



Grundlagen der Digitaltechnik

Foliensatz 1: Einführung

G. Kemnitz

Institut für Informatik, TU Clausthal (EDS_F1)

13. April 2023



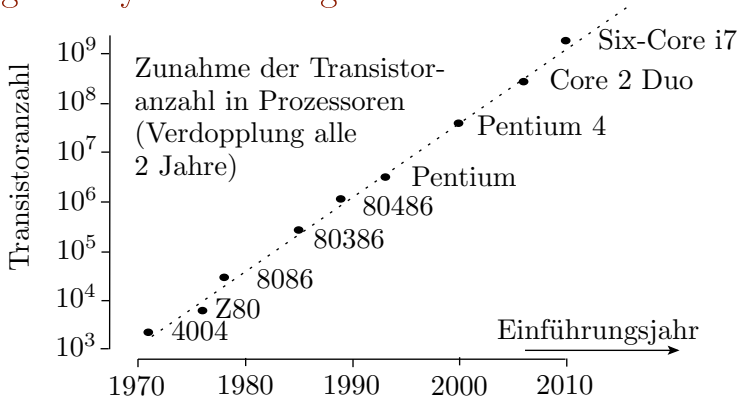
Organisation der Lehrveranstaltung

- Informationen, Foliensätze, Übungsaufgaben:
techwww.in.tu-clausthal.de
- Organisation und Ablaufplan: siehe Webseite
- Mi. 10:15 bis 11:45 Vorlesung.
- Do. 10:15 bis 11:45 abwechselnd Vorlesung und Übung
- 3 mal Labor- statt Hörsaalübung (20.04., 04.05. und 22.06.):
 - Gruppe 1: Do. 8:30 bis 10:00
 - Gruppe 2: Do. 10:15 bis 11:45
- Hausübungen: Aufgaben und Abgabetermine siehe Web-Seite.
Prüfungsvoraussetzung, Bonuspunkte für die Prüfung.

Prüfung:

- Schriftlich voraussichtlich Anfang Oktober.
- Hilfsmittel: eigene Ausarbeitungen, Mitschriften, Foliensätze, eigene Hausübungen, ...

Digitale Systeme sind groß



Nach dem Mooreschen Gesetz verdoppelt sich die Transistoranzahl digitaler Schaltungen alle 2 Jahre. Die größten derzeit gefertigten Schaltkreise enthalten über zehn Milliarden Transistoren.



Wenn es die Möglichkeit gibt, innerhalb weniger Jahre funktionierende digitale Schaltungen mit 10^9 Transistoren zu entwickeln, herzustellen und in Betrieb zu nehmen, sollte es für einen angehenden Bachelor of Science möglich sein, funktionierende Schaltungen aus einigen Hundert bis Tausend Gattern zu verstehen, zu entwerfen und auszuprobieren.

Wie beherrscht man so große Entwürfe?

- rechnergestützt, teilautomatisiert,
- hierarchisch, Bausteinkonzept.

Bausteine für digitale Funktionseinheiten:

- Standardschaltkreise,
- Rechnerschaltkreise,
- programmierbare Logikschaltkreise,
- Sensor- und Aktoreinheiten mit digitalen Schnittstellen, ...



Aufbau der Vorlesung

F1: Einführung anhand von zwei Beispielenentwürfen:

- 1 Klassisch mit Standardschaltkreisen.
- 2 Rechnergestützt in einer Hardware-Beschreibungssprache mit programmierbaren Logikschaltkreisen.

Lernziele:

- Entwurf erfolgt nach einfachen Regeln.
- Herausforderung ist die Beherrschung der Größe.
- Digitaler Schaltungsentwurf ist heute hauptsächlich Programmieren.



Weitere Foliensätze

F2: Simulation

- Einführung in VHDL (Hallo Welt, Signale, Datentypen, imperative Modelle, ereignisgesteuerte Simulation).
- Strukturbeschreibung (Schnittstellen, Instanziierung und Verbinden von Teilschaltungen, Testrahmen).
- Laufzeittoleranz (Glitches, Simulation von Zeittoleranzen, Laufzeitanalyse).
- Speicher (Latches, Register, Verarbeitung + Abtastung, Register-Transfer-Funktionen, adressierbare Speicher).

F3: Synthese und Schaltungsoptimierung

- Synthese (Verarbeitungsfunktionen, Register-Transfer-Funktionen, typische Beschreibungsfehler, Constraints).



- Asynchrone Eingabe (Abtastung, Initialisierung, Entprellen, asynchrone Schnittstellen mit und ohne Übertragung des Sendertaktes).
- Schaltungsoptimierung (Energieverbrauch, Schaltungsumformung, KV-Diagramm, Verfahren von Quine und McCluskey, reduziertes geordnetes binäres Entscheidungsdiagramm (ROBDD)).

F4: Rechenwerke und Operationsabläufe

- Rechenwerke (Addierer, Subtrahierer, Zähler, Multiplizierer, Komparatoren, Block-Shifter, ...).
- Automaten (Entwurf mit KV-Diagrammen, Beschreibung in VHDL, redundante Zustände, Spezifikation und Entwurf).
- Operationsabläufe (serielle Schnittstelle, serieller Addierer, Dividierer).



F5: Vom Transistor zur Logikschaltung

- Gatterentwurf (MOS-Transistoren als Schalter, FCMOS-Gatter, deaktivierbare Treiber, Transfergatter und Multiplexer, geometrischer Entwurf).
- Signalverzögerung (Inverter, Logikgatter, Puffer).
- Latches und Register.
- Blockspeicher (SRAM, Mehrport- und Assoziativspeicher, DRAM, Festwertspeicher).
- Programmierbare Logikschaltkreise.

F6: Rechner

- CORDIC (Algorithmus, erstes Simulationsmodell, Festkommazahlenformate, Algorithmus weiter optimiert).
- Minimalprozessoren (Befehlssatz, Datentypen, ...)
- Pipeline-Erweiterung.



Inhalt F1: Einführung

Standardschaltkreise

- 1.1 Entwurf eines Zählers
- 1.2 Test der Zählfunktion
- 1.3 Zustandsregister
- 1.4 Leiterplattenentwurf

VHDL + FPGA

- 2.1 Einfache Gatterschaltung
- 2.2 Increment Rechenwerk
- 2.3 Zähler und Ampelsteuerung
- 2.4 Simulation



Standardschaltkreise

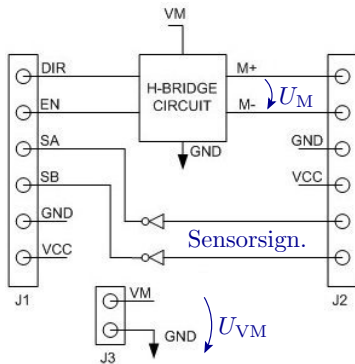


1. Standardschaltkreise

Aufbau, Funktion und Schaltplan



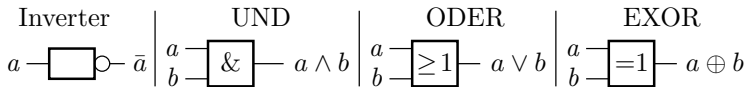
Dir	En	Motorspannung U_M
-	0	0 V
0	1	U_{VM}
1	1	$-U_{VM}$



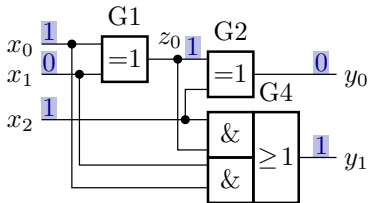
- Baugruppe mit H-Brücke.
- Funktion als Tabelle: $U_M = 0$ aus, $U_M = U_{VM}$ vorwärts, ...
- Schaltplan: Bauteile und ihre elektrischen Verbindungen.



Logische Grundbausteine



b	a	\bar{a}	$a \wedge b$	$a \vee b$	$a \oplus b$
0	0	1	0	0	0
0	1	0	0	1	1
1	0		0	1	1
1	1		1	1	0



x_2	x_1	x_0	z_0	y_1	y_0
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1	1	1	0
1	1	0			
1	1	1			



Logikfamilie

Schaltkreise unterschiedlicher Funktion mit gleichem elektrischen Anschlussverhalten, gleicher Versorgungsspannung, ...

Klassiker war die 74...-Logikfamilie. Schaltungstechnik TTL (Transistor-Transistor-Logik). Veraltet.

In der Übung Nachfolger 74HC... CMOS-Logikfamilie. Schaltkreisnummern und Anschlusswerte kompatibel zur 74er-Serie.

74HC00 4 × NAND2

74HC02 4 × NOR2

74HC04 6 × Inverter

74HC08 4 × AND2

74HC10 3 × NAND3

74HC11 3 × AND3

74HC14 6 × Inverter (Schwellwertschalter mit Hysterese)

74HC20 2 × NAND4

74HC30 1 × NAND8

74HC32 4 × OR2

74HC74 2 × D-Flipflop

74HC75 4 × Latch

74HC86 4 × EXOR

74HC174 6-Bit-Register

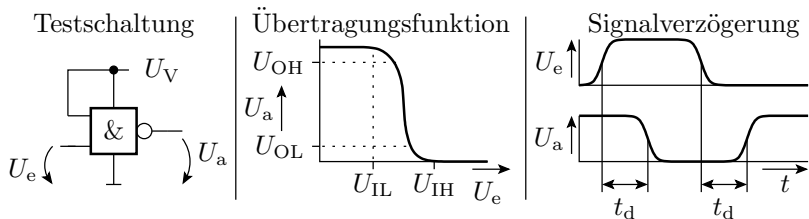
...



1. Standardschaltkreise

<p>74HC00: 4 × NAND2</p>	<p>74HC86 4 × EXOR</p>
<p>74HC174: 6 Bit-Register mit Takt- und Rücksetzeingang</p> <p>Funktion: Bei $\bar{R} = 0$ werden alle 6 Speicherzellen gelöscht: $Q_i = 0$ Sonst bei steigender Flanke an T Übernahme der Eingabe: $Q_i = D_i$ Sonst speichern.</p>	
<p>U_V Versorgungsspannung \perp Masse</p>	

Elektrische Eigenschaften

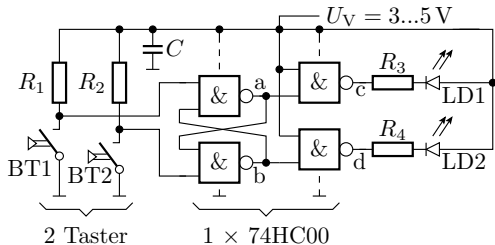


U_{OH} minimale Spannung für eine 1 am Ausgang
 U_{IH} minimale Spannung für eine 1 am Eingang
 U_{OL} maximale Spannung für eine 0 am Ausgang
 U_{IL} maximale Spannung für eine 0 am Eingang

U_V Versorgungsspannung
 t_d Verzögerungszeit

Typische elektrische Eigenschaften der 74HC-Familie	U_V	U_{OH}	U_{IH}	U_{IL}	U_{OL}	t_d
	2 V	1,9 V	1,5 V	0,5 V	0,1 V	100 ns
	4,5 V	4,4 V	3,2 V	1,4 V	0,1 V	20 ns
	6 V	5,9 V	4,2 V	1,8 V	0,1 V	16 ns

Untersuchung einer Beispielschaltung



Was passiert, wenn man die Taster nacheinander wie folgt drückt:

BT1	BT2	a	b	c	d	LD1	LD2
aus	aus	0	1	1	0	aus	an
an	aus						
aus	aus						
aus	an						



Einige wichtigste Regeln für den Entwurf

- Alle Versorgungsanschlüsse aller Schaltkreise mit $U_V = 2...6\text{ V}$ und alle Masseanschlüsse mit Masse verbinden.
- Stützkondensator von 10/100nF zwischen U_V - und Masseanschluss der Schaltkreise.
- Ein Ausgang kann mehrere Eingänge oder eine Low-Current LED treiben.
- LEDs brauchen immer einen Vorwiderstand zur Strombegrenzung.
- Ungenutzte Eingänge nicht offen lassen, sondern mit U_V oder Masse verbinden.
- Ausgänge nie miteinander, mit U_V oder Masse verbinden.
- Wenn Eingangsspannungen $> U_V$ oder < 0 nicht ausschließbar sind, z.B. bei Eingabe über Stecker, 100Ω Schutzwiderstand in Reihe schalten.



Entwurf eines Zählers



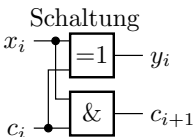
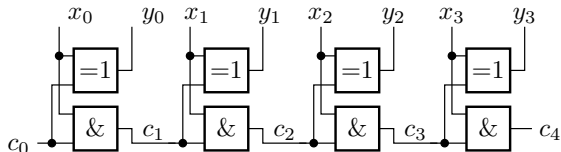
Entwurf der Zählfunktion

Aufgabe sei der Entwurf eines 4-Bit-Zählers aus Gattern und einem 4-Bit-Register. Schaltungsentwurf für die Zähloperation:

$$\begin{array}{r}
 \text{Zähloperation} \\
 x_3 \ x_2 \ x_1 \ x_0 \\
 + \qquad \qquad \qquad c_0 \\
 \hline
 y_3 \ y_2 \ y_1 \ y_0
 \end{array}
 \qquad
 \begin{array}{r}
 \text{Beispiel} \\
 1 \ 0 \ 1 \ 1 \\
 + \qquad \qquad \qquad 1 \\
 \hline
 1 \ 1 \ 0 \ 0
 \end{array}$$

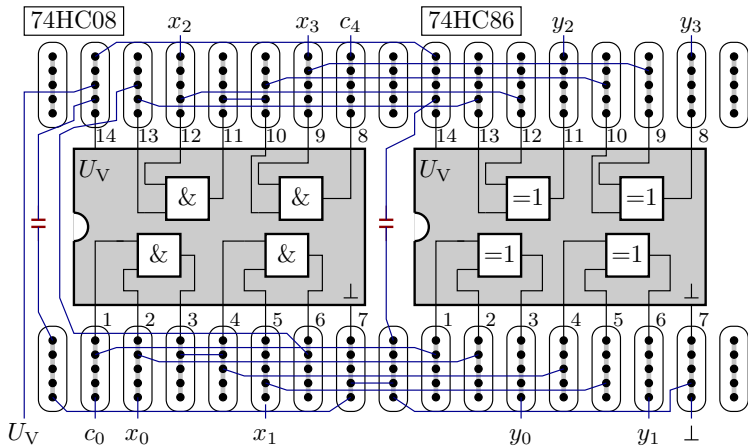
für jedes Bit
Wertetabelle

x_i	c_i	c_{i+1}	s_i
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Erfordert einen 74HC08 (4×AND2) und einen 74HC86 (4×EXOR).

Platzierung und Verdrahtung



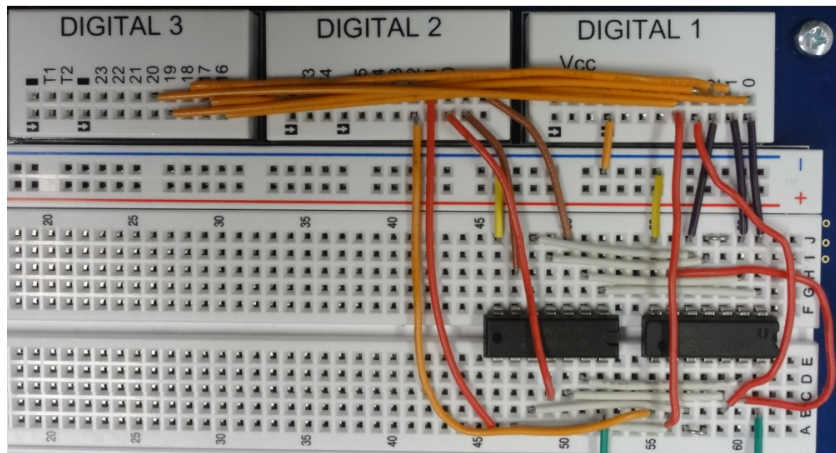
Anordnung auf einem Steckbrett. Blau gesteckte Drahtbrücken, rot Stützkondensatoren. Eingerahmte Punkte sind intern verbunden.



Test der Zählfunktion

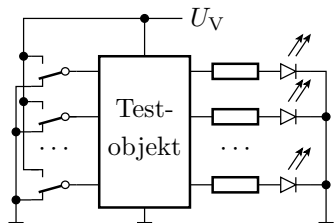


Schaltungsaufbau mit »Electronics Explorer«

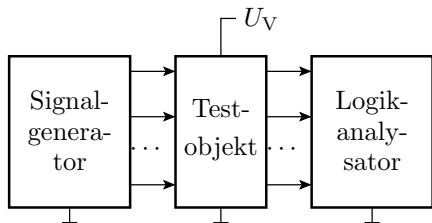


Testen

Test des logischen Verhaltens



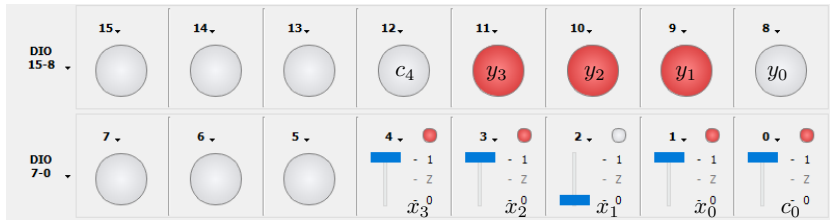
Test des Zeitverhaltens



Zum Test des logischen Verhaltens werden die Eingaben über Schalter angelegt und die Ausgabewerte mit Leuchtdioden oder einem Multimeter überwacht. Kontrolliert wird die Funktion nach der Wertetabelle. Bei einem Test des Zeitverhaltens werden die Eingaben in Echtzeit von einem Signalgenerator erzeugt und die Ausgabe mit einem Logikanalysator aufgezeichnet. Dabei werden zusätzlich die Signalverzögerungen kontrolliert.

Test mit dem »Electronics Explorer«

»Electronics Explorer«: Steckbrett mit Spannungsquellen, Signalquellen, Messwandlern, ... Ansteuerung und Messwernerfassung über PC-Programm. Test des logischen Verhaltens: Programmoberfläche mit LEDs, Schaltern, ...

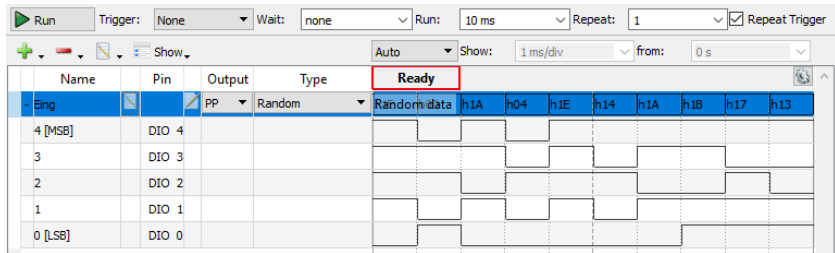


- Nacheinander mit Schaltern alle Testeingaben einstellen und
- LED-Ausgaben mit Sollwerten der Wertetabelle vergleichen.



Test des Zeitverhaltens

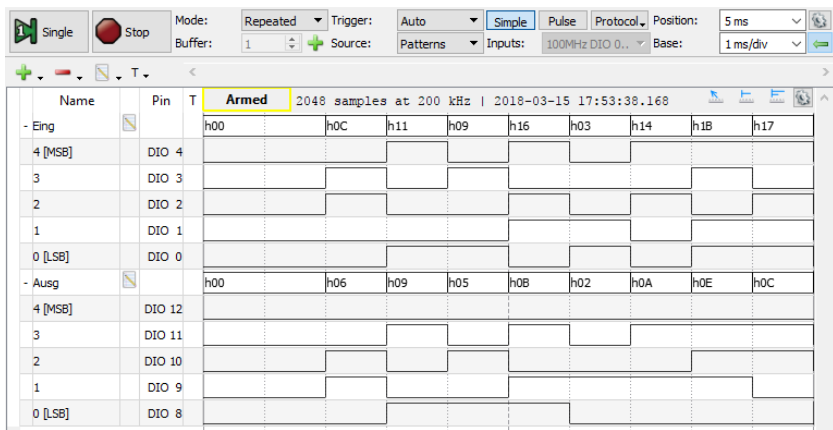
Bedienoberfläche des Signalgenerators. Testeingaben sind im Beispiel Zufallswerte. Anschluss durch Namenszuordnung.



Im Beispiel bilden die Eingänge x_0 bis x_3 einen Bus und c_0 ist ein Einzelsignal. Für alle Eingangesignale ist eingestellt »10 MHz« und Zufallssignale. Nach Konfiguration starten mit »Run«.



Ergebnisaufzeichnung mit Logikanalysator

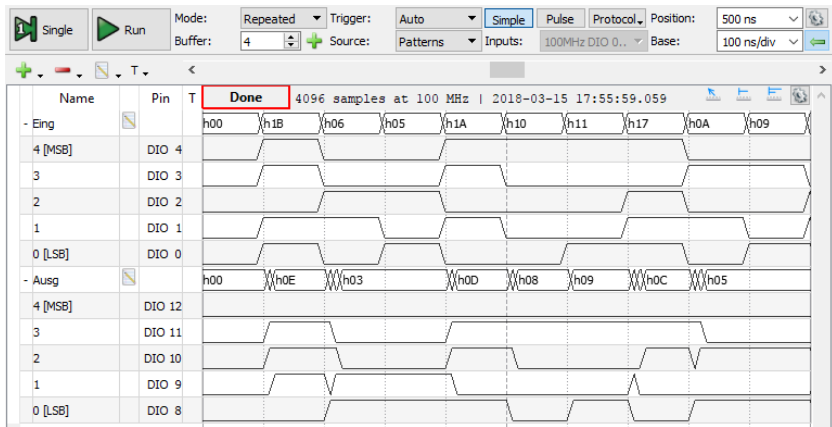


Ein Logikanalysator zeichnet digitale Abtastwerte auf. Erforderliche Einstellungen: abzutastende Signale, Abtastfrequenz, Trigger (Bedingung für den Aufzeichnungsbeginn), ...



1. Standardschaltkreise

2. Test der Zählfunktion



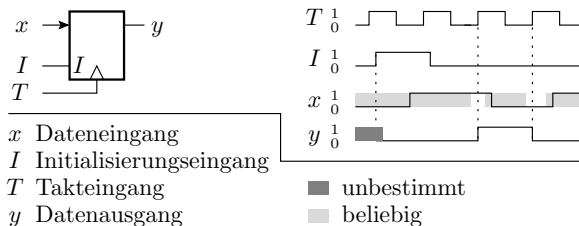
Im Beispiel werden alle Ein- und Ausgangssignale mit 100 MHz, d.h. mit 10 Werten je Eingabeänderung aufgezeichnet. Damit sind Verzögerungen mit einer Auflösung in 10ns-Schritten erkennbar.



Zustandsregister

Schaltungen mit Registern

Ein Register besitzt i. Allg. einen Initialisierungs- und einen Takteingang sowie Dateneingänge. Bei aktivem Initialisierungssignal (im Bild $I = 1$), Übernahme eines Anfangswertes, meist null. Sonst bei aktiver Taktflanke (hier steigender) Datenübernahme, sonst speichern.



Register dienen zur Abtastung, zur Verzögerung um einen Takt und als Zustandsspeicher für Schaltungen mit Gedächtnis.



Leiterplattenentwurf

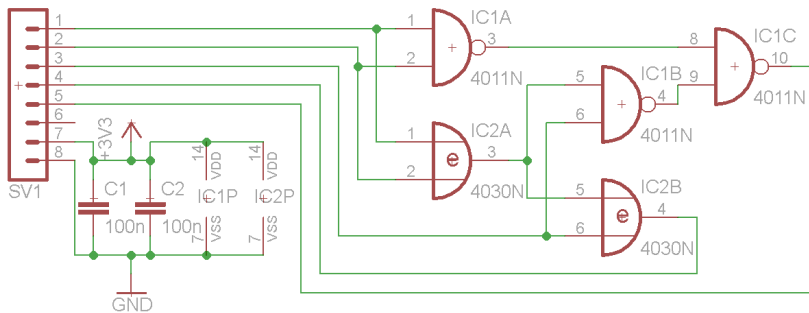


Leiterplattenentwurf

Weitere Schritte nach dem Entwurf des Schaltplans:

- Rechnereingabe in das Entwurfssystem (z.B. Eagle).
- Kontrolle der elektrischen Anschlussregeln, z.B. dass jedes Signal genau eine Quelle hat, ...
- Simulation mit Testbeispielen,
- Platzierung und Verdrahtung,
- Fertigung,
- Test gefertigter Baugruppen ohne Versorgungsspannung auf Bestückungs- und Verbindungsfehler,
- Test unter Spannung mit Beispieleingaben.

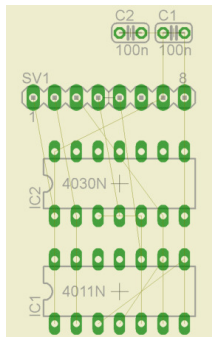
Schaltplaneingabe in Eagle



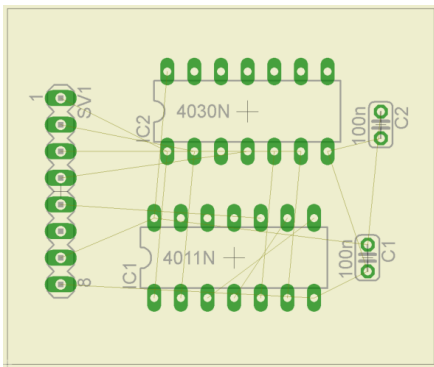
- zwei Schaltkreise, einer mit 4 NAND2 und einer mit 4 EXOR.
- Stecker mit 3 Eingängen, 2 Ausgängen, U_V und Masse.
- Impliziter Anschluss von U_V und Masse an die Schaltkreise.
- Stützkondensatoren bei der Platzierung unmittelbar an den Schaltkreisen anordnen.

Platzierung

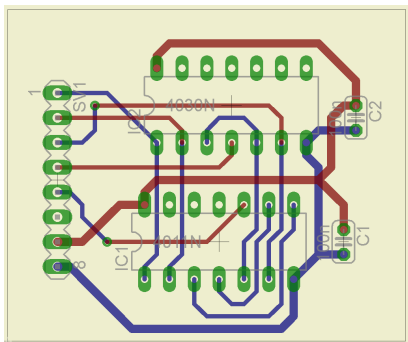
Nach Wechsel von der Schaltungs- zur Geometrieansicht.



Bauteile platziert. Die dünnen Linien sind noch anzuordnende Verbindungen



Verdrahtung



(blau – Unterseite, rot – Oberseite; grün – Lötaugen, bei zweiseitigen Leiterplatten mit Durchkontaktierungen (Vias)).



Zusammenfassung traditioneller Digitalentwurf

Zusammenfügen passgerechter Bausteine nach einfachen Regeln.
Überwiegend Fleißarbeit. Hauptaufwand Inbetriebnahme, Test und Fehlersuche.

Außer Schaltkreisen mit Gattern und Registern gibt es auch höher integrierte Schaltkreise mit

- Zählern, Schieberegistern, Multiplexern,
- Rechenwerken, Rechnerbausteinen, Prozessoren, ...

Entwicklung in den vergangenen 30 Jahren:

- Zunehmend komplexere Funktionseinheiten als Standardschaltkreise.
- ASICs (anwendungsspezifische Schaltkreise).
- Programmierbare Logikschaltkreise.
- Kleinere Bauteile und höhere Packungsdichte auf Leiterplatten.



VHDL + FPGA



Hardware-Programmierung

Heutiger Stand der Technik für den Entwurf digitaler Schaltungen ist die Beschreibung der Zielfunktion in einer

Hardware-Beschreibungssprache. Simulation auf dem Rechner.

Prototyp-Fertigung mit programmierbaren Logikschaltkreisen (FPGA **F**ield **P**rogrammable **G**ate **A**rray). FPGAs bestehen aus

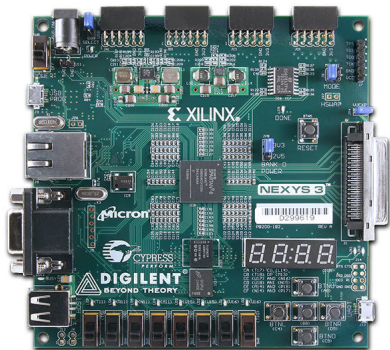
- programmierbaren Logikblöcken,
- programmierbaren EA-Schaltungen,
- einem programmierbaren Verbindungsnetzwerk und
- optional weiteren konfigurierbaren Schaltungsblöcken, z.B. Blockspeichern, Multiplizierern, Taktversorgung (Taktteiler und -vervielfacher, Zeitversatzkorrektur) und Prozessorkernen.

Programmierung ähnlich wie bei Mikrorechnern über eine serielle Programmierschnittstelle (JTAG, ISP,...). Programmierdauer Sekunden bis Minuten.

Prototyp-Plattform für die Laborübungen

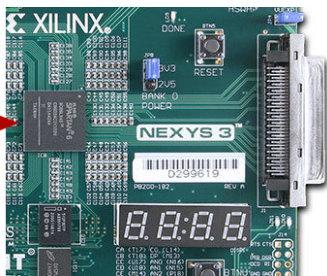
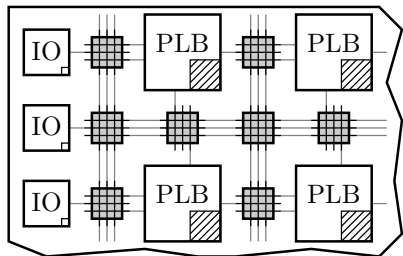
Prototyp-Board Nexys 3:

- Programmierbarer Logikschaltkreis
Größenordnung 10^6 Gatter.
- 100MHz Quarztakt,
- 8 Leuchtdioden,
- 4 7-Segment-Anzeigen,
- 8 Schalter, 5 Taster,
- 1MB SRAM,
- Anschlüsse für USB-Tastatur/Maus, USB-UART, VGA,
- Kameras, Touch-Screen, ...



Ausreichend für alles, was im Studium gelehrt wird.

Der Programmierbare Logikschaltkreis



programmierbares
Verbindungsnetzwerk



programmierbare Eingabe-Ausgabe-Schaltung



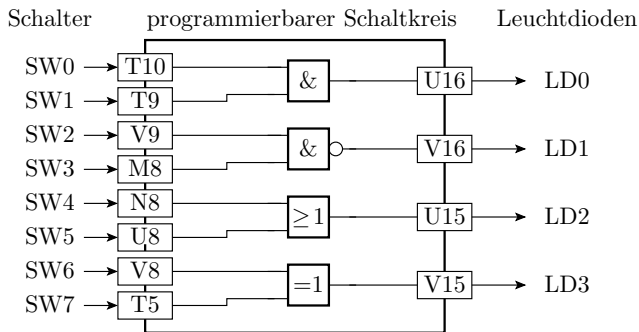
programmierbarer
Logikblock

Die Konfiguration wird ähnlich wie ein Programm auf dem PC entwickelt, übersetzt, ...



Einfache Gatterschaltung

Einprogrammieren von Logikfunktionen



Im ersten Beispiel sollen vier Gatter so einprogrammiert werden, dass die Eingänge von Schaltern steuer- und die Ausgänge mit LEDs beobachtbar sind. Die Kästchen mit »T10« etc. sind die Bezeichner der Schaltkreisanschlüsse, an denen die Schalter und LEDs auf der Baugruppe angeschlossen sind.



Beschreibung in VHDL

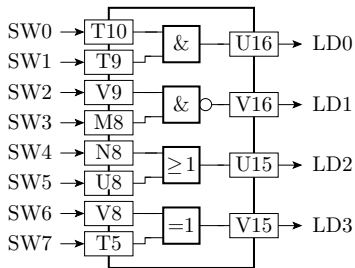
Eine VHDL-Beschreibung besteht aus

- Schnittstellenbeschreibung und
- einer Beschreibung der Realisierung.

Die Schnittstelle definiert für die Anschlüsse Bezeichner, Flussrichtung (Ein-/Ausgang) und den Datentyp:

```
entity Gatterschaltung is  
  port (SW0, SW1, SW2, SW3, SW4, SW5,  
        SW6, SW7: in std_logic;  
        LD0, LD1, LD2, LD3: out std_logic);  
end entity;
```

In der Realisierung stehen im Beispiel Signalzuweisungen mit logischen Verknüpfungen:



```

architecture test of Gatterschaltung is
begin
  LD0 <= SW0 and SW1;
  LD1 <= SW2 nand SW3;
  LD2 <= SW4 or SW5;
  LD3 <= SW6 xor SW7;
end architecture;
  
```

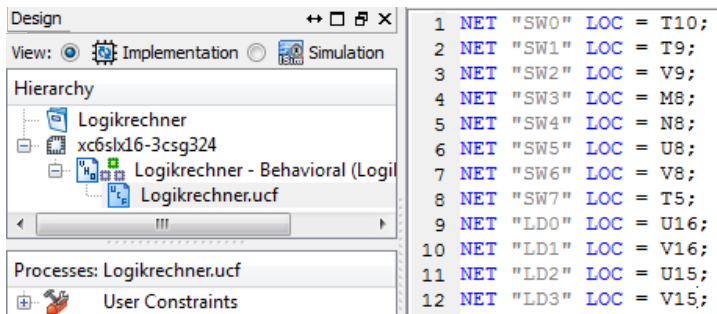
Entwurf

The screenshot shows the Xilinx IDE interface. The top toolbar has 'Implementation' selected. The 'Hierarchy' pane on the left shows a project named 'Logikrechner' with a sub-project 'Logikrechner - Behavioral (L...'. The 'Processes' pane on the left lists the following steps: Design Summary/Reports, Design Utilities, User Constraints, Synthesize - XST, Implement Design, Generate Programming File, Configure Target Device, and Analyze Design Using ChipScope. The main editor window displays the following VHDL code:

```
4 -----  
5 library IEEE;  
6 use IEEE.STD_LOGIC_1164.ALL;  
7  
8 entity Logikrechner is  
9   Port ( SW0, SW1, SW2, SW3, SW4, SW5,  
10         SW6, SW7 : in  STD_LOGIC;  
11         LD0, LD1, LD2, LD3 : out STD_LOGIC);  
12 end Logikrechner;  
13  
14 architecture Behav. of Logikrechner is  
15 begin  
16   LD0 <= SW0 and SW1;  
17   LD1 <= SW2 nand SW3;  
18   LD2 <= SW4 or SW5;  
19   LD3 <= SW6 xor SW7;  
20 end Behavioral;
```

- Projekt anlegen, einige Konfigurationen vornehmen, ...
- Beschreibung eingeben, Syntaxtest, optional Simulation, ...
- Übersetzen (Synthesize bis Configure Target Device)

Das Constraint-File



```
1 NET "SW0" LOC = T10;
2 NET "SW1" LOC = T9;
3 NET "SW2" LOC = V9;
4 NET "SW3" LOC = M8;
5 NET "SW4" LOC = N8;
6 NET "SW5" LOC = U8;
7 NET "SW6" LOC = V8;
8 NET "SW7" LOC = T5;
9 NET "LD0" LOC = U16;
10 NET "LD1" LOC = V16;
11 NET "LD2" LOC = U15;
12 NET "LD3" LOC = V15;
```

Die Constraint-Datei enthält alle zusätzlichen Informationen zur Vorgabe der Zielfunktion, die nicht in der VHDL-Datei stehen: die Pin-Namen der Schaltungsanschlüssen (s.o.), Taktfrequenz, obere und untere Schranken für Verzögerungen, ...

Synthese

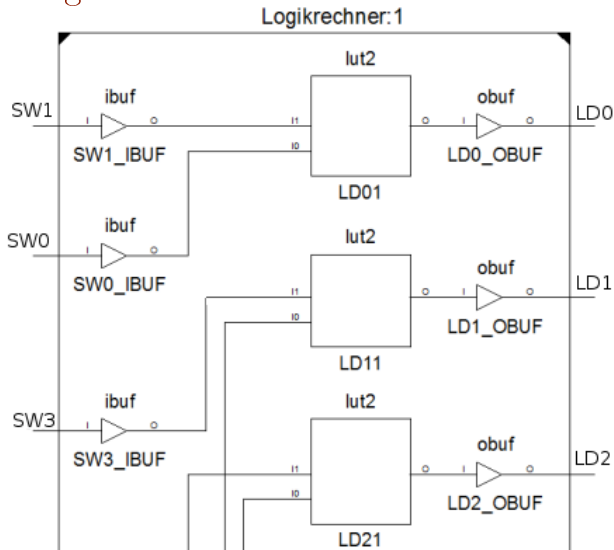
Berechnung einer Schaltung aus der VHDL-Beschreibung. Im Beispiel ist das trivial, weil die Schaltung nur aus vier Gattern besteht. Für größere Entwürfe wird die Zielfunktion mit Bitvektoren, arithmetischen Operatoren, Fallunterscheidungen, Unterprogrammen, ... beschrieben. Die Synthese muss daraus die logischen Funktionen extrahieren, optimieren, mit Teilschaltungen nachbilden, ...

Unser programmierbarer Schaltkreis hat statt Gatter als logische Grundbausteine Tabellenfunktionen (LUT **L**ook-**U**p **T**able, kleine programmierbare Speicher).

An den Anschlüssen werden Buffer eingefügt, die die internen kleineren Spannungspegel (0/1V) auf die größeren Anschlusspegel (0/2,5...3,3V) umsetzen.

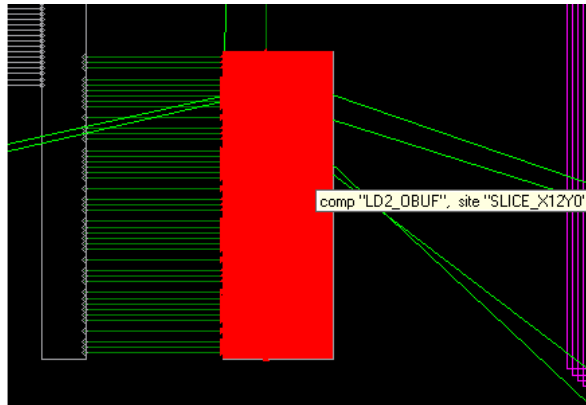
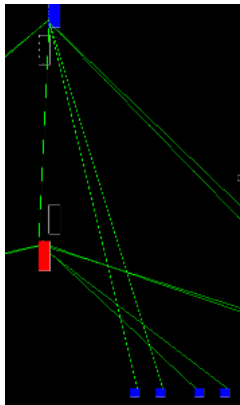
x_0	x_1	\wedge	$\bar{\wedge}$	\vee	\oplus
0	0	0	1	0	0
0	1	0	1	1	1
1	0	0	1	1	1
1	1	1	0	1	0

Syntheseresult



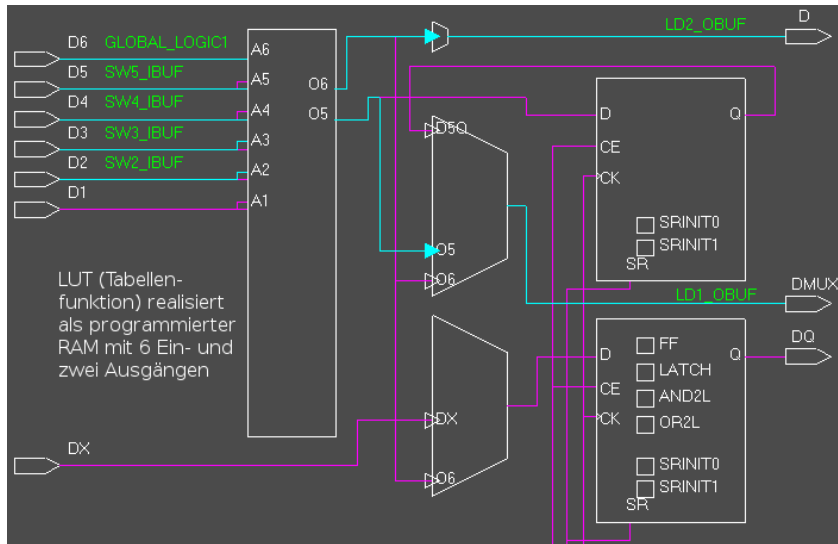
Verdrahtung

Nach der Synthese folgt die Platzierung der einzelnen Funktionsblöcke und ihre Verdrahtung. Die blauen Quadrate sind Pins und die Rechtecke sind programmierbare Logikblöcke (slices).





Schaltung in dem rot hervorgehobenen Slice:





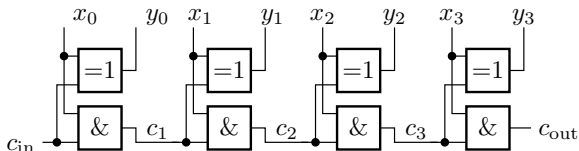
Increment Rechenwerk

Increment Rechenwerk

Die Inkrementoperation

$$c_{out} \ \& \ \mathbf{y} \leq \mathbf{x} + c_{in}$$

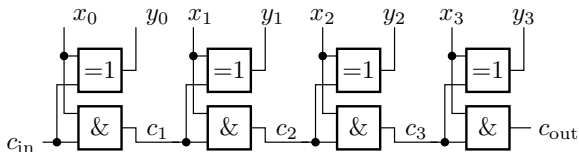
(\mathbf{x} , \mathbf{y} – 4-Bit-Vektoren) lässt sich mit derselben Schaltung wie auf Folie 19 implementieren:



```

entity Increment4Bit is
  port (cin: in std_logic;
    x: in std_logic_vector(3 downto 0);
    y: out std_logic_vector(3 downto 0);
    cout: out std_logic);
end entity;
    
```

Beschreibung mit logischen Operatoren



```

architecture Gatter_Arch of Increment4Bit is
  signal c: std_logic_vector(4 downto 0);
begin
  c(0) <= cin;
  y(0) <= x(0) xor c(0); c(1) <= x(0) and c(0);
  y(1) <= x(1) xor c(1); c(2) <= x(1) and c(1);
  y(2) <= x(2) xor c(2); c(3) <= x(2) and c(2);
  y(3) <= x(3) xor c(3); c(4) <= x(3) and c(3);
  cout <= c(4);
end architecture;
    
```

Generierungsschleife

Die vier bis auf die Indizes gleichen Zeilen können auch zu einer Schleife zusammengefasst werden. Das ist keine Ablaufschleife, deren Anweisungen im Schleifenkörper hintereinander mit dem Index 0 bis 3 abgearbeitet werden, sondern eine Generierungsschleife, die bei der Übersetzung durch je eine Anweisungspaar für jeden Indexwert ersetzt wird.

```
architecture Gen_Arch of Increment4Bit is  
  signal c: std_logic_vector(4 downto 0);  
begin  
  c(0) <= cin;  
  genHA: for i in 0 to 3 generate  
    y(i) <= x(i) xor c(i);   c(i+1) <= x(i) and c(i);  
  end generate;  
  cout <= c(4);  
end architecture;
```

Beschreibung mit dem Additionsoperator

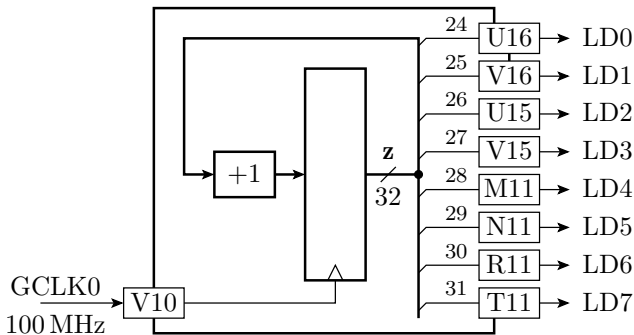
```
use ieee.numeric_std.all;
...
architecture Num_Arch of Increment4Bit is
  signal sum: unsigned(4 downto 0);
begin
  sum <= ('0' & x) + cin;
  y <= sum(3 downto 0);
  cout <= sum(4);
end architecture;
```

- Das Package `ieee.numeric_std` definiert u. a. den Datentyp `unsigned` und den Additionsoperator dafür.
- `x`, `y` müssen vom Typ `unsigned(3 downto 0)` und `cin` vom Typ `unsigned(0 downto 0)` sein. Erfordert Typumwandlungen.
- `»'0' & ...«` hängt an `»x«` zur Verlängerung auf 5 Bit eine führende Null an. Erforderlich für Übertragsberechnung.



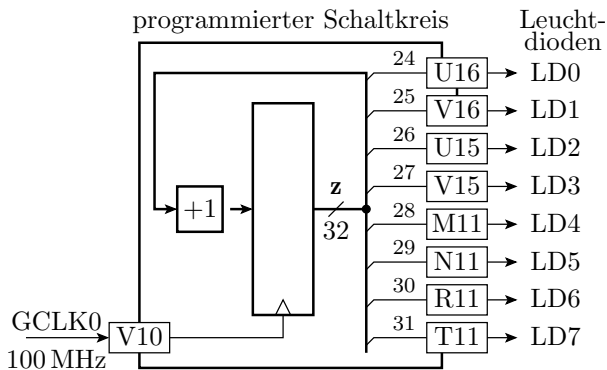
Zähler und Ampelsteuerung

Zähler als Taktteiler

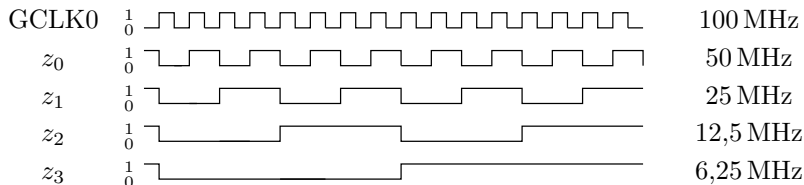


An Pin »V10« liegt auf der Baugruppe der 100MHz-Takt an. Dieser soll mit einem 32-Bit-Zähler gezählt und die höchstwertigen 8 Bit auf LEDs ausgegeben werden. Jedes Zählbit halbiert den Takt:

$$f(z_i) = 100 \text{ MHz} \cdot 2^{-(i+1)}$$



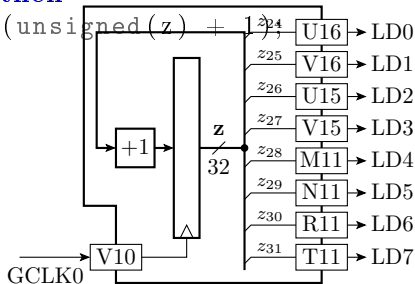
f	$t_P = \frac{1}{f}$
3 Hz	0,34 s
1,5 Hz	0,67 s
745 mHz	1,34 s
373 mHz	2,68 s
186 mHz	5,37 s
93 mHz	10,7 s
47 mHz	21,5 s
23 mHz	43,0 s





```
entity Taktteiler is
  port (GCLK0: in std_logic;
        LD0, LD1, ..., LD7: out std_Logic);
end entity;

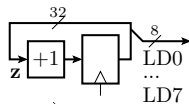
architecture test of Taktteiler is
  signal z: std_logic_vector(31 downto 0);
begin
  process (GCLK0)
  begin
    if rising_edge(GCLK0) then
      z <= std_logic_vector(unsigned(z) + 1);
    end if;
  end process;
  LD0 <= z(24);
  LD1 <= z(25);
  ...
  LD7 <= z(31);
end architecture;
```



Die Beschreibungsschablone für eine Register-Transfer-Funktion ist ein Prozess¹ mit dem Takt in der Weckliste (hier GCLK0) und Signalzuweisungen nur bei aktiver Taktflanke:

```

process (GCLK0)
begin
    if rising_edge(GCLK0) then
        z <= std_logic_vector(unsigned(z) + 1);
    end if;
end process;
    
```



Die Beschreibungsschablone einer einfachen kombinatorischen Schaltung, hier die Verbindung der Registerausgänge mit den Schaltkreisausgängen zu den LEDs, bleiben weiterhin die nebenläufige Signalzuweisungen ohne Prozessrahmen:

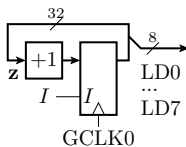
```
LD0 <= z(24);
```

¹Prozess ist ein Rahmen, in dem die Anweisungen bei der Simulation imperativ, d.h. wie bei einem normalen Programm hintereinander, statt nebenläufig abgearbeitet werden.

Die meisten Register-Transfer-Funktionen haben eine asynchrone Anfangsinitialisierung für die Reset-Funktion des Gesamtsystems. Bei dieser steht zusätzlich zum Takt das Initialisierungssignal in der Weckliste. Wenn es aktiv ist (im Beispiel bei $I='1'$), übernimmt das Register eine Konstante (im Beispiel alles null). Alle anderen Signalzuweisungen erfolgen nur bei inaktivem Initialisierungssignal und aktiver Taktflanke:

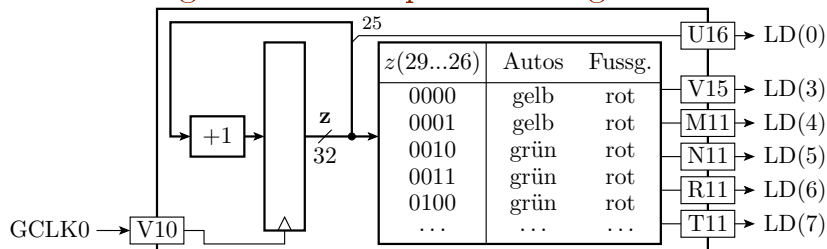
```

process (I, GCLK0)
begin
    if I = '1' then
        z <= (others => '0')2;
    elsif rising_edge(GCLK0) then
        z <= std_logic_vector(unsigned(z) + 1);
    end if;
end process;
    
```



²Das ist eine Zuordnungsliste, die allen anderen Bits, den der Wert nicht explizit zugeordnet wird, z.B. mit $0=>'1'$, den Wert '0' zugeordnet.

Erweiterung zu einer Ampelsteuerung



Komplexere kombinatorische Funktionen werden durch Prozesse mit allen Eingabesignalen in der Weckliste beschrieben, die bei jedem Aufwecken alle Ausgabesignale neu berechnen:

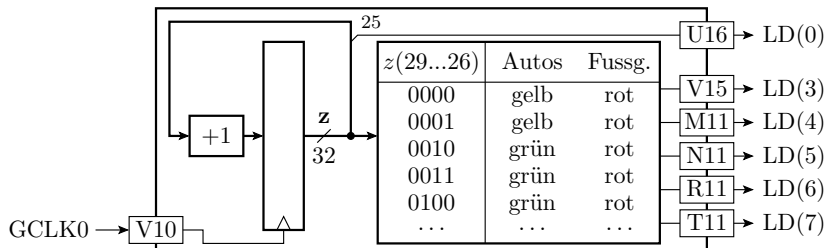
```
process (z)
```

```
begin
```

```
LD(0) <= z(25);  -- Ausgabe von z(25) auf eine LED
```

```
<Beschreibung der Tabelle siehe nächste Folie>
```

```
end process;
```



Tabellenbeschreibung mit einer Case-Anweisung:

```

case z(29 downto 26) is
  when "0000" | "0001"
    => LD(7 downto 3) <= b"010_01"; — A: gelb , F: rot
  when "0010" | "0011"
    => LD(7 downto 3) <= b"100_01"; — A: grün , F: rot
    — ab hier selbst weiterentwickeln
  when others
    => LD(7 downto 3) <= b"001_01"; — A: rot , F: rot
end case;
  
```



Simulation



Simulation

Die Simulation benötigt ein Testobjekt. Das sei die Zählfunktion:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Inc is
  port (x: in  std_logic_vector(3 downto 0);
        y: out std_logic_vector(3 downto 0));
end entity;

architecture behavioral of Inc is
begin
  y <= std_logic_vector(unsigned(x) + 1);
end architecture;
```

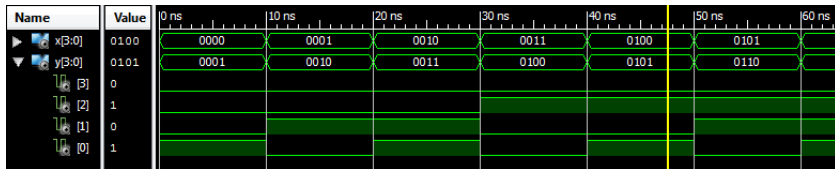
Zusätzlich wird ein Testrahmen zur Erzeugung der Eingangssignale benötigt. Das ist eine Entwurfseinheit mit dem Testobjekt als Teilschaltung und einem Prozess zur Erzeugung der Eingabe.



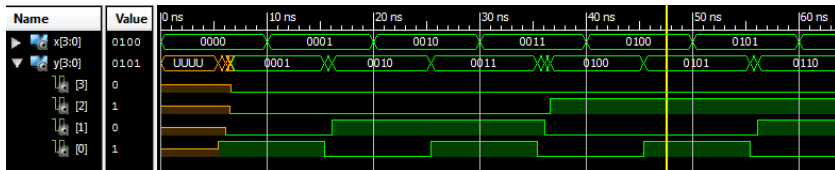
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity IncTB is end entity;
architecture behavioral of IncTB is
  signal x: std_logic_vector(3 downto 0):=(others=>'0');
  signal y: std_logic_vector(3 downto 0));
begin
  -- Einbindung des Testobjekts (UUT, Unit under Test)
  uut: entity work.Inc port map ( x=>x, y=>y);
  process -- Erzeugung der Eingaben
  begin
    for i in 1 to 10 loop
      wait for 10 ns;
      x <= std_logic_vector(unsigned(x) + 1);
    end loop;
    wait;
  end process;
end architecture;
```

Simulationsarten

- Simulation des Verhaltens ohne Verzögerungen:



- Simulation der synthetisierten und verdrahteten Schaltung (mit Verzögerungen)



Zusammenfassung

Im modernen Digitalentwurf wird die Zielfunktion in einer Hochsprache mit Datentypen und -objekten, arithmetischen und logischen Operatoren, Fallunterscheidungen, Schleifen, Unterprogrammen, ... beschrieben.

Eine so beschriebene Schaltung kann simuliert und innerhalb weniger Minuten übersetzt, in einem Schaltkreis geladen und getestet werden. Ähnlich wie Software-Entwurf.

Die bisher eingeführten Beschreibungsmittel sind bereits ausreichend, um die Zielfunktionen digitaler Schaltungen mit einigen hundert bis tausend Gattern zu beschreiben, zu simulieren und erfolgreich in Betrieb zu nehmen.