**BTE5024-Digital – Projekt**

**Entwicklung einer DCF77-Funkuhr**



**Zweck:** Dokumentation

**Institution:** Berner Fachhochschule, Technik und Informatik

**Betreuer:** Mähne Thorsten, Dozent BFH

**Autoren:** Fabian Rihs

Daniel Zwygart

**Datum:** Burgdorf, Juni 2015

**Inhaltsverzeichnis**

[1 Einleitung 2](#_Toc421469466)

[2 Konzeptionierung 3](#_Toc421469467)

[3 Das DCF77-Signal 4](#_Toc421469468)

[4 Beschreibung der Module 6](#_Toc421469469)

[4.1 Topschema 6](#_Toc421469470)

[4.1.1 DCF77-Decoder 9](#_Toc421469471)

[4.1.2 Clock-Modul 14](#_Toc421469472)

[4.1.2.1 Counter und Counter3\_9 16](#_Toc421469473)

[4.1.3 Divider 18](#_Toc421469474)

[4.1.4 Display-Modul 21](#_Toc421469475)

[4.1.4.1 Multiplexer 24](#_Toc421469476)

[4.1.4.2 BCD2SEG 26](#_Toc421469477)

[4.1.4.3 Exnor 27](#_Toc421469478)

[5 Synthesereport 28](#_Toc421469479)

[6 Erkenntnisse 29](#_Toc421469480)

[7 Fazit 29](#_Toc421469481)

[8 Abbildungsverzeichnis 30](#_Toc421469482)

# Einleitung

Im Rahmen der Projektarbeit im Modul BTE5024-Digital wurde eine Funkuhr entwickelt, die das Zeitsignal des DCF77-Senders auswertet und es zum Stellen der internen Quarzuhr nutzt. Weiter kann über einen Schalter zwischen der Darstellung der Zeit und des Datums auf dem LCD gewechselt werden. Zur Realisierung standen 16 Lektionen im Unterricht zur Verfügung.

Um die Uhr so zu realisieren, dass sie im Alltag auch nutzbar wäre, wurde viel Wert auf das Empfangen und Verarbeiten des DCF-Signales gelegt. Durch Anwendung von Wissen aus der Signalverarbeitung konnte das System optimiert werden, so dass auch Signale mit bis zu 10% Falschpegeln pro Bit verarbeitet werden können. Dies wird mit einer Mittelwertbildung über das abgetastete Signal erreicht. Weiter soll die Anzeige der Uhr beim Betrachter einen professionellen Eindruck hinterlassen. Daher wurden die Punkte zur Trennung der Ziffern verwendet und speziell bei der Zeitanzeige blinken die Punkte mit der Sekundenanzeige. Für interessierte User sowie zum optischen überprüfen des Empfanges wurde auf der Rückseite noch das DCF-Signal auf einer LED visualisiert.

Die folgende Dokumentation zeigt weitere Einzelheiten der Planung sowie der Umsetzung und der anschliessenden Tests. Es wird auf jedes verwendete Modul eingegangen und zum Schluss unsere Erkenntnisse aus der Arbeit präsentiert.

# Konzeptionierung

Antenne

DIP-Switch

Clock-Modul

DCF77-Decoder

Display-

Modul

LED

Display

Minute

Stunde

Jahr

Monat

Tag

Auswahl Uhrzeit/Datum

Sync

Clock-Divider

Abbildung 1 Grobschema

Um das Projekt möglichst effizient zu bearbeiten, stellten wir das Konzept am Anfang gemeinsam auf. Danach wurden die einzelnen Funktionsblöcke aufgeteilt, um die Zeit optimal nutzen zu können. Abbildung 1 zeigt das Ergebnis unseres Grobkonzeptes. Von der Antenne wird das DCF-Signal zum Decoder geführt wo es dekodiert wird. Um feststellen zu können, ob überhaupt ein Signal empfangen wird, stellen wir das Signal auf einer LED dar. Nachdem ein Protokoll empfangen und die Parität überprüft wurde, wird die Uhr mit einen Sync-Impuls geladen. Mit einem DIP-Schalter kann ausgewählt werden, ob das Datum oder die Uhrzeit angezeigt werden soll. Nachdem alle Module beschrieben waren, fügten wir die Module gemeinsam zusammen. Auch die gefundenen Fehler oder Probleme lösten wir gemeinsam.

Aufteilung der Module:  
Daniel Zwygart: DCF77-Decoder, Uhr  
Fabian Rihs: Display-Modul, DCF77-Decoder Testbench, Clock-Divider

# Das DCF77-Signal

Der Zeitzeichensender DCF77 ist ein Langwellensender in Mainflingen bei Frankfurt am Main, der die meisten funkgesteuerten Uhren im westlichen Europa mit der Normalzeit versorgt. Seine im Sekundenrhythmus gesendeten Zeitzeichen übertragen die mitteleuropäische Zeit bzw. mitteleuropäische Sommerzeit. Der Sender in Mainflingen arbeitet auf der Frequenz 77,5kHz mit einer Leistung von 50kW. Das DCF77-Signal ist abhängig von der Tages- und Jahreszeit bis zu einer Entfernung von etwa 2000km zu empfangen. Die Zeitinformationen werden als digitales Signal zusätzlich zur Normalfrequenz (der Trägerfrequenz des Senders, also 77,5kHz) übertragen. Das geschieht durch negative Modulation des Signals (Absenken der Trägeramplitude auf etwa 15%) im Sekundentakt. Der Beginn der Absenkung liegt jeweils auf dem Beginn der Sekunden 0 bis 58 innerhalb einer Minute. In der letzten Sekunde erfolgt keine Absenkung, wodurch die nachfolgende Sekundenmarke den Beginn einer Minute kennzeichnet und der Empfänger synchronisiert werden kann. Die Länge der Amplitudenabsenkungen am Beginn der Sekunden steht jeweils für den Wert eines binären Zeichens: 100ms Absenkung stehen für den Wert „0“, 200ms für „1“. Damit stehen innerhalb einer Minute 59 Bit für Informationen zur Verfügung.

|  |  |  |
| --- | --- | --- |
| **Bit** | **Bedeutung** | |
| 0 | Start einer neuen Minute (ist immer „0“) | |
| 1-14 | Wetterinformationen der Firma MeteoTime sowie Informationen des Katastrophenschutzes | |
| 15 | Rufbit | |
| 16 | „1“: Am Ende dieser Stunde wird MEZ/MESZ umgestellt. | |
| 17 | „0“: MEZ, „1“: MESZ | |
| 18 | „0“: MESZ, „1“: MEZ | |
| 19 | „1“: Am Ende dieser Stunde wird eine Schaltsekunde eingefügt. | |
| 20 | Beginn der Zeitinformation (ist immer „1“) | |
| 21 | Minute (Einer) | Bit für 1 |
| 22 | Bit für 2 |
| 23 | Bit für 4 |
| 24 | Bit für 8 |
| 25 | Minute (Zehner) | Bit für 10 |
| 26 | Bit für 20 |
| 27 | Bit für 40 |
| 28 | Parität Minute | |
| 29 | Stunde (Einer) | Bit für 1 |
| 30 | Bit für 2 |
| 31 | Bit für 4 |
| 32 | Bit für 8 |
| 33 | Stunde (Zehner) | Bit für 10 |
| 34 | Bit für 20 |
| 35 | Parität Stunde | |

|  |  |  |
| --- | --- | --- |
| 36 | Kalendertag (Einer) | Bit für 1 |
| 37 | Bit für 2 |
| 38 | Bit für 4 |
| 39 | Bit für 8 |
| 40 | Kalendertag (Zehner) | Bit für 10 |
| 41 | Bit für 20 |
| 42 | Wochentag | Bit für 1 |
| 43 | Bit für 2 |
| 44 | Bit für 4 |
| 45 | Monatsnummer (Einer) | Bit für 1 |
| 46 | Bit für 2 |
| 47 | Bit für 4 |
| 48 | Bit für 8 |
| 49 | Monatsnummer (Zehner) | Bit für 10 |
| 50 | Jahr (Einer) | Bit für 1 |
| 51 | Bit für 2 |
| 52 | Bit für 4 |
| 53 | Bit für 8 |
| 54 | Jahr (Zehner) | Bit für 10 |
| 55 | Bit für 20 |
| 56 | Bit für 40 |
| 57 | Bit für 80 |
| 58 | Parität Datum | |
| 59 | keine Sekundenmarke | |

Quelle: http://de.wikipedia.org/wiki/DCF77

# Beschreibung der Module

## Topschema

Das Top-Schema besteht aus vier Teilmodulen. Zusammen verarbeiten sie ein DCF77-Signal und stellen die Zeit sowie das Datum an einem 7-Segment-LCD dar.

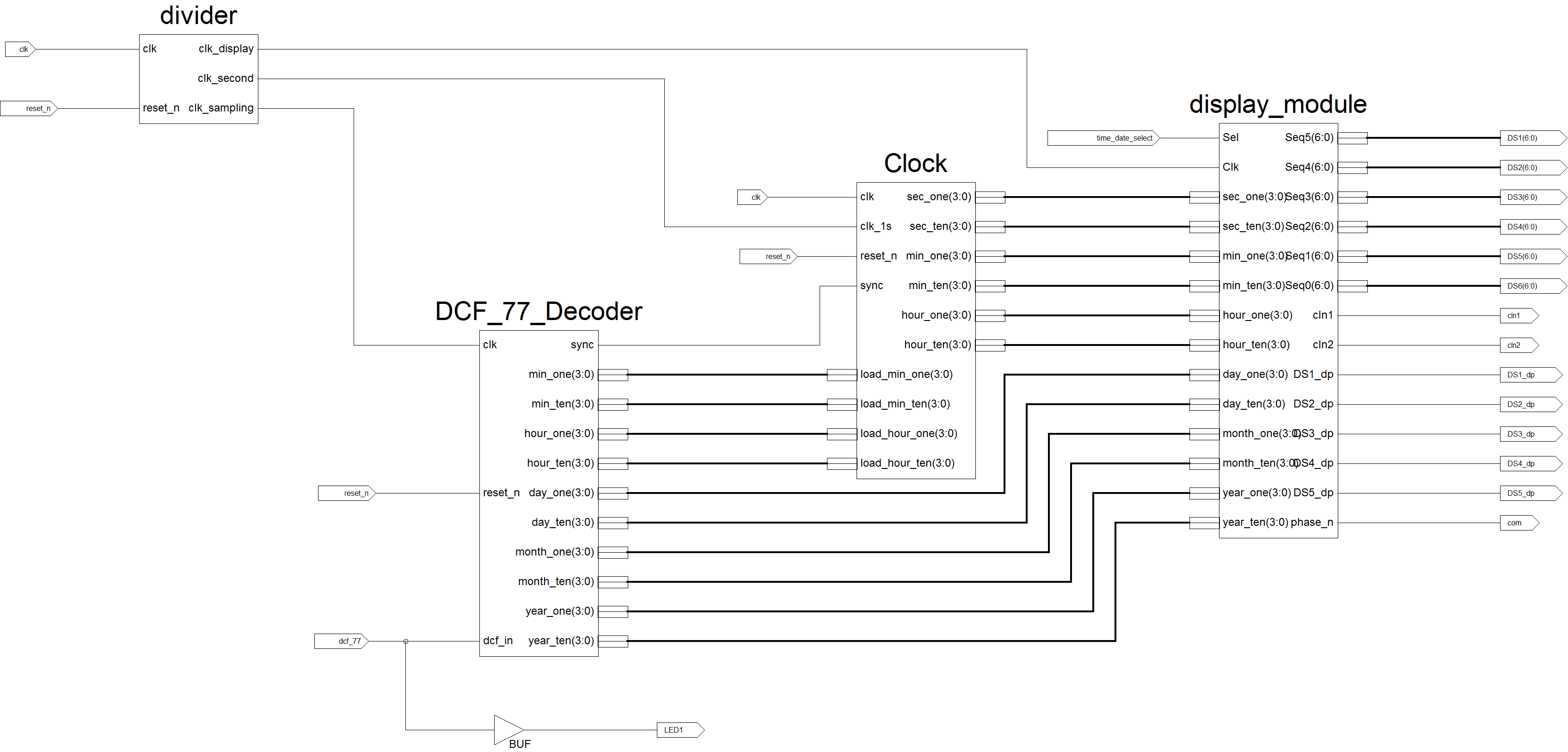


Abbildung 2 Topschema

**Inputs:**

|  |  |
| --- | --- |
| clk : in STD\_LOGIC; | Takt (Duty-Cycle 50%) 32.768kHz |
| reset\_n: in STD\_LOGIC; | Synchroner Reset |
| time\_data\_select | Auswahl Zeit oder Datumsanzeige |
| dcf\_77 | DCF77-Signal |

**Outputs:**

|  |  |
| --- | --- |
| DS1(6:0) bis DS6(6:0) | LCD Segmente (Datum und Zeit) |
| cln1, cln2 | LCD Doppelpunkte (Blinken im 0.5Hz Takt) |
| dS1\_dp bis DS5\_dp | LCD Punkte unten (Trennpunkte bei Datum) |
| com | LCD Takt |
| LED1 | LED zur Darstellung des DCF-Signals |

**Beschreibung**

Auf der Eingangsseite stehen ein 32.768kHz Takt und ein DCF-Signal sowie ein Taster zur Verfügung. Aus den 32.768kHz wird zuerst ein 128Hz Sampletakt ein 32Hz Displaytakt und ein 1Hz Sekundentakt generiert. Mit dem 128Hz Sampletakt wird das DCF-Signal eingelesen. Sobald ein korrektes Telegramm eingelesen wurde wird ein sync ausgelöst mit welchem das Clock-Modul aufgefordert wird die angelegte Zeit zu übernehmen. Die Uhr zählt dann im 24h Format von diesem Wert aus weiter. Können die Telegramme immer erfolgreich empfangen werden wird die Uhr nach jeder Minute neu gestellt. Das Display-Modul entscheidet dann anhand der Schalterstellung ob es das Datum oder die Zeit darstellen soll. Dazu wird der BCD-Code in 7-Segment-Code umgewandelt und anschliessend in Phase oder in Gegenphase zu com an das Display angelegt. Dies ist nötig, da das LCD keine DC-Spannungen an den Eingängen verträgt.

**Testbench**

Die Testbench des Top-Schema nutzt die gleiche Funktion wie die Testbench des DCF-Decoders um ein Telegramm zu versenden. Getestet wurde in erster Linie die Synchronisierung des kompletten Systems. Dabei ist die korrekte Übergabe der Zeit vom Decoder zur Uhr der entscheidendste Punkt.

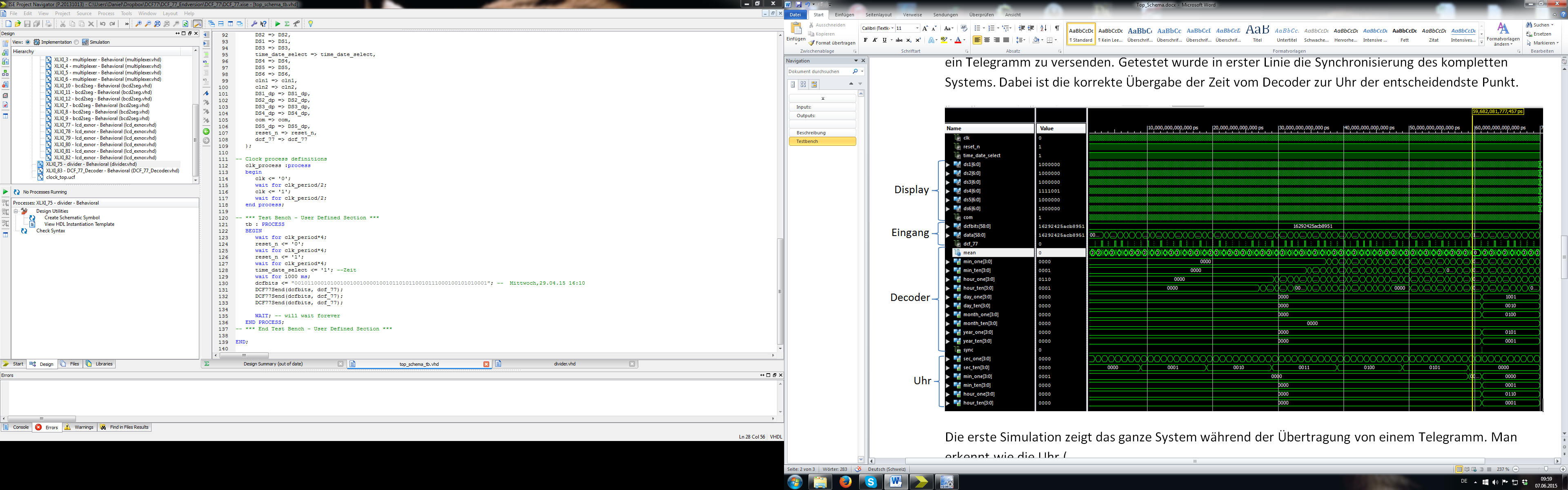


Abbildung 3 Testbench Topschema: Einlesen des Telegramms

Die erste Simulation zeigt das ganze System während der Übertragung von einem Telegramm. Man erkennt wie die Uhr bereits zählt und nach 60s gestellt wird. Der Vergleich von dcfbits und data zeigt, dass in data der richtige Wert eingelesen wurde. Um zu erkennen ob das Display auch aktualisiert wird folgt ein Zoom in den Bereich der 60sten Sekunde.

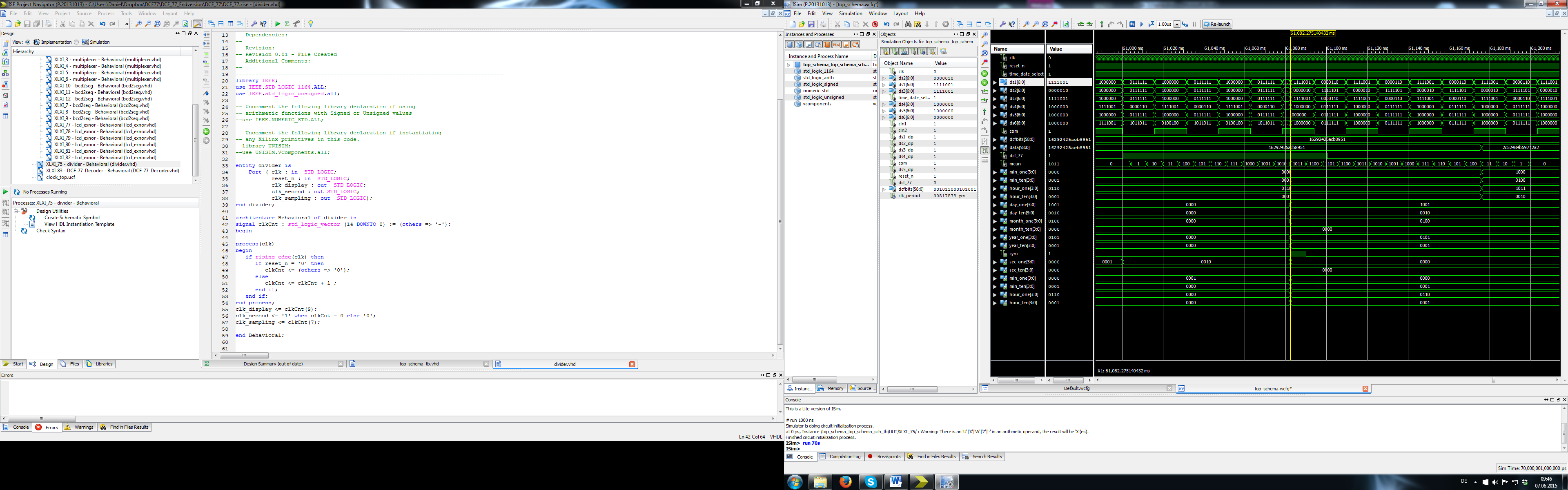


Abbildung 4 Testbench Topschema: Synchronisation

Sobald sync high wird erkennt man, dass das Display seine Anzeige ändert. Somit funktioniert auch hier die Synchronisation. Auf den Wert der jeweiligen Bits der Segmente wird hier nicht getestet, da das Display-Modul selber getestet wurde und somit davon ausgegangen wird, dass dieses Funktioniert. Beim Signal mean erkennt man auch die Abtastung des DCF-Signals wodurch der Wert in mean zuerst Steigt und anschliessend wieder fällt, je nach Zustand des DCF-Signals.

### DCF77-Decoder

Dieses Modul liest ein DCF77-Telegramm während einer Minute ein und extrahiert daraus die aktuelle Zeit und das aktuelle Datum. Sobald ein Telegramm empfangen wurde wird ein sync ausgegeben.

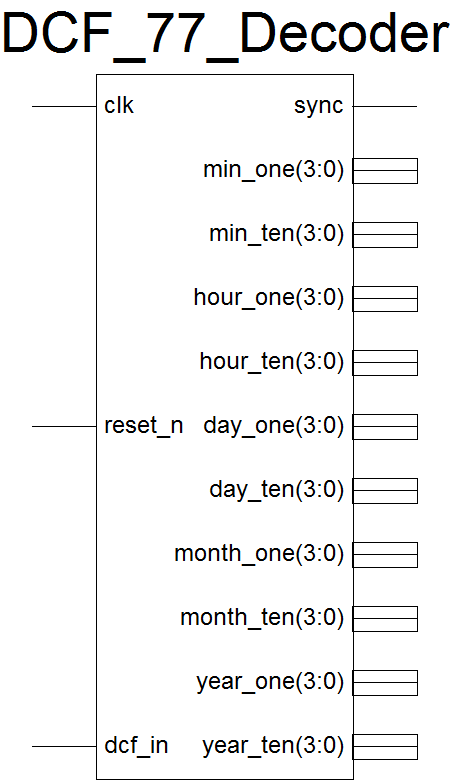


Abbildung 5 Symbol DCF77-Decoder

**Inputs:**

|  |  |
| --- | --- |
| clk : in STD\_LOGIC; | Abtastfrequenz (Duty-Cycle 50%) |
| reset\_n: in STD\_LOGIC; | Synchroner Reset |
| dcf\_in: in STD\_LOGIC; | DCF-Signal |

**Outputs:**

|  |  |
| --- | --- |
| sync : out STD\_LOGIC; | Zeit Synchronisierungsignal: ‘1‘ während einem Takt wenn Zeit an Ausgängen gültig |
| min\_one : out STD\_LOGIC\_VECTOR (3 downto 0); | Binäre Einerstelle der Minute nur gültig wenn sync=‘1‘ |
| min\_ten : out STD\_LOGIC\_VECTOR (3 downto 0); | Binäre Zehnerstelle der Minute nur gültig wenn sync=‘1‘ |
| hour\_one : out STD\_LOGIC\_VECTOR (3 downto 0); | Binäre Einerstelle der Stunde nur gültig wenn sync=‘1‘ |
| hour\_ten : out STD\_LOGIC\_VECTOR (3 downto 0); | Binäre Zehnerstelle der Stunde nur gültig wenn sync=‘1‘ |
| day\_one : out STD\_LOGIC\_VECTOR (3 downto 0); | Binäre Einerstelle des Tages immer gültig ab erstem korrektem Telegramm |
| day\_ten : out STD\_LOGIC\_VECTOR (3 downto 0); | Binäre Zehnerstelle des Tages immer gültig ab erstem korrektem Telegramm |
| month\_one : out STD\_LOGIC\_VECTOR (3 downto 0); | Binäre Einerstelle des Monats immer gültig ab erstem korrektem Telegramm |
| month\_ten : out STD\_LOGIC\_VECTOR (3 downto 0); | Binäre Zehnerstelle des Monats immer gültig ab erstem korrektem Telegramm |
| year\_one : out STD\_LOGIC\_VECTOR (3 downto 0); | Binäre Einerstelle des Jahres immer gültig ab erstem korrektem Telegramm |
| year\_ten : out STD\_LOGIC\_VECTOR (3 downto 0)); | Binäre Zehnerstelle des Jahres immer gültig ab erstem korrektem Telegramm |

**Generic**

|  |  |
| --- | --- |
| fa : integer := 128; | Abtastfrequenz des DCF-Signal in Hz |
| one\_max : integer := 230; | Obere Schranke für eine 1 im Signal in ms |
| one\_min : integer := 170; | Untere Schranke für eine 1 im Signal in ms |
| zero\_max : integer := 130; | Obere Schranke für eine 0 im Signal in ms |
| zero\_min : integer := 80; | Untere Schranke für eine 0 im Signal in ms |
| nothing\_trigger : integer := 40; | Max mean Wert welcher noch als kein stabiles Signal gedeutet wird in ms |
| zero\_time\_to\_sync: integer := 1300); | Minimale Pausenzeit nach Telegramm in ms |

**Beschreibung**

Der Kern des DCF77-Decoders ist die Unterscheidung zwischen 1 und 0 des DCF-Signals und das korrekte Synchronisieren auf die neue Zeit. Bei der Konzeptionierung dieses Moduls wurde von einer schlechten Signalqualität ausgegangen, da die effektive Signalqualität nicht bekannt war. Es wurde ein Ansatz aus der Signalverarbeitung umgesetzt welcher auf dem Bilden des Mittelwerts basiert. Dabei wird das Eingangssignal gewichtet und aufsummiert. Der Maximalwert welcher durch eine lange high Zeit des DCF-Signals erreicht wird lässt darauf zurückschliessen wie lange das Signal high war. So kann zwischen 100ms und 200ms high welches einem 0 bzw. 1 im Telegramm entspricht unterschieden werden. Es folgt die exakte Beschreibung des Algorithmus sowie die dazu gehörige Simulation bzw. Visualisierung und Tests des Algorithmus in MATLAB. Es ist hilfreich die MATLAB-Simulation zu betrachten um die folgenden Beschreibung des Ablaufs nach zu vollziehen.

Gewichtung Eingangssignal: logisch 1 = 1, logisch 0 = -1  
mean Zähler für den momentanen Mittelwert  
zero\_time Zähler für die Pausenzeit zwischen zwei Telegrammen

Bei jedem Taktzyklus wird der Eingangszustand eingelesen und anhand der Gewichtung dem momentanen Wert von mean hinzuaddiert. Umso länger das DCF-Signal logisch 1 bleibt umso höher wird also der Zählerstand von mean.

Es gibt 5 verschiedene Levels für mean welche das Verhalten des restlichen Ablaufes beeinflussen:  
one\_max, one\_min Wertebereich für ein eingelesenes 1 des Telegramms.  
zero\_max, zero\_min Wertebereich für ein eingelesenes 0 des Telegramms.  
nothing\_trigger Minimalwert damit von einem stabilen logischen 1 des DCF-Signales ausgegangen werden kann.  
zero\_time\_to\_sync Zählerwert für zero\_time welcher erreicht werden muss damit sync ausgelöst wird.

In der Variable max\_holder wird der Maximal von mean gespeichert. Unterschreitet mean den nothing\_trigger Wert wird max\_holder anhand von one bzw. zero\_max, min ausgewertet und anschliessend gelöscht.

Solange mean unterhalb von nothing\_trigger ist wird der Zähler zero\_time inkrementiert und so die vergangene 0 Zeit gemessen. Überschreitet mean anschliessend wieder zero\_min wird der Zählerstand ausgewertet und anschliessend wieder gelöscht. Ist zero\_time>= zero\_time\_to\_sync

wird die Checksumme des Telegramms überprüft und falls diese auch stimmt ein sync ausgegeben. Zudem wird das Datum in den Ausgangsbuffer gespeichert.

**Simulation des Algorithmus in MATLAB**

Um den Algorithmus zu testen wurde er zuerst in MATLAB umgesetzt. Die folgende Abbildung zeigt den Verlauf der Zähler während eines Ausschnittes der Übertragung. Der MATLAB-Code befindet sich im Dateianhang zu diesem Dokument.

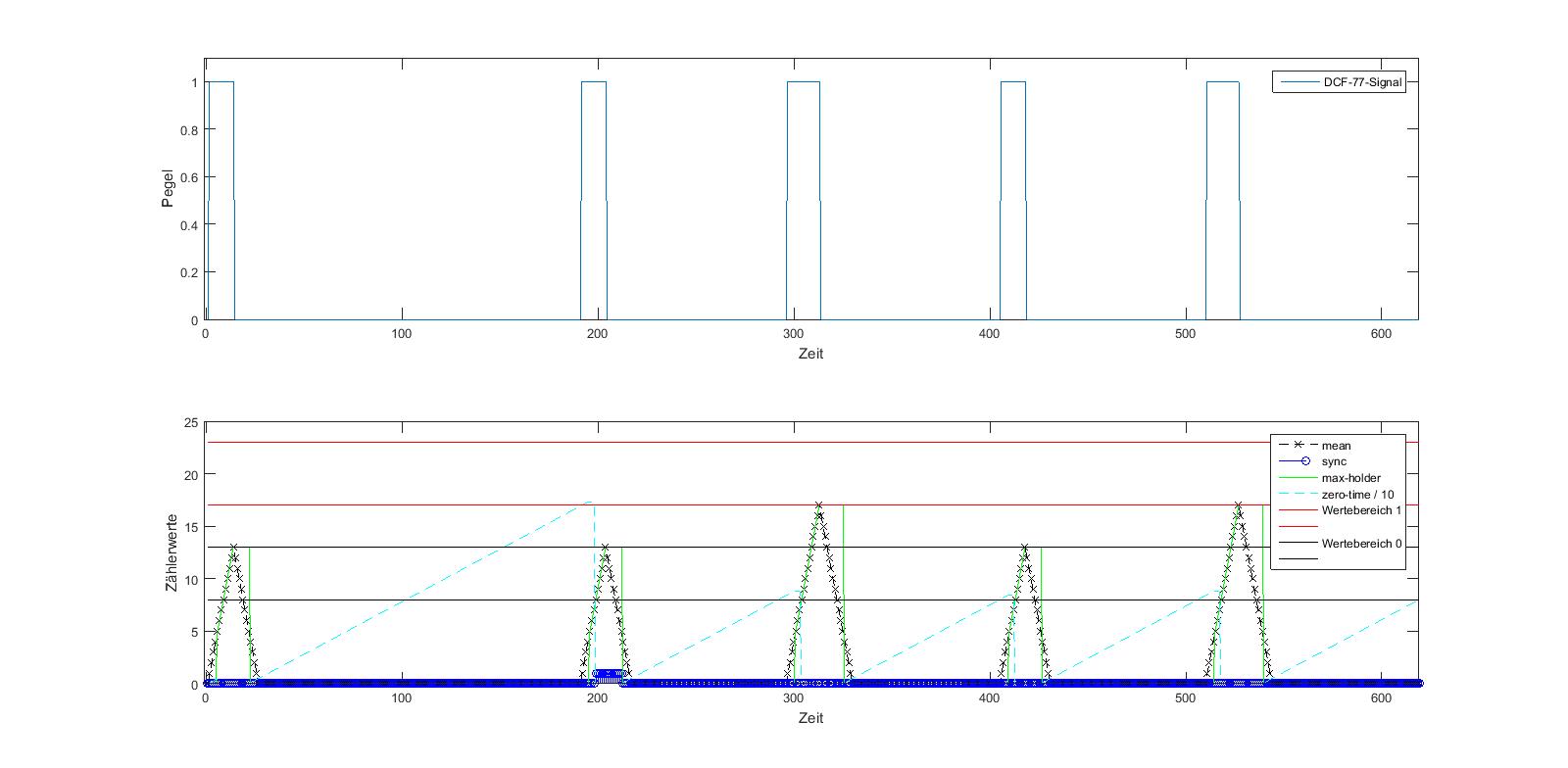


Abbildung 6 Simulation mit MATLAB

**Sync Verzögerung und Einflüsse Signalqualität**

Da das sync erst nach dem Überschreiten des Minimalwertes für eine 1 (one\_min) ausgelöst wird hat das sync eine Verzögerung von 80ms. Hinzu kommt noch ein weiterer Fehler durch die nicht sehr hohe Abtastrate von 128Hz (7,8125ms), welche aus Platzgründen auf ein vielfaches von 2 gelegt werden musste. Die maximale Verzögerung kann somit nicht grösser als 87,8125ms sein sofern das Signal in dieser Zeit stabil auf 1 bleibt. Möchte man das sync optimieren würde die Möglichkeit bestehen eine Uhr mit ms-Zähler zu bauen welche man dann nicht auf 0 sondern auf 80ms reseten würde. Beim verwendeten CPLD ist dies jedoch aus Platzgründen nicht möglich. Optisch ist der Fehler nicht Erkennbar da zum Zeitpunkt des sync die Uhr bereits 0 anzeigt und somit kein Flackern der Sekundenanzeige entsteht.

Durch das Bilden des Mittelwertes hat dieses Modul einen entscheidenden Vorteil gegenüber der herkömmlichen Auswertung mit Flankentriggerung. Die Anforderungen an das Signal können durch die hier verwendete Methode stark gesenkt werden. Somit kann auch an schlechten Standorten schnell ein korrektes Telegramm empfangen werden. Durch die gewählten Schranken von 170-230ms für eine 1 und 80-130ms für eine 0 können 20 bis 30ms des Pulses unstabil sein ohne das Einlesen zu beeinflussen. Weiter wird das sync nicht fälschlicherweise zu früh ausgelöst wenn ein einzelner Störimpuls empfangen wird. Pulse von bis zu 40ms werden komplett ignoriert und erst bei einem Impuls von 80ms wird das sync ausgelöst.

Ein schlechtes Signal kann das sync zusätzlich um mehrere 10ms verzögern im Gegenzug gewinnt man ein sehr stabiles System welches selbst mit einer sehr schlechten Signalqualität zurechtkommt. Sollte trotzdem ein Fehler eingelesen werden gibt es als weitere Sicherheit noch die Checksumme womit das Signal dann falls nötig verworfen werden kann.

**Testbench**

Um den Decoder zu Testen wurde eine Testbench geschrieben, welcher man ein Testprotokoll übergeben kann, welches dann als dcf\_in Signal am Modul angelegt wird. In dieses Telegramm lassen sich ohne weitere Aufwände Fehler einbauen. So kann die Checksummeprüfung und die Extrahierung von Datum und Zeit getestet werden.

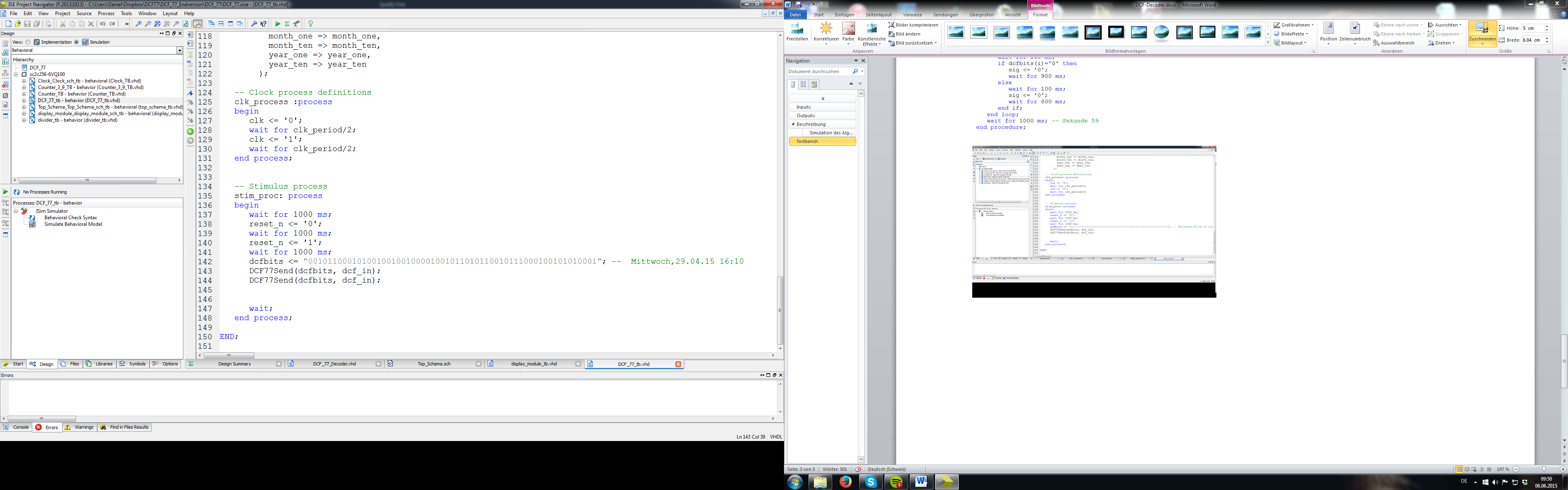


Abbildung 7 Code Testbench DCF77-Modul

Um das Telegramm in das DCF-Format zu überführen wurde eine Funktion definiert. Dieser kann die Signalleitung und das zu versendende Telegramm übergeben werden. Die Funktion generiert 100ms und 200ms Impulse je nach Zustand des jeweiligen Bits im Telegramm. Danach wird 900ms bzw. 800ms gewartet. Dies wird für alle Bits im Telegramm wiederholt und am Schluss wird eine weitere Sekunde gewartet um den Schluss des Telegramms zu simulieren.

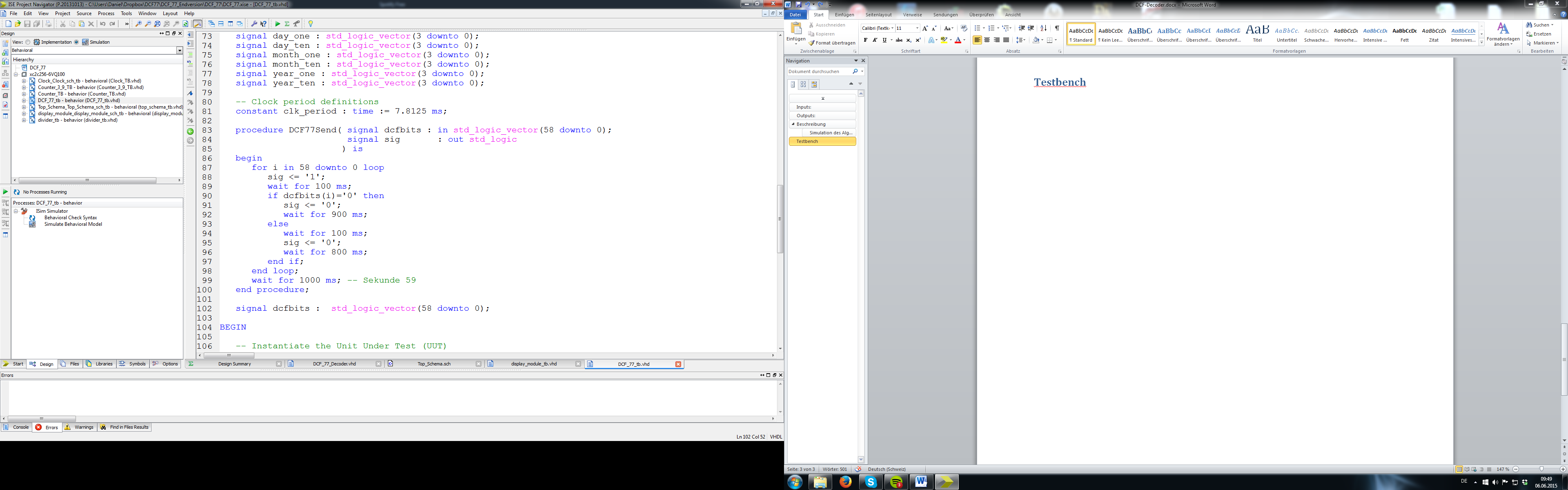


Abbildung 8 Code Testbench: Funktion zum Simulieren von Daten

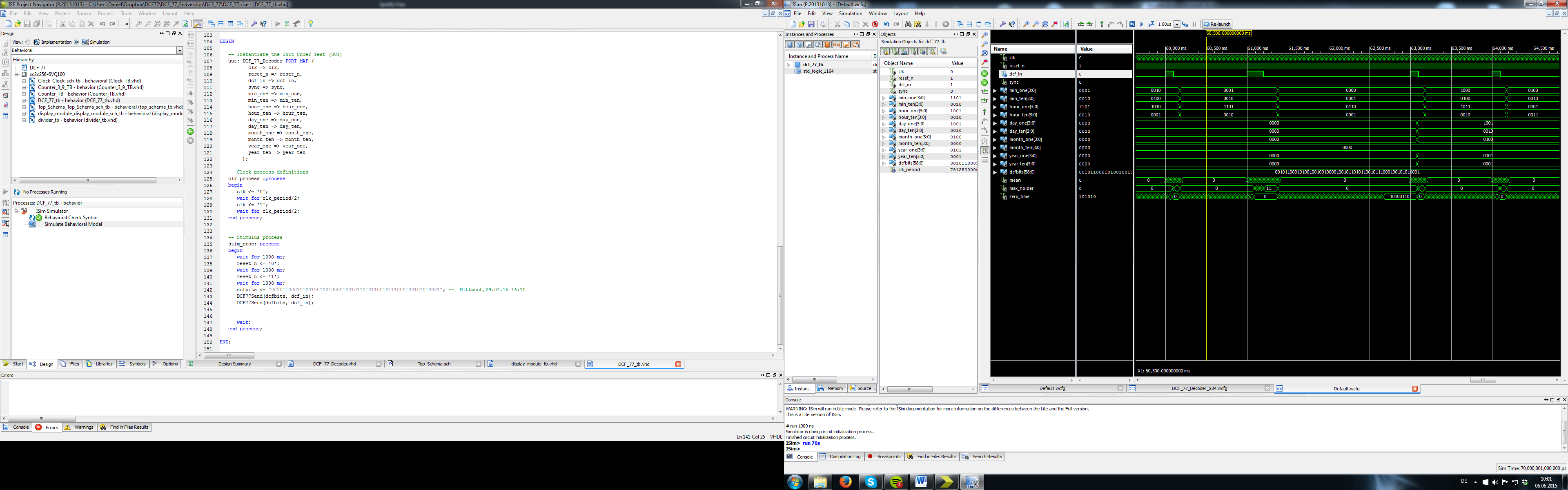


Abbildung 9 Testbench DCF77-Decoder

Der Telegrammausschnitt zeigt die zwei letzten und zwei ersten Bits einer Übertragung. Man erkennt wie mean während dcf\_in auf high ist hochzählt und wieder runter sobald dcf\_in auf low ist. Max\_holder behält den Maximalwert und sobald mean den nothing\_trigger unterschreitet wird das Bit gelesen. In der langen Pause zwischen dem letzten und ersten Bit zählt zero\_time bis zu seinem Maximalwert und bleibt dort stehen, bis mean wieder den zero\_min Wert überschreitet. Zu diesem Zeitpunkt wird auch das sync ausgelöst.

### Clock-Modul

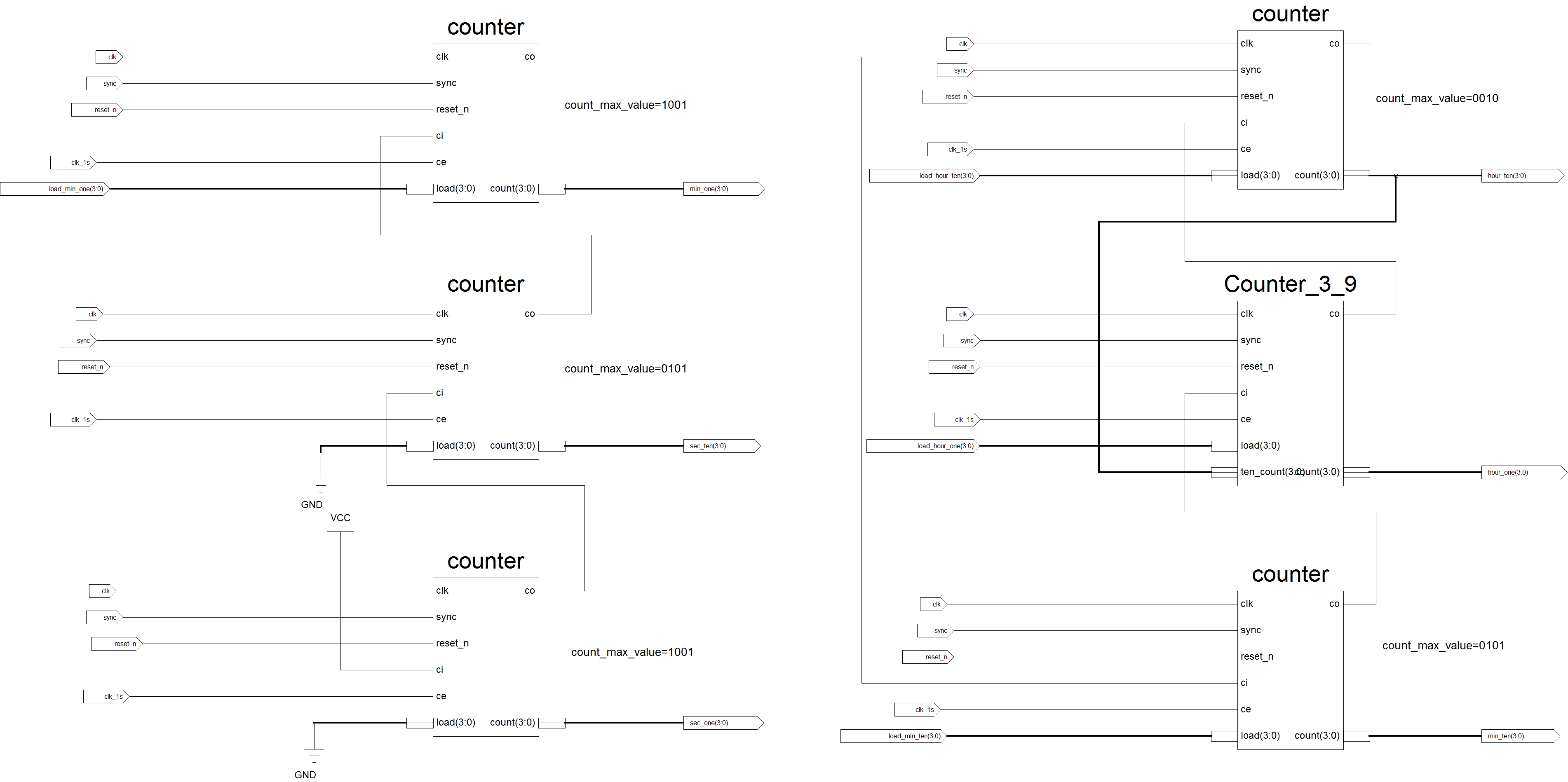
Das Clock-Modul ist eine fortlaufend zählende Uhr welche durch einen positiven Pegel auf sync mit einem Wert geladen werden kann. 

Abbildung 10 Aufbau der Uhr

**Inputs:**

|  |  |
| --- | --- |
| clk : in STD\_LOGIC; | Takt (Duty-Cycle 50%) |
| clk\_1s : in STD\_LOGIC; | Sekundentakt  clk\_1s <= '0';wait for clk\_period\_1s-clk\_period/2;  clk\_1s <= '1';wait for clk\_period/2; |
| reset\_n: in STD\_LOGIC; | Synchroner Reset |
| sync : out STD\_LOGIC; | Zeit Synchronisierungsignal: ‘1‘ während einem Takt. |
| load\_min\_one : out STD\_LOGIC\_VECTOR (3 downto 0); | Binäre Einerstelle der Minute wird geladen wenn sync=‘1‘ |
| load\_min\_ten : out STD\_LOGIC\_VECTOR (3 downto 0); | Binäre Zehnerstelle der Minute wird geladen wenn sync=‘1‘ |
| load\_hour\_one : out STD\_LOGIC\_VECTOR (3 downto 0); | Binäre Einerstelle der Stunde wird geladen wenn sync=‘1‘ |
| load\_hour\_ten : out STD\_LOGIC\_VECTOR (3 downto 0); | Binäre Zehnerstelle der Stunde wird geladen wenn sync=‘1‘ |

**Outputs:**

|  |  |
| --- | --- |
| sec\_one : out STD\_LOGIC\_VECTOR (3 downto 0); | Binäre Einerstelle der Sekunde |
| sec\_ten : out STD\_LOGIC\_VECTOR (3 downto 0); | Binäre Zehnerstelle der Sekunde |
| min\_one : out STD\_LOGIC\_VECTOR (3 downto 0); | Binäre Einerstelle der Minute |
| min\_ten : out STD\_LOGIC\_VECTOR (3 downto 0); | Binäre Zehnerstelle der Minute |
| hour\_one : out STD\_LOGIC\_VECTOR (3 downto 0); | Binäre Einerstelle der Stunde |
| hour\_ten : out STD\_LOGIC\_VECTOR (3 downto 0); | Binäre Zehnerstelle der Stunde |

**Generic**

Die Zähler werden im Schema durch setzen der generic Variable auf ihre jeweiligen Endwerte initialisiert. Dadurch mussten nur zwei verschiedene Zählertypen realisiert werden.

**Beschreibung**

Die Uhr zählt wie jede andere Uhr im 24 Stunden Format. Solange clk\_1s auf 1 ist sind die Zähler freigegeben und zählen bei jeder negativen Flanke von clk Eins weiter. Daher darf clk\_1s pro Sekunde nur während einer clk Periode freigegeben sein. Der sync Eingang verhält sich ebenfalls so, dass der Eingang solange gelesen wird, wie sync auf 1 ist und negative Flanken an clk erscheinen.

**Testbench**

In der Testbench wird das korrekte laden sowie das korrekte zählen überprüft. Dazu werden die Eingangtakte simuliert und Daten an den Load-Eingängen angelet, welche nach dem Reset geladen werden.

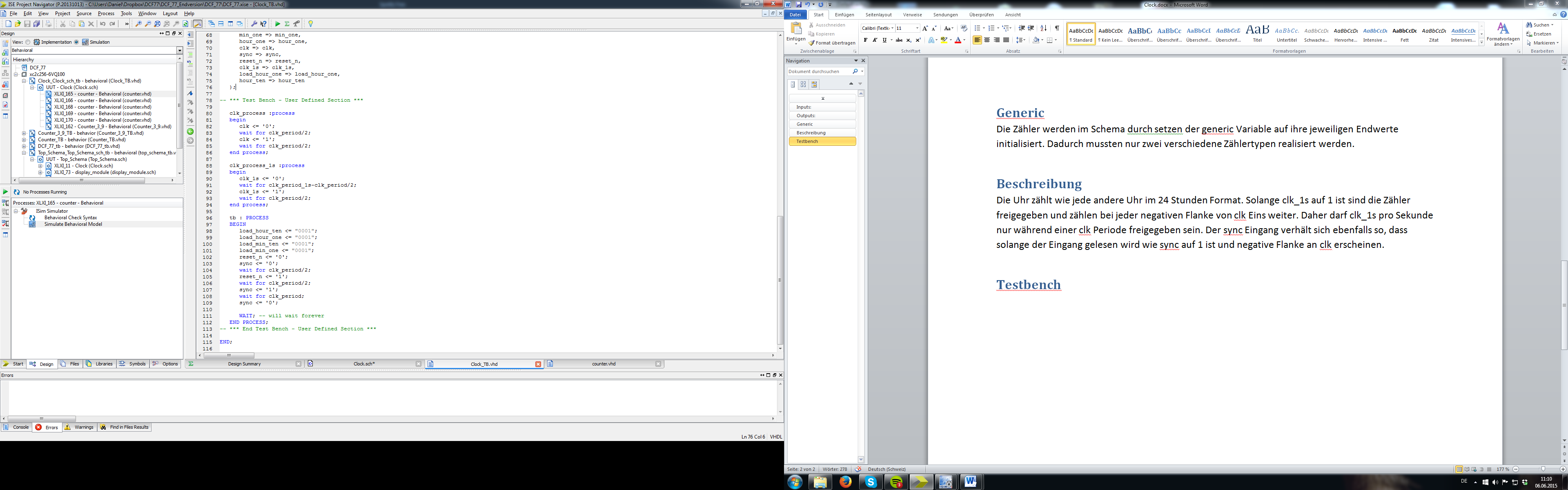


Abbildung 11 Code Testbench Uhr

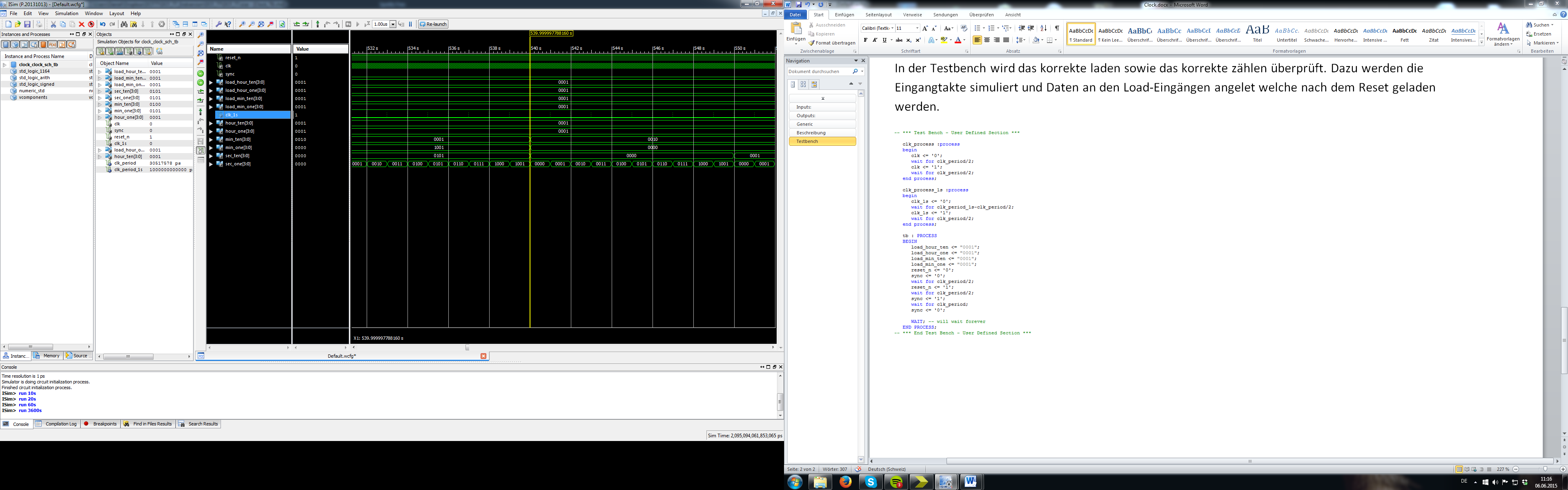


Abbildung 12 Testbench Uhr

Die Simulation zeigt denn Überlauf bei 11:19:59 zu 11:20:00 welcher so verläuft wie erwartet.

#### Counter und Counter3\_9

Es wurden zwei Zählermodule umgesetzt. Dies hat den Grund, dass 5 der 6 Zähler einen fixen Endwert besitzen und einer einen Variablen.

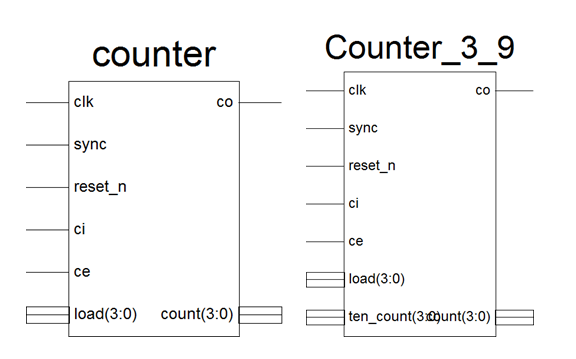


Abbildung 13 Symbol counter und Counter\_3\_9

**Inputs:**

|  |  |
| --- | --- |
| clk : in STD\_LOGIC; | Takt (Duty-Cycle 50%) |
| ce : in STD\_LOGIC; | Zählerfreigabe ( input für Sekundentakt) |
| reset\_n: in STD\_LOGIC; | Synchroner Reset |
| sync : out STD\_LOGIC; | Zeit Synchronisierungsignal: ‘1‘ während einem Takt. |
| ci : in STD\_LOGIC; | Übertragseingang |
| load : out STD\_LOGIC\_VECTOR (3 downto 0); | Ladewert wenn sync=‘1‘ und falling\_edge(clk) |
| ten\_count : in STD\_LOGIC\_VECTOR (3 downto 0); | Zählerstand Zehnerstelle für Umschaltung 3 und 9 Stunden Zählung |

**Outputs:**

|  |  |
| --- | --- |
| co : out STD\_LOGIC; | Übertragsausgang |
| count : out STD\_LOGIC\_VECTOR (3 downto 0); | Zählerausgang |

**Generic**

|  |  |
| --- | --- |
| count\_max\_value : STD\_LOGIC\_VECTOR (3 downto 0) := "0000"); | Maximalwert Zähler |

**Beschreibung**

Im Universalzähler counter kann der maximale Wert bis zu welchem gezählt werden soll durch ein generic gesetzt werden. Im counter\_3\_9 wird anhand der Zehnerstelle, also dem Wert des nachfolgenden Zählers, entschieden ob bis 3 oder 9 gezählt wird. So wird zwischen 19Uhr und 23Uhr unterschieden. Ansonsten sind die Module wie Standardzähler. Die exakte Realisierung kann dem Code entnommen werden, welcher mit der Dokumentation mitgeliefert wird.

**Testbench**

In der Testbench wird das korrekte laden sowie das korrekte zählen überprüft. Dazu werden die Eingangtakte simuliert und Daten an den Load-Eingängen angelet welche nach dem Reset geladen werden. Danach werden die ce sowie ci Eingange angeregt.

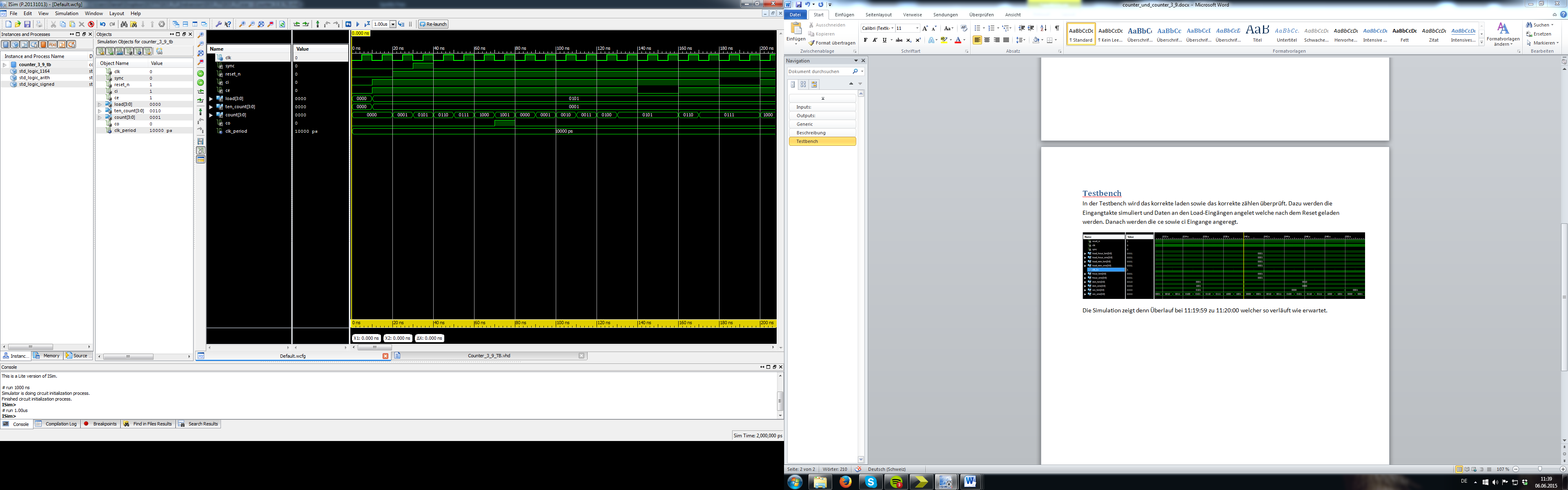


Abbildung 14 Testbench Counter

Die Simulation zeigt den Fall wo eine 5 geladen wird. Danach werden ce und ci abgeschaltet um zu testen, ob der Zähler aufhört zu zählen. Da ten\_count auf 1 ist gibt der Zähler einen Übertrag an co aus beim Zählerwert 9.

### Divider

Dieses Modul generiert aus der Uhrenquarzfrequenz (32768Hz) verschiedene andere Frequenzen, welche die anderen Module benützen.

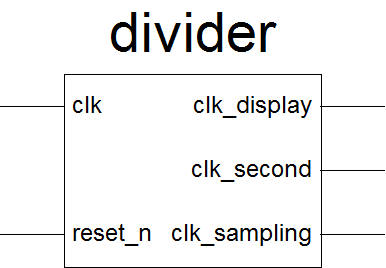


Abbildung 15 Symbol Divider

**Inputs:**

|  |  |
| --- | --- |
| clk : in STD\_LOGIC | Uhrenquarzfrequenz 32768Hz(Duty-Cycle 50%) |
| reset\_n: in STD\_LOGIC | Synchroner Reset |

**Outputs:**

|  |  |
| --- | --- |
| clk\_display : out STD\_LOGIC | Clockfrequenz für die Ansteuerung des Displays (32Hz, 50% Duty-Cycle) |
| clk\_second : out STD\_LOGIC | Clockimpuls für das CarryEnable (ce) der Uhr. Dieser Impuls ist nur gerader während einer Periode von der Uhrenquarzfrequenz aktiv und dies alle 1s. |
| clk\_sampling : out STD\_LOGIC | Clockfrequenz für die Abtastfrequenz des DCF77-Moduls. (128Hz, 50% Duty-Cycle) |

**Beschreibung**

Als Zählvariable für den Clock-Divider eignet sich ein std\_logic\_vector. Dies weil die Uhrenquarzfrequenz gerade so ist, dass man mit 15 FF’s in Serie eine Frequenz von 1Hz generieren kann. Auch muss man sich keine Gedanken über den Überlauf machen, da dieser Automatisch geschieht. Der ganze Prozess des Dividers ist der folgende:

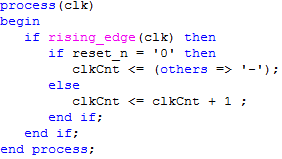


Abbildung 16 Prozess Divider

Sobald eine steigende Flanke der Uhrenquarzfrequenz detektiert wird, wird der Zähler bei Normalbetrieb um 1 inkrementiert. Da egal ist, welchen Zählerstand der Zähler nach dem Reset hat, wird der Zähler wahrscheinlich auch bei reset\_n = 0 inkrementiert. Um den Divider zu simulieren, muss der Resetwert wie folgt geändert werden: *clkCnt <= (others => ‚0‘);*

Die Generierung der Ausgangssignale ist sehr einfach: 

Abbildung 17 Ausgangszuweisung Divider

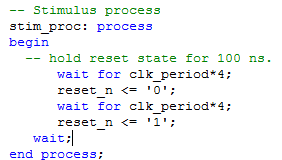
Der Clock für das Display ist jetzt gerade die 9. Stelle der Count-Variable. Dies ergibt eine Frequenz von:

Der Sekundentakt ist nur gerade aktiv, wenn alle 15 Bits des Zählers 0 sind. Das heisst, der clk\_second ist pro Sekunde während einer Taktperiode aktiv.

Der Clock für das DCF77-Modul ist jetzt gerade die 7. Stelle der Count-Variable. Dies ergibt eine Frequenz von:

**Testbench**

Die Testbench des Zählers ist einfach. Zuerst wird der Divider kurz geresetet, danach läuft er einfach weiter.



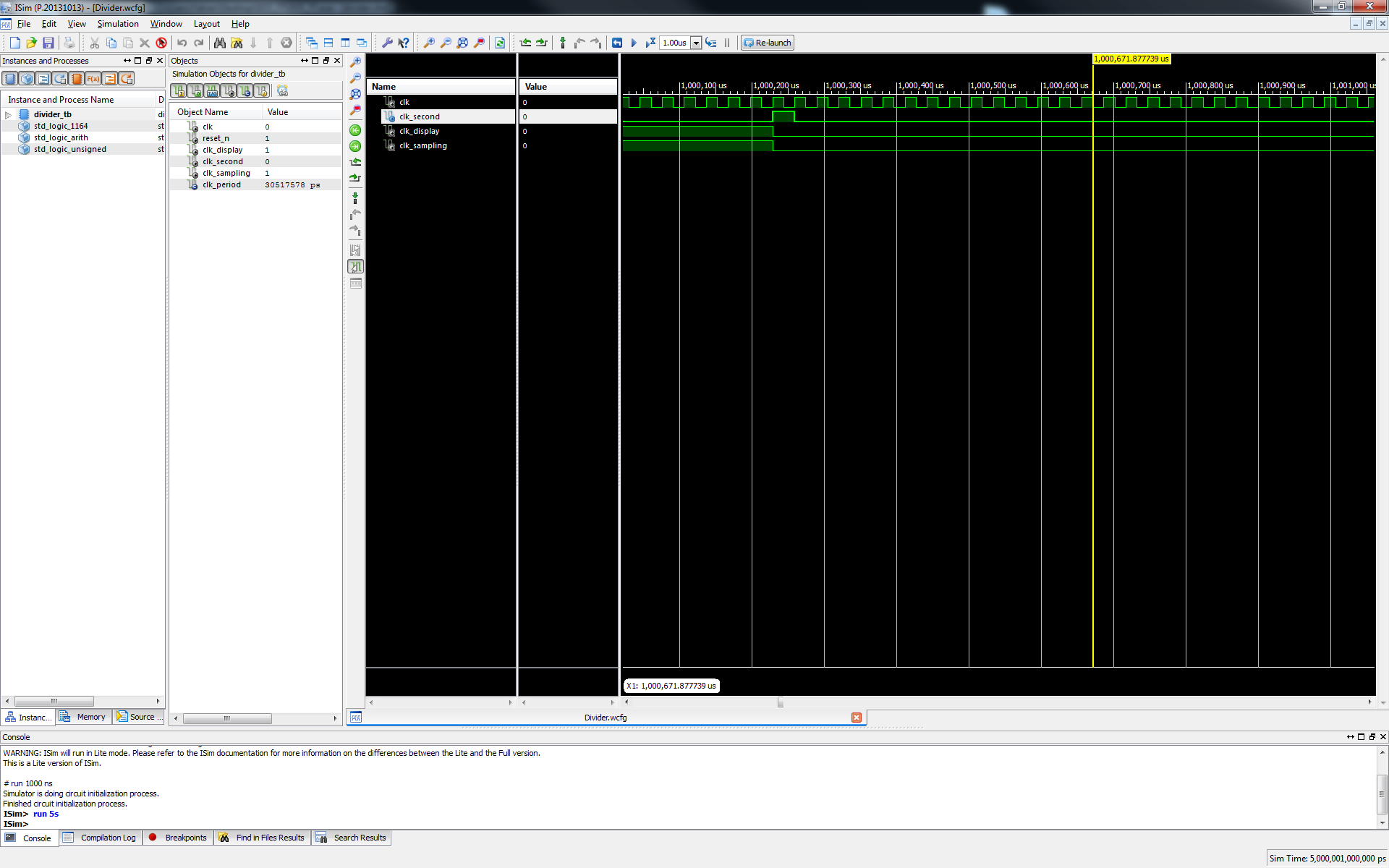


Abbildung 18 Testbench Divider: Sekundentakt

In diesem Bild sieht man sehr gut, wie der Sekundentakt während genau einer Periode der Uhrenquarzfrequenz aktiv ist.

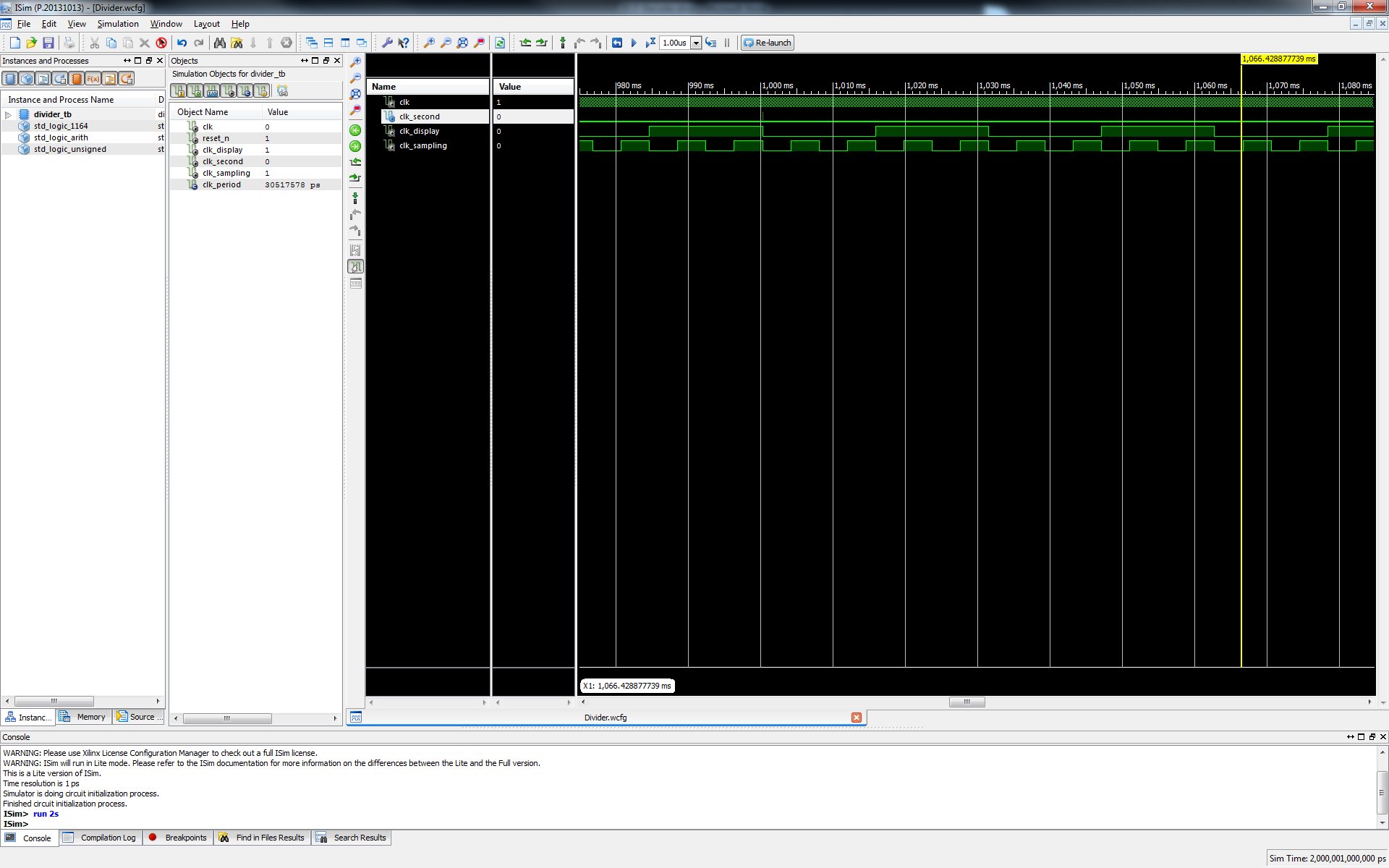


Abbildung 19 Testbench Divider: Sampling- und Displaytakt

Hier kann man erkennen, dass die Frequenz des Samplingtaktes etwa 128Hz ist. Der Displaytakt ist 3mal langsamer, also 32 Hz.

### Display-Modul

Dieses Modul wird dazu benötigt, das 7-Segment-LCD-Display anzusteuern. Da das Display keine DC-Spannung verträgt, müssen alle Ansteuerungssignale getaktet werden.

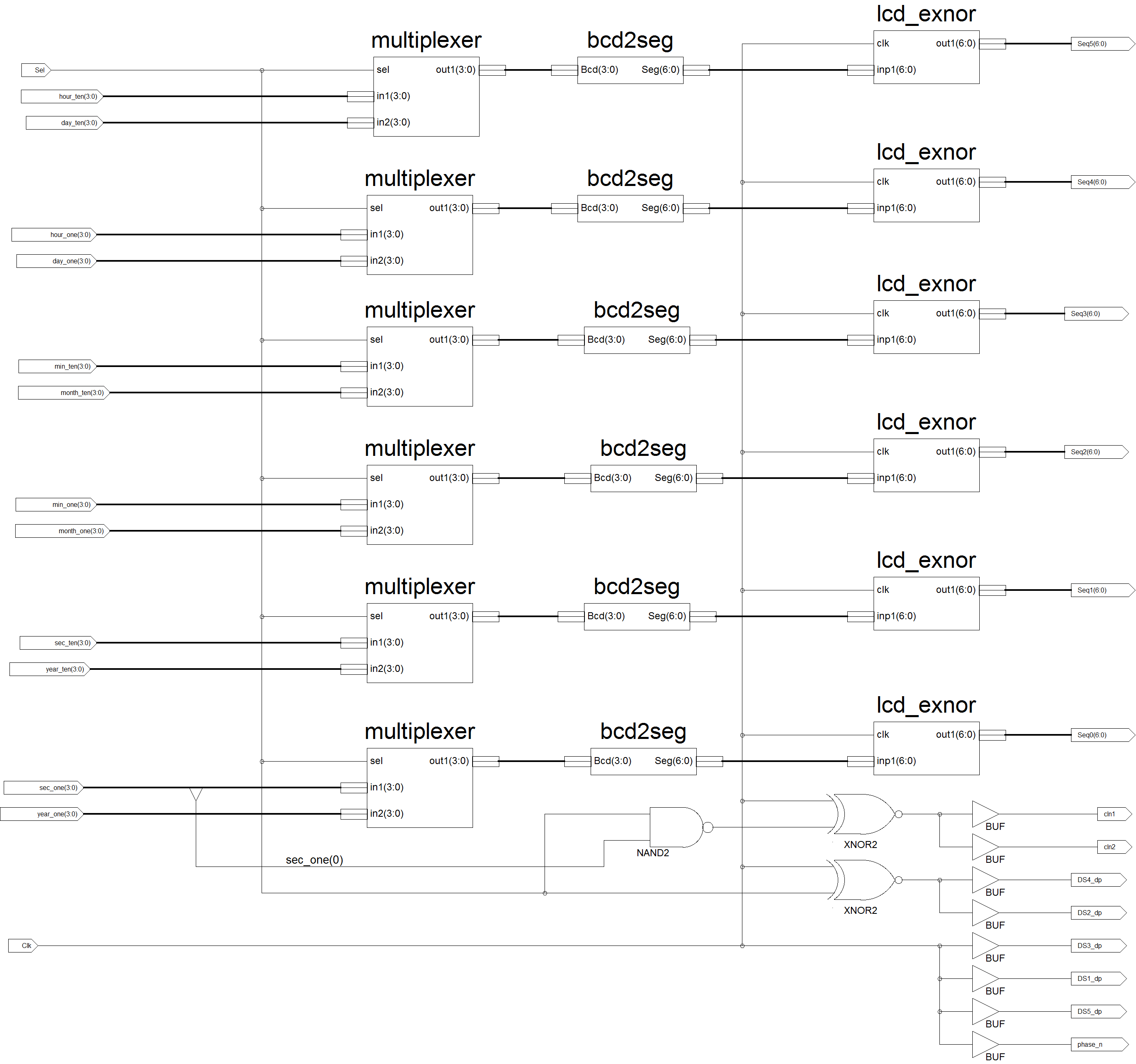


Abbildung 20 Aufbau des Display-Moduls

**Inputs:**

|  |  |
| --- | --- |
| clk : in STD\_LOGIC | Displayfrequenz (32Hz) |
| sel : in STD\_LOGIC | Select-Signal, 1 = Uhrzeit anzeigen  0 = Datum anzeigen |
| hour\_ten : in STD\_LOGIC\_VECTOR (3 downto 0); | Eingang Zehnerstelle Stunde |
| day\_ten : in STD\_LOGIC\_VECTOR (3 downto 0); | Eingang Zehnerstelle Tag |
| hour\_one : in STD\_LOGIC\_VECTOR (3 downto 0); | Eingang Einerstelle Stunde |
| day\_one : in STD\_LOGIC\_VECTOR (3 downto 0); | Eingang Einerstelle Tag |
| min\_ten : in STD\_LOGIC\_VECTOR (3 downto 0); | Eingang Zehnerstelle Minute |
| month\_ten : in STD\_LOGIC\_VECTOR (3 downto 0); | Eingang Zehnerstelle Monat |
| min\_one : in STD\_LOGIC\_VECTOR (3 downto 0); | Eingang Einerstelle Minute |
| month\_one : in STD\_LOGIC\_VECTOR (3 downto 0); | Eingang Einerstelle Monat |
| sec\_ten : in STD\_LOGIC\_VECTOR (3 downto 0); | Eingang Zehnerstelle Sekunde |
| year\_ten : in STD\_LOGIC\_VECTOR (3 downto 0); | Eingang Zehnerstelle Jahr |
| sec\_one : in STD\_LOGIC\_VECTOR (3 downto 0); | Eingang Einerstelle Sekunde |
| year\_one : in STD\_LOGIC\_VECTOR (3 downto 0); | Eingang Einerstelle Jahr |

**Outputs:**

|  |  |
| --- | --- |
| Seg5 : out STD\_LOGIC\_VECTOR (6 downto 0); | Ausgang Segment 1 |
| Seg4 : out STD\_LOGIC\_VECTOR (6 downto 0); | Ausgang Segment 2 |
| Seg3 : out STD\_LOGIC\_VECTOR (6 downto 0); | Ausgang Segment 3 |
| Seg2 : out STD\_LOGIC\_VECTOR (6 downto 0); | Ausgang Segment 4 |
| Seg1 : out STD\_LOGIC\_VECTOR (6 downto 0); | Ausgang Segment 5 |
| Seg0 : out STD\_LOGIC\_VECTOR (6 downto 0); | Ausgang Segment 6 |
| cln1 : out STD\_LOGIC | Ausgang Doppelpunkt1 |
| cln2 : out STD\_LOGIC | Ausgang Doppelpunkt2 |
| DS1\_dp : out STD\_LOGIC | Ausgang Punkt zwischen Seg0 und Seg1 |
| DS2\_dp : out STD\_LOGIC | Ausgang Punkt zwischen Seg1 und Seg2 |
| DS3\_dp : out STD\_LOGIC | Ausgang Punkt zwischen Seg2 und Seg3 |
| DS4\_dp : out STD\_LOGIC | Ausgang Punkt zwischen Seg3 und Seg4 |
| DS5\_dp : out STD\_LOGIC | Ausgang Punkt zwischen Seg4 und Seg5 |
| phase\_n : out STD\_LOGIC | Gemeinsamer Anschluss LCD |

**Beschreibung**

Zuerst wird mit dem Multiplexer ausgewählt, welches Signal (Uhrzeit oder Datum) angezeigt werden soll. Danach wird aus dem 4Bit Breiten BCD-Code die Ansteuerungssignale für die einzelnen Segmente generiert. Anschliessend werden die 7Bit Breiten Ansteuerungssignale für das LCD mit dem Takt EXNOR verknüpft und an das LCD ausgegeben. Der gemeinsame Anschluss des LCD (phase\_n) wird direkt an den Takt gelegt. Die Punkte 1, 3 und 5 werden auch an den Takt angelegt, da sie nie leuchten sollen. Die Doppelpunkte werden so verknüpft, dass sie nur wenn die Zeit dargestellt wird im 0.5Hz Takt blinken. Die beiden verbleibenden Punkte leuchten nur dann, wenn das Datum angezeigt wird.

**Testbench**

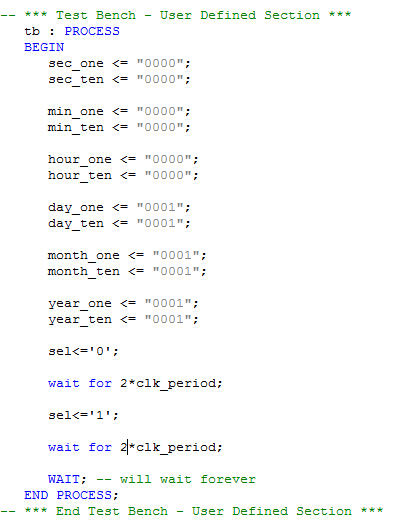


Abbildung 21 Code Testbench Display-Modul

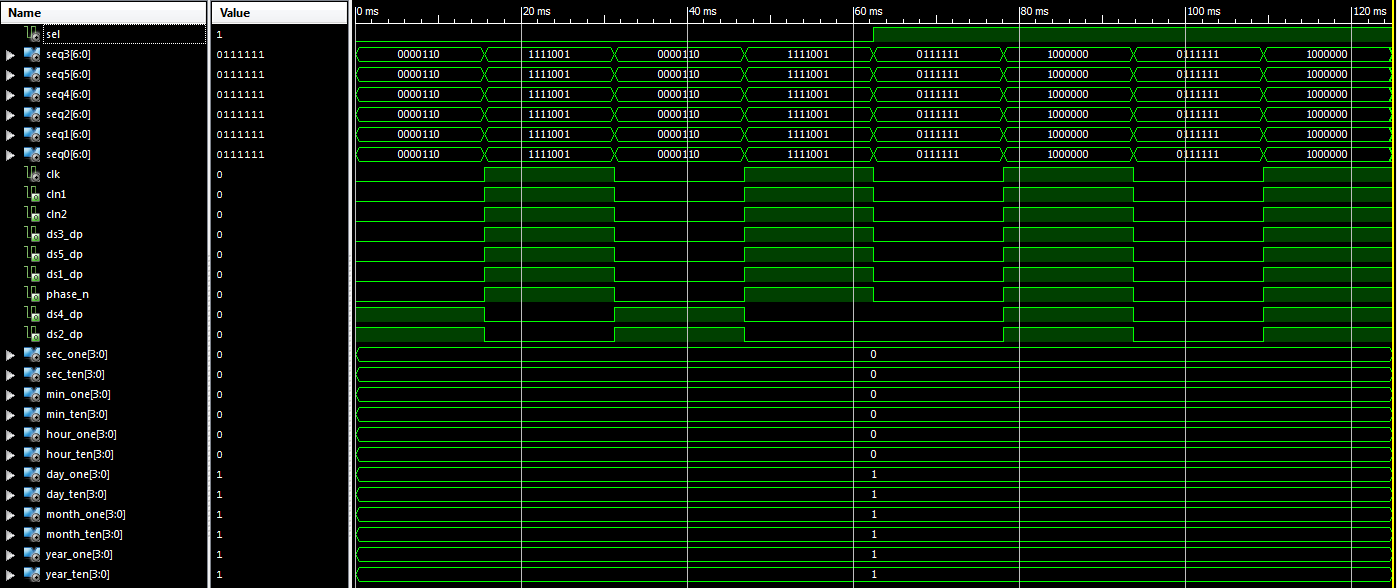


Abbildung 22 Testbench Display-Modul

Zuerst ist Sel = 0, das heisst das Datum 11.11.11 soll angezeigt werden. Während dieser Zeit ist ds4\_dp und ds2\_dp gegenphasig zu phase\_n. Damit leuchten die beiden Punkte beim Datum. Wenn der Takt ‚1‘ ist, sind bei den Segmenten nur gerade 2Bit ‚0‘, das heisst die beiden Segmente zur Darstellung der ‚1‘ leuchten. Wenn der Takt ‚0‘ ist sind die Bits gerade umgekehrt und zeigen auch wieder die ‚1‘ an.

Im zweiten Schritt wird Sel = 1, das heisst die Uhrzeit 00:00:00 soll angezeigt werden. Leider sieht man in dieser Darstellung nicht, wie die beiden Doppelpunkte blinken. Dies ist der Fall, weil sie jede Sekunde invertiert werden und bei einer Simulationsdauer von 60ms ist dies nicht sichtbar. Wenn der Takt ‚1‘ ist, sind bei den Segmenten gerade 6Bit ‚0‘, das heisst die sechs Segmente zur Darstellung der ‚0‘ leuchten. Wenn der Takt ‚0‘ ist sind die Bits gerade umgekehrt und zeigen auch wieder die ‚0‘ an. Wie man sieht leuchten die beiden Punkte nicht mehr, da sie gleichphasig wie der Takt (phase\_n) sind.

#### Multiplexer

Dieses Modul kann je nach Select-Signal die Uhrzeit oder das Datum zum Ausgang multiplexen.

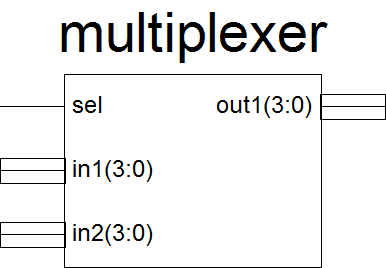


Abbildung 23 Symbol Multiplexer

**Inputs:**

|  |  |
| --- | --- |
| sel : in STD\_LOGIC | Auswahl, welcher Eingang zum Ausgang gemultiplext werden soll |
| in1 : in STD\_LOGIC\_VECTOR (3 downto 0); | Eingang für die Uhrzeit |
| In2 : in STD\_LOGIC\_VECTOR (3 downto 0); | Eingang für das Datum |

**Outputs:**

|  |  |
| --- | --- |
| out1 : in STD\_LOGIC\_VECTOR (3 downto 0); | Ausgangssignal des Multiplexer  out1 <= in1, wenn sel = 1  out1 <= in2, wenn sel = 0 |

**Beschreibung**

Das Ganze Modul besteht aus einem Prozess, welcher je nach Eingangssignal (sel) das Datum oder die Uhrzeit mit dem Ausgang verbindet.

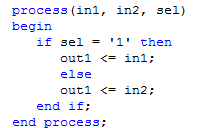


Abbildung 24 Prozess Multiplexer

**Testbench**

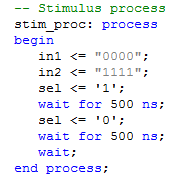


Abbildung 25 Code Testbench Multiplexer

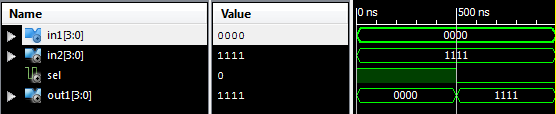


Abbildung 26 Testbench Multiplexer

Der Multiplexer funktioniert. Wenn sel = 1 wird in1 zum Ausgang weitergeleitet und wenn sel = 0 wird in2 zu Ausgang weitergeleitet.

#### BCD2SEG

Dieses Modul wandelt den BCD-Code in Ansteuerungssignale für die einzelnen Segmente des Displays um.

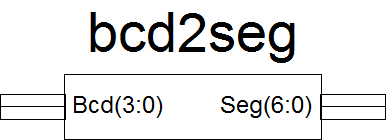


Abbildung 27 Symbol bcd2seg

**Inputs:**

|  |  |
| --- | --- |
| Bcd : in STD\_LOGIC\_VECTOR (3 downto 0) | Eingangssignal im BCD-Kode |

**Outputs:**

|  |  |
| --- | --- |
| Seg : out STD\_LOGIC\_VECTOR (6 downto 0) | Ausgangssignal, welches die Ansteuerungssignale für die einzelnen Segmente des Displays enthält. |

**Beschreibung**

Je nach Eingangssignal werden mit einer Case-Struktur die entsprechenden Codes an den Ausgang gelegt.

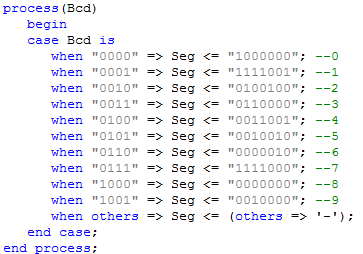


Abbildung 28 Prozess bcd2seg

**Testbench**

Auf eine erneute Simulation wurde verzichtet, da das Modul aus dem letzten Semester übernommen wurde.

#### Exnor

Da die 7-Segment-Anzeige keine DC-Spannung verträgt, werden die Ansteuerungssignale des Displays getaktet.

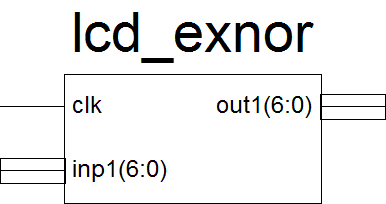


Abbildung 29 Symbol lcd\_exnor

**Inputs:**

|  |  |
| --- | --- |
| clk : in STD\_LOGIC | Clocksignal |
| inp1 : in STD\_LOGIC\_VECTOR (6 downto 0); | Ansteuerungssignale für die einzelnen Segmente des Displays |

**Outputs:**

|  |  |
| --- | --- |
| out1 : out STD\_LOGIC\_VECTOR (6 downto 0); | Getaktete Ansteuerungssignale für die einzelnen Segmente des Displays |

**Beschreibung**

Der Eingang wird Bitweise XNOR mit dem Eingangsclock verknüpft und am Ausgang angelegt.

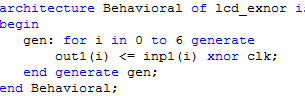


Abbildung 30 Code bcd2seg

**Testbench**

Auf eine Testbench wurde verzichtet.

# Synthesereport

Nach der Umsetzung des Projekts wurde die Ausgabe des Syntesereports auf die Plausibilität überprüft. Die verwendeten Gatter decken sich mit denen, die wir für die Umsetzung vorgesehen hatten. In der untenstehenden Tabelle, sieht man die verwendeten Ressourcen und wo sie eingesetzt werden.

|  |  |  |
| --- | --- | --- |
| Macro Statistics | Anzahl | Verwendung |
| **# ROMs** | **6** | **-** |
| 10x7-bit ROM | 6 | Laden der Initialwerte diverser Register |
| **# Counters** | 9 | **-** |
| 15-bit up counter | 1 | Zähler für den Clock-Divider |
| 4-bit up counter | 6 | Zähler für die Uhr. |
| 5-bit updown counter | 1 | Zähler für den Mittelwert (mean) des DCF77-Decoders |
| 8-bit up counter | 1 | zero\_time Counter für den DCF77-Decoder |
| **# Registers** | **9** | **-** |
| 1-bit register | 2 | Register month\_ten / Register sync |
| 2-bit register | 1 | Register day\_ten |
| 4-bit register | 4 | Register für day\_one, month\_one, year\_one und year\_ten |
| 5-bit register | 1 | max\_holder für den DCF77-Decoder |
| 59-bit register | 1 | Gesamtes Datenregister des empfangenen DCF77-Signals |
| **# Comparators** | **15** | **-** |
| 5-bit comparator greatequal | 3 | Vergleich mit mean-Counter und den Triggerwerten zur Erkennung eines ‚0‘ oder ‚1‘ |
| 5-bit comparator greater | 2 |
| 5-bit comparator less | 5 |
| 5-bit comparator lessequal | 3 |
| 8-bit comparator greatequal | 1 | Detektieren der Pause vor sync |
| 8-bit comparator less | 1 | Obere Zählgrenze des zero\_time-Counters des DCF77-Decoders |
| **# Xors** | **45** | **-** |
| 1-bit xor2 | 42 | Taktung der Steuerleitungen des LCD-Displays |
| 1-bit xor23 | 1 | Berechnung der Parität des DCF77-Decoders |
| 1-bit xor7 | 1 |
| 1-bit xor8 | 1 |

# Erkenntnisse

Wir haben bei dieser Projektarbeit viel Neues über Hardware sowie vor allem über VHDL gelernt. Die Ansteuerung eines LCDs ohne DC-Anteil war für uns Neuland. Wir haben erkannt, dass es wichtig ist bei kritischen Modulen wie dem LCD, alle Ansteuerungssignale in einem Modul zu vereinen. So kann gewährleistet werden, dass das Display immer korrekt angesteuert wird, auch wenn andere Module auf dem Schema geändert werden.   
Weiter konnten wir unser Wissen aus der Signalverarbeitung einbringen. Das Vorgehen mit der vorgängigen Simulation in MATLAB und anschliessender Umsetzung in VHDL hat sehr gut funktioniert.   
Wir hatten zuerst einige Mühe mit der Implementierung eines generischen Modules in ein Schema, da wir nicht wussten wie man die generic Variablen dort setzen kann. Dank Internet konnten wir dieses Problem jedoch selbständig lösen und so wieder etwas Neues lernen.   
Die wohl wichtigste Erkenntnis ist, dass ein asynchroner Reset auf diesem Board nicht zu funktionieren scheint. Dieses Problem hat uns einige Stunden gekostet, da es keine Fehlermeldungen beim Implementieren gab. Module welche einzeln getestet funktioniert hatten, funktionierten plötzlich integriert in das Gesamtschema nicht mehr. Durch umbauen des ganzen Systems auf einen synchronen Reset konnte dann die gesamte Schaltung zum Laufen gebracht werden.

# Fazit

Wir sind mit unserem Resultat sehr zufrieden. Die Vorgaben konnten alle erfüllt werden und wir konnten viele neue Erkenntnisse aus der Projektzeit mitnehmen. Durch die anderen parallel laufenden Module blieb leider nicht genügend Zeit um weitere Funktionalitäten einzubauen. Trotzdem ist es uns gelungen eine sehr stabile Auswertung des DCF-Signales zu realisieren und eine schöne Darstellung der Zeit auf dem Display zu erhalten.

Als gute Erweiterung könnte man weitere Daten aus dem Signal auslesen und diese versuchen auf dem Display darzustellen. Auch könnte eine Weckfunktion implementiert werden. Ob dies ohne Platzoptimierungen der bereits geschriebenen Module möglich wäre ist unklar, da die Auslastung des CPLD bereits bei rund 80% liegt. Die Aufgabenstellung war sehr interessant und vielseitig. Für jede Person bestand die Möglichkeit in dem Bereich, in welchem man gerne mehr investieren möchte, auch etwas tiefer in die Materie einzutauchen.

Das Projekt hat uns Spass gemacht und uns in unserem Studium weiter gebracht. Somit haben wir unser Ziel erreicht.

# Abbildungsverzeichnis

[Abbildung 1 Grobschema 3](#_Toc421469436)

[Abbildung 2 Topschema 6](#_Toc421469437)

[Abbildung 3 Testbench Topschema: Einlesen des Telegramms 7](#_Toc421469438)

[Abbildung 4 Testbench Topschema: Synchronisation 8](#_Toc421469439)

[Abbildung 5 Symbol DCF77-Decoder 9](#_Toc421469440)

[Abbildung 6 Simulation mit MATLAB 11](#_Toc421469441)

[Abbildung 7 Code Testbench DCF77-Modul 12](#_Toc421469442)

[Abbildung 8 Code Testbench: Funktion zum Simulieren von Daten 12](#_Toc421469443)

[Abbildung 9 Testbench DCF77-Decoder 13](#_Toc421469444)

[Abbildung 10 Aufbau der Uhr 14](#_Toc421469445)

[Abbildung 11 Code Testbench Uhr 15](#_Toc421469446)

[Abbildung 12 Testbench Uhr 15](#_Toc421469447)

[Abbildung 13 Symbol counter und Counter\_3\_9 16](#_Toc421469448)

[Abbildung 14 Testbench Counter 17](#_Toc421469449)

[Abbildung 15 Symbol Divider 18](#_Toc421469450)

[Abbildung 16 Prozess Divider 19](#_Toc421469451)

[Abbildung 17 Ausgangszuweisung Divider 19](#_Toc421469452)

[Abbildung 18 Testbench Divider: Sekundentakt 20](#_Toc421469453)

[Abbildung 19 Testbench Divider: Sampling- und Displaytakt 20](#_Toc421469454)

[Abbildung 20 Aufbau des Display-Moduls 21](#_Toc421469455)

[Abbildung 21 Code Testbench Display-Modul 23](#_Toc421469456)

[Abbildung 22 Testbench Display-Modul 23](#_Toc421469457)

[Abbildung 23 Symbol Multiplexer 24](#_Toc421469458)

[Abbildung 24 Prozess Multiplexer 24](#_Toc421469459)

[Abbildung 25 Code Testbench Multiplexer 25](#_Toc421469460)

[Abbildung 26 Testbench Multiplexer 25](#_Toc421469461)

[Abbildung 27 Symbol bcd2seg 26](#_Toc421469462)

[Abbildung 28 Prozess bcd2seg 26](#_Toc421469463)

[Abbildung 29 Symbol lcd\_exnor 27](#_Toc421469464)

[Abbildung 30 Code bcd2seg 27](#_Toc421469465)