1.4 WüHDL
--
Ein erster Überblick

Zur Vorlesung Rechenanlagen

SS 2019

Wir haben an Beispielen schon gesehen, wie man das Verhalten von Bausteinen in WüHDL beschreiben kann. Da man jedes Stück Hardware im Grunde als Baustein mit den nach außen geführten Signalen als Schnittstelle auffassen kann, kann man die Definition des Verhaltens stets als

- eine **Architecture** auffassen:
  - Spezifikation des Verhaltens durch
    - Programmcode (Prozesse), d.h. man beschreibt das Verhalten mit dem Repertoire einer gewöhnlichen Programmiersprache (Ada ähnlich), oder
    - Strukturbeschreibung, d.h. man definiert das Verhalten durch Instanziierung und Verbindung anderer Komponenten (Schaltkreise)

Wir wollen nun die wichtigsten Konzepte, die wir in den Beispielen schon benutzt haben, zusammenfassen:

Architekturen -- die Definition der Komponenten

Definitionsmöglichkeiten

Man hat schon aus der Syntax ersichtlich mehrere Möglichkeiten, das Verhalten von Komponenten zu definieren. In WüHDL sehen wir folgende Optionen vor:

- Beschreibung durch Prozesse
- Beschreibung durch nebenläufige Signalzuweisungen (concurrent signal assignments)
- Beschreibung durch Instanziierung und Verbindung von Komponenten.

Die erste Variante kann man als Verhaltensdefinition auffassen, während die letztere eine Strukturdefinition ist (Aufbau über Instanzen vorhandener Komponenten). Die zweite Variante kann man sowohl als Verhaltens- als auch als Strukturdefinition auffassen (vgl. spätere Kapitel). Wir klammern sie vorläufig aus.

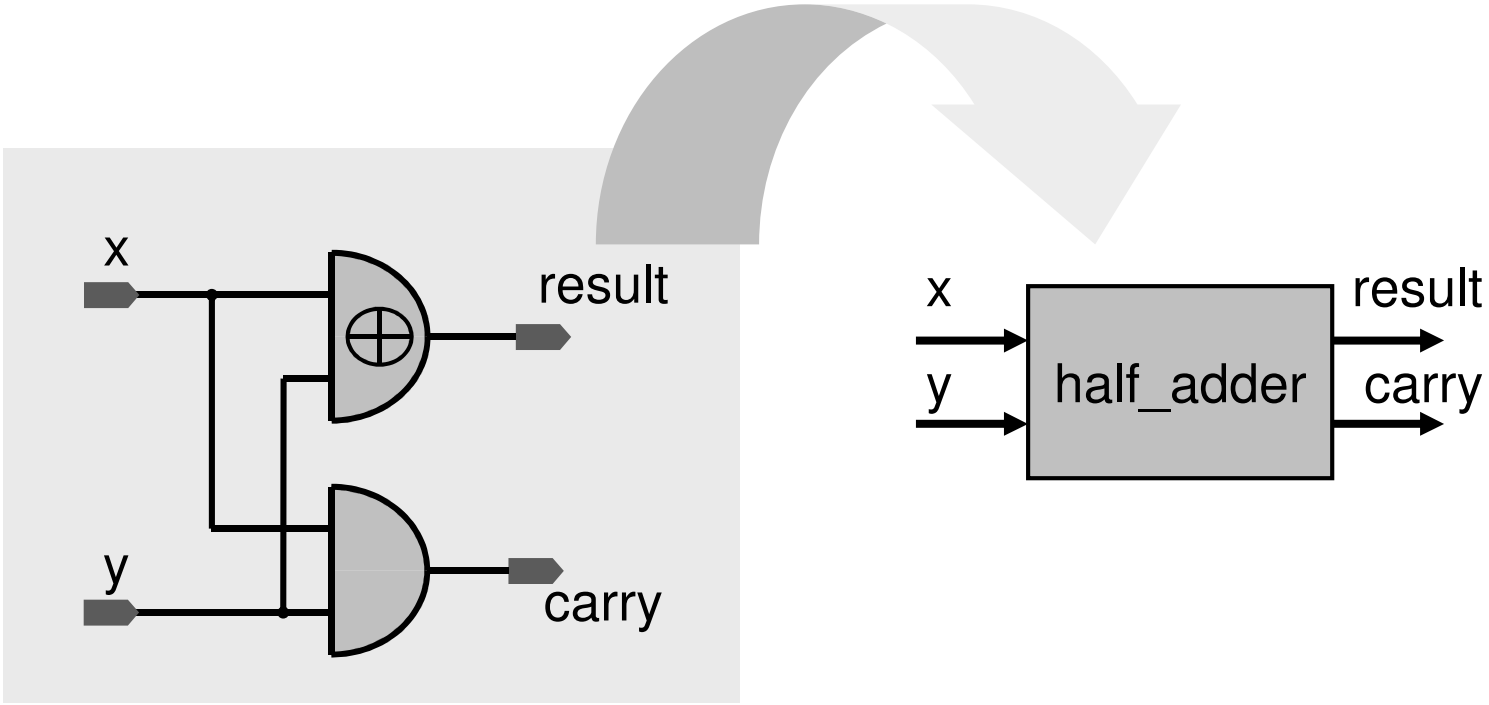
Architekturen -- Definition durch Prozesse

Das Verhalten eines Stückes Hardware kann man durch mehrere nebenläufige Prozesse beschreiben, die als sequentielle Programme bis zur Terminierung ausgeführt werden. Innerhalb eines Prozesses können neue Waveforms vom Auswertezeitpunkt in die Zukunft hin Signalen zugewiesen werden. Dadurch entstehende Änderungen an Signalen können dann wieder andere Prozesse zur Auswertung anstossen:

Triviales Beispiel

```
ARCHITECTURE behavior1 OF half_adder IS
BEGIN
  PROCESS (x, y)
            -- x,y Signale bei deren Änderung der Prozess anläuft
            -- man nennt diese Liste auch Sensitivitätsliste
   BEGIN
         result <= x XOR y;
         carry <= x AND y;
   END PROCESS
END behavior1;
```

Architekturen -- Definition durch strukturelle Beschreibung



Häufig ist es bequem und auch intuitiv, eine Komponente durch einen **Schaltplan** über anderen Komponenten zu definieren, d.h. man benutzt

- Subkomponenten und
- entsprechende Verdrahtung

Architekturen -- Beispiel zu Schaltplänen

Folgendes Programmstück beschreibt das Verhalten eines Halbaddierers durch Instanziierung von Komponenten:

```
ARCHITECTURE structure OF half_adder IS
    COMPONENT xor2
        PORT (a,b: IN BIT; c: OUT BIT);
    END COMPONENT;
    COMPONENT and2
        PORT (a,b: IN BIT; c: OUT BIT);
    END COMPONENT;
BEGIN
    G0: xor2 PORT MAP (x,y,result); -- positionelle Zuordnung
    G1: and2 PORT MAP (a => x; b => y; c => carry );
    -- besser: Zuordnungsliste formaler Port => Signal oder Port
END structure;
```

Architekturen -- Beispiel zu Schaltplänen

Anmerkungen:

Unser Beispiel ist sehr speziell, weil alle Signale des Schaltkreises zugleich auch Ports der Komponente Halfadder sind.

Bei komplexeren Schaltkreisen hat man meist viele Signale, die nur als Mittler zwischen den beteiligten Komponenten fungieren, aber selbst nach außen nicht sichtbar sind. Diese Signale sind dann als lokale Signale in der Architecture zu deklarieren.

Viele Entwicklungswerkzeuge sehen für solche Strukturbeschreibungen spezielle Unterstützung vor:

Einen **graphischen Symboleditor**, mit dem man Komponentensymbole mit Portbezeichnungen zeichnen kann und einen **graphischen Schaltplaneditor**, mit dem man Symbole von Komponenten platzieren und deren Ports verdrahten kann.

Die Benutzung solcher Tools nimmt einem die mühevolle Arbeit ab, WüHDL bzw. VDHL Code eingeben zu müssen:

Symboleditor → entity declaration

Schaltplaneditor → architecture Definition mit Instanziierungen

Konstanten, Variablen und Signale

Konstanten

Konstanten stehen für Objekte, deren Wert nur einmal festgelegt und dann nicht mehr geändert werden kann. Ihr Wert muss zur Übersetzungszeit, d.h. wenn eine WüHDL Spezifikation von einem Übersetzerprogramm übersetzt wird, feststehen. Sie beschreiben in der Regel typische Größen eines Entwurfs.

Beispiele*:*

```
CONSTANT clockperiod: TIME := 10 ns; -- 100Mhz
CONSTANT pi : REAL := 3.1415;
CONSTANT defaultcode: BIT_VECTOR(0 TO 3) :="1001";
```

Konstanten, Variablen und Signale

Variablen

- unterscheiden sich nicht von Variablen anderer höherer Programmiersprachen.
- haben "kein Gegenstück in der Hardware“ !!
- werden nur im Rahmen von Zwischenrechnungen zur Verhaltensdefinition verwendet (nur lokal in Prozessen, Funktionen, Prozeduren, Paketen, ...nicht in Architectures ...).

Beispiele

```
VARIABLE row, column : INTEGER RANGE 0 TO 31;
VARIABLE sum : NATURAL := 0;
VARIABLE  feld : Two_Dim (0 TO 15, 0 TO 31);
```

Konstanten, Variablen und Signale

Signale

- sind in anderen Programmiersprachen nicht zu finden.
- modellieren den zeitlichen Ablauf des Verhaltens der Hardware, sie bringen überhaupt den Zeitbegriff erst ins Spiel.
- sind die einzige Möglichkeit für nebenläufig arbeitende Komponenten, miteinander zu kommunizieren.
- können über jedem Typ definiert sein.
- tragen Werte, die stets eine Abbildung von TIME in den Wertebereich des Typs sind, während der Wert einer Variablen nur ein Wert aus dem Wertebereichs des Typs der Variablen ist!

Beispiele

```
SIGNAL clk, resetn      : BIT;
SIGNAL counter          : INTEGER RANGE 0 TO 31;
SIGNAL reg_ram          : Two_Dim (0 TO 15, 0 TO 31);
```

Variablen versus Signale

Verschiedene Syntax für Zuweisungen

- <variable_assignment> ::= <target > **:=** <expression> **;**
- <signal_assignment >::=
  <target> **<=** [**TRANSPORT**] <waveform_element>
  { **,** <waveform_element> }* **;**

  <waveform_element> ::=
  <value_expression> [**AFTER** <time_expression >] |
  NULL [**AFTER** <time_expression>]

Variablen weist man Werte ihres Typs zu, Signalen weist man Wellenformen (Signalwerte) über ihrem Typ zu. Dabei beziehen sich die Zeitangaben (AFTER) relativ auf den Zeitpunkt der Zuweisung. Fehlt die Zeitangabe, dann ist die Verzögerung = *0* (!).

Verzögerungsarten bei Signalen

Mit einem Signal ist als Wert stets eine zeitdiskrete **Waveform** über dem Wertebereich seines Typs assoziiert. Mit einer Zuweisung fügt man nun neue Schaltereignisse in die Waveform ein, die **ab dem aktuellen Zeitpunkt** stattfinden können. Es stellt sich folgende Frage:

? Wie verfährt man, wenn hinter dem aktuellen Zeitpunkt schon Änderungen in der Waveform stehen?

Eine Zuweisung an ein Signal bedeutet eine **Fortführung** der Waveform **aus dem Wissen** über das Systemverhalten **bis zum aktuellen Zeitpunkt**.

Es gibt nun zwei Fälle:

- Es sollen Ereignisse vor schon bestehenden Ereignissen eingfügt werden.
- Es sollen Ereignisse nach schon bestehende Ereignissen eingefügt werden.

Verzögerungsarten bei Signalen

- Einfügen **vor** schon bestehenden Ereignissen:

  Führt zu einem Löschen aller nachfolgenden Ereignisse

  Grund: Aus dem Wissen über das Verhalten des Systems bis zum aktuellen Zeitpunkt ist die bisherige Waveform bestimmt. Nun entsteht aber ein Ereignis, das dieses Wissen zurücknimmt. Man muss also die für die Zukunft getroffenen Änderungen zurücknehmen.

- Einfügen **hinter** schon bestehenden Ereignissen

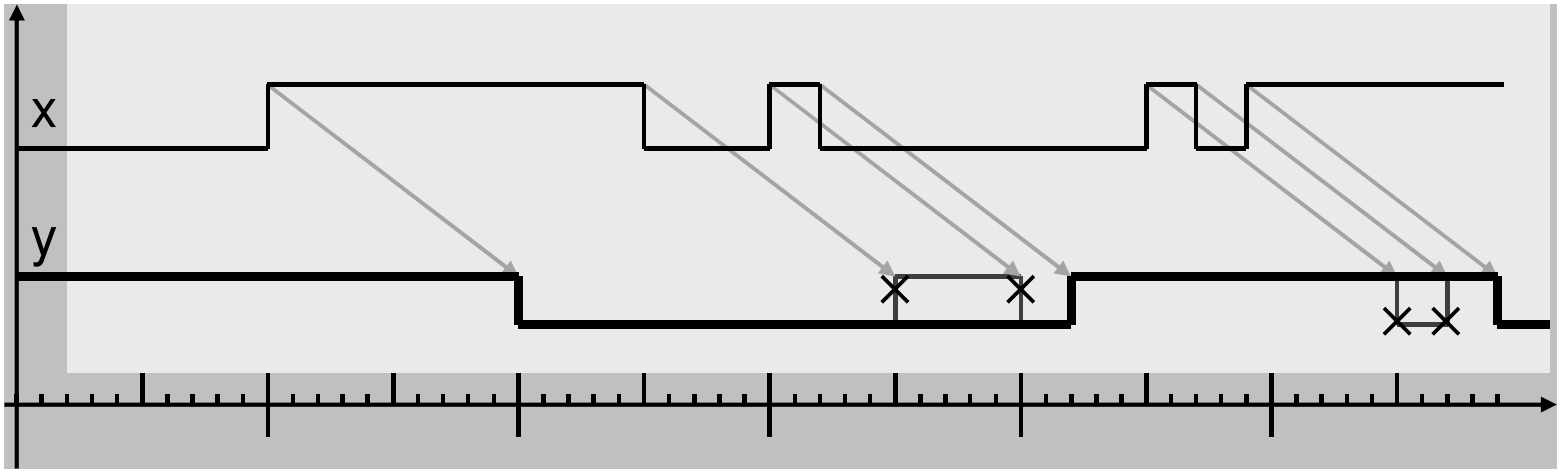
  WüHDL unterscheidet dazu zwischen 2 Arten der Verzögerung

  - **träge Verzögerung** (inertial delay)
  - **nichtträge Verzögerung** (transport delay)

Die Verzögerungsart INERTIAL

- INERTIAL ist die **Default Einstellung**
- beim Einfügen eines Ereignisses werden alle vorhergehenden Ereignisse zwischen aktuellem Zeitpunkt und dem Einfügezeitpunkt gelöscht, bis auf das erste Vorgängerereignis, falls existent, das denselben Wert erzeugt.
- dieses Modell erzeugt keine Signalpulse an Komponenten, die kürzer sind als deren Verzögerung:

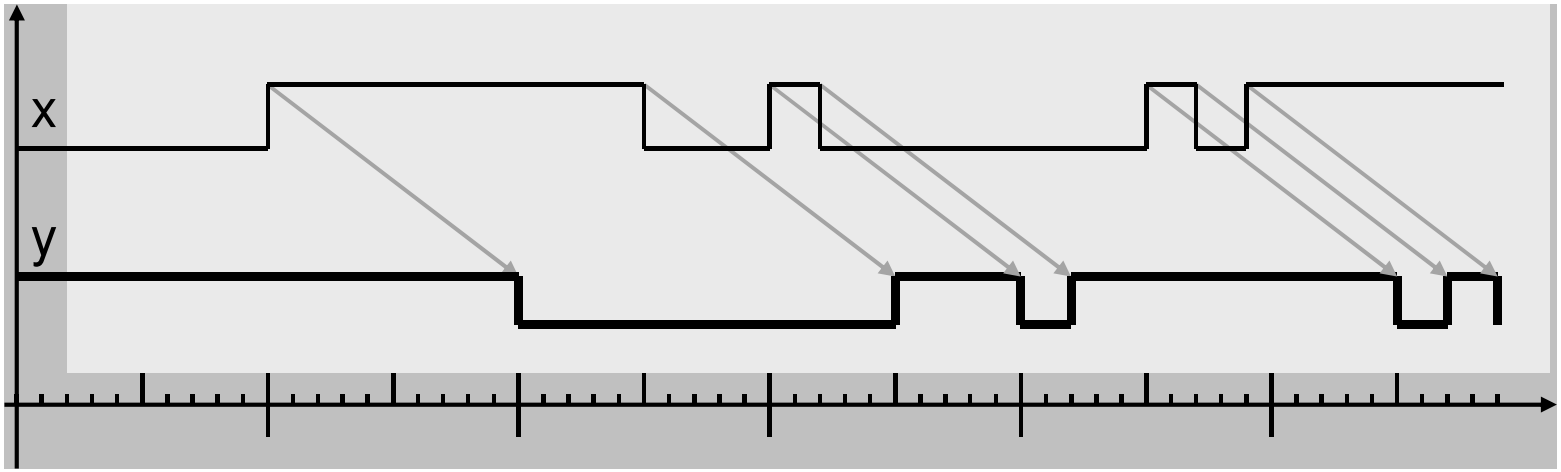
y <= not x AFTER 10 ps;



Die Verzögerungsart TRANSPORT

- Transportverzögerung muss explizit durch Voranstellen des Schüsselworts TRANSPORT erzwungen werden.
- Beim Einfügen eines Ereignisses bleiben alle vorhergehenden Ereignisse zwischen aktuellem Zeitpunkt und dem Einfügezeitpunkt erhalten.
- Dieses Modell kann sehr kurze Signalpulse an Komponenten erzeugen, die physikalisch ggf. gar nicht mehr entstehen:

y <= TRANSPORT not x AFTER 10 ps;



Fazit: Variablen versus Signale

- Deklaration
  - Variablen werden in Prozessen / Unterprogrammen deklariert und sind auch nur da sichtbar.
  - Signale können nicht in Prozessen / Unterprogrammen deklariert werden.
- Verwendung
  - Variablen werden zum Abspeichern temporärer Werte benutzt.
  - Signale stehen üblicherweise für Verbindungen in der Hardware.
- Wertzuweisung
  - bei Variablen erfolgt sofort, wenn die Variablenzuweisung ausgeführt wird.
  - bei Signalen erfolgt erst, nachdem der zuweisende Prozess terminiert ist.

Ausführung eines WüHDL Modells

Eine WüHDL Spezifikation besteht bis jetzt also aus

- einer Menge von Entities
- einer Architektur zu jeder Entity
- einer Menge von Instanzen von Bausteinen, zu denen selbst wieder
  - ⊙ Prozesse und
  - ⊙ Instanzen von Bausteinen

  definiert sind.

Ist die Architektur einer Bausteininstanz *A* wieder durch Instanzen $\{b_1,...,b_n\}$ von Bausteinen definiert, so kann man dies auch auffassen als eine Instanziierung von Bausteinen $A.b_1, ... , A.b_n$. Man spricht in diesem Zusammenhang auch von einer **Entwurfshierarchie**.

Entwurfshierarchie bei WüHDL Modellen

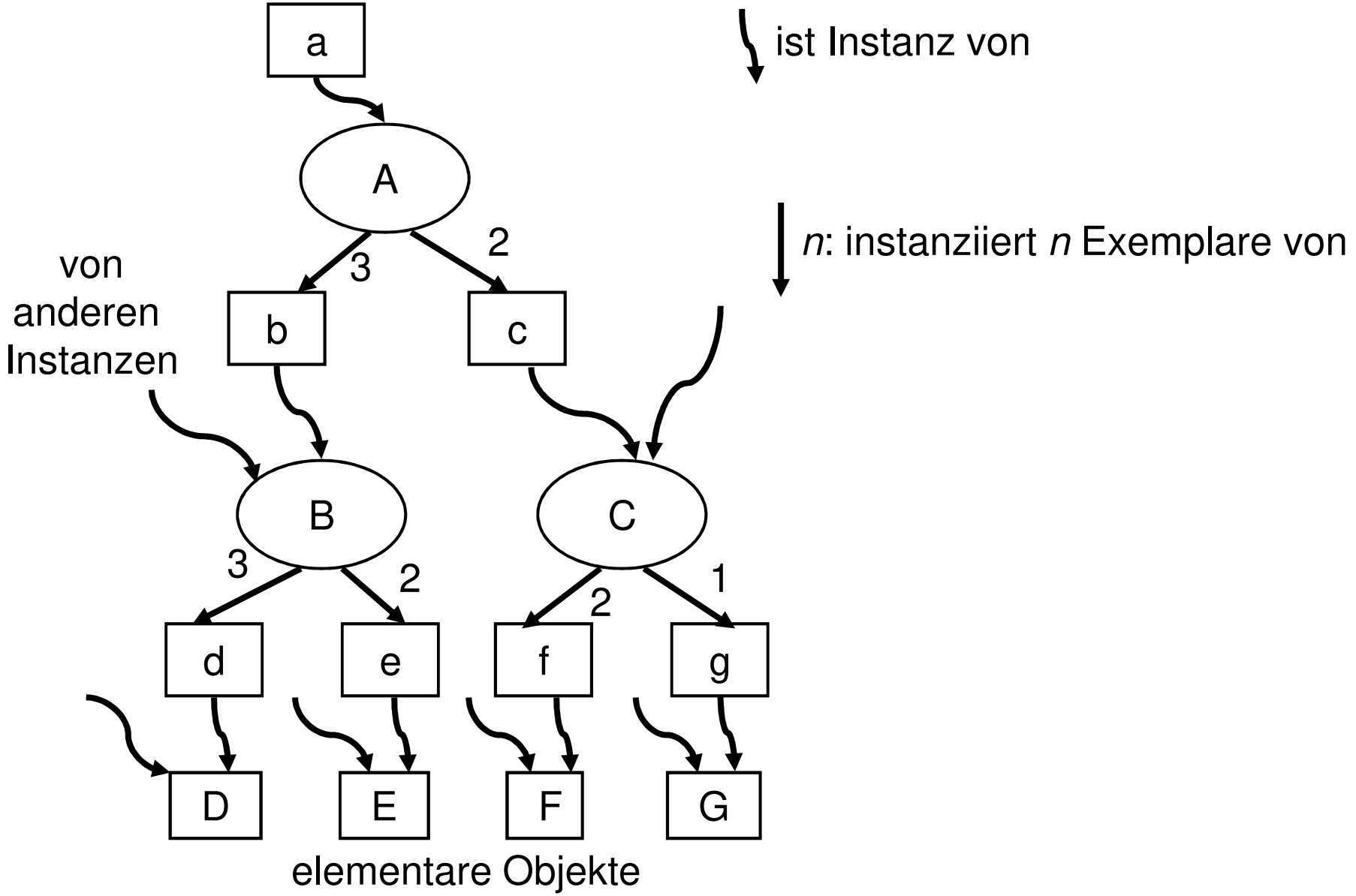
Hierarchie ist ein fundamentales Konzept zur Konstruktion komplexer Systeme, weil man eine riesige Anzahl von Instanzen auf sehr kompakte Art und Weise definieren kann. Man beschreibt Komponenten, die andere Komponenten instanziieren nur einmal, bei jeder Instanziierung der Komponente werden aber auch alle Subkomponenten instanziiert.

Man erhält auf diese Weise eine sehr kompakte Spezifikation eines riesigen Objekts bestehend aus tausenden Instanzen elementarer Objekte.

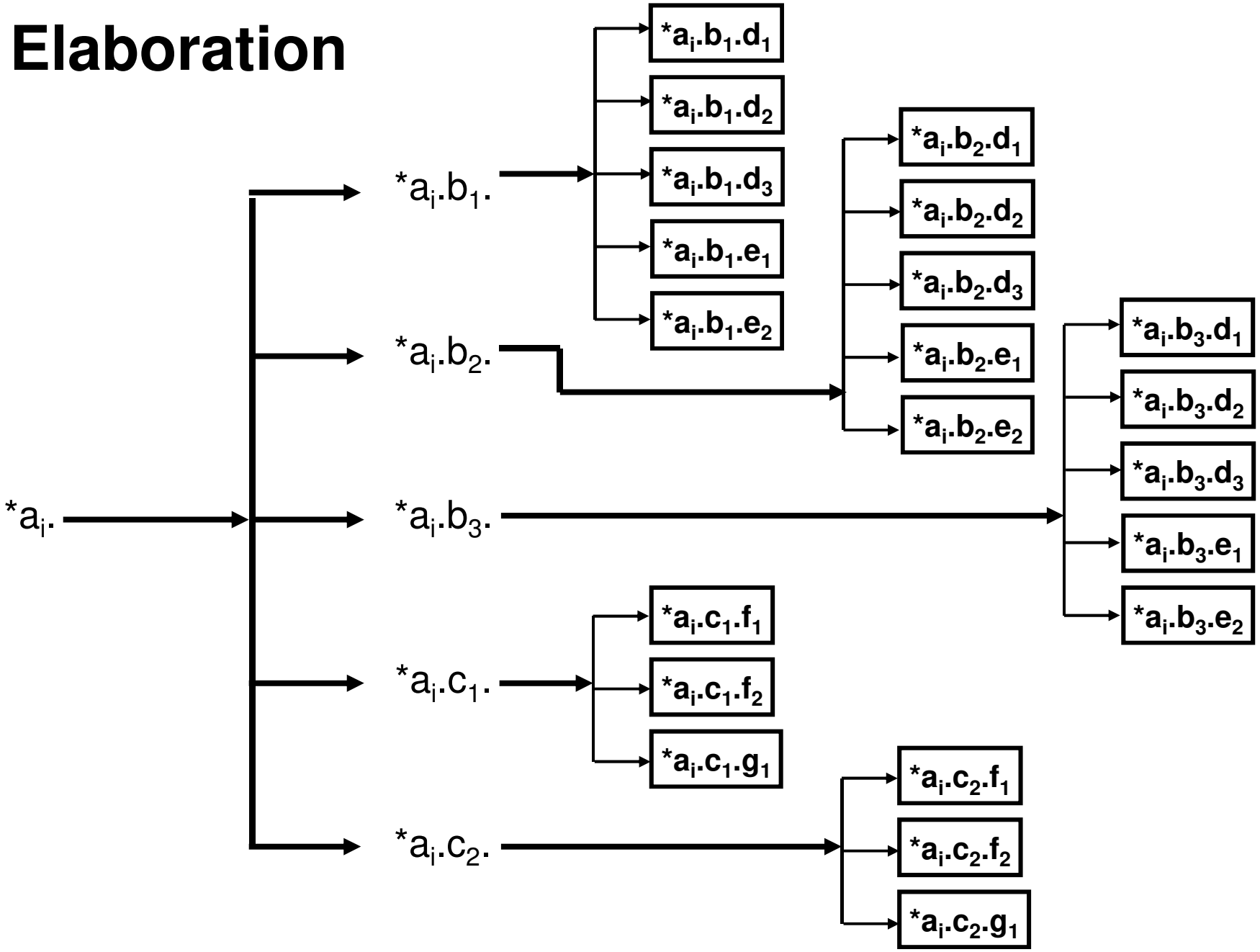
Elementar in WüHDL sind Prozesse und Signale, da sie keine Subobjekte instanziieren.

Folgendes Bild soll die Kraft der Hierarchie verdeutlichen:

Entwurfshierarchie -- ff



Elaboration



Ausführung eines WüHDL Modells -- ff

Nach der Elaboration der durch die WüHDL Modelle definierten Hierarchie hat man es also im Grunde nur mit einer Vielzahl von Instanzen von Prozessen und Signalen zu tun. Dabei muss folgendes gelten:

- Jede Signalinstanz eines unaufgelösten Signaltyps wird von höchstens einer Prozessinstanz zugewiesen. Da dies zwingend erfordert, dass in unserer Beschreibung ein Signal höchstens in einem Prozess eine Zuweisung erhalten darf, nennen wir diesen Prozess auch **den Treiber** des Signals.

  Nach Elaboration bedeutet dies, dass jede Instanz eines Treiberprozesses eine entsprechende Instanz eines Signals treibt.
- Zu jeder Signalinstanz eines aufgelösten Typs wird zu jeder Prozessinstanz, die dem Signal etwas zuweist, ein Treiber angelegt.

  Aufgelöste Signalinstanzen haben also mehrere Prozessinstanzen als Treiber.

Ausführung eines WüHDL Modells -- ff

Die Prozessinstanzen stehen nun für sequentielle Programme, wobei zu jeder Prozessinstanz auch entsprechend lokale Variablen instanziiert werden.

Ferner hängt nach dem von uns bisher eingeführten Konzept jeder Prozess von einer Menge von Signalen ab, auf deren Änderung er reagieren muss.

Wir können also jeder Prozessinstanz eine Menge von Signalinstanzen zuordnen, gegenüber denen er sensitiv ist, die **Sensitivitätsliste**.

Die Auswertung startet nun stets zum Zeitpunkt 0.

Alle Signale und Variablen vom Typ T werden mit T'LOW, d.h. dem niedrigsten Wert des Typs initialisiert, bzw. T'LOW von 0 bis unendlich (Signale).

Die aktuelle Zeit (NOW) ist 0.

DELTA Zyklen

Nun ist das Spiel eigentlich stets dasselbe:

- Starte alle Prozessinstanzen, die sensitiv auf eine Signalinstanz sind, die ein Schaltereignis hat, und lasse sie bis zur Terminierung laufen. Zu Beginn sind dies alle(!) Prozessinstanzen, da sich alle Signalinstanzen initialisieren.
- Sind die Prozessinstanzen terminiert, dann haben sie den Zustand ihrer lokalen Variablen verändert, und Zuweisungen an Signaltreiberinstanzen gemacht. Nun werden die Signaltreiber aller Signalinstanzen ausgewertet und für Signalinstanzen, die eine Zuweisung im Signaltreiber erhalten, ein nächstes Ereignis berechnet (dies kann auch Verzögerung 0 haben).

  Bei aufgelösten Signalen wird der Auflösungsfunktion ein Vektor übergeben, dessen Komponenten aus allen Zuweisungswerten von Treibern bestehen, die in diesem Zyklus eine Zuweisung machen. Als Wert wird der zurückgelieferte, aufgelöste Wert genommen.

DELTA Zyklen -- ff

- Setze nun die aktuelle Zeit auf das Minimum aller Startzeiten von nächsten Ereignissen auf Signalinstanzen.
- Beginne wieder mit einem Delta Zyklus

Probleme machen Änderungen, die mit Verzögerung 0 geschehen. Diese können dazu führen, dass Prozessinstanzen in einem Zeitpunkt sogar mehrfach (sogar unendlich oft) anlaufen und wieder terminieren. Hier kann sich also die Simulation aufhängen.

Vorsicht bei der Modellierung.

Problematisch sind auch Prozesse mit leerer Sensitivitätsliste. Diese laufen automatisch immer wieder an, wenn sie nicht explizit suspendiert werden.