Vorlesung 5

Thema 1: Setup und Hold

In Vorlesung wurden Flipflops beschrieben.

Folien 4 - 14

***

Hold-Zeit

Stellen wir uns vor, wir haben zwei Flipflops und die kombinatorische Logik dazwischen. Auf steigende Taktflanke Ck(i) ändert sich der Ausgang des Flip-Flops 1 (Folie 7). Genau genommen, das zweite Latch im Flipflop FF1 wird transparent und leitet den Wert vom seinem Eingang bis zum Ausgang Q1 durch. Gleichzeitig kommt auch das erste Latch im Flipflop 2 in Speicherzustand. Der Eingang D2 wird vom Latch getrennt. Wichtig ist das folgende: Die Änderung am D2 (verursacht durch die Änderung an Q1 wegen der Taktflanke Ck(i)) darf nicht passieren bevor das Latch 1 in Flipflop 2 in Speichermodus kommt.

Wir definieren als Hold Time die Zeit die Eingang D2 nach der aktiven Taktflanke noch unverändert bleiben muss (also im D2(i) Zustand) so dass D2(i) im Flipflop gespeichert wird (Folie 5).

Folien 6 – 9 zeigen die zeitliche Reihenfolge von Vorgängen nach der aktiven Taktflanke.

Folie 6: Taktflanke

Folie 7: Nach der hold-Zeit werden die Schaltern geöffnet

Folie 8: Q1 ändert sich

Folie 9: D2 ändert sich

Die D2 – Änderung passiert hach der hold-Zeit, das ist in Ordnung.

Es wird ein Slack (etwa wie Zeitlücke) wie folgend definiert:

Slack = Ck1 + Delay – (Ck2 + Thold)

Ck1/2 ist der Moment der steigenden Taktflanke am Takteingang FF1/FF2.

Beachten wir, dass verschiedene Flip-Flops in verschiedenen Momenten das Taktsignal empfangen können.

Delay ist die Gesamtverzögerung des Signals von der steigenden Taktflanke am FF1 bis zum Dateneingang des FF2.

Thold ist die hold-Zeit.

Wenn Slack > 0, ist „alles in Ordnung“.

Wenn sich D2 zu schnell ändert, also wenn sich D2 ändert bevor Latch 1/FF2 in Speichermodus kommt wird der falsche Zustand D2(i+1) im Flipflop 2 gespeichert. Wir reden von einer Hold Time Verletzung.

Hold Time Verletzung passiert wenn sich Niveau am Eingang D2 zu schnell ändert. Die Ursache könnte ein schlechtes Design des Flipflops sein oder, dass der Takt Ck2 später ankommt als Ck1. Das letzte könnte bei einem nichtoptimalen Taktbaum passieren. Verzögerung in der kombinatorischen Logik zwischen den Flipflops hilft.

Folien 10 – 14 zeigen der Fall einer hold-Time Verletzung. Die Ursache ist hier ein zu spät eintreffendes Ck2-Signal, da wir einen Takt-Buffer zwischen zwei Flip-Flops haben.

Folie 11: Taktflanke am FF1

Folie 12: Nach einer Delay-Zeit kommt die Änderung an D2, Ck hat sich am FF2-Takteingang noch nicht geändert. FF2/Latch1 ist noch transparent und nimmt das neue D2 Wert auf.

Folie 13: Erst jetzt kommt die Taktflanke an FF2-Takteingang

Folie 14: Das neue D2 (i+1) Wert wird am Q2 sichtbar –> Hold-Verletzung

Folien 16 - 32

Setup-Zeit

Die Änderung am D2, verursacht durch die Änderung an Q1 bei der Taktflanke Ck(i), soll geschehen bevor die nächste Taktflanke Ck(i+1) das Flipflop 2 erreicht und das Latch 1/FF2 den transparenten Modus verlässt.

Wir definieren die Setup-Time (Folie 17) als den letzten Zeitpunkt vor der aktiven Taktflanke, wo sich D noch ändern muss, so dass die Änderung sicher gespeichert wird. (Oft können die D-Änderungen noch kurze Zeit nach der Taktflanke gespeichert werden, in dem Fall ist die setup-Zeit negativ.)

Wenn sich D2 nach der Setupzeitpunkt ändert, wird noch der alte Zustand (D2(t)) auf die Taktflanke Ck(i+1) gespeichert, was falsch ist. Wir reden von einer Setup Time Verletzung.

Setup-Zeit-Verletzung passiert wenn sich Niveau am Eingang D2 zu langsam ändert. Das passiert am meistens wenn die Taktfrequenz zu hoch ist oder die kombinatorische Logik zu langsam.

Folien 18 – 24 zeigen die zeitliche Reihenfolge von Vorgängen nach der aktiven Taktflanke im Fall ohne setup-Zeitverletzung.

Folie 18: Taktflanke i am FF1 (Ck1 (i))

Folien 19 - 22: Nach einer Verzögerung ändert sich D2 = D2(i+1). Die Verzögerung setzt sich aus der Verzögerung im FF1 und der Verzögerung in der kombinatorischen Logik (der Kreis im Bild) zusammen.

Folie 23: Taktflanke i+1 am FF2 (Ck2 (i+1)). Der neue Zustand D2(i+1) wird gespeichert.

Auch hier wird ein Slack (Zeitlücke) definiert:

Slack = Ck2 - Tsetup - (Ck1-Tck) - Delay

Ck2 ist der Moment der steigenden Taktflanke (i+1) am Takteingang FF2.

Ck1-Tck ist der Moment der steigenden Taktflanke (i) am Takteingang FF1. Tck ist die Taktperiode.

Delay ist die Gesamtverzögerung des Signals von der steigenden Taktflanke am FF1 bis zum Dateneingang des FF2.

Tsetup ist die setup-Zeit.

Wenn Slack > 0, ist „alles in Ordnung“.

Folien 25 – 32 zeigen der Fall einer setup-Time Verletzung

Folie 26: Taktflanke i

Folien 27-29: Das Signal ist noch von FF1 nach D2 „unterwegs“

Folie 30: Taktflanke i+1

Folie 31: Setup-Zeitpunkt (definiert als Ck2(i+1)+Tsetup), D2 hat sich noch nicht geändert

Folie 32: Erst jetzt haben wir die D2(i+1) Änderung -> Setup-Verletzung

Folie 33

***

Beachten wir das folgende – Setup-Zeit Verletzungen kann man durch langsamere Taktfrequenz verhindern. Hold-Zeit Verletzungen kann man, wenn sie vorhanden sind, nicht mehr entfernen.

Wenn eine Schaltung Hold-Zeit Probleme hat, kann man sie in der Regel nicht verwenden.

Hold-Zeit Probleme verhindert man im Design durch eine künstliche Taktverlangsamung am Empfänger Flipflop. Diese nennt man Clock Uncertainty. Auf diese Weise wird Synthese Tool gezwungen D2 in Bezug auf Ck-Eingang am FF2 zu verlangsamen. Das erreicht das Tool z.B. durch Einfügen von Invertern im Datenpfad.

Thema 2: Kodierer (Encoder)

Folie 35

***

Um eine Information bearbeiten zu können muss man sie in einem geeignetem Zahlsystem z.B. im binären System darstellen.

Nahmen wir z.B. an, wir haben eine Tatstatur, jeder Taste gehört ein Digitaleingang. Ein Kodierer ist ein kombinatorisches Netzwerk. Es erzeugt den binären Code, der der aktivierten Taste entspricht.

Der einfachste Kodierer setzt voraus, dass nur ein Eingang in einem Moment aktiv ist – das heißt es wird in einem Moment nur eine Taste gedrückt.

Nehmen wir als Beispiel acht Eingänge A0 bis A7.

Folie 36

***

Kombinatorische Tabelle dieses Kodierers wird in Folie 36 gezeigt.

Diese Tabelle kann mit ODER Gattern realisiert werden

Y0 = A1 || A3 || A5 || A7

Y1 = A2 || A3 || A6 || A7

Y2 = A4 || A5 || A6 || A7

Wir sehen, dass ein Kodierer umgekehrte Funktionalität wie der Decoder hat.

Folie 37

***

Beachten wir, dass der Kodierer nur dann richtig funktioniert, wenn nur ein Eingangssignal aktiv ist. Wenn z.B. A3 und A4 gleichzeitig aktiv werden, bekommen wir am Ausgang den Code Y0 = Y1 = Y2 = 1 also 7 statt 3 oder 4.

In den Systemen wo mehrere Eingänge gleichzeitig aktiv werden können werden die Prioritätskodierer benutzt. Diese erzeugen den Kode des Eingangs mit höchster Priorität – z.B. den größeren Kode.

Folie 38/39

***

Man kann den Prioritätskodierer mithilfe eines einfachen Kodierers und eines Prioritäts-Netzwerks aufbauen. Das Prioritätsnetzwerk soll gewährleisten, dass nur ein Ausgang aktiv ist, ungeachtet von der Zahl der aktiven Eingängen. Z.B., wenn A3 und A4 Eingänge aktiv sind, soll nur AP4 aktiv werden.

Das Prioritätsnetzwerk kann mit folgenden Funktionen beschrieben werden:

AP7 = A7

AP6 = A6 & !A7

AP5 = A5 & !A6 & !A7

...

AP0 = A0 & !A1 & !A2 & !A3 & !A4 & !A5 & !A6 & !A7

Diese Funktionen können wie in Folie 39 realisiert werden.

Folie 40

***

Ein Prioritätskodierer hat oft die Signale Prio_Input und Prio_Output die benutzt werden können um einen größeren Kodierer als Kaskade von mehreren kleineren zu realisieren.

Prio Output ist die ODER Funktion vom Prio_Input und allen Eingängen.

Die Ausgänge sind nur dann Aktiv wenn Prio Input = 0 ist.

Thema 3: Minimierung von Schaltfunktionen - Karnaugh Tabellen

Folie 42

***

Wir haben gesehen, dass man eine kombinatorische Tabelle als eine disjunktive Normalform darstellen kann.

Jeder Zeile mit dem Ergebnis Eins entspricht eine UND Funktion, die Gesamttabelle ist ODER Funktion von einzelnen Zeilen.

Wir haben gesehen, dass die Normalform oft vereinfacht werden kann – ein Beispiel ist:

AB || !AB = B

& schreiben wir hier als Produkt.

Folie 43

***

Karnaugh-Tabelle ist eine Graphische Darstellung der Wahrheitstabelle. Es ist aus einer Kanaugh-Tabelle leicht zu erkennen ob eine Normalform vereinfacht werden kann und wie.

Eine Karnaugh-Tabelle für n Eingangsvariablen hat 2^n Felder.

Am Rand der Tabelle werden die Variablen beschriftet – jede Zeile gehört einer Variable (negiert oder nicht-negiert) oder einem Produkt von zwei/drei (negierten oder nicht-negierten) Variablen – die negierte Variable wird mit Null oder !Xi beschriftet.

Wichtig ist, dass sich horizontal und vertikal benachbarte Felder nur in genau einer Variablen unterscheiden dürfen. Gray Code wird verwendet.

Mithilfe von Wahrheitstabelle wird in einzelnen Feldern Eins eigetragen wenn für die gegebene Variablen-Kombination die entsprechende Zeile eins ist.

Karnaugh-Diagramme eignen sich für die Vereinfachung von Funktionen mit maximal ca. 4–6 Eingangsvariablen; bis 4 Variablen sind sie übersichtlich

Folie 44

***

Diese Folie zeigt die Karnaugh-Tabelle der Funktion Y = DCBA + DCB!A + D!CBA D!CB!A

Folien 45/46

***

Beachten wir folgendes – Wenn wir ein Block mit Einsen haben und wenn in diesem Block einige Variablen alle Kombinationen durchlaufen, können wir diese aus dem UND Produkt eliminieren. Der Block wird nur durch die Feste Variablen dargestellt.

Folie 47

***

Wenn wir bis zwei Variablen an einem Rand der Tabelle haben (wie hier DC links und BA oben), und Gray Code verwenden, kann für jeden 2x1 Block eine Variable eliminiert werden, für jeden 2x2 Block zwei Variablen, für jeden 2x4 Block drei Variablen.

Folie 48

***

Die Minimierung wird wie folgend gemacht:

Man versucht, möglichst viele horizontal und vertikal benachbarte Felder, die eine 1 enthalten, zu rechteckigen zusammenhängenden Blöcken zusammenzufassen. Als Blockgröße sind alle Potenzen von 2 erlaubt

Dabei sind alle 1-Felder mit Blöcken zu erfassen

Ein Block kann unter Umständen über den rechten bzw. unteren Rand des Diagramms fortgesetzt werden

Dies erklärt sich folgendermaßen: KV-Diagramme für drei Variablen müssen im Grunde als Zylinder verstanden werden. Die Felder ganz links und ganz rechts bzw. oben und unten sind also benachbart.

Von den ermittelten Blöcken sind so viele auszuwählen, dass alle 1-Felder überdeckt werden.

Folie 49

***

Die gebildeten und ausgewählten Blöcke/Päckchen wandelt man nun in Konjunktionsterme um. Dabei werden Variablen innerhalb eines Blockes, die in allen Formenkombinationen auftreten, weggelassen.

Diese UND-Verknüpfungen werden durch ODER-Verknüpfungen zusammengefasst und ergeben eine disjunktive Minimalform.

Weitere Varianten von K-Tabellen (keine Folien)

Häufig gibt es Boolesche Funktionen mit Wahrheitstabellen, in denen nicht für jede Kombination der Eingangsvariablen ein Wert der Ausgangsvariablen definiert sein muss. Man nennt solche Ausgangszustände Don’t-Care-Terme und bezeichnet sie mit X

Ein Nachteil der ursprünglichen Variante des K-Diagramms ist, dass es für mehr als 4 Eingangsvariablen nicht geeignet ist. Um mit einer beliebigen Zahl von Eingangsvariablen zu arbeiten, kann die verallgemeinerte Form des Symmetriediagramms verwendet werden.

Bei diesem Verfahren wird die Entstehung eines K-Diagramms mit n Eingangsvariablen dadurch erklärt, dass ein K-Diagramm mit n-1 Eingangsvariablen gespiegelt und dadurch verdoppelt wird. Daher die Bezeichnung "Symmetriediagramm". Die neu entstehende Hälfte des K-Diagramms entspricht dann der neu hinzugekommenen Eingangsvariable in nicht-negierter Form, während die bereits bestehende Hälfte der Eingangsvariable in negierter Form entspricht.

Thema 4: Glitch

Folie 51/52

***

Die Verwendung von minimaler Logik führt oft zu einem Problem, genannt Glitch.

Hier geht es um das Zeitverhalten von kombinatorischen Netzen. Wir haben beim Inverter gesehen, dass sich das Ausgangsniveau nach einer Verzögerung

Ändert. Im Fall von einfachsten logischen Funktionen führt es nur zu einer Verspätung des Ausgangssignals.

Im Falle von komplexeren Funktionen, kann es passieren, dass der Ausgangssignal kurze Zeit ein Niveau nimmt, was dem Ausgang nicht entspricht. Es kann auch passieren, dass wir einen kurzen Impuls bekommen während wir erwarten, dass sich das Niveau nicht ändert. Solche Impulse stellen kurze Störungen dar – sie werden Glitch genannt.

Nehmen wir als Beispiel ein Multiplexer aus AND and OR gattern.

Die Funktion des Multiplexers wird wie folgend beschrieben:

F = !Sel A + Sel B

Nehmen wir an, dass beide Eingänge Eins sind: A = B = 1. Sel ist anfangs 1 und ändert sich auf 0. Wir erwarten, dass sich der Ausgang nicht ändert da beide Eingänge identisch sind.

Wenn wir aber die Verzögerung des Signals durch Inverter berücksichtigen, bekommen wir einen Zeitdiagram wie in Folie 52 unten.

Ein kurze Zeit sehen beide AND Gatter den Select Eingang 0, wir bekommen für eine kurze Zeit 0 am Ausgang.

Di Frage ist, ob solch ein Glich problematisch ist.

Folie 53

***

Falls die kombinatorische Logik zwischen zwei flankengetriggerten Flip-Flops steht, also falls wir eine synchrone sequenzielle Schalung haben, ist ein Glich unproblematisch wenn es kürzere Zeit wie die Taktperiode dauert.

Manchmal aber wird kein Taktsignal zur Synchronisierung verwendet, und dann könnte solch ein Glitch problematisch sein.

Folie 54

***

Man kann die Möglichkeit eines Glitches leicht aus der Karnaugh Tabelle erkennen. Folie 54 zeigt die Karnaugh-Tabelle für den Multiplexer.

Zwei Gruppen sind getrennt, aber liegen nah einander. Wenn sich die Variable Sel von 1 auf 0 oder 1 auf 0 ändert, für A = B = 1, wird die Gruppe 1 „ausgeschaltet“ (bzw. ihre UND Funktion wird 0) und 2 eingeschaltet (bzw. ihre UND Funktion wird 1). Wenn das nicht synchron passiert, können wir 0 als Glitch bekommen.

Folie 55

***

Man kann ein Glitch verhindern indem man eine zusätzliche Gruppe (oder Kontur) 3 hinzufügt die als Brücke zwischen den Gruppen 1 und 2 dient. Das Produkt A & B entspricht dieser Gruppe. Beim Sel Änderung (für A = B = 1) wird die Gruppe 3 nicht ausgeschaltet da Select nicht als Variable in dieser gruppe vorhanden ist. Das verhindert ein 0-Glitch.

Folie 56

***

Die kombinatorische Schaltung, welche der Tabelle entspricht, ist in Folie 56 gezeigt. Glitch-freie Schaltungen sind normalerweise komplizierter als die minimalen Schaltungen.

Folie 57

***

Beachten wir, dass die kombinatorischen Schaltungen, implementiert als disjunktive Normalform, kein 1-Glitch erzeugen können unter Annahme dass sich nur eine Eingangsvariable ändert.

Logische null bekommt man am Ausgang nur wenn alle UND Gatter null sind. Einzige Möglichkeit für Glitch 1 wäre wenn ein UND Gate kurze Zeit 1 wird. Das kann nicht passieren wenn sich nur eine Variable ändert.

Folie 58

***

An dieser Stelle könnte man erwähnen, dass die kombinatorischen Schaltungen auch als konjunktive Normalform implementiert werden können. Das wäre eine UND Verknüpfung von vielen ODER Funktionen. Mit ODER „Summen“ würde man die Zeilen in der Wahrheitstabelle beschreiben, die null sind. (Variablen die 1 sind werden negiert.)

Eine Konjunktive Normalform kann keine 0-Glitches haben wenn sich nur eine Variable ändert.

Konjunktive Normalform ist für die Funktionen geeignet, die viele „Einsen“ als Ergebnis haben.

Thema 5: Grey Code

Bei Carnaugh Tabellen wurde Grey code erwähnt.

Folie 60

***

Die Folie 60 zeigt als Vergleich den binären- und Grey Code

Grey Code hat die Eigenschaft, dass sich immer nur ein Bit ändert wenn man hochzählt.

Das ist im Sinne von Glitches vorteilhaft. Wir haben gesehen, dass man kombinatorische Schaltungen glitchfrei machen kann unter der Bedingung, dass sich nur eine Variable ändert. Wenn wir z.B. ein Grey Code Zähler als Eingang für eine kombinatorische Schaltung nehmen, ist das gesichert.

Zweite Verwendung vom Grey Code ist die Zeitmessung. Im Fall vom binären Code besteht die Gefahr, dass man die Zeit-Bits falsch speichert wenn das gemessene Signal im Moment eintrifft wenn sich mehrere Bits gelichzeitig ändern. (Oft sind die Verzögerungen von einzelnen Bit verschieden und eine gewisse Zeit ist der Code falsch.) Im Fall von Grey Code besteht die Gefahr nicht, da sich nur ein Bit ändert.

Man kann aus einem binären Code den Grey Code wie folgend herleiten.

G0 = B1 exor B0

G1 = B2 exor B1

…

Gn-1 = Bn-1

Es gilt auch:

Bn-1 = Gn-1

Bn-2 = Bn-1 exor Gn-2

…

B0 = B1 exor G0

In Vorlesung 6 zeigen wir wie man einen Grey Code Zähler implementiert.

Thema 6: Statemaschine

Finite State Maschine

Folie 64

***

Zustandsmaschinen werden für Ansteuerung von digitalen Systemen verwendet. Man kann sie mit Programmen vergleichen, wo der Programmcode fest ist.

Die Ausgangssignale von Zustandsmaschinen, und allgemein von sequentiellen Schaltungen, hängen nicht nur von momentanen Werten der Eingangsvariablen sondern auch von deren Reihenfolge. Dieses Verhalten kann man nur dann realisieren wenn die Schaltung Speicherelemente enthält.

Falls eine Statemaschine n Speicherelemente hat, kann sie sich theoretisch in 2^n verschiedenen Zustände befinden. Da es eine endliche Zahl von möglichen Zuständen gibt, nennt man solche Zustandsautomate „finite-state“ Maschinen.

Den Zustand des Speicherelements nennt man Zustandsvariable.

Folie 65

***

Die Zustandsmaschinen kann man in zwei Klassen unterteilen.

Beim Moore typ hängt der Ausgang der Zustandsmaschine nur von der Zustandsvariable – d.h. nur vom Zustand der Maschine.

Beim Mealy Typ hängt der Ausgang auch von den Eingängen.

Mealy Maschinen können oft mit weniger Zuständen realisiert werden, brauchen aber kombinatorische Schaltung für die Erzeugung von Ausgangssignalen. Moore Typ Automaten sind einfacher zu beschreiben, brauchen aber oft mehr Zuständen.

Folie 66

***

Man kann in einem Zustandsautomat jede Art von Speicherzellen verwenden um den Zustand zu speichern.

Wenn alle Speicherzellen in Statemaschine den Zustand gleichzeitig ändern, z.B. auf steigende Taktflanke, nennen wir dieses Netzwerk synchron. Synchrone Zustandsmaschine verwenden Filp-Flops als Speicherelemente.

Man kann auch Zustandsmaschinen bauen, deren Zustand sich durch Änderung von verschiedenen Eingangssignalen ändert. Solche Netzwerke nennen wir asynchron. Asynchrone Zustandsmaschinen verwenden Latches als Speicherelemente.

Folie 67

***

Wir werden uns mit asynchronen Zustandsmaschinen befassen.

Die Funktionalität einer Zustandsmaschine kann man z.B. mit einem Zustandsdiagramm beschrieben. Solch ein Zustandsdiagram hat für eine

Statemaschine die gleiche Bedeutung wie eine Wahrheitstabelle für eine kombinatorische Schaltung. Ein Zustandsdiagramm kann entweder graphisch oder als Verilog/VHDL Code dargestellt werden.

Folie 68

***

Zustandsmaschine für einen Timer

Der Timer besteht aus folgenden Komponenten – einem Zähler, einem Komparator, einem Startkopf, einem Drehregler für die Zeiteinstellung und einem Lautsprecher/Klingel. Der Komparator vergleicht den Zähler-Zustand mit der eingestellten Zeit.

Die Eingänge für die State-Maschine sind das Startsignal und der Komparator-Ausgang. Die Ausgangssignale sind ein Reset Signal für den Zähler und ein Signal für den Lautsprecher.

Die Statmeschine braucht noch ein Taktsignal und ein asynchrons Reset.

Folien 69 - 74

***

Der Zustandsdiagramm könnte wie folgend aussehen (Code):

Input clk, reset, start, comp;

Output resetcounter, beep;

Reg [1:0] State;

Parameter IDLE = 2‘b00, RESETCNT = 2‘b01, COUNT = 2’b11, STOP = 2’b10;

Assign resetcounter = (State == RESETCNT);

Assign beep = (State == STOP);

Always @ (posedge clk or posedge reset) Begin

If (reset) State <= IDLE;

Else begin

```
Case (State)

IDLE: begin

If (Start) State <= RESETCNT;

//!Else State <= IDLE;

End

RESETCNT: begin

State <= COUNT;

//Counter <= 0;

End

COUNT: begin

//Counter <= Counter + 1;

If (comp) State <= STOP;

End

STOP: begin

State <= IDLE;

End

Endcase

End//not reset

End//always
```

Folie 69 zeigt auch die graphische Darstellung des Zustandsdiagrams.

Beachten wir, dass die Schleifen im Code weggelassen werden. Oft enthält der Code der Statemaschine auch die Digitalschaltungen, die die Statemaschine ansteuert.

Wir haben für die Zustände Gray Code verwendet da sich bei jedem Übergang nur ein Flipflop ändert was weniger Glitches verursacht.