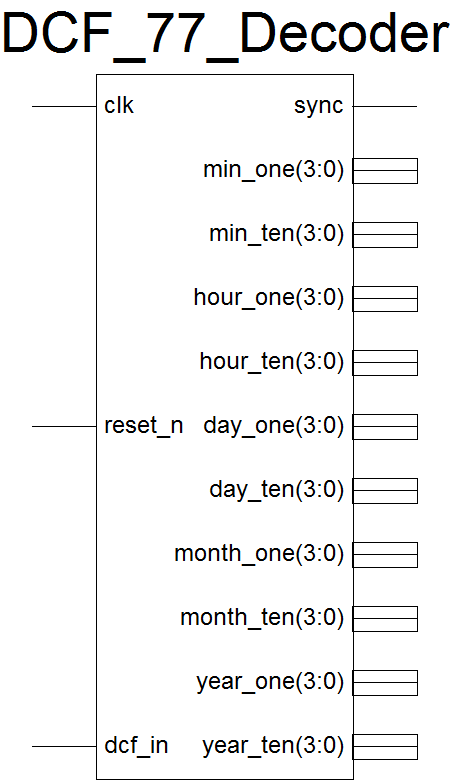
DCF-77-Decoder

Dieses Modul liest ein DCF-77 Telegramm während einer Minute ein und extrahiert daraus die aktuelle Zeit und das aktuelle Datum. Sobald ein Telegramm empfangen wurde wird ein sync ausgegeben.



# 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 DCF-77-Decoders ist die Unterscheidung zwischen 1 und 0 des DCF-Signals und des korrekten Synchronisierens auf die neue Zeit. Bei der Konzeptionierung dieses Moduls wurde von einer schlechten Signalqualität ausgegangen, da die effektive Signalqualität nichtbekannt 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-Signal 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, 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.

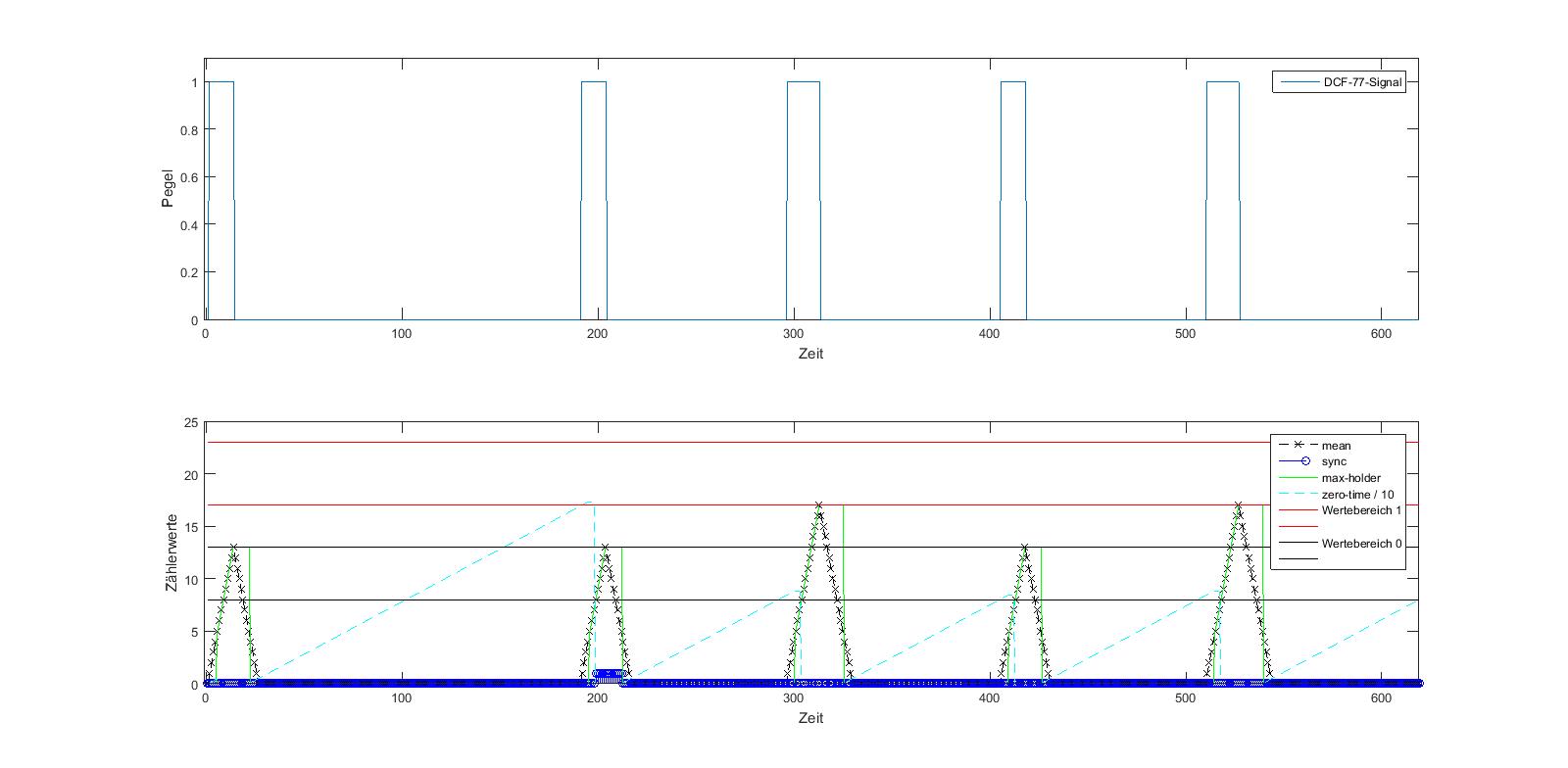
Es gibt 5 verschiedene Levels für mean welche das Verhalten des restlichen Ablaufes beeinflussen:  
one\_max, one\_min Wertebereich für ein eingelesenes 1 des Telegramms.  
zero\_max, zero\_min Wertebereich für ein eingelesenes 0 des Telegramms.  
nothing\_trigger Minimalwert damit von einem stabilen 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.



### Sync Verzögerung und Einflüsse Signalqualität

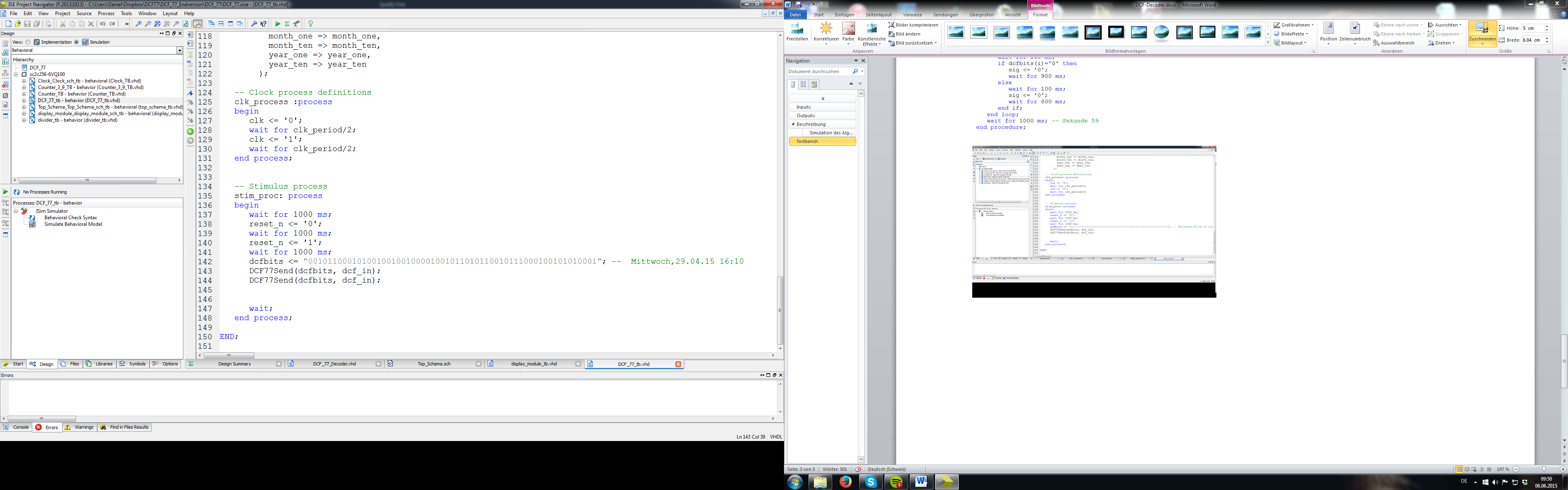
Da das sync erst nach dem Überschreiten des Minimalwertes für eine 1 (one\_min) ausgelöst wird hat das sync eine Verzögerung von 80ms. Hinzu kommt noch ein weiterer Fehler durch die nicht sehr hohe Abtastrate von 128Hz (7,8125ms), welche aus Platzgründen auf ein vielfaches von 2 gelegt werden musste. 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 dies auch 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.

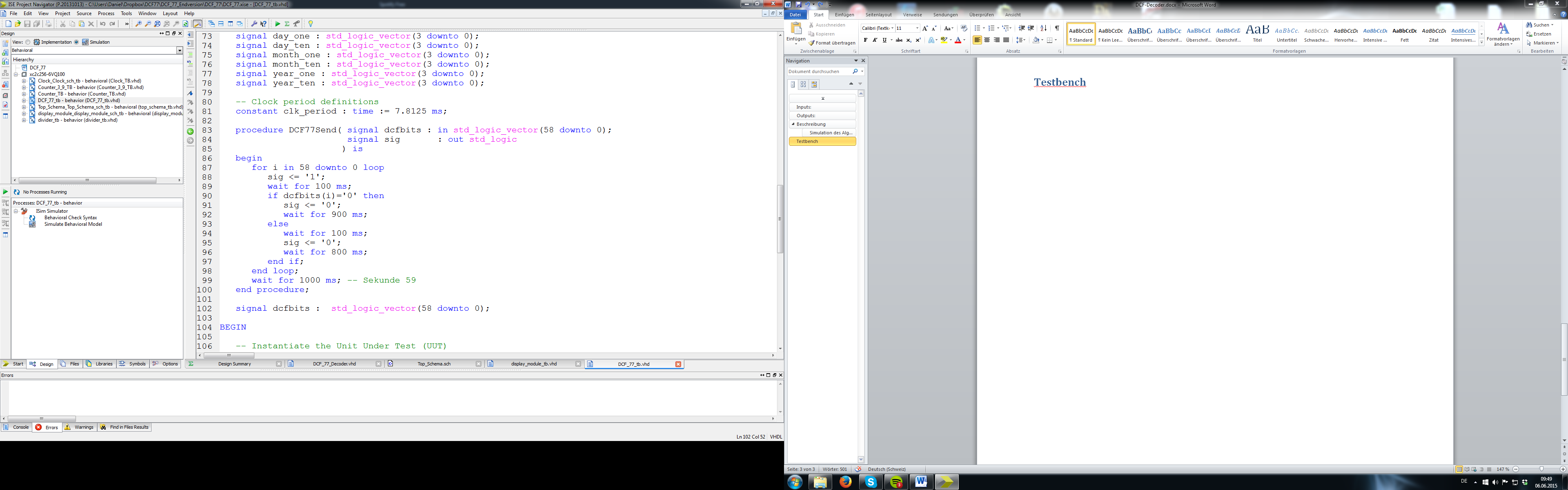
Ein schlechtes Signal kann das sync zusätzlich um mehrere 10ms verzögern im Gegenzug gewinnt man ein sehr stabiles System welches selbst mit einer sehr schlechten Signalqualität zurechtkommt. Sollte trotzdem ein Fehler eingelesen werden gibt es als weitere Sicherheit noch die Checksumme womit das Signal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

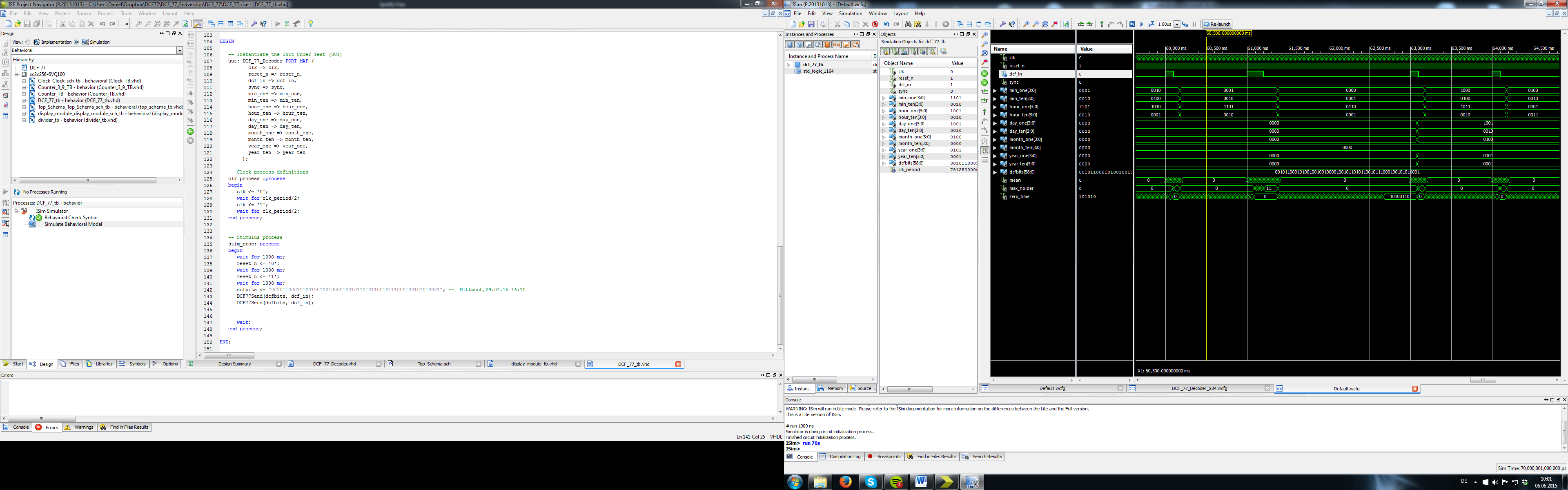
# Testbench

Um den Decoder zu Testen wurde eine Testbench geschrieben, welcher man ein Testprotokoll übergeben, 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.



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.





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.