

Modellbildung und Simulation

1.1 Entwurf digitaler Schaltungen

Die heutigen digitalen Schaltkreise, Baugruppen und Systeme bestehen aus Millionen von logischen Bausteinen. Kein Mensch ist mehr in der Lage, diese Bausteine auch nur zu zählen. Wie werden so große Schaltungen erfolgreich entworfen? Diese Frage hat zu einer starken Annäherung des digitalen Schaltungsentwurfs an die Methoden und Techniken der Software-Entwicklung geführt.

1.1.1 Der Umgang mit großen Schaltungen

Die Prinzipien, auf denen der Entwurf, die Beschreibung und der Test von großen Schaltungen basieren, lauten

- rechnergestützt, hierarchisch, parametrisiert,
- Synthese und Simulation.

Hierarchie digitaler Systeme

Digitale Geräte bestehen aus Baugruppen, Baugruppen aus Schaltkreisen und anderen Bauelementen, Schaltkreise aus Funktionsblöcken etc. (Abb. 1.1). Eine hierarchische Beschreibung verweist auf jeder Hierarchieebene auf die verwendeten Bauteiltypen und beschreibt, wie Instanzen von diesen zum übergeordneten System zusammengesetzt werden. Die Bauteiltypen werden oft als fertige Produkte oder Entwurfsbeschreibungen übernommen und idealerweise mehrfach genutzt.

Der manuelle Arbeitsaufwand wird von der Typenvielfalt und weniger von der Systemgröße bestimmt. Der Entwickler muss sich mit dem Anschlussverhalten der Bausteine auf der betrachteten Hierarchieebene, aber nicht mit deren interner Realisierung befassen. Die Auflösung der Hierarchie in eine flache Beschreibung aus Tausenden oder Millionen von Gattern oder Transistoren

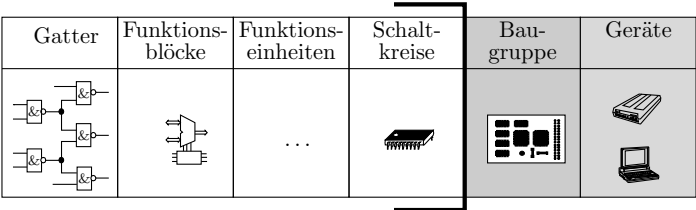


Abb. 1.1. Die Hierarchie digitaler Systeme

und deren weitere Verarbeitung (Analyse, Simulation, Synthese, Platzierung und Verdrahtung) erfolgen rechnergestützt.

Parametrisierte Schaltungen

Ein parametrisiertes Schaltungsmodell ist ein Datensatz für einen Algorithmus, der die Schaltung generiert. In einer hierarchischen Beschreibung sind UND-Gatter mit zwei, drei etc. Eingängen unterschiedliche Bauteiltypen. Um die Anzahl der zu unterscheidenden Typen im System überschaubar zu halten, werden ähnliche Funktionen zu Funktionsklassen zusammengefasst. Ein parametrisiertes UND könnte z.B. den Grundtyp UND und den Parameter Eingangsanzahl haben. Andere mit parametrisierten Modellen beschreibbare Schaltungsklassen sind Speicher, Register, Rechenwerke und Datenflussumschalter. Die Generierung der aufgelösten Schaltungen erfolgt genau wie die Auflösung der Hierarchie erst im Rechner, so dass nur der Rechner, nicht aber der Entwickler mit der tatsächlichen Schaltungsgröße und Typenvielfalt konfrontiert wird.

Synthese

Die Synthese sucht für eine in einer Hardware-Beschreibungssprache beschriebene Funktion eine optimierte Schaltung. Sie ist üblicherweise in folgende Schritte untergliedert [16] (Abb. 1.2):

- Register-Transfer-Synthese,
- Generierung,
- Logikoptimierung und
- Technologieabbildung.

Die Register-Transfer-Synthese arbeitet ähnlich wie ein Software-Compiler. Die Beschreibung der Zielfunktion wird zuerst analysiert, in eine interne Datenstruktur übersetzt und letztendlich auf einen Datenflussgraphen, d.h. eine Schaltung abgebildet. Logische Operatoren werden durch Logikgatter, arithmetische Operatoren durch Rechenwerke, Fallunterscheidungen durch Datenflussumschalter, Signale und Variablen durch Verbindungen oder Register etc.

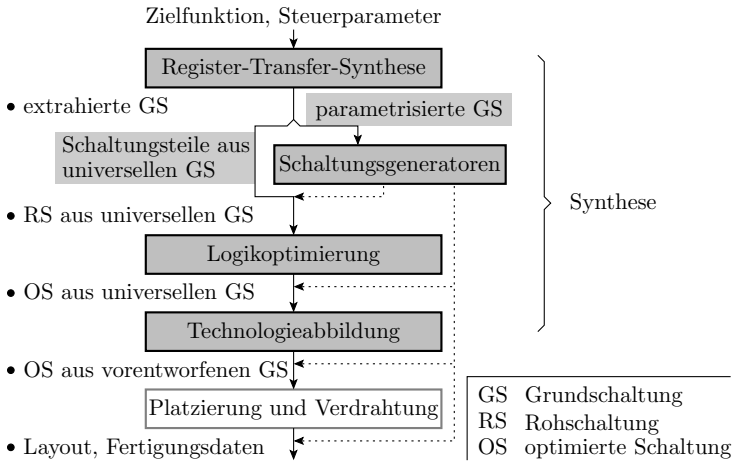


Abb. 1.2. Synthese

nachgebildet. Die extrahierten Gatter, Rechenwerke, Datenflussumschalter, Register etc. sind zum Teil parametrisierte Modelle mit variabler Anschlussanzahl, die an Schaltungsgeneratoren weitergereicht werden, die dafür die Schaltungen generieren. Die verbleibenden Teilschaltungen – Nachbildungen der Ausdrücke mit den logischen Grundoperatoren (Inverter, UND2, ODER2¹ etc.) – werden einzeln oder zusammen mit den generierten Teilschaltungen unter Ausnutzung der Gesetze der Schaltalgebra und anderer Regeln optimiert. Die nachfolgende Technologieabbildung ersetzt die universellen Grundsaltungen durch vorentworfenene (technologieabhängige) Grundsaltungen. Das ist keine einfache Eins-zu-eins-Abbildung, sondern oft eine Zusammenfassung mehrerer universeller Grundsaltungen zu einer Schaltung aus mehreren vorentworfenen Schaltungen, z.B. mehrere UND-Gatter mit zwei Eingängen zu einem NAND-Gatter mit mehr als zwei Eingängen und einem Ausgabeinverter. Nach der Synthese folgt der geometrische Entwurf.

Er umfasst die Anordnung der Teilschaltungen auf einem Schaltkreis oder einer Leiterplatte (Platzierung) und die geometrische Anordnung der Verbindungen (Verdrahtung). Aus der geometrischen Beschreibung werden abschließend die Fertigungsdaten und die tatsächlichen Verzögerungszeiten für die Simulationsmodelle berechnet.

Die Schaltungsgeneratoren für parametrisierte Schaltungen und der Einsatz vorentworfenener Teilschaltungen spielen in diesem Konzept eine Schlüsselrolle. Die Algorithmen für die Logikoptimierung verkraften nur Schaltungen aus bis zu einigen Hundert Gattern. Die Generierung erzeugt bereits lokal optimierte Schaltungen und unterliegt nicht dieser Größenbeschränkung. Damit die auf dem Standardweg zu verarbeitenden Schaltungsbeschreibungen

¹ UND- und ODER-Gatter mit zwei Eingängen

die beherrschbare Größe nicht überschreiten, werden die generierten und vor-entworfenen Teilschaltungen teilweise erst nach der Optimierung, nach der Technologieabbildung oder bereits als fertige Layouts zur Gesamtschaltung hinzugefügt.

Der Entwickler arbeitet bei der Synthese ähnlich wie bei einem Software-Entwurf überwiegend mit Funktionsbeschreibungen in einer Programmiersprache. Die »Millionen von Einzelgattern« erscheinen nur als Aufwands- und Gütekenngößen (Flächenverbrauch, Geschwindigkeit etc.) in den Meldungen der Entwurfswerkzeuge.

Simulation

Beim Entwurf großer Schaltungen entstehen zahlreiche Fehler, die idealerweise vor der Fertigung gefunden und beseitigt werden. Das erfordert ausgiebige Simulationen. Eine Schaltungssimulation benötigt zwei Verhaltensmodelle, ein Modell für das Testobjekt und einen Testrahmen. Das Simulationsmodell des Testobjekts ist praktisch ein ausführbares Programm, das die Ausgabesignale und die internen Signale aus den Eingabesignalen berechnet. Der Testrahmen ist gleichfalls ein ausführbares Programm. Er enthält das Testobjekt als Teilprogramm, erzeugt die Eingabesignale und wertet die Ausgabesignale aus. Jede Simulation ist ein Kompromiss zwischen dem Programmier- und dem Rechenaufwand auf der einen Seite und der Leistungsfähigkeit bei der Fehlererkennung auf der anderen Seite. Die Entwicklung ausreichend aussagekräftiger Simulationsmodelle und die Auswertung der Simulationsergebnisse ist meist erheblich aufwändiger als der eigentliche Schaltungsentwurf.

1.1.2 Modelle und Entwurfsraum

Definition 1.1 (Modell) *Ein Modell ist ein Mittel, um einen Zusammenhang zu veranschaulichen. Es stellt die wesentlichen Sachverhalte dar und verbirgt unwesentliche Details.*

Definition 1.2 (formales Modell) *Ein formales Modell ist eine eindeutige Beschreibung der wesentlichen Merkmale, in der Regel ein auf einem Rechner abarbeitbares Simulationsmodell.*

Definition 1.3 (informales Modell) *Ein informales Modell ist eine Beschreibung wesentlicher Zusammenhänge in einer anschaulichen Form ohne Rücksicht auf Eindeutigkeit oder Vollständigkeit.*

Definition 1.4 (halbformales Modell) *Ein halbformales Modell ist eine Beschreibung, aus der Fachleute in der Regel dasselbe herauslesen.*

Moderne informationsverarbeitende Systeme werden objektorientiert entworfen. Objektorientierung ist in ihrem Kern eine Teile-und-Herrsche-Methode:

Gliedere den Entwurf in überschaubare und damit beherrschbare Teilaufgaben.

Ausgangspunkt sind die zu verarbeitenden Datenobjekte. Datenobjekte, die in derselben oder in ähnlicher Weise verarbeitet werden, werden zu Klassen zusammengefasst und um Bearbeitungsmethoden ergänzt. Beim digitalen Schaltungsentwurf sind die Objekte Schaltungsbeschreibungen, Simulationsdaten, Testdaten und vieles mehr. Die Methoden sind Analyse, Visualisierung, Verarbeitung etc. Der Entwurfsraum beschreibt das Klassifizierungsschema der Entwurfsobjekte und die Beziehungen zwischen ihnen.

Datenobjekte sind im Sinne der Informatik Modelle, die jeweils bestimmte Sachverhalte beschreiben. Bei manuellen Entwürfen werden überwiegend informale und halbformale Modelle entwickelt, bei denen die Anschaulichkeit im Vordergrund steht. Die rechnergestützte Verarbeitung verlangt formale Modelle, die die bearbeiteten Sachverhalte eindeutig und vollständig beschreiben. Der Entwurfsraum für einen digitalen Schaltungsentwurf untergliedert die Entwurfsobjekte außer nach der Modellkategorie (informal, halbformal und formal) noch nach der Hierarchieebene, der sie zugeordnet sind (Gesamtsystem, Teilsystem etc.), und nach der Beschreibungsart (Funktionsbeschreibung, Strukturbeschreibung, geometrische Beschreibung und Testbeschreibung, Abb. 1.3).

Beschreibungsart Hierarchieebene	Funktionsbeschreibung	Strukturbeschreibung	geometrische Beschreibung	Testbeschreibung
Gesamtsystem				
Teilsysteme				
Funktionsblöcke	Entwurfsraum			
Gatter				
Transistoren				

Abb. 1.3. Entwurfsraum digitaler Schaltungen

Eine Funktionsbeschreibung ist ein Algorithmus oder Formalismus, der die Abbildung der Eingaben einer Schaltung auf ihre Ausgaben beschreibt. Das kann ein ganz normales Programm sein. Funktionsbeschreibungen dienen zur Simulation der Zielfunktion und als Funktionsvorgabe für die Schaltungssynthese.

Eine Strukturbeschreibung beschreibt die Zusammensetzung eines Systems aus Teilsystemen. Sie enthält zusätzliche Informationen über die Realisierung und kann durch Einsetzen der Funktionsbeschreibungen für die Teilschaltungen in eine Funktionsbeschreibung des Gesamtsystems umgerechnet

werden. Jeder Struktur ist damit eindeutig eine Funktion zugeordnet. Umgekehrt kann jede Funktion durch praktisch unbegrenzt viele Schaltungen nachgebildet werden, die sich im Aufwand und anderen Optimierungskenngrößen unterscheiden. Die Suche nach einer geeigneten Struktur für eine Funktion ist praktisch eine Optimierungsaufgabe.

Eine geometrische Beschreibung beschreibt die Abmessungen und die Anordnung der einzelnen Teilschaltungen und Verbindungen auf einer Leiterplatte oder auf einem Schaltkreis. Sie enthält die Strukturbeschreibung, ist aber für strukturbasierte Algorithmen viel zu überladen. Die Struktur ist eindeutig zugeordnet. Umgekehrt kann jede Strukturbeschreibung durch unbegrenzt viele unterschiedliche geometrische Beschreibungen nachgebildet werden. Das Ergebnis des geometrischen Entwurfs bildet die Grundlage für die Generierung der Fertigungsdaten sowie für die Berechnung der tatsächlichen Verzögerungszeiten, des tatsächlichen Schaltungsaufwands etc.

Die Testbeschreibungen werden zur Aufdeckung von Entwurfs- und Fertigungsfehlern benötigt. Sie beschreiben Testabläufe, Testeingaben, Soll-Ausgaben, Problemöglichkeiten, Plausibilitätskriterien für die Ausgaben etc. Die Entwicklung der Testbeschreibungen verursacht in der Regel mehr als die Hälfte des Gesamtentwurfsaufwands (Abb. 1.4).

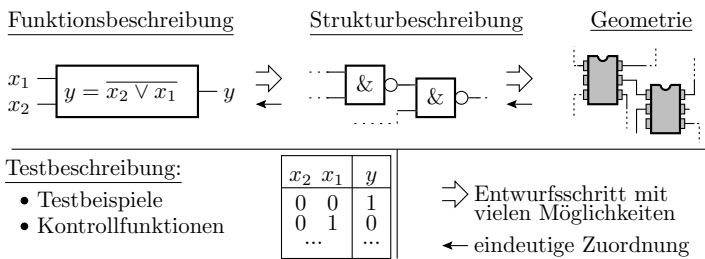


Abb. 1.4. Beschreibungsarten und ihre Beziehungen untereinander

1.1.3 Entwurfsablauf

Das Voranschreiten eines Entwurfsprojekts lässt sich als eine Bewegung in dem Entwurfsraum in Abb. 1.3 beschreiben. Es gibt zwei Grundstrategien: Top-Down- und Bottom-Up-Entwurf. Ein Top-Down-Entwurf beginnt mit der Spezifikation der Anforderungen. Zuerst werden die Ziele und die wesentlichen Anforderungen zusammengestellt. Darauf aufbauend werden schrittweise informale und halbformale bis hin zu formalen, im Rechner simulierbaren Funktionsbeschreibungen entwickelt. Der Eintrittspunkt in den rechnergestützten Entwurf ist die obere linke Ecke des Entwurfsraums. Von dort aus startet die Entwicklung der Struktur. Zuerst wird die Gesamtfunktion entworfen und

durch eine Struktur aus miteinander kommunizierenden Teilschaltungen nachgebildet. Danach wird rekursiv absteigend mit den Teilschaltungen genauso verfahren. Später im Entwurfsprozess werden die Teilschaltungen platziert und verdrahtet. Der Entwurfsraum füllt sich tendenziell von oben links nach unten rechts. Nach jedem Entwurfsschritt folgen Simulationen und andere Kontrollen, ob die gefundene Lösung die Entwurfsziele befriedigt (Abb. 1.5).

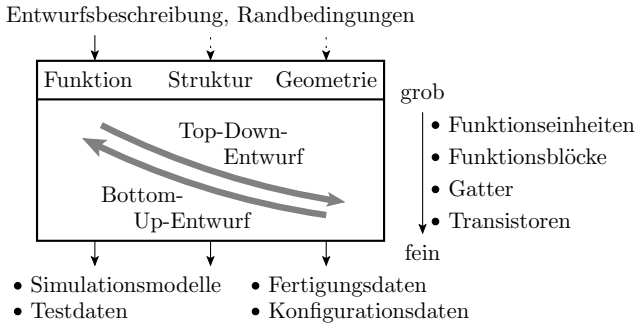


Abb. 1.5. Entwurfsstrategien

Ein Bottom-Up-Entwurf durchläuft den Entwurfsraum in entgegengesetzter Richtung. Aus Bauteilen mit bekannter Funktion und Geometrie werden größere Funktionseinheiten zusammengesetzt. Die Optimierungsziele hinsichtlich

- der Größe,
- der Geschwindigkeit,
- des Stromverbrauchs und
- der Testbarkeit

lassen sich dabei besser berücksichtigen. Aber es ist schwer, auf diese Weise eine vorgegebene Zielfunktion nachzubilden. Der Bottom-Up-Entwurf dient deshalb meist dazu, Schaltungen quasi auf Vorrat zu konstruieren und in Bibliotheken oder in Form von Schaltungsgeneratoren für den Top-Down-Entwurf bereitzustellen. Vorentworffene Schaltungen kann es für jede Hierarchieebene geben: Gatter, Rechenwerke, Prozessoren, Speichergeneratoren etc. Praktische Entwürfe arbeiten entsprechend nach einer Mischstrategie, bei der sich Top-Down- und Bottom-Up-Entwurf in der Mitte treffen.

Mensch-Maschine-Interaktion

Die vielen unterschiedlichen Beschreibungsarten und die Werkzeuge² zu ihrer Bearbeitung werden vom Rechner in einer für den Entwickler handhabbaren

² übliche Bezeichnung für Entwurfsprogramme

Form verwaltet. Um die Daten- und Werkzeugverwaltung in einem Entwurfssystem für digitale Schaltungen zu verstehen, muss man danach fragen, wie der Entwickler damit arbeitet. Er bekommt informale Zielvorgaben und setzt sie manuell in formale, für den Rechner verständliche Funktionsbeschreibungen um. Die Schnittstelle für die Rechnereingabe bilden Programmiersprachen und Eingabemasken. Rechnerintern werden alle neuen und geänderten Eingabedaten zuerst analysiert. Die Analyse generiert Fehlermeldungen oder Erfolgsmeldungen und weiterverarbeitbare Datenobjekte. Die Werkzeuge für die Weiterverarbeitung sind Simulatoren, Analysewerkzeuge zur Bestimmung von Kennwerten, z.B. für die Geschwindigkeit, Synthesewerkzeuge zur Umwandlung von Funktions- in Strukturbeschreibungen und einige mehr. Der Entwickler gibt nach jeder Eingabeänderung das zu bearbeitende Entwurfsobjekt³ und das Bearbeitungsziel (nur Analyse, auch Simulation etc.) vor. Der Rechner arbeitet, wenn kein Fehler auftritt, alle Schritte bis zu diesem Entwurfsziel ab und liefert Rückmeldungen in Form von Erfolgsmeldungen, Warnungen, Gütekennzahlen etc. Aus diesen Rückmeldungen schätzt der Entwickler ab, was weiter zu tun ist. Entweder er bessert iterativ nach, indem er seine Eingabebeschreibung ändert und die Berechnung neu startet, oder er beginnt mit dem nächsten Entwurfsschritt. Insgesamt ähnelt dieser Prozess einem Regelkreis mit dem Entwickler als Regler und dem Rechner als Regelstrecke (Abb. 1.6).

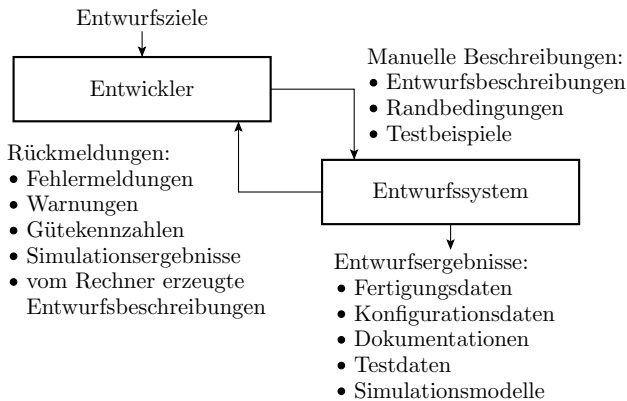


Abb. 1.6. Die Mensch-Maschine-Interaktion in einem Entwurfsprozess

Entwurfsautomatisierung

Ein Entwurfsschritt beinhaltet in der Regel Entscheidungen zwischen vielen Alternativen. Das gilt für die Festlegung der Funktion, die strukturelle

³ in einer hierarchischen Beschreibung das oberste Objekt in der Beschreibungshierarchie

Nachbildung der Zielfunktion und für die geometrische Anordnung und Verdrahtung der Strukturelemente. Entwurfswerkzeuge zur Automatisierung von Entwurfsschritten arbeiten deterministisch. Das Ergebnis ist eine Funktion der Entwurfseingabe sowie von optionalen Steuerparametern. Ohne Steuerparameter legt die Entwurfsvorgabe die Lösung, die gefunden wird, eindeutig fest. Bei dieser Einschränkung des Lösungsraums fallen natürlich nicht nur schlechte, sondern auch gute Lösungen weg (Abb. 1.7).

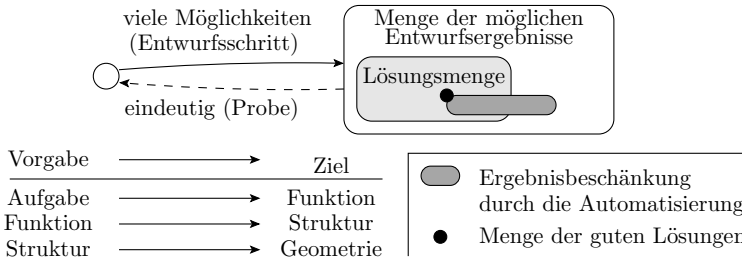


Abb. 1.7. Beschränkung des Lösungsraums von Entwurfsschritten durch die Automatisierung

Die Beschränkung des Lösungsraums hat zur Folge, dass ein einzelner automatisierter Entwurfsschritt im Durchschnitt schlechtere Ergebnisse als eine gute Entwurfsentscheidung in einem Handentwurf liefert. Gut und schlecht bezieht sich dabei auf Kenngrößen wie den Schaltungsaufwand, die Geschwindigkeit etc. Aber die Automatisierung hat auch prinzipielle Vorzüge. Der Rechner ist viel schneller und kann in kürzerer Zeit viel mehr Lösungsmöglichkeiten untersuchen und vergleichen. Daraus folgt eine spezielle Optimierungsstrategie. Ähnlich wie bei einem genetischen Algorithmus wird die Eingabebeschreibung variiert. Gute Lösungsdetails werden mit anderen guten Lösungsdetails kombiniert. Ein mit dieser Art der Optimierung gut vertrauter Entwickler ist durchaus in der Lage, Schaltungen zu entwerfen, die ähnlich wenig Chipfläche benötigen und vergleichbar schnell sind wie ein professioneller Handentwurf. Dazu kommen die Vorteile einer Entwurfsautomatisierung, nämlich

- deutlich kürzere Gesamtentwurfszeiten,
- besser vorhersagbare Güte- und Aufwandskennzahlen für die Projekt- und Kostenplanung und
- eine geringere zu erwartende Anzahl von Entwurfsfehlern.

1.1.4 VHDL als formale Beschreibungsplattform

Definition 1.5 (Syntax einer Programmiersprache) Die *Syntax* einer Programmiersprache ist ein System von Regeln, nach denen erlaubte Konstrukte (Bezeichner, Anweisungen etc. bis hin zu kompletten Programmen) aus Zeichen eines Zeichenvorrats zusammengesetzt werden.

Definition 1.6 (Entwurfseinheit) Eine Entwurfseinheit (*entity**) ist eine abgegrenzte Funktionseinheit, die über Anschlussignale mit anderen Entwurfseinheiten kommuniziert.

Definition 1.7 (Entwurfsbeschreibung) Eine Entwurfsbeschreibung (*architecture**) beschreibt die innere Struktur oder Funktion einer Entwurfseinheit.

Definition 1.8 (Testrahmen) Ein Testrahmen (*testbench**) ist eine Entwurfseinheit zur Beschreibung eines Testablaufs.

Definition 1.9 (Anschlussignal) Ein Anschlussignal (*port**) ist ein Signal, über das eine Entwurfseinheit mit ihrer Umgebung kommuniziert.

Definition 1.10 (Prozess) Ein Prozess ist ein Programmrahmen für eine imperative Beschreibung eines Teilsystems in einem nebenläufig arbeitenden Gesamtsystem.

Definition 1.11 (Package) Ein Package ist eine Sammlung von Vereinbarungen und Unterprogrammen für die Nutzung in Entwurfseinheiten und anderen Packages.

Definition 1.12 (Bibliothek) Eine Bibliothek (*library**) ist eine Sammlung von Entwurfseinheiten und Packages.

(* – VHDL-Schlüsselwort zur Vereinbarung oder Referenzierung eines Objekts von diesem Typ.)

Eine formale, manuell lesbare Beschreibung erfolgt in der Informatik vorzugsweise in einer künstlichen Sprache. In den Anfangszeiten des digitalen Schaltungsentwurfs hatten viele Entwurfswerkzeuge ihre eigenen Sprachen zur Beschreibung der Entwurfsobjekte. Inzwischen haben sich zumindest für die manuellen Aufgabenbereiche wenige universelle, standardisierte Hardware-Beschreibungssprachen durchgesetzt.

Die in Europa verbreitetste Hardware-Beschreibungssprache ist VHDL. Andere Hardware-Beschreibungssprachen sind Verilog und System-C [31]. Das Akronym VHDL steht für

V	VHSIC (Very High Speed Integrated Circuits)
H	Hardware
D	Description
L	Language.

Mit VHDL können Funktionen, Strukturen und Testabläufe vom Gesamtsystem bis herunter zum Gatterniveau formal beschrieben werden (Abb. 1.8).

Als Begleitmaterial zum Buch wurden zahlreiche Beschreibungsbeispiele in VHDL entwickelt und mit dem frei verfügbaren Simulator GHDL [3] und dem ebenfalls für Studenten frei verfügbaren Entwurfssystem ISE von Xilinx [5] getestet. Die Beispiele stehen im Internet unter [27] und laufen

Beschreibungsart Ebene in der Hierarchie	Funktionsbeschreibung	Strukturbeschreibung	geometrische Beschreibung	Test-Beschreibung
Gesamtsystem	+	+		+
Teilsysteme	+	+		+
Funktionsblöcke	+	+		+
Gatter	+	+		+
Transistoren	·	·		·

+ üblich
· möglich

Abb. 1.8. Mit VHDL beschreibbare Entwurfsobjekte

unter Linux und Windows. Der VHDL-Simulator GHDL ist ein Kommandozeilenprogramm, das VHDL-Programme in die Sprache »C« und weiter in ausführbare Programme übersetzt. Die Visualisierung der berechneten Signalverläufe erfolgt mit dem gleichfalls unter Linux und Windows frei verfügbaren Programm GTKWAVE [4]. Das Entwurfssystem ISE dient zum Entwurf digitaler Schaltungen mit programmierbaren Logikschaltkreisen von Xilinx. Für die Beispiele genügt die für Studenten freie Web-Version. Um die beschriebenen Schaltungen nicht nur zu analysieren und zu simulieren, sondern auch auszuprobieren, wird zusätzlich ein Prototyp-Board benötigt. Eine gute und kostengünstige Wahl ist das Spartan-3-Board von Digilent Inc. [2], für das es gute Praktikumsanleitungen und sogar ein Lehrbuch mit Übungsbeispielen gibt [17].

Das Grundkonzept von VHDL

VHDL wurde ursprünglich zur Simulation digitaler Schaltungen entwickelt, lehnt sich an die Programmiersprache Ada an und ist unter IEEE Std.1076 standardisiert [9]. Ein VHDL-Projekt besteht aus Entwurfseinheiten und Packages, die in Bibliotheken verwaltet werden. Entwurfseinheiten dienen zur Beschreibung von Schaltungsmodellen und Testrahmen. Packages sind Sammlungen von Datentypen, Konstanten und Unterprogrammen.

Abbildung 1.9 zeigt die Grundstruktur einer VHDL-Entwurfseinheit. Die Beschreibung der Sprache erfolgt in der Backus-Naur-Form (BNF). Schlüsselwörter sind **fett** und Platzhalter (Nicht-Terminal-Symbole) für einzusetzende Beschreibungselemente *kursiv* dargestellt. Beschreibungselemente in [...] sind optional. Sie können, müssen aber nicht enthalten sein. Beschreibungselemente in {...} können beliebig oft enthalten sein. VHDL unterscheidet nicht zwischen Groß- und Kleinschreibung. Hier im Buch wird für alle vordefinierten Bezeichner aus standardisierten Bibliotheken zur Unterscheidung von selbst definierten Bezeichnern die Schriftart CAPITALIZED verwendet. Kommentare

beginnen mit »--«. Weitere Regeln, die Liste der Schlüsselworte, Bildungsregeln für Bezeichner etc. sind im Anhang A.1.1 nachzulesen. Das Standardwerk zu VHDL ist [11]. Lehrbücher sind z.B. [16, 17, 26, 31, 33, 37, 41].

```
-- Vorspann
1: library IEEE;
2: use IEEE.STD_LOGIC_1164.all;
-- Schnittstellenbeschreibung
3: entity Bezeichner_Entwurfseinheit is
4:   [generic (Liste_der_Konfigurationsparameter);]
5:   [port(Liste_der_Anschlussignale);]
6: end entity;
-- Beschreibung der Funktion oder Struktur
7: architecture Beschreibungsname of Bezeichner_Entwurfseinheit is
8:   { Vereinbarung }
9: begin
10:  { Beschreibungselement_für_die_Struktur_oder_Funktion }
11: end architecture;
```

Abb. 1.9. Beschreibungsstruktur einer VHDL-Entwurfseinheit

Die Beschreibung einer Entwurfseinheit besteht aus dem Vorspann, einer Schnittstellenbeschreibung und einer oder mehreren Beschreibungen der Funktion oder Struktur. Der Vorspann importiert Bezeichner für Datentypen, Unterprogramme etc. aus Packages in den Namensraum der Entwurfseinheit. Zeile 1 definiert den Bezeichner IEEE⁴ als Bibliothek. Zeile 2 importiert alle Vereinbarungen aus dem Package IEEE.STD_LOGIC_1164. Die Schnittstelle einer Schaltung beschreibt die Anschlussignale (Zeile 5) und für parametrisierte Entwurfsbeschreibungen auch die Konfigurationsparameter (Zeile 4) als Liste. Die Beschreibung des Innenlebens erfolgt getrennt von der Schnittstelle (Zeile 7 bis 11) und hat einen eigenen Bezeichner (Zeile 7). Der Beschreibungsname dient in übergeordneten Strukturbeschreibungen zur Unterscheidung zwischen unterschiedlichen Beschreibungsversionen mit derselben Schnittstelle. Im Vereinbarungsteil der Beschreibung (Zeile 8) werden hauptsächlich die internen Signale und Konstanten festgelegt. Der Anweisungsteil (Zeile 10) kann drei Arten von Beschreibungselementen enthalten:

- Beschreibungen zur Einbindung von Teilschaltungen,
- Prozesse als Rahmen für imperative Beschreibungen und
- nebenläufige Anweisungen⁵.

⁴ IEEE ist eine Bibliothek mit standardisierten Packages, die in jeder VHDL-Umgebung verfügbar ist.

⁵ nebenläufige Signalzuweisungen und nebenläufige Prozeduraufrufe als Kurzschreibweisen für spezielle Prozesse mit nur einer Anweisung

Zur Simulation muss eine Entwurfseinheit als Teilschaltung in einen Testrahmen eingebettet werden. Ein Testrahmen ist eine Entwurfseinheit ohne Konfigurationsparameter und ohne Anschlüsse. Das kleinste ausführbare VHDL-Programm ist ein Testrahmen mit einem Prozess, der eine Meldung ausgibt und sich beendet (Abb. 1.10).

```

1: entity HalloWelt is
2: end entity;

3: architecture Verhalten of HalloWelt is
4: begin
5:   process
6:   begin
      -- Meldung ausgeben und Prozess beenden
7:   report "Hallo Welt";
8:   wait;
9:   end process;
10: end architecture;

```

⇒ WEB-Projekt: P1.1/HalloWelt.vhdl

Abb. 1.10. Ausführbares VHDL-Programm (Zeile 1-2: Schnittstelle; Zeile 3-10: Beschreibung; Zeile 5-9: Prozess; Zeile 7: Textausgabe; Zeile 8: Anweisung, die den Prozess schlafen legt)

Ein Prozess ist der Programmrahmen für eine imperative Beschreibung. Die eingerahmte Anweisungsfolge wird zur Simulation in eine Endlosschleife übersetzt und zyklisch Schritt für Schritt abgearbeitet. Nach der letzten Anweisung folgt wieder die erste Anweisung. Jeder Prozess muss deshalb mindestens eine Warteanweisung enthalten, die ihn schlafen legt und den Kontrollfluss an den Simulator zurückgibt. Im Beispiel hat die Warteanweisung keine Weckbedingung, d.h., der Prozess terminiert dauerhaft. Da es der einzige Prozess ist, endet auch die gesamte Simulation nach der Textausgabe.

Die Simulation soll mit dem Simulator GHDL erfolgen. Die Befehle für die Analyse, das Übersetzen und den Start der Simulation des VHDL-Programms lauten:

Kommando	Aktion
ghdl -a HalloWelt.vhdl (analyze)	Syntaxtest und Ablegen der vorübersetzten Beschreibung in der Arbeitsbibliothek »work«
ghdl -m HalloWelt (make)	Zusammensetzen aller Beschreibungsteile zu einem ausführbaren Programm
ghdl -r HalloWelt (run)	Start der Simulation

Das Beispielprogramm gibt nach dem Simulationsstart nur die Meldung

HalloWelt.vhdl:7:5:@0ms:(report note): Hallo Welt

aus und beendet sich.

Außer Entwurfseinheiten gibt es in VHDL eine zweite Art von Beschreibungen. Das sind die Packages. Ein Package besteht aus einem Kopf mit Vereinbarungen von Datentypen, Konstanten, Unterprogrammen etc. und einem Körper mit den Beschreibungen der Unterprogramme. Im Package in Abb. 1.11 sind im Kopf eine Textkonstante und eine Funktion vereinbart. Die Funktion bildet aus einem Eingabetext einen Ausgabertext. Aus der Beschreibung im Package-Körper ist abzulesen, dass der übergebene Text einfach nur unverändert zurückgegeben wird.

```

1:  package HalloWelt_pack is
2:    constant c: STRING := "Hallo Zeichenkette";
3:    function SchreibeText(s: STRING) return STRING;
4:  end package;

5:  package body HalloWelt_pack is
6:    function SchreibeText(s: STRING) return STRING is
7:      begin
8:        return s;
9:      end function;
10: end package body;
```

⇒WEB-Projekt: P1.1/HalloWelt_pack.vhdl

Abb. 1.11. Package für das Projekt »Hallo Welt«; Zeile 1-4: Kopf; Zeile 5-10: Körper; Zeile 2: Konstantenvereinbarung; Zeile 3: Funktionsvereinbarung; Zeile 6-9: Funktionsbeschreibung

Zur Nutzung des Packages muss es zuerst analysiert und in eine Bibliothek – standardmäßig die Arbeitsbibliothek `WORK` – abgelegt werden. Von dort aus können die Vereinbarungen – im Beispiel die der Konstanten »c« und der Funktion »SchreibeText(...)« – von jeder Entwurfseinheit und jedem anderen Package importiert werden.

Abbildung 1.12 zeigt ein erweitertes Hallo-Welt-Programm, das die beiden Package-Vereinbarungen nutzt. Der Bibliotheksbezeichner `WORK` für die Arbeitsbibliothek ist vordefiniert. Der Vorspann beginnt deshalb gleich mit der Use-Anweisung (Zeile 1). Die Schnittstellenbeschreibung (Zeile 2 und 3) ist wieder leer. Die beiden ersten Warteanweisungen (Zeile 9 und 11) haben eine Wartezeit als Weckbedingung und legen den Prozess für 1 ms Simulationszeit⁶ schlafen. Nach dem Aufwachen gibt der Prozess jeweils die nächste Meldung aus. Die erste Meldung (Zeile 8) erscheint zur Simulationszeit »@0ms«. Die zweite Meldung mit der Textkonstante »c« (Zeile 10) wird zur Simulationszeit »@1ms« und die dritte Meldung mit dem Rückgabewert der Funktion (Zeile

⁶ Die Simulationszeit ist eine Rechengröße im Simulator, die im Beispiel von den beiden ersten Warteanweisungen jeweils um die Wartezeit weitergestellt wird.

12) zur Simulationszeit »@2ms« ausgegeben. Die letzte Warteanweisung (Zeile 13) legt den Prozess dauerhaft schlafen und beendet die Simulation.

```

1: use work.HalloWelt_pack.all;
2: entity HalloWelt1 is
3: end entity;

4: architecture Verhalten of HalloWelt1 is
5: begin
6:   process
7:   begin
8:     report "Hallo Welt";
9:     wait for 1 ms;
10:    report c;
11:    wait for 1 ms;
12:    report SchreibeText("Hallo Funktionsaufruf");
13:    wait;
14:   end process;
15: end architecture;
```

⇒ WEB-Projekt: P1.1/HalloWelt1.vhdl

Abb. 1.12. Nutzung des Packages

Zur Ausführung werden zuerst das Package und dann die Entwurfseinheit analysiert. Dann wird aus beiden ein ausführbares Programm gebaut und dieses gestartet:

```

ghdl -a HalloWeltPack.vhdl
ghdl -a HalloWelt1.vhdl
ghdl -m HalloWelt1
ghdl -r HalloWelt1
```

Der Aufruf mit Option »m« bedeutet »make«. Er baut das ausführbare Simulationsobjekt neu zusammen und aktualisiert, falls die Quelldateien geändert wurden, vorher automatisch die Objektdaten in den Bibliotheken. Die Dateien mit den VHDL-Beschreibungen und ein Hilfetext dazu stehen im Web unter [27].

1.1.5 Zusammenfassung und Übungsaufgaben

Der Entwurf digitaler Schaltungen gleicht heute in vielerlei Hinsicht einem Software-Entwurf. Er erfolgt rechnergestützt und in einer Hochsprache. Die Beschreibungen sind hierarchisch strukturiert. Die Routinearbeit erledigt der Rechner. Der manuelle Entwurfsaufwand entfällt überwiegend auf den Test und die Fehlersuche. Mit den Millionen von Gattern der fertigen Schaltung beschäftigt sich ein Hardware-Entwickler genauso wenig wie ein Software-Entwickler mit den Millionen von Maschinenbefehlen seines fertigen Programms. Es gibt aber auch Unterschiede. Der Hardware-Entwurf

kennt zwei zusätzliche Beschreibungsarten: Strukturbeschreibungen und geometrische Beschreibungen. Das vergrößert den Entwurfsraum und die Entwurfskomplexität erheblich. Die Kenngrößen für die Optimierung – Aufwand, Geschwindigkeit, Stromverbrauch etc. – haben bei Hardware einen höheren Stellenwert und sind erst später im Entwurfsprozess genau bestimmbar. Ein Hardware-Entwurf ist dadurch kein vollständig automatisierbarer Vorgang wie das Übersetzen eines Programms, sondern eine Art manuell gesteuerte heuristische Suche. Ein Hardware-Entwickler nutzt in wesentlich größerem Umfang vorentworfene Teilsysteme und investiert wesentlich mehr Zeit in die Optimierung, den Test und das Ausprobieren unterschiedlicher Alternativen. Das erfordert ein größeres Detailverständnis darüber, wie der Rechner Entwurfs- und Testbeschreibungen analysiert, simuliert und in Schaltungen übersetzt. Eine Schlüsselstellung nimmt dabei die Hardware-Beschreibungssprache ein, über die der Entwickler mit dem Rechner kommuniziert.

Aufgabe 1.1

Was ist in VHDL eine Entwurfseinheit und was ist ein Package?

Aufgabe 1.2

- Was sind eine Funktionsbeschreibung, eine Strukturbeschreibung und eine geometrische Beschreibung?
- Wozu dient der Testrahmen in einem Simulationsmodell?
- Was sind die Vorteile und die Nachteile formaler und informaler Modelle?

Aufgabe 1.3

Schreiben Sie ein VHDL-Programm, das fünfmal hintereinander im Abstand von zwei Millisekunden simulierter Zeit eine Meldung mit dem Nachrichtentext »Hallo« ausgibt. Testen Sie das Programm mit GHDL oder einem anderen VHDL-Simulator.

Aufgabe 1.4

Schauen Sie sich im Anhang A.1.3 die Bildungsregeln für Bezeichner an. Sind die Zeichenfolgen »To«, »3dimensional«, »bereit«, »tTyp« und »_escape« zulässige VHDL-Bezeichner?

1.2 Funktion, Struktur und Simulation

Definition 1.13 (Kombinatorische Schaltung) *Eine kombinatorische Schaltung (engl. combinational circuit, im deutschen Sprachraum auch Schaltnetz) ist eine digitale Schaltung ohne Speicherverhalten, die kontinuierlich binäre Eingabesignale auf binäre Ausgabesignale abbildet.*

Definition 1.14 (Kombinatorische Funktion) Eine kombinatorische Funktion beschreibt das logische Eingabe-Ausgabe-Verhalten einer kombinatorischen Schaltung ohne Berücksichtigung des Zeitverhaltens.

Definition 1.15 (Signal) Ein Signal ist der zeitliche Werteverlauf einer physikalischen Größe.

Definition 1.16 (Datentyp) Der Datentyp beschreibt den Wertebereich eines Datenobjekts und optional die Codierung der darstellbaren Werte.

1.2.1 Signale und Datentypen

In der digitalen Schaltungstechnik wird die zu verarbeitende Information durch Signale dargestellt. Ein Signal ist der zeitliche Werteverlauf einer physikalischen Größe. In einer Schaltung ist die physikalische Größe in der Regel die Spannung zwischen der signalführenden Leitung und dem Bezugspunkt ($\perp$). Die Digitaltechnik unterscheidet nur zwischen großen und kleinen Werten. Dazwischen gibt es einen Sicherheitsbereich. In diesem Bereich ist der logische Wert ungültig und darf nicht weiterverarbeitet werden. Der Sicherheitsbereich verhindert eine Fehlklassifikation von »0« als »1« und umgekehrt (Abb. 1.13).

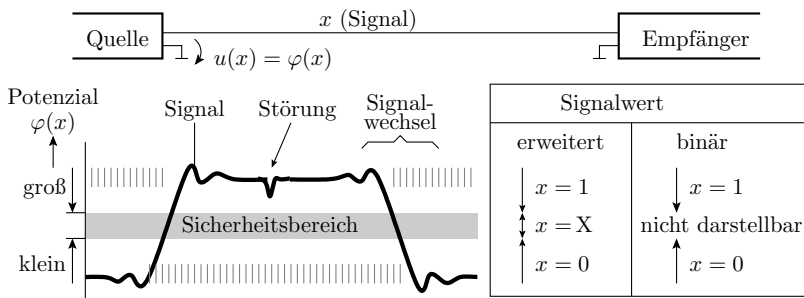


Abb. 1.13. Signal zur Darstellung von Logikwerten

Binäre Datentypen

Jedes Datenobjekt hat einen Typ, der den Wertebereich und die Wertedarstellung beschreibt. Ein zweiwertiges Datenobjekt wird umgangssprachlich als Bit bezeichnet. Die vordefinierten bitorientierten VHDL-Datentypen sind:

```

type BIT(1) is ('0', '1')(2);
type BOOLEAN(1) is (FALSE, TRUE)(3);
type STD_LOGIC(4) is ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');
  
```

(⁽¹⁾ – definiert im Package STD.STANDARD; (⁽²⁾ – die Elemente sind druckbare Zeichen; (⁽³⁾ – die Elemente sind symbolische Konstanten; (⁽⁴⁾ – definiert im Package IEEE.STD_LOGIC_1164; 'U' – nicht initialisiert; 'X' – unbestimmt, 'Z' inaktiv/hochohmig; 'W', 'L', 'H' – schwache, von einer normalen Quelle überschreibbare Werte; '-' – beliebig). Die Datentypen BIT und STD_LOGIC stellen die beiden zu unterscheidenden Logikwerte als druckbare Zeichen dar. Im Weiteren gilt die Zuordnung, dass kleine Werte durch das Zeichen »0« und große Werte durch das Zeichen »1« dargestellt werden. Der Typ STD_LOGIC kann zusätzlich zu »0« und »1« auch eine Reihe von Pseudo-Werten darstellen, die alle bei der Simulation eine spezielle Bedeutung haben. Der Pseudo-Wert für ungültig ist »X«. Der Typ BOOLEAN stellt binäre Informationen mit den symbolischen Wahrheitswerten FALSE und TRUE dar.

Halbformale Beschreibungen (Datenblätter etc.) verwenden statt »0« und »1« oft die Bezeichnungen »inaktiv« und »aktiv«. Diese Bezeichnungen beziehen sich auf die Funktion eines Signals. Es gibt z.B. Freigabesignale für Treiber, Initialisierungssignale für Register und Schreib- und Lesesignale für Blockspeicher. Aktiv bedeutet, dass die zugeordnete Funktion ausgeführt wird, und inaktiv, dass sie nicht ausgeführt wird. Ein aktives Schreibsignal bewirkt einen Schreibvorgang, ein aktives Lesesignal einen Lesevorgang. Der aktive Signalwert, der die Funktion auslöst, kann dabei entweder »groß« oder »klein« sein. Ist der aktive Signalwert der große Wert, spricht man von einem high-aktiven und sonst von einem low-aktiven Signal.

Bitvektoren

Größere Informationseinheiten werden in einer digitalen Schaltung mit mehreren Bits dargestellt. Eine Information aus n Bits erlaubt es, bis zu

$$N \leq 2^n \quad (1.1)$$

Werte zu unterscheiden (Abb. 1.14 a). Umgekehrt verlangt die Unterscheidung von $N > 2$ Werten – z.B. von N Zahlenwerten – einen Bitvektor mit einer Bitanzahl von mindestens

$$n \geq \log_2(N) \quad (1.2)$$

Signalwerte aus mehreren Einzelbits sind gültig, wenn alle Bits gültig sind (Abb. 1.14 b).

Bitvektoren der Sprache VHDL sind eindimensionale Felder mit dem Elementtyp BIT bzw. STD_LOGIC:

```
type BIT_VECTOR is array(4) (NATURAL range <>)(5) of BIT(6,7);  
type STD_LOGIC_VECTOR is array (NATURAL range <>) of STD_LOGIC;(8)
```

(⁽⁴⁾ – Schlüsselwort für die Definition von Feldern; (⁽⁵⁾ – Beschränkung des Indexbereichs; (⁽⁶⁾ – Elementtyp; (⁽⁷⁾ – definiert im Package STD.STANDARD; (⁽⁸⁾ – definiert im Package IEEE.STD_LOGIC_1164). Den einzelnen Werten eines Bitvektors können wieder Bedeutungen zugeordnet sein, z.B. die Zustände

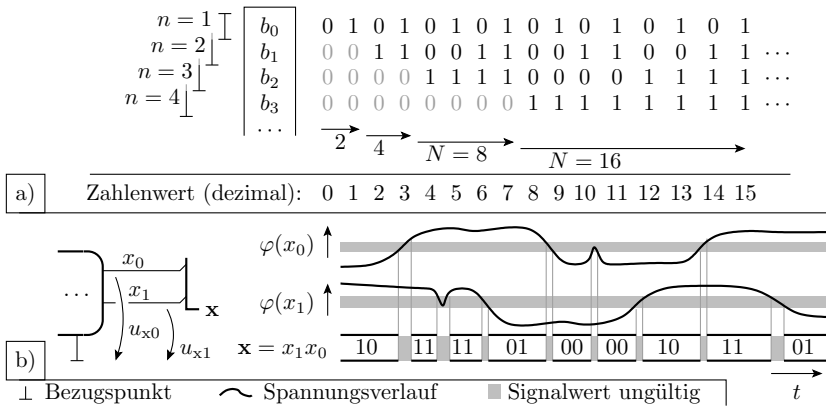


Abb. 1.14. Informationsdarstellung mit Bitvektoren a) Bitanzahl und die Anzahl der darstellbaren Werte b) Signalverlauf und Gültigkeitsfenster

einer Steuerung oder Zahlenwerte. Für die Darstellung von Zahlenwerten gibt es aber auch eigene Typen. Der vordefinierte Basistyp für ganze Zahlen ist `INTEGER`. Von `INTEGER` sind zwei Untertypen abgeleitet:

```
subtype NATURAL is INTEGER range 0 to INTEGER'HIGH;
subtype POSITIVE is INTEGER range 1 to INTEGER'HIGH;
```

(`'HIGH` – Attribut für Zahlentypen, das den größten darstellbaren Wert zurückgibt). Zahlentypen werden z.B. für die Elementauswahl in Vektoren und Feldern benötigt.

Die Beschreibung »(NATURAL **range** <>)« beschränkt den Indexbereich auf den Wertebereich von NATURAL. Das bedeutet, dass für die Indexgrenzen und Indexwerte nur ganzzahlige Werte von null bis zum Maximalwert von `INTEGER` zugelassen sind. Wir kommen auf die hier eingeführten Datentypen später zurück.

Simulierte Zeit

Signale sind zeitliche Werteverläufe, die durch Wert-Zeit-Tupel dargestellt werden. Die Zeit hat in VHDL den vordefinierten Basistyp `TIME`. Zeitangaben bestehen immer aus einem Wert und einer Maßeinheit. Zulässige Maßeinheiten der Zeit sind fs (Femtosekunden), ps (Pikosekunden), ns (Nanosekunden), us (Mikrosekunden), ms (Millisekunden), sec (Sekunden), min (Minuten) und hr (Stunden) (siehe später physikalische Typen, Abschnitt 3.2.3). Für Verzögerungszeiten ist davon ein Untertyp abgeleitet, der nur nicht negative Zeiten darstellen kann:

```
subtype DELAY_LENGTH is TIME 0 fs to TIME'HIGH;
```

1.2.2 Signalflussplan

Der Signalflussplan (Schaltplan oder Signalflussgraph) stellt ein signalverarbeitendes System als Graph mit den Verarbeitungsbausteinen als Knoten und den Signalen als Kanten dar. Die Kanten sind gerichtet. Jedes Signal hat eine Quelle, die den Signalverlauf einspeist, und einen oder mehrere Empfänger, die den Signalverlauf auswerten⁷. Der Signalfluss einer kombinatorischen Schaltung ist darüber hinaus zyklensfrei und verläuft von den Eingängen zu den Ausgängen. In der graphischen Darstellung verläuft der Signalfluss vorzugsweise von links nach rechts. Abbildung 1.15 zeigt ein Beispiel. Die Eingangssignale x_1 bis x_4 werden von der Schaltungsumgebung bereitgestellt. Die drei Gatter bilden jeweils die Eingangssignale auf der linken Seite des Symbols auf das Ausgabesignal auf der rechten Seite ab.

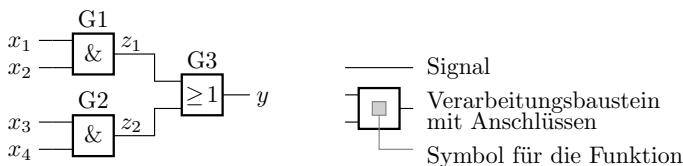


Abb. 1.15. Beispiel für einen Signalflussplan

Logische Grundbausteine

Signalflusspläne sind Strukturbeschreibungen. Auf der Gatterebene sind die Grundbausteine logische Gatter mit den Funktionen UND, ODER etc. Ihre Symbole sind gleich mehrfach in unterschiedlicher Weise standardisiert. Die im Weiteren verwendeten Symbole lehnen sich an den Standard IEC 60617-12 an (Abb. 1.16). Auf der linken Symbolseite werden die Eingangssignale und auf der rechten Symbolseite das Ausgabesignal angeschlossen. Die Gattersymbole außer dem Treiber und dem Inverter können auch mehr als zwei Eingänge haben.

Ein weiterer Grundbaustein ist der Multiplexer, ein Signalflussumschalter. Er hat mindestens zwei Dateneingänge x_i und einen Auswahleingang s (engl. select). In Abhängigkeit vom Wert am Auswahleingang wird eines der Dateneingangssignale an den Ausgang y weitergeleitet. Signalverarbeitende Bausteine mit Funktionen, für die es kein spezielles Symbol gibt, werden im Weiteren durch abgerundete Rechtecke mit der Angabe der nachgebildeten Funktion dargestellt.

⁷ Bei Signalen mit mehreren Quellen ist gleichzeitig nur eine Quelle aktiv.

logische Funktion	Symbol	logische Funktion	Symbol
Treiber (Identität): $y \leq x$	$x \rightarrow y$	Inverter: $y \leq \bar{x}$	$x \rightarrow \neg y$
UND: $y \leq x_0 \wedge x_1 \wedge \dots$	$x_0 \vdots \begin{matrix} \boxed{\&} \\ \vdots \\ x_{n-1} \end{matrix} y$	NAND: $y \leq \overline{x_0 \wedge x_1 \wedge \dots}$	$x_0 \vdots \begin{matrix} \boxed{\&} \\ \vdots \\ x_{n-1} \end{matrix} \neg y$
ODER: $y \leq x_0 \vee x_1 \vee \dots$	$x_0 \vdots \begin{matrix} \boxed{\geq 1} \\ \vdots \\ x_{n-1} \end{matrix} y$	NOR: $y \leq \overline{x_0 \vee x_1 \vee \dots}$	$x_0 \vdots \begin{matrix} \boxed{\geq 1} \\ \vdots \\ x_{n-1} \end{matrix} \neg y$
XOR: $y \leq x_0 \oplus x_1 \oplus \dots$	$x_0 \vdots \begin{matrix} \boxed{=1} \\ \vdots \\ x_{n-1} \end{matrix} y$	XNOR: $y \leq \overline{x_0 \oplus x_1 \oplus \dots}$	$x_0 \vdots \begin{matrix} \boxed{==} \\ \vdots \\ x_{n-1} \end{matrix} y$
Multiplexer: wenn $s = 0$ dann $y \leq x_0$ sonst $y \leq x_1$	$x_0 \vdots \begin{matrix} 0 \\ 1 \end{matrix} y$ x_1 s	allg. Funktion: $y \leq f(\mathbf{x})$	$x_0 \vdots \begin{matrix} f() \\ \vdots \\ x_{n-1} \end{matrix} y_0 \dots y_{m-1}$

Abb. 1.16. Symbole für die logischen Grundgatter ($\leq$ – Signalzuweisung)

Signalflusspläne auf der Funktionsblockebene

Auf den Hierarchieebenen oberhalb der Gatterebene sind die Verarbeitungsbauusteine Teilschaltungen mit komplexeren Funktionen, die zum Teil auch über Bitvektoren, die ihrerseits wieder Zahlen, Aufzählungstypen und aus diesen zusammengesetzte Typen repräsentieren können, verbunden. Wegen der großen Vielfalt möglicher Funktionen haben die Teilschaltungen auf diesen Beschreibungsebenen meist das allgemeine Symbol und einen Verweis auf eine Funktionsbeschreibung. Es gibt aber auch komplexe Standardschaltungen mit speziellen Symbolen, z.B. Register, Rechenwerke und Blockspeicher. Signale mit Typen, die mehr als zwei Werte darstellen können, – Bitvektortypen, Zahlentypen etc. – werden als dicke Linien gezeichnet. In einer formalen Darstellung ist an jede dieser Linien der Datentyp zu schreiben. Bei Bitvektoren wird oft nur die Bitanzahl angegeben. Einmündungen und Abzweigungen von Einzelbits und Teilvektoren werden durch schräge Anschlusslinien symbolisiert.

Abbildung 1.17 zeigt eine Beispielschaltung. Der Signalvektor $\mathbf{x}$ wird hier von den beiden linken, nur halb dargestellten Teilschaltungen geliefert. Ein Teil der Signale (x_2 bis x_6) wird von einer Funktion $f(\mathbf{x})$ auf einen Signalvektor $\mathbf{y}$ abgebildet. Von dem Vektor $\mathbf{y}$ reicht der Multiplexer bei $x_7 = 0$ den unteren und sonst den oberen Teilvektor weiter.

Signalflusspläne sind oft nur informale Modelle, die bestimmte Aspekte und Strukturmerkmale veranschaulichen. Informal bedeutet, dass die Schaltungen nicht vollständig oder nicht eindeutig beschrieben sind. Es können z.B. Signale fehlen, Typangaben weggelassen sein oder die Signalauswahl bei Verzweigungen und Einmündungen ist nicht eindeutig. Das ist der Tribut an die Anschaulichkeit. Für formale, simulierbare und synthetisierbare Beschreibungen

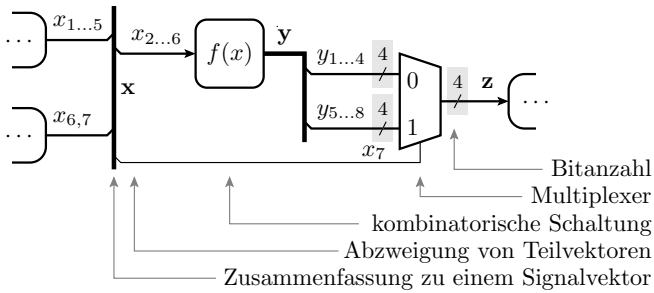


Abb. 1.17. Signalflussplan auf der Funktionsblockebene

gen ist ein Programm in einer Hardware-Beschreibungssprache besser geeignet.

1.2.3 Imperative Funktionsmodelle

In einem imperativen Funktionsmodell wird die Abbildung der Signalverläufe an den Eingängen auf die Signalverläufe an den Ausgängen einer Schaltung durch nacheinander abzuarbeitende Anweisungen beschrieben. Die wichtigsten Beschreibungselemente sind Zuweisungen, Ausdrücke und Fallunterscheidungen. In den späteren Kapiteln kommen noch Schleifen und Unterprogramme hinzu.

Zum Erlernen einer Hardware-Beschreibungssprache ist es zweckmäßig, die behandelten Beschreibungskonstrukte mit einem Simulator auszuprobieren. Abbildung 1.18 zeigt einen Testrahmen für den Test der Beschreibungselemente, die in diesem Abschnitt behandelt werden. Im Vorspann werden die Bibliotheksbezeichner definiert und Vereinbarungen aus Packages importiert. Bis auf Weiteres genügen zwei Packages. Das standardisierte Package IEEE.STD_LOGIC_1164 ist in jeder VHDL-Entwicklungsumgebung verfügbar und definiert die Typen STD_LOGIC und STD_LOGIC_VECTOR sowie die Operatoren und Funktionen für diese beiden Typen. Das Package »Tuc.Ausgabe« stellt eine Minimalmenge von Funktionen für die Textausgabe der Simulationsergebnisse bereit. Um es nutzen zu können, muss zuerst die Bibliothek »Tuc« erzeugt werden. Die Anleitung dazu und auch alle behandelten Beispiele stehen im Web [27]. Der Testrahmen selbst ist eine Entwurfseinheit ohne Anschlussignale mit einem Prozess, der die zu testenden Anweisungen enthält. Prozesse werden in eine Endlosschleife übersetzt. Damit die zu testenden Anweisungen nur einmal ausgeführt werden, muss am Ende der Anweisungsfolge eine Warteanweisung ohne Weckbedingung stehen (vgl. Abschnitt 1.1.4). In den nachfolgenden VHDL-Beispielen werden, damit diese kurz und übersichtlich bleiben, nur die Vereinbarungen und die Anweisungen aufgelistet. Diese sind getrennt nach Vereinbarungen für die Entwurfseinheit, Vereinbarungen für den Testprozess, Anweisungen für den Testprozess und

weiteren nebenläufigen Beschreibungselementen der Entwurfseinheit an den gekennzeichneten Stellen in Abb. 1.18 einzufügen. Die vollständigen Simulationsbeschreibungen stehen im Web. Der Dateiname steht jeweils im Kasten rechts unter der Beschreibung. Zusätzlich sind auf der Web-Seite Hilfetexte zur Ausführung der Simulationen und zur Visualisierung der berechneten Signalverläufe zu finden.

<code>-- Vorspann</code>	<code>Testprozess: process</code>
<code>library IEEE;</code>	<code>-- Vereinbarungen im Testprozess</code>
<code>use IEEE.STD_LOGIC_1164.all;</code>	<code>...</code>
<code>library Tuc;</code>	<code>begin</code>
<code>use Tuc.Ausgabe.all;</code>	<code>-- Anweisungsfolge im Testprozess</code>
<code>entity test is end entity;</code>	<code>...</code>
<code>architecture TR of test is</code>	<code>wait;</code>
<code>-- Vereinbarungen der</code>	<code>end process;</code>
<code>-- Entwurfseinheit</code>	<code>-- weitere nebenläufige</code>
<code>...</code>	<code>-- Beschreibungselemente</code>
<code>begin</code>	<code>...</code>
	<code>end architecture;</code>

Abb. 1.18. Testrahmen für VHDL-Beschreibungselemente

Definition von Datenobjekten und Zuweisungen

Die Definition von Datenobjekten erfolgt in VHDL für alle Objekttypen nach demselben Schema:

```
constant Bezeichner{Bezeichner}: Typ [:=Anfangswert];
variable Bezeichner{Bezeichner}: Typ [:=Anfangswert];
signal Bezeichner{Bezeichner}: Typ [:=Anfangswert];
```

Hinten dem Schlüsselwort für den Objekttyp werden ein oder mehrere neue Bezeichner eingeführt. Diesen wird ein Datentyp und optional ein Anfangswert zugeordnet. Signale beschreiben die Kommunikation zwischen Prozessen und können nur als Schnittstellensignale oder interne Signale einer Entwurfseinheit vereinbart werden. Variablen als imperative Beschreibungsmittel lassen sich nur in imperativen Beschreibungsumgebungen, d.h. in Prozessen und Unterprogrammen, vereinbaren.

Der Anfangswert ist der Wert, mit dem ein Datenobjekt zu Beginn der Simulation initialisiert wird. Ohne explizite Angabe legt der Datentyp den Anfangswert fest. Bei Aufzählungstypen wird das Datenobjekt ohne explizite Angabe des Anfangswertes mit dem ersten Wert der Aufzählung und bei Zahlentypen mit dem Wert der linken Bereichsgrenze initialisiert (siehe später

Abschnitt 3.2). Die nachfolgende Tabelle zeigt die impliziten Anfangswerte der eingeführten elementaren Datentypen aus Abschnitt 1.2.1.

BIT	STD_LOGIC	BOOLEAN	CHARACTER	INTEGER	NATURAL	POSITIVE
'0'	'U'	FALSE	NUL	INTEGER'LOW	0	1

('LOW – Attribut, das den kleinsten darstellbaren Wert eines Zahlentyps zurückgibt).

Bei der Vereinbarung von Bitvektoren ist der Indexbereich anzugeben. Die Vereinbarung kann aufsteigend oder abfallend erfolgen, z.B.:

```
signal s: STD_LOGIC_VECTOR(0 to 3) := "1100";
signal f: STD_LOGIC_VECTOR(3 downto 0) := "1100";
```

Für STD_LOGIC_VECTOR ist der Indexbereich auf den Wertebereich von NATURAL beschränkt. Bei aufsteigender Vereinbarung wird bei der Zuweisung einer Bitvektorkonstante der erste Bitwert der Konstanten an das Vektorelement mit dem kleinsten und der letzte Bitwert an das Vektorelement mit dem größten Index zugewiesen. Bei einer absteigenden Vereinbarung ist die Zuordnung umgekehrt, der erste Bitwert wird wie bei der Zahlendarstellung dem Vektorelement mit dem größten Index zugeordnet. Die beiden Signalvereinbarungen oben weisen den einzelnen Vektorelementen zueinander gespiegelte Anfangswerte zu:

Vektorelement	s(0)	s(1)	s(2)	s(3)	f(3)	f(2)	f(1)	f(0)
Anfangswert	1	1	0	0	1	1	0	0

Ohne die Angabe des Anfangswertes werden die Vektorelemente mit den typspezifischen Anfangswerten initialisiert. Für Elemente vom Typ BIT ist das »0« und für Elemente vom Typ STD_LOGIC »U«.

Textausgabe

Bereits die Kontrolle der Anfangswerte, mit denen die Datenobjekte zum Simulationsbeginn initialisiert werden, erfordert eine Simulation mit Textausgaben. Der vordefinierte Datentyp für Einzelzeichen ist CHARACTER. Das ist ein Aufzählungstyp. Der Wertebereich ist der Latin-1-Zeichensatz, der wiederum den ASCII-Zeichensatz als Teilmenge enthält (siehe Anhang A.1.4, Abb. A.1). Die großen deutschen Umlaute »Ä«, »Ö« und »Ü« und das Zeichen für »ß« fehlen und sind selbst in Kommentaren verboten. Die kleinen Umlaute »ä«, »ö« und »ü« sind erlaubt. Alle druckbaren Zeichen sind in einfache Hochkommas eingerahmt. Die nicht druckbaren Zeichen, z.B. »LF« für den Zeilenvorschub, sind symbolische Konstanten. Zeichenketten sind eindimensionale Felder von Zeichen:

```
type STRING is array (POSITIVE range <>) of CHARACTER;
```

Der Indexbereich von Zeichenketten ist auf den Zahlenbereich POSITIVE beschränkt. Der kleinste zulässige Indexwert ist »1«. Zeichenkettenkonstanten werden in doppelte Hochkommas eingerahmt, z.B. "Text". Bis auf Weiteres genügen folgende Textverarbeitungsfunktionen:

- Konkatenation (Verketten) von Zeichen und Zeichenketten mit dem Operator »&«, z.B.
`"Hallo" & ' ' & "Welt" ↦ "Hallo Welt"`
- Umwandlung der Werte von Datenobjekten in eine Textdarstellung mit der im Package »Tuc.Ausgabe« überladenen Funktion `str(Datenobjekt)` und
- Schreiben eines Textes auf die Standardausgabe mit `write(Zeichenkette)`.

Im folgenden Beispiel werden eine Variable vom Typ INTEGER und eine Variable von Typ POSITIVE mit impliziten Anfangswerten vereinbart. Im Anweisungsteil werden beide Anfangswerte mit einem vorangestellten Text auf dem Bildschirm des Simulationsrechners ausgegeben:

```
-- Vereinbarungen im Testprozess
variable a: INTEGER;
variable b: POSITIVE;
-- Anweisungsfolge im Testprozess
write("Wert von a: " & str(a));
write("Wert von b: " & str(b));
```

⇒ WEB-Projekt: P1.2/Test_Anfangswerte.vhdl

Zur Simulation werden die beiden Variablenvereinbarungen und die beiden Anweisungen in den Testrahmen in Abb. 1.18 an den gekennzeichneten Stellen eingefügt, das VHDL-Programm analysiert, zusammengebaut und ausgeführt. Für die Variable *a* wird der kleinste mit INTEGER darstellbare Wert und für die Variable *b* der kleinste mit POSITIVE darstellbare Wert ausgegeben:

```
> Wert von a: -2147483648
> Wert von b: 1
```

Der Testrahmen mit den beiden Vereinbarungen und den beiden Anweisungen steht in der Datei »Test_Anfangswerte.vhdl« im Unterverzeichnis »P1.2« auf der Web-Seite [27].

Zum Erlernen von VHDL ist es zu empfehlen, mit den kleinen vorgegebenen Beispielen zu beginnen, diese zu testen, abzuwandeln und wieder zu testen. Das ist der einfachste Weg zur Einarbeitung in eine neue Beschreibungssprache und in eine neue Entwicklungsumgebung.

Variablenzuweisungen und Ausdrücke

Variablen sind imperative Beschreibungsmittel. Sie können nur innerhalb von Prozessen und Unterprogrammen vereinbart und genutzt werden und dienen dort zur Aufbewahrung von Zwischenergebnissen. Die Wertzuweisung erfolgt genau wie die Zuweisung der Anfangswerte bei der Deklaration mit dem Zuweisungsoperator »:=«:

Variablenname := *Ausdruck*;

Links vom Zuweisungsoperator steht der Bezeichner der Variablen und rechts ein Ausdruck zur Berechnung des zuzuweisenden Wertes. Der Wert eines Ausdrucks kann eine Konstante, der Wert einer Variablen, der Wert eines Signals, das Ergebnis einer Operation oder das Ergebnis einer Funktion sein. Die Definition ist rekursiv. Die Operanden und Aufrufparameter der Funktionen sind wiederum Ausdrücke, d.h., es können entweder Konstanten, Variablen, Signale, Operationen oder Funktionen sein (Abb. 1.19 a). In der digitalen Schaltungstechnik spielen vor allem die logischen Ausdrücke mit dem Wertebereich {0,1} und den Operatoren Invertierung, UND, ODER und exklusives ODER (XOR, modulo-2-Addition) eine Rolle. Abbildung 1.19 b zeigt die Wertetabellen und die Schlüsselworte der von VHDL unterstützten logischen Operatoren.

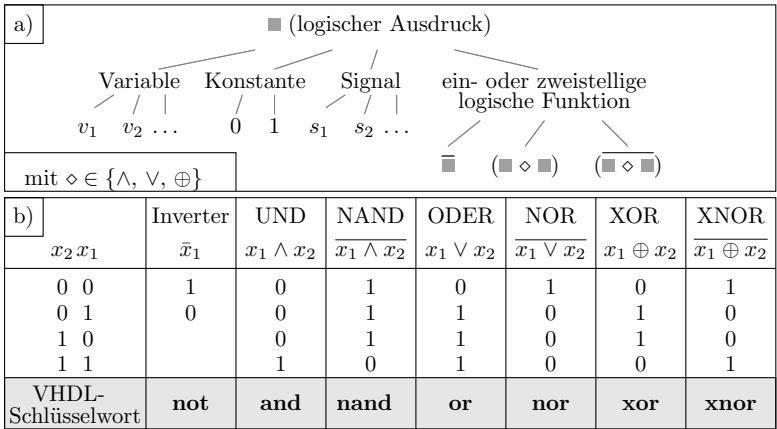


Abb. 1.19. Logische Ausdrücke a) rekursive Definition b) Wertetabellen und VHDL-Schlüsselworte der von VHDL unterstützten logischen Operatoren

Für STD_LOGIC sind die Operatoren für den Teilwertebereich {0,1} in derselben Weise überladen. In Verhaltensmodellen, die auch den Pseudo-Wert »X« (ungültig) nutzen, wird der Wert »X« wie ein Signalwert behandelt, der entweder »0« oder »1« sein kann. Eine logische Verknüpfung eines ungültigen Wertes mit einem anderen Wert ergibt in der Regel wieder einen ungültigen Wert. Ausgenommen davon sind die UND-Verknüpfung mit »0«, die immer »0« ergibt, und die ODER-Verknüpfung mit »1«, die immer »1« ergibt:

$\overline{\overline{X}} = X$

$X \wedge 0 = 0$

$X \vee 0 = X$

$X \oplus 0 = X$

$X \wedge 1 = X$

$X \vee 1 = 1$

$X \oplus 1 = X$

$X \wedge X = X$

$X \vee X = X$

$X \oplus X = X$

(1.3)

Ein Ausdruck ist eine Textbeschreibung für einen baumartigen Berechnungsfluss. Beginnend mit den Operationen in den innersten Klammern werden diese ausgeführt und die Ergebnisse an die Operationen in den übergeordneten Klammern weitergegeben. Abbildung 1.20 a zeigt das am Beispiel für den Ausdruck

$$(\bar{x}_1 \wedge x_2) \vee (\bar{x}_3 \wedge (1 \oplus x_4)) \quad (1.4)$$

Bei der Nachbildung durch eine Schaltung wird der Berechnungsfluss durch einen Signalfluss nachgebildet. Aus den Kanten des Berechnungsbaums werden Signale und aus den Operatoren signalverarbeitende Bausteine. Die Graphenstruktur bleibt komplett erhalten (Abb. 1.20 b).

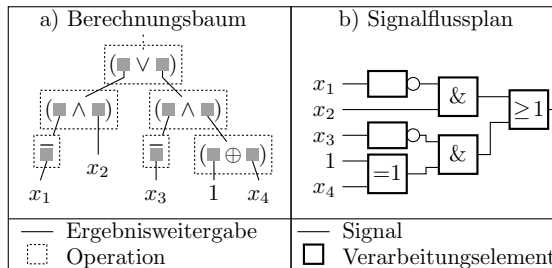


Abb. 1.20. Berechnungs- und Signalfluss für den Ausdruck aus Gleichung 1.4

Kombinatorische Funktionen mit mehreren Ausgängen verlangen eine Beschreibung mit mehreren Ausdrücken. Bei der Simulation werden die Teilergebnisse der Berechnungsbäume der einzelnen Ausdrücke in Variablen zwischengespeichert. Im Signalflussplan werden aus den Berechnungsbäumen strukturgleiche Datenflussgraphen, die ihre Zwischenergebnisse in Signalen weitergeben.

Ausdrücke als Grundlage der Logikoptimierung

Jeder Ausdruck ist in einen Berechnungsbaum und einen strukturgleichen Signalflussplan umrechenbar und umgekehrt. Dieser Zusammenhang spielt eine Schlüsselrolle bei der Spezifikation, Simulation und Optimierung kombinatorischer Schaltungen. Simuliert wird der Berechnungsfluss. Die Optimierung erfolgt zu einem großen Teil auf der Ebene der logischen Ausdrücke mit Hilfe der Regeln der Schaltalgebra. Das Entwurfsergebnis ist der strukturgleiche Signalflussplan.

Zwei anschauliche Optimierungstechniken sind die Konstantenelimination und die Verschmelzung. Die Konstantenelimination ersetzt zweistellige Logikoperationen mit konstanten Operanden durch das Operationsergebnis – eine Konstante oder eine Operation mit nur einem Operanden (Abb. 1.21 a). Die Schaltung links in Abb. 1.21 b entspricht dem Ausdruck

$$(((x_1 \wedge 0) \wedge x_2) \vee x_3) \oplus 1) \quad (1.5)$$

Nach einer schrittweisen Anwendung der Regeln aus Abb. 1.21 a bleibt am Ende nur noch ein Ausdruck mit einer invertierten Eingabegröße übrig:

$$\dots = (((0 \wedge x_2) \vee x_3) \oplus 1) = ((0 \vee x_3) \oplus 1) = (x_3 \oplus 1) = \bar{x}_3$$

Die korrespondierende minimierte Schaltung ist ein Inverter.

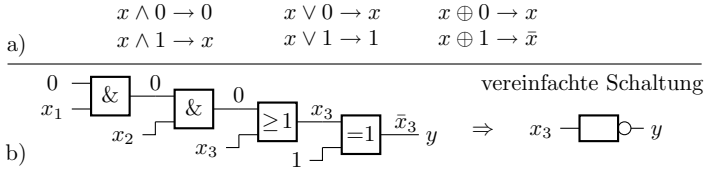


Abb. 1.21. Konstantenelimination a) Vereinfachungsregeln b) Vereinfachung der Schaltung für den Ausdruck in Gleichung 1.5

Die Verschmelzung fasst gleiche Teilberechnungen zusammen. Die beiden Ausdrücke

$$y_1 := x_3 x_2 x_1 \vee (x_4 \oplus x_2 x_1)$$

$$y_2 := x_5 x_4 \vee x_2 x_1$$

enthalten zusammen dreimal den Teilausdruck $x_2 x_1$. Zur Verschmelzung wird der Wert des gemeinsamen Teilausdrucks vorab berechnet und einer Variablen zugewiesen, die den Teilausdruck in den nachfolgenden Ausdrücken ersetzt:

$$z := x_2 x_1$$

$$y_1 := x_3 z \vee (x_4 \oplus z)$$

$$y_2 := x_5 x_4 \vee z$$

Jede eingesparte Operation im Berechnungsfluss bildet sich auch im Signalfluss auf eine eingesparte Operation ab. Die dreifache Verwendung des Wertes der Variablen z entspricht im Datenfluss einem Signal z mit drei Empfängern (Abb. 1.22). Durch die Verschmelzung werden in der Beispielschaltung die beiden grau unterlegten Gatter G1a und G1b eingespart.

Klammernsetzung in Ausdrücken

Ein Ausdruck setzt eine Berechnung aus ein- und zweistelligen Operationen zusammen. Die Abarbeitungsreihenfolge wird im einfachsten Fall mit Klammern festgelegt. In der Mathematik gilt, dass die Negation, falls keine Klammern gesetzt sind, zuerst ausgeführt wird und dass UND Vorrang vor ODER

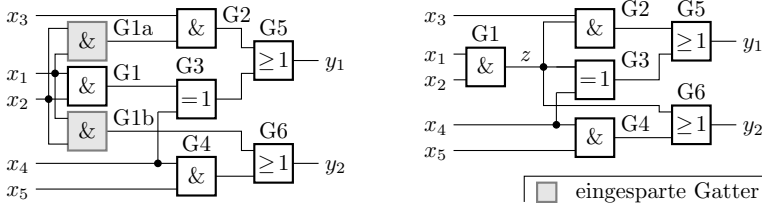


Abb. 1.22. Verschmelzung

hat. Im Ausdruck in Gleichung 1.6 links wird zuerst die Invertierung, dann werden die UND-Verknüpfungen und abschließend die ODER-Verknüpfung ausgeführt. Die Ausführungsreihenfolge der beiden UND-Verknüpfungen ist vertauschbar. Im Ausdruck rechts ist dieselbe Ausführungsreihenfolge mit Klammern festgelegt:

$$x_1 \vee \bar{x}_2 x_3 x_4 = (x_1 \vee (((\bar{x}_2) \wedge x_3) \wedge x_4)) \quad (1.6)$$

In VHDL unterscheiden sich die Vorrangregelungen geringfügig von denen in der Mathematik. Die Invertierung wird auch hier, falls keine Klammern gesetzt sind, zuerst ausgeführt. Die zweistelligen logischen Operatoren sind jedoch alle gleichberechtigt. Wenn die Ausführungsreihenfolge weder mit Klammern festgelegt noch vertauschbar ist, gibt es eine Fehlermeldung. Der ungeklammerte Ausdruck in Anweisung A1 des nachfolgenden Programmausschnitts ist z.B. nicht zulässig, weil nicht festgelegt ist, ob die ODER-Verknüpfung Vorrang hat oder ob die UND-Verknüpfungen Vorrang haben. In Anweisung A2 ist die unzulässige Mehrdeutigkeit mit einer Klammer um die beiden UND-Verknüpfungen beseitigt:

```
variable x: STD_LOGIC_VECTOR(4 downto 1);
...
A1: ... x(1) or not x(2) and x(3) and x(4); -- Analysefehler
A2: ... x(1) or (not x(2) and x(3) and x(4)); -- zulässig
```

Signalzuweisungen

Ein Signal beschreibt einen zeitlichen Werteverlauf und dient als Beschreibungsmittel für die Kommunikation zwischen Prozessen und Teilschaltungen. Ein digitales Signal wird durch einen Anfangswert und eine zeitlich geordnete Folge von Wert-Zeit-Tupeln beschrieben⁸. Der Anfangswert gilt bis zum Zeitwert des ersten Tupels und der Wert eines jeden Tupels bis zum Zeitwert des Folgetupels. Eine Signalzuweisung weist dem Signal jeweils ein oder mehrere

⁸ Diese Darstellung ist für Signale, deren Werte sich nur zu diskreten Zeitpunkten ändern und sonst konstant bleiben, wesentlich kompakter als Abtastfolgen, wie sie in der digitalen Signalverarbeitung und in der digitalen Mess-, Steuer- und Regelungstechnik verwendet werden.

Wert-Zeit-Tupel zu. Jedes Tupel besteht aus einem Ausdruck W zur Berechnung des neuen Signalwertes und einem Ausdruck t_d für die Verzögerungszeit:

Signalname <= [VM] W [after t_d]{, W after t_d };

(VM – Verzögerungsmodell). Der Zeitwert, ab dem ein neu zugewiesener Signalwert gilt, ist jeweils die Summe aus der aktuellen Simulationszeit t_{sim} und der Verzögerungszeit t_d . Verzögerungszeiten haben den Untertyp DELAY_LENGTH für nicht negative Zeitangaben (vgl. Abschnitt 1.2.1). Bei der Simulation läuft eine Art Uhr für die Simulationszeit mit, die nur erhöht, aber nicht zurückgestellt werden kann. Die berechneten Signalverläufe vom Simulationsbeginn bis zur aktuellen Simulationszeit können nicht mehr verändert werden. Zugewiesene Tupel, deren Zeitwerte noch nicht erreicht sind, werden als schwebend bezeichnet. Der Signalverlauf, den sie beschreiben, ist noch nicht endgültig. Eine neue Signalzuweisung löscht alle schwebenden Zuweisungen, deren Zeitwerte größer als die der neu zugewiesenen Tupel sind. Den Umgang mit schwebenden Zuweisungen mit kleineren Zeitwerten legt das Verzögerungsmodell fest. Vorerst werden die Beispiele so gewählt, dass früher zugewiesene Wert-Zeit-Tupel zu den Ausführungszeitpunkten der nachfolgenden Signalzuweisungen nicht mehr schwebend sind. Damit spielt das Verzögerungsmodell vorerst keine Rolle.

Die meisten Signalzuweisungen weisen nur ein Wert-Zeit-Tupel nach dem Standardverzögerungsmodell zu:

Signalname <= W [after t_d];

Ohne die optionale After-Klausel ist die Verzögerungszeit null. Der Zeitwert, ab dem der neue Signalwert gilt, ist die aktuelle Simulationszeit. In Abb. 1.23 wird einem Signal y mit dem Anfangswert »0« innerhalb eines Prozesses eine Änderungsfolge »1-0-1« zugewiesen. Zum Ausprobieren sind die Signalvereinbarungen in den Vereinbarungsteil der Entwurfseinheit und die Signalzuweisungen in den Anweisungsteil des Testprozesses im Testrahmen in Abb. 1.18 einzufügen. Der komplette Testrahmen steht auf der Web-Seite [27] in der angegebenen Datei.

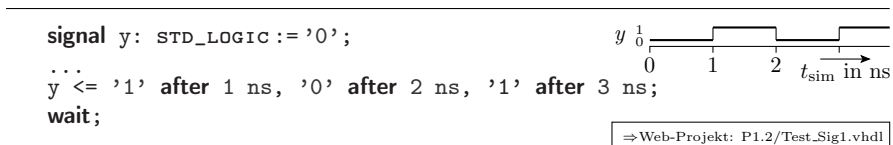


Abb. 1.23. Zuweisung einer Änderungsfolge »1-0-1«

Bei der Ausführung der Simulation ist zusätzlich im Kommandozeilenaufzuruf der Dateiname für die Abspeicherung der berechneten Signalverläufe anzugeben:

```
ghdl -r Test_Sig1 --wave=Test_Sig1.ghw
```

Das Visualisierungsprogramm GTKWAVE wird anschließend mit dieser Datei und einer Sav-Datei mit den Visualisierungseinstellungen gestartet:

```
gtkwave Test_Sig1.ghw [Test_Sig1.sav]
```

Visualisierungseinstellungen sind z.B. die Auswahl der darzustellenden Signale und das darzustellende Zeitfenster. Auf der Web-Seite [27] steht für jedes Projekt, in dem Signalverläufe ausgewertet werden, eine geeignete Sav-Datei.

Die aktuelle Simulationszeit spielt für die Berechnung der Signalverläufe eine Schlüsselrolle. Sie kann innerhalb eines Prozesses mit der vordefinierten Funktion NOW abgefragt und mit Warteanweisungen, die eine Weckbedingung haben, weitergestellt werden. In VHDL wird zwischen folgenden Weckbedingungen unterschieden:

- Änderung eines Signals aus einer Weckliste
`wait on Weckliste;`
- Weiterrücken der Simulationszeit um eine Verzögerungszeit
`wait for Verzögerungszeit;`
- eine beliebige Bedingung, die Signalwerte oder die Simulationszeit auswertet
`wait until Bedingung;`

Eine Warteanweisung legt den Prozess, in dem sie ausgeführt wird, schlafen. Während er schläft, werden andere Prozesse weiterbearbeitet, die Simulationszeit weitergestellt und die Signalwerte aktualisiert. Auch bei einer Zuweisung mit der Verzögerungszeit »0« (Delta-Delay-Modell) ist der neue Signalwert im Prozess erst nach der nächsten Warteanweisung sichtbar.

Abbildung 1.24 zeigt eine typische Prozessbeschreibung für die Erzeugung eines Testeingabesignals für eine kombinatorische Schaltung. Das zu erzeugende Signal ist im Beispiel ein 5-Bit-Vektor mit dem bitweisen Anfangswert »nicht initialisiert«. Im Prozess wird für jeden Testschritt eine bestimmte Zeit gewartet und danach ein neuer Wert zugewiesen. Am Ende der Anweisungsfolge steht die obligatorische Warteanweisung ohne Weckbedingung, die den

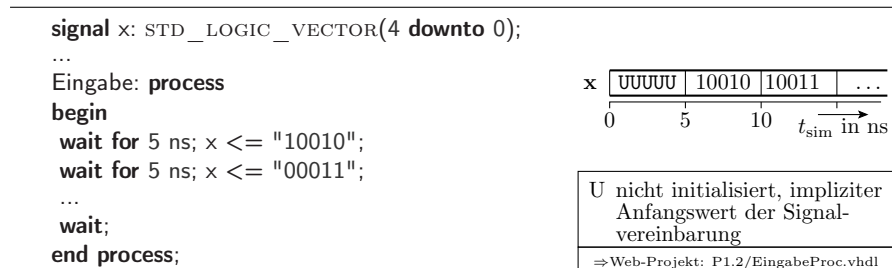


Abb. 1.24. Prozessbeschreibung zur Erzeugung eines Testeingabesignals

Prozess dauerhaft schlafen legt. Ohne diese würde der Prozess die Simulationszeit immer Weiterschalten und dem Signal einen periodischen Verlauf zuweisen. Die Simulation müsste von außen beendet werden, unter Linux z.B. mit »Ctrl-C«.

Abbildung 1.25 a zeigt ein Beispielprogramm zur Signalerzeugung, das auch die Wait-On-Anweisung nutzt. Das erzeugte Signal n ist vom Typ NATURAL und wird mit »0« initialisiert. Die erste Zuweisung weist ihm zum Simulationszeitpunkt null mit den Verzögerungen 1 ns und 2 ns die Werte »1« und »2« zu. Dann wird nacheinander auf beide Änderungen des Signals (d.h. bis zu den Simulationszeitpunkten 1 ns und 2 ns) gewartet. Daran schließt sich eine Wartezeit von 0,5 ns an. Die zweite Zuweisung erhöht den aktuellen Signalwert zum Simulationszeitpunkt von 2,5 ns, der dann »2« ist, wieder mit einer Verzögerung von 1 ns und 2 ns, d.h. zu den Zeitpunkten 3,5 ns und 4,5 ns, auf »4« und »6« und wartet dann wieder auf beide Signaländerungen. Um das Beispiel schrittweise testen zu können, enthält es nach jeder Zuweisung und nach jeder Warteinstruktion eine Ausgabeinstruktion. Der Ausgabetext besteht jeweils aus der Anweisungsbezeichnung, der aktuellen Simulationszeit und dem aktuellen Signalwert. Abbildung 1.25 b zeigt den berechneten Signalverlauf und veranschaulicht die Wirkungsweise der einzelnen Anweisungen.

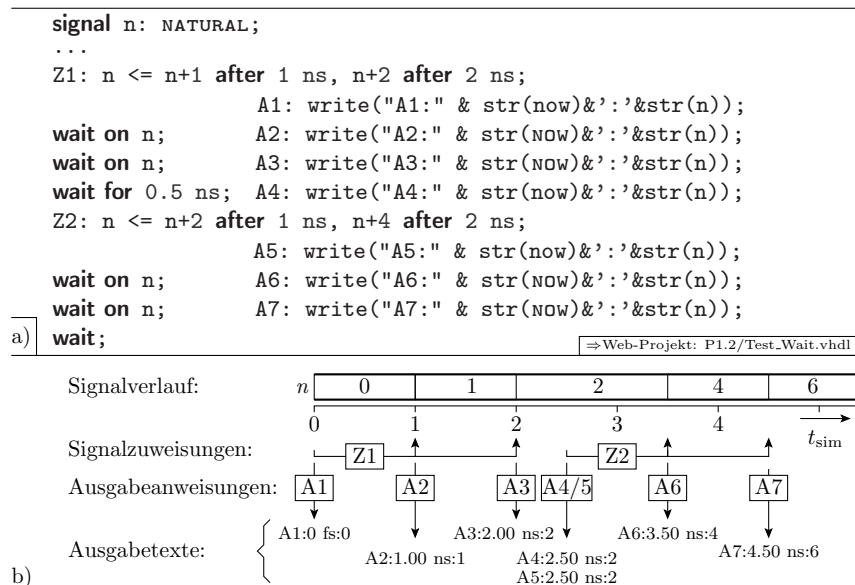


Abb. 1.25. a) Signalerzeugung mit Wait-On-Anweisungen b) erzeugter Signalverlauf und Wirkungsweise der einzelnen Anweisungen

Signale gehören zu den komplizierteren Beschreibungsmitteln digitaler Schaltungen. Es sei deshalb empfohlen, die vorgegebenen Beispiele selbst mit dem Simulator auszuprobieren und zu verändern.

1.2.4 Ereignisgesteuerte Simulation

Die Simulation einer digitalen Schaltung erfolgt ereignisgesteuert. Vorerst werden nur kombinatorische Schaltungen betrachtet. Das einfachste Funktionsmodell einer kombinatorischen Schaltung ist ein Prozess, der die Ausgabewerte aus den aktuellen Eingabewerten berechnet, die berechneten Werte den Ausgabesignalen zuweist, auf die nächste Eingabeänderung wartet und bei jeder Eingabeänderung den gesamten Ablauf wiederholt. Die Ausgabeänderungen sind wiederum Weckereignisse für Prozesse, die die Signale weiterverarbeiten.

Abbildung 1.26 a zeigt eine kombinatorische Schaltung aus fünf Gattern und Abb. 1.26 b eine Funktionsbeschreibung der Schaltung durch einen Prozess. Die Anweisungsfolge innerhalb des Prozesses berechnet zuerst die Ausgabewerte der Gatter G1 bis G3 und speichert sie in Variablen. Aus den zwischengespeicherten Werten werden danach die Ausgabewerte der Schaltung berechnet und mit der Verzögerung $t_d = 2\text{ ns}$ an die beiden Bits des Ausgabesignals zugewiesen. Abschließend wartet der Prozess auf eine Änderung des Signals x , d.h. auf eine Änderung von mindestens einem der Eingabebits. Die Anweisungsfolge in einem Prozess wird zur Simulation in eine Endlosschleife eingebettet, so dass nach dem Aufwachen des Prozesses wieder die gesamte Anweisungsfolge von vorn ausgeführt wird.

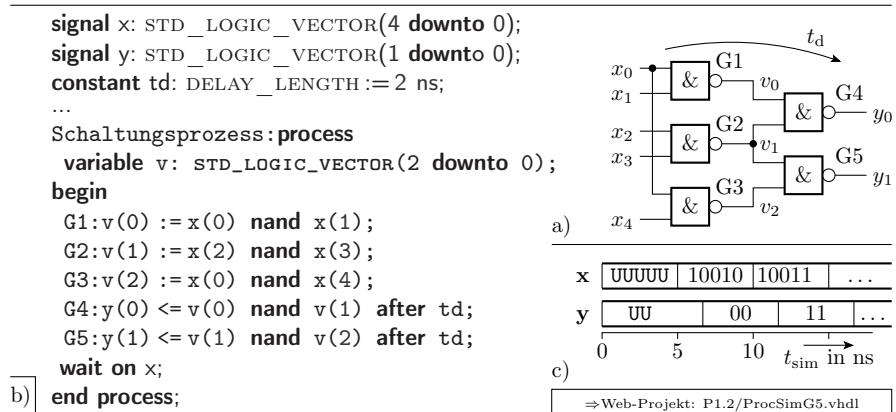


Abb. 1.26. a) Kombinatorische Schaltung b) Prozess zur Funktionsbeschreibung c) Simulationsausgabe (Eingabebereitstellung mit dem Prozess aus Abb. 1.24)

Zur Simulation der Beispielschaltung sind die einzelnen Beschreibungselemente an den richtigen Stellen in den Testrahmen in Abb. 1.18 einzusetzen: die

Signalvereinbarungen in den Vereinbarungsteil und der Prozess in den Anweisungsteil der Entwurfseinheit. Zur Simulation wird ein weiterer Prozess benötigt, der die Eingabesignale bereitstellt. Das ist im zugehörigen Web-Projekt der Prozess aus Abb. 1.24. Er erzeugt ein Eingabesignal x , das zum Simulationsbeginn mit »UUUUU« initialisiert wird, nach 5 ns den Wert »10010« und nach 10 ns den Wert »10011« zugewiesen bekommt. Das berechnete Ausgabe-signal ist zu Beginn »UU« und übernimmt nach jeder Eingabeänderung mit einer Verzögerung von zwei Nanosekunden den Funktionswert für den neuen Eingabevektor, für den ersten »00« und für den zweiten »11«. Abbildung 1.26 c zeigt den berechneten Signalverlauf. Zur genaueren Untersuchung, wie die beiden Prozesse arbeiten, können in die Prozesse Anweisungen zur Textausgabe eingefügt werden. Der experimentierfreudige Leser wird merken, dass es gar nicht so einfach ist, sinnvolle und auswertbare Simulationsausgaben zu erhalten.

Alternativ zu dem Modell in Abb. 1.26 b kann jedes Gatter einer kombinatorischen Schaltung auch durch einen eigenen Prozess beschrieben werden. Abbildung 1.27 a zeigt eine Beispielschaltung aus drei Gattern und Abb. 1.27 b das zugehörige Simulationsmodell. Der erste Prozess stellt die Testeingabesignale bereit. Der zweite und der dritte Prozess beschreiben je ein UND-Gatter und der vierte Prozess ein ODER-Gatter. Die Tabelle in Abb. 1.27 c veranschaulicht den Simulationsablauf. Zum Simulationsbeginn sind alle Signale mit »0« initialisiert. Der Simulator weckt nacheinander alle Prozesse und führt sie bis zur ersten Warteanweisung aus. Die Prozesse der Gatter G1 bis G3 weisen ihren Ausgabesignalen nach 1 ns bzw. 2 ns je den Wert »0« zu. Diese Zuweisungen bewirken jedoch keine Signaländerungen. Der Eingabeprozess legt sich sofort schlafen, wacht nach 1 ns wieder auf und weist für den Aufwachzeitpunkt $t_{\text{sim}} = 1 \text{ ns}$ an x_3 eine Änderung nach »1« zu. Diese Änderung weckt den Prozess G2, der z_2 für den Simulationszeitpunkt 2 ns eine »0« (auch keine Änderung) zuweist. Das nächste Mal wacht der Eingabeprozess zum Simulationszeitpunkt $t_{\text{sim}} = 3 \text{ ns}$ auf und weist den Eingabesignalen x_1 und x_4 eine Änderung nach »1« zu. Das weckt nacheinander die Prozesse G1 und G2. Die Signalzuweisung in G1 an z_1 erzeugt keine Änderung, die in G2 an z_2 erzeugt für $t_{\text{sim}} = 4 \text{ ns}$ eine Änderung nach »1«. Diese Änderung weckt zum Simulationszeitpunkt $t_{\text{sim}} = 4 \text{ ns}$ den Prozess G3. In G3 wird dem Ausgabesignal y für den Simulationszeitpunkt $t_{\text{sim}} = 6 \text{ ns}$ der geänderte Wert »1« zugewiesen. Abbildung 1.27 d zeigt die berechneten Signalverläufe. Die Simulation aller Gatter durch einen eigenen Prozess bildet das Zeitverhalten einer Schaltung wesentlich genauer nach als ein einzelner Prozess. Der Preis dafür ist ein viel größerer Rechenaufwand für die Simulation.

Für wichtige Arten von Prozessbeschreibungen gibt es in VHDL Kurz-schreibweisen. Verhaltensmodelle für Schaltungen werden in der Regel – wie auch in den bisherigen Beispielen – durch einen Prozess mit genau einer Warteanweisung mit Weckliste am Ende der Anweisungsfolge beschrieben. Die Kurzschreibweise hierfür ist ein Prozess mit Weckliste. Bei diesem steht die

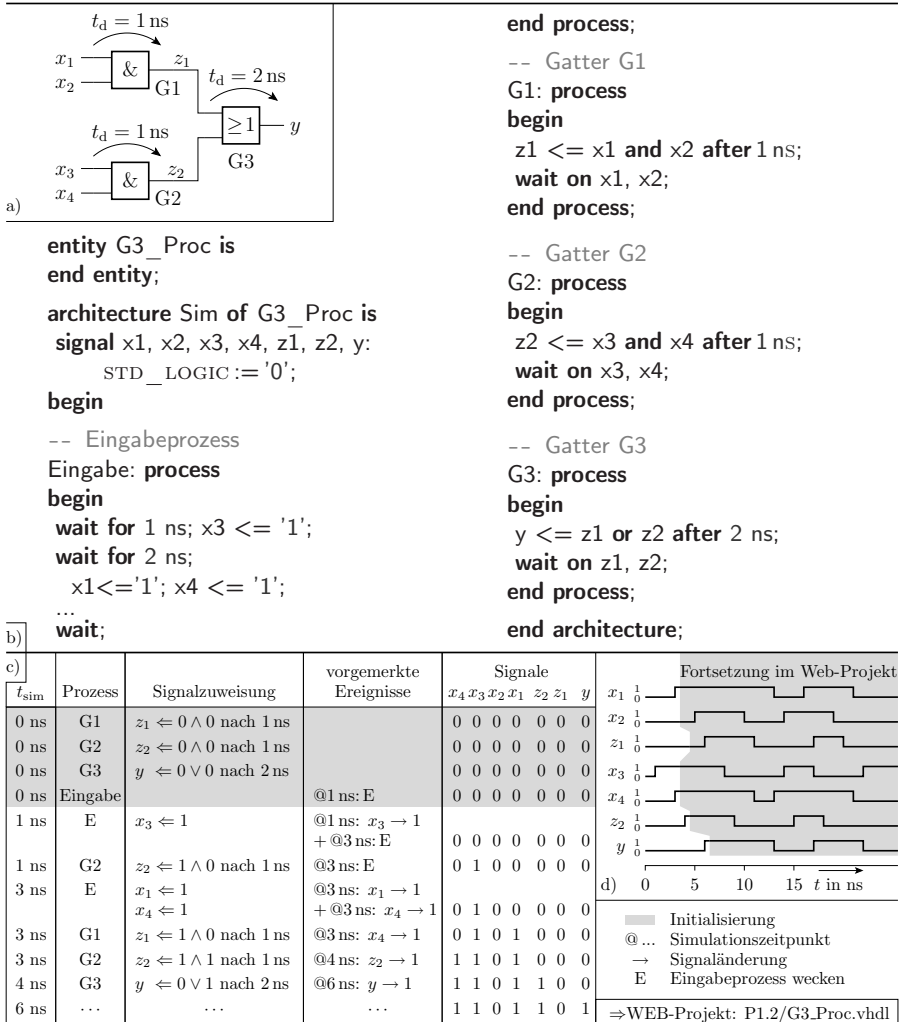


Abb. 1.27. Simulation einer kombinatorischen Schaltung mit einem Prozess je Gatter a) Beispielschaltung b) VHDL-Beschreibung c) Tabelle der Ereignisse und Aktionen während der Simulation d) berechnete Signalverläufe

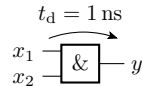
Weckliste hinter dem Schlüsselwort »process« und die Warteanweisung entfällt (Abb. 1.28 b). In einem solchen Prozess sind keine weiteren Warteanweisungen erlaubt. Diese Spezialschreibweise dient vor allem zur Vermeidung des Fehlers »vergessene Warteanweisung«. Ein Prozess ohne Warteanweisung hat die unschöne Eigenschaft, dass er den Kontrollfluss nicht mehr abgibt, so dass die Simulation praktisch stehen bleibt.

-- a) Prozess mit genau einer Warteanweisung mit Weckliste

```

process
begin
  z1 <= x1 and x2 after 1 ns;
  wait on x1, x2;
end process;

```



-- b) funktionsgleiche Beschreibung durch einen Prozess mit Weckliste

```

process (x1, x2)
begin
  z1 <= x1 and x2 after 1 ns;
end process;

```

-- c) funktionsgleiche Beschreibung durch eine nebenläufige Signalzuweisung

```

z1 <= x1 and x2 after 1 ns;

```

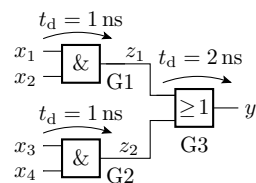
Abb. 1.28. Kurzschreibweisen für kombinatorische Prozesse

Für einen Prozess mit Weckliste gibt es wiederum eine Kurzschreibweise. Wenn der Anweisungsteil nur aus einer Anweisung besteht und die Weckliste alle Eingabesignale enthält, kann der Prozessrahmen weggelassen werden. Die resultierende Kurzschreibweise ist eine nebenläufige Signalzuweisung (Abb. 1.28 c). Abbildung 1.29 zeigt als Beispiel eine verkürzte Beschreibung der drei Gatter aus Abb. 1.27 mit nebenläufigen Signalzuweisungen. Der Eingabeprozess muss in der Langform übernommen werden. Die Signalzuweisungen G1 bis G3 werden zur Simulation wieder durch ihre Prozesse ersetzt, so dass die Beschreibung in 1.29 in dasselbe ausführbare Simulationsprogramm wie die Beschreibung in Abb. 1.27 übersetzt wird.

```

architecture Sim of G3_NLZ is
  signal x1, x2, x3, x4, z1, z2, y: STD_LOGIC := '0';
  begin
    -- nebenläufige Signalzuweisungen
    G1: z1 <= x1 and x2 after 1 ns;
    G2: z2 <= x3 and x4 after 1 ns;
    G3: y <= z1 or z2 after 2 ns;
    -- Eingabeprozess
    Eingabe: process
    -- weiter wie in Abb. 1.27
  end architecture;

```



⇒ WEB-Projekt: P1.2/G3_NLZ.vhdl

Abb. 1.29. Beschreibung der Gatterprozesse aus Abb. 1.27 b durch nebenläufige Signalzuweisungen

1.2.5 Strukturbeschreibung

Eine Strukturbeschreibung beschreibt die Zusammensetzung einer Gesamtschaltung aus Teilschaltungen. Wie bereits in Abschnitt 1.2.2 eingeführt, ist das ein Graph mit den Bauteilen als Knoten und den Signalen als Kanten. Die rechnerinterne Darstellung besteht aus der Vereinbarung aller Bauteile und Signale und einer Verbindungsliste, die entweder den Bauteilanschlüssen Signale oder den Signalen Bauteilanschlüsse zuordnet. In VHDL erfolgt die Zuordnung bauteilorientiert.

Schnittstellenvereinbarung

Jede Entwurfseinheit – unabhängig davon ob sie eine Teilschaltung, eine Gesamtschaltung oder einen Testrahmen beschreibt – hat in VHDL eine Schnittstelle. Die Schnittstellenvereinbarung definiert die Anschluss-signale und für parametrisierte Schaltungen zusätzlich die Parameter. Sie hat die folgende Form:

```
entity Bezeichner_Entwurfseinheit is
  [generic (Liste_der_Konfigurationsparameter);]
  [port (Liste_der_Anschluss-signale);]
end entity;
```

Die Liste der Anschluss-signale legt für alle Anschluss-signale einen Bezeichner, die Signalflossrichtung, den Datentyp und optional einen Standardwert fest:

```
Signalname {,Signalname}: Richtung Typ [:=Standardwert]
{; Signalname {,Signalname}: Richtung Typ [:=Standardwert]}
```

Mögliche Signalflossrichtungen sind:

```
in      -- Eingang
out     -- Ausgang
inout   -- Ein- und Ausgang
buffer  -- Ausgang mit rücklesbarem Wert
```

Die Liste der Konfigurationsparameter hat eine ähnliche Beschreibungsstruktur, nur dass die Parameter lokal statische (zum Übersetzungszeitpunkt bekannte) Konstanten sein müssen und keine Signalflossrichtung haben:

```
Parametername {,Parametername}: Typ [:=Standardwert]
{; Parametername {,Parametername}: Typ [:=Standardwert]}
```

Die Standardwerte haben nur für Eingabesignale und Parameter eine Bedeutung. Das sind die Werte dieser Objekte, wenn ihnen in der übergeordneten Beschreibung kein Signal bzw. kein Parameterwert zugeordnet wird.

Abbildung 1.30 zeigt als Beispiel die Beschreibung eines UND-Gatters. Die Schnittstelle definiert die Verzögerungszeit als Konfigurationsparameter und legt die Typen und Signalflossrichtungen der drei Anschluss-signale fest. Die

Verzögerungszeit des Gatters hat den Typ `DELAY_LENGTH` und den Standardwert 0 ns. Die Anschlussignale *a* und *b* sind Eingabesignale und *c* ist ein Ausgabesignal, alle mit dem Typ `STD_LOGIC`. Die Beschreibung besteht im Beispiel nur aus einer nebenläufigen Signalzuweisung. Wie im Vorabschnitt behandelt, ist das die Kurzschreibweise für einen Prozess mit den beiden Eingabesignalen in der Weckliste und der Signalzuweisung im Anweisungsteil.

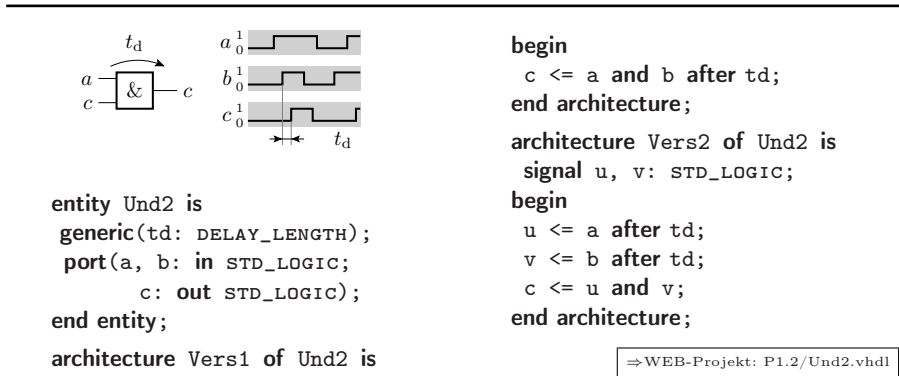


Abb. 1.30. UND-Gatter mit Schnittstelle und zwei Beschreibungen

In VHDL kann eine Entwurfseinheit auch wie in Abb. 1.30 mehrere Beschreibungen haben. In der ersten Beschreibungsversion wird im Beispiel die UND-Verknüpfung der beiden Eingabesignale verzögert an das Ausgabesignal zugewiesen und in der zweiten Beschreibung werden die Eingabesignale zuerst verzögert und dann UND-verknüpft. Beschreibungen derselben Entwurfseinheit werden am Beschreibungsnamen unterschieden. Die zweite und alle weiteren Beschreibungen können in derselben oder in einer anderen Datei stehen. Die Schnittstellenbeschreibung darf auch bei einer Aufteilung auf mehrere Dateien nur einmal enthalten sein. Bei der Analyse muss immer zuerst die Schnittstellenbeschreibung vorübersetzt und in der Bibliothek abgelegt werden, bevor die Beschreibungen analysiert werden können.

Teilschaltungsinstanziierung

Eine Teilschaltungsinstanz besteht aus einem Verweis auf eine Entwurfseinheit, optional einen Verweis auf eine ihrer Beschreibungen und einer Zuordnung von Anschlussignalen und Parameterwerten:

```

Teilschaltungsname: entity Bibliothek.Entwurfseinheit[(Beschreibung)]
  [generic map(Zuordnungsliste_Parameter)]
  port map(Zuordnungsliste_Anschlüsse);

```

Entwurfseinheiten desselben Projekts stehen in der Regel in der Arbeitsbibliothek `WORK`. Die Zuordnungslisten können namensbasiert oder positionsbasiert aufgebaut sein. Eine namensbasierte Zuordnungsliste besteht aus Zuordnungstupeln der Form:

$$BS \Rightarrow BZ\{, BS \Rightarrow BZ\}$$

(*BS* – Bezeichner in der Schnittstellendefinition der Teilschaltung; *BZ* – Bezeichner des zugeordneten Datenobjekts in der Strukturbeschreibung). Eine positionsbasierte Zuordnungsliste ist eine kommaseparierte geordnete Liste der zugeordneten Objekte:

$$BZ\{, BZ\}$$

Diese Objekte müssen in der durch die Schnittstelle vereinbarten Reihenfolge zugeordnet werden. Für beide Arten der Zuordnungsliste müssen die Typen der einander zugeordneten Objekte übereinstimmen und für die Zuordnungsliste der Anschlüsse müssen zusätzlich die Signalfussrichtungen zueinander passen. Passen heißt, dass Signale, außer in wenigen später behandelten Ausnahmen, genau eine Quelle und mindestens einen Empfänger haben. Nicht verwendeten Anschlüssen und Parametern wird entweder nichts oder das Schlüsselwort »open« zugeordnet.

Beispiel sei die Strukturbeschreibung in Abb. 1.31 für die Gatterschaltung aus Abb. 1.27. Die Schnittstellenbeschreibung definiert vier Eingabe- und ein Ausgabesignal, die alle vom Typ `STD_LOGIC` sind. Die internen Signale werden im Vereinbarungsteil der Beschreibung vereinbart. Die Instanziierung der drei enthaltenen Gatter folgt im Anweisungsteil. Die Gatter `G1` und `G2` sind Entwurfseinheiten vom Typ »Und2« mit der Beschreibung »Vers1« aus der

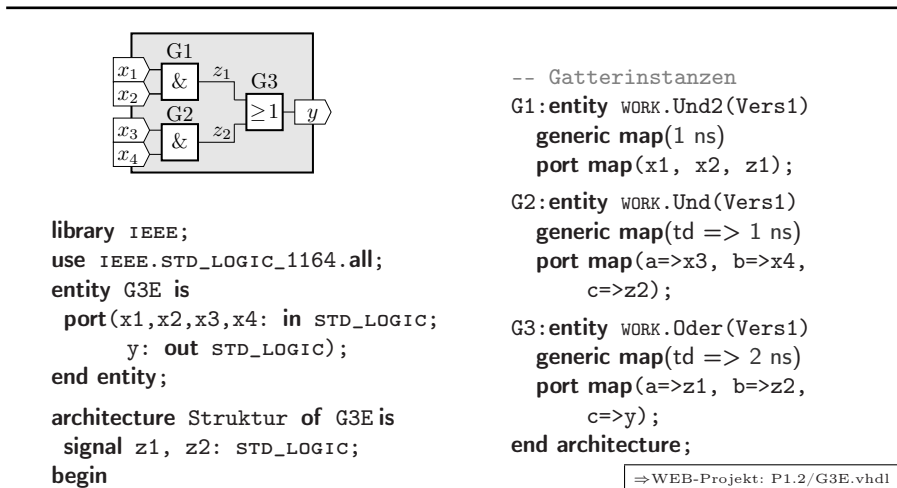


Abb. 1.31. Beispiel einer Strukturbeschreibung

Arbeitsbibliothek WORK. Der Teilschaltungstyp »Oder2« für G3 sei genauso wie der Typ »Und2« in Abb. 1.30 definiert, nur mit »or« statt »and« in der Signalzuweisung. Der Bibliotheksbezeichner WORK ist vordefiniert und verlangt im Gegensatz zu den Bibliotheksbezeichnern IEEE und »Tuc« keine Vereinbarung im Vorspann. Für G1 erfolgt die Zuordnung der Parameter und Anschlussignale positionsbasiert und für die anderen beiden Gatter namensbasiert. Die positionsbasierte Zuordnung ist kürzer, die namensbasierte verständlicher und weniger fehleranfällig.

1.2.6 Testrahmen

Die oberste Hierarchieebene in einem Simulationsmodell ist der Testrahmen. Das ist eine Entwurfseinheit ohne Anschlussignale und Parameter. In der Regel wird das Testobjekt als Schaltungsinstanz, statt wie in Abb. 1.27 als System aus mehreren Prozessen, eingebunden. Das ist übersichtlicher, änderungsfreundlicher und weniger fehleranfällig.

Abbildung 1.32 zeigt ein funktionsgleiches Simulationsmodell zu Abb. 1.27. Das Testobjekt ist eine Bauteilinstanz. Der Prozess für die Eingabebereitstellung ist übernommen. Zur Simulation und auch für andere Bearbeitungsschritte wird die Hierarchie der Strukturbeschreibung aufgelöst. Dazu werden von unten beginnend alle Teilschaltungsinstanzen durch die Prozesse, die ihre Funktion beschreiben, ersetzt. Im Beispiel entsteht praktisch wieder das Modell aus Abb. 1.27.

```
entity Test_G3E is end entity;
```

```
architecture Test of Test_G3E is
```

```
  signal e1, e2, e3, e4, a: STD_LOGIC := '0';
```

```
begin
```

```
  TObj: entity WORK.G3E(Struktur)
```

```
    port map(x1=>e1, x2=>e2, x3=>e3, x4=>e4, y=>a);
```

```
  Eingabe: process
```

```
  begin
```

```
    wait for 1 ns;
```

```
    e3 <= '1';
```

```
    wait for 2 ns;
```

```
    e1 <= '1';
```

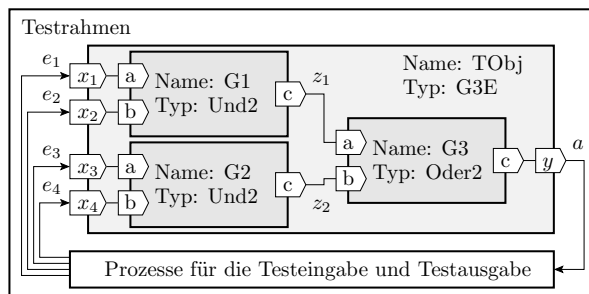
```
    e4 <= '1';
```

```
    ...
```

```
  wait;
```

```
end process;
```

```
end architecture;
```



⇒ WEB-Projekt: P1.2/Test_G3E.vhdl

Abb. 1.32. Testrahmen mit dem Testobjekt als Schaltungsinstanz

1.2.7 Instanziierung von Komponenten

Eine ältere, umständlichere Art der Instanziierung von Teilschaltungen verweist statt auf Bibliotheksobjekte auf Komponenten:

```
Teilschaltungsname: [component] Bezeichner_Komponente
[generic map(Zuordnungsliste_Parameter)]
port map(Zuordnungsliste_Anschlüsse);
```

Eine Komponente ist ein Platzhalter, der syntaktisch bis auf das Schlüsselwort genauso wie die Schnittstelle einer Entwurfseinheit vereinbart wird:

```
component Bezeichner_Komponente is
[generic (Liste_der_Konfigurationsparameter);]
port (Liste_der_Anschlussignale);
end component;
```

Die Vereinbarungen der Komponenten können im Vereinbarungsteil der Entwurfseinheit stehen oder aus einem Package importiert werden. Die Zuordnung der Entwurfseinheit und der Entwurfsbeschreibung zu einer Komponente kann implizit oder explizit erfolgen. Implizit wird die gleichnamige Entwurfseinheit mit derselben Schnittstelle, die zuletzt analysiert wurde, und deren zuletzt analysierte Beschreibung zugeordnet. Die explizite Zuordnung erfordert eine spezielle Konfigurationsbeschreibung, die hier im Buch weder behandelt noch benutzt wird. Andere Bücher und auch das Entwurfssystem ISE von »Xilinx« bevorzugen jedoch diese Art der Instanziierung von Teilschaltungen. Deshalb ist es notwendig, wenigstens zu wissen, dass es sie gibt und dass es außer der zusätzlichen Zuordnungsebene keinen semantischen Unterschied zu der hier verwendeten Art der Instanzbildung gibt.

1.2.8 Zusammenfassung und Übungsaufgaben

Eine digitale Schaltung ist ein System aus signalverarbeitenden Bausteinen, die über Signale miteinander kommunizieren. In einer formalen Beschreibung hat jedes Signal und jedes andere Datenobjekt einen Typ, der den Wertebereich festlegt. Eingeführt wurden die standardisierten VHDL-Typen für Bits, Bitvektoren, Zahlen, die Simulationszeit und für die Textverarbeitung.

Das Funktionsmodell einer Teilschaltung ist ein Prozess. Von den imperativen innerhalb eines Prozesses nutzbaren Beschreibungselementen wurden Variablenzuweisungen, Signalzuweisungen, Ausdrücke und Warteanweisungen behandelt. Ausdrücke beschreiben einen baumartigen Berechnungsfluss, der sich durch einen strukturgleichen Signalfuss nachbilden lässt und umgekehrt. Sie bilden die Basis für viele Schaltungsoptimierungen, von denen die Konstantenelimination und die Verschmelzung behandelt wurden. Die Wirkung jeder Zuweisung lässt sich einzeln im Simulator ausprobieren.

Die zeitlichen Abläufe werden mit Hilfe von Signalzuweisungen und Warteanweisungen nachgebildet. Das Funktionsmodell einer kombinatorischen

Schaltung ist ein Prozess, der die Ausgabewerte aus den aktuellen Eingabewerten berechnet, verzögert zuweist, sich bis zur nächsten Eingabeänderung schlafen legt und nach dem Aufwachen dieselbe Berechnungsfolge wiederholt. In einer Schaltung aus mehreren Teilschaltungen wird jede Teilschaltung durch einen solchen Prozess simuliert. Zusätzlich sind Prozesse zur Bereitstellung der Eingabesignale erforderlich, die im einfachsten Fall aus einer Folge von Signalzuweisungen und Warteanweisungen bestehen.

Zur hierarchischen Strukturierung werden die Prozesse zu Entwurfseinheiten zusammengefasst, mit Schnittstellen versehen und in übergeordnete Entwurfseinheiten als Instanzen eingebunden. Die oberste Ebene in der Beschreibungshierarchie eines Simulationsmodells ist der Testrahmen, eine Entwurfseinheit ohne Anschlüsse. Bei der Schaltungsanalyse vor der Simulation wird die Hierarchie wieder in einzelne über Signale kommunizierende Prozesse – das eigentliche Simulationsmodell – aufgelöst.

Aufgabe 1.5

Gegeben sind die Zeitverläufe der Spannungspotenziale auf den vier Bit-Leitungen eines Signals

signal `b`: `STD_LOGIC_VECTOR(3 downto 0)`;

in Abb. 1.33. Kennzeichnen Sie die Zeitbereiche, in denen das Signal ungültig ist. Ordnen Sie den Zeitfenstern, in denen es gültig ist, die Signalwerte zu.

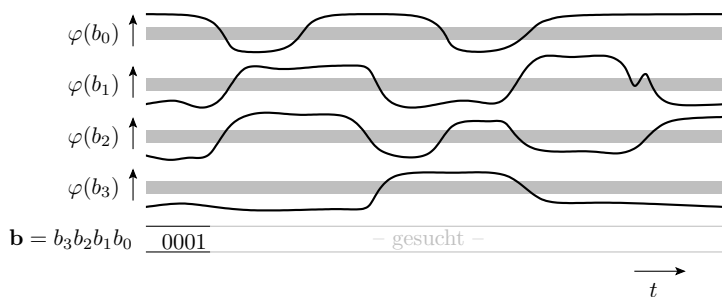


Abb. 1.33. Signalverlauf zu Aufgabe 1.5

Aufgabe 1.6

- Wie viele Werte lassen sich mit einem Byte (8 Bit) unterscheiden?
- Wie groß muss ein Bitvektor sein, damit er 25 Zustände unterscheiden kann?

Aufgabe 1.7

Gegeben sind die nachfolgenden Vereinbarungen und Anweisungen:

```
-- Vereinbarungen der Entwurfseinheit
signal a: INTEGER := 0;

-- Vereinbarungen im Testprozess
variable b: INTEGER := 0;

-- Anweisungsfolge im Testprozess
A1: a <= a + 1 after 0.5 ns; b := b + 1;
A2: wait for 1 ns;
A3: a <= a + 1 after 0.5 ns; b := b + 1;
A4: a <= a + 1 after 0.5 ns; b := b + 1;
A5: wait for 1 ns;
```

Welche Werte haben das Signal *a* und die Variable *b* nach Abarbeitung jeder der Anweisungszeilen A1 bis A5?

Aufgabe 1.8

Gegeben ist der folgende logische Ausdruck:

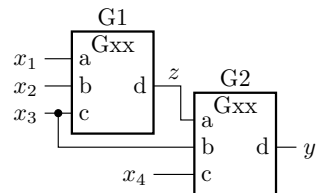
$$\overline{x_1 \wedge x_2 \wedge \bar{x}_3} \wedge \overline{\bar{x}_1 \wedge \bar{x}_2 \wedge \bar{x}_3}$$

- Zeichnen Sie den Berechnungsbaum unter Verwendung der ein- und zweistelligen logischen Operationen Invertierung, UND und ODER.
- Zeichnen Sie den Signalflussplan unter Verwendung von Invertern sowie UND- und NAND-Gattern mit *n* Eingängen.
- Der Ausdruck sei die logische Funktion einer Schaltung. Das Eingangssignal sei vom Typ `STD_LOGIC_VECTOR`, das Ausgabesignal vom Typ `STD_LOGIC` und die Verzögerungszeit 2 ns. Beschreiben Sie die Signalvereinbarungen und die Signalzuweisung in VHDL.

Aufgabe 1.9

Gegeben sind die Funktionsbeschreibung des Teilschaltungstyps »Gxx« und der Signalflussplan einer Schaltung mit zwei Instanzen von diesem Typ:

```
entity Gxx is
  port(a, b, c: in STD_LOGIC;
        d: out STD_LOGIC);
end entity Gxx;
architecture fkt of Gxx is
begin
  d <= (a and not c) or (b and c) after 1 ns;
end architecture;
```



⇒WEB-Projekt: P1.2/Gxx.vhdl

- a) Beschreiben Sie die Struktur der Gesamtschaltung in VHDL. Der Bezeichner der Gesamtschaltung sei »S2B«, der Beschreibungsname »stk« und der Datentyp der Anschlüsse der Gesamtschaltung STD_LOGIC.
- b) Entwickeln Sie einen Testrahmen »Test_S2B« mit dem Beschreibungsnamen »t1«, der die Strukturbeschreibung der Gesamtschaltung als Instanz und einen Prozess, der nacheinander im Abstand von 5 ns die Eingabewerte »0000«, »0001«, »0011« und »0111« den zu einem Vektor zusammenfassenden Eingabesignalen $x_4 \dots x_1$ zuweist, enthält.

1.3 Laufzeittoleranz

Definition 1.17 (Laufzeittoleranz) *Eine Schaltung ist laufzeittolerant, wenn ihre Funktion nicht von den internen Signalverzögerungen abhängt, solange diese innerhalb ihrer zulässigen Toleranzbereiche liegen.*

Eine kombinatorische Schaltung bildet die Werte der Eingabesignale verzögert auf die Werte der Ausgabesignale ab. Die Abbildungsfunktion ist eindeutig. Im stationären Zustand, nach Abklingen aller Änderungsvorgänge, ist jedem Eingabewert genau ein Ausgabewert zugeordnet. Das Verzögerungsverhalten wird von dem bisherigen Modell jedoch nur grob angenähert. Die physikalischen Größen zur Darstellung der Signalwerte (Spannungen, Ströme etc.) ändern sich stetig. Sie springen nicht, wie in den bisherigen Simulationsmodellen angenommen, zwischen »0« und »1« hin und her, sondern sie sind bei jedem Signalwechsel kurzzeitig ungültig (vgl. Abschnitt 1.2.1). Bei manchen Eingabeänderungen wechseln die Ausgaben sogar mehrfach zwischen »0« und »1«, bevor der stationäre Wert erreicht ist. Darüber hinaus unterliegen die Zeitparameter erheblichen fertigungs- und temperaturbedingten und von anderen Einflussgrößen abhängigen Streuungen. Eine digitale Schaltung arbeitet nur dann zuverlässig, wenn ihre Funktion nicht von Zeitparametern abhängt, wenn diese innerhalb ihrer zulässigen Toleranzbereiche liegen.

1.3.1 Glitches

Ein Glitch ist ein kurzer Impuls, der durch Verknüpfung von Signalen mit zeitversetzten Änderungen entsteht. Abbildung 1.34 zeigt eine kleine Gatterschaltung mit simulierten Beispielsignalverläufen. Bei einer steigenden Flanke an x_2 und einer verzögerten fallenden Flanke an z entsteht ein kurzer Impuls am Ausgang y . Auftreten und Länge der Glitches hängen von den Flankenversätzen der Eingabesignale und von den Verzögerungen innerhalb der Schaltung ab und sind in der Regel nicht genau vorhersagbar.

Glitches können in einer kombinatorischen Schaltung bereits dann entstehen, wenn sich nur ein einzelnes Eingabebit ändert, und zwar, wenn der Signalfluss rekonvergent ist. Rekonvergent bedeutet, dass der Signalfluss sich

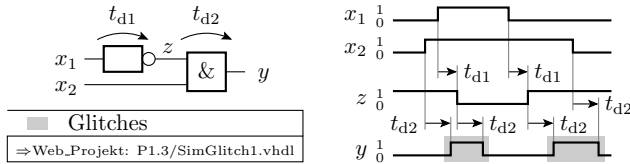


Abb. 1.34. Glitches verursacht durch geringfügig zeitversetzte Signaländerungen an unterschiedlichen Eingängen

verzweigt und später wieder zusammentrifft. Entlang der unterschiedlichen Signalpfade werden die Änderungen unterschiedlich stark verzögert, so dass an den Eingängen des Funktionsblocks, an dem der Signalfluss wieder zusammentrifft, Signale mit zeitversetzten Änderungen ankommen. In Abb. 1.35 a verzweigt das Signal x_1 . Auf dem unteren Zweig wird es mit x_2 verknüpft und verzögert. Auf dem oberen Pfad trifft es unverzüglich auf das Gatter G2. Bei $x_2 = 1$ und einer steigenden Flanke an x_1 wechselt dadurch die Ausgabe kurzzeitig auf »0« (Abb. 1.35 b).

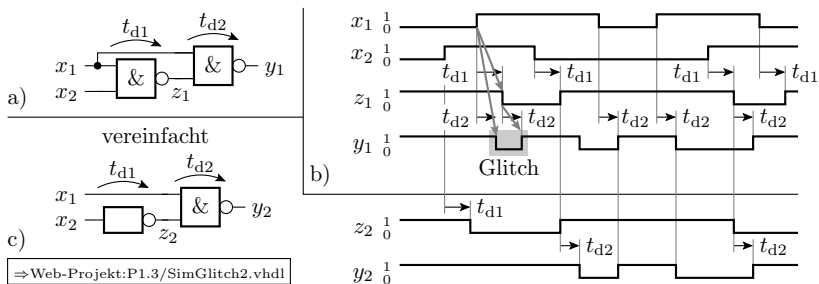


Abb. 1.35. a) Schaltung mit rekonvergentem Signalfluss b) simulierte Zeitverläufe mit einem Glitch c) funktionsgleiche Schaltung ohne rekonvergenten Signalfluss

Das Entstehen von Glitches hängt nicht von der Funktion, sondern von der Struktur einer Schaltung ab. Der Ausdruck, der die logische Funktion der Schaltung in Abb. 1.35 a beschreibt, kann z.B. wie folgt umgeformt werden:

$$y = \overline{\overline{x_2}x_1}x_1 = \overline{(\overline{x_2} \vee \overline{x_1})}x_1 = \overline{x_2}x_1$$

Die Schaltung für den vereinfachten Ausdruck enthält keine rekonvergenten Signalpfade. Das schließt aus, dass bei der Änderung eines einzelnen Eingabebits ein Glitch entsteht (Abb. 1.35 c). Zeitnahe Änderungen an mehreren Eingängen können natürlich weiterhin Glitches verursachen. Wenn die Schaltungsstruktur einer kombinatorischen Funktion unbekannt, noch zu entwerfen oder zu synthetisieren ist, kann nicht vorhergesagt werden, ob oder unter welchen Bedingungen Glitches im Einsatz auftreten werden. In einem synthesebasierten Entwurf gilt deshalb die Grundregel:

Eine Schaltung ist nur dann laufzeittolerant, wenn ihre Funktion nicht von möglicherweise auftretenden oder nicht auftretenden Glitches abhängt.

1.3.2 Das Verzögerungsmodell einer Signalzuweisung

Glitches vervielfachen die Anzahl der Änderungsereignisse bei der Ausbreitung eines Signals in einer Schaltung. Das treibt den Rechenaufwand für die Simulation in die Höhe. Auf der anderen Seite benötigt jede physikalische Signaländerung eine gewisse Zeit, so dass zumindest die sehr kurzen Glitches, die sich nach dem bisherigen Simulationsmodell ergeben, in Wirklichkeit gar nicht auftreten. Auch um Rechenzeit zu sparen, sind kurz aufeinanderfolgende Signaländerungen in der Regel zu unterdrücken. Die Mindestlänge der Glitches, die die Simulation berechnet und weiterleitet, wird mit dem Verzögerungsmodell der Signalzuweisungen eingestellt.

Ohne Angabe hat eine Signalzuweisung das Standardverzögerungsmodell. Mit diesem werden bei der Zuweisung eines neuen Wert-Zeit-Tupels alle schwebenden Zuweisungen gelöscht. Es gibt eine Ausnahme. Wenn die schwebenden Zuweisungen den neuen Wert bereits für einen früheren Simulationszeitpunkt vorgemerkt haben, wird der frühere Zeitpunkt beibehalten. Insgesamt werden mit dem Standardverzögerungsmodell alle Glitches bis zur Länge der Verzögerungszeit unterdrückt. In Abb. 1.36 a wird das Eingangssignal x mit einer Verzögerung von 10 ns an das Ausgabesignal y zugewiesen. Die erste Eingabeänderung wird wirksam, die zweite ist bei der Zuweisung der dritten Änderung noch schwebend und wird gelöscht. Die dritte Zuweisung erreicht zwar ihren Ausführungszeitpunkt, bewirkt aber keine Ausgabesignaländerung. Die vierte Zuweisung wird wieder vor ihrem Ausführungszeitpunkt gelöscht und die

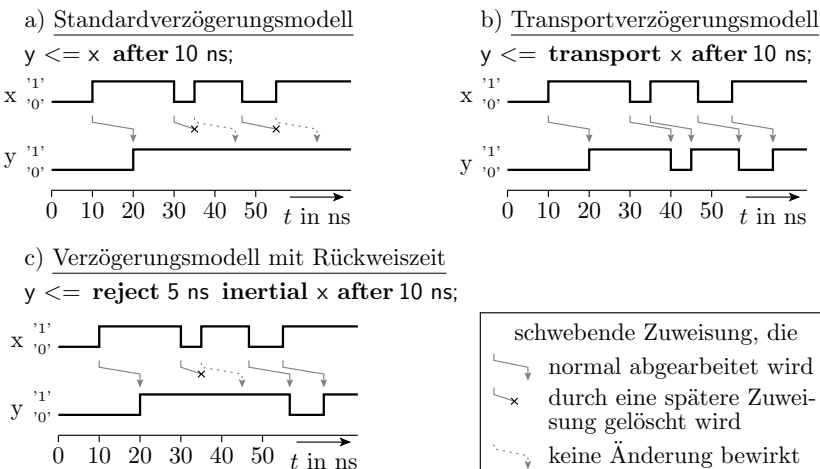


Abb. 1.36. Signalzuweisungen mit unterschiedlichen Verzögerungsmodellen

fünfte bewirkt wieder keine Ausgabesignaländerung. Die Mindestdauer eines Eingabeimpulses, der auf einen Ausgabeimpuls abgebildet wird, ist 10 ns.

Die anderen Verzögerungsmodelle sind in der Signalzuweisung anzugeben:

```
Signalname <= VM W [after td]{, W after td};
VM ⇒ transport|[reject tr] inertial]
```

(VM – Verzögerungsmodell; W – Ausdruck zur Berechnung des neuen Wertes; t_d – Verzögerungszeit; t_r – Rückweiszeit). Das Transportverzögerungsmodell (Schlüsselwort »transport«) deaktiviert die Glitch-Unterdrückung. Bei der Zuweisung eines neuen Wert-Zeit-Tupels bleiben alle schwebenden Zuweisungen, deren Zeitwerte vor dem der neuen Zuweisung liegen, erhalten. Schwebende Zuweisungen für spätere Zeitpunkte werden unabhängig vom Verzögerungsmodell immer gelöscht. Die Signalzuweisung in Abb. 1.36 b erzeugt eine exakte zeitversetzte Kopie des Eingabesignals auf der rechten Zuweisungsseite.

Zwischen den beiden Grenzfällen – Löschen aller oder nur der späteren schwebenden Zuweisungen – gibt es ein weiteres Verzögerungsmodell mit einstellbarer Glitch-Unterdrückungsdauer. Das ist das Reject-Inertial-Modell:

```
y <= reject tr inertial W after td{, W after td};
```

(t_d – Verzögerungszeit; t_r < t_d – Rückweiszeit). Bei einer neuen Zuweisung werden bei diesem Verzögerungsmodell nur die schwebenden Zuweisungen für spätere Zeitpunkte als

$$t_{\text{sim}} + t_d - t_r$$

(t_{sim} – aktuelle Simulationszeit der Signalzuweisung) gelöscht. Dabei gibt es wieder dieselbe Ausnahme wie bei dem Standardverzögerungsmodell. Wenn die schwebenden Zuweisungen den neuen Wert bereits für einen früheren Simulationszeitpunkt vorgemerkt haben, wird der frühere Zeitpunkt beibehalten. Eine Signalzuweisung mit diesem Verzögerungsmodell unterdrückt alle Glitches bis zur Länge der Rückweiszeit t_r (Abb. 1.36 c).

Abbildung 1.37 zeigt ein Beispielprogramm, um die Wirkung der unterschiedlichen Verzögerungsmodelle genauer zu untersuchen. Die Signale sind vom Typ NATURAL und haben den Anfangswert null. Zum Simulationsbeginn werden allen drei Signalen fünf Änderungen im Abstand von 1 ns mit von »1« hochzählenden Werten zugewiesen. Nach einer Wartezeit von 2,5 ns werden zwei weitere Änderungen nach 3 ns und 4 ns zugewiesen, diesmal jedoch mit unterschiedlichen Verzögerungsmodellen.

Mit dem Standardverzögerungsmodell löscht die zweite Zuweisung die schwebenden Änderungen für alle Zeitpunkte größer der aktuellen Simulationszeit von 2,5 ns. Von der ersten Zuweisung werden nur die Änderungen nach 1 ns und 2 ns ausgeführt. Mit dem Transportverzögerungsmodell bleiben die schwebenden Änderungen für alle Zeitpunkte größer der aktuellen Simulationszeit von 2,5 ns plus der Verzögerungszeit von 3 ns erhalten, so dass auch die Änderungszuweisungen für 3 ns, 4 ns und 5 ns ausgeführt werden. Bei der Zuweisung mit Rückweiszeit werden die schwebenden Änderungen für alle Zeitpunkte größer der aktuellen Simulationszeit von 2,5 ns plus der Verzögerungszeit von 3 ns

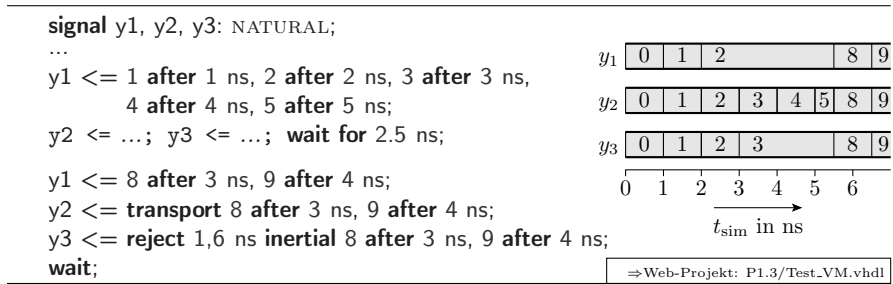


Abb. 1.37. Test der Wirkung der Verzögerungsmodelle (... – Zuweisung wie an y_1)

abzüglich der Rückweiszeit von 1,6 ns gelöscht. Die schwebende Änderung für 3 ns wird ausgeführt und die schwebenden Änderungen für 4 ns und 5 ns werden gelöscht.

1.3.3 Simulation mit Halte- und Verzögerungszeiten

Der Toleranzbereich für die Verzögerung einer kombinatorischen Schaltung – sei es ein einzelnes Gatter oder eine Schaltung aus vielen Gattern – lässt sich mit zwei Worst-Case-Parametern beschreiben:

- der minimalen Haltezeit t_h (hold time) und
- der maximalen Verzögerungszeit t_d (delay time).

Die minimale Haltezeit t_h – kurz Haltezeit – ist die Zeit, die der alte gültige Ausgabewert nach einer Eingabeänderung garantiert noch erhalten bleibt. Die maximale Verzögerungszeit t_d – kurz die Verzögerungszeit – ist die Zeit nach Anlegen des neuen gültigen Eingabewertes, nach der spätestens der neue gültige Wert ausgegeben wird (Abb. 1.38).

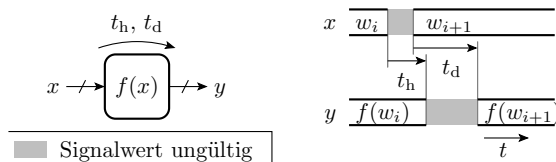


Abb. 1.38. Simulation mit Halte- und Verzögerungszeiten

Das VHDL-Modell für dieses Zeitverhalten ist ein Prozess, der bei jeder Eingabeänderung geweckt wird. Innerhalb des Prozesses wird abgefragt, ob der alte Eingabewert vor der Änderung gültig war und ob der neue Eingabewert gültig ist. Im ersten Fall wird dem Ausgabesignal nach der Haltezeit der Wert »ungültig« zugewiesen, im zweiten Fall nach der Verzögerungszeit der neue gültige Funktionswert:

```

process(x)
begin
  if vorherige_Eingabe_gültig then
    y <= ungültig after th;
  end if;
  if neuer_Eingabewert_gültig then
    y <= transport f(x) after td;
  end if;
end process;

```

($f(x)$ – Vorschrift zur Berechnung der Ausgabewerte aus den Eingabewerten, vorerst ein Ausdruck, nach der Einführung von VHDL-Funktionen meist ein Funktionsaufruf). Die zweite Zuweisung muss mit dem Transportverzögerungsmodell erfolgen, damit die möglicherweise noch schwebende Zuweisung des Pseudo-Wertes »ungültig« nicht gelöscht wird.

Dasselbe Verhalten kann verkürzt mit nur einer einzigen nebenläufigen Signalzuweisung beschrieben werden:

```
y <= ungültig after th, f(x) after td;
```

Eine nebenläufige Signalzuweisung wird in einen Prozess übersetzt, der bei jeder Eingabeänderung geweckt wird (vgl. Abschnitt 1.2.4). Es sei unterstellt, dass der Funktionswert einer ungültigen Eingabe gleichfalls ungültig ist. Dann wird bei einer Eingabeänderung nach »ungültig« sowohl nach der Haltezeit als auch nach der Verzögerungszeit der Wert »ungültig« zugewiesen. Die zweite Zuweisung hat keine Wirkung. Bei der nachfolgenden Eingabeänderung nach »gültig« hat die Zuweisung von »ungültig« nach der Haltezeit auch keine Wirkung. Erst die Zuweisung des neuen gültigen Funktionswertes ändert die Ausgabe.

Abbildung 1.39 demonstriert das am Beispiel eines Inverters. Die Signalzuweisung an x in Abb. 1.39 a stellt das Testeingabesignal bereit. Sie wird in

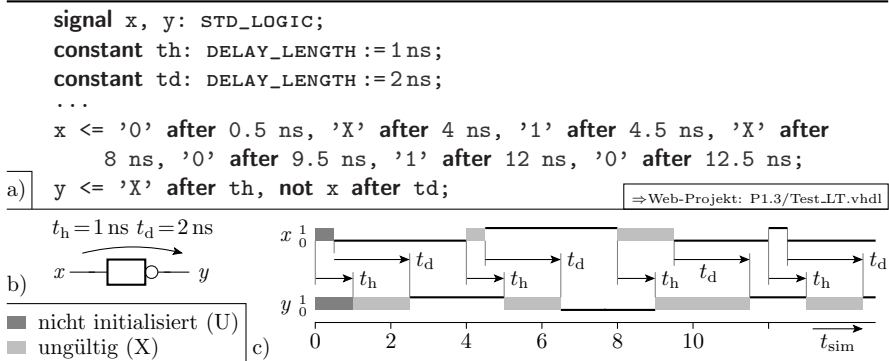


Abb. 1.39. Simulation eines Inverters mit Halte- und Verzögerungszeit a) Simulationsmodell b) Schaltung c) Simulationsergebnis

einen Prozess mit einer leeren Weckliste übersetzt. Ein solcher Prozess wird nur einmal zum Simulationsbeginn ausgeführt. Er weist im konkreten Fall einmalig einen Signalverlauf zu und legt sich danach dauerhaft schlafen. Die zweite Zuweisung beschreibt den Inverter in Abb. 1.39 b mit einer Haltezeit t_h und einer Verzögerungszeit t_d . Der Prozess, der die Zuweisung im Simulationsmodell einrahmt, wird außer zum Simulationsbeginn auch bei jeder Änderung von x geweckt. Wenn das Eingangssignal ungültig wird, speichert der Ausgang für die Haltezeit noch den alten Wert und wird dann »ungültig«. Nach Anliegen eines neuen gültigen Eingabewertes übernimmt der Ausgang den invertierten Wert erst nach der Verzögerungszeit (Abb. 1.39 c).

Abbildung 1.40 zeigt eine weitere Beispielsimulation mit Halte- und Verzögerungszeiten, diesmal von einer Schaltung aus drei Gattern. Die UND-Gatter reagieren im Beispiel auf Eingabeänderungen frühestens nach 0,5 ns und spätestens nach 1 ns. Für das ODER-Gatter liegt die Verzögerung im Bereich zwischen 0,5 ns und 2 ns. Das hat im Beispiel zur Folge, dass das Ausgabe-signal die meiste Zeit ungültig ist. Die mehrfachen Wechsel der berechneten Signale zwischen »U« und »X« haben im Beispiel keine Bedeutung. Sie kommen dadurch zustande, dass bei der Simulation mit STD_LOGIC zwischen zwei Ursachen für ungültige Werte unterschieden wird. Wenn die Ursache eine fehlende Initialisierung ist, wird ungültig durch »U« und sonst durch »X« dargestellt. Diese Unterscheidung dient zur Aufdeckung von Initialisierungsfehlern in Schaltungen mit Speicherverhalten.

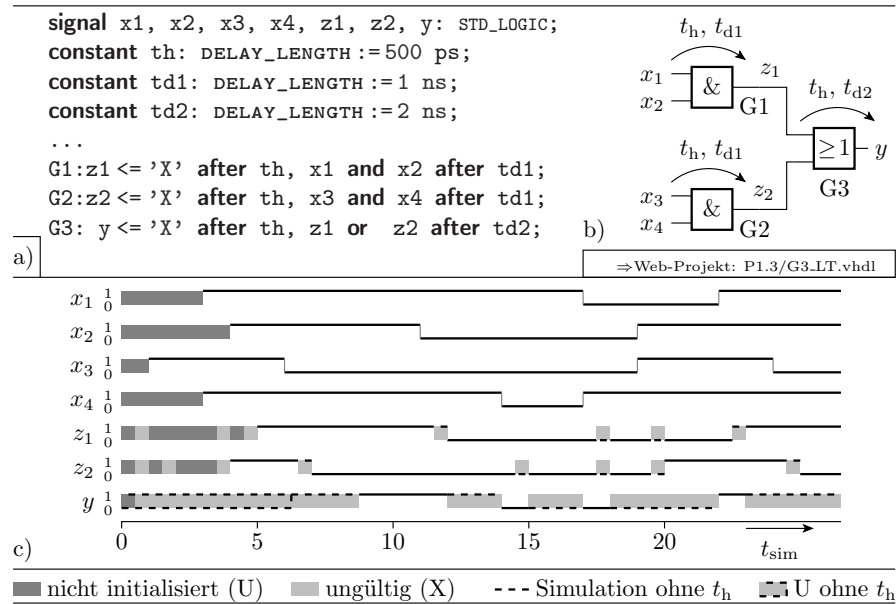


Abb. 1.40. Simulation einer 3-Gatter-Schaltung mit Halte- und Verzögerungszeiten a) VHDL-Beschreibung b) Schaltung c) Simulationsergebnis

Die gestrichelten Linien an den berechneten Signalverläufen zeigen die Werteverläufe, die sich bei einer Simulation ohne Haltezeiten ergeben. Ohne Haltezeiten werden die Gültigkeitsfenster nicht mitberechnet. Ein ungültiges Signal hat nicht unbedingt den falschen Wert. In unserem Modell bedeutet ungültig lediglich, dass der Wert nicht garantiert richtig ist. Eine Schaltung, die ungültige Werte weiterverarbeitet, funktioniert möglicherweise, nur kann die Simulation dafür nicht garantieren.

Der Nachweis, dass eine Schaltung laufzeittolerant ist, erfordert eine Simulation mit Halte- und Verzögerungszeiten, damit auch die Gültigkeitsfenster mitberechnet werden.

1.3.4 Laufzeitanalyse

Die Laufzeitanalyse ist ein einfaches Verfahren zur Berechnung der Halte- und der Verzögerungszeit einer Gesamtschaltung aus den Halte- und Verzögerungszeiten der Teilschaltungen, aus denen sie besteht. In einer Schaltung mit einem gerichteten Datenfluss addieren sich die Halte- und Verzögerungszeiten entlang aller Signalpfade. Die Gesamthaltezeit ist die Zeit, die sich die Ausgabe nach einer Eingabeänderung garantiert nicht ändert. Das ist die Haltezeit des Pfades mit der kleinsten Summe von Einzelhaltezeiten. Die Gesamtverzögerungszeit ist die Zeit, nach der das System spätestens den Funktionswert der neuen Eingabe ausgibt. Das ist entsprechend die Verzögerungszeit entlang des längsten Pfades mit der größten Verzögerung.

Abbildung 1.41 zeigt eine Schaltung aus fünf Gattern. Die Tabelle daneben zeigt für alle Pfade durch die Schaltung die aufsummierten Halte- und Verzögerungszeiten. Die kürzeste Haltezeit hat der Pfad von x_5 über G4 nach y_2 . Seine Haltezeit ist die Gesamthaltezeit. Die längste Verzögerungszeit, die gleichzeitig die Gesamtverzögerungszeit ist, hat der Pfad über G2-G4-G5.

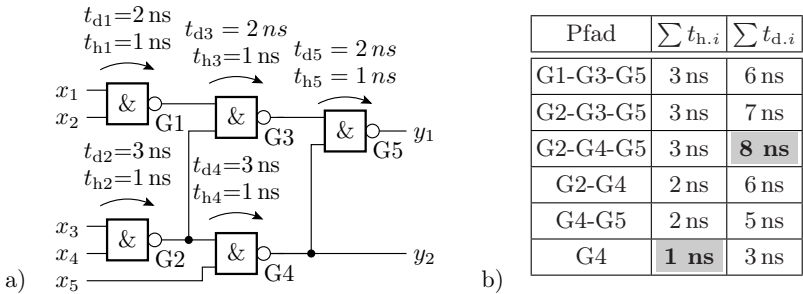


Abb. 1.41. Beispiel für die Laufzeitanalyse a) Schaltung b) Tabelle der Halte- und Verzögerungszeiten aller Pfade

Für größere Schaltungen sind Algorithmen vom Typ »Wiederhole für jeden Pfad« ungünstig, weil die Pfadanzahl exponentiell mit der Schaltungsgröße zunehmen kann. Die Laufzeitanalyse lässt sich auch mit einem Algorithmus nach dem Schema »Wiederhole für alle Teilschaltungen« beschreiben, bei dem der Rechenaufwand nur linear mit der Schaltungsgröße zunimmt. Dazu werden die Eingänge der Gesamtschaltung, die Eingänge aller Teilschaltungen, die Ausgänge der Teilschaltungen und die Ausgänge der Gesamtschaltung je zu einer Signalgruppe zusammengefasst und für jede Signalgruppe die Halte- und die Verzögerungszeitdifferenz zur Signalgruppe S0 berechnet⁹. Beginnend an den Schaltungseingängen werden dabei für jede Teilschaltung zwei Operationen ausgeführt:

- Berechnung der Halte- und Verzögerungszeitdifferenz für die Gruppe der Eingabesignale durch Minimum- bzw. Maximumbildung und
- Berechnung der Halte- und Verzögerungszeitdifferenz für die Gruppe der Ausgabesignale durch Addition der Parameter der Teilschaltung.

Abbildung 1.42 a zeigt die Beispielschaltung aus Abb 1.41 mit den eingezeichneten Signalgruppen. Abbildung 1.42 b veranschaulicht den Rechenweg. Die Halte- und Verzögerungszeitdifferenzen der Gruppen S1 und S2 zur Gruppe S0 sind gleich den Halte- und Verzögerungszeiten der dazwischen liegenden Gatter. Gruppe S3 übernimmt für die Haltezeitdifferenz das Minimum und für die Verzögerungszeitdifferenz das Maximum der Gruppen S1 und S2. Für Gruppe S5 werden die Halte- und Verzögerungszeiten des Gatters dazwischen dazu addiert etc. Die grauen Balken zeigen, wie sich das Ungültigkeitsfenster der Signalgruppe S0 auf die Ungültigkeitsfenster der im Datenfluss nachfolgenden Signale fortpflanzt. Der Rechenaufwand verhält sich proportional zur Anzahl der Teilschaltungen.

1.3.5 Zusammenfassung und Übungsaufgaben

Bei einer realen Schaltung ist die Verzögerung zwischen den Eingabe- und den Ausgabeänderungen kein exakt angebbarer Wert. Denn jedes Signal ist beim Übergang zwischen zwei gültigen Werten kurzzeitig ungültig. Bis zum Erreichen des stationären Wertes können mehrfache Wechsel (Glitches) auftreten. Diese und weitere Eigenarten digitaler Schaltungen lassen sich nur mit Toleranzbereichen für die Zeitparameter modellieren. Eine kombinatorische Schaltung hat eine (minimale) Haltezeit, nach der nach einer Eingabeänderung garantiert noch der alte, und eine (maximale) Verzögerungszeit, nach der spätestens der neue Wert am Ausgang anliegt. In der Zeit dazwischen kann für den Ausgabewert nicht garantiert werden. Das »nicht garantiert« wird durch den Pseudo-Wert »ungültig«, dargestellt. Die Halte- und die Verzögerungszeiten einer Gesamtschaltung lassen sich mit einer Laufzeitanalyse aus den Halte- und Verzögerungszeiten der Teilschaltungen berechnen.

⁹ Die Signalgruppe S0 ist die Gruppe der Eingabesignale der Gesamtschaltung.

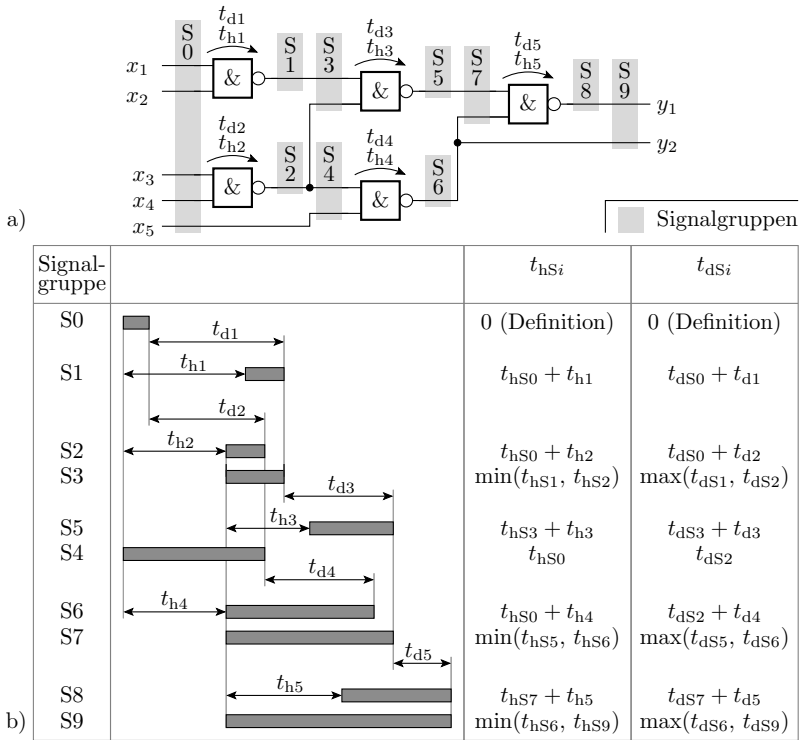


Abb. 1.42. Laufzeitanalyse mit Signalgruppen a) Schaltung b) Berechnung

Aufgabe 1.10

- Entwickeln Sie einen Testrahmen zur Simulation der Schaltung in Abb. 1.43 mit den vorgegebenen Eingabesignalverläufen. Die Gatter und Eingabesignalverläufe sollen durch nebenläufige Signalzuweisungen beschrieben und die Signale z_i und y zum Simulationsbeginn die Anfangswerte haben, die sie hätten, wenn die Anfangseingabewerte vor dem Simulationsbeginn schon länger als die maximale Verzögerungszeit anliegen würden.
- Bestimmen Sie die Verläufe der internen Signale und des Ausgabesignals zuerst ohne Rechner und überprüfen Sie das Ergebnis durch Simulation.

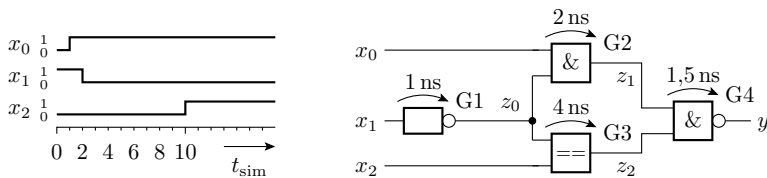


Abb. 1.43. Schaltung und Eingabesignale zu Aufgabe 1.10

- c) Einer der Eingangessignalwechsel verursacht zwei Signalwechsel am Ausgang. Welcher Signalwechsel ist das und unter welcher Bedingung kommt der Mehrfachsignalwechsel zustande?

Aufgabe 1.11

Innerhalb eines Prozesses werden folgende Signalzuweisungen und Warteanweisungen abgearbeitet:

```
y <= '0', 'X' after 3 ns, '1' after 7 ns, 'X' after 8 ns, '1' after 10 ns,
  '0' after 11 ns, '1' after 13 ns, '1' after 15 ns, 'X' after 18 ns, '0' after 20 ns;
wait for 5 ns;
y <= '1' after 12 ns;
-- y <= transport '1' after 12 ns;
-- y <= reject 8 ns inertial '1' after 12 ns;
```

Welche der schwebenden Änderungen werden gelöscht, wenn die zweite Signalzuweisung

- mit dem Standardverzögerungsmodell
- mit dem Verzögerungsmodell »transport«
- mit dem Verzögerungsmodell »reject 8 ns inertial«

erfolgt und welchen Signalverlauf erzeugen die drei Zuweisungen, wenn die erste Signalzuweisung zum Zeitpunkt $t_{\text{sim}} = 0$ ausgeführt wird? Überprüfen Sie das Ergebnis mit dem Simulator.

Aufgabe 1.12

Bestimmen Sie für die Schaltung in Abb. 1.44 die Signalverläufe der internen Signale z_0 bis z_2 und des Ausgabesignals y für den Eingabesignalverlauf

```
x <= '0', '1' after 10 ns, '0' after 20 ns;
```

Alle Signale seien vom Typ STD_LOGIC und haben zum Simulationsbeginn den Pseudo-Wert 'U' (uninitialised, nicht initialisiert).

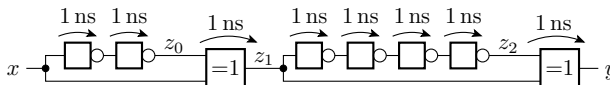


Abb. 1.44. Schaltung zu Aufgabe 1.12

Aufgabe 1.13

Gegeben ist die nachfolgende Schaltungsbeschreibung:

```
-- Vereinbarungen der Entwurfseinheit
signal x: STD_LOGIC_VECTOR(4 downto 0) := "01011";
signal z: STD_LOGIC_VECTOR(4 downto 0);
signal y: STD_LOGIC;
-- nebenläufige Zuweisungen im Beschreibungsteil der Entwurfseinheit
G1: z(0) <= 'X' after 4 ns, x(0) and x(1) after 8 ns;
G2: z(1) <= 'X' after 5 ns, x(2) or x(3) after 6 ns;
G3: z(2) <= 'X' after 3 ns, z(0) or z(1) after 6 ns;
G4: z(3) <= 'X' after 4 ns, z(1) and x(3) after 7 ns;
G5: z(4) <= 'X' after 3 ns, z(3) or x(4) after 5 ns;
G6: y    <= 'X' after 5 ns, z(2) or z(4) after 7 ns;
```

- Zeichnen Sie den Signalflussplan und lesen Sie für alle Teilschaltungen die Halte- und die Verzögerungszeiten ab.
- Bestimmen Sie die Halte- und die Verzögerungszeit der Gesamtschaltung.

1.4 Register

Nach einer Eingabeänderung sind die Signalwerte an den Ausgängen einer kombinatorischen Schaltung für eine gewisse Zeit ungültig. Ungültig kann bedeuten, dass die Spannungswerte zur Signaldarstellung im verbotenen Bereich liegen oder dass zu nicht genau bekannten Zeitpunkten Signaländerungen auftreten können. Bei der Simulation pflanzen sich die Zeitfenster mit gültigen und ungültigen Werten entlang der Signalfade fort. Da die Haltezeiten stets kleiner als die Verzögerungszeiten sind, nimmt die Breite der Gültigkeitsfenster entlang der Signalfade ab. Wie aus Abb. 1.45 ablesbar, ist die Breite t_{GE} des Gültigkeitsfensters am Ende eines Signalfades gleich der Breite t_{GA} des Gültigkeitsfensters am Anfang abzüglich der Differenz aus den Verzögerungszeiten und den Haltezeiten entlang des Pfades. Für den Pfad in der Abbildung gilt z.B.

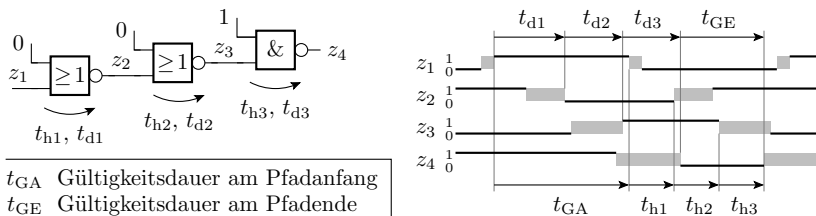


Abb. 1.45. Abnahme der Breite des Gültigkeitsfensters entlang eines Signalfades

$$t_{GE} = t_{GA} + \sum_{i=1}^3 t_{hi} - \sum_{i=1}^3 t_{di}$$

Damit die Daten an den Pfadenden weiterverarbeitet werden können, müssen sie innerhalb ihrer Gültigkeitsfenster abgetastet werden. Abtasten bedeutet, dass die Daten in einem schmalen Zeitfenster übernommen und die übrige Zeit gespeichert werden. Die Abtastung verbreitert die Gültigkeitsfenster, in dem die Daten weiterverarbeitet werden können.

1.4.1 Register als Abtastelemente

Das Funktionselement für die Signalabtastung heißt Register. Es hat das abzutastende Signal $\mathbf{x}$ als Eingang, ein typgleiches abgetastetes Signal $\mathbf{x}'$ als Ausgang und einen Takteingang T (Abb. 1.46). Der Takt ist ein Speziaalsignal, das mit hoher Zeitgenauigkeit periodisch zwischen »0« und »1« umschaltet und dessen Flanken die Abtastzeitpunkte festlegen. Die Abtastflanken – das können die steigenden, die fallenden oder beide Taktflanken sein – werden als aktive Taktflanken bezeichnet.

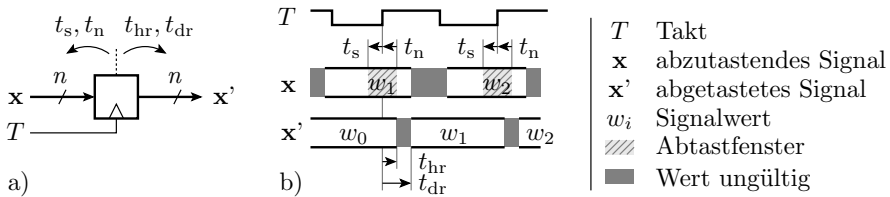


Abb. 1.46. Register a) Symbol b) Zeitverhalten

Das Abtastverhalten eines Registers wird durch die folgenden vier Zeitparameter beschrieben:

- die Vorhaltezeit t_s ,
- die Nachhaltezeit t_n ,
- die Registerhaltezeit t_{hr} und
- die Registerverzögerungszeit t_{dr} .

Die allgemeine Funktion eines Registers ist anhand der Signalverläufe in Abb. 1.46 b dargestellt. Bei jeder aktiven Taktflanke – in der Abbildung der steigenden Flanke – wird das Ausgabesignal nach der Registerhaltezeit t_{hr} ungültig und übernimmt nach der Registerverzögerungszeit t_{dr} den abgetasteten Wert. Voraussetzung für eine korrekte Datenübernahme ist, dass der Eingabewert im gesamten Abtastfenster stabil und gültig ist. Das Abtastfenster beginnt dabei um die gegebene Vorhaltezeit vor und endet nach der gegebenen Nachhaltezeit nach der aktiven Taktflanke. Meist kann entweder die Vorhaltezeit oder die Nachhaltezeit gleich null gesetzt werden.

Aus der Spezialaufgabe eines Taktes, die Abtastzeitpunkte festzulegen, resultieren folgende Besonderheiten. Taktsignale werden im Gegensatz zu anderen Signalen ohne Zeittoleranzen und unbestimmte Zwischenzustände modelliert. Die physikalisch bedingten zeitlichen Unsicherheiten der Taktflanken stecken mit in den Zeitparametern der Register. Takte müssen Glitch-frei sein und mit geringen Verzögerungsunterschieden an die Takteingänge der einzelnen Register und Registerzellen geführt werden. Das erfordert spezielle Taktversorgungsschaltungen, die in der Regel manuell zu entwerfen sind (siehe später Abschnitt 4.3.6).

Das Abtasten ist eine Grundfunktion für alle sequenziellen Abläufe in einer digitalen Schaltung. Deshalb wird eine spezielle Notation eingeführt. Das Signal, aus dem durch Abtastung ein Signal x gebildet wird, wird mit x^+ bezeichnet, das einmal abgetastete Signal von x mit x' , das zweimal abgetastete Signal mit x'' etc. In VHDL werden wir für $x^+ \gg x_next$ «, für $x' \gg x_del$ «, für $x'' \gg x_del2$ « etc. schreiben.

1.4.2 VHDL-Abtastprozesse

Das Verhaltensmodell für ein Register ist ein Abtastprozess. Ein Abtastprozess ist ein Prozess, der nur von Taktänderungen geweckt wird, nur bei einer aktiven Taktflanke Signalzuweisungen ausführt und der optional die Vor- und Nachhaltebedingungen kontrolliert. Dafür werden zusätzliche, noch nicht eingeführte VHDL-Beschreibungsmittel benötigt:

- Signalattribute,
- Fallunterscheidungen im Kontrollfluss und
- Anweisungen zur Kontrolle zuzusichernder Bedingungen.

Attribute sind in VHDL Beschreibungsmittel, um zusätzliche Eigenschaften von Objekten abzufragen. Mit Signalattributen kann z.B. abgefragt werden, ob ein bestimmtes Signal den Prozess geweckt hat, ob ein anderes Signal zum Weckzeitpunkt bereits hinreichend lange den aktuellen Wert führt und wann ein Signal das letzte Mal seinen Wert geändert hat (Tabelle 1.1).

Mit den Attributen 'STABLE(...) und 'LAST_EVENT können die Vor- und Nachhaltebedingungen kontrolliert werden. Die Abfrage, ob ein Prozess von der steigenden oder fallenden Flanke geweckt wurde, erfolgt vorzugsweise mit den in dem standardisierten Package IEEE.STD_LOGIC_1164 definierten Funktionen:

```
function RISING_EDGE(signal s: STD_LOGIC) return BOOLEAN is
begin
  return s'EVENT and s='1' and LAST_VALUE(s)='0';10
end function;
```

¹⁰ vereinfachte Beschreibung ohne Berücksichtigung schwacher Signalwerte (schwache Signalwerte siehe später Abschnitt 4.1.6)

Tabelle 1.1. Signalattribute

Attribut	Ergebnistyp	Ergebnis
s' EVENT	BOOLEAN	TRUE, wenn der Prozess durch eine Änderung von s geweckt wurde
s' STABLE(t)	BOOLEAN	TRUE, wenn seit einer Zeit t keine Signaländerung stattgefunden hat
s' LAST_EVENT	DELAY_LENGTH	Zeit seit der letzten Änderung von s
s' LAST_VALUE	$tTyp$	Wert vor der letzten Änderung von s
s' DELAYED(t)	$tTyp$	das um t verzögerte Signal zu s
...	...	

(s – Signal mit dem Datentyp $tTyp$; t – Variable oder Konstante vom Typ DELAY_LENGTH)

```

function FALLING_EDGE(signal  $s$ : STD_LOGIC) return BOOLEAN is
begin
  return  $s'$ EVENT and  $s$ ='0' and LAST_VALUE( $s$ )='1';
end function;

```

Bei einer steigenden Flanke ist der Wert des Signals s , das den Prozess geweckt hat, vor der Änderung »0« und nach der Änderung »1«. Bei einer fallenden Flanke sind die beiden Werte vertauscht. Ein Prozess, der nur den Takt in der Weckliste hat, wird bei der Simulation immer genau einmal geweckt, ohne dass eine Taktflanke aufgetreten ist, und zwar zum Simulationsbeginn. Damit beim Simulationsbeginn keine Datenübernahme simuliert wird, sind die Flankenbedingungen zusätzlich mit »T'EVENT« UND-verknüpft.

Binäre Fallunterscheidungen im Kontrollfluss werden mit der If-Anweisung beschrieben:

```

if  $b$  then
  Anweisung {Anweisung}
end if;

```

Die Bedingung b ist dabei ein Ausdruck oder eine Funktion vom Typ BOOLEAN, z.B. der Rückgabewert der Funktion RISING_EDGE(...) für »steigende Flanke«. Die Kontrolle zugesicherter Bedingungen erfolgt vorzugsweise mit der Assert-Anweisung

```

[assert  $b$ ] report  $m$  [severity  $sl$ ];

```

(b – zugesicherte Bedingung; m – Text für die Fehlermeldung; sl – Fehler schwere). Die Fehlerschwere klassifiziert die Bildschirmausgabe nach ihrer Bedeutung für die Simulation und hat den im Package STD.STANDARD vordefinierten Typ

```

type SEVERITY_LEVEL is (
  NOTE,           -- Kontrollausgabe

```

```

WARNING,    -- Warnung
ERROR,      -- Fehler
FAILURE);   -- schwerer Fehler/Simulationsabbruch

```

Ohne Assert-Klausel wird die Report-Anweisung immer ausgeführt, ohne Severity-Klausel erhält die Ausgabe die Fehlerschwere NOTE.

Abbildung 1.47 zeigt eine Verhaltensbeschreibung für einen Register-Abtastprozess. Die Typangabe $tTyp$ steht für einen beliebigen Typ mit einem Pseudo-Wert für ungültig, z.B. STD_LOGIC oder STD_LOGIC_VECTOR. In der Weckliste darf nur der Takt stehen (Zeile 6). Bei einer steigenden Taktflanke (Zeile 8) wird, wenn die Vorhaltebedingung erfüllt ist (Zeile 9), an die Ausgabe nach der Haltezeit der Pseudo-Wert »ungültig« und nach der Verzögerungszeit der aktuelle Eingabewert zugewiesen (Zeile 10). Die Nachhaltebedingung kann nur rückwirkend kontrolliert werden. Der Prozess wird für die Nachhaltezeit schlafen gelegt (Zeile 11). Wenn sich während dieser Zeit die Eingabe verändert hat, wird die Ausgabe auf ungültig gesetzt und eine Warnung ausgegeben (Zeile 12 bis 14). Wenn bereits die Vorhaltebedingung verletzt war, wird gleichfalls die Ausgabe auf ungültig gesetzt und eine Warnung ausgegeben (Zeile 16 bis 19).

```

1: entity Register is
2:   generic(ts, tn, thr, tdr: DELAY_LENGTH);
3:   port( T: in STD_LOGIC;
4:         x: in tTyp; y: out tTyp);
5: end entity;
...
6: process(T)
7: begin
8:   if RISING_EDGE(T) then
9:     if x'LAST_EVENT>ts then
10:      x_del <= ungültig after thr, x after tdr;
11:      wait for tn;
12:      if x'LAST_EVENT<tn then
13:        x_del <= ungültig;
14:        report "Nachhaltebedingung verletzt"
            severity WARNING;
15:      end if;
16:     else
17:      x_del <= ungültig after thr;
18:      report "Vorhaltebedingung verletzt"
            severity WARNING;
19:     end if;
20:   end if;
21: end process;

```

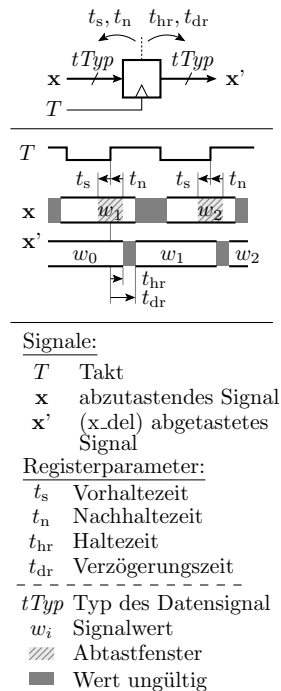


Abb. 1.47. Verhaltensmodell für ein Register

In den nachfolgenden Beispielen werden in den Abtastprozessen die Meldungen weggelassen und für die Nachhaltezeit $t_n = 0$ unterstellt. Das vereinfacht die Registerbeschreibung auf folgende Beschreibungsschablone:

```

6: process(T)
7: begin
8:   if RISING_EDGE(T) then
9:     if x'LAST_EVENT>ts then
10:      x_del <= ungültig after thr, x after tdr;
11:     else
12:      x_del <= ungültig after thr;
13:     end if;
14:   end if;
15: end process;

```

1.4.3 Verarbeitung plus Abtastung

Mit Abtastprozessen werden nicht nur Register, sondern auch kombinatorische Schaltungen, deren Ausgaben abgetastet oder kombinatorische Schaltungen, die abgetastete Signale verarbeiten, modelliert. Abbildung 1.48 zeigt eine kombinatorische Schaltung mit einem Abtastregister am Ausgang.

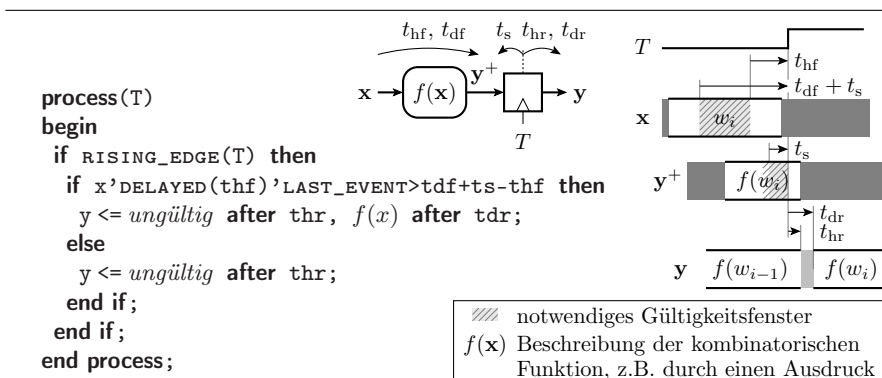


Abb. 1.48. Verarbeitung plus Ausgabeabtastung

Der Prozess wird wie der eines Registers bei jeder Taktänderung geweckt. Zur Kontrolle der Vorhaltebedingung wird das Eingangssignal x mit dem Attribut 'DELAYED(...)' um die Haltezeit der vorgelagerten kombinatorischen Schaltung verzögert und für das verzögerte Signal kontrolliert, dass seine letzte Änderung mindestens eine Zeit $t_{df} + t_s - t_{hf}$ zurückliegt. Wenn die Vorhaltebedingung erfüllt ist, übernimmt der Registerausgang nach einer Haltezeit den Pseudo-Wert »ungültig« und nach der Verzögerungszeit den abgetasteten Funktionswert. Sonst übernimmt der Registerausgang nur den Wert »ungültig« und behält ihn bis zur nächsten aktiven Taktflanke. Die kombinatorische

Funktion kann dabei mit einem Ausdruck, einem Funktionsaufruf oder einer Folge von Anweisungen nachgebildet werden. Bei einer Nachbildung durch mehrere Anweisungen sind die Zwischenergebnisse in Variablen weiterzureichen. Signale sind als Zwischenspeicher ungeeignet. Denn sie übernehmen die ihnen zugewiesenen Werte erst, nachdem der Prozess sich bis zur nächsten Taktänderung schlafen gelegt hat.

Wenn eine Verarbeitungseinheit abgetastete Signale verarbeitet, ist das Verhaltensmodell ähnlich (Abb. 1.49). Die Eingangssignale der kombinatorischen Schaltung ändern sich nur nach einer aktiven Taktflanke. Das Verhaltensmodell ist wieder ein Abtastprozess, der bei jeder aktiven Taktflanke geweckt wird, die Vor- und die Nachhaltebedingungen überprüft, das Ausgangssignal nach einer Haltezeit invalidiert und, wenn die Vor- und die Nachhaltebedingungen eingehalten sind, nach einer Verzögerungszeit den neuen gültigen Ausgabewert zuweist. Nur die Zeitparameter errechnen sich etwas anders. Die Vor- und die Nachhaltezeit der Gesamtschaltung sind die des Abtastregisters und die Halte- und die Verzögerungszeit sind jeweils die Summe der des Registers und der der Verarbeitungsschaltung.

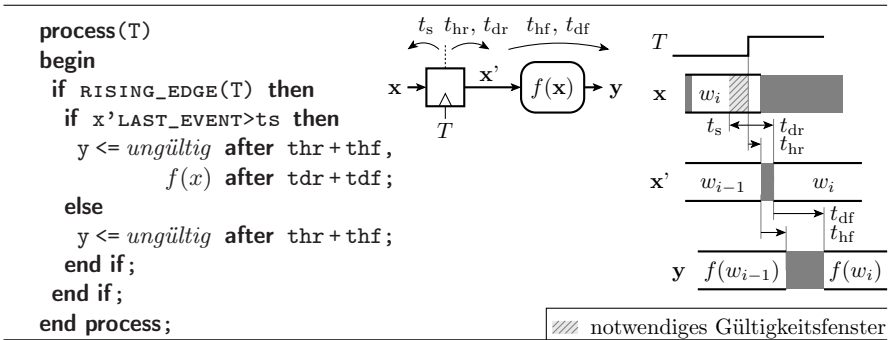


Abb. 1.49. Abtastung plus Verarbeitung

1.4.4 Register-Transfer-Funktion

Eine Register-Transfer-Funktion ist die Funktion einer kombinatorischen Schaltung, die zwischen Eingabe- und Ausgaberegistern eingebettet ist (Abb. 1.50 a). Die Eingaberegister übernehmen mit jeder aktiven Taktflanke neue Eingabewerte und die Ausgaberegister die neuen Ergebnisse. Für die Verarbeitung von der Eingabeübernahme bis zur Ergebnisübernahme steht genau die Dauer einer Taktperiode zur Verfügung (Abb. 1.50 b). Damit die Vorhalte- und die Nachhaltebedingungen des Ausgaberegisters erfüllt sind, muss gelten

$$T_P \geq T_{P\min} = t_{dr} + t_{df} + t_s \quad (1.7)$$

$$t_n \leq t_{n\max} = t_{hr} + t_{hf} \quad (1.8)$$

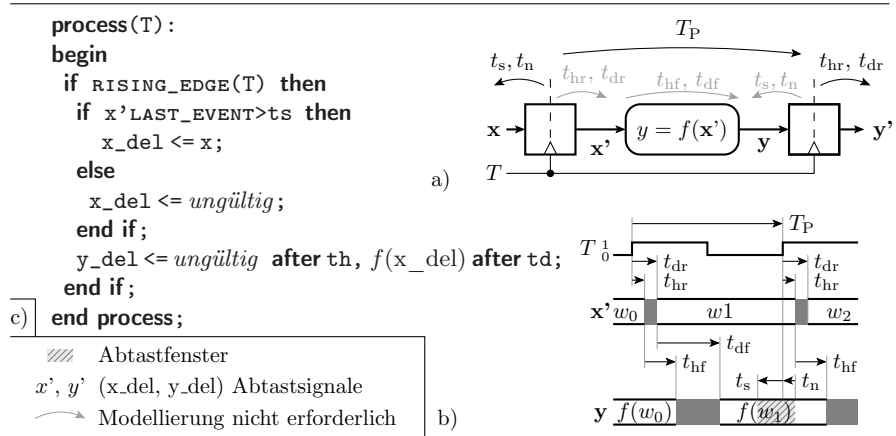


Abb. 1.50. Register-Transfer-Funktion a) Schaltungsstruktur b) Signalverläufe c) VHDL-Beschreibung

(T_P – Dauer einer Taktperiode; t_{dr} – Verzögerung des Eingaberegisters; t_{df} – Verzögerung der Verarbeitungsschaltung; t_s – Vorhaltezeit des Ergebnisregisters; t_n – Nachhaltezeit des Ergebnisregisters; t_{hr} – Haltezeit des Eingaberegisters; t_{hf} – Haltezeit der Verarbeitungsschaltung). Die Dauer einer Taktperiode darf nicht kleiner als die Summe der Verzögerungszeiten und der Vorhaltezeit des Ausgaberegisters sein und die Nachhaltezeit des Ausgaberegisters darf nicht größer als die Summe der Haltezeiten sein. Die maximal zulässige Taktfrequenz ist der Kehrwert der minimalen Taktperiode:

$$f_T \leq \frac{1}{T_{P\min}} = \frac{1}{t_{dr} + t_{df} + t_s} \quad (1.9)$$

Die minimale Taktperiode und die maximale Nachhaltezeit lassen sich mit einer Laufzeitanalyse berechnen. Wenn die Gleichungen 1.7 und 1.8 erfüllt sind, sind auch die Vorhalte- und Nachhaltebedingungen für die Ausgabeabtastung immer erfüllt. Ohne dass sich die fehlererkennenden Eigenschaften verschlechtern, kann die Simulation auf ihre Kontrolle verzichten. Für die Register-Transfer-Funktion genügt eine verzögerungsfreie Simulation ohne Berechnung der Signalgültigkeit. In einer synchronen Schaltung braucht die Simulation praktisch nur für Eingaberegister die Vorhalte- und die Nachhaltebedingungen zu kontrollieren und nur für Ausgaberegister und Ausgabefunktionen Halte- und Verzögerungszeiten zu berücksichtigen. Für die restlichen Schaltungsteile, die von Registern eingerahmt sind, genügt das viel einfachere Verhaltensmodell ohne Kontrollen und ohne Verzögerungen (Abb. 1.50 c).

1.4.5 Taktversatz

In der Schaltung in Abb. 1.51 besteht zwischen der Übernahme- und der Übernahmeflanke für das Eingaberegister und der Übernahmeflanke für das Ergebnisregister ein

konstanter Taktversatz. Der Taktversatz kann entweder vorsätzlich eingebaut oder von unbeabsichtigten Laufzeitunterschieden entlang der Taktleitungen verursacht sein.

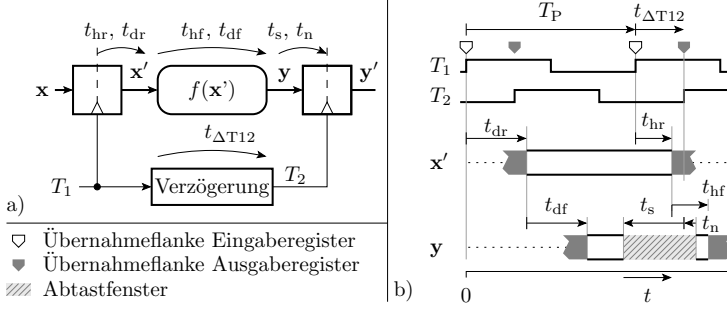


Abb. 1.51. Register-Transfer-Funktion mit Taktversatz a) Schaltung b) Signalverläufe

Damit die Vor- und die Nachhaltebedingungen in dieser Schaltung erfüllt sind, muss gelten:

- Die Summe der Verzögerungszeiten und der Vorhaltezeit muss kleiner als die Summe aus der Taktperiode und dem Taktversatz sein

$$t_{dr} + t_{df} + t_s \leq T_P + t_{\Delta T12} \quad (1.10)$$

- Die Summe der Haltezeiten muss größer als die Summe aus dem Taktversatz und der Nachhaltezeit sein

$$t_{hr} + t_{hf} \geq t_n + t_{\Delta T12} \quad (1.11)$$

($t_{\Delta T12}$ – Taktversatz zwischen den Takten T_1 und T_2). Aus Ungleichung 1.11 ergibt sich der maximal zulässige Taktversatz:

$$t_{\Delta T12} \leq t_{hr} + t_{hf} - t_n$$

Mit dem maximal zulässigen Taktversatz beträgt die minimale Taktperiode

$$T_{Pmin} = t_{dr} + t_{df} + t_s - (t_{hr} + t_{hf} - t_n)$$

und die maximal zulässige Taktfrequenz

$$f_T \leq \frac{1}{t_{dr} + t_{df} + t_s - (t_{hr} + t_{hf} - t_n)} \quad (1.12)$$

Sie ist der Kehrwert der Summe der Verzögerungszeiten, der Vorhaltezeit und der Nachhaltezeit des Ergebnisregisters abzüglich der Summe der Haltezeiten. Die maximal erzielbare Taktfrequenz für eine Register-Transfer-Funktion

hängt folglich nicht nur von den Verzögerungszeiten, sondern in gleichem Maße auch von den Haltezeiten ab. Bei einer Übertragung über eine reflexionsfreie Leitung ist z.B. die Haltezeit fast so groß wie die Verzögerungszeit. Modelliert als Register-Transfer-Funktion darf das Senderegister einer Leitungsübertragung das nächste Datenpaket lange, bevor das vorherige Datenpaket das Empfangsregister erreicht hat, losschicken. Bei einer typischen Ausbreitungsgeschwindigkeit von $10 \frac{\text{cm}}{\text{ns}}$ – das ist etwa ein Drittel der Lichtgeschwindigkeit – und einer Übertragungsrate von $10^9 \frac{\text{Bit}}{\text{s}}$ sind auf einer 10 m langen Datenleitung gleichzeitig bis zu 100 Datenbits unterwegs. Das Prinzip, gleichzeitig auf diese Weise mehrere Datensätze bearbeiten oder übertragen zu lassen, wird auch als Wellen-Pipeline bezeichnet.

1.4.6 Zusammenfassung und Übungsaufgaben

Eine kombinatorische Schaltung arbeitet nur dann zuverlässig, wenn ihre Ausgabesignale innerhalb ihrer Gültigkeitsfenster mit Registern abgetastet werden. Ein Register übernimmt seine Eingabewerte mit der aktiven Taktflanke und speichert sonst seinen Zustand. Ein Takt ist ein Speziaalsignal, das periodisch, Glitch-frei und mit hoher Zeitgenauigkeit zwischen »0« und »1« wechselt. Das Verhaltensmodell eines Registers ist ein Abtastprozess, der nur von einer Taktänderung geweckt wird und in dem bei einer aktiven Taktflanke, wenn das Eingangssignal im Abtastfenster stabil ist, der Wert des abzutastenden Signals übernommen, gespeichert und für die nachfolgenden Schaltungsteile bereitgestellt wird. Bei einer Verletzung der Vor- oder Nachhaltebedingung ist der gespeicherte Wert ungültig und darf nicht weiterverarbeitet werden.

Auch kombinatorische Funktionseinheiten mit Eingabeabtastung oder Ausgabeabtastung können, um Simulationsaufwand zu sparen, als Abtastprozesse modelliert werden. Eine eingangs- und ausgangsseitig mit Registern eingerahmte kombinatorische Funktionseinheit wird als Register-Transfer-Funktion bezeichnet. Das Besondere an einer Register-Transfer-Funktion ist, dass sich die Zeitbedingungen für eine zuverlässige Datenübernahme auch ohne Simulation kontrollieren lassen, so dass ein stark vereinfachtes Verhaltensmodell genügt.

Aufgabe 1.14

Beschreiben Sie die Schaltung in Abb. 1.52 a mit dem Zeitverhalten in Abb. 1.52 b durch einen Abtastprozess. Die Nachhaltezeit sei $t_n = 0$ und alle Signale sollen den Typ `STD_LOGIC` haben. Bei Verletzung der Vorhaltebedingungen ist der gespeicherte Wert zu invalidieren.

- a) Bestimmen Sie die Werte von t_1 bis t_7 des Zeittoleranzschemas in Abb. 1.52 b aus den Zeitparametern der Schaltung.

- b) Entwickeln Sie die VHDL-Beschreibung für den Prozess unter Nutzung der Ergebnisse aus Aufgabenteil a.

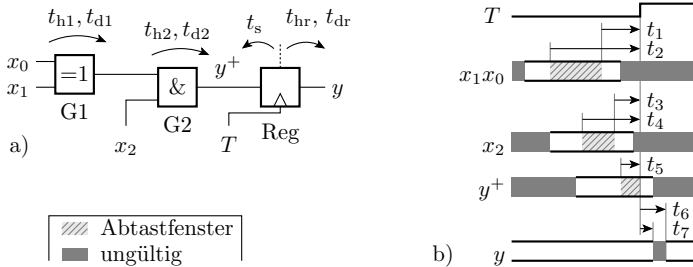


Abb. 1.52. a) Schaltung b) Zeitverhalten zu Aufgabe 1.14

Aufgabe 1.15

Abbildung 1.53 zeigt den Signalflussplan einer Register-Transfer-Funktion, die, wenn das höchste Operandenbit $x_{n-1} = 1$ ist, den Operanden x bitweise invertiert und eine »1« addiert.

- Welcher funktionale Zusammenhang besteht zwischen der Operandenbitbreite n und der maximalen Taktfrequenz der Schaltung?
- Mit welcher Taktfrequenz darf die Schaltung bei einer Operandenbreite $n = 16$ maximal betrieben werden?

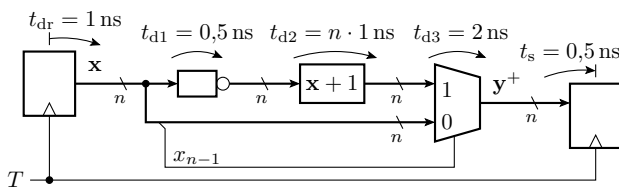


Abb. 1.53. Schaltung zu Aufgabe 1.15

Aufgabe 1.16

In welchem Zeitbereich muss der Taktversatz $t_{\Delta T12}$ in der Schaltung in Abb. 1.54 liegen, damit die Vorhalte- und die Nachhaltebedingung des Ergebnisregisters auch noch bei einer Taktfrequenz von $f_T = 100 \text{ MHz}$ erfüllt sind?

vorhersagbare Kombination aus Nullen und Einsen. Das kann auch ein unzulässiger, überhaupt nicht vorgesehener Bitvektorwert sein, der die Schaltung in einen Zustand versetzt, in dem sie nicht mehr funktioniert (Systemabsturz, siehe später Abschnitt 1.6.3). Der mittlere zeitliche Abstand zwischen den Fehlfunktionen, die die fehlende Abtastung verursacht, kann Sekunden, aber auch Tage oder Wochen betragen. Die ausgelösten Fehlfunktionen haben Zufallscharakter. Fehler, die sehr selten Fehlfunktionen verursachen, bleiben oft beim Test unerkannt und beeinträchtigen später im Einsatz die Zuverlässigkeit [29]. Die korrekte laufzeitrobuste Schaltung hat zusätzlich das in Abb. 1.56 grau unterlegt gezeichnete Abtastregister, das das asynchrone Eingabesignal vor der Weiterverarbeitung zum Takt ausrichtet. Nur so kann sichergestellt werden, dass auch y zu jedem Abtastzeitpunkt gültig ist.

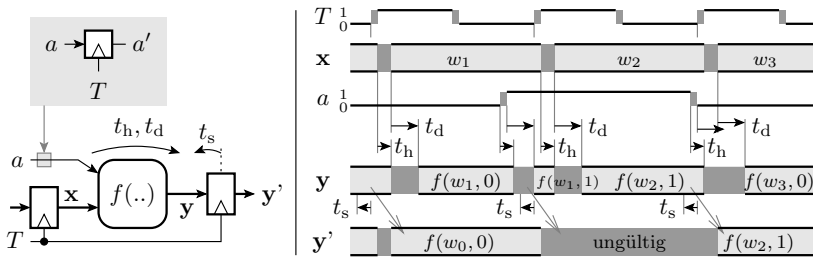


Abb. 1.56. Verarbeitung eines nicht abgetasteten Signals und Fehlerbeseitigung

1.5.2 Entprellen von Tasten

Manuell betätigte Tasten und Schalter liefern nicht nur asynchrone Eingabesignale. Sie prellen auch noch. Prellen bedeutet, dass die Schaltkontakte – ein Masse-Feder-System – bei Betätigung mechanisch schwingen und möglicherweise mehrfach öffnen und schließen. Die Prelldauer ist deutlich kürzer als die minimale Betätigungsdauer und liegt im Millisekundenbereich.

Als asynchrone Signale müssen Tastensignale vor ihrer Weiterverarbeitung abgetastet werden. Die Entprellung erfolgt im einfachsten Fall mit einer Abtastperiode, die größer als die maximale Prelldauer und kürzer als die minimale Betätigungsdauer ist (Abb. 1.57 a). Mit dieser Abtastperiode ist garantiert, dass bei jedem Ein-Aus-Wechsel das Tastensignal höchstens einmal während der Zeit, in der die Taste möglicherweise prellt, abgetastet wird. Der Abtastwert einer prellenden Taste kann »0« oder »1« sein. Vorher hat er garantiert den einen und nachher garantiert den anderen Wert, so dass das abgetastete Signal nach jeder Tastenbetätigung garantiert nur eine Schaltflanke hat. In Abb. 1.57 b wird das abgetastete Eingabesignal nochmal abgetastet, um ein Signal y zu bilden, das nach jeder fallenden Flanke des abgetasteten Eingabesignals für die Dauer eines Taktes »1« und sonst »0« ist.

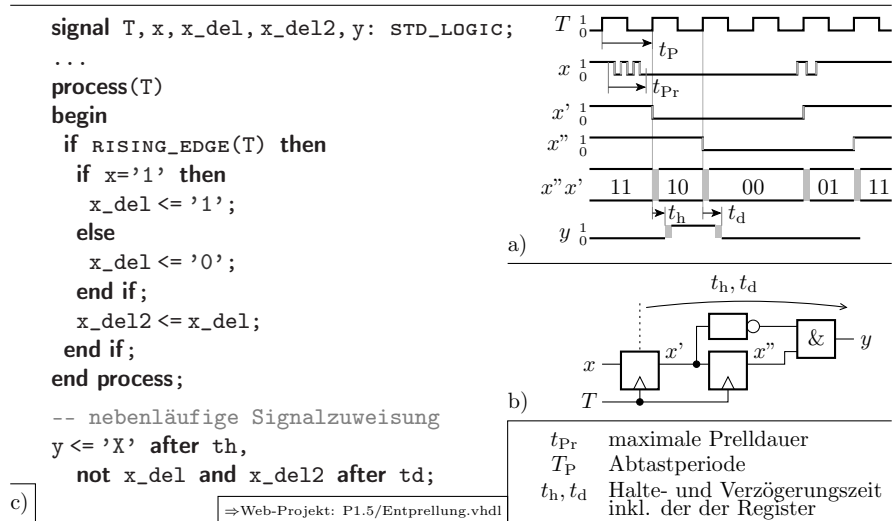


Abb. 1.57. Entprellen und Flankenerkennung für eine Tasteneingabe a) Schaltung b) Signalverläufe c) VHDL-Beschreibung

In der VHDL-Beschreibung in Abb 1.57c werden die beiden Register im selben Abtastprozess durch je eine Signalzuweisung beschrieben. Bei der Abtastung eines binären asynchronen Signals ist der Abtastwert immer »0« oder »1«, aber nicht »X«. Im Programmbeispiel wird Abtastwerten ungleich »1« eine »0« zugeordnet. Zutreffender wäre eine zufällige Auswahl von »0« oder »1. Die zweite Abtastung ist eine Register-Transfer-Funktion und wird als solche ohne Zeitverhalten modelliert, da sich die Einhaltung der Zeitbedingungen auch ohne Simulation überprüfen lassen. Die bei den Signalzuweisungen im Abtastprozess unberücksichtigten Registerhalte- und Registerverzögerungszeiten seien in der Halte- und der Verzögerungszeit der kombinatorischen Ausgabefunktion, die als nebenläufige Signalzuweisung beschrieben ist, mit eingerechnet.

1.5.3 Asynchrone Initialisierung

Die einzelnen Bits eines Registers haben nach Zuschalten der Versorgungsspannung und beim Start einer Simulation einen unbestimmten Anfangswert. Bei der Modellierung mit dem vom VHDL-Standard empfohlenen Bit-typ STD_LOGIC ist das der symbolische Wert für nicht initialisiert »U« (uninitialized, vgl. Abschnitt 1.2.1). Bevor der Registerinhalt ausgewertet werden darf, muss ein definierter Wert in das Register geschrieben werden. Dazu haben Register oft einen zusätzlichen Initialisierungseingang, der, wenn er aktiviert wird, für jedes Bit einen definierten Wert einstellt. Abbildung 1.58 zeigt das erweiterte Datenflusssymbol und ein vereinfachtes Verhaltensmodell

ohne Berücksichtigung von Verzögerungen. Der Registerprozess hat, damit er auch zur Initialisierung geweckt wird, zusätzlich das Initialisierungssignal in der Weckliste. Bei einer Aktivierung des Initialisierungssignals wird der Anfangswert – eine vordefinierte Konstante aus Nullen und Einsen – und sonst bei jeder aktiven Taktflanke der Eingabewert zugewiesen.

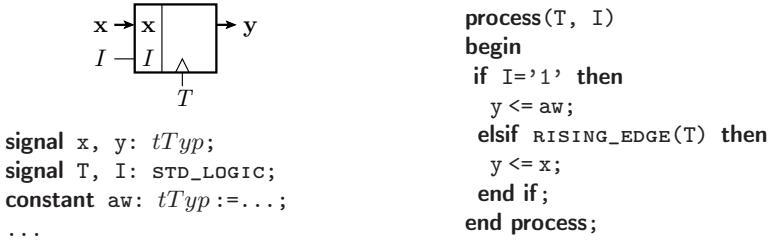


Abb. 1.58. Register mit Initialisierungseingang (*tTyp* – beliebiger in Hardware darstellbarer Datentyp)

Die Initialisierung eines Registers ist laufzeitkritisch. Abbildung 1.59 a zeigt das am Beispiel einer Schaltung mit einem initialisierbaren Eingaberegister und einem einfachen Ausgaberegister. Wenn das Initialisierungssignal zu kurze Zeit aktiv ist, liegt, selbst wenn das Eingaberegister seinen Anfangswert korrekt übernimmt, der Initialwert am Registerausgang möglicherweise zu kurze Zeit stabil an, so dass das nachfolgende Register im ersten Schritt einen ungültigen Wert übernimmt (Abb. 1.59 b). Die in Abb. 1.59 c und 1.59 d gezeigten Fehlersituationen unterstellen, dass das Register aus Master-Slave-Flipflops besteht (siehe später Abschnitt 4.3.5). Ein Master-Slave-Flipflop besteht aus einer Master- und einer Slave-Speicherzelle. Der Master übernimmt

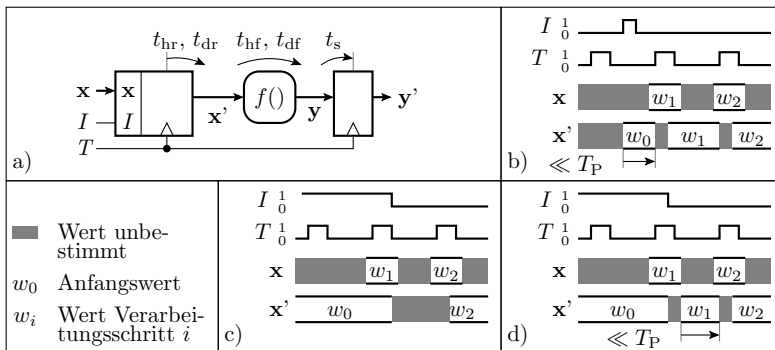


Abb. 1.59. Registerinitialisierung a) Schaltung mit initialisierbarem Eingaberegister b-d) mögliche Fehlfunktionen ohne Abtastung

den Eingabewert in der Taktphase vor der aktiven Taktflanke und gibt ihn in der Taktphase nach der aktiven Flanke an den Slave weiter. Wenn das Initialisierungssignal, das im Slave den Anfangswert einstellt, in der Übernahmephase des Slaves deaktiviert wird, übernimmt der Registerausgang in diesem Moment den Wert, der vor der letzten aktiven Flanke in den Master übernommen wurde. Dieser Wert liegt dann am Registerausgang kürzer als eine Taktperiode an (Abb. 1.59 c). Dadurch ist es möglich, dass das nachfolgende Register im zweiten Arbeitsschritt einen ungültigen Wert übernimmt. Bei einer etwa zeitgleichen Deaktivierung des Initialisierungssignals mit der Deaktivierung des Slave-Übernahmesignals entscheiden geringfügige Laufzeitunterschiede darüber, welche Slaves eines Registers den Initialisierungswert beibehalten und welche den Wert aus ihrem Master übernehmen. In diesem Fall ist der Ausgabewert des Registers bis zur nächsten aktiven Taktflanke unbestimmt und damit ungültig (Abb. 1.59 d).

Zur Vermeidung von Initialisierungsfehlern ist das Initialisierungssignal abzutasten. Abbildung 1.60 zeigt eine typische Initialisierungsschaltung mit einer Reset-Taste und einer Power-On-Reset-Logik. Die Reset-Taste liefert ein asynchrones low-aktives Signal. Die Power-On-Reset-Logik ist hier eine Schaltung, die nach Zuschalten der Versorgungsspannung ein low-aktives Initialisierungssignal mit einem Kondensator noch für eine gewisse Zeit auf »0« hält. Auch dieses Signal ist asynchron. Beide Signale werden invertiert, ODER-verknüpft, abgetastet und auf die Initialisierungseingänge der internen Register geführt. Die abgetastete ODER-Verknüpfung ist ein Bitsignal, das nur »0« oder »1« sein kann. Sein Wert ändert sich nur mit der aktiven Taktflanke. Damit steht den Register-Transfer-Funktionen hinter den initialisierbaren Registern auch während der Initialisierung und im ersten Schritt nach der Initialisierung die komplette Taktperiode zur Verfügung.

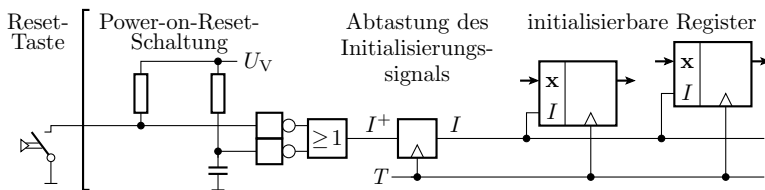


Abb. 1.60. Erzeugung eines abgetasteten Initialisierungssignals

1.5.4 Asynchrone parallele Schnittstelle

Asynchrone parallele Eingabedaten haben immer einen zufälligen Zeitversatz zum Empfängerakt. Um sie verarbeiten zu können, muss ihr Gültigkeitsfenster mitübertragen oder aus den empfangenen Signalverläufen bestimmt

werden. Im einfachsten Fall wird der Takt der Signalquelle, an dem die Daten ausgerichtet sind, mitgesendet.

In Abb. 1.61 liefert die Signalquelle einen n -Bit-Datenvektor $\mathbf{x}$ und einen Sendetakt T_x mit der halben Frequenz, mit der der Sender die Daten bereitstellt. Durch die Halbierung werden mit beiden Taktflanken Daten übertragen. Das Gültigkeitsfenster soll immer mit der Taktflanke beginnen und mindestens eine Länge t_g haben. Der Empfänger muss alle empfangenen Signale, d.h. sowohl die Datensignale als auch den Übertragungstakt, abtasten, um sie zeitlich an seinem eigenen Takt auszurichten. Der Zeitversatz zwischen den Flanken des Abtasttaktes T und den Taktflanken von T_x ist eine Zufallsgröße, die zwischen null und der Periodendauer T_P des Abtasttaktes liegt. Um sicherzustellen, dass jeder übertragene Datenwert mindestens einmal in seinem Gültigkeitsfenster abgetastet wird, darf die Periode des Abtasttaktes nicht kürzer als die Länge des Gültigkeitsfensters der empfangenen Daten sein:

$$T_P < t_g \quad (1.13)$$

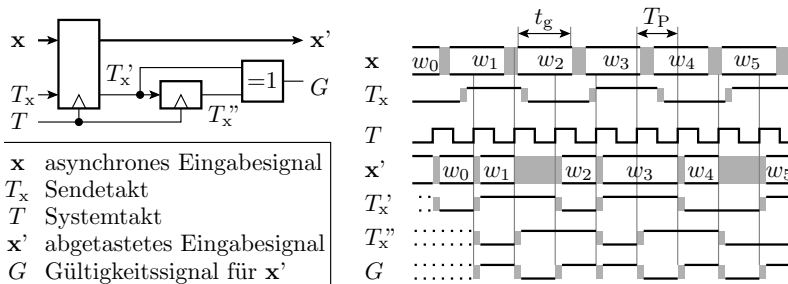


Abb. 1.61. Schnittstellenschaltung für die asynchrone parallele Eingabe

Nach der Abtastung wird anhand des Abtastwertes von T_x entschieden, ob der abgetastete Datenwert neu und gültig war. Der Abtastwert von T_x ist immer »0« oder »1«. Gültigkeitsbedingung für die abgetasteten Datenwerte ist, dass sich der aktuelle Abtastwert von T_x vom vorherigen, d.h. von dem doppelt abgetasteten Wert unterscheidet. Die Funktion zur Bildung des Gültigkeitssignals G ist eine EXOR-Verknüpfung.

Abbildung 1.62a zeigt eine VHDL-Beschreibung für die Schnittstellenschaltung aus Abb. 1.61. Die Eingabedaten und der Sendetakt werden ohne Kontrolle der Vorhalte- und der Nachhaltebedingungen abgetastet. Für den Sendetakt wird der Wertebereich der Abtastwerte wie in Abb. 1.57b auf »0« und »1« begrenzt. Die EXOR-Verknüpfung zur Bildung des Gültigkeitssignals ist durch eine nebenläufige Signalzuweisung beschrieben.

Nach einer Schnittstellenschaltung folgt üblicherweise eine kombinatorische Verarbeitungsschaltung mit Ausgabeabtastung, die zusammen mit der

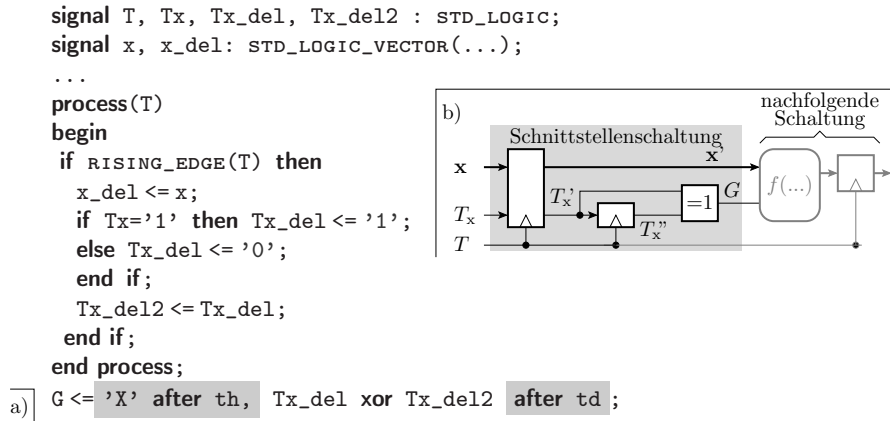


Abb. 1.62. a) VHDL-Beschreibung der asynchronen parallelen Schnittstellenschaltung aus Abb. 1.61 b) typische Schaltungseinbindung

EXOR-Verknüpfung eine Register-Transfer-Funktion bildet (Abb. 1.62 b). In diesem Fall kann das Gültigkeitssignal G auch ohne die in Abb. 1.62 a grau unterlegten Beschreibungsbestandteile für das Zeitverhalten simuliert werden.

1.5.5 Zusammenfassung und Übungsaufgaben

Die Eingabedaten einer digitalen Schaltung verhalten sich asynchron zum Schaltungstakt. Sie müssen vor ihrer Verarbeitung mit Registern abgetastet und bei mechanischen Eingabeelementen zusätzlich entprellt werden. Zum Entprellen muss die Abtastperiode größer als die maximale Prelldauer und kleiner als die minimale Betätigungsdauer sein. Auch asynchrone Initialisierungssignale sind, bevor sie auf die Initialisierungseingänge der Register geführt werden dürfen, abzutasten. Zur Übernahme asynchroner Daten von einer anderen Schaltung sind die empfangenen Daten gleichfalls zuerst abzutasten. Die Gültigkeit der empfangenen Daten wird rückwirkend aus den Abtastwerten bestimmt.

Aufgabe 1.17

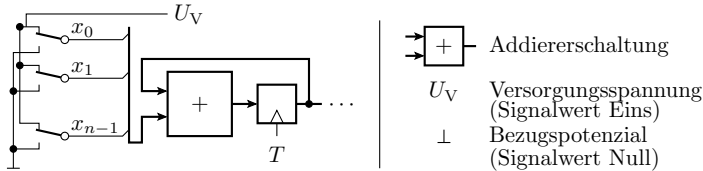
Für ein Gerät mit zwei mechanischen Tasten mit einer maximalen Prelldauer von $t_{Pr} = 20 \text{ ms}$ und einer Betätigungsdauer größer $0,2 \text{ s}$ ist eine Eingabeschaltung zu entwickeln, die beim Drücken von genau einer Taste i ein Ausgangssignal y_i erzeugt, das an der aktiven Taktflanke ausgerichtet und bei jeder Tastenbetätigung genau einen Takt lang »1« ist. Die Tasten-Eingabesignale seien low-aktiv (gedrückt »0«, nicht gedrückt »1«).

- a) In welchem Bereich muss die Taktfrequenz der Schaltung liegen?

- Entwerfen Sie eine Schaltung mit Tasten, Pull-Up-Widerständen¹¹, Gattern und Registern.
- Beschreiben Sie den Abtastprozess und die Ausdrücke zur Bildung der Ausgabewerte in VHDL.

Aufgabe 1.18

Im nachfolgenden Schaltungsausschnitt wird zum Inhalt eines Registers eine über mehrere Schalter einstellbare Konstante addiert.



- Unter welcher Bedingung kann es zu einem unerwarteten Fehlverhalten kommen?
- Wie ist die Schaltung zu verändern, damit sie zuverlässig arbeitet?

Aufgabe 1.19

Bei einer asynchronen Übertragung seien die übertragenen Datenworte des Signals $\mathbf{x}$ immer dann gültig, wenn das mitübertragene Gültigkeitssignal $g \neq 0$ ist (Abb. 1.63).

- Mit welcher Taktfrequenz muss die Empfangsschaltung die beiden Signale abtasten?
- Wie ist das Gültigkeitssignal für die abgetasteten Datenworte zu bilden, damit es für jedes abgetastete gültige Datenwort genau einen Takt lang aktiv ist?

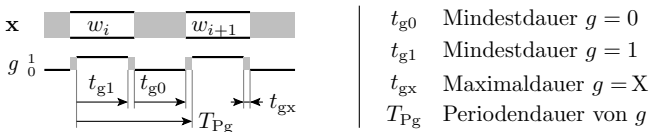


Abb. 1.63. Signalverlauf zu Aufgabe 1.19

¹¹ Ein Pull-Up-Widerstand verbindet das Signal mit der Versorgungsspannung und zieht den Signalwert bei nicht betätigter Taste auf »1« (siehe später Abschnitt 4.1.6).

1.6 Sequenzielle Schaltungen

Definition 1.18 (Sequenzielle Schaltung) *Eine sequenzielle Schaltung (engl. sequential circuit, im deutschen Sprachraum auch Schaltwerk) ist eine digitale Schaltung mit Speicherverhalten, die in diskreten Zeitschritten aus aktuellen Zuständen und Eingaben Folgezustände und Ausgaben bildet.*

Definition 1.19 (Zustandsgraph) *Ein Zustandsgraph ist eine Verhaltensbeschreibung für eine sequenzielle Schaltung, die den gespeicherten Zustand durch Knoten und die Zustandsübergänge durch Kanten darstellt. In der Grundform sind die Übergangsbedingungen Mengen von Eingabewerten und die gesteuerten Aktionen Ausgabewerte.*

Definition 1.20 (Operationsablaufgraph) *Ein Operationsablaufgraph ist ein erweiterter Zustandsgraph, der zusätzlich Operationen steuert und in den Übergangsbedingungen der Kanten Operationsergebnisse auswertet.*

Definition 1.21 (Zustandscodierung) *Die Zustandscodierung ist die Festlegung der Bitvektorkonstanten, mit denen die Zustände eines Zustandsgraphen oder eines Operationsablaufgraphen in der Schaltung dargestellt werden.*

Definition 1.22 (Software-orientierte Ablaufbeschreibung) *Eine Software-orientierte Ablaufbeschreibung beschreibt Operationsabläufe mit den in höheren Programmiersprachen gebräuchlichen Kontrollflusselementen Sequenz, Fallunterscheidung und Schleife sowie mit Warteanweisungen zum Weitschalten der Simulationszeit.*

Sequenziell bedeutet zeitlich nacheinander. Eine sequenzielle Schaltung berechnet ihre Ausgaben in mehreren Zeitschritten. Sie besteht aus kombinatorischen Schaltungen für die eigentliche Verarbeitung und aus Registern für die Abtastung und Aufbewahrung der Zwischenergebnisse für die nachfolgenden Berechnungsschritte.

1.6.1 Endlicher Automat

Ein endlicher Automat ist ein allgemeines Funktionsmodell zur Beschreibung sequenzieller Abläufe. Er definiert einen sequenziellen Ablauf durch

- endliche Wertebereiche für die Eingabe, den Zustand und die Ausgabe,
- einen Startzustand,
- eine Übergangsfunktion und
- eine Ausgabefunktion.

Die Wertebereiche sind Symbolmengen:

- Eingabemenge $\Sigma = \{E_1, E_2, \dots\}$
- Zustandsmenge $S = \{Z_1, Z_2, \dots\}$

- Ausgabemenge $\Pi = \{A_1, A_2, \dots\}$

Die Übergangsfunktion beschreibt die Abbildung der Eingabemenge und der Zustandsmenge auf die Zustandsmenge

$$f_s : \Sigma \times S \rightarrow S$$

und die Ausgabefunktion die Abbildung der Eingabemenge und der Zustandsmenge auf die Ausgabemenge

$$f_a : \Sigma \times S \rightarrow \Pi.$$

In einer digitalen Schaltung werden die Eingabe-, die Zustands- und die Ausgabemenge durch Bitvektoren dargestellt. Die Anzahl der mit einem Bitvektor darstellbaren Werte gehorcht Ungleichung 1.1:

$$N \leq 2^n$$

(N – Anzahl der darstellbaren Werte; n – Bitanzahl). Sie ist eine Zweierpotenz. Die Anzahl der darstellbaren Werte ist dabei in der Regel größer als die Anzahl der darzustellenden Werte. Die ungenutzten Bitvektorstellen dürfen im fehlerfreien Betrieb nicht auftreten. Anderenfalls ist das Verhalten nicht festgelegt. Die Übergangs- und die Ausgabefunktion werden durch kombinatorische Schaltungen nachgebildet. Der Zustand benötigt ein Register zur Speicherung. Der Startzustand ist der Initialisierungswert des Zustandsregisters. Das Zustandsregister richtet alle Zustandsübergänge an der aktiven Taktflanke aus.

Zustandsgraph

Die Funktion eines Automaten wird gern in der Form eines Zustandsgraphen spezifiziert und beschrieben. Ein Zustandsgraph stellt die Zustände als Knoten und die Zustandsübergänge als Kanten dar (Abb. 1.64). Jeder Zustand hat eine Bezeichnung – eine Nummer, eine Bitvektorkonstante oder einen

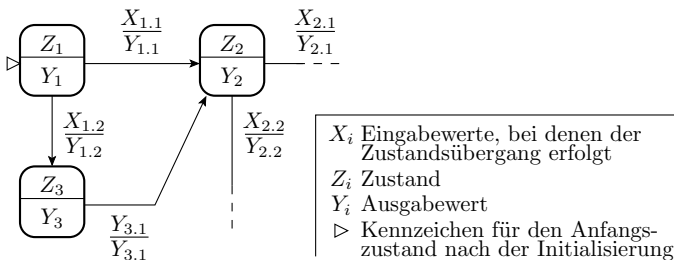


Abb. 1.64. Zustandsgraph

symbolischen Namen. Der Startzustand wird mit einem kleinen Dreieck gekennzeichnet. Die Kanten sind gerichtet und mit den Übergangsbedingungen beschriftet. In der Grundform ist die Übergangsbedingung die Menge der Eingabewerte, bei denen der Übergang erfolgen soll. Keine Angabe bedeutet, dass der Übergang bei jeder Eingabe erfolgen soll. In erweiterten Modellen kann die Übergangsbedingung ein beliebiger logischer Ausdruck sein. Die Ausgabe-werte werden entweder den Knoten oder den Kanten zugeordnet. Die Ausgabe in einem Knoten gilt für alle Kanten, die den Knoten verlassen. Wenn einem Knoten eine Ausgabe zugeordnet ist, dann darf den abgehenden Kanten keine Ausgabe zugeordnet sein.

Endliche Automaten werden unterteilt in Mealy-Automaten, Moore-Automaten und autonome Automaten. Ein Mealy-Automat ist ein allgemeiner Automat mit mehreren abgehenden Kanten je Zustand, denen zumindest zum Teil unterschiedliche Ausgabewerte zugeordnet sind. Die Ausgabe ist damit von der Eingabe abhängig. Abbildung 1.65 a zeigt als Beispiel den Zustands-graphen für einen Vorwärts-Rückwärts-Zähler, der bei der Eingabe »V« zyklisch die Zustandsfolge »A-B-C-D« und bei der Eingabe »R« die umgekehrte Zustandsfolge »A-D-C-B« durchläuft. Bei dem dritten Wert aus der Eingabe-menge »H« behält der Automat seinen aktuellen Zustand bei. Jedem Zu-standsübergang ist ein Ausgabewert zugeordnet.

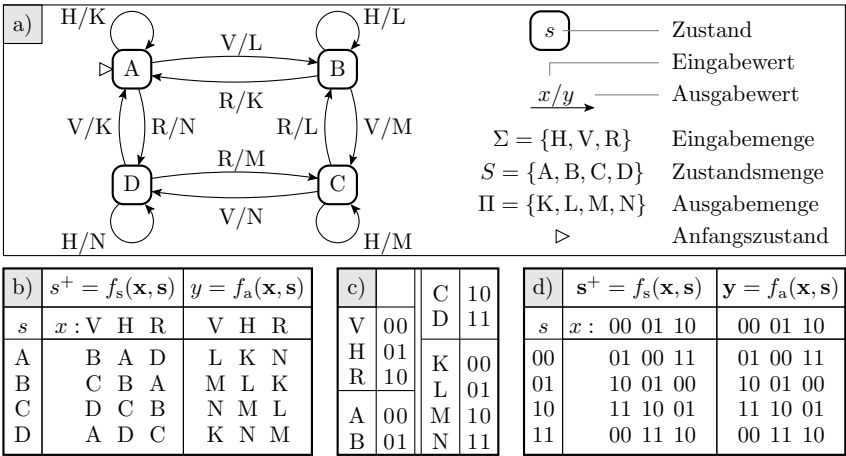


Abb. 1.65. Vorwärts-Rückwärts-Zähler als Mealy-Automat a) Zustandsgraph b) tabellarische Übergangs- und Ausgabefunktion mit symbolischen Werten c) Codierung d) bitorientierte Übergangs- und Ausgabefunktion

Die Übergangsfunktion und die Ausgabefunktion lassen sich als Tabellen darstellen. Die Zeilen repräsentieren die Ist-Zustände, die Spalten die Eingabewerte und die Einträge die zugeordneten Folgezustände und Ausgabewerte (Abb. 1.65 b). Die Werte sind wie im Zustandsgraph Symbole, die im nächs-

ten Entwurfsschritt durch Bitvektorkonstanten ersetzt werden (Zustandscodierung). Die Unterscheidung von drei bzw. vier Werten verlangt nach Gleichung 1.2 mindestens 2 Bit. Abbildung 1.65c zeigt eine Beispielcodierung und Abb. 1.65d die Tabellen mit der Übergangs- und der Ausgabefunktion, in denen die Symbole durch die zugeordneten Bitvektoren ersetzt sind. In Abschnitt 2.2.4 wird der hier begonnene Entwurf bis zu einer Schaltung aus Logikgattern und Speicherzellen weitergeführt.

Ein Moore-Automat hat eine einfachere Ausgabefunktion. Die Ausgabe-werte sind den Knoten zugeordnet und damit nicht von der Eingabe abhängig. Der Moore-Automat in Abb. 1.66 hat dieselbe Übergangsfunktion und dieselbe Ausgabemenge wie der Mealy-Automat in Abb. 1.65. Nur sind die Ausgabewerte den Zuständen zugeordnet, so dass die Tabelle der Ausgabewerte mit einer Spalte auskommt.

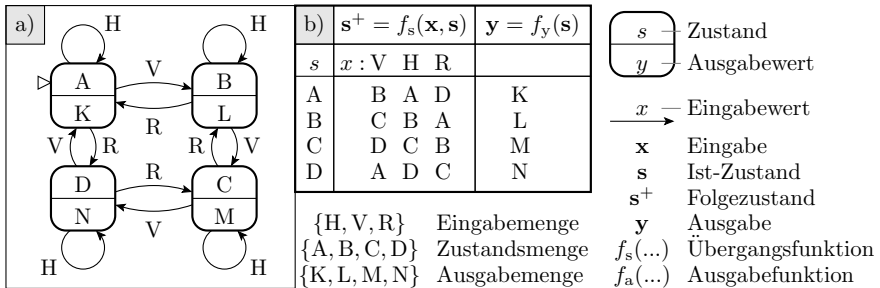


Abb. 1.66. Moore-Automat a) Zustandsgraph b) tabellarische Übergangs- und Ausgabefunktion

Ein weiterer Sonderfall ist ein autonomer Automat. Das ist ein Automat ohne Eingabe. Bei einem autonomen Automaten hat jeder Zustand genau einen Nachfolger. Die Zustände werden entweder zyklisch oder einmalig durchlaufen (Abb. 1.67). Die Ausgabe ist den Zuständen zugeordnet. Wenn im Zustandsgraphen eines Moore-Automaten oder eines autonomen Automaten den Zuständen keine Ausgaben zugeordnet sind, gilt der Zustand gleichzeitig als Ausgabe.

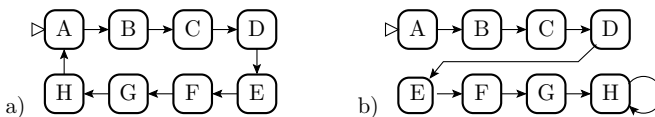


Abb. 1.67. Zustandsgraphen autonomer Automaten a) zyklisch b) zyklensfrei

1.6.2 Vom Zustandsgraph zur VHDL-Beschreibung

In einer digitalen Schaltung wird die Übergangsfunktion und die Ausgabefunktion je durch eine kombinatorische Schaltung und der Zustand durch ein Register nachgebildet (Abb. 1.68). Der Startzustand ist der Anfangswert des Zustandsregisters. Er wird über ein spezielles Initialisierungssignal eingestellt (vgl. Abschnitt 1.5.3). Damit die Funktion der Schaltung nicht von Laufzeiten abhängt, müssen das Eingabe- und das Initialisierungssignal zeitlich an der aktiven Taktflanke ausgerichtet sein. Die Register hierfür zählen nicht zur Schaltung des Automaten. Die kombinatorischen Schaltungen für die Übergangs- und die Ausgabefunktion bilden ständig mit einer gewissen Zeitverzögerung aus dem Eingangssignal x und dem Ist-Zustand s den Folgezustand s^+ und den Wert des Ausgabesignals y . Der Folgezustand wird mit der nächsten aktiven Taktflanke in das Zustandsregister als Ist-Zustand übernommen. Das Ausgabesignal y wird entweder direkt ausgegeben oder zur Verbreiterung des Gültigkeitsfensters mit einem Ausgaberegister abgetastet. Auch das Ausgaberegister wird nicht zum Funktionsmodell des Automaten gezählt.

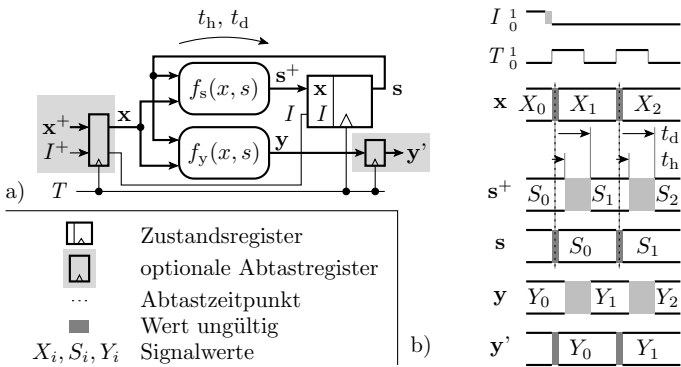


Abb. 1.68. Nachbildung eines Automaten durch eine digitale Schaltung a) Schaltungsstruktur b) Zeitverhalten

Die VHDL-Beschreibungsmittel für dieses Modell sind Abtastprozesse und kombinatorische Prozesse. Die Übergangsfunktion ist praktisch eine Register-Transfer-Funktion. Nach den Gleichungen 1.7 und 1.8 muss die Taktperiode größer als die Summe des Maximums der Verzögerung des Zustands- und des Eingaberegisters plus der Verzögerung der Schaltung für die Übergangsfunktion plus der Vorhaltezeit des Zustandsregisters sein. Die Summe der Haltezeiten muss größer als die Nachhaltezeit des Zustandsregisters sein (vgl. Abschnitt 1.4.4). Für die Ausgabefunktion zwischen dem Eingabe- und dem Zustandsregister am Eingang und dem Ausgaberegister am Ausgang gelten analoge Beziehungen. Das Eingabe-Ausgabe-Verhalten der beiden kombina-

torischen Schaltungen wird vorzugsweise mit VHDL-Auswahlanweisungen beschrieben.

Die VHDL-Auswahlanweisung

Eine Auswahlanweisung wertet einen Ausdruck aus. In Abhängigkeit von seinem Wert werden unterschiedliche Anweisungsfolgen abgearbeitet (Abb. 1.69). Auswahlwerte, für die dieselben Anweisungen abgearbeitet sind, können mit dem Symbol »|« zusammengefasst werden. Der Auswahlwert »others« steht für alle anderen Auswahlwerte.

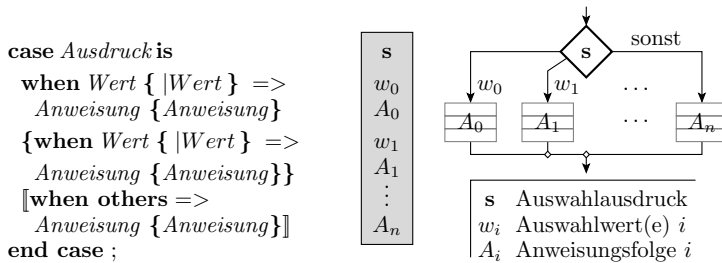


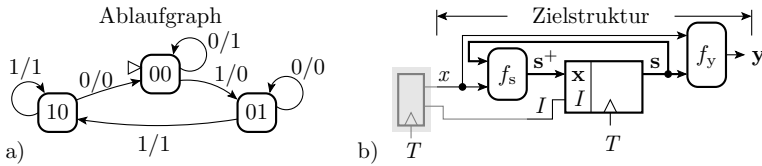
Abb. 1.69. Auswahlanweisung

Zur Nachbildung eines Automaten ist der Ausdruck für die Auswahl entweder das Zustandssignal oder das Eingangssignal oder eine Verknüpfung beider Signale. Die auszuführenden Anweisungen für die einzelnen Auswahlfälle sind entweder eine weitere Fallunterscheidung oder die Zuweisung eines konstanten Wertes an das Zustands- oder das Ausgabesignal.

Eine Beispielbeschreibung für einen Mealy-Automaten

Abbildung 1.70 zeigt den Zustandsgraph und die Zielstruktur des Beispielautomaten. Die Eingabemenge umfasst nur die Werte »0« und »1« und die Zustandsmenge die Bitvektorstufen »00«, »01« und »10«. Das Zustandsregister besitzt einen Initialisierungseingang zur Einstellung des Startzustands, der, wie aus dem Zustandsgraphen ersichtlich ist, »00« sein soll.

Für das Eingabe- und das Initialisierungssignal ist unterstellt, dass sie vor dem Automaten mit dem Takt des Automaten abgetastet werden. Die Übergangsfunktion ist entsprechend eine Register-Transfer-Funktion, die bis zu ihrer maximalen Taktfrequenz nach Gleichung 1.9 lauffest ist. Eine Kontrolle des Zeitverhaltens ist im Simulationsmodell nicht erforderlich. Die Standardbeschreibungsschablone ist ein Abtastprozess mit dem Takt und dem Initialisierungssignal in der Weckliste (Abb. 1.71). Wenn der Prozess aufwacht und das Initialisierungssignal aktiv ist, wird dem Zustandssignal der Anfangswert »00« zugewiesen, sonst bei einer aktiven Taktflanke der Folgezustand.



```

signal x, y, T, I: STD_LOGIC;
signal s: STD_LOGIC_VECTOR(1 downto 0);

```

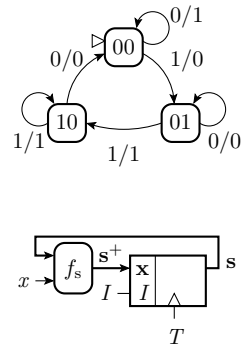
Abb. 1.70. Mealy-Automat a) Zustandsgraph b) Zielstruktur

Der Folgezustand ergibt sich dabei jeweils aus dem aktuellen Zustand und der Eingabe. In der Prozessbeschreibung in Abb. 1.71 ist der Auswahl Ausdruck eine Konkatenation (Verkettung) des Zustands- und des Eingangssignals zu einem gemeinsamen Bitvektor. So genügt eine Fallunterscheidung. Alle Auswahlwerte mit demselben Folgezustand sind zu einem Auswahlfall zusammengefasst. Alle anderen im Zustandsgraph nicht berücksichtigten Fälle – ungültiger Ist-Zustand, ungültige Eingabe etc. – erfasst der Sonst-Zweig, der dem Folgezustand den Pseudo-Wert »ungültig« zuweist. Ein ungültiger Zustand darf nicht weiterverarbeitet werden bzw. signalisiert eine Fehlfunktion.

```

process(I, T)
  variable v: STD_LOGIC_VECTOR(2 downto 0);
begin
  if I='1' then
    s <= "00";
  elsif RISING_EDGE(T) then
    v := s & x;
    case v is
      when "00"&'0' | "10"&'0' => s <= "00";
      when "01"&'0' | "00"&'1' => s <= "01";
      when "10"&'1' | "01"&'1' => s <= "10";
      when others => s <= "XX";
    end case;
  end if;
end process;

```



⇒ Web-Projekt: P1.6/BspMealy.vhdl

Abb. 1.71. Abtastprozess zur Beschreibung der Übergangsfunktion und des Zustandsregisters für den Mealy-Automaten in Abb. 1.70

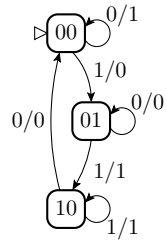
Die Ausgabefunktion ist als kombinatorischer Prozess beschrieben (Abb. 1.72). Ihre Eingangssignale – das sind der Ist-Zustand s und das Eingangssignal des Automaten x – stehen in der Weckliste. Der Ausgabewert wird auch hier mit einer Auswahlanweisung mit einer Konkatenation des Eingabewertes und des Zustands als Auswahl Ausdruck gebildet. Alle Kanten mit dem

Ausgabewert »0« und alle Kanten mit dem Ausgabewert »1« sind zu einem Auswahlfall zusammengefasst. Der Sonst-Fall erfasst alle unzulässigen Eingabewerte und Zustände und weist der Ausgabe den Pseudo-Wert »ungültig« zu.

```

process(x, s)
  variable v: STD_LOGIC_VECTOR(2 downto 0);
begin
  v := s & x;
  case v is
    when "00"&'1' | "01"&'0' | "10"&'0' => y <= '0';
    when "00"&'0' | "01"&'1' | "10"&'1' => y <= '1';
    when others => y <= 'X';
  end case;
end process;

```



⇒ Web-Projekt: P1.6/BspMealy.vhdl

Abb. 1.72. Beschreibung der Ausgabefunktion zu Abb. 1.70

Moore-Automat

Bei einem Moore-Automaten sind die Ausgabewerte den Zuständen zugeordnet. Der Beispielautomat in Abb. 1.73 hat dieselbe Zyklusstruktur wie der Mealy-Automat in Abb. 1.70. Die Übergangsfunktion ist somit identisch. Die Ausgabefunktion wertet nur den Zustand des Automaten aus. In der Weckliste des beschreibenden Prozesses steht entsprechend nur das Zustandssignal. Für den Zustand »01« wird eine »0«, für die Zustände »00« und »10« eine »1« und sonst, wenn der Zustand keinen zulässigen Wert hat, wird der Pseudo-Wert »ungültig« an das Ausgabesignal zugewiesen.

```

signal x, y, T, I: STD_LOGIC;
signal s: STD_LOGIC_VECTOR(1 downto 0);
...
-- kombinatorischer Ausgabeprozess
process(s)
begin
  case s is
    when "01"      => y <= '0';
    when "00" | "10" => y <= '1';
    when others    => s <= 'X';
  end case;
end process;

```

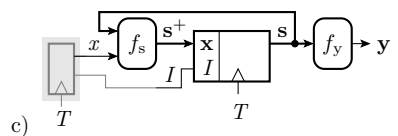
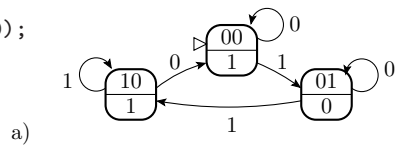


Abb. 1.73. Moore-Automat mit der Übergangsfunktion des Mealy-Automaten in Abb. 1.70 a) Zustandsgraph b) VHDL-Beschreibung der Ausgabefunktion c) Zielstruktur

Autonomer Automat

Das Beispiel für einen autonomen Automaten sei der 4-Bit-Johnson-Zähler in Abb. 1.74. Ein Johnson-Zähler ist ein rückgekoppeltes Schieberegister, bei dem der Ausgang der letzten Speicherzelle invertiert auf den Eingang der ersten Speicherzelle zurückgeführt ist. Er wird mit »alles Null« initialisiert, läuft zyklisch mit Einsen voll, bis die erste Eins hinten ankommt. Dann läuft er mit Nullen voll, bis die erste Null hinten ankommt. Die Zykluslänge ist doppelt so groß wie die Registerlänge.

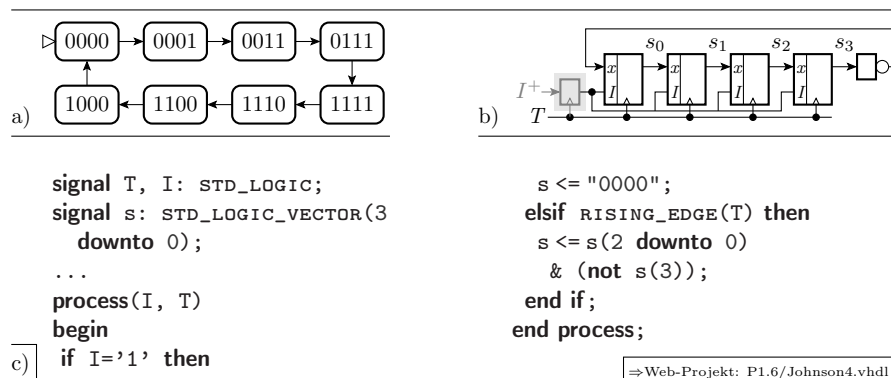


Abb. 1.74. 4-Bit-Johnson-Zähler a) Zustandsgraph b) Schaltungsstruktur c) VHDL-Funktionsbeschreibung

Das Zustandssignal ist im allgemeinen Fall ein Bitvektor der Länge n . Die gesamte Funktion lässt sich in einem einzelnen Abtastprozess beschreiben, bei dem der Takt und das Initialisierungssignal in der Weckliste stehen. Bei aktivem Initialisierungssignal wird das Zustandssignal zurückgesetzt. Sonst wird bei jeder aktiven Taktflanke den Zustandsbits $i = 1$ bis $n - 1$ der Ist-Wert des Vorgängerbits und dem niederwertigsten Bit der negierte Ist-Wert des höchstwertigen Bits zugewiesen.

1.6.3 Unzulässige Zustände, Systemabsturz und Watchdog

Ein Automat mit n Speicherzellen hat 2^n Zustände. Davon wird meist nur ein geringer Teil für das Soll-Verhalten genutzt. Die übrigen Zustände sind unzulässig und sollten im normalen Betrieb nie erreicht werden. Größere digitale Systeme sind statisch gesehen nicht fehlerfrei, unterliegen Störungen und werden nicht immer so bedient, wie es die Zielfunktion vorsieht. Im Fehlerfall können Automaten auch in unzulässige Zustände übergehen. Ein Teil der unzulässigen Zustände kann der Automat nicht selbstständig wieder verlassen.

Er liefert dann bis zur Neuinitialisierung keine oder keine sinnvollen Ausgaben mehr. Ein solcher Systemzustand wird als Systemabsturz bezeichnet.

Abbildung 1.75 a zeigt die komplette Zyklusstruktur des Johnson-Zählers aus Abb. 1.74. Außer den acht Zuständen, die er im fehlerfreien Fall zyklisch durchläuft, besitzt er acht unzulässige Zustände, die auch einen Zyklus bilden. Wenn der Johnson-Zähler aufgrund einer Fehlfunktion in einen der unzulässigen Zustände übergeht, gibt es zumindest im bisherigen Funktionsmodell keinen Weg zurück. Er bleibt bis zur Neuinitialisierung im Zyklus der unzulässigen Zustände.

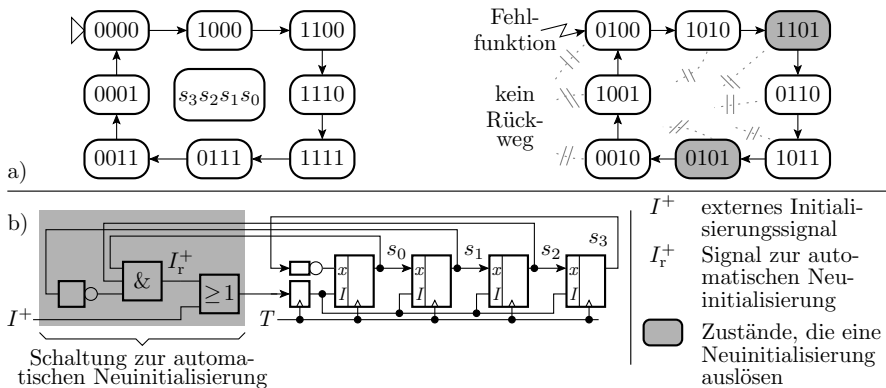


Abb. 1.75. a) Zyklusstruktur des 4-Bit-Johnson-Zählers mit unzulässigen Zuständen b) Schaltungserweiterung mit automatischer Neuinitialisierung

Ein Automat lässt sich theoretisch auch so konstruieren, dass auf einen unzulässigen Zustand im Zustandsgraph nach einer begrenzten Anzahl von Zustandsübergängen immer ein zulässiger Zustand folgt. In Abb. 1.75 b ist der Johnson-Zähler um eine Schaltung zur automatischen Neuinitialisierung erweitert, die, wenn er in einen der unzulässigen Zustände »0101« oder »1101« übergeht, das Initialisierungssignal aktiviert. Mit der nächsten aktiven Taktflanke startet eine Neuinitialisierung, nach der der Zähler wieder korrekt arbeitet.

Große digitale Systeme, z.B. Rechner, haben Tausende von Zustandsbits und eine unüberschaubar große Anzahl unzulässiger Zustände. Bei solchen Systemen erfolgt die Detektierung eines Systemabsturzes durch Zeitüberwachung und der automatische Neustart in der Regel durch einen Watchdog. Ein Watchdog ist ein Zähler für Zeitimpulse, der vom zu überwachenden System in regelmäßigen Zeitabständen zurückgesetzt wird. Wenn das System abstürzt, bleiben die Rücksetzimpulse aus. Der Zähler läuft über und aktiviert das Initialisierungssignal des überwachten Systems (Abb. 1.76).

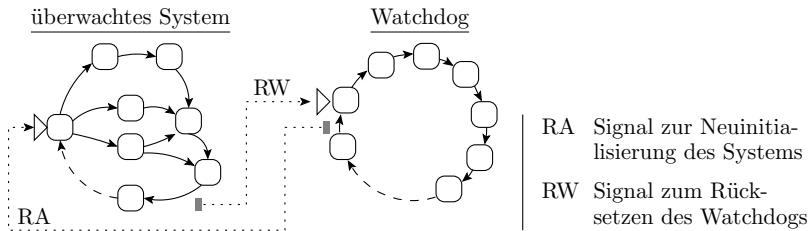


Abb. 1.76. Zeitüberwachung mit Watchdog

1.6.4 Entwurf eines Zahlenschlosses

In diesem Abschnitt wird eine Steuerung für ein Zahlenschloss mit mehreren Zifferntasten, einer Rücksetztaste und einem Ausgabesignal, das zum Testen eine Leuchtdiode ansteuert, entworfen. Wenn nach der Betätigung der Rücksetztaste die richtige Ziffernfolge eingetippt wird, soll die Leuchtdiode leuchten und sonst ausgeschaltet bleiben. Alle Anschluss-signale sind low-aktiv. Bei einer Tastenbetätigung ist das zugehörige Eingabesignal »0« und sonst »1«. Die Leuchtdiode leuchtet bei einer »0« am Ausgang.

Mechanische Tasten liefern asynchrone Eingabesignale und prellen. Jedes Tastensignal muss mit einer Periodendauer, die länger als die maximale Prellzeit und kürzer als die minimale Betätigungsdauer ist, z.B. mit $T_p = 10\text{ ms}$ abgetastet werden (siehe Abschnitt 1.5.2). Das abgetastete Rücksetzsignal ist das Initialisierungssignal des Zustandsregisters. Für die Zifferntasten wird ein zweites Abtastsignal benötigt, um Eingabeänderungen zu erkennen. Es genügt jedoch, wenn aus der zweiten Abtastung zu erkennen ist, dass im Zeitschritt zuvor keine Taste gedrückt war, d.h. wenn nur die UND-Verknüpfung x_U der Abtastwerte aller abgetasteten Signale von Zifferntasten noch einmal abgetastet wird. Der eigentliche Automat besteht aus je einer kombinatorischen Schaltung für die Übergangs- und für die Ausgabefunktion und dem Zustandsregister. Die Übergangsfunktion wertet den Ist-Zustand, die abgetasteten Ziffernsignale, deren UND-Verknüpfung x_U und die abgetastete UND-Verknüpfung x_U' aus. Das Ausgabesignal soll aus den Zuständen gebildet werden (Moore-Automat). Abbildung 1.77 zeigt die beschriebene Schaltung, in der die Übergangsfunktion und die Ausgabefunktion des eingebetteten Automaten noch zu entwerfen sind.

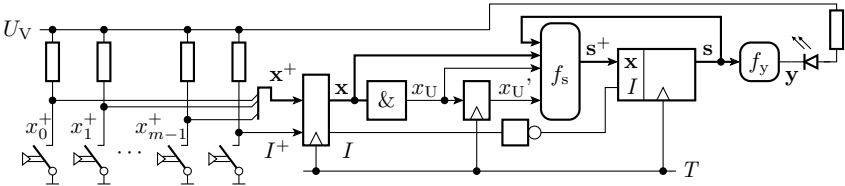


Abb. 1.77. Gesamtschaltung der Steuerung für das Zahlenschloss

Im nächsten Schritt wird der Zustandsgraph entwickelt (Abb. 1.78 a). In seinem Kern ist ein Zahlenschloss ein Akzeptorautomat. Die Eingabe einer Ziffernfolge beginnt mit der Initialisierung, die den Automaten in den Startzustand Z_0 versetzt. Von dort aus führt eine Zustandsfolge Z_i zum Endzustand Z_n (Eingabe akzeptiert). Der Index i ist dabei jeweils die Anzahl der bisher richtig eingegebenen Ziffern. Im Endzustand Z_n soll die Leuchtdiode leuchten ($y = 0$) und in allen andern Zuständen soll sie ausgeschaltet sein ($y = 1$).

```

signal T, I, I_next, xu, xu_del, y: STD_LOGIC;
signal s: STD_LOGIC_VECTOR(2 downto 0);
signal x, x_next: STD_LOGIC_VECTOR(3 downto 0);

```

```

...
-- c) Beschreibung der Abtastregister
process(T)
begin
  if RISING_EDGE(T) then
    x <= x_next;
    I <= I_next;
    xu_del <= xu;
  end if;
end process;

-- d) nebenläufige Bildung von xu
xu <= x(0) and x(1) and x(2) and x(3);

-- e) Abtastprozess der Uebergangsfunktion

```

```

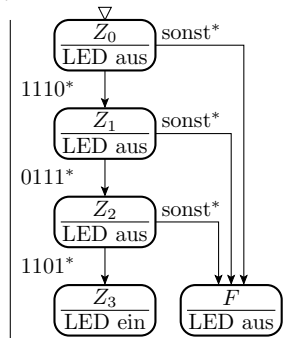
process(I, T)
  variable v: STD_LOGIC_VECTOR(6 downto 0);
begin
  if I = '0' then
    s <= "000";
  elsif RISING_EDGE(T) and xu = '0' and
    xu_del = '1' and s(2) = '0' then
    v := s & x;
    case v is
      when "000" & "1110" => s <= "001"; -- 1. Geheimzahl: Taste 0
      when "001" & "0111" => s <= "010"; -- 2. Geheimzahl: Taste 3
      when "010" & "1101" => s <= "100"; -- 3. Geheimzahl: Taste 1
      when others => s <= "111";
    end case;
  end if;
end process;

```

```

-- f) nebenläufige Anweisung zur Bildung von y
y <= not s(2) or s(1) or s(0);

```



a) $* \wedge (x_U = 0) \wedge (x_U' = 1)$

b)

Zustand	Codierung
Z_0	000
Z_1	001
Z_2	010
Z_3	100
F	111

Abb. 1.78. Fortsetzung des Entwurfs der Zahlenschlosssteuerung a) Zustandsgraph b) Zustandscodierung c) Abtastregister d) UND-Gatter e) Übergangsfunktion und Zustandsregister f) Ausgabefunktion

Ein Zustandsübergang soll nur erfolgen, wenn im vorherigen Abtastschritt keine Taste betätigt war ($x_U = 1$), im aktuellen Abtastschritt eine Taste betätigt ist ($x_U = 0$) und keiner der Endzustände erreicht ist. Wenn die richtige Taste betätigt wird, ist der Abtastwert dieser Taste »0« und der der anderen Tasten »1«. Für ein Zahlenschloss mit $m = 4$ Zifferntasten sei die korrekte Ziffernfolge null-drei-eins. Bei Betätigung der nullten Taste ist der Eingabevektor $\mathbf{x} = 1110$, bei Betätigung der dritten Taste $\mathbf{x} = 0111$ und bei Betätigung der ersten Taste $\mathbf{x} = 1101$. Die Betätigung einer falschen Taste oder von mehreren Tasten soll den Automaten in einen Fehlerzustand F versetzen, der gleichfalls ein Endzustand ist.

Bei einem Automatenentwurf mit den bisher eingeführten VHDL-Beschreibungsmitteln folgt nach der Aufstellung des Zustandsgraphen die Zustandscodierung. Für jeden symbolischen Zustand ist eine Bitvektorkonstante festzulegen, mit der er in der Schaltung dargestellt wird. Die Zustandscodierung ist in Abb. 1.78 b so gewählt, dass die beiden Endzustände Z_3 und F an $s_2 = 1$ erkannt werden. Der erste Abtastprozess (Abb. 1.78 c) beschreibt die Abtastregister für das Eingabesignal, das Initialisierungssignal und das Signal x_U , das in einer nebenläufigen Anweisung aus dem abgetasteten Zifferntastenvektor gebildet wird. Der zweite Abtastprozess (Abb. 1.78 e) beschreibt die Übergangsfunktion und das Zustandsregister des Automaten. Bei einem aktiven Abtastwert des Initialisierungssignals wird dem Zustandssignal der Wert für den Zustand Z_0 zugewiesen, sonst bei einer aktiven Taktflanke, wenn ein Tastendruck erkannt wird und sich der Automat in keinem Endzustand befindet, erfolgt eine Auswahl nach dem Ist-Zustand und dem Abtastwert der Eingabe. Im Zustand Z_0 erfolgt bei Eingabe des Wertes »1110« für die erste Geheimzahl der Übergang nach Z_1 , von dort, wenn die zweite Eingabe richtig ist, nach Z_2 etc. In allen anderen Fällen wechselt der Automat in den Fehlerzustand F mit dem Zustandswert $\mathbf{s} = 111$. Das Ausgabesignal y wird in einer nebenläufigen Anweisung gebildet (Abb. 1.78 f).

Die Zurückführung einer Aufgabe auf einen Automatenentwurf teilt den Entwurf in einen kreativen Teil zur Entwicklung der Schaltung mit dem Automaten als Black-Box und einen eher rezeptartigen Teil, in dem die Automatenbeschreibung selbst durch eine Schaltung nachgebildet wird. Der vorgelagerte kreative Teil bestimmt dabei die Struktur und die Güte der Lösung. Ohne den Trick mit dem Signal x_U für »keine Taste gedrückt« hätte man zwischen den Zuständen für »Warte auf richtige Tasteneingabe« zusätzlich Zustände für »keine Taste gedrückt« einfügen müssen, was die Zustandsanzahl fast verdoppelt hätte. Mit dem Trick, Zustandsübergänge an weitere Bedingungen als nur die steigende Taktflanke zu binden, entfallen zahlreiche Kanten.

1.6.5 Operationsablaufgraph

Die bisherige Beschreibung einer sequenziellen Schaltung durch einen Zustandsgraphen hat einen entscheidenden Schwachpunkt. Die aus den Werten aller Eingabesignale gebildeten Eingabemengen und die aus den Zuständen

aller Speicherelemente gebildeten Zustandsmengen wachsen beide exponentiell mit der Bitanzahl. Ab etwa zehn Zuständen und 20 Kanten, d.h. ab etwa zusammen vier Eingabe- und Zustandsbits, wird ein Zustandsgraph unübersichtlich.

Wie lassen sich die zu verarbeitenden und die gespeicherten Informationen besser strukturieren?

Das Vorbild ist ein Programm auf einem Rechner. Ein Programm hat einen Kontrollfluss, der Operationen steuert und zum Teil von Operationsergebnissen gesteuert wird. Die möglichen Operationen für Daten sind eingeschränkt (Zählen, Addieren, bitweise Logikoperationen etc.). Aus der Programmierung ist bekannt, dass sich so jeder Algorithmus beschreiben lässt, ohne dass die Beschreibungsgröße exponentiell mit der Anzahl der zu verarbeitenden und zu speichernden Bits zunimmt.

Ein Operationsablaufgraph ist ein erweiterter Zustandsgraph, der zusätzlich Operationen steuert und in den Übergangsbedingungen der Kanten Operationsergebnisse auswertet. Für die Spezifikation einer sequenziellen Schaltung mit einem Operationsablaufgraphen sind zuerst die Operationen und die Operanden, die für die Nachbildung der Zielfunktion benötigt werden, als Register-Transfer-Funktionen zusammenzustellen. Im zweiten Schritt wird ein Ablaufgraph konstruiert, der die Abfolge der Operationen unter teilweiser Auswertung der Berechnungsergebnisse beschreibt. Die folgenden Beispiele zeigen, dass bereits eine einfache Zählfunktion und die Auswertung des Zählstandes die Beschreibung von Abläufen und deren Nachbildung durch sequenzielle Schaltungen drastisch vereinfachen kann. Größere Beispiele folgen später.

Beispiel Dreiecksignalgenerator

Der zu entwerfende Dreiecksignalgenerator soll seinen Ausgabewert immer alternierend aufwärts und abwärts zählen. Abbildung 1.79 a zeigt das zu generierende 4-Bit-Dreiecksignal. Die Zykluslänge beträgt $2 \cdot (2^4 - 1) = 30$. Ein Automat nach dem bisherigen Schema hätte 30 Zustände, die zyklisch durchlaufen werden.

In einem Programm würde man die gegebene Zielfunktion mit einem Zähler beschreiben, der in einer Schleife immer bis zu seinem Maximalwert aufwärts und anschließend bis zu seinem Minimalwert abwärts zählt. Bevor der Operationsablauf festgelegt werden kann, sind die zu manipulierenden Datenobjekte und die auf sie anzuwendenden Operationen zu spezifizieren. Der Dreiecksignalgenerator benötigt ein Zählregister und die drei auf das Zählregister anwendbaren Operationen: »clr« (Zählstand auf null setzen), »inc« (Inkrement, Aufwärtszählen) und »dec« (Dekrement, Abwärtszählen, Abb. 1.79 b). Der Operationsablaufgraph kommt mit zwei Zuständen aus. Bei der Initialisierung des Automaten ist auch der Zähler zu initialisieren. Im Zustand »0«

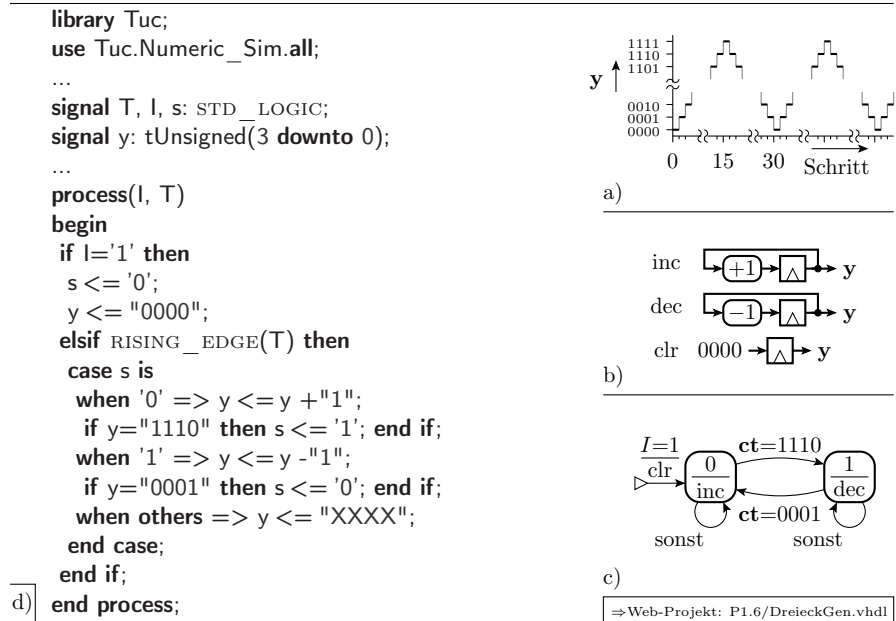


Abb. 1.79. 4-Bit-Dreiecksignalgenerator a) zu generierendes Dreiecksignal b) Operationen des Zählregisters c) Operationsablaufgraph d) VHDL-Beschreibung

zählt der Zähler bis zum Maximalwert. Dann wechselt der Automat in den Zustand »1« und lässt den Zähler bis zum Minimalwert abwärts zählen. Dann wechselt er zurück zum Anfangszustand »0« und beginnt den Zyklus von vorn.

In der VHDL-Beschreibung sind sowohl die Zähloperationen als auch die Übergangsfunktion des Kontrollflussautomaten Register-Transfer-Funktionen, die in einem gemeinsamen Abtastprozess beschrieben werden können. Bei aktivem Initialisierungssignal werden der Zählerstand und der Automatenzustand auf null gesetzt. Im anderen Fall, bei einer aktiven Taktflanke, erfolgt eine Fallunterscheidung nach dem Zustand. Im Zustand $s = 0$ wird der Zählstand um »1« erhöht und abgefragt, ob der vorletzte Zählstand erreicht ist. Wenn ja, wird dem Automatenzustand der Wert für »Abwärtszählen« zugewiesen. Man beachte, dass y und s Signale in einem Abtastprozess sind, die ihre zugewiesenen Werte erst mit der nächsten aktiven Taktflanke übernehmen. Deshalb muss der Folgezustand immer einen Zählschritt vor dem Endzustand zugewiesen werden. Im Zustand $s = 1$ wird bis »0000« abwärtsgezählt und einen Schritt zuvor wieder der Automatenzustand für »Aufwärtszählen« zugewiesen.

Das Simulationsmodell enthält eine Addition und eine Subtraktion. Dafür wird ein Bitvektortyp benötigt, für den diese beiden arithmetischen Operationen definiert sind. Im Weiteren werden, wie auch hier im Beispiel, vorzeichenfreie Zahlen mit dem Typ »tUnsigned« dargestellt. Für vorzeichenbehaftete

Zahlen wird der Typ »tSigned« benutzt. Beide Typen sind in dem eigenen Package »Tuc.Numeric_Sim«, das mit auf der Web-Seite steht, definiert und sind mit dem Typ STD_LOGIC_VECTOR eng verwandt (siehe später Abschnitt 3.2.7).

1.6.6 Beispiel Wegemessung mit Quadratur-Encoder

Ein Quadratur-Encoder ist ein Signalgeber für eine inkrementelle Wegemessung. Er besteht aus einer rotierenden Scheibe mit Löchern und einer Doppellichtschranke (Abb. 1.80 a). Die Löcher in der Scheibe und die beiden Lichtschranken sind dabei so angeordnet, dass bei einer Rotation der Scheibe an den Lichtschrankenausgängen Impulse mit einem Tastverhältnis 1:1 erzeugt werden, die um eine viertel Periode zueinander versetzt sind. Bei einer Vorwärtsbewegung wechselt das Ausgangssignal zuerst am Ausgang *a* und bei einer Rückwärtsbewegung zuerst am Ausgang *b* (Abb. 1.80 b).

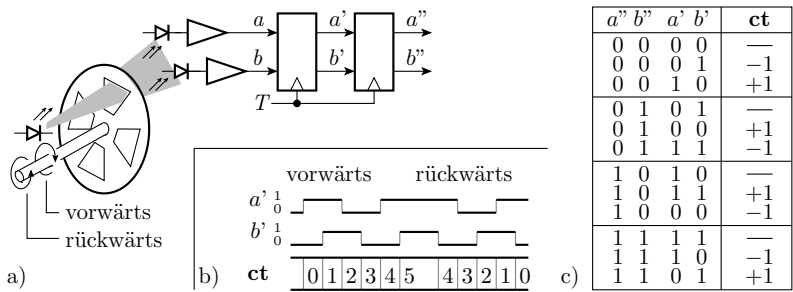


Abb. 1.80. Wegemessung mit einem Quadratur-Encoder a) Messanordnung mit Abtast- und Zustandsregister b) abgetastete Encoder-Ausgaben und Zählverhalten c) tabellarische Ausgabefunktion

Ein digitales Wegemesssystem mit einem Quadratur-Encoder besteht aus dem Abtastregister für die beiden asynchronen Eingabesignale, einem Vorwärts-Rückwärts-Zähler und einem Steuerautomaten. Der Steuerautomat hat vier Zustände. Das sind in diesem speziellen Fall die vier Variationen der Abtastwerte *a''b''* des Zeitschritts zuvor. Der Folgezustand ist praktisch gleich der abgetasteten Eingabe. Abbildung 1.80 c zeigt die den Zustandsübergängen zugeordneten Operationen. Bei einer Vorwärtsdrehung schaltet zuerst *a* ein (*a''b''a'b'* = 0010), dann schaltet *b* ein (*a''b''a'b'* = 1011), dann *a* aus (*a''b''a'b'* = 1101) und zum Schluss *b* aus (*a''b''a'b'* = 0100). Bei jedem dieser Übergänge ist der Zählstand zu erhöhen. Wenn zuerst *b* einschaltet (*a''b''a'b'* = 0001), dann *a* einschaltet (*a''b''a'b'* = 0111), dann *b* ausschaltet (*a''b''a'b'* = 1110) und zum Schluss *a* ausschaltet (*a''b''a'b'* = 1000), hat sich das Encoder-Rad je einen Schritt zurückgedreht. Der Zählstand ist um »1« zu verringern.

In der nachfolgenden VHDL-Beschreibung sind die Encoder-Signale zu einem 2-Bit-Vektor zusammengefasst. Die einfach und die zweifach abgetasteten Eingabewerte bilden zusammen einem 4-Bit-Vektor wie in der Tabelle in Abb. 1.80 c:

```

signal T, I, I_del: STD_LOGIC;
signal ab: STD_LOGIC_VECTOR(1 downto 0);
signal ab_del: STD_LOGIC_VECTOR(3 downto 0);
signal ct: tSigned(15 downto 0);
...
-- Abtastprozess
process(T)
begin
  if RISING_EDGE(T) then
    I_del <= I;
    ab_del(1 downto 0) <= ab; -- a'b' <= ab
    ab_del(3 downto 2) <= ab_del(1 downto 0); -- a''b'' <= a'b'
  end if;
end process;
-- Uebergangsfunktion
process(I_del, T)
begin
  if I_del='1' then ct <= "0000000000000000";
  elsif RISING_EDGE(T) then
    case ab_del is
      when "0010" | "0100" | "1011" | "1101" => ct <= ct+"01";12
      when "0001" | "0111" | "1000" | "1110" => ct <= ct-"01";
      when "0000" | "0101" | "1010" | "1111" => null;
      when others => ct <= "XXXXXXXXXXXXXXXXXX"; -- 16 x 'X'
    end case;
  end if;
end process;

```

⇒ WEB-Projekt: P1.6/QuadEnc.vhdl

Die Gesamtbeschreibung ist in zwei Abtastprozesse aufgeteilt, einen mit und einen ohne Initialisierungssignal in der Weckliste. Außer dem Abtastregister benötigt auch das Zustandsregister keine Initialisierung, weil nach zwei Takten mit gültigen Eingaben automatisch ein gültiger Zustand übernommen wird. Das Initialisierungssignal für den Zähler darf dann aber erst nach den beiden ersten aktiven Taktflanken, wenn das Zustandsregister einen gültigen Wert hat, deaktiviert werden. Der Zähler im zweiten Prozess wird durch ein 16-Bit-Signal vom Typ »tSigned« nachgebildet. Das ist der für arithmetische Operationen für vorzeichenbehaftete Zahlen definierte Typ im Package »Tuc.Numeric_Sim« (siehe Anhang A.2.4). Der Zählprozess setzt den Zähler bei aktivem Initialisierungssignal zurück und wählt sonst bei einer steigenden Taktflanke die Zähloperation entsprechend der Tabelle in Abb. 1.80 c aus. Für die vier Bitvektorstufen, die bei der Vorwärts- bzw. Rückwärtsbewegung auf-

¹² Die führende »0« repräsentiert das Vorzeichen (siehe später Abschnitt 2.4.2).

treten, wird der Zähler um den Wert eins erhöht bzw. verringert. Den vier Bitvektoren, bei denen der erste Abtastwert mit dem zweiten Abtastwert übereinstimmt, ist die Null-Anweisung (keine Operation) zugeordnet¹³. Alle anderen Fälle – gleichzeitige Änderung beider Encoder-Werte, ungültige Eingaben, etc. – dürfen im korrekten Betrieb und bei der Simulation nicht auftreten. Um potenzielle fehlerhafte Zustände und Eingaben aufzudecken, wird bei ihrem Auftreten der Zählwert invalidiert.

1.6.7 Software-orientierte Ablaufbeschreibung

Für komplexere Schaltungen ist die Aufstellung der Zielfunktion bereits ein arbeitsaufwändiger Prozess, der in Teilschritte untergliedert werden muss. Ein bewährter Zwischenschritt ist die Beschreibung der geplanten Operationsabläufe als normales Programm mit Fallunterscheidungen und Schleifen. Ein solches Programm ist übersichtlicher und änderungsfreundlicher als eine Funktionsbeschreibung mit einem Zustands- oder einem Operationsablaufgraphen. Die Aufteilung in Zeitschritte erfolgt mit Warteanweisungen.

Abbildung 1.81 a zeigt den Signalverlauf des Dreiecksignalgenerator aus Abb. 1.79 und Abb. 1.81 b eine Software-orientierte Ablaufbeschreibung mit einem Struktogramm. Das Verhalten wird mit einer Endlosschleife, in der in einer ersten eingebetteten Schleife der Ausgabewert aufwärts und in einer zweiten eingebetteten Schleife abwärts gezählt wird, beschrieben. In jedem Schleifendurchlauf wird für eine Taktperiode gewartet. Ein Modell dieser Art ist auch in VHDL schnell geschrieben und verhält sich bei der Simulation wie die Zielfunktion. Die Nachbildung eines solchen Programmes durch einen Zustands- oder Operationsablaufgraphen erfolgt nach einem einfachen Rezept. Jeder Wartezustand wird zu einem Ablaufzustand. Die Operationen und Ausgaben werden jeweils dem nächsten Wartezustand zugeordnet. Komplexere Beispiele folgen später.

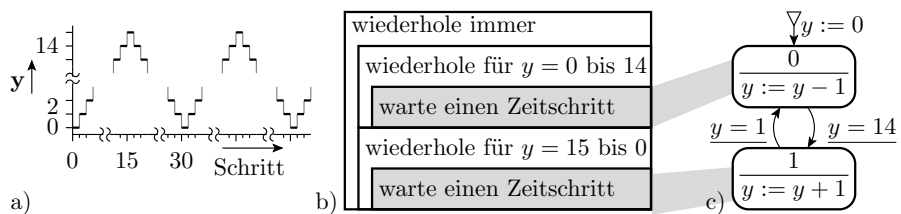


Abb. 1.81. Erzeugung eines Dreiecksignals a) zu erzeugender Signalverlauf b) Software-orientierte Ablaufbeschreibung mit einem Struktogramm c) Nachbildung des Struktogramms durch einen Operationsablaufgraphen

¹³ Eine VHDL-Auswahlanweisung ordnet jedem Auswahlwert mindestens eine Anweisung zu. Die Dummy-Anweisung für »tue nichts« ist in VHDL die Null-Anweisung.

1.6.8 Zusammenfassung und Übungsaufgaben

Eine sequenzielle Schaltung berechnet ihre Ausgabewerte in mehreren Schritten. Ein universelles Funktionsmodell dafür ist ein Zustandsgraph, der auf dem Modell des endlichen Automaten basiert und das Verhalten durch Zustände und Zustandsübergänge beschreibt. In der Schaltungsnachbildung des Automaten werden die Zustände durch ein Zustandsregister, die Abbildung der Eingabe und des Ist-Zustands auf den Folgezustand durch eine kombinatorische Schaltung mit der Übergangsfunktion und die Abbildung der Eingabe und des Zustands auf die Ausgabe durch eine kombinatorische Schaltung mit der Ausgabefunktion beschrieben. Der Anfangszustand ist der Initialwert des Zustandsregisters. Die Nachbildung eines Zustandsgraphen in VHDL erfolgt nach einfachen Schablonen, in der Regel mit Fallunterscheidungen. Die Übergangsfunktion kann immer mit im Abtastprozess des Zustandsregisters beschrieben werden. Die Ausgabesignale werden vorzugsweise in einem kombinatorischen Prozess oder mit nebenläufigen Signalzuweisungen erzeugt.

Eine sequenzielle Schaltung besitzt in der Regel unzulässige Zustände, die zwar nicht im normalen Betrieb, aber durch Fehlfunktionen erreicht werden und zu einem Systemabsturz führen können. Die übliche Fehlerbehandlung bei einem Systemabsturz ist die Neuinitialisierung, entweder manuell über eine Reset-Taste oder automatisch mit einem Watchdog.

Zustandsgraphen sind nur für Funktionen mit wenigen zu unterscheidenden Zuständen und Eingabewerten geeignet. Zielfunktionen mit großen Zustands- und Eingabemengen lassen sich genau wie ein Programm auf einem Rechner durch eine Abfolge von Operationen modellieren. Dazu werden Operanden als in Registern gespeicherte Signale und die Operationen als Register-Transfer-Funktionen definiert. Der Ablauf wird durch einen Operationsablaughen beschrieben. Allein mit einer Zählfunktion als gesteuerte Operation lassen sich auf diese Weise recht komplexe Abläufe in einer anschaulichen Form darstellen und in VHDL beschreiben. In praktischen Entwürfen ist meist vor dem eigentlichen Automatenentwurf eine Skizze der Gesamtschaltung mit der Übergangs- und der Ausgabefunktion als Black-Box zu entwerfen. Ein anderer nützlicher Entwurfszwischenschritt ist die Beschreibung und Simulation der geplanten Operationsabläufe im Stile eines normalen imperativen Programms mit Warteanweisungen zum Weiterschalten der Simulationszeit. Weiterführende und ergänzende Literatur siehe [14, 24, 33, 36, 41, 45].

Aufgabe 1.20

Gegeben sind die Schaltung und die Zeitverläufe in Abb. 1.82.

- Bestimmen Sie für alle Werte des Ist-Zustands s_1s_0 und des Eingabesignals x den Folgezustand $s_1^+s_0^+$ und den Ausgabewert y .
- Zeichnen Sie den Zustandsgraphen des Automaten.
- Ergänzen Sie die fehlenden Signalverläufe für den Zustand und die Ausgabe in Abb. 1.82 b.

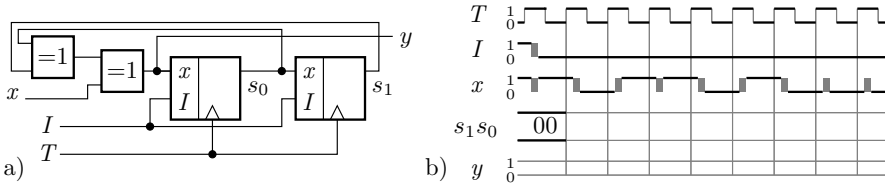


Abb. 1.82. Schaltung und Signalverläufe zu Aufgabe 1.20

Aufgabe 1.21

Zur Funkübertragung von Daten werden balancierte Codes eingesetzt. Das sind Codes, bei denen innerhalb eines bestimmten Zeitfensters die Anzahl der übertragenen Nullen gleich der Anzahl der übertragenen Einsen ist. Für einen so codierten Datenstrom ist eine Überwachungsschaltung zu konstruieren, die die Differenz zwischen der Anzahl der empfangenen Einsen und der Anzahl der empfangenen Nullen $N_1 - N_0$ überwacht und, wenn diese den Wert -3 unterschreitet oder den Wert drei überschreitet, das Fehlersignal y aktiviert. Die nachfolgende Tabelle zeigt einen Beispielablauf:

Eingabe x	1	0	0	0	0	0	1	0	1	1	1	1	1	1	1	...
$N_1 - N_0$	0	1	0	-1	-2	-3	-3	-2	-3	-2	-1	0	1	2	3	3
Ausgabe y	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	...

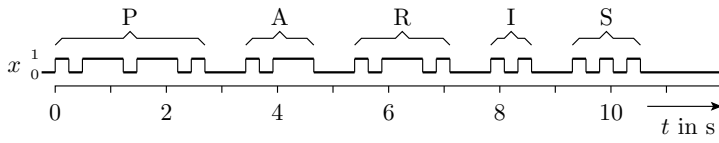
Entwerfen Sie einen Zustandsgraphen zur Nachbildung dieser Funktion.

Aufgabe 1.22

Morsezeichen bestehen aus Punkten und Strichen. Punkte sind kurze Impulse. Ihre Länge soll in einem Bereich von $t_P = 200...300$ ms liegen. Striche sind lange Impulse, die eine Zeitdauer zwischen $t_S = 600...900$ ms haben sollen. Die Taktfrequenz der zu entwerfenden Schaltung sei $f_T = 20$ Hz und die Prellzeit der Eingabesignale viel kleiner als die Taktpériode. Entwerfen Sie eine Schaltung mit einer Eingabetaste für das Morsecode, einer Eingabetaste zum Initialisieren, einem bitorientierten Ausgabesignal p , das nach Empfang eines Punktes, einem bitorientierten Ausgabesignal s , das nach Empfang eines Strichs und einem bitorientierten Ausgabesignal err , das nach Empfang eines Impulses unzulässiger Breite für einen Takt aktiviert (auf »1« gesetzt) wird.

- Skizzieren Sie die Gesamtschaltung mit einem Automaten, dessen Übergangs- und Ausgabefunktion als Black-Box dargestellt ist.
- Entwerfen Sie den Operationsablaufgraphen unter Verwendung eines Zählers.
- Beschreiben Sie die Gesamtfunktion in VHDL durch Abtastprozesse und kombinatorische Prozesse.

- d) Entwerfen Sie einen VHDL-Testprozess, der den Morse-Code für das Wort »Paris« als Testeingabesignal bereitstellt.





<http://www.springer.com/978-3-642-17446-9>

Technische Informatik

Band 2: Entwurf digitaler Schaltungen

Kemnitz, G.

2011, XIII, 503 S. 372 Abb., Softcover

ISBN: 978-3-642-17446-9