CPLD Einführung

Version: 0.0.1
Datum: 27.01.2013
Autor: Werner Dichler

Inhalt

Inhalt 2
Programmierbare Logik 3
Hersteller / Typ 3
Chip Aufbau 4
Test-Konfiguration 11
Projekt erstellen 11
Synthese 17
Simulation 24

Darstellungen wurden z.T. von Xilinx Datenblättern übernommen!

Programmierbare Logik

Hersteller / Typ

Xilinx Inc.
Website: www.xilinx.com

XC9572XL-10VQG44C

XC9572XL	Geräte-Klasse
-10	Geschwindigkeitseinstufung (speed grade)
VQG	Gehäuse Typ
44	Anzahl der Pins
C	Temperatur Bereich

44-Pin, Flash-basierender CMOS CPLD

Grundlegende Eigenschaften

Parameter	Wert
Makrozellen:	72
Register:	72
nutzbare Gatter:	1600
I/O Pins:	34 (3x GCK, 2x GTS, 1x GSR)
Programmierzyklen:	10000
5V tolerante IO-Pins:	ja
Gehäuse:	PLCC, VQFP
Temperatur Bereich:	Industrie-Bereich: -40°C bis +85°C, Kommerziell: 0°C bis +70°C

Elektrische Eigenschaften

Parameter	Wert
Betriebsspannung:	+3.0V bis +3.6V
Ausgangstreiber Versorgung:	+2.3V bis +2.7V / 3.0V bis 3.6V
Low-Eingangsspannung:	0.0V bis 0.8V
High-Eingangsspannung:	2.0V bis 5.5V
Low-Ausgangsspannung:	Max. 0.4V
High-Ausgangsspannung:	Min. 2.4V

Timing Eigenschaften

Parameter	Wert
IO to output valid (T_{PD}):	Max. 10.0ns
Internal operating frequency (f_{System}):	Max. 100MHz
IO setup time before p-term clock input (T_{PSU}):	Min. 2.1ns
IO hold time after p-term clock input (T_{PH}):	Min. 4.4ns
P-term clock output valid (T_{PCO}):	Max. 10.2ns
P-term S/R to output valid (T_{PAO}):	Max. 15.3ns
Asynchr. preset/reset pulse width (T_{APRPW}):	Min. 7.0ns
P-term clock pulse width (T_{PLH}):	Min. 7.0ns

Chip Aufbau

Interne Beschaltung

Der CPLD (Complex Programmable Logic Device) von Xilinx besteht grundsätzlich aus einer Ein-/Ausgangsbeschaltung, einem globalen Verbindungsnetzwerk und den Funktionsblöcken. Ein Funktionsblock besteht aus 18 Makrozellen, die jeweils nahezu identisch aufgebaut sind.

Sämtliche IO-Pins können als Ein- und auch als Ausgang verwendet werden. Einige Pins haben eine Zweitfunktion: globale Taktleitungen, globale Set-/Reset-Eingänge und globale Output-Enable. Werden diese Signale benötigt, so müssen die jeweiligen Pins reserviert werden.

Über eine JTAG Schnittstelle wird die Konfiguration übertragen. Diese Pins sind dezidiert für diese Funktion ausgelegt und können nicht anderweitig verwendet werden.

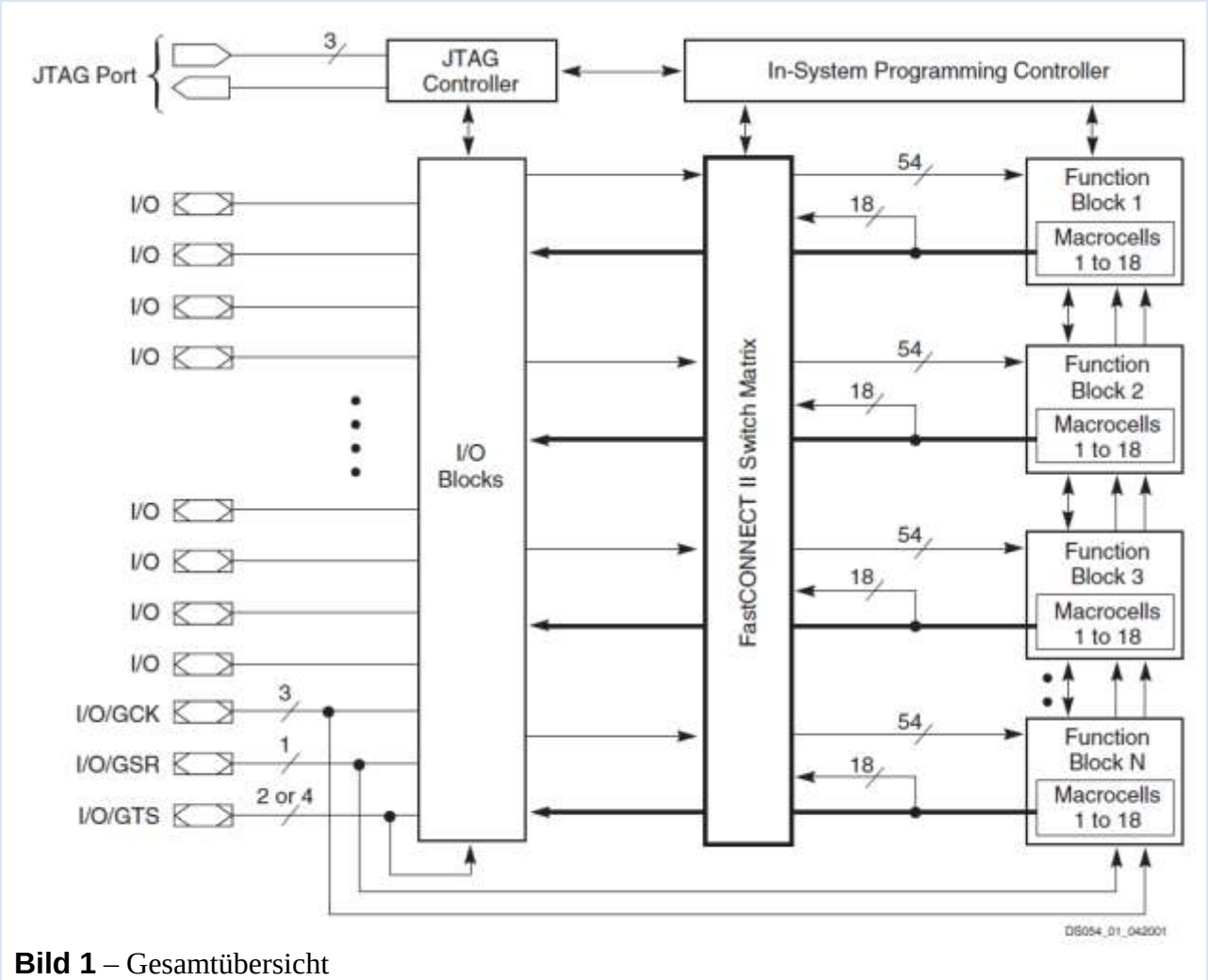

Bild 1 – Gesamtübersicht

Das globale Verbindungsnetzwerk dient für unterschiedliche Eingangskonfigurationen der Makrozellen. Zur Auswahl stehen sämtliche IO-Eingänge (34) und die Ausgänge der insgesamt 72 Makrozellen (4 Funktionsblöcke zu 18 Makrozellen).
Aus diesen 34+72 Eingängen können 54 ausgewählt werden, die einem Funktionsblock zur Verfügung stehen. Die ausgewählten Signale stehen des weiteren auch noch negiert zur Verfügung. Die 18 Makrozellen innerhalb eines Funktionsblocks haben die gleiche Eingangs-konfiguration.

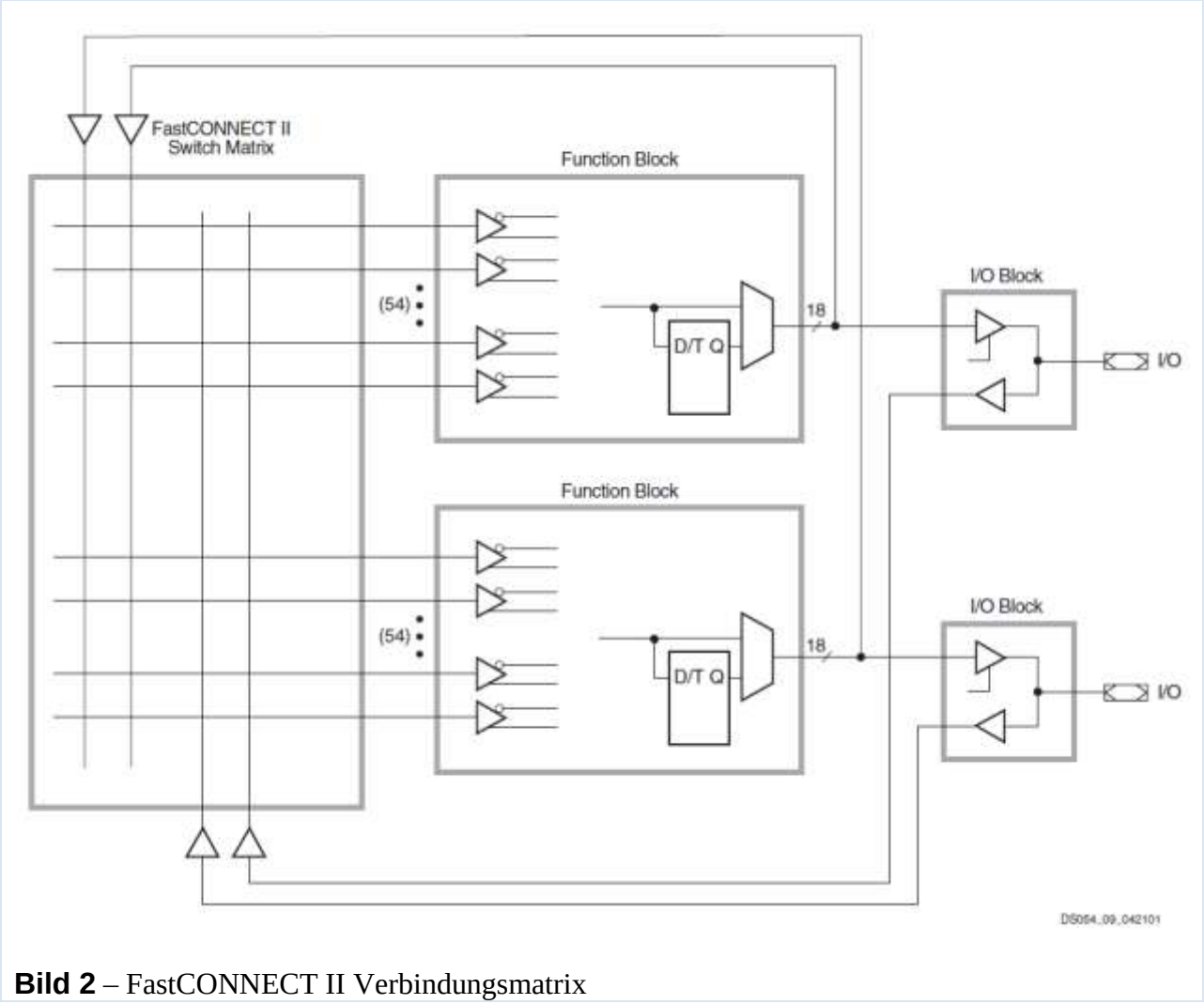

Bild 2 – FastCONNECT II Verbindungsmatrix

Eine Makrozelle besteht grundsätzlich aus dem kombinatorischen Teil, einer Signal-Auswahl-Logik und dem Register.
Jede Makrozelle kann unterschiedlich verwendet werden. Sie kann als Register-Logik oder als eine reine kombinatorische Logik verwendet werden. Der kombinatorische Teil besteht aus auswählbaren Und-Verknüpfungen mit anschließender Oder-Verknüpfung. Der gesamte Ausdruck wird dann noch mit einem Exklusiv-Oder verbunden.
Die Eingänge des Registers können mit unterschiedlichen Signalen verbunden werden. Die Eingänge S, R, D und C können mit einem globalen Signal oder mit einem Ausgang der kombinatorischen Verknüpfung verbunden werden.
Jede Makrozelle liefert ein Signal zum globalen Verbindungsnetzwerk. Da unterschiedliche Chips unterschiedliche Anzahl an Pins haben, werden immer nur bestimmte Makrozellen-Ausgänge direkt mit den IO-Pins verbunden. Bei einem Chip mit 34 IO-Pins z.B. nur 34 Makrozellen.

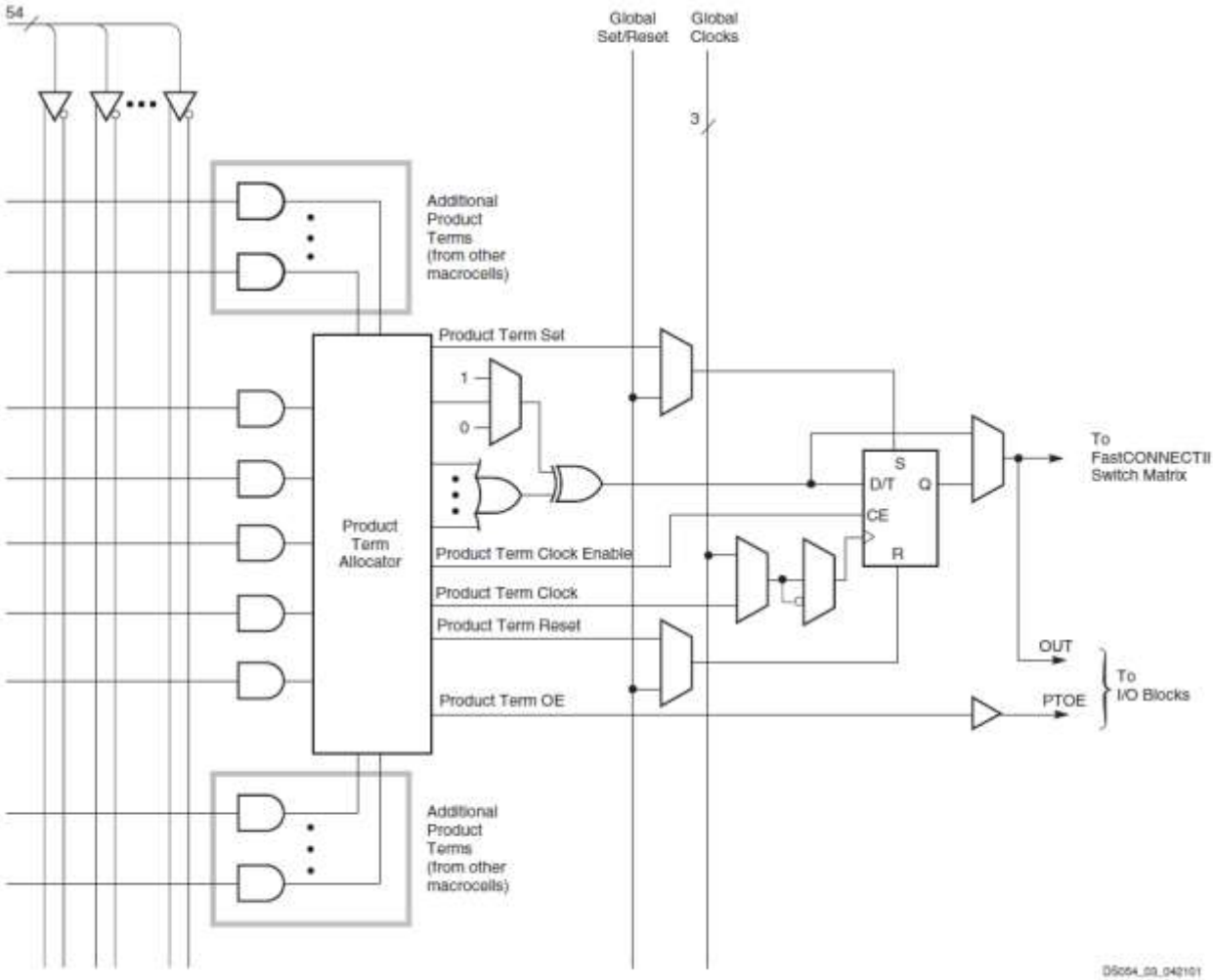

Bild 3 – Makrozelle

Um den kombinatorischen Teil einer Makrozelle noch flexibler zu gestalten, können von benachbarten Makrozellen noch zusätzliche Verknüpfungen verwendet werden.

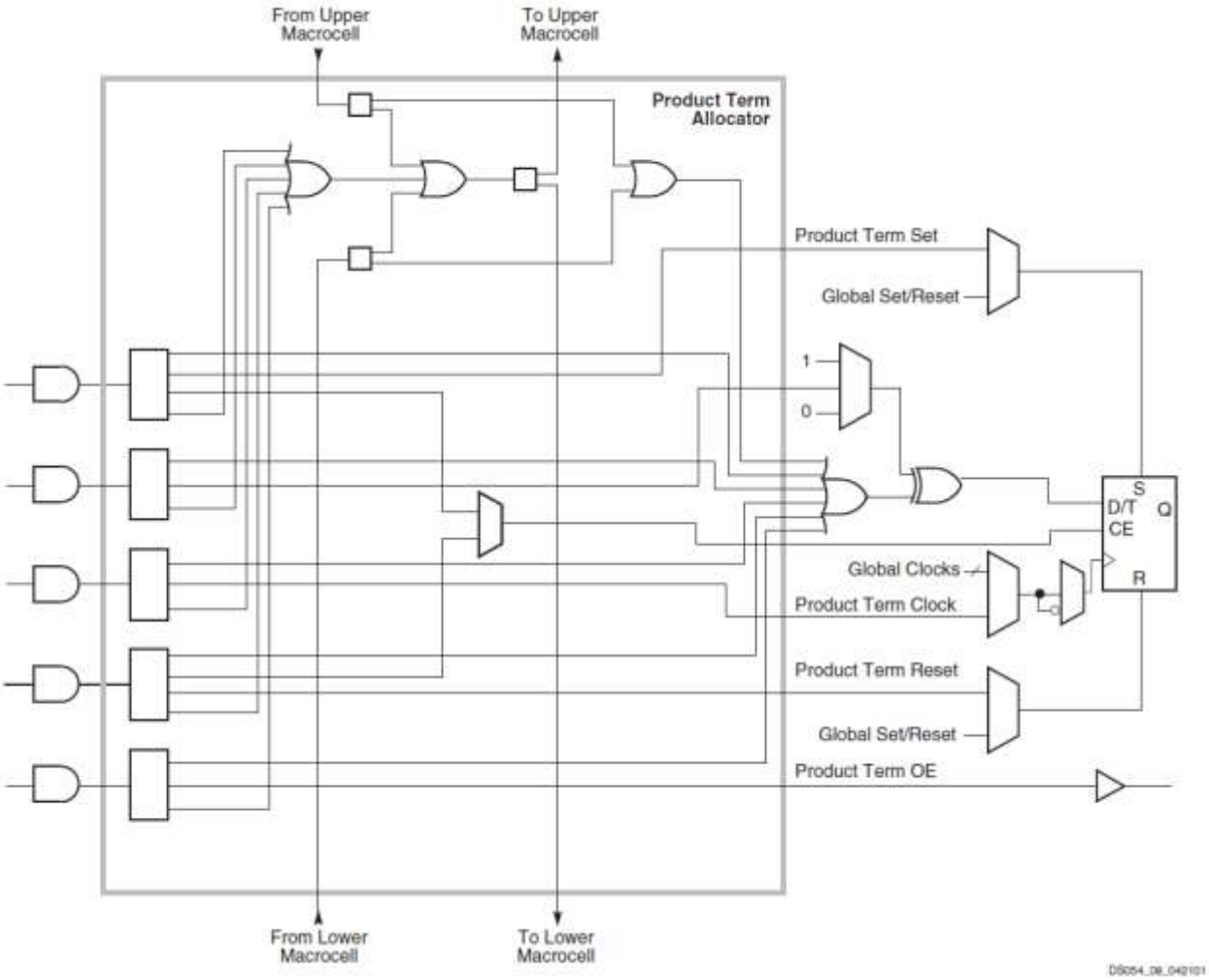

Bild 4 – Product Term Allocator - Logik

Jeder IO-Pin kann als Eingang und Ausgang verwendet werden. Der Eingangspuffer ist immer verbunden. Die Ausgangs-Funktion kann durch den Ausgangspuffer aktiviert und deaktiviert werden. Die Anstiegs-Zeit kann für externe Beschaltungen optimiert werden.

Wird ein Pin nicht benötigt, so kann man diesen offen lassen, über einen großen Widerstand an ein Potential schalten (Bus-Hold) oder als zusätzlichen Masse-Pin verwenden. Die Bus-Hold-Funktion ist bei einer Eingangskonfiguration immer aktiviert. Diese Funktion haltet den letzten aktiven Zustand mit einer hochohmigen Rückkopplung. Somit werden nicht verbundene Eingänge auf ein fixes Potential gehalten.

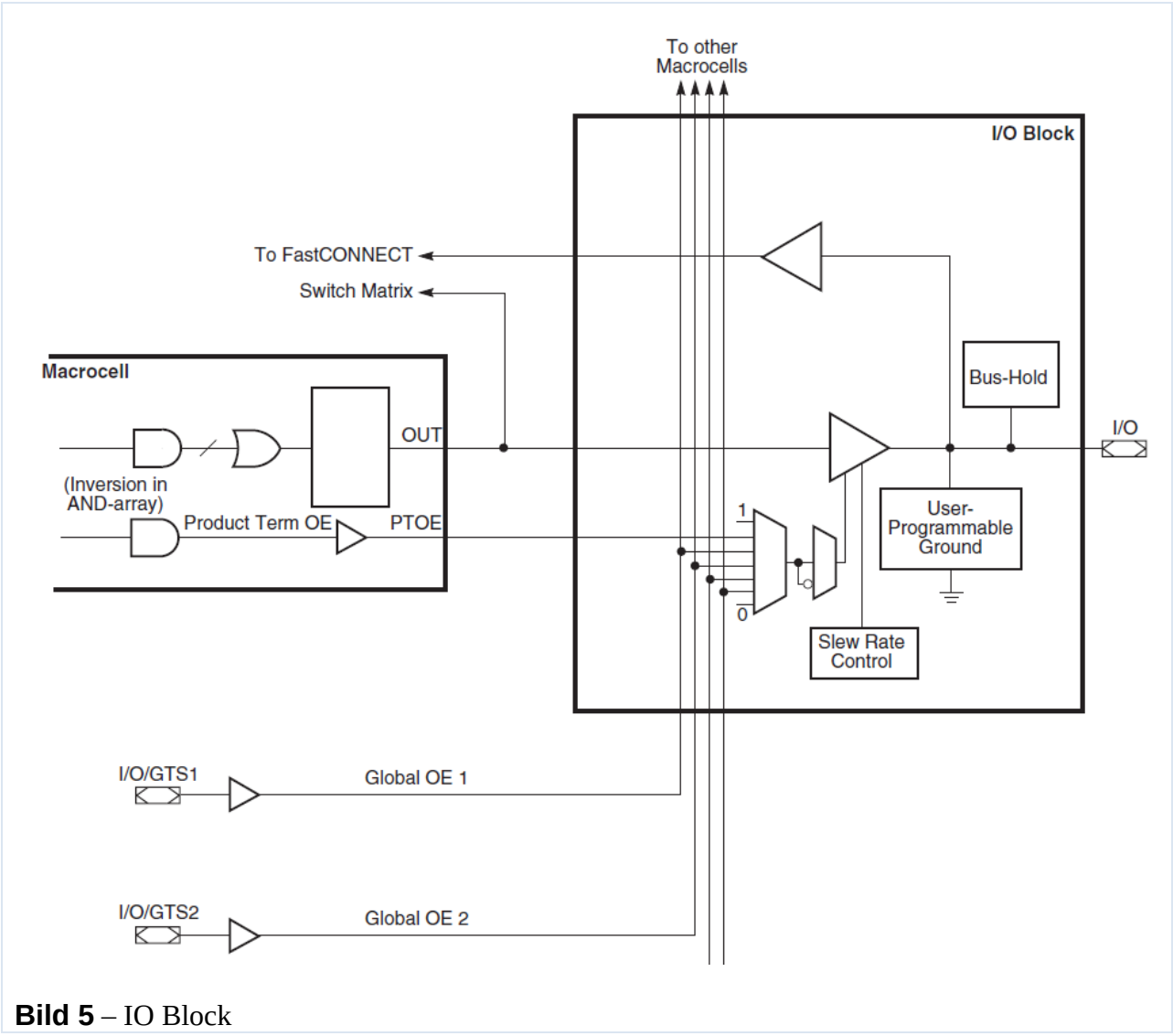

Bild 5 – IO Block

Interne Timing Parameter

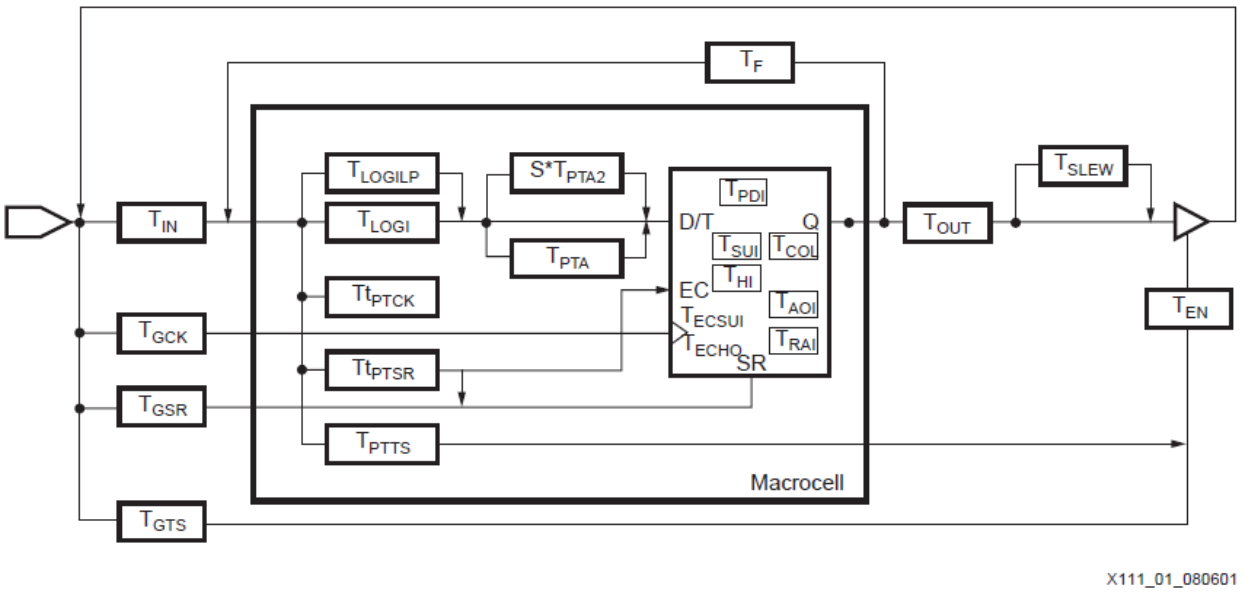

Bild 6 – Timing Modell

Symbol	Parameter	Wert
Buffer Delays		
T_{IN}	Input buffer delay	Max. 3.5ns
T_{GCK}	GCK buffer delay	Max. 1.8ns
T_{GSR}	GSR buffer delay	Max. 4.5ns
T_{GTS}	GTS buffer delay	Max. 7.0ns
T_{OUT}	Output buffer delay	Max. 3.0ns
T_{EN}	Output buffer enable delay	Max. 0.0ns
Product Term Control Delays		
T_{PTCK}	Product term clock delay	Max. 2.7ns
T_{PTSR}	Product term set/reset delay	Max. 1.8ns
T_{PTTS}	Product term 3-state delay	Max 7.5ns
Internal Register and Combinatorial Delays		
T_{PDI}	Combinatorial logic propagation delay	Max. 1.7ns
T_{SUI}	Register setup time	Min. 3.0ns
T_{HI}	Register hold time	Min. 3.5ns
T_{ECSUI}	Register clock enable setup time	Min. 3.0ns
T_{ECHO}	Register clock enable hold time	Min. 3.5ns
T_{COI}	Register clock to output valid time	Max. 1.0ns
T_{AOI}	Register async. SR to output delay	Max. 7.0ns
T_{RAI}	Register async. SR recover before clock	Min. 10.0ns
T_{LOGI}	Internal logic delay	Max. 1.8ns
T_{LOGILP}	Internal low power logic delay	Max. 7.3ns
Feedback Delays		
T_F	Fast connect feedback delay	Max. 4.2ns
Time Adders		
T_{PTA}	Incremental product term allocator delay	Max. 1.0ns
T_{SLEW}	Slew-rate limited delay	Max. 4.5ns

Pin-Belegung

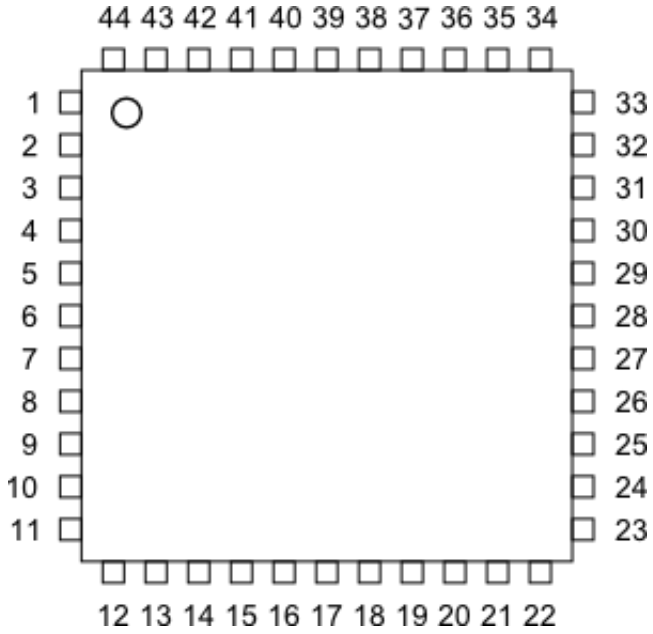

Bild 7 – Pin-Belegung

Pin Nummer	Pin Funktion	Pin Nummer	Pin Funktion
1	IO1-14 / GCK3	23	IO4-14
2	IO1-15	24	TDO
3	IO1-17	25	GND
4	GND	26	VCCIO
5	IO3-2	27	IO4-15
6	IO3-5	28	IO4-17
7	IO3-8	29	IO2-2
8	IO3-9	30	IO2-5
9	TDI	31	IO2-6
10	TMS	32	IO2-8
11	TCK	33	IO2-9 / GSR
12	IO3-11	34	IO2-11 / GTS2
13	IO3-14	35	VCCINT
14	IO3-15	36	IO2-14 / GTS1
15	VCCINT	37	IO2-15
16	IO3-17	38	IO2-17
17	GND	39	IO1-2
18	IO3-16	40	IO1-5
19	IO4-2	41	IO1-6
20	IO4-5	42	IO1-8
21	IO4-8	43	IO1-9 / GCK1
22	IO4-11	44	IO1-11 / GCK2

Test-Konfiguration

Projekt erstellen

Die Entwicklungsumgebung kann auf der Website von Xilinx herunter geladen werden. Für einfache Projekte ist eine freie Webpack-Lizenz ausreichen.
Nach dem Installieren wird der "ISE Project Navigator" geöffnet und ein neues Projekt angelegt. Im ersten Wizard-Fenster wird der Projekt-Name und der Speicherort eingetragen. Wird die Logik mit dem Schematic-Editor aufgebaut, so wählt man "Schematic" als Top-Level-Source-Typ.
Das zweite Wizard-Fenster beinhalten Angaben zum verwendeten CPLD-Chip. Dabei ist es wichtig den richtigen Typen, das richtige Gehäuse und die richtige Geschwindigkeitseinstufung auszuwählen. Wurde ein falsches Gehäuse ausgewählt, so kann die Pin-Zuordnung nicht richtig ausgeführt werden.
Die Geschwindigkeitseinstufung ist für die genaue Zeitsimulation wichtig. Die CPLDs werden nach den maximalen Delay-Zeiten der einzelnen Blöcke sortiert. Bei einem Chip mit einem definierten Speed-Grade von z.B. 10 werden angegebene Zeiten nicht überschritten. Je kleiner der Speed-Grade, desto kleiner sind die maximalen Verzögerungszeiten und der Chip kann mit einem höheren Takt verwendet werden.

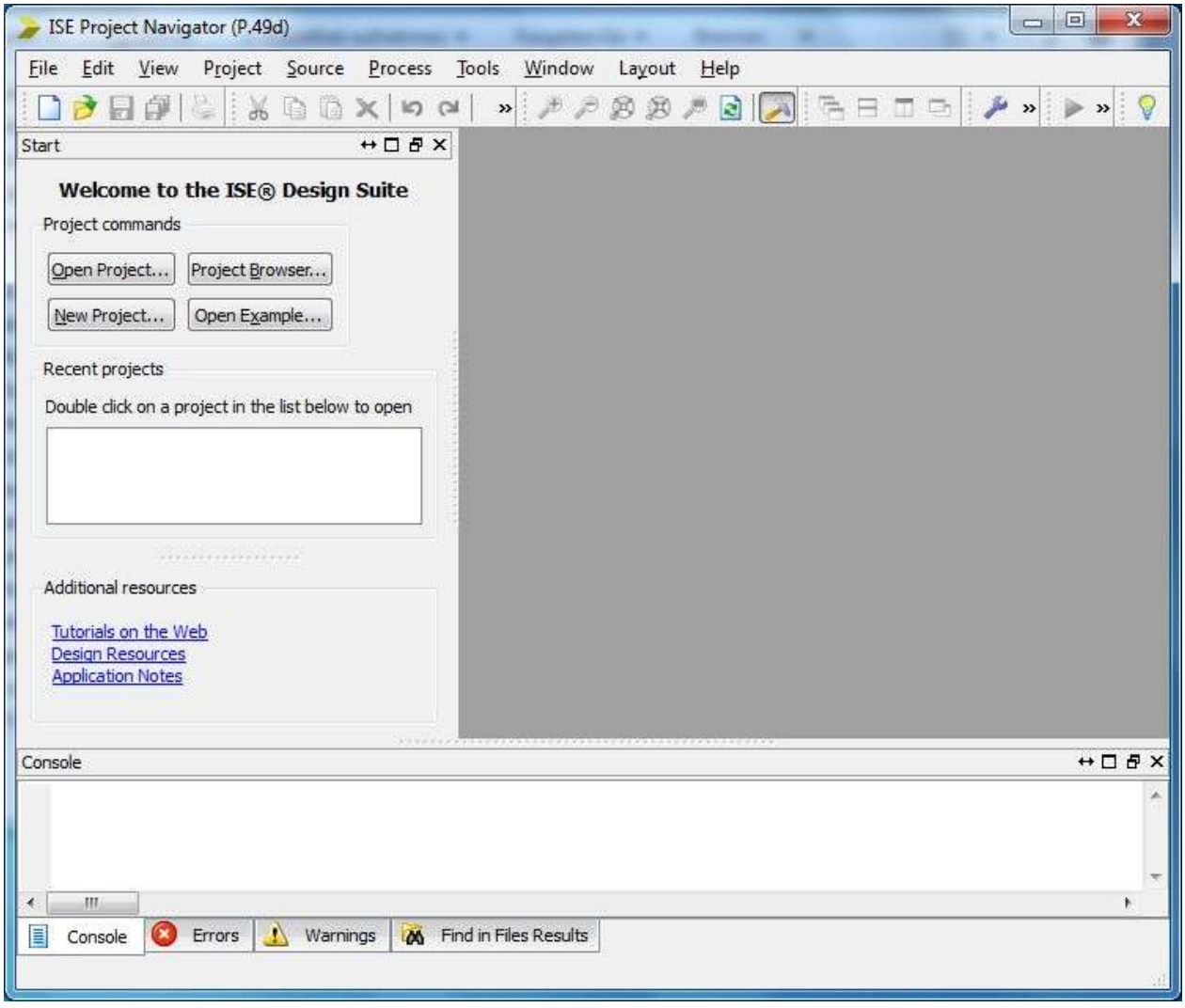

Bild 8 – Entwicklungsumgebung (ISE Project Navigator)

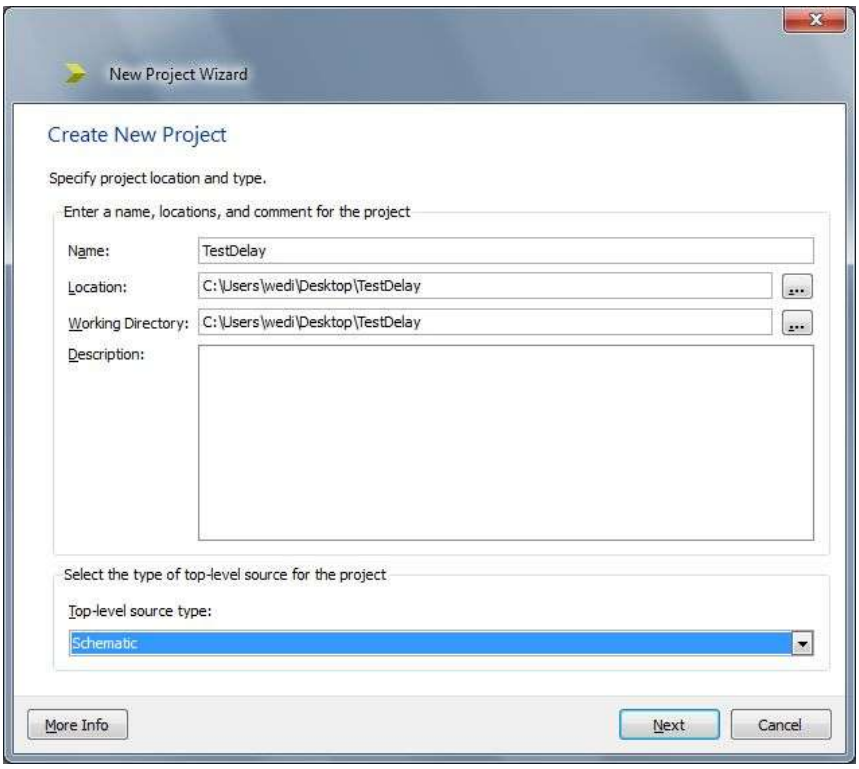

Bild 9 – Projekt-Wizard

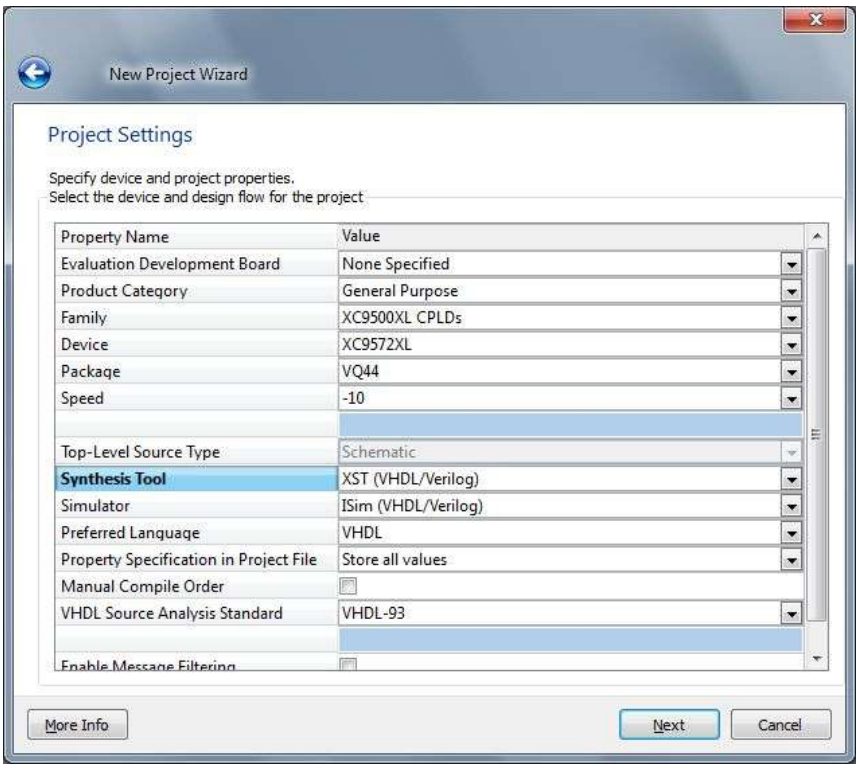

Bild 10 – Projekt-Wizard

Für die Logikbeschreibung wird eine neue Schematic-Datei angelegt. Mit dem Schematic-Editor kann die Logik mit einfachen Symbolen zusammengebaut werden. Dieser Logikentwurf ist für einfache Projekte und für Top-Level-Designs hilfreich. Somit erhält man beim Top-Level-Design sogleich die Diagramme für die Dokumentation.

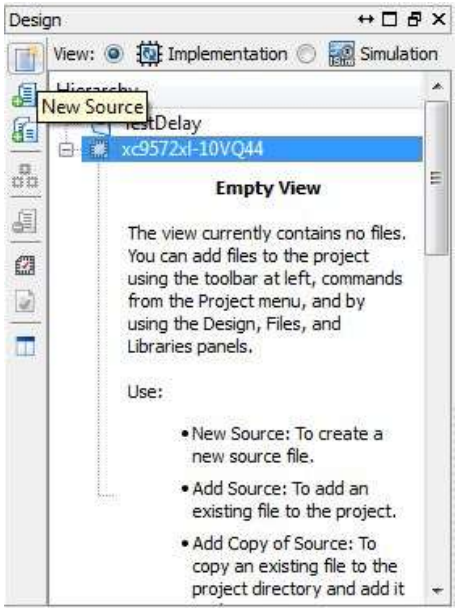

Bild 11 – Neue Datei anlegen

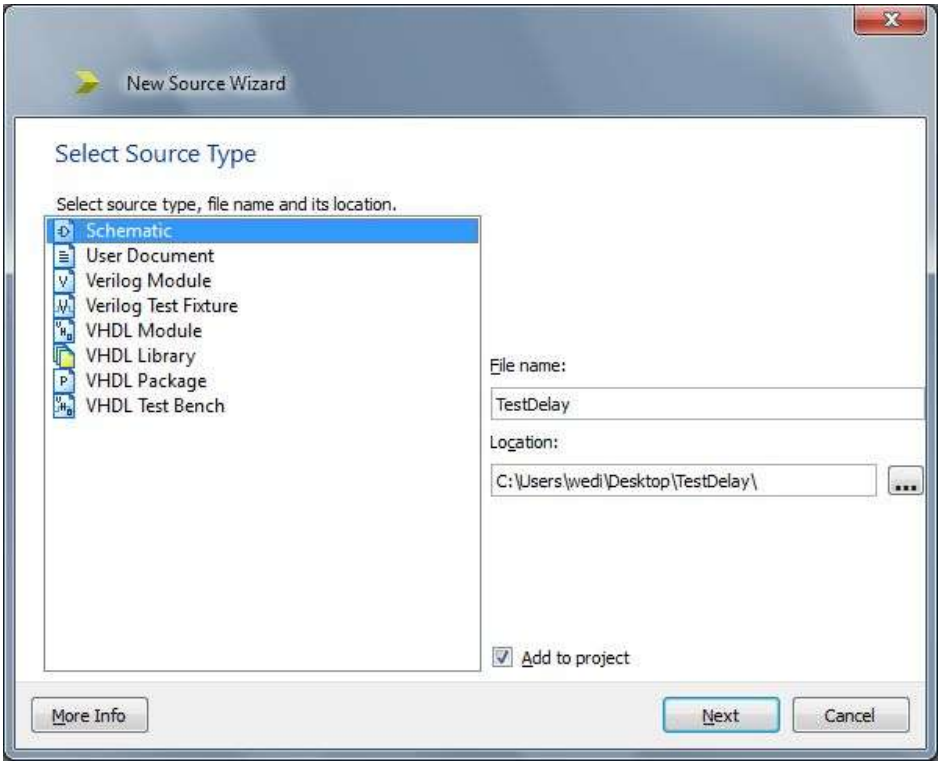

Bild 12 – Neue Schematic Datei

Beim Logikentwurf werden die benötigten Symbole platziert und mit dem "Add Wire"-Tool verbunden. Ein- und Ausgänge werden mit dem "Add IO Marker"-Tool hinzugefügt. Bei einem Top-Level-Design werden normalerweise Ein-/Ausgangspuffer benötigt. Diese werden normalerweise automatisch bei der Synthese hinzugefügt, können aber auch manuell platziert werden.

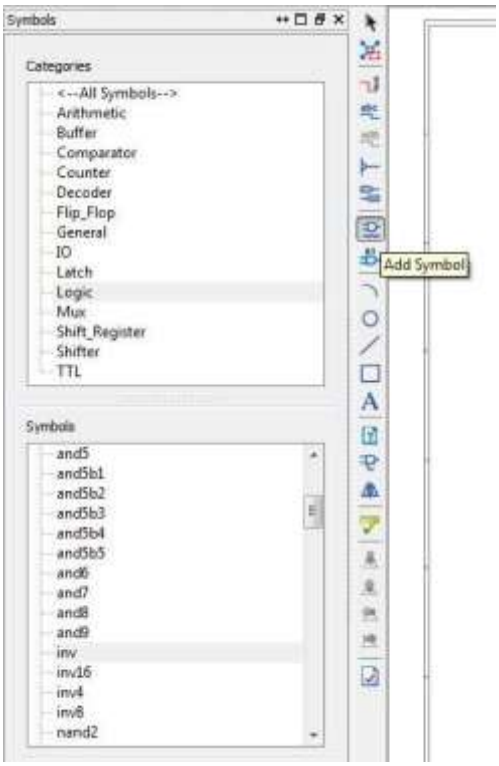

Bild 13 – Symbol platzieren

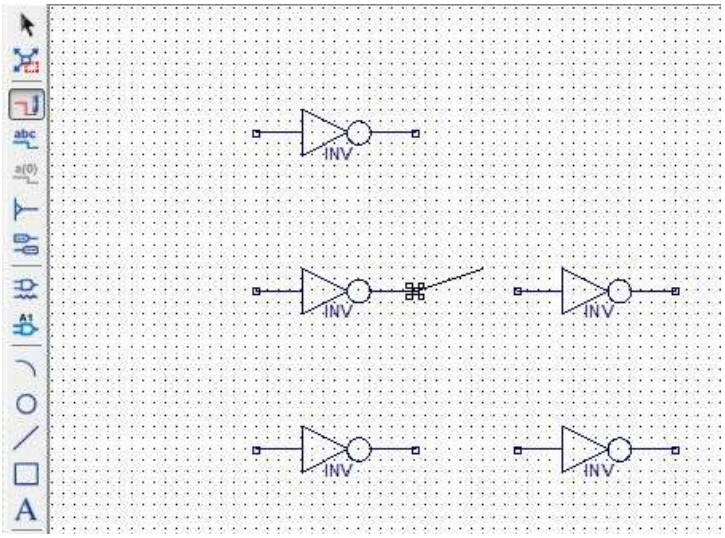

Bild 14 – Verbindungen zeichnen

Durch Doppel-Klick auf das Eingangs-Symbol kann ein anderer Name zugewiesen werden. Dieses Zuweisen ändert eigentlich den Namen der Leitung zwischen dem Eingangs-Symbol und dem Oder-Symbol. Eine Leitung entspricht einem Signal in VHDL.
Es können sämtliche Signale benannt werden. Damit man bei der Simulation auch innere Zustände findet und darstellen kann, sollte man jedes zu betrachtende Signal benennen.

Damit die Symbole etwas geordneter platziert werden können, werden "Align"-Werkzeuge zur Verfügung gestellt (Edit → Align Instances / Align I/O Marker Origins / Align Text). Für zusätzliche Erklärungen oder Kommentare stehen diverse Zeichen-Funktionen zur Verfügung. Ein vordefinierter Header kann unter (Add Symbol → General → title) gefunden werden.

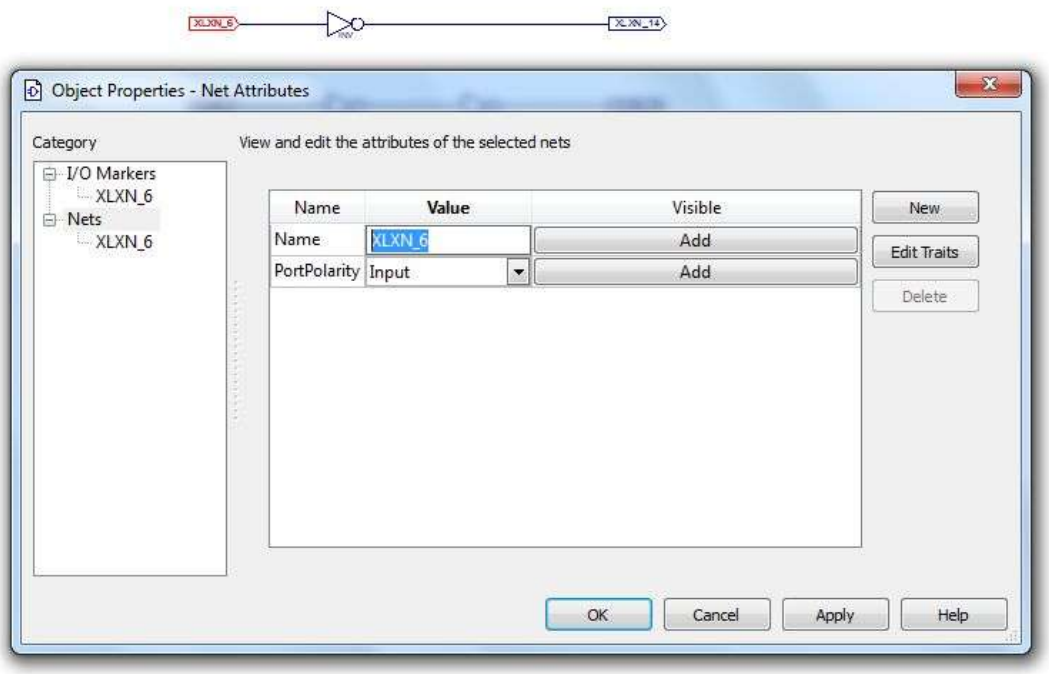

Bild 15 – Benennung der Ein-/Ausgänge

Eine Logik kann durch "Check Schematic" auf Fehler überprüft werden. Sobald die Logik fertiggestellt und keine Fehler mehr vorhanden sind, kann der Entwurf für weitere Prozesse verwendet werden (Synthese, Simulation).

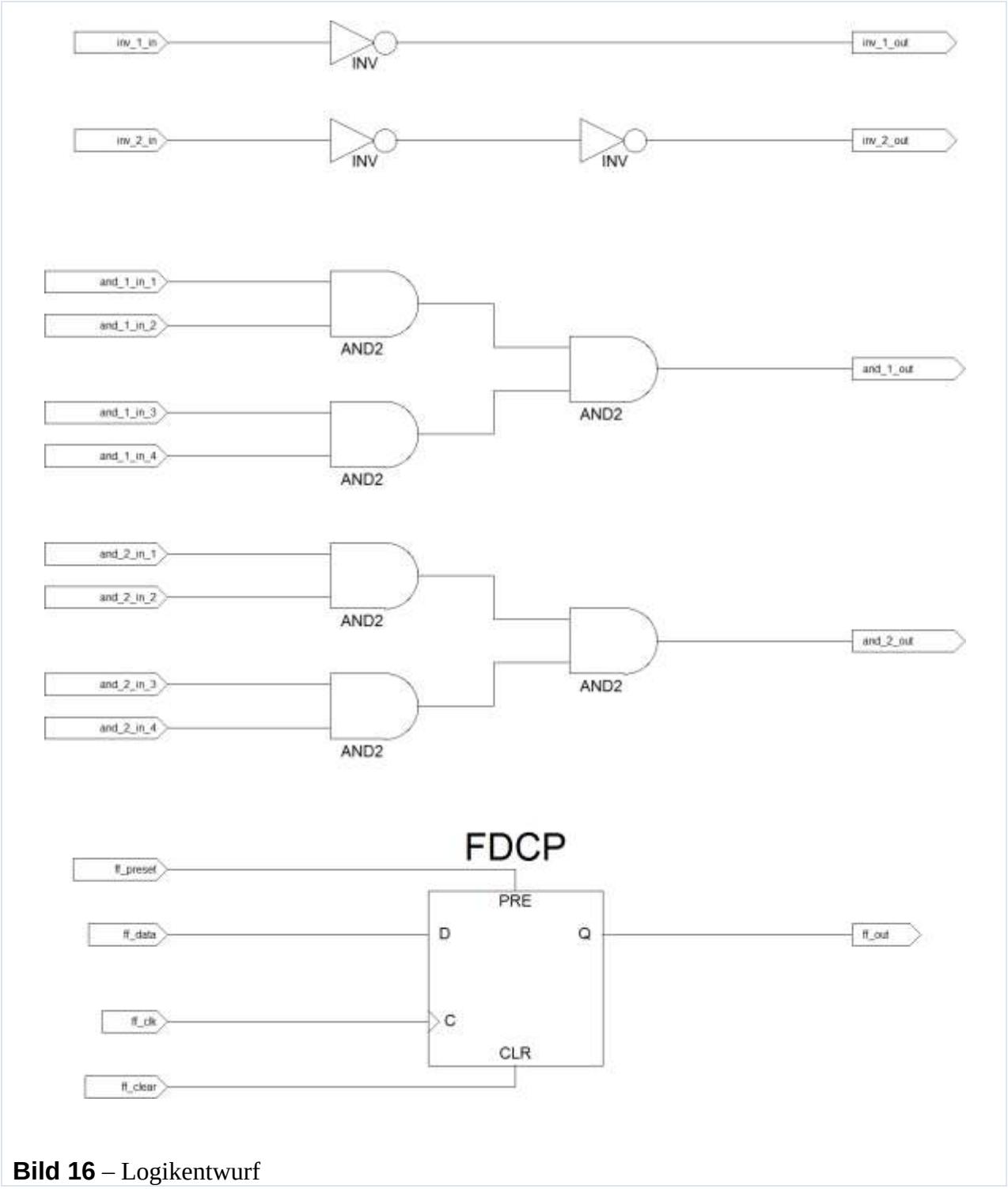

Bild 16 – Logikentwurf

Synthese

Mit Hilfe der Synthese werden die Logikdiagramme auf VHDL/Verilog konvertiert und des weiteren auf eine allgemeinere Darstellung gebracht. Dabei werden einige Optimierungsschritte durchlaufen, um komplexe Entwürfe ressourcenschonend unterzubringen. Der Fitter ist für das auswählen der richtigen CPLD-Logikblöcke und Verbindungen. Die Datei nach dem Fitter ist nur noch für einen bestimmten Chip geeignet.
Bevor die Prozesse gestartet werden, sollten die Einstellungen überprüft werden (Processes → Implement Design → Rechts-Klick → Process-Properties).

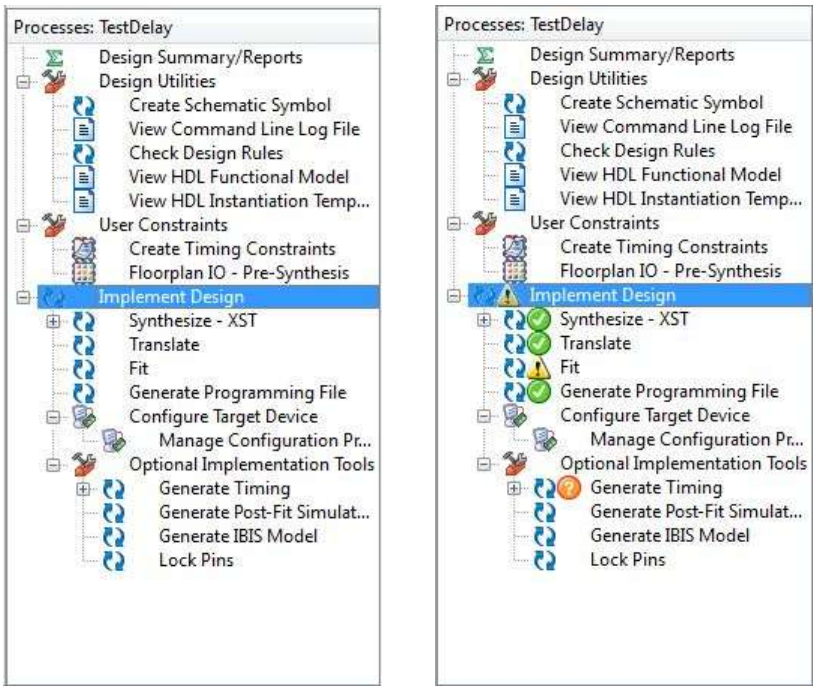

Bild 17 – Prozess-Übersicht (vor/nach der Synthese)

Nach der erfolgreichen Synthese sollte nur eine Warnung ausgegeben worden sein. Diese Warnung bezieht sich auf ein veraltetes Modul der Entwicklungsumgebung, welches anders aufgerufen wird, als gewünscht. Die Warnung kann ignoriert werden.

```
WARNING:Cpld - Unable to retrieve the path to the iSE Project Repository. Will
    use the default filename of 'TestDelay.ise'.
```

Bild 18 – Warnung

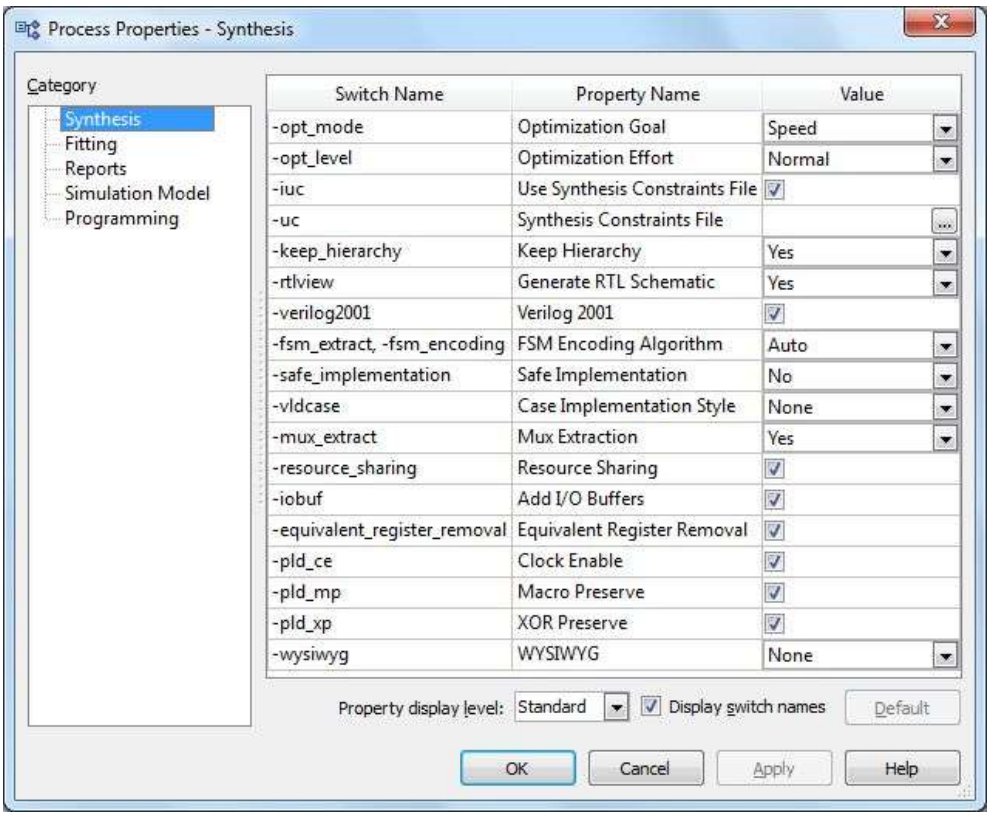

Bild 19 – Synthese Einstellungen

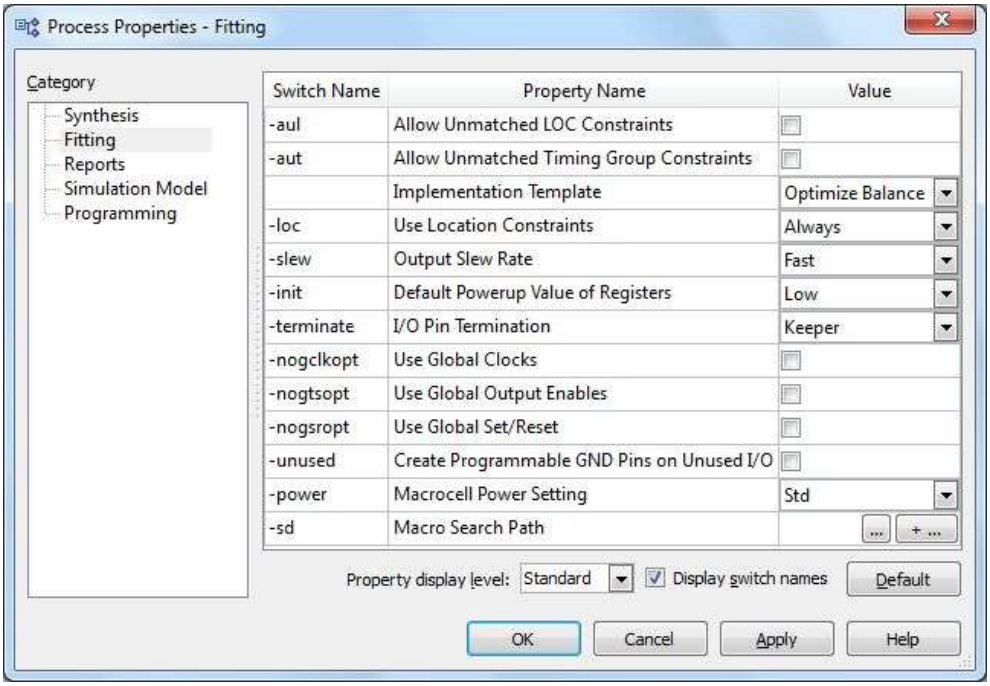

Bild 20 – Fitter Einstellungen

Die Synthese-Ergebnisse sollten überblicksmäßig überprüft werden (z.B. wieviele Register wurden instanziiert, welche Gleichungen wurden implementiert, welche Pins wurden zugewiesen, ...).
Bei unserem Design wurde ein Register, 16 Pins und 5 Makrozellen verwendet. Da bei den "Fitting"-Einstellungen die globalen Takt, Set und Reset Pins abgewählt wurden, wurden die Register Eingänge mit einfachen Pins realisiert (Required/Mapped).

RESOURCES SUMMARY

Macrocells Used	Pterms Used	Registers Used	Pins Used	Function Block Inputs Used
5/72 (7%)	8/360 (3%)	1/72 (2%)	19/34 (56%)	14/216 (7%)

PIN RESOURCES

Signal Type	Required	Mapped	Pin Type	Used	Total
Input	14	14	I/O	16	29
Output	5	5	GCK/IO	0	3
Bidirectional	0	0	GTS/IO	2	2
GCK	0	0	GSR/IO	1	1
GTS	0	0			
GSR	0	0			

Bild 21 – Synthese Ergebnisse

Bei den Gleichungen sieht man die Auswirkung des Optimierers. Die Logikschaltung mit den zwei Invertern wurde minimiert und durch eine einfache Leitung ersetzt.

Equations

```
********** Mapped Logic **********
and_1_out <= (and_1_in_2 AND and_1_in_1 AND and_1_in_4 AND
    and_1_in_3);
and_2_out <= (and_2_in_2 AND and_2_in_1 AND and_2_in_4 AND
    and_2_in_3);
FDCPE_ff_out: FDCPE port map (ff_out,ff_data,ff_clk,ff_clear,ff_preset);
inv_1_out <= NOT inv_1_in;
inv_2_out <= inv_2_in;
Register Legend:
    FDCPE (Q,D,C,CLR,PRE,CE);
    FTCPE (Q,D,C,CLR,PRE,CE);
    LDCP (Q,D,G,CLR,PRE);
```

Bild 22 – Synthese Ergebnisse

Optimierung Lokal Deaktivieren

Um bestimmte Logik-Optimierungen zu unterbinden, können Restriktionen hinzugefügt werden (z.B. belasse diese Verbindung = keep).
Wenn man dem Synthese-Programm nur vorgibt, dass er die Logik so implementieren soll, wie sie dargestellt wird, so wird der kombinatorische Teil dennoch optimiert. Das Ergebnis entspricht der vorgegebenen Logik.
Will man logische Blöcke in getrennten Makrozellen, so muss die Verbindung dazwischen explizit gekennzeichnet werden (keep).

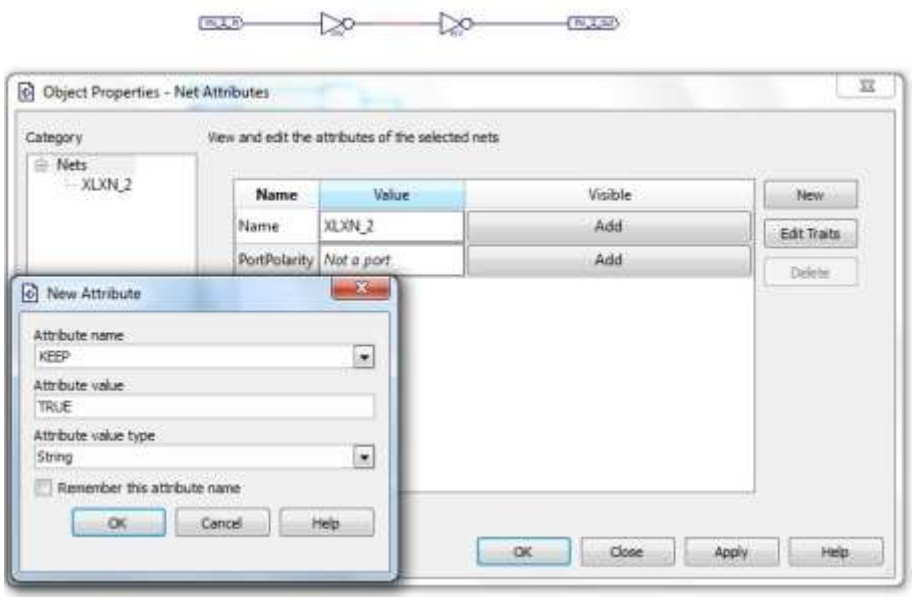

Bild 23 – Restriktion Hinzufügen (Constraint)

Bild 24 – Logikentwurf

Nach dem Einfügen der Restriktionen sehen die Gleichungen etwas verändert aus. Es wurden "Zwischen"-Gleichungen eingeführt. Diese repräsentieren die innere Verbindung.

Equations

```
********** Mapped Logic **********
XLXN_2 <= NOT inv_2_in;
XLXN_4 <= (and_2_in_2 AND and_2_in_1);
XLXN_5 <= (and_2_in_4 AND and_2_in_3);
and_1_out <= (and_1_in_2 AND and_1_in_1 AND and_1_in_4 AND
    and_1_in_3);
and_2_out <= (XLXN_4 AND XLXN_5);
FDCPE_ff_out: FDCPE port map (ff_out,ff_data,ff_clk,ff_clear,ff_preset);
inv_1_out <= NOT inv_1_in;
inv_2_out <= NOT XLXN_2;
Register Legend:
    FDCPE (Q,D,C,CLR,PRE,CE);
    FTCPE (Q,D,C,CLR,PRE,CE);
    LDCP (Q,D,G,CLR,PRE);
```

Bild 25 – Syntheseergebnisse

Pin-Zuweisung

Die Pin-Zuweisung erfolgt über den "Floorplan IO"-Prozess. Nach dem ersten Starten der Anwendung wird eine UCF-Datei angelegt, welche für die Projekt-Constraints verwendet werden.
Danach werden sämtliche Pins aufgelistet, die vom Design benötigt werden. Durch einfaches Drag&Drop können die Ein-/Ausgänge bestimmten Pins zugewiesen werden.

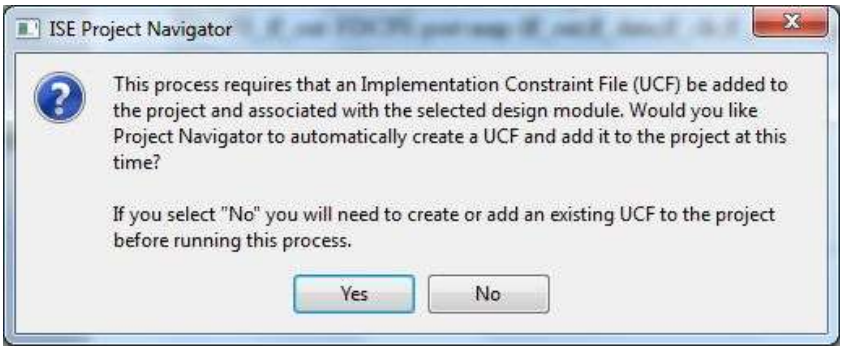

Bild 26 – Hinzufügen einer UCF-Datei

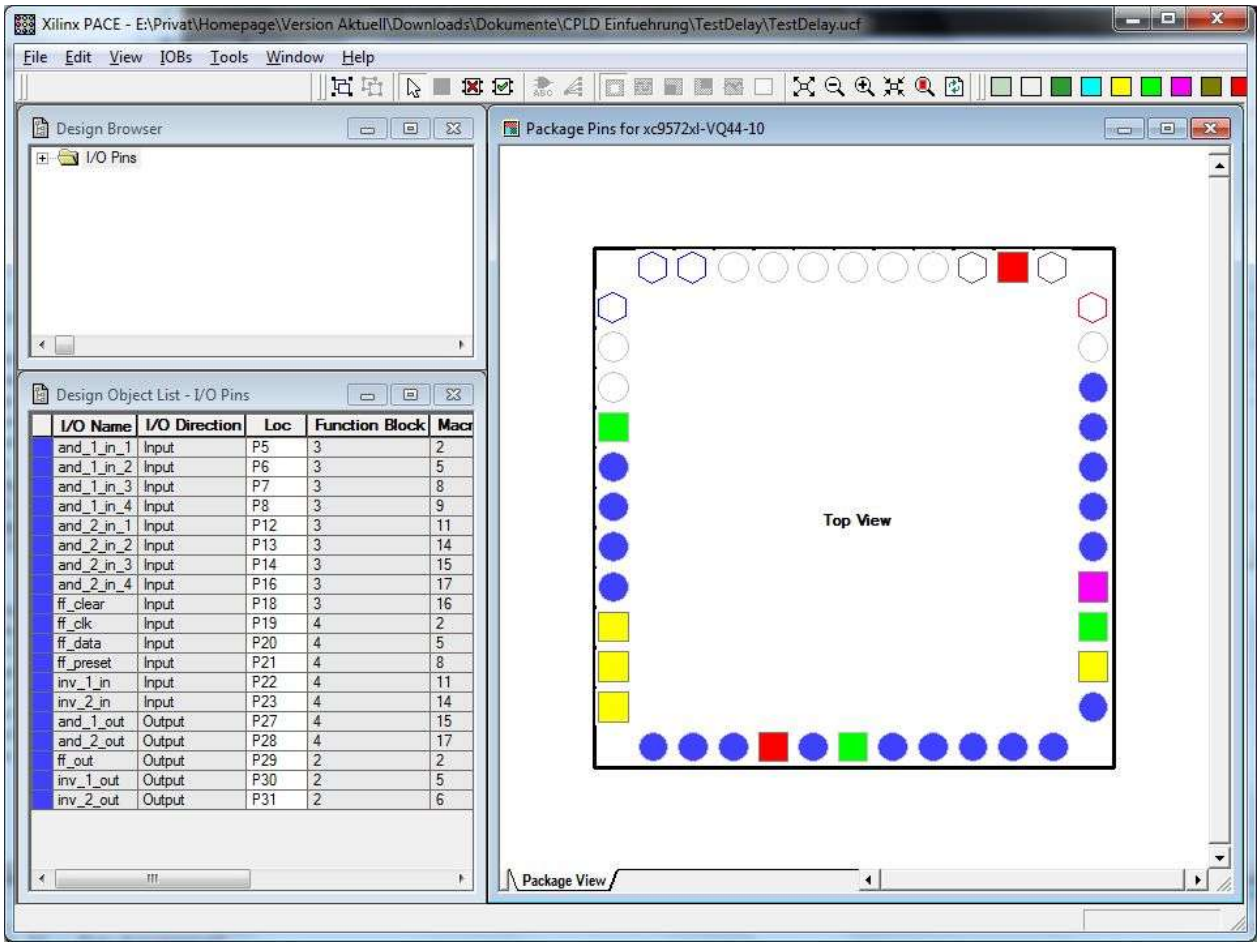

Bild 27 – Pin-Zuweisung

Nach dem Speichern der Pin-Konfiguration und einer erneuten Synthese sieht man bereits in der Zusammenfassung, dass die globalen Takt und Set/Reset Eingänge nicht verwendet wurden.

PIN RESOURCES

Signal Type	Required	Mapped	Pin Type	Used	Total
Input	14	14	I/O	19	29
Output	5	5	GCK/IO	0	3
Bidirectional	0	0	GTS/IO	0	2
GCK	0	0	GSR/IO	0	1
GTS	0	0			
GSR	0	0			

Bild 28 – Synthese Ergebnis

Pin List

Pin Num	Pin Type	Assigned Signal
1	I/O/GCK3	KPR
2	I/O	KPR
3	I/O	KPR
4	GND	GND
5	I/O	and_1_in_1
6	I/O	and_1_in_2
7	I/O	and_1_in_3
8	I/O	and_1_in_4
9	TDI	TDI
10	TMS	TMS
11	TCK	TCK
12	I/O	and_2_in_1
13	I/O	and_2_in_2
14	I/O	and_2_in_3
15	VCCINT	VCC
16	I/O	and_2_in_4
17	GND	GND
18	I/O	ff_clear
19	I/O	ff_clk
20	I/O	ff_data
21	I/O	ff_preset
22	I/O	inv_1_in
23	I/O	inv_2_in
24	TDO	TDO
25	GND	GND

Bild 29 – Verwendete Pins

Simulation

Timing Analyse / Behavioural Simulation

Für sequentielle Systeme liefert die Timing Analyse genügend Information, ob der Entwurf alle Zeitvorgaben einhält. Sämtliche kombinatorischen Blöcke werden über Register gepuffert. Somit müssen „nur“ die Setup- und Hold-Zeiten der Register überprüft werden. Die statische Timing Analyse ermittelt die maximale und minimale Verzögerung durch sämtliche kombinatorischen Blöcke. Ist der Maximal-Wert addiert mit der Setup-Zeit kleiner als die Taktperiode, so ist die Setup-Zeitvorgabe erfüllt. Ist der Minimal-Wert größer als die Hold-Zeit, so ist die Hold-Zeitvorgabe erfüllt.
Erfüllt das System die geforderten Zeitvorgaben, so ist die Erfüllung der gewünschten Funktion noch nicht garantiert. Um die Funktion testen zu können, werden Verhaltenssimulationen (Behavioural Simulation) durchgeführt. Diese Simulationen berücksichtigen keine Verzögerungszeiten durch kombinatorische Blöcke, denn diese wurden bereits über die statische Analyse überprüft.

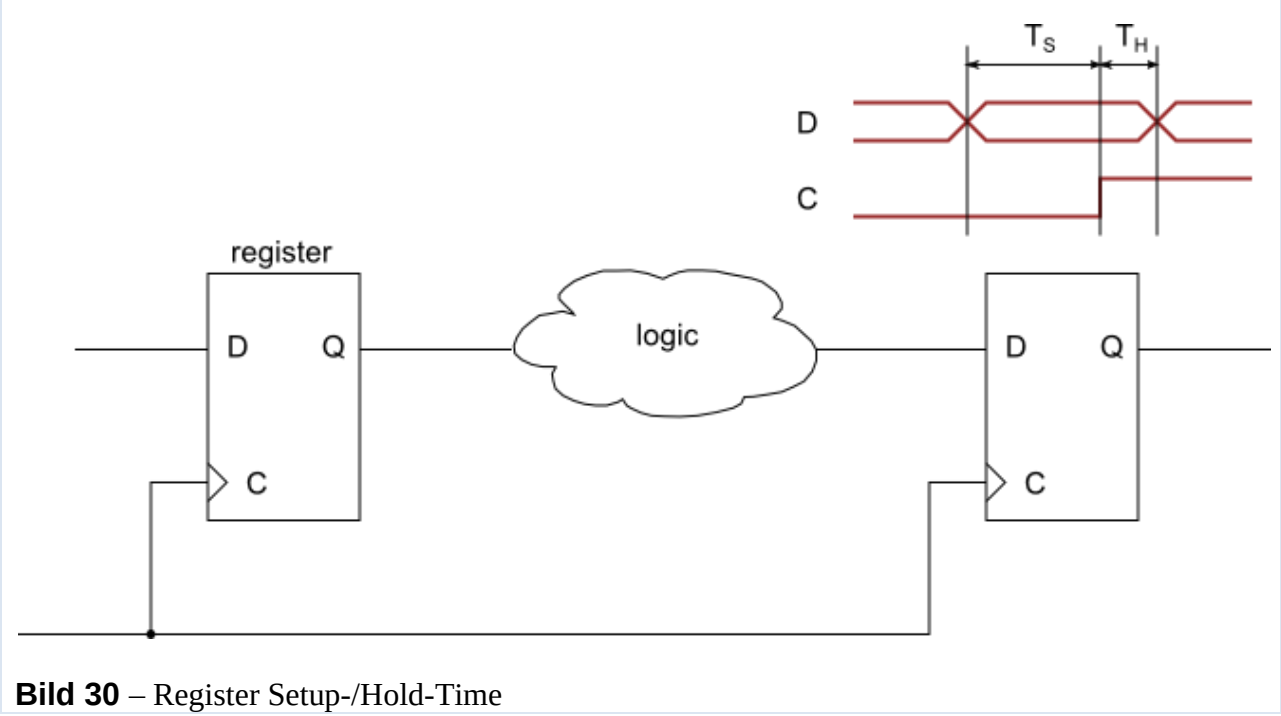

Bild 30 – Register Setup-/Hold-Time

Post-Fit Simulation

Möchte man die Verzögerungszeiten der einzelnen Blöcke ebenfalls simulieren, so verwendet man die Post-Fit Simulation. Diese Simulation kann nur durchgeführt werden, wenn man den CPLD-Chip bereits ausgewählt hat. Es werden die realen Zeiten verwendet. Diese sind vom verwendeten Typ und vom Speed-Grade des Chips abhängig.
Die Post-Fit Simulation wird meistens nur dann angewendet, wenn der Entwurf auch reine kombinatorische Blöcke enthält. Somit kann die minimal und maximal benötigte Verzögerung durch die Kombinatorik ermittelt werden.

Vor dem Beginn der Simulation muss für die Konfiguration ein Post-Fit Simulationsmodell erstellt werden (Generate Post-Fit Simulation Model). Danach wird eine Testumgebung angelegt. Dazu wechselt man vom Implementations-Modus in den Simulations-Modus und wählt die Post-Fit Simulation aus. In der Hierarchie fügt man eine neue „VHDL Test Bench“-Datei hinzu. Diese beinhaltet den automatisch generierten Rumpf, welche mit ein paar VHDL-Zeilen erweitert wird.

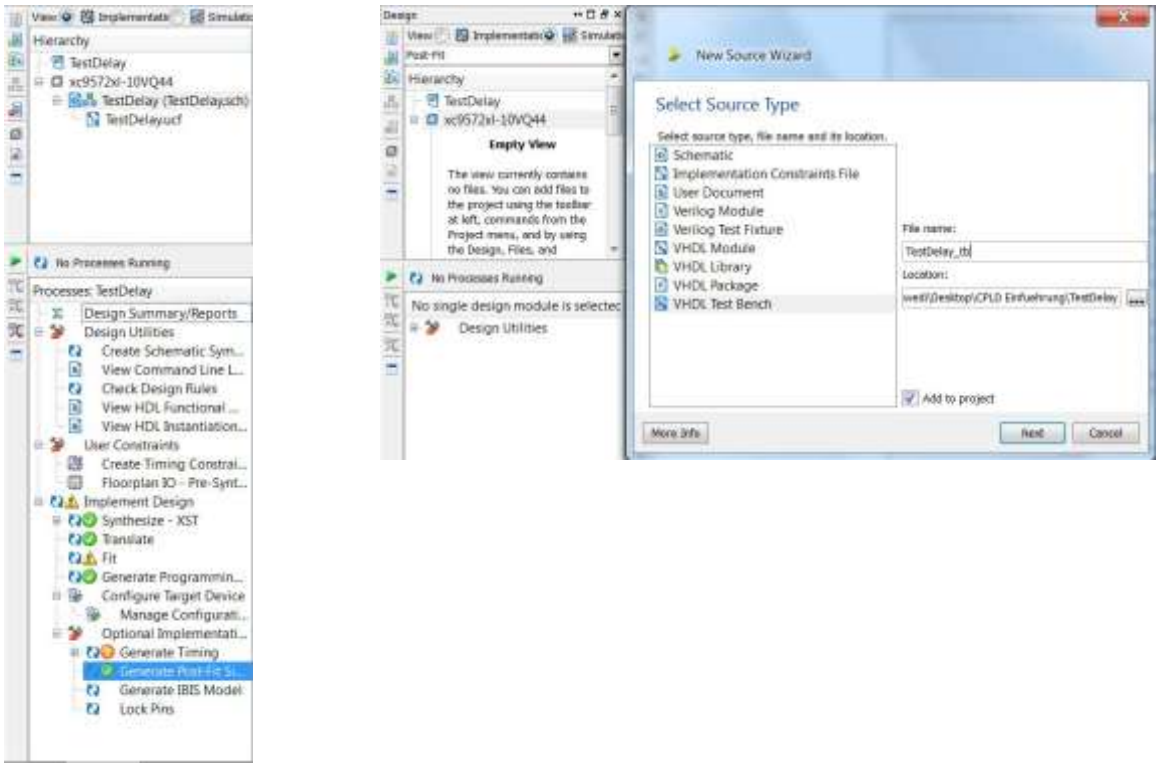

Bild 31 –Testbench Vorbereitungen

```
--------------------------------------------------------------------------------
-- Title :   Testbench for TestDelay
-- Project : -
--------------------------------------------------------------------------------
-- Abstract:
-- Generates some signals to test the logic.
--------------------------------------------------------------------------------
-- Company : -
-- Copyright (c)
--------------------------------------------------------------------------------
-- Author:   Dichler W.
-- Revision: 0.0.1
-- Date:     27.01.2013
--------------------------------------------------------------------------------
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

LIBRARY UNISIM;
USE UNISIM.Vcomponents.ALL;

ENTITY TestDelay_tb IS
END ENTITY TestDelay_tb;

ARCHITECTURE behavioral OF TestDelay_tb IS

   COMPONENT TestDelay
      PORT (
         inv_1_in   : IN  STD_LOGIC;
         inv_1_out  : OUT STD_LOGIC;
         inv_2_in   : IN  STD_LOGIC;
         inv_2_out  : OUT STD_LOGIC;
         and_1_in_1 : IN  STD_LOGIC;
         and_1_in_2 : IN  STD_LOGIC;
         and_1_in_3 : IN  STD_LOGIC;
         and_1_in_4 : IN  STD_LOGIC;
         and_1_out  : OUT STD_LOGIC;
         and_2_in_1 : IN  STD_LOGIC;
         and_2_in_2 : IN  STD_LOGIC;
         and_2_in_3 : IN  STD_LOGIC;
         and_2_in_4 : IN  STD_LOGIC;
         and_2_out  : OUT STD_LOGIC;
         ff_preset  : IN  STD_LOGIC;
         ff_data    : IN  STD_LOGIC;
         ff_clk     : IN  STD_LOGIC;
         ff_clear   : IN  STD_LOGIC;
         ff_out     : OUT STD_LOGIC

      );
   END COMPONENT;

   SIGNAL inv_1_in   : STD_LOGIC;
   SIGNAL inv_1_out  : STD_LOGIC;
   SIGNAL inv_2_in   : STD_LOGIC;
   SIGNAL inv_2_out  : STD_LOGIC;
   SIGNAL and_1_in_1 : STD_LOGIC;
   SIGNAL and_1_in_2 : STD_LOGIC;
   SIGNAL and_1_in_3 : STD_LOGIC;
   SIGNAL and_1_in_4 : STD_LOGIC;
   SIGNAL and_1_out  : STD_LOGIC;
   SIGNAL and_2_in_1 : STD_LOGIC;
   SIGNAL and_2_in_2 : STD_LOGIC;
   SIGNAL and_2_in_3 : STD_LOGIC;
   SIGNAL and_2_in_4 : STD_LOGIC;
   SIGNAL and_2_out  : STD_LOGIC;
   SIGNAL ff_preset  : STD_LOGIC;
   SIGNAL ff_data    : STD_LOGIC;
   SIGNAL ff_clk     : STD_LOGIC;
   SIGNAL ff_clear   : STD_LOGIC;
   SIGNAL ff_out     : STD_LOGIC;

BEGIN

   UUT: TestDelay
      PORT MAP(
         inv_1_in   => inv_1_in,
         inv_1_out  => inv_1_out,
         inv_2_in   => inv_2_in,
         inv_2_out  => inv_2_out,
         and_1_in_1 => and_1_in_1,
         and_1_in_2 => and_1_in_2,
         and_1_in_3 => and_1_in_3,
         and_1_in_4 => and_1_in_4,
         and_1_out  => and_1_out,
         and_2_in_1 => and_2_in_1,
         and_2_in_2 => and_2_in_2,
         and_2_in_3 => and_2_in_3,
         and_2_in_4 => and_2_in_4,
         and_2_out  => and_2_out,
         ff_preset  => ff_preset,
         ff_data    => ff_data,
         ff_clk     => ff_clk,
```

```
         ff_clear   => ff_clear,
         ff_out     => ff_out
      );

   tb : PROCESS
   BEGIN
      inv_1_in   <= '0';      -- initialize inputs

      inv_2_in   <= '0';

      and_1_in_1 <= '0';
      and_1_in_2 <= '0';
      and_1_in_3 <= '0';
      and_1_in_4 <= '0';

      and_2_in_1 <= '0';
      and_2_in_2 <= '0';
      and_2_in_3 <= '0';
      and_2_in_4 <= '0';

      ff_preset  <= '0';
      ff_clear   <= '1';
      ff_data    <= '0';
      ff_clk     <= '0';

      WAIT FOR 50 NS;         -- test combinatorial delay
      inv_1_in   <= '1';

      inv_2_in   <= '1';

      and_1_in_1 <= '1';
      and_1_in_2 <= '1';
      and_1_in_3 <= '1';
      and_1_in_4 <= '1';

      and_2_in_1 <= '1';
      and_2_in_2 <= '1';
      and_2_in_3 <= '1';
      and_2_in_4 <= '1';

      ff_clear   <= '0';      -- test register
      WAIT FOR 25 NS;
      ff_data    <= '1';

      WAIT FOR 25 NS;
      ff_clk     <= '1';
      WAIT FOR 25 NS;
      ff_clk     <= '0';

      WAIT FOR 25 NS;
      ff_clear   <= '1';

      WAIT FOR 25 NS;
      ff_clear   <= '0';

      WAIT FOR 25 NS;
      ff_preset  <= '1';

      WAIT;                   -- wait forever
   END PROCESS;

END ARCHITECTURE behavioral;
```

Quelltext 1 –Testumgebung

Ist die Testumgebung bereit, so kann die Simulation gestartet werden. Nach dem Öffnen von ISim sieht man bereits einige Signalverläufe. Damit die Übersicht erhöht wird, sollte man die Signale gruppieren (Rechtsklick „New Divider“). Für spätere Simulationen kann die aktuelle Darstellung gespeichert werden.
Stellt man die internen Signale dar, so sieht man die Verzögerungszeiten der einzelnen CPLD-Blöcke (Eingangspuffer, interne Logik, ...). Die Gesamtverzögerung kann auch manuell, durch das angegebene Zeitmodel des verwendeten Chips, ermittelt werden.

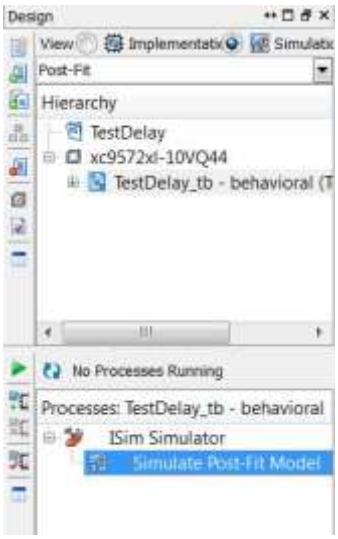

Bild 32 – Simulation Starten

Bild 33 – Simulation der Kombinatorik

Das Zeitverhalten ist in der Xilinx-Dokumentation genau beschrieben. Mit Hilfe des Zeitmodells kann die Simulation exakt nachvollzogen werden.
Bei der Logik mit einem Inverter sind vier Verzögerungen in Serie. Die Verzögerung des Eingangspuffers, die Verzögerung der internen Logik, die Verzögerung der kombinatorischen Logik und die Verzögerung des Ausgangspuffers. Die Gesamtverzögerung beträgt max. 10ns. Es handelt sich um einen Maximalwert und nicht um einen realen Wert, da die Herstellerangaben Maximalwert-Angaben sind.

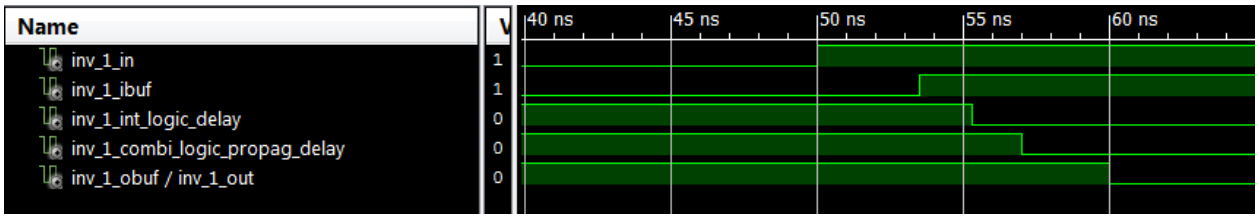

Bild 34 – Logikblock mit einem Inverter (Max-Delay / Setup-Time Simulation)

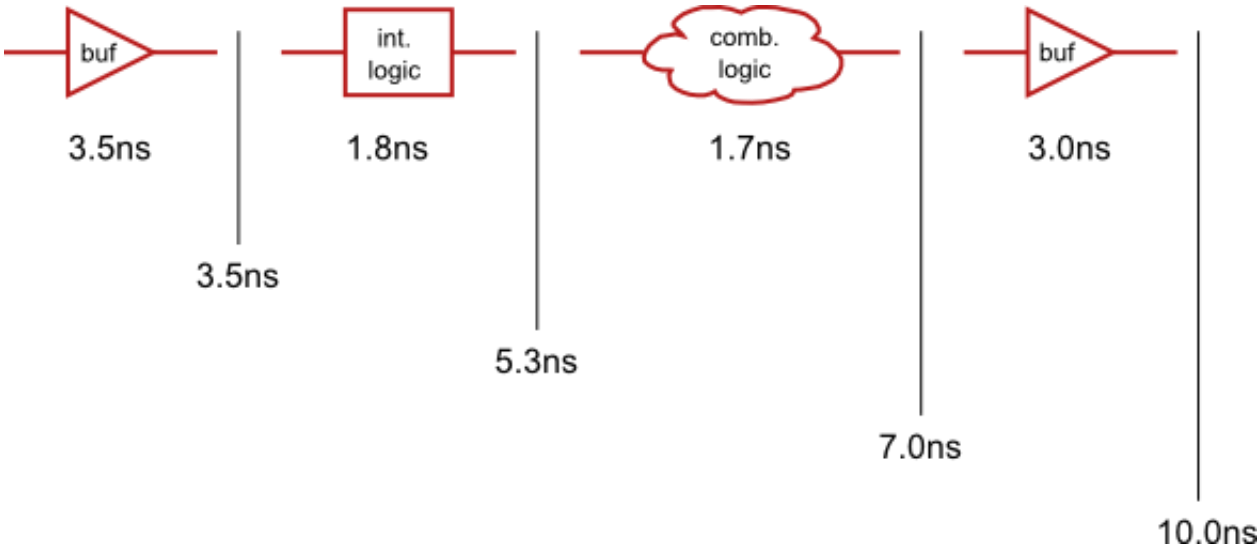

Bild 35 – Berechnung der Verzögerungszeit

Bei der Logik mit zwei Invertern wurden drei zusätzliche Verzögerungen hinzugefügt. Der der zweite Inverter hat eine eigene Makrozelle zugewiesen bekommen. Die Gesamtverzögerung beträgt somit max. 17.7ns.

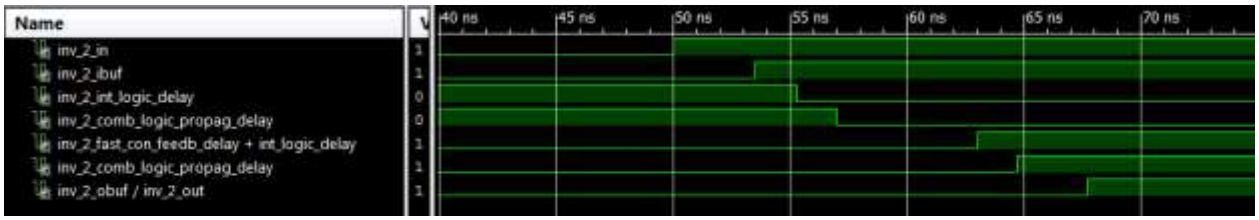

Bild 36 – Logikblock mit zwei Invertern (Max-Delay / Setup-Time Simulation

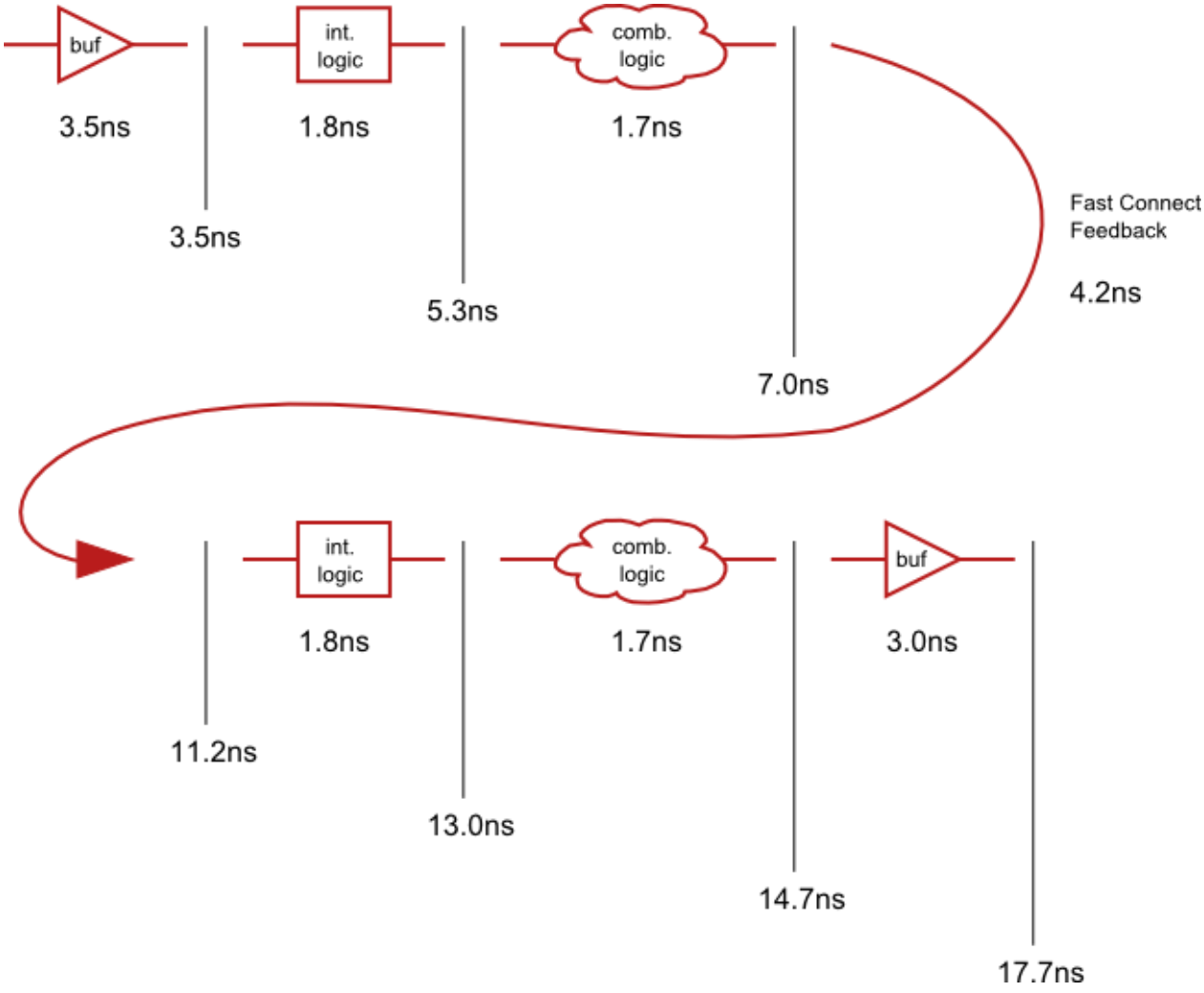

Bild 37 – Berechnung der Verzögerungszeit