INVENT a CHIP

GEFÖRDERT VOM

Bundesministerium
für Bildung
und Forschung

VDE

Wettbewerb für Schüler*innen zum Chipdesign

VHDL-Coding-Style für IaC-Challenge

Institut für Mikroelektronische Systeme
Fachgebiet Architekturen und Systeme
Prof. Dr.-Ing. H. Blume

1 VHDL-Modulbeschreibung

In diesem Dokument wird exemplarisch dargestellt, wie VHDL-Module für die Invent a Chip-Challenge aufgebaut sein sollen. Es ist grundsätzlich darauf zu achten,

- dass der Quelltext möglichst übersichtlich formatiert ist und
- Signalnamen sinnvoll und eindeutig vergeben werden.

Die farbliche Hervorhebung des Quelltextes kann von der hier gezeigten Version abweichen, je nachdem welcher Editor verwendet wird.

1.1 Kommentare

Kommentare helfen beim Verständnis des Quellcodes für einen selbst und insbesondere andere. Zeilen-Kommentare folgen in VHDL auf einen Doppel-Bindestrich:

```
-- Dies ist ein Kommentar
```

1.2 Einrückung

Einrückungen des Quellcodes helfen der Übersichtlichkeit enorm. Generell sollte pro Zeile lediglich eine Anweisung stehen. Die Einrückung innerer Anweisungen in eine übergeordnete kann durch Leerzeichen, z.B. 2 oder 4 Leerzeichen pro Einrückungsebene erfolgen. Die Eingabe des Tabulatorzeichen im Web-Editor ist durch die Überbelegung mit Interface-Optionen herausfordernd.

Mehrere konkrete Beispiele sind im Folgenden dargestellt.

1.3 Struktur: entity

```
entity <entity_name> is
generic(
        -- Definition von „Generics“
        );
port(
        -- zuerst Eingangssignale
        clk       : in  std_ulogic;
        reset     : in  std_ulogic;
        data_in : in  std_ulogic_vector(7 downto 0);
        next_in : in  std_ulogic_vector(15 downto 0);

        -- dann Ausgangssignale
```

```
        out_1   : out std_ulogic;
        out_2   : out std_ulogic_vector(7 downto 0)
        );
end <entity_name>;
```

1.4 Struktur: architecture und processes

```
architecture <architecture_name> of <entity_name> is
-- Definition von Signalen, Konstanten und Registern
        signal reg_0_ff, reg_0_nxt : std_ulogic_vector(7 downto 0);
        signal reg_1_ff, reg_1_nxt : std_ulogic_vector(7 downto 0);
        -- <reg_name>_ff beschreibt Ausgangssignale eines Registers
        -- <reg_name>_nxt ist Eingangssignal im nächsten Taktzyklus

        signal count_ff , count_nxt : unsigned(7 downto 0);
        signal count_inc, count_rst : std_ulogic;
        -- Kontrollsignale direkt nach der Definition von Registern anlegen
        -- (in diesem Fall Beispielsignale für einen Zähler)

        -- Weitere Signale und Register …

begin
-- erster Prozess -> Alle Register anlegen
process(clk, reset)
begin
        if reset = '1' then
                -- reset ist asynchron und “high-aktiv”
                reg_0_ff <= (others => '0');
                reg_1_ff <= (others => '1');
                count_ff <= (others => '0');
                ...
        elsif rising_edge(clk) then
                -- Registerwerte nur bei steigender Taktflanke setzen
                reg_0_ff <= reg_0_nxt;
                reg_1_ff <= reg_1_nxt;
                count_ff <= count_nxt;
                ...
        end if;
end process;
```

```
-- zweiter Prozess -> kombinatorischer Prozess für eine FSM (falls das
-- Module eine enthält)
process(<fsm input signals>)
begin
        ...
end process;


-- im Weiteren -> Nur kombinatorische Prozess
-- Wichtig: Korrektheit der “Sensitivity List” überprüfen
-- -> in kombinatorischen Prozessen alle ausgewerteten Signale hinzufügen

-- Beispiel mit den vorher angelegten Registern
process(reg_0_ff, reg_1_ff, count_rst, count_ff, count_inc)
begin
        if <condition> then
                reg_0_nxt <= (others => '0');
                reg_1_nxt <= (others => '0');
        else
                reg_0_nxt <= reg_0_ff;
                reg_1_nxt <= reg_1_ff;
        end if;

        if count_rst = '1' then
                count_nxt <= (others => '0');
        elsif count_inc = '1' the
                count_nxt <= count_ff + 1;
        else
                count_nxt <= count_ff;
        end if;
end process;


-- Im Anschluss: kombinatorische Prozesse
<process_name_1> : process(<sensitivity list>)
begin
        ...
end process;


-- Verbindung der Registerausgangssignale mit den Ausgangssignalen der “Entity”
out_1 <= <internal signal>;
out_2   <= reg_1_ff;


end architecture <architecture_name>;
```

1.5 Bibliotheken

Grundsätzlich stehen, falls nicht anders angegeben, ausschließlich die folgenden Standard-Bibliotheken aus VHDL zur Verfügung.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
```

1.6 Reset

Der Reset eines Moduls soll, falls nicht anders angegeben, asynchron und bei einer logischen 1 (*high-active*) am Reset-Signal (*reset*) erfolgen.

1.7 Signale

Einige verbreitete Standardsignale sollten immer den gleichen Namen haben. Beispielsweise:

- „clk“ für das Taktsignal,
- „reset“ für einen high-active (aktiv bei logisch 1) reset-Signal,
- „enable“ für einen high-active (aktiv bei logisch 1) enable-Signal.

1.8 Konstanten

Konstanten sollten in VHDL möglichst im CAPS_CASE (komplett großgeschrieben mit Wortverknüpfungen durch Unterstriche) erfolgen.