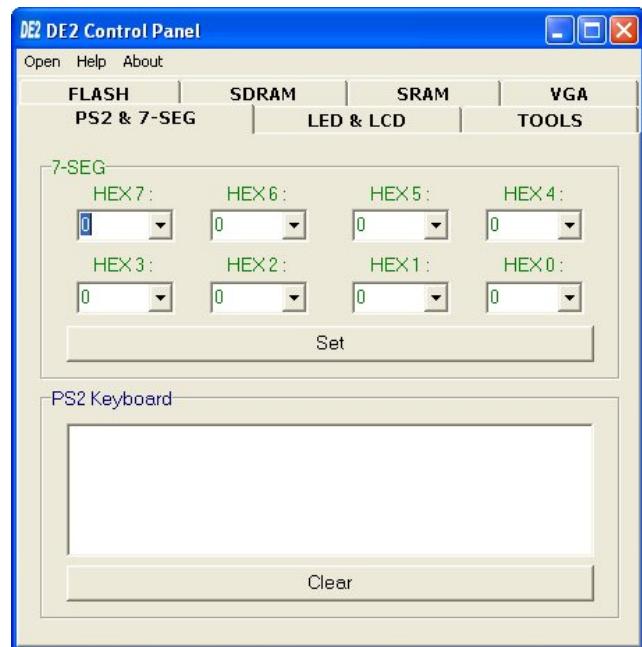
## 7. Das DE2 Control Panel

Die verschiedenen IO-Elemente des DE2 Boards lassen sich über eine Steuertafel (Control Panel) auf dem Rechner ohne spezielle Programmierung auch einzeln ansprechen. Dazu ist zunächst unter Quartus II das Assembler Programm **DE2\_USB\_API.sof** auf das Board mittels des Quartus Programmers herunterzuladen. Anschließend kann die Steuertafel durch Aufruf der Datei **DE2\_Control\_Panel.exe** gestartet werden. Um den zugehörigen Kommunikationskanal zu öffnen, ist dort in der Menüzeile **Open → Open USB Port 0** einzustellen. Dort kann der Kommunikationskanal auch wieder geschlossen werden, wenn eine neue SOF-Datei auf das DE2 Board heruntergeladen werden soll (letzteres ist anderenfalls nicht möglich, solange wie das Control Panel in Betrieb ist ).

Entsprechend der aufgeführten Karteireiter lassen sich Ein- und Ausgaben über den kontrollierenden Rechner oder eine an dem Board angeschlossene PS2 Tastatur direkt vornehmen. Darüber ist auch eine Funktionskontrolle der IO-Bausteine und des DE2 Boards möglich. Sie sollen im Folgenden kurz beschrieben werden.

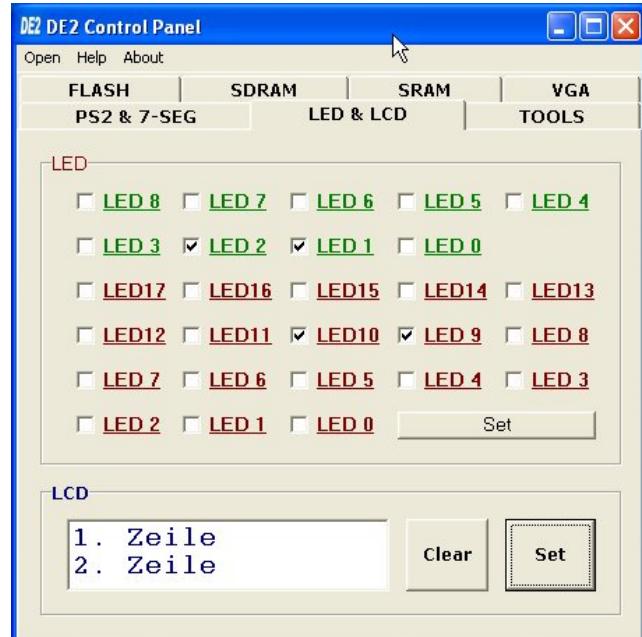
### 7.1 Reiter PS2 & 7-SEG

- Die acht 7-Segmentanzeigen können unabhängig von einander auf einen Hex-Wert eingestellt werden.
- Die Eingabe über eine Tastatur am PS2-Anschluss des DE2 Boards wird auf der Kontrolltafel ausgegeben und kann dort wieder gelöscht werden. Es ist zu beachten, dass die Tastenbelegung dabei der einer englischen Tastatur entspricht (z.B. sind y und z vertauscht).



### 7.2 Reiter LED & LCD

- Jede der 9 grünen und der 18 roten LEDs kann einzeln ein oder aus geschaltet werden.
- Auf dem LCD Display mit 2 x 16 Zeichen können über die **Rechnertastatur** Texte und Symbole ausgegeben werden. Allerdings sind nicht alle Umlaute und Sonderzeichen darstellbar bzw. werden teilweise durch taiwanische Schriftzeichen ersetzt.



### 7.3 Reiter SDRAM / SRAM

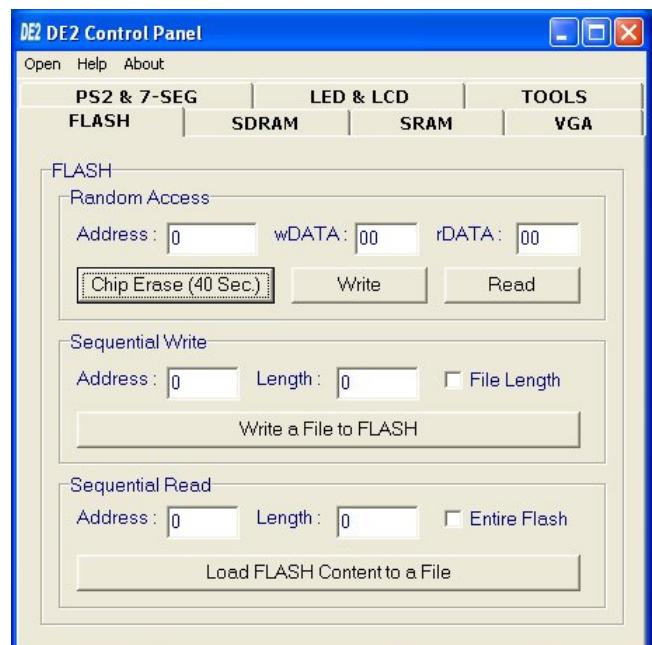
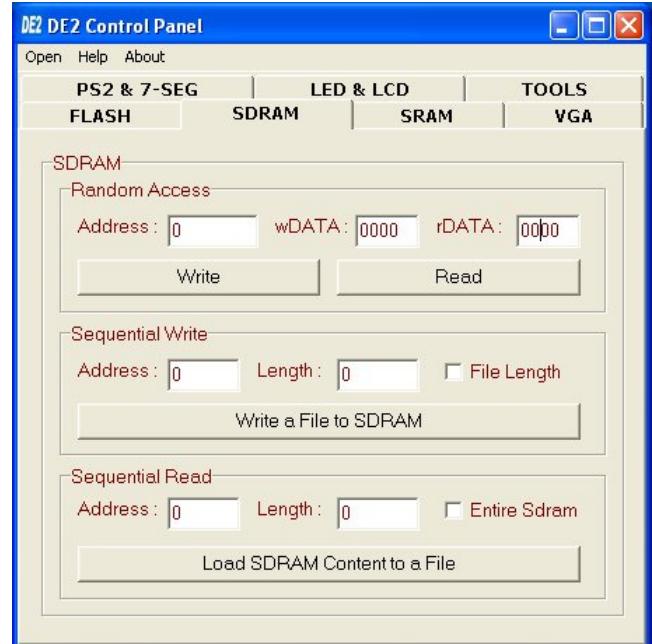
Auf dem Board befindet sich ein SDRAM Speicher mit einer Kapazität von 8 MByte und ein SRAM Speicher mit 512 KByte, organisiert in 16 Bit Wörtern. Beide Speicher können über das Control Panel auf die gleiche Weise beschrieben oder ausgelesen werden. Da es sich um externen Speicher handelt, bleiben die gespeicherten Daten dort bei der Umprogrammierung des FPGA Bausteins erhalten. Aber sie gehen bei der Ausschaltung des Boards verloren.

- Im **Random Access** Mode können Daten manuell als 4-stelliger Hex-Wert (wDATA) an der Adresse (Address) geschrieben oder von der Adresse gelesen (rDATA) werden.
- Im **Sequential Write** Mode kann der Inhalt einer Datei an definierter Stelle (Address) ganz (File Length) oder davon eine definierte Anzahl von Zeichen in den Speicher geschrieben werden. Die Daten müssen innerhalb der Datei im Binärcode (0, 1) oder im HEX-Code (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F) vorliegen. Letztere werden in Quartus II als HEX-Dateien bezeichnet. Sie können Doppelpunkte als Trennzeichen enthalten. Soll eine Datei komplett in den Speicher geschrieben werden, muss das Kästchen „File Length“ markiert werden.
- Im **Sequential read** Mode werden analog zum Schreiben der Daten (siehe oben) diese an definierter Stelle und in definierter Länge des Speichers gelesen und in eine Datei auf der Festplatte geschrieben. Ist das Kästchen „Entire SDRAM“ markiert, wird der gesamte Speicherinhalt in einer Datei abgelegt.

### 7.4 Reiter FLASH

Flash-Speicher zeichnen sich dadurch aus, dass sie ihre Information auch bei fehlender Versorgungsspannung behalten. Allerdings können diese nicht einfach überschrieben werden, wie beim SRAM / SDRAM Speicher, sondern dazu muss der Speicher von 4 MByte mit 8 Bit Wörtern vorher komplett gelöscht (**Chip Erase (40 Sec)**) werden. Der Löschvorgang ist aufgrund der dazu notwendigen hohen elektrischen Feldstärke mit einer Alterung der Speicherzellen verbunden, so dass die Anzahl der Löschvorgänge in der Praxis auf einige Tausend begrenzt ist, bevor der Speicher unbrauchbar wird. Auch kann der Speicher nur als ganzes gelöscht werden. Während des Löschvorgangs, der etwa 20 s benötigt, darf die Steuertafel nicht geschlossen werden!

Ansonsten entsprechen die Zugriffsmöglichkeiten (Schreiben / Lesen) der SDRAM / SRAM Kontrolle.



## 7.5 Reiter VGA

Die Registerkarte VGA erlaubt, auf das über die VGA Schnittstelle ausgegebene Monitorbild (siehe Deckblatt) Einfluss zu nehmen.

- Wird „Cursor Enable“ ausgewählt, kann über die X- / Y-Schieber an den Seiten des Testbildes (siehe Registerkarte rechts) ein grünes Cursor-Kreuz in die VGA-Ausgabe eingeblendet und positioniert werden.
- Es kann ein Testbild, wie nebenstehend zu sehen, ausgegeben werden, wenn „Default Image“ ausgewählt wird. Alternativ kann auch ein im SRAM abgelegtes Bild ausgegeben werden, was jedoch weitere Einstellungen erfordert (siehe Tools).

Um ein anderes Bild darzustellen, ist wie folgt vorzugehen:

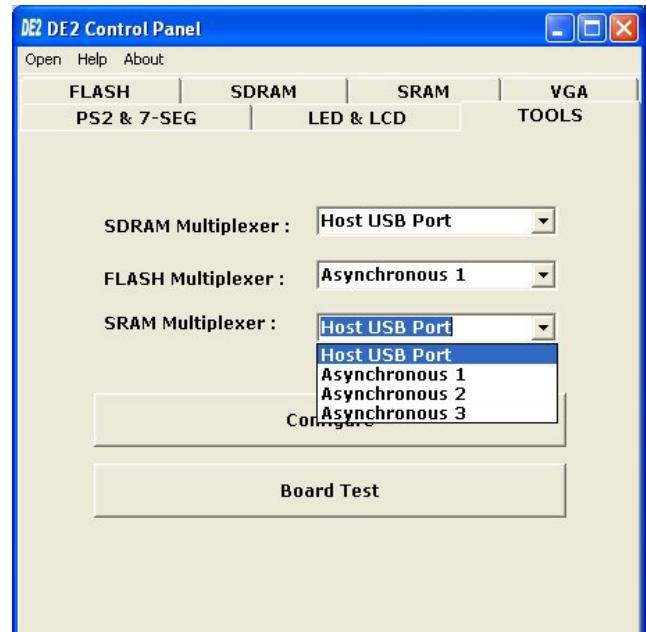
- Mit Hilfe der Registerkarte SRAM ist ein geeigneter Datensatz (z.B. die Datei **picture.dat** aus der DE2core Bibliothek in das SRAM zu laden. Es ist darauf zu achten, dass das Kästchen „File Length“ markiert ist.
- Auf der Registerkarte TOOLS ist wie im folgenden Abschnitt beschrieben „Asynchronous 1“ als Multiplexer für den SRAM Speicher auszuwählen. Nach Betätigung der „Configure“ Taste besteht anschließend ein Datenpfad zwischen SRAM Speicher und dem DAC Konverter zur VGA Ausgabe.
- Nach Abwahl des Kästchens „Default Image“ sollte das dortige Bild auf dem VGA Monitor ausgegeben werden.

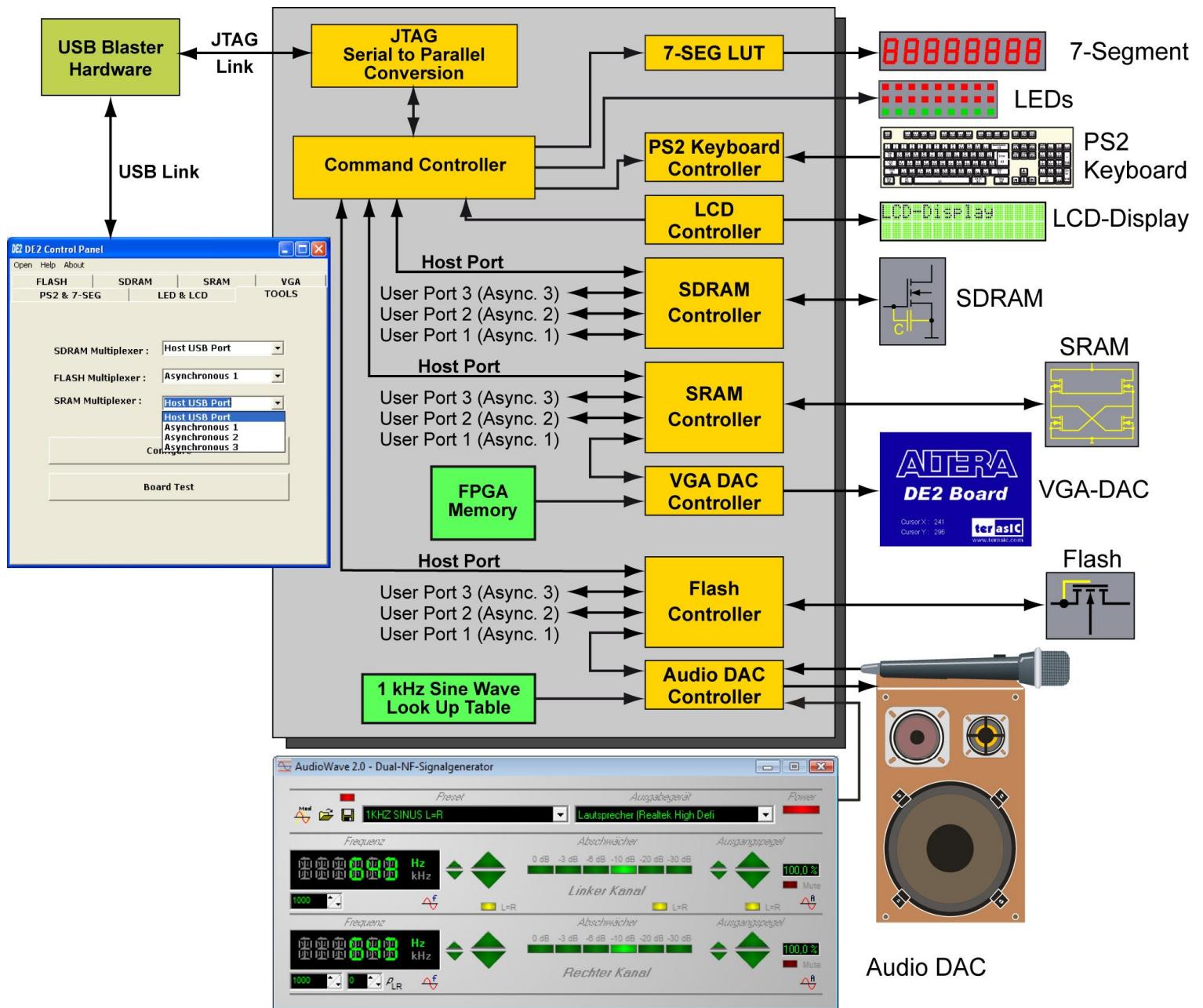


## 7.6 Reiter TOOLS

Mit Hilfe des Werkzeugs TOOLS kann die Kommunikation mit den verschiedenen Speicherbausteinen eingestellt und konfiguriert werden. Einen Überblick des Zusammenspiels der verschiedenen (Ein- / Ausgabe-) Elemente zeigt das folgende, detaillierte Blockschaltbild der Steuertafel.

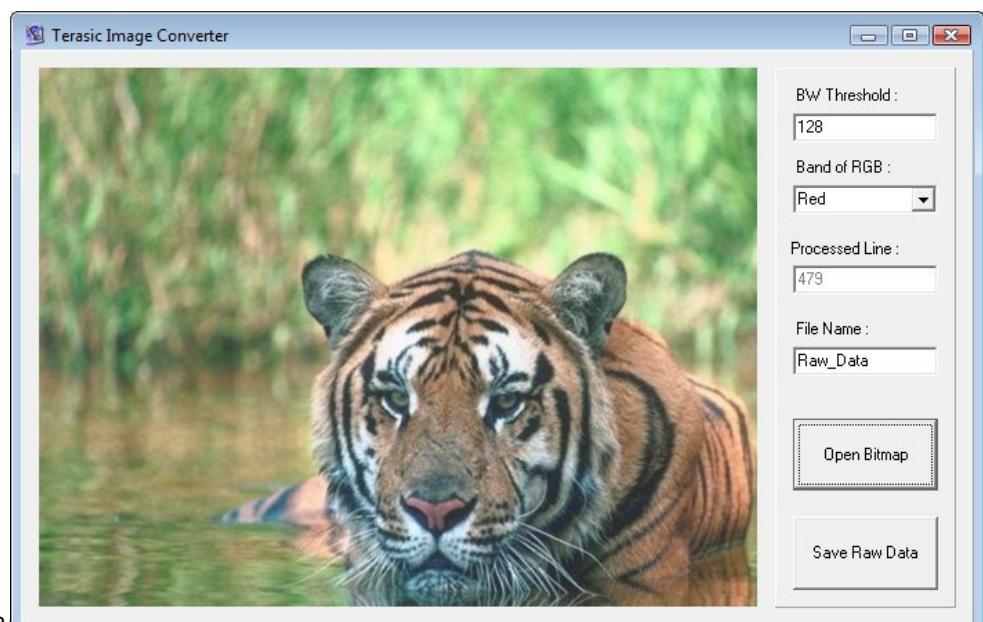
Exemplarisch soll beschrieben werden, wie mit Hilfe der Registerkarte TOOLS ein im SRAM gespeicherten Bild über den VGA Monitor dargestellt wird. Wie die Registerkarte zeigt, besitzt jeder der Speicher einen **Multiplexer**, der es ihm erlaubt, über den USB Port oder einem von drei asynchronen Ports zu kommunizieren. Wie schon oben beschrieben, ist hierüber die Ausgabe eines im SRAM gespeicherten Bildes auf dem VGA Monitor möglich, da wie das Blockschaltbild zeigt, über den User Port 1 eine Verbindung zwischen dem SRAM Controller und dem VGA DAC Controller geschaltet werden kann. Ähnlich kann eine Verbindung zwischen dem Flash Controller / Speicher und dem Audio DAC Controller geschaltet werden, um akustische Signale auszugeben.





## 7.7 Erzeugung von SRAM – Bildern

Die Erzeugung von SRAM und VGA konformer Bilder ist mit Hilfe des Terasic Image Converters „**ImgConv.exe**“ möglich. Dazu muss das Ausgangsbild in Corel PhotoPaint (Photoshop liefert eine Fehlermeldung) auf eine Größe von 640 x 480 Pixel skaliert und im Windows Bitmap Format abgespeichert werden. Nach Aufruf des Converters kann die transformierte Datei als **Graubild** in das SRAM geladen werden.



## 8. Die DE2core Library

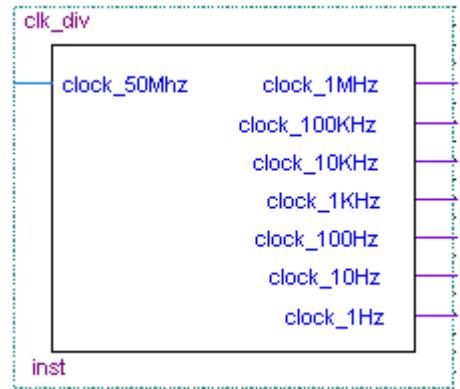
Spezielle VHDL Programme zur Nutzung des DE2 Boards sind in der DE2core Bibliothek abgelegt. Sie sollte in das Verzeichnis „quartus\libraries“ unter Quartus II kopiert werden und unter **Assignments → Settings → Libraries** als **Global Library** eingebunden werden. Diese Module dienen insbesondere zur Ansteuerung der verschiedenen Ein- und Ausgabeelemente wie Oszillatoren, LCD-Display, 7-Segment Anzeigen, Touch Panel externer Monitor, Maus und Tastatur. Es stehen die folgenden, editierbaren Module bzw. VHDL Programme zur Verfügung, die auch als Komponenten eingebunden werden können. Zum Teil greifen diese auf weitere Komponenten zurück.

- CLK\_DIV
- HEX7SEG
- DE2\_I2SOUND
- AUD\_SP
- AUD\_PS
- LCD\_DISPLAY
- LCD\_LINE
- CHAR\_ROM
- ASCII\_ROM16
- HEX\_ROM16
- SYMBOL\_ROM32
- SMILY\_ROM
- KEYBOARD
- SCAN2CHAR
- MOUSE
- VGA\_SYNC
- SRAM8
- SRAM16
- DE2\_TOUCHPANEL
- TP\_SENSOR
- TP\_HEXPAD
- TP\_SYMBOL
- TP\_RADBUT
- TP\_DISP\_HEX
- TP\_DISP\_ASCII
- POTENTIOMETER
- TP\_BAR

Eine detaillierte Beschreibung der einzelnen Module erfolgt im weiteren Teil.

## **8.1 Die Generation unterschiedlicher Taktfrequenzen**

Auf dem DE2 Board befinden sich zwei quarz-stabile Taktgeneratoren für 50 und 27 MHz. Häufig werden allerdings auch andere Frequenzen benötigt. Dafür kann grundsätzlich der PLL Wizard (Kapitel 6) benutzt werden. Für einfach dezimal gestaffelte Frequenzen wie 10 MHz, 1 MHz, ... , 1 Hz bietet sich aber auch das VHDL Modul clk\_div an.



### ***8.1.1 Das VHDL Modul clk\_div***

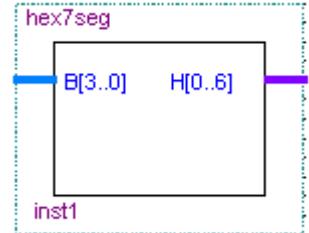
Bei einer Eingangsfrequenz von 50 MHz stellt das Modul diverse, um den Faktor 5 / 10 herunter geteilte Taktraten zur Verfügung. Bei anderen Eingangsgraten ändern sich die Ausgangstakte entsprechend. Die einfache Struktur dieses Moduls macht eine weitere Beschreibung überflüssig, und es sei auf das zugehörige VHDL Programm verwiesen, von dem hier nur die ENTITY als COMPONENT Anweisung vorgestellt wird.

### ***8.1.2 Die VHDL Komponente***

```
COMPONENT clk_div IS
    PORT (clock_50Mhz : IN STD_LOGIC;
          clock_1MHz : OUT STD_LOGIC;
          clock_100KHz : OUT STD_LOGIC;
          clock_10KHz : OUT STD_LOGIC;
          clock_1KHz : OUT STD_LOGIC;
          clock_100Hz : OUT STD_LOGIC;
          clock_10Hz : OUT STD_LOGIC;
          clock_1Hz : OUT STD_LOGIC);
END COMPONENT clk_div;
```

## **8.2 Die Sieben-Segment-Ausgabe**

Zur Ansteuerung der 8-stelligen Sieben-Segment-Ausgabe auf dem DE2 Board kann das VHDL Modul hex7seg eingesetzt werden, um Daten im Hexadezimalcode anschaulich auszugeben.



### ***8.2.1 Das VHDL Modul hex7seg***

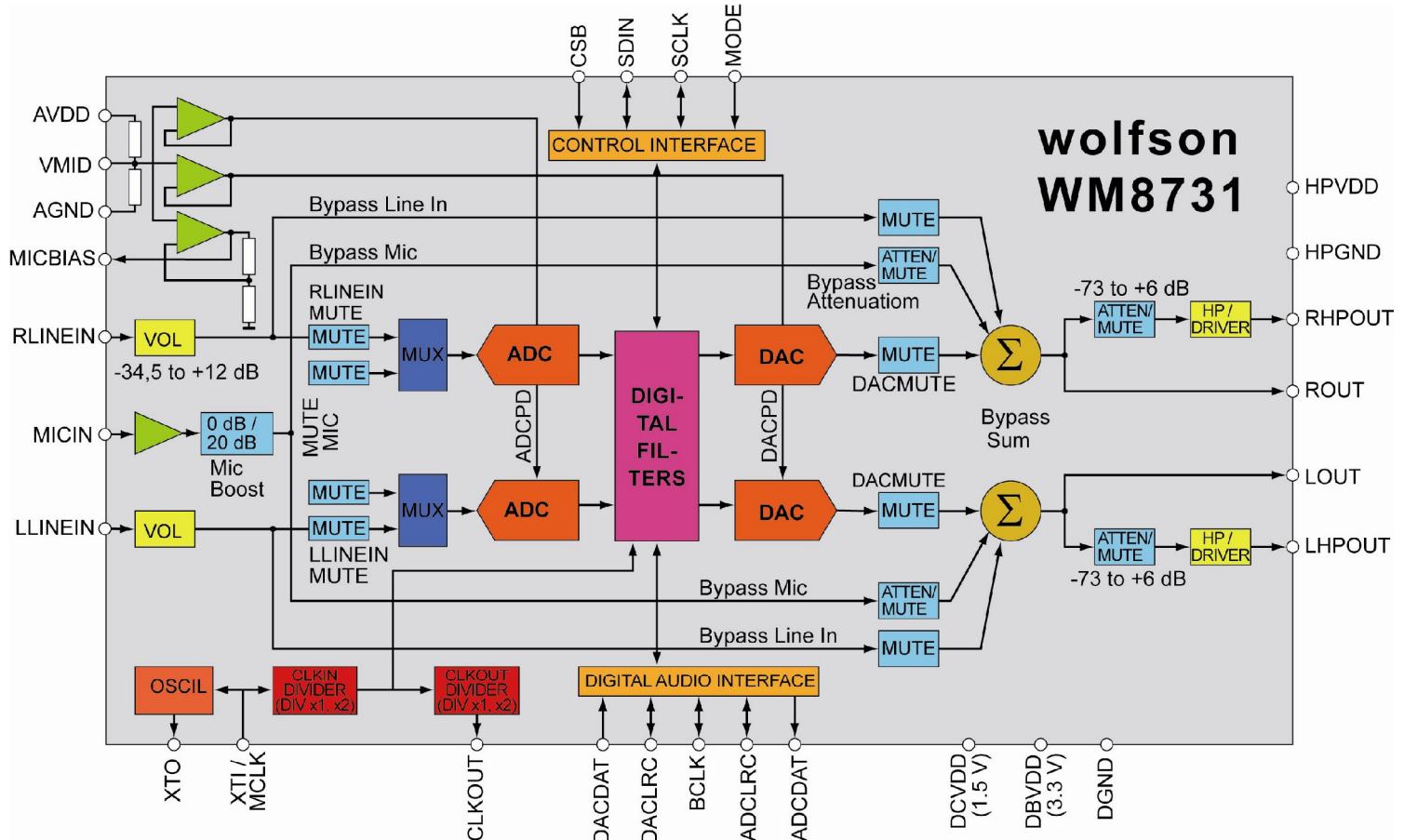
Das Modul beinhaltet einen einfachen Decoder der die Ansteuerung übernimmt. Der Decoder kann auch als Komponente in andere VHDL Programme eingebunden werden, wobei auf die Definition der Ein- und Ausgangsbussignale geachtet werden sollte.

### ***8.2.2 Die VHDL Komponente***

```
COMPONENT hex7seg IS
    PORT (B : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
          H : OUT STD_LOGIC_VECTOR(0 TO 6));
END COMPONENT hex7seg;
```

## 8.3 AUDIO CODEC

Zur Ein- und Ausgabe von Audiosignalen steht auf dem DE2 Board der universell nutzbarer Stereo Codec WM8731 der Firma Wolfson zur Verfügung. Er bietet vielfältige Möglichkeiten zur Umsetzung analoger Audiosignale in digitale Abtastwerte und umgekehrt. Es stehen verschiedene Abtastfrequenzen und Wortbreiten zur Auswahl. Der Codec bietet unterschiedliche Auswahlmöglichkeiten für den **Mikrophoneingang**, den linken und den rechten **Line-In** Kanal, die getrennt und gemeinsam genutzt werden können. Ein- und Ausgangsverstärkung lassen sich in festen Schritten verändern. Der **Frequenzgang** kann unterschiedlich angepasst werden (De-Emphas) etc. Einen groben Überblick über seine Signalverarbeitungsmöglichkeiten bietet das folgende Blockdiagramm.



Diese vielfältigen Nutzungsmöglichkeiten erfordern eine recht komplexe Ansteuerung. Von der Firma Terasic stehen dazu anpassbare **Verilog Module** zur Verfügung, die auf dem Cyclon II Baustein implementiert werden können. Um die Handhabung des Codecs relativ einfach und VHDL kompatibel zu gestalten, existieren in der DE2core Bibliothek drei VHDL Komponenten, deren Handhabung und Einstellungen im Folgenden beschrieben werden. Alle möglichen Einstellungen werden zu Anfang in einem ROM Speicher (Verilog Modul **CLOCK\_500.V**) abgelegt, womit der Codec jeweils über eine **I2C Schnittstelle** initialisiert wird. Die wichtigsten Einstellungsmöglichkeiten, auf die auch im Praktikum zugegriffen wird, seien im Folgenden beschrieben. Für alle anderen Einstellungen sei auf das Datenblatt der Firma Wolfson verwiesen.

### 8.3.1 Das Initialisierungs ROM

Der Initialisierungsspeicher ist in 10 Einheiten ( $\text{ROM}[0] \dots \text{ROM}[rs]$ ) unterteilt. In jeder Einheit wird ein 4 stelliger Hexwert abgelegt, der sich aus einem Register- (7 Bit) und einem Datenwert (9 Bit) zusammensetzt. Die jeweilige Funktion wird durch die entsprechenden Datenbits festgelegt. Standardeinstellungen (Default Werte) sind im Folgenden fett und rot gekennzeichnet. Diese Initialisierungsdaten werden im Verilog Programm **CLOCK\_500.v** (siehe das unten gezeigte Listing) definiert und übernommen, um den Audiocodec entsprechend einzustellen. Änderungen bei der Arbeitsweise der Sound Ein- und Ausgabe sind hier vorzunehmen.

Initialisierungs ROM				
ROM	Register-wert	Datenwert	Hexwert	Funktion / Einstellung
<b>ROM[0]</b>	0000 110	0 0000 0000	h0c00	Power down = aus (alles ist an)
<b>ROM[1]</b>	0000 111	0 1100 <b>10</b> 10	h0e5b	Eingangsdatenlänge (Standard)
		11		32 Bit
		10		24 Bit (Standard)
		01		20 Bit
		00		16 Bit
<b>ROM[2]</b>	0000 100	0 0001 <b>0000</b>	h0810	Auswahl des Eingangssignals (Standard)
		0		Line In Eingang
		1		Mikrofone
		0		20 dB Verstärkung des Mikrofonpegels
		1		0 dB Verstärkung des Mikrofonpegels
		0		Kein Bypass von Line In zum Ausgang
		1		Bypass von Line In zum Ausgang
		0 0001 xx0x		Kein Bypass des Mikrofons zum Ausgang
		1		Bypass des Mikrofons zum Ausgang
<b>ROM[3]</b>	0001 000	0 0000 <b>0 0000</b>	h1000	Abtastrate (Standard 48 kHz)
		0 00		48 kHz (Standard)
		1 10		32 kHz
		0 11		8 kHz
		1 11		96 kHz
<b>ROM[4]</b>	0000 000	0 0001 <b>0111</b>	h0017	Eingangsempfindlichkeit des <b>linken</b> Kanals in 1,5 dB Schritten (Standard 0 dB)
		0 000x xxxx		0 0000 = - 34 dB bis 1 1111 = + 12 dB
<b>ROM[5]</b>	0000 001	0 0001 <b>0111</b>	h0217	Eingangsempfindlichkeit des <b>rechten</b> Kanals in 1,5 dB Schritten (Standard 0 dB)
		0 000x xxxx		0 0000 = - 34 dB bis 1 1111 = + 12 dB
<b>ROM[6]</b>	0000 010	0 0 <b>111 1001</b>	h0479	Ausgangspegel des <b>linken</b> Kanals in 1 dB Schritten (Standard 0dB)
		0 0xxx xxxx		000 0000 bis 010 1111 stumm geschaltet 011 0000 = -73 dB bis 111 1111 = +6B
<b>ROM[7]</b>	0000 011	0 0 <b>111 1001</b>	h0679	Ausgangspegel des <b>rechten</b> Kanals in 1 dB Schritten (Standard 0dB)
		0 0xxx xxxx		000 0000 bis 010 1111 stumm geschaltet 011 0000 = -73 dB bis 111 1111 = +6B
<b>ROM[8]</b>	0000 101	0 0000 0000	h0a00	Filtereinstellungen (Highpass an, De-Emphasis aus)
<b>ROM[rs]</b>	0001 001	0 0000 0001	h1201	Aktivierung des Bausteins

### 8.3.2 Das Verilog Programm CLOCK\_500

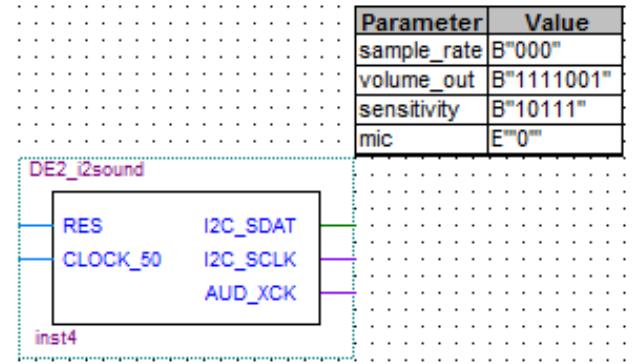
```

1 // Initialisierungsprogramm für den Audiocodec WM8731 auf dem DE2 Board
2 // Angepasst als Komponente für das Modul DE2_i2sound.vhd
3
4 `define rom_size 6'd8           //6bits dezimal mit Inhalt "001000"
5
6 module CLOCK_500 (CLOCK,
7                   CLOCK_500,
8                   DATA,
9                   ENDT,
10                  RESET,
11                  VOL_OUT,
12                  VOL_IN,
13                  MIC,
14                  SR,
15                  GO,
16                  CLOCK_2);
17     input CLOCK;                //50MHz
18     input ENDT;                 //aus I2C-Modul
19     input RESET;
20     input [6:0]VOL_OUT;         //Ausgangspegel
21     input [4:0]VOL_IN;          //Eingangsempfindlichkeit
22     input [2:0]SR;              //SampleRate Input
23     input MIC;                 //Auswahl Mic / Line In
24     output CLOCK_500;          //to I2C
25     output [23:0]DATA;          //to I2C
26     output GO;                 //to I2C
27     output CLOCK_2;            //Audio-XCK
28
29 reg [10:0]COUNTER_500; //register COUNTER_500 mit 11 bits increment mit 50MHz
30
31 wire CLOCK_500=COUNTER_500[9];      //CLOCK_500 taktet mit 48,828 kHz
32 wire CLOCK_2=COUNTER_500[1];        //CLOCK_2 taktet mit 12,5 MHz
33
34 reg [15:0]ROM[`rom_size:0];
35 reg [15:0]DATA_A;
36 reg [5:0]address;
37 wire [23:0]DATA={8'h34,DATA_A};
38
39 wire GO =((address <= `rom_size) && (ENDT==1))? COUNTER_500[10]:1;
40 always @ (posedge RESET or posedge ENDT) begin
41   if (!RESET) address=0;
42   else
43     if (address <= `rom_size) address=address+1;
44 end
45
46 always @ (posedge ENDT) begin
47 // Initialisierungs ROM:
48   ROM[0]= 16'h0c00;             //power down (alles ist an)
49   ROM[1]= 16'h0e5b;             //DSP, 24bit, LRP-2nd, master
50   ROM[2]= {12'h081,1'b0,MIC,2'b0}; //Auswahl Mic / Line In
51   ROM[3]= {8'h10,3'b0,SR[2:0],2'b0}; //Einstellung der Abtastrate
52   ROM[4]= {8'h00,3'b0,VOL_IN[4:0]}; //left linein volume
53   ROM[5]= {8'h02,3'b0,VOL_IN[4:0]}; //right linein volume
54   ROM[6]= {8'h04,1'b0,VOL_OUT[6:0]}; //sound out volume
55   ROM[7]= {8'h06,1'b0,VOL_OUT[6:0]}; //sound out volume
56   ROM[`rom_size]= 16'h1201;       //device ist aktiv
57   DATA_A=ROM[address];
58 end
59
60 always @ (posedge CLOCK ) begin
61   COUNTER_500=COUNTER_500+1;
62 end
63 endmodule

```

### 8.3.3 Das VHDL Modul DE2\_I2SOUND

Das Modul DE2\_I2SOUND.VHD umfasst im Zusammenspiel mit mehreren Verilog Komponenten die Einstellung aller Parameter, die für eine definierte Arbeitsweise notwendig sind (vergleiche die obige Tabelle). Es dient somit der **Initialisierung**, und es muss in jedem Design, welches die Sound Ein- und Ausgabemöglichkeiten des DE2 Boards nutzt, enthalten sein. Innerhalb einer Generic Anweisung lassen sich die **Abtastrate** (sample\_rate), der **Ausgangspegel** (volume\_out), die **Eingangsempfindlichkeit** (sensitivity) und das **Eingangssignal** (mic) einstellen wie unten gezeigt. Die Übernahme der Arbeitsparameter erfolgt mit einer Signalflanke am Eingang RES. Alle Ausgänge sind auf dem DE2 Board fest verdrahtet. Die Aufgaben der einzelnen Ein- und Ausgänge sind in der folgenden Tabelle zusammengefasst.



DE2_I2sound	
Anschluss	Funktion
<b>Generic:</b>	
sample_rate	Abtastrate (standardmäßig 48 kHz)
volume_out	Ausgangslautstärke (standardmäßig 0 dB)
sensitivity	Eingangsempfindlichkeit (standardmäßig 0 dB)
mic	Auswahl der Signalquelle (standardmäßig Line In)
<b>Eingänge:</b>	
RES	Bei einer Signalflanke werden die eingestellten Arbeitsparameter in den Codec übernommen.
CLOCK_50	Externes 50 MHz Taktsignal (PIN_N2)
<b>Ausgänge:</b>	
I2C_SDAT	Serieller Datenkanal des I2C Buses auf dem DE2 Board (PIN_A6)
I2C_SCLK	Taktkanal des I2C Buses auf dem DE2 Board (PIN_B6)
AUD_XCK	Systemtakt des Audio Codecs (PIN_A5)

```

1 -----
2 -- Funktion : Steuerprogramm für den WM8731 Audiocodec
3 -- Filename : DE2_i2sound.vhd
4 -- Beschreibung : Über eine Generic Anweisung lassen sich die Abtastrate,
5 --                   die Ausgangslautstärke, die Eingangsempfindlichkeit und
6 --                   das Eingangssignal (Line In oder Mic) einstellen.
7 --                   Die Einstellungen werden durch eine Signalflanke auf dem
8 --                   Eingang RES übernommen.
9 --                   Es werden die Komponenten i2c.v, keytr.v, clock_500.v
10 --                  und der PLL Baustein audio_clk.vhd benutzt.
11 --
12 -- Standard : VHDL 1993
13 -- Author : Ulrich Sandkuehler
14 -- Revision : Version 1.0 10.09.2008
15 LIBRARY ieee;
16 USE ieee.std_logic_1164.all;
17

```

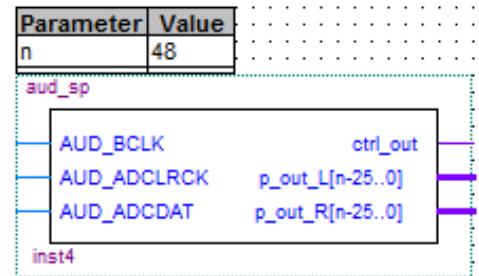
```

18 ENTITY DE2_i2sound IS
19 -- Einstellung der Abtastrate:
20   generic(sample_rate : std_logic_vector(2 downto 0)
21           := "000";      -- 48 kHz
22   --           := "110";      -- 32 kHz
23   --           := "011";      -- 8 kHz
24   --           := "111";      -- 96 kHz
25   -- Einstellung der Ausgangslautstärke zwischen Minimum und Maximum
26   -- "0000000" bis "0101111" Stummschaltung:
27     volume_out : std_logic_vector(6 downto 0)
28   --           := "0110000";    -- -73 dB Minimum
29   --           := "1111001";    -- 0 dB
30   --           := "1111111";    -- 6 dB Maximum
31   -- Einstellung der Eingangsempfindlichkeit zwischen Minimum und Maximum:
32     sensitivity : std_logic_vector(4 downto 0)
33   --           := "00000";      -- -34,5 dB Minimum
34   --           := "10111";      -- 0 dB
35   --           := "11111";      -- 12 dB Maximum
36   -- Auswahl des Mikrofons / Line In Eingangs:
37     mic : std_logic
38   --           := '0');        -- Line In Eingang
39   --           := '1');        -- Mikrofon Eingang
40
41   port(RES : IN STD_LOGIC;
42         CLOCK_50 : IN STD_LOGIC;
43         I2C_SDAT : INOUT STD_LOGIC;
44         I2C_SCLK : OUT STD_LOGIC;
45         AUD_XCK : OUT STD_LOGIC);
46 END DE2_i2sound;

```

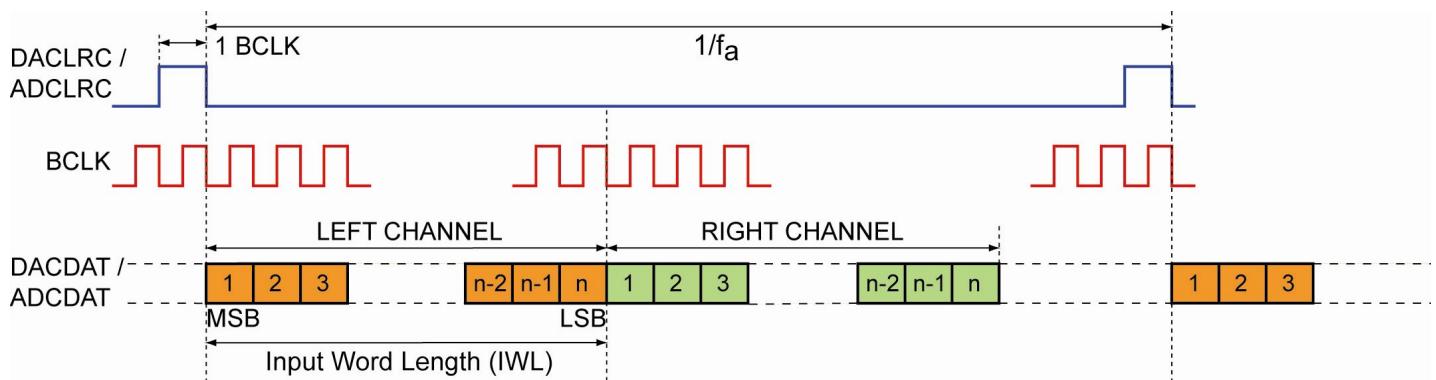
### 8.3.4 Das VHDL Modul AUD\_SP

Das Modul AUD\_SP.VHD bietet einen direkten Zugang zu den digital gewandelten Abtastwerten des linken (`p_out_L`) und des rechten (`p_out_R`) Audiokanals. Sie werden im 2-Komplement dargestellt und mit einer standardmäßigen Auflösung von je 24 Bit ( $n/2$ ) parallel ausgegeben. Sobald ein neuer Abtastwert vorliegt, wird dies durch einen Puls auf dem Ausgang `ctrl_out` angezeigt. Alle Eingänge sind fest verdrahtet. Aufgabe und Funktion aller Ein- und Ausgänge werden in der folgenden Tabelle beschrieben. Eine Änderung der Standardauflösung von 24 Bit im Initialisierungsrom erfordert auch eine entsprechende Anpassung der Busbreiten ( $n$ ) der Ausgangsparameter (Generic) dieses Moduls.



AUD_SP	
Anschluss	Funktion
<b>Generic:</b>	
<code>n</code>	Gesamtbusbreite der Ausgangskanäle (standardmäßig $n=48$ )
<b>Eingänge:</b>	
<code>AUD_BCLK</code>	Systemtakt des Audio Codecs (PIN_B4)
<code>AUD_ADCLRCK</code>	Highpegel kündigt eine neue Datensatzübertragung an (PIN_C5)
<code>AUD_ADCDAT</code>	Serielle Daten vom AD-Umsetzer des Codecs (PIN_B5)
<b>Ausgänge:</b>	
<code>ctrl_out</code>	Zeigt das Anliegen eines neuen Datenworts durch einen positiven Impuls an.
<code>P_out_L(n-25..0)</code>	$n$ Bit Abtastdatenwort des linken Kanals im 2-Komplement (signed)
<code>P_out_R(n-25..0)</code>	$n$ Bit Abtastdatenwort des rechten Kanals im 2-Komplement (signed)

Der Datenverkehr auf den Ein- und Ausgängen des Moduls wird noch einmal im folgenden Datenflussdiagramm für die Abtastfrequenz  $f_a$  dargestellt. Der Bittakt `AUD_BCLK` steuert den Ablauf.



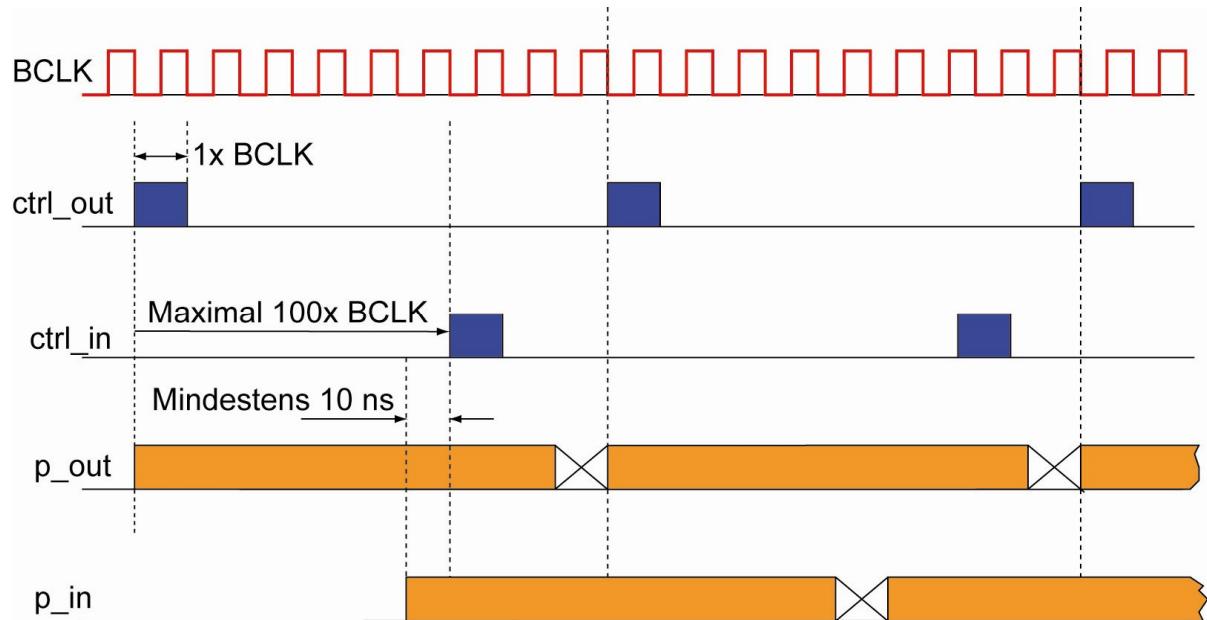
### 8.3.5 Das VHDL Modul AUD\_PS

Das Modul AUD\_PS bildet quasi das Gegenstück zum Modul AUD\_SP. Über die Eingänge (`p_out_L`) und (`p_out_R`) können digitale Datenworte parallel mit einer Standardauflösung von 24 Bit eingelesen werden. Der Audio Codec wandelt diese anschließend in analoge Signale um, die am Line Out Ausgang zur Verfügung stehen. Das Vorliegen eines neuen Datenwortes muss am Eingang `ctrl_in` durch einen positiven Impuls angezeigt werden. Aufgabe und Funktion aller Ein- und Ausgänge werden in der folgenden Tabelle beschrieben. Für andere Auflösungen muss die Busbreite über den Generic Parameter `n` angepasst werden.

Parameter	Value
<code>n</code>	48
<code>aud_ps</code>	
<code>AUD_BCLK</code>	<code>AUD_DACDAT</code>
<code>AUD_DACLRCK</code>	
<code>ctrl_in</code>	
<code>p_in_L[n-25..0]</code>	
<code>p_in_R[n-25..0]</code>	
<code>inst2</code>	

AUD_PS	
Anschluss	Funktion
<b>Generic:</b>	
<code>n</code>	Gesamtbusbreite der Eingangskanäle (standardmäßig <code>n=48</code> )
<b>Eingänge:</b>	
<code>AUD_BCLK</code>	Systemtakt des Audio Codecs (PIN_B4)
<code>AUD_DACLRCK</code>	Highpegel kündigt eine neue Datensatzübertragung an (PIN_C6)
<code>Ctrl_in</code>	Serielle Daten vom AD-Umsetzer (PIN_B5)
<code>P_in_L(n-25..0)</code>	<code>n</code> Bit Datenwort für den linken Kanal
<code>P_in_R(n-25..0)</code>	<code>n</code> Bit Datenwort für den rechten Kanal
<b>Ausgang:</b>	
<code>AUD_DACDAT</code>	Serielle Daten zum DA-Umsetzer des Codecs (PIN_A4)

Der Datenverkehr entspricht der folgenden Darstellung. Der Ablauf wird vom Bittakt `AUD_BCLK` gesteuert ähnlich wie schon im Modul AUD\_SP. Darüber hinaus ist bei dem Zusammenspiel der beiden Module AUD\_SP und AUD\_PS darauf zu achten, dass am Eingang `ctrl_in` spätestens 100 Bittakte (BCLK), nachdem der Ausgang `ctrl_out` einen HIGH-Pegel geliefert hat, ebenfalls ein High-Pegel anliegt. Dies spielt z.B. dann eine Rolle, wenn zwischen beiden Anschlüssen (`P_out_LR` und `P_in_LR`) ein digitales Filter geschaltet werden soll. Den Signalverlauf zeigt noch einmal das folgende Diagramm.



## 8.4 Das LCD Display

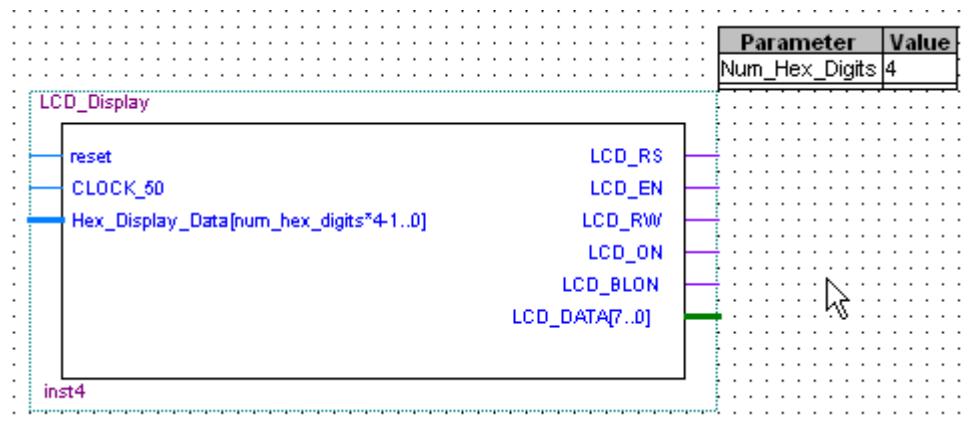
Das LCD-Display auf dem DE2 Board ermöglicht es, in zwei Zeilen bis zu 2 x 16 alphanumerische und grafische Zeichen auszugeben. Dazu besitzt das Display ein integriertes Character ROM, in dem der nebenstehend angegebene Zeichensatz in Form einzelner 5x8 Pixel-Matrizen abgelegt ist:

Jedes der insgesamt 256 möglichen Zeichen (von denen allerdings nicht alle genutzt werden) kann durch einen 8 Bit Wert (Obere HEX Zahl, Untere HEX Zahl) ausgewählt und dargestellt werden. Die entsprechenden zwei Hex-Werte zur Auswahl des Zeichens sind der nebenstehenden, grafischen Tabelle zu entnehmen. Einen Sonderfall stellen dabei Hexwerte dar, die für eine Darstellung eine feste obere Hexzahl von 0 besitzen, während für die untere Hexzahl der darzustellende Hexadezimalwert genommen wird. Dies vereinfacht die Ausgabe von numerischen Werten.

### 8.4.1 Das VHDL Modul *LCD\_Display*

Um das LCD-Display korrekt anzusteuern, ist das unten gezeigte LCD-Modul notwendig, welches in Form eines VHDL Programms alle notwendigen Steuersignale erzeugt, sowie die auszugebenden Zahlenwerte im Hexadezimalformat übernimmt.

		Obere HEX Zahl															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Untere HEX Zahl	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E
	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F



Dabei haben die Anschlüsse die folgenden Funktionen:

LCD_Display	
Anschluss	Funktion
<b>Generic:</b>	
Num_Hex_Digits	Anzahl der übertragenen Hexwerte (standardmäßig 4)
<b>Eingänge:</b>	
reset	Active low arbeitender Eingang, um die Ansteuerung des Displays neu zu initialisieren.
CLOCK_50	Externes 50 MHz Taktsignal (PIN N2)
Hex_Display_Data	Darzustellender Binärwert variabler Stellenzahl (Num_Hex_Digits*4). Anzahl und Anordnung der Hex-Werte müssen durch Editieren des VHDL Programms
Num_Hex_Digits	Anzahl der eingelesenen Hexadezimalwerte (standardmäßig = 4)
<b>Ausgänge:</b>	
LCD_RW	LCD Read/Write Select, 0 = Command, 1 = read (PIN K4)
LCD_EN	LCD Enable (PIN K3)
LCD_RS	LCD Command / Data Select, 0 = Command, 1 = Data (PIN K1)
LCD_ON	LCD Power ON / OFF (PIN L4)
LCD_BLON	LCD Back Light On / Off (nicht verdrahtet!) (PIN K2)
<b>Bidirektional:</b>	
LCD_DATA	Gemultiplexte Daten zur Ansteuerung des Displays (PIN xx)

Alle Ausgänge sind nun mit den entsprechenden Pins des Cyclon Bausteins auf dem DE2 Board fest zu verbinden. Dabei sollten die angegebenen Bezeichnungen benutzt werden, um eine „automatische“ Verdrahtung insbesondere der Ausgänge zu ermöglichen. Eine weitere Behandlung der Ausgänge erübrigt sich deshalb. Die Hintergrundbeleuchtung des Displays (LCD\_BLON) kann allerdings nicht genutzt werden, da sie auf dem DE2 Board nicht verdrahtet ist.

Standardmäßig werden über den Eingang Hex\_Display\_Data 16 Bit (4 Hexwerte) eingelesen. Um mehr oder weniger Werte einzulesen und auf dem 2 x 16 Zeichen umfassenden Display an der richtigen Position darzustellen, ist ein Eingriff in das zugehörige VHDL Programm **LCD\_Display.vhd** notwendig. Dazu ist in einer GENERIC-Anweisung zunächst die gewünschte Stellenzahl anzugeben:

```
GENERIC(Num_Hex_Digits: Integer:= 4);
```

Weitere vor oder nach den Hex-Werten auszugebenden Zeichen müssen in dem folgenden Programmausschnitt wie gewünscht definiert werden. Dazu sollte das Modul LCD\_Display aus der Bibliothek in das **Projektverzeichnis** kopiert und dort editiert werden. Jedes Zeichen ist dabei durch zwei Hex-Werte definiert, wie in der zu Anfang dargestellten Tabelle zu sehen war.

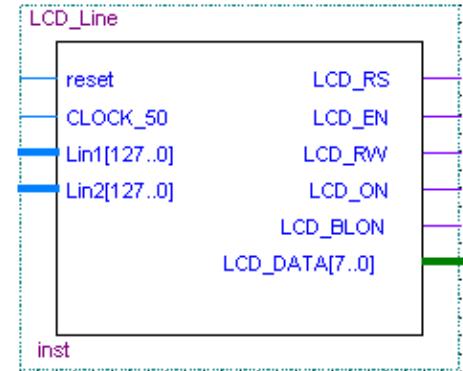
```
LCD_display_string <= (
-- ASCII hex values for LCD Display
-- Enter Live Hex Data Values from hardware here
-- LCD DISPLAYS THE FOLLOWING:
-----
--| Count=XXXX           |
--| DE2                 |
-----
-- Line 1
X"43",X"6F",X"75",X"6E",X"74",X"3D",
X"0" & Hex_Display_Data(15 DOWNTO 12),X"0" & Hex_Display_Data(11 DOWNTO 8),
X"0" & Hex_Display_Data(7 DOWNTO 4), X"0" & Hex_Display_Data(3 DOWNTO 0),
X"20",X"20",X"20",X"20",X"20",X"20",
-- Line 2
X"44",X"45",X"32",X"20",X"20",X"20",X"20",X"20",
X"20",X"20",X"20",X"20",X"20",X"20");
```

Das modifizierte Programm kann dann als Symbol, wie oben gezeigt oder als Komponente eingebunden werden:

```
COMPONENT LCD_Display IS
  GENERIC(Num_Hex_Digits: Integer:= 4);
  PORT(reset, CLOCK_50 : IN STD_LOGIC;
         Hex_Display_Data : IN STD_LOGIC_VECTOR((Num_Hex_Digits*4)-1 DOWNTO 0);
         LCD_RS, LCD_EN, LCD_RW : OUT STD_LOGIC;
         LCD_ON, LCD_BLON : OUT STD_LOGIC;
         LCD_DATA : INOUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END COMPONENT LCD_Display;
```

#### 8.4.2 Das VHDL Modul LCD\_Line

Bei dem Modul LCD\_Line handelt es sich um eine modifizierte Version des Moduls LCD\_Display. Während es beim letzteren notwendig war, das Programm LCD\_Display.vhd entsprechend der gewünschten Ausgabe zu modifizieren, werden hier die auszugebenen Zeichen entsprechend der zu Anfang aufgeföhrten ASCII-Tabelle direkt über die Eingänge Lin1 und Lin2 getrennt für die beiden Zeilen übergeben. 8 Bit definieren dabei jeweils ein Symbol, so dass eine Zeile mit 16 Zeichen 128 Bits erfordert.



Die entsprechende Komponente des Programms LCD\_Line.vhd lautet dabei wie folgt:

```
COMPONENT LCD_Line IS
  PORT(reset, CLOCK_50 : IN STD_LOGIC;
        Lin1, Lin2 : IN STD_LOGIC_VECTOR(127 DOWNTO 0);
        LCD_RS, LCD_EN, LCD_RW : OUT STD_LOGIC;
        LCD_ON, LCD_BLON : OUT STD_LOGIC;
        LCD_DATA : INOUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END COMPONENT LCD_Line;
```

Alle Ausgänge des LCD\_Line Moduls entsprechen dem Modul LCD\_Display und können der zugehörigen Tabelle entnommen werden.

#### 8.5 Einsatz und Arbeitsweise von Character ROMs

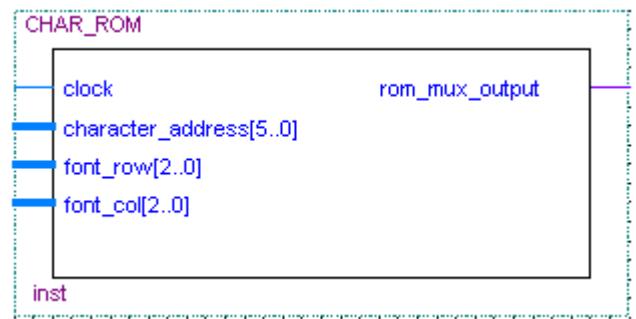
Im Gegensatz zur 7-Segmentausgabe ist bei es bei einer pixelorientierten Darstellung möglich, sehr unterschiedliche Zeichen klar und gut erkennbar darzustellen. Dies ist allerdings mit einem sehr viel höheren Aufwand verbunden, da schon bei einer groben Auflösung mit nur 8 x 8 Punkten nun 64 Bit pro Zeichen codiert werden müssten, was den Speicherbedarf unnötig erhöht.

Deshalb hat man spezielle Speicherbausteine sogenannte „Zeichengeneratoren“ oder Character ROMs entwickelt, in denen alle Muster für einen ganzen Zeichensatz fest abgelegt sind. Für 64 verschiedene Zeichen kommt man so mit 6 Bit pro Zeichen ( $2^6 = 64$ ) anstatt 64 Bit aus, wenn man jedes Pixel einzeln ansteuern müsste. Über diese 6 Bit Adresse können dann die z.B. auf einem Monitor darzustellenden Symbole aus dem Character ROM ausgelesen werden.

Die DE2core Bibliothek enthält Character Roms unterschiedlicher Auflösung (8 x 8 Pixel bzw. 16 x 16 Pixel) mit modifizierbaren Zeichensätzen unterschiedlicher Länge. Für spezielle Anwendungen können kleine Grafiken, Icons oder Symbole abgelegt werden. Ihre Arbeitsweise und ihr Einsatz sollen im Folgenden zunächst anhand des Moduls **CHAR\_ROM** genauer beschrieben werden.

### 8.5.1 Das VHDL Modul CHAR\_ROM

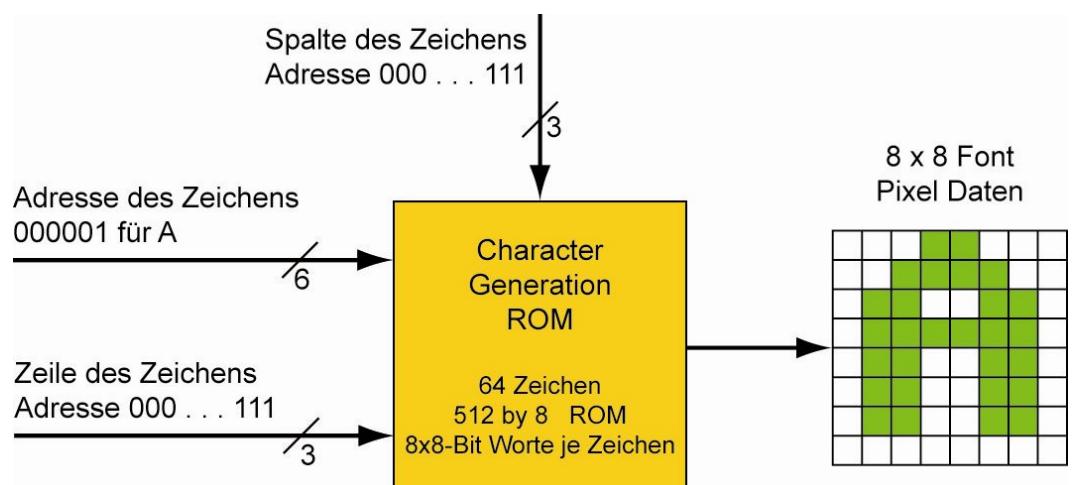
Das nebenstehende Modul Char\_ROM enthält 64 verschiedene Zeichen. Jedes Zeichen wird dabei in Form einer 8 x 8 Matrix abgespeichert, wie dies die unten stehende Grafik zeigt. Die Ansteueradressen für die einzelnen Zeichen sind dort gleichzeitig im 2-stelligen Oktalcode (MSO, LSO) angegeben.



Die Hexadezimalzeichen 0 ... F tauchen unter der oktalen Adresse 60 ... 77 zum Teil noch einmal auf. Sie können dadurch sehr einfach aufgerufen werden, indem vor einem 4 Bit Hexwert die Bits „11“ vorgehängt werden, so dass sich das 6-Bit Adresswort für den Binärwert xxxx entsprechend 11xxxx für den Zeichengenerator ergibt.

Die Arbeitsweise eines Character ROMs für das oben gezeigte Beispiel lässt sich am unten stehenden Bild beschreiben. Eines der 64 abgespeicherten Zeichen, hier der Buchstabe A, kann über den 6 Bit Bus „Adresse des Zeichens“ (`character_address[5..0]`) ausgewählt werden. Die einzelnen Pixelzeilen (`font_row`) und Pixelspalten (`font_col`) des Zeichens können dann über je 3 Bit breite Busse ausgewählt und an einem Ausgang (`rom_mux_out`) Pixel für Pixel entsprechend dem Taktsignal `CLOCK` zur Verfügung gestellt werden. Zusammengefasst ergibt sich für 64 Zeichen a' 8 x 8 Bit somit ein Speicherbedarf von 4 Kbit. Bei höherer Auflösung oder einem umfangreichen Zeichensatz ist der Speicherplatzbedarf entsprechend größer.

	Oberer Oktalwert (MSO)							
	0 (000)	1 (001)	2 (010)	3 (011)	4 (100)	5 (101)	6 (110)	7 (111)
U	0 (000)	C	H	P	X		(	0
n	1 (001)	A	I	Q	V	!	)	1
t	2 (010)	B	J	R	Z	"	#	2
e	3 (011)	C	K	S	C	#	+	3
r	4 (100)	D	L	T	+	S	,	4
O	5 (101)	E	M	U	U	X	-	5
k	6 (110)	F	N	V	↑	B	.	6
t	7 (111)	G	O	W	*	/	?	F



Alle benötigten Anschlüsse werden in der folgenden Tabelle noch einmal zusammengefasst:

CHAR_ROM	
Anschluss	Funktion
<b>Eingänge:</b>	
clock	Pixeltakt (Auslesetakt), z.B. das VGA_CLK Signal des VGA_SYNC Moduls
character_address[5..0]	Auswahl eines der 64 Zeichen (6 Bit Bus)
font_row[2..0]	Pixelauswahl Zeile (3 Bit Bus)
font_col[2..0]	Pixelauswahl Spalte (3 Bit Bus)
<b>Ausgang:</b>	
rom_mux_output	Bitwert (0 oder 1) des ausgewählten Pixels

### 8.5.2 Das VHDL Programm

Die Realisation eines Character ROM Bausteins lässt sich am besten an dem zugehörigen VHDL Programm nachvollziehen. Das unten aufgeführte Programmlisting enthält im Wesentlichen die Komponente **altsyncram**, welche Bestandteil der Library of Parameterized Modules (**LPM**) ist. Dieser Festwertspeicher kann mit dem MegaWizard Plug-In Manager erstellt und parametrisiert werden. Er enthält später alle Zeichen und Pixel für das Character Rom. Die wichtigsten seiner Designgrößen sind in der unten stehenden Tabelle zusammengefasst. Weitere Angaben lassen sich in der Online-Hilfe zu „altsyncram“ finden. Die Werte beziehen sich auf das oben beschriebene CHAR\_ROM. Werte für das unten beschriebene ASCII\_ROM16 sind in Klammern angegeben. Weitere Modifikationen zur Speicherung von Symbolen in unterschiedlicher Anzahl und Auflösung sind möglich.

altsyncram		
Parameter	Wert	Funktion
clock_enable_input_a	"BYPASS"	Enable Funktionen für Ein- und Ausgänge, die bei CYCLON II Bausteinen nicht weiter berücksichtigt wird.
clock_enable_output_a	"BYPASS"	
init_file	"TCGROM.MIF" ("ARIAL16.MIF")	Dateiname für die CHAR_ROM Font Daten mit allen Pixelwerten
intended_device_family	"Cyclone II"	Baustein Familie
lpm_hint	"ENABLE_RUNTIME_MOD=YES, INSTANCE_NAME=ROM"	Altera-spezif. Parameter zur Aktivierung des In-System Memory Content Editors
lpm_type	"altsyncram",	Entity Name für die LPM
numwords_a	512 (2048)	Anzahl der Worte im Speicher (Zeichenanzahl x Pixelhöhe)
operation_mode	"ROM"	Speichertyp
outdata_aclr_a	"NONE"	Async. Löschen der Ausgangsdaten
outdata_reg_a	"UNREGISTERED"	Register Funktion des Ausgangs
widthad_a	9 (11)	Adressbreite des Speichers
width_a	8 (16)	Wortlänge in Bit (Zeichenbreite)
width_bytlena_a	1	Breite des Enable Eingangs
Anschluss	Funktion	Ergänzungen
clock_0	Pixel Takt	Takt des Speichers
address_a	Zeichen- und Zeilenadresse aus widthad_a Bits	character_address & font_row
Q_a	Ausgangswort mit width_a Bits	

In dem obigen altsyncram ist standardmäßig der „**In-System Memory Content Editor**“ aktiviert (Wert des Parameters `lpm_hint`), so dass der Speicherinhalt des ROMs im laufenden Betrieb über die JTAG Schnittstelle editiert und modifiziert werden kann (vergl. Kapitel 1.9.4 der Vorlesung). Wichtigstes Element ist die sogenannte MIF-Datei (`init_file`), die unter Quartus II die Daten für einen zu programmierenden ROM-Speicher (altsyncram) enthält.

### 8.5.3 Die Struktur der MIF-Datei

Unter Quartus II wird der Dateninhalt von einem ROM Speicher, der auf dem FPGA arrangiert werden soll, in einer editierbaren ASCII Datei vom Typ \*.mif untergebracht. Der Anfang und das Ende der hier benutzten MIF-Datei **tcgrom.mif** ist unten dargestellt:

```

depth = 512;
Width = 8;
Address_radix = oct;
Data_radix = bin;
% Character Generator ROM Data %
Content
Begin
000 : 00111100 ; % **** %
001 : 01100110 ; % ** **
002 : 01101110 ; % ** ***
003 : 01101110 ; % ** ***
004 : 01100000 ; % **
005 : 01100010 ; % ** *
006 : 00111100 ; % **** %
007 : 00000000 ; %

010 : 00011000 ; % **
011 : 00111100 ; % **** %
012 : 01100110 ; % ** **
013 : 01111110 ; % ***** %
014 : 01100110 ; % ** **
015 : 01100110 ; % ** **
016 : 01100110 ; % ** **
017 : 00000000 ; %

020 : 01111100 ; % **** %
021 : 01100110 ; % ** **
022 : 01100110 ; % ** **
023 : 01111100 ; % **** %
024 : 01100110 ; % ** **
025 : 01100110 ; % ** **
026 : 01111100 ; % **** %
027 : 00000000 ; %

.
.

770 : 01111110 ; % ***** %
771 : 01100000 ; % **
772 : 01100000 ; % **
773 : 01111000 ; % **** %
774 : 01100000 ; % **
775 : 01100000 ; % **
776 : 01100000 ; % **
777 : 00000000 ; %

End;

```

Diese wie auch alle anderen Größen aus der Tabelle **altsyncram** sind gegebenenfalls zu modifizieren, wenn die Anzahl oder die Auflösung der Zeichen geändert wird, wie dies z.B. im Character ROM **ASCII\_ROM16** der Fall ist. Entsprechendes gilt natürlich auch für die Busbreiten in der Tabelle CHAR\_ROM. Die Adressen und die Daten können dabei in den folgenden Formaten angegeben werden. Lehrzeilen sind erlaubt. Kommentare bis zum Zeilenende können durch das Zeichen % gekennzeichnet werden.

Bezeichnung	Bedeutung
<b>bin</b>	binär
<b>oct</b>	oktal
<b>hex</b>	hexadezimal
<b>dec</b>	dezimal mit Vorzeichen
<b>uns</b>	dezimal ohne Vorzeichen

Die Entity aus dem folgenden VHDL Programm kann natürlich auch als Komponente in ein anderes Programm eingebaut werden.

```

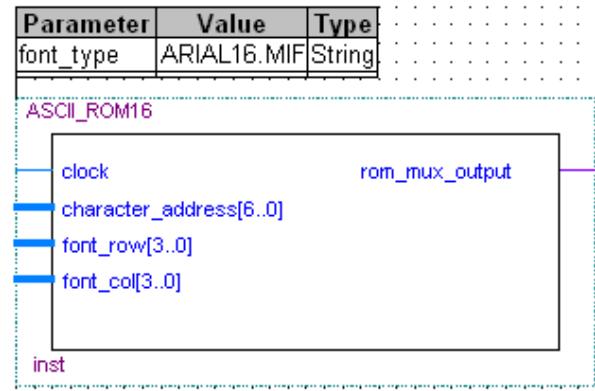
1  -----
2  -- Funktion : Character Rom fuer 64 Zeichen
3  -- Filename : CHAR_ROM.vhd
4  -- Beschreibung : 8x8 Pixel, 64 Zeichen CHAR_ROM mit Aktivierung des
5  --                      In-System Memory Content Editors.
6  --                      Benötigt wird die Datei tcdrom.mif
7  -- Standard : VHDL 1993
8  -- Author   : Ulrich Sandkuehler
9  -- Revision : Version 1.2 29.03.2009
10 -----
11 library IEEE;
12 use IEEE.STD_LOGIC_1164.all;
13 use IEEE.STD_LOGIC_UNSIGNED.all;
14
15 ENTITY CHAR_ROM IS
16     PORT(clock          : IN STD_LOGIC;
17           character_address : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
18           font_row, font_col  : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
19           rom_mux_output    : OUT STD_LOGIC);
20 END CHAR_ROM;
21
22 ARCHITECTURE BEHAVIOR OF CHAR_ROM IS
23
24 COMPONENT altsyncram
25     GENERIC (clock_enable_input_a  : STRING;
26               clock_enable_output_a : STRING;
27               init_file            : STRING; -- Fontsatz für CHAR_ROM
28               intended_device_family: STRING;
29               lpm_hint              : STRING;      -- In-System Memory Content Editors
30               lpm_type              : STRING;
31               numwords_a            : NATURAL;   -- Zeichenhöhe in Pixel x Zeichenanzahl
32               operation_mode        : STRING;
33               outdata_aclr_a       : STRING;
34               outdata_reg_a         : STRING;
35               widthad_a             : NATURAL;   -- Adressbreite für "numwords_a"
36               width_a               : NATURAL;   -- Zeichenbreite in Pixel
37               width_bytenea_a       : NATURAL);
38     PORT (clock0 : IN STD_LOGIC ;
39            address_a : IN STD_LOGIC_VECTOR (8 DOWNTO 0);
40            q_a : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
41 END COMPONENT;
42
43 SIGNAL rom_data: STD_LOGIC_VECTOR(7 DOWNTO 0);
44 BEGIN
45
46 rom_mux_output <= rom_data ((CONV_INTEGER(NOT font_col(2 downto 0)))); 
47
48 altsyncram_component : altsyncram
49 GENERIC MAP(clock_enable_input_a => "BYPASS",
50               clock_enable_output_a => "BYPASS",
51               init_file            => "TCGROM.MIF", -- Fontsatz für CHAR_ROM
52               intended_device_family=> "Cyclone II",
53               lpm_hint              => -- Aktivierung In-System Memory Content Editors
54                           "ENABLE_RUNTIME_MOD=YES, INSTANCE_NAME=ROM",
55               lpm_type              => "altsyncram",
56               numwords_a            => 512, -- Zeichenhöhe in Pixel x Zeichenanzahl
57               operation_mode        => "ROM",
58               outdata_aclr_a       => "NONE",
59               outdata_reg_a         => "UNREGISTERED",
60               widthad_a             => 9, -- Adressbreite für "numwords_a"
61               width_a               => 8, -- Zeichenbreite in Pixel
62               width_bytenea_a       => 1)
63 PORT MAP(clock0 => clock,
64           address_a => character_address & font_row,
65           q_a => rom_data);
66
67 END BEHAVIOR;

```

#### 8.5.4 Das VHDL Modul ASCII\_ROM16

Während der Zeichengenerator Char\_ROM auf möglichst wenig Speicherbedarf (4 kBit) ausgelegt war, zeichnet sich das Modul ASCII\_ROM16 durch einen auf 128 Symbole erweiterten 7 Bit ASCII Zeichensatz aus, wie er unten aufgeführt ist. Darin sind die sonst üblichen Steuerzeichen (weißer Hintergrund) durch Sonderzeichen (Umlaute, griechische Buchstaben) ersetzt worden. Alle anderen Zeichen behalten ihren üblichen Bit-Wert wie angegeben. Die grün hinterlegten Zeichen existieren als Typ CHARACTER auch in VHDL. Jedes der darstellbaren 128 Zeichen kann

später durch die Ansteuerung mit diesem, angegeben 7 Bit Wert ausgewählt werden. Hexadezimalwerte lassen sich einfach durch "000xxxx" aufrufen. Alle Zeichen werden im Character ROM mit einer Auflösung von 16 x 16 Pixel abgespeichert. Dies erhöht zwar den Speicherbedarf auf insgesamt 32 KBit, verbessert aber die Qualität der Darstellung deutlich. Auch ist es möglich, den Schriftsatz selbst zu variieren, indem man z.B. unterschiedliche Fontsätze verwendet. Der gewünschte Schriftsatz kann über eine GENERIC Anweisung definiert werden. Für die Typen **Arial**, **Times Roman** und **Courier New** existieren dazu die MIF-Dateien **arial16.mif**, **timesroman16.mif** und **courier16.mif**. Die Konstruktion und Variation solcher Zeichensätze ist unter Photoshop relativ einfach möglich, soll hier aber nicht weiter behandelt werden. Für die Ein- und Ausgänge dieses Zeichengenerators gilt das schon für das Modul CHAR\_ROM gesagte. Alle Signalbusse sind hier allerdings um 1 Bit breiter. Sehen Sie sich auch das entsprechende VHDL Programm ASCII\_ROM16.vhd an!



#### 8.5.5 Der modifizierte ASCII Code

ASCII Tabelle (ISO 7-Bit)								
LSB \ MSB	000 (0)	001 (1)	010 (2)	011 (3)	100 (4)	101 (5)	110 (6)	111 (7)
0000 (0)	0	ä		0	@	P	'	p
0001 (1)	1	ö	!	1	A	Q	a	q
0010 (2)	2	ü	"	2	B	R	b	r
0011 (3)	3	Ä	#	3	C	S	c	s
0100 (4)	4	Ö	\$	4	D	T	d	t
0101 (5)	5	Ü	%	5	E	U	e	u
0110 (6)	6	€	&	6	F	V	f	v
0111 (7)	7	α	'	7	G	W	g	w
1000 (8)	8	β	(	8	H	X	h	x
1001 (9)	9	γ	)	9	I	Y	i	y
1010 (A)	A	δ	*	:	J	Z	j	z
1011 (B)	B	Δ	+	;	K	[	k	{
1100 (C)	C	λ	'	<	L	\	l	
1101 (D)	D	π	-	=	M	]	m	}
1110 (E)	E	Σ	.	>	N	^	n	~
1111 (F)	F	Ω	/	?	O	_	o	μ

### 8.5.6 Die Komponente des ASCII\_ROM16 Moduls

Die Deklaration der Ein- und Ausgänge des ASCII\_ROM16 Moduls lassen sich aus der zugehörigen Komponente entnehmen.

```
COMPONENT ASCII_ROM16 IS
    GENERIC(font_type : STRING := "ARIAL16.MIF");
    PORT(clock : IN STD_LOGIC;
          character_address : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
          font_row, font_col : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
          rom_mux_output : OUT STD_LOGIC);
END COMPONENT ASCII_ROM16;
```

### 8.5.7 Spezielle Zeichen Generatoren

Weiter Zeichengeneratoren, die für spezielle Aufgaben entwickelt wurden, lassen sich in der DE2core Bibliothek finden. Einige Beispiele beschreibt die folgende Tabelle. Alle Ein- und Ausgänge entsprechen den Modulen CHAR\_ROM und ASCII\_ROM16, wobei die Busbreiten der Anschlüsse der jeweiligen Zeichengröße / Auflösung und -anzahl entsprechend angepasst werden müssen.

Generatoren für Sonderzeichen		
Name	Zeichenzahl x Auflösung	Zeichen / Symbole / Bemerkungen
HEX_ROM16	16 x 16	Enthält nur die HEX-Symbole (1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F) des ASCII_ROM16
SYMBOL_ROM32	16 x 32	😊 😞 😟 ☎ 🎗️ 🎵 🎣 🎪 🎩 🎫 🎧 🎤 🎭 🎮
SMILY_ROM	2 x 64	😊 😞

```
component HEX_ROM16 is      -- KOMPONENTE ZUR GENERIERUNG DER HEX ZEICHEN:
    port(clock : in std_logic;           -- Pixeltakt
          character_address : in std_logic_vector(3 downto 0); -- HEX Zeichen
          font_row, font_col : in std_logic_vector(3 downto 0); -- Pixeladress
          rom_mux_output : out std_logic);                      -- Pixelwert (0/1)
end component HEX_ROM16;
```

```
COMPONENT SYMBOL_ROM32 IS
    PORT(clock : IN STD_LOGIC;
          character_address : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
          font_row, font_col : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
          rom_mux_output : OUT STD_LOGIC);
END COMPONENT SYMBOL_ROM32;
```

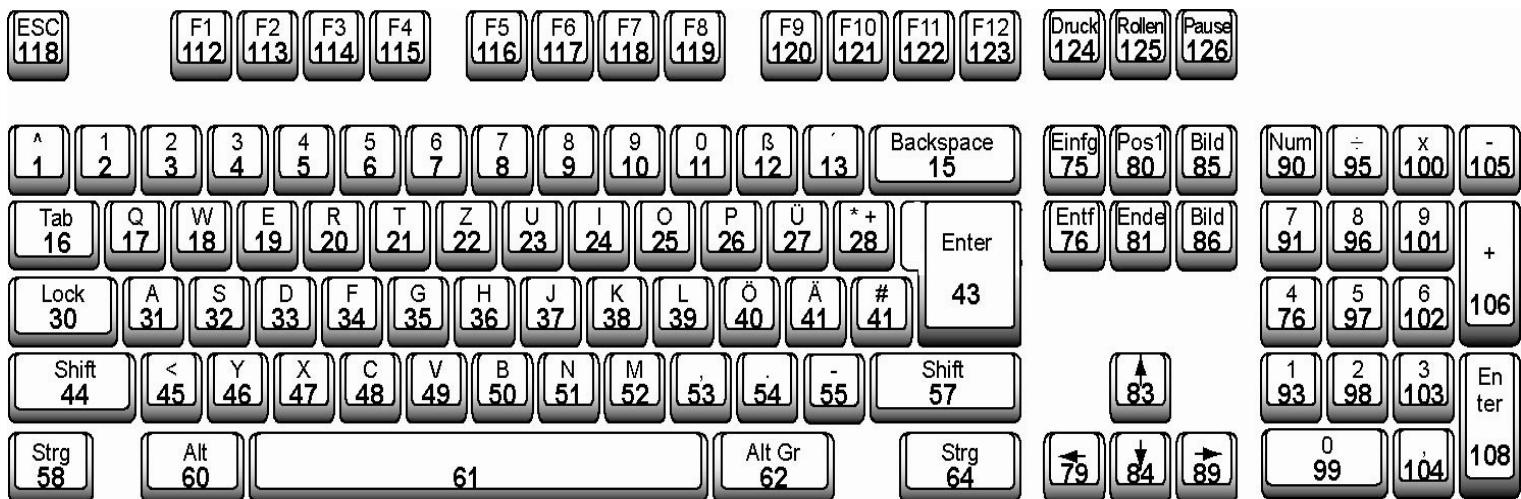
```
component SMILY_ROM is
    generic(N : positive := 6;
            DATEI : string := "SMILY.mif");
    port(CLOCK : in std_logic;
          symb_address : in std_logic;
          font_row, font_col : in std_logic_vector(N-1 downto 0);
          rom_mux_output : out std_logic);
end component SMILY_ROM;
```

## 8.6 Die PS2 Schnittstelle

Auf dem DE2 Board steht eine PS2-Schnittstelle zur Verfügung, an die eine Mouse oder eine Tastatur angeschlossen werden können. Die Schnittstelle umfasst 6 separate Leitungen, von denen aber neben den zwei Anschlüssen zur Stromversorgung nur zwei weitere als Daten- und Takteleitung genutzt werden und direkt mit dem FPGA-Baustein verbunden sind: `PS2_CLK` ist an `PIN_D26` angeschlossen und `PS2_DAT` an `PIN_C24`. Beide Leitungen arbeiten normalerweise bidirektional. Alle Daten werden dabei seriell übertragen.

### 8.6.1 Die Tastatur

Das KEYBOARD Modul wertet nur die Daten einer Tastatur an der PS2 Schnittstelle selbst aus, und verzichtet auf einen bidirektionalen Betrieb, um die Funktion und die Umsetzung der Daten hier möglichst einfach zu gestalten.



### 8.6.2 Der Scancode

Alle Taster einer (PS2) Tastatur (Keyboard) sind elektrisch in Form einer Matrix angeordnet, deren Elemente oben durch eine Nummer gekennzeichnet sind, und liefern bei ihrer Betätigung einen individuellen **Scancode**. Während beim hier nicht genutzten bidirektionalen Betrieb verschiedene Steuersignale eine Auswahl unterschiedlicher Scancodes z.B. für eine deutschsprachige oder englischsprachige Tastatur erlauben, wird hier nur der beim Einschalten von der Tastatur vorgegebene Standardscancode benutzt, wie er in der nachfolgenden Tabelle aufgelistet ist. Ein Sonderfall stellt dabei der Ziffernblock dar, dessen Scancode durch die NumLock Taste umgeschaltet wird. Davon wird hier allerdings kein Gebrauch gemacht, und es wird nur der ursprüngliche Code gezeigt und ausgewertet. Es existieren nicht für alle aufgeführten Nummern auch reale Tasten und damit auch keine Scancodes (grau unterlegt).

Jeder Scan Code besteht aus zwei Codefolgen. Beim Drücken einer Taste wird ein sogenannter **Make Code** generiert. Beim Loslassen wird anschließend ein **Break Code** generiert. Er entspricht dem Make Code mit vorangestellter `xF0` Bitfolge, wie in der Tabelle zu sehen ist. Bei gedrückter Taste wird der Make Code periodisch wiederholt. Anhand des Scan Codes kann erkannt werden, ob mehr als eine Taste betätigt wurde, und in welcher Reihenfolge sie wieder gelöst werden. Aufgabe des später vorgestellten KEYBOARD Moduls ist es, diese von der Tastatur gelieferten Codes aus dem seriellen Bitstrom herauszulesen und auszugeben.

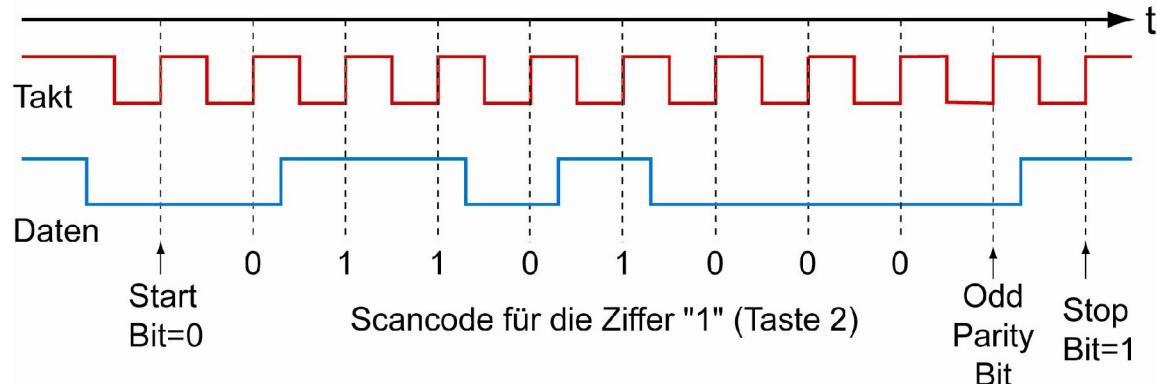
Scancode für eine deutschsprachige PS/2 Tastatur											
Key #	Char-acter	Make Code	Brake Code	Key #	Char-acter	Make Code	Brake Code	Key #	Char-acter	Make Code	Brake Code
1	^	0E	F0 0E	43	Enter	5A	F0 5A	85	Bild↑	E0 7D	E0 F0 7D
2	1	16	F0 16	44	Shift	12	F0 12	86	Bild↓	E0 7A	E0 F0 7A
3	2	1E	F0 1E	45	<	61	F0 61	87			
4	3	26	F0 26	46	Y	1A	F0 1A	88			
5	4	25	F0 25	47	X	22	F0 22	89	→	E0 74	E0 F0 74
6	5	2E	F0 2E	48	C	21	F0 21	90	Num	77	F0 77
7	6	36	F0 36	49	V	2A	F0 2A	91	7 Pos 1	6C	F0 6C
8	7	3D	F0 3D	50	B	32	F0 32	92		6B	F0 6B
9	8	3E	F0 3E	51	N	31	F0 31	93	1 Ende	69	F0 69
10	9	46	F0 46	52	M	3A	F0 3A	94			
11	0	45	F0 45	53	,	41	F0 41	95	÷	4A	F0 4A
12	ß	4E	F0 4E	54	.	49	F0 49	96	8↑	75	F0 75
13	'	55	F0 55	55	-	4A	F0 4A	97	5	73	F0 73
14				56				98	2↓	72	F0 72
15	Back	66	F0 66	57	Shift	59	F0 59	99	0 Einfo	70	F0 70
16	Tab	0D	F0 0D	58	Strg	14	F0 14	100	x	7C	F0 7C
17	Q	15	F0 15	59				101	9 Bild↑	7D	F0 7D
18	W	1D	F0 1D	60	Alt	11	F0 11	102	6→	74	F0 74
19	E	24	F0 24	61	Leer	29	F0 29	103	3 Bild↓	7A	F0 7A
20	R	2D	F0 2D	62	Alt Gr	E0 11	E0 F0 11	104	Entf	71	F0 71
21	T	2C	F0 2C	63				105	-	7B	F0 7B
22	Z	35	F0 35	64	Strg	14	F0 14	106	+	79	F0 79
23	U	3C	F0 3C	65				107			
24	I	43	F0 43	66				108	Enter	5A	F0 5A
25	O	44	F0 44	67				109			
26	P	4D	F0 4D	68				110	Esc	76	F0 76
27	Ü	54	F0 54	69				111			
28	+	5B	F0 5B	70				112	F1	05	F0 05
29		5D	F0 5D	71				113	F2	06	F0 06
30	Lock			72				114	F3	04	F0 04
31	A	1C	F0 1C	73				115	F4	0C	F0 0C
32	S	1B	F0 1B	74				116	F5	03	F0 03
33	D	23	F0 23	75	Einfo	E0 70	E0 F0 70	117	F6	0B	F0 0B
34	F	2B	F0 2B	76	Entf	E0 71	E0 F0 71	118	F7	83	F0 83
35	G	34	F0 34	77				119	F8	0A	F0 0A
36	H	33	F0 33	78				120	F9	01	F0 01
37	J	3B	F0 3B	79	←	E0 6B	E0 F0 6B	121	F10	09	F0 09
38	K	42	F0 42	80	Pos 1	E0 80	E0 F0 80	122	F11	78	F0 78
39	L	4B	F0 4B	81	Ende	E0 69	E0 F0 69	123	F12	07	F0 07
40	Ö	4C	F0 4C	82				124	Druck	E0 12	E0 F0 7C
41	Ä	52	F0 52	83	↑	E0 75	E0 F0 75	125	Rollen	E0 7C	E0 F0 12
42	#	5D	F0 5D	84	↓	E0 72	E0 F0 72	126	Pause	E0 7E	E0 F0 7E

### 8.6.3 Das Übertragungsprotokoll

Da die Codeübertragung seriell erfolgt, müssen die Scan Codes in ein Übertragungsprotokoll eingebettet werden. Dieses Protokoll besteht aus den folgenden Komponenten:

- Ein Start Bit (logisch 0),
- 8 Datenbits beginnend mit dem LSB,
- Ein Paritätsbit für ungerade Parität,
- Ein Stop Bit (logisch 1).

Eine denkbare Bitfolge zeigt das folgende Diagramm.



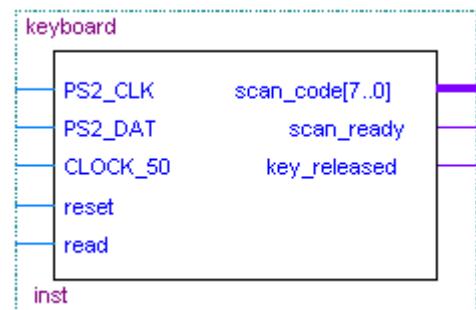
Eine Übertragung läuft dabei in den folgenden Schritten ab:

1. Die Tastatur überprüft den inaktiven Zustand der Daten- und Takteleitung anhand des logisch 1 Signals auf beiden Leitungen.
2. Die Takteleitung wird für 35 µs auf logisch 0 gesetzt.
3. Der Takt startet mit einer Periodenlänge von 70 µs. Mit der ersten Flanke wird das Startbit übertragen.
4. Es folgen die ersten 8 Datenbits des Make bzw. des Brake Codes.
5. Im Anschluss an die Daten wird das Odd Parity Bit und das Stop Bit übertragen.

Dieses Muster wiederholt sich, bis alle Daten übertragen wurden. Danach werden die Leitungen wieder inaktiv geschaltet.

### 8.6.4 Das VHDL Modul KEYBOARD

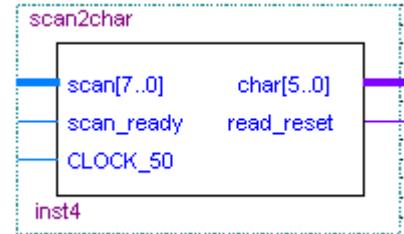
Das nebenstehende Modul KEYBOARD konvertiert die seriellen Daten einer Tastatur in einen parallelen 8 Bit Scan Code. Zur Unterdrückung eventueller externer Störeinflüsse bei der Datenübertragung enthält es ein einfaches Filter in Form eines 8 Bit Schieberegisters, um kurzzeitige Störimpulse auszublenden. Die übertragenden Daten werden dabei mit 50 MHz abgetastet (`CLOCK_50`) und über den Eingang `PS2_DAT` in das Schieberegister eingelesen. Nur wenn alle 8 aufeinander folgenden Abtastwerte identisch sind, wird ein neuer 8 Bit Datenwert generiert und an den Ausgang `scan_code[7..0]` angelegt. Das Anliegen eines neuen Wertes wird durch ein log. 1 Signal am Ausgang `scan_ready` signalisiert. Bevor ein weiterer Scancode ausgegeben werden kann, muss dieses Signal zurückgesetzt werden. Dazu ist ein log. 1 Impuls am Eingang `read` notwendig. Der Ausgang `key_released` signalisiert durch einen log. 1 Impuls das Loslassen einer Taste. Dieses Signal kann genutzt werden, um die Anzahl von Tastenbefätigungen zu erfassen. Die Aufgaben der einzelnen Anschlüsse sind in der folgenden Tabelle noch einmal aufgelistet.



KEYBOARD	
Anschluss	Funktion
<b>Eingänge:</b>	
PS2_CLK	Von der Tastatur vorgegebener Takt (PIN D26)
PS2_DAT	Serieller Datenstrom von der Tastatur (PIN C24)
CLOCK_50	50 MHz Systemtakt (PIN N2)
reset	Rücksetzen des Moduls (active low)
read	Rücksetzen der Ausgangssignale scan_ready und key_released
<b>Ausgang:</b>	
Scan_code[7..0]	Ausgabe des parallelen 8 Bit Scancodes
scan_ready	Signalisiert das Vorliegen eines neuen Scancodes
key_released	Signalisiert das Loslassen einer Taste (log. 1 Impuls)

### 8.6.5 Das VHDL Modul SCAN2CHAR

Um das Symbol einer betätigten Taste darzustellen, muss der generierte und vom Modul KEYBOARD gelieferte Scancode für ein CharROM aufbereitet werden. Diese Aufgabe wird von dem Modul SCAN2CHAR übernommen. Es liest den generierten Scancode am Eingang **scan[7..0]** ein und liefert entsprechend dem unten dargestellten Zeichenvorrat am Ausgang **char[5..0]** den 6 Bit Adresscode zur Ansteuerung des **CHAR\_ROM** Moduls. Da dieser sehr begrenzt ist, werden auch nur einfache Tastenbetätigungen ausgewertet, und Kombinationen mit Umschalttasten wie Shift, Strg, Alt, Alt Gr nicht ausgewertet. Tastenbetätigungen, denen kein einfaches Zeichen zugeordnet ist, werden als Leerzeichen (Space) ausgegeben und dargestellt. Dies vereinfacht die Funktionsweise des Moduls deutlich. Somit sind nur die unten aufgelisteten Zeichen darstellbar. Nicht aufgeführte Zeichen sind nicht darstellbar, auch wenn sie im CHAR\_ROM Modul existieren.



Außer der beschriebenen Codeumsetzung übernimmt das Modul auch die Generierung des Rücksetzsignals für das **scan\_ready** Signal, welches daraus abgeleitet wird. Darüber hinaus benötigt das Modul noch den boardeigenen Systemtakt von 50 MHz (**CLOCK\_50**).

SCAN2CHAR	
Anschluss	Funktion
<b>Eingänge:</b>	
scan[7..0]	Vom Ausgang des KEYBOARD Moduls einzulesender Scancode
scan_ready	Signalisierung eines neuen Scancodes durch das KEYBOARD Moduls
CLOCK_50	50 MHz Systemtakt (PIN N2)
<b>Ausgang:</b>	
char[5..0]	Ausgabe des Tastenzeichens (Adresscode für CHAR_ROM)
read_reset	Rücksetzimpuls für das scan_ready Signal

Von SCAN2CHAR umgesetzte Zeichen											
Make Code	Zeichen	Adresse [oktal]	Make Code	Zeichen	Adresse [oktal]	Make Code	Zeichen	Adresse [oktal]	Make Code	Zeichen	Adresse [oktal]
		00	4D	P	20		space	40	45	0	60
1C	A	01	15	Q	21			41	16	1	61
32	B	02	2D	R	22			42	1E	2	62
48	C	03	1B	S	23	5D	#	43	26	3	63
23	D	04	2C	T	24			44	25	4	64
24	E	05	3C	U	25			45	2E	5	65
2B	F	06	2A	V	26			46	36	6	66
34	G	07	1D	W	27			47	3D	7	67
33	H	10	47	X	30			50	3E	8	70
43	I	11	46	Y	31			51	46	9	71
2B	J	12	35	Z	32			52			72
42	K	13			33	5B	+	53			73
4B	L	14			34	41	,	54			74
3A	M	15			35	4A	-	55			75
31	N	16			36	49	.	56			76
44	O	17	45	←	37			57			77

## 8.7 Der Betrieb einer Maus an der PS2 Schnittstelle

Die PS2-Schnittstelle ermöglicht neben dem oben beschriebenen Anschluss einer Tastatur auch den Betrieb einer schnurgebundenen Computer Maus. Während allerdings die Tastatur über unidirektionale Schnittstellen betrieben werden konnte, erfordert der Mausbetrieb **bidirektionale Tristate-Schnittstellen** für Takt und Daten.

### 8.7.1 Die Betriebsweise der Maus

Die Maus muss vor dem Einschalten mit dem DE2 Board verbunden sein. Im regulären Betrieb (Screaming Mode) überträgt die Maus ihre Daten in drei aufeinander folgenden Bytes, deren Zusammensetzung in der folgenden Tabelle beschrieben wird.

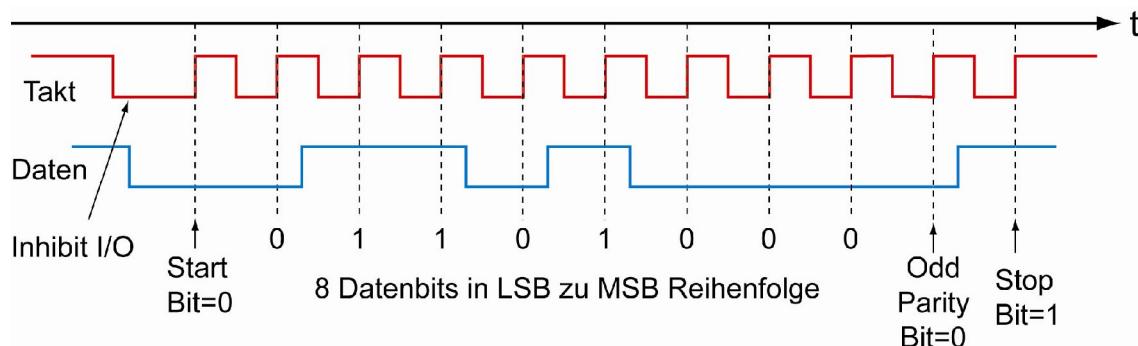
Maus Datenformate								
Bit	7	6	5	4	3	2	1	0
Byte 1	Yc	Xc	Ys	Xs	1	M	R	L
Byte 2	x7	x6	x5	x4	x3	x2	x1	x0
Byte 3	y7	y6	y5	y4	y3	y2	y1	y0

**L** Statusbit der linken Taste (1 gedrückt)  
**M** Statusbit der mittleren Taste (1 gedrückt)  
**R** Statusbit der rechten Taste (1 gedrückt)  
**x7...x0** zurückgelegte X-Distanz im 2-Komplement  
**y7...y0** zurückgelegte Y-Distanz im 2-Komplement  
**Xc** Überlauf für die X-Daten (1 Überlauf)  
**Yc** Überlauf für die Y-Daten (1 Überlauf)  
**Xs** Vorzeichen der X-Daten (1 negativ)  
**Ys** Vorzeichen der Y-Daten (1 negativ)

Die zurückgelegten Distanzen (Maus Positionen) und die Statusbits der Maustasten werden dabei 100 Mal pro Sekunde aktualisiert und übertragen.

### 8.7.2 Das Übertragungsprotokoll

Der Signalverlauf auf den Anschlussleitungen ähnelt dem Protokoll für die Tastaturansteuerung. Der zeitliche Ablauf der Datenübertragung kann dort nachgelesen werden.



### 8.7.3 Das VHDL Modul MOUSE

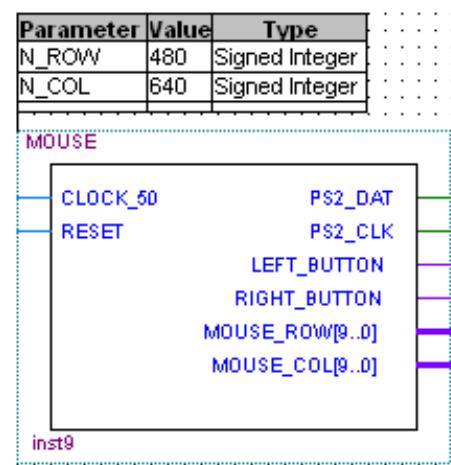
Das eigentliche VHDL Programm zur Ansteuerung der Maus ähnelt der Tastaturansteuerung, ist aber deutlich komplexer. Auch hier wird ein Taktfilter zur Unterdrückung von Störungen auf der Verbindungsschnur, ein Serien-Parallel und ein Parallel-Serien Wandler bestehend aus zwei Schieberegistern benutzt. Eine Zustandsmaschine steuert zusätzlich die verschiedenen Betriebsarten. Für weitere Einzelheiten sei auf das entsprechende VHDL Programm verwiesen.

Zum Betrieb des MOUSE Moduls ist ein Systemtakt von 50 MHz erforderlich. Das optionale **RESET** Signal arbeitet aktiv high. Bei nicht Gebrauch kann es auf Masse gelegt werden. Die Betätigung der linken bzw. der rechten Maustaste liefert ein log. 1 Signal an den zugehörigen Ausgängen **LEFT\_BUTTON** und **RIGHT\_BUTTON**.

Die aktuelle Position des Maus-Zeigers steht in XY-Koordinaten an den Ausgängen **MOUSE\_ROW** und **MOUSE\_COL** mit einer Auflösung von 10 Bit zur Verfügung.

Dies begrenzt die Koordinatenwerte auf 0 – 1023. Der maximale Bereich kann separat für Zeile und Spalte über die Generic Parameter **N\_ROW** und **N\_COL** eingestellt werden. Standardmäßig ist eine VGA Auflösung definiert. Beim Einschalten (Initialisierung, Reset) befindet sich der Cursor in der Mitte des Darstellungsbereichs (**N\_ROW/2**, **N\_COL/2**).

Eine Zusammenfassung aller Ausgangs- und Eingangssignale liefert die folgende Tabelle:



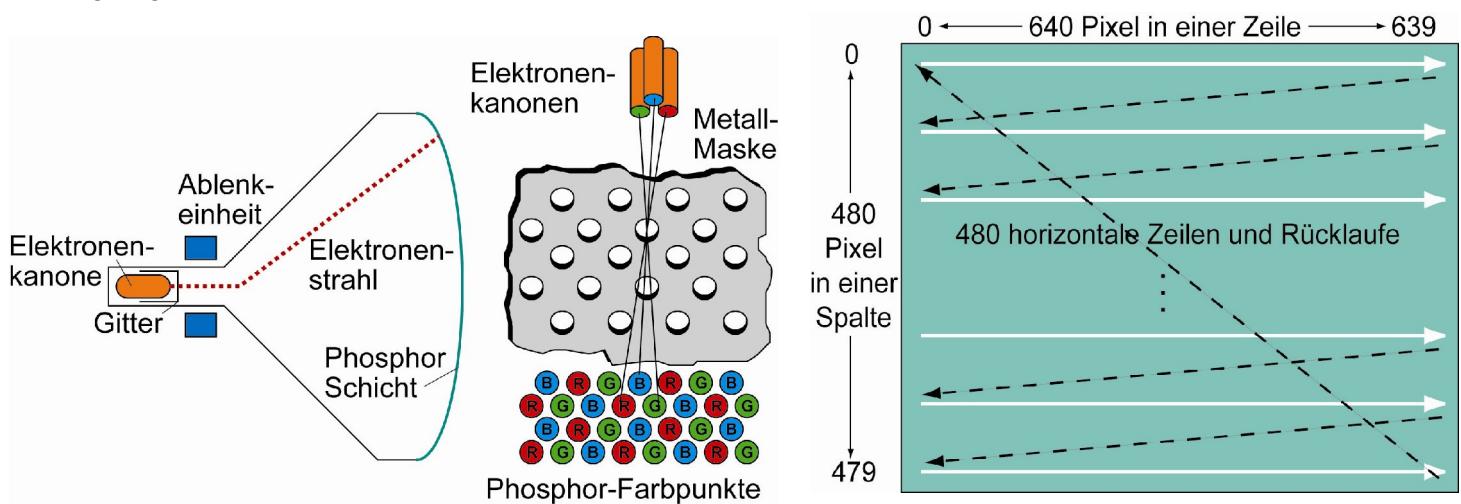
MOUSE	
Anschluss	Funktion
<b>Generic Parameter</b>	
<b>N_ROW</b>	Maximale Zeilenzahl < 1024
<b>N_COL</b>	Maximale Spaltenzahl < 1024
<b>Eingänge:</b>	
reset	Active high arbeitender Eingang, um die Ansteuerung der Maus neu zu initialisieren.
CLOCK_50	Externes 50 MHz Taktsignal (PIN N2)
<b>Ausgänge:</b>	
LEFT_BUTTON	Linke Maustaste, gedrückt = 1, nicht gedrückt = 0
RIGHT_BUTTON	Rechte Maustaste, gedrückt = 1, nicht gedrückt = 0
MOUSE_ROW[9..0]	Cursor Position in Y-Richtung (10 Bit Auflösung)
MOUSE_COL[9..0]	Cursor Position in X-Richtung (10 Bit Auflösung)
<b>Bidirektional:</b>	
PS2_DAT	PS2 Anschluss (PIN C24)
PS2_CLK	PS2 Anschluss (PIN D26)

## 8.8 Der Betrieb von externen VGA Monitoren

Das DE2 Board besitzt einen VGA-Anschluss, um eine Ausgabe auf einem Monitor zu ermöglichen. Dazu muss der FPGA Baustein entsprechend kompatible Monitor-Signale erzeugen. Für die Generierung dieser Signale wird das Modul VGA\_SYNC benutzt, hinter dem sich ein VHDL Programm zur Erzeugung der Synchronisationssignale verbirgt. Das Programm kann für unterschiedliche Monitorauflösungen modifiziert werden. Dazu ist es notwendig, die Arbeitsweise und den Aufbau des VHDL Programms zu verstehen.

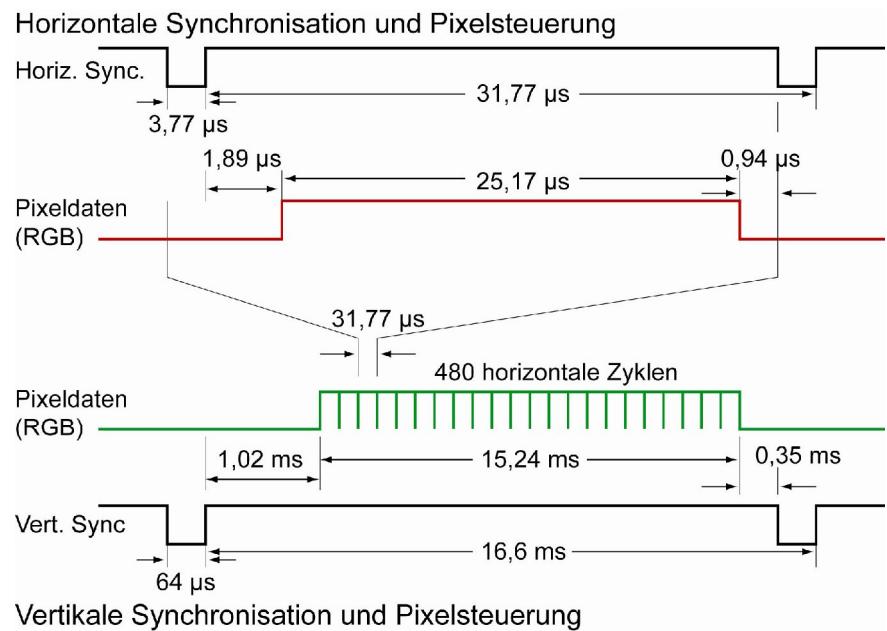
### 8.8.1 Die Synchronisation eines VGA-Monitorsignals

Der Aufbau und der zeitliche Ablauf eines Monitorsignals richten sich auch heute noch nach den ehemaligen Erfordernissen zur Steuerung eines Elektronenstrahls in einer klassischen Kathodenstrahlröhre (CRT: Cathode Ray Tube). Drei unabhängige aber gemeinsam abgelenkte Elektronenstrahlen treffen dabei auf einzelne Farbpunkte (RGB), wie dies untenstehend gezeigt ist. Die horizontale und vertikale Steuerung der Elektronenstrahlen erfolgt dabei durch entsprechende Synchronisationssignale, so dass der Strahl zeilenweise über den Bildschirm geführt wird. Während des Bildaufbaus muss ausreichend Zeit für den **Zeilenrücklauf** des Strahls berücksichtigt werden, wie dies für einen VGA-Monitor dargestellt ist. Für Monitore mit anderen Auflösungen gilt entsprechendes.



Nachdem alle Zeilen vom Strahl abgefahren wurden, kehrt der Elektronenstrahl zu seinem Ausgangspunkt zurück. Für diesen sogenannten **Bildrücklauf** muss das Synchronisationssignal wiederum ausreichend Zeit bereitstellen, während der wie schon beim Zeilenrücklauf keine Pixel angesteuert werden. Dieser Vorgang wiederholt sich je nach Bildwiederholrate etwa 60 – 100 Mal pro Sekunde. Er bestimmt auch bei den heutigen LCD-Displays (TFTs) aus Kompatibilitätsgründen den Aufbau und die Struktur eines Videosignals, wie es hier am Beispiel eines VGA-

Monitorsignals gezeigt wird. Bei einer höheren Auflösung oder einer höheren Bildfrequenz ändert sich der qualitative Ablauf nicht; nur die Dauer der verschiedenen Signale für die horizontale und vertikale Synchronisation ist entsprechend kürzer und die Anzahl der dargestellten Pixel variiert. Bei TFTs ist allerdings eine Bildwiederholrate von „nur“ 60 Hz völlig ausreichend für eine flimmerfreie Darstellung.



Der Abstand zwischen zwei horizontalen Synchronisationsimpulsen ist gleich dem Kehrwert der **Synchronisationsrate**; der Abstand der vertikalen Synchronisationsimpulse ist gleich dem Kehrwert der **Bildwiederholrate** (vergleiche Auch die Tabelle „Monitorsignale“).

Der Bildaufbau beginnt in der linken oberen Ecke des Schirms. Von hieraus werden auch die Pixel gezählt. Nach dem Schreiben einer Zeile wird der Spaltenindex (**pixel\_col**) zurückgesetzt und der Zeilenindex (**pixel\_row**) um eins erhöht. Die Signale für die vertikale (**VGA\_HS**) und horizontale (**VGA\_VS**) Synchronisation werden dabei für einen bestimmten Indexwert (Zählerwert) von logisch 1 kurzfristig auf 0 gesetzt. Dieser Ablauf ist dem VHDL Programm am Ende der Beschreibung zu entnehmen. Während des Zeilenrücklaufs wird der Elektronenstrahl unterdrückt (dunkel geschaltet: **VGA\_BLANK** → ‘0’). In dieser Zeit kann z.B. ein Bildwiederholspeicher neu beschrieben werden. Nachdem der Schirm vollgeschrieben wurde, beginnt der Prozess von neuem.

### 8.8.2 Das VGA\_SYNC Modul im Detail

Die Aufgabe des nebenstehenden VGA\_SYNC Moduls besteht nun darin, die in der Tabelle VGA\_SYNC aufgelisteten Signale zu erzeugen und aufzubereiten. Je nach der gewünschten **Monitorauflösung** (640 x 480, 800 x 600, 1024 x 768, 1280 x 1024 Pixel) und der dargestellten Farben muss das zugehörige VHDL Programm entsprechend modifiziert werden (siehe Zeile 37 ff des Programms). Dabei ist Folgendes zu beachten:

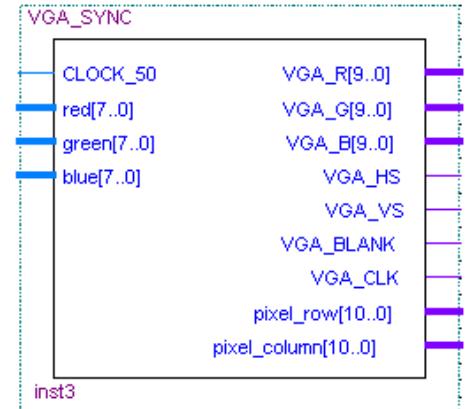
#### CLOCK\_50

Das Modul nutzt das boardeigene 50 MHz Oszillatorsignal, um über ein als Komponente einzubindendes PLL Modul (**video\_PLL**,

**vhd**) daraus den je nach Auflösung notwendigen Takt zwischen 25 und 135 MHz zu erzeugen. Dieser kann dem VHDL Programm bzw. der **Tabelle Monitorsignale** am Ende der Beschreibung entnommen werden. Das PLL Modul muss dazu entsprechend der notwendigen Taktfrequenz unter Quartus II mit Hilfe des **MegaWizard Plug-In Managers** modifiziert bzw. entworfen werden. Allerdings lassen sich damit nicht alle in der Tabelle aufgeführten Taktraten exakt erzeugen. Solange die Abweichung aber nicht mehr als etwa 5 % beträgt, sollte der Monitor trotzdem auf die erzeugten Signale synchronisieren können. Für die Programmierung des PLL Moduls sei auf das entsprechende Kapitel 6 „Der PLL Wizard“ verwiesen.

#### Red[7..0], green[7..0], blue[7..0]

In der ursprünglichen Version wurden nur drei „binäre“ Farbanteile als Eingabe definiert, so dass sich insgesamt daraus nur die gezeigten 8 Farben darstellen ließen. Dies vereinfachte zunächst die Eingabe, aber für das DE2 Board werden die Eingangsfarben auf je 8 Bit erweitert, um das gesamte Farbspektrum besser wiederzugeben. Eine Anpassung der dargestellten Farbwerte kann im VHDL Programm vorgenommen werden. Dazu sind neben den Busbreiten in der Entity die Programmzeilen 127 – 129 anzupassen.



R	G	B	oktal	Farbe
1	1	1	7	weiß
1	1	0	6	gelb
1	0	1	5	magenta
1	0	0	4	rot
0	1	1	3	türkis
0	1	0	2	grün
0	0	1	1	blau
0	0	0	0	schwarz

#### VGA\_R[9..0], VGA\_G[9..0], VGA\_B[9..0]

Die digitalen Ausgangssignale für die RGB-Farben steuern den 10 Bit Digital / Analog Konverter (DAC) ADV7123 von Analog Devices auf dem DE2 Board an, der analoge Farbsignale (RGB) mit 10 Bit Auflösung an den Monitor liefert. Dementsprechend haben sie grundsätzlich eine Wortbreite von 10 Bit. Wie oben beschrieben werden davon allerdings nur die oberen 8 Bits genutzt. Die beiden LSBs werden im **VGA\_SYNC** Modul dauerhaft auf log. 0 gesetzt. Eine Modifikation dieser Einstellungen kann in den Programmzeilen 126 ff vorgenommen werden.

#### VGA\_HS, VGA\_VS

Hauptaufgabe des VGA\_SYNC Moduls ist die Erzeugung der Signale für die horizontale (**VGA\_HS**) und die vertikale Synchronisation (**VGA\_VS**). Dazu werden geeignete „Pixelzähler“ im VHDL Programm benutzt, und die Synchronisationssignale entsprechend dem jeweiligen Zähler-

stand auf Null oder Eins gesetzt. Bei Änderung der Auflösung müssen entsprechend der **Tabelle Monitorsignale** die folgenden Konstanten im VHDL Programm (Zeile 38 – 46) angepasst werden:

<b>CONSTANT</b> H_pixels_across	-- Gesamtzahl der horizontal dargestellten Pixel
<b>CONSTANT</b> H_sync_low	-- Beginn des horizontalen Synchron. Impulses
<b>CONSTANT</b> H_sync_high	-- Ende des horizontalen Synchron. Impulses
<b>CONSTANT</b> H_end_count	-- Maximalwert des Pixelzählers (horizontal)
<b>CONSTANT</b> V_pixels_down	
<b>CONSTANT</b> V_sync_low	-- Beginn des vertikalen Synchron. Impulses
<b>CONSTANT</b> V_sync_high	-- Ende des vertikalen Synchron. Impulses
<b>CONSTANT</b> V_end_count	-- Maximalwert des Pixelzählers (vertikal)

Bevorzugte Einstellungen für verschiedene Auflösungen sind in der Tabelle am Ende des Abschnitts grün hinterlegt. Außerdem ist das Video\_PLL Modul an den entsprechenden Takt anzupassen.

#### VGA\_BLANK, VGA\_CLK

Für die Steuerung des Video DAC Bausteins ADV7123 muss das VGA\_SYNC zwei weitere Signale generieren. Das **VGA\_BLANK** Signal wird während der Zeilen- oder Bildrückläufe auf logisch 0 gesetzt und unterdrückt dadurch die Ausgabe von den RGB-Pixelwerten. Bei Einsatz eines Bildwiederholspeichers kann es zur Steuerung des Schreib Enables benutzt werden.

Das Signal **vga\_clk** liefert den Pixeltakt, wie er von der PLL-Schaltung erzeugt wird, für den DAC Baustein aus. Das Taktsignal kann aber auch von anderen Komponenten genutzt werden, wenn eine pixelgenaue Zeitsteuerung benötigt wird.

**pixel\_row[10..0], pixel\_col[10..0]**

Diese beiden Signale geben die XY-Adresse des gerade angesteuerten Pixels an. Sie ändert sich entsprechend dem Pixeltakt, und sie können genutzt werden, um über die Farbeingänge **red**, **green**, **blue** die Farbe des gerade ausgegebenen Pixels festzulegen. Die Festlegung kann auf unterschiedliche Art und Weise erfolgen. Das höchstwertigste Bit (MSB) ist bei geringeren Auflösungen (Zeilen- / Spaltenzahl < 1024) gleich Null. Sie werden z.B. auch zur Ansteuerung eines CHAR\_ROM Moduls benutzt.

VGA_SYNC	
Anschluss	Funktion
<b>Eingänge:</b>	
CLOCK_50	Externes 50 MHz Taktsignal ( <b>PIN_N2</b> )
red[7..0]	Rotanteil für die RGB Ausgabe (8 Bit)
green[7..0]	Grünanteil für die RGB Ausgabe (8 Bit)
blue[7..0]	Blauanteil für die RGB Ausgabe (8 Bit)
<b>Ausgänge:</b>	
VGA_R[9..0]	10 Bit Rot-Signale für DAC ( <b>PIN_xx</b> )
VGA_G[9..0]	10 Bit Grün-Signale für DAC ( <b>PIN_xx</b> )
VGA_B[9..0]	10 Bit Blau-Signale für DAC ( <b>PIN_xx</b> )
VGA_HS	Signal für die horizontale Synchronisation ( <b>PIN_A7</b> )
VGA_VS	Signal für die vertikale Synchronisation ( <b>PIN_D8</b> )
VGA_BLANK	Steuersignal für Video DAC Baustein (Zeilenrücklauf) ( <b>PIN_D6</b> ) VGA_BLANK = 1 signalisiert die Ausgabe von Pixelwerten VGA_BLANK = 0 signalisiert die Änderung von Pixelwerten
VGA_CLK	Pixeltakt für den Video DAC Baustein ( <b>PIN_B8</b> )
pixel_row[10..0]	Zeilenposition (Adresse) des gerade angesteuerten Pixels
pixel_col[10..0]	Spaltenposition (Adresse) des gerade angesteuerten Pixels

Alle Ein- und Ausgangssignale des Moduls VGA\_SYNC werden noch einmal in der obigen Tabelle aufgelistet. Für eine korrekte, automatische Verdrahtung der Anschlüsse auf dem DE2 Board durch Import einer \*.csv Datei sollten nur angegebenen Bezeichnungen benutzt werden. Für ein detaillierteres Verständnis sei auf den folgenden Programmauszug verwiesen.

### 8.8.3 Das VHDL Programm VGA\_SYNC

```

1 -----
2 -- Funktion : VGA_SYNC Modul in 3 x 8 Bit Farbauflösung für einen
3 -- Monitor mit 640 x 480 Pixeln
4 -- Filename : VGA_SYNC.VHD
5 -- Beschreibung : Das VGA_SYNC Modul steuert einen VGA-kompatiblen Monitor
6 -- so an, dass für jede RGB Farbe 8 Bit bei unterschiedlich
7 -- einstellbarer Auflösung (siehe unten) nutzbar sind.
8 -- Das Programm erzeugt alle notwendigen Synchronisations-
9 -- und Steuersignale.
10 -- Standard : VHDL 1993
11 -- Author : Ulrich Sandkuehler
12 -- Revision : Version 2.0 30.03.2009
13 -----
14 LIBRARY IEEE;
15 USE IEEE.STD_LOGIC_1164.all;
16 USE IEEE.STD_LOGIC_ARITH.all;
17 USE IEEE.STD_LOGIC_UNSIGNED.all;
18
19 ENTITY VGA_SYNC IS
20     PORT(CLICK_50 : IN STD_LOGIC;
21             red, green, blue : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
22             VGA_R, VGA_G, VGA_B : OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
23             VGA_HS, VGA_VS, VGA_BLANK, VGA_CLK : OUT STD_LOGIC;
24             pixel_row, pixel_column : OUT STD_LOGIC_VECTOR(10 DOWNTO 0));
25 END VGA_SYNC;
26
27 ARCHITECTURE a OF VGA_SYNC IS
28 SIGNAL horiz_sync, vert_sync, pixel_clock_int : STD_LOGIC;
29 SIGNAL video_on_int, video_on_v, video_on_h : STD_LOGIC;
30 SIGNAL h_count, v_count :STD_LOGIC_VECTOR(10 DOWNTO 0);
31
32 -- To select a different screen resolution, clock rate and refresh rate pick a set
33 -- of new video timing constant values from table at end of code section enter eight
34 -- new sync timing constants below and adjust PLL frequency output to pixel clock
35 -- rate from table using MegaWizard to edit video_PLL.vhd
36
37 -- Horizontal Timing Constants      640x480   800x600   1024x768   1280x1024
38 CONSTANT H_pixels_across : Natural := 640; --640;    800    1024    1280
39 CONSTANT H_sync_low : Natural := 664; --664;    840    1032    1328
40 CONSTANT H_sync_high : Natural := 760; --760;    968    1176    1512
41 CONSTANT H_end_count : Natural := 800; --800;    1056   1344    1712
42 -- Vertical Timing Constants
43 CONSTANT V_pixels_down : Natural := 480; --480;    600    768    1024
44 CONSTANT V_sync_low : Natural := 491; --491;    601    771    1025
45 CONSTANT V_sync_high : Natural := 493; --493;    605    777    1028
46 CONSTANT V_end_count : Natural := 525; --525;    628    806    1054
47
48 COMPONENT video_PLL25      -- Clock/MHz:      25      40      65      110
49     PORT(inclk0 : IN STD_LOGIC := '0';
50           c0 :      OUT STD_LOGIC);
51 END COMPONENT;
52
53 BEGIN
54
55 -- PLL below is used to generate the pixel clock frequency
56 -- Uses DE2's 50 MHz clock for PLL's input clock
57 video_PLL_inst : video_PLL25
58 PORT MAP(inclk0    => CLICK_50,
59            c0        => pixel_clock_int);
60
61 -- video_on is high only when RGB pixel data is being displayed
62 -- used to blank color signals at screen edges during retrace
63         video_on_int <= video_on_H AND video_on_V;
64 -- output pixel clock and video on for external user logic
65         VGA_CLK <= pixel_clock_int;
66 VGA_BLANK <= video_on_int;

```

```

68 PROCESS
69 BEGIN
70   WAIT UNTIL(pixel_clock_int'EVENT) AND (pixel_clock_int='1');
71
72 -- Generate Horizontal Timing Signals for Video Signal:
73 -- H_count counts pixels (#pixels across + extra time for sync signals)
74 --
75 -- Horiz_sync  -----
76 -- H_count      0           #pixels          sync low      end
77 --
78   IF (h_count = H_end_count) THEN h_count <= "000000000000";
79           ELSE h_count <= h_count + 1;
80   END IF;
81
82 -- Generate Horizontal Sync Signal using H_count
83   IF (h_count <= H_sync_high) AND
84       (h_count >= H_sync_low) THEN horiz_sync <= '0';
85           ELSE horiz_sync <= '1';
86   END IF;
87
88 -- Generate Vertical Timing Signals for Video Signal:
89 -- V_count counts rows of pixels (#pixel rows down + extra time for V sync signal)
90 --
91 -- Vert_sync  -----
92 -- V_count      0           last pixel row      V sync low      end
93 --
94   IF (v_count >= V_end_count) AND
95       (h_count >= H_sync_low) THEN v_count <= "000000000000";
96   ELSIF (h_count = H_sync_low) THEN v_count <= v_count + 1;
97   END IF;
98
99 -- Generate Vertical Sync Signal using V_count
100  IF (v_count <= V_sync_high) AND
101      (v_count >= V_sync_low) THEN vert_sync <= '0';
102          ELSE vert_sync <= '1';
103  END IF;
104
105 -- Generate Video on Screen Signals for Pixel Data
106 -- Video on = 1 indicates pixel are being displayed
107 -- Video on = 0 retrace - (user logic can update pixel memory
108 -- without needing to read memory for display)
109   IF (h_count < H_pixels_across) THEN video_on_h <= '1';
110           pixel_column <= h_count;
111   ELSE video_on_h <= '0';
112   END IF;
113
114   IF (v_count <= V_pixels_down) THEN video_on_v <= '1';
115           pixel_row <= v_count;
116           ELSE video_on_v <= '0';
117   END IF;
118
119 -- Put all video signals through D-FFs to elminate any small timing delays
120 -- that cause a blurry image:
121   VGA_HS <= horiz_sync;
122   VGA_VS <= vert_sync;
123
124 -- Also turn off RGB color signals at edge of screen during vertical and
125 -- horizontal retrace:
126   IF video_on_int = '1' THEN
127       VGA_R <= red & "00";
128       VGA_G <= green & "00";
129       VGA_B <= blue & "00";
130   ELSE
131       VGA_R <= "0000000000";
132       VGA_G <= "0000000000";
133       VGA_B <= "0000000000";
134   END IF;
135 END PROCESS;
136 END a;

```

### 8.8.3 Monitorsignale in Abhangigkeit von der gewunschten Auflosung

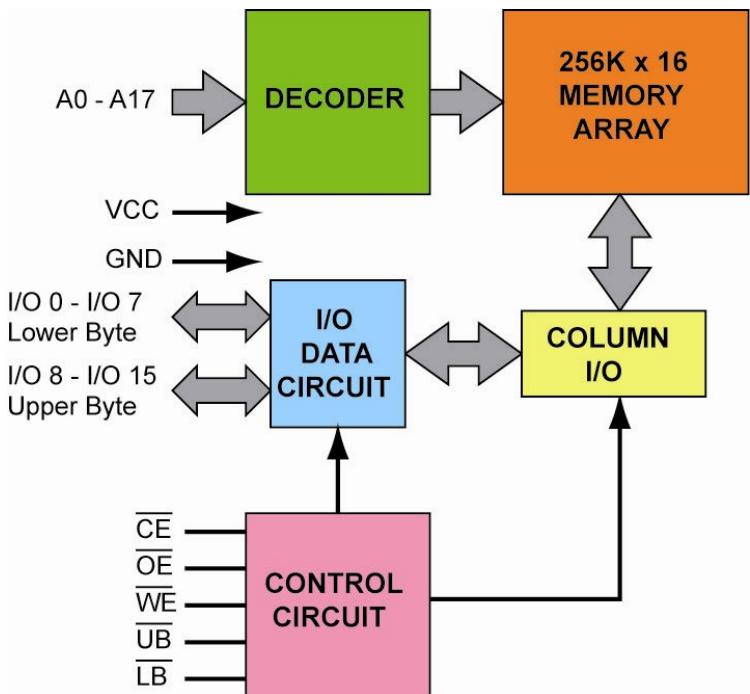
					Horizontal Timing					Vertical Timing				
Auflosung	Bild-rate	Bemerkung	Horizont. Synchron.-rate	Clock MHz	across	low	high	count	down	low	high	count	Flag +/-	VESA ?
640x480	60 Hz	Non-Interlaced mode	31.5 kHz	25.175	640	664	760	800	480	491	493	525		
640x480	60 Hz	Non-Interlaced mode	31.5 kHz	25.175	640	672	768	800	480	490	492	525		Alternate
640x480	63 Hz	Non-Interlaced mode (non-standard)	32.8 kHz	28.322	640	680	720	864	480	488	491	521		
640x480	70 Hz	Non-Interlaced mode (non-standard)	36.5 kHz	31.500	640	680	720	86	480	488	491	521		
640x480	72 Hz	Non-Interlaced mode	37.9 kHz	31.5	640	664	704	832	480	489	492	520		VESA
800x600	56 Hz	Non-Interlaced mode	35.1 kHz	36	800	824	896	1024	600	601	603	625		
800x600	56 Hz	Non-Interlaced mode	35.4 kHz	36	800	832	976	1016	600	604	606	634		Alternate
800x600	60 Hz	Non-Interlaced mode	37.9 kHz	40	800	840	968	1056	600	601	605	628	+/+	VESA
800x600	60 Hz	Non-Interlaced mode	37.9 kHz	40	800	848	1000	1056	600	605	607	633		Alternate
800x600	72 Hz	Non-Interlaced mode	48 kHz	50	800	856	976	1040	600	637	643	666	+/+	VESA
1024x768	60 Hz	Non-Interlaced mode	48.4 Hz	65	1024	1032	1176	1344	768	771	777	806	-/-	VESA
1024x768	60 Hz	Non-Interlaced mode (non-standard dot-clock)	48.4 kHz	62	1024	1064	1240	1280	768	774	776	808		
1024x768	70 Hz	Non-Interlaced mode	56.5 kHz	75	1024	1048	1184	1328	768	771	777	806		VESA
1024x768	70 Hz	Non-Interlaced mode (non-standard dot-clock)	56.25 kHz	72	1024	1056	1192	1280	768	770	776	806		
1024x768	76 Hz	Non-Interlaced mode	62.5 kHz	85	1024	1032	1152	1360	768	784	787	823		
1280x1024	59 Hz	Non-Interlaced mode (non-standard)	63.6 kHz	110	1280	1320	1480	1728	1024	1029	1036	1077		
1280x1024	61 Hz	Non-Interlaced mode	64.25 kHz	110	1280	1328	1512	1712	1024	1025	1028	1054		
1280x1024	74 Hz	Non-Interlaced mode	78.85 kHz	135	1280	1312	1456	1712	1024	1027	1030	1064		

## 8.9 Externer SRAM Speicher

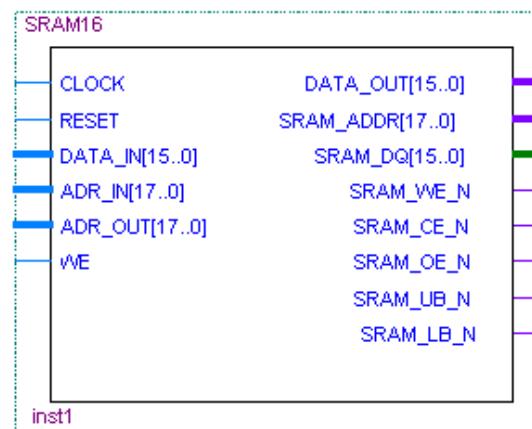
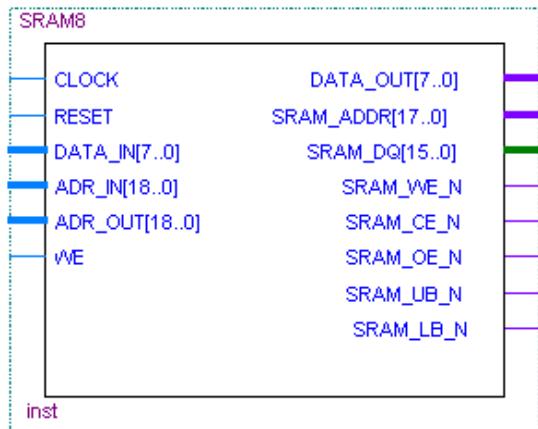
Auf dem DE2 Board steht ein schnelles, externes SRAM Speichermodul IS61LV25616 von ISSI mit einer Kapazität von 256K x 16 Bit zur Verfügung, welches mit dem FPGA fest verdrahtet ist. Das Modul verträgt Taktraten bis zu 100 MHz und eignet sich deshalb sehr gut als Bildspeicher für einen externen VGA Monitor oder das Touch Panel.

### 8.9.1 Die Arbeitsweise des SRAM Speichers

Entsprechend seiner Kapazität von 256 K Wörtern verfügt das SRAM über 18 Adressleitungen (**A0 – A17**), um die 16 Bit Speicherzellen auszuwählen. Ihr Dateninhalt wird über denselben 16 Bit breiten Bus (I/O) zum Schreiben und zum Lesen angefragt. Dabei wird das 16 Bit Datenwort in ein oberes (Upper Byte) und ein unteres Byte (Lower Byte) unterteilt, die über die negierten Enable-Eingänge **UB** und **LB** ausgewählt werden können. Der Eingang **CE** (Chip Enable) dient zur Aktivierung des Bausteins und **OE** (Output Enable) zur Aktivierung des Ausgangs. Zum Lesen muss **WE** (Write Enable) logisch 0 sein zum Lesen logisch 1. All diese Eingänge arbeiten **active low**. In der hier betrachteten Betriebsart werden sie alle bis auf WE fest auf logisch „0“ gesetzt.



### 8.9.2 Die VHDL Module SRAM8 und SRAM16



Um diesen Speicher anzusprechen, existieren zwei VHDL Module, die mit dem oben dargestellten SRAM Speicherbaustein als Kern ein **Dual Port RAM** nachbilden. Diese Module SRAM8.VHD und SRAM16.VHD unterscheiden sich nur in der Speicherorganisation (Adressierung). SRAM8 adressiert 512 K Speicherzellen a' 8 Bit, während SRAM16 256 K Zellen a' 16 Bit adressiert. Dementsprechend unterscheiden sich die Breiten der Daten- und der Adressbusse, wie in den Symbolen gezeigt.

Da es sich hier nicht um ein „echtes“ Dual Port RAM handelt, können Daten im Gegensatz zu einem solchen SRAM nicht gleichzeitig gelesen und geschrieben werden. Deshalb können bei Einsatz dieses RAMs z.B. als Bildwiederholspeicher nur während der Austastlücken (Zeilenrücklauf) Pixeldaten in den Speicher geschrieben werden, da das Lesen Vorrang hat. Zur Steuerung kann z.B. der Ausgang **VGA\_BLANK** des VGA\_SYNC Moduls benutzt werden.

Da beim Einschalten des DE2 Boards der externe Speicher zufällig mit Nullen und Einsen belegt ist, müssen alle Speicherzellen zunächst mit Nullen beschrieben werden. Dazu dient der zusätzliche **RESET Eingang**. Bei einer logischen 0 am Reset Eingang wird der Speicher entsprechend

der Taktrate mit Nullen vollgeschrieben und somit gelöscht. Dies erfordert, dass der Reset Eingang ausreichend lange auf Low Potential liegt (> 10 ms bei 50 MHz Taktfrequenz). Um alle Speicherzellen der Reihe nach zu adressieren, wird ein Zähler auf dem FPGA implementiert, der Bestandteil des unten aufgelisteten VHDL Programms ist.

Alle SRAM\_xx Anschlüsse sind auf dem DE2 Board fest verdrahtet. Für eine einfache, automatische Zuordnung der Anschlüsse sollten die vorgegebenen Namen übernommen werden.

Die Aufgaben der verschiedenen Anschlüsse kann der folgenden Tabelle entnommen werden:

SRAM8 / SRAM16	
Anschluss	Funktion
<b>Eingänge:</b>	
CLOCK	Speicher Takt
RESET	Löschen des gesamten Speichers
DATA_IN[7..0]	Datenwort am Eingang (8 Bit / 16 Bit)
ADR_IN[18..0]	Schreibadresse für Dateneingang (19 Bit / 18 Bit)
ADR_OUT[18..0]	Leseadresse für Datenausgang (19 Bit / 18 Bit)
WE	Write Enable (active low)
<b>Ausgänge:</b>	
DATA_OUT[7..0]	Datenwort am Ausgang (8 Bit / 16 Bit)
SRAM_ADDR[17..0]	SRAM Speicheradresse (PIN xx)
SRAM_DQ[15..0]	SRAM Datenbus (I/O) (PIN xx)
SRAM_WE_N	SRAM Write Enable (active low) (PIN AE10)
SRAM_CE_N	SRAM Chip Enable (,0') (PIN AC11)
SRAM_OE_N	SRAM Output Enable (PIN AD10)
SRAM_UB_N	SRAM Upper Byte Enable (PIN AF9)
SRAM_LB_N	SRAM Lower Byte Enable (PIN AE9)

### 8.9.3 Das VHDL Programm SRAM8

Die Nachbildung eines Dual Port RAMs durch geeignete Ansteuerung des externen SRAM Speichers kann dem folgenden VHDL Programm entnommen werden. Auch sei auf die Aufteilung der 16 Bit Speicherworte in ein Upper und ein Lower Byte hingewiesen. Der Aufruf eines **Adresszählers** ermöglicht das Löschen des Speichers.

Anwendungen für das DE2-70 oder das NEEK Board erfordern eine Anpassung der Speichermodule, da dort andere externe SRAM Speicherbausteine mit größerer Kapazität benutzt werden.

```

1 -----  

2 -- Funktion : Ansteuerung des extern SRAM Speichers auf dem DE2 Board  

3 -- Filename : SRAM8.vhd  

4 -- Beschreibung : Das externe Single-Port 256k x 16 SRAM wird als Pseudo-  

5 -- Dual-Port Speicher mit gebuffertem Eingang betrieben.  

6 -- Das Steuermodul SRAM stellt je 19 Bit Schreib- und  

7 -- Leseadressen (ADR_IN, ADR_OUT) und 8 Bit -daten (DATA_IN,  

8 -- DATA_OUT) zur Verfügung. RESET löscht den Speicherinhalt.  

9 -- Alle SRAM Anschlüsse sind fest auf dem DE2 Board verdrahtet.  

10 -- Standard : VHDL 1993  

11 -- Author : Ulrich Sandkuehler  

12 -- Revision : Version 1.7 19.5.2009  

13 -----  

14 library ieee;  

15 use ieee.std_logic_1164.all;  

16 use ieee.std_logic_unsigned.all;  

17  

18 entity SRAM8 is  

19     port (CLOCK, RESET : in std_logic; -- Takt, Löschen des Speichers  

20             DATA_IN      : in std_logic_vector(7 downto 0);  

21             ADR_IN       : in std_logic_vector(18 downto 0);  

22             ADR_OUT      : in std_logic_vector(18 downto 0);  

23             DATA_OUT      : out std_logic_vector(7 downto 0);  

24             WE           : in std_logic; -- Write Enable active low  

25             SRAM_ADDR    : out std_logic_vector(17 downto 0);  

26             SRAM_DQ      : inout std_logic_vector(15 downto 0);  

27             SRAM_WE_N    : buffer std_logic;  

28             SRAM_CE_N, SRAM_OE_N, SRAM_UB_N, SRAM_LB_N : out std_logic);  

29 end SRAM8;  

30  

31 architecture VERHALTEN of SRAM8 is  

32 signal WR        : STD_LOGIC;  

33 signal REG_DATA : std_logic_vector(15 downto 0);  

34 signal REG_ADR  : std_logic_vector(18 downto 0);  

35  

36 begin  

37 process(CLOCK)  

38 variable CNT : std_logic_vector(18 downto 0); -- Adresszählvariable  

39 begin  

40     if rising_edge(CLOCK) then  

41         if RESET = '0' then          -- LÖSCHEN DES SPEICHERS:  

42             WR <= '0';                -- Schreiben  

43             REG_DATA <= (others => '0'); -- 0-Daten  

44             CNT := CNT + 1;           -- Hochzählen des Zählers  

45             REG_ADR <= CNT;          -- Adresse der gelöschten Speicherzelle  

46         else  

47             WR <= WE;               -- Register für Write Enable (active low)  

48             REG_DATA <= DATA_IN & DATA_IN; -- Register für Eingangsdaten  

49             if (WE = '0') then REG_ADR <= ADR_IN;   -- Umschaltung Schreib- /  

50                 else REG_ADR <= ADR_OUT;           -- Leseadressen  

51             end if;  

52         end if;  

53     end if;  

54 end process;  

55  

56 SRAM_WE_N <= WR;                      -- SRAM Write Enable  

57 SRAM_ADDR <= REG_ADR(17 downto 0);      -- SRAM Speicheradresse  

58 SRAM_DQ  <= REG_DATA WHEN (SRAM_WE_N = '0') -- SRAM Dateneingabe  

59                   ELSE "ZZZZZZZZZZZZZZZ";  

60 DATA_OUT <= SRAM_DQ(7 downto 0) when REG_ADR(18) = '0' else  

61                     SRAM_DQ(15 downto 8);           -- SRAM Datenausgabe  

62 SRAM_CE_N <= '0';                      -- SRAM Chip Enable  

63 SRAM_OE_N <= '0';                      -- SRAM Output Enable  

64 SRAM_UB_N <= not REG_ADR(18);          -- SRAM Enable für oberes Daten Byte  

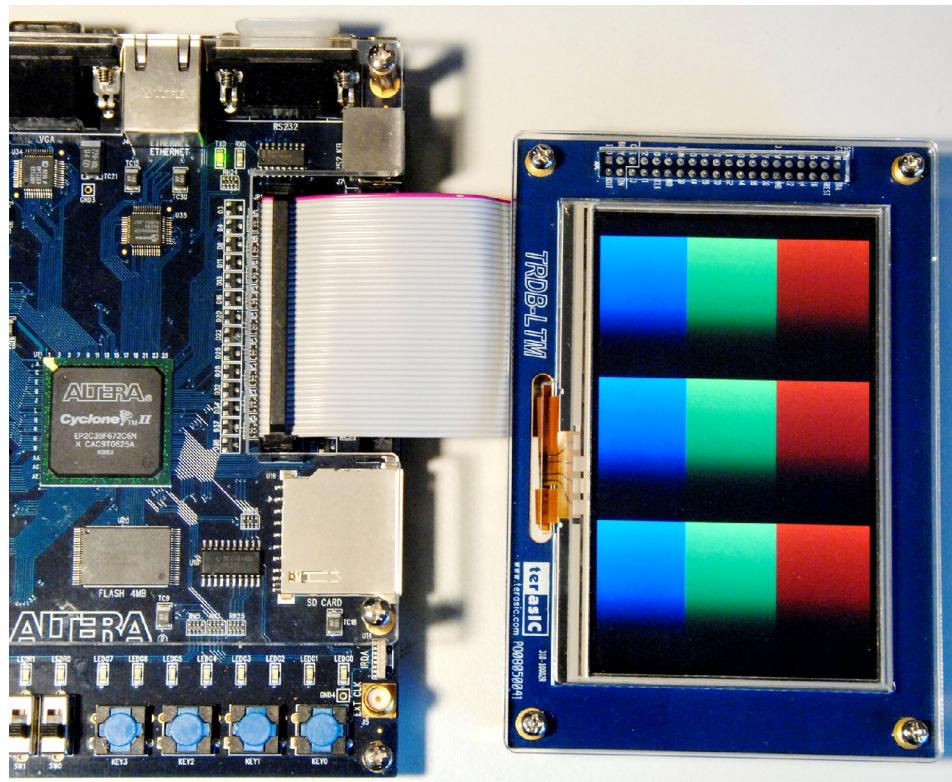
65 SRAM_LB_N <= REG_ADR(18);              -- SRAM Enable für unteres Daten Byte  

66 end VERHALTEN;

```

## 9. Das Touch Panel (TP)

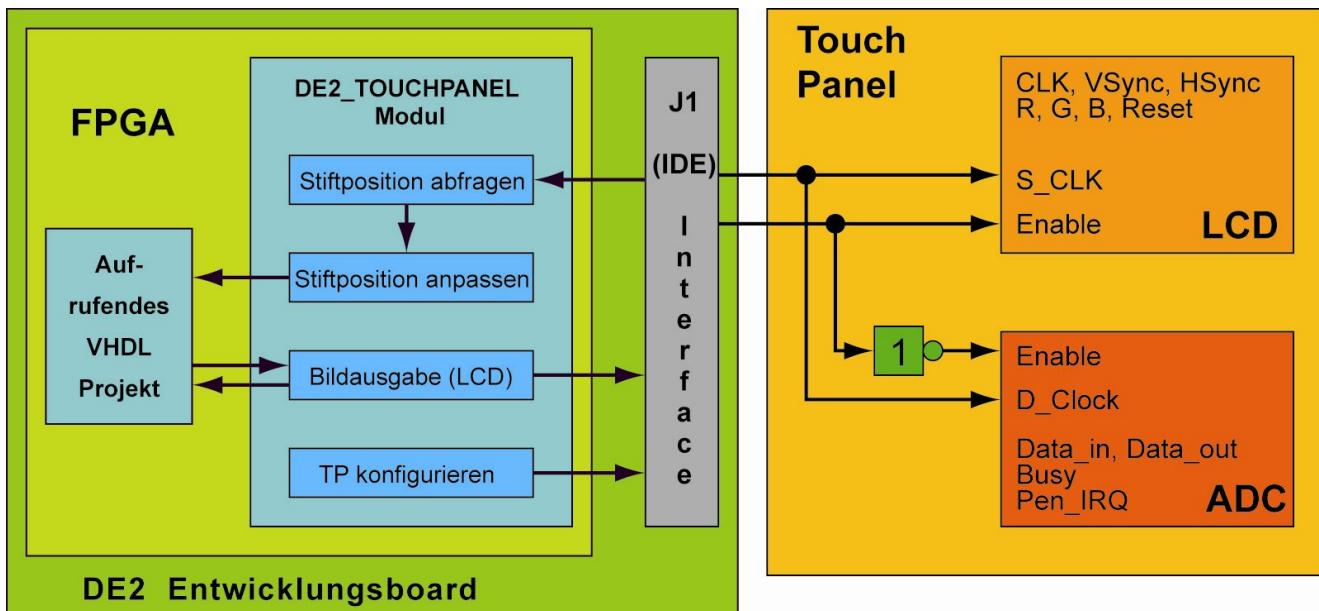
Das DE2 Board bietet die Möglichkeit, über die 40-polige **JP1 Schnittstelle** (GPI\_0) ein LCD Touch Panel von Terasic anzuschließen und zu betreiben. Dies ermöglicht nicht nur die Ausgabe von Daten wie auf einem VGA Monitor, sondern auch die Eingabe über eine berührungsempfindliche Oberfläche, die auf den Druck eines Stiftes oder Fingers reagiert. Das Display mit einer Diagonale von 4,3" (11 cm) und einer **Auflösung** von 480 x 800 Punkten und 8 Bit je Farbe (RGB) wird dabei über ein einfaches IDE-konformes Kabel angeschlossen. Die Ansteuerung ist jedoch



sehr viel komplexer als bei einem einfachen VGA Monitor. Deshalb sollen im Folgenden nur die wichtigsten Aspekte beschrieben werden, um zumindest eine ungefähre Vorstellung von der Arbeitsweise des Moduls zu bekommen. Alles Weitere kann in den technischen Unterlagen von Terasic nachgelesen werden. Für die spätere Ansteuerung des TP's wird auch hier ein fertiges VHDL Modul DE\_TOUCHPANEL.VHD bzw. NEEK\_TOUCHPANEL.VHD benutzt. Letzteres Modul dient der Ansteuerung des Touch Panels, welches fest auf dem „NIOS II Embedded Evaluation Kit“ (NEEK) integriert ist. Diese Module sind dem VGA\_SYN Modul nachempfunden, um möglichst einfach und kompatibel zu bleiben. Vorhandene Programme mit klassischer VGA Ausgabe können so leicht auf eine TP Ausgabe angepasst werden.

### 9.1 Die Arbeitsweise des Touch Panels

Die Arbeitsweise des Gesamtsystems kann in mehrere Komponenten unterteilt werden, wie dies im unten stehenden Blockschaltbild angedeutet ist. Diese sollen im Folgenden genauer betrachtet werden.



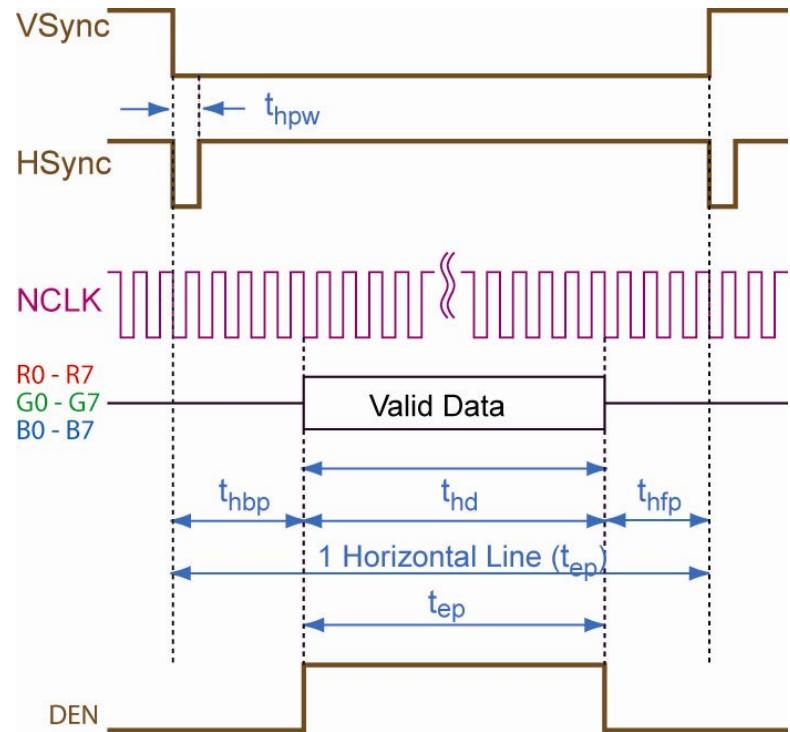
### 9.1.1 Die J1 Schnittstelle

Das DE2 Board enthält zwei externe, 40-polige Schnittstellen, die als J0 und J1 bezeichnet sind und beide einen IDE-konformen Stecker besitzen. Das Touch Panel (TP) kann über ein IDE-Kabel mit der weiter innen liegenden J1 Schnittstelle (GPIO\_0) verbunden werden. Von den 40 Leitungen sind jedoch nur 36 nutzbar, die in der folgenden Tabelle zusammen mit ihren Funktionen aufgeführt sind. Sie sind direkt mit 36 Anschlüssen (Pins) des Cyclon II FPGAs verbunden. Die restlichen Anschlüsse dienen der Stromversorgung. Dies macht es erforderlich, dass zwei Anschlüsse (GPIO\_0[0], GPIO\_0[34]) je zwei Funktionen übernehmen müssen, um alle 38 Signale des TPs zu übertragen. Alle anderen Signale werden eins zu eins durchgeleitet.

Belegung der J1 Schnittstelle			
FPGA Pin	VHDL-Label IDE-Anschuss	Signalname	Signalrichtung in / out TP
D25	GPIO-0[0]	ADC Pen Interrupt	out
J22	GPIO-0[1]	ADC Serieller Datenausgang	out
E26	GPIO-0[2]	ADC Busy	out
E25	GPIO-0[3]	ADC Serieller Dateneingang	in
F24	GPIO-0[4]	ADC/LCD Serieller Bit-Takt	in
F23	GPIO-0[5]	LCD Blau 3	in
J21	GPIO-0[6]	LCD Blau 2	in
J20	GPIO-0[7]	LCD Blau 1	in
F25	GPIO-0[8]	LCD Blau 0	in
F26	GPIO-0[9]	LCD Pixel-Takt	in
N18	GPIO-0[10]	LCD Device Enable	in
P18	GPIO-0[11]	LCD Horizontal Sync	in
G23	GPIO-0[12]	LCD Vertical Sync	in
G24	GPIO-0[13]	LCD Blau 4	in
K22	GPIO-0[14]	LCD Blau 5	in
G25	GPIO-0[15]	LCD Blau 6	in
H23	GPIO-0[16]	LCD Blau 7	in
H24	GPIO-0[17]	LCD Grün 0	in
J23	GPIO-0[18]	LCD Grün 1	in
J24	GPIO-0[19]	LCD Grün 2	in
H25	GPIO-0[20]	LCD Grün 3	in
H26	GPIO-0[21]	LCD Grün 4	in
H19	GPIO-0[22]	LCD Grün 5	in
K18	GPIO-0[23]	LCD Grün 6	in
K19	GPIO-0[24]	LCD Grün 7	in
K21	GPIO-0[25]	LCD Rot 0	in
K23	GPIO-0[26]	LCD Rot 1	in
K24	GPIO-0[27]	LCD Rot 2	in
L21	GPIO-0[28]	LCD Rot 3	in
L20	GPIO-0[29]	LCD Rot 4	in
J25	GPIO-0[30]	LCD Rot 5	in
J26	GPIO-0[31]	LCD Rot 6	in
L23	GPIO-0[32]	LCD Rot 7	in
L24	GPIO-0[33]	ADC/LCD Reset ('0' aktiv)	in
L25	GPIO-0[34]	ADC/LCD Chip Select	in
L19	GPIO-0[35]	LCD Serieller Daten Eingang	in

### 9.1.2 Die Kommunikation (LCD)

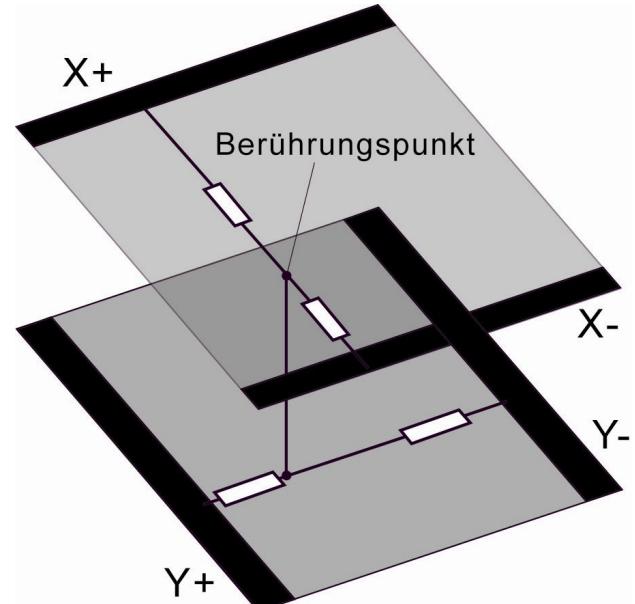
Bei der Kommunikation zwischen FPGA und TP sendet das FPGA zu Anfang eine **LCD-Konfiguration** an das LCD-Treiber-IC. Sie enthält u.a. Angaben zur Auflösung, Gammakorrektur und Leuchtstärke der Hintergrundbeleuchtung. Sie wird hier nicht weiter betrachtet und ihre Einstellungen bleiben unverändert. Erst danach erfolgt die Übertragung der **Synchronisationssignale** und **Pixeldaten** (je 8 Bit parallel für R, G, B). Der **Pixel-Takt** NCLK gibt dazu das Zeitraster vor. Da jede Bildzeile einen nicht sichtbaren Anfangs- und Endbereich besitzt, definiert ein **Device Enable** (DEN) Signal den dargestellten Bereich von 800 Pixeln für eine Zeile. Der zeitliche Ablauf entspricht qualitativ weitgehend der Ansteuerung eines VGA Monitors, wie er im Kapitel 8.8 beschrieben wurde. Dort übernahm das Signal VGA\_BLANK die Rolle des DEN Signals. HSync und VSync haben hier dieselbe Funktion wie beim VGA\_SYNC Modul. Unabhängig von der LCD Darstellung ist das Display darüber hinaus berührungsempfindlich. Die Realisierung dieser „**Touch Funktion**“ soll im Folgenden betrachtet werden.



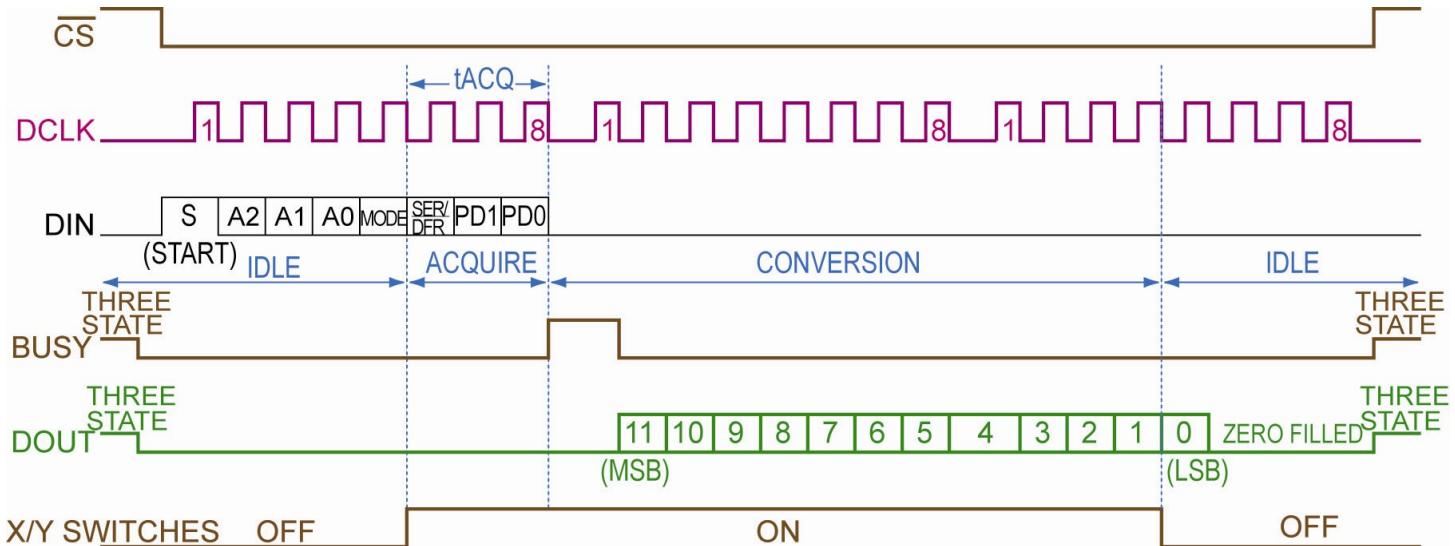
### 9.1.3 Die Positionierung (ADC)

Jede Berührung mit Stift oder Finger löst zunächst ein **Interruptsignal** aus und bestimmt anschließend die Koordinaten des Berührungs punktes. Dazu ist das Display zusätzlich mit zwei gegenüberliegenden, transparenten, elektrisch leitfähigen Folien aus Indiumzinnoxidschichten (ITO-Schichten) überzogen. Beide Folien werden mit kaum sichtbaren Abstandshaltern von einander getrennt. Bei horizontal bzw. vertikal anliegender Gleichspannung ( $X+$   $X-$ ,  $Y+$   $Y-$ ) an den gegenüber liegenden Seiten einer Folie, entsteht über dieser Folie ein Spannungsgefälle wie nebenstehend dargestellt. Wird das „Touch Panel“ mit einem Stift berührt, so verbinden sich die beiden Folien an der Druckstelle miteinander und erzeugen so einen elektrischen Kontakt (Kurzschluss).

Die jeweils andere Folie empfängt nun das Spannungspotential, welches im Spannungsgefälle entlang der Folie der Druckstelle entspricht. Der positionsabhängige Spannungswert wird möglichst hochohmig gemessen, damit der Widerstand der anderen Folie den Messwert so wenig wie möglich verfälscht. Hintereinander wird so jeweils der Spannungswert für die X- und anschließend für die Y-Position erfasst und über einen **Analog-Digital-Converter** (ADC) mit 12 Bit ausgegeben. Das Touch Panel besitzt somit eine Auflösung von 4096 Punkten in beiden Richtungen. Dieser Bereich umfasst allerdings auch Randbereiche außerhalb des sichtbaren LCD-Displays. Um kompatibel mit dem LCD Bereich zu bleiben, wird im DE2\_TOUCHPANEL Modul bzw. dessen VHDL Komponenten Fläche und Auflösung entsprechend angepasst, so dass auch hier die gleiche Zählweise für Zeile und Spalte (beginnend links oben) und eine **Auflösung** von  $480 \times 800$  für die „Pen-Punkte“ gilt.



Die Ansteuerung des ADCs über die J1 Schnittstelle erfolgt seriell. Er wird über eine '0' am Eingang „**CS**“ (Chip Select) angesprochen. Die Empfangsbereitschaft wird mit einer '0' am Ausgang „**BUSY**“ signalisiert. Während der darauf folgenden acht Bit-Takten am Takt-Eingang „**DCLK**“ wird über den Daten-Eingang „**DIN**“ das 8-Bit lange Konfigurationswort seriell eingelesen. Nach dem 9. Bit-Takt erscheint bereits das erste digitalisierte Positions-Bit (MSB) am Daten-Ausgang „**DOUT**“. Während der nächsten 11 Bit-Takte werden die weiteren Bits der Position ausgegeben. Vom 22. Bit-Takt, mindestens 3 Bit-Takte lang oder bis zu dem Zeitpunkt, an dem das „**CS**“-Signal beendet wird, bleibt der Ausgang „**DOUT**“ im Zustand '0'. Die Anforderungen für die X- und die Y-Koordinaten erfolgen unabhängig von einander.



Das 8 Bit lange Konfigurationswort auf der Leitung **DIN** bestimmt unter Anderem, ob eine X- oder Y-Position angefordert wird und mit welcher Auflösung gearbeitet wird. Das Chip Select Signal CS ist erforderlich, da die Taktleitung (DCLK) für den ADC Baustein auch vom LCD Baustein mit benutzt wird.

## 9.2 VHDL Module zur grafischen Nutzung des TP's

Um die diversen Ein- und Ausgabemöglichkeiten des Touch Panels komfortabel nutzen zu können, werden im Folgenden verschiedene Module vorgestellt. Für spezielle Anwendungen lassen sich diese Module entsprechend modifizieren und auf die persönlichen Bedürfnisse zuschneiden.

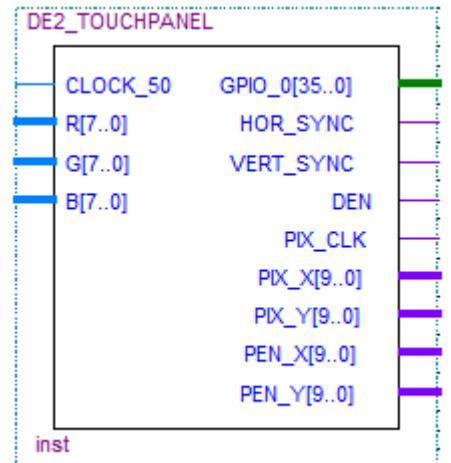
Diese Module eignen sich insbesondere auch für das NEEK Board, welches nur sehr eingeschränkte Ein- und Ausgabemöglichkeiten bietet, so dass viele IO Funktionen auf das Touch Panel begrenzt sind.

### 9.2.1 Das DE2\_TOUCHPANEL Modul

Die externen Anschlüsse des Moduls orientieren sich aus Kompatibilitätsgründen weitestgehend an dem VGA\_SYNC Modul. Dazu kommt der externe GPIO\_0 Anschluss Bus (J1 Steckerleiste auf dem DE2 Board) für das Touch Panel an sich. Neu ist die Ausgabe der Position eines Fingers oder besser Stiftes über die Ausgänge **PEN\_X[9..0]** und **PEN\_Y[9..0]** mit einer Auflösung von 480 x 800 Punkten. Dabei werden die Koordinaten des jeweils letzten Berührungs punktes solange gespeichert (ausgegeben), bis eine neue Berührung erfolgt.

Zur Ansteuerung des Touch Panels greift das Modul noch auf mehrere weitere Komponenten zurück, um die Synchronisation und Initialisierung mit der Konfigurationsdatei für die Ansteuerung des TP's durchzuführen. Diese Dateien sind zum Teil in Verilog realisiert und befinden sich ebenfalls in der DE2core Library:

- **TP\_SYNC.VHD** (Generierung der Synchronisationssignale auf dem DE2 Board),
- **adc\_spi\_controller.v** (Verilog Datei von Terasic zur Steuerung des ADCs),



- `Lcd_spi_controller.v` (Verilog Datei von Terasic zur Steuerung des LCDs),
- `Reset_Delay.v` (Verilog Datei von Terasic),
- `PEN_ADJ.VHD` (Anpassung der Stiftpositionen an das LCD Koordinatensystem).

Die Anschlüsse haben die folgenden Funktionen:

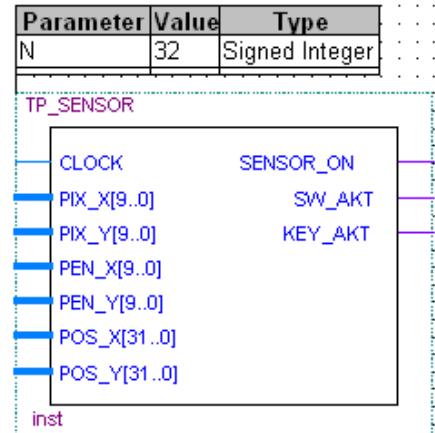
DE2_TOUCHPANEL	
Anschluss	Funktion
<b>Eingänge:</b>	
CLOCK_50	Externes 50 MHz Taktsignal (Systemtakt) (PIN N2)
R[7..0]	8 Bit Farbwert für Rot
G[7..0]	8 Bit Farbwert für Grün
B[7..0]	8 Bit Farbwert für Blau
<b>Ausgänge:</b>	
GPIO_0[35..0]	J1 (IDE) Schnittstelle für das Touch Panel (PIN XX)
HOR_SYNC	Taktausgabe für die vertikale Synchronisation
VERT_SYNC	Taktausgabe für die vertikale Synchronisation
DEN	VGA (Device) Enable (BLANK)
PIX_CLK	Pixeltakt
PIX_X[9..0]	X-Positionsausgabe der LCD Pixel
PIX_Y[9..0]	Y-Positionsausgabe der LCD Pixel
PEN_X[9..0]	X-Positionsausgabe des Berührungs punktes
PEN_Y[9..0]	Y-Positionsausgabe des Berührungs punktes

Das entsprechende Modul für das NEEK Board wird weiter unten beschrieben.

### 9.2.2 Das TP\_SENSOR Modul

Das TP\_SENSOR Modul bietet einen recht universellen Zugang zu den Ein- Ausgabemöglichkeiten des Touch Panels sowohl für das DE2 Board wie auch für das NEEK Board, indem es einen **virtuellen Schalter** bzw. **Taster** auf einen definierten Bereich des Touch Panels realisiert. Es bildet die Basiskomponente für unterschiedliche, weitere Anwendungen. Dazu wird standardmäßig (Generic Anweisung) ein 32 x 32 Pixel großer, **quadratischer Bereich** festgelegt, der beliebig auf dem Touch Panel positioniert werden kann. Über die **Positionsangaben** (`POS_X`, `POS_Y`) wird hier und auch bei allen anderen Modulen immer die linke, obere Ecke des Display Bereiches in Pixeln (Spaltennummer, Zeilennummer im **Integer Format**) festgelegt. Für eine manuelle Eingabe könnte die Größe des Bereichs entsprechend auf 64 x 64 oder mehr Pixel erhöht werden, um ihn „fingergerecht“ zu gestalten. Dazu steht eine einfache **GENERIC Anweisung** zur Verfügung. Bei der Nachbildung des **Tasters** (Ausgang `KEY_AKT`) liefert jede Berührung des Sensors einen einzelnen, etwa 100 ms langen, positiven Impuls. Gleichzeitig ändert sich mit jeder Berührung auch der Ausgangswert des virtuellen **Schalters** (`SW_AKT`), indem er quasi hin und her toggelt. Im Ausgangszustand liefert er eine log. 0. Für eine fehlerfreie Funktion ist es erforderlich, dass sich bei jeder Berührung die Stift Position um wenigstens ein Pixel ändert, auch darf der Zeitabstand zwischen zwei Berührungen etwa 50 ms nicht unterschreiten. Etwas Erfahrung bei der Berührung des Touch Panels ist hier hilfreich.

Über das Ausgangssignal `SENSOR_ON` kann der **Sensorbereich** für unterschiedliche „Schalterstellungen“ als Reaktion auf eine Berührung farbig gekennzeichnet werden (**Eingabemodus**). Alternativ kann die Farbdarstellung des Sensorbereichs in Abhängigkeit von einem Steuersignal geändert werden, so dass sich eine mehrfarbige LED nachbilden lässt (**Ausgabemodus**). Zusätzlich können auch Schalter (`SW_AKT`) und Taster Aktivitäten (`KEY_AKT`) ebenso ausgewertet werden, um je nach Schalterstellung die Farben von Text und Hintergrund zu ändern (z.B. zu invertieren) und so die Stellung des virtuellen Schalters zu signalisieren.



Die praxisnahe Einbindung dieses und weiterer Module beschreibt das Programm **TP\_DESIGN.VHD** am Schluss dieser Ausführungen. Es bildet eine nützliche Basis zum Verständnis der Arbeitsweise auch aller anderen TP Module.

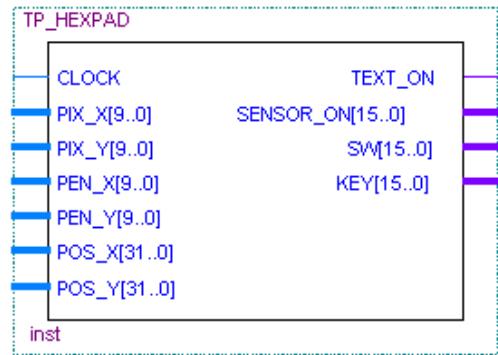
Die Anschlüsse haben die folgenden Funktionen:

TP_SENSOR	
Anschluss	Funktion
<b>GENERIC Parameter:</b>	
N (= 32)	Seitenlänge des Sensorbereiches in Pixeln
<b>Eingänge:</b>	
CLOCK	Pixel Takt (vom DE2_TOUCHPANEL)
PIX_X[9..0], PIX_Y[9..0]	Positionsübergabe der LCD Pixelwerte (vom DE2_TOUCHPANEL)
PEN_X[9..0], PEN_Y[9..0]	Positionsübergabe des PEN Berührungs punktes (vom DE2_TOUCHPANEL)
POS_X, POS_Y	Positionierung des Sensors (obere, linke Ecke) auf dem TP in Pixeln (Integer)
<b>Ausgänge:</b>	
SW_AKT	Schalter Aktivität (0 oder 1) Toggeln
KEY_AKT	Taster Aktivität (1 Impuls)
SENSOR_ON	Hintergrund des aktiven Sensorbereichs

### 9.2.3 Das TP\_HEXPAD Modul

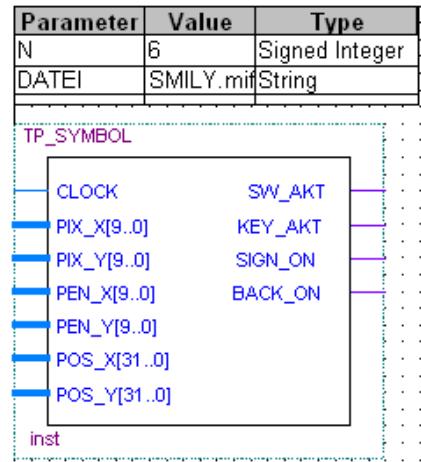
Dieses Eingabemodul benutzt die Komponenten TP\_SENSOR und HEX\_ROM16, um ein 4 x 4 Tastenfeld (Keypad) mit allen Hexadezimalzeichen zu definieren und darzustellen. Jede Taste übernimmt dabei sowohl Taster- wie auch Schalterfunktion, so dass man über 16 virtuelle Schalter / Taster (SW\_AKT, KEY\_AKT) verfügt. Die Farben der HEX-Zeichen und der Taster können im einbindenden Design (VHDL Programm) über TEXT\_ON bzw. SENSOR\_ON definiert werden.

Es ist zu beachten, dass mit jedem Aufruf der Komponente TP\_HEXPAD zusätzliche 4 Kbit an ROM-Speicher für die mit eingebundenen HEX\_ROM16's auf dem FPGA benötigt werden.



Alle Ein- und Ausgänge sind wie folgt definiert:

TP_HEXPAD	
Anschluss	Funktion
<b>Eingänge:</b>	
CLOCK	Pixel Takt (vom DE2_TOUCHPANEL)
PIX_X[9..0], PIX_Y[9..0]	Positionsübergabe der LCD Pixelwerte (vom DE2_TOUCHPANEL)
PEN_X[9..0], PEN_Y[9..0]	Positionsübergabe des PEN Berührungs punktes (vom DE2_TOUCHPANEL)
POS_X, POS_Y	Positionierung des Keypads (obere, linke Ecke) auf dem TP in Pixeln
<b>Ausgänge:</b>	
TEXT_ON	Textdarstellung aktiv
SENSOR_ON[15..0]	Sensorbereich aktiv
SW_AKT[15..0]	Schalter Aktivität (0 oder 1) Toggeln
KEY_AKT[15..0]	Taster Aktivität (1 Impuls)



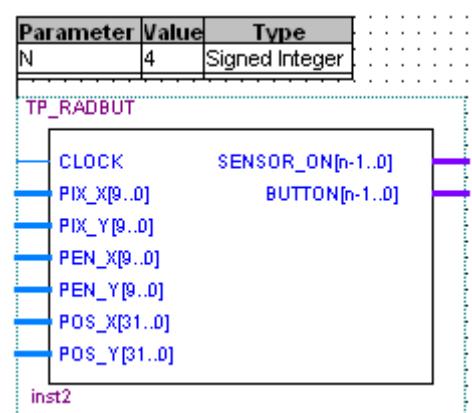
#### 9.2.4 Das TP\_SYMBOL Modul

TP\_SYMBOL bildet einen einzelnen (großen) virtuellen Schalter / Taster nach, der bei jeder Betätigung zwischen zwei unterschiedlichen Symboldarstellungen auf dem Taster hin und her schaltet, die in einem speziellen Character Rom (z.B. **SMILY\_ROM**) mit einer Auflösung von  $64 \times 64$  Pixeln abgelegt sind. Exemplarisch werden hier zwei Smilies gezeigt. Alternativ können auch andere ROM Dateien gleicher Größe und Struktur über den **GENERIC Parameter „DATEI“** eingebunden werden. Die Größe wird dabei durch den Parameter **N** zu  $2^N$  festgelegt. Ansonsten entspricht die Arbeitsweise des Moduls der Komponente TP\_SENSOR.

TP_SYMBOL	
Anschluss	Funktion
<b>GENERIC Parameter:</b>	
N (= 6)	Seitenlänge des Sensorbereiches in $2^{**N}$ Pixeln
DATEI (= „Smily.mif“)	Datei mit zwei Symbolen (hier ☺ ☻)
<b>Eingänge:</b>	
CLOCK	Pixel Takt (vom DE2_TOUCHPANEL)
PIX_X[9..0], PIX_Y[9..0]	Positionsübergabe der LCD Pixelwerte (vom DE2_TOUCHPANEL)
PEN_X[9..0], PEN_Y[9..0]	Positionsübergabe des PEN Berührungs punktes (vom DE2_TOUCHPANEL)
POS_X, POS_Y	Positionierung des Sensors (obere linke Ecke) auf dem TP in Pixeln (Integer)
<b>Ausgänge:</b>	
SW_AKT	Schalter Aktivität (0 oder 1) Toggeln
KEY_AKT	Taster Aktivität (1 Impuls)
SIGN_ON	Symbol des Sensorbereichs aktiv
BACK_ON	Hintergrund des Sensorbereichs aktiv

#### 9.2.5 Das TP\_RADBUT Modul

Das Modul TP\_RADBUT realisiert eine horizontale Leiste von standardmäßig 4 **Radio Buttons**, d.h. es ist immer nur der zuletzt ausgewählte Taster aktiv und der vorher betätigte Button wird zurückgesetzt. Der aktive Taster wird am Ausgang **BUTTON** dadurch gekennzeichnet, dass von den N Bits nur immer ein Bit logisch „1“ ist. Alle anderen Bits sind logisch „0“. Vor der ersten Auswahl sind alle Bits logisch „0“. Durch Editieren des Programms kann die Größe und Anordnung der Buttons geändert werden. Eine „Beschriftung“ kann außerhalb z.B. in dem aufrufenden Programm erfolgen. Die Anzahl der von einander abhängigen Buttons kann durch den GENERIC Parameter N variiert werden.



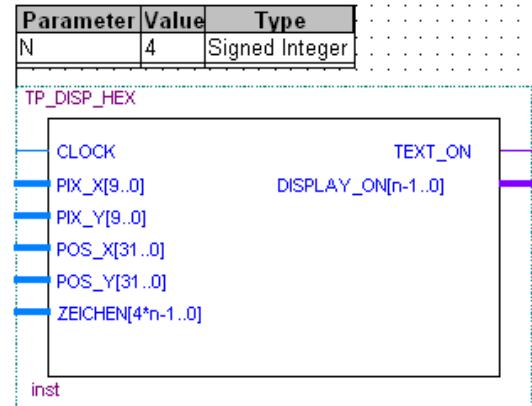
Die Anschlüsse haben die folgenden Funktionen:

TP_RADBUT	
Anschluss	Funktion
<b>GENERIC Parameter:</b>	
N (= 4)	Anzahl der Radio Buttons
<b>Eingänge:</b>	
CLOCK	Pixel Takt (vom DE2_TOUCHPANEL)
PIX_X[9..0], PIX_Y[9..0]	Positionsübergabe der LCD Pixelwerte (vom DE2_TOUCHPANEL)
PEN_X[9..0], PEN_Y[9..0]	Positionsübergabe des PEN Berührungs punktes (vom DE2_TOUCHPANEL)
POS_X, POS_Y	Positionierung der Tasterzeile (linke, obere Ecke) auf dem TP in Pixeln (Integer)
<b>Ausgänge:</b>	
BUTTON[N-1..0]	Taster Aktivitäten (1 aus N)
SENSOR_ON[N-1..0]	Hintergrund der Sensorbereiche

### 9.2.6 Das TP\_DISP\_HEX Modul

Zwei weitere Module dienen allein der **Ausgabe von Daten**.

Dabei greift das Modul TP\_DISP\_HEX ebenfalls auf die Komponenten TP\_SENSOR und HEX\_ROM16 zurück, um einen (mehrstelligen) **Ausgabebereich** zu definieren. Standardmäßig werden 4 horizontal angeordnete Stellen der Größe 32 x 32 Pixel angezeigt. Auch hier ist zu beachten, dass mit jedem Aufruf der Komponente TP\_DISP\_HEX zusätzliche 4 Kbit an ROM-Speicher für die mit eingebundenen HEX\_ROM16's auf dem FPGA benötigt werden. Über **TEXT\_ON** bzw. **SENSOR\_ON** ist wie beim Modul HEX\_PAD eine **farbige Gestaltung** möglich.

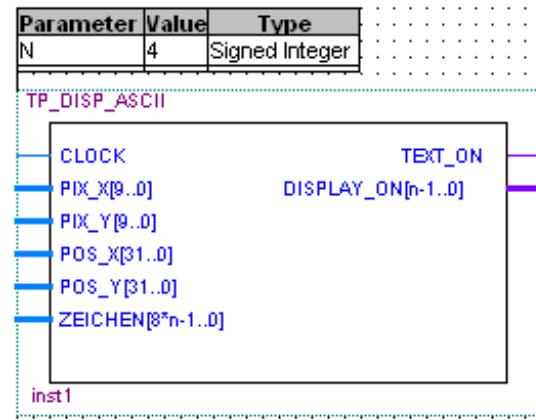


Die Anschlüsse haben die folgenden Funktionen:

TP_DISP_HEX	
Anschluss	Funktion
<b>GENERIC Parameter:</b>	
N (= 4)	Anzahl der dargestellten HEX-Zeichen
<b>Eingänge:</b>	
CLOCK	Pixel Takt (vom DE2_TOUCHPANEL)
PIX_X[9..0], PIX_Y[9..0]	Positionsübergabe der LCD Pixelwerte (vom DE2_TOUCHPANEL)
POS_X, POS_Y	Positionierung des Displays (obere linke Ecke) auf dem TP in Pixeln (Integer)
ZEICHEN[4*N-1..0]	Je 4 Bit für jedes dargestellte HEX-Symbol
<b>Ausgänge:</b>	
TEXT_ON	Textdarstellung aktiv
DISPLAY_ON	Displaybereich aktiv

### 9.2.7 Das TP\_DISP\_ASCII Modul

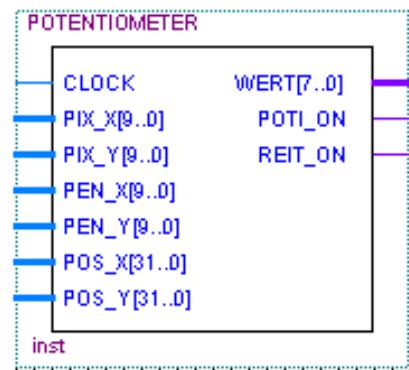
Das zweite Ausgabemodul erlaubt nicht nur HEX-Werte sondern alle ASCII Zeichen entsprechend der modifizierten **7 Bit ASCII Tabelle** in Kapitel 8.5.5 darzustellen. Die Arbeitsweise und das Design entspricht dem Modul TP\_DISP\_HEX, nur dass das HEX\_ROM16 durch das ASCII\_ROM16 mit 128 anstatt 16 Zeichen ersetzt wurde. Dementsprechend erhöht sich der Bedarf an ROM-Speicher auf 32 Kbit je eingebundener Komponente. Um den Aufruf der Zeichen zu vereinfachen und dem Modul LCD\_DISPLAY anzupassen, werden je ASCII Zeichen **8 Bit** anstatt 7 Bit benutzt, wobei das MSB aber grundsätzlich logisch „0“ ist. Die Codierung der Zeichen ist der ASCII Tabelle in Kapitel 8.5.5 zu entnehmen.



TP_DISP_ASCII	
Anschluss	Funktion
<b>GENERIC Parameter:</b>	
N (= 4)	Anzahl der dargestellten ASCII-Zeichen
<b>Eingänge:</b>	
CLOCK	Pixel Takt
PIX_X[9..0], PIX_Y[9..0]	Positionsübergabe der LCD Pixelwerte
POS_X, POS_Y	Positionierung des Displays (obere, linke Ecke) auf dem TP in Pixeln (Integer)
ZEICHEN[8*N-1..0]	Je 8 Bit für jedes dargestellte ASCII-Zeichen
<b>Ausgänge:</b>	
TEXT_ON	Textdarstellung aktiv
DISPLAY_ON	Displaybereich aktiv

### 9.2.8 Das POTENTIOMETER Modul

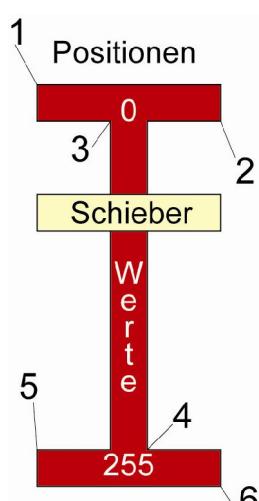
Das Modul POTENTIOMETER generiert auf dem Touch Panel ein einfaches, vertikales **Schiebepotentiometer**, wie es unten mit allen Positionsnummern zur Generierung im VHDL Programm gezeigt wird. Es ermöglicht eine **analoge Eingabe** digitaler Werte. Mit der Berührung des vertikalen Reiterbalkens erscheint ein Schieber (Reiter), der sich in einem Bereich von 0 – 255 Pixeln verschieben lässt, so dass ein 8 Bit Ausgabewert WERT [7..0] analog der Reiterstellung generiert und ausgegeben wird. Die farbige Darstellung kann auch hier über die Ausgänge POTI\_ON und REIT\_ON im aufrufenden Programm beeinflusst werden.



Die **Positionsangaben** (**pos\_x**, **pos\_y**) beziehen sich auch hier auf die linke, obere Ecke des Potentiometers in Pixeln (Nummer von Spalte und Zeilen im **Integer Format**). Es sollte darauf geachtet werden, dass der Darstellungsbereich des Touch Panels nicht überschritten wird!

Die Abmessungen und Positionen des Potentiometers werden innerhalb des VHDL Programms definiert. Der entsprechende Programmausschnitt ist unten dargestellt. Er kann für eine andere z.B. horizontale Form des Potentiometers editiert werden. Auch ließe sich der ausgegebene Wertebereich über die Länge des Potentiometers modifizieren.

Für den Ausgabewert 0 befindet sich der Reiter (Schieber) am unteren Anschlag des Potentiometers. Der Maximalwert von 255 wird am oberen Anschlag definiert.



```

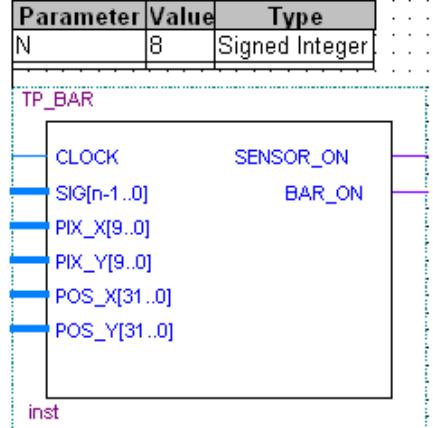
-- Alle darstellungsrelevanten Positionen auf dem TP:
type POSITION is array (1 to 6, 1 to 2) of integer;
constant POSI : POSITION :=
    (( 0,      0), -- (1) obere linke Ecke oberer Balken      1-----
    (31,      7), -- (2) untere rechte Ecke oberer Balken      -----2
    (12,      7), -- (3) obere linke Ecke mittlerer Balken     3|
                -- |||
                -- Länge des mittleren Balkens = 256 Pixel        |||
                -- |||
(19,   262), -- (4) untere rechte Ecke mittlerer Balken     |4
( 0,   262), -- (5) linke oberer Ecke unterer Balken       5-----
(31,   269));-- (6) rechte untere Ecke unterer Balken      -----6

```

POTENTIOMETER	
Anschluss	Funktion
<b>Eingänge:</b>	
CLOCK	Pixel Takt (vom DE2_TOUCHPANEL)
PIX_X[9..0], PIX_Y[9..0]	Positionsübergabe der LCD Pixelwerte (vom DE2_TOUCHPANEL)
PEN_X[9..0], PEN_Y[9..0]	Positionsübergabe des PEN Berührungs punktes (vom DE2_TOUCHPANEL)
POS_X, POS_Y	Positionierung des Potentiometers (linke, obere Ecke) auf dem TP in Pixeln (Integer)
<b>Ausgänge:</b>	
WERT[7..0]	8 Bit Ausgabewert
POTI_ON	Potentiometer aktiv
REIT_ON	Schiebereiter aktiv

### **9.2.9 Das TP BAR Modul**

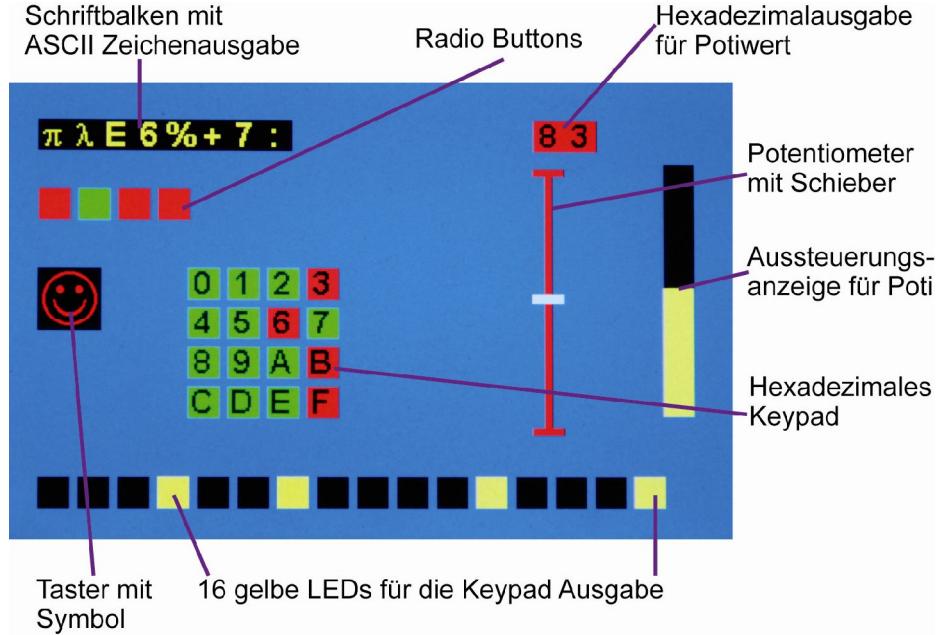
Während das Potentiometer die analoge Eingabe digitaler Werte gestattet, ermöglicht das Modul TP\_BAR die analoge, visuelle Ausgabe digitaler Werte in Form einer **Aussteuerungsanzeige**, deren Balkenlänge entsprechend dem Signalwert **SIG[n-1..0]** variiert. Standardmäßig können 255 Werte dargestellt werden. Eine Adaptierung auf andere Wertebereiche ist möglich. Die Farbe des Aussteuerungsbalkens sollte über **BAR\_ON** im einbindenden Programm eingestellt werden (siehe TP\_DISPLAY).



TP_BAR	
Anschluss	Funktion
<b>GENERIC Parameter:</b>	
N (= 8)	Anzahl der Signalbits
<b>Eingänge:</b>	
CLOCK	Pixel Takt (vom DE2_TOUCHPANEL)
SIG[n-1..0]	Darzustellendes Eingangssignal mit N Bit
PIX_X[9..0], PIX_Y[9..0]	Positionsübergabe der LCD Pixelwerte (vom DE2_TOUCHPANEL)
POS_X, POS_Y	Positionierung des Potentiometers (linke, obere Ecke) auf dem TP in Pixeln (Integer)
<b>Ausgänge:</b>	
SENSOR_ON	Balkenbereich aktiv
BAR_ON	Aussteuerungsbereich aktiv

### 9.3 Die Zusammenarbeit der Module im Programm TP DESIGN

Die Einbindung der verschiedenen, oben vorgestellten, interaktiven I/O Elemente für das Touch Panel wird exemplarisch in dem VHDL Programm **DESIGN.VHD** vorgeführt, welches die nebenstehend gezeigte Darstellung auf dem TP ausgibt. Auf dieses Programm in der DE2core Library kann für eigene Designs zurückgegriffen werden. In ihm wird die Art der **Darstellung** (Farben) der aktiven (Taster / Schalter) und nicht aktiven Sensorbereiche (Symbol- und Zeichenausgabe, Aussteuerungsanzeige) festgelegt. Auf dem Touch Panel können so die unterschiedlichen Funktionsweisen der TP Module untersucht werden.



In dem VHDL Programm TP\_DESIGN werden die verschiedenen, benötigten Komponenten aufgelistet und die Übergabe aller Modulwerte beschrieben. Von besonderem Interesse ist der vom Pixel Takt gesteuerte **Prozess zur Darstellung der Farben** der verschiedenen Elemente. Er sollte vornehmlich beachtet werden und als Basis für eigene Design dienen!

Das Display zeigt die folgenden virtuellen I/O-Elemente:

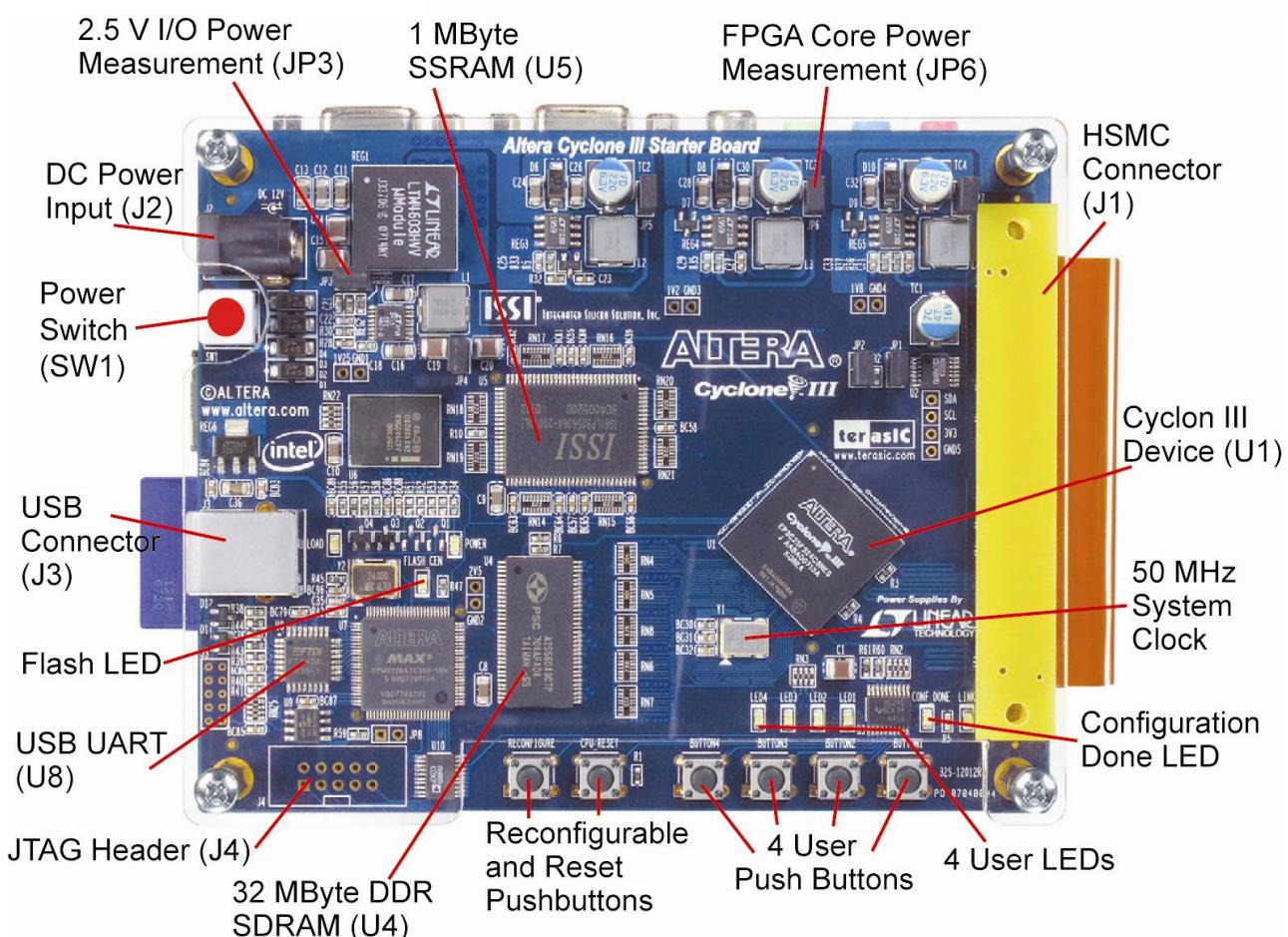
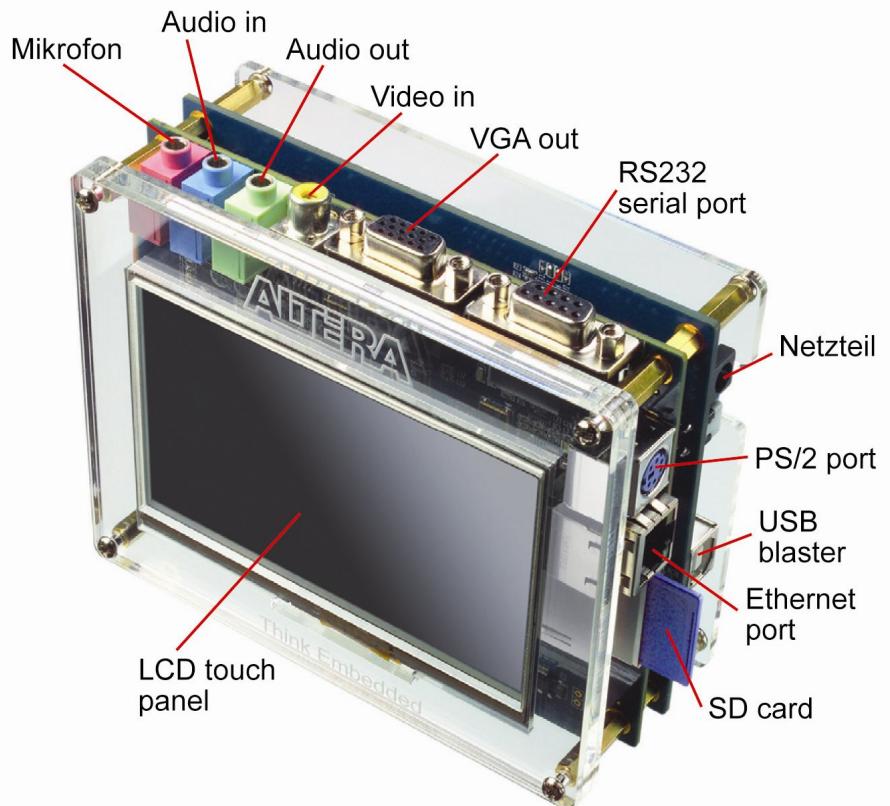
- Ein Schriftbalken mit 7 gelben ASCII Zeichen auf schwarzem Hintergrund.
- Ein großer Taster mit wechselndem Symbol (Smily).
- Ein Keypad zur Hexadezimalen Eingabe (16 Taster / Schalter).
- Eine LED Zeile mit gelben Leuchtdioden, die auf die hexadezimalen Keypad Eingaben reagiert.
- Ein Schiebepotentiometer mit 256 Werten.
- Eine Balkenanzeige (Aussteuerungsanzeige) für die analoge Ausgabe der Potentiometer-einstellung.
- Eine hexadezimale Ausgabe für den Wert des Potentiometers (Reiterstellung).
- Eine horizontale Leiste von 4 Radio Buttons (grün ist aktiv).

## 10. Das NEEK Entwicklungsboard

Das „Nios II Embedded Evaluation Kit“ (NEEK Entwicklungssystem) ermöglicht unter Quartus II die Entwicklung eines eigenständigen Prozessorsystem mit einem Touch Panel zur Ein- und Ausgabe auf kleinstem Raum.

### 10.1. Aufbau und Komponenten

Das NEEK Board besteht aus zwei mit einander verschraubten Platinen, dem „**Altera Cyclone III Starter Board**“ und dem „**Multimedia Touch Panel Daughter Board**“ (MTDB).



Das „Altera Cyclone III Starter Board“ ist ein vollwertiges FPGA-Entwicklungsboard und kann eigenständig mit Quartus II betrieben werden. Es ist auf komplexe Schaltungslogiken mit „Nios II“ Prozessoren ausgelegt, kann aber auch für komplexe Schaltnetze benutzt werden, wie sie sich auf dem DE2 Board implementieren lassen. Bestückt ist es mit einem Altera **Cyclone III FPGA** des Typs **EP3C25F324C6**. Konfiguriert und programmiert wird der FPGA im JTAG-Mode über den USB-Blaster. Für komplexere Anwendungen stehen zusätzlich 32 MB DDR-SDRAM-Speicher, 1 MB SSRAM-Speicher und 16 MB Flash-Speicher zur Verfügung. An Bedienungs- und Anzeigeelementen sind allerdings lediglich 4 frei verfügbare LEDs und 4 frei verfügbare Taster vorhanden. Insofern sollte die Ein- und Ausgabe von Datenwerten oder Steuerfunktionen auf das Touch Panel des MTDBs verlagert werden.

Der „**84 Pin HSMC Adapter**“ ist die einzige Anschlussmöglichkeit zur weiteren Peripherie. Standardmäßig ist der HSMC Adapter mit dem MTDB verbunden und steht somit für andere Anwendungen nicht mehr zur Verfügung.

Das „Multimedia Touch Panel Daughter Board (MTDB)“ dient als Peripherieerweiterung des „Starter Boards“ und verfügt über einen 24-Bit Stereo-Audiocodec mit drei Klinkenanschlüssen, einen Video-Dekoder, einen VGA-Ausgang, einer seriellen RS232-Schnittstelle, einen Ethernet-Transceiver, einen SD-Speicherkarten-Slot, einen PS/2-Anschluss und ein farbiges 4,3 Zoll LCD Touch Panel (TP), welches baugleich zu dem TP im DE2-System ist. Die Abbildung zeigt das gesamte NEEK-Entwicklungssystem.

Die Eigenschaften des FPGAs EP3C25F324C6 (fett gekennzeichnet) sowie anderer Mitglieder der Cyclon III Familie können der folgenden Tabelle entnommen werden:

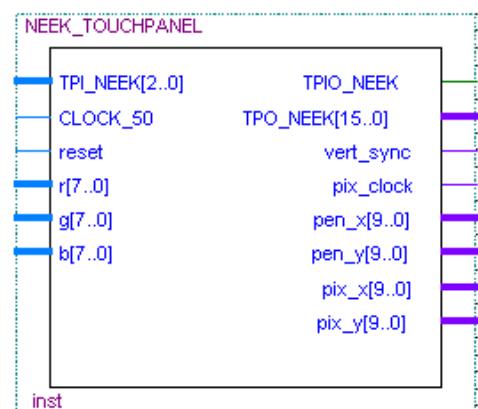
Eigenschaften der Cyclon III Familie von ALTERA								
	EP3C5	EP3C10	EP3C16	<b>EP3C25</b>	EP3C40	EP3C55	EP3C80	EP3C120
<b>Logic Elements (LEs)</b>	5.136	10.320	15.408	<b>24.624</b>	39.600	55.856	81.264	119.088
<b>Total RAM kBits</b>	414	414	504	<b>594</b>	1.134	2.340	2.745	3.888
<b>Embedded Multipliers</b>	23	23	56	<b>66</b>	126	156	244	288
<b>Global Clock Networks</b>	10	10	20	<b>20</b>	20	20	20	20
<b>PLLs</b>	2	2	4	<b>4</b>	4	4	4	4

## 10.2 Das NEEK Board als Ersatz für das DE2 Entwicklungsboard

Entwicklungen für das Touch Panel in Kombination mit dem DE2 Entwicklungsboard können, so weit keine weiteren mechanischen Schalter, Taster LEDs oder 7-Segmentanzeigen auf dem Board benötigt werden, mit kleinen Modifikationen auf das kompaktere „NIOS II Embedded Evaluation Kit“ (NEEK) übertragen werden. Anstelle der I/O-Elemente des DE2 Boards können die in Kapitel 9.2 vorgestellten, virtuellen I/O Module benutzt werden. Dabei ist auf die folgenden Modifikationen zu achten. Zur Ansteuerung des Touch Panels muss das DE2\_TOUCHPANEL Modul durch das NEEK\_TOUCHPANEL ersetzt werden. Auch die geänderte Pinbelegung des FPGAs ist zu berücksichtigen.

## 10.3 Das Modul NEEK\_TOUCHPANEL

Das Modul NEEK\_TOUCHPANEL ersetzt für das NEEK Board das Modul DE2\_TOUCHPANEL des DE2 Boards zur Ansteuerung des TPs. Um den Austausch möglichst einfach zu gestalten, wurde darauf geachtet, alle Funktionen möglichst kompatibel zu gestalten. Das Modul greift auf zum Teil bekannte Komponenten zurück, um die Synchronisation und Initialisierung mit der Konfigurationsdatei des TP's durchzuführen. Diese Dateien sind zum Teil in Verilog realisiert und befinden sich ebenfalls in der DEcore Library:



- adc\_spi\_controller.v (Verilog Datei von Terasic),
- lcd\_spi\_controller.v (Verilog Datei von Terasic),
- Reset\_Delay.v (Verilog Datei von Terasic),
- PEN\_ADJ.VHD (Anpassung der Stiftpositionen an das LCD Koordinatensystem).
- TP\_SYNC\_NEEK.VHD (Generierung der Synchronisationssignale auf dem NEEK Board),

Die letzte Komponente ist neu und speziell für das NEEK Board notwendig. Die Anschlüsse haben die folgenden Funktionen:

NEEK_TOUCHPANEL	
Anschluss	Funktion
<b>IO-Port:</b>	
TPIO_NEEK	I/O HMSC Adapter (fest verdrahtet) (PIN T2)
<b>Eingänge:</b>	
TPI_NEEK[2..0]	I/O HMSC Adapter (fest verdrahtet) (PIN XX)
CLOCK_50	Externes 50 MHz Taktsignal (Systemtakt) (PIN V9)
R[7..0]	8 Bit Farbwert für Rot
G[7..0]	8 Bit Farbwert für Grün
B[7..0]	8 Bit Farbwert für Blau
<b>Ausgänge:</b>	
TPO_NEEK[15..0]	I/O HMSC Adapter (fest verdrahtet) (PIN XX)
HOR_SYNC	Taktausgabe für die vertikale Synchronisation
VERT_SYNC	Taktausgabe für die horizontale Synchronisation
DEN	VGA (Device) Enable (BLANK)
PIX_CLK	Pixeltakt
PIX_X[9..0]	X-Positionsausgabe der LCD Pixel
PIX_Y[9..0]	Y-Positionsausgabe der LCD Pixel
PEN_X[9..0]	X-Positionsausgabe des Berührungs punktes

## **10.4 Die Pin Belegung des NEEK Boards**

Darüber hinaus muss die PIN Belegung für den programmierbaren Baustein Cyclon III FPGA EP3C25F324C6 auf dem NEEK Board importiert werden. Die Datei liegt als editierbare, Excel kompatible Datei **NEEK\_Pin\_Assignments.csv** vor:

```
# Copyright (C) 1991-2009 Altera Corporation
# Your use of Altera Corporation's design tools, logic functions
# and other software and tools, and its AMPP partner logic
# functions, and any output files from any of the foregoing
# (including device programming or simulation files), and any
# associated documentation or information are expressly subject
# to the terms and conditions of the Altera Program License
# Subscription Agreement, Altera MegaCore Function License
# Agreement, or other applicable license agreement, including,
# without limitation, that your use is for the sole purpose of
# programming logic devices manufactured by Altera and sold by
# Altera or its authorized distributors. Please refer to the
# applicable agreement for further details.

# Quartus II Version 9.0 Build 132 02/25/2009 SJ Web Edition
# File: D:\DIGITALE_SYSTEME\Beispielprojekte\NEEK_FRAMES\NEEK_FRAMES.csv
# Generated on: Fri Apr 17 12:07:52 2009

# Note: The column header names should not be changed if you wish to import this .csv file into the Quartus II software.
```

To,Location

# Takt:  
CLOCK\_50,PIN\_V9

#Taster:  
KEY[0],PIN\_F1  
KEY[1],PIN\_F2  
KEY[2],PIN\_A10  
KEY[3],PIN\_B10

#LEDs:  
LED[0],PIN\_P13  
LED[1],PIN\_T1  
LED[2],PIN\_N12  
LED[3],PIN\_N9

#HSMC-Adapter Inputs:  
TPI\_NEEK[0],PIN\_N17  
TPI\_NEEK[1],PIN\_L18  
TPI\_NEEK[2],PIN\_K18

#HSMC-Adapter In/Outputs:  
TPIO\_NEEK,PIN\_T2

#HSMC-Adapter Outputs:  
TPO\_NEEK[0],PIN\_R4  
TPO\_NEEK[1],PIN\_T17  
TPO\_NEEK[2],PIN\_T18  
TPO\_NEEK[3],PIN\_L16  
TPO\_NEEK[4],PIN\_M17  
TPO\_NEEK[5],PIN\_N6  
TPO\_NEEK[6],PIN\_M13  
TPO\_NEEK[7],PIN\_N13  
TPO\_NEEK[8],PIN\_D14

TPO\_NEEK[9],PIN\_R17  
TPO\_NEEK[10],PIN\_M14  
TPO\_NEEK[11],PIN\_L13  
TPO\_NEEK[12],PIN\_M6  
TPO\_NEEK[13],PIN\_V18  
TPO\_NEEK[14],PIN\_U18  
TPO\_NEEK[15],PIN\_R5

# EEPROM I2C:  
eeprom\_scl,PIN\_H6  
eeprom\_dat,PIN\_D3

# Reset:  
pld\_clear\_n,PIN\_N2

# Audiocodec:  
AUD\_BCLK,PIN\_E17  
AUD\_XCK,PIN\_A1  
AUD\_DACDAT,PIN\_R1  
AUD\_DACLRCK,PIN\_R2  
AUD\_ADCDAT,PIN\_A9  
AUD\_ADCLRCK,PIN\_M1  
I2C\_SDAT,PIN\_E1  
I2C\_SCLK,PIN\_F3

#PS/2 ??? Belegung ist falsch ???
HC\_PS2\_CLK,PIN\_M5
HC\_PS2\_DAT,PIN\_T1

# DDR SDRAM Interface:  
mem\_dqs[0],PIN\_U3  
mem\_dqs[1],PIN\_T8  
mem\_dm[0],PIN\_V3  
mem\_dm[1],PIN\_V8

```

mem_ba[0],PIN_V11
mem_ba[1],PIN_V12
mem_cas_n,PIN_T4
mem_cke[0],PIN_R13
mem_cs_n[0],PIN_V1
mem_ras_n,PIN_V16
mem_we_n,PIN_U15
mem_clk[0],PIN_U2
mem_clk_n[0],PIN_V2
mem_addr[0],PIN_U1
mem_addr[1],PIN_U5
mem_addr[2],PIN_U7
mem_addr[3],PIN_U8
mem_addr[4],PIN_P8
mem_addr[5],PIN_P7
mem_addr[6],PIN_P6
mem_addr[7],PIN_T14
mem_addr[8],PIN_T13
mem_addr[9],PIN_V13
mem_addr[10],PIN_U17
mem_addr[11],PIN_V17
mem_addr[12],PIN_U16
mem_dq[0],PIN_U4
mem_dq[1],PIN_V4
mem_dq[2],PIN_R8
mem_dq[3],PIN_V5
mem_dq[4],PIN_P9
mem_dq[5],PIN_U6
mem_dq[6],PIN_V6
mem_dq[7],PIN_V7
mem_dq[8],PIN_U13
mem_dq[9],PIN_U12
mem_dq[10],PIN_U11
mem_dq[11],PIN_V15
mem_dq[12],PIN_U14
mem_dq[13],PIN_R11
mem_dq[14],PIN_P10
mem_dq[15],PIN_V14

# Flash and SSRAM bridge:
tristate_bus_address[1],PIN_E12
tristate_bus_address[2],PIN_A16
tristate_bus_address[3],PIN_B16
tristate_bus_address[4],PIN_A15
tristate_bus_address[5],PIN_B15
tristate_bus_address[6],PIN_A14
tristate_bus_address[7],PIN_B14
tristate_bus_address[8],PIN_A13
tristate_bus_address[9],PIN_B13
tristate_bus_address[10],PIN_A12
tristate_bus_address[11],PIN_B12
tristate_bus_address[12],PIN_A11
tristate_bus_address[13],PIN_B11
tristate_bus_address[14],PIN_C10
tristate_bus_address[15],PIN_D10
tristate_bus_address[16],PIN_E10
tristate_bus_address[17],PIN_C9
tristate_bus_address[18],PIN_D9
tristate_bus_address[19],PIN_A7
tristate_bus_address[20],PIN_A6
tristate_bus_address[21],PIN_B18
tristate_bus_address[22],PIN_C17

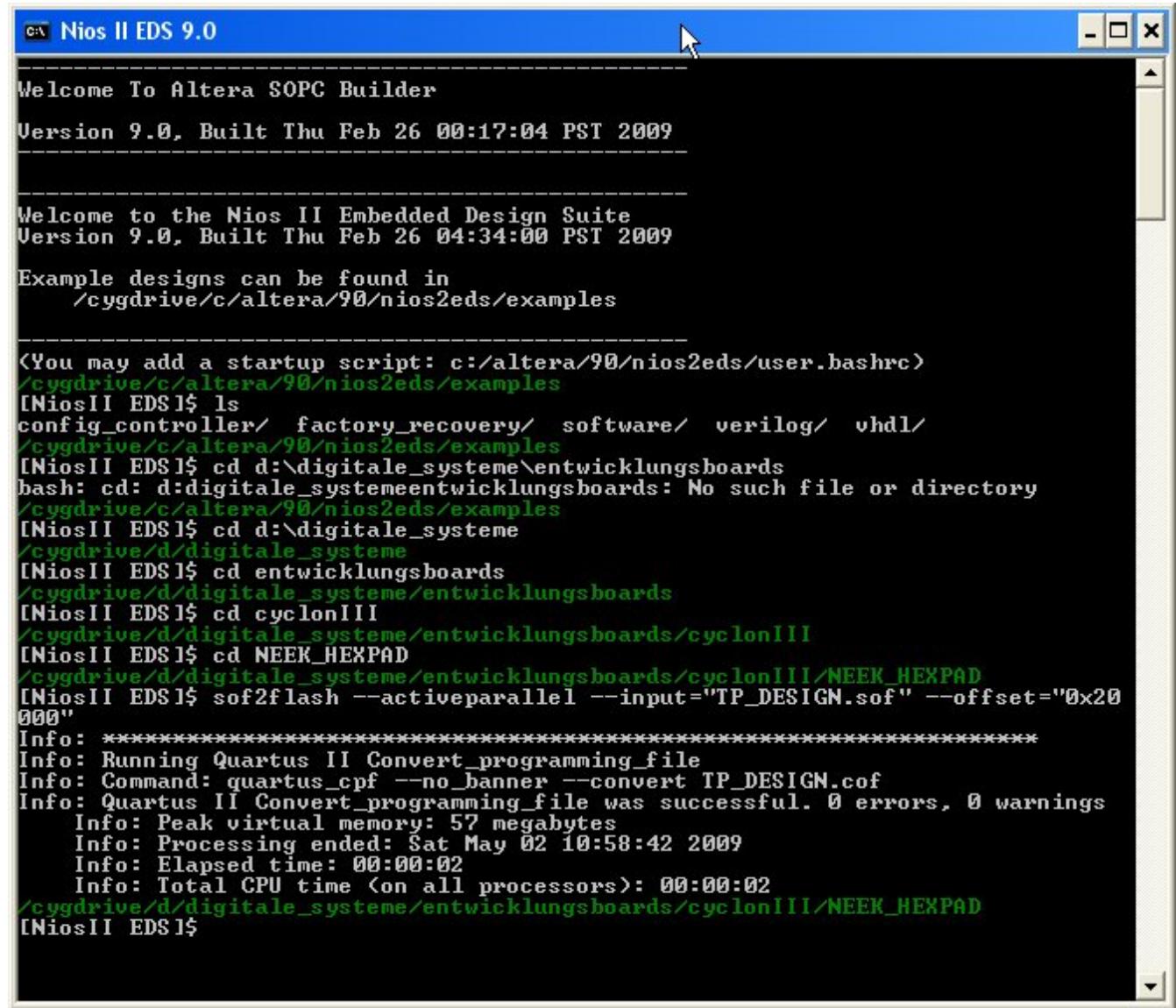
tristate_bus_address[23],PIN_C18
flash_select_n,PIN_E2
flash_read_n,PIN_D17
flash_write_n,PIN_D18
flash_reset_n,PIN_C3
tristate_bus_data[0],PIN_H3
tristate_bus_data[1],PIN_D1
tristate_bus_data[2],PIN_A8
tristate_bus_data[3],PIN_B8
tristate_bus_data[4],PIN_B7
tristate_bus_data[5],PIN_C5
tristate_bus_data[6],PIN_E8
tristate_bus_data[7],PIN_A4
tristate_bus_data[8],PIN_B4
tristate_bus_data[9],PIN_E7
tristate_bus_data[10],PIN_A3
tristate_bus_data[11],PIN_B3
tristate_bus_data[12],PIN_D5
tristate_bus_data[13],PIN_B5
tristate_bus_data[14],PIN_A5
tristate_bus_data[15],PIN_B6
tristate_bus_data[16],PIN_C16
tristate_bus_data[17],PIN_D12
tristate_bus_data[18],PIN_E11
tristate_bus_data[19],PIN_D2
tristate_bus_data[20],PIN_E13
tristate_bus_data[21],PIN_E14
tristate_bus_data[22],PIN_A17
tristate_bus_data[23],PIN_D16
tristate_bus_data[24],PIN_C12
tristate_bus_data[25],PIN_A18
tristate_bus_data[26],PIN_F8
tristate_bus_data[27],PIN_D7
tristate_bus_data[28],PIN_F6
tristate_bus_data[29],PIN_E6
tristate_bus_data[30],PIN_G6
tristate_bus_data[31],PIN_C7
ssram_outputenable_n,PIN_E9
ssram_chipenable_n,PIN_F9
ssram_bw_n[0],PIN_F10
ssram_bw_n[1],PIN_F11
ssram_bw_n[2],PIN_F12
ssram_bw_n[3],PIN_F13
ssram_adsc_n,PIN_F7
ssram_bwe_n,PIN_G13
ssram_clk,PIN_A2

```

## **10.5 Starten eines Projektes von der SD-Karte**

Während das DE2 Board in der Regel von einem externen Rechner über die USB-Blaster Schnittstelle programmiert werden muss, besteht beim NEEK Board auch die Möglichkeit, das Programm neben anderen Programmen und Anwendungen auf einer **SD-Karte** (maximal 2 GB, nicht vom Typ SDHC) vorher zu speichern und bei Bedarf auszuwählen und zu starten. Dies erfordert jedoch einige Vorarbeiten, damit die richtigen Daten am richtigen Ort auf der SD-Karte landen. Die dazu notwendigen Schritte sollen im Folgenden beschrieben werden.

Um aus der existierende \*.sof Datei die benötigte \*.flash Datei zu generieren, die auf einer SD-Karte gespeichert werden kann, muss als erstes die „**NIOS II Konsole**“ im „Nios II EDS“-Ordner (z.B.: C:\altera\90\nios2eds\Nios II Command Shell.bat) gestartet werden, worauf sich die ersten Zeilen des unten gezeigten Fensters öffnen.



```
c:\ Nios II EDS 9.0
Welcome To Altera SOPC Builder
Version 9.0, Built Thu Feb 26 00:17:04 PST 2009

-----
Welcome to the Nios II Embedded Design Suite
Version 9.0, Built Thu Feb 26 04:34:00 PST 2009

Example designs can be found in
/cygdrive/c/altera/90/nios2eds/examples

<You may add a startup script: c:/altera/90/nios2eds/user.bashrc>
/cygdrive/c/altera/90/nios2eds/examples
[NiosII EDS]$ ls
config_controller/ factory_recovery/ software/ verilog/ vhdl/
/cygdrive/c/altera/90/nios2eds/examples
[NiosII EDS]$ cd d:\digitale_systeme\entwicklungsboards
bash: cd: d:digitale_systemeentwicklungsboards: No such file or directory
/cygdrive/c/altera/90/nios2eds/examples
[NiosII EDS]$ cd d:\digitale_systeme
/cygdrive/d/digitale_systeme
[NiosII EDS]$ cd entwicklungsboards
/cygdrive/d/digitale_systeme/entwicklungsboards
[NiosII EDS]$ cd cyclonIII
/cygdrive/d/digitale_systeme/entwicklungsboards/cyclonIII
[NiosII EDS]$ cd NEEK_HEXPAD
/cygdrive/d/digitale_systeme/entwicklungsboards/cyclonIII/NEEK_HEXPAD
[NiosII EDS]$ sof2flash --activeparallel --input="TP_DESIGN.sof" --offset="0x2000"
Info: ****
Info: Running Quartus II Convert_programming_file
Info: Command: quartus_cpf --no_banner --convert TP_DESIGN.cof
Info: Quartus II Convert_programming_file was successful. 0 errors, 0 warnings
  Info: Peak virtual memory: 57 megabytes
  Info: Processing ended: Sat May 02 10:58:42 2009
  Info: Elapsed time: 00:00:02
    Info: Total CPU time (on all processors): 00:00:02
/cygdrive/d/digitale_systeme/entwicklungsboards/cyclonIII/NEEK_HEXPAD
[NiosII EDS]$
```

Mit `cd ordnername` kann nun wie oben gezeigt zu dem gewünschten Projekt Verzeichnis navigiert werden. Der Befehl

```
sof2flash --activeparallel --input="NEEK_FRAMES.sof" --offset="0x20000"
```

erzeugt dann die benötigte FLASH-Datei. Diese ist in `*_hw.flash` umzubenennen. Sie beschreibt die Konfiguration der Hardware.

Darüber hinaus wird (formal) eine „Software“ Flash Datei benötigt. Da diese in reinen VHDL-Designs ohne implementiertem NIOS Prozessor jedoch nicht gelesen wird, kann dazu eine beliebige Datei wie **Dummy\_sw.flash** benutzt werden. Zusätzlich kann in einer formatierten Textdatei **info.txt** eine erläuternde Beschreibung untergebracht werden. Um die richtige Formatierung beizubehalten, sollte diese Datei z.B. in WORD editiert werden.

Alle drei Dateien sind in ein neues, anschaulich benanntes Unterverzeichnis im Ordner „**Altera\_EEK\_Applications**“ auf der SD-Karte zu kopieren. Der Name dieses Unterverzeichnisses wird beim Einschalten des NEEK Boards bei eingelegter SD-Karte im Anwendungsverzeichnis (**Application Selektor**) mit aufgeführt und kann so über den Touch Screen ausgewählt und geladen werden.

## 11. Literatur

Zum Praktikum und den verschiedenen Entwicklungsboards (UP1 - 3, DE1, DE2) von Altera ist folgendes Buch in der 4. Auflage besonders empfehlenswert:

- Hamblen, J.O.; Hall, T.S; Furman, M.D.; 2008: "Rapid Prototyping of Digital Systems - SOPC Edition. Springer Verlag.

Dieses Buch bildet auch die Basis vieler Praktikumsversuche und einige der hier benutzten VHDL Module wurden dort entnommen. Auch die Internetseite des Autors ist empfehlenswert:

<http://users.ece.gatech.edu/~hamblen/DE2/>

Des Weiteren sei auf die Internetseite von Altera <http://www.altera.com> aufmerksam gemacht, wo sich vieles an weiterer Information, Literatur und (Online-) Tutorien zu unterschiedlichsten Fragestellungen finden lässt.

Das DE2 Board selbst stammt von der Taiwanesischen Firma Terasic, wo dieses auch bestellt werden kann. Hier <http://www.terasic.com> finden sich auch weiteres Zubehör, Datenblätter etc.