Entwurf eines Aufzugs in VHDL

Frank Pilhofer

Inhaltsverzeichnis

1. Überblick 2
2. Der Aufzug 2
    2.1. Spezifikation der Ports 3
    2.2. Aufzugsautomat 3
3. Der Controller 4
    3.1. Spezifikation der Ports 5
    3.2. Die Mathematik des Controllers 5
    3.3. Interne Variablen und Signale 6
4. Das System 7
5. Aufgaben 7
    5.1. Entwurf einer struktur-orientierten Beschreibung des Controllers 7
    5.2. Gerechtigkeit 7
    5.3. Sicherheit 8
6. Anhang 9
    6.1. Der Aufzug in VHDL 9
    6.2. Der Controller in VHDL 10
    6.3. Das System in VHDL 11
    6.4. Beispielsimulationen 12

Dieses Dokument ist auch im WWW abrufbar

`http://www.uni-frankfurt.de/~fp/AlI/Aufzug/`

1. Überblick

Dieser Text beschreibt und erläutert die Implementation des Aufzugsystems in VHDL. Einige Simulationsbeispiele befinden sich im Anhang. Das gesamte System zeichnet sich durch seine Flexibilität aus. Es ist nicht auf 3 Stockwerke beschränkt, sondern fährt beliebig viele Stockwerke an. Die genaue Anzahl wird mit einem *generic* Parameter festgesetzt. Um dies zu erreichen, verwendet die Implementation die Möglichkeiten zur Bitmanipulation von *Smax*. Die Programme können nur nach vorheriger Übersetzung des *Smax*-Paketes kompiliert und simuliert werden.

Die Implementation teilt sich in drei Teile. Der erste Teil ist der Aufzug selber, womit der sich bewegende Kasten gemeint ist. Der Aufzug ist als dummer Automat konstruiert; er erhält Befehle und führt sie aus. Der zweite Baustein ist der Controller, er wertet den Zustand des Aufzuges aus, nimmt die Bedienungen der Passagiere entgegen und gibt dem Aufzug dementsprechende Befehle zurück. Diese ersten zwei Bausteine sind zwar separat implementiert, können jedoch nicht einzeln existieren. Aus diesem Grunde wurde der dritte Teil, abstrakt *System* genannt, geschaffen. Das *System* verkabelt beide Bausteine miteinander und legt die Anzahl der zu befahrenden Stockwerke fest.

Um die gewünschte Flexibilität zu erreichen, mußte ein Weg gefunden zu werden, die Stockwerke beliebig skalierbar zu verwalten. Die Lösung ist, das "Gebäude" als Bitfeld zu kodieren. Jedes Stockwerk wird durch ein Bit in diesem Bitfeld repräsentiert; das oberste Stockwerk das höchstwertige Bit, und das Erdgeschoß das unterste Bit.

Dieser Text ist als Beschreibung und Dokumentation zum VHDL-Text gedacht. Die VHDL-Beschreibung ist ohne diesen Text nicht allzu verständlich, und dieser Text macht ohne das danebenliegende Programm wahrscheinlich auch recht wenig Sinn.

2. Der Aufzug

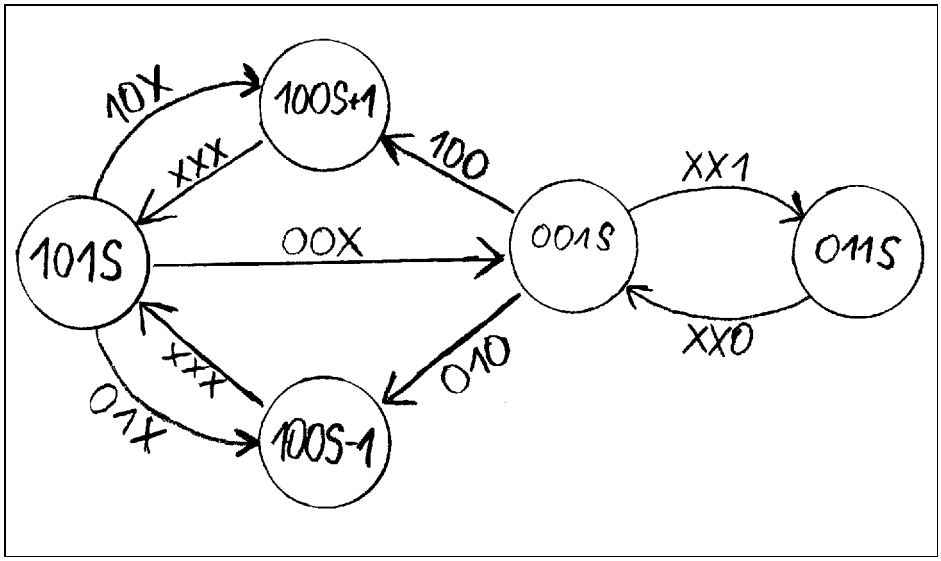
Wie bereits erwähnt ist der Aufzug ein blöder Automat, der einfach nur das tut was man ihm sagt. Um die Aufgabe noch etwas interessanter zu gestalten, hat der Aufzug eine Tür, die korrekt geöffnet und geschlossen werden muß. Weiterhin hält der Aufzug nicht in Stockwerken an, die nicht bedient werden sollen.

2.1. Spezifikation der Ports

Name	Richtung	Typ	Beschreibung
progress	in	bit	Bei jeder aufsteigenden Flanke dieses Signals führt der Automat einen Zustandsübergang aus
go_up	in	bit	Befehl, aufwärts zu fahren
go_down	in	bit	Befehl, abwärts zu fahren
open_door	in	bit	Wenn 1, dann soll die Tür geöffnet werden, wenn 0, dann soll die Tür geschlossen werden oder geschlossen bleiben.
at	out	bit_vector	Das Stockwerk, in dem sich der Aufzug befindet. Wenn sich der Aufzug bewegt, dann wird das Stockwerk angezeigt, in dem der Aufzug als nächstes ankommen wird. Genau ein Bit in diesem Vektor ist gesetzt.
door_open	out	bit	Status der Tür, ob geöffnet (1) oder geschlossen (0)
moving	out	bit	Bewegungszustand: Aufzug bewegt sich (1) bzw. Aufzug ist gestoppt (0)

2.2. Aufzugsautomat

Der Aufzug kann wie folgt als Automat beschrieben werden. Da sich der Automat auf jedem Stockwerk gleich verhält, ist hier nur ein Zyklus dargestellt. "*S*" ist die Kodierung des jeweiligen Stockwerks.



Zustandskodierung: ´moving´ ´door_open´ ´mvwait´ ´at´
Kodierung der Eingabe: ´go_up´ ´go_down´ ´open_door´

Die Zustände können wie folgt beschrieben werden:

001S Sozusagen der Basiszustand. Der Aufzug steht mit geschlossener Tür wartend in einem Stockwerk. Von hier kann der Aufzug entweder nach oben oder unten fahren sowie die Tür öffnen.

011S Die Tür ist geöffnet. Die Tür muß vor der Abfahrt erst wieder geschlossen werden.

100S+1 Der Aufzug hat sich nach oben in Bewegung gesetzt.

100S-1 Der Aufzug hat sich nach unten in Bewegung gesetzt.

101S Der Aufzug ist kurz davor, das nächste Stockwerk zu erreichen. Von diesem Zustand aus kann der Aufzug entweder anhalten (zurück nach 001S) oder aber nach oben oder unten weiterfahren.

Es ist sofort ersichtlich, daß diese Spezifikation nicht Narrensicher ist. Etliche Übergänge des Automaten sind unspezifiziert. So darf es z.B. nicht passieren, daß im Zustand **011S** der Befehl zur Abfahrt gegeben wird, ohne die Tür zu schließen. Um so etwas zu verhindern, gibt es ja auch den Controller.

3. Der Controller

Nachdem an dem Aufzug einiges an Intelligenz gespart wurde, benötigt der Controller gleich eine doppelte Portion davon. Er muß den Aufzug steuern und muß dabei aufpassen, diesen nicht in unspezifizierte Zustände zu schicken. Auch sollte der Aufzug weder in den Boden gerammt noch aus der Decke herausgeschossen werden. Gleichzeitig muß der Controller die Aufträge von außen entgegennehmen und diese früher oder später möglichst gerecht bedienen. Und zum Schluß war da noch das berühmte Kaugummi auf dem Knopf, d.h. auch ein permanentes Knopfdrücken in einem beliebigen Stockwerk darf den Aufzug nicht zu sehr von seinem korrekten Tun ablenken.

Es zeigt sich, daß gerade dieses Kaugummi ein größeres Problem für die Logik darstellt. Eine einfache Möglichkeit wäre es, nicht auf gedrückte Knöpfe zu reagieren sondern auf die Taktflanke eines Knopfes. Das führt jedoch zu Fehlverhalten des Aufzuges: Sobald das Kaugummi auf dem Knopf klebt, wird es in diesem Stockwerk keine Taktflanke mehr geben. Passagiere, die in diesem Stock zusteigen wollen, dürfen sich dann wundern, weshalb der Aufzug stets an ihnen vorbeifährt.

Auch stellt sich die Frage, wie mit Knopfdrücken im Stockwerk, in dem sich der Aufzug gerade befindet, umgegangen wird. Auch hier gibt es die zunächst offensichtliche Methode, dieses offensichtlich unsinnige Drücken zu ignorieren. Bei näherer Betrachtung ist das Drücken jedoch nicht immer sinnlos. Wenn der Aufzug seine Tür geschlossen hat, aber noch nicht abgefahren ist, so darf der Knopfdruck natürlich nicht vergessen werden. Vielmehr wird von dem Aufzug erwartet, daß er seine Tür noch einmal öffnet. Leider kommt hier wieder das Kaugummi ins Spiel; falls ein Kaugummi auf dem Knopf klebt, darf das den Aufzug natürlich nicht am Abfahren hindern.

Es wird daher der folgende Algorithmus gewählt:

- Generell werden alle Knopfdrücke im aktuellen Stockwerk ignoriert, mit den folgenden Ausnahmen:
- Flankenwechsel von '0' auf '1' werden nicht ignoriert
- Wenn der Aufzug sich bereits bewegt, oder zumindest der Befehl zur Abfahrt bereits gegeben wurde, wird der Knopfdruck auch entgegengenommen
- Wenn der Aufzug sowieso nichts zu tun hat, wird der Knopfdruck auch akzeptiert.

Eine kleine Lücke in der ganzen Intelligenz bereitet das Kaugummi noch immer. Sobald jetzt ein Knopf ständig gedrückt ist, und keine anderen Aufträge anliegen, stellt sich der Aufzug in das jeweilige Stockwerk und macht ständig die Tür auf und zu und auf und zu und ... aber ansonsten würde ja auch niemand den Kaugummi bemerken und endlich mal entfernen.

3.1. Spezifikation der Ports

Name	Richtung	Typ	Beschreibung
inknopf	in	bit_vector	Bedienungsknöpfe innerhalb des Aufzugs
outknopf	in	bit_vector	Bedienungsknöpfe in den Stockwerken. ***inknopf*** und ***outknopf*** werden identisch behandelt.
at	in	bit_vector	Standort des Liftes. Wenn sich der Aufzug noch bewegt, zeigt dieser Port das Stockwerk an, das als nächstes erreicht wird. Nur ein Bit dieses Vektors ist gesetzt.
door_open	in	bit	Zustand der Aufzugstür: offen (1) oder geschlossen (0)
moving	in	bit	Bewegungszustand des Aufzugs: fahrend (1) oder stehend (0)
go_up	out	bit	Befehl an den Aufzug, aufwärts zu fahren
go_down	out	bit	Befehl an den Aufzug, abwärts zu fahren
open_door	out	bit	Befehl an den Aufzug, die Tür zu öffnen (1) bzw. die Tür zu schließen oder sie geschlossen zu halten (0)

3.2. Die Mathematik des Controllers

Die Bitfeldmathematik aus dem *Smax*-Packet hilft dem Controller, die Knöpfe zu verwalten und die Befehle zu berechnen. Alle Knopfdrücke werden in dem Bitfeld ***requests*** verwaltet. Wenn ein Bit dieses Feldes '1' ist, so muß dieses Stockwerk bedient werden. Zusammen mit dem Port ***at*** ergeben sich die folgenden Berechnungsmöglichkeiten:

requests and ***at***	ist genau dann '1', wenn das aktuelle Stockwerk bedient werden soll. Da at das nächste Stockwerk anzeigt, solange sich der Aufzug bewegt, zeigt dieser Ausdruck, ob der Aufzug im nächsten Stockwerk anhalten soll. Wird dieser Ausdruck '1', so halten wir also den Aufzug an, indem ***go_up*** und ***go_down*** beide auf '0' gesetzt werden.
requests and (***at*** - 1)	(***at***-1) ist eine Maske, in der alle Bits unterhalb des aktuellen Stockwerks auf '1' stehen. Wenn dieser Ausdruck ungleich 0 ist, so gibt es Stockwerke unterhalb des aktuellen, die bedient werden müssen.
requests and not (***at*** - 1)	not (***at***-1) ist eine Maske, in der alle Bits oberhalb des aktuellen Stockwerks auf '1' stehen. Wenn dieser Ausdruck ungleich 0 ist, so gibt es Stockwerke oberhalb des aktuellen, die bedient werden müssen.
requests = 0	In diesem Fall hat das System nichts mehr zu tun.

3.3. Interne Variablen und Signale

Die Variablen und Signale, die innerhalb des Controllers verwendet werden, benötigen wahrscheinlich auch noch einige Erklärung. Das Problem des Controllers ist die Asynchronizität, mit der er fertig werden muß. Auf jede Änderung von außen muß er reagieren, doch leider halten sich diese äußeren Ereignisse nicht immer an den Zeitplan. Die internen Variablen existieren einzig zur Koordination. So ist beispielsweise der Fall denkbar, daß (a) der Controller den Befehl zur Abfahrt in einem Stockwerk gibt; (b) bevor der Aufzug sich in Bewegung gesetzt hat, wird der Knopf des aktuellen Stockwerks gedrückt. Da der Port ***moving*** noch auf 0 steht, denkt der Controller, daß der Aufzug noch im Stockwerk steht und gibt den Befehl, die Tür zu öffnen. (c) Die Tür öffnet sich, und mit offener Tür fährt der Aufzug los. Deshalb muß sich der Controller selber merken, welche Befehle er schon erteilt hat.

idle Dieses Signal wird auf '1' gesetzt, falls keine weiteren Stockwerke anzufahren sind. In diesem Fall wird ein permanenter Knopfdruck im aktuellen Stockwerk nicht mehr ignoriert, so daß die Tür im aktuellen Stockwerk wieder geöffnet wird.

movereq Wird auf '1' gesetzt, wenn der Befehl zur Abfahrt erteilt wird, und zurückgesetzt, wenn der Aufzug tatsächlich am Fahren ist, um genau den oben beschriebenen Effekt zu vermeiden. Sobald dieser Befehl erteilt wurde, werden auch wieder Knopfdrücke im Stockwerk nicht mehr ignoriert.

oldreq speichert den letzten Status der Knöpfe zur Feststellung einer Taktflanke eines beliebigen Knopfes.

doorstat Dieser Bitvektor besteht aus 2 Bits, in denen eingetragen ist, welcher Befehl an die Tür gegeben wurde. ***doorstat(0)*** ist '1', wenn der Befehl "Tür schließen" gegeben aber noch nicht ausgeführt wurde; ***doorstat(1)*** ist '1', wenn der Befehl "Tür öffnen" gegeben aber noch nicht ausgeführt wurde. Diese 2 Bits sind wahrscheinlich die obskursten im Controller, doch zumindest das letztere ist durchaus nötig. Es verhindert, daß der Befehl zum Abfahren gegeben wird bevor sich die Aufzugstür ordnungsgemäß geschlossen hat.

going_up Diese Variable ist zentral für die Gerechtigkeit des Systems. Falls diese Variable '1' ist, so fährt der Aufzug nach oben, falls '0', so fährt er nach unten. Die Fahrtrichtung wird nur geändert, wenn in der jeweiligen Richtung keine Anforderungen jenseits des aktuellen Stockwerks mehr vorliegen.

4. Das System

Aufzug und Controller sind die zwei Teile des Gesamtsystems, und dieses kleine Programmstück ist der Kleber zwischen ihnen. Hier wird die Anzahl der zu versorgenden Stockwerke festgelegt; die Ausgänge des Aufzugs werden mit den Eingängen des Controllers verkabelt und umgekehrt. Eine kleine Aufgabe übernimmt das System allerdings doch: es versorgt den Aufzugsautomaten mit einem Taktsignal. Alle 100ns wird das ***progress***-Signal invertiert, so daß der Aufzugsautomat alle 200ns einen Zustandsübergang vollführt.

Die Vektoren ***inknopf*** und ***outknopf*** werden überhaupt nicht verkabelt. Doch von hier aus können sie im Simulator mit *force*-Anweisungen nach Belieben verändert werden.

5. Aufgaben

5.1. Entwurf einer struktur-orientierten Beschreibung des Controllers

Auch schon bei einer moderaten Gebäudehöhe von 5 Stockwerken besitzt der Controller bereits 15 Bits an internen Zuständen und 3 Ausgabebits, d.h. der komplette Automat hätte $2^{15+3}=262144$ Zustände. Der Controller hat in dieser Konfiguration (da Gleichbehandlung von inneren und äußeren Knöpfen) 12 Eingangsbits. Auch unter der Annahme, daß nur ein kleiner Prozentsatz der Zustände erreichbar ist, ergäbe das ganze noch ein eindrucksvolles Wirrwarr, das ich mir hier erspare.

5.2. Gerechtigkeit

- Die Tür des Aufzuges wird immer geschlossen.
- Ist die Tür geschlossen, und existiert keine Anfrage im aktuellen Stockwerk, und existieren Anfragen in anderen Stockwerken, so wird auf jeden Fall der Befehl zur Abfahrt in Richtung einer anderen Anfrage gegeben (vorletzter *elsif*-Zweig des Controllers).
- Der Aufzug wechselt seine Fahrtrichtung nur, wenn in der jeweiligen Fahrtrichtung keine Anfragen jenseits des aktuellen Stockwerks mehr zu befriedigen sind (ebenfalls im vorletzten *elsif*-Zweig des Controllers; siehe die *Smax*-Befehle)
- Wenn in der Gegenrichtung der Aufzugfahrt noch Anfragen vorliegen, so wechselt der Aufzug früher oder später seine Fahrtrichtung, denn im untersten (obersten) Stockwerk können keine weiteren Anfragen unterhalb (oberhalb) des aktuellen Stockwerks vorliegen.

Die obigen Punkte stellen sicher, daß jede Anfrage früher oder später erfüllt wird. Eine kleine Ausnahme gibt es allerdings noch: Wenn jemand im aktuellen Stockwerk gerade dann auf den Knopf drückt, wenn der Aufzug die Tür schließt, so wird die Tür erneut geöffnet. Es ist also möglich, den Aufzug durch geschicktes Drücken in einem Stockwerk festzuhalten.

In CTL müßte beschrieben werden, daß es global auf allen Pfaden unvermeidlich ist, in einen Zustand zu gelangen, in dem ***requests(Stockwerk)*** gleich '0' ist. Wie an dem Programmtext abzulesen ist, wird dieses Bit nur dann von '1' auf '0' zurückgesetzt, wenn sich der Aufzug in dem Stockwerk befindet und darüber hinaus die Tür geöffnet war und nun wieder geschlossen wird, d.h. die Anforderung befriedigt wurde. Insgesamt muß es heißen: ***AF(requests(Stockwerk)='0')=1***. Die gegebene Bedingung muß für alle Zustände des Automaten erfüllt sein, denn entweder es liegt keine Anfrage in diesem Stockwerk vor und das Bit ist sowieso '0', oder wir müssen tatsächlich dorthin fahren. An dieser Stelle darf nicht gefordert werden, daß in ***requests*** alle Bits zu '0' werden, da natürlich ständig Knöpfe gedrückt werden können. Allerdings muß der Ausdruck im Normalfall noch auf die tatsächlich erreichbaren Zustände im Automaten beschränkt werden. Und im Sinne der obigen Ausnahme muß auch noch als Fairneßbedingung angegeben werden, daß niemand so blöd ist und regelmäßig aufs Knöpfchen drückt.

5.3. Sicherheit

Wie im VHDL-Programm nachgelesen werden kann, können die Befehle ***go_up*** bzw. ***go_down*** nur dann erteilt werden, wenn sich der Aufzug nicht bewegt. In Hinsicht auf die Bewegungsrichtung können dem Aufzug daher nie widersprüchliche Befehle erteilt werden.

In CTL könnte formuliert werden, "von allen Zuständen, in denen sich der Aufzug nach oben bewegt, ist es unvermeidlich, daß der Aufzug anhält, und bis dorthin wird der Befehl, nach oben zu fahren, beibehalten":

go_up $\Rightarrow$ AU (***go_up*** and not ***go_down***, not (***go_up*** or ***go_down***))

und natürlich symmetrisch

go_down $\Rightarrow$ AU (***go_down*** and not ***go_up***, not (***go_up*** or ***go_down***))

Auch hier sollte natürlich noch alles auf die erreichbaren Zustände eingeschränkt werden.