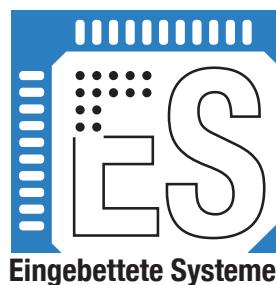


Praktikum
Grundlagen von Hardwaresystemen
Sommersemester 2015

Versuch 3:



10. November 2016

Fachbereich 12: Informatik und Mathematik
Institut für Informatik
Professur für Eingebettete Systeme
Prof. Dr. Uwe Brinkschulte
Andreas Lund

Johann Wolfgang Goethe-Universität
Frankfurt am Main

Inhaltsverzeichnis

1 Einleitung	2
2 Grundlagen	3
2.1 D-Flipflop	3
2.2 Binär Kodierte Dezimalzahlen (BCD)	4
2.3 Entwurf von Zählereinheiten	5
2.3.1 BCD-Zähler als asynchroner Mod-10-Zähler	7
2.4 Xilinx Spartan 6 FPGA	10
2.5 FPGA-Board Nexys 3	11
2.5.1 Routing von Signalen und Pins	12
2.5.2 Warnhinweise	13
2.5.3 Anschließen der Platine	13
2.5.4 Download des Bitstreams	13
3 Anmerkungen und Tipps	16
4 Vorbereitungsaufgaben	17
5 Praktikumsaufgaben	19

Kapitel 1

Einleitung

Dieser Versuch soll die Funktionsweise eines asynchronen BCD-Zählers vermitteln, mit welchem abschließend eine digitale Stoppuhr auf rekonfigurierbarer Hardware umgesetzt werden soll. Als Zielplattform verwenden Sie eine bereitgestellte FPGA-Entwicklungsplattform des Herstellers Xilinx. Das Ziel dieses Versuches ist unter anderem die Erläuterung des Zusammenhangs zwischen einer Hardwarebeschreibungssprache (VHDL, Verilog, System C), der Schaltungssynthese und der FPGA-Prototyp-Plattform. Die Aufgaben in diesem Versuch sind im Wesentlichen:

- Umsetzung eines asynchronen BCD-Zählers in VHDL
- Implementierung einer digitalen Stoppuhr
- Einbindung einer 7-Segment Anzeige als visuelle Ausgabe
- Inbetriebnahme der „Nexys 3“ FPGA-Platine
- Funktionstest der Stoppuhr mit Hilfe der FPGA-Platine.

Kapitel 2

Grundlagen

2.1 D-Flipflop

Speicherglieder gehören zu den Elementarbausteinen einer digitalen Rechenanlage. Sie können Schaltvariablen aufnehmen, speichern und abgeben. Da nur die Verarbeitung von binären Schaltvariablen betrachtet wird, muss ein Speicherglied die Eigenschaft haben, eine Variable mit den Werten 0 und 1 aufnehmen zu können. Speicherglieder mit dieser Eigenschaft sind bistabile Kipplieder, auch **Flipflops** genannt. Ein bistabiles Kipplied kann, wie der Name sagt, zwei stabile Zustände einnehmen: einen Zustand 1 (Setzzustand) und einen Zustand 0 (Rücksetzzustand). Ein **Flipflop** ist also ein Speicherglied mit zwei stabilen Zuständen, das aus jedem der beiden Zustände durch eine geeignete Ansteuerung in den anderen Zustand übergeht. Die Ansteuerung kann sein:

- taktunabhängig
- taktabhängig
 - taktzustandsgesteuert
 - taktflankengesteuert

Diese unterschiedliche Art der Ansteuerung führt zu verschiedenen Flipflop-Typen. Das transparente D-Flipflop (auch D-Latch genannt) hat zwei Eingänge: Einen Takteingang C und einen Dateneingang D. Solange am Takteingang eine 0 anliegt, behält das Flipflop seinen momentanen Zustand bei. Liegt an C eine Eins an, übernimmt das Flipflop den am Dateneingang anliegenden Wert als neuen Zustand. Jetzt betrachten wir das Blockschaltbild des D-Flipflops (Abb. 2.1), die zugehörige Zustandsfolgetabelle, das Zustandübergangsdiagramm und die Übergangsfunktion.

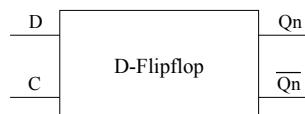


Abbildung 2.1: Blockschaltbild eines D-Flipflops

C D	Q_n	Q_{n+1}
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	1
1 1	1	1

Tabelle 2.1: Zustandsfolgetabelle für ein D-Flipflop mit Zustandssteuerung

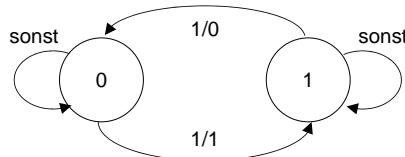


Abbildung 2.2: Zustandübergangsdiagramm

Die Kanten des Diagramms sind mit den jeweiligen Werten für das Paar (C,D) versehen.
Übergangsfunktion:

$$Q_{n+1} = D$$

2.2 Binär Kodierte Dezimalzahlen (BCD)

In der Digitalelektronik wird als Zahlensystem das Dualsystem benutzt. Die beiden Zustände werden im Dualsystem durch die beiden Signale 0 und 1 abgebildet. Der Mensch denkt und rechnet im Dezimalsystem, in dem Zahlen mittels den Ziffern 0 - 9 dargestellt werden. Für einen Programmierer ist es deshalb erforderlich, lange Ziffernfolgen mit vielen Bits des Dualsystems schnell in Dezimal umzuwandeln und auszuwerten, oder gar eigene Ziffernfolgen des Dualsystems in das System einzugeben. Eine Möglichkeit hierfür ist, Dezimalzahlen binär zu kodieren. Eine solche Darstellung nennt man Binär kodierte Dezimalzahlen (BCD-Code). Hierbei wird zunächst jede Stelle einer Dezimalzahl unter Benutzung des binären Zeichenvorrats (0,1) abgebildet. Für die Darstellung der Dezimalziffern 0 bis 9 werden somit mindestens 4 Binärstellen benötigt. Zahlen werden deshalb im BCD-Format stets in 4-Bit-Blöcken gruppiert, wodurch sich insgesamt 16 verschiedene Zahlenwerte (0-15) darstellen lassen. Da für die Darstellung von dezimalen Zahlen jedoch nur 10 Werte (0-9) notwendig sind, unterteilt man die BCD-Ziffernfolgen in gültige (0-9) und ungültige (10-15) Bereiche. Alle gültigen Ziffernfolgen werden als Tetraden bezeichnet und ungültige als Pseudotetraden.

Folglich lässt sich mit 4 Tetraden (16 Bits) ein dezimaler Zahlenwert innerhalb des Bereichs von 0 bis 9999 darstellen. Dabei wird jede Tetrade als eine eigene Ziffer im Dezimalsystem interpretiert. Von rechts nach links gesehen ist wie im Dezimalsystem die Bildung von Einer, Zehner, Hunderter, Tausender usw. Stellen möglich.

2.3 Entwurf von Zählereinheiten

Zählereinheiten sind heute ein elementarer Bestandteil zahlreicher integrierter Schaltkreise. Die Verwendung eines Zählers ist dabei oftmals vielseitig und unspezifisch. Auf Schaltungsebene unterscheidet man zwischen synchronen und asynchronen Zählereinheiten. Da Zähler im Gegensatz zu gewöhnlichen Schaltnetzen einen aktuellen Zählerzustand aufweisen, werden Sie zur Gruppe der Schaltwerke gezählt. Die Speicherung des Zählerstandes erfolgt dabei binärkodiert in Flipflops. Hierbei werden vorzugsweise aus Einfachheitsgründen taktflankengesteuerte D-Flipflops verwendet. Der Unterschied zwischen synchroner und asynchroner Funktionsweise liegt in der Verschaltung der Flipflops und der damit verbundenen Ansteuerung des Taktsignals. Während bei der synchronen Variante der aktuelle Zählerstand für alle Flipflops simultan zum selben Taktzyklus abgespeichert wird, erfolgt dies bei asynchronem Prinzip sequentiell zu jeweils zeitlich versetzten Taktzyklen. Dies wird bei synchroner Variante durch eine parallele Anordnung der Flipflops erreicht. Zusätzlich ist ein Schaltnetz bestehend aus Voll- und Halbaddierern vorgeschaltet, welches die fortlaufende Inkrementierung des aktuellen Zustandes in beliebigen Zählerschritten ermöglicht.

Im Folgenden dient das vereinfachte Beispiel eines 4-Bit Zählers, welches die grundlegenden Funktionsweisen beider Varianten genauer verdeutlichen soll. Tabelle 2.2 repräsentiert die allgemeingültige Zustandsübergangstabelle des Zählers, in welcher die Zustandsabfolge und dessen Wertebereich [0,15] zeilenorientiert aufgeführt ist. Die Speicherung des aktuel-

Taktzyklus	Q				Q+			
	D	C	B	A	D	C	B	A
1	0	0	0	0	0	0	0	1
2	0	0	0	1	0	0	1	0
3	0	0	1	0	0	0	1	1
4	0	0	1	1	0	1	0	0
5	0	1	0	0	0	1	0	1
6	0	1	0	1	0	1	1	0
7	0	1	1	0	0	1	1	1
8	0	1	1	1	1	0	0	0
9	1	0	0	0	1	0	0	1
10	1	0	0	1	1	0	1	0
11	1	0	1	0	1	0	1	1
12	1	0	1	1	1	1	0	0
13	1	1	0	0	1	1	0	1
14	1	1	0	1	1	1	1	0
15	1	1	1	0	1	1	1	1
16	1	1	1	1	0	0	0	0

Tabelle 2.2: Zustandsübergangstabelle eines 4-Bit Zählers mit Überlauf

len Zustandes Q sowie dessen Folgezustand Q+ erfolgt in den Flipflops A,B,C und D. Die Bit-Reihenfolge entspricht der alphabetischen Anordnung, wobei Flipflop A das niederwertigste und Flipflop D das höchstwertigste Bit repräsentiert. Ausgehend vom Anfangszustand $Q_0 = (0_D, 0_C, 0_B, 0_A)$ inkrementiert der Zähler fortlaufend zu jedem Taktsignal seinen Zustand bis der natürliche Überlauf in Zustand $Q_{15} = (1_D, 1_C, 1_B, 1_A)$ erreicht wird. Anschlie-

ßend kehrt der Zähler in den Anfangszustand Q_0 zurück und der Zählvorgang beginnt von vorne.

Ausgehend auf dem in Tabelle 2.2 vorgestellten Verhalten zeigt Abbildung 2.3 die Umsetzung einer synchron-basierten Variante des 4-Bit Zählers, bestehend aus den vier Flipflops A,B,C, und D, drei Voll- und einem Halbaddierer. Nennenswert und deutlich zu erkennen ist das taktsynchrone Speicherverhalten der Flipflops. Der aktuelle Zustand Q ist an den Ausgängen Q_A, Q_B, Q_C und Q_D zu finden. Durch eine direkte Rückkopplung der Ausgänge auf ein additives Schaltnetz wird der Zustand $Q+$ errechnet, der wiederrum auf die Daten-eingänge der Flipflops zurückgeführt wird und dessen Speicherung zum nächsten Taktsignal (Clk) synchron erfolgt. Der Vorteil dieser Variante liegt in der einfachen Funktionsweise und der individuellen Zählerschrittgröße, die jedoch einen erheblichen Mehraufwand an Hardware nach sich zieht.

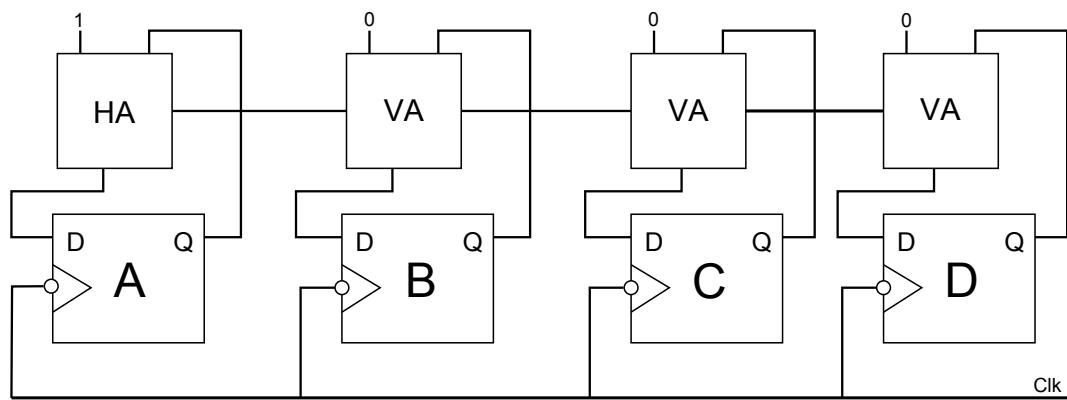


Abbildung 2.3: 4-bit synchrones Zählwerk

Gegenläufig ist der Aufbau eines asynchronen Zählerwerkes mit geringerem Hardware-Aufwand verbunden. Anstelle eines aufwendigen Addierer-Schaltnetzes werden ausschließlich Flipflops zur Berechnung und Speicherung des fortlaufenden Zahlerstandes benötigt. Diese werden in sequentieller Reihenfolge hintereinander an den Ausgängen miteinander gekoppelt, wie in Abbildung 2.4 gezeigt. Für jeden Zählerstand findet ein asynchroner Wechsel statt,

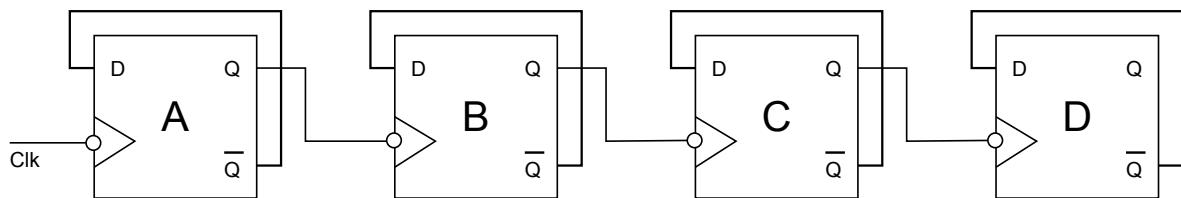


Abbildung 2.4: 4-bit asynchrones Zählwerk

bei welchem die beteiligten Speicherglieder ihren Zustand zyklisch und nacheinander ändern. Dabei wird der Effekt des natürlichen Kippverhaltens (Toggling) der Flipflops ausgenutzt ($\text{Übergang } 1 \rightarrow 0$), um sowohl das Inkrementierverhalten wie auch das taktversetzte Speichern abzubilden. Die Anzahl verschiedener Taktsignale entspricht somit der Anzahl der Flipflops, wobei der Ausgang Q eines Flipflops als Taktsignal für das darauffolgende Flipflop dient. Dies kann aus der zugehörigen Zustandswechselfolge der Flipflops unter Berücksichtigung der Zustandübergangstabelle 2.2 ermittelt werden. Hierbei sind alle Taktzyklen relevant, bei welchem

ein Flipflop seinen internen Zustand von Q nach $Q+$ ändert. Dementsprechend ergibt sich für die Flipflops A,B,C und D die in Tabelle 2.3 aufgeführte Wechselsequenz mit zugehörigem Takt-Signal. Im Gegensatz zur synchronen Variante ist eine Anpassung der Zählerschrittgröße nicht

Flipflop	Taktzyklus	Taktsignal
A	alle	Clk
B	2,4,6,8,10,12,14,16	Q_A
C	4,8,12,16	Q_B
D	8,16	Q_C

Tabelle 2.3: Zustandswechselsequenz der Flipflops bei asynchroner Taktkopplung der Ausgänge

möglich, da stets zyklisch inkrementiert wird. Ein weiterer Nachteil liegt im Rücksetzverhalten des Zählers. Während die synchronisierte Variante zu jedem beliebigen Taktzyklus in den Zustand Q_0 zurückgesetzt werden kann, ist dies bei asynchroner Funktionsweise nur über den natürlichen Überlauf des Zählers möglich. Synchronisierte Varianten werden deshalb überwiegend für die Erzeugung individueller Zahlenfolgen bevorzugt eingesetzt. Hingegen ist das Verhalten asynchroner Zähler statisch und überwiegend hardwaregebunden. Diese Varianten werden deshalb oft in hochfrequenten Schaltungsbereichen verwendet, wie beispielsweise bei der Erzeugung eines dynamischen Taktsignals durch einen Verteiler.

2.3.1 BCD-Zähler als asynchroner Mod-10-Zähler

Ein markantes Beispiel für die Verwendung asynchroner Zähler, ist die Erzeugung einer binär kodierten dezimalen Zahlenfolge (BCD). Solche BCD-Zähler finden beispielsweise noch immer überwiegend Einsatz in digitalen Uhrwerken. Basierend auf dem BCD-Darstellungsformat (Abs. 2.2) wird eine dezimale Zahlenfolge ziffernweise in Binärdarstellung abgebildet, zyklisch inkrementiert und nach erreichen des Zählerstandes 9 zum Anfangswert 0 zurückgesetzt. Für die Umsetzung des BCD-Zählers soll hier vorzugsweise ein 4-Bit asynchrones Zählwerk verwendet werden. Jedoch wird anstelle des normalen Zählerüberlaufs ($\text{mod } 16$) eine Modulo 10 Funktion benötigt. Um dieses Verhalten sukzessiv und methodisch zu entwerfen, ist eine Analyse der internen Flipflop-Zustände notwendig. Dies erfolgt mit Hilfe der aufgeführten Zustandsübergangstabelle 2.4.

Takt	Q				Q+			
	D	C	B	A	D	C	B	A
1	0	0	0	0	0	0	0	1
2	0	0	0	1	0	0	1	0
3	0	0	1	0	0	0	1	1
4	0	0	1	1	0	1	0	0
5	0	1	0	0	0	1	0	1
6	0	1	0	1	0	1	1	0
7	0	1	1	0	0	1	1	1
8	0	1	1	1	1	0	0	0
9	1	0	0	0	1	0	0	1
10	1	0	0	1	0	0	0	0

Tabelle 2.4: Zustandsübergangstabelle des asynchronen BCD-Zählers

Anhand der Tabelle lässt sich erkennen, zu welchem Taktzyklus ein Flipflop seinen internen Zustand wechselt. Dies entspricht jedem Eintrag bei dem der aktuelle Zustand Q des entsprechenden Flipflops nicht mit dem Folgezustand $Q+$ übereinstimmt. Für die insgesamt vier vorhandenen Flipflops lassen sich daher die folgenden Zustandsübergänge zu den folgenden Taktzyklen erkennen.

Flipflop	Taktzyklus	Taktsignal
A	alle	Clk
B	2,4,6,8	-
C	4,8	Q_B
D	8,10	-

Tabelle 2.5: Zustandswechselfolge der Flipflops eines BCD-Zählers (mod 10)

Aufgrund der in Tabelle 2.5 aufgeführten Wechselfolge wird ersichtlich, dass durch eine direkte Kopplung der Ausgänge Q für die beiden Flipflops B und D kein vollständig übereinstimmendes Taktsignal generiert werden kann. Deshalb ist es notwendig, eine möglichst minimale Überdeckung der benötigten Taktzyklen mit den tatsächlich verfügbaren Taktsignalen Clk, Q_A, Q_B und Q_C aus Tabelle 2.3 zu finden. Eine gültige Überdeckung betreffend der beiden Flipflops B und D wird durch die erweiterte Zustandswechselfolge in Tabelle 2.6 dargestellt. Hier wurde für Flipflop B und D jeweils der Ausgang Q_A als geeignetes Taktsignal gewählt. Anzumerken ist, dass die in den Klammern aufgeführten Taktzyklen als Folge der Überdeckung mit dem jeweiligen Taktsignal nun miteinbezogen werden müssen, obwohl diese gemäß Zustandsübergangstabelle nicht erforderlich wären. Dadurch wird ein unbeabsichtigtes, fehlerbehaftetes Speichern in den besagten Flipflops vermieden.

Flipflop	Taktzyklus	Taktsignal
A	alle	Clk
B	2,4,6,8,(10)	Q_A
C	4,8	Q_B
D	(2),(4),(6),8,10	Q_A

Tabelle 2.6: Erweiterte Zustandswechselfolge der Flipflops eines BCD-Zählers (mod 10)

Basierend auf der in Tabelle 2.6 erweiterten Wechselfolge kann nun das Verhalten eines jeden Flipflops durch seinen Zustand im zugehörigen Anwendungsdiagramm bestimmt werden. Da vier Flipflops notwendig sind, werden vier KV-Diagramme mit vier Argumenten benötigt. Die vom Binärzähler als ungültig definierten Zustände '10' bis '15' sowie die nicht relevanten werden dabei nicht berücksichtigt und mit "don't cares" (-) gekennzeichnet.

$D_A:$	Q_C	Q_A	Q_B	Q_D
1 0 2 -	0 1 3 -	0 5 7 -	0 4 6 -	1 -
10 8	11 9	15 13	14 12	-

$D_B:$	Q_C	Q_A	Q_B	Q_D
- 0 - -	1 1 0 -	1 5 0 -	1 4 6 -	-
2 -	3 -	7 -	7 -	-
10 8	11 9	15 13	14 12	-

$D_C:$	Q_C	Q_A	Q_B	Q_D
- 0 - -	1 3 7 -	5 0 6 -	4 - -	-
2 -	3 -	7 -	6 -	-
10 8	11 9	15 13	14 12	-

$D_D:$	Q_C	Q_A	Q_B	Q_D
- 0 - -	1 3 7 -	5 0 1 6	4 - -	-
2 -	3 -	7 -	6 -	-
10 8	11 9	15 13	14 12	-

Mit Hilfe einer graphischen Minimierung durch geeignete Verbund-Bildung, ergeben sich die folgenden Funktionsgleichungen für die Ansteuerung der Flipflops.

$$\begin{aligned} D_A &= \overline{Q_A} \\ D_C &= \overline{Q_C} \\ D_B &= \overline{Q_B} \wedge \overline{Q_D} \\ D_D &= Q_B \wedge Q_C \end{aligned}$$

Für das vollständige Schaltwerk des BCD-Zählers gilt also die folgende Realisierung:

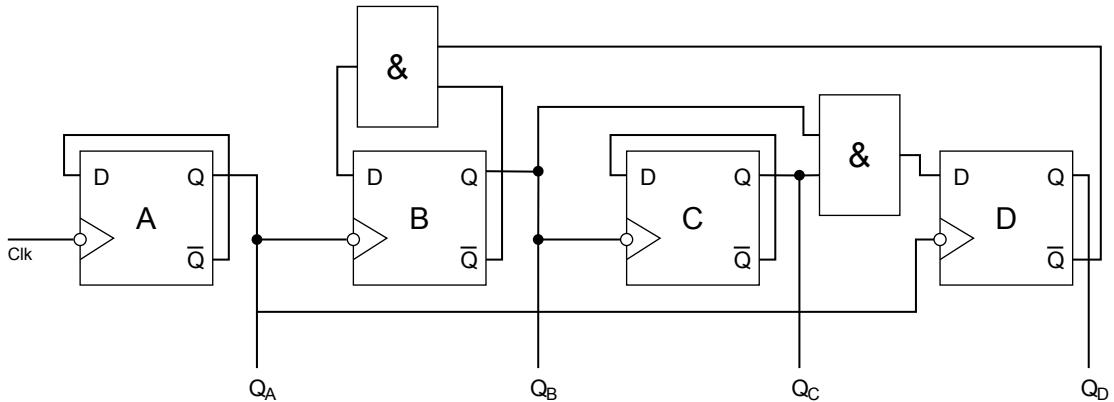


Abbildung 2.5: BCD-Zähler mit asynchronem Schaltwerk

2.4 Xilinx Spartan 6 FPGA

FPGAs (Field Programmable Gate Arrays) bestehen aus flächig angeordneten Logikblöcken (Basiszellen), zwischen denen Verbindungskanäle verlaufen. Die Verschaltung der Logikblöcke untereinander, sowie deren jeweilige Funktion sind vom Anwender programmierbar (konfigurierbar). Man spricht hierbei auch von einer Personalisierung des Bausteins.

In einem **FPGA** gibt es im wesentlichen zwei Arten von Blöcken: Logikblöcke (Logikzellen) und I/O-Blöcke (Input/Output-Blöcke). Die Logikblöcke implementieren boolesche Funktionen, während die I/O-Blöcke das elektrische Verhalten der Ein- und Ausgänge bestimmen (z. B. Schaltschwellen). Die jeweiligen Bezeichnungen differieren jedoch von Hersteller zu Hersteller. Erwähnenswert sind besonders die relativ weit verbreiteten Abkürzungen CLB für Configurable Logic Block und IOB für Input/Output-Block der Firma Xilinx, die 1985 das erste **FPGA** vorstellte. In der Abbildung 2.6 wird eine mögliche Abbildung eines Ein-Bit-Volladdierers auf die Logikressourcen eines **FPGAs** gezeigt.

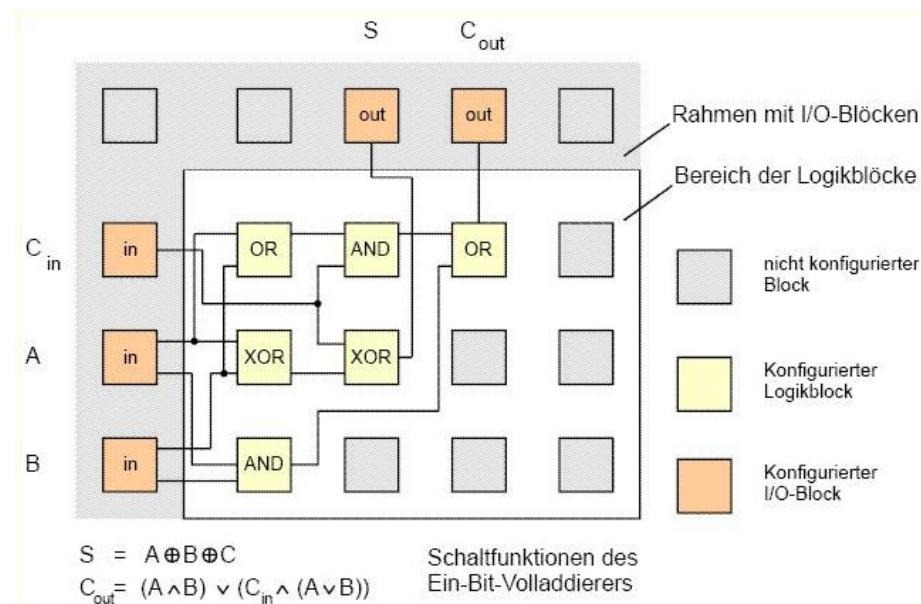


Abbildung 2.6: Abbildung eines Ein-Bit-Volladdierers auf die Logikressourcen eines FPGAs

Deutlich zu erkennen ist die unterschiedliche Länge der einzelnen Leitungssegmente. Die regelmäßige Anordnung der Logikblöcke und Verdrahtungskanäle sind von einem Rahmen aus I/O-Blöcken umgeben. Diese beinhalten Signalaufbereitung, Pegelanpassung, etc. und teilweise auch Register zum zwischenspeichern von Ein- und Ausgangssignalen. Sie stellen über die Anschlußpins die Verbindung des Bausteins zu anderen Schaltungsteilen her.

Zur Realisierung boolescher Funktionen in den Basiszellen (CLB = configurable logic block) der **FPGAs** verwendet man heutzutage im Wesentlichen Lookup-Tables (LUTs). Eine LUT ist einfach eine Wertetabelle, in der die durch den Logikblock zu realisierenden Funktion abgelegt ist, im Grunde also einfach ein winziger RAM-Baustein.

Die Bitkombination am Eingang adressiert eine der Speicherzellen der Lookup-Table. Das dort abgespeicherte Bit wird dann ausgegeben. Die Verwendung von Lookup-Tables ermöglicht die Realisierung beliebiger boolescher Funktionen; alternativ können sie als adressierbarer Speicher verwendet werden.

Der Ausgang wird einmal direkt in das Verbindungsnetzwerk herausgeführt und einmal über ein D-Flipflop.

Die Kapazität der LUT liegt typischerweise bei 64 Bit. Damit hat das RAM 6 Adressleistungen und somit kann jede Boolesche Funktion mit 6 Variablen implementiert werden.

2.5 FPGA-Board Nexys 3

Das Nexys 3 Developer Board ist eine vielseitige Basisplatine zu Verifikations- und Testzwecken von FPGA-Entwürfen. Neben dem eigentlichen FPGA-Chip besitzt die Platine eine umfangreiche Peripherie, die dynamisch mit dem Logik-Baustein verbunden und angesteuert werden kann. Die Platine (Abb. 2.7) bietet hierzu folgende für die Versuche interessante Elemente:

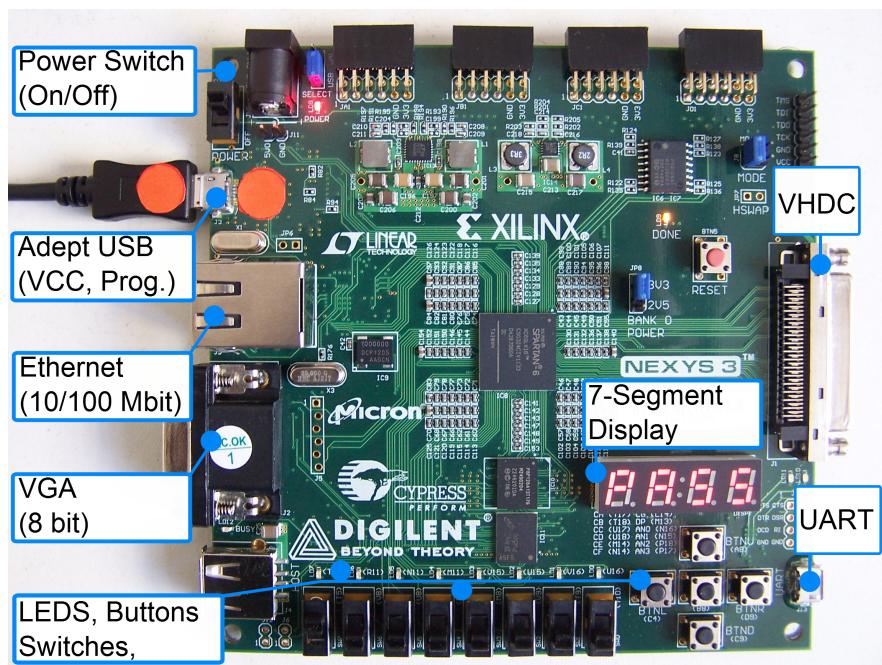


Abbildung 2.7: Abbildung der Nexys 3 FPGA-Platine

- Spartan-6 (XC6SLX16) FPGA von Xilinx
- Analoger VGA-Ausgang mit 256 Farben
- 10/100 Mbit Ethernet-Schnittstelle
- VHDC- und UART-Schnittstelle
- 8 LEDs
- 4 Taster, 8 Schiebeschalter
- eine 4-stellige 7-Segment Anzeige

Neben den oben genannten Bauteilen befinden sich auch die folgenden zugehörigen Anschlüsse auf der Platine:

- Adept USB-Anschluss, zur Konfiguration und Spannungsversorgung mit seriell Mikro-USB Kabel
- VGA-Port
- RJ-45 Ethernet Buchse
- 68-Pin VHDCI Connector
- Mikro-USB-UART Anschluss

Des Weiteren führen zusätzliche Pmod-Steckerleisten verschiedene Signale des FPGAs nach außen. Diese werden in den Versuchen jedoch nicht benötigt.

2.5.1 Routing von Signalen und Pins

Um die internen logischen Signale eines VHDL-Entwurfs den entsprechenden physikalischen Pins des FPGA-ICs zuzuweisen, ist ein explizites Routing (Vernetzung) durch den Anwender notwendig. Dies erfolgt mittels einer Look-Up Table, die durch eine zusätzliche Datei, dem sogenannten User Constraint File (UCF), repräsentiert wird. Strukturell betrachtet, beinhaltet das UCF daher eine Menge von statischen Routing-Anweisungen, die jeweils eine logische Quelle und ein physikalisches Ziel spezifiziert. Der Aufbau und Syntax einer UCF-Datei ist anhand des folgenden Entity-Beispiels dargestellt:

```

1 ENTITY DUMMY IS PORT
2 (
3     CLK_SIGNAL : OUT STD_LOGIC;
4     OUTPUT_SIGNAL : OUT STD_LOGIC_VECTOR(2 DOWNTO 0)
5 );
6 END DUMMY;
7

```

Listing 2.1: Beispiel Entity DUMMY mit einem Ein-/Ausgang

```

1 # Eingangstaktsignal #
2 NET "CLK_SIGNAL" LOC = "V10";
3 # Ausgangssignale #
4 NET "OUTPUT_SIGNAL<0>" LOC = "T17";
5 NET "OUTPUT_SIGNAL<1>" LOC = "T18";
6 NET "OUTPUT_SIGNAL<2>" LOC = "U17";

```

Listing 2.2: UCF-Datei für Entity DUMMY

Eine gültige Routing-Anweisungen innerhalb des UCF-Formats erfolgt zeilenweise, beginnend mit der Anweisung „NET“ gefolgt von der logischen Signalquelle und dem letztlichen Ziel-Pin. Alle in der UCF-Datei aufgeführten Signale werden mit den in und out-Ports der zugehörigen Top-Level Entity des Projekts assoziiert. Achten Sie deshalb bei der Programmierung stets darauf, dass alle Signale der Top-Level Entity im UCF korrekt und vollständig geroutet sind. Fehlerhaftes und unvollständiges Routing kann zur Beschädigung der Platine führen. Damit der ISE Compiler die UCF-Datei letzten Endes erkennt, muss die Dateierweiterung stets „.ucf“ lauten. Für das Anlegen und Editieren der Datei kann aber jeder unicode-fähige Texteditor (UTF-8) oder das bereits vorhandene ISE Webpack verwendet werden. Um

eine fertige UCF-Datei dem Projekt hinzuzufügen, klicken Sie in dem Fenster **Processes** unter dem Punkt **User Constraints** auf **Edit Constraints (Text)**. Die Abfrage wird mit **Yes** beantwortet. Anschließend kann die UCF-Datei auch im ISE Webpack editiert werden.

2.5.2 Warnhinweise



Abbildung 2.8: Warnhinweis für elektrostatisch empfindliche Geräte

Die Platine ist elektrostatisch empfindlich, d.h. winzige statische Aufladungen können die ICs auf der Platine zerstören. Bitte erden Sie sich daher vor der Handhabung (Metallteile des Computergehäuses, Wasserrohre, etc. anfassen). Es gelten im Grunde die gleichen Verhaltensregeln wie bei PC-Einsteckkarten (Grafikkarten, etc.). Insbesondere ist das relevant bei niedriger Luftfeuchtigkeit (kalte Wintertage) und bei ungünstiger Kombination aus Bekleidung und Fußboden. Die Platine sollte möglichst am Rand, dort wo sich keine Leiterbahnen befinden, angefasst werden.

Die Platine wird über den Mikro USB-Anschluss an der linken Seite mit Spannung versorgt. Die Platine ist somit verpolungssicher. Die Spannung sollte idealerweise stabilisiert 3.3V betragen. Bei höheren Spannungen ist eine zusätzliche externe Spannungsquelle (12V Netzteil) notwendig, die im Praktikum jedoch nicht verwendet wird.

Es empfiehlt sich darauf zu achten, dass keine losen Metallteile (Schlüssel, Geldmünzen, etc.) auf dem Tisch oder in der Nähe der Platine liegen, da es sonst zu Kurzschlüssen kommen kann.

2.5.3 Anschließen der Platine

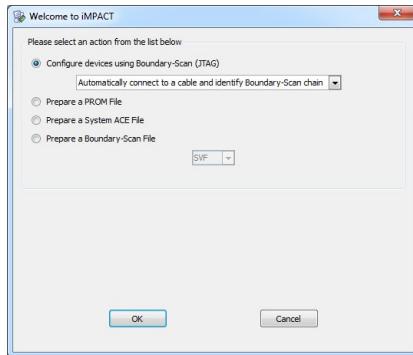
Um einen Bitstream (die Konfigurationsdaten) in den FPGA zu laden, wird die Platine mittels des beiliegenden USB-Kabels mit dem Computers verbunden. Achten Sie dabei darauf, dass die Markierungen (roter Punkt nach oben) übereinstimmen. Vermeiden Sie dabei unnötige Gewalt oder gar höhere Druckeinwirkungen auf den USB-Anschluss der Platine. Anschließend können Sie das Programm ISE iMPACT starten, um das FPGA zu konfigurieren.

2.5.4 Download des Bitstreams

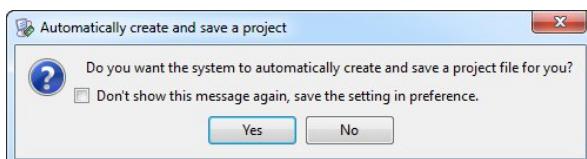
Um aus der VHDL-Quelldatei einen Datenstrom für den FPGA (Bitstream) zu erzeugen, muss die Synthese, das Place&Route und die eigentliche Bitstream-Erzeugung durchlaufen werden (wie oben beschrieben über Generate Programming File). Anschließend kann das Kompilat mittels ISE iMPACT zur Platine übertragen und das FPGA somit konfiguriert werden. Bevor Sie dies tun, verbinden Sie zuvor die Platine wie in Abschnitt 2.5.3 beschrieben mit dem PC.

Für die Benutzung von iMPACT gehen Sie wie folgt vor:

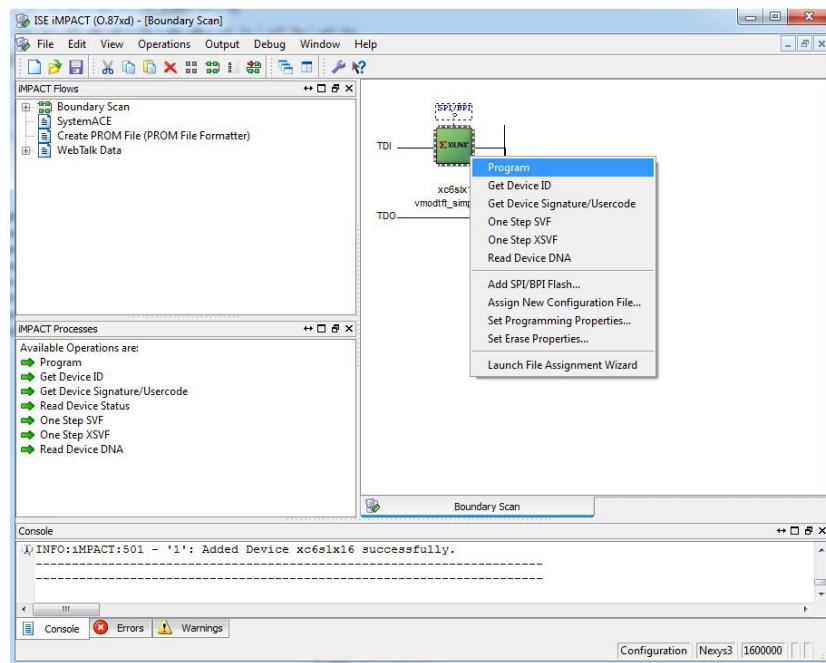
- Öffnen Sie unter **Start (WinIcon Taskleiste u.l.)→Alle Programme →Xilinx ISE Design Suite 13.5→ISE Design Tools→64-bit Tools iMPACT**.
- Der automatische Projekt Wizard öffnet sich. Klicken Sie auf OK um das System automatisch zu konfigurieren.



- Der Auto-Assign-Dialog öffnet sich. Bestätigen Sie auch diesen mit YES. Wählen Sie dann aus dem Windows-Öffnen-Dialog das Verzeichnis ihres ISE Projektes mit dem jeweiligen Bitstream aus, dessen Bezeichner äquivalent zu der Top-Modul Entity ist. Daraufhin erscheint nochmals der Windows-Öffnen-Dialog, klicken Sie hier aber auf Bypass. Dieser Schritt kann jeder Zeit manuell getätigert werden, indem Sie im Hauptfenster auf die Schaltfläche des Zielchips (xc6stx16) mit der rechten Maustaste klicken und **Assign New Configuration File** auswählen.



- Nachdem alle Einstellungen übernommen wurden, klicken Sie mit der rechten Maustaste auf den Zielchip (xc6stx16) und wählen **Program**. Sollte ein Übertragungsfehler auftreten und die Verbindung abgebrochen werden (**Program Failed**), wiederholen Sie diesen letzten Schritt.



Als erster Test der Platine, oder wenn man Zweifel an der Funktionsfähigkeit der Platine hat, sollte man den folgenden Bitstream in den FPGA laden: s3demo.bit. War der Ladevorgang erfolgreich, dann sollten alle 7-Segment-Anzeigen von 1 - 9 durchzählen.

Kapitel 3

Anmerkungen und Tipps

- Im Schematics Editor des ISE Webpack kann die Plane(Zeichenfläche) dynamisch vergößert werden, sofern Sie für ihren Schaltungsentwurf mehr Platz benötigen sollten. Geben Sie hierfür unter **Edit→Change Sheet Size** die gewünschte Größe ein.
- Die D-Flipflops der in Abschnitt 2.3.1 vorgestellten BCD-Zählereinheit sind taktflankengesteuert und reagieren ausschließlich auf fallende Flanken ($1 \rightarrow 0$ Übergang).
- die Nexys FPGA-Platine ist mit einer umfangreichen Peripherie ausgestattet. Für genauere technische Daten und Details nutzen Sie das ausführliche Handbuch, welches Sie in der Zip-Datei finden (Nexys3_Manual.pdf)
- Die Stromversorgung der Platine findet die über das beiliegende Mikro-USB-Kabel statt. Ein zusätzliches Netzgerät ist nicht vonnöten. Vermeiden Sie beim Anschließen des Kabels hektische oder gar ruckartige Zug- und Druckbewegungen.
- Nachdem der Synthese-Prozess eines VHDL-Entwurfs in Webpack vollständig durchlaufen wurde, können wichtige Kerndaten (Taktfrequenz, Gatterlaufzeiten, etc.) der Schaltung aus dem Log-Bericht ausgelesen werden. Den Log-Bericht finden Sie unter **Project→Design/Summary Reports→Synthesis Report**.

Kapitel 4

Vorbereitungsaufgaben

- Aufgabe 1.** Erweitern Sie den in Abschnitt 2.3.1 vorgestellten 4-Bit BCD-Zähler um eine asynchrone Rücksetzfunktion. Sofern eine logische 1 an einem zusätzlichen Reset-Eingang anliegt, soll der Zähler schrittweise auf Null zurückgesetzt werden, d.h. der Rücksetzvorgang dauert mehrere Tackphasen an, bis alle Flipflops wieder den Zustand 0 speichern. Neben Takteingang (CLK) soll der Zähler auch über einen zusätzlichen Output mit der Bezeichnung CLR (Cleared) verfügen. Dieser indiziert eine logische 1 genau dann, wenn der Zähler den Zustand Null speichert. Zeichnen Sie den zugehörigen überarbeiteten Schaltplan. Ihre Modifikationen sollten ausschließlich aus NAND- sowie UND-Gattern (jeweils 2 Eingänge, 1 Ausgang) bestehen.
- Aufgabe 2.** Entwerfen Sie den Schaltplan für einen weiteren 3-Bit asynchronen BCD-Zähler, welcher basierend auf dem in Abschnitt 2.3.1 vorgestellten Konzept eine Ziffernfolge von 0 bis 5 durchläuft (Modulo 6). Der Zähler soll ausschließlich aus D-Flipflops und UND-/NAND-Gattern aufgebaut sein. Stellen Sie hierfür zunächst die Zustands-Übergangstabelle auf und leiten Sie anhand dieser die Abhängigkeiten für die Dateneingänge der Flipflops ab. Geben Sie dabei für jedes Flipflop das zugehörige KV-Diagramm an. Berücksichtigen Sie ebenfalls ein asynchrones Rücksetzen des Zählers über einen zusätzlichen Eingang sowie einen Ausgang CLR aus Aufgabe 1.
- Aufgabe 3.** Entwerfen Sie ein asynchrones Zählwerk für eine digitale Stoppuhr. Die Stoppuhr besteht aus einer Anzeige für 6 Ziffern. Das zugehörige Zählwerk soll aus sechs kaskadierten BCD-Zählern aufgebaut sein, um eine exakte Zeitmessung in Minuten, Sekunden, 10tel und 100stel Sekunden zu ermöglichen. Verwenden Sie für die Darstellung von 10tel/100stel Sekunden ausschließlich zwei gekoppelte BCD-Zähler aus Aufgabe 1 und für die Minuten/Sekunden je einen BCD-Zähler aus Aufgabe 1 und Aufgabe 2. Zeichnen Sie den zugehörigen Schaltplan. Nehmen Sie dabei jeden der Zähler als ein eigenes Gatter mit dem folgendem Schaltsymbol an.

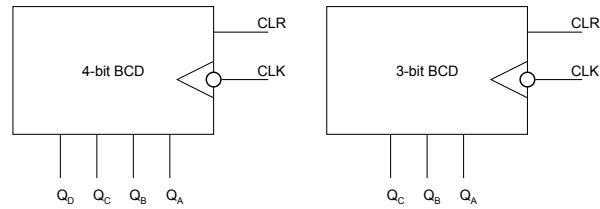


Abbildung 4.1: Schaltsymbol für einen 3 bzw. 4-bit BCD_Zähler

Aufgabe 4. Überlegen Sie sich, welche Probleme beim Rücksetzen des aus Aufgabe 3 implementierten Zählwerks auftreten können und wie sich diese eventuell beheben lassen.

Kapitel 5

Praktikumsaufgaben

- Aufgabe 1.** Implementieren Sie eine VHDL-Verhaltensbeschreibung für ein taktflankengesteuertes D-Flipflop. Erstellen Sie eine Entity D_Flipflop mit den Eingängen D, CLK und dem Ausgang Q sowie \bar{Q} , und modellieren Sie das Verhalten des D-Flipflops mittels if/then-Statements. Achten Sie bei der Beschreibung darauf, dass das FlipFlop **ausschließlich auf fallende Flanken ($1 \rightarrow 0$ Übergang)** reagiert. Testen Sie anschließend ihr Flipflop unter Berücksichtigung aller Eingabebelegungen im Simulator.
- Aufgabe 2.** Implementieren Sie den modifizierten 4-Bit BCD-Zähler aus Vorbereitungsaufgabe 1 mittels Schemata in VHDL. Für die UND-/NAND-Gatter können Sie die bereits vorhandenen Entwürfe der vorangegangenen Versuche verwenden. Denken Sie daran, für jede der Entitäten vorab das Schaltsymbol zu erzeugen, bevor Sie mit der Vernetzung des Zählers beginnen. Testen Sie anschließend ihren Zähler unter Berücksichtigung aller Eingabebelegungen.
- Aufgabe 3.** Implementieren sie den 3-Bit BCD-Zähler aus der Vorbereitungsaufgabe mittels Schemata. Überprüfen Sie beim Testen die Modulo 6 Operation sowie das korrekte Rücksetzverhalten des Zählers unter Berücksichtigung aller Eingabebelegungen.
- Aufgabe 4.** Implementieren Sie das Zählwerk einer digitalen Stoppuhr basierend auf dem Schaltplan aus Vorbereitungsaufgabe 3 mittels Schemata. Kaskadieren Sie die 3- und 4-Bit BCD-Zähler in der entsprechenden sequentiellen Reihenfolge. Das Zählwerk soll über einen Reset- sowie Takteingang verfügen. Als Ausgabe dienen die Ziffern der einzelnen Zähler, wobei jede Ziffer durch einen separaten 3- bzw. 4-Bit breiten Ausgabevektor zusammengefasst wird. D.h., insgesamt gibt es 6 Ausgabevektoren. Testen Sie abschließend das Zählwerk im Simulator und analysieren Sie graphisch, wieviel Taktzyklen für das Rücksetzen im Worst-Case-Fall benötigt werden.
- Aufgabe 5.** Erweitern Sie nun das Zählwerk zu einem vollständigen Stoppuhr-Schaltkreis. Laden Sie hierzu die Zip-Datei mit den noch fehlenden Komponenten für die Ansteuerung der 7-Segment Anzeige der Platine von der Website herunter. Dieses umfasst einen Taktgeber (CLK_CTRL) sowie eine Steuereinheit(SEG_CTRL). Verknüpfen Sie Ihr Zählwerk mit den zusätzlichen Einheiten gemäß dem Schema in Abbildung 5.1.
Die Stoppuhr besitzt neben Takt-(CLK_100MHz) und Reseteingang (RESET) noch zwei zusätzliche Steuereingänge CTRL und HOLD um auf spätere Schalttereingaben der Platine zu reagieren. Als Ausgang dient ein 12-bit breiter Ausgabevektor der Steuerein-

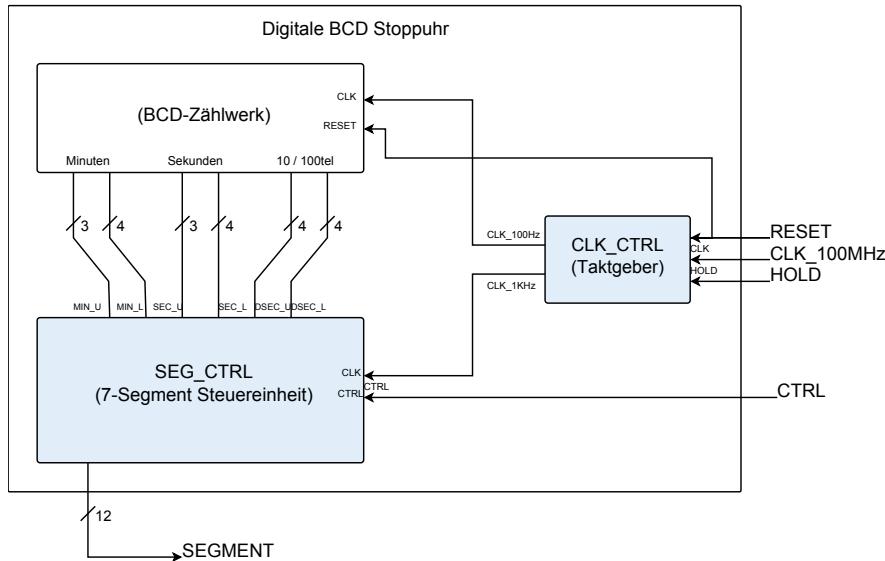


Abbildung 5.1: Verknüpfung des Zählwerks mit der Anzeige der Stoppuhr

heit(SEG_CTRL), wobei die Elemente SEGMENT[0:7] den einzelnen Segmenten A-P und SEGMENT[8:11] die Ziffern AN0-AN3 der Anzeige repräsentieren.

- Aufgabe 6.** Testen Sie die Stoppuhr nun auf dem FPGA. Bevor Ihr Entwurf aber letztlich auf der Platine getestet werden kann, muss das Routing zwischen den Pins des FPGAs und den VHDL-Signalen der Stoppuhr angegeben werden. Legen Sie hierfür eigens eine UCF-Datei im Projekt an. Achten Sie beim Implementieren der Datei auf die in Abschnitt 2.5.1 gegebenen Hinweise und routen Sie alle In-/Outputs ihrer Stoppuhr auf die entsprechenden Pins des FPGAs, die in Tabelle 5.1 festgehalten sind.

Nachdem Sie das Routing abgeschlossen haben, müssen alle aufgeführten Synthesese Schritte im Process-Fenster bis einschließlich **Generate Programming File** durchlaufen werden. Anschließend können Sie das fertige Kompilat des Entwurfs mittels iMPACT auf die Platine downloaden. Ermitteln Sie die maximale Taktfrequenz ihrer Stoppuhr-Schaltung. Diese ist im Synthesebericht unter dem Punkt Clocking-Frequency vermerkt.

	VHDL Signal	FPGA Pin
Eingänge		
1	CLK_100MHz	V10
2	CTRL	T10
3	HOLD	T9
4	RESET	V9
Ausgänge		
1	SEGMENT(0)	T17
2	SEGMENT(1)	T18
3	SEGMENT(2)	U17
4	SEGMENT(3)	U18
5	SEGMENT(4)	M14
6	SEGMENT(5)	N14
7	SEGMENT(6)	L14
8	SEGMENT(7)	M13
9	SEGMENT(8)	N16
10	SEGMENT(9)	N15
11	SEGMENT(10)	P18
12	SEGMENT(11)	P17

Tabelle 5.1: Spartan 6 FPGA-Pin Belegung