Rechnerorganisation Praktikum

signed/unsigned-Darstellung, Loops

Philipp Habermann, Uffke Drechsler, Jonas Tröger

Architektur Eingebetteter Systeme
Institut für Technische Informatik und Mikroelektronik
Technische Universität Berlin

WS 2015/16

Integer

- Integer sind Ganzzahlen
- Wir haben sie bisher zur
  - Adressierung von Array-Indizes
  - Einschränkung von Array-Ranges

  benutzt
- Sie können aber auch als Signale oder Variablen genutzt werden
- Dann häufig als Zähler (wie im letzten Aufgabenblatt)
- Außerdem sind sie häufig genutzte `generic`-Parameter
- Und können zum Rechnen auf Vektoren benutzt werden
- Beides wird in den folgenden Folien angesprochen werden

Motivation: Arithmetik

- Wir kennen die Bibliothek `IEEE.NUMERIC_STD` bereits
- Damit wandeln wir `std_logic_vector` in `integer` um:

```
architecture foo of bar is
    ...
    signal int : integer;
    signal slv : std_logic_vector(3 downto 0);
    ...
begin
    ...
    int <= to_integer(unsigned(slv));
    ...
end architecture;
```

- So könnten wir prinzipiell auch mit `std_logic_vector` rechnen
- Müssten dafür aber jeden Vektor erst in einen Integer umwandeln
- Und das ist umständlich und wird zudem schnell unübersichtlich!

Motivation: Arithmetik

- Deshalb bringt `IEEE.NUMERIC_STD` zwei neue Datentypen mit
  - `signed`
    - Array von `std_logic`-Werten
    - Wird vorzeichenbehaftet (2-Komplement) interpretiert
    - Kann also negativ *oder* positiv sein
  - `unsigned`
    - Array von `std_logic`-Werten
    - Wird *nicht* vorzeichenbehaftet interpretiert
    - Kann also stets *nur* positiv sein
- Auf diesen sind alle gängigen Rechenoperationen definiert
- Sowie sämtliche Vergleichsoperationen: `>`, `>=`, `=`, `/=`, `<=`, `<`

Rechenoperationen

- `IEEE.NUMERIC_STD` definiert die arithmetischen Operationen
  - `+` (Addition)
  - `-` (Subtraktion)
  - `*` (Multiplikation)
  - `/` (Division)
  - `mod` (Modulo für pos. Zahlen)
  - `rem` (Modulo für neg. Zahlen)
- Und zwar sowohl für jeweils zwei `signed`- bzw. `unsigned`-Werte
- sowie für einen `unsigned`-Werte und einen positiven `integer`-Wert
- als auch für einen `signed`-Wert und einen `integer`-Wert

Rechenbeispiele


```
architecture behavirol of bar is
    ...
    signal sig_1 : unsigned(2 downto 0) :=  "010";
    signal sig_2 : unsigned(3 downto 0) := "1000";
    signal res_1 : unsigned(2 downto 0);
    signal res_2 : unsigned(3 downto 0);
    ...
begin
    ...
    -- "010"(2) - "010"(2) = "000"(0)
    res_1 <= sig_1 - sig_1;

    -- "010"(2) + "1000"(8) = "1010"(10)
    res_2 <= sig_1 + sig_2;

    -- "1000"(8) / 4 = "0010"(2)
    res_2 <= sig_2 / 4

    -- "1000"(8) mod "010"(2) = "0000"(0)
    res_2 <= sig_2 mod sig_1
    ...
end architecture;
```

Loops

- Es gibt noch eine dritte Kontrollstruktur: den `loop`
- Ein `loop` ist sequentiell, es gibt *kein* nebenläufiges Pendant

```
foo: process (...)
    while (cond) loop
    -- sequentielle Anweisungen
    end loop;

    for i in <range> loop
    -- sequentielle Anweisungen
    end loop;
end process;
```

- Die Deklaration der Zählvariable im `for-loop` erfolgt implizit
- Sie kann bspw. zum Zugriff auf Teile eines Arrays verwendet werden
- Der Zählvariable im `loop` einen Wert zuzuweisen, ist unzulässig