Aritmética em VHDL

Hans Schneebeli
Depto de Engenharia Elétrica – UFES

Introdução

Os tipos std_logic_vector (definido em ieee.std_logic_1164) e bit_vector (pré-definido) não permitem operações numéricas. Assim é invalida a operação

```
q <= q + 1;
```

se q for definido como um vetor de bits ou um std_logic_vector.

Por outro lado, VHDL tem um tipo inteiro pré-definido (integer) que pode servir de base para novos tipos inteiros usando-se intervalos.

```
signal q : integer range 0 to 255;
```

Observar que para síntese é muito importante que se defina o intervalo, pois caso este não seja definido o signal terá, por default, 32 bits.

Dois intervalos já são pré-definidos:

natural	- de 0 a 2^{32} - 1
positive	– de 1 a 2^{32} - 1

Implementações não padrão

Para se poder fazer operações artiméticas com std_logic_vector foram desenvolvidas algumas pacotes. Um deles é o pacote std_logic_arith, desenvolvido pela Synopsys (Existem também um pacote disponibilizado pela Mentor Graphics) [1]. No entanto, mesmo que seja definida na maior parte das implementações como parte da bibliotecas ieee (use ieee.std_std_logic_arith), ela não é padrão e não faz parte da biblioteca ieee. Este pacote define tipos signed e unsigned além das operações aritméticas necessárias e as rotinas de conversão conv_signed, conv_unsigned e conv_integer. Mas ainda assim, não é possível se somar dois sinais do tipo std_logic_vector sem conversões explicitas. Para se poder fazer isto, foram desenvolvidos dois pacotes, também não padrão, std_logic_unsigned e std_logic_signed, que interpretam os sinais do tipo std_logic_vector como inteiros sem sinal ou com sinal respectivamente. Como não pode haver esta ambiguidade, somente um destes pacotes pode ser usado.

Implementação padrão

A maneira padrão de se fazer operações aritméticas com bit_vector ou std_logic_vector é com o uso dos pacotes padrão para síntese (Padrão IEEE 1076.3). Embora haja um pacote numeric_bit, o pacote numeric_std é mais usado.

Ambos pacotes definem os tipos unsigned e signed como vetores do tipo base (bit ou std_logic). Na verdade, as definições de std_logic_vector e de unsigned (em numeric_std) são identicas, ou seja,

```
type unsigned is array(natural range <>) of std_logic
```

A diferença reside na definição de operadores aritméticos e de conversão que existem no pacote numeric_std. Por esse motivo, a conversão entre unsigned e std_logic_vector é simples, usando-se apenas o nome do tipo de dados.

```
Slv <= std_logic_vector(u);
u   <= unsigned(slv);
```

Assim é possível se definir um sinal (ou uma variável) como abaixo e se poder fazer atribuições e comparações.

```
signal contador : unsigned(11 downto 0); -- 12 bits
      ...
      if contador = 202  then
            contador <= 0;
      else
            contador <= contador + 1;
      end if;
      ...
```

No entanto, não é possível se fazer operações aritméticas do tipo contador <= contador + 1 se contador for definido como std_logic_vector. Neste caso a conversão deve ser explicita.

```
signal contador: std_logic_vector(11 downto 0); -- 12 bits
      ...
      if contador = 202  then
            contador <= 0;
      else
            contador <= std_logic_vector(unsigned(contador) + 1);
      end if;
      ...
```

Primeiro, contador é convertido para um sinal do tipo unsigned (apenas reinterpretação dos bits pois não é necessário nenhum circuito), feita a operação de incremento e o resultado é convertido então de novo para std_logic_vector.

Operações

As operações abaixo são definidas no pacote numeric_std.

Operações aritméticas		
Arg1	**Arg2**	**Resultado**
unsigned	unsigned	unsigned
unsigned	integer	unsigned
integer	unsigned	unsigned
signed	signed	signed
signed	integer	signed
integer	signed	signed

Conversões

Considerando os sinais abaixo

```
signal slv: std_logic_vector(10 downto 0);
```

```
signal u: unsigned(10 downto 0);
signal s: signed(10 downto 0);
signal ui: integer range 0 to 2047;
signal si: integer range -1024 to 1023;
```

as seguintes conversões podem ser feitas usando o pacote numeric_std. As conversões são igualmente válidas para variáveis.

De std_logic_vector para unsigned ou signed

```
u <= unsigned(slv);
s <= signed(slv);
```

De unsigned ou signed para std_logic_vector

```
slv <= std_logic_vector(u);
slv <= std_logic_vector(s);
```

De unsigned ou signed para integer

```
ui <= to_integer(u);
si <= to_integer(s);
```

De integer para unsigned ou signed

Neste caso deve-se especificar o tamanho do vetor resultado

```
u <= to_unsigned(ui,11);
s <= to_signed(si,11);
```

ou melhor usando-se o atributo de comprimento

```
u <= to_unsigned(ui,u'length);
s <= to_signed(si,s'length);
```

De integer para std_logic_vector

Primeiro deve-se converter para unsigned (ou signed) e então para std_logic_vector.

```
slv <= std_logic_vector(to_unsigned(ui,slv'length));
slv <= std_logic_vector(to_signed(si,slv'length));
```

De std_logic_vector para integer

Primeiro deve-se especificar como o vetor de bits deve ser interpretado (unsigned ou signed) e então converte-lo para integer.

```
us <= to_integer(unsigned(slv));
is <= to_integer(signed(slv));
```

Resumo

Type Conversion	numeric_std
std_logic_vector -> unsigned	unsigned(*arg*)
std_logic_vector -> signed	signed(*arg*)
unsigned -> std_logic_vector	std_logic_vector(*arg*)
signed -> std_logic_vector	std_logic_vector(*arg*)
integer -> unsigned	to_unsigned(*arg*,*size*)
integer -> signed	to_signed(*arg*,*size*)
unsigned -> integer	to_integer(*arg*)
signed -> integer	to_integer(*arg*)

`integer -> std_logic_vector`	`integer -> unsigned/signed ->std_logic_vector`
`std_logic_vector -> integer`	`std_logic_vector -> unsigned/signed ->integer`
`unsigned + unsigned -> std_logic_vector`	`std_logic_vector(arg1 + arg2)`
`signed + signed -> std_logic_vector`	`std_logic_vector(arg1 + arg2)`

Redimensionamento

Existe a função resize que permite se redimensionar (para mais ou para menos) um sinal do tipo signed ou unsigned. Ela é particularmente importante para manipulação de números com sinal, pois ela os redimensiona preservando seu valor.

Referencias

[1] -, **IEEE library pitfalls** in **GHDL guide** (acessado através de http://ghdl.free.fr/ghdl/IEEE-library-pitfalls.html#IEEE-library-pitfalls).

[2] Jim Lewis , **VHDL Math Tricks of the Trade** (acessado através de http://www.synthworks.com/papers/vhdl_math_tricks_mapld_2003.pdf).

[3] Hendry, DC. **Arithmetic using numeric_std** (acessado através de http://wwwcad.eng.abdn.ac.uk/~eng186/eg3560/lecture14.pdf).

[4] IEEE. **Código fonte do pacote numeric_std** (acessado em http://www.cs.umbc.edu/help/VHDL/numeric_std.vhdl).

[5] Microelectronics Design Center. **VHDL Sources**. (acessado em http://dz.ee.ethz.ch/support/ic/hdl/vhdlsources.en.html).