VHDL-Kurzreferenz

Thomas B. Preußer

5. Mai 2009

1 Allgemeines

Diese Kurzreferenz gibt einen schnellen Einstieg in die Beschreibung grundlegender Designbestandteile in VHDL. Sie liefert keine formale oder gar erschöpfende Beschreibung der VHDL-Syntax, sondern stellt gebräuchliche Grundmuster anhand von kurzen Beispielen in VHDL-Quellcode dar. In diesen Beispielen wird durch den Schriftsatz unterschieden zwischen:

- **der wesentlichen Grundstruktur**,
- üblichen Bestandteilen und
- *beispielspezifischen Codeelementen*.

Zur Dimensionierung binärkodierter Signale ist häufig der Zweierlogarithmus, genauer $\lceil \mathrm{ld}\, 2 \rceil$, über die Größe des Zustandsraumes zu bestimmen, wozu folgende VHDL-Funktion dienen soll:

```
function log2ceil(arg : positive) return natural is
  variable tmp : positive;
  variable log : natural;
begin
  if arg = 1 then  return  0; end if;

  tmp := 1;
  log := 0;

  while arg > tmp loop
    tmp := tmp * 2;
    log := log + 1;
  end loop;
  return log;

end;
```

2 Register

Register werden mit Hilfe getakteter Prozesse beschrieben. Das Schema wird hier anhand eines Abwärtszählers demonstriert, der nach dem *N*-ten Dekrement ein Fertig-Signal (in Form des Vorzeichenbits) generiert.

```
constant N : positive := ...;

signal Cnt      : unsigned(log2ceil(N) downto 0); – Zählerregister
signal CntInit  : std_logic;                      – Steuerung: Initialisierung
signal CntDec   : std_logic;                      – Steuerung: Dekrementierung
signal CntDone  : std_logic;                      – Fertig

...

process(clk)
begin
    if clk'event and clk = '1' then
        if rst = '1' then
            Cnt <= (others => '0');
        else
            if CntInit = '1' then
                Cnt <= to_unsigned(N-1, Cnt'length);
            elsif CntDec = '1' then
                Cnt <= Cnt - 1;
            end if;
        end if;
    end if;
end process;
CntDone <= Cnt(Cnt'left);
```

3 Multiplexer

Für alle Multiplexerbeispiele gelten folgende Annahmen:

```
type tData is ...;              – Typ der geschalteten Signale
type tCtrl is ...;              – Typ des Steuereingangs
signal x0, x1, x2, x3 : tData;  – Eingangswerte
signal y              : tData;  – Multiplexer-Ausgang
signal s              : tCtrl;  – Steuereingang
```

2:1-MUX:

```
y <= x1 when s = '1' else x0;
```

Zur Steuerung kann auch direkt ein beliebiger anderer BOOLEsch evaluierender Ausdruck verwendet werden, häufig z.B. Vergleicher.

n:1-MUX:

In Verallgemeinerung der obigen Syntax:

```
y <= x0 when s = "00" else
     x1 when s = "01" else
     x2;
```

Breite Multiplexer sollten zur Verringerung der kombinatorischen Tiefe bevorzugt als parallele **select**-Anweisung oder in einem Prozess als **case**-Statement realisiert werden. Unter Umständen kann sogar die dazu notwendige Umkodierung komplexerer Bedingungen in ein binär kodiertes Steuersignal noch Vorteile bringen.

```
with s select
   y <=  x0 when "00",
         x1 when "01",
         x2 when "10",
         x3 when others;
```

Basiert der Typ von *s* auf der 9-wertigen Logik (`std_logic`, `std_logic_vector`, `unsigned`, ...), so ist die abschließende Bedingung notwendig als **others** zu wählen, da eine erschöpfende (auch die Metawerte umfassende), aber auch synthetisierbare Fallunterscheidung ansonsten unmöglich ist.

Sind die Eingangssignale Elemente eines einzelnen Feldes, so lassen sich auch gut Multiplexer mit großem oder generisch adaptierbarem Fanin als einfache Feldindizierung beschreiben:

```
constant N : positive := ...;   – Fanin

type tDatas   is array(natural range<>) of tData;      – Feld von Eingangssignalen
subtype tCtrl is unsigned(log2ceil(N)-1 downto 0);     – Typ des Steuersignals

signal x      : tDatas(0 to N-1);                      – Eingangssignale

...

y <= x(to_integer(s));
```

Entspricht die Größe des Feldes keiner Zweierpotenz, so übersteigt der von *s* abgedeckte Indexbereich den tatsächlichen. Diese Multiplexer-Beschreibung wird dann zwar immer noch korrekt synthetisiert, veranlasst das Synthesewerkzeug aber zur Ausgabe einer Warnung. Diese kann durch folgende Struktur vermieden werden:

```
subtype tData is std_logic_vector(...);
...

process(s, x)
   variable tmp : tDatas(0 to 2**log2ceil(N)-1);
begin
   tmp := (others => (others => '-')); – Default: Don't Care
   tmp(x'range) := x; – Fixiere tatsächliches Eingangspräfix
   y <= tmp(to_integer(s));
end process;
```

4 Breite Gatter

signal x : std_logic_vector(...);– Eingänge
signal y_OP : std_logic; – <OP>-Verknüpfung

Die iterative Berechnung über **generate**-Statements oder in Prozessen ist allgemein möglich und erlaubt auch die Verwendung von berechneten Einzelausdrücken anstatt expliziter Eingangssignale. In einem Prozess genügt die iterative Aktualisierung einer einzigen temporären Variablen. In **generate**-Statements muss explizit ein Feld von Zwischenergebnissen angelegt werden:

```
process(x)
    variable t : std_logic;
begin
    t := '0'; – Neutraler Initialwert
    for i in x'range loop
        t := t xor x(i);
    end loop;
    y_XOR <= t;
end process;
```

```
signal t : std_logic_vector(x'low to x'high+1);
...
t(t'high) <= '0'; – Neutraler Startwert
gen: for i in x'range generate
    t(i) <= t(i+1) xor x(i);
end generate;
y_XOR <= t(t'low);
```

Für die Operationen `UND` und `ODER` ist ferner die Nutzung der Multiplexer-Syntax kurz und elegant:

```
y_OR  <= '1' when x /= (x'range => '0') else '0';
y_AND <= '1' when x  = (x'range => '1') else '0';
```

5 1-aus-n-Kodierung

signal hot : std_logic_vector(0 to N-1); *– 1-aus-N-Code*
signal bin : std_logic_vector(log2ceil(N)-1 downto 0); *– Binärcode*

Binär -> 1-aus-n-Code:

```
process(bin)
begin
    hot <= (others => '0');
    hot(to_integer(bin)) <= '1';
end process;
```

1-aus-n-Code -> Binär:

```
process(hot)
    variable t : std_logic_vector(bin'range);
begin
    t := (others => '0');
    for i in hot'range loop
        if hot(i) = '1' then
            t := t or std_logic_vector(to_unsigned(i, bin'length));
                – ODER vermeidet Prioritätenlogik und erlaubt balancierten Baum
        end if;
    end loop;
    bin <= t;
end process;
```