Présentation de GHDL

Tristan Gingold

1. Conception des circuits intégrés.
2. VHDL, un langage matériel.
3. GHDL, une implémentation de VHDL.
4. Conclusion.

http://ghdl.free.fr/

Fabrication des circuits intégrés

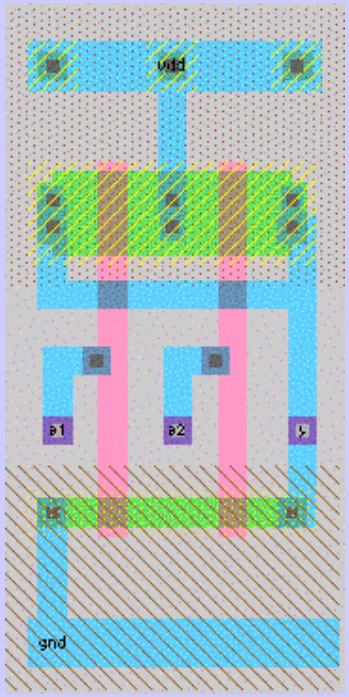
- Matière de base : silicium (élément simple et très répandu).
  - Purification et mise sous forme de cylindre.
  - Découpe en galettes (wafer).

- Ensemble de traitements physiques et chimiques, appliqués successivement :
  - Dopage (par diffusion),
  - Métallisation,
  - Gravure.

- Test final, découpage et packaging.

Temps de fabrication relativement long.

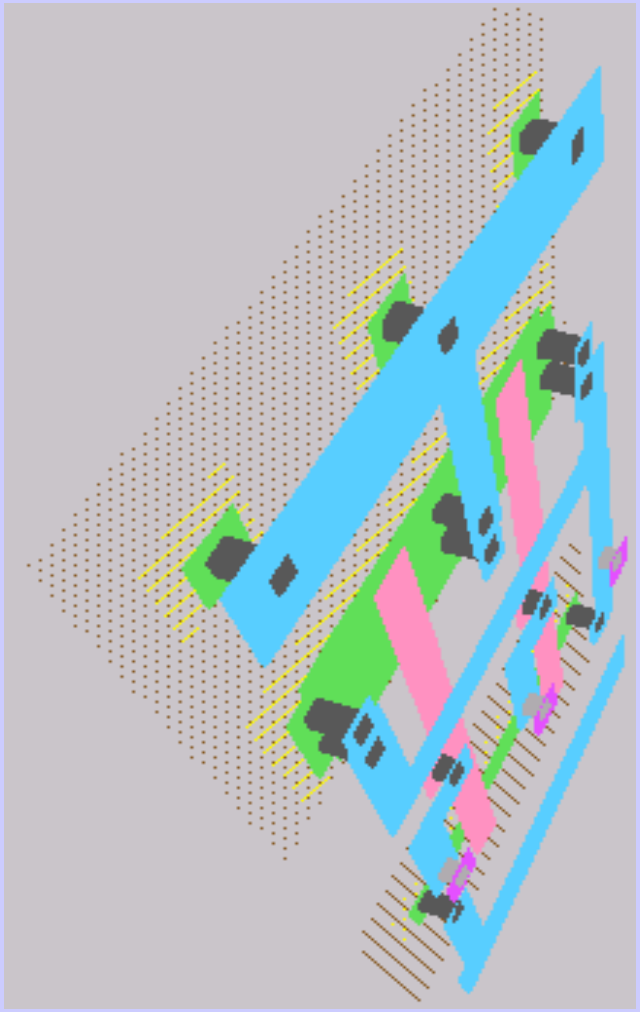
Niveau masques



Dessin des masques par CAO:

- Chaque couleur représente un masque pour un traitement.
- Toujours utilisé pour la conception des cellules.
- Aide : vérification des règles de conception (distance minimale).

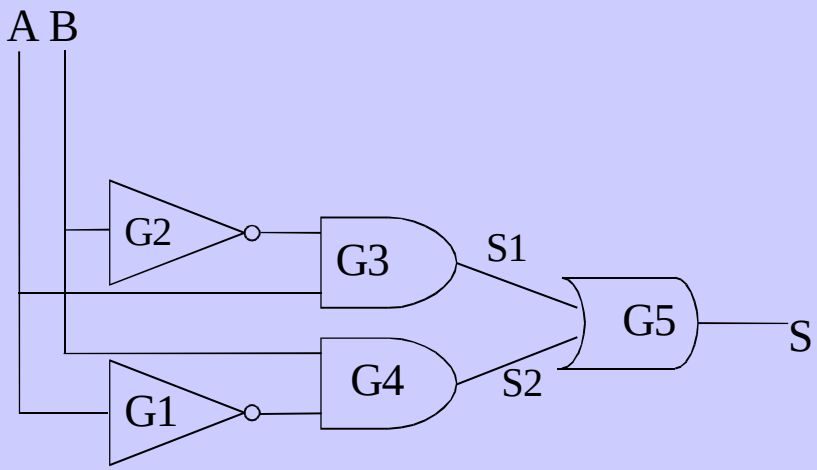
Porte NAND



Niveau Netlist

Exemple : porte *OU-exclusif*

Schéma :



Fichier :

```
Input A, B;
Output S;
Net S1, S2;
Net nA, nB;

G1: Not (nA, A);
G2: Not (nB, B);
G3: And (S1, nB, A);
G4: And (S2, B, nA);
G5: Or (S, S1, S2);
```

→ Premiers outils de CAO (placement/routage)

Niveau RTL

- RTL : register transfer language.
- Exemple : porte *OU-exclusif* :

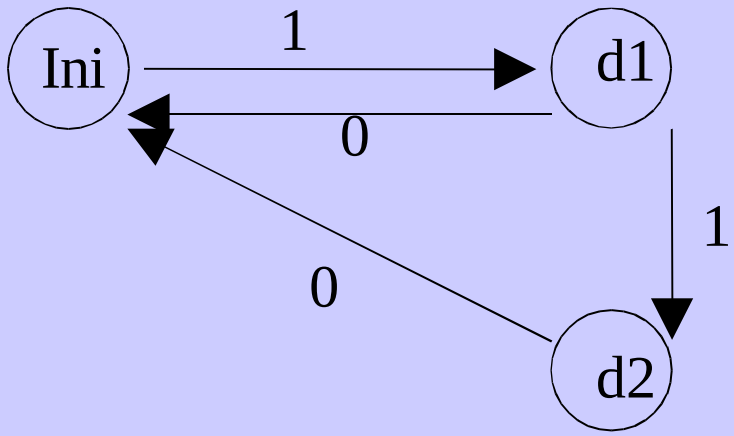
```
input A, B;
output S;

S := (A and (not B)) or ((not A) and B);
```

➔Abstractions supplémentaires.

➔Outils d’optimisation.

Niveau Synthèse

- Exemple : FSM (finite state machine) pour la détection de deux 1 consécutifs.




```
Input I;
Clock CLK;
Reset RST;
Output Detect;

State Ini, d1, d2;
Init Ini;

Ini: d1 when I = 1;
      Ini otherwise;
d1: d2 when I = 1;
     Ini otherwise;
d2: Ini otherwise;
Detect := State = d2;
```

Niveau comportemental

- Description très haut niveau : pas de contraintes temporelles, types complexes.
- Exemple : PGCD

```
int pgcd (int a, int b)
{
  if  (b > a)
    return pgcd (b, a);
  if (b == 0)
    return a;
  return pgcd (b, a % b);
}
```

VHDL, histoire

- ~1980 : Appel d'offre du DOD pour la conception d'un Hardware Design Language (HDL).
- La proposition de IBM, Texas Instrument et Intermetrics (Very High Speed Integrated Circuit HDL ou VHDL 7.2) est retenue.
- 1985 : La spécification de VHDL est publique.
- 1987 : Première normalisation par IEEE (1076).
- 1993 : Deuxième normalisation.
- 1998 : Ajout des variables partagées.
- 2001 : Troisième normalisation (corrections).
- 200x : Prochain standard (!)

Aperçu : Netlist

```
entity and2 is
  port (a, b: in bit;
        o : out bit);
end and2;


architecture netlist of and2 is

   component nand2
      port (a, b : in bit; o: out bit);
   end component;

   component not1
      port (i: in bit; o : out bit);
   end component;

   signal s1: bit;
begin
   g1: nand2 port map (a => a, b => b, o => s1);
   g2: not1 port map (i => s1, o => o);
end netlist;
```

Aperçu : RTL

```
entity and2 is
  port (a, b: in bit;
        o : out bit);
end and2;

architecture rtl of and2 is
begin
   o <= not (a and b);
end rtl;


entity full_adder is
  port (a, b, ci: in bit;
        o, co: out bit);
end entity;

architecture rtl of full_adder is
begin
   co <= (a and b) or (b and ci) or (a and ci);
   o <= (a xor b) xor ci;
end rtl;
```

Aperçu : synthèse

```
entity detect2 is
    port (i : in bit;
          clk, rst: in bit;
          detect : out bit);
end detect2;

architecture fsm of detect2 is
  type state_type is (Ini, d1, d2);
  signal state, n_state : state_type;
begin
  detect <= '1' when state = d2
       else '0';

  process (clk, rst)
  begin
    if rst = '1' then
      state <= Ini;
    elsif clk = '1' and clk'event then
      state <= n_state;
    end if;
  end process;

  process (i, state)
  begin
    if i = '0' then
      n_state <= Ini;
    else -- i = '1'
      case state is
        when Ini =>
          n_state <= d1;
        when d1 =>
          n_state <= d2;
        when d2 =>
          n_state <= d2;
      end case;
    end if;
  end process;
end fsm;
```

Aperçu : comportemental

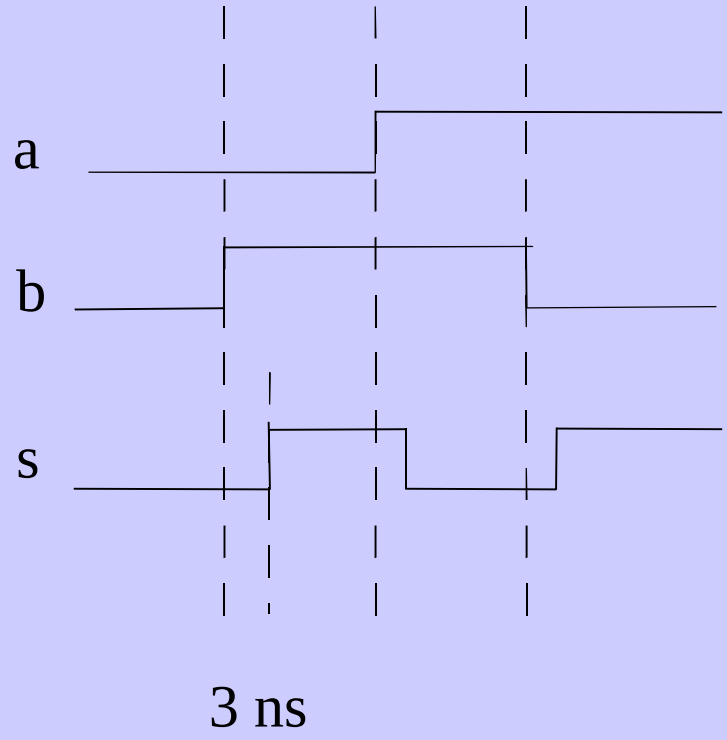
```
entity pgcd is
  port (a, b: natural;
        res : out natural);
end pgcd;

architecture hll of pgcd is
   function pgcd_func (a, b: natural)
     return natural
   is
   begin
     if b > a then
        return pgcd_func (b, a);
     elsif b = 0 then
        return a;
     else
        return pgcd_func (b, a mod b);
     end if;
   end pgcd_func;
begin
    res <= pgcd_func (a, b);
end hll;
```

Simulation (comportementale)

```
entity gate_tb is
end gate_tb;

architecture tb of gate_tb is
  component gate is
    port (a, b : in bit;
          o : out bit);
  end component;

  signal a, b, s: bit;
begin
  my_gate: gate port map (a, b, s);

  process
  begin
    a <= '1'; b <= '1';
    wait for 1 ns;
    assert s = '1';

    a <= '0'; b <= '1';
    wait for 1 ns;
    assert s = '0';

    a <= '1'; b <= '0';
    wait for 1 ns;
    assert s = '0';

    a <= '0'; b <= '0';
    wait for 1 ns;
    assert s = '1';

    wait;
  end process;
end tb;
```

Simulation (temporelle)

```
entity my_xor is
  port (a, b: in bit;
        o : out bit);
end my_xor;

architecture timing of my_xor is
begin
   o <= a xor b after 3 ns;
end timing;
```



Les acteurs

- Vendeurs de CAO : Mentor, Cadence, Synopsys...
- Vendeurs de FPGA : Xilinx, Altera, Actel, Cypress...
- En open-source :
  - Icarus Verilog : simulateur et synthétiseur verilog.
  - Savant (Université de Cincinnati) : outil de recherche qui a eu plusieurs contrats DARPA. L'analyseur génère du code C++, le simulateur est générique.
  - Alliance (LIP6, Jussieu) : chaîne de CAO universitaire. Le simulateur VHDL ne gère qu'un sous-ensemble très restreint (pas de process).
  - ...

GHDL, vue générale

- Compilateur au sens classique du terme.
- Plusieurs phases :
  - Scan : reconnaissance des tokens.
  - Parse : construction de l'arbre syntaxique.
  - Sem : analyse sémantique.
  - Canon : transformations de certaines constructions.
  - Translate : traduction en arbre bas niveau.

GHDL, vue générale (2)

- Briques supplémentaires :
  - Libraries : gestion des bibliothèques VHDL (regroupement d'unités).
  - Name_table : gestion de la table des noms, avec les particularités de VHDL :
    - Portées (scopes).
    - Surchage.
    - Visibilité potentielle.

GHDL, la traduction

- Translate : traduction en arbre de bas niveau (Ortho).
  - Simplification des types (ex : linéarisation des tableaux).
  - Création des instances et des sous-programmes associés.
  - Génération des appels aux routines de la run-time pour les traitements complexes (création des signaux, enregistrement des process, affectation des signaux, fichiers...).
  - Codage (mangling) des noms.
  - Génération des structures pour la hiérarchie.

L'interface Ortho

- API abstraite définissant un arbre.
- Uniquement des appels de fonctions.
- Contraintes sévères
  - Pas de cycles.
  - Pas de possibilité de relire le résultat.
- Types de noeuds:
  - Types : booléen, signés, non signés, énumeration, tableau, structure, pointeur.
  - Noms (lvalue) : paramètres, variables, constantes...
  - Expressions.
  - Sous-programmes.
  - Instructions.

Interfaçage avec GCC

- GHDL est vu comme un front-end, à l'image de C, C++, ObjC, Java, Ada95…
- Il existe une implémentation de Ortho pour GCC.
- Binding en Ada95 au niveau procédural et au niveau types simples (enum, pointeur).
- Fonctions C pour « exporter » les macros de GCC.
- Programme C pour générer des morceaux de code Ada à partir de certains fichiers de GCC.
- Mise à jour du binding au fur et à mesure des versions de GCC.
- Driver indépendant de GCC : ghdldrv.

GRT, GHDL run time

- Échéancier à temps discret :
  - Acteurs : les process (un sous-programme et une pile par process).
  - Événements : portés par les signaux.
- Gestion des fichiers.
- Génération d’une trace (VCD).
- Gestion des erreurs.
- Routines auxiliaires (allocation mémoire, copie, exp).
- Ligne de commande, affichage de l'aide...

Traduction des entity/architecture

```
entity my_gate is
   port (a, b : in bit;
         o : out bit);
end my_gate;
```

```
struct my_gate_struct
{
    bit_signal_ptr a;
    bit_signal_ptr b;
    bit_signal_ptr o;
};
```

```
architecture test of my_gate is
   signal s : bit;
   …
end test;
```

```
struct my_gate__test_struct
{
   struct my_gate_struct ENTITY;
   bit_signal_ptr s;
   …
};
```

Traduction des process

```
process (a, b)
begin
   s <= a or b;
end process;
```

```
void my_gate__test_P0
   (struct my_gate__test_struct *INSTANCE)
{
   bit t;
   t = *(INSTANCE->a) | *(INSTANCE->b);
   __grt_assign_signal_B2 (INSTANCE->s, t, 0);
}
```

Construction des process

```
void my_gate__test__INIT (struct my_gate__test_struct *INSTANCE)
{
  INSTANCE->s = __ghdl_create_signal_B2 (&bit_type_desc);
  …
  __ghdl_create_sensitized_process
    (&my_gate__test_P0, INSTANCE);
  __ghdl_process_add_sensitivity (INSTANCE->a);
  __ghdl_process_add_sensitivity (INSTANCE->b);
  __ghdl_process_add_driver (INSTANCE->s);

  …
}
```

Traduction des tableaux (1/4)

- En VHDL :
  - Les tableaux ont des limites.
  - Ils peuvent être multidimensionnels.
  - Les indexes ont une direction.
- Exemples :
  - `type bit_vector is array (natural range <>) of bit;`
  - `subtype byte is bit_vector (7 downto 0);`
  - `type memory is array (0 to 1023, 0 to 31) of bit;`

Traduction des tableaux (2/4)

Les limites sont stockées en mémoire :

```
type integer is range
  -2147483648 to 2147483647;
```

```
struct integer_BOUNDS_TYPE
{
   integer left;
   integer right;
   bool dir;
   unsigned int length;
};

struct integer_BOUNDS_TYPE
  integer_RANGE =
{
  -2147483648,
   2147483647,
  dir_to,
  4294967296
};
```

Traduction des tableaux (3/4)

Le contenu est linéarisé :

```
type bit_vector is
  array (natural range <>) of bit;
```

→

```
struct bit_vector__BOUNDS
{
  struct integer__BOUNDS dim1;
};

struct bit_vector__ARRAY
{
   struct bit_vector__BOUNDS *bounds;
   bit *VAL;
};
```

```
variable v : bit_vector (7 downto 0);
```

→

```
struct bit_vector__BOUNDS
  v_BOUNDS =
{
  { 7, 0, dir_downto, 8 }
};
..
  bit v[8];
```

Traduction des tableaux (4/4)

Accès à un élément :

```
type arr_type is
   array (0 to 15, 7 downto 0) of bit;
variable v : arr_type;
..
 .. := v (i, j);
```

```
struct arr_type__BOUNDS v__B =
{
  {0, 15, dir_to, 16},
  {7, 0, dir_downto, 8}
};
bit v[128];
..
if (v__B.dim1.dir = dir_to)
  index1 = i - v__B.dim1.left;
else
  index1 = v__B.dim1.left - i;
if (index1 < 0 || index1 >= v__B.dim1.length)
  error ();
.. // Idem pour index2
.. := v[index1 * v__B.dim1.length + index2];
```

Types non statiques

```
--  Déclaration d'une fonction.
function get_word_size return natural;

--  Déclaration d'une constante (qui n'est pas localement statique).
constant word_size : natural := get_word_size;

--  Attention, les limites du type word_type ne sont pas connues lors
--  de la compilation.
subtype word_type is bit_vector (word_size - 1 downto 0);

--  La taille de data_bus non plus.
signal data_bus: word_type;

--  La taille de la structure bus_rec non plus…
type bus_rec is record
begin
   data : word_type;
   r_w : bit;
end record;
```

Retour de types composites

```
function bin_image (v : natural)
   return string
is
   variable res : string (1 to 31);
   variable t : natural := v;
   variable index : natural := 31;
begin
   loop
      if t mod 2 = 1 then
         res (index) := '1';
      else
         res (index) := '0';
      end if;
      t := t / 2;
      index := index - 1;
     exit when t = 0;
   end loop;
   return res (index to res'right);
end bin_image;
```

Conclusion : futur…

- Implémentation complète de la norme 87, 93, 98 puis 2000.
- Optimisation du run-time.
- Optimisation du code généré.
- Portage sur d'autres plateformes.
- Implémentation de normes voisines (VHPI, AMS).
- Vers la synthèse.