1. EINLEITUNG

1.1 Lernziele

Dieses Labor verfolgt folgende Ziele:

1. Sie sollen den im Laufe des Semesters behandelten Stoff der Lehrveranstaltungen "Digitaltechnik" anwenden und vertiefen und dadurch ein besseres Verständnis für digitale Schaltungen erlangen.
2. Sie sollen Verfahren zur Schaltungsanalyse durch Simulation erlernen.
3. Sie sollen moderne Electronic-Design-Methoden für das Entwerfen, Editieren und Simulieren von digitalen elektronischen Schaltungen auf dem Rechner kennen lernen und anwenden.
4. Sie sollen lernen, sich in ein komplexes Programmsystem einzuarbeiten und die dazu notwendige Information aus (Online-)Handbüchern und anderer Literatur zu beschaffen.

Um diese Ziele zu erreichen, werden Simulationen auf Rechnern mit Hilfe einer Entwicklungssoftware durchgeführt.

Als Entwicklungssoftware wird Quartus Prime von Intel gewählt, weil ...

- es sich um eine industrieorientierte Entwicklungssoftware handelt,
- es die Möglichkeit der hardwareorientierte Umsetzung der Simulationen durch die Programmierung von Logikbausteinen ermöglicht,
- eine kostenlose Version (20.1) unter https://fpgasoftware.intel.com/?edition=lite verfügbar ist, die alle wichtigen Funktionen und Schaltelemente beinhaltet und Ihnen auch noch nach Abschluss dieses Labors zur Verfügung steht.

1.2 Intel Quartus Prime Lite

Detaillierte Informationen zur Installation und eine Kurzanleitung finden Sie auf dem Server.

2. Versuche

2.1 Versuch 2: Kombinatorische Logik

2.1.1 Grundgatter

In dieser Übung lernen Sie die Grundgatter digitaler Schaltungen kennen, wie sie im Quartus Prime Block Diagram einzugeben sind und wie man einfach alle Kombinationen der Eingangssignale als Stimulus im Waveform Editor erzeugen und übersichtlich anzeigen kann. Da die Bibliothek der Demoversion nur die Symboldarstellungen digitaltechnischer Gatter nach US-Norm enthält, dient diese Übung auch zur Wiederholung dieser Darstellungsart.

2.1.1.1 Durchführung

1. Geben Sie die Schaltung gemäß der Abbildung 1 ein. Als Quelle für die Eingangssignale A und B ist die digitale Quelle Input aus der Bibliothek „primitives“ einzufügen. Durch Klick auf PIN_NAME können Sie den Namen verändern. Achten Sie darauf, dass Sie die Anschlüsse sinnvoll benennen. Kompilieren Sie die Schaltung, wie im Tutorial beschrieben.

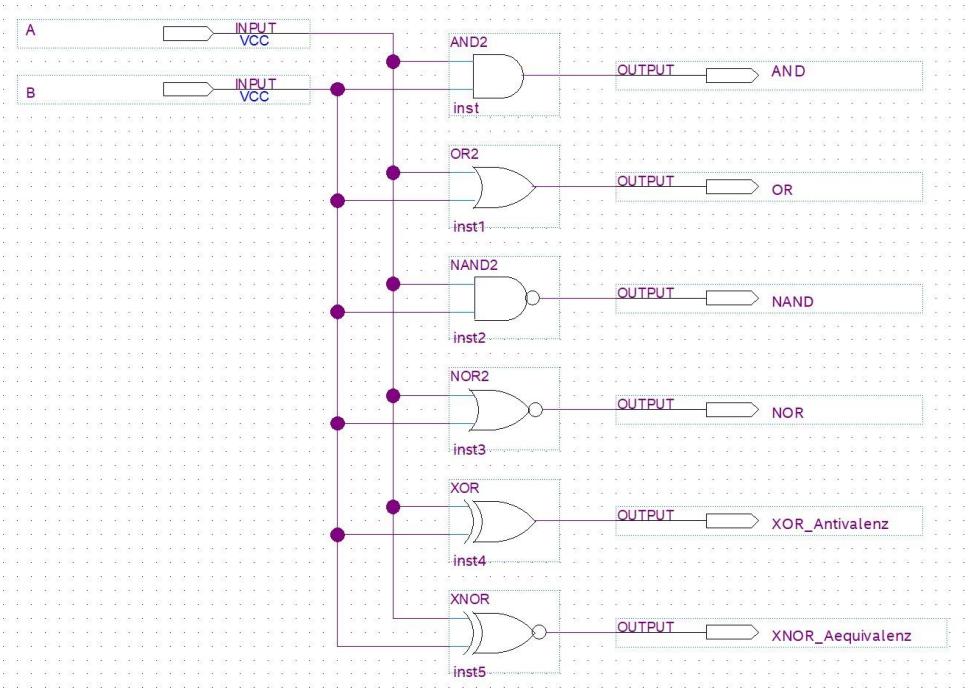


Abbildung 1: Grundgatter

2. Nach Öffnen des Waveform Editors und Erstellen der Datei stellen Sie die End Time auf 100 µs bei einer Grid Size von 10 µs ein. Fügen Sie alle Ein- und Ausgangssignale in das Fenster ein. Damit alle vier möglichen Kombinationen der beiden Eingangsvariablen A und B zyklisch durchlaufen werden, definieren Sie für A ein Clock-Signal mit Periodendauer von 20µs und für B mit einer Periodendauer von 40µs.
3. Führen Sie nach dem Speichern eine Simulation der Schaltung durch. Oftmals ist es vorteilhaft, mehrere Binärsignale als gewichtete Summe darzustellen. Dieses soll hier mit den Eingangs-signalen A und B geübt werden. Mit der Funktion Grouping (Signale selektieren, rechte Maustaste: Grouping – Group) können Sie Signale zusammenfassen und z.B. binär oder dezimal darstellen. Die Gewichtung erfolgt hierbei über die Reihenfolge von höherwertigen zu niederwertigen Bits von oben nach unten. So sind die vier möglichen Eingangskombinationen übersichtlich durch die Ziffern 0 bis 3 darstellbar.

2.1.2 Versuch 2: Entwickeln einer Digitalschaltung aus einer Wahrheitstabelle

2.1.2.1 Durchführung

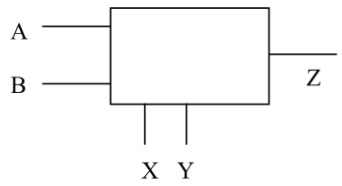
Entwickeln Sie eine Digitalschaltung, welche die rechts stehende Wahrheitstabelle auf möglichst einfache Weise realisiert

1. Vereinfachen Sie die Logikfunktion mit einem Verfahren ihrer Wahl.
2. Geben Sie in Quartus Prime die Schaltung ein. Es sollen ausschließlich NAND-Gatter verwendet werden.
3. Testen Sie die Schaltung mit einer Timing-Simulation.

D	C	B	A	Z
0	0	0	0	0
0	0	0	1	0
0	0	1	0	x
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	x
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	x
1	1	1	1	0

2.1.3 Versuch 2 (optional): Digitalschaltung für Universalgatter

Für das Universalgatter von Aufgabe 3 aus Versuchstermin 1 soll eine Schaltung in Quartus Prime entworfen werden und die Funktion der Schaltung mittels Simulation überprüft werden. Das Universalgatter mit den zwei Eingängen A und B soll in Abhängigkeit der beiden Steuereingänge X und Y die Verknüpfungsfunktionen gemäß der folgenden Tabelle in möglichst einfacher Weise realisieren:



X	Y	Z
0	0	A *NAND* B
0	1	A *NOR* B
1	0	A *EXOR* B
1	1	1

2.1.3.1 Durchführung

1. Soweit nicht schon erfolgt ermitteln Sie die einfachste logische Schaltfunktion zur Realisierung der gewünschten Verknüpfungen.
2. Geben Sie in Quartus Prime die zugehörige Schaltung ein.
3. Überprüfen Sie die Funktion der Schaltung mit einer Simulation.

2.1.4 Versuch 3: Codewandler

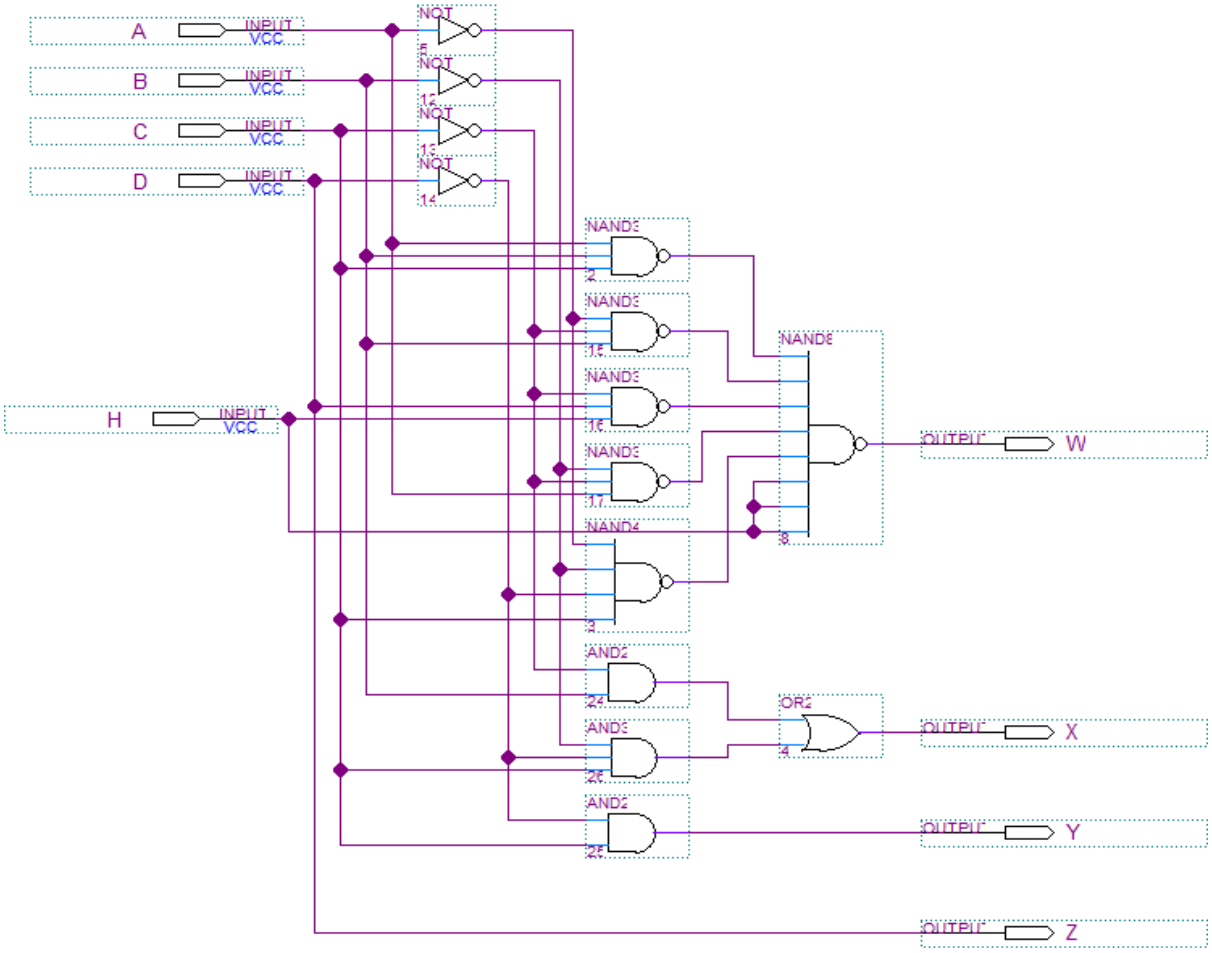


Abbildung 2: Codewandler

Die obige Schaltung zeigt einen Codewandler, der Glixon-Code in Dualcode (BCD-Code) umwandelt. Der Glixon-Code ist ein einschrittiger Code, der für die Dezimalwerte 0 bis 9 gemäß der nebenstehenden Wertetabelle definiert ist:

Glixon-Code

Dezimalwert	D	C	B	A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	0	1	0
4	0	1	1	0
5	0	1	1	1
6	0	1	0	1
7	0	1	0	0
8	1	1	0	0
9	1	0	0	0

2.1.4.1 Durchführung:

1. Öffnen Sie in Quartus Prime die Schaltung *codewandler_01.bdf* (zum Download auf dem Server bereitgestellt).
2. Überprüfen Sie mit Hilfe einer Simulation, ob die Schaltung korrekt arbeitet. Definieren Sie dazu die Eingangssignale in geeigneter Weise und testen Sie die Funktion.

2.1.4.2 Ausarbeitung

Erstellen Sie die KV-Diagramme für die Ausgangssignale und ermitteln Sie daraus die Booleschen Funktionen in einfachster Form. Vergleichen Sie das Ergebnis mit der Schaltung.

2.1.5 Versuch 3: Addier- und Subtrahierwerk

In dieser Aufgabe soll ein hierarchisches Design realisiert werden. Dabei ist zuerst ein Halbaddierer einzugeben. Aus zwei Halbaddierern wird dann ein 1-Bit-Volladdierer zusammengesetzt. Aus vier Volladdierern wird schließlich ein 4-Bit-Addierwerk gebildet und dieses noch zu einem umschaltbaren 4-Bit-Subtrahierwerk erweitert.

2.1.5.1 Durchführung:

1. Geben Sie in Quartus Prime einen Halbaddierer aus NAND-Gattern (**nand2**) mit den Eingangssignalen A und B und den Ausgangssignalen S und Ue ein. Und kompilieren Sie die Schaltung.
2. Legen Sie an A und B Signale an und überprüfen Sie mit einer Simulation die Funktion, indem Sie durch geeignete Eingangssignaldefinition alle Eingangskombinationen durchfahren. Anzeige: A, B, S, Ue, {Ue S}.
3. Erstellen Sie mittels „**Create/ Update → Create Symbol Files for Current File“** unter dem Menüpunkt „File“ ein Symbol Halbaddierer mit den Eingängen A und B und den Ausgängen S und UE.
4. Nach erfolgreichem Test erstellen Sie eine neue Schaltung Volladdierer. Benutzen Sie hierzu den erstellten Halbaddierer. Diesen finden Sie über die Schaltfläche Symbol Tool unter der aktuellen Project Library. Setzen Sie „Set as Top-Level Entity“ auf den Volladdierer.
5. Über „Hierarchy“ können Sie innerhalb Ihres Designs navigieren.
6. Testen Sie nun die Schaltung ähnlich wie in Punkt 2.
7. Anschließend wird die Schaltung analog zu Punkt 3 in ein Modul umgeformt.
8. Nun sind vier Volladdierer analog zu Punkt 4 zu einem Vier-Bit-Addierwerk zusammenzusetzen und exemplarisch zu testen.
9. Simulieren Sie im „Run Functional Simulation“ und „ Run Timing Simulation“ Modus und interpretieren Sie die Ergebnisse.
10. Erweitern Sie das 4-Bit-Addierwerk aus Punkt 8 zu einem umschaltbaren 4-Bit-Addier- und Subtrahierwerk, indem Sie durch Hinzufügen von EXOR-Gattern und je einem Steuersignal pro 4-Bit Eingangszahl eine Inversion der 4-Bit-Eingangszahlen A[3..0] oder B[3..0] erzeugen und zusammen mit dem Carry-In Signal schließlich auch eine 4-Bit Subtraktion bewerkstelligen.
11. Testen Sie abschließend auch die 4-Bit Subtraktion exemplarisch.

2.2 Versuch 4: Sequentielle Logik

2.2.1 Versuch 4: RS-Flipflop aus Grundgattern

1. Realisieren Sie ein RS-Flipflop mit NOR-Gattern
2. Testen Sie das Flipflop mit einer Simulation, indem Sie die Eingangssignale so definieren, dass kein verbotener Zustand anliegt.
3. Definieren Sie nun die Eingangssignale so, dass auch der verbotene Zustand vorkommt und beobachten Sie, welches Ergebnis die Simulation liefert. Simulieren Sie sowohl im Modus „Run Functional Simulation“ als auch im „Run Timing Simulation“ Modus.
4. Vermeiden Sie den verbotenen Zustand, indem Sie die Schaltung so abändern, dass sie dominant setzend wirkt.

2.2.2 Versuch 4: D-Flipflop aus Grundgattern

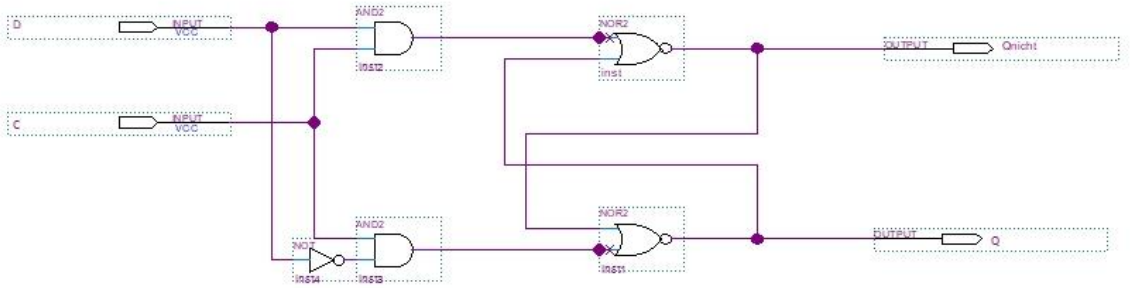


Abbildung 3: D-Flipflop

1. Geben Sie die Schaltung eines zustandsgesteuerten D-Flipflops gemäß Abbildung 3 ein.
2. Testen Sie das Flipflop mit einer Simulation (Grid Size 1µs; End Time 100µs). Vorschlag für die Eingangssignale:
   C: Clock Period 10 µs
   D: Clock Period 12 µs
3. Erklären Sie die Arbeitsweise des zustandsgesteuerten D-Flipflops anhand der Simulations-ergebnisse aus Punkt 2.

2.2.3 Versuch 4: JK-MS-Flipflop aus Grundgattern

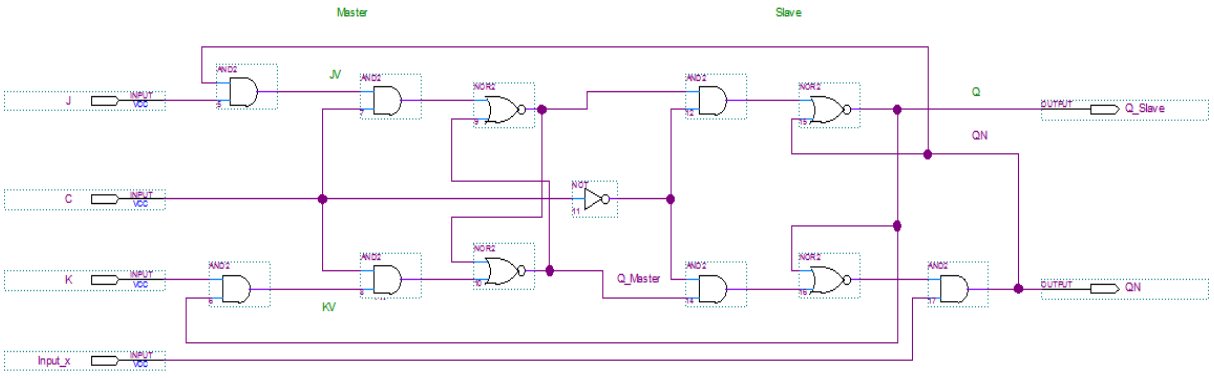


Abbildung 4: JK-MS-Flipflop

1. Geben Sie in Quartus Prime ein taktzustandsgesteuertes JK-Master-Slave-Flipflop gemäß Abbildung 4 ein. Es soll nur aus den logischen Grundgattern NOR, AND und NOT aufgebaut werden.
2. Wozu dient das AND2-Gatter vor dem QN-Ausgang mit dem Eingangssignal Input_x?
3. Testen Sie die Schaltung mit einer Simulation (Grid Size 1µs; End Time 100µs). Stellen Sie dazu die Signale J, K, C, die Vorbereitungssignale JV und KV, sowie die FF-Zustände Q_Master und Q_Slave im Waveform Editor dar. Vorschlag für die Eingangssignale:
   J: Clock Period 10 µs
   K: Clock Period 7 µs
   C: Clock Period 3 µs
   Input_x: Clock Period 31 µs.
4. Erklären Sie die Arbeitsweise des JK-Master-Slave-Flipflops anhand der Simulationsergebnisse aus Punkt 3.

2.2.4 Versuch 5: Schaltungsanalyse

Auf dem Server befindet sich die Schaltung *analyse_stud_01.bdf*. Kopieren Sie diese Datei in Ihr Projektverzeichnis auf der Harddisk und entfernen Sie gegebenenfalls den Schreibschutz.

2.2.4.1 Durchführung

Analysieren Sie die Schaltung *analyse_stud_01.bdf* und beantworten Sie folgende Fragen:

1. Welche Funktion erfüllt die Schaltung?
2. Welches ist das Ausgangssignal?
3. Nach welchem Prinzip arbeitet die Schaltung?

2.2.5 Versuch 5: Asynchronzähler

2.2.5.1 Durchführung

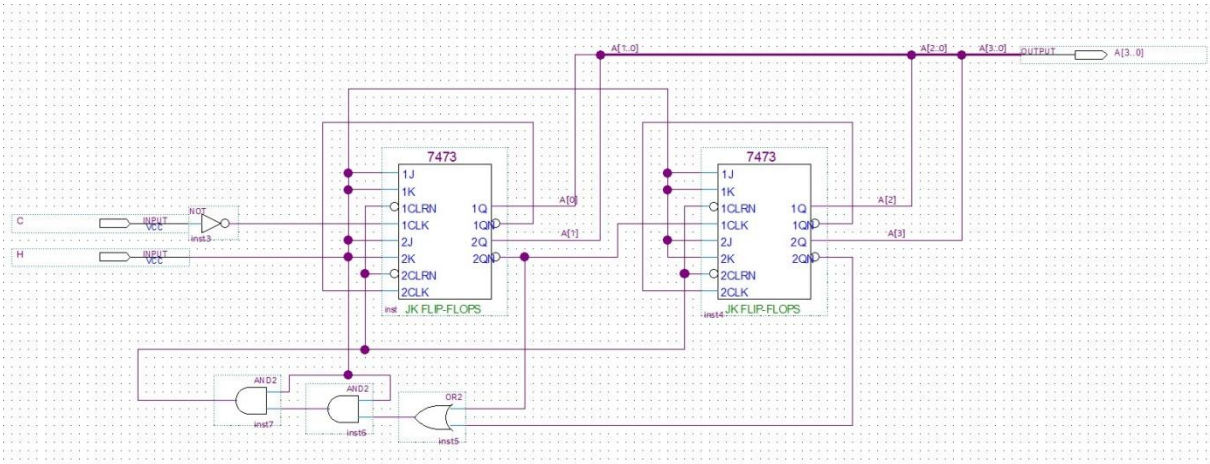


Abbildung 5: Asynchronzähler

1. Nach welchem Prinzip arbeitet die oben abgebildete Zählerschaltung?
2. Geben Sie die Schaltung des asynchronen Dezimalzählers in Quartus Prime ein. Nutzen Sie hierzu das Bauteil 7473 aus der Bibliothek „megafunctions“ und beachten Sie zur Erzeugung der Busverdrahtung die Anleitung im Quartus Prime Tutorial.
3. Führen Sie eine Simulation über 3 µs mit einer Periodendauer des Taktsignals C von 200 ns durch.
4. Analysieren Sie das Verhalten bei verschiedenen Taktfrequenzen. Setzen Sie dazu Periodendauer der Taktsignals C auf die Werte a) 60 ns , b) 30 ns c) 20 ns und wiederholen Sie jeweils die Simulation mit entsprechend angepassten Werten für die End Time.
5. Wozu dienen die AND-Gatter 6 und 7 im unteren Teil der Schaltung?

2.2.5.2 Auswertung

1. Ermitteln Sie für den Fall b) Periodendauer 30 ns, wann der Zählerausgang gültige Werte liefert.
2. Welche maximale Zählfrequenz ist etwa möglich?

2.2.6 Versuch 5: Synchronzähler

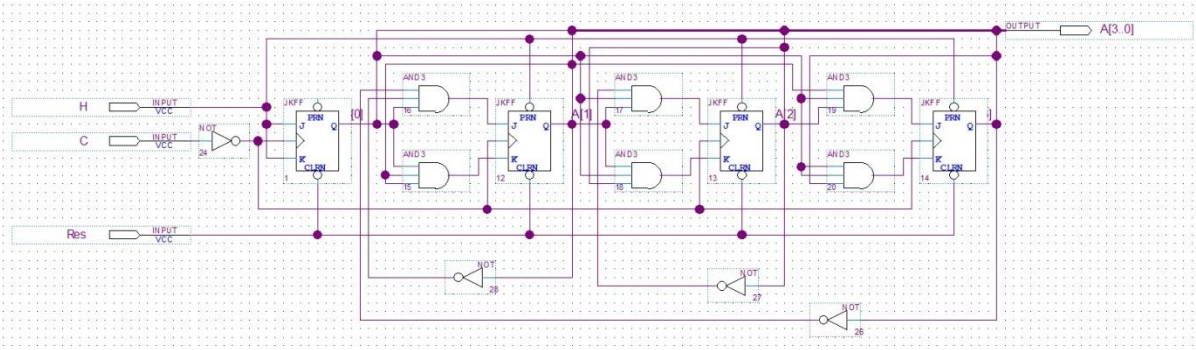


Abbildung 6: Synchronzähler

2.2.6.1 Durchführung

1. Laden Sie die Schaltung des obigen Synchronzählers in Quartus Prime. Die Datei heißt *szaehler_01.bdf* und befindet sich auf dem Server.
2. Führen Sie eine Simulation über 3 µs mit einer Periodendauer von 200 ns durch.
3. Analysieren Sie das Verhalten bei verschiedenen Taktfrequenzen. Setzen Sie dazu die Periodendauer auf die Werte a) 60 ns, b) 20 ns und c) 10 ns und wiederholen Sie jeweils die Simulation mit entsprechend angepassten Werten für die End Time.

2.2.6.2 Ausarbeitung

Zeigen Sie anhand der Simulation, warum mit Synchronzählern höhere Zählraten als mit Asynchronzählern möglich sind.

3. Anhang

3.1 Literatur

Intel Corporation (2019). Quartus® Prime Introduction Using Schematic Designs**.** Abgerufen 31.März.2021, von https://ftp.intel.com/Public/Pub/fpgaup/pub/Teaching_Materials/current/Tutorials/Schematic/Quartus_II_Introduction.pdf

Bauer, H.-P.
Digitaltechnik, Vorlesungsskript WS 2019 für Bac EIT.