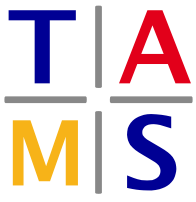
VHDL Schaltungs- und Systementwurf

3

Entwurf einer Weckuhr



Andreas Mäder

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

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

Jetzt wird ein Wecker, als kleines System gekoppelter Automaten und Komponenten, mit Hilfe der VHDL Simulations- und Syntheseprogramme entworfen. Dabei sollen die Erfahrungen der letzten beiden Aufgabenblätter, zum Umgang mit den Werkzeugen sowie zur synthesegerechten VHDL-Modellierung, eingebracht werden.

Arbeitsweise

Es soll ein hierarchisch aufgebauter, synthetisierbarer VHDL-Entwurf erstellt werden — in Hinblick auf eine spätere Standardzell- oder FPGA-Realisierung.

1. Entwerfen Sie synthetisierbare VHDL Verhaltensmodelle auf Register-Transfer Ebene für alle Teilkomponenten (Blöcke) der Schaltung.
2. Simulieren Sie diese Schaltungen in geeigneten Testumgebungen, um die Funktionalität zu prüfen.
3. (optional) Neben der Funktion ist auch die Synthetisierbarkeit der VHDL-Eingabe, durch Abbildung der (Teil-)Entwürfe auf die Zellbibliothek, zu überprüfen.
4. Anschließend wird der Wecker aus den einzelnen Komponenten als Strukturbeschreibung zusammengesetzt.
5. Auch diese Beschreibung muss simuliert werden.
6. Die Gesamtschaltung ist zu synthetisieren und anschließend als VHDL-Gatternetzliste zu simulieren.

Aufgabe 3.1

Entsprechend der oben beschriebenen Vorgehensweise sollen Sie einen **Wecker** entwerfen. Dabei sind keine besonderen Vorgaben einzuhalten, sie können also Ihren Ideen freien Lauf lassen...

Eine mögliche Unterteilung der Schaltung in einzelne Komponenten ist in Abb. 1 dargestellt, dazu gleich einige Anmerkungen:

- Die Skizze stellt hier nur einen ersten Schritt in einem hierarchischen Entwurf dar. Weitere Unterteilungen, außer der skizzierten Referenz auf `bcddec` in `outmux`, können zwar vorgenommen werden, sind aber nach den bisherigen Erfahrungen nicht notwendig.
- Zusätzliche Blöcke auf dieser Ebene könnten bei erweiterter Funktionalität benötigt werden, beispielsweise für zentrale Kontrolleinheiten.
- Die nachfolgenden Beschreibungen der einzelnen Komponenten sollen jeweils nur einen ersten Eindruck von einer „Mindestfunktionalität“ liefern. Erweiterungen sind sicherlich denkbar für:
  - komfortablere Einstellmöglichkeiten für die Uhr- und Weckzeit:
    vorwärts-/rückwärts zählen
    mehrere Geschwindigkeiten
    ...
  - erweiterte Weckfunktionen:
    Nachweckautomatik
    mehrere (datumsabhängige, s. Aufgabe **4**) Weckzeiten
    ...
- Zusätzliche Ein- und Ausgänge sind zwar möglich, es ist aber darauf zu achten, dass ihre Zahl nicht zu sehr steigt, da der Entwurf sonst als Standardzellentwurf *padlimitiert* wird (d.h. die Fläche des Entwurfs wird durch die Anzahl der I/O-Pads bestimmt und nicht durch die Komplexität der internen Logik).

  Werden noch viele zusätzliche Ein- oder Ausgänge benötigt, so ist zu überlegen, ob sich durch geeignete Multiplexlösungen, die zu einer Mehrfachbenutzung der Eingänge führen, nicht wieder externe Anschlüsse einsparen lassen.
- Um die Ausgabe einfacher zu realisieren und um das spätere Interface zu dem DCF 77 Decoder zu vereinfachen, sollte für die interne Zahlendarstellung eine BCD-Codierung gewählt werden. Vorteile: einfache Ein-/Ausgabe; Nachteil: die Arithmetik muss man selber „schreiben“.

  Die Ein- und Ausgänge der einzelnen Strukturelemente können jeweils hinsichtlich der tatsächlich benötigten Bitbreiten minimiert werden — dementsprechend wird keine „richtige“ BCD-Codierung benutzt.

Wie schon oben erwähnt ist die Beschreibung der Blöcke als ein erster Vorschlag und nicht als vollständige Spezifikation zu verstehen — entsprechende Beispiele sind in dem Verzeichnis `templates` vorhanden, sie können aber je nach Bedarf modifiziert werden!

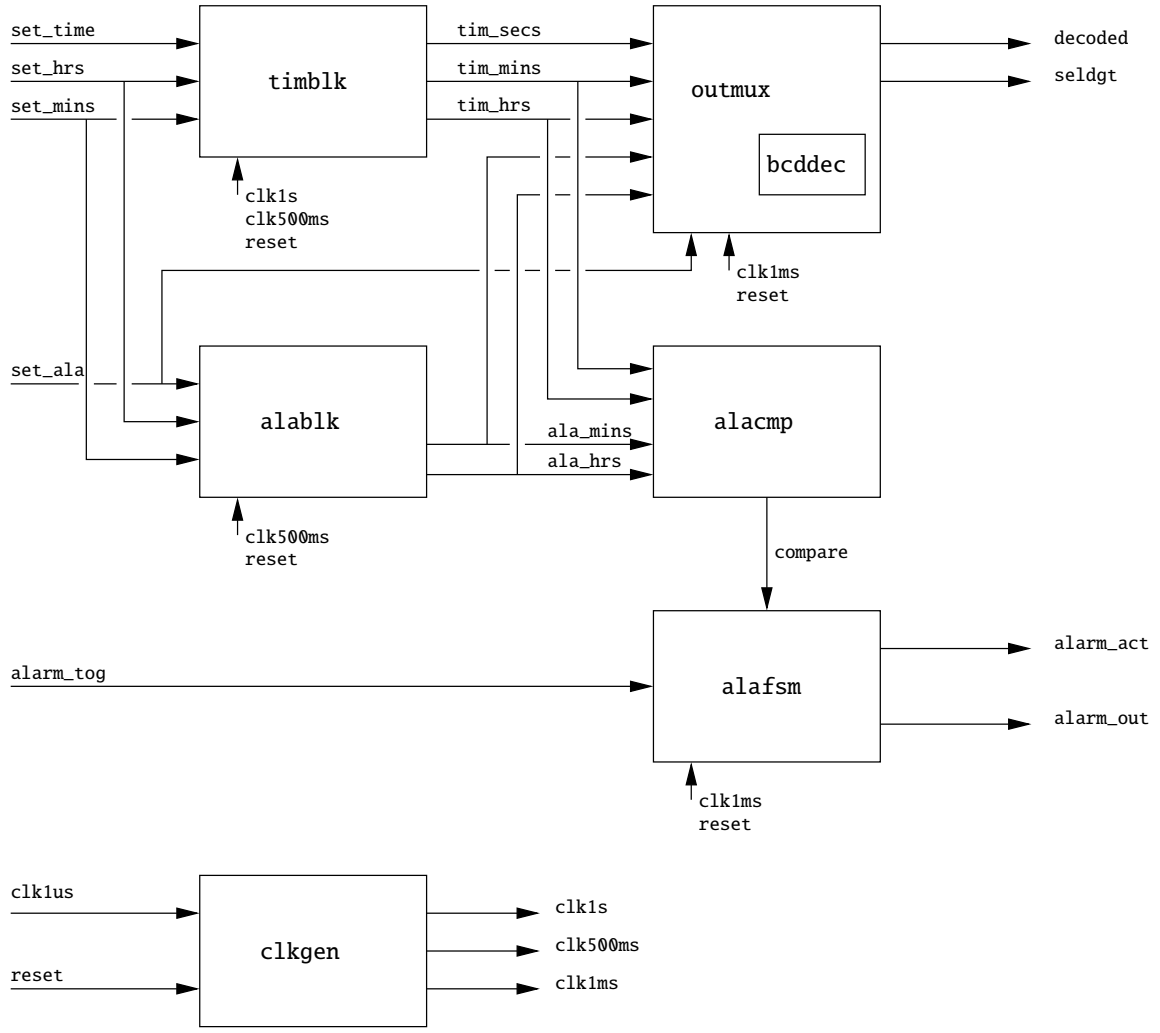


Abbildung 1: Blockstruktur der Weckuhr

`clkgen` : Taktuntersetzer, für die intern benötigten Arbeitstakte.

Signal	Wirkung	Funktion
`reset`	active Low	asynchrones Reset
`clk1us`	Vorderflanke	externer Takt 1MHz
`clk1ms`	Flanke	interner Takt 1kHz
`clk500ms`	Flanke	interner Takt 2Hz
`clk1s`	Flanke	interner Takt 1Hz

`alafsm` : Automat zur Wecksteuerung, siehe Aufgabenblätter **1** und **2**.

Neben der hier vorgestellten Partitionierung in `alafsm` und `alacmp`, ist es auch möglich die Funktion des Vergleichers direkt in `alafsm` zu integrieren.

timblk : entspricht dem eigentlichen Uhrwerk.

`set_time = '0'` die Uhrzeit wird im Sekundentakt raufgezählt, dazu sind entsprechende Modulo-Zähler zu entwerfen.

`set_time = '1'` setzt die Sekunden auf "00" zurück und bereitet das manuelle Stellen der Uhrzeit vor.

`set_time = '1'` $\wedge$ `set_hrs = '1'` Uhrzeit stellen: zählt die Stunden mit der 500ms-Clock (rauf).

`set_time = '1'` $\wedge$ `set_mins = '1'` Uhrzeit stellen: zählt die Minuten mit der 500ms-Clock (rauf), dabei findet kein Stundenübertrag statt.

Tipp: Der Aufbau als *synchroner* Zähler ist am einfachsten zu implementieren und am sichersten im Betrieb.[1]

Signal	Wirkung	Funktion
reset	active Low	asynchrones Reset
clk1s	Vorderflanke	„normaler“ Takt
clk500ms	Vorderflanke	Takt zum Stellen
set_time	active High	Stellen der Uhrzeit
set_hrs	active High	Stellt Stunden
set_mins	active High	Stellt Minuten
tim_secs1	4-bit Bus	Sekunden 1er
tim_secs10	3-bit Bus	Sekunden 10er
tim_mins1	4-bit Bus	Minuten 1er
tim_mins10	3-bit Bus	Minuten 10er
tim_hrs1	4-bit Bus	Stunden 1er
tim_hrs10	2-bit Bus	Stunden 10er

alablk : Register und Zähler für den Weckmechanismus.

`set_alarm = '1'` $\wedge$ `set_hrs = '1'` Weckzeit stellen: zählt die Stunden mit der 500ms-Clock (rauf).

`set_alarm = '1'` $\wedge$ `set_mins = '1'` Weckzeit stellen: zählt die Minuten mit der 500ms-Clock (rauf), dabei findet kein Stundenübertrag statt.

Signal	Wirkung	Funktion
reset	active Low	asynchrones Reset
clk500ms	Vorderflanke	Takt zum Stellen
set_alarm	active High	Stellen der Alarmzeit
set_hrs	active High	Stellt Stunden
set_mins	active High	Stellt Minuten
ala_mins1	4-bit Bus	Minuten 1er
ala_mins10	3-bit Bus	Minuten 10er
ala_hrs1	4-bit Bus	Stunden 1er
ala_hrs10	2-bit Bus	Stunden 10er

[1] Alle Stellen des Zählers werden in einem VHDL-Prozess – mit einer Clock – gesetzt.

alacmp : Vergleicher für das Auslösen des Alarms.

Signal	Wirkung	Funktion
tim_mins1	4-bit Bus	Uhr-Minuten 1er
tim_mins10	3-bit Bus	Uhr-Minuten 10er
tim_hrs1	4-bit Bus	Uhr-Stunden 1er
tim_hrs10	2-bit Bus	Uhr-Stunden 10er
ala_mins1	4-bit Bus	Alarm-Minuten 1er
ala_mins10	3-bit Bus	Alarm-Minuten 10er
ala_hrs1	4-bit Bus	Alarm-Stunden 1er
ala_hrs10	2-bit Bus	Alarm-Stunden 10er
compare	active High	Uhrzeit = Alarmzeit

outmux : steuert die 7-Segment Anzeige an.

`set_alarm = '1'` zum Stellen der Weckzeit wird die Ausgabe auf die aktuelle Weckzeit umgeschaltet; normalerweise wird die Uhrzeit angezeigt.

- Um nicht für jede Ziffer einen eigenen 7-bit Bus als Ausgang vorzusehen, soll die Ausgabe im Zeitmultiplex betrieben werden. Wegen der Trägheit des Auges kann man die Ziffern auf 7-Segment LEDs nacheinander ausgeben, dazu werden zwei Ausgangsbusse benutzt: einer aktiviert genau eine einzige der sechs Anzeigen (seldgt), der andere liefert den zugehörigen Wert der 7-Segmente (decoded) parallel an alle Anzeigen. Über eine Zeitschleife werden beide Werte (gleichzeitig) weitergeschaltet.
- Eine Erweiterung der Funktion wäre, dafür zu sorgen, dass die Ziffer Null an der Stelle der 10-er Stunden nicht angezeigt wird, sondern diese Stelle der Anzeige leer bleibt.

Signal	Wirkung	Funktion
reset	active Low	asynchrones Reset
clk1ms	Vorderflanke	Takt für internen Automaten
set_alarm	active High	Umschalten auf Alarmzeit
tim_secs1	4-bit Bus	Uhr-Sekunden 1er
tim_secs10	3-bit Bus	Uhr-Sekunden 10er
tim_mins1	4-bit Bus	Uhr-Minuten 1er
tim_mins10	3-bit Bus	Uhr-Minuten 10er
tim_hrs1	4-bit Bus	Uhr-Stunden 1er
tim_hrs10	2-bit Bus	Uhr-Stunden 10er
ala_mins1	4-bit Bus	Alarm-Minuten 1er
ala_mins10	3-bit Bus	Alarm-Minuten 10er
ala_hrs1	4-bit Bus	Alarm-Stunden 1er
ala_hrs10	2-bit Bus	Alarm-Stunden 10er
decoded	7-bit Bus	7-Segmente
seldgt	6-bit Bus	6-Stellen