Digital Design

WS 2019

Vorbesprechung:

Mo: 10^{15} - 12^{00} HS 13

Fr: 11^{15} - 13^{00} EI 10

Folien in Towel

Prüfung: Potentielle Prüfungsfragen in rot.

Taschenrechner (nicht programmierbar) Beispiele können sich WH :)

mind 49 <48 ist negativ
Pro Theorie 4 - 8 Punkte

KV Diagramm kommt fast immer + AOI/OAI 1,5 h

Beispielfragen

(1 Wo for Pr max!)
(Stoff erklären nicht :))

Inhalt:

1 Grundlagen → chips
2 ASIC-Fertigung
3 CMOS-Logik
4 Design-Flow
5 Speichertechnologien
6 ASIC-Zieltechnologien → VHDL
7 Temperatur & Verlustleistung
8 Datenblatt-Angaben
9 Synchrones Design & Metastabiliät
10 Defekte
11 Test
12 Logikanalysator

Literatur:

- Dipital Design – Principles & Practises
- Buch 2

Allgemeine Erkenntnisse

Laufzeit

Rekonstruktivität (mehr als High/Low)

Temperatur

parallele Prozesse → nicht alles nacheinander

Modellierung v Abkürzungen

Hierarchien

Fehlermodelle

Das sind alles Abstraktionen

Vorgänge, die sich aus R, C etc ergeben

→ auf

oder so :)

Grundlagen

Physik — elektromagnetisches Feld, Thermodynamik

Transistoren — Schwellspannung, Ausgangsstrom

Transistor: (Halbleiter) Halbleiterbauelement = Steuerbarer Widerstand

ALU: arithmetic logic unit

Schalter — Steuerung

Gatter — logische Pegel / Verknüpfung

} DiDe

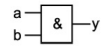
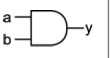
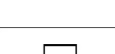
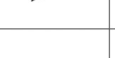
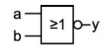
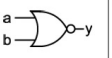
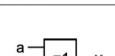
Assembler: übersetzt Assembler in Maschinensprache

Registertransfer RTL (Abstraktionsebene von integrierten Schaltkreisen) High level darstellungen, ALU

Assembler — Registerfile, Befehl, Sprung

Hochsprache Datentyp, Zweisung

Algorithmen Regler, Signalverarbeitung

Bezeichnung	Schaltsymbol nach IEC 60617	ANSI Schaltsymbol	Boolescher Ausdruck	Wahrheitstabelle
AND			$y = a \,\&\, b$	b a | $a \,\&\, b$ 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR			$y = a \vee b$	b a | $a \vee b$ 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NOT			$y = \overline{a}$	a | $\overline{a}$ 0 | 1 1 | 0
NAND			$y = \overline{a \,\&\, b}$	b a | $\overline{a \,\&\, b}$ 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR			$y = \overline{a \vee b}$	b a | $\overline{a \vee b}$ 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
XOR			$y = a \oplus b$	b a | $a \oplus b$ 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
XNOR			$y = \overline{a \oplus b}$	b a | $\overline{a \oplus b}$ 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

verschiedene Abstraktionsebenen

VO: Gatter, Schalter und ihre Optimierungen

CAC?

analog: alle Punkte / Spannungspegel interessant

digital: Unterscheidung high / low

Spannungen: analoges Signal mit digitalem Logikpegel

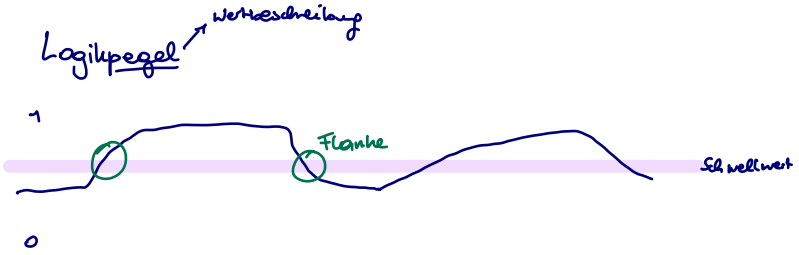
"Jetzt neu: digital"
↳ einfacher zu bedienen
↳ reproduzierbar (teilweise)
↳ billiger (Trend)
↳ weniger Fehleranfällig

Warum digital ?

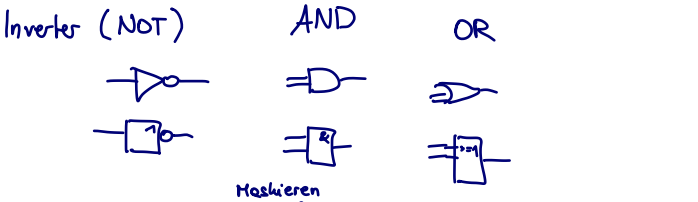
- Störfester (wenn Fehler dann markante)
- Speichern (analogsignal speichern ist schwieriger) Computer ohne Speichermöglichkeit ;(
- Reproduzierbar
- Programmierbarkeit
- Einfache Schaltung (Fertigung)
  Zusammenstöpselung

... aber IRL ist Analog! ⟶ Digitale Betrachtung erlaubt höheren Abstraktionsgrad

Temperatur, Spannungssignale ...



Flanke: steigend 0 → 1
fallend 1 → 0
↑ Zeitpunkt



INV (Eingang & Ausgang) → und zu oder

Bodsche Algebra

- AND
- Const → Ausgang ist immer 0
- BUF nur 1 signal betrachtet
- OR
- XOR
- INV (negiert)
- NAND (not and)
- NOR
- NINV = BUF

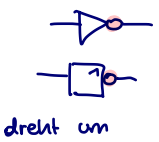
Interrupt - Mask

Kombinatorische Logik

hängt nur von Eingang ab

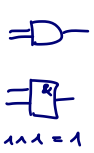
e_1	e_2	konstant 0	$e_1 \wedge e_2$ AND	$e_1 \wedge \neg e_2$	BUF e_1	$\neg e_1 \wedge e_2$	BUF e_2	$e_1 \oplus e_2$ XOR	$\neg(e_1 \vee e_2)$ NOR	$\neg(e_1 \oplus e_2)$ XNOR	$e_1 \vee \neg e_2$	$\neg e_1$	$\neg e_1 \vee e_2$	$\neg(e_1 \wedge e_2)$ NAND	konstant 1
0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	0	0	1	0	0	1	1
1	1	0	1	0	1	0	1	0	0	1	1	0	1	0	1

Inverter (NOT)



dreht um

AND

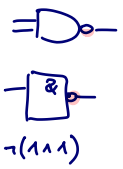


1 ∧ 1 = 1

OR

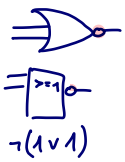
0 ∨ 0 = 0

NAND



¬(1 ∧ 1)

NOR



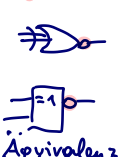
¬(1 ∨ 1)

alle Funktionen darstellbar (wegen demorgan!)

XOR

Antivalenz

XNOR



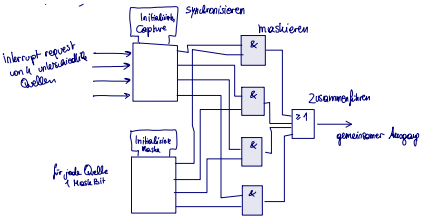
Äquivalenz

Bei gesetztem Inversion (Trojaner)

Addierer, Halbaddierer, Parity

Maskieren:

	Eingang		Ausgang	
AND	0	⇒	0	erzwingt Sperrung der Eingänge (nur 1)
OR	1	⇒	1	erzwingt Sperrung der Eingänge (nicht beide)

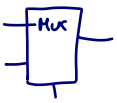


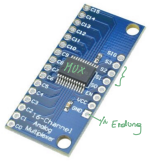
Zusammenführen:

	Eingang		Ausgang	
AND	0	⇒	0	
OR	1	⇒	1	

Wenn man die Eingänge und Ausgänge invertiert, wird AND ↔ OR

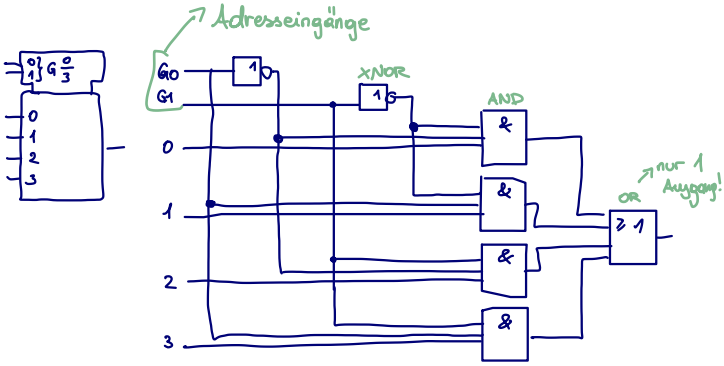
Multiplexer





Auswahl aus mehreren Signalen

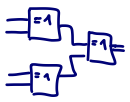
n Dateneingänge → $\log_2 n$ Adresseingänge



Threshold Gate

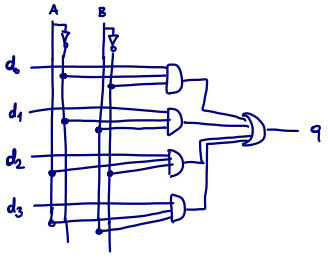
bei >k Eingängen geht Schaltung auf 1.

Parity



XOR even parity

XNOR odd parity



Sequentielle Logik

Anwendung:
Register, Schieberegister, Serielle Parallel Konverter, CRC Generator, FSM ...

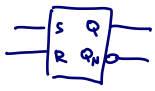
hängt von Eingang und Zustand ab

Enthält explizite Speicherelemente / Speichert durch Rückkopplung sj ⇒ hat Inneren Zustand
(wird bestätigt und so gehalten)

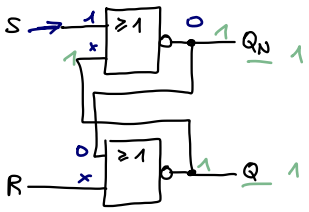
Zustand durch Vorgeschichte bestimmt (Fernsehlautstärke)

SR Latch (verriegelt)

S Set Eingang
R Reset Eingang
Q Ausgang
Q_N Verneinter Ausgang



Prinzipschaltung:



Setzen: S → Q = 1
Löschen: R → Q = 0
Halten: S, R inaktiv → Q hält

$Q = Q_N$ weil S, R gleichzeitig

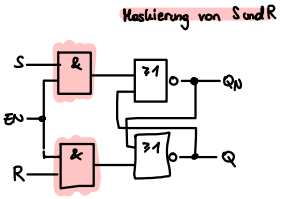
⇓

Timing mit "Haven"

SR Latch mit Enable

wie SR Latch; S, R inaktiv wenn enable = 0
Enabe steuert S, R
EN = 1 → S

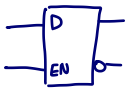
Maskierung von S und R



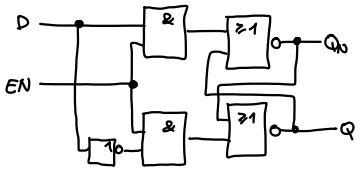
Wenn bei Enable nichts geschalten ist, geht das Ding nicht

Problem: Es kann "Metastabilität" auftreten!

D Latch

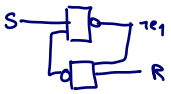


Eingang für D ... Data
Set/Reset gleichzeitig nicht mehr möglich
($Q_N = \neg Q$)

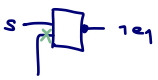


BSP

S Ram Zelle



Inverter



Latch vs Flipflop

Latch	Flipflop
Pegel bewirkt Zustandsänderung	bei aktiver Taktflanke
kein Taktsignal	synchron: mit Taktflanke passiert was

D - Flip Flop

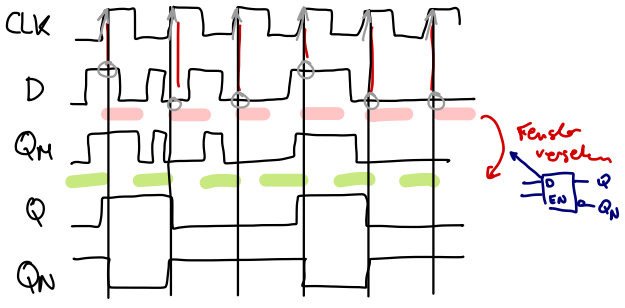
CLK ist wichtigstes Element

ENABLE ist nicht immer Aktiv

nur hold / übernehmen

D → Q_M → Q

Flanke rauf

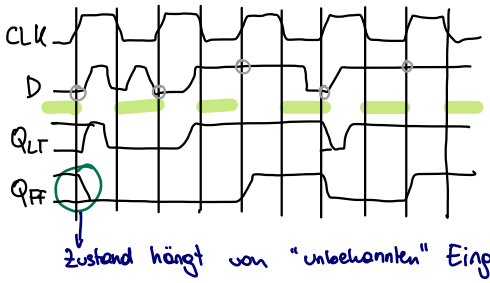


PR: Ausgang von Flipflop aufgrund von Flanken bestimmen

Latch

CLK low → undurchlässig

CLK high → durchlässig



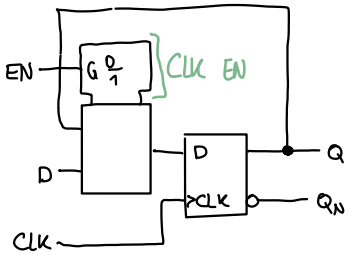
Zustand hängt von "unbekannten" Eingangssignal ab

D-Flipflop

steigende Taktflanke

Clock Enable

- Steuert Übernahme von D nach Q
- Es werden nicht immer alle Funktionseinheiten benötigt



(Ohne Takt ist alles initialisiert-!)

plötzlich / "Metaebene"

Preset erzwingt $Q = 1$

Clear erzwingt $Q = 0$

für initialisierung wichtig! (Durch spez. Gatter)

D	CLK	PRE	CLR	Q	Q_N	
0	↑	1	1	0	1	
1	↑	1	1	1	0	
	0	1	1	lok. Q	lok. Q_N	
	1	1	1	lok. Q	lok. Q_N	
		0	1	1	0	preset
		1	0	0	1	clear

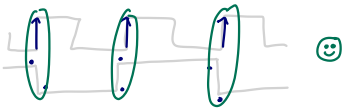
Toggle Flip Flop

mit jeder steigenden Flanke ändert sich der Ausgang (Taktänderung)

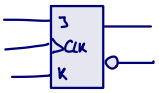
toggle: bei aktiver Flanke wird Ausgang invertiert

↳ Umschalter

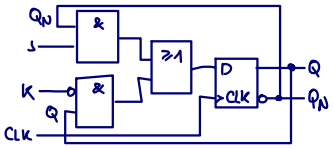
hold: Q hält Zustand bis zur nächsten aktiven Flanke



JK Flip Flop (jump and kick)

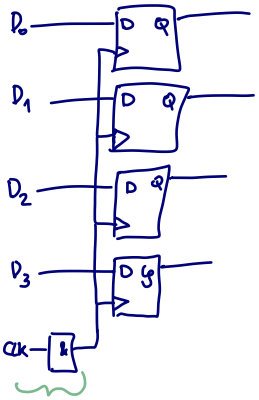


- ähnlich Mux, Toggle FF
- J – Set Eingang
  K – Clear Eingang
  K = 1 clear
  J = 1 set



$J \wedge K = 1$ → toggle (odefiniert, nicht instabil)

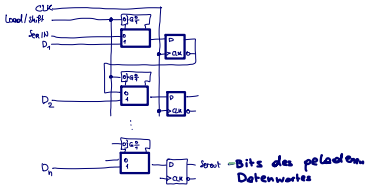
Some random shit ... ?



Takt ist maskiert → timing durch einander für Register schlecht!

Seriell - parallel Konverter

- Kette von DFF als Schieberegister
- Aktive Taktflanke: Q des vorherigen FF als D übernommen
- LOAD 1 → Datenwort synchron übernommen
  0 → Mux legt Q als D an



Linear Feedback Shift Register

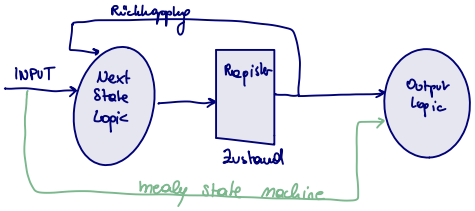
- für Polynomdivison
- arbeitet zyklisch
- Fall Length LFSR hängt von Anzapfungen (Polynom) ab
- hat Eigenschaften einer Zufallsfolge

Finite State Machines

synchrone Zustandswechsel

Moore : Ausgangssignal durch Zustand

Mealy : Ausgangssignal durch Eingang und Zustand



BSP: Ampel braucht (= output logic) kombinatorische Logik
nicht nur 0/1

Zustand: 3 Register → 8 Zustände

PR : Blöcke für Zustandsdiagramm / Beschriften

Boolsche Algebra

Identität, Nullelement, Indempotenz, Komplement, Involution, Assoziativität, Kommutativität, Distributivität, Überdeckung, Kombination

De Morgan $\neg F(x_1 \dots, \wedge, \vee) = F(\neg x_1, \dots, \vee, \wedge)$

Shannon: 1 Variable loswerden → Variable auf 1 setzen (MUX maskiert)

$$(a \wedge b) \vee (b \wedge c) = (b \wedge \underbrace{(a \wedge 1) \vee (1 \wedge c)}_{b=1}) \vee (\neg b \wedge \underbrace{(a \wedge 0) \vee (0 \wedge c)}_{b=0})$$

Minterm: UND von Variablen $\wedge$ $\neg$ (Variable ≤ 1)

Maxterm: OR $\vee$ $\neg$ (Variable ≤ 1!)

KV Diagramm

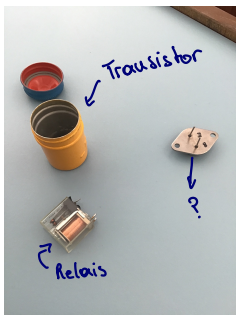
KNF: Nullen DNF auslesen → Invertieren

DNF: 1 bei OR

KNF: 0 bei AND

PR KNF, DNF angeben

ASIC Fertigung



MOTIVATION:

Integrated Circuit; schneller, kleiner, leistungsfähiger, billiger, störsicherer

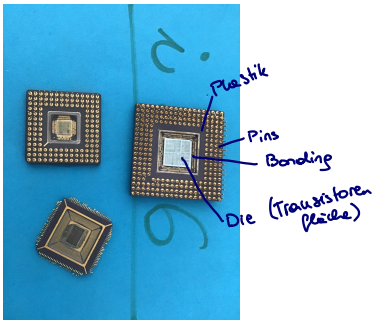
Technology Roadmap ITRS → (Moores Law: Halbleiterindustrie wächst sehr schnell an!)

1971 – 2300 Trans ~ 2008 731 · 10^6 Trans

Rechner werden schneller durch Microchips, Speicher wächst nicht so schnell ⇒ Cache

5 GHz war/ist die Obergrenze

Digitale Logik



Isolierung: Silizium Dioxid SiO_2 (0 Kelvin)

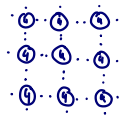
Schaltbare Verbindungen: Dotiertes Silizium (Silizium & Germanium) 2 B Transistoren

Feste Verbindungen: früher PolySi, heute Kupfer, Aluminium

aus Sand → Rohmaterial

Halbleiter: Silizium, diamantähnliche Kristallstruktur

isoliert bei 0 Kelvin



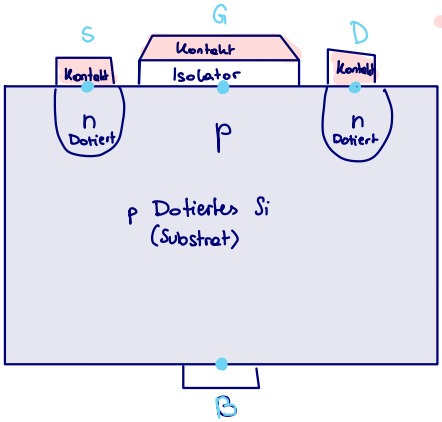
Dotieren:

- n Dotierung: Elektronenüberschuss (Elektronendonatoren) Arsen, Antimon
- p Dotierung: zu wenig e^- (Elektronenakzeptoren) Bor, Indium

→ es gibt auch Dotierungsstärke!, Legierung ist eine Art Dotierung

pn Übergang: keine Leitung aber mit elektrischen Feld angelegt fließt Strom zw np

Metall Oxid Semiconductor Transistor



Polysilicium, heute Kupfer

Source

Drain

Gate

Basis

4 mögliche Beschaltungsarten

?

Chip Fertigung

→ Sand ~99 % Reinheit durch Tiegelziehen (rund!)

→ Runterschneiden → es entstehen "Wafer"

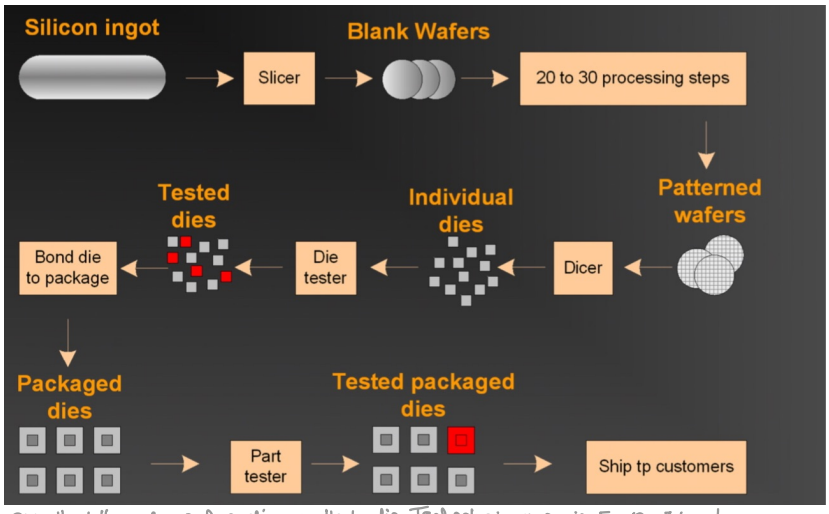
→ Muster hineinätzen mittels Photolithographie

→ zerschneiden (je größer desto weniger Verschnitt)

→ testen (kaputte makieren) mit Prüfspitzen

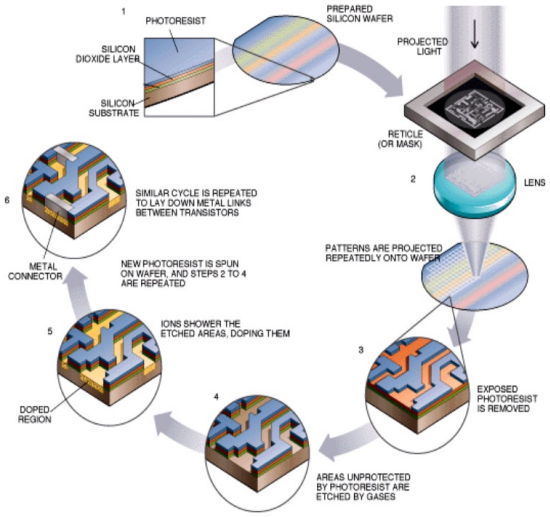
→ Packaging: Gehäuse mit Anschlüssen (Flip Chip → kein Package)

→ Bonding: Drähte mit Kontakten auf Packagingpins



Chip Herstellung so aufwendig → heute die Technologie von in 5–10 Jahren! entiert sich erst nach 10^6 chips oder so...

Photolithographie - Prozess:

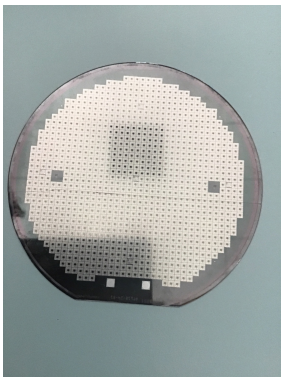


R

Wozu drehen die Masken?

Wie wird Licht richtig fokussiert?

Laserstrahl Elektronenstrahl



Wafer (weifer)

Feature Size λ

Maß für Technologiefortschritt

Kanallänge 2λ → 0.014 µm (Haar 100µm)

feature size wächst ⇒ Transistor kleiner, schneller, leistungsfähiger

Aber: Je geringer die Spannung U, desto weiter muss man Frequenz senken!

Subthreshold

Prozessparameter:

Kanal Länge 2λ, Breite 3λ

Oxid Dicke $T_{ox} < 1nm$

Diffusionsbereich Länge 3λ

Metallverbindungen Breite 3λ, Abstand 3λ

Größe der Dies bleibt konstant

Interconnect:

Verbinden der Leiterbahnen (früher Löten, heute Isolation & Vias)

Moderne Formen von Chips

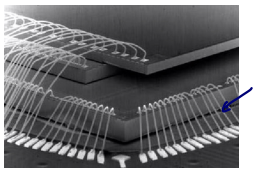
SoC – System on a Chip

Früher: Für jede Funktion ein anderer Chip

Heute: IP module, Schaltbilder, Codestücke auf 1

SOC – Pro: Kosten, Performance, Größe

SOC – Contra: Komplexität (mehr Prozessschritte), nicht alles integrierbar, suboptimale Technologie



Stacked Die Package

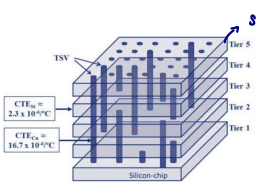
Bondings

MCM – Multi Chip Modul: Mehrere Dies planar → kleiner, billiger

mems: Hebelarm aus Silicium auf Chip (Widerstand – für Beschleunigungssensoren)

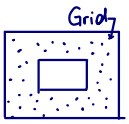
Stacked Die Package: mehrere Dies gestapelt, Entfernung Thermisch

↳ Through Silicon Via (TSV)



surface mount

TSV mit Layern



Grid

Ball grid array verschmelzen sofort einkapseln

Grenzen der Technologie:

Miniaturisierung: Elektron ist kleinstes Ladungsding → Quantenphysik

Isolatorproblem: nicht ∞ belastbar (zu hohe Feldstärke ☹)

Tunnelströme: steigen exp(x) bei kurzem Kanal (Isolier nicht 100%) Gate-Oxid

Quantenmechanik

Wellenlänge von Licht, Molekülgröße Photolack, thermische Leitfähigkeit begrenzt, Kosten

Wafer Dotierung stört im Silizium → Silicon an Isolator

Materialien:

1 V/m → 1 nV/m

1 GV/m → 1 V/nm

Transistor der Zukunft: Strained Silicon, Gate aus Metall, High k, SOI, 3D → Gate beidseitig

SOI: Isolator als Träger

Finfet: 3 Kanäle

Alternativen:

Quantencomputer (ja)

Nanotubes (vielleicht)

Molekular Elektronik (eher nicht)

Geschwindigkeit: Wellenausbreitung (natur)

Ladevorgänge (unvermeidbar)

verzögerung

Bewegung Ladungsträger (unvermeidbar)

Mit welcher Geschwindigkeit breitet sich eine Welle aus?

Komplexität: Aufwand, Design Crisis, Ausbeute, Anzahl an Pins, Leistungsverbrauch

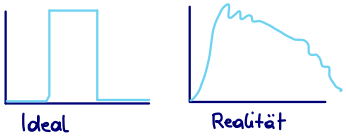
Multiprozessorchips: parallel programming

Aufbau logischer Gatter

Feldeffekttransistor

3 Anschlüsse: Gate, Drain, Source

Funktioniert wie ein Schalter (eigentlich analoges Bauelement



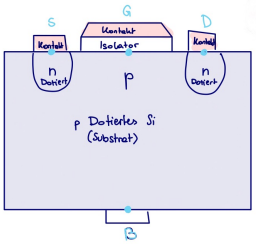
(Fast) kein Strom für Energiefluss nötig → Energieeffizient

Bipolartransistor hat immer Basisstrom für Funktion

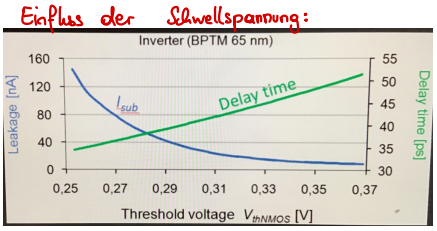
n Kanal Enhancement MOSFet

Spannung an Gate und Source → sperrt, kein Stromfluss
Schwellspannung U_{th} >100µV

Spannung an Gate, Source, Drain → Stromfluss



Einfluss der Schwellspannung:



Was passiert im Fet:

p → 3 ↑ zu wenig
n → 5 ↑ Überschuss

elektrische Kraft
"Gitterkraft"

U_{GS} bewirkt e Feld

$U_{GS} = U_{th}$ p Si löcher gefüllt → Elektronen passieren

Potentialunterschied von n und p → Gatter sind schneller bei hohen Temperaturen

Eingangskennlinie

Schwellspannung über Dotierung einstellbar

Leakage current (Strom der fließt obwohl es aus ist)

Je höher Schw Sp desto langsamer Trans

Ausgangsstrom I_{DSS}: Maximaler Strom den fet bei "Schalter geschlossen" führen kann; Regulierbar über Kanallänge L: Kanalbreite W → Formfaktor

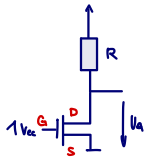
Formfaktor: $\frac{W}{L}$ W>L guter Leiter, großes W, größerer Transistor (Treiberstärke veränderbar)

p Kanal fet um Faktor 2 schlechter als n Kanal fet

n Kanal fet
- logisch 1: Schalter weak geschlossen
- logisch 0: Schalter offen strong

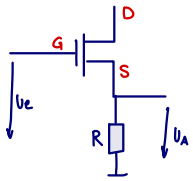
p Kanal fet
- logisch 1 offen strong
- logisch 0 geschlossen weak

Source schaltung ist besser als Source folger



Spannung wird invertiert

Strong 0



weak 1

CMOS Logik

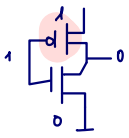
Kombi von pKanal & nKanal
↳ beide logischen Pegel aktiv
0 ↑ 1 gleichzeitig
zueinander komplementär

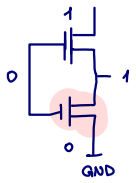
kombinatorische Logikzellen

CMOS Inverter (2 Transistoren)

Wenn man Abstraktion ausreizt, wird Kennlinie analog

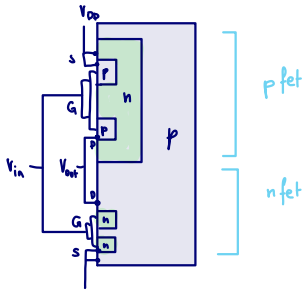
pfet auf
nfet zu





pfet zu
nfet auf

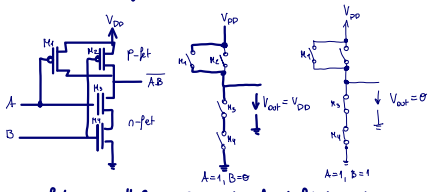
Technologie:



B – Anschluss "ganz unten"

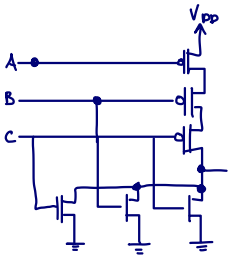
pFet in nFet integriert (Gates beide)

NAND



p-fets parallel → 0 an A oder B für Y = 1
n fets in Serie → 1 an A und B für Y = 0

NOR

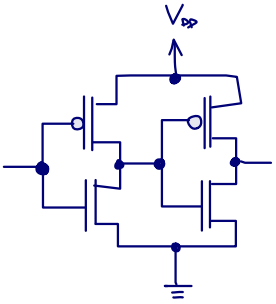


p Fet in Serie, 0 an A,B,C
↳ Y = 1

n Fet parallel, 1 an A,B,C
↳ Y = 0

alle 3 Leitan nur wenn Nuller anliegen

Buffer



2 Inverter, sonst nur schwache Wirkung

p Stack: aus pfet Aus: 1

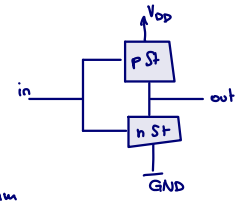
AND durch Serie

OR durch Parallel

Inversion am Schluss

Invertierter Eingang

$F(\neg x_1, \dots x_n, \vee, \wedge)$



zum Realisieren von Nand, nor, etc ...

n Stack: aus nfet Aus: 0

AND durch Serienschaltung

OR durch Parallelschaltung

Invertierter Eingang

nur AND/OR mit Inversion am Schluss

$\neg F(x_1, \dots x_n, \wedge, \vee)$

Tristate:

P	n	
ein	ein	→ V_{DD} mit GND → geht nix (Kurzschluss)
aus	aus	→ "Tristate", meistens Floating Pin (0/1/nix)

mehrere Ausgänge zsm schalten, nur einer aktiv

Wie funktioniert Tristate?

open drain

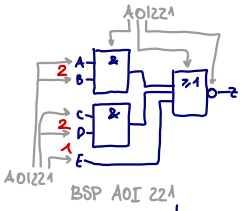
kein p Stack (Ausgang 0 durch n Stack erzwungen)

ist ein "And" Schaltung, Ausgang 1 durch R → Spannung bricht bei parasitären Störern

RC braucht länger als Gate Delay

Was ist Unterschied zu Tristate / open drain?)

AOI: & OAI:



BSP AOI 221

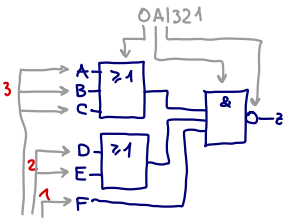
↓

2 eingang
2 eingang
1 eingang

1) Gleichung aufstellen — n Stack = Strong 0

2) Inversion zu Eingängen → p Stack = Strong 1

3) AND → Serie
OR → Parallel



BSP OAI 321

3 eingang
2 eingang
1 eingang

N Stack aus AOI / OAI bauen

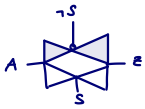
Inverter dazuschalten (→ was passiert...)

n Stack bauen → für p Stack: n Stack komplett invertieren, Serie parallel schalten

BSP Alarmanlage

p Stack zeichnen, n Stack zeichnen, verknüpfen

Transmission Gate

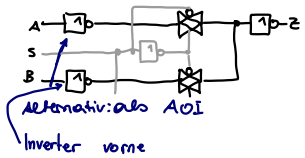


- p Kanal / n kanal fet parallel ⇒ INVERTER
- Schaltbare Verbindung zw 2 Leitungen
- bei A unbekannt ob high / low

Buffer: nicht schaltbar

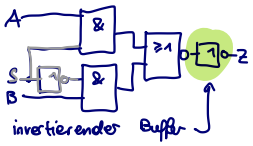
- Transmission Gate ist schaltbar

MUX mit Transmission Gate



Alternativ: als AOI

Inverter vorne



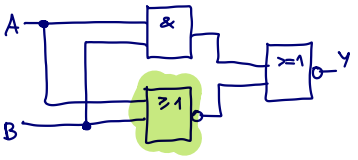
invertierender Buffer

Transmissiongate beidseitig abgeschirmt!

XOR

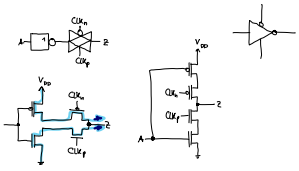
als Transmissiongate, 3 Eingänge, einer als schaltbarer Inverter (ungünstig) → 2 Transistoren in Serie :(

Lösung: XOR als AOI - NOR Gatter davor



Getakteter Inverter

- wie TG
- Signal invertiert, Takt = Steuersignal
- Inverter + Schalter → getakteter Inverter
- einlaufen benötigt Zeit: hold & set
- Buffer am Ende um Bitflip zu verhindern



Bei Latch & Flipflop

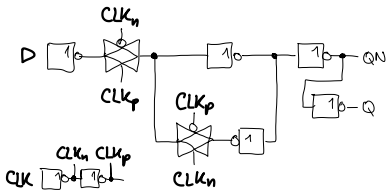
Gattereinheit GE

2 Transistoren → 0.5 GE
4 Transistoren → 1 GE

Latches Transparent, Hold (Setup Time & Hold Time)

Decision Window: Wenn Schalter betätigt, dann fix (bei Flanke metastabil!)

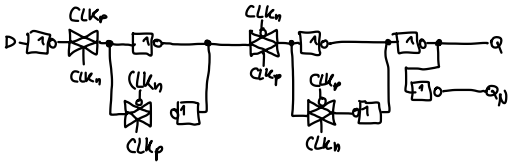
BSP Speicher kann immer metastabil werden → Grenze muss immer verschwimmen



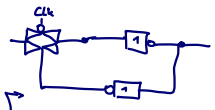
Aufwand: 7 Inv + 2 TG = 18 Trans ⇒ 9,5 GE

D Flip Flop Data, Clock

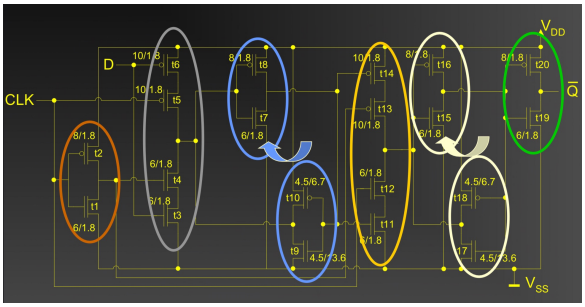
2 latches in Master - Slave



Theoretischer Aufbau mit TG



Quasistatisch (Kurzschluss von Nanosek)



Praktischer Aufbau - Implementierung

Pfeile: Speicherschleife Speicher kann immer metastabil werden → Grenze muss immer verschwimmen

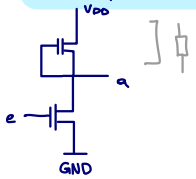
4,5 / 13,5 W klein / L größ 8/1.8 breiter weil p-Kanal fet (wg Löcher) 6/1.8

SR Latch: Metastabilität (Kurzer Puls, gleichzeitige Aktivierung)

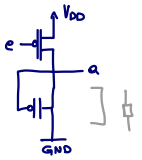
Register Array aus Flip Flops

Speicher: FlipFlop, SRAM, DRAM (D → S → D)

NMOS (nur n Kanal fet)



PMOS (nur p Kanal fet)



Fet als Widerstand

- Ausgangs I abhängig von Ausgangs U → saturation reach
- Formfaktor W/L

:

Bipolare Logikfamilien

TTL
Transistor-Transistor-Logic
Durch Bipoartransistoren
Stromverstärkung von 50 oder 100 fach
74xx früher Biplolar

ECL
Emitter Coupled Logic
Schnelles umschalten
nicht kompatibel zu CMOS/TTL
Glasfaser
→ Umschalter von Strompfaden

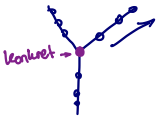
Bi CMOS Logik
hohe Ströme (Motoren etc)
BSP Autofensterheber
Kontaktfehler Fensterheber
↓
extern Löten, Treiber im Baustein

Design Flow eines ASIC

VHDL

Y Diagramm (Denkhilfe)

Sichtweisen auf Schaltung: Verhalten, Struktur, Geometrie



Systemebene
Algorithmische Ebene
RTL Ebene
Logik Ebene
Schaltkreisebene

Konvergenz: Design Flow

Designschritte beschreiben Prozessübergang

behavioural

- Specs – Exakte Formulierung des Produktes (2B Synchronisierungsschaltung, viele non functional Parameter)
- Validation
- Design Entry – Specs in Computer/Compiler übersetzen / Basis / Dokumentation → Möglichkeiten
- Behavourial Simulation

Pre Layout

structural

- Compilation – verhaltensbasierte in strukturelle Darstellung (Analyse, Elaboration, Transformation, Minimierung, Timing Analyse)
- Functional Simulation
- Technology Mapping – Target Library (welche Flit pitts, ...) → Zieltechnologie festlegen (Generische Gatter auf Logikelemente)
- Pre Layout Gate-Level Simulation

POST LAYOUT

- Partitioning and Placement – Designaufteilung (wenige Verbindungen, Größe, Speed, Verlustleistung) & Verteilung Logikelemente / Pinbelegung & Routing
- Postlayout
- Manufactory & Download
- Test
- Chip Complete

Ebene	Verhalten	Struktur	Geometrie
System	Inputs : Keyboard Output: Display Funktion: Umrechnung,...	Speicher, CPU, IO, Control	Q1, GND, PLCC84, VDD, D0, D1, RW, A7, A6, A5
Algorithmisch	while input Read „Schilling" Calulate Euro Display „Euro"	Speicher, 16, µP, IO-Ctrl, RS232 Interface, PS/2 Interface	µP, PS/2, IO-Ctrl, RS232

Registertransfer (RTL)	case A
	when `1` then
	nextB <= C;
	nextstate <= idle;
		RAM, Register, ALU, Counter	REG, ALU, Counter

Logik	D = NOT E C = (D OR B) AND A	E, B, A, C, &, INV1, AND2 x3, OR2	
Schaltkreis	$\frac{dU}{dt} = R\frac{dI}{dt} + \frac{I}{C} + L\frac{d^2I}{dt^2}$		

Detailliertere Auflistung des Design Flows

FBGA - Ball Grid Array
BUS - Ein Weg viele Daten

① Specs

② Design Hierarchie:
Volladdierer aus 2 Halbaddierern
Vectored Instance (Parallelschaltung → BUS)
Schematic Entry ↳ 1:4 : 4 Bit Latch

BCD (Binary Coded Decimal) 0001 – 0111 → 0-7

③ SW vs HW
Hochsprache vs VHDL → alle in Assembler

Compiler → SW: Zeitfreie Beschreibung, Funktion ohne Timing
→ HW: Timing notwendig!
"file open" / "wait" in HW nicht implementierbar
→ VHDL (nie für HW konzipiert)
→ Verilog – Simulationssprache
→ C

④ Optimierung: Kosten & Nutzen
Synthese: Optimierungskriterium (speed v area v effort) OK
Randbedingungen (Timing, Power, Sperren) einhalten RB

⑤ Strategien für Partitioning:
Konstruktiv – Zamstöpseln
Iterativ – Sachaufteilung, 2 Elemente vertauschen → in Richtung Optimum
Simmulated Annealing: Schmelztigel → je kühler umso weniger will ich Verschlechterung.

⑥ Placement
Floorplanning
Placement
Grundproblem: Leitungen optimieren bevor sie verlegt wurde
↳ Lösung: Heuristik
Delay: Gatter, Leitungen
massives Problem zw Gate Delay & Interconnect Delay

⑦ Routing: Verbindungsherstellung
Interconnect - half pitch
CLK: minimaler Delay / Skew (Treiber & Buffer zum Ausgleichen) ← Zeitversatz
Stromversorgung: Strom / Querschn. $\frac{1mA}{\mu m^2}$ maximal, sonst Elektromigration → Chip stirbt
↳ gewaltige Stromdichte!!! 0,5 MW zu 230V!

Wir haben: Positionen, Library, Routing ...

Simulatoren: kapazitive Kopplungen, ...

Was versteht man unter Back-Annotation?

Verifikationsschritte

Validation - "Do I built the thing right or do I build the right thing ?

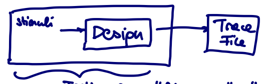
Simulation - "If SW Developer had to build planes" Flugzeug

ASIC - sehe nur Pins außen! (Zugänglichkeit)

Fertigungskosten, Dauer, Prototypen

Testbench: Hardwarecodierung von TR in größere Beschreibung → Signale durch → auslesen

Testbench - simuli
→ nur bestimmte Simulationen
| Trace Files
\ assertions



Testbench: "file open", "wait verwendbar, weil es nur simuliert, nicht gebaut wird

ist keine Garantie!

Arten von Simulationen (auf allen Ebenen)

- Behavourial Simulation: Testbench, specs, whats happening ... (Des → Beh. → Syn)
- Functional Simulation: Code richtig interpretiert, Timing! (Syn → Fun → Tech)
- Pre Layout Gate Level Simulation: Latch, FlipFlops
- Post Layout ... Simulation: Timing — wichtigster Schritt

Simulation notwendig, aber nicht auf allen Ebenen, auf jeden Fall Post Layout Sim!

Viele Sims gut zum genauen Fehlerorten.
↳ Fehlermaskierung fällt auf!

Ablauf einer Simulation:

mixed Level: Alle Layer

mixed mode: analog & digital

sign-off: Vorlagen von mixed level/mode als "Abkommen"

① Logikpegel
tristate? 0 & 1 zu wenig
→ 9 stellige Logikpegel IEEE

AND	U	X	0	1	Z	W	L	H	-
U	U	U	0	U	U	U	0	U	U
X	U	X	0	X	X	X	0	X	X
0	0	0	0	0	0	0	0	0	0
1	U	X	0	1	X	X	0	1	X
Z	U	X	0	X	X	X	0	X	X
W	U	X	0	X	X	X	0	X	X
L	U	0	0	0	0	0	0	0	0
H	U	X	0	1	X	X	0	1	X
-	U	X	0	X	X	X	0	X	X

Signal Resolution Table für ein OR Gatter

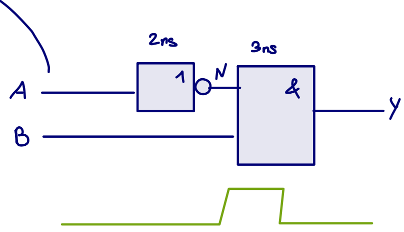
② Ereignisgesteuerte Simulation

BSP | Annahme: vorher stabil

t	event	A	B	N	Y
0- (vor Null!)	init	1	0	0	0
0	A↓	0	0	0	0
0+Δ (Deltatiming, findet gleichzeitig statt, ist aufgeteilt)	B↑	0	1	0	0
2	A↑	1	1	0	0
2+Δ	N↑	1	1	1	0
3	B↓	1	0	1	0
~~3+Δ~~	~~Y↓~~				~~0~~ Bleibt 0 (theoretisch)
2+2ns=4	N↓	1	0	0	0
2+3ns=5	Y↓	1	0	0	1
3+3ns=6	Y↓	1	0	0	0

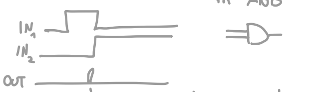
(0 → A↓: unverändert)

gleichzeitig, Ablauf nach Reihenfolge :)



Abbruchbedingung angeben

anderes BSP 2 Signale, zeitgleich in AND



↓kurz auf 1 WG Δ !

Δ ... Virtuell

Schaltung neigt zu Glitches: (Problem von 3 und 3+Δ) bei guter Schaltung eigentlich wurst.

Statische Timinganalyse

Prototyp: notwendig, nicht hinreichend wegen Cornercases

wie lange muss Taktperiode sein, dass sich "Logikwolke" ausgeht? → Backtracking (langsamster)

$$\frac{1}{f_{clk,max}} = \max\left(t_{delay,DATA,ij}\right) + t_{su} \dots - \min\left(t_{delayD,CLK,i}\right)$$

(normalerweise 0, kompliziert → wird weggelassen)

$f_{clk,max}$: sucht langsamstes und rekt das als Timing

Statisch vs Simulation

- Statisch: ✓ kritischer Pfad, ✗ vorteilhaften Pfad
- Simulation: Info über Eingangsvektoren ✓, plausiblen Pfad ✓

Library Datasbook

charakterisiert verfügbare Basiszellen, definiert Timing (Variationen durch Derating-Faktoren)

$$t_{delay} = t_{int} + C_{load} \cdot t_{ext}$$

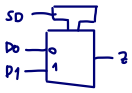
Hohe Temperatur und niedrige Versorgung verlangsamt den Chip

W-L einstellen: W-L größer → Kapazität steigt

Mux: $t_{delay} = t_{int} + C_{load}\, t_{ext}$

Fixes Delay & lastabhängiges Delay

BSP: D0↑ schneller, aber stärker lastabhängig



from input	to output	intrinsic	extrinsic (ns/pF)
D0↓	Z↓	1.42	2.10
D0↑	Z↑	1.23	3.66
D1↓	Z↓	1.42	2.10
D1↑	Z↑	1.23	3.66
SD↓/↑	Z↓	1.42	2.10
SD↓/↑	Z↑	1.08	3.66

Derating Factors

Kühl: schneller warm: langsam

wenig Spannung: langsam viel Spannung: schnell

Corner Cases unbeliebt → Wahrscheinlichkeitsverteilung

	4,5 V	4,75 V	5 V	5,25 V	5,5 V
−40°	0.72	0.73	0.68	0.64	0.61
0	1	0.93	0.87	0.82	0.78
25°	1.14	1.07	1	0.94	0.9
85°	1.5	1.4	1.3	1.26	1.2
100°	1.6	1.48	1.41	1.34	1.28
125°	1.76	1.65	1.56	1.74	1.41

(1,76: worst case)

WORST Case: 1,176 (fast doppeltes delay!)

Formale Verifikation:

vollständige Überprüfung → Darstellung des Designs als Modell (gibts Deadlock? wird Zustand A erreicht?) (Verifikation, Werkzeitation)

modell & formal

- Model Checking → Überprüfen von Bedingungen/Eigenschaften
  - vollautomatisch, kommerzielle Tools, OUT: pass/fail (nur obs geht oder nicht), Nachträgliche Prüfung (für welchen Timing geht das noch dann?)
- Equivalence Checking → Übereinstimmung von zwei Modellen
  - Formal: manuell, Mathematisch, Parameter als Variable, Design Entscheidungen

Design Tools

unabhängig:

- Design Entry, Logic Compiler,
- Simulation, PPR

Laufzeit erst wenn es fertig ist → Heuristiken zur Berechnung verwenden

Speichertechnologien

Speicher: strukturierte Anordnung von Informationsträgern

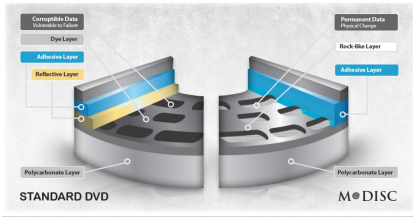
Speicher: strukturierte Anordnung von Informationsträgern

Funktionsprinzip:

Analoger Speicher: Magnetisiert, Lichtton bei alten Kinofilmen

Digitaler Speicher: 1 bit = 1 Informationsträger mit 2 unterscheidbaren Ladungszuständen
Spannungsregeln
Magnetisierungsrichtungen Festplatten
Topologien (Schmelzvorgang)
optischen Reflexionseigenschaft BluRay

BSP Optischer Speicher



Eigenschaften

im Betrieb beschreiben: Reg File, Read/Write Memory

im Betrieb nicht beschreiben: Read only Memory (ROM) → non volatile (nicht flüchtig), schreibgeschützt

nur schreiben, nicht lesen: Write only WOM, Log File (Blackbox)

Auslesen von Speicherinhalten im Betrieb immer möglich!

Speicherzugriff:

Random Access: zu jedem Zeitpunkt (Wahlfrei) zugreifen mittels Adresse

FILO: First in Last out (keine Adressen notwendig!) a,b,c,d → [d|c|b|a] → d,c,b,a

FIFO: First in First out (Adresse nötig!, mehr Rechenaufwand) a,b,c,d → [d|c|b|a] → a,b,c,d

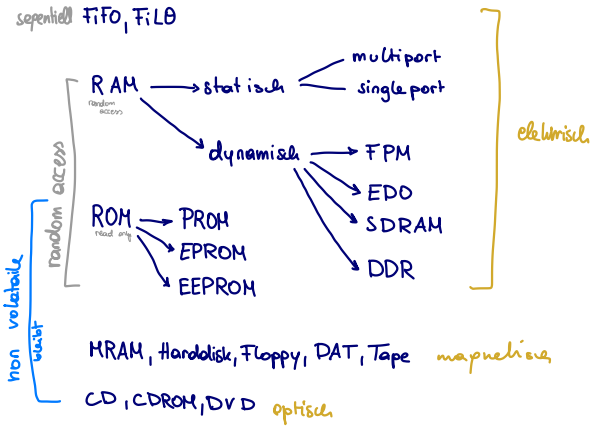
NVM: Non volatile Memory (Inhalt bleibt erhalten) ROM, Harddisk, CDROM

VM: Volatile Memory (weg, wenn Strom aus) Ram, SRAM, DRAM

(Festplatte - Mischform, Wahlfrei, kriegs aber erst wenn Daten "vorbeikommen")

BSP
FPGA → serial ROM
Festplatte → Zugriffszeit abh. von Position

Arten von Speichern



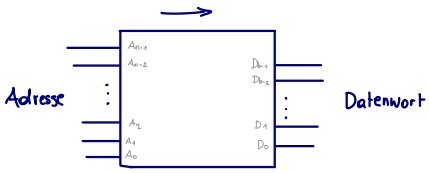
MRAM: Speicherbaustein mit Bitzellen auf magnetischer Basis
nicht flüchtig

RWM und ROM sind beide RAM

ROM (Read only memory)

Arten 1)

- Adresse anlegen → Daten raus
- Jede kombinatorische Logikfunktion kann dargestellt werden
- kombinatorisches Bauelement



Anwendung: Logikfunktion, Prozessor (Befehlscode), Decoder, Linearisierung,

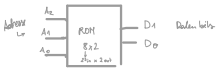
Schaltung entwerfen die Maximum von 2 Zahlen ausgibt.

BSP Tabellenberechnung: EPPROM als Befehlsspeicher

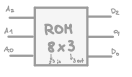
Adresse	Daten	
0x08	00001000 00010100	add r1 r2 r5
0x09	00001001 01001001	cmp r2 r0 r4
0x0A	00001010 11000101	beq r4 r7 r3
0x0B	00001011 11100010	or r7 r6 r5

EPPROM

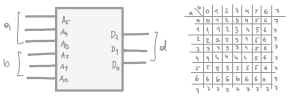
BSP Wahrheitstabelle



BSP Gray Decoder



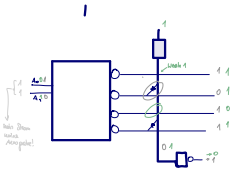
BSP

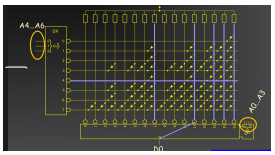


Prinzip:

1) Decoder wandelt Adresse um (Ausgänge invers)
2) Bits von Datenwort in Spalten → Pull up auf 1
3) Diode an Kreuzpunkt von Spalte/Zeile → auf 0 (sonst 1)
4) Programmierung: Setzen von Dioden

2 Adressbits → 2^2 Möglichkeiten!





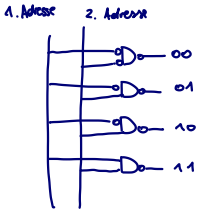
Problem: ≥ 2 Adressbits → 2^n Gates mit n Eingängen?!
1M x 1ROM → 1048576 Gates mit je 20 Eingängen!

Lösung: 2D Codierung

Decoder: Je Ausgangbit 1 AND mit je 1 Eingang (teilweise inv.) → genau 1 spricht an

MUX: Je Eingangbit 1 AND mit je 1 Adressbit + 1 Datenbit

↳ Alle ANDs vereint!



Mask ROM:

Dioden setzen oder nicht (als Traus realisiert)
Maske → Jeder Fehler ist fatal (Speicher nicht veränderbar)
Fertigung
PRO: Billig, Störfest, nicht programmierbar
Mehrere Datenbits → mehrere Arrays (BSP McDonalds Spielzeug)

Floating Gate (ex):

Gate hat keinen Anschluss → Tunneleffekt
Programmieren → wird Ladungsträger: Schwellwert
Störungen: Bit fällt um (nur bestimmte # von ~10.000 Prog. Zyklen)

OTP ROM

PROM
Überall Dioden
selbst Programmieren (1x brennen)

BSP 3 bit Decoder, 2 bit MUX
möglichst quadratisch → Hoher Gewinn
2B MUX 4x Bauen (array!)

UV EPROM:

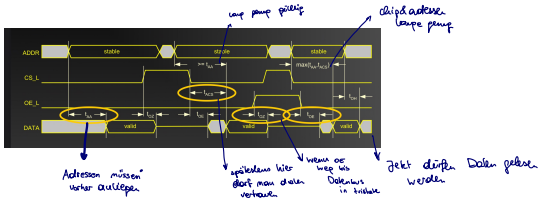
Erasable
Transistoren mit Floating Gate
löschbar mit UV (Fenster - zukleben)
Störanfällig
Daher RegFile: Invertieren, alles ausgleichen

EEPROM:

Wie UVEPROM

electric erasable
Einzelne Zellen löschbar (elektr. Feld)
Write slower
≈ 10.000 Zyklen pro Zelle

ROM Timing:



Warum haben UV Eproms Fenster? zum Löschen

Solid state drive
:: ↳ Alterung, Isolator leiert aus
:: ↳ als Regfile ca. 10ms!

Steuersignale

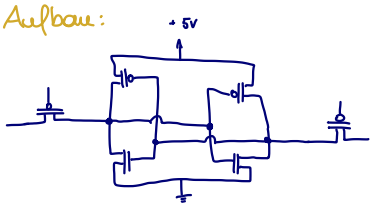
Adressen: Speicherzellenwahl
Chip select: "du bist gemeint"
Output enable: Ausgangstreiber (low aktiv)

Tristate bei 0

Datenbus, Ram, Peripherie

SRAM (static random access memory)

Aufbau:



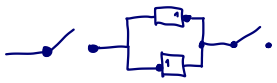
Symbolisch wie Latchzelle

"Inverterloop"

Inverterschleife wird von Latchzelle durch 2 Transistoren angesteuert

↳ 2p Kanal fet (1, 0 erzwingen)

Vereinfachung



Kein Buffer nötig, weil geordnete Struktur!

Write: Kein Transistor nötig (nur einfacher Fet) weil einer immer Strom

Kein Transitiongate für Feedback nötig (wenn Treiberstärke richtig) → Feedback schwach

Speicherzellen einfacher gehalten durch Verwendung von Steuerlogik und Ein/Ausgangsbuffer für alle Zellen

Steuersignale:

gleich wie ROM (Adressen, Chip Select, Output enable, Write enable)

Bus Zusammenhängen: Write & Read (In|Output) über gleiche Pins geführt → Hälfte der Pins gespart

Readtiming wie bei ROM

Writetiming Daten müssen vorher & nachher eine Zeit lang stabil sein, Adressen müssen gehalten werden

Rising von Chip Select und Write Enable, Pulsbreite = Write CWP

Synchrones SRAM

Eingangssignale Synchron in Register

Read Daten asynchron flow through output

synchron: Pipelined output

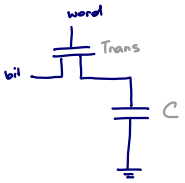
Zugriffe: überlappbar, über Pipeline

Cache|Burst Mode: ganzes Paket einlesen (automatic. increment.)

DRAM

Aufbau

Bit wird als Spannung in Kondensator abgelegt (HI / LO)
Fet Schalter offen → Ladung erhalten
Write (WR): Laden / entladen des Fet
Read (RD): Auslesen der Spannung → Sense Amplifier (Auswertung)
→ Konsumiert Ladung (→ write Back nötig!)



tl;dr:
Kondensator: geladen oder nicht?
Schalter: Schreiben und lesen

Refresh: Abfangen bevor bit kippt! (10- 100ms!)
Timer läßt C ständig auf $3 \cdot 10^{-15}$ F

Steuersignale

halbe Adressanzahl
Write Enable: RD oder WR → anderes muss stoppen!
Row, Col hintereinander anlegen
Ras Row Adress Strobe
Cas Colum Adress Strobe

Read Timing: Cas = Output enable
Flanke: gültig
Pegel: output enable
Ras weg → Zeile refresh

Write Timing: Colum adress an → lesen stoppen!

SRAM vs DRAM

Störfestigkeit: Einzelfall (nicht verallgemeinerbar!) ↳ wieviele Bits sind stand der Technik?

DRAM = 4 · SRAM
6 Trans → 1 Trans, 1 Cond

DRAM: hohe Speicherdichte, als Hauptspeicher

SRAM: höhere Leistung, schnell, kein Refresh, als Cache für embedded speicher

Zugriffsarten

Page Mode: (2 fach schnellerer Datenzugriff in ROW)

Zugriff → RAS active, ROW adress in Latch → Zugriff auf gleiche Page ROW, nur CAS Pulse nötig

EDO: Extended Data Out Ausgangssignal von OE → Überlappen von Zugriffen

Schematisierte Zugriffsarten

Standart: row - col - data - row - col - data

Page Mode: row - col - data - col - data

EDO: row - col < data / col > data

SDRAM: row · col - - data - row - col
DDRAM: - - row - col - - data

SDRAM = Synchro Dynamic RAM nur eine Taktflanke)

DDRAM = Double Data Ram (Speicher einbetten → Takten → Pipelinen)

verwendet beide Taktflanken → wird schneller → ca 50% von Tastverhältnis, sonst ist Timing dahin!

DDR 3 Signallaufzeiten, Reflexionen → noch schneller

standartisiert, geht bis in Leiterplatte (wie beschaffen, das Timing passt?)

MRAM

Aufbau:

"Spindependent tunneling"

Write: Magnet drehen (umpolen) → mittels Spule, Magnetfeld

Read: Strom → Tunnelstrom groß oder klein?

alternativ: GMR / Pseudo Spin Valve

Features:

non volotaile

kleine Einzelspeicher wie Ram ansprechen

nicht für Cache

so schnell wie DRAM, 40% kleiner

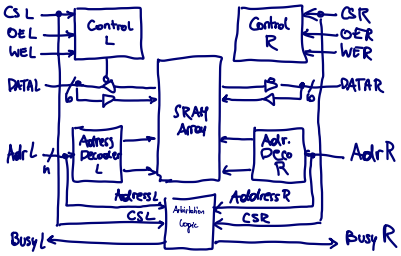
Stromsparend

Multiport RAM

Ein Ramarray mit Werten, mehrere Ports greifen zu

Doppelte Steuerlogik für RD, WR, CS, OE + "Busy" Signal

Doppelte Adressdecoder & Doppelte Interfaces (Pins)



Arbitration Logic: WIR will a Busy Flag

Semaphor Bit: Koordinierter Zugriff auf gemeinsame Ressourcen

Setzen von S. Bit → Request
Löschen von S. Bit → Release

DPRAM mit integriert

Anwendung:

DPRAM: Adresse hat Bedeutung

BSP Temperatur: WO? Wie viel?
→ Daten nicht immer refreshed
Block: WR auf Adresse während RD

FiFo: Adresse sinnlos
Empty Flag: Blockieren
Kopplung von Zeitverhalten

FiFo Prinzip

nicht wahlfrei zugreifen, Speichert in Tiefe

WR/RD überlappbar

Flags: (Hilfe für Verkopplung von 2 Systemen, die Daten austauschen)

Full Flag: alle Speicherzellen geschrieben, noch nicht gelesen

Half full Flag (Half emty flag): Vorwarnung für Speicherverwaltung

Emty Flag: Keine Daten zum RD im Speicher (WR, sonst RD undefined!)

Ringspeicher: Daten listenweise sortieren (internes Pointermanagement)
Was lesen wo nix steht → Pointervergleich
Fifo voll: schreiben?
Writepointer = Readpointer?

Errordetection & Error correction

Speicher schützen? → Fehlererkennung mittels Redundanz

D=0 < D=1

Hamming distanz 4 bit für 16 < 5 bit für 16

EDC Bitfehler erkennen

ECC Bitfehler erkennen & korrigieren

Parity: Prüfbit an Datenwort anhängen D=2

Hamming Code: Multibitfehler → besser für breitere Datenworte

Blocksignator CRC: Hash bilden → Länger gespeicherte / übertragene Blöcke, in Hardware: linear Feedback chip Register, kein unbegrenzter Schutz!

Aliasing

ex:

Kondensator in CMOS ?

—||—  Trenchcondensator

ex: https://en.wikichip.org/wiki/intel/loihi

Zieltechnologien

ASIC = Application Specific Integrated Circuit

Die Aufbau → Interconnect (Metallisierung) 5-12 Layer
↘ Zellen (Silicium) 5-10 Layer (inkl Transistoren)

Full Custom ASIC: Alles vom Designer festgelegt
anwenderspezifisch

Standard Cell ASIC: Zellen in Library mit fertigen Bauelementen
CBIC (bereits getestet → funktionieren)
Block mit Anschlüssen.

BSP CBIC in Software:
Vorgefertigte Funktionen
Elemente sind Betriebsgeheimnis

Zellen Library

für Standard Cells, FPGAs, Gate Array Macros
von Hersteller / Vendor (koop. mit Hersteller)
vorgefertigte Funktionen

IP Core (fertige Funktionseinheit, kann als Macro eingebunden werden)

Hard Macro: Black Box (Leitobahnen, Dotierungen vorhanden)

Soft Macro (bevorzugt) selbst synthetisieren → technologieunabhängig, VHDL, System on a Chip

Gate Array (generische Lösung) MGA (Mask Gate Array)

Structured GA vs Structured Cell
↓ billiger
↓ weiter optimierbar
Cores frei wählbar

Wafer haben Basiszellen → als Macros realisiert

Metallisierung von Anwender entschieden

- channelless Gate Array: Trans Anschlüsse wählbar, Kanäle besetzt, Interconnect: Unbenützte Trans
- channeled Gate Array: Trans Anschlüsse fix, # an Kanälen frei → für Interconnect
- structured Gate Array: (embedded GA) gewisse Stücke für RAM, ROM, Basiszelle frei gehalten

Programmable Logic Device: alle Layer fix, Interconnect programmierbar, vor. Matrix aus Macrozellen

ROM, PLA, PAL, CPLD, FPGA

ROM: (Logik wandelt Adresse in Daten um)
↳ programmierung: EPROM (elektr, lösch), PROM & OTP (elektr, perma), mask progr. ROM
↳ löschen: UV EPROM (UV Licht), EPROM (elektrisch)

Programmable Array Logic

PAL (nur AND Array)

PLA (AND, OR als Core Programmierbar)

Heft to know:
FPGA = BGA = Ball grid Array

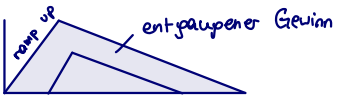
ASIC Technologien

		cell design	cell placement	cell connect
Full Custom		user	user	user
Semi custom	Standard Cell	library	user	user
	MGA: channelless, channeled, structured	fixed	fixed	user
	PLD: ROM, PAL, PLA, CPLD, FPGA	fixed	fixed	fixed (also programmable!)

3 Arten von Programmierbarkeit: Interconnect, Macrozelle, I/O Zelle

ASIC → Break Even (Marktanalyse)
FPGA (billig) → Massenproduktion teuer
CBIC (teuer) → Massenproduktion billig

Gewinnmodelle



ASIC ~ 0,5 Jahre
FPGA ~ 1 WO
selbst designt :-

Programmierbare Logic Devices (CPLD & FPGA)

programmierbar bei FPGA: Funktion in Macrozellen, Funktion in I/O Zellen, Auswählen der Verbindungen

programmiert durch "schaltbare Verbindungen" (wenn Verbindungen weg ist es ein ROM Transistor)

Antifuse

irreversibel (durch thermische Zerstörung v Isolationsschicht)

non-volatile (bootfähig), radiationhard

Kopierschutz

✗ Alterungsprobleme (irgendwann wird nicht mehr gut isoliert)
"Kontaktwiderstand"

1ns delay (zw Wolfram, Si, Al)

Transistor

irreversibel (gründliche Simulation)

SRAM Konfiguration

"in System" Speicher überschreibbar → volatile

✗ störanfälliger, Platzbedarf

EEPROM Konfiguration

non volatile, überschreibbar

so groß wie Antifuse

✗ störanfällig

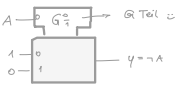
Programmierbare Logikzellen – Macrozellen

MUX basierte Logikzellen: mittels Shannons Erweiterungstheorem (nicht zu verwechseln mit Microcontroller!)

Funktionsweise: Die Variablen / Logikfunktionen werden schrittweise mit 0, 1 ersetzt → MUX Baum? durch zweistufige Logik ersetzbar

Jede Variable kann beim G-Teil angelegt werden

BSP:



Realisierung: Inverter, AND, OR, Identität werden alle mit einem MUX dargestellt.

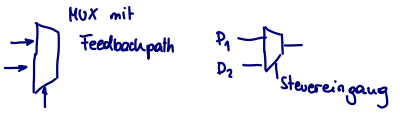
Alle können mit 2 Variablen (als Latches = $\frac{1}{2}$ FF (Flipflop)) manche mit 3/4 Variablen

ACT 2: MUX hat 2 Eingänge

BSP Supercluster

→ R und C müssen nicht / können verbunden sein

→ Combinational Logic – Carry Logic, MUX Logic (Geschwindigkeit ist Kriterium)



(siehe Datenblatt)

R: FlipFlop löschen, CLK setzen

Look up table basierte Logikzellen (LUT)

Verknüpfungen von n Variablen eindeutig darstellbar (Wahrheitstabelle)

4 in 1 out (1 bit Speicher (bestimmt Verknüpfung), 4 Adressen (Eingang), Wahrheitstabelle) → allg. als $2^n \times 1$ Bit

LUT theoretisch als RAM verwendbar (zu teuer)

konstantes Timing

BSP Altera / Intel Logic Element

DSP Block ist Multiplier (zur Berechnung von FFT, Faltung)

LAB – Logic Array Block

Wird sehr schnell sehr komplex!

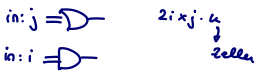
adder carry chain:
https://www.intel.com/content/www/us/en/programmable/documentation/wtw1441782332101.html#wtw1443170033156

PAL Logikzellen

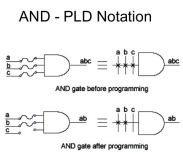
Pull up R, Collector & Treiber / open drain

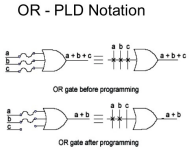
Direkte Implementierung der DNF

Wired AND, connecting OR → FF für sequentielle Logik



Notation: Horizontal AND, Vertikal signale, Kreuzungspunkte – Programmieren eher





Verbindungen die wir nicht wollen: Transistoren

XOR, invertieren

Wie realisiert man programmierbaren Inverter? Dioden

Wozu braucht man programmierbaren Inverter?

Parallel Expander: doppelte Durchlaufzeit, Kaskadierung von Stufen

CPDL durch großes Interconnect

Logic Expander: zusätzliche Durchlaufzeit, löst einzelne Produktterme (und verarbeitet diese schon zuvor!)

BSP $(\neg A \wedge C \wedge D) \vee (\neg B \wedge C \wedge D) \vee (A \wedge B) \vee (B \wedge \neg C) =$ Vorverarbeitung, stellt 4 nach 2!

$= [\neg(A \wedge B) \wedge C \wedge D] \vee [B \wedge \neg(\neg A \wedge C)]$

programmierbarer Inverter: Verarbeitung / Umwandlung & richtige Ausgabe inverser Funktionen

BSP minimale DNK aus KV Diagramm

HOW? Bitstream an bestimmte Zellen

LAB:

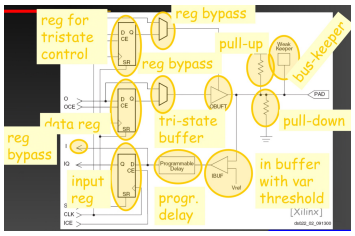
Scrubbing: Konfigurationsspeicher während dem Betrieb überschreiben

Programmierbare I/O-Zellen

Wir wünschen uns:

Eingang - Super Schwellwerte (versch. Spannungen), direkt Input/ Input FF, Timing mit Verzögerung

Ausgang - Super Schwellwerte, passives Pull Up / Pull down, direkt Output/ FF Latch, Tristate & Open drain, Flanke/Treiber wählbar

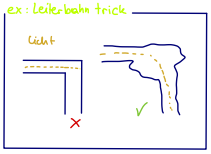


PAD als Eingang → out tut puffern ⇒ tristate (OBUFT)
in tut puffen ⇒ weg (IBUFT)

bus keeper = schwacher Treiber

entweder pull up oder pull down (beides nicht möglich!)

MUX = Steuereingang kann an ROM hängen



Programmierbares Interconnect

Wir bekommen alles, was wir uns wünschen, aber die Kapazität ist weg

nix kurzschließen

(maximale Flexibilität (viele Verbindungen) ∧ langsam) ≠ maximale Geschwindigkeit ∧ minimale Kapazität)

→ Direct connect (nomen est omen) - Signal wird mittels Meshierung gewählt

→ carry logic (ohne Antifuse!)

Konfig. Error → Radiation hard für FPGA

Signale in PIA (völlig sym. globaler Bus von jeder Quelle zu jeder Senke!

Quellen: ∀ I/O Pins, ∀ dedicated Input, ∀ Macrocell Outputs

Senken: LAB Inputs

CPLD - Interconnect

hochregulär & symmetrisch (einfache Struktur, voll verbunden, PPR einfach → Timing vorhersehbar ✓)

bestimmt Funktion (Flut & Struktur fix → PLD)

→ Konfiguration

on chip flash memory OR In-system Programmierung OR JTAG (over system/USB)

non volatile → Security bit (damit niemand klaut) → Reverse engeneering mittels Bitstream kann die auf "Gebeimcode" → nichts ist sicher!

Actel Interconnect

Antifuse → effizienk Verbindung

lokal = regulär, highway = variabel (Timing, PPR schwer, highway = Flaschenhals)

bestimmt Funktion (MUX, Signale wohin?)

FPGA Interconnect

hochkomplexes Netzwerk (mehrstufig, Timing kann vorhersehbar)

Routing, Funktion getrennt (Flut durch LUT)

→ Konfiguration (ich muss SRAM Programmieren)

durch externen Microcontroller konfiguriert OR Interfaces OR Auswahl durch Pinbeschaltung

or (seriell von PC, Config device)

· seriell, parallel, jtag

HardCopy Devices

FPGA → ASIC

1 Nacht Lieferzeit & ich darf selbst basteln!

PS: Umprogrammieren von FPGAs im Betrieb = Hurra: Design!

Datenblatt für Labor: EP1 S25F672CC
Konvektion:
transient:
crowbar:

Datenblatt - Angaben

Grenzen:
Absolute Maximum Ratings: Limits (geht man drüber, ist es hin)

Temperaturen 1) ambient: 0 – 70°C kommerziell
-40 – 85°C industriell
2) junction: 0 – 85
-40 – 100
-55 – 125 military

Temperatur

Fehlerrate steigt mit Temperatur exponentiell, chip wird langsamer

→ junction Temperatur (unmittelbar am Die)

messbar: case & ambient
Gehäusetemperatur Umgebungstemperatur

vereinfachte Wärmeleitungsgleichung: $T_{junction} = T_{ambient} + P \cdot \vartheta_{JA}$

P... Verlustleistung

ϑ_{JA} Wärmewiderstand (junction to ambient) [K/W] (Kelvin pro Watt)

ϑ Wärmewiderstand: Maß für Vermögen zur Wärmeleitung

spezifisch: Konstante
absolut: [K/W], je größer ϑ desto ...

Was ist Wärmewiderstand? Einheit

ϑ_{JC} Wärmewiderstand junction/case → Gehäusetyp

ϑ_{CA} Wärmewiderstand case/ambient → Gehäuse, Einbaulage, Kühlkörper, Lüfter

ϑ_{JA} Wärmewiderstand junction/ambient → $\vartheta_{JA} = \vartheta_{JC} + \vartheta_{CA}$

- Lüfter (erzwungene Konvektion)
- Kühlrippen

Warum schmilzt Diode nicht?

Daher: Temperaturfühler (Diode hängt an Die, direkte Messung von $T_{junction}$

$$\left.\frac{dU_D}{dT}\right|_{I_D} = -2\,mV/K$$

Komponenten der Verlustleistung

Batterie:

statischer Anteil: tritt auf wenn Chip nicht getaktet ≙ zusätzlicher Strom trägt nicht zur Erwärmung des Chips bei
↳ quiescent current

Bei CMOS: Tunnelstrom, Leckstrom, Sperrstrom, Ströme über Pull up Widerstände

dynamischer Anteil: Ladeströme dominieren (switching current) OR transiente Kurzschlüsse (crowbar current)

Verlustleistung durch periodisches Schalten von 0 auf V_{DD}: $P_{laede} = C_{zepu} \cdot f \cdot V_{DD}^2$
↳ selten umladen, geringe Spannung V_{DD}

Verlustleistung proportional zu Kapazität, Frequenz, Versorgungsspannung

Beispiel zur Verlustleistung berechnen

transiente Kurzschlüsse: kurzzeitig leiten beide gut

kleine Asymmetrie der Schaltzeitpunkte

selten umschalten

Energie sparen: CLK Gating Dynamic Voltage & Frequency Scaling

LV... Low Voltage

LVTTL ...

Versorgungsspannung:

Reduktion von Versorgungsspannung auf Hälfte

→ geringere Verlustleistung

→ für CMOS supi

→ oft nur bei Core (I/O bleibt)

Ungeeignete Versorgung:

HV: Chip heiß → Ausfall → Gate Oxid

LV: Chip langsam → hohe Ströme!

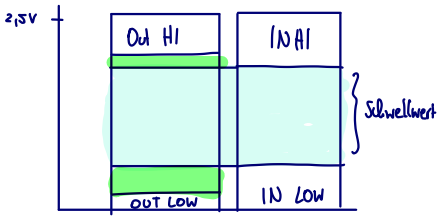
Spannungspegel & Strömungsabstand

Spannungspegel am Eingang: Eingangsspannung beachten!

Spannungspegel
- sicher als 0 (unterhalb Eingang)
- sicher als 1 (oberhalb Eingang)

!! Schwellwert! (Pegel = Grenzwert!)
↳ aufpassen, sonst out of spec!

Störspannungsabstand: Sicherheitsabstand zu zugehörigen Ausgangspegeln



Kompatibilität:

hängt von Höhe des Schwellwertes ab; (bzw "undefined" ab!)

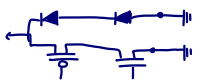
Wenn Schwellwert zu niedrig, nicht möglich

(muss direkt an Grenze zu Output High / Input High liegen!)

Klemmdioden:

wie Schalter (schließen sobald U an IO Pin > V_{DD} + 0.6V)
< GND − 0.6V

unerwünschte Überspannung abgeleitet (halten aber auch nur begrenzt!)



Spannungspegel am Ausgang:

Logikpegel ideal: 1 = V_{DD} 0 = GND

Logikpegel real: Verschiebung durch Schwellspannung / Spannungsabfälle wg I_{out}

Grenzwerte (Abhängig von Logikfamilie / Versorgungsspannung) min U_{out} = 1, max U_{out} = 0

V_{IH} (High level input voltage) → kl U die noch als 1 erkannt wird

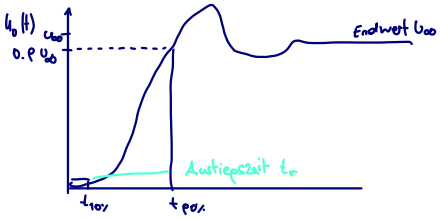
V_{IL} (Low level input voltage) gr Spannung, die als 0 wird

V_{OH} (High level output voltage) für 1 zumindest Spannung X

V_{OL} (Low level output voltage) für 0 höchstens gegebene Spannung X

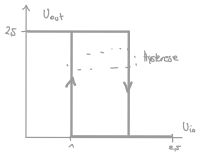
Flankensteilheit: In Datenblatt durch t_R (Input rise time)
t_F (Input fall time)

t_r Rise Time (Anstiegszeit)



Dauer des Signalanstiegs von 10% auf 90% des Endwertes

BSP Schmitt Trigger Eingang: Hysterese



100mV Hysterese um 1,2V herum

Toleranz der Hysterese im Datenblatt auffindbar

Hysterese, Schwellspannung etc benennen können

Treiber:

Ausgangsstrom

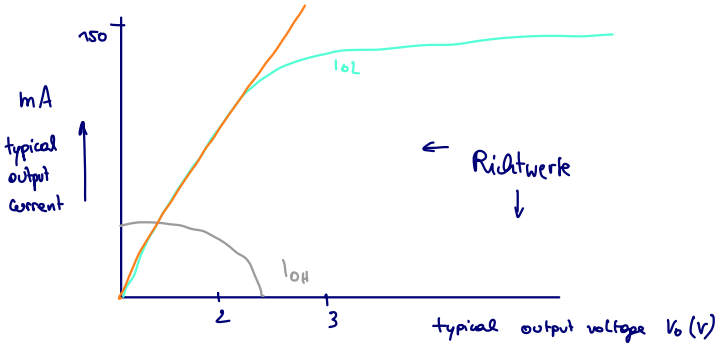
Wirkung Ausgangsströme I_{out}

1) Abweichung v I_{out} vom Idealwert wg Spannungsabfall an R_{out} (Ausgangswiderstand)

2) zu hoher Ausgangsstrom: GW für 0,1 nicht mehr eingehalten
Signalverzögerung steigt, Flanken flach
Ausgangstreiber überlastet

zulässige Ausgangsströme, Typische Ausgangsströme aus Datenblatt auslesen

Ausgangskennlinie



← Richtwerte ↓

typical output current: Abhängig von Exemplarstreuung, V_{DD}, T

Worst Case Angaben:

unter schlechtesten Bedingungen immer noch gültig

alle anderen Annahmen optimistisch → Mehrkosten (von Murphy)

↳ Typical Values (unbrauchbar), Messergebnisse v Prototyp (Funktionsnachweis)

Steigung der Kennlinie:

Ausgangswiderstand:

- Abweichung von Uout vom Idealwert
- verzögert Umladen / RC Verhalten
- An Steigung d. Ausgangskennlinie ablesen
  $r_a = \frac{\Delta U_a}{\Delta I_a}$ (Am Bild orange)
- von Logikpegel, Ausgangsstrom abhängig (konstant)
- Durch Treiberstärke eingestellt
- Idealwert Null ≠ erreichbar!

Eingangsströme:
Angeschlossener Eingang belastet vorhergehenden Ausgang

Fanout:

Wie viele Eingänge an Ausgangstreiber (meist max kapazitive Belastung!) sehr klein, stört nicht

↳ Überzählige Eingänge: Wenn benötigte # von Inputs nicht verfügbar, wird größeres Gatter gewählt, überzähliger Eingang mit anderem Paar zusammenschließen (belastet neue Quelle stärker) / logisch neutralisieren (über R zum Schutz vor Spannungsspitzen) an V_{DD}, GND

↳ Offene Eingänge: parasitäre Effekte bei bestimmten Spannungen
(in FPGA automat. korrekt behandelt) (bei CMOS typischerweise „0" undefiniert)

Spannungswerte sind labil / störempfindlich → subtile Fehler, unnötiger Leistungsverbrauch

jede Störung hat keinen Gegner → Pegel geht in irgendeine Richtung

PLL .. Phase locked Loop

Spezifikation des Zeitverhaltens

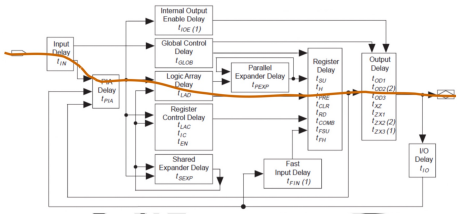
Timing - Angaben → kombinatorische Logik: Propagation Delay, Output Enable, Output Disable

→ sequentielle Logik: Flip Flop — Input → Setup Hold window; output → Clock to Data out — Taktflanke, Dauer, Delays, Setuptime

Steuersignale (CLK enable, ...) wie lange bis wirklich ⊖

→ Interconnect: versch Segmente, Parameter

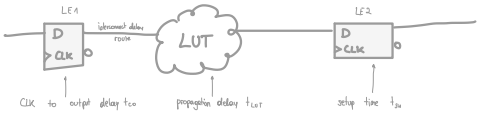
Timing Modell



Pfad, wo Daten durchwandern.

Summe aller relevanten Delays im Signalpfad

BSP zum Timing:



Achtung ob Pin oder Array → km/h

t_{SU} - setup time : min 10 ps

t_{CO} - clock to output: 176 ps (max

t_{LUT} - time for lookup table

Taktflanke kommt wann welches delay?

$$t_{CO} + t_{LUT} + t_{SU} + t_{route} = X \text{ ps}$$

minimale Taktperiode: ~ 1ns → man könnte 1GHz takten (etwas abenteuerlich)

Maximale Taktfrequenz: $f_{max} = \frac{1}{1ns} = 1\,GHz$

Speedgrade Zahl die angibt,

nicht direkt in Performance umrechenbar (nicht prop!)

Je größer Speedgrade desto langsamer/billiger der Chip

Interconnect timing: eher selten verwendet

I/O Element timing: normalerweise (setup, hold, ...) normalerweise von tools, nur reinschauen wenn was nicht hinhaut

Phase, Blindstrom bei DC nicht problematisch!

Chip treibt Ausgänge ⇒ Strom * Spannung (Rechenleistung muss nicht abgezogen werden)

/n ... Teile durch n

Optimierung des Zeitverhaltens

Einflussfaktoren: (Derating Factors)

- Versorgungsspannung (RC Ladekurve)
  - R durch Rout, Din von Fet an out, R der Leitung
  - C durch Kapazität Leitung, Gatter, Lasten, Programmierbare Verbindungen
  - Kapazitive Last ≫ Ohmsche Last!
  - Einfluss Lastenkapazität – Größenordnung 10 pF ... 1/4 ns!
  - An C_{IOTB}, C_{IOLR} ablesbar
- Temperatur, Treiberstärke

Eingänge, Routing, Exemplarstreuungen

Hohe Temperatur und niedrige Versorgung verlangsamen den Chip

Induktivität?

Optimierung:

- starke Treiber (kostet Chipfläche)
- Fan out verringern (1 out mit 10 zu 2 out mit 5)
- Sinnvolle Constraints (Timing Con → Optimierung auf wesentliches, nur unbedingt nötige Pins festlegen, Layout Vorgaben)

Slew Rate

maximale Steilheit des Spannungsanstiegs (am Out)

hohe SlewRate: Out liefert hohe Stromspitzen (verursachen Störung)

Vermindert Delay

Taktnetz: (Sollte man verwenden!)

Flanken überall gleichzeitig → verzweigtes Netz, hohes Fan Out

Spikes (Fischgerippe!)

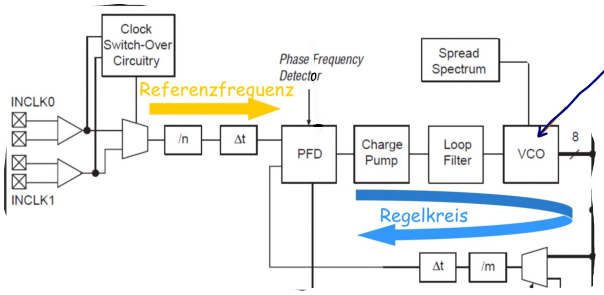
BSP Stratix

Clocking: hierarchisch 4 PLLs + 2 enhanced PLLs

Features: Signal Tap (Logik Analysator

Speednote Pinning

PPL (phase locked loop)



Frequenzmultiplizierer VCO

- DC höher Frq höher
- generiert Frequenz (spannungsgesteuerter Oszi)
- so geführt, dass er input entspricht

Phasenreferenz PFD: schaut wie 2 Eingänge zueinander sind gelockt → nur Phase einstellen

(aka Phasendetector)

Spread Spectrum: VCO wird verstellt damit Stromspitze nicht passiert

Synchrones Design

boolsche Logik: eindeutig zeitfreie Beschreibung, unabhängig von Implementierung (kombinatorische | sequentielle Logik)

↳ Realität: Delays (nicht vermeidbar, nicht deterministisch)

Einflussfaktoren auf Delay: Logikstufen (durch Technologie fixiert), Routing, Exemplarstreuung (worst case annehmen)

Delaykomponenten: Gate Delay, Interconnect Delay

Verhalten eines Chips ist nicht exakt vorhersagbar!

⇒ ∃ Grenzen für Design (km/h hat Grenzen, Laufzeit nie gleich → Skew)

Skew: Unterschied im Signaldelay $t_{skew} = |t_{dly1} - t_{dly2}|$

Folgen → (theoretisch): eingeschwungener Zustand ist Y θ

→ (praktisch): Glitches ⇒ Hazard

Glitch: Puls, den wir nicht wollen

Puls: Folge von entgegengesetzt gerichteten Flanken

Hazard: Möglichkeit für Glitch

Hazard:

"Möglichkeit für Entstehung von Glitches"

⚠ Gefährlich: Glitch wird in Zustand übernommen wo Flanke auswirkung hat

Glitch wird in Zustand übernommen wo transienter falscher Wert in Speicherelement

Handshake:

Regelschleife für Datenfluss zwischen SRC & SNK

→ robuster Betrieb

"Kunst" asynchrone Regelungen richtig koordinieren

Arten von Hazards

static 1: hätte gerne 1er, kriege Nullen...

static 0: hätte gerne 0er, kriege Einser

dynamic Glitch: Transitions, die wir nicht wollen

BSP Asynchroner Teiler

mehrfach verzweigter rückgekoppelter

Teiler würde mit Glitches hin werden!

Datenkonsistenz: Daten für gleichen Zeitabschnitt richtig korreliert

(Asynchroner Zähler ≠ optimal!)

⇒ Bei Boolscher Logik sind explizite Zeitbezüge nicht darstellbar!

beschreibt in|out zeitfrei

kontinuierlich gültige Eingänge

(skew macht ungültige Zwischenzustände!)

Grundproblem: Wir dürfen nur mit konsistenten eingeschwungenen Zuständen arbeiten

Lösung: synchrones Design

∃ asynchron, mit bounded delay & delay insensitive (erst in RasterVO ☺)

wie wir Synchrones Design bauen

Lösung im Zeitbereich finden

Eingänge: verarbeitet, nachdem alle Einschwingvorgänge sicher beendet

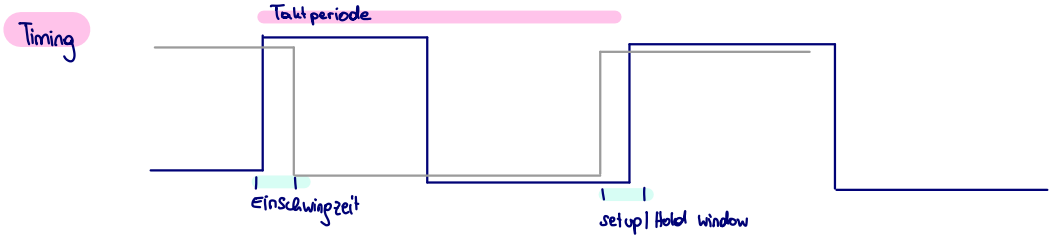
Taktsignal: Gibt Zeitpunkte an, wo Daten gültig

Quelle — Senke

(wird nach bestimmter Zeit verwendet, SRC sendet wieder Daten)

BSP SRC schickt um 7 Uhr Daten → SNK übernimmt um 7⁰⁰
SRC schickt um 8 Uhr nächste Daten
→ globaler Plan nötig

1) lokaler Plan muss von 🚶 umgesetzt werden
2) Uhren müssen gleich gehen (Taktflanke müssen gleichzeitig ankommen!)



nur noch mit eingeschwungenen Zuständen arbeiten:

Digital: nur 2 Logikpegel (Spannungsverläufe irrelevant)

Synchron: Zeitachse wird diskret (dazwischen egal)

Vorteile: Abstraktion ist hoch, Wahrheitstabelle mit Vorigem Zustand, Einschwingung egal, Timing nachträglich, Trennung von Takt und Daten (Timing einfach),

Effizienz (1 Steuersignal, single rail Datenkodierung, minimale Flanken, Takt = Zeit (einfach generierbar))

aber: Ermittlung von Taktfrequenz erfordert Kenntnis ALLER Laufzeiten

PROBLEME:

Timing Analyse nur mit idealem Taktnetz machbar!

Worst Case Annahme ist pessimistisch und hat keine Robustheit wenn überschritten (Graceful Degradation → kleiner Fehler, große Auswirkung)

Taktverteilung durch Verzweigung sehr groß; Minimierung von Delay, Skew ist aufwendig

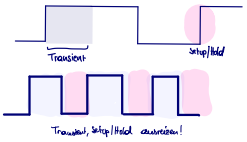
Verlustleistung durch hohen Leistungsverbrauch im Taktnetz mittels gleichzeitigem (stromspitzen) Schalten / ständigem (unnötig) Schalten

EMV (Elektromagnetische Verträglichkeit) durch Konzentration der Energie der Störaussendung auf eine Komponente

Erweiterbarkeit: neue Timinganalysen, Technologiewechsel, Spec-Anforderungen durch verknüpfte IP Cores

Grenzen

1) extreme Taktfrequenzen lassen keinen Spielraum! ideal: kein Einschwing, diskrete Zeite



Realität: Transiente Vorgänge & Setup/Hold möglichst agressiv ausreizen

Rasant steigende Taktraten → Situation noch schlechter!

2) Schnittstellen zur Umwelt

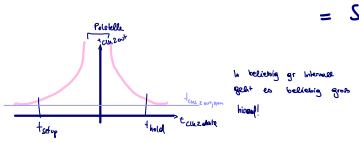
zwangsläufig Asynchron → $P_{violate} = \frac{T_0}{T_{clock}} > 0$ $= 0$ kein Setup/Hold Window/ Taktperiode ∞

3) CLK Domains mit Schnittstellen

Zwangsläufige Setup/Hold violation! (ähnlich dem unbestimmten Skew)
wegen beliebiger Phasenlage

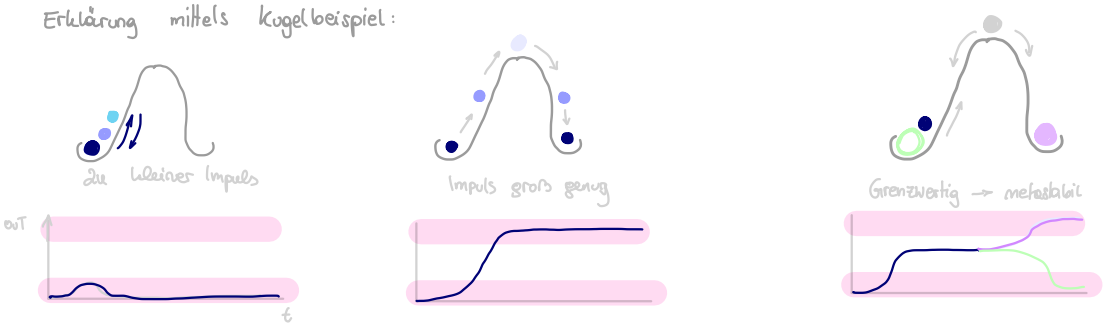
Metastabilität

Setup Hold Time: Decision Window: .. vor Beginn Änderung am Eingang, TG umschalten, keine Flanken am Eingang
= Setup time + hold time



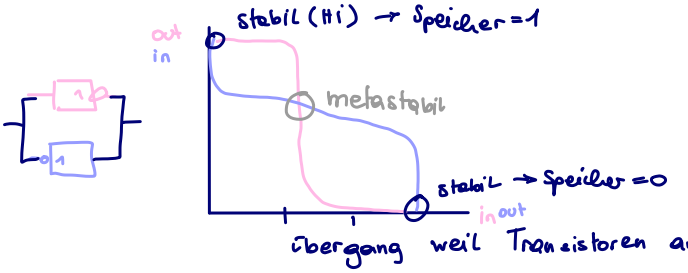
Ausgang bleibt für unbestimmte Zeit in metastabilem Zustand, dann geht er zufällig auf 1 oder 0. "Arbiträres Verhalten"

Erklärung mittels Kugelbeispiel:



Dynamik des Systems bringt mich in stabilen Punkt zurück!

Kennlinie von Inverter



Übergang weil Transistoren analoge Bauelemente

Beschreibt analoges Übertragungsverhalten
→ metastabiler Zustand kann überhapen werden

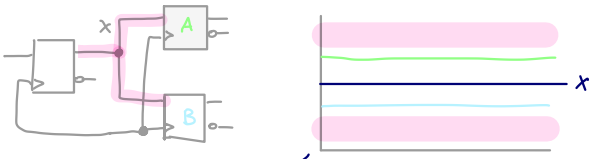
Folgen

Folgezustand? → Metastabilität führt zu Fortpflanzung (Fehler wird von Folge FF übernommen)

→ Metastabilität führt zu Inkonsistenz (Gleiches Signal, FFs interpretieren es anders!)

BSP Kennlinie

BSP Inkonsistente Wahrnehmung



Lösung

Resolutiontime

$$t_r = T_{clk} - t_{co} - t_{comb} - t_{su}$$

Zeit die für das Auflösen der Metastabilität maximal zur Verfügung steht.

Upset: Auflösen braucht länger als t_r.

MTBU Mean Time Between Upset

Formel erklären, anwenden

$$MTBU = \exp\left(\frac{t_r}{\tau_c}\right) \cdot \frac{1}{T_0 \, f_{clk} \, \lambda_{dat}}$$

$$MTBU = \exp\left(\frac{\text{Resolutiontime}}{\text{Bauteilparameter Zeitkonstante}}\right) \cdot \text{Taktperiode} \cdot \frac{1}{\text{Bauteilparameter}} \cdot \frac{1}{\text{Flankenrate}}$$

Bauteilparameter: Decision Window Breite

$$\frac{1}{MTBU} = R_{upset} = \exp\left(\frac{-t_r}{\tau_c}\right) \frac{T_0}{T_{clk}} \cdot \lambda_{dat}$$

Upset Rate

$\exp\left(\frac{-t_r}{\tau_c}\right)$: WSK das Metastabiler Zustand in t_r erhalten wieder

$\frac{T_0}{T_{clk}}$: WSK für Flanke in Decision Window

λ_{dat}: Flankenhäufigkeit

Besserer Ansatz für Metastabilität mittels MTBU

λ_{dat}, f_{clk} klein → fix durch Applikation

τ_c, T_0 klein → fix durch Speedgrade, Technologie

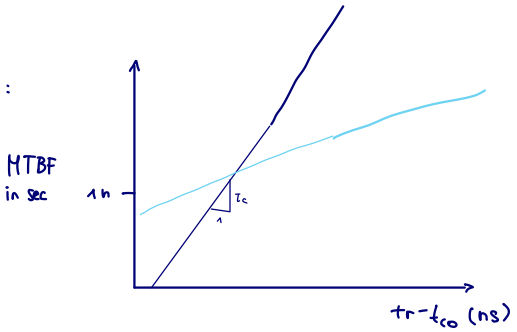
t_r sehr groß → Starker einfluss, Grundidee Synchronizer

konventionell: nicht vermeidbar, nur hinreichend unwahrscheinlich machbar!

adaptiert synchron: vermeidbar, absolutfreier Takt nötig

asynchron: vermeidbar, Handshaking (statt starrer Takt)

Technologie:



$f_{dat} = 1\,MHz$

$f_{clk} = 10\,MHz$

BSP Metastabilität

FF mit 100MHz Takt und 500kHz Eingangssignal.
t_{comb} = 3ns, t_{SU} = 0.5ns,
τ_c = 0.25ns, T_0 = 0.1ns
t_{CO} = 1.5ns

$$t_r = \frac{1}{f_{clk}} - t_{CO} - t_{comb} - t_{SU} = 5ns$$

$$\lambda_{dat} = 2 f_{dat} = 1MHz$$

$$MTBU = \exp\left(\frac{t_r}{\tau_c}\right) \cdot \frac{1}{T_0 \cdot f_{clk} \cdot \lambda_{dat}} =$$

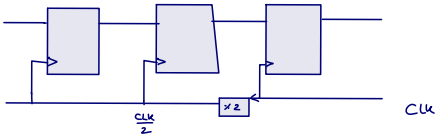
$$= e^{20} \cdot 10^{10} \cdot 10^{-8} \cdot 10^{-6} s = 4.8 \cdot 10^4 s$$

- ein Upset alle 13 Stunden.
- Bei 16 Kanälen also mehr als ein Upset stündlich.
- Bei 10000 Produkten/1Kanal im Schnitt alle 5s !

1) Resolution Time erhöhen

Andere Lösung: Synchronizer

a) 2. EingangsFF mit geteiltem Takt

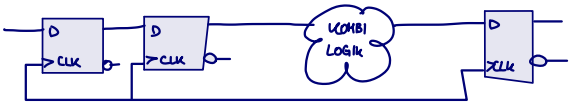


MTBU besser, Skew shit

b) n EingangsFF verknüpft

MTBU nicht so..., dafür problemlos
↳ Industrielösung

oder 2 Flop Synchronizer



Recontime between upsets messen:
↳ FlipFlop, 2 unabh. Oszis, FlipFlop
↳ Late transition Detection

Synchr Des bildet verbindung zw
Realdaten ↓ Zeitmodell

Wie kommt man auf $MTBU = \exp\left(\frac{t_r}{\tau_c}\right) \cdot \frac{1}{f_{clk} \cdot T_0 \cdot \lambda_{dat}}$?

1) Einfaches Modell: als Inverter modellieren (linear) → geschlossene Schleife (homogenen Fall) betrachten

2) Dynamik: Näherung mittels Dyn Modell 1. Ordnung RC Glied, Symmetrie, symmetrische Versorgung

3) Differentialgleichungen aufstellen

- Vorwärtspfad: $u_2 = -A \cdot u_1 - R \cdot C \cdot \frac{du_2}{dt}$
- Rückwärtspfad: $u_1 = -A \cdot u_2 - R \cdot C \cdot \frac{du_1}{dt}$

$u_R = R \cdot i_R$, $i_C = C \cdot \frac{du_C}{dt}$

- Laplace: $L\left\{\frac{du(t)}{dt}\right\} = s \cdot U(s) - u^0$

$U_2 = -A \cdot U_1 - \tau \cdot \left(s \cdot U_2 - u_2^0\right)$

$U_1 = -A \cdot U_2 - \tau \cdot \left(s \cdot U_1 - u_1^0\right)$

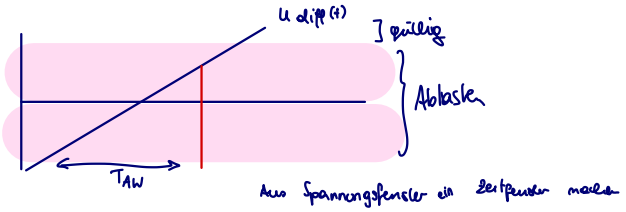
- Lösung im Zeitbereich:

$$u_2(t) = \frac{u_2^0 - u_1^0}{2} \cdot \exp\left(\frac{A-1}{\tau} \cdot t\right) + \frac{u_2^0 + u_1^0}{2} \cdot \exp\left(-\frac{A+1}{\tau} \cdot t\right)$$

4) Näherungsweise Lösen:

$$u_2(t) \approx \frac{u_2^0 - u_1^0}{2} \cdot \exp\left(\frac{A-1}{\tau} \cdot t\right)$$

Aperture Window:



Aus Spannungsfenster ein Zeitfenster machen

$$T_{AW} = \frac{2 u_{o,border}}{S}$$

Kalibrierung

$$T_{AW} = \frac{2 u_{o,border}}{S} = \frac{2 U_{o,border}}{S} \cdot \exp\left(-\frac{A-1}{\tau} \cdot t_{res}\right)$$

$$\Rightarrow T_{AW} = T_{wo} \cdot \exp\left(-\frac{A-1}{\tau} t_{res}\right)$$

Flanke:

$$\lambda_{upset} = \lambda_{dat} \cdot \frac{T_{AW}}{T_{Clk}} \rightarrow MTBU = \frac{1}{\lambda_{upset}} = \frac{1}{\lambda_{dat}} \cdot \frac{T_{Clk}}{T_{AW}}$$

T_0, τ_c ermitteln:

Rechnen:

Fehlerrate: $\lambda_{dat}\, f_{clk}\, T_0\, e^{-\frac{t_r}{\tau_c}}$

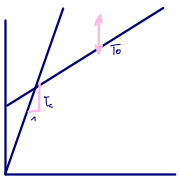
$\ln$ Fehlerrate $\ln\left(\lambda_{dat} \cdot f_{clk} \cdot T_0\right) - \frac{1}{\tau_c} \cdot t_r$

$\ln\left(\frac{1}{\text{Fehlerrate}}\right) = -\ln\left(\lambda_{dat} f_{clk} T_0\right) + \frac{1}{\tau_c} \cdot t_r$

$\ln\left(\lambda_{dat} f_{clk} T_0\right) \rightarrow d + k \cdot x$

Geradengleichung!

in Praxis nicht möglich, Punktwolke eher → Näherungsgerade



experimentell: Variieren von t_{res}, MTBU messen, log Graph,

Anstieg: τ_c, Offset → T_0

Trends: Es wird nicht wirklich besser...

Wenn wir jetzt alles nach synchronisiertem Timing ablaufen lassen, führt das zu Problemen (Ampel, Mikrowelle, Gaslicht)

Daher: Handshake → "Halt stop, ich bin noch nicht fertig!"

→ Metastability vermeiden?

asynchrone Inputs, multiple Clock domains, clock divider ;

supply drop, hohe Temperaturen ;

Defekte und Fehler

Hardwaretrojaner ?

Fehlerquellen Zeitdruck - Fremddesign - Komplexität - Strukturgrößen - kleinere Versorgungsspannungen (auf C, FlipFlop) - steigende Taktraten

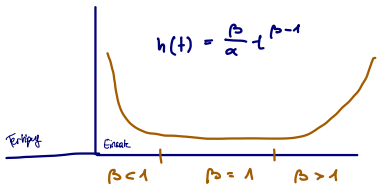
widrige Umgebungsbedingungen (Computer im Auto, -40° - 100°C) - Laien

Fehlertypen

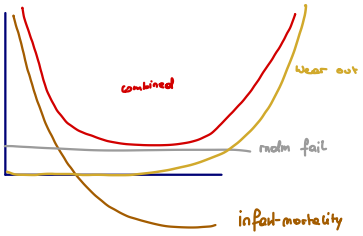
Design	Fertigung (Defekt)	Betrieb (Ausfall) THR-Systeme
Fehlerhafte Specs	Wafer	Komponentenausfall
Validation + Simulation	Factory Test	Fehlertoleranz

Fehlerquellen: Wafer, Prozess, Packaging, Transport, Bestückung, System integration

Badewannenkurve



Herleitung



Fehlerursachen im Einsatz

1) Electrical Stress (tritt kontinuierlich über Lebensdauer auf)

- Electro Statical Discharge

BSP: Ballon reiben, aufladen, anderen Elektrisieren

Was tun wir dagegen? Antistatische Böden, Erdungsbändchen, Klemmdioden (in Chip)

- Electrical Over Stress

Spannungsspitzen über Spannungsversorgung in Leitung, Design ∴, Blitzschlag

Was tun wir dagegen? Sicherungsdeckdosen gegen Überspannung

- Latch up

$\beta \rightarrow 100$ Kette verstärkung

bzw

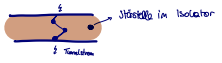
BSP Thyristor Überlappendes N P-P N-N P

Leiten sich gegenseitig weiter, hört nur auf wenn man es abschaltet

2) Intrinsisch (im Silicium)

Fehlerquellen: Gateoxid Wear out, Ionic contamination, Oberflächenladungen, Kristalldefekte, Piping

Gateoxid



- wear out: Tunnelströme (Leckt)
- break down: Isolator reißt

heisse Elektronen → stark beschleunigt, trifft wo auf → Lawinen effekt

3) Extrinsisch

- Elektromigration ← "Elektronenwind"

Stoßprozesse e^- gehen zum Pluspol → Gitter wandert (darstellbar mittels Wüstenlinie ← Hillock) → Unterbrechungen / Kurzschlüsse (geht nur bei DC!)
dh. Voiding → nur mikroskopisch!

$$MTTF = \frac{A}{J^2} \exp\left(\frac{E}{kT}\right)$$

J... Stromdichte, je größer die Stromdichte, desto mehr sinkt die Temperatur

T... Temperatur in Kelvin

k ... Boltzmankonstante

A ... Konstante

E_{act} = 0.5V – 1.5V

Fehlerrate: Häufigkeit von Auftreten von Fehlern

Reliability: WSK, das System noch t noch funktioniert

MTTF (Mean Time to failure) Erwartungswert von Start bis zu Ausfall

Black's law berechnen (Kelvin in Grad!!!)

BSP$_1$ "bsp für Kinderkrankheit"

kleinerer Querschnitt → Wann wirds hin?
(halber $\frac{1}{2}$)

halber Querschnitt → Doppelte Stromdichte
MTTF ist proportional mit $\frac{1}{J^2}$ → auf $\frac{1}{2^2} = 25\%$ reduziert

BSP$_2$ Elektromigration reduziert Al Verbindung in Chip

$$\frac{MTTF_{mil}}{MTTF_{com}} = \frac{\frac{A}{J^2}}{\frac{A}{J^2}} \cdot \frac{\exp\left(\frac{E}{k(125+273)}\right)}{\exp\left(\frac{E}{k(75+273)}\right)} = \frac{1}{26} \approx 4\%$$

mil ... military
com ... comercial

k.. Boltzmankonstante

Lebensdauer bei AC Chip höher, da DC Chip durch Relais (Dünen) altert.

- Microcracks (Dehnungskräfte)

Materialien (zB Kupfer) dehnen sich bei Erwärmung aus. (Kupfer wandert, Silicium reisst)

Kräfte bei Fertigung, Lagerung, Leistungsverbrauch

- Stressinduzierte Migration

Mechanische Spannung ⇒ Materialwanderung dh. wir brauchen keine Spannung, wird auch so hin.
Folge: Voids
Altern Chips auch wenn sie nicht in Betrieb sind? → Ja wg Temp. einfluss

- weiters noch:

Die Attach → Luft als Isolator

Bonding → Stifft zu nahe bzw Whisker, Kurzschluss...

Popcorn → Feuchtigkeit + Wärme = Dampf = Die bricht 

Korrision → Oxidation

Soft errors → Bitflips durch Strahlung ...

Ausfallursachen im Betrieb:

gate oxid: oxid dünn - Feld nahe - kleiner Isolator → kleine Tunnelströme - Isolator groß
Tunnelströme → Ausfall in Gateoxid ⟹ Lebensdauer von Gate Oxid abhängig

Electromigration

Electrostatic discharge

↳ Parameterabhängig

Konklusio: Nichts hält ewig, MTTF ist einstellbar (Kompromiss: Performance, MTTF, Preis!)

"in dubio pro Ausfall" – Ausfallfaktoren

Vermeiden (bei Herstellung): Unreine Wafer / Kristalle / Gate oxid, Prozessverunreinigung, kein ESD / Latch up Schutz, unreliables Package

Vermeiden (bei Verwendung): Hohe Temperatur (zyklen) / Spannungen / Strombelastungen, ESD durch Laien herbeiführen, Grenzwertbetrieb

Arrhenius Gleichung: $F_{i\,\text{Fehlerrate}} = C \exp\left(\frac{-E_{act}}{kT}\right) = \frac{1}{MTTF}$ → Temperaturerhöhung bewirkt exp() ansteigt!

beim Messen stellt man fest: +50° – 150° → 5000 fache!

Burn-in: Chip vor Verkauf heizen, bei Badewannenkurve so Fehlerrate geringer halten.

(nützen der Arrhenius Gleichung) $AF_T \times AF_V$

Alter = Temp * Voltage

normal / stress

wissen ?

$$AF_T = \exp\left(\frac{E_{act}}{k \cdot T_{normal}} - \frac{E_{act}}{k \cdot T_{stress}}\right)$$

$$AF_V = \exp\left[\gamma \left(U_{stress} - U_{normal}\right)\right]$$

BSP Rein hypothetisch: Chip rennt 4 Tage / 24h bei $T_j = 130$, 6V
Weibe Betriebsdauer bei $T_j = 70$, 5V

Faktoren berechnen → Multiplizieren

$AF_T = \exp\left(\frac{0.7}{8.6 \cdot 10^{-5} \cdot 70} - \frac{0.7}{8.6 \cdot 10^{-5} \cdot 130}\right) \approx 34.2$

$AF_V = \exp\left(3.2\,(6-5)\right) \approx 24.5$

$34{,}2 \times 24{,}5 \approx 838 \cdot 4\,\text{Tage} \approx 9$ Jahre (theoretisch → praktisch ist chip um 9 Jahre älter – zu viel!)

Fehlermanifestation

elektrisch: Kontaktprobleme durch Unterbrechung, hohen R, Wackelkontakt

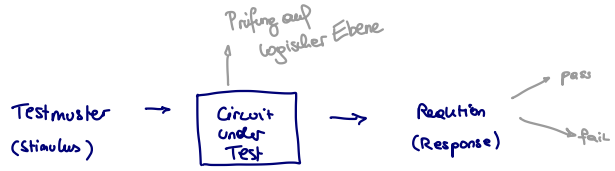
Material geht spazieren: Isolationsprobleme (Gate oxid)

dynamische Fehler durch reduzierte Geschwindigkeit

parametrische Fehler: zu viel Strom, Schwellwertverschiebung, ... Fehler in Kontinuum → "Jetzt reicht's!"

Speicherfehler durch Bit-Flips

Testen: Prinzip



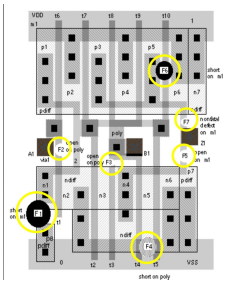
BSP Taschenrechner... Muss ich jetzt jede Rechnung der Welt testen?

Fault Maping

Defekte : physikalisch

Test : überprüft nur Logik (eine Ebene drüber)

BSP



F_1 Kurzschluss statt Durchlass

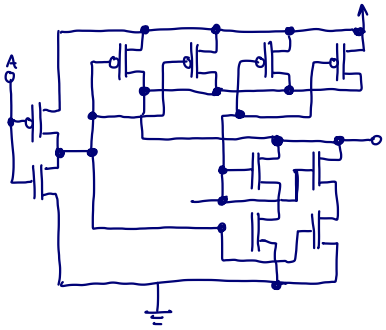
F_2, F_3 Staubkorn beim Bedrucken

F_4 Verbindung zu viel

F_5 Unterbrechung

F_6 Kurzschluss

F_7 : Electromigration (is nicht ganz



Knoten passt oder

stuck-at-1 (immer auf 1)

stuck-at-0 (immer auf 0 mit GND)

} dannn gibt es zu einem Zeitpunkt immer nur einen!

↳ Praxis : Single Stuck at (Relevanteste Fehler zu finden)

Praxis: Fehler finden mittels Anlegen von (bestimmten Stuck at Fehlern) an bestimmten Knoten!

Schrittweises "simulieren"

Stuck at open theoretisch möglich - simulation zu aufwendig

Der Test

Motivation: Rule of ten (Kosten bei Fehler verzehnfachen sich je Produktionsschritt → Chip € <<< System €

Testqualität: Wie gut (akurat) ist der Test den wir machen?

Defekt Level: Anteil an fehlerhaften Produkten im Verkauf – Delay Fault

Test coverage: 100% habe die Fehler, von denen ich ausging, alle gefunden (aber die anderen nicht!)

"Test coverage ist 100%." sagt nichts über Fehlersicherheit aus!

→ Abwesenheit von Fehlern lässt sich nicht beweisen!

Stuck-at-Fehlermodell

Fault Coverage	average defect level	average quality
50%	7%	93%
95%	1%	99%
99.9%	0.01%	99.99%

⇒ Einfacher Test erkennt viele Defekte.
Wenige restliche Defekte benötigen sehr großen Aufwand.

Erweitertes Fehlermodell durch gegenläufige Flanken mit Zeitmessung dazwischen

Testen auf verschiedenen Ebenen macht Sinn!

Systemausfall ist teuer! (Rule of 10)

↳ kosten auf Systemebene bei defectlevel 5.0% schon bei $5 \cdot 10^6$ €!

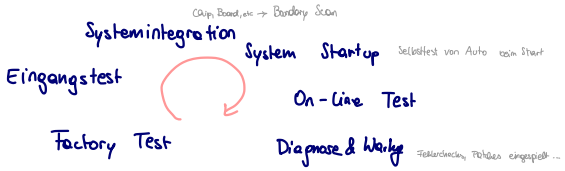
↳ BSP: 9h Stillstand ⇒ – 60 mio

Ziele des Testens

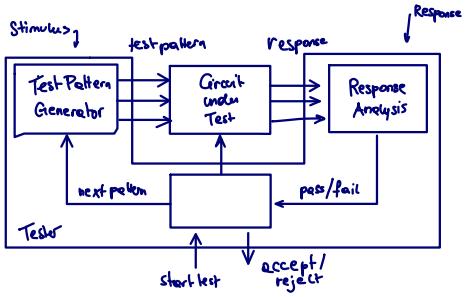
Defect Detection, Defect Location

(Reparatur, Prozessfeedback, Verantwortliche)

Testen in allen "Lebensphasen":



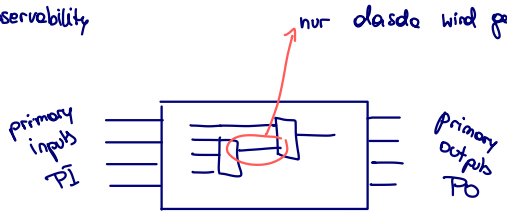
Testablauf



Tester macht prozess automatisiert.

Aber wie kommen wir auf die Muster?

"Testbarkeit" = Controllability + Observability



Wer testet den Tester?

Der Test selbst → best → Tester Defekt → saubere defekt

→ best nicht → tester Defekt

Steuerbarkeit über PIs (Controllability) → bestimmten Pegel zuordnen

Beobachtbarkeit über POs (Observability)

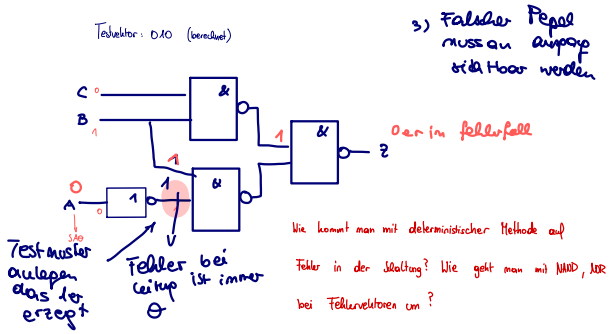
Test – Pattern – Generator

1) Brute force: Exhaustive Test – wir probieren alles durch
n Eingänge → 2^n Testvektoren

"leistbar" für kleine Blöcke, schließlich für viele große Blöcke
32 in → 43s Test aber 64 in → 580 Jahre

2) Deterministic Testing: Fehlerziele vorgeben, Vektoren zu Fehlern suchen
Wir nehmen Fehlermodell (2B stuck at ...)

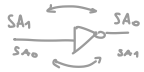
Activation: in fehlerfreiem Zustand einstellen
Justification: Bedingungen für Primary Inputs stellen
Propagation (nicht wegmaskieren!) Weiterleitung zu Primary Outputs sensitized path
Justification Bedingungen für Primary Inputs stellen



3) Falscher Pegel muss an Ausgang sichtbar werden

Wie kommt man mit deterministischer Methode auf Fehler in der Schaltung? Wie geht man mit NAND, XOR bei Fehlervektoren um?

Fehlersimulation Liste aller Fehler durch Fehlerdominanz/equivalenz Fehler verkleinern (Testvektor ermitteln/weg von Liste, ähnliche streichen)

Fehleräquivalenz: Fehler an verschiedenen Stellen haben die gleiche Auswirkung 

(Inverter)

Fehlerdominanz: Gleiche Fehlervektoren sind nicht unterscheidbar
↳ Test Vector Compaction: dort wo man bei Comul Coverage auf 100% kommt

Trotz super Testing haben wir immer noch Probleme: "hard to detect" faults, PI Bedingungen nicht leicht lösbar, Aufwand, redundante Logik
↳ testbar durch Zerlegung in Einzelteile
↳ Redundante Fehlstellen sind wegkürzbar und damit nicht erkennbar
⇒ Test Coverage ist dann Testvektorgeneration ist hard!

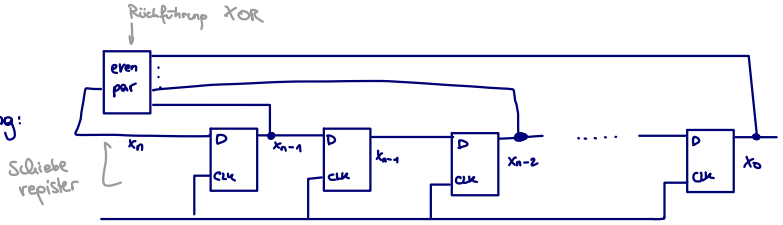
3) Nondeterministic Testig: Wir tippen "irgendwas" ein (bei 1% Coverage Ergebnis raus!)

mit Hardware (LFSR) on chip generierbar, erkennt "non target" Faults, pseudo deterministische Zufallsfolge
gewisse Eingangsmuster verboten → sonst mehrere Bustreiber

Linear Feedback Shift Register ·LFSR·

- periodische Mustersequenz (wie Schieberegister ohne XOR), Ins durch Polynom (achtung auf maximum length sequence),
- wird als Zufallsgenerator verwendet (Seed, reseed initialisiert vorgegebene Vektoren durch Sequenzvorkommen erzwingen)
- Polynom: Periode muss dem Maximum entsprechen $X^n = X^{n-1} \otimes X^{n-3} \otimes \dots \otimes X^0$

Implementierung:



Gegeben ist ein Polynom, zeichnen sie das LSFR, bestimmen sie die Ausgänge von 4 Zyklen (Wahrheitstabelle)

Response Analysis: Verhalten (Referenzdaten aus Simulation) → Abweichung = Fehler
→ Analyse von Abweichung: Diagnose

Massiver Speicherplatz für Referenz-Responses!

Sequentielle Logik testen – der Scan Path

kurze Wh. Sequentielle Logik
hat Speichernde Elemente (FF / Latch)
für Fehlererkennung: bestimmter Zustand von Testvektor

neue Dimension: Zustand (Testvektor anlegen, Zustand anfahren!) → mehr Testvektoren

eventuelle Probleme: Wenn Schaltung kein Reset hat weiß man nicht in welchen Zustand zum Testbeginn (Wir ham aber Reset :)
(Trotzdem wächst Testaufwand exponentiell!)

BSP Overflow testen ist ein Problem!
Carrybit von 32 bit Counter
1) Counter auf 0 (mit Reset :)
2) 2^{32} Taktzyklen warten, dann Overflow
→ kosten 10€ / Chip / Feature ... TEUER

BSP Controller im Auto alle einzeln testen → dann zsmhängen → alles Funktioniert ist nicht garantiert!

Wie umgehen wir das Geldproblem?

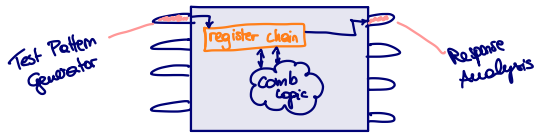
Functional-Test: Black Box → Funktioniert? es wird nicht hineingeschaut → nur für unkomplexes

Structural-Test: White Box → System zerlegt, einzeln getestet es wird mehr (aber Designfehler nicht erkennbar) Großes Ganzes betrachtet.

Fertigungstest: Fehler im Transistor → Gatterfehler

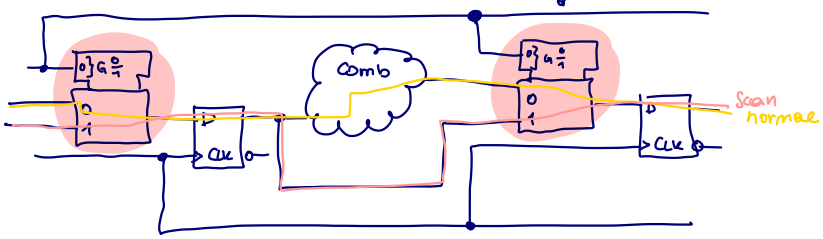
Methode: Scan Test

Im Chip sind quasi schon eingebaute Funktionen, die beim Testen helfen. Sie verdrahten die Register anders (zu einem großen Schieberegister) und werden dann mit Testmustern getestet. Die Schaltung ist nur statisch (nicht in Betrieb) testbar.
Testpunkte werden generiert (an fast jedes Gatter, kleine Blöcke werden exhaustive getestet)



Speicherelemente zu Schieberegistern!

Das gestaltet man mittels Overhead, beispielsweise wird ein zusätzliches MUX zum Testen davor ASIC Design Tool in der Fertigung eingebaut
Scan → Potential für Reversed engineering



Register (alle MUX)
- Data_out über D
- Scan_out über clk

Was muss man tun damit man Scantest machen kann?
Wie generiert man Testmuster?
MUX, Scanenable ...

Ablauf:

1) Testmodus aktivieren: Multiplexer schaltet FF Eingang an anderen FF Ausgang ⇒ Schieberegister (umgehen von kom. Logik)
2) Testvektor anlegen: Seriell in Scan chain geshiftet
3) anwenden: Verarbeitungsschritt an Pins durchführen
4) Response: Zustände von FFs seriell auslesen

Varianten:
Full Scan: MUX einbauen → alle FFs eingebunden → Chip langsamer
Partial Scan: MUX nur an unkritische Pfade → nicht alle FFs → Nachteil bei sequentieller Logik
Multiple Scan chain: Aufteilung auf disjunkte Scan Chains → rasches Lesen, viele Pins

Vorteile: sequentielles Testproblem auf kombinatorisches zurückführen, Scan chain automatisch generierbar, Wenige Pins für Registerketten zugriff
Setzen von internen Zuständen, gute Testbarkeit (weil jedes FF ist Testpunkt)

Nachteil: Shiftdauer bei langen Scan Chains, "tote Pins"

IDDQ-Test: wieviel Strom zieht er?
ungewöhnlich hoher Stromverbrauch → eventuell Fehler!
(Abh. von Testvektor!)
POR verdächtiger Chip

Overhead: eingebautes Testdesign

Speicher & Marchtests

dh. wie teste ich, ob Speicher funktionieren?

Fehlermodelle:

- Stuck-at Fault: Bitzelle sitzt auf 0/1
- Transition Fault: nur in eine Richtung schaltbar
- Coupling Fault: Aktivität einer Zelle beeinflusst nächste (zB: Bit fällt um → Nachbarbit kippt)
  - ↳ Neighbourhood: Schreiben in Nachbarzelle

Tests von Speicherblöcken: mittels Scantest zu aufwendig,
mittels Funktionalem Test möglich
mittels March Test empfohlen

March Test:

- findet Coupling Faults
- O(n) nur Linear
- in HW einfach implementierbar (auch nur programmierbar!)

March C Test Algorithmus:

1) alle Speicherzellen = 0
2) aufsteigend: Zellen durchgehen, überall wo 0 steht 1 setzen. (wenn 1 steht → Fehler!)
3) aufsteigend: Zellen durchgehen, überall wo 1 steht 0 setzen. (wenn 1 steht → Fehler!)
4) 0 in allen Zellen Prüfen.
5) absteigend: Zellen durchgehen, überall wo 0 steht 1 setzen. (wenn 1 steht → Fehler!)
6) absteigend: Zellen durchgehen, überall wo 1 steht 0 setzen. (wenn 1 steht → Fehler!)

gehe zu 4.

Umsetzung in FPGAs:

nach Fertigung: Grundfunktion testen (prog. bei OTP nicht möglich)

nach Programmierung: Rücklesen, Simulation
(Test im Programmiergerät, Test im Design)

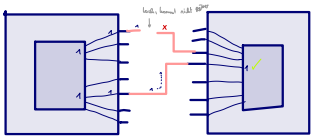
Programmierbarer & nicht programmierbarer Baustein sind gleich gut testbar.

Boundary Scan (System integration → hat nichts mit Scantest gemein!)

Problem: Zugänglichkeit bei unbestückten Leiterplatten

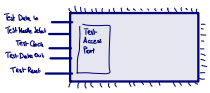
Wir testen: bestückte Leiterplatte mittels Flying Probes, Nadelbettadapter, Funktionstest und Boundary Scan

Wie funktioniert's?



DR-Cells zur Umsetzung (normal, Schieberegister, ausgeben/einlesen von Daten)
oft spezielle Zellen in Libraries verfügbar!

JTAG Test Access Port: aka TAP Controller ist ein Testcontroller.



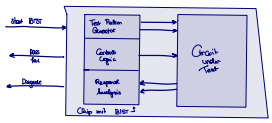
MISR: Multiple Shift Index Register

Built in Self Test (BIST)

Design for Test: Testbarkeit = Designanforderung, Integration von Bist Logik ("unnötiges" FF, ... → Leitung, an sich selbst geschlossenes Flipflop ist schlecht für Test!, Schaltung, die gewisse Taktfrequenz braucht auch!)

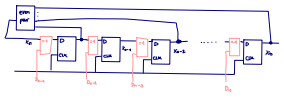
Regeln für Design-for-Test:

- ~~Clock gating~~
- ~~Self resetting~~
- ~~Redundante Logik~~
- Partitionierung (mit Bus) ✓
- Trennen von Takt + Signalen ✓
- Initialisierung für sequentielle Logik ✓
- Testhilfen in Zählerketten ✓



Implementierung:

Test Pattern Generator: rückgekoppeltes Schieberegister, Pseudo Random Generator

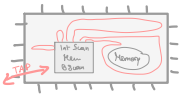


Response Analyzer: Multiple Input Shift Register (MISR) Erweitertes LFSR

response data compaction: Sequenz von Responses ("Compaction") mit MISR auf Signatur → Aliasing (auch einzelne Fehler zu richtiger Signatur!) (WSLK bei $p \sim 2^{-R}$)

Collar Logik isoliert Speicher während Test!

BSP: 5 Pins für boundary Scan, internal scan, Speichertest, Konfiguration



Vorteile: kritisches Interface auf Chip, Interface einfach nach außen,
ad speed testen, gute Testpunktzugänglichkeit, Kosten klein,
Overhead bei 10%, rasche Diagnose, keine Designmacht,
für Start up / on-Line Test verwendbar

on Line Test: Defekte während Betrieb (Raumfahrt, Weiche)
↳ Innerer Zustand bei Test gleich!
Zeitverhalten gleich!
Auto 300.000 km → funktioniert Airbag?

Der Logikanalysator

elektronisches Messgerät, das den Zeitverlauf von digitalen Signalen aufzeichnet/anzeigt

Multimeter

1 Kanal

statische Größen messen (Relative Kennwerte)

gibt Effektivwert an

für rasche Änderungen

Oszilloskop

2-4 Kanäle

dynamisch (Spannungsverlauf über Zeit)

Frequenzbereich bis mehrere GHz

(gut für Interleaving)

kontinuierliche Amplitude, kontinuierliche Zeit

Timing Analyzer

bis 512 Kanäle

um Prozessor zu debuggen (ganze Busse anschauen)

nur Logikpegel messbar

viele Messfunktionen

Frequenz: bis einzelne GHz

State Analyzer

bis 512 Kanäle

nur synchrone Zeitverläufe sichtbar

nur Logikpegel messbar

viele Messfunktionen

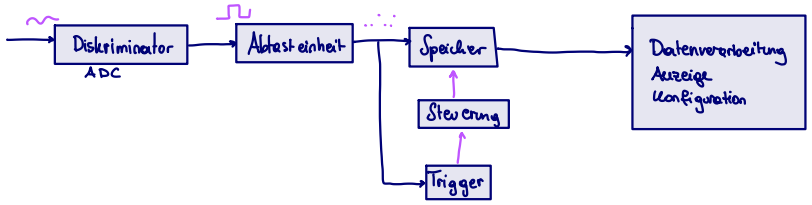
Frequenz: bis GHz

↳ fast so hoch wie bei Timing Analyzer

diskrete Amplitude, kontinuierliche Zeit

diskrete Amplitude, diskrete Zeit

Aufbau (Funktionsblöcke)



Diskriminator

ADC – Analog Digital Converter (bei Speicheroszi liefert ADC feinstufige Info)

Schwellwert: user defined / nach Logik standards

fein einstellbar (mit Logik ist es grenzenlos zoombar)

→ Interpretation kann anders als HW untersuchung sein!

Schwellwert – Problem: Schwellwert LA ≠ Schwelkwert HW

→ falsche Pegel

(Aber nur wenn Signalpegel in falschem Bereich)

→ falsche Flanken

(Flankenverschiebung nur bei flachen Flanken relevant)

Abtasteinheit

zeitlich diskretisieren (Achtung: Unterabtastung → verlorene Daten!)

Takt als Abtastungsbasis
- Timing Analyzer: hochfrequenter Takt → alle Details wie Speicheroszi
- State Analyzer: Target liefert Takt → Zustandsabtastung ist Fehleranfällig (bei Phasenverschiebung!)

Sampling – Problem

- Unterabtastung (Timing Mode)
  Pulse verloren, Flanken verschoben
  QD Shannontheorem
- Phasenverschiebung (State Mode)
  verschiedene Interpretationen von Setup / Hold etc...

Speicher

Samples ablegen (in Echtzeit, Bearbeitung & Darstellung offline)

ein großer Speicher ist nicht die optimale Lösung

Tiefe: 1 M Sample (bei 1GHz nach 1ms voll → nicht alle States durchsuchbar → Trigger)

Aufzeichnungsmodi (für Timing Mode)

Trade off: Je schneller getastet wird, desto weniger Speicher wird gebraucht (weil alle eng beisammen)
je genauer die Speicherticks, desto geringer die Abtastrate

Transitional Sampling: Punkte merken wo sich Zeitpunkt ändert (Flanke + Zeitpunkt)

Trigger

≠ Sampling, was nur einen Punkt verwendet!

bestimmt, wann Sampling beginnt / aufhört (LA kann h auf Trigger warten)

Triggerbedingungen: Flanke, Zustand, Pulsdauer, ...

(if then else nicht nützlich)

Markierung: für Datenauffindung für später!

Pretrigger: Zeichnet auf, was vor Trigger passiert ist (nicht überschreibbar)

Posttrigger: Was war nachher (Trigger → noch halber Speicher dann stop)

} Speicher als FiFo

trigger levels möglich

store, stop, AND, OR möglich

(im State mode komplexer)

LA zeichnet ständig auf, wenn "Trigger" aktiviert dann Info speichern

Unterschied von Trigger / Sampler

Darstellung durch Timing Diagramm, State Diagramm, State Listing

Cursor

horizontale Zeitbereichmarkierung (direkte Wertablesung)

automatisch positioniert

Differenzmessung

Probing:

für mechanische & elektrische Verbindungen, Pegelanpassung, Impedanzanpassung

Probleme: Zeitverzüge bei langen Kabeln

Lösung: Flying leads >> Samtec probes (Stecker)

Teilweise angelötet (einweg → Sockel in Design vorhanden

Kenndaten eines LA

logic analyzer channels

pattern generator channels → Ausgänge zum Signal/Stimuli erzeugen

high speed timing zoom

maximum timing sample rate (Deppensicher) kann beliebig gezoomt werden

maximum state clock rate

maximum state data rate

maximum memory depth

supported signal types: single ended (normale Logikpegel erzeugbar)

Grundlagen

Der Einsatz digitaler Elektronik in sogenannten „Embedded Systems“ prägt unsere Gesellschaft.
Digitale Elektronik bietet gegenüber der analogen Vorteile wie einfache Speicherbarkeit, bessere Störsicherheit und höheren Abstraktionsgrad.
Die Kunst des digitalen Design besteht darin, die Grenzen digitaler Logik zu verstehen, die imma-nenten Idealisierungen zu bedenken und durch Kenntnis des gesamten Entwicklungsprozesses die Gesamtkosten eines Design zu minimieren.
Kombinatorische Logik funktioniert unabhängig von der Vorgeschichte.
Sequentielle Logik hat im Gegensatz dazu ein Gedächtnis (Rückkopplung oder Speicher) und daher einen inneren Zustand.
Kombinatorische Grundfunktionen sind Inverter, AND, OR, NAND, NOR, XOR und XNOR.
Aus diesen Funktionen lassen sich komplexere Funktionen mit mehr Eingängen wie z.B. ein Multiplexer oder ein Threshold Gate realisieren.
Sequentielle Grundfunktionen sind das Latch und das Flip-Flop. Es gibt jeweils mehrere Varianten.
Beim D-Latch steuert ein „Enable“-Signal, ob die aktuellen Daten durchgelassen oder die alten Daten gehalten werden.
Das D-Flip-Flop übernimmt mit der aktiven Takt-flanke neue Daten und hält diese bis zur nächsten.
Zusatzfunktionen beim Flip-Flop sind „Preset“ und „Clear“ sowie „Clock Enable“.
Eine State-Machine durchläuft synchron eine Sequenz von Zuständen.
Die Abfolge der Zustände wird durch die Eingangssignale gesteuert.
Bei der Moore-State Machine hängen die Ausgänge nur vom Zustand ab, bei der Mealy-State Machine werden sie zusätzlich auch (asynchron) von den Eingängen beeinflusst.
Die Darstellung einer State-Machine erfolgt im Zustandsgraph und im Zustandsdiagramm.
Die Boolsche Algebra umfasst mehrere Theoreme, wie z.B. jene von De Morgan und Shannon.
Jede Boolsche Funktion ist durch ihre Wahrheitstabelle eindeutig beschrieben.
Eine grafische Darstellung der Wahrheitstabelle ist im Karnaugh-Veitch-Diagramm möglich.
Aus Wahrheitstabelle bzw. KV-Diagramm lassen sich Minterme bzw. Maxterme ableiten und durch deren Verknüpfung schließlich die disjunktive bzw. die konjunktive Normalform.

Asic Fertigung

Die technologische Entwicklung im Bereich der ASICs ist extrem dynamisch.
Der bekannteste Indikator dafür ist das Moore‘sche Gesetz:
Die Komplexität (Anzahl von Transistoren in einem Design) verdoppelt sich alle 1,5 Jahre.
Siliziumdioxid ist ein Isolator, polykristallines Silizium ein Leiter, und mittels Dotierung lassen sich mit Silizium auch Schalter (Transistoren) realisieren.
Damit ist Silizium der ideale Ausgangsstoff für digitale Logik.
Ausgehend vom Rohstoff Quarz wird über komplexe Fertigungsschritte ein Chip gefertigt: Schmelzvorgänge, Dotierung, Oxidation, Metallisierung
Mittels Photolithographie werden dabei die gewünschten Strukturen hergestellt. Diese sind über Masken definiert.
Der charakteristische Parameter einer Technologie ist die Feature-Size l.
Die einzelnen Transistor-Strukturen werden über den metallischen Interconnect verbunden.
Der fertige Die wird getestet und in ein Gehäuse gepackt.
Aktuelle Trends bei der ASIC-Fertigung sind Silicon on Insulator, Multichip-Module, Thorugh-Silicon Via und System on a chip.
Eine Reihe technologischer Grenzen scheint das weitere Wachstum der Entwicklung zu begrenzen. Bisher wurden solche Grenzen jedoch stets überwunden – nicht zuletzt aufgrund der immensen Forschungsaufwände in diesem Bereich.

Aufbau Logischer Gatter

Grundelement der digitalen Logik ist der Enhance-ment-FET, wobei bei CMOS der n-Kanal-Typ und der p-Kanal komplementär zum Einsatz kommen.
Die wichtigsten Parameter des FET sind Schwell-spannung und Ausgangsstrom (bzw. Formfaktor)
Im Idealfall verhält sich ein FET wie ein Schalter: der n-Kanal-FET schließt bei 1 am Steuereingang, der p-Kanal-FET bei 0.
Die Idealisierung als Schalter funktioniert nur unter geeigneten Randbedingungen.
Bei genauerer Betrachtung (im Zeit oder Amplitudenbereich) verhält sich der FET wie ein analoges Bauelement.
Der Inverter ist die Grundstruktur aller Logik-funktionen. Er lässt sich technologisch einfach implementieren.
Ersetzt man die beiden Einzeltransistoren durch einen sog. n-Stack bzw. p-Stack, so lassen sich bei geeigneter Abstimmung allgemeine logische Funktionen wie AOI und OAI implementieren, sowie als Sonderfälle auch NAND und NOR.
Nicht invertierende Funktionen können in CMOS nicht einstufig realisiert werden.
Weitere typische Elemente sind Transmission Gate, Multiplexer und getakteter Inverter.
Mittels getakteter Inverter kann ein Latch realisiert werden, durch Master/Slave Kombination zweier Latches ein Flip-Flop.
Aufgrund der Einschwingzeit der Datenpfade (und insbesondere der Speicherschleife) darf innerhalb des „Decision-Window“ (Summe aus Setup- und Hold-Time) keine Änderung der Daten erfolgen, sonst kann Metastabilität auftreten.
Die Realisierung eines ganzen Speichers mittels Flip-Flop oder Latch ist sehr ungünstig, effizienter sind hier SRAM oder DRAM.
Neben dem komplementären Ausgang gibt es den Tri-State Ausgang sowie den Open Drain Ausgang.
Die CMOS-Technologie ist derzeit am weitesten verbreitet, in besonderen Anwendungen findet man jedoch auch bipolare Logikfamilien wie TTL oder ECL, oder auch Bi-CMOS (für hohe Treiber-leistung).

Design Flow eines Asci

Der Design-Flow eines ASIC bzw. FPGA umfasst die folgenden Schritte: Spezifikation, Design-Entry, Compliation, Technology-Mapping,Partitioning, Floorplanning, Placement & Routing (PPR),Fertigung /Download

Der Design-Prozess ist höchst komplex und daher fehleranfällig. An vielen Stellen ist daher eine Verifikation nötig, und im Fehlerfall müssen die einzelne Design-Schritte wiederholt werden. Das Design ist also ein iterativer Prozess.

Die Verifikation umfasst folgende Schritte: Validation (Prüfen der Spezifikation),Simulation (Prüfen der virtuellen Implementierung) und Test (Prüfen des physikalischen Designs)

Das Y-Diagramm erlaubt eine Veranschaulichung des Design-Prozesses. Es umfasst die 3 Achsen Verhalten, Struktur und Geometrie. Durch konzentrische Kreise werden die Abstraktions-ebenen dargestellt.

Der Design-Prozess beginnt auf einer hohen Abstraktionsebene (typ. RTL) auf der Verhaltens- (und/oder Struktur-) achse. Mittels Tool-Support gelangt man über Struktur und die Geometrieachse zu niedrigeren Abstraktionsebenen und schließlich ins Zentrum des Diagramms.

In HDLs lässt sich vieles einfach beschreiben, eine Abbildung auf HW erweist sich jedoch bei der Synthese oft als zu aufwendig oder unmöglich. Durch einen Optimierungsprozeß wird eine Kosten- funktion minimiert bzw. eine Nutzenfunktion maximiert, jeweils unter Einhaltung gegebener Randbedingungen. Partitioning, Placement und Routing sind solche Optimierungsprozesse. Vielfach werden hier aufgrund der Komplexität heuristische Methoden den geschlossenen Lösungen vorgezogen. Für Partitioning und insbesondere Placement und Routing ist eine Abschätzung der Signallaufzeiten essenziell. Diese erweist sich jedoch aufgrund der Dominanz des Interconnect-Delay für neuere Technologien als zunehmend schwieriger. Simulation sollte auf möglichst vielen Ebenen durchgeführt werden, um Fehler rasch und eindeutig identifizieren zu können.

Die Signal-Resolution Table gibt Aufschluss darüber, welcher Ausgangspegel aus dem Zusammenwirken mehrerer Eingangspegel an einem bestimmten Logikelement entsteht. Bei der ereignisgesteuerten Simulation werden die Ereignisse nach einer Liste chronologisch abgearbeitet, neue Folge-Ereignisse werden in der Liste ergänzt. Gleichzeitigkeit wird durch die „Delta-Time“ berücksichtigt.

Die statische Timinganalyse sucht systematisch das Design nach den langsamsten Datenpfaden ab. Das Timing ist in Libraries definiert. Variationen in der Temperatur oder der Versorgungsspannung werden durch Derating-Factors berücksichtigt. Die formale Verifikation erlaubt eine lückenlose Überprüfung des Designs nach bestimmten Kriterien. Voraussetzung ist aber das Vorliegen eines entsprechenden

Speichertechnologien

Digitale Speicher sind eine strukturierte Anord-nung von Bits mit zwei klar unterscheidbaren Zuständen.
Neben Aufbau und physikalischem Speicherprinzip sind Volatility, Beschreibbarkeit und Random Access charakteristische Merkmale.
Neben ihrer eigentlichen Funktion als Daten-speicher sind Speicher auch für vielfältige logische Funktionen verwendbar.
Zur Vereinfachung der Decodierlogik sind Spei-cher als zweidimensionales Array strukturi
Bei den ROMs unterscheidet man zwischen Mask-ROM, OTP, UV-EPROM und EEPROM.
Das SRAM beruht wie das Latch auf einer Speicherschleife aus rückgekoppelten Invertern, ist aber platzsparender.
Das DRAM verwendet einen Kondensator als Speicherelement, eine Speicherzelle ist daher deutlich kleiner als beim SRAM. Es ist allerdings ein periodischer Refresh erforderlich.
Das MRAM ist ein schneller nicht-flüchtiger Speicher. Es beruht auf magnetischer Polarisation.
Multiport-Speicher und FIFO erlauben die Kopplung unsynchronisierter Systeme.
Durch Hinzufügen von Prüfbits können Bitfehler erkannt und eventuell auch korrigiert werden. Übliche EDC/ECC-Verfahren sind Parity, Hamming Code und CRC.

Zieltechnologien

Der Full-Custom-ASIC bietet maximale Flexibili-tät für Optimierungen, wird jedoch aus Aufwands-gründen nur für Spezialanwendungen eingesetzt.
Der Standardzellen IC (CBIC) geht von einer Library aus vorgefertigten und getesteten Logik-zellen aus. Dies vereinfacht das Design, bei der Fertigung sind aber alle Layer kundenspezifisch.
Beim Gate Array sind Basisfunktionen vorgegeben, durch kundenspezifische Metallisierungslayer kann über Makros jede Funktion implementiert werden. Man unterscheidet zwischen channelled, chanelless und structured GAs.
Time to market ist ein entscheidendes Erfolgs-kriterium für ein Design. Das ist ein entscheiden-des Argument für programmierbare Logik.
Bei den programmierbaren Logikbausteinen (Pro-grammable Logic Devices, PLDs) unterscheidet man zwischen ROM, PAL/PLA und FPGA/CPLD.
Bei den FPGAs sind die Funktion der Logikzellen, Funktion der I/O-Blöcke und Verbindungen programmierbar.
Programmiert werden schaltbare Verbindungen, (Antifuse oder TG über EPROM bzw. SRAM).
Programmierbare Logikzellen lassen sich auf der Basis von Multiplexern, Look-up Tables (LUT) oder Wired AND (PAL-Struktur) realisieren.
Bei den I/O-Blöcken sind üblicherweise Ausgangspolarität, Verzögerungen, Latches im Datenpfad, Treiberstärke/Anstiegszeit, Pull-ups, Tri-State etc. programmieren.
Beim Interconnect bedeutet jede programmierbare Verbindung eine Vezögerung. Es muß daher ein Tradeoff zwischen Flexibilität und Geschwindigkeit gefunden werden.
Der Interconnect ist bei den CPLDs völlig regulär und sein Timing daher einfach vorhersagbar.
Die wesentlich leistungsfähigeren FPGAs haben auch einen komplexeren Interconect, der dadurch auch im PPR sowie im Zeitverhalten schwerer beherrschbar ist.
Die Konfiguration ist bei EEPROM- und Antifuse-basierten CPLDs ist nicht-flüchtig; bei den SRAM-basierten FPGAs muss sie bei jedem power-on nachgeladen werden.
Hierzu gibt es vielfältige Möglichkeiten.

Datenblattangaben

Die Temperatur hat einen wesentlichen Einfluß auf die Funktion eines Chips
Entscheidend ist in jedem Fall die „Junction Temperature“. Sie ergibt sich aus Umgebungs-temperatur, Verlustleistung und thermischem Widerstand.
Für die Umgebungstemperatur sind verschiedene Bereiche spezifiziert: commercial, industrial und military.
Die Verlustleistung umfasst drei Komponenten: statische Ströme, Ladevorgänge und transiente Kurzschlusse.
Dominant sind im aktiven Betrieb die dynamischen Ladeströme. Sie sind proportional zur Takt-frequenz und zum Quadrat der Spannung.
Um ein Hin- und Herschalten der Eingänge zu vermeiden, müssen entweder die Signalflanken entsprechend steil sein, oder man verwendet Schmitt-Trigger-Eingänge. Diese weisen eine Hysterese auf.
Um eine Störspannungsabstand zu gewährleisten sehen die Spezifikationen für die Ausgänge engere Grenzen vor als für die Eingänge.
Der Ausgangsstrom ist stets begrenzt (Ausgangs-widerstand). An einen Ausgang darf daher nur eine begrenzte Anzahl von Eingängen ange-schlossen werden (Fan-Out).
Ausserdem ergibt der Ausgangswiderstand im Zusammenwirken mit den Kapazitäten der Leitungen und der Eingänge ein RC-Glied, das die Umschaltvorgänge verzögert.
Typische Timing-Angaben sind Setup- und Hold-Time, Durchlaufzeit, clock-to-output-delay sowie Routing delay.
Hohe Temperatur, niedrige Versorgungsspannung und hohes Fan-Out machen einen Chip langsamer.
Das Routing ist ein entscheidender Einflussfaktor für das Timing
Das Taktnetz spielt eine besondere Rolle im Interconnect. Zur Erreichung eines geringen Delay und eines geringen Skew trotz des hohen Fan-Out und des weit verzeigten Netzes werden spezielle starke Treiber und spezielle Topologien eingesetzt.
Taktnetze sind daher stets mit gesonderten Pins verbunden und sind im Design gesondert zu behandeln.

Synchrones Design

In realen Schaltungen sind Signallaufzeiten
unvermeidlich.
Die Differenzen dieser Laufzeiten (Skew)
variieren in nicht vorhersagbarer Weise.
Boole'sche Logik geht von kontinuierlich gültigen
und konsistenten Eingängen aus.
Zeitliche Zusammenhänge sind in Boole'scher
Logik nicht beschreibbar.
Genau diesen Mangel muss eine Design-Methode
kompensieren.

Synchrones Design verwendet Zeitbedingungen, um die die Datenkonsistenz zu beurteilen.
Diese Methode ist effizient und bewährt.
Der Schluss von Zeitbedingung auf Konsistenz ist aber indirekt und willkürlich.
Synchrones Design ist daher keine natürliche Lösung des Grundproblems.
Dieser Mangel wird mit steigenden Taktfrequenzen immer deutlicher spürbar.
Bei Timing Violations und an den Grenzen von Clock Domains kann es zu Metastabilität kommen.
Eine Verletzung der Setup-/Hold time kann zu beliebig langem Verweilen des Ausgangs eines Flip-Flop in einem undefinierten Pegel führen.
Übernimmt ein nachfolgendes Flip-Flop diesen un-definierten Zustand („Upset"), kommt es zu Fortpflanzung und zu inkonsistenter Interpretation.
Die Wahrscheinlichkeit dafür kann berechnet wer-den. Sie steigt exponentiell mit sinkenden Timing-Reserven, hängt aber auch von der Technologie ab.

Defekte und Fehler

Fehlerquellen sind über den gesamten Lebens-zyklus eines Chips verteilt: vom Design über die Fertigung und Inbetriebnahme bis zur Applikation.
Designfehler werden durch Simulation entdeckt.
Ursachen von Defekten bei der Fertigung liegen in Wafer-Material, Lithographie, Entwicklung & Ätzen (Verunreinigungen), Bonding, Packaging,...
Die Badewannenkurve beschreibt die Verteilung der Ausfälle über die Betriebszeit: Nach einer hohen Ausfallsrate zu Beginn (infant mortality) folgt eine Periode mit konstanter, niedriger Ausfallrate (useful life), danach steigt die Ausfallrate stark an (wear-out).
Häufigste Ursachen für Defekte im Betrieb sind Gate-Oxide-Breakdown, Electromigration und Electrical Overstress.
Beim Gate-Oxide-Breakdown bilden Störstellen in der extrem dünnen Isolationsschichte des Gate-Oxid einen leitenden Pfad. Solche Störstellen entstehen z. B. durch unreines Material bzw. durch Beanspruchung durch hohe Felder.
Electromigration ist die Verschiebung von Material durch einen Elektronenwind, der sich bei extrem hoher Stromdichte bildet.
Gemäß Black's Law sinkt die MTTF quadratisch mit d. Stromdichte und exponentiell mit d. Temperatur
Electrical Overstress ist eine Überbeanspruchung durch zu hohe Spannungen, wie z. B. Spannungs-spitzen bzw. Blitzschläge.
Hohe Temperatur, Temperaturzyklen sowie hohe Spannungen und Ströme wirken sich negativ auf die Lebensdauer eines Chips aus. Ebenso kann unsach-gemäßes Handling zur Zerstörung führen (ESD).
Der Einfluss der Temperatur auf die Lebenserwar-tung eines Chips wird durch die Arrhenius-Glei-chung beschrieben: Temperaturerhöhung bewirkt ein exponentielles Ansteigen der Fehlerrate.
Beim Burn-in versucht man die Phase der infant mortality durch Temperaturzyklen noch vor der Auslieferung zu überwinden.
Physikalische Defekte können sich in verschieden-ster Weise auf die logische Funktion eines Chips auswirken. Typische Manifestationen sind Kontakt-probleme, Isolationsprobleme, parametrische Fehler, dynamische Fehler und Speicherfehler.
Das Stuck-at Fehlermodell nimmt als Fehlermani-festation an, dass ein Schaltungsknoten auf einem bestimmten Logikpegel „festsitzt". Dieses Modell trifft zwar selten wirklich zu, hat sich in der praktischen Anwendung jedoch bewährt.

Testing

Gemäß der „Rule of ten“ steigen die Kosten für einen Defekt mit jedem Assemblierungsschritt um den Faktor 10.

Tests werden nicht nur in der Fertigung durch-geführt. Sie begleiten den Chip während seines gesamten Lebenszyklus.

Übliche Maße für die Testqualität sind Defect Level und Test Coverage.

Fehler in redundanter Logik können prinzipiell durch einen Test nicht erkannt werden.

Die Testbarkeit ist bestimmt durch Beobachtbarkeit und Steuerbarkeit.

Beim Exhaustive Testing werden alle möglichen Testvektoren angelegt und das Testobjekt somit vollständig getestet. Diese Methode führt in der Praxis meist zu einer unrealistisch hohen Anzahl von Testvektoren.

Beim Deterministic Testing wird versucht, systematisch eine Liste von Testvektoren zu erstellen. Dabei bedient man sich der Fehlersimulation sowie einiger Techniken zur Reduktion (Äquivalenz, Dominanz)

Beim nondeterministic Testing werden die Testvektoren nach dem Zufallsprinzip gewählt. In der Praxis bewährt sich diese Methode (bei sinnvoller Handhabung) sehr gut.

Der Test sequentieller Logik ist besonders problematisch, da zunächst das Testobjekt in einen bestimmten Zustand gebracht werden muss.

Der Scan-Test vermeidet dieses Problem, indem alle Register zu einem Schieberegister verbunden werden. Mit diesem wird der Testvektor an die verbleibende kombinatorische Logik geführt.

Der Scan Test ist ein struktureller Test: Er überprüft (im Gegensatz zum funktionalen Test) nicht die spezifizierte Funktion der Gesamtanordnung, sondern nur die Funktion der verwendeten logischen Gatter.

Beim Speicher erweist sich aufgrund seiner einfachen Funktionalität und der etwas anderen Fehlermodelle ein funktionaler Test als die bessere Lösung.

March-Tests bieten eine besonders hohe Effizienz beim Speichertest.

Für das Testen von Leiterplatten hat wurde der Boundary-Scan standardisiert.

Der Test-Access Port ermöglicht eine effiziente Kommunikation mit der Testlogik auf dem Chip.

Beim Built-in Self-Test befindet sich die Testlogik auf dem Chip. Dies verursacht zwar einen HW-Overhead, erhöht die Test-Performance jedoch entscheidend und erlaubt eine Wiederverwendung in anderen Testphasen.

Für das Generieren von pseudozufälligen Testmustern werden Linear Feedback Shift Register (LFSR) verwendet.

Die Compaction der Responses erfolgt in einem Multiple Input Shift Register (MISR). Sie führt zu Aliasing.

Der Logikanalysator

Ein Logikanalysator kann wesentlich mehr Kanäle gleichzeitig darstellen als ein Oszilloskop, eignet sich aber nicht zur Darstellung des analogen Signalverlaufs.
Das Eingangssignal wird zunächst digitalisiert, wobei der Schwellwert an den Logikstandard der HW anzupassen ist.
Für die nachfolgende zeitliche Diskretisierung wird ein Takt benötigt. Im Timing Mode wird dieser vom Messgerät geliefert, im State Mode vom Target.
Bei entsprechender Überabtastung kann im Timing Mode der Zeitverlauf noch recht genau verfolgt werden.
Im State Mode ist die zeitliche Position der Flanken nicht mehr genau sichtbar, man erkennt dafür die Folge der synchronen Zustände.
Da der Speicher begrenzt ist, muss im Timing Mode ggf. die Abtastrate verringert werden um ein genügend großes Intervall aufzuzeichnen. Für Signale mit geringer Aktivität ist Transitional Sampling vorteilhaft.
Der Trigger dient als Referenzpunkt für die Auf-zeichnung. Als Triggerbedingung wird daher üb-licherweise das interessierende Event formuliert.
Im Pretrigger-Bereich ist die Aktivität unmittel-bar vor dem Trigger sichtbar, im Posttrigger die unmittelbar nachfolgende Aktivität.
Die aufgezeichneten Daten können entweder in Form einer Liste oder als Timing-Diagramm dargestellt werden.
Richtiges Probing ist wesentlich, um unverfälschte Messergebnisse zu erhalten.

Anhang: Beispiele

Theoriefragen – 2.6.2015

Frage 1:

Nennen Sie 5 Vorteile der Digitalisierung von Signalen bzw. Information!

Frage 2:

Sie wollen ein Logiksignal vom Ausgang eines Gatters G1 (Ausgangswiderstand R = 100Ω) zum Eingang eines Gatters G2 (Eingangskapazität C = 0,1pF) leiten (Leitung ideal). Schätzen Sie ab, welches Delay sich durch die RC-Konstante ergibt!

Frage 3:

Was besagt das Theorem von Shannon? Wo wird es praktisch verwendet?

F(A,B,C) ----> ----
Folie 37 und für MUX

FPGA

Frage 4:

Was versteht man unter Bonding?

Frage 5:

Fehler die kein/n oder Chip verwirft

Erläutern Sie, wie physikalische Defekte zu dynamischen Fehlern führen können!

Folie 49, 51, 50

Frage 6:

Wozu verwendet man eine Antifuse?

OTP FPGA, Verbindungen

Frage 7:

Was versteht man unter infant mortality?

Badewannenkurve (Achsen beschriften)

Frage 8:

Was besagt die „Rule of Ten“?

Beispiel 1

Entwerfen Sie einen 16-bit Pseudo-Zufallsgenerator mit folgenden Schritten: Schieberegister mit Rückkopplungen (Test Kapitel)

(a) Stellen Sie den Aufbau eines 16-bit Schieberegisters aus Flip-Flops dar. Schließen Sie jeweils Takteingang, Dateneingang und Ausgang korrekt an!

(b) Ergänzen Sie diese Schaltung geeignet zu einem LFSR mit dem Polynom
X16 = X8 ⊕ X4 ⊕ X3 ⊕ X0
(Hinweis: verwenden Sie einen Block namens „odd Parity“ bzw. „even Parity“ für die Realisierung eines XOR mit mehr als 2 Eingängen)

(c) Stellen Sie dar, wo man die Zufallszahl abgreifen kann

(d) Was ist der Unterschied zwischen diesem Pseudo-Zufallsgenerator und einem „echten“ Zufallsgenerator (bezogen auf die gelieferten Zahlenfolgen am Ausgang)?

(e) Nehmen Sie an, der Generator sei auf den Wert „0110110100111100“ initialisiert. Welchen Wert hat der Ausgang nach 3 Taktzyklen (= aktiven Taktflanken)?

a) 16 bit Schieberegister

b) Verbindung von 0, 3, 4, 8 weg zu even Par (Folie 31 LFSR)

c) Bei Ausgang von FF

d) Bei Pseudo: Reproduzierbar, periodische Zufallsfolge

e) Folie 32 LFSR

Tabelle: Parity über 0, 3, 4, 8 berechnen, um 1 verschieben, ganze Kette

Beispiel 1

Gegeben ist die Multiplexer-basierte Implementierung einer Funktion F(A,B,C) laut Abbildung 1.1.

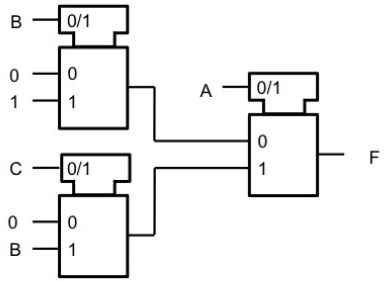


Abbildung 1.1: Implementierung der Funktion F mittels Multiplexern

(a) Ermittlen Sie *anhand der Schaltung* (nicht Wahrheitstabelle) die Funktion F(A,B,C)!

(b) Zeichnen Sie das KV-Diagramm und geben sie die *konjunktive* Normalform an!

(c) Geben Sie für die verwendeten Multiplexer eine Realisierung mittels Transmission Gates und Invertern an (mit geeigneter Entkopplung von Eingang und Ausgang)! Wie viele Transistoren benötigt ein Multiplexer, wie viele die gesamte Implementierung von F(A,B,C) lt. Abbildung 1.1 (für interne Verbindungen können Sie die Entkopplung nun einsparen)?

(d) Entwerfen Sie eine alternative Realisierung von F(A,B,C) mittels AOI bzw. OAI! Welche Variante (OAI oder AOI) ist günstiger und warum?

(e) Stellen Sie den Aufbau Ihrer Lösung als Transistorschaltung aus p-Stack und n-Stack dar (vergessen Sie nicht, den Ausgang zu kennzeichnen)! Falls Sie Inverter benötigen, geben Sie einmal den Aufbau eines Inverters aus Transistoren an, und verwenden Sie in Ihrer Schaltung für F(A,B,C) zur besseren Übersichtlichkeit nur das Invertersymbol.

(f) Wie viele Transistoren benötigen Sie nun für die Implementierung von F(A,B,C)? Vergessen Sie nicht, die Transistoren für die Inverter zu berücksichtigen!

a)

mittels Shanon lösen

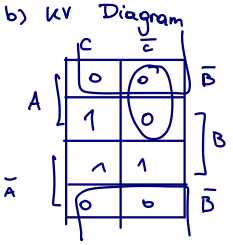
$(\bar{A} \wedge F_1) \vee (A \wedge F_2)$

$F_1 = (\bar{B} \wedge 0) \vee (B \wedge 1) = B$

$F_2 = (\bar{C} \wedge 0) \vee (C \wedge B) = B \wedge C$

$(\bar{A} \wedge B) \vee (A \wedge B \wedge C)$

b) KV Diagram



c) Folie 47 MUX Realisierung

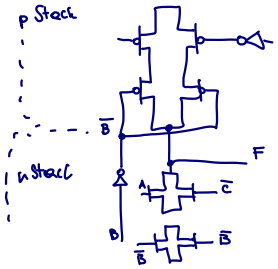
Trans: 12

in/out Buffer wg (2 inverter hinter einander)

d) AOI

OAI

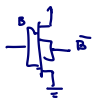
a) $(\bar{A} \wedge B) \vee (B \wedge C)$



e) f) 12 Trans = 10 + 2 (für Inv)

KNF: nur a inv, b/c nicht

Inverter zeichnen!



Beispiel 2

Gegeben ist der in Abbildung 2.1 dargestellte Synchronizer mit 500MHz Takt (f_{clk}). Ein asynchrones Eingangssignal (f_{input}) mit 93MHz liegt am Eingang von Flip-Flop *FF1*. Der Ausgang von *FF1* geht direkt an den Eingang von Flip-Flop *FF2*. Laut Datenblatt haben die beiden Flip-Flops folgende Parameter:

setup time t_{SU} = 220ps; hold time t_H = 160ps; clock to output delay t_{CO} = 350ps;
metastability characteristics: τ_C = 50ps ; T_0 = 100ps

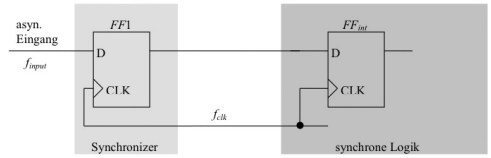


Abbildung 2.1

(a) Wie groß ist die Resolution Time für FF_{int}? Welche MTBU ist zu erwarten?

Zur Erhöhung der MTBU soll der Synchronizer zweistufig gemacht werden, indem ein weiteres Flip-Flop, *FF2*, mit identischen Eigenschaften zwischen den Ausgang von *FF1* und den Eingang von FF_{int} geschaltet wird. Damit ergibt sich die in Abbildung 2.2 dargestellte Schaltung.

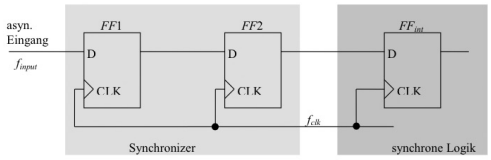


Abbildung 2.2

(b) Welche Resolution Time und MTBU ergeben sich mit diesem neuen Synchronizer?

(c) Ihr Chef möchte, dass Sie FF2 mit invertiertem Takt ansteuern, damit die Latenzzeit des Synchronizers vermindert wird. Er argumentiert, dass seine Lösung zwar schlechtere MTBU aufweist als die nach Abb. 2.2, aber immer noch bessere als jene nach Abb. 2.1. Ist das korrekt? Begründen Sie ihre Antwort!

a) Resolution Time für FF_1

Was von Taktperiode übrigbleibt

2ns → 2000 ps

$t_r = 2000 - 350 - 220 = 1430$ ps
(350: clk, 220: setup)

Formel:

$$MTBU = \frac{1}{f_{clk}} \cdot \frac{1}{t_{dat}} \cdot T_0 \exp\left(\frac{t_r}{\tau_c}\right)$$

$$= \frac{1}{500 \cdot 10^6} \cdot \frac{1}{e3 \cdot 10^6} \cdot 100 \cdot 10^{12} \cdot \exp\left(\frac{1430\,ps}{50\,ps}\right)$$

b) Resolution time:

Achtung MTBU nicht immer ausrechnen!

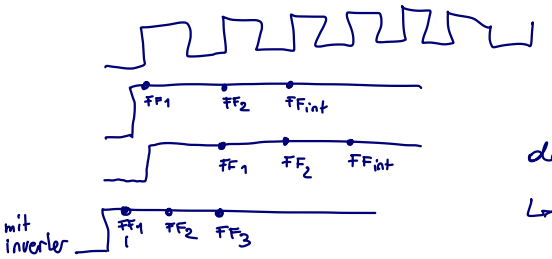
$t_{res} = T_{clk} - t_{co} - t_{su}$

$t_{res}' = 2T_{clk} - 2t_c - t_{su}$

$\rightarrow t_{res}' = 2\,t_{res} = 2860$ ps

c) normal:
bis zu 3 Taktflanken (2–3 Taktzyklen)

invertier bei Takt:
FF2 übernimmt bereits nach 1/2 Zyklus



delay bei 2 - 3 Taktzykl.
↳ schneller mit Latenz

Restime mit inv. $t_{res} = \frac{T_{clk}}{2} - t_{co} - t_{su} + \underbrace{\frac{T_{clk}}{2} - t_{co} - t_{su}}_{\text{2. Stufe}} = T_{clk} - 2t_{co} - t_{su}$

Chef hat unrecht!

Rechenbeispiele

Beispiel 1

Gesucht ist eine Schaltung, für ein 2-aus-3 Threshold Gate mit Hysterese: Sie soll den Ausgang (y) genau dann auf 1 setzen, wenn an zumindest zwei der drei Eingänge (a,b,c) logisch 1 anliegt. Der Ausgang wird genau dann auf 0 gesetzt, wenn an allen drei Eingängen logisch 0 anliegt. Für alle anderen Eingangskombinationen wird der letzte gesetzte Wert gehalten (Hysterese).

(a) Da die Schaltung ein „Gedächtnis" für den letzten gesetzten Wert benötigt, brauchen Sie ein Speicherelement. Verwenden Sie dafür ein SR-Latch mit Beschaltung laut Abbildung 1.1. Erstellen Sie KV-Diagramm und DNF für die kombinatorischen Blöcke „set" und „reset"!

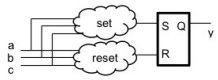


Abbildung 1.1

Nehmen Sie für (b) bis (e) an, Sie hätten zur Implementierung nur NOR-Gatter (mit bis zu 4 Eingängen) und Inverter zur Verfügung.

(b) Wie würden Sie das SR-Latch mit NOR Gattern realisieren (Schaltplan)?

(c) Beim SR-Latch gibt es eine illegale Eingangskombination für R und S. Welche ist dies und wie verhält sich Ihr Latch dann?

(d) Kann diese Kombination in Ihrer Implementierung des Threshold Gate auftreten? Falls ja, für welches (a,b,c)? Falls nein, warum nicht?

(e) Betrachten Sie die kombinatorische Funktion für die„set" Logik: Wie können Sie diese nur mit NOR und Invertern realisieren (Schaltplan)?

(f) Ihr Chef hat vor seinem Urlaub die in Abb. 1.2 dargestellte Implementierung für die set-Logik skizziert. Beschalten Sie die Eingänge e1 … e4 passend!

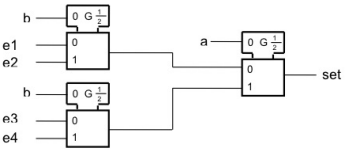


Abbildung 1.2

(g) Erarbeiten Sie eine alternative Implementierung der gesamten Threshold-Funktion mittels LUT (ohne Verwendung eines SR-Latch!): Erstellen Sie dazu die Wahrheitstabelle y(a,b,c,y').

(h) Wie viele Eingänge und Ausgänge benötigt die LUT, wie werden diese beschaltet (Skizze)? Geben Sie den Speicherinhalt der LUT an!

→ Wert muss gehalten werden

a) SET: Wahrheitstabelle

a	b	c	set	reset
0	0	0	0	1
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

b) Folie 39 – SR LATCH

c) illegal: auf beiden Q/$\bar{Q}$ ein 0
obwohl $\bar{Q} \neq Q$!
R=1, S=1, bei Q/$\bar{Q}$ kommt 0 raus

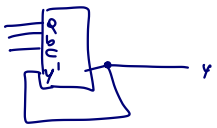
d) zum auftreten set & reset ein 1 in Wahrheitstabelle, ≠ nicht!
(Nein)

e) DNF bauen
und Terme invertieren, oder

2B: $(A \wedge B) \vee (A \wedge C)$
$(\bar{A} \vee \bar{B}) \vee (\bar{A} \vee \bar{C})$ nor

f) mit Shannon zerlegen

g) mittels Codetabelle
in: a,b,c
out: y



in			out	
a	b	c	y'	y
0	0	0	0	0
1	1	0	*	1
1	0	1	*	1
0	1	1	*	1
1	1	1	*	1
1	0	0	1	1
0	1	0	1	1
0	0	1	1	1

→ 11/16 auf 1

h) In: 4
Speicherinhalte: ?

Beispiel 3

Gegeben ist die Schaltung in Abbildung 3.1.

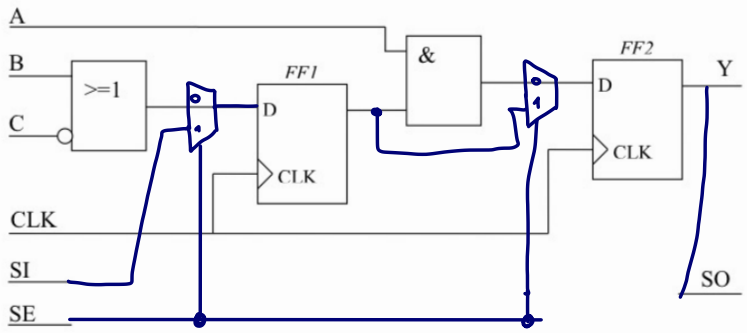


Abbildung 3.1

(a) Wie müssen Sie ein D-Flip-Flop erweitern, um daraus ein Scan-Register zu machen? Skizzieren Sie diese Erweiterung!

(b) Erweitern Sie die beiden Flip-Flops in Abbildung 3.1 entsprechend!

(c) Bilden Sie mit den beiden so entstandenen Scan-Registern eine Scan Chain. Schließen Sie die Signale SI (scan in), SO (scan out) und SE (scan enable) entsprechend an.

(d) Betrachten Sie das AND-Gate: Welche Testmuster müssen Sie an dieses Gatter anlegen, um alle Stuck-at Fehler an seinen Eingängen und seinem Ausgang zu finden?

(e) Beschreiben Sie den Ablauf eines Scan Tests anhand der Schaltung für das Beispiel eines SA0 am unteren Eingang des AND-gates.

a. Welchen Pegel müssen Sie dort anlegen um den Fehler zu aktivieren?

b. Welche Sequenz von Signalen an SE, SI und CLK benötigen Sie, um das zu erreichen?

c. Welche Sequenz von Signalen müssen Sie als nächstes anlegen, um den Fehler an SO sichtbar zu machen?

Scantest

a) FF1 : Overhead - Scan Register hinzufügen

b) MUX vor D eingänge SI / SE ausschließen

c) ✓

d) F 23/24 Fehlerdominanz ... :

e)
a) bei SE an 1
Reintakten (2x)
Umschalten (normal)
Takten (und)
2x Takten → Muster