Revue du projet GenCod

Adrien Guatto et Marc Pouzet
LRI

Mardi 5 Octobre 2010

Contexte scientifique

Question soulevée dans le cadre du projet GenCod

Générer une description de matériel à partir de programmes écrits en Scade 6.

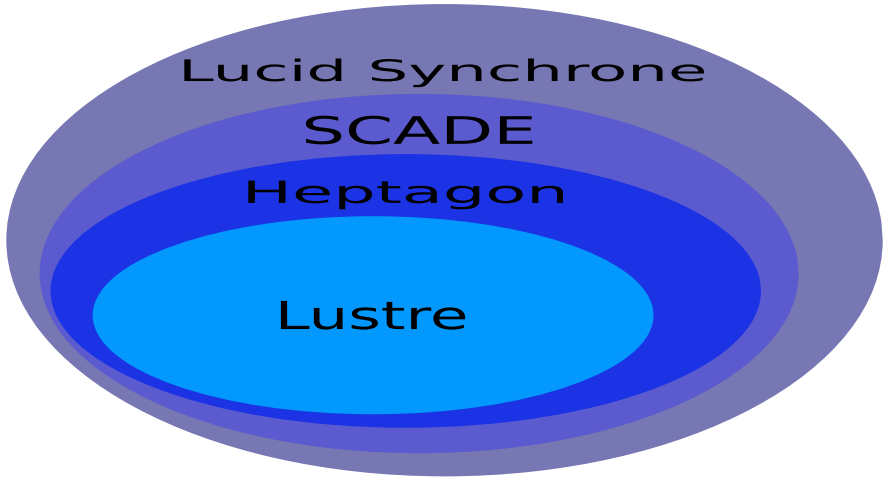
Approche retenue

- Génération modulaire de code VHDL à partir de code data-flow.
- Mise en oeuvre dans un prototype (Heptagon).

Prototype

- Un sous-ensemble du langage (académique) Lucid Synchrone (Pouzet et al.).
- Traits principaux (data-flow, automates, tableaux) présents dans SCADE 6.

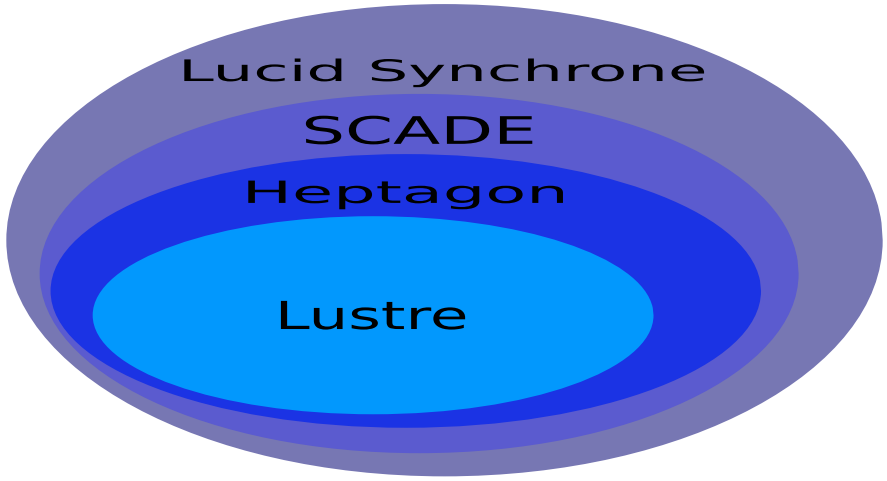
Réalisation



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.

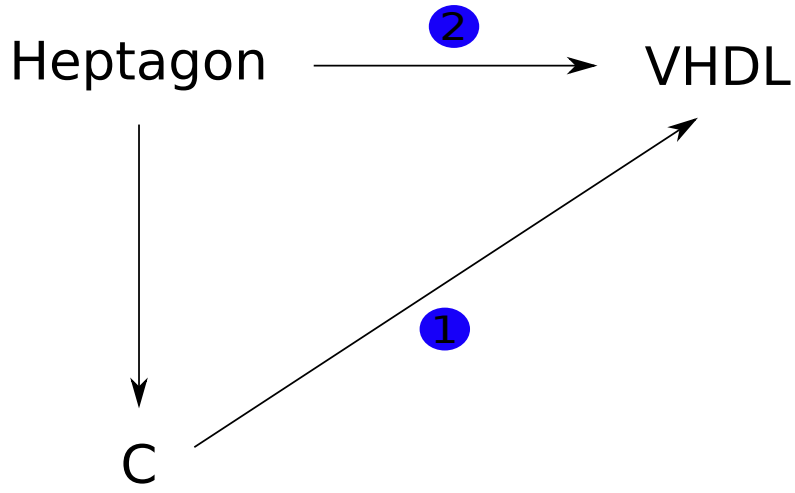
Réalisation



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, on considère la génération de code depuis un langage intermédiaire data-flow “gardé” (utilisée dans les langages Lucid Synchrone et Scade 6).
La compilation est une suite de réécritures du programme data-flow.

- Chacune est formellement spécifiée et concise.
- Chaque étape intermédiaire peut-être vérifiée à posteriori.

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*	*f*	*t*	...
top	*f*	*f*	*t*	*f*	...
o	1	2	0	1	...

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(tick)));
  o = _v_27 + _v_28;
  _v_26 = 0 fby o
tel
```

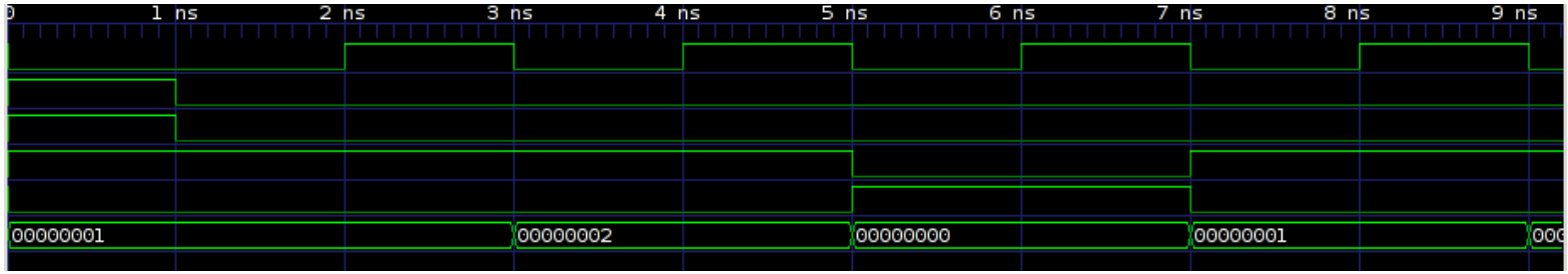
Le code est normalisé et ordonnancé.

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, h_v_26)
    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 - caractéristiques du code VHDL

Simulation comportementale



Synthèse

L'outil industriel *Xilinx XST* synthétise une netlist avec des propriétés satisfaisantes :

- Un additionneur 32 bits pour **o**.
- Un registre 32 bits pour **h_v_26**.

Exemple 2 : compteur de bits - code original

```
node countbits(x : bool^n; mask : bool^n; sw_mode : bool)
      returns (o : int)
var bits : bool^n;
let
  automaton
    state Simple
      do bits = x;
      unless sw_mode then Masked
    state Masked
      do bits = map (&) <<n>>(x, mask);
      unless sw_mode then Simple
  end;

  o = fold (+) <<n>> (map int_of_bool <<n>>(bits), 0);
tel
```

Exemple 2 : compteur de bits - exemple de sortie attendue

Ici, $n = 3$.

x	$[t, f, f]$	$[t, t, t]$	$[t, t, t]$	$[t, t, t]$	...
$mask$	$[t, f, t]$	$[t, f, t]$	$[t, f, t]$	$[t, f, t]$	...
sw_mode	f	t	t	f	...
o	1	2	3	3	...

Exemple 2 : compteur de bits - code MiniLS

```
node countbits(x : bool^n; mask : bool^n;
               sw_mode : bool) returns (o : int)
var t_71_t_66_87 : bool; t_62_t_57_86 : bool;
    t_53_nr_31_81 : bool; t_49_nr_29_80 : bool;
    ck_33 : __states__1; ck_28 : __states__1;
    pnr_19 : bool;
let
  t_71_t_66_87 = (sw_mode when __Masked__3(ck_33));
  t_62_t_57_86 = (sw_mode when __Simple__2(ck_33));
  t_53_nr_31_81 = (false when __Masked__3(ck_28));
  t_49_nr_29_80 = (false when __Simple__2(ck_28));
  ck_33 =
    __Simple__2 fby
      merge ck_28
        (__Masked__3 ->
           (__Masked__3 when __Masked__3(ck_28)))
        (__Simple__2 ->
           (__Simple__2 when __Simple__2(ck_28)));
  ck_28 =
    merge ck_33
      (__Masked__3 ->
        if t_71_t_66_87
        then (__Simple__2 when __Masked__3(ck_33))
        else (__Masked__3 when __Masked__3(ck_33)))
      (__Simple__2 ->
        if t_62_t_57_86
        then (__Masked__3 when __Simple__2(ck_33))
        else (__Simple__2 when __Simple__2(ck_33)));
  pnr_19 =
    false fby merge ck_28
      (__Masked__3 -> t_53_nr_31_81)
      (__Simple__2 -> t_49_nr_29_80);
  o =
    fold (+)<<n>>
      (map (int_of_bool())<<n>>
         (merge ck_28
            (__Masked__3 ->
               map (&)<<n>>
                 (x when __Masked__3(ck_28),
                  mask when __Masked__3(ck_28)))
            (__Simple__2 ->
               (x when __Simple__2(ck_28)))),
       0)
tel
```

Exemple 2 : compteur de bits - code MiniLS sans reset

```
node countbits(rst_2 : bool; x : bool^n; mask : bool^n; sw_mode : bool)
      returns (o : int)
var t_71_t_66_87 : bool; t_62_t_57_86 : bool; t_53_nr_31_81 : bool;
    t_49_nr_29_80 : bool; ck_33 : __states__1; ck_28 : ; pnr_19 : bool;
let
  ...
  ck_33 =
    if rst_2
    then __Simple__2
    else (__Simple__2 fby
            merge ck_28
              (__Masked__3 -> (__Masked__3 when __Masked__3(ck_28)))
              (__Simple__2 -> (__Simple__2 when __Simple__2(ck_28))));
  pnr_19 =
    if rst_2
    then false
    else (false fby
            merge ck_28 (__Masked__3 -> t_53_nr_31_81)
                        (__Simple__2 -> t_49_nr_29_80);
  o = fold (+) <<n>>(map int_of_bool <<n>>
                       (rst_2^n,
                        merge ck_28
                          (__Masked__3 ->
                             map (&) <<n>> (x when __Masked__3(ck_28),
                                            mask when __Masked__3(ck_28)))
                          (__Simple__2 -> (x when __Simple__2(ck_28)))),
                     0)
tel
```

Exemple 2 : compteur de bits - code MiniLS final

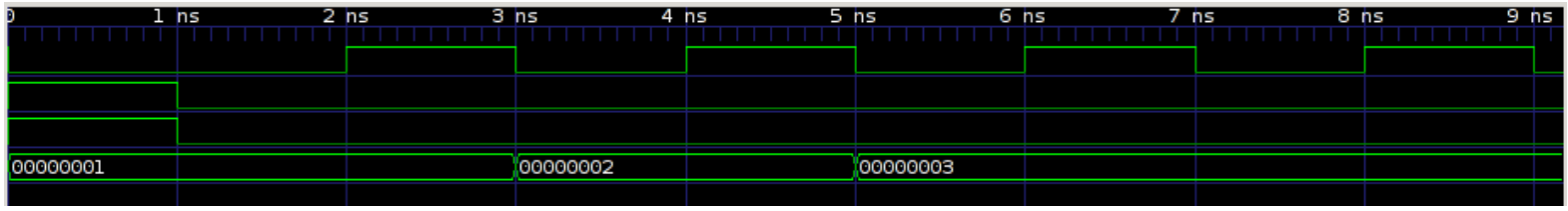
```
node countbits(rst_2 : bool; x : bool^n; mask : bool^n;
               sw_mode : bool) returns (o : int)
var ...;
let
  _v_116 = rst_2^n;
  ck_33 =
    merge rst_2
      (true -> (__Simple__2 when true(rst_2)))
      (false -> (_v_111 when false(rst_2)));
  pnr_19 =
    merge rst_2
      (true -> (false when true(rst_2)))
      (false -> (_v_113 when false(rst_2)));
  t_71_t_66_87 = (sw_mode when __Masked__3(ck_33));
  t_62_t_57_86 = (sw_mode when __Simple__2(ck_33));
  ck_28 = merge ck_33 (__Masked__3 -> ...)
                      (__Simple__2 -> ...);
  _v_114 = (x when __Masked__3(ck_28))
         & (mask when __Masked__3(ck_28));
  _v_110 = merge ck_28
      (__Masked__3 ->
         (__Masked__3 when __Masked__3(ck_28)))
      (__Simple__2 ->
         (__Simple__2 when __Simple__2(ck_28)));
  t_49_nr_29_80 = (false when __Simple__2(ck_28));
  t_53_nr_31_81 = (false when __Masked__3(ck_28));
  _v_115 =
    merge ck_28 (__Masked__3 -> ...)
                (__Simple__2 -> ...));
  _v_112 =
    merge ck_28 (__Masked__3 -> t_53_nr_31_81)
                (__Simple__2 -> t_49_nr_29_80);
  _v_147 = _v_116[0];
  _v_111 = __Simple__2 fby _v_110;
  _v_113 = false fby _v_112;
  _v_143 = _v_116[2];
  _v_144 = _v_115[2];
  _v_148 = _v_115[0];
  z_136 = int_of_bool(_v_143, _v_144);
  z_134 = int_of_bool(_v_147, _v_148);
  _v_145 = _v_116[1];
  _v_146 = _v_115[1];
  z_135 = int_of_bool(_v_145, _v_146);
  _v_117 = [z_136; z_135; z_134];
  _v_142 = _v_117[0];
  _v_140 = _v_117[2];
  _v_141 = _v_117[1];
  z_137 = ( + )(_v_142, 0);
  z_138 = ( + )(_v_141, z_137);
  z_139 = ( + )(_v_140, z_138);
  o = z_139
tel
```

Exemple 2 : compteur de bits - code VHDL final

```
...;

entity countbits is
  port (signal clk_1 : in std_logic;
        signal hw_rst_3 : in std_logic;
        signal rst_2 : in std_logic;
        ...);
end entity countbits;

architecture rtl of countbits is
  ...;
  component int_of_bool
    port (signal clk_1 : in std_logic;
          signal hw_rst_3 : in std_logic;
          signal rst_2 : in std_logic;
          signal b : in std_logic;
          signal o_o : out integer);
  end component;
  for int_of_bool3: int_of_bool
    use entity work.int_of_bool;
  ...;
begin
  update : process (clk_1, hw_rst_3, z_135, ...)
    variable h_v_116 : std_logic_vector (0 to 2);
    ...;
  begin
    ...;
    h_v_114 := (x and mask);
    ...;
    if (hw_rst_3 = '1') then
      h_v_111 <= h_u_Simple_u_2;
    elsif rising_edge(clk_1) then
      h_v_111 <= h_v_110;
    end if;
    ...;
    arg_ck_3 <= clk_1;
    arg_v_143 <= h_v_143;
    arg_v_144 <= h_v_144;
    ...;
    z_137 := (h_v_142 + 0);
    z_138 := (h_v_141 + z_137);
    z_139 := (h_v_140 + z_138);
    o := z_139;
    o_o <= o;
  end process update;
  int_of_bool3: int_of_bool
  port map (clk_1 => arg_ck_3,
            hw_rst_3 => hw_rst_3,
            rst_2 => arg_v_143,
            b => arg_v_144,
            o_o => z_136);
  ...;
end architecture rtl;
```

Exemple 2 : compteur de bits - caractéristiques du code VHDL

Simulation comportementale



Synthèse

Xilinx XST synthétise une netlist avec des propriétés satisfaisantes :

- Deux additionneurs 32 bits.
- Un registre 1 bit.

Conclusion

- Le langage traité est un sous-ensemble de Scade 6.
- Le prototype est raisonnablement petit.
- Les langages intermédiaires data-flow des compilateurs synchrones sont un bon point d'entrée pour la génération de code VHDL.