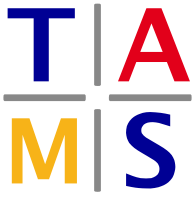
VHDL Schaltungs- und Systementwurf

4

Eine DCF 77 gesteuerte Weckuhr



Andreas Mäder

Universität Hamburg – MIN – Fachbereich Informatik
Arbeitsbereich Technische Aspekte Multimodaler Systeme

https://tams.informatik.uni-hamburg.de

Mit diesem letzten Aufgabenblatt soll nun (als krönender Abschluss) ein kleiner IC entworfen werden, der ergänzend zur Weckuhr ein Interface zu dem Zeitzeichensender DCF 77 besitzt.

Diese Schaltung soll auf einer FPGA-Prototypenplatine realisiert werden. Um aufzuzeigen wie die weiteren Schritte im Falle einer Standardzellentwurfs aussehen würden, ist geplant auch ein Chiplayout zu erstellen.

Das DCF 77 Signal

Das Zeitzeichensignal übermittelt codiert die aktuelle Zeit und das Datum.

- Innerhalb einer Minute werden 59-bit übertragen, die codiert die Information der *nächsten folgenden* Minute beinhalten.
- Die Übertragung findet synchron zum Sekundenrhythmus statt, wobei ein genauer Sekundenbeginn festgelegt wird.
- Durch das *Fehlen* der letzten (59.) Sekunde wird gekennzeichnet, dass mit der nächsten übertragenen Sekunde eine neue Minute beginnt.
  Minute : $+1$ für sie gilt die vorher gesendete Information
  Sekunde : 0

Das übertragene Signal hat (nach der Digitalisierung) den, in Abbildung 1 dargestellten, zeitlichen Signalverlauf. Die einzelne Bits der Codierung werden durch folgendes Schema übertragen:

Codiertes Bit	0-Pegel [ms]	1-Pegel [ms]
'0'	100	900
'1'	200	800

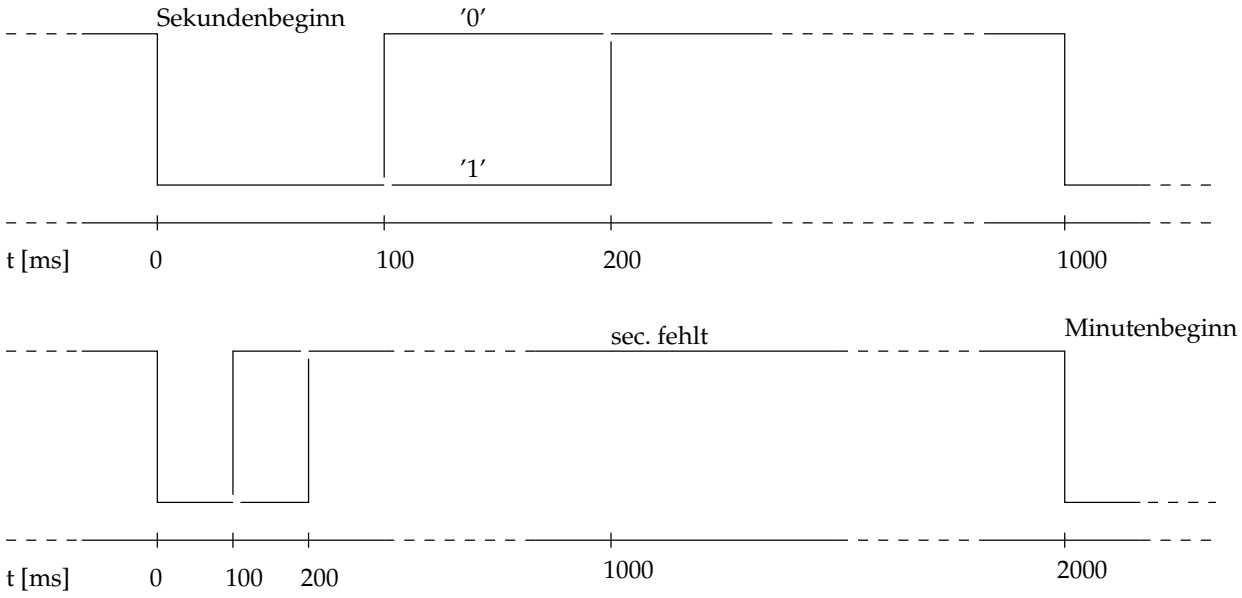


Abbildung 1: Zeitschema des DCF 77 Signals

Codierung des DCF 77 Signals

Sek.	Bit	Wert	Bedeutung	Beispiel
0	0		Minutenbeginn	0
1	∗		nicht belegt, i.A. wird 0 gesendet	0
⋮				
14	∗		– s.o. –	0
15	0 / 1		1: Reserveantenne an	0
16	0 / 1		1: Stundensprung folgt (z.B. Sommerzeit)	0
17	0 / 1	2	Zeitzonenbits: geben die Abweichung von der	1
18	0 / 1	1	Weltzeit in Stunden an (MEZ = 1, MESZ = 2)	0
19	0 / 1		1: Schaltsekunde folgt	0
20	1		Start der Zeitcodierung	1
21	0 / 1	1	Minuten – Stelle 1	1
22	0 / 1	2	– s.o. –	0
23	0 / 1	4	– s.o. –	1
24	0 / 1	8	– s.o. –	0 = 5
25	0 / 1	10	Minuten – Stelle 10	1
26	0 / 1	20	– s.o. –	1
27	0 / 1	40	– s.o. –	0 = 3
28	0 / 1		Prüfbit zu 21…27 (even)	0
29	0 / 1	1	Stunden – Stelle 1	1
30	0 / 1	2	– s.o. –	0
31	0 / 1	4	– s.o. –	0
32	0 / 1	8	– s.o. –	1 = 9
33	0 / 1	10	Stunden – Stelle 10	1
34	0 / 1	20	– s.o. –	0 = 1
35	0 / 1		Prüfbit zu 29…34 (even)	1
36	0 / 1	1	Kalendertag – Stelle 1	0
37	0 / 1	2	– s.o. –	1
38	0 / 1	4	– s.o. –	0
39	0 / 1	8	– s.o. –	0 = 2
40	0 / 1	10	Kalendertag – Stelle 10	0
41	0 / 1	20	– s.o. –	0 = 0
42	0 / 1	1	Wochentag	1
43	0 / 1	2	– s.o. –	0
44	0 / 1	4	– s.o. –	0 = 1 (Mo)
45	0 / 1	1	Kalendermonat – Stelle 1	0
46	0 / 1	2	– s.o. –	1
47	0 / 1	4	– s.o. –	0
48	0 / 1	8	– s.o. –	0 = 2
49	0 / 1	10	Kalendermonat – Stelle 10	1 = 1
50	0 / 1	1	Kalenderjahr – Stelle 1	1
51	0 / 1	2	– s.o. –	0
52	0 / 1	4	– s.o. –	0
53	0 / 1	8	– s.o. –	1 = 9
54	0 / 1	10	Kalenderjahr – Stelle 10	1
55	0 / 1	20	– s.o. –	0
56	0 / 1	40	– s.o. –	0
57	0 / 1	80	– s.o. –	0 = 1
58	0 / 1		Prüfbit zu 36…57 (even)	1
59	–		Marke fehlt, weder 0 noch 1 wird gesendet	
	Beispiel:		Mo. 02.12.19: 19.35	

Arbeitsweise

Der komplette Designablauf für ein IC in Standardzell-Technik ist in Abbildung 2 dargestellt; im Bereich der VHDL-Eingabe, Simulation und Synthese entspricht er dem Vorgehen bei der letzten Aufgabe, ist jetzt aber um die Anbindung an das abschließende Layout (Platzierung & Verdrahtung) erweitert.

1. Beschreibung der einzelnen Komponenten (Hauptblöcke) der Schaltung durch VHDL (Verhaltens-) Modelle. In der Regel wird schon synthetisierbarer RT-Code eingegeben, so dass Algorithmen- und Architekturdesign (Abb. 2) zusammenfallen.
2. Simulation der Schaltungen in geeigneten Testumgebungen.
3. (optional) Um dabei auch die Synthetisierbarkeit der VHDL-Eingabe zu gewährleisten, empfiehlt es sich die jeweiligen Komponenten mit den Synthesewerkzeugen auf die Zellbibliothek abzubilden.
4. Anschließend wird der Chip aus den vorgefertigten Komponenten in Form einer VHDL-Strukturbeschreibung aufgebaut.
5. Die Simulation der Gesamtstruktur soll die einwandfreie Funktion beim Zusammenspiel der einzelnen Teile des ICs gewährleisten. Bestehen zwischen den einzelnen Komponenten zeitliche Abhängigkeiten, die in der „standalone“ Simulation noch nicht bemerkt worden sind, so werden diese jetzt festgestellt.
6. Der anschließende Syntheseschritt bildet die VHDL-Beschreibungen des Designs auf eine Struktur aus Elementen der $0,35\,\mu m$ AMS-Standardzellbibliothek ab.

   Bei der Synthese können schon Padzellen für den späteren Chip eingefügt werden. Die, durch die VHDL-Entities vorgegeben Hierarchie, bleibt bei der Synthese erhalten.
7. Unter Beibehaltung der „alten“ Testumgebung kann jetzt die synthetisierte Netzliste simuliert werden. Als Netzlistenformat sind hier VHDL *und* Verilog möglich.

   Da eine komplette Simulation der Gatternetzliste häufig lange dauert, können gegebenenfalls Teile der Schaltung als VHDL-Verhaltensmodelle simuliert werden, was sich über geeignete Konfigurationen (`configuration`) einstellen lässt.
8. Über eine Verilog-Netzliste wird der Entwurf in die Platzierungs- und Verdrahtungswerkzeuge transferiert.
9. Anschließend werden die Standardzellen platziert und verdrahtet, womit das Layout fertig ist und die Maskendaten für eine Chipfertigung erzeugt werden können.

10. Nach dem Layout folgen noch mehrere Testprozeduren, die gewährleisten sollen, dass der Entwurf auch *nach* der Fertigung funktioniert. Dies sind Konsistenzprüfungen und elektrische Checks:

- Sind genügend Padzellen für die Spannungsversorgung vorhanden?
- Sind die Ausgänge einzelner Stufen durch zu viele nachgeschaltete Eingänge oder zu lange Leitungen überlastet?
- Sind die Leitungen (Netze) zu lang?

Dabei werden aus dem Layout zusätzliche Verzögerungen, durch die Kapazitäten der Leitungen, für eine nachfolgende Simulation extrahiert.

11. *Golden Simulation*
Mit einer abschließenden post-Layout Simulation wird jetzt noch einmal die Funktion der Schaltung überprüft.

Der größte Teil des Arbeitsaufwands ($> 70\%$) wird bei den ersten Schritten, also im Bereich des *Algorithmen- und Architekturdesigns*, anfallen: entsprechend den Punkten 1 – 5 der Liste.

Anmerkung: Zum Verständnis der unterschiedlichen Simulationsschritte sind hier noch einmal die verschiedenen „Arten“ aufgezählt:

2. Simulation von Teilschaltungen als VHDL-Verhaltensmodell *ohne Verzögerungszeiten*.

5. Simulation des hierarchischen Designs als VHDL-Verhaltensmodell *ohne Verzögerungszeiten*. Eine zeitliche Synchronisation von Ereignissen findet in einem (hoffentlich) synchronem Entwurf ausschließlich über das *Taktschema* statt.

7. Simulation des Designs als Gatternetzliste. Dabei werden die jeweiligen *Gatterverzögerungszeiten* der Herstellerbibliotheken in der Simulation berücksichtigt.

Im Vergleich zu den vorherigen Simulationen ist der Aufwand um Größenordnungen größer: Verhaltensmodelle : 100 Prozesse
Gatternetzliste : 10 000 – 100 000 Prozesse

11. post-Layout Simulation des Designs. Diese abschließende Simulation entspricht dem Zeitverhalten, das der Hersteller nach der Fertigung gewährleistet, bestehend aus den *Gatterverzögerungszeiten und leitungsbedingten Verzögerungen zwischen den Gattern*. Die Leitungslaufzeiten wurden aus dem fertigen Layout extrahiert.

Bei einem FPGA-Entwurf finden die Synthese- und Layoutschritte (6, 8…10) in der FPGA-Entwicklungsumgebung statt. Da die FPGA-Hersteller (Intel, Xilinx etc.) die Eigenheiten ihrer Bausteine direkt in der Software berücksichtigt haben, sind Platzierung und Verdrahtung (= Mapping im FPGA), sowie die abschließenden Tests, deutlich einfacher zu handhaben, als beim Chipentwurf. Die „Art der Schritte“ und die Komplexität (der Algorithmen) sind aber vergleichbar.

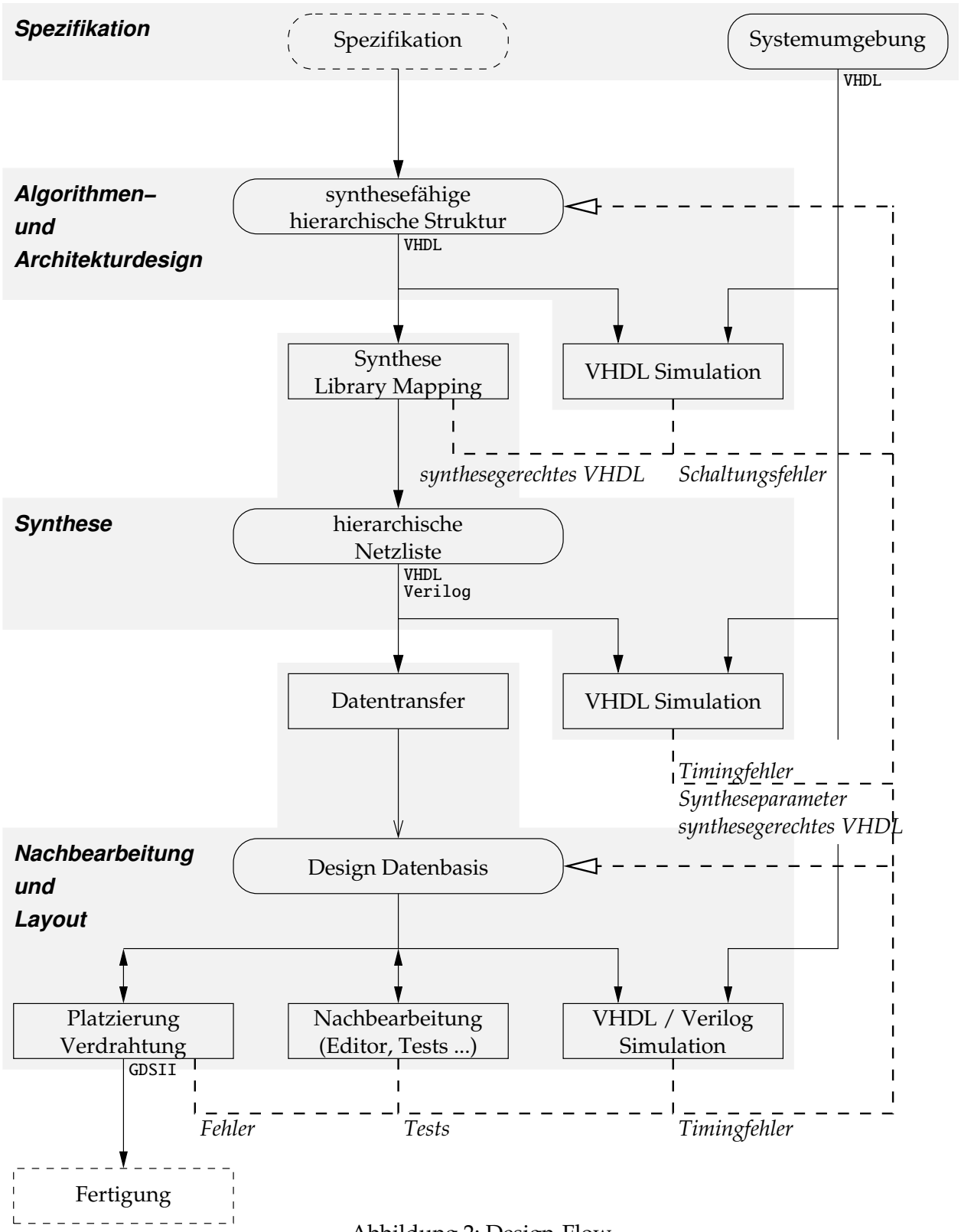


Abbildung 2: Design-Flow

Aufgabe 4.1

Entwerfen sie einen **Automaten**, der das DCF 77 Zeitsignal decodiert; er soll dann die Uhr (den Wecker) aus Aufgabenblatt **3** steuern.

Verfahren Sie dabei wie im letzten Versuch: VHDL-Eingabe und Simulation mit einer Testumgebung... Für die Simulation des Automaten wird ein Modell des DCF 77-Senders bereitgestellt:

`dcf77.vhd`	VHDL-Entity zur Simulation des DCF 77-Senders.
`dcffsm.vhd`	Entity-Deklaration des Decodierautomaten.

Der neu zu entwerfende Decoder des DCF 77 Signals soll in der Uhrenschaltung folgende Aufgaben übernehmen:

1. Er soll sich auf den Sekundentakt synchronisieren, also die Rückflanke des Zeitsignals.
2. Läuft der Automat synchron zum Zeittakt soll die übertragene Information (Nutzbit '0' oder '1') erkannt werden.
3. Durch das Fehlen der 59. Sekunde kann sich die Schaltung dann auf den Minutenrhythmus einstellen und die gewonnene Information auswerten — vorher können die empfangenen Nutzbits nicht im Code eingeordnet werden.
4. Wurde eine Minute fehlerfrei empfangen, so soll beim nächsten Minutenbeginn
   - die empfangene Zeit in `timblk` geladen werden. Dies entspricht dem Stellen der Uhr durch den DCF-Empfänger.
   - das empfangene Datum in der Anzeige erscheinen. Wird das Funksignal nicht richtig empfangen, so sollte die Datumsanzeige abgeschaltet sein.
   - der Sekundentakt des Taktgenerators (`clkgen`) synchronisiert werden.

Zur Funktionsweise und zur Auswertung des Zeitsignals noch einige Anmerkungen:

- Wird kein DCF 77 Signal empfangen oder ist der Empfang zu sehr gestört, so muss die Uhr autark laufen (selbstverständlich). Die Uhr wird nur, wie unter Punkt 4 beschrieben, synchronisiert, wenn die Information „richtig" empfangen wurde. Sonst läuft sie unabhängig vom DCF-Decoder; dies ist auch der Fall während der jeweils 60 Sekunden die der Automat braucht um die Information „einzusammeln".
- Es ist möglich, dass das DCF 77 Signal durch Rauschen gestört ist. Durch den Aufbau des Analog-Empfangsteils – der Ausgang ist über einen pull-up Widerstand geschaltet – wirken sich Störungen überwiegend als *Peaks* während des 0-Pegels aus.

  Um jetzt das Signal sicher abzutasten ist vorgesehen den Eingang im 1kHz Takt auszuwerten (= 1 ms). Außerdem ist dadurch eine hinreichend genaue Synchronisation auf die Rückflanke sichergestellt.

  Durch die Überabtastung ist es dann möglich auftretende Fehler bis zu einem gewissen Grad zu tolerieren und andererseits, bei zu sehr gestörten Signalen wo die Synchronisation auf den Sekundenbeginn schon unsicher wird, den Automaten in einen „Wartezustand" zurückzusetzen.

- Läuft der Automat synchron zum Sekunden-/Minutentakt, so sollte die Nutzinformation in (Schiebe-) Registern gespeichert werden. Dies sind hier nur die Bits für die Uhrzeit und das Datum — eine Auswertung der Prüfbits ist (meiner Meinung nach) nicht notwendig, da durch die Überabtastung schon eine ausreichende Kontrolle des Eingangssignals stattfindet. Außerdem ließ sich aus der vorliegenden Literatur keine eindeutige Aussage treffen, über welche Bits genau die Prüfbits berechnet werden (d.h. die Angaben zu den Prüfbits in der Beschreibung des DCF 77-Codes können falsch sein).
- Entsprechend der vorgestellten Spezifikation hat der Automat die folgenden Ein- und Ausgänge:

Signal	Wirkung	Funktion
`reset`	active Low	asynchrones Reset
`clk1ms`	Vorderflanke	Takt für DCF 77-Decoder
`dcfsig`		DCF 77 Eingangssignal
`dcfsok`	active High	fehlerfreier Empfang (LED)
`dcfload`	active High	empfangenen Code laden Uhr synchronisieren
`dcf_mins1`	4-bit Bus	Minuten 1er
`dcf_mins10`	3-bit Bus	Minuten 10er
`dcf_hrs1`	4-bit Bus	Stunden 1er
`dcf_hrs10`	2-bit Bus	Stunden 10er
`dcf_wday`	3-bit Bus	Wochentag
`dcf_day1`	4-bit Bus	Kalendertag 1er
`dcf_day10`	2-bit Bus	Kalendertag 10er
`dcf_month1`	4-bit Bus	Monat 1er
`dcf_month10`	1-bit	Monat 10er
`dcf_year1`	4-bit Bus	Jahr 1er
`dcf_year10`	4-bit Bus	Jahr 10er

- Bei der Realisierung des Automaten sind mehrere Varianten denkbar:
  - zwei gekoppelte Automaten, einer für die Sekunden, ein zweiter für die Minuten.
  - ein synchroner Automat der beide Funktionen beinhaltet.

  Tipp: Am einfachsten erscheint mir folgende Variante: Ein Automat, der zusätzlich über externe Zähler (Anzahl der Fehler, Anzahl der Millisekunden, aktuelle Sekundenposition…) und über Zustandsregister/Flags (derzeitiges Nutzbit…) gesteuert wird. Der Automat selber kommt dann mit wenig internen Zuständen, entsprechend den Phasen des DCF-Signals, aus.

  Bei der Entwicklung empfiehlt es sich, erst eine Maschine zu entwerfen, die sich auf die Sekunden einsynchronisieren und diese erkennen kann. Anschließend kann der Automat über zusätzliche Register erweitert werden um die Minuten zu erkennen und die Nutzinformation auszuwerten.

- Für die *sichere* Auswertung des (möglicherweise gestörten) DCF 77 Signals können die unterschiedlichsten Strategien verfolgt werden. Es muss nur berücksichtigt werden, dass
  – Zeitverschiebungen der Flanken auftreten können.
  – Peaks/Einbrüche in den Pegeln möglich sind.

  Als Beispiel werden jetzt zwei (mögliche) Zeitschema vorgestellt:

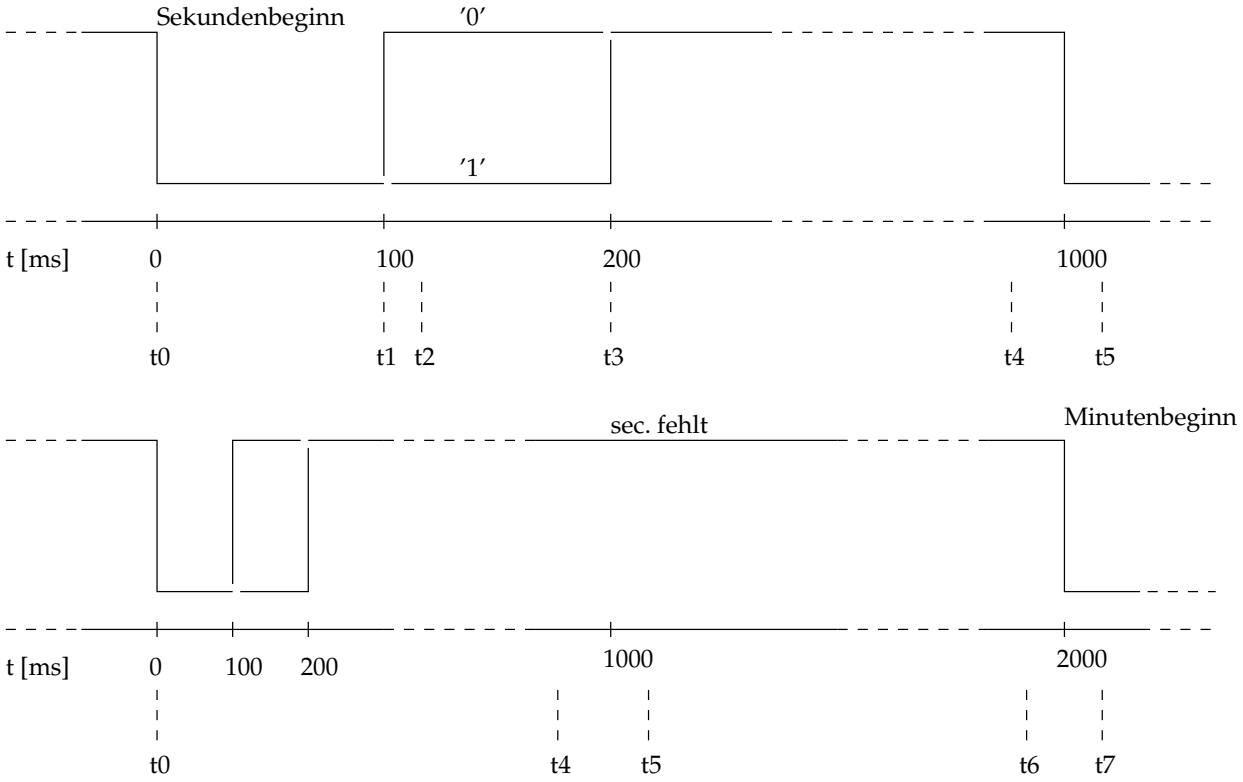


Abbildung 3: Auswertung des DCF 77 Signals – Beispiel 1

t_0	Sekundenbeginn, Übergang in aktiven Zustand
$t_0 - t_1$	0-Pegel, höchstens n-Fehler
$t_1 - t_2$	Entscheidung ob '0' oder '1' übertragen wird als Majoritätsprüfung (Maximum von n-Samples, Erster Wert mit n-Samples…)
$t_2 - t_3$	Pegel muss dem in $t_1 - t_2$ festgelegten Wert entsprechen, max. n-Fehler
$t_3 - t_4$	1-Pegel, höchstens n-Fehler
$t_4 < 1000$	Übergang in Wartezustand (*vor* nächster Rückflanke) um sich bei Taktverschiebungen auf den folgenden Sekundenbeginn einzusynchronisieren
$t_4 - t_5$	Zeitintervall für nächsten *gültigen* Sekundenbeginn
$t_5 - t_6$	1-Pegel, höchstens n-Fehler
$t_6 - t_7$	Zeitintervall für nächsten *gültigen* Minutenbeginn, wenn der Sekundenbeginn bei $t_4 - t_5$ ausbleibt

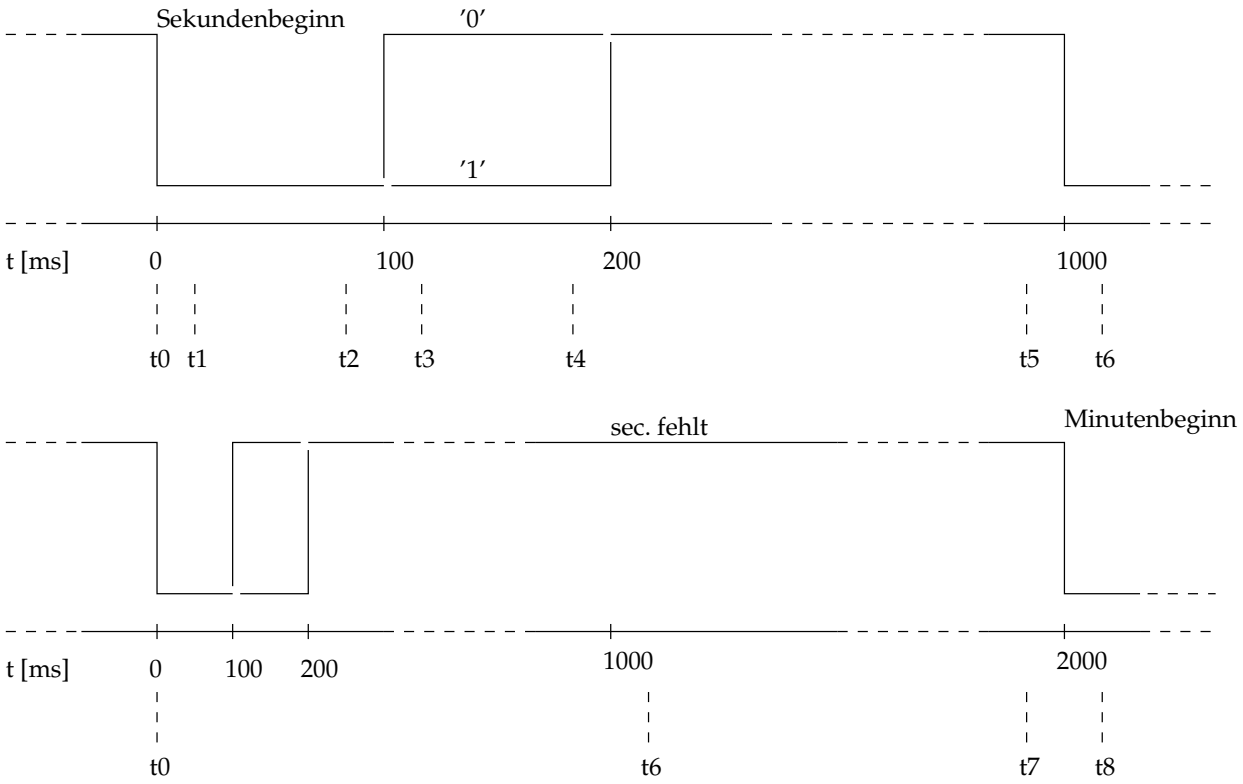


Abbildung 4: Auswertung des DCF 77 Signals – Beispiel 2

t_0	Sekundenbeginn, Synchronisation des Automaten
$t_0 - t_1$	keine Auswertung, Unsicherheitsintervall
$t_1 - t_2$	0-Pegel, höchstens n-Fehler
$t_2 - t_3$	keine Auswertung, Unsicherheitsintervall
$t_3 - t_4$	Entscheidung ob '0' oder '1' übertragen wird: n% der Samples muss von einer Sorte sein
$t_4 - t_5$	keine Auswertung
$t_5 < 1000$	Übergang in Wartezustand (*vor* nächster Rückflanke) um sich bei Taktverschiebungen auf den folgenden Sekundenbeginn einzusynchronisieren
$t_5 - t_6$	Zeitintervall für nächsten *gültigen* Sekundenbeginn
$t_6 - t_7$	keine Auswertung, wenn der Sekundenbeginn bei $t_5 - t_6$ ausbleibt
$t_7 - t_8$	Zeitintervall für nächsten *gültigen* Minutenbeginn, wenn der Sekundenbeginn bei $t_5 - t_6$ ausbleibt

Beide Methoden zählen „Fehler“ in dem Eingangssignal, um Aussagen über die Empfangsgüte zu machen. Alternativ dazu kann man das Signal auch vorverarbeiten und damit Peaks/Einbrüche herausfiltern. So kann beispielsweise über eine feste Anzahl von Samples die Majorität als Ausgangssignal genommen werden. Dieses wird dann vom eigentlichen Automaten verarbeitet, der dann aber keinerlei Fehlertoleranz mehr aufweisen muss und dementsprechend einfacher wird.

Aufgabe 4.2

Bauen Sie die Gesamtschaltung der **DCF 77 gesteuerten Weckuhr** auf, wie in Abbildung 5 skizziert. Die Schaltung soll dann als FPGA-Prototyp realisiert und bis zum fertigen Standardzell-Layout entworfen werden.

Entsprechend der oben beschriebenen Funktionsweise der DCF-Decoders müssen einige der zuvor entworfenen Komponenten modifiziert werden:

`timblk` : wird erweitert, um die Werte aus `dcffsm` übernehmen zu können.

`dcfload = '1'` Register für die Sekunden wird zurückgesetzt und die Werte für Minuten und Stunden werden in die entsprechenden Register geladen.

`clkgen` : wird modifiziert, so dass der abgeleitete Sekundentakt durch `dcfload` mit den Sekunden des DCF-Signals synchronisiert wird.

`dcfload = '1'` der Sekundentakt wird so synchronisiert, dass nach 1 sek. $\pm\,\varepsilon$ die nächste Vorderflanke erzeugt wird.

Achtung: die Synchronisation betrifft *nur* den Sekundentakt, der Takt für `clk1ms` der den DCF-Empfänger steuert, muss davon unbeeinflusst weiter erzeugt werden.

Außerdem muss dafür gesorgt werden, dass `timblk` und ggf. `outmux` Taktsignale erhalten, um die „gültigen“ Werte des DCF-Empfängers (`dcf_...`) zu laden.

`outmux` : gibt zusätzlich neben der Zeit das Datum aus.

`dcfload = '1'` die Datumswerte werden in interne Register geladen.

`disp_wday = '1'` $\wedge$ `dcfsok = '1'` der Wochentag wird auf der Anzeige ausgegeben, vorausgesetzt der Empfang des DCF-Signals ist gewährleistet.

`disp_date = '1'` $\wedge$ `dcfsok = '1'` das Datum wird auf der Anzeige ausgegeben, vorausgesetzt der Empfang des DCF-Signals ist gewährleistet.

Abhängig von der Implementation der Wochentags-/Datumsausgabe (eine oder mehrere Anzeigen, getrennte LEDs für den Tag ...) können sich hier natürlich noch Änderungen ergeben. Außerdem sind Erweiterungen denkbar, wie die Ausgabe eines „richtig empfangenen“ Datums, unabhängig von `dcfsok`, bis der nächste Tag beginnt oder die Uhr manuell gestellt wird.

Hier noch einige Überlegungen zur Beschaltung der Multiplex BCD-Anzeige:

- Um die Anzeige flimmerfrei zu halten, sollte eine Wiederholfrequenz > 100 Hz erreicht werden — Durchschalten der Bits in `seldgt`.
- Damit das Tastverhältnis (Zeitanteil wann eine Ziffer leuchtet zu der Dunkelzeit) nicht zu klein wird, was zu einer vergleichsweise *dunklen* Ausgabe führen würde, ist unter Umständen der Ausgangsbus `decoded` doppelt auszuführen (`seldgt` würde dann jeweils zwei Ziffern aktivieren).

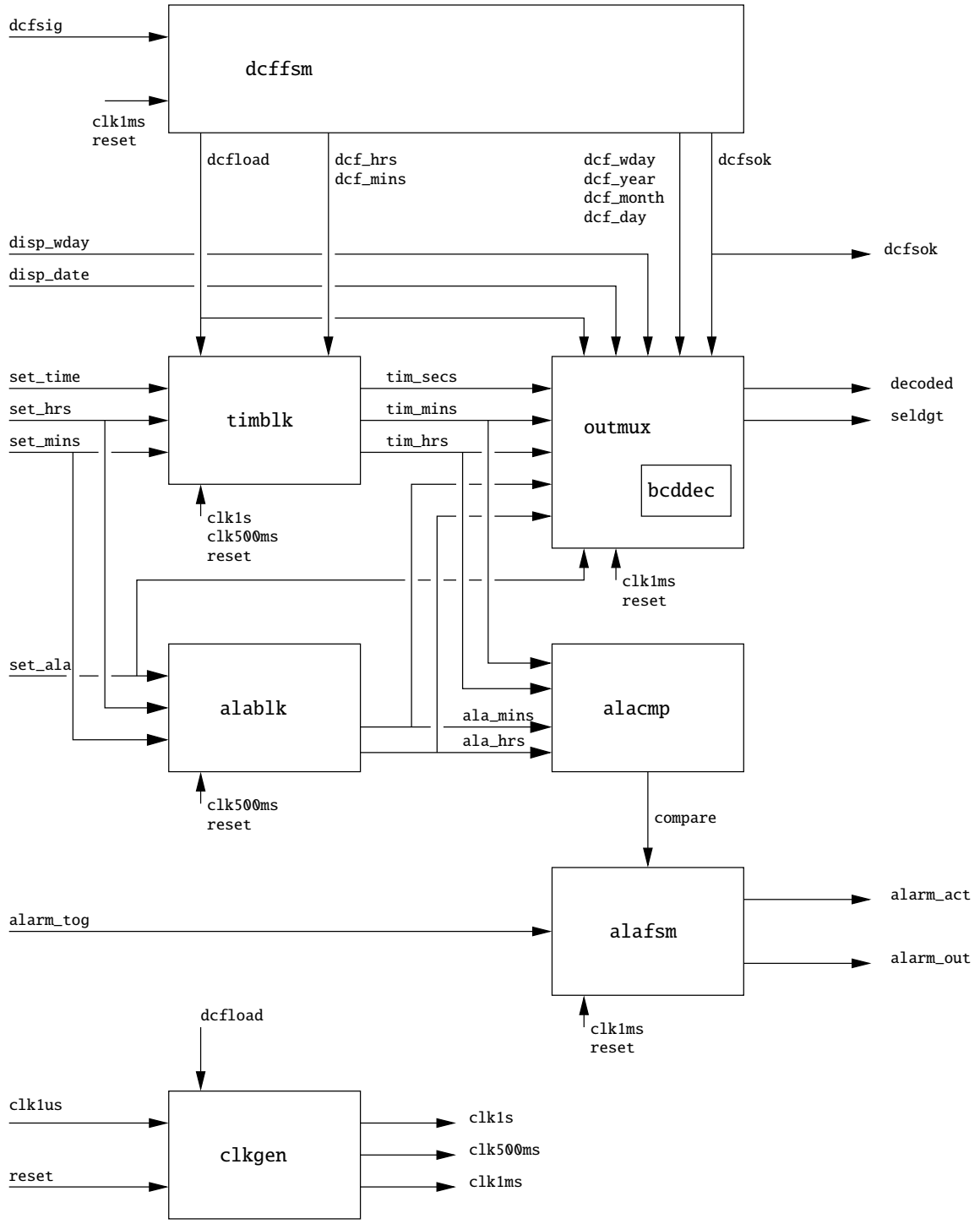


Abbildung 5: Blockstruktur der DCF 77 gesteuerten Weckuhr