Digitalelektronisches Praktikum
Versuch 6

Moritz Breipohl
mbreipohl@techfak.uni-bielefeld.de

Markus Rothgänger
mrothgaenger@techfak.uni-bielefeld.de

Gruppe 5

Tutor: Lukas Schmidt, Robin Ewers

5. Juli 2018

Theorie/Allgemeines[2]

Was ist ein FPGA

FPGA steht für "Field Programmable Gate Array", also ein Chip, auf welchem mehrere Logikblöcke liegen, welche verschiedene Funktionalitäten wie zum Beispiel AND oder OR Gattern anbieten. Das besondere ist jedoch, dass diese zunächst nicht miteinander verbunden sind. Mithilfe einer HDL (Hardware Description Language) kann man beschreiben, wie diese Bauteile miteinander verbunden sein sollen, um bestimmte Schaltungen zu realisieren. Das heißt, dass im Gegensatz zu einer CPU nicht auszuführende Instruktionen programmiert werden, sondern die eigentliche Schaltung und Verbindung auf kleinstem Raum.
Dies wird häufig für das Prototyping von Schaltungen und neuen Chips eingesetzt, aber auch für hochspezialisierte Schaltungen wie Netzwerkschnittstellen. Ein anderes Anwendungsgebiet ist die Produktion in sehr kleiner Serie, die zwar einen bestimmte Chip benötigen, bei denen es sich aber nicht lohnt, extra einen speziellen Chip zu designen, zu testen und zu bestellen.

Wie arbeitet man mit einem FPGA?

Es wurde mit der Software Vivado gearbeitet, einem Tool, mit dem man grafisch das logische Layout des Boards festlegen kann. Da es bereits vordefinierte Blöcke gibt, kann man diese per Drag&Drop platzieren und deren Ein- und Ausgänge mit anderen Blöcken verbinden. Es gibt auch Komponenten wie Hardware- Ein- und Ausgänge, so dass der Chip auch mit anderen Bauteilen kommunizieren kann. Wenn das Design fertiggestellt ist, muss dieses kompiliert werden. Der Computer überprüft zunächst, ob die Verbindungen, die angegeben wurden gültig sind (z.B. dass nicht mehrere Ausgänge an einen Eingang angeschlossen werden, ohne einen Multiplexer davorzuschalten). Im weiteren Verlauf versucht der Computer, die optimale Verbindung der Logikblöcke auf dem Chip zu ermitteln, welche dann die gewünschte Funktionalität liefern. Das Resultat ist ein sogenannter Bit-Stream. Dieser kann auf das FPGA geladen werden und die Funktion ist praktisch gegeben.

Versuchsaufbau

Die allgemeine Benutzung der Software Vivado wird hier nicht erläutert.

Aufgabe

Es wurden insgesamt sechs Aufgaben bearbeitet. Alle arbeiteten mit den Komponenten auf dem Board.

- 1. In der ersten Aufgabe sollten sechs Segmente einer Sieben-Segment-Anzeige mit Hilfe von Vier Schaltern angesteuert werden. Dabei sollte der angezeigte Wert der Anzeige dem Binär kodierten Wert der Schalter entsprechen. Der Block zum Ansteuern der Sechs Segmente war vorgegeben. Die Funktionalität der Implementierung sollte (durch die Simulation mit der *Testbench* und auf dem Board) verifiziert und die Aufgabe der *Testbench* erklärt werden. Des weiteren sollte herausgearbeitet werden, welches Segment nicht angesteuert wird, sowie, welche Aufgabe das Ausgangssignal mit dem Code AN besitzt.

- 2. Im zweiten Teil war die Schaltung aus der ersten Aufgabe so zu erweitern, so dass auch das siebte Segment angesteuert wird. Dazu sollte eine aufgestellte Schaltlogik mithilfe des Karnaugh-Plans minimiert werden. Diese minimierte Schaltung wurde in das Block

Design eingebaut. Auch dieses Block Design wurde durch die Simulation sowie am Board verifiziert.

- 3. Hier wurde das Erstellen eines *IP-Core* (aus Zeitmangel) übersprungen. Es war herauszustellen, was bei dem Ansteuern aller vier Sieben-Segment-Anzeigen zu beachten ist. Diese Schaltung war zu skizzieren, zu implementieren und zu testen.
- 4. In der vierten Aufgabe sollten die mit den Schaltern eingestellten Werte erst bei einem Knopfdruck auf die Anzeige übernommen werden. Dazu waren alle möglichen Einbaustellen für Speicherelemente aufzuzählen und gegeneinander abzuwägen. Die Schaltung war zu implementieren und zu testen.
- 5. Im fünften Teil war das Zurücksetzen der Anzeige auf den Wert *0000* bei Knopfdruck (eines weiteren Knopfes) zu implementieren. Es sollte erklärt werden, wieso sowohl der Knopf zum Zurücksetzen, als auch der Knopf zum Speichern zu drücken war, um die Anzeige zurückzusetzen. Eine Lösung dieses Problems war zu finden.
- *6. übersprungen*
- 7. Ein 16-Bit Zähler sollte erstellt werden, welcher bei einem Knopfdruck auf Null gesetzt werden konnte. Die Schaltung war zuerst mit zwei 8-Bit Addern zu skizzieren, dann zu realisieren und schließlich zu testen.

Aufbau und Erläuterung

Verwendete Bauteile

Basys 3 FPGA-Board, USB-Kabel, Computer mit Vivado Software.

Block Design Bauteile

Allgemein sind alle Eingänge linksbündig und alle Ausgänge rechtsbündig. Hier eine Liste der Bauteile mit einer kurzen Erklärung der Funktionsweise

- *Constant:* Konstanter Wert, dessen Bitweite eingestellt werden kann.
- *Slice:* Herausschneiden von bestimmten Bits des Eingangsbitstroms.
- *Concat:* Hintereinanderfügen von zwei Bitströmen variabler Breite zu einem einzigen Bitstrom, dessen Breite der Summe der Breiten der Eingangsbitströme entspricht.
- *and_4:* AND-Gatter mit vier Eingängen.
- *and_3:* AND-Gatter mit drei Eingängen.
- *or_3:* OR-Gatter mit drei Eingängen.
- *Utility Vector Logic:* Mehrzweck-Baustein; wurde in den Designs nur als Negator benutzt.

Sechs-Segment-Anzeige

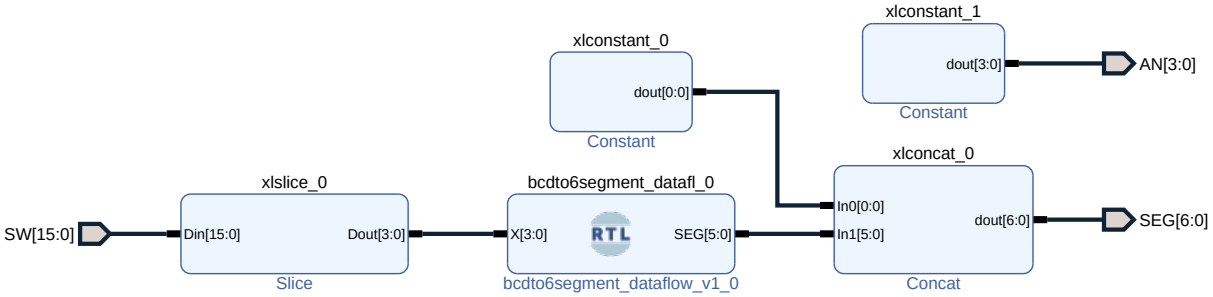

Abbildung 1: Block Design zum Ansteuern von Sechs der Sieben Segmente einer Anzeige

Das Blockdesign zu dem ersten Aufbau war im wesentlichen vorgegeben. Wie in Abbildung 1 zu sehen, wurde zuerst der Bitstrom aus dem Eingang der Schalter mit Hilfe des *slice* Bausteins auf die unteren vier Bit verkleinert. Diese dienen dem aus den Aufgabendaten entnommenen Modul zum ansteuern der sechs Segmente als Eingang (in der Abbildung mit *bcdto6segment_dataflow* bezeichnet). Der Ausgang des Moduls wurde mit einer ein Bit Konstante mit dem Wert 0 verknüpft und der entstehende sieben Bit breite Strom wurde an den Ausgang *SEG* geleitet. Dieser steuert die Segmente direkt an. Zusätzlich sollte eine vier Bit breite Konstante mit dem Wert 1 an den Ausgang *AN* angelegt werden. Die Bedeutung dieses Wertes wird im folgenden Erläutert.

Test und Auswertung der Waveforms

Bei einer Simulation mit der initialen Konfiguration stieß die Testbench auf einen Fehler. Dieser besagte, dass der Wert an der Anode falsch gesetzt sei. Ein Blick in das Handbuch[1] des Boards klärte auf, dass die Anode den an dem Ausgang *SEG* anliegenden Wert auf die Ziffern der Anzeige weitergibt, für die das Bit an der Anode Null ist. Ein Binärcode von 1110 bedeutet also, dass die letzte Ziffer aktiv ist, die anderen nicht. Wurde der Wert der Konstante am Ausgang *AN* also auf 14 (Wert des Binärcodes 1110) gesetzt, so lief der Test ohne Fehler. Hieraus wird deutlich, dass die Testbench die Funktionsweise und die Vorraussetzungen für eine spezifische Schaltung verifiziert, ähnlich zu einem Unit-Test.

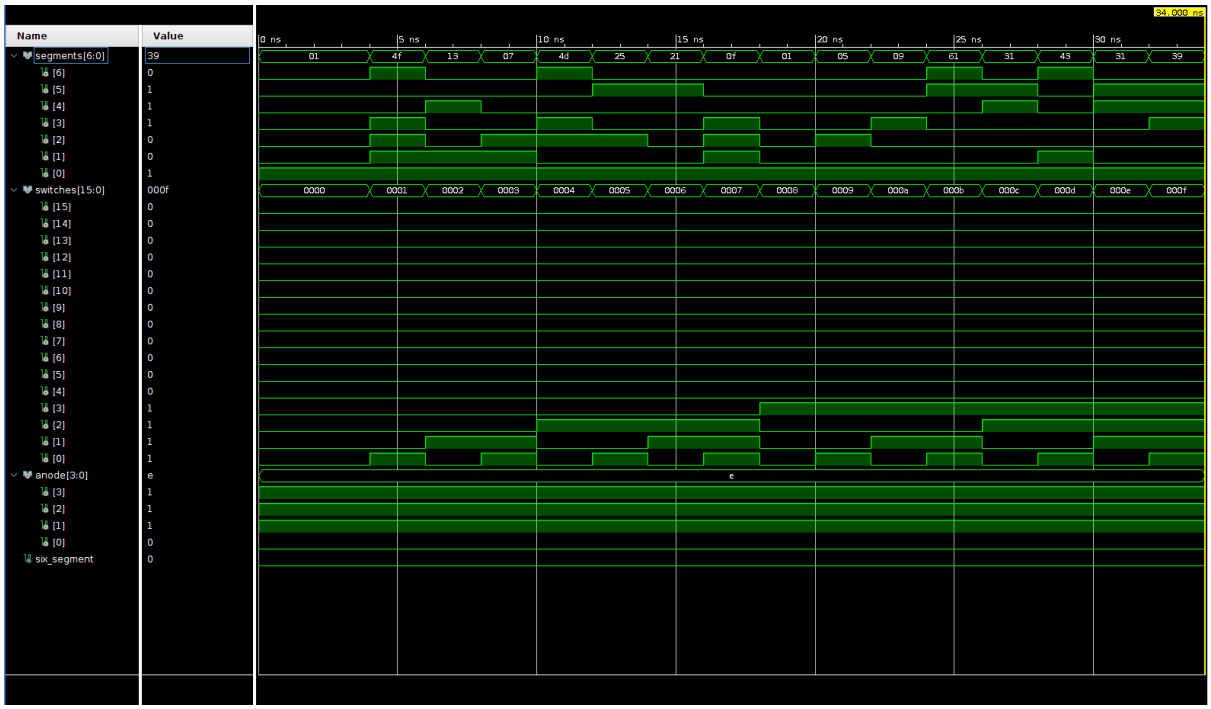

Abbildung 2: Screenshot des Waveforms der Testbench zur Sechs-Segment Anzeige

Die Waveform-Anzeige in der Software bietet die Möglichkeit, die Auswirkung von in der Testbench konfigurierten Eingangsbelegungen auf die Ausgänge zu analysieren. Somit kann das vollständige Verhalten der Schaltung verifiziert werden. Für diesen Versuch war also in der Testbench ein Hochzählen mithilfe der letzten vier Bits des Eingangs der Schalter konfiguriert. Konkret wurde für diesen Versuch aus den Waveforms deutlich, dass das Element, welches von dem untersten Bit des Ausgangs angesteuert wird, nicht geschaltet wird. Dieses Bit hat immer den Wert 1 (durch die Konstante im Block Design), somit leuchtet die Anzeige nicht. In dem praktischen Test wurde deutlich, dass es sich um das mittlere Segment handelt. Beispielsweise ist in Abbildung 2 zu erkennen, dass eine Null am Eingang (also der erste Abschnitt im Diagramm) in einer Ausgangsbelegung resultiert, indem alle bis auf das mittlere Element Null sind. Es wird also eine 0 als Ziffer dargestellt.

Sieben-Segment-Anzeige

Zuerst wurde eine Logiktabelle (Tabelle 1) angelegt in welcher nur das siebte Element betrachtet wurde. Alle Eingangsbelegungen der vier Eingangsbits wurden hier berücksichtigt. Ein Übertrag in einen Karnaugh-Plan (Tabelle 2) und das Erstellen einer Disjunktiven Normalform durch Umkreisen der Einsen führt zu folgender Schaltformel:

$$S_0 = \overline{x_3}\,\overline{x_2}\,\overline{x_1} \vee \overline{x_3}\,x_2\,x_1\,x_0 \vee x_3\,x_2\,\overline{x_1}\,x_0$$

x_3	x_2	x_1	x_0	Output
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Tabelle 1: Logiktabelle zum mittleren Segment

$x_3x_2x_1x_0$	00	01	11	10
00	1	1	0	0
01	0	0	1	0
11	1	0	0	0
10	0	0	0	0

Tabelle 2: Karnaugh-Plan zur Schaltung des mittleren Segments

Die Schaltfunktion wurde also mit AND- und OR-Gattern in das Block-Design eingefügt. Dabei wurden zuerst die unteren vier Bits des Eingangs isoliert und negiert. Diese Bits dienten dann als Eingang für die Gatter. Schließlich wurde der Ausgang der Schaltung mit dem Ausgang des Steuerungsmoduls der sechs Segmente konkateniert. Daraus resultierte das Design aus Abbildung 3.

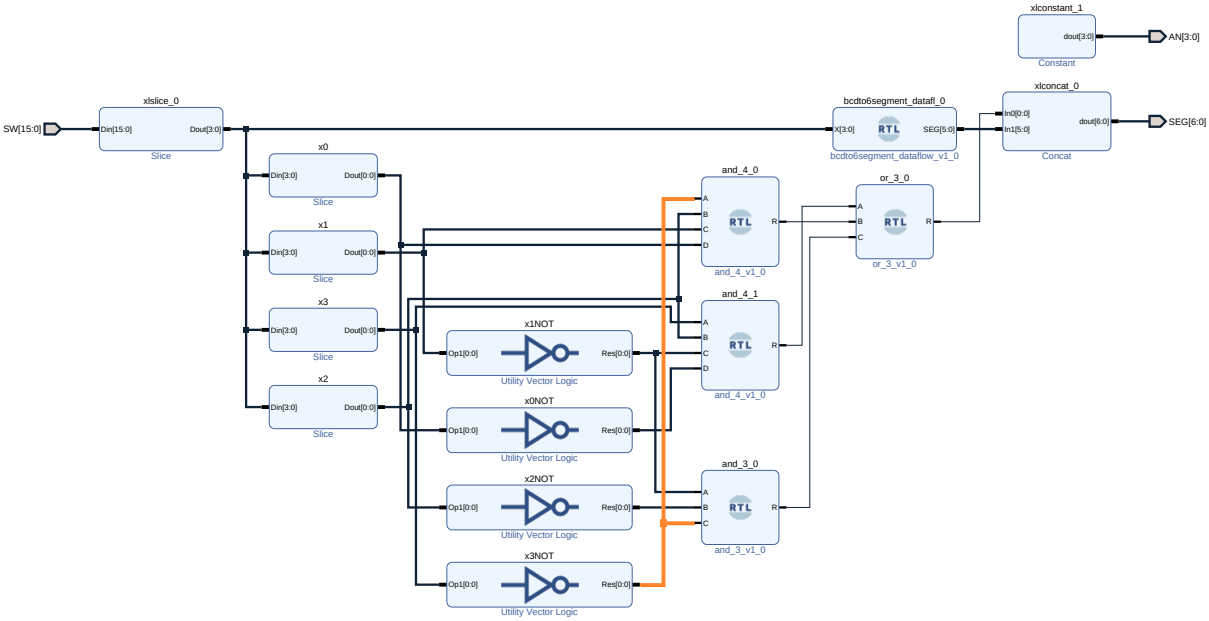

Abbildung 3: Block Design zum Ansteuern von der vollen Sieben-Segment-Anzeige

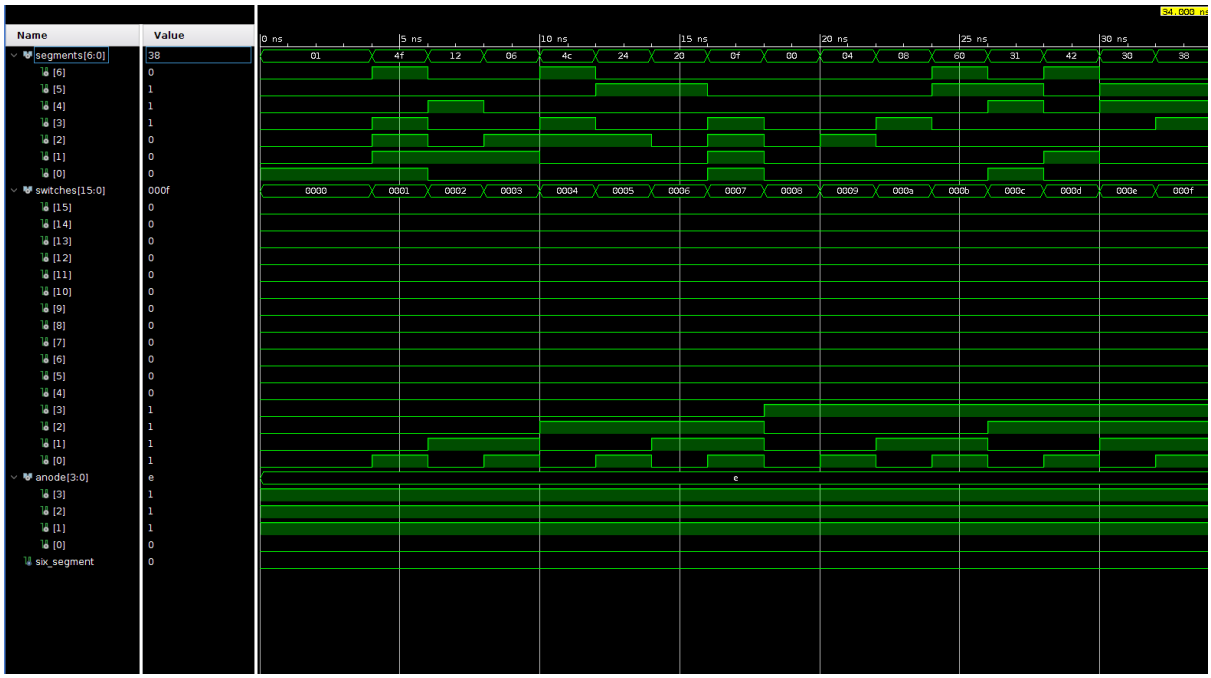

Abbildung 4: Screenshot des Waveforms der Testbench zur Sieben-Segment Anzeige

Die Testbench lief ohne Fehler und die resultierenden Waveforms (Abbildung 3) zeigten, dass nun auch das Signal des untersten Ausgangsbit bei Variation der Eingänge verändert wurde. Ein praktischer Test lieferte das gewünschte Ergebnis.

Hierarchischer Entwurf

Problem bei einem Ansteuern von allen Ziffern auf dem Board ist, dass es nur einen Eingangskanal für alle vier Anzeigen gibt. Hier wird der Anodeneingang wichtig.
Idee war, dass jede der vier Anzeigen nacheinander zum leuchten gebracht werden. Also wird zuerst der Wert der unteren vier Bit auf die letzte Ziffer übertragen, dann die nächsthöheren Bit auf die zweite Ziffer usw. Dies muss sehr schnell geschehen, so dass das menschliche Auge ein konstantes Leuchten wahrnimmt. Hier kommt ein Takteingang ins Spiel.

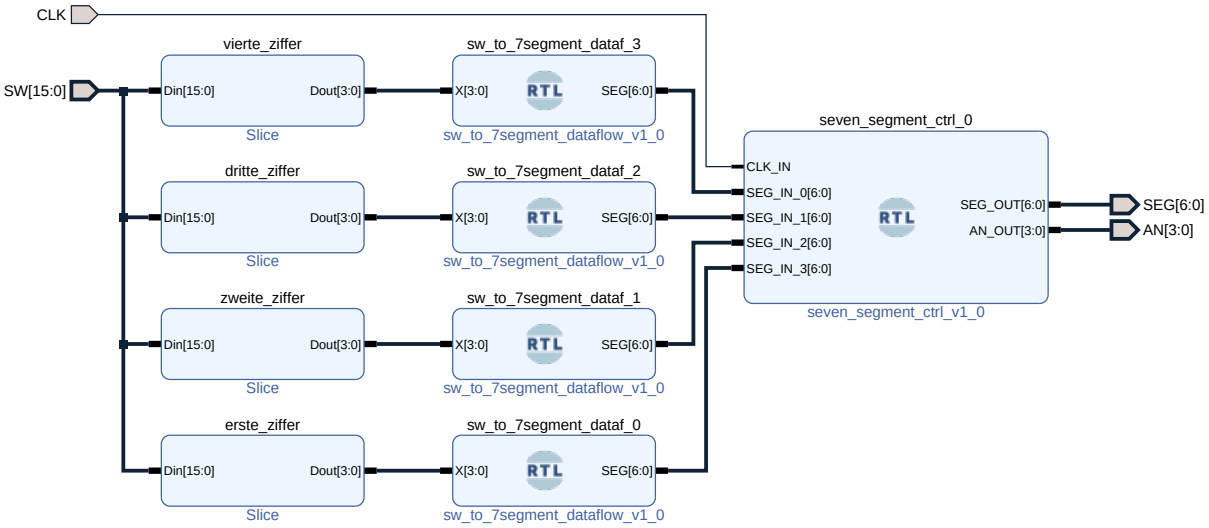

Abbildung 5: Block Design zum Ansteuern aller vier Ziffern

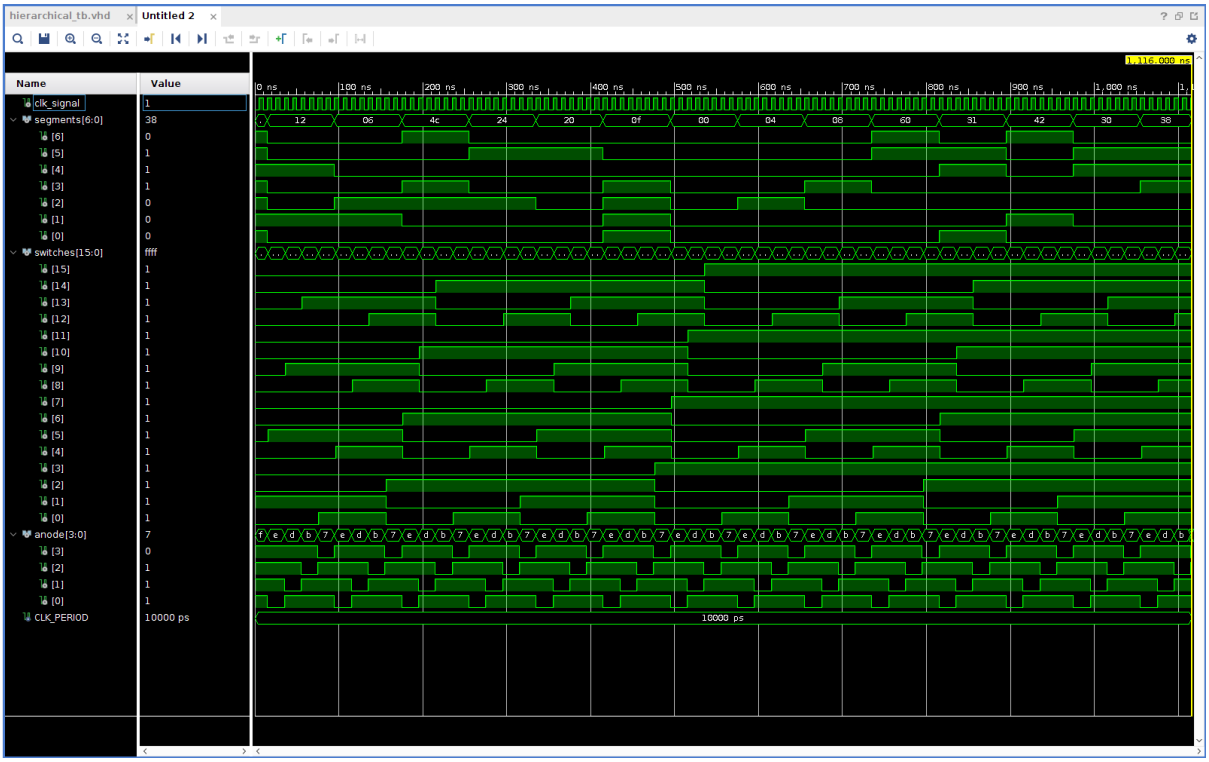

Abbildung 6: Waveforms der Testbench beim Ansteuern aller vier Ziffern

In diesem Aufbau (Abbildung 5) wurde das Modul *sw_to_7segment_dataf* im Design verwendet. Dieser Baustein stellt die Funktionalität aus Aufgabe 2 zur Verfügung, jedoch in einem Baustein vereinigt. Diesen Baustein selbst zu erstellen war ursprünglich Teil der Aufgabe, wurde von uns jedoch übersprungen. Es wurde also das Eingangssignal der Schalter zuerst zugeschnitten, sodass jeweils vier Bit in ein Modul zur Ansteuerung einer Anzeige geleitet wurden. Das Rotieren durch die Anode und den entsprechenden Eingang übernimmt ein ebenfalls vorgegebener Baustein mit dem Namen *seven_segment_ctrl*. Dieser hat neben den Vier sieben Bit breiten Eingängen einen Eingang mit dem der Takt zum Umschalten bestimmt wird. Hier wurde ein neuer Eingang dem Design hinzugefügt, welcher einen Takt (Clock) mit einer Frequenz von 100 MHz erzeugte. Die Testbench lief ohne Fehler durch (es musste eventuell der Test manuell gestartet werden, die Waveforms sind in Abbildung 6) und auch das Ergebnis auf der Hardware war wie erwartet. Es war kein flackern zu sehen und mit den obersten vier Bit konnte die erste Ziffer, mit den niedrigsten vier Bit die letzte Ziffer angesteuert werden. Eine Verifizierung der Funktion war hier erstmals nicht nur anhand der Waveforms möglich. Diese zeigen nur den Ausgang zu den Segmenten sowie die Belegung der Schalter. Dennoch ist anhand der Waveforms die korrekte Belegung der Anode zu erkennen. An der Anode ist immer nur ein Bit nach dem anderen aktiv (also Null).

Anzeige bei Tastendruck übernehmen

Um diese Aufgabe zu realisieren, muss die aktuelle Belegung der Anzeige gespeichert werden. Es wurde ein beliebig breites Speicherregister (*register_gen_bit_enable*) mit Speicherung auf steigender Taktflanke (am *Enable* Eingang) bereitgestellt. Hier gibt es mehrere Möglichkeiten zur Platzierung des Registers.

- 1. Vor dem Aufteilen des Eingagnssignals. Hier ist der Vorteil, dass nur ein einziges 16-Bit Register benötigt wird.
- 2. Nach dem Aufteilen. Die Speicherung der Werte jeder einzelnen Ziffer ist so unabhängig möglich.
- 3. Nach den Logikbausteinen zum Schalten der sieben Segmente. Ähnlich zu 2., das Register ist jedoch sieben statt vier Bit breit.

Ein Speichern von Daten nach dem Controller Baustein ist nicht Sinnvoll, da der Takt das Signal ständig verändert bzw. je nach Speicherzeitpunkt die Werte einer anderen Ziffer gespeichert würden. Am sinnvollsten erschien uns zu diesem Zeitpunkt eine Speicherung der 16-Bit Rohdaten, da die Anpassung am geringsten war. Als Eingangssignal um das Register zu aktivieren nutzten wir den Knopf auf dem Board mit der Bezeichnung *BTN_C*.

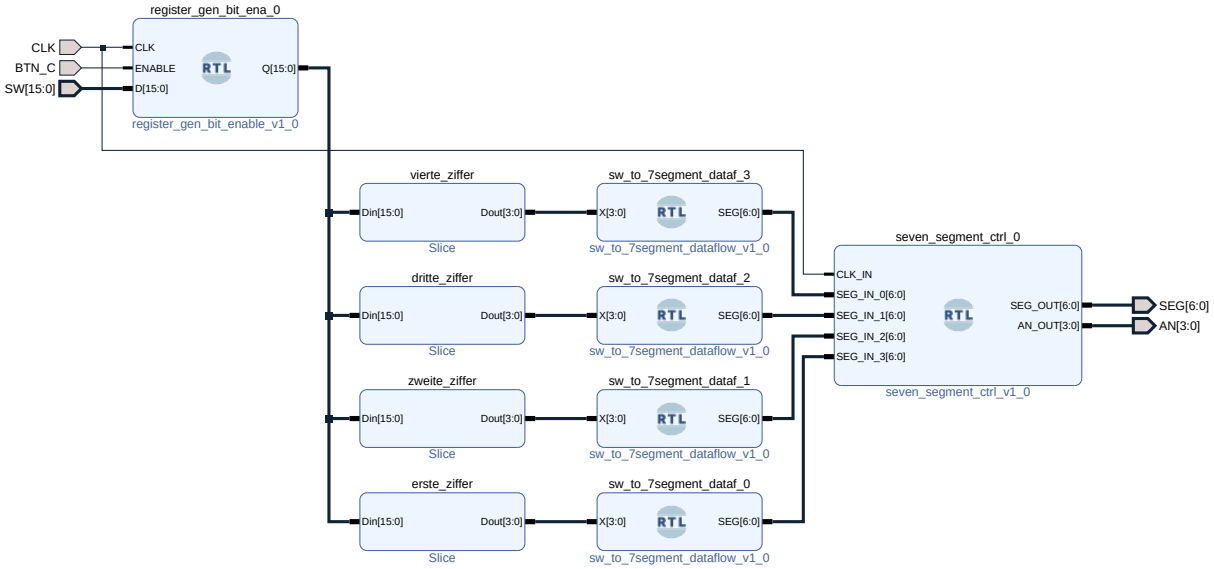

Abbildung 7: Block Design mit Speicherregister

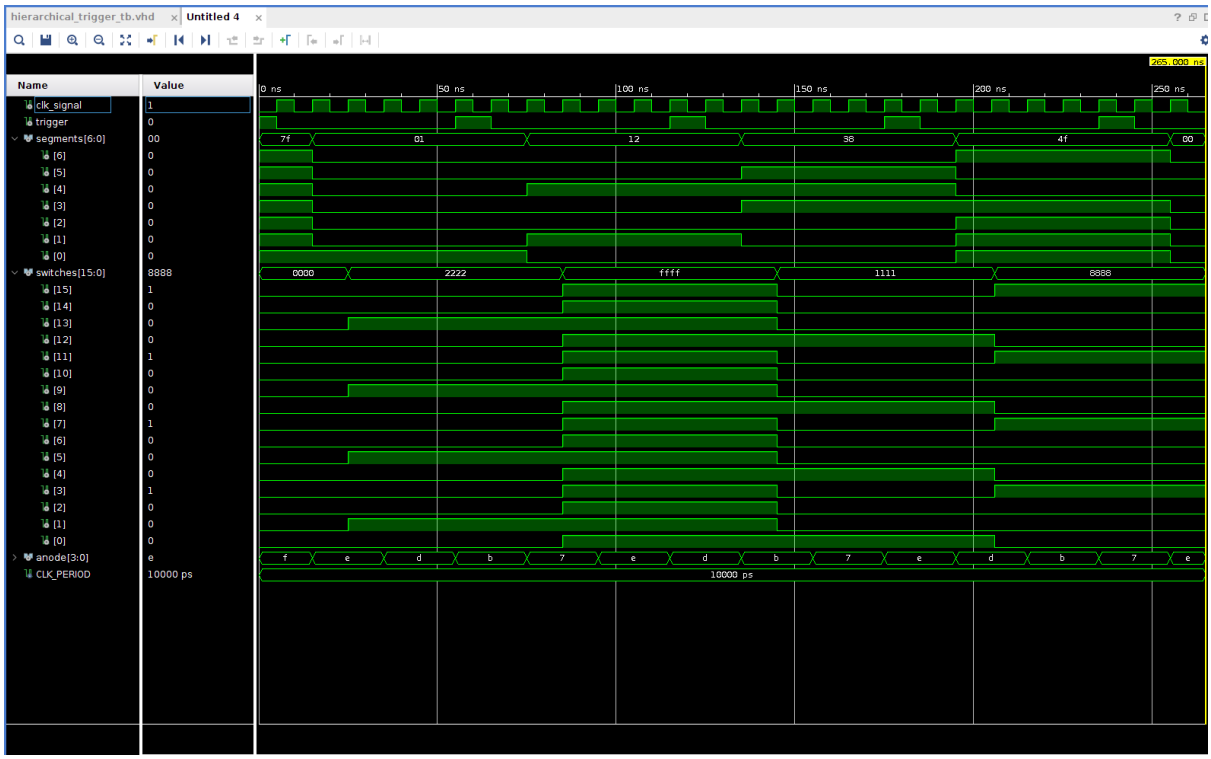

Abbildung 8: Waveforms zur Speicherung auf Tastendruck

Wie im Aufbau (Abbildung 7) und den Waveforms der entsprechenden Testbench (Abbildung 8) zu sehen, werden die an den Schaltern des Boards eingestellten Werte erst bei der nächsten Taktflanke nach drücken des Knopfes *BTN_C* (im Waveform *trigger*) auf den Ausgang zu den Segmenten übernommen.

Zurücksetzen der Anzeige

Hier wurde deutlich, dass die Entscheidung, das Register an den Anfang des Datenstroms zu platzieren, für diese Aufgabe nicht optimal war. Problem war, dass uns vier Bit Multiplexer zur Verfügung standen, wir zur Lösung der Aufgabe in dieser Konfiguration allerdings zwischen zwei 16-Bit Daten wechseln müssten. Daher wurde die Schaltung so umgebaut, dass vier Vier-Bit-Register den Zustand der Anzeige nach dem Aufteilen in die vier Bitblöcke Speichern. So konnten wir dem Design pro Ziffer einen Multiplexer hinzufügen, welcher beim Drücken von *BTN_D* den Eingang des Registers mit einem vier Bit breiten Nullvektor belegte. Sobald der Knopf losgelassen wurde, wurde wieder die Belegung aus den Schaltern benutzt. Der Aufbau des Beschriebenen ist in Abbildung 9 dargestellt. Die Testbench wurde ohne Fehler durchlaufen und die resultierenden Waveforms sind in Abbildung 10 zu sehen. Um die Anzeige wirklich Zurückzusetzen, musste der Null-Wert in das Register gespeichert werden. Daher war ein gleichzeitiges Drücken von *BTN_D* (zum Wählen des Nullvektors) und *BTN_C* (zum Speichern des Nullvektors im Register) notwendig. Eine Lösung dieses Problems hätte realisiert werden können, wenn der Knopf zum Wählen des Nullvektors gleichzeitig die Register aktiviert.

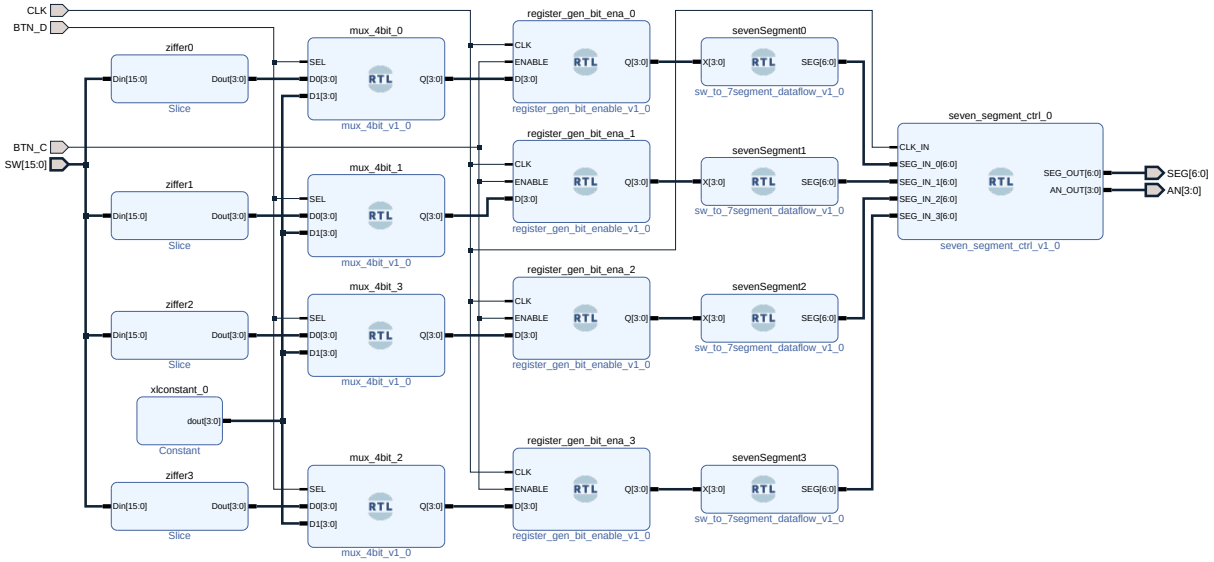

Abbildung 9: Block Design zum Zurücksetzen der Anzeige

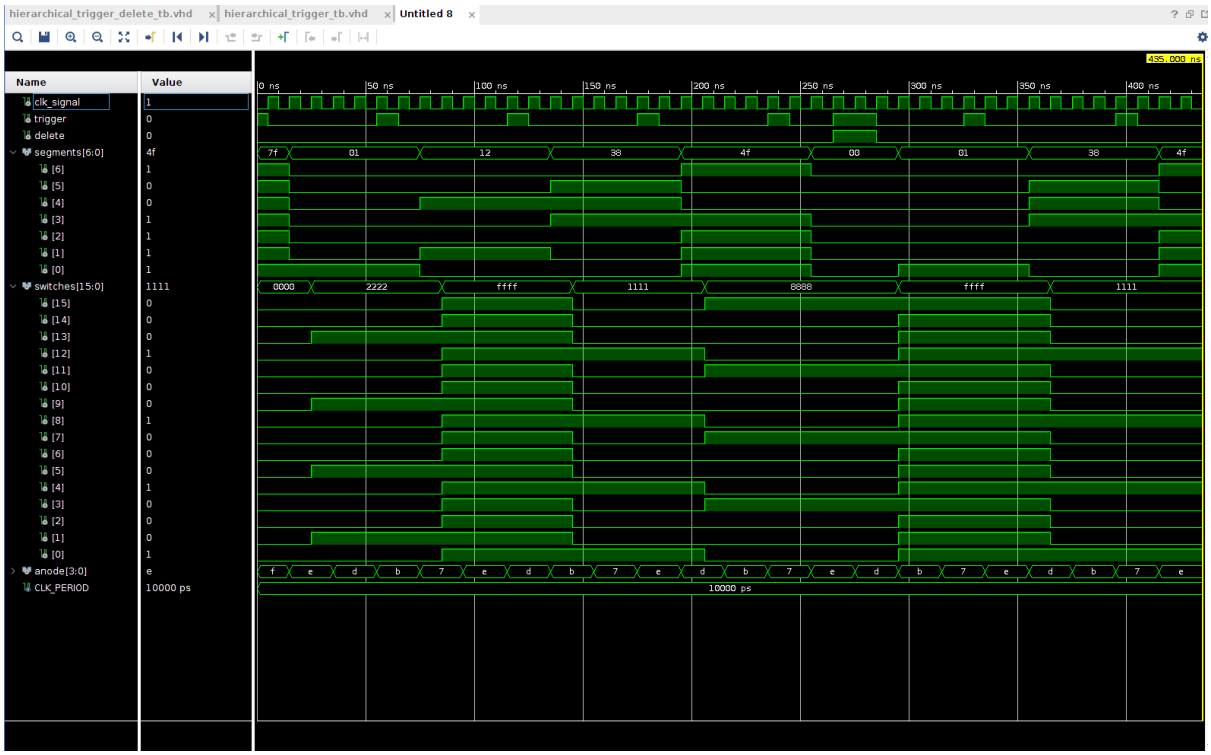

Abbildung 10: Waveforms zum Zurücksetzen der Anzeige

16-Bit Zähler

Um 16 Bit hochzählen zu können, wurden Addiererbausteine benötigt. Vorgegeben waren *add_8bit_v1_0*, welche nur acht Bit addieren können. Diese besitzen drei Eingänge und zwei Ausgänge: zwei jeweils acht Bit breite Inputs, die aufeinander addiert werden sollen, so wie ein Carry-Eingang, welcher ein Bit groß ist und auf das niederwertigste Bit addiert wird. Zudem einen acht Bit breiten Output, welcher das Ergebnis der Addition bereitstellt und einen ein Bit breiten Carry-Ausgang, welcher eine Eins liefert, wenn die Addition ein Ergebnis größer als acht Bit liefert.
Um daraus einen 16 Bit breiten Addierer zu basteln brauchten wir zwei dieser Bausteine. Einen für die unteren 8 Bit und einen für die höheren acht Bit. Um den Überlauf der unteren acht Bit zu zählen, wurde der Carry-Ausgang des ersten Addierers mit dem Carry-Eingang des zweiten Addierers verbunden. Da hier in jedem Schritt eine Eins addiert werden sollte, wurde der erste Eingang des ersten Addierers eine Konstante mit acht Bit Breite und Wert Eins verbunden. Der andere Eingang bezog die unteren 8 Bit des Register, aufgespalten durch einen Slice-Baustein. Für den Adder der höheren acht Bit wurden die höherwertigen acht Bit am Ausgang des Registers, wieder durch einen Slicer bezogen, sowie eine acht Bit breite Konstante mit dem Wert Null angeschlossen. Um hier hochzählen zu können, wurde der Carry-Ausgang des anderen Addierers mit dem Carry-Eingang dieses Addierers verbunden. Dies resultiert darin, dass wenn die niederwertigen acht Bit ihren Speicherbereich überschreiten, die höherwertigen acht Bit einmal hochzählen.
Die Zwischenergebnisse wurden in dem bereits bekannten Register gespeichert. Dieses hat bei einem Taktsignal auf dem Eingang *CLK* den Ausgang mit den Werten am Input überschrieben. Somit liegt am Eingang des Registers immer der bereits nächste, um 1 erhöhte Wert, wird aber nur bei einem CLK-Signal gespeichert. Zu beachten ist, dass das Taktsignal an dem Register durch den vorgegebenen *frequency_divider_v1_0* reduziert wurde, da ein Zähler, welcher mit 100 MHz hochzählt, nicht für das menschliche Auge sichtbar ist.
Hinter dem Register hängt die bereits zuvor verwendete Schaltung zum decodieren und darstellen auf dem Sieben-Segment-Display des Boards.
Zunächst lag das Problem vor, dass der Zähler immer zwei addierte und nicht eins. Nach etwas Suche klärte sich das Problem aber: Die Konstante *xlconstant_carry_0* war versehentlich mit dem Wert Eins statt Null belegt. Nachdem das Problem behoben war, lief die Testbench einwandfrei und auch auf dem Board konnte die Schaltung verifiziert werden. Bei Druck des Knopfes *BTN_D* wurde der Zähler auf Null zurückgesetzt.

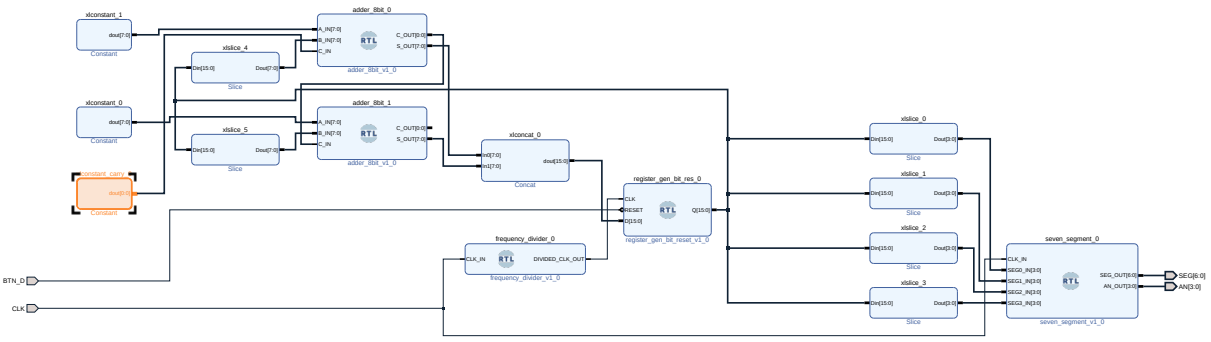

Abbildung 11: Blockdesign des 16-Bit Zählers

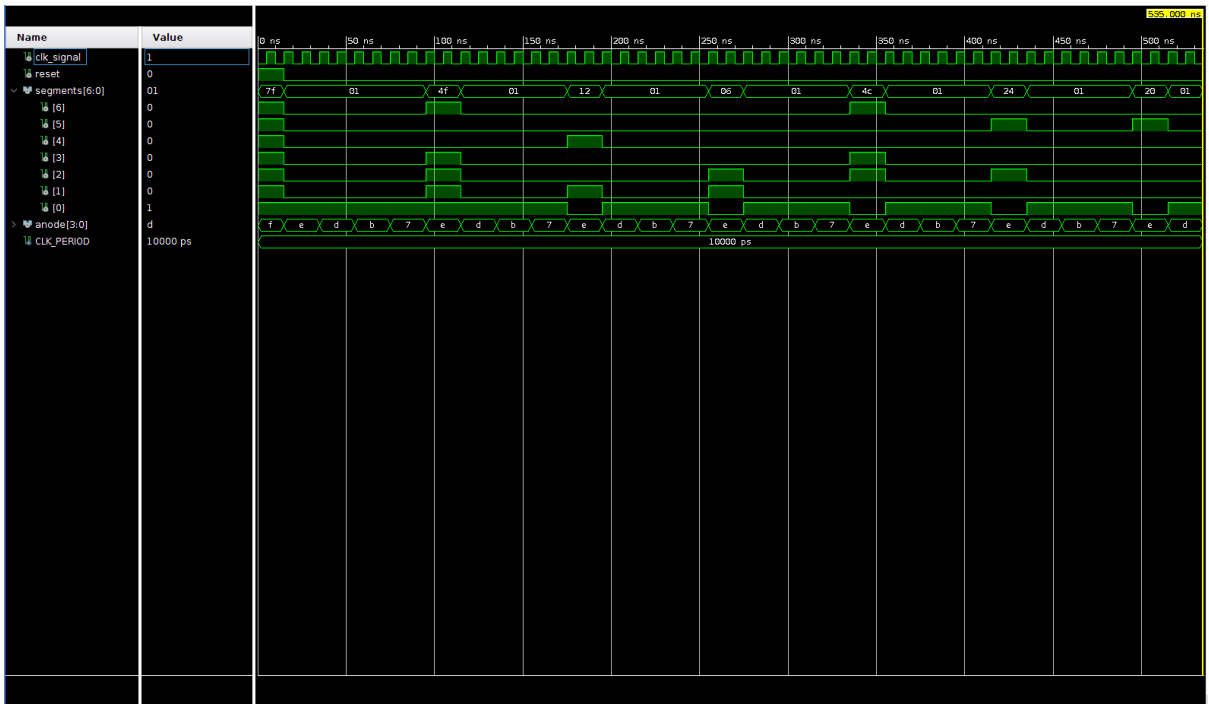

Abbildung 12: Waveforms zum Counter

Fazit

FPGAs sind äußert vielseitige Werkzeuge, mit denen man kleinere Spielereien bauen kann, aber auch professionell arbeiten kann. Mithilfe von Tools wie Vivado kann man auch ohne großes Verständnis der zugrunde liegenden HDL bereits Projekte mit den FPGAs realisieren. Interessant ist dabei die andere Denkweise als beim Programmieren: Man muss sich Gedanken um die einzelnen Signale machen und wie diese interpretiert werden und wie man diese verknüpft, um bestimmte Schaltungen zu verwirklichen.

Literaturverzeichnis

[1] Handbuch Basys 3
`https://reference.digilentinc.com/_media/reference/programmable-logic/basys-3/basys3_rm.pdf` 05.07.2018 13:55
[2] Field Programmable Gate Array
`https://de.wikipedia.org/wiki/Field_Programmable_Gate_Array` 05.07.2018 15:09