1 Hardwareentwurf

1.1 Grundlagen

POSITIVE	natürliche Zahlen	$\mathbb{N}$
NATURAL		$\mathbb{N}_0$
INTEGER	ganze Zahlen	$\mathbb{Z}$
REAL	reelle Zahlen	$\mathbb{R}$
BOOLEAN	(true, false), (low, high)	
BIT	('0','1')	
CHARACTER	(...,'A','B',...,'a','b',...,'0',...)	
TIME	hr,min,sec,ms,us,ns,ps,fs	
SEVERITY_LEVEL	note, warning, error, failure	

Defintion der Bedeutung

<typedeclaration> ::= TYPE <identifier> IS {<scalartypedef>|<composite(td)>| ...};
z.B.
TYPE STRING IS ARRAY (positive range <>) OF character;
TYPE BITVECTOR IS ARRAY (natural range <>) OF bit;
TYPE Xvector32 IS ARRAY (0 TO 31) OF X;

Konkatenation von Worten

u & v	Konkatenation
u(i TO j)	Teilwort

1.2 Grundbausteine der Digitaltechnik

1.2.3 WüHDL-Beschreibung eines Bausteins

Definition von Bausteinen

entity	Definition der Bausteinschnittstelle (Ports)
component	Definition eines Bausteins zur Benutzung
Instanzierungen	Erzeugung von Instanzen von Komponenten und deren Verdrahtung durch Anweisungen

Beschreibung eines Bausteins

Entity	Spezifikation einer Schnittstelle
Architecture	Spezifikation des Verhaltens (Prozesse, Strukturbeschreibung: durch Verhalten anderer Komponenten)

Entity Declaration

ENTITY <entity_identifier> IS
    [<generic_delcaration>][<port_declaration>]
END[ENTITY][<entity_identifier>];

Port Declaration

PORT ({<port_list> [<mode>] <type_id>;}*);

Beispiel

```
ENTITY inverter IS
    PORT(x: IN bit;y: OUT bit);
END inverter;
```

Port-Arten

- IN: Eingangsport (wird nicht verändert)
- OUT: Ausgangsport (wird verändert, aber nicht von außen)
- INOUT: wird von Komponente oder von außen verändert, kann Einfluss auf Verhalten der Komponente haben

Architecture Declaration

ARCHITECTURE <arch_identifier> OF <entity_identifier> IS
    [<local_delcarations>]
    BEGIN [<statements>]
END[ARCHITECTURE][<architecture_identifier>];

Local Declarations

{ USE < ... >| TYPE < ... >| CONSTANT < ... >| SIGNAL < ... >| VARIABLE < ... >|
COMPONENT < ... >| FUNCTION < ... >| PROCEDURE < ... >}*

Architekturen

- Beschreibung durch Prozesse
- Beschreibung durch nebenläufige Signalzuweisungen
- Beschreibung durch Instanzierung und Verbindung von Komponenten

Prozesse

sichtbar:

- Ports vom Modus IN,INOUT
- lokal in der Architecture definierte Signale
- Ports vom Modus OUT, INOUT und lokal definierte Signale

Sensitivitätsliste:
Prozesse ist sensitiv auf Ports der entity vom Modus IN,INOUT,lokale Signale

Beispiel zu Prozessen

```
ARCHITECTURE behavior1 OF inverter IS
    CONSTANT tdhl: TIME := 25ps;
    CONSTANT tdlh: TIME := 35ps;
BEGIN
PROCESS(x)
    BEGIN
        IF x = '0'
            THEN y <= not x AFTER tdlh;
            ELSE y <= not x AFTER tdhl;
        END IF;
    END PROCESS;
END behavior1;
```

END terminiert nicht den Prozess (Ende der Deklaration)
Angabe einer Sensitivitätsliste ist äquivalent zu einer WAIT-Anweisung

NAND

```
ENTITY nand2 IS
        PORT(x,y: IN bit;z: OUT bit);
END nand2;

ARCHITECTURE behavior1 OF nand2 IS
        CONSTANT tdhl: TIME := 35ps;
        CONSTANT tdlh: TIME := 50ps;
BEGIN
PROCESS(x,y) – sensitiv auf IN,INOUT ports
        BEGIN
                IF x = '0' OR y = '0'
                        THEN z <= 1 AFTER tdlh;
                        ELSE z <= 0 AFTER tdhl;
                END IF;
        END PROCESS;
END behavior1;
```

Beispiel zu Instanzierung

```
ENTITY dlatch IS
        PORT(clk,d: IN bit;q,qz: OUT bit);
END dlatch;

ARCHITECTURE structure OF dlatch IS
        COMPONENT inverter PORT(x:IN bit; y: OUT bit);
        END COMPONENT
        ...
BEGIN
        INV: inverter PORT MAP (x => clk, y => clk_z);
        ...
END structure;
```

Problem: Keine Überwachung der Setup oder Hold-Zeiten

Überwachung von Bedingungen

```
ARCHITECTURE structure OF dlatch IS
        ...CONSTANT setup: Time := 40ps; CONSTANT hold: TIME := 20ps;
BEGIN
        Setup_hold_check: PROCESS(clk,d)
        BEGIN IF clk'event AND clk = '0'
                THEN
                        ASSERT clk'last_event > hold
                        REPORT '"Hold-Violation'" SEVERITY error;
                ELSE IF ...
                END IF;
        END Setup_hold_check;
END structure;
```

Testbenches und formale Verifikation

Prozess in welchem nach und nach (zeitlich) versch. Wert an ein Bausteinsystem angelegt werden und die Ausgänge überprüft werden

Attribute von Signalen

- sig'stable(T) TRUE, wenn sich sig in den letzten T Zeiteinheiten geändert hat
- sig'event TRUE, wenn zum akt. Zeitpunkt eine Änderung erfolgt
- sig'last_event TIME seit letzter Änderung
- sig'last_value gleicher Typ wie sig mit Wert vor letzter Änderung

Resolved Types

Package IEEE.std_logic_1164

U	uninitialized	Z	High Impedance (Hochohmig)
X	Forcing unknown	W	Weak unknown
0	Forcing 0	L	Weak 0
1	Forcing 1	H	Weak 1

Konstanten

einmal festgelegt: CONSTANT pi: REAL := 3.1415;

Variablen (nur lokal, nicht in Architecture)

haben kein Gegenstück in der Hardware
VARIABLE sum: NATURAL := 0;

SIGNALE ($\Rightarrow$ parallele Verarbeitung möglich)

modellieren den zeitlichen Ablauf des Verhaltens der Hardware
zeitdiskrete Wellenform, Waveform über dem Wertebereich des Typs

- einzige Möglichkeit für nebenläufig arbeitende Komponenten zur Kommunikation
- können über jedem Typ definiert sein
- Abbildung von TIME in den Wertebereich des Typs

Zuweisungen

- Variablen: <target> := <expression>
- Signal: <target> <= <waveform_element> {<wav_elem>}*;
- waveform_element: <value expression>|NULL [AFTER <time_expr>];

Entwurfshierarchie bei VHDL Modellen

Hierarchie ist ein fumdamentales Konzept zur Konstruktion komplexer Systeme
$\Rightarrow$ kompakte Spezifikation eines riesigen Objekts

- Jede Signalinstanz eines unaufgelösten Signaltyps wird von höchstens einer Prozessinstanz zugewiesen
- Jedes Signal darf in einem Prozess höchstens eine Zuweisung erhalten
  dieser Prozess ist der Treiber des Signals
- Zu jeder Signalinstanz eines aufgelösten Typs werden zu jeder Prozessinstanz, die dem Signal etwas zuweist, ein Treiber angelegt
  $\Rightarrow$ Der Auflösungsfunktion wird Vektor übergeben, der aus Zuweisungswerten von Treibern besteht

1.4 Schaltfunktion und boolsche Ausdrücke

Boolesche Ausdrücke in WüHDL

AND,OR,NAND,NOR,XOR,XNOR mit gleicher Präzedenz (v.l.n.r), lediglich NOT hat hörere Präzedenz

Concurrent Signal Assignments

im Rumpf der Architecture

- zugewiesene Signale: OUT-Ports, lokale Signale
- zu jedem Signal: höchstens eine Zuweisung
- rechte Seite: nur lokale Signale oder IN-Ports
- lokale Signale: keine zyklische Abhängigkeit

Beispiel

BEGIN
$s <= (a\ AND\ b) OR\ NOT c$;
$t <= (a\ XOR\ c) NAND b$;
PROCESS...
END;

Jede einzelne Signalzuweisung ist Kurznotation für Prozess!
Keine Angabe von Verzögerungen

2 Hardwareentwurf

2.1 Schaltkreise

Hierarchischer Entwurf

- Schaltkreise durch Instanzierung von Komponenten und das Binden ihrer Ports an Signale oder Ein/Ausgangsprots
- kein festes Bausteinsystem, sondern in der COMPONENT Deklaration festgelegt
- Komponenten können dabei selbst wieder durch Schaltkreise modelliert sein
- $\Rightarrow$ kurze und übersichtliche Definition von großen Schaltkreisen

Prozesse und concurrent signal assignments können im Grunde auch als Bausteininstanzen aufgefasst werden

Generische Definition

VHDL sieht dafür den Parameter GENERIC in der Definition der ENTITIES vor

hier hängt n von der Breite des Eingabevektors ab

Hazardanalyse

in VHDL: keine Hazardanalysen möglich $\Rightarrow$ konkrete Verzögerung

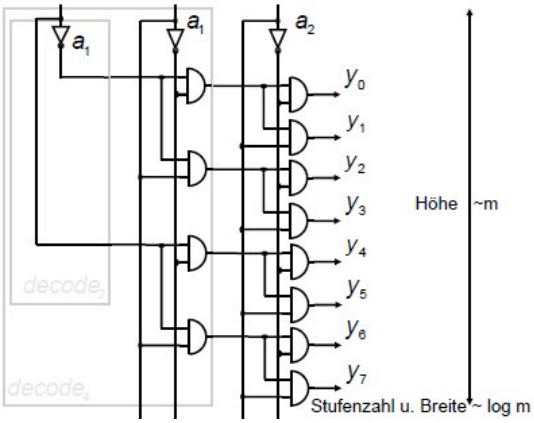
Beispiele:

- PGPC (Parity Generator Parity Checker)

```
ENTITY pgpc IS
        GENERIC (n: POSITIVE)
        PORT(x: IN BIT_VECOTR(0 TO n-1);y: OUT BIT);
END ENTITY pgpc;
ARCHITECTURE structure OF pgpc IS
        COMPONENT pgpc –Benutzt sich selbst
                GENERIC (n: POSITIVE);
                PORT(x: IN BIT_VECTOR(0 TO n-1);y: OUT BIT);
        END COMPONENT
        COMPONENT xor2
                PORT(x,y: IN BIT, z: OUT BIT);
        END COMPONENT;
        SINGAL msp, lsp: BIT;
        BEGIN
                verankerung2: IF n=2 GENERATE
                        X0: xor2 PORT MAP (x=>x(0),y=>x(1),z=>y);
                        END GENERATE
                verankerung3: IF n=3 GENERATE
                        X1: xor2 PORT MAP (x=>x(0),y=>x(1),z=>msp);
                        X2: xor2 PORT MAP (x=>x(2),y=>msp,z=>y);
                        END GENERATE
                rekursion: IF n>3 GENERATE
                    lsb: pgpc
                        GENERIC MAP (n/2)
                        PORT MAP (x=>x(n-n/2 TO n), y=>lsp);
                    msb: pgpc
                        GENERIC MAP (n-n/2)
                        PORT MAP (x=>x(0 TO n-n/2-1), y=>msp);
                result: xor2 PORT MAP (x=>msp, y=>lsp, z=>y);
                END GENERATE;
        END ARCHITECTURE;
```

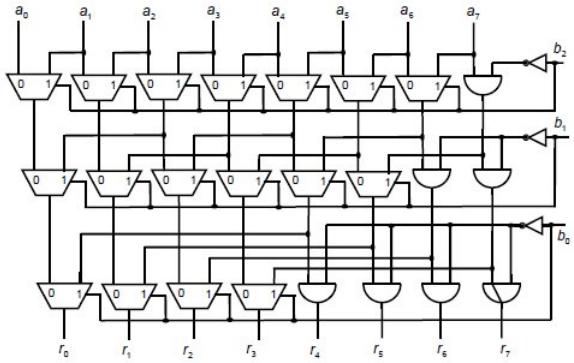
- Zahlen
  $u_n : B^n \mapsto \mathbb{N}_0$ mit $u_n(a_0, ..., a_{n-1}) := 2^{n-1} \sum_{i=0}^{n-1} a_i 2^{-i}$
  ist die Darstellung als (unsigned) ganze Zahl

- Decoder
  bildet Binärcode ab auf einem 1 aus m Code (one hot Code)
  decode$_m$: $B^n \mapsto B^m$ mit
  $decode_m(a_0, ..., a_{n-1}) = (y_0, ..., y_{m-1}) \Leftrightarrow \forall i : y_i = (u_n(a) = i)$
  Decoder rekursiv aus kleinere Dekodern zusammensetzen

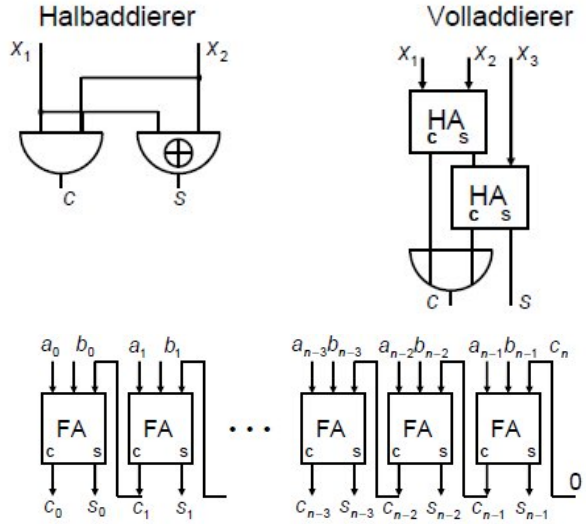


- Linksshifter
  Operand a mit Wortbreite n hat Shiftweite i als unsigned Zahl
  Für jede zweierpotenz einen Shifter SHLi bereitschtellen, der schiebt, wenn diese mit 1 besetzt ist und dann diese Bausteine hintereinander schalten
  Grundstruktur: $shl_i(a, s) = SHL(a, s \cdot 2^i)$

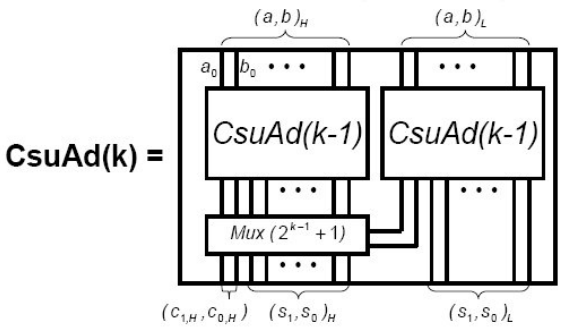
Beispiel: 8-Bit Linksshifter



- Ripple Carry Addierer
  ADDU: Addition vorzeichenloser Zahlen
  $addu : B^{2n} \mapsto B^{n-1}$ mit $addu(a,b) = (ov,s) \Leftrightarrow 2^n ov + u(s) = u(a) + u(b)$
  Halbaddierer (half adder: Summe zweier Bits als zweistellige Zahl
  ripple carry Addierer: von weniger signifikanten Stelle zur signifikanten hin mit Volladdierer und jeweils Übertrag weiterleiten



- Conditional Sum Adder (seriell addieren)
  Idee: Berechne parallel die Summe und die Summe plus eins für die signifikantere Hälfte und die weniger signifikante Hälfte
  Rekursives Bildungsgesetz:



Und für ein Bitpaar haben wir