Revue du projet GenCod

Adrien Guatto et Marc Pouzet

Mardi 5 Octobre 2010

Contexte scientifique

Question soulevée dans le cadre du projet GenCod

Comment générer une description de matériel à partir de programmes écrits en Scade ?

Contexte scientifique

Question soulevée dans le cadre du projet GenCod

Comment générer une description de matériel à partir de programmes écrits en Scade ?

Approche retenue

Étudier la question et proposer une solution dans le cadre d'un langage laboratoire simplifié mais suffisamment proche de Scade 6 pour un transfert ultérieur vers celui-ci.

Contexte scientifique

Question soulevée dans le cadre du projet GenCod

Comment générer une description de matériel à partir de programmes écrits en Scade ?

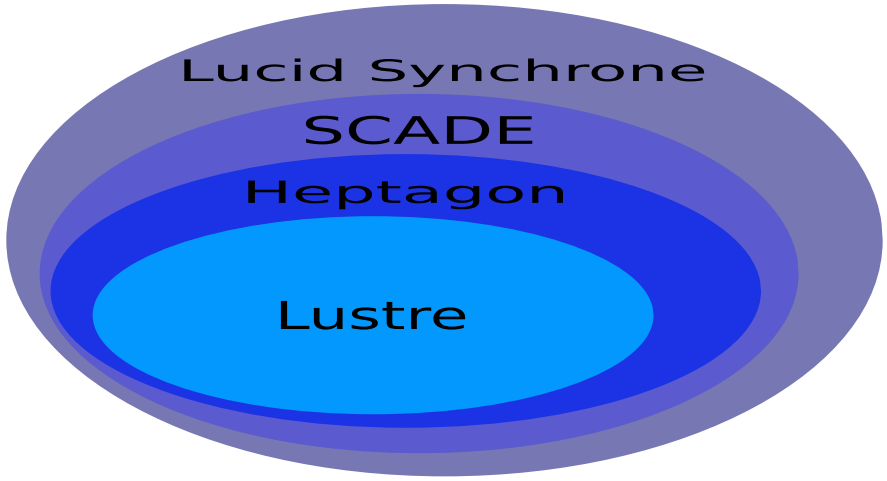
Approche retenue

Étudier la question et proposer une solution dans le cadre d'un langage laboratoire simplifié mais suffisamment proche de Scade 6 pour un transfert ultérieur vers celui-ci.

Véhicule de l'expérience

Heptagon est un sous-ensemble de Lucid Synchrone, langage synchrone académique (Pouzet et al.) dont les principes de constructions furent repris dans Scade 6

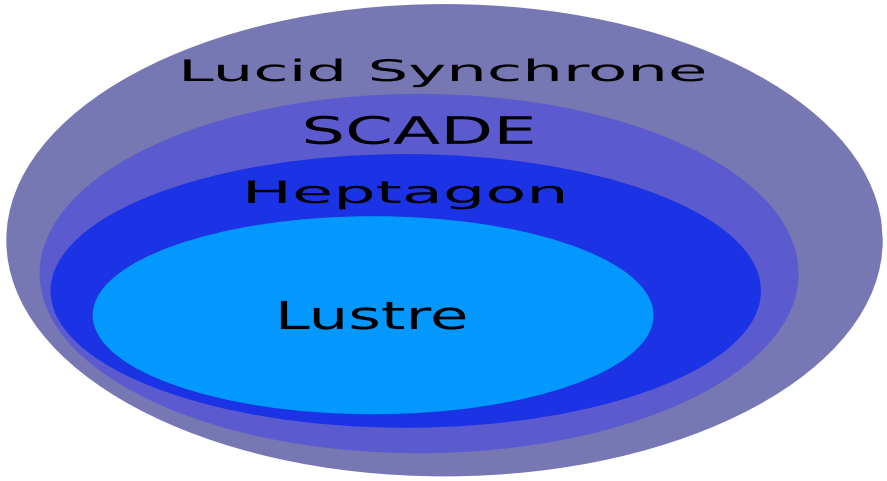
Réalisation



Les traits marquants d'Heptagon sont :

- Les programmes sont structurés en noeuds contenants des équations de suites ou des automates.
- Le processus de compilation vérifie des propriétés de sûreté et raffine le programme jusqu'à générer du code impératif (e.g. langage C).
- Il est possible de générer du code VHDL.

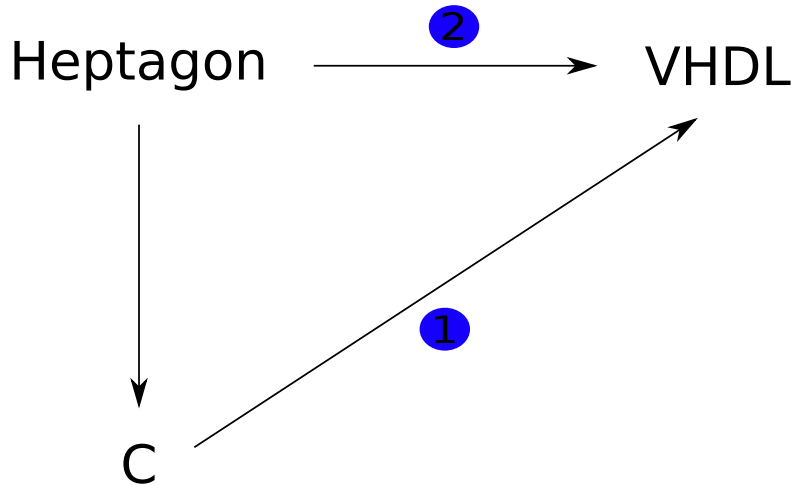
Réalisation



Les traits marquants d'Heptagon sont :

- Les programmes sont structurés en noeuds contenants des équations de suites ou des automates.
- Le processus de compilation vérifie des propriétés de sûreté et raffine le programme jusqu'à générer du code impératif (e.g. langage C).
- Il est possible de générer du code VHDL.

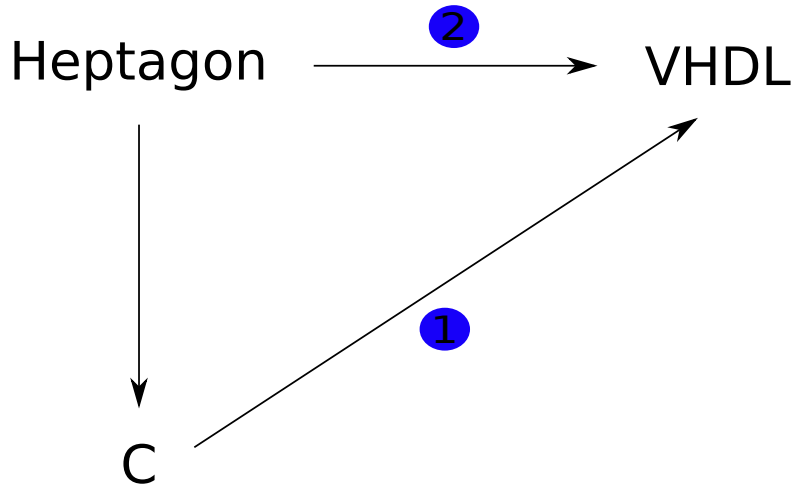
Génération de code VHDL



Pour produire du code VHDL, deux points de départ sont possibles :

1. Le code C produit par le compilateur original (Scade ou Heptagon). Cette approche est celle de la société GeenSoft.
2. La représentation intermédiaire à flots de données interne au compilateur du langage synchrone.

Génération de code VHDL



Pour produire du code VHDL, deux points de départ sont possibles :

1. Le code C produit par le compilateur original (Scade ou Heptagon). Cette approche est celle de la société GeenSoft.
2. La représentation intermédiaire à flots de données interne au compilateur du langage synchrone.

Nous avons retenu le second cheminement, et allons l'illustrer sur des exemples dans les transparents suivant.

Exemple 1 : compteur - code original

```
node compteur(tick, top : bool) returns (o : int)
var pres : int;
let
  pres = if tick then 1 else 0;
  reset o = (0 fby o) + pres every top;
tel

node main() returns (o : int)
let
  o = compteur(true  fby  true fby false fby true,
               false fby false fby true fby false);
tel
```

Exemple 1 : compteur - exemple de sortie attendue

$tick$	t	t	t	f	$\ldots$
top	f	f	t	f	$\ldots$
$compteur(tick, top)$	1	2	1	1	$\ldots$

Exemple 1 : compteur - code MiniLS

```
node compteur(tick : bool; top : bool) returns (o : int)
var pres : int;
let
  o = (if top then 0 else 0 fby o) + pres;
  pres = if tick then 1 else 0
tel
```

Le code n'est plus formé que d'équations.

Exemple 1 : compteur - code MiniLS sans reset

```
node compteur(rst_2 : bool; tick : bool; top : bool)
     returns (o : int)
let
  o = (if top then 0 else (if rst_2 then 0 else (0 fby o)))
    + (if tick then 1 else 0)
tel
```

La réinitialisation logique est explicitée via un paramètre du nœud.

Exemple 1 : compteur - code MiniLS final

```
node compteur(rst_2 : bool; tick : bool; top : bool) returns (o : int)
var _v_28 : int; _v_27 : int; _v_26 : int;
let
  _v_27 =
    merge top
      (true -> (0 when true(top)))
      (false ->
        (merge rst_2
           (true -> (0 when true(rst_2)))
           (false -> (_v_26 when false(rst_2)))
         when false(top)));
  _v_28 =
    merge tick (true -> (1 when true(tick))) (false -> (0 when false(ti
  o = _v_27 + _v_28;
  _v_26 = 0 fby o
tel
```

Le code est normalisé et ordonnancé.

Exemple 1 : compteur - code VHDL

```
use work.compteur.all;

library ieee;
use ieee.std_logic_1164.all;

entity compteur is
  port (signal clk_1 : in std_logic;
        signal hw_rst_3 : in std_logic;
        signal rst_2 : in std_logic;
        signal tick : in std_logic;
        signal top : in std_logic;
        signal o_o : out integer);
end entity compteur;

architecture rtl of compteur is
  signal h_v_26 : integer;
begin
  update : process (clk_1, hw_rst_3, rst_2,
                    tick, top)
    variable h_v_27 : integer;
    variable h_v_28 : integer;
    variable o : integer;
  begin
    case top is
      when '1' => h_v_27 := 0;
      when '0' => case rst_2 is
                    when '1' => h_v_27 := 0;
                    when '0' => h_v_27 := h_v_26;
                  end case;
    end case;
    case tick is
      when '1' => h_v_28 := 1;
      when '0' => h_v_28 := 0;
    end case;
    o := (h_v_27 + h_v_28);
    if (hw_rst_3 = '1') then
      h_v_26 <= 0;
    elsif rising_edge(clk_1) then
      h_v_26 <= o;
    end if;
    o_o <= o;
  end process update;
end architecture rtl;
```

Exemple 1 : compteur - comportement du code VHDL