Probeklausur

Technische Grundlagen der Informatik (TechGI) - Digitale Systeme

Universität:	Technische Universität Berlin
Kurs/Modul:	Technische Grundlagen der Informatik (TechGI) - Digitale Systeme
Bearbeitungszeit:	120 Minuten
Erstellungsdatum:	September 6, 2025

Bearbeitungszeit: 120 Minuten.

Aufgabe 1.

(a) Vereinfachen Sie die Boolesche Funktion

$$F(A, B, C) = A\,\overline{B}\,C \;+\; A\,B\,\overline{C} \;+\; \overline{A}\,B\,C.$$

(b) Geben Sie die Wahrheitstabelle von F in allen 8 Kombinationen von A, B, C an.

(c) Skizzieren Sie eine Schaltungsrealisation von F nur mit NOT-, AND- und OR-Gattern. Beschreiben Sie die Verknüpfungen.

(d) Zeigen Sie, dass F sich als Summe zweier Produkte minimieren lässt, und geben Sie eine minimale Form an.

Aufgabe 2.

(a) Gegeben sei ein 2-Bit-Zähler realisiert durch zwei JK-Flipflops (Q_1, Q_0) mit $J_0 = K_0 = 1$ und $J_1 = K_1 = Q_0$. Bestimmen Sie die nächsten Zustände (Q_0', Q_1') in Abhängigkeit von (Q_0, Q_1).

(b) Zeichnen Sie den Zustandsgraphen des Zählers mit den Zuständen $00, 01, 10, 11$ und beschreiben Sie den Zyklus.

(c) Erklären Sie, wie ein asynchroner Reset implementiert werden könnte und welche Auswirkungen dies auf das Zählen hat.

(d) Geben Sie eine grobe gate-level-Implementierungsidee für die JK-Eingänge an, um das Zählen-Verhalten zu realisieren.

Aufgabe 3.

(a) Definieren Sie eine 1-Bit-ALU-Einheit mit den Operationen $0: A \wedge B, \quad 1: A \vee B, \quad 2: A \oplus B, \quad 3: \neg A$ und einem 2-Bit-Selektor. Beschreiben Sie die Output-Funktionen.

(b) Erweitern Sie die ALU zu einer 2-Bit-Ganzzahlausgabe über zwei 1-Bit-ALUs mit Ripple-Verarbeitung. Skizzieren Sie die Sign- und Carry-Information.

(c) Geben Sie eine kurze Begründung, warum ein Carry-Lookahead-Ansatz Vorteile gegenüber Ripple hat.

Aufgabe 4.

(a) Beschreiben Sie die Prinzipien eines 4-Bit-Registers, realisiert durch D-Flipflops $D_0, \ldots, D_3$ mit Enable. Geben Sie an, wie der Zustand bei Enable=1 aktualisiert wird.

(b) Implementieren Sie ein 4-Bit-Schieberegister mit linkem Shift und einem Parallelladepfad. Beschreiben Sie die Funktionsweise.

(c) Gegeben seien die Bits $A_3A_2A_1A_0$. Erläutern Sie, wie man durch eine einfache Schaltung eine Lese-/Schreibsteuerung realisieren könnte.

Aufgabe 5.

(a) Beschreiben Sie das Konzept einer 4-Eingang-LOOKUP-TABLE (LUT). Welche Art von Logik kann sie implementieren?

(b) Geben Sie für eine gegebene Wahrheitstabelle die Funktionszuordnung in Form eines LUT-Eintrags an.

(c) Diskutieren Sie, wie LUTs als Bausteine in programmierbaren Logikbausteinen verwendet werden können. Nennen Sie Vor- und Nachteile.

Lösungen

Bearbeitungszeit: 120 Minuten.

Aufgabe 1.

(a) Lösung:

$$F(A,B,C) \;=\; A\,\overline{B}\,C \;+\; A\,B\,\overline{C} \;+\; \overline{A}\,B\,C.$$

Um eine kompakte Minimierung zu erhalten, gruppieren wir die Terme nach C und $\overline{C}$: $F \;=\; C\,(A\,\overline{B} \;+\; \overline{A}\,B) \;+\; AB\,\overline{C} \;=\; C\,(A \oplus B) \;+\; AB\,\overline{C}$.

Damit erhält man eine minimale Darstellung als Summe zweier Produkte, wenn man XOR als Baustein zulässt:

$$\boxed{F = AB\,\overline{C} \;+\; (A \oplus B)\,C.}$$

Hinweis: In reinem SOP mit nur AND/OR/NOT bleibt die minimale Form drei Produkte:

$$F = A\,\overline{B}C \;+\; AB\,\overline{C} \;+\; \overline{A}BC.$$

(b) Lösung: Wahrheitstabelle (A,B,C) $\rightarrow F$:

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

(c) Lösung: Schaltungsrealisation mit NOT-, AND-, OR-Gattern (ohne XOR). Nutze die gegebene SOP:

$$F = T_1 \;+\; T_2 \;+\; T_3, \quad T_1 = A\,\overline{B}\,C, \quad T_2 = AB\,\overline{C}, \quad T_3 = \overline{A}\,B\,C.$$

Damit erhält man eine Umsetzung durch drei UND-Terme, deren Ergebnisse durch eine Vieleing-ODER-Verknüpfung zusammengeführt werden.

(d) Lösung: Im rein-AND/OR/NOT-SINN lässt sich F nicht zuverlässig als Summe von zwei Produkttermen geringeren Grades schreiben, ohne XOR zu verwenden. Eine zulässige zwei-Produkte-Form unter Erweiterung um XOR ist:

$$F = AB\,\overline{C} \;+\; (A \oplus B)\,C,$$

wobei (AB) durch $(A\overline{B} + \overline{A}B) dargestellt wird und damit die ursprngliche Funktion deckt$: $(A \oplus B)C \;=\; (A\overline{B}C) \;+\; (\overline{A}BC)$.

Aufgabe 2.

(a) Lösung: Für JK-FF mit J0=K0=1 (Toggle) gilt

$$Q_0' = \overline{Q_0}.$$

Für den zweiten Flipflop gilt J1=K1=$Q_0, d.h. Q_1' = \begin{cases} Q_1, & Q_0 = 0, \\ \overline{Q_1}, & Q_0 = 1. \end{cases}$ Kombiniert man die beiden, ergibt sich

$$Q_0' = \overline{Q_0}, \qquad Q_1' = Q_1 \oplus Q_0.$$

(b) Lösung: Zustandsgraph (Zustände Q1Q0 = 00, 01, 10, 11). Unter Beachtung von Q0' und Q1' ergibt sich folgende Abbildung der nächsten Zustände (Q1',Q0'):

- $00 \to 01 - 01 \to 11 - 11 \to 00 - 10 \to 10$

Zyklus: Ein 3-Zustands-Zyklus $00 \to 01 \to 11 \to 00$ (Periode 3) sowie ein eigenständiger Zustand 10 mit Selbstschleife.

(c) Lösung: Ein asynchroner Reset würde die Klarleitungen CLR_BAR der beiden JK-Flipflops parallel verbinden. Beim Assert-Status (typischerweise niedrig aktiv) werden Q0=0 und Q1=0 unabhängig von Takt und J/K-Eingängen. Nach Freigabe kehrt der Zähler in den normalen Zählbetrieb zurück (hier z. B. 00). Die Zählfolge wird durch den Reset unterbrochen bzw. auf den Startwert gesetzt.

(d) Lösung: Grobe gate-level-Implementierungsidee: - J0 = K0 = 1 (LSB-Flipflop toggelt bei jedem Clock) - J1 = K1 = Q0 (MSB-Flipflop toggelt nur, wenn LSB-Qu bit Q0 = 1) Damit realisiert man das gewünschte Zählverhalten als JK-basierten Binärcounter. Eine einfache Darstellung: Verdrahtung von J0/K0 auf 1; J1/K1 an Q0 anschließen.

Aufgabe 3.

(a) Lösung: Eine 1-Bit-ALU mit 2-bit Selektor S = (S1,S0) besitzt folgende Ausgabenfunktion F(A,B,S): - S = 00: F = A B - S = 01: F = A B - S = 10: F = A B - S = 11: F = ¬A

Man kann dies kompakt über eine mehrstellige Multiplexer-Implementation ausdrücken:

$$F(A,B,S) = (\neg S_1 \wedge \neg S_0)(A \wedge B) \ \vee \ (\neg S_1 \wedge S_0)(A \vee B) \ \vee \ (S_1 \wedge \neg S_0)(A \oplus B) \ \vee \ (S_1 \wedge S_0)\overline{A}.$$

(b) Lösung: 2-Bit-Ganzzahlausgabe über zwei 1-Bit-ALUs mit Ripple-Verarbeitung. Bezeichne die Bits der Operanden als A1A0 und B1B0. Taktimpuls-sequentiell:

- LSB-Stufe (Stufe 0): - S0 = A0 B0 (Operation 2) - C1 = A0 B0 (Carry-Bildung aus der LSB-Stufe, z. B. per UND-Gatter) - MSB-Stufe (Stufe 1) mit Ripple: - D1 = A1 B1 (Operation 2) - S1 = D1 C1 (Operation 2, inkl. C1 als Cin) - C2 = (A1 B1) (C1 D1) (Carry-out der MSB-Stufe)

Damit ergibt sich die zweibitige Ganzzahlausgabe Sum = S1S0 und ggf. C2 als Carry-out. Diese Umsetzung nutzt zwei 1-Bit-ALUs (Operation 2) sowie einfache Gates (UND/ODER) zur Carry-Bildung.

(c) Lösung: Carry-Lookahead bietet Vorteile bei der Reduktion der Verzögerung infolge serieller Carry-Berechnung. Für zwei Bits lassen sich Propagate- und Generate-Signale definieren: - P0 = A0 B0, G0 = A0 B0 - P1 = A1 B1, G1 = A1 B1 Carry-Kette: - C1 = G0 - C2 = G1 (P1 G0) Summe: - S0 = P0 - S1 = P1 C1 Dieses Lookahead-Verfahren reduziert die Zirkulation des Carry-Bandes gegenüber reinem Ripple.

Aufgabe 4.

(a) Lösung: Prinzipien eines 4-Bit-Registers realisiert durch D-Flipflops D0, D1, D2, D3 mit Enable. Jedes FF besitzt D-Eingang, Clock-Eingang und Enable. Wird Enable=1 gesetzt, übernimmt D-input den neuen Zustand zum nächsten Clockedge; bei Enable=0 bleibt der Zustand stabil. Typischer Aufbau:

- D0..D3 jeweils mit einem Multiplexer am D-Eingang, der zwischen externem Dateneingang (D0..D3) und dem aktuellen Qn-Wert (Q0..Q3) je nach Enable schaltet. - Wenn Enable=1, Dn = externer Eingangn; wenn Enable=0, Dn = Qn (Latch-Verhalten).

Die Zustandsgleichungen lauten: Qn(new) = Enable ? Dn : Qn(old).

(b) Lösung: 4-Bit-Schieberegister mit linkem Shift und Parallelladepfad. Typischer Aufbau mit D-Flipflops D3..D0 und MUXen:

- Parallelladepfad: Wenn Load=1, werden D3,D2,D1,D0 mit den Eingängen ParallelIn[3:0] belegt. - Shiftpfad nach links: D3(Connection zu D2), D2(zu D1), D1(zu D0), D0(SerialIn). Die Selecting-Logik zwischen Shift- und Load-Pfad wird durch einen 2-to-1 MUX pro Bit realisiert, gesteuert durch Load bzw. Shift-Signal.

Funktionsweise: Bei Load=1 werden die 4 Bits parallel in das Register geladen; bei Load=0 erfolgt ein Linksshift, wobei Bit 0 das seriellinhausale Eingangsbit erhält und die übrigen Bits eine Verschiebung nach oben durchführen.

(c) Lösung: Gegeben A3A2A1A0. Lese-/Schreibsteuerung realisiert man durch eine 2-zu-1-Mux- oder eine kontrollierte Busverbindung:

- Write-Phase: Datenbus (z. B. aus A3A2A1A0) wird in das Register geladen (parallel). - Read-Phase: Registerausgänge Q3Q2Q1Q0 werden auf dem Bus ausgegeben.

Eine einfache Umsetzung ist ein Vierfach-MUX, der je Bit den Eingang Dn (Write-Bus) oder den bus-Ausgang Qn (Read) je nach RW-Signal durchschaltet. RW=1 bedeutet Lesen (Ausgang auf Bus), RW=0 bedeutet Schreiben (Bus auf Eingang Dn).

Aufgabe 5.

(a) Lösung: Eine 4-Eingang-LOOKUP-TABLE (LUT) implementiert eine beliebige Boolesche Funktion f(A,B,C,D) mit 4 Eingängen. Die LUT speichert für jede Eingangsbelegung einen Ausgangswert (0 oder 1). Damit kann jede Boolesche Logik mit bis zu 4 Eingängen realisiert werden, egal welche Struktur dahinter (KNOTENstruktur, MUX-basiert, ROM-basiert etc.).
(b) Lösung: Für eine gegebene Wahrheitstabelle wird die Funktionszuordnung als LUT-Eintrag festgelegt. Die übliche Vorgehensweise: - Ordne die Eingänge A,B,C,D in einer festen Reihenfolge (z. B. A als MSB, D als LSB). - Die LUT besitzt 16 Bits, wobei das i-te Bit den Funktionswert f(i) angibt, mit i 0,...,15 und i gleich der Binärdarstellung von (A,B,C,D). - Der LUT-Eintrag ist damit der 16-Bit-Vektor [f(0000), f(0001), ..., f(1111)].

Beispiel (Beispiel-Funktion F(A,B,C,D) = A B C D): Die Ausgabewerte lauten (für i = 0..15): 0,1,1,0,1,0,0,1,1,0,0,1,0,1,1,0. Damit ergibt sich der LUT-Eintrag LUTword(F) = 0b01101001 10010011 0? (16 Bit) → genau: 0 1 1 0 1 0 0 1 1 0 0 1 0 1 1 0 entsprechend der Reihenfolge f(0000) bis f(1111). In hex: 0x6996.
(c) Lösung: LUTs als Bausteine in programmierbaren Logikbausteinen (z. B. FPGAs) bieten Vorteile wie grosse Flexibilität, geringe Tiefe der Logikpfade bei vielen Funktionen, einfache Rekonfiguration und gute Platz-Nutzen-Verhältnisse für komplexe Funktionen. Nachteile umfassen größere Speicherkosten pro Funktion, potenziell höhere Leckströme, und Abhängigkeit der Laufzeit von LUT-Grösse sowie begrenzte lokale Schaltgeschwindigkeit bei sehr großen Funktionen. In FPGA-Architekturen dienen LUTs als fundamentale Bausteine, die durch Register-Stacks, Routing und Lookahead-Logik zu komplexen Schaltungen verbunden werden.