# Introduction au logiciel Tetramax (`tmax`, mode graphique)

**Mounir Benabdenbi**

Le Laboratoire CIME dispose de x licences du logiciel Tetramax de Synopsys, destiné au test des circuits intégrés. Ce logiciel peut fonctionner en mode graphique (**`tmax`**) ou en mode texte (**`tmax -shell fichier.cmd`**) Pour des raisons de commodité, **vous utiliserez le mode texte pour vos expérimentations si le nombre de licences est insuffisant**!!!. Rester en mode graphique monopolise une licence, et empêche les autres de travailler.

En utilisation « simple », le logiciel fonctionne de manière très intuitive. Nous détaillons ici les deux modes de fonctionnement principaux : le mode **ATPG** et le mode « **simulation de fautes** ». L'utilisation de ces deux modes repose sur une phase commune de chargement, de contrainte et de vérification de la netlist. Nous commencerons donc par étudier cette phase commune, puis les deux alternatives.

**Phase Commune**

**Etape 1 :** Allez dans le répertoire **`~`** `(racine de votre compte)`, et exécutez la commande Unix suivante :
**`source .cshrc_TP`** pour définir les variables d'environnement et accéder aux logiciels Synopsys.

**Etape 2 :** Allez dans votre répertoire de travail. Nous supposerons par la suite que le répertoire de travail est
**`~/TP-Test/TP1/`**

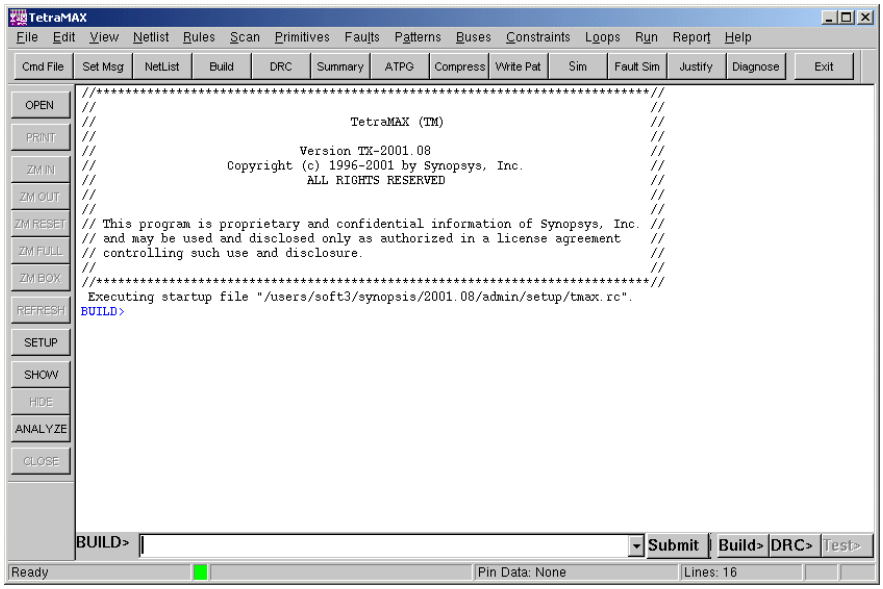
**Etape 3 :** Examinez le fichier **`mux.vhd`** suivant dont on cherche à calculer les vecteurs de test avec l'ATPG de **Tetramax**.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY mux IS
PORT(
  a     : IN BIT;
  b     : IN BIT;
  com   : IN BIT;
  s     : OUT BIT;
  vdd   : IN BIT;
  vss   : IN BIT
);
END mux;
```

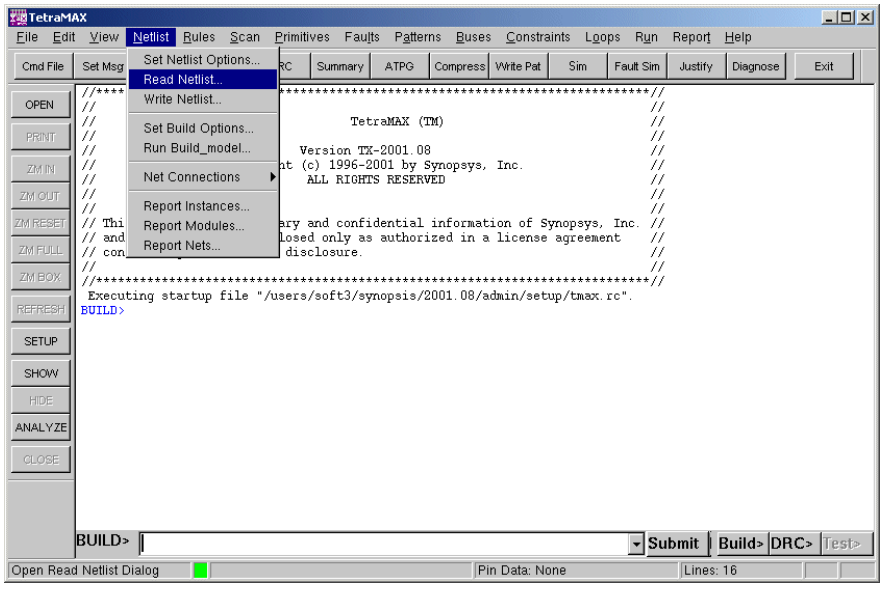
```
ARCHITECTURE RTL OF mux IS
  SIGNAL out_and2       : BIT;
  SIGNAL out_and1       : BIT;
  SIGNAL com_not        : BIT;

  COMPONENT o2_x2
  PORT(
  i0    : IN BIT;
  i1    : IN BIT;
  q     : OUT BIT;
  vdd   : IN BIT;
  vss   : IN BIT
   );
  END COMPONENT;

  COMPONENT inv_x1
  PORT(
  i     : IN BIT;
  nq    : OUT BIT;
  vdd   : IN BIT;
  vss   : IN BIT
   );
  END COMPONENT;

  COMPONENT a2_x2
  PORT(
  i0    : IN BIT;
  i1    : IN BIT;
  q     : OUT BIT;
  vdd   : IN BIT;
  vss   : IN BIT
   );
  END COMPONENT;

BEGIN
  or1 : o2_x2
  PORT MAP (
    i0 => out_and1,
    i1 => out_and2,
    q => s,
    vdd => vdd,
    vss => vss
  );
  inv : inv_x1
  PORT MAP (
    i => com,
    nq => com_not,
    vdd => vdd,
    vss => vss
  );
  and2 : a2_x2
  PORT MAP (
    i0 => com_not,
    i1 => b,
    q => out_and2,
    vdd => vdd,
    vss => vss
  );
  and1 : a2_x2
  PORT MAP (
    i0 => com,
    i1 => a,
    q => out_and1,
    vdd => vdd,
    vss => vss
  );
END RTL;
```

Ce fichier comprend trois entrées logiques (**`a, b et cmd`** ), deux alimentations (**`vdd`** et **`vss`**) et une sortie (**`s`**). Les vecteurs que doit calculer l'ATPG sont donc assez simples.

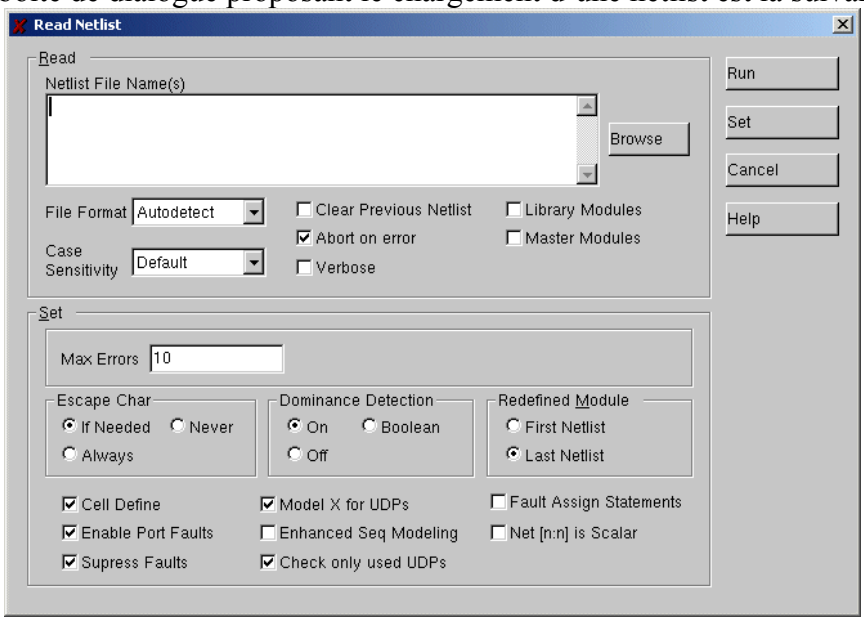
**Etape 4 :** Exécutez la commande **`tmax`**. L'écran suivant apparaît :




**Etape 5 :** Pour commencer, il faut charger la bibliothèque des portes de base, pour que chaque bloc fonctionnel puisse être reconnue correctement. Pour cela, cliquez sur le bouton « **`Netlist`** » ou choisissez l'option « **`Netlist/Read netlist…`** » depuis le menu principal :

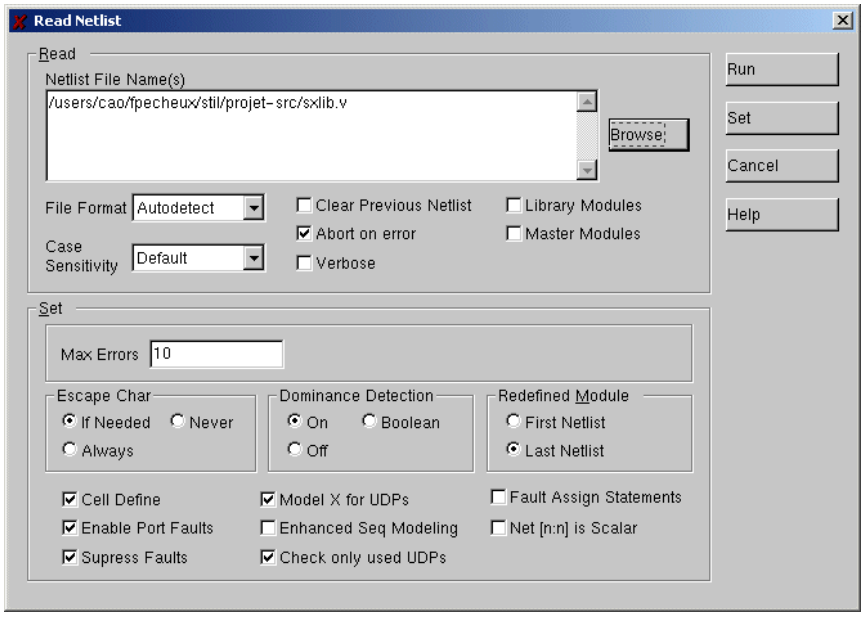


La boîte de dialogue proposant le chargement d'une netlist est la suivante :

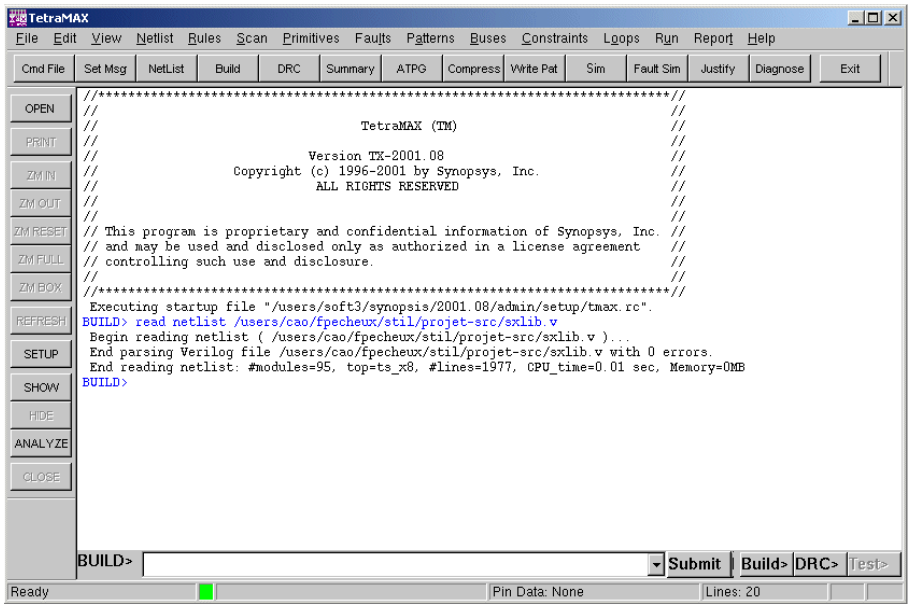



**Etape 6 :** Pour procéder au chargement du fichier comprenant le comportement des portes logiques de base **`sxlib.v`** (au format Verilog), cliquez sur « **`Browse`** » et choisissez le fichier **`sxlib.v`** dans l'arborescence.

La boîte de dialogue devient alors :

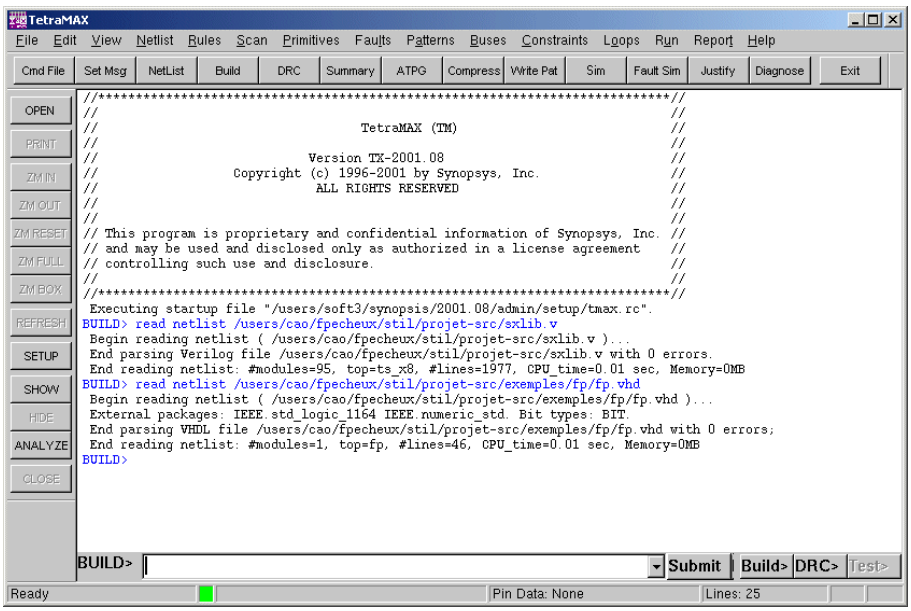


**Etape 7 :** Une fois cliqué sur le bouton « **Run** » de la boîte de dialogue, **tmax** affiche l'écran suivant :



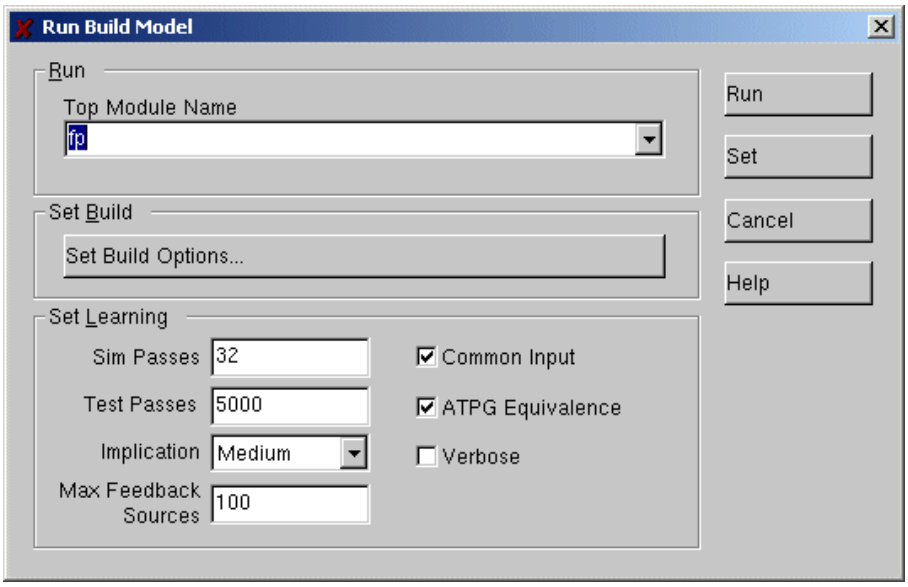

**tmax** est donc dans le mode **BUILD**, comme indiqué par le prompt **BUILD>**, et le chargement de la bibliothèque des cellules de base s'est correctement déroulé. Notez que la commande de l'interface graphique que vous venez d'effectuer se traduit du point de vue du logiciel **tmax** par l'exécution de la commande affichée en bleu. **C'est cette même commande que vous devrez ajouter à votre fichier de commande en mode texte !!!**

**Etape 8 :** Il faut ensuite charger le fichier **mux.vhd**, contenant la netlist du composant dont on cherche à déterminer les vecteurs. La lecture de la netlist se fait de la même manière « **Netlist/Read Netlist…** », etc. Vous devriez alors obtenir l'écran **tmax** suivant :

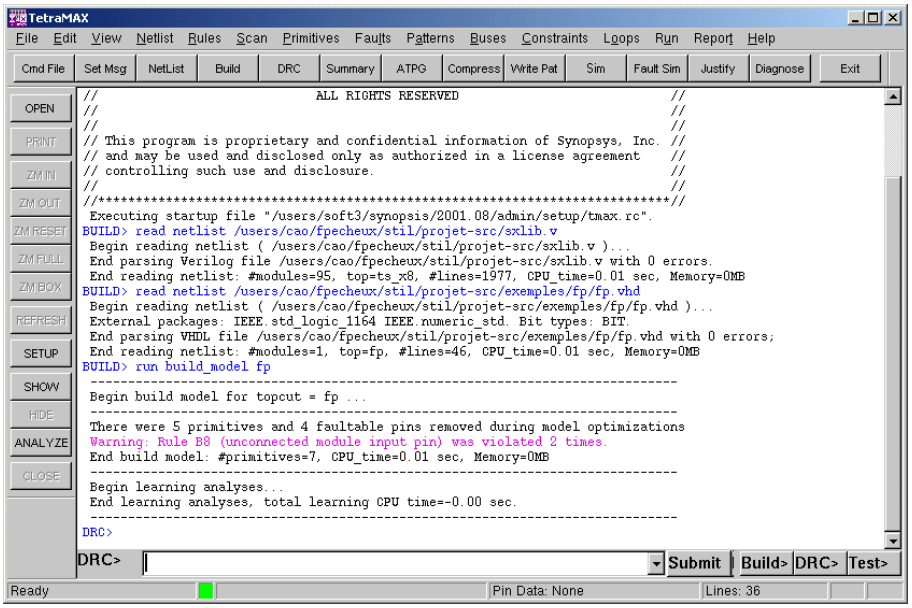



indiquant que la netlist a été correctement lue

**Etape 9 :** Il faut ensuite « construire » le projet à l'aide de la commande **`build`**. Cette opération est une sorte d'élaboration (au sens VHDL) pour le test, qui permet de vérifier que tous les composants nécessaires sont effectivement présents en mémoire. Il faut en pratique donner à **`tmax`** le nom du composant sur lequel on souhaite opérer. Tetramax étant plutôt « user-friendly », si l'on clique sur le bouton « **`Build`** », on peut constater que **`tmax`** a déjà rempli ce champ avec **`mux`**, le nom du modèle :

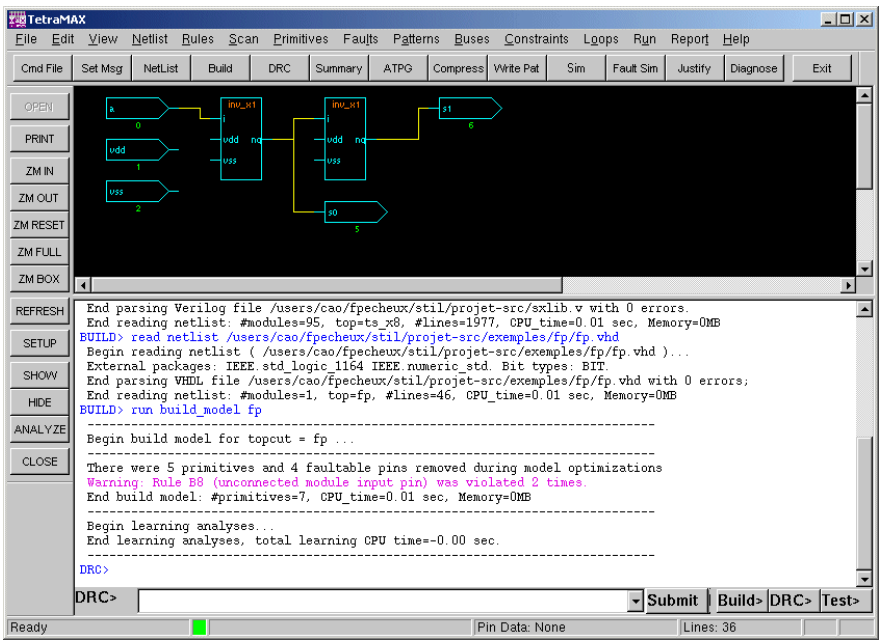



**Etape 10 :** Une fois cliqué sur « **`Run`** », **`tmax`** affiche l'écran suivant :



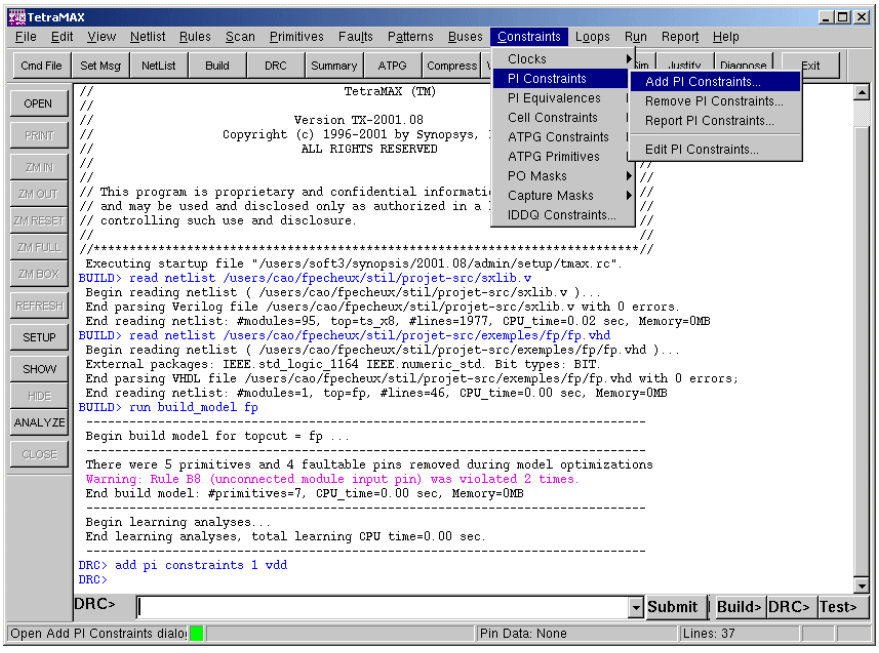

Le message affiché en violet par ce dernier écran montre que durant la phase de **`Build`**, **`tmax`** a déterminé que les entrées d'alimentation **`vdd`** et **`vss`** étaient non connectées. Le message est un avertissement et non une erreur.

Si vous le souhaitez, vous pouvez visualiser le circuit **mux** tel qu'il a été analysé par Tetramax en cliquant sur le bouton «**Open GSV** » puis sur « show all » situé sur la partie haute de la fenêtre. Vous obtenez alors le schéma suivant :

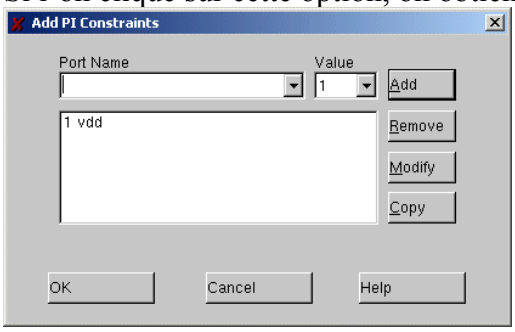



Le schéma confirme bien toutes les étapes que nous avons vues jusqu'à présent. Notez que le logiciel **tmax** est passé tout seul de la phase **BUILD** à la phase **DRC**. Il s'attend donc à ce que l'on lance la phase de vérification du modèle.

**Etape 11 :** Avant de procéder à cette vérification, et pour régler le problème des alimentations, il faut « contraindre » **tmax** en imposant des valeurs particulières sur les entrées que sont **vdd** et **vss**. Cette opération se fait avec l'option « **Constraints/PI Constraints/Add PI Constraints…** » du menu principal de **tmax** :

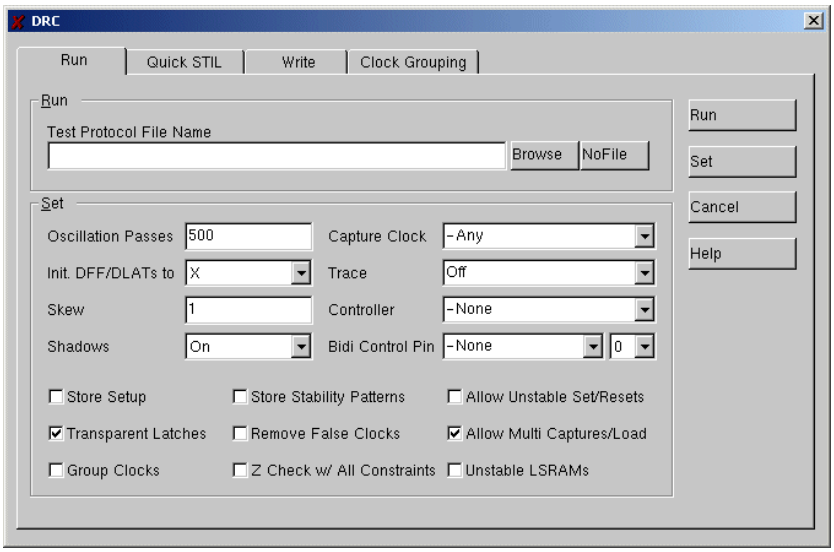


Si l'on clique sur cette option, on obtient la boîte de dialogue suivante :

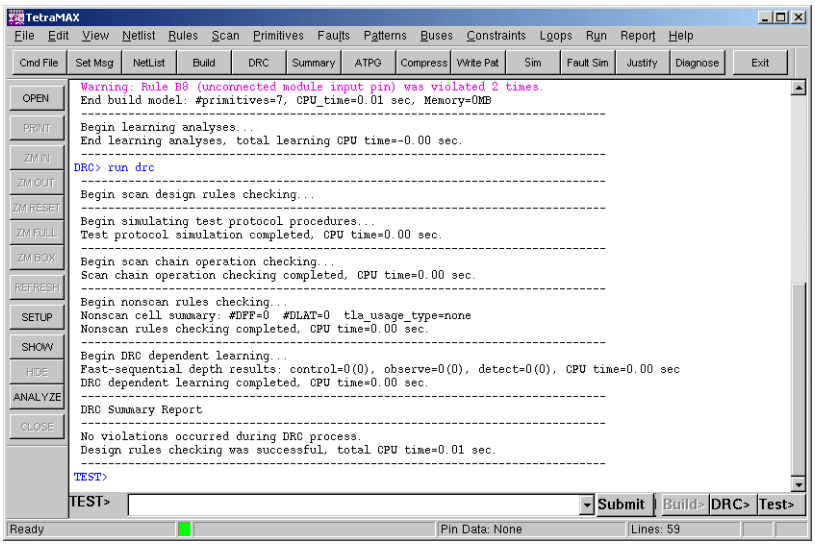



**Etape 12 :** Il faut alors choisir le **Port name** d'entrée que l'on souhaite contraindre (ici **`vdd`** et **`vss`**), choisir la valeur de contrainte (Champ **`Value`**) et cliquer sur **`Add`**. Il faut répéter cette opération pour toutes les contraintes.

**Etape 13 :** La prochaine étape est le DRC, qui vérifie qu'aucune règle de conception n'est violée par ce modèle. Il suffit de cliquer sur le bouton « **`DRC`** » pour obtenir la boîte de dialogue du DRC:



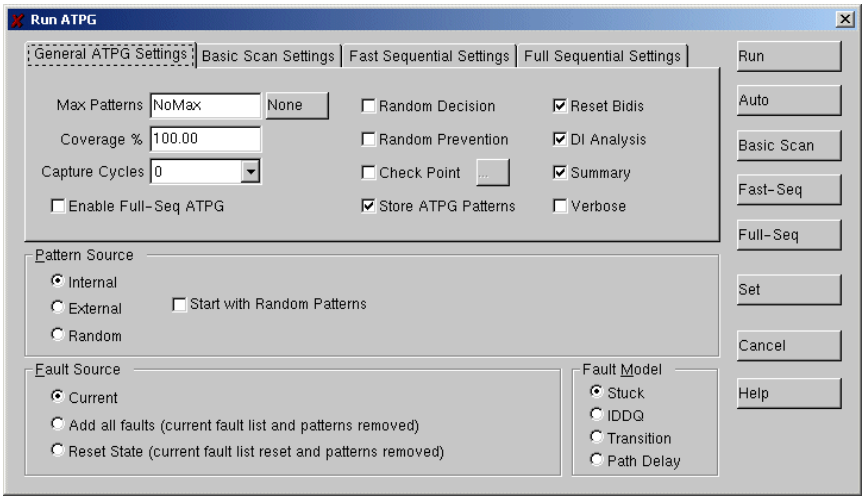

On se contente ici de cliquer sur **`Run`**, ce qui permet d'obtenir l'écran **`tmax`** suivant :



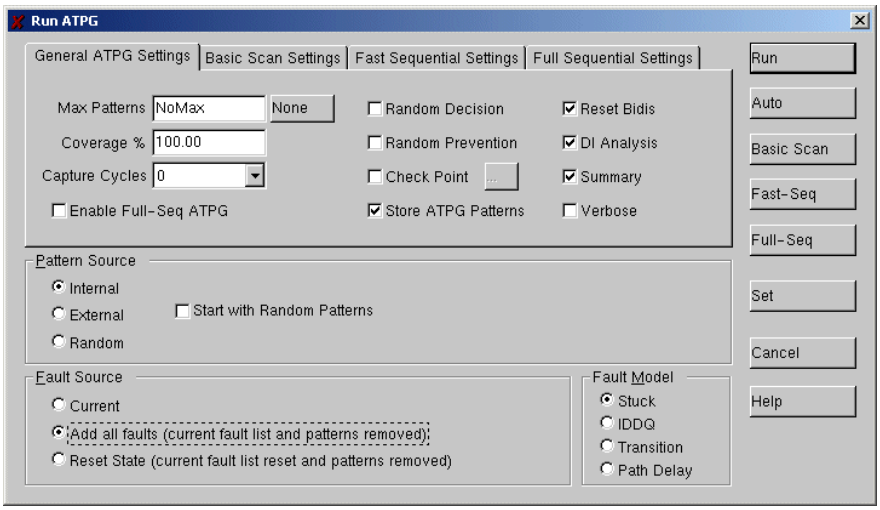

**`tmax`** nous indique qu'il n'a détecté aucune violation de règle durant cette phase. Le logiciel passe alors en mode **`TEST`**. C'est ici que l'on choisit le mode de fonctionnement « ATPG » ou « Simulation de fautes ».

## Mode ATPG

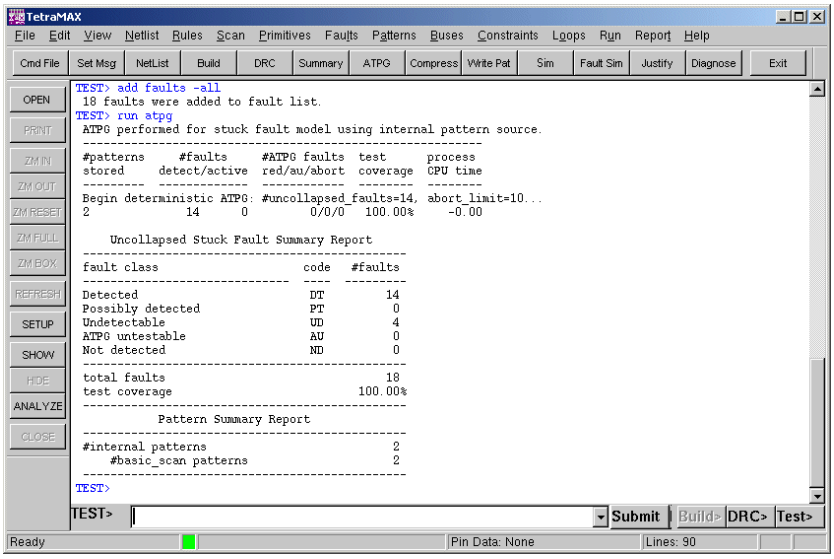
**Etape 14 :** Pour cela, cliquez sur le bouton « **`ATPG`** » pour obtenir la boîte de dialogue suivante :



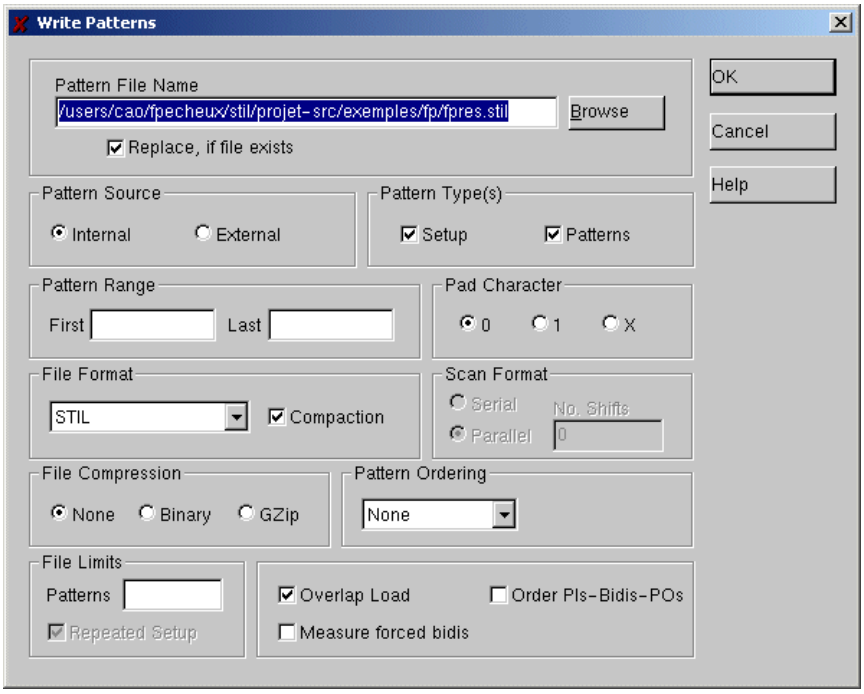

Cochez la « **`fault source`** » en ajoutant toutes les fautes…




Puis « **Run** ». Vous obtenez alors l'écran **tmax** suivant, indiquant que l'ATPG a correctement généré les vecteurs.



**Etape 15 :** Il reste alors à sauvegarder les patterns, ce qui se fait en cliquant sur « **`Write Pat`** » :




Il faut spécifier le nom du fichier de sortie, et le format (ici **`STIL`**). N'oubliez pas de cocher « **`Replace, if file exists`** » pour générer un fichier dans tous les cas.

**Etape 16 :** Vous pouvez maintenant sortir de **`tmax`**, pour visualiser le fichier de patterns au format **`STIL`** suivant :

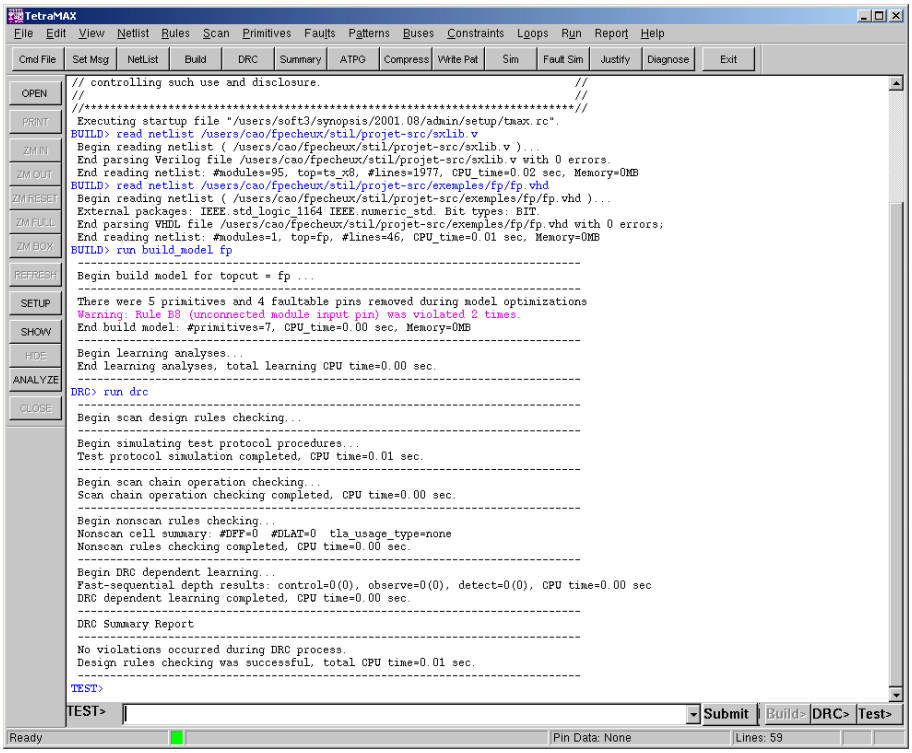
```
STIL 1.0 {
   Design P2000.9;
}
Header {
   Title "  TetraMAX (TM)  2001.08-i010809_170343 STIL output";
   Date "Mon Feb  3 10:00:29 2003";
   History {
      Ann {*     Uncollapsed Stuck Fault Summary Report *}
      Ann {* ----------------------------------------------- *}
      Ann {* fault class                     code   #faults *}
      Ann {* ------------------------------  ----  --------- *}
      Ann {* Detected                         DT         14 *}
      Ann {* Possibly detected                PT          0 *}
      Ann {* Undetectable                     UD          4 *}
      Ann {* ATPG untestable                  AU          0 *}
      Ann {* Not detected                     ND          0 *}
      Ann {* ----------------------------------------------- *}
      Ann {* total faults                                18 *}
      Ann {* test coverage                            100.00% *}
      Ann {* ----------------------------------------------- *}
      Ann {*  *}
      Ann {*            Pattern Summary Report *}
      Ann {* ----------------------------------------------- *}
      Ann {* #internal patterns                          2 *}
      Ann {*     #basic_scan patterns                    2 *}
      Ann {* ----------------------------------------------- *}
      Ann {*  *}
      Ann {* rule  severity  #fails  description *}
      Ann {* ----  --------  ------  ---------------------------------- *}
      Ann {* B8    warning        2  unconnected module input pin *}
      Ann {*  *}
      Ann {* There are no clocks *}
```

```
      Ann {* port_name         constraint_value *}
      Ann {* ----------------  ---------------- *}
      Ann {* vdd                 1 *}
      Ann {* vss                 0 *}
      Ann {*  *}
      Ann {* There are no equivalent pins *}
      Ann {* There are no net connections *}
   }
}
Signals {
   "a" In; "vdd" In; "vss" In; "s0" Out; "s1" Out;
}
SignalGroups {
   "_default_In_Timing_" = '"a" + "vdd" + "vss"'; // #signals=3
   "_pi" = '"a" + "vdd" + "vss"'; // #signals=3
   "_default_Out_Timing_" = '"s0" + "s1"'; // #signals=2
   "_po" = '"s0" + "s1"'; // #signals=2
}
ScanStructures {
   // Uncomment and modify the following to suit your design
   // ScanChain "chain_name" { ScanIn "chain_input_name"; ScanOut "chain_output_name"; }
}
Timing {
   WaveformTable "_default_WFT_" {
      Period '100ns';
      Waveforms {
         "_default_In_Timing_" { 0 { '0ns' D; } }
         "_default_In_Timing_" { 1 { '0ns' U; } }
         "_default_In_Timing_" { Z { '0ns' Z; } }
         "_default_In_Timing_" { N { '0ns' N; } }
         "_default_Out_Timing_" { X { '0ns' X; } }
         "_default_Out_Timing_" { H { '0ns' X; '40ns' H; } }
         "_default_Out_Timing_" { T { '0ns' X; '40ns' T; } }
         "_default_Out_Timing_" { L { '0ns' X; '40ns' L; } }
      }
   }
}
PatternBurst "_burst_" { PatList {
   "_pattern_" {
   }
}}
PatternExec {
   PatternBurst "_burst_";
}
Procedures {
   "capture" {
      W "_default_WFT_";
      F { "vdd"=1; "vss"=0; }
      "forcePI": V { "_pi"=###; "_po"=\j XX; }
      "measurePO": V { "_po"=##; }
   }
   // Uncomment and modify the following to suit your design
   // load_unload {
      // V { } // force clocks off and scan enable pins active
      // Shift { V { _si=#; _so=#; }} // pulse shift clocks
   // }
}
MacroDefs {
   "test_setup" {
      W "_default_WFT_";
      V { "vdd"=1; "vss"=0; }
   }
}
Pattern "_pattern_" {
   W "_default_WFT_";
   "precondition all Signals": C { "_pi"=000; "_po"=\j XX; }
   Macro "test_setup";
   "pattern 0": Call "capture" {
      "_pi"=110; "_po"=LH; }
   "pattern 1": Call "capture" {
      "_pi"=010; "_po"=HL; }
}
```

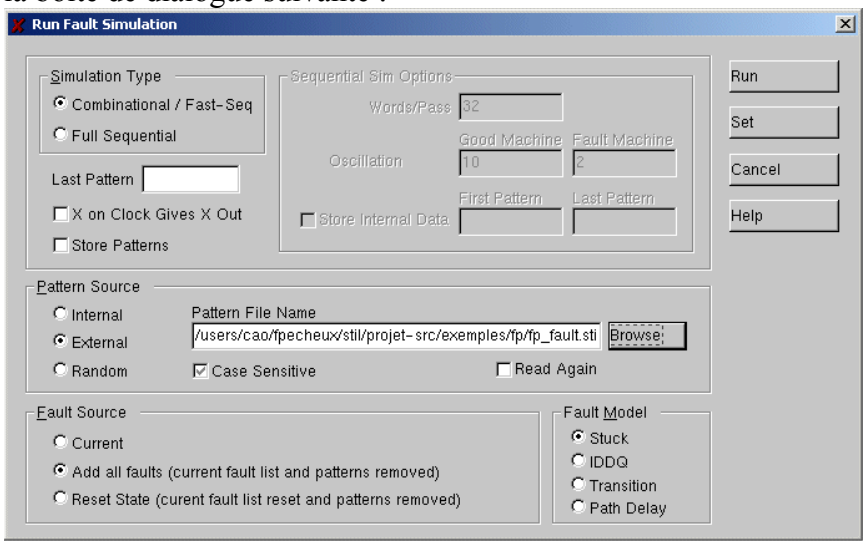
La fin de ce fichier montre que l'ATPG a effectivement calculé les vecteurs de test précédemment décrits.

## Mode Simulation de fautes

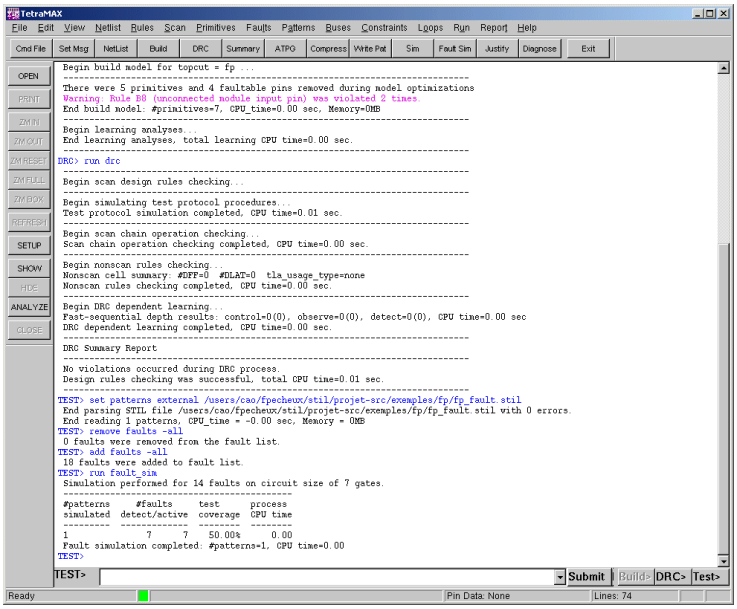
En mode simulation de fautes, il faut effecteur la phase commune, dont un résumé est indiqué ci –dessous :




**Etape 14 (Fault Sim):** Mais il faut choisir cette fois le bouton « **`Fault sim`** », pour obtenir la boîte de dialogue suivante :




On choisit alors la source de patterns externe que l'on souhaite valider, et on n'oublie pas de cocher « **`Add all faults`** » pour indiquer au logiciel de tester toutes les fautes. Le fichier **`fp_fault.stil`** utilisé ici est une recopie pure et simple du fichier généré par l'ATPG, mais dont on a supprimé le dernier vecteur de test (il n'en reste donc plus qu'un seul). L'exécution du simulateur de fautes doit donc donner un taux de couverture de 50%, à comparer aux 100% du mode ATPG.




CQFD !!!

## Tetramax en mode texte (`tmax_shell_exec`)

En mode texte, Tetramax s'utilise avec un fichier de commande. Un exemple de fichier de commande est le suivant :

```
# tmax.cmd
read netlist /users/enseig/trncomun/tutorial_tetramax/sxlib.v
read netlist /users/enseig/trncomun/tutorial_tetramax/fp.vhd
run build_model fp
add pi constraints 1 vdd
add pi constraints 0 vss
run drc
add faults -all
run atpg
write patterns fpres.stil -internal -format stil -replace
# set patterns external /users/enseig/trncomun/
tutorial_tetramax/fp_fault.stil
# remove faults -all
# run fault_sim
```

Ce fichier sert de script à l'interpréteur de commande de Tetramax, nommé **`tmax_shell_exec`**. Pour exécuter plus vite vos essais sans monopoliser les licences, vous devrez donc entrer :

```
tmax_shell_exec < tmax.cmd >& tmax.res
```

où **`tmax.cmd`** est votre fichier de commande et **`tmax.res`** le fichier de log généré par Tetramax.