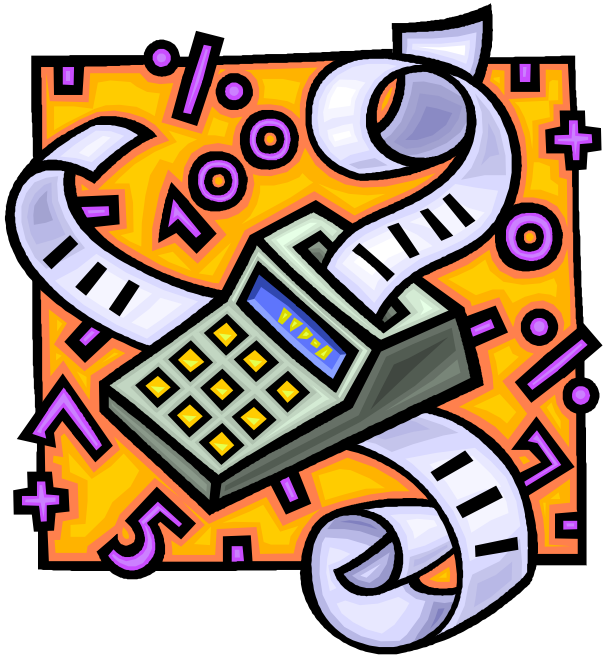
Arithmétique entière en VHDL

Arithmétique entière en VHDL : sujets de ce thème

- Exemple: conversion de couleurs de RGB à CMYK.
- Types à utiliser pour la synthèse.
- Opérateurs et fonctions arithmétiques.
- Conversion de types.



Exemple: conversion de couleurs de RGB à CMYK

- Les images sont habituellement encodées dans l'espace à trois dimensions RGB.
- Une imprimante utilise plutôt l'espace CMY: cyan (C), magenta (M) et jaune (Y), les couleurs complémentaires de rouge, vert et bleu.
- Les équations de conversion, pour des valeurs exprimées sur 8 bits, sont:
  C = 255 – R; M = 255 – G; Y = 255 – B
- Une imprimante utilise aussi une cartouche noire (K) pour économiser les couleurs et obtenir une meilleure qualité de noir: système CMYK.
- On réduit les valeurs (C, M, Y) de façon à utiliser le plus d'encre noire possible.

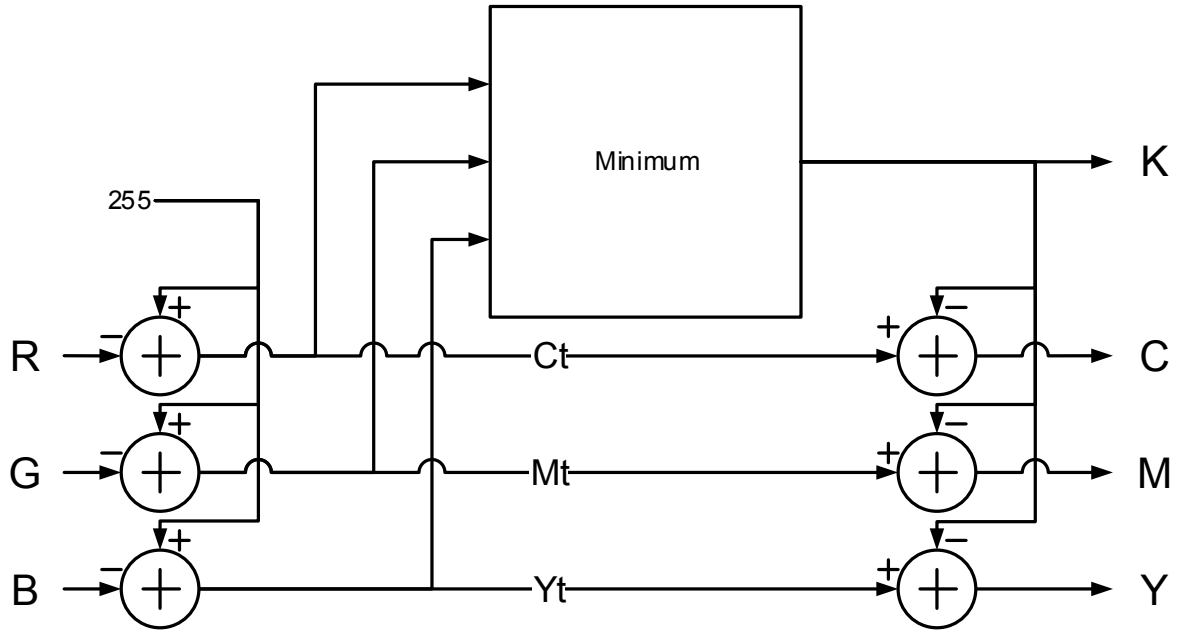
- Par exemple, on pourrait remplacer CMY = (250, 200, 200) par (50, 0, 0) + (200, 200, 200), où la dernière quantité correspond à du gris, obtenu de l'encre noire.
- On a donc les équations suivantes pour la conversion de RGB à CMYK:

$$Ct = 255 - R$$
$$Mt = 255 - G$$
$$Yt = 255 - B$$
$$K = \min(Ct, Mt, Yt)$$
$$C = Ct - K$$
$$M = Mt - K$$
$$Y = Yt - K$$

A. Stodghill, Tip o'day: ask for a a refill, Green Options, 2007/06/18. Consulté le 4 septembre 2009, tiré de http://greenoptions.com/tag/ink-cartridge

Conversion de couleurs de RGB à CMYK: analyse du problème

- Besoin de représenter des valeurs numériques, quels types choisir?
  - Valeurs entières
  - Entre 0 et 255 inclusivement
- Options:
  - `real`: non! trop précis et pas synthétisable
  - `integer`, `natural`: peut-être; `positive`: non (min: 1)
  - `signed`, `unsigned`: peut-être
  - `std_logic_vector`: non, trop bas niveau
- Besoin de l'opération de soustraction
- Besoin de l'opération minimum



$$Ct = 255 - R; Mt = 255 - G; Yt = 255 - B$$

$$K = \min(Ct, Mt, Yt)$$

$$C = Ct - K; M = Mt - K; Y = Yt - K$$

Types VHDL à utiliser pour la synthèse

- **`real`: ça dépend**
  - pour des valeurs constantes: ok
  - pour des registres et valeurs intermédiaires: non! trop précis et *pas synthétisable*.
- **`integer`, `natural`, `positive`: acceptables**
  - Bien supportés par les synthétiseurs pour les opérations arithmétiques.
  - Bonne abstraction par rapport à un vecteur de bits.
  - Important de spécifier la gamme de valeurs possibles de façon à contraindre les ressources matérielles utilisées pour les représenter.
    Par défaut: 32 bits pour chaque signal.
- **`signed`, `unsigned`: acceptables**
  - Définis dans le package normalisé `numeric_std`, comme des tableaux de `std_logic`.
  - Bien supportés par les outils de synthèse:
    un fil/un registre par bit
  - Le package `numeric_std` redéfinit les opérateurs de VHDL pour ces deux types.
- **`std_logic_vector`: pas pour l'arithmétique**
  - Défini dans le package `std_logic_1164` comme un tableau de `std_logic`.
  - Des packages populaires incluent des définitions pour les opérations arithmétiques, mais ils ne sont pas normalisés et *leur utilisation n'est pas recommandée pour représenter des nombres*.

Opérateurs et fonctions arithmétiques VHDL à utiliser pour la synthèse

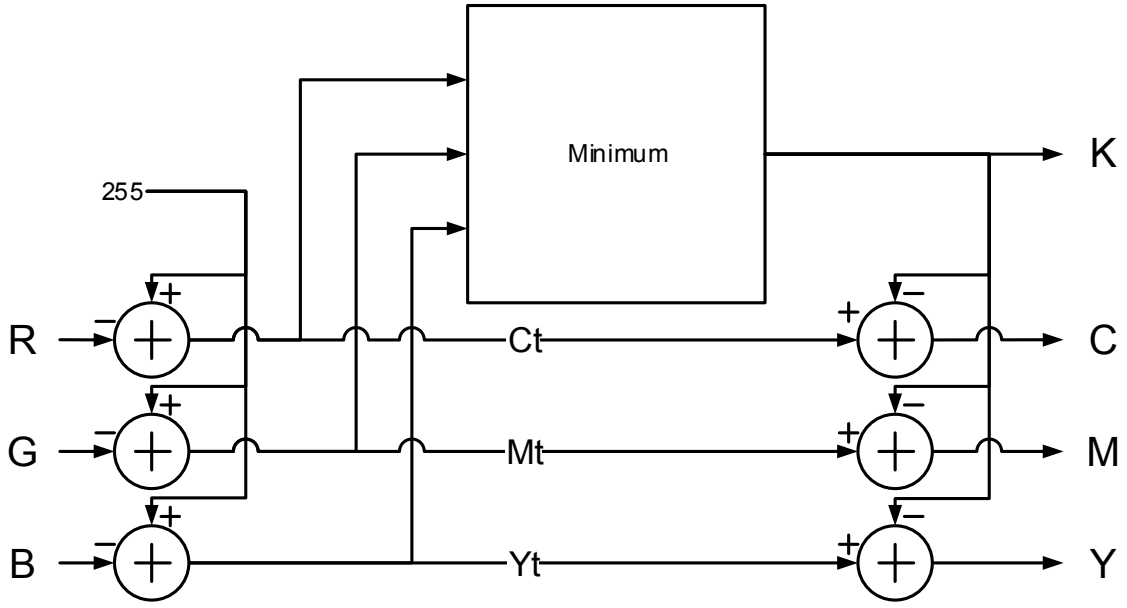
- +, -, *, abs
  - Ces opérateurs sont synthétisables.
- / , mod, rem
  - Ces opérateurs ne sont synthétisables que si l'opérande de droite est une puissance de deux.
- shift_left(arg, count), shift_right(arg, count), resize(arg, size)
  - Ces fonctions sont synthétisables.

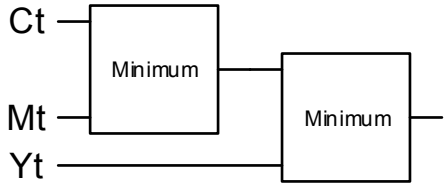
```
-- Exemples de code synthétisable
-- Les tailles des expressions
-- et des destinations doivent correspondre!
A <= B + C + abs(D);
E <= B / 64;
F <= shift_left(B, 3);
G <= resize(A, G'length);
```

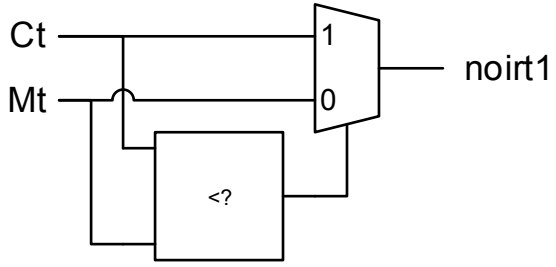
Package `numeric_std`
Opérateur définis pour les types `signed` et `unsigned`
http://www.eda.org/rassp/vhdl/models/standards/numeric_std.vhd

Opérateur ou fonction	Type opérande 1	Type opérande 2	Type résultat
+, -, *, /, mod, rem	signed	signed	signed
	signed	integer	signed
	unsigned	unsigned	unsigned
	unsigned	natural	unsigned
abs(arg), -	signed		signed
shift_left(arg, count), shift_right(arg, count) (décalage)	signed	natural	signed
	unsigned	natural	unsigned
resize(arg, size) (redimension avec extension du signe)	signed	natural	signed
	unsigned	natural	unsigned

Conversion de couleurs de RGB à CMYK: code VHDL








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

entity convRGB2CMYK is
   port (
   rouge, vert, bleu : in unsigned(7 downto 0);
   cyan, magenta, jaune, noir : out unsigned(7 downto 0)
);
end convRGB2CMYK;

architecture arch2 of convRGB2CMYK is

begin

   process(rouge, vert, bleu)
   variable cyant, magentat, jaunet, noirt1, noirt2 : unsigned(7 downto 0);
   begin

      cyant := 255 - rouge;
      magentat := 255 - vert;
      jaunet := 255 - bleu;

      if cyant < magentat then noirt1 := cyant; else noirt1 := magentat; end if;
      if noirt1 < jaunet then noirt2 := noirt1; else noirt2 := jaunet; end if;

      cyan <= cyant - noirt2;
      magenta <= magentat - noirt2;
      jaune <= jaunet - noirt2;
      noir <= noirt2;

   end process;

end arch2;
```

Conversion de couleurs de RGB à CMYK : banc d'essai

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

entity convrgb2cmyk_tb is
end convrgb2cmyk_tb;

architecture TB_ARCHITECTURE of convrgb2cmyk_tb is

signal rouge, vert, bleu, cyan, magenta, jaune, noir : UNSIGNED(7 downto 0);

type pixelRGB_type is record
    rouge : integer range 0 to 255;
    vert : integer range 0 to 255;
    bleu : integer range 0 to 255;
end record;

type tableau_pixelRGB is array (natural range <>) of pixelRGB_type;
constant vecteurs : tableau_pixelRGB := (
(0,0,0),(1,1,1),(2,2,2),(100,100,100),(128,128,128),(200,200,200),
(254,254,254),(255,255,255),
(0,255,255),(255,0,255),(255,255,0),(0,0,255),(0,255,0),(255,0,0),
(1,100,254),(1,254,100),(100,1,254),(100,254,1),(254,1,100),(254,100,1)
--(-1,-1,-1),(-1,0,0),(0,-1,0),(0,0,-
1),(256,256,256),(256,0,0),(0,256,0),(0,0,256)
);
```

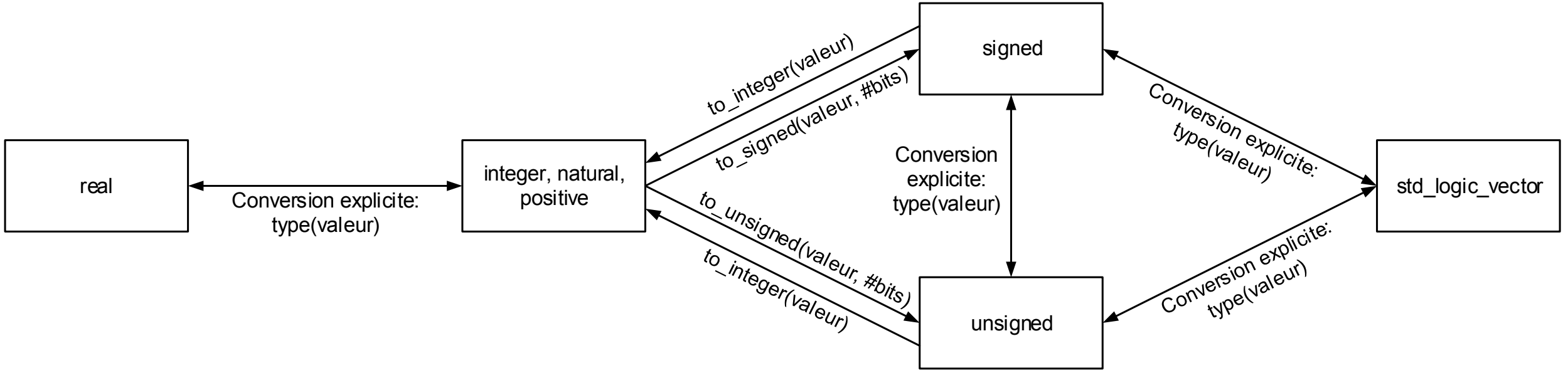
```
begin

    UUT : entity convrgb2cmyk(arch2)
    port map (rouge, vert, bleu, cyan, magenta, jaune, noir);

    process
    begin
        for k in vecteurs'range loop
            rouge <= to_unsigned(vecteurs(k).rouge, 8);
            vert <= to_unsigned(vecteurs(k).vert, 8);
            bleu <= to_unsigned(vecteurs(k).bleu, 8);

            wait for 10 ns;

            assert 255 - (to_integer(noir) + to_integer(cyan)) = vecteurs(k).rouge
            report "erreur pour l'entrée " & integer'image(k) &
            ", rouge: " & integer'image(vecteurs(k).rouge) & ", cyan: " &
                integer'image(to_integer(cyan))
            severity error;

            assert 255 - (to_integer(noir) + to_integer(magenta)) = vecteurs(k).vert
            report "erreur pour l'entrée " & integer'image(k) &
            ", vert: " & integer'image(vecteurs(k).vert) & ", magenta: " &
                integer'image(to_integer(magenta))
            severity error;

            assert 255 - (to_integer(noir) + to_integer(jaune)) = vecteurs(k).bleu
            report "erreur pour l'entrée " & integer'image(k) &
            ", bleu: " & integer'image(vecteurs(k).bleu) & ", jaune: " &
                integer'image(to_integer(jaune))
            severity error;

        end loop;

        assert false report "simulation terminée" severity failure;

    end process;
end TB_ARCHITECTURE;
```

Assigner la valeur d'une expression à un objet

- Pour assigner la valeur d'une expression à un objet (signal, variable), il faut:
  - que les types soient compatibles
  - que les dimensions soient les mêmes
- Il faut aussi penser au sens de l'expression!


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

entity democode2 is
   port (
   A8, B8 : in signed(7 downto 0);
   R8 : out signed(7 downto 0);
   R9, S9, T9 : out signed(8 downto 0);
   R7 : out signed(6 downto 0)
   );
end;

architecture arch of democode2 is

begin

   R8 <= A8 + B8; -- ok, mais risque de débordement

   S8 <= A8 + 100; -- ok, mais risque de débordement

   R9 <= (A8(A8'left) & A8) + (B8(B8'left) & B8); -- ok

   S9 <= resize(A8, S9'length) + resize(B8, S9'length); -- ok

   --T9 <= A8 + B8; -- non, largeurs incompatibles!

   R7 <= A8(6 downto 0) + B8(6 downto 0); -- ok, mais ...

end arch;
```

Conversions de types: vue d'ensemble



Vous devriez maintenant être capable de …

- Utiliser correctement les types VHDL pour représenter des nombres dans un modèle. (B3)
- Choisir le type numérique le plus approprié à un problème en tenant compte de la précision des calculs et de la complexité du circuit synthétisé. (B3)
- Utiliser correctement les opérateurs arithmétiques et de comparaison dans un modèle VHDL. (B3)
- Utiliser correctement les fonctions de conversion entre les types numériques. (B3)

Code	Niveau (http://fr.wikipedia.org/wiki/Taxonomie_de_Bloom)
B1	Connaissance - mémoriser de l'information.
B2	Compréhension – interpréter l'information.
B3	Application – confronter les connaissances à des cas pratiques simples.
B4	Analyse – décomposer un problème, cas pratiques plus complexes.
B5	Synthèse – expression personnelle, cas pratiques plus complexes.