Klausur zur Vorlesung Digitaltechnik

Name, Vorname: _____ Matrikelnummer: ⌊⌋⌊⌋⌊⌋⌊⌋⌊⌋⌊⌋⌊⌋

Aufgabe 1 Verständnisfragen (10 Punkte)(5 × 2)

Beantworten die folgenden Fragen mit *wenigen kurzen Sätzen*.

a) Was ist der (obere und untere) *Störabstand* bei der Darstellung von Binärwerten als Spannungsbereiche und wofür wird dieser benötigt?
Der Eingangsbereich muss breiter definiert werden als der Ausgangsbereich, um Störungen bei der Übertragung von einem Gatter-Ausgang zum nächsten Gattereingang zuzulassen. Die Differenz zwischen Eingangs- und Ausgangsbereich ist der Störabstand.

1 Punkt Differenz Eingang/Ausgang, 1 Punkt für Störung bei Übertragung [V3F22]

b) Wodurch unterscheiden sich 'X' und 'Z' bei der Vierwertigen Logik?
'X' beschreibt einen Fehler (bspw. mehrere widersprüchliche Treiber), während 'Z' in der Regel einen bewusst erzeugten Zustand beschreibt (bspw. kein Treiber).

Je ein Punkt für Beschreibung beider Konzepte, [V6F26]

c) Wie entstehen *Glitches* in kombinatorischen Schaltungen und wann stellen diese in synchronen sequentiellen Schaltungen kein Problem dar?
Sobald sich ein Eingangssignal über mehrere Pfade auf einen Ausgang auswirkt, können unterschiedliche Leitungs- und Gatterlaufzeiten auf diesen Pfaden zu mehrfachen Umschalten des Ausgangs nach dem einmaligen Umschalten des Eingangs führen. In synchronen sequentiellen Schaltungen ist das kein Problem, wenn der Glitch weit genug vor der Setup-Zeit des nächsten Registers liegt.

1 Punkt für verschiedene Pfadlaufzeiten, 1 Punkt für Abstand zur Setup-Zeit, [V6F41]

d) Wodurch unterscheiden sich *Flip-Flops und Latches*?
Flip-Flops sind Taktflanken gesteuert, während Latches Taktphasen gesteuert sind.

Je ein Punkt für Beschreibung beider Konzepte [V7F23]

e) Wie unterscheiden sich Shifter mit konstanter Shift-Weite von *Barrel-Shiftern* bezüglich ihrer kombinatorischen Verzögerungszeit?
Ein Shift mit konstanter Weite benötigt keine Basisgatter und verursacht nur Leitungsverzögerung. Ein Barrel-Shifter besteht hingegen aus einer Kaskade von Multiplexern. Seine kombinatorische Verzögerung steigt signifikant mit der Datenbreite an.

Je 1 Punkt für keine bzw. Bitbreiten-abhängige Verzögerungszeit. [V12F32f]

Klausur zur Vorlesung Digitaltechnik

Name, Vorname: _____ Matrikelnummer: ⌴⌴⌴⌴⌴⌴⌴

Aufgabe 2 Zahlendarstellungen und binäre Subtraktion **(10 Punkte)(5 + 5)**

a) Vervollständigen Sie die folgende Tabelle *vorzeichenloser* Zahlendarstellungen. Alle Einträge einer Zeile sollen dabei den gleichen numerischen Wert repräsentieren. Verwenden Sie *möglichst kurze* Ziffernfolgen (ohne führende Nullen).

Dezimal	Binär	Hexadezimal
181_{10}	$1011\ 0101_2$	$B5_{16}$
41_{10}	$10\ 1001_2$	29_{16}
1984_{10}	$111\ 1100\ 0000_2$	$7C0_{16}$

Ein Punkt je Dezimal- und Binäreintrag. 0.5 Punkte je Hexadezimaleintrag. [V2F20, Ü2.3.1]

b) Wandeln Sie $a = 60_{10}$ und $b = -15_{10}$ in 1 Byte breite *Zweierkomplement-Zahlen* um. *Subtrahieren* Sie die Binärdarstellungen voneinander $(a - b)$. Wandeln Sie das Ergebnis ins *Dezimalformat und* ins 12 bit *Hexadezimalformat* um. Der *Lösungsweg* wird bewertet.

- Umrechnung ins Zweierkomplement:
  $60_{10} = 0011\ 1100_2$
  $-15_{10} = \overline{0000\ 1111_2} + 1 = 1111\ 0001_2$

- Subtrahend negieren:
  $15_{10} = 0000\ 1111_2$

- Addition:
  $0011\ 1100_2$
  $+\ 0000\ 1111_2$
  $=\ 0100\ 1011_2$

- Umrechnen ins Dezimalformat:
  $0100\ 1011_2 = 64_{10} + 8_{10} + 2_{10} + 1_{10} = 75_{10}$ ✓

- Umrechnen ins Hexadezimalformat nach (vorzeichenbehafteter) Bitbreitenerweiterung:
  $0100\ 1011_2 = 0000\ 0100\ 1011_2 = 04B_{16}$

Je ein Punkt pro Teilschritt. [V2F31-35, Ü2.5]

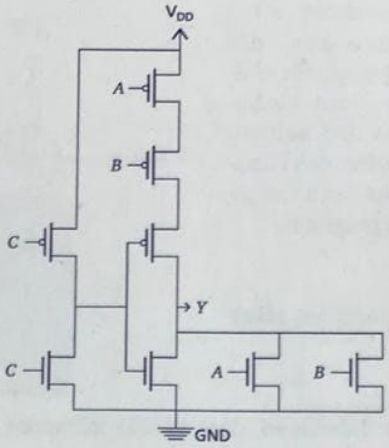
Name, Vorname: _____ Matrikelnummer: ⌊⌋⌊⌋⌊⌋⌊⌋⌊⌋⌊⌋⌊⌋

Aufgabe 3 Realisierung von logischen Funktionen (15 Punkte)(8 + 7)

Die folgenden Teilaufgaben hängen *nicht* voneinander ab.

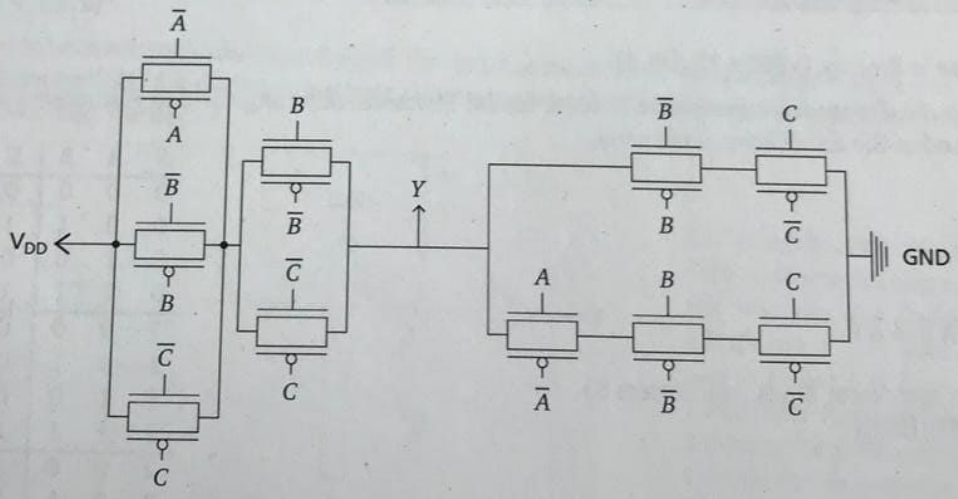
a) Zeichnen Sie eine *CMOS* Schaltung zur Realisierung von $Y = \overline{A+B}\ C$. Verwenden Sie ausschließlich *positive Eingangsliterale*.

Umwandlung nach DeMorgan und Involution liefert $Y = \overline{A}\,\overline{B}\,\overline{\overline{C}}$. C muss daher zunächst separat invertiert werden.



0.5 Punkte je Transistor,
2 Punkte für korrekte Verbindungen.
2 Punkte für Idee eines separaten Inverters.
[V3F41, V5F10, Ü3.5.2, Ü4.2]

b) Geben Sie die von folgender Schaltung realisierte boole'sche Funktion in *konjunktiver Normalform* an.



Aus der Schaltung kann die nebenstehende Wahrheitswertetabelle abgelesen werden. Daraus kann die KNF $Y = (A+B+\overline{C})\,(\overline{A}+B+\overline{C})\,(\overline{A}+\overline{B}+\overline{C})$ abgelesen werden.
2 Punkte je korrektem Maxterm (Erkennen und boole'scher Ausdruck)
1 Punkte für korrekte KNF
[V3F45, Ü4.3]

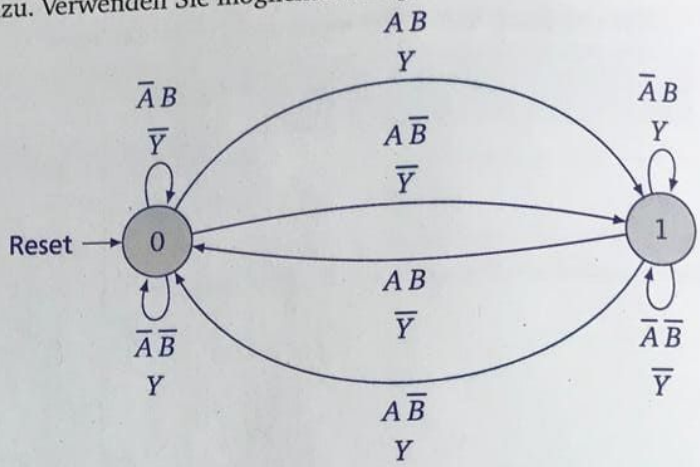
A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Klausur zur Vorlesung Digitaltechnik

Name, Vorname: _____ Matrikelnummer: ⌊⌋⌊⌋⌊⌋⌊⌋⌊⌋⌊⌋⌊⌋

Aufgabe 4 Endliche Automaten (35 Punkte)(6+5+3+7+6+8)

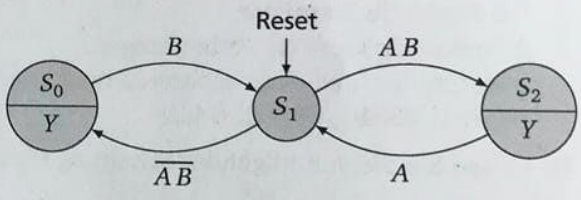
a) Zeichnen Sie das Diagramm eines *Mealy-Automaten* mit zwei Eingängen A und B und einem Ausgang Y, welcher genau dann $Y = 1$ ausgibt, wenn B der *geraden Parität* der seit dem Reset gelesenen Bitfolge von A entspricht. Das aktuell gelesene A zählt bereits zu dieser Bitfolge hinzu. Verwenden Sie möglichst wenige Zustände.



Zwei Zustände repräsentieren die bisherige Parität der A Bitfolge. Für $A = 1$ muss daher der Zustand gewächselt werden. Wegen gerader Parität ist der Startzustand 0. Für einen Mealy-Automaten steht die Ausgabe an den Kanten. Für jeden Zustandswechsel sind dabei zwei Kanten notwendig. $Y = 1$ wird genau dann ausgegeben, wenn B dem Zielzustand entspricht.
0.5 Punkt je Zustand
1 Punkt für Reset
0.5 Punkte je Kannte (Bedingung und Ausgabe)
[V3F11+15, V8F8, Ü8.5]

b) Vervollständigen Sie die folgende Tabelle so, dass sie das taktweise Ein-/Ausgabeverhalten des nebenstehenden Automaten mit *asynchronem Reset* wiedergibt.

Reset	1	1	0	0	0	0	0	0	1	0
A	1	0	0	0	0	1	0	0	0	0
B	1	1	1	0	1	1	1	0	1	0
Y	0	0	0	1	1	0	1	1	0	0



0.5 Punkte je Eintrag [V8F8+35, Ü8.4]

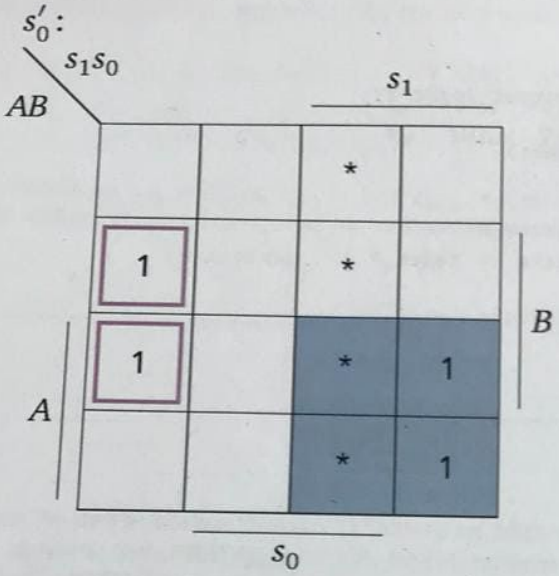
c) Geben Sie die *Zustandsübergangstabelle* ohne binäre Zustandskodierung für den Automaten aus Aufgabenteil b) an. Verwenden Sie dabei *keine don't cares*.

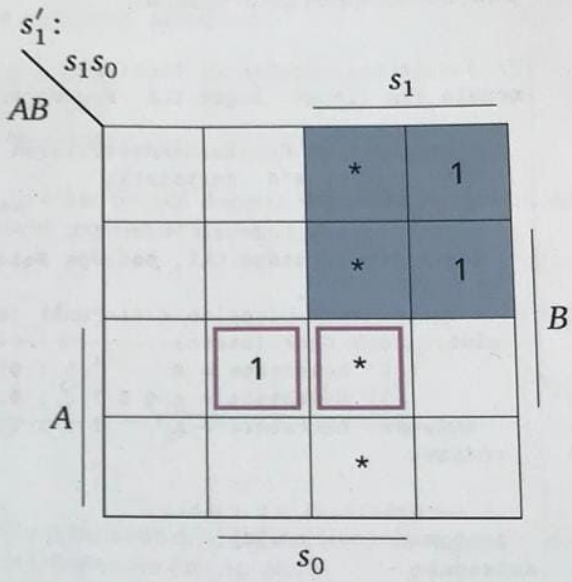
S	A	B	S'
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	2
2	0	0	2
2	0	1	2
2	1	0	1
2	1	1	1

Achtung: $\overline{A\,B} \neq \overline{A}\,\overline{B}$

1 Punkt je korrektem Block (mit festem S)
[V8F14+32, Ü8.5]

d) Geben Sie die Zustandsübergangsfunktionen für den Automaten aus Aufgabenteil b) mit kanonischer Zustandskodierung ($S_0 = 00, S_1 = 01, S_2 = 10$) an. Verwenden Sie dafür die folgenden Karnaugh-Diagramme.



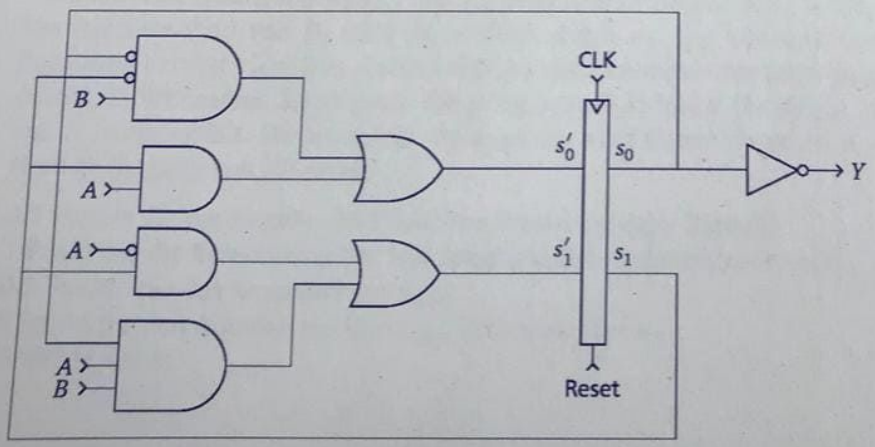


$$s_0' = s_1 A + \overline{s_0}\,\overline{s_1}\,B$$

$$s_1' = s_1 \overline{A} + s_0 A B$$

1 Punkt für Einsen je Karnaugh-Diagramm,
1 Punkt für Don't Cares in Karnaugh-Diagrammen (ungenutzte Zustandskodierung $s_1 = s_0 = 1$)
1 Punkt je Primimplikant (Markierung und boole'scher Ausdruck)
[V5F32+36, Ü5.4, Ü8.2]

e) Zeichnen Sie das *Schaltwerk inklusive Ausgabelogik* für den Automaten aus Aufgabenteil b) mit der Zustandskodierung aus Aufgabenteil d). Sie dürfen invertierte Gattereingänge (Inverterblasen) verwenden. Die Reset-Logik muss nicht berücksichtigt werden.



Die Ausgabelogik kann direkt aus dem Automatendiagramm abgelesen werden: Die Indizes aller Zustände mit $Y = 1$ sind gerade, also $s_0 = 0$. Daher $Y = \overline{s_0}$.
1 Punkt für Register mit CLK,
2 Punkte für je Eingabefunktion,
1 Punkt für Ausgabefunktion,
[V8F31, Ü8.2b]

f) Implementieren Sie den Automaten aus Teilaufgabe b) als *SystemVerilog* Modul namens `fsm` inklusive *Modulsch*
stelle. Kommentieren Sie Ihre Lösung.

```
// Modulschnittstelle (1 Punkt)
module fsm (input  logic CLK, Reset, A, B, output logic Y);

  // Deklaration der Zustandsvariablen (1 Punkt)
  logic [1:0] state, nextstate;

  // Zustandsspeicher (1 Punkt)
  always_ff @(posedge CLK, posedge Reset) state <= Reset ? 1 : nextstate;

  // Zustandsübergangslogik (1 Punkt je Übergang)
  always_comb case (state)
          0: nextstate = B     ? 1 : 0;
          1: nextstate = A & B ? 2 : 0;
    default: nextstate = A     ? 1 : 2;
  endcase

  // Ausgabelogik (1 Punkt)
  assign Y = ~state[0];
endmodule
```

Punkte laut Kommentare + 1 Punkt für Kommentare [V11F38, Ü11.3]

Name, Vorname: _____ Matrikelnummer: ⌊⌋⌊⌋⌊⌋⌊⌋⌊⌋⌊⌋⌊⌋

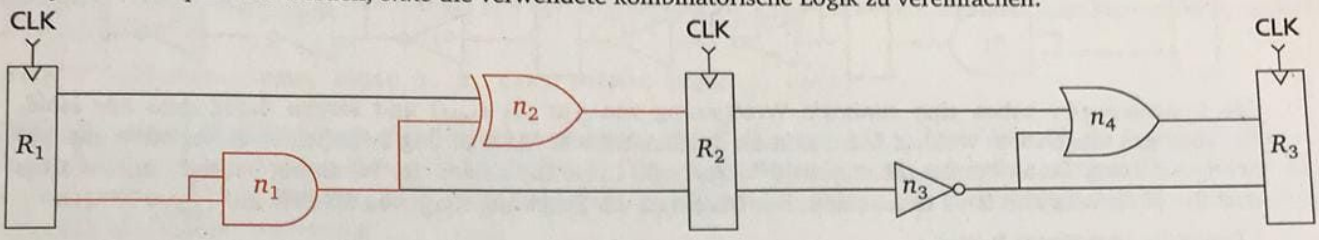
Aufgabe 5 Timing Analyse

(10 Punkte)(3 + 3 + 4)

Für diese Aufgabe werden *ausschließlich* die wie folgt spezifizierten Logikgatter verwendet:

	XOR	AND	OR	NOT
t_{cd}	8 ns	4 ns	3 ns	2 ns
t_{pd}	10 ns	7 ns	5 ns	2 ns

Darüber hinaus sei $t_{ccq} = 0{,}5\,\text{ns}$, $t_{pcq} = 1\,\text{ns}$, $t_{setup} = 2\,\text{ns}$ und $t_{hold} = 3\,\text{ns}$ für alle Register. Folgendes Schaltwerk soll analysiert und optimiert werden, *ohne* die verwendete kombinatorische Logik zu vereinfachen:



a) Geben Sie den *kritischen Pfad* der Schaltung an. Mit welcher Frequenz kann das Schaltwerk *maximal getaktet* werden, ohne die *Setup-Bedingung* von R_2 und R_3 zu verletzen? Begründen Sie Ihre Antwort.
Der kritische Pfad von R_1 nach R_2 verläuft durch n_1 und n_2. Die maximale Gesamtverzögerung der kombinatorischen Schaltung beträgt $t_{pd} = 17\,\text{ns}$. Zusammen mit der Ausgabeverzögerung t_{pcq} von R_1 und t_{setup} von R_2 ergibt dies einen kritischen Pfad der Länge 20 ns. Dies begrenzt die Taktrate auf maximal 50 MHz. Alle Pfade zwischen R_2 und R_3 sind kürzer.

0.5 Punkte für die Angabe oder Markierung des kritischen Pfades (welche Gatter),
1 Punkt für die Berechnung der minimalen Taktperiode,
0.5 Punkte für die Umrechnung in die maximale Taktrate,
1 Punkt für den Ausschluss eines kritischen Pfades von R_2 nach R_3
[V9F21+32, Ü9.4, Ü9.5]

b) Wird die *Hold-Bedingung* von R_2 und R_3 eingehalten? Begründen Sie Ihre Antwort.
Der kürzeste Pfad von R_2 nach R_3 verläuft durch n_3. Die minimale Gesamtverzögerung der kombinatorischen Schaltung beträgt $t_{cd} = 2\,\text{ns}$. Zusammen mit der minimale Ausgabeverzögerung t_{ccq} von R_2 erreicht eine Signaländerung R_3 frühestens 2,5 ns nach der steigenden Taktflanke. Da R_3 ein größeres t_{hold} hat, ist die Hold-Bedingung von R_3 nicht erfüllt. Die minimale Verzögerung aller Gatter zwischen R_1 und R_2 ist größer als t_{hold}. Daher ist die Hold-Bedingung von $R2$ erfüllt.

0.5 Punkte für die Angabe des kürzesten Pfades (welche Gatter),
1 Punkt für die Berechnung der frühestmöglichen Änderungszeit bei R_3,
0.5 Punkte für den Vergleich mit t_{hold},
1 Punkt für den Ausschluss einer t_{hold} Verletzung bei R_2
[V9F21f, Ü9.4]

Name, Vorname: _____

Matrikelnummer: ⌴⌴⌴⌴⌴⌴⌴

Aufgabe 6 Sequentielle Addierer

(10 Punkte)(10)

Ein *sequentieller Addierer* berechnet die Summe zweier Zahlen mit einer sequentiellen statt einer kombinatorischen Schaltung, wobei in jedem Takt ein Ergebnis-Bit generiert wird. Dazu sind neben den arithmetischen Ein- und Ausgaben weitere Steuersignale notwendig:

- `CLK` - das Taktsignal
- `START` - wird für einen Takt auf 1 gesetzt, wenn eine neue Berechnung starten soll
- `DONE` - wird für einen Takt auf 1 gesetzt, wenn eine Berechnung fertig ist.

Ein solcher Addierer benötigt unabhängig von der Bitbreite der Eingänge nur einen einzigen Volladdierer mit folgender Modulschnittstelle:

```
module fulladder(input logic a, b, cin, output logic s, cout);
```

Der Volladdierer wird taktweise mit den entsprechenden Eingabebits beschaltet. Dazu können die Eingaben bspw. in Registern gespeichert werden, deren Inhalte in jedem Takt um ein Bit nach rechts geschoben werden. Die Summen-Bits können auch in einem Shift-Register gesammelt werden. Für das Übertragen des carry-Bits in den nächsten Takt ist ebenfalls ein Register notwendig.

Implementieren Sie einen sequentiellen Addierer mit folgender Schnittstelle in *SystemVerilog*. *Kommentieren* Sie Ihre Lösung. Beachten Sie, dass die *Summe ein Bit breiter* als die Summanden sind.

```
module seqadd
 #(parameter    WIDTH = 8)        // Bitbreite der Eingaben
  (input  logic CLK,              // Takt
                START,            // Eingaben liegen an
   input  logic [WIDTH-1:0] A, B, // zu addierende Eingaben
   output logic [WIDTH:0]   S,    // Summe der Eingaben
   output logic DONE);            // Berechnung fertig

  // Zähler für erzeugte Ausgabebits (2 Punkte)
  logic[$clog2(WIDTH+2)-1:0] cnt;
  always_ff @(posedge CLK) cnt <= START ? WIDTH+1 : cnt > 0 ? cnt - 1 : cnt;

  // Eingabe shift register: LSBs werden verrechnet (2 Punkte)
  logic [WIDTH-1:0] as, bs;
  always_ff @(posedge CLK) as <= START ? A : as >> 1;
  always_ff @(posedge CLK) bs <= START ? B : bs >> 1;

  // Volladdierer Einbinden (2 Punkte)
  logic cin, cout, sum;
  fulladder inst (as[0], bs[0], cin, sum, cout);

  // Carry übertragen (1 Punkt)
  always_ff @(posedge CLK) cin <= START ? 1'b0 : cout;

  // Volladdierer Ausgaben speichern (1 Punkt)
  always_ff @(posedge CLK) S   <= {sum, S[WIDTH:1]};

  // Abschluss der Berechnung signalisieren (1 Punkt)
  always_ff @(posedge CLK) DONE <= cnt == 1 ? 1 : 0;
endmodule
```

Punkte laut Kommentare + 1 Punkt für Kommentare [Transferaufgabe aus V10-V12]