Integration continue et vérification de code VHDL en Python.

Cayetano Santos - L2it / In2p3

13e Journées Informatiques IN2P3/IRFU

15–17 nov. 2021

Outline

Problématiques liés à la conception numérique

C'est quoi un FPGA ?

Matrice logique pré cablé.

Sur mesure On décrit un comportement logique adapté aux besoins

Re-programmable Modifiable à souhait

Performant Pipelining et parallelisme avec des cycles de quelques ns.

Bas niveau Connexion d'éléments logiques (registres, LUTS, DSP, etc.)

Parallelisme Véritable matrice avec des centaines de pins en I/O

Example Out (10 bits) <= [[*A* (16b) + B (6b)] xor C (4b)] X D (10b)

C'est quoi un FPGA ?

Matrice logique pré cablé.

Sur mesure On décrit un comportement logique adapté aux besoins

Re-programmable Modifiable à souhait

Performant Pipelining et parallelisme avec des cycles de quelques ns.

Bas niveau Connexion d'éléments logiques (registres, LUTS, DSP, etc.)

Parallelisme Véritable matrice avec des centaines de pins en I/O

Example Out (10 bits) <= [[*A* (16b) + B (6b)] xor C (4b)] X D (10b)

C'est quoi un FPGA ?

Matrice logique pré cablé.

Sur mesure On décrit un comportement logique adapté aux besoins

Re-programmable Modifiable à souhait

Performant Pipelining et parallelisme avec des cycles de quelques ns.

Bas niveau Connexion d'éléments logiques (registres, LUTS, DSP, etc.)

Parallelisme Véritable matrice avec des centaines de pins en I/O

Example Out (10 bits) <= [[*A* (16b) + B (6b)] xor C (4b)] X D (10b)

C'est quoi un FPGA ?

Matrice logique pré cablé.

Sur mesure On décrit un comportement logique adapté aux besoins

Re-programmable Modifiable à souhait

Performant Pipelining et parallelisme avec des cycles de quelques ns.

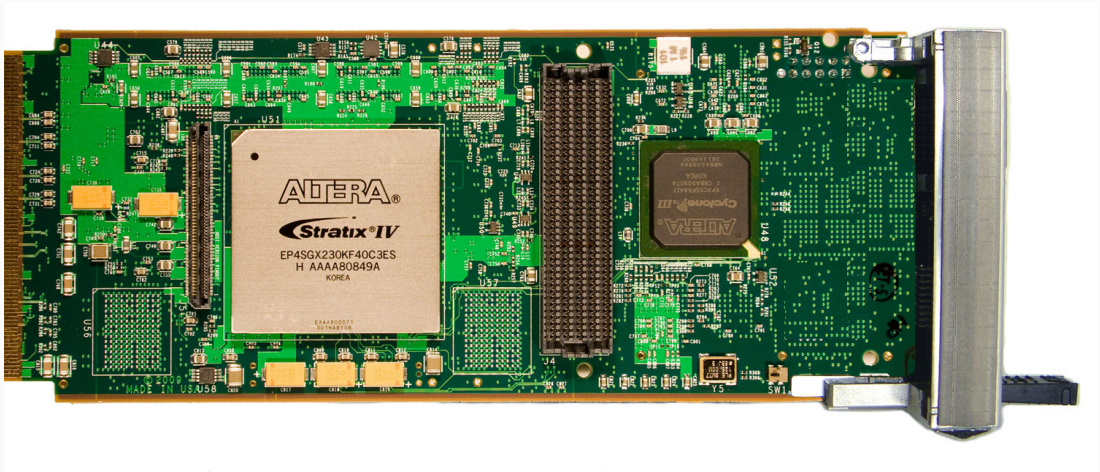
Bas niveau Connexion d'éléments logiques (registres, LUTS, DSP, etc.)

Parallelisme Véritable matrice avec des centaines de pins en I/O

Example Out (10 bits) <= [[*A* (16b) + B (6b)] xor C (4b)] X D (10b)

C'est quoi un FPGA ?

Matrice logique pré cablé.



Sur mesure On décrit un comportement logique adapté aux besoins

Re-programmable Modifiable à souhait

Performant Pipelining et parallelisme avec des cycles de quelques ns.

Bas niveau Connexion d'éléments logiques (registres, LUTS, DSP, etc.)

Parallelisme Véritable matrice avec des centaines de pins en I/O

Example Out (10 bits) <= [[*A* (16b) + B (6b)] xor C (4b)] X D (10b)

C'est quoi un FPGA ?

Matrice logique pré cablé.

Sur mesure On décrit un comportement logique adapté aux besoins

Re-programmable Modifiable à souhait

Performant Pipelining et parallelisme avec des cycles de quelques ns.

Bas niveau Connexion d'éléments logiques (registres, LUTS, DSP, etc.)

Parallelisme Véritable matrice avec des centaines de pins en I/O

Example Out (10 bits) <= [[*A* (16b) + B (6b)] xor C (4b)] X D (10b)

VHDL : language matériel (pas de main !)

```
6   -- pragma translate_off
7   library osvvm;
8   use osvvm.AlertLogPkg.all;
9   -- pragma translate_on
10
11  entity mux_fifo is
12    generic (
13      g_enabled_channels : std_logic_vector := X"FF";
14      -- pragma translate_off
15      mux_fifo_id       : alertlogidtype   := GetAlertLogID("mux-fifo");
16      -- pragma translate_on
17      g_extend          : natural range 0 to 1
18      );
19    port (
20      rst       : in    std_logic;
21      clk       : in    std_logic;
22      -- fifo if
23      fifo_rd_if : inout t_fifo_rd_if_array;
24      -- out if
25      dataout    : out   std_logic_vector;
26      wr_en      : out   std_logic;
27      full       : in    std_logic
28      );
29  begin
30
31    -- pragma translate_off
32    process(clk) is
33    begin
34      if (clk'event and clk = '1') then
35        if (wr_en) then
36          Log(mux_fifo_id, "writting data " &
37              to_string(to_integer(unsigned(dataout(dataout'left downto 8)))) &
38              "." &
39              to_hstring(dataout(7 downto 0)), DEBUG);
40        end if;
41      end if;
42    end process;
43    -- pragma translate_on
44
45  end entity mux_fifo;
46
47  architecture simple of mux_fifo is
48
49    type t_state is (s_wait, s_capture, s_write);
50
51    signal fifo_rd_if_rd    : std_logic_vector(fifo_rd_if'range) := (others => '0');
52    signal fifo_rd_if_empty : std_logic_vector(fifo_rd_if'range) := (others => '0');
53    signal state            : t_state;
54
55  begin
```

Cadencé par événements (horloge hardware)

Concurrent / sequentiel

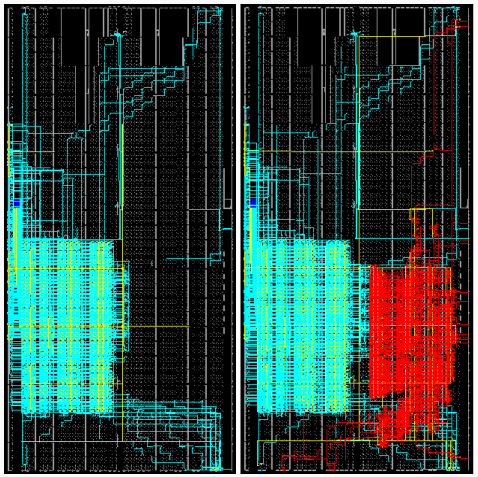
Fortement hierarchisé en blocks

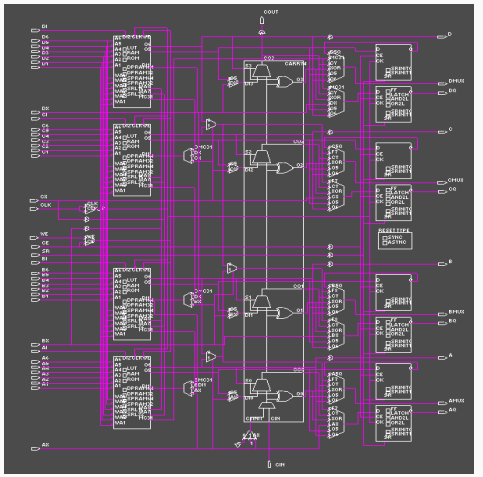
Adapté déscriptions bas niveau

Conversion du code en binaire (bitstream)

Synthèse logique RTL

Mapage, placement de logique et routage





Temps total excédant *des dizaines d'heures*

Téléchargement sur cible et exécution

Cycle de développement très lent. . .

VHDL : language matériel (pas de main !)

```
6   -- pragma translate_off
7   library osvvm;
8   use osvvm.AlertLogPkg.all;
9   -- pragma translate_on
10
11  entity mux_fifo is
12    generic (
13      g_enabled_channels : std_logic_vector := X"FF";
14      -- pragma translate_off
15      mux_fifo_id       : alertlogidtype   := GetAlertLogID("mux-fifo");
16      -- pragma translate_on
17      g_extend          : natural range 0 to 1
18      );
19    port (
20      rst       : in    std_logic;
21      clk       : in    std_logic;
22      -- fifo if
23      fifo_rd_if : inout t_fifo_rd_if_array;
24      -- out if
25      dataout    : out   std_logic_vector;
26      wr_en      : out   std_logic;
27      full       : in    std_logic
28      );
29  begin
30
31    -- pragma translate_off
32    process(clk) is
33    begin
34      if (clk'event and clk = '1') then
35        if (wr_en) then
36          Log(mux_fifo_id, "writting data " &
37              to_string(to_integer(unsigned(dataout(dataout'left downto 8)))) &
38              "." &
39              to_hstring(dataout(7 downto 0)), DEBUG);
40        end if;
41      end if;
42    end process;
43    -- pragma translate_on
44
45  end entity mux_fifo;
46
47  architecture simple of mux_fifo is
48
49    type t_state is (s_wait, s_capture, s_write);
50
51    signal fifo_rd_if_rd    : std_logic_vector(fifo_rd_if'range) := (others => '0');
52    signal fifo_rd_if_empty : std_logic_vector(fifo_rd_if'range) := (others => '0');
53    signal state            : t_state;
54
55  begin
```

Cadencé par événements (horloge hardware)

Concurrent / sequentiel

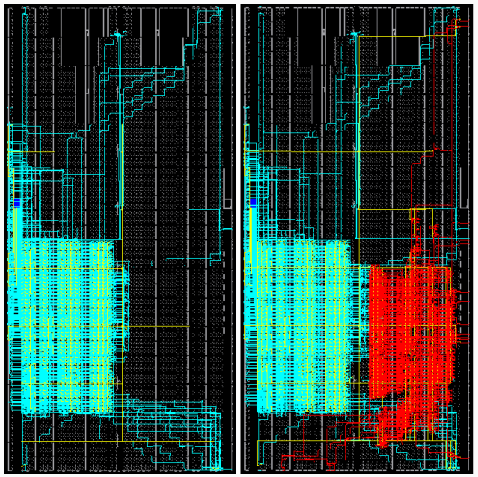
Fortement hierarchisé en blocks

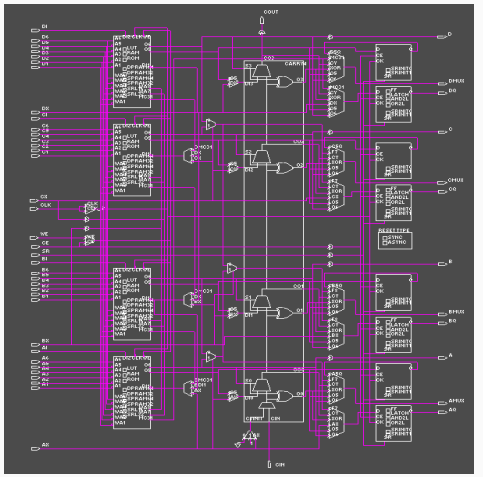
Adapté déscriptions bas niveau

Conversion du code en binaire (bitstream)

Synthèse logique RTL

Mapage, placement de logique et routage



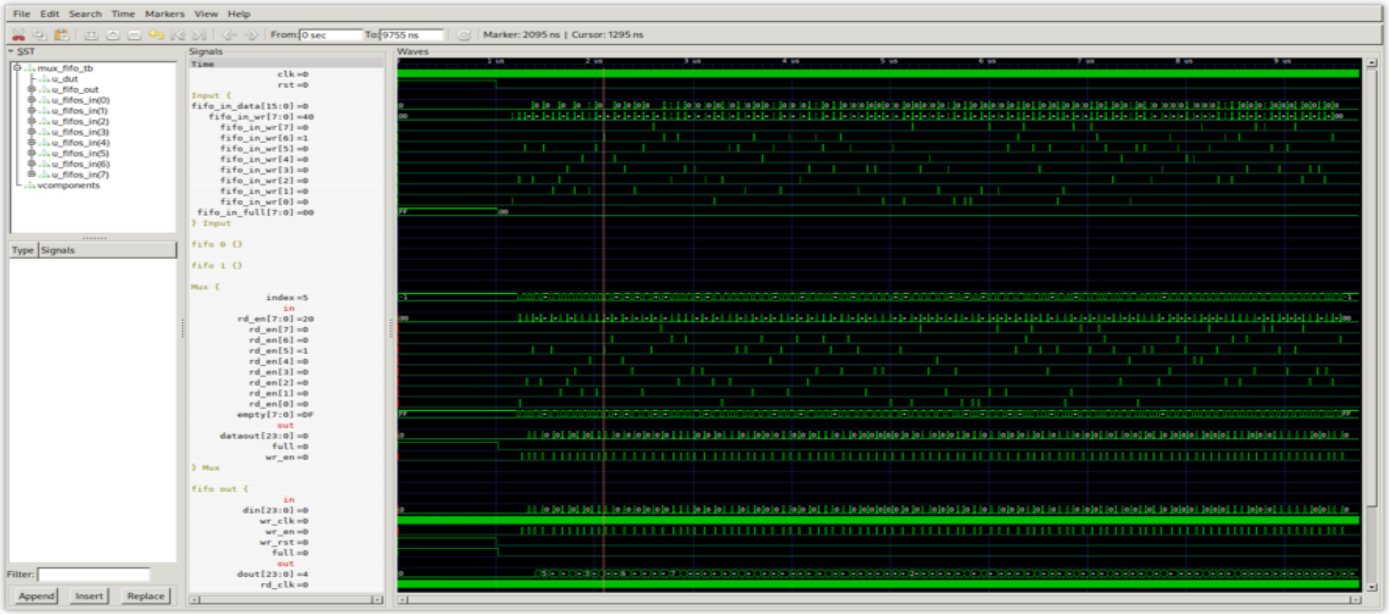


Temps total excédant *des dizaines d'heures*

Téléchargement sur cible et exécution

Cycle de développement très lent. . .

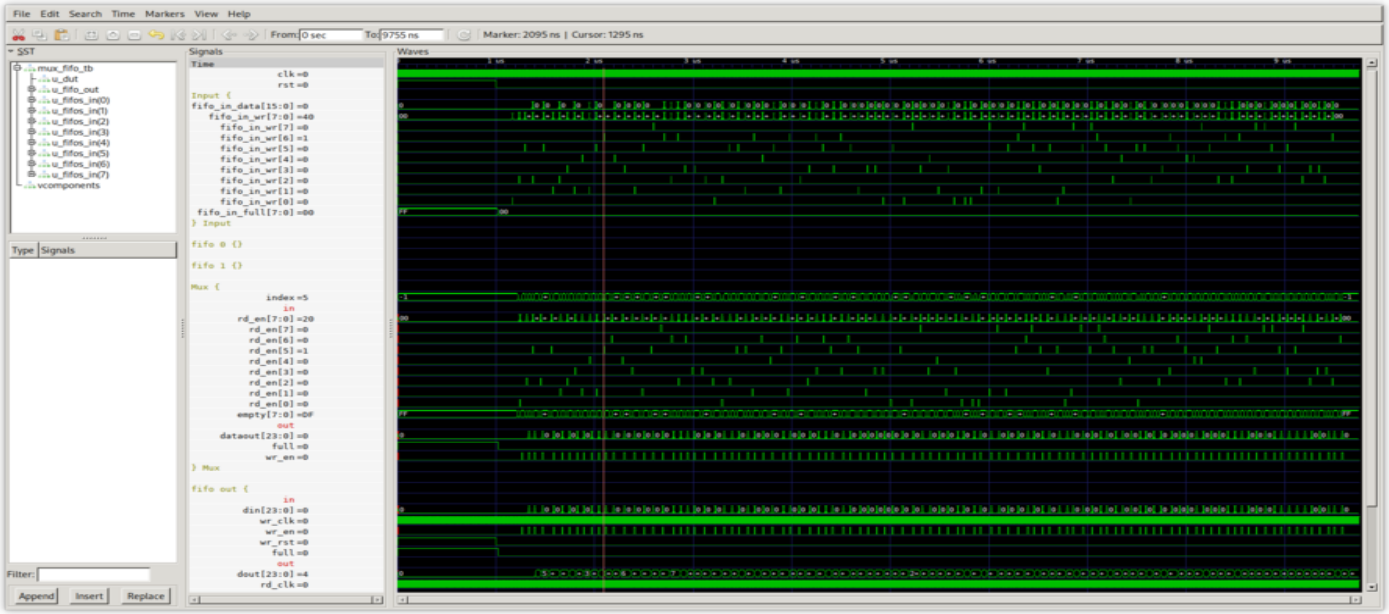
Tests : approche traditionnelle



Chronogramme d'événements logiques

- VHDL - language de bas niveau, peu flexible, peu adapté aux tests sophistiqués
- Peu adapté au codage algorithmique complexe actuel
- Peu de possibilités de vérification software simultanée, peu de tests unitaires
- Pas représentatif de la réalité, temps de simulation trop courts
- Peu de possibilités d'automatisation possible, souvent on stocke le trames sur disque
- Faible vérification fonctionnelle, couverture de code . . .
- Vérification (visuelle !) à partir d'un chronogramme . . . erreurs non detectés.

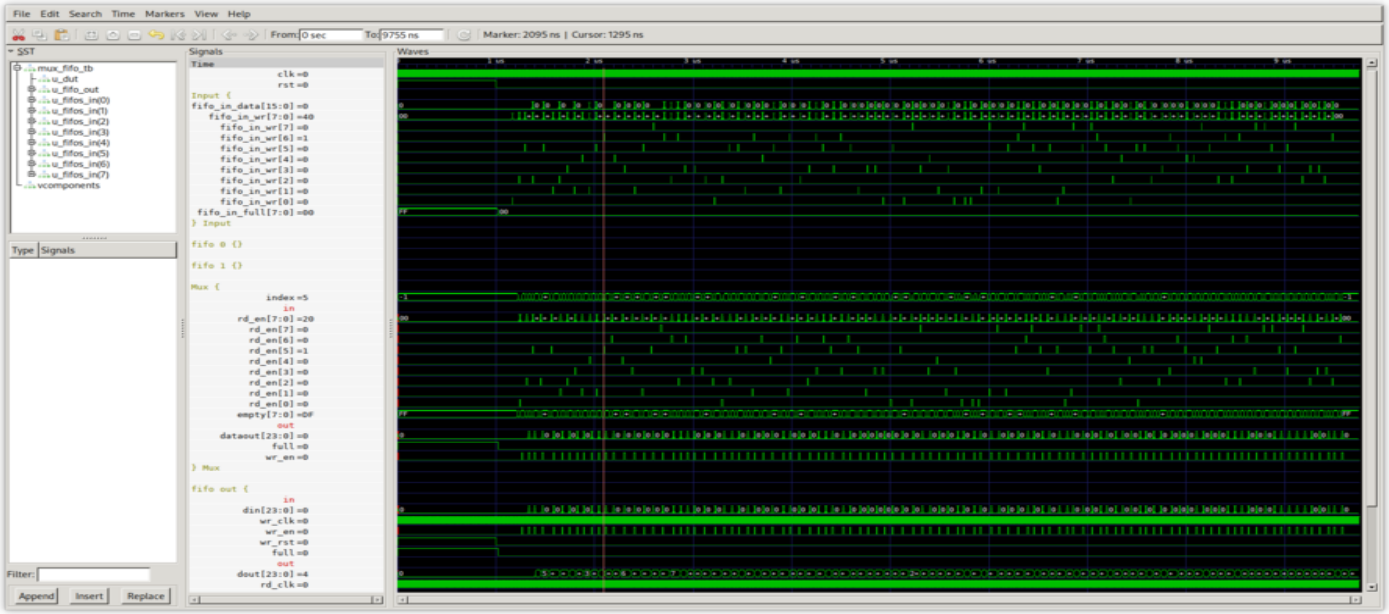
Tests : approche traditionnelle



Chronogramme d'événements logiques

- VHDL - language de bas niveau, peu flexible, peu adapté aux tests sophistiqués
- Peu adapté au codage algorithmique complexe actuel
- Peu de possibilités de vérification software simultanée, peu de tests unitaires
- Pas représentatif de la réalité, temps de simulation trop courts
- Peu de possibilités d'automatisation possible, souvent on stocke le trames sur disque
- Faible vérification fonctionnelle, couverture de code . . .
- Vérification (visuelle !) à partir d'un chronogramme . . . erreurs non detectés.

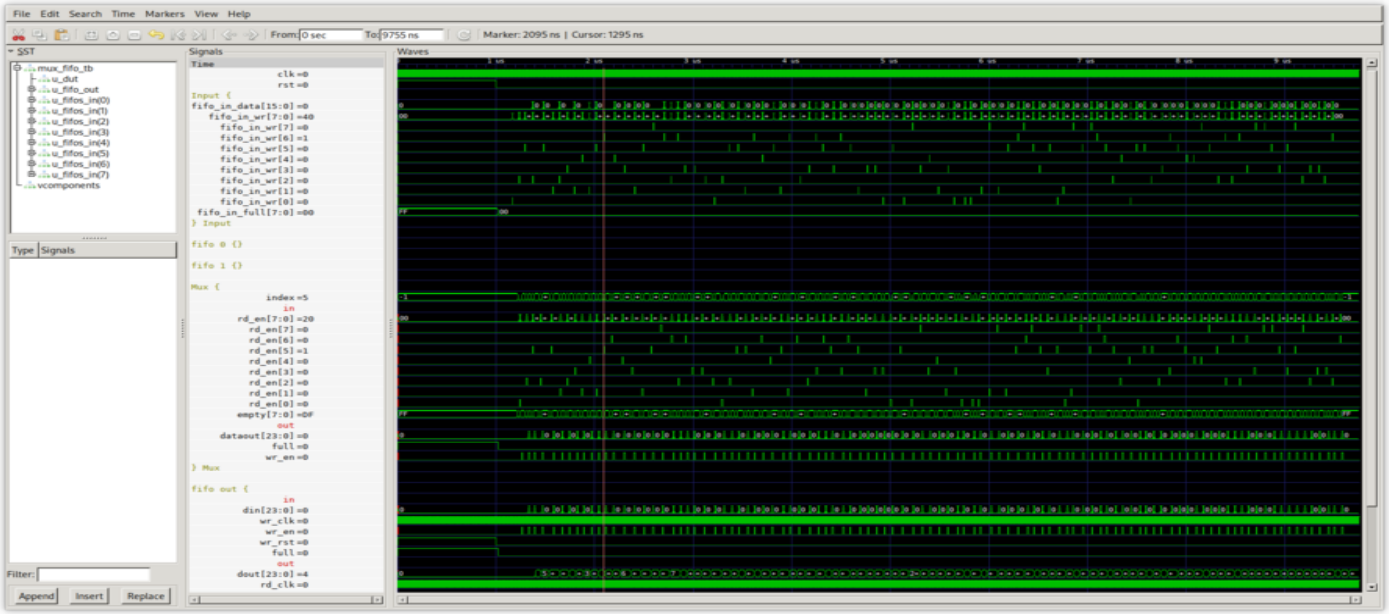
Tests : approche traditionnelle



Chronogramme d'événements logiques

- VHDL - language de bas niveau, peu flexible, peu adapté aux tests sophistiqués
- Peu adapté au codage algorithmique complexe actuel
- Peu de possibilités de vérification software simultanée, peu de tests unitaires
- Pas représentatif de la réalité, temps de simulation trop courts
- Peu de possibilités d'automatisation possible, souvent on stocke le trames sur disque
- Faible vérification fonctionnelle, couverture de code . . .
- Vérification (visuelle !) à partir d'un chronogramme . . . erreurs non detectés.

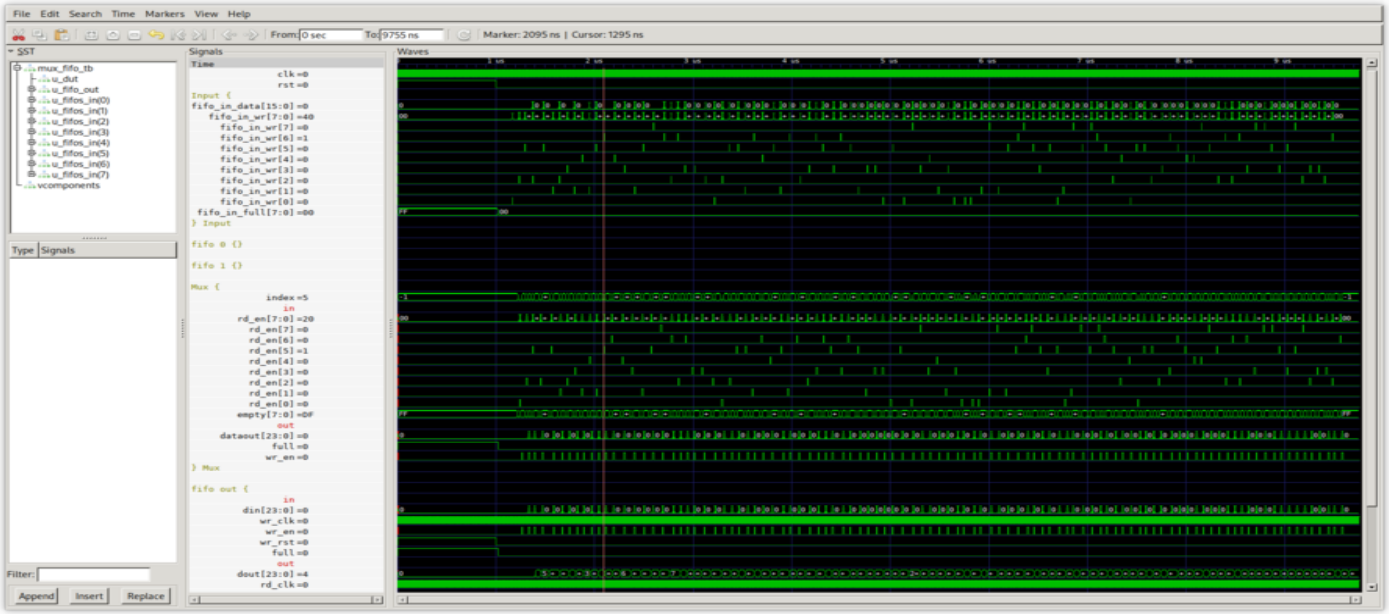
Tests : approche traditionnelle



Chronogramme d'événements logiques

- VHDL - language de bas niveau, peu flexible, peu adapté aux tests sophistiqués
- Peu adapté au codage algorithmique complexe actuel
- Peu de possibilités de vérification software simultanée, peu de tests unitaires
- Pas représentatif de la réalité, temps de simulation trop courts
- Peu de possibilités d'automatisation possible, souvent on stocke le trames sur disque
- Faible vérification fonctionnelle, couverture de code . . .
- Vérification (visuelle !) à partir d'un chronogramme . . . erreurs non detectés.

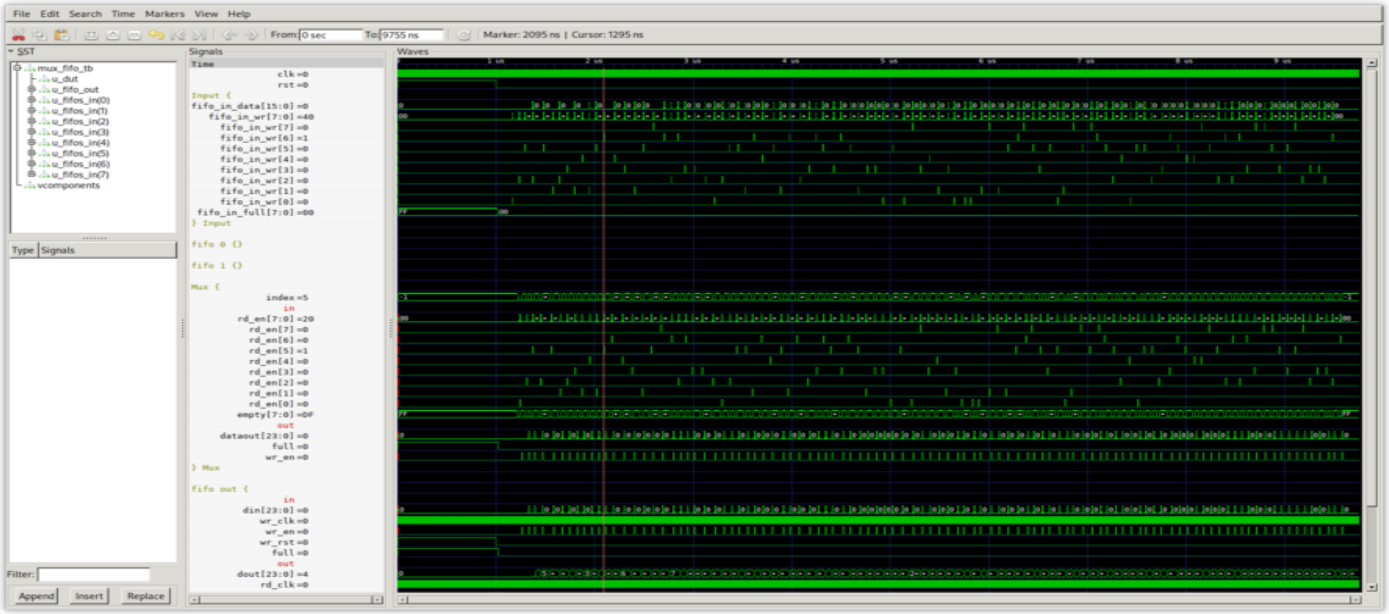
Tests : approche traditionnelle



Chronogramme d'événements logiques

- VHDL - language de bas niveau, peu flexible, peu adapté aux tests sophistiqués
- Peu adapté au codage algorithmique complexe actuel
- Peu de possibilités de vérification software simultanée, peu de tests unitaires
- Pas représentatif de la réalité, temps de simulation trop courts
- Peu de possibilités d'automatisation possible, souvent on stocke le trames sur disque
- Faible vérification fonctionnelle, couverture de code . . .
- Vérification (visuelle !) à partir d'un chronogramme . . . erreurs non detectés.

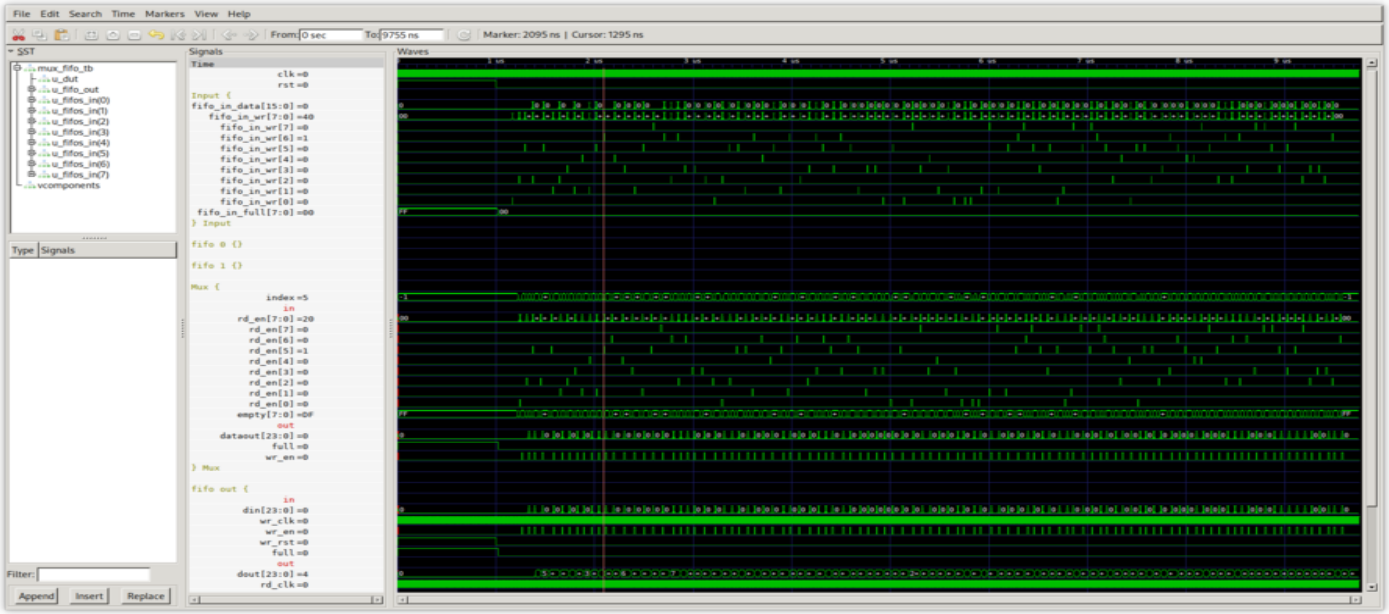
Tests : approche traditionnelle



Chronogramme d'événements logiques

- VHDL - language de bas niveau, peu flexible, peu adapté aux tests sophistiqués
- Peu adapté au codage algorithmique complexe actuel
- Peu de possibilités de vérification software simultanée, peu de tests unitaires
- Pas représentatif de la réalité, temps de simulation trop courts
- Peu de possibilités d'automatisation possible, souvent on stocke le trames sur disque
- Faible vérification fonctionnelle, couverture de code . . .
- Vérification (visuelle !) à partir d'un chronogramme . . . erreurs non detectés.

Tests : approche traditionnelle

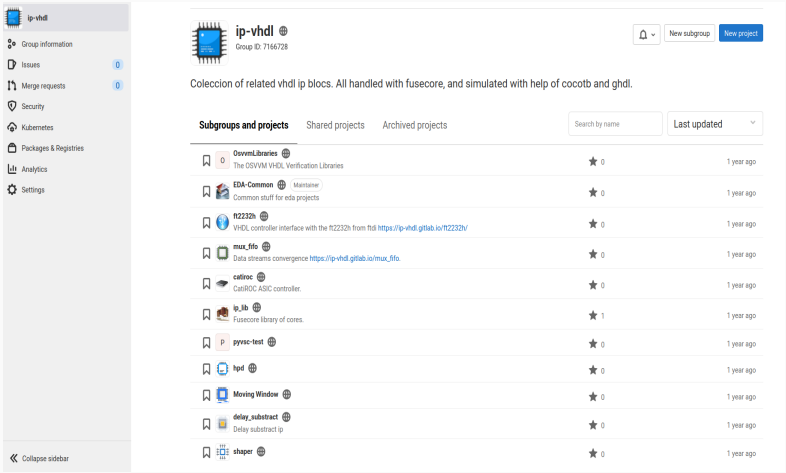


Chronogramme d'événements logiques

- VHDL - language de bas niveau, peu flexible, peu adapté aux tests sophistiqués
- Peu adapté au codage algorithmique complexe actuel
- Peu de possibilités de vérification software simultanée, peu de tests unitaires
- Pas représentatif de la réalité, temps de simulation trop courts
- Peu de possibilités d'automatisation possible, souvent on stocke le trames sur disque
- Faible vérification fonctionnelle, couverture de code ...
- Vérification (visuelle !) à partir d'un chronogramme ... erreurs non detectés.

Méthode de conception alternative

Approche "logicielle"



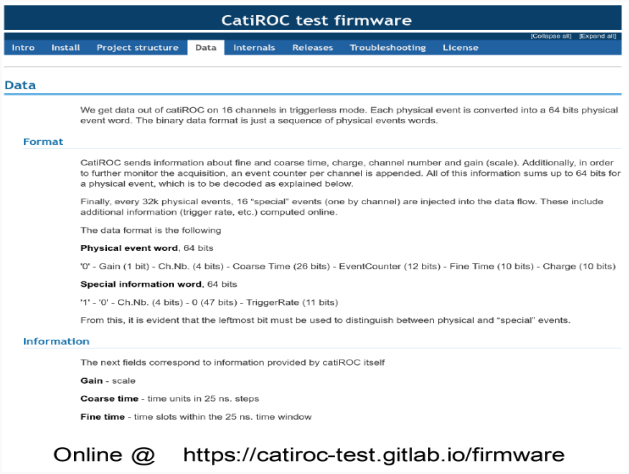
Méthodes de conception *"software"*

- Méthodes logicielles prouvés
- On adapte des méthodologies similaires
- Forge gitlab

Intégration de la notion de qualité logicielle

- Documentation integrée
- Communication (pages, code, etc.)
- Suivi de versions, ré-utilisation de code

Outils ouverts et en licence libre
Notions de CLI, KISS, scripts, makefiles, etc.



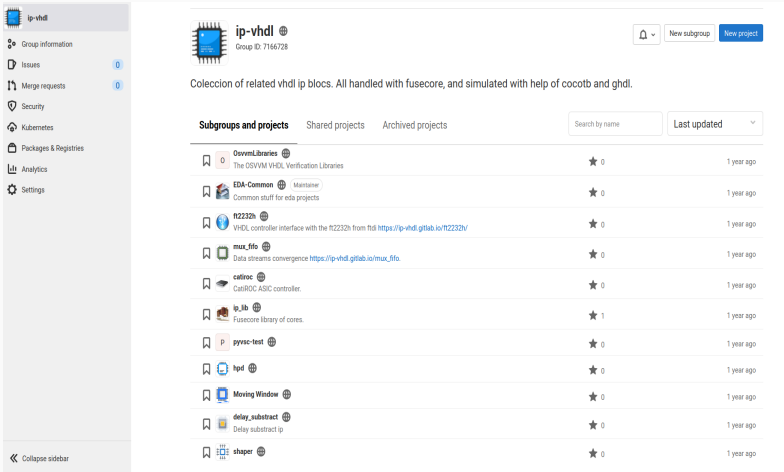
Gestion de bibliothèques de composants
Fusesoc

Outils de vérification haut niveau
Osvvm, UVVM, VUnit, etc.

Outils de simulation
GHDL, Cocotb, GtkWave, etc.

Génération de documentation
Doxygen, etc.

Approche "logicielle"



Méthodes de conception *"software"*

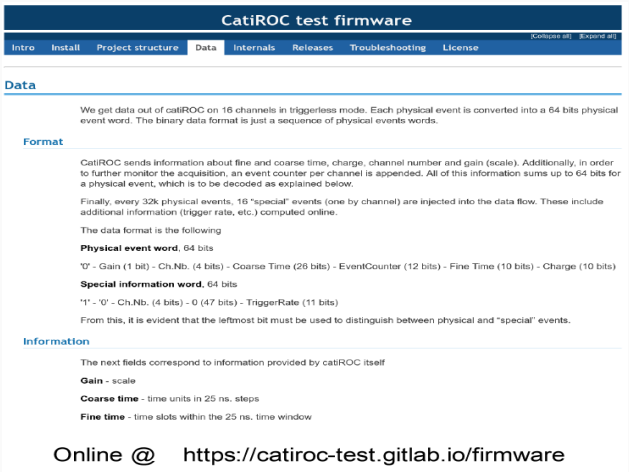
- Méthodes logicielles prouvés
- On adapte des méthodologies similaires
- Forge gitlab

Intégration de la notion de qualité logicielle

- Documentation integrée
- Communication (pages, code, etc.)
- Suivi de versions, ré-utilisation de code

Outils ouverts et en licence libre

Notions de CLI, KISS, scripts, makefiles, etc.



Gestion de bibliothèques de composants

Fusesoc

Outils de vérification haut niveau

Osvvm, UVVM, VUnit, etc.

Outils de simulation

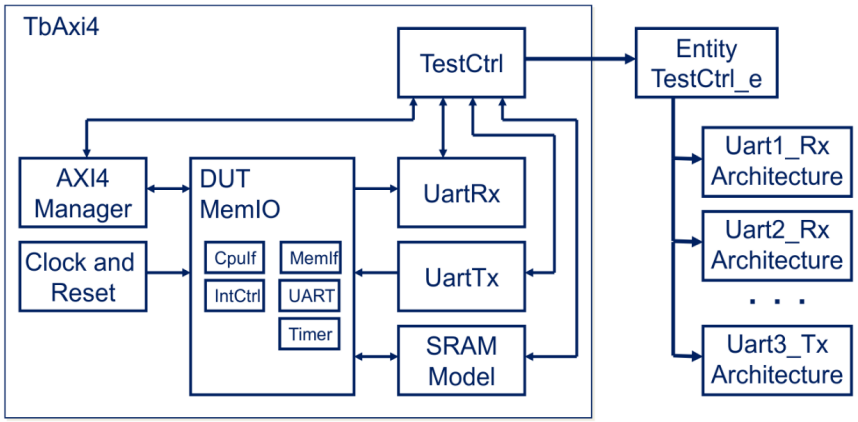
GHDL, Cocotb, GtkWave, etc.

Génération de documentation

Doxygen, etc.

Vérification et simulation

Open Source VHDL Verification Methodology



OSVVM

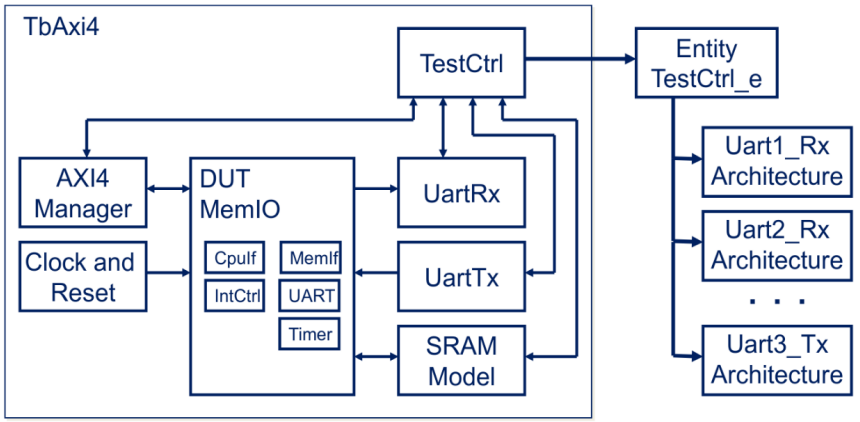
- Bibliothèque de vérification de composants VHDL
- Simplifie des taches complexes
- Couverture fonctionnelle, tests random, structures de données avancés. . .
- Tests aléatoires intelligents, contraintes, etc.
- Messages, logs, scoreboards, etc.

GHDL : gcc/llvm VHDL toolkit

- Logiciel libre d'analyse, compilation et simulation
- Libre, mur et très rapide
- Technologies gcc et llvm comme backend
- Multi plataforme
- Analyse de couverture de code
- En développement actif, très bon support
- Fonctionnalités avancées (std 2008, etc.)
- Interface cli : possibilité de créer des scripts
- Très bon support pour Osvvm, Uvvm, VUnit, etc.

Vérification et simulation

Open Source VHDL Verification Methodology



OSVVM

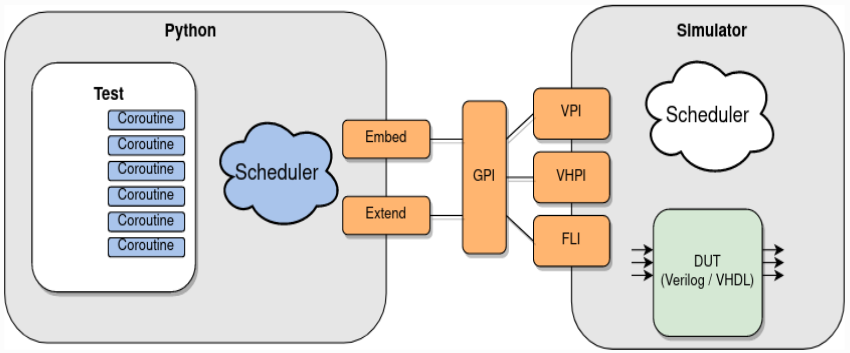
- Bibliothèque de vérification de composants VHDL
- Simplifie des taches complexes
- Couverture fonctionnelle, tests random, structures de données avancés. . .
- Tests aléatoires intelligents, contraintes, etc.
- Messages, logs, scoreboards, etc.

GHDL : gcc/llvm VHDL toolkit

- Logiciel libre d'analyse, compilation et simulation
- Libre, mur et très rapide
- Technologies gcc et llvm comme backend
- Multi plataforme
- Analyse de couverture de code
- En développement actif, très bon support
- Fonctionnalités avancées (std 2008, etc.)
- Interface cli : possibilité de créer des scripts
- Très bon support pour Osvvm, Uvvm, VUnit, etc.

Testbench haut niveau

Bibliothèque Cocotb



- Python - haut niveau, simple et puissant
- Ecosystème complet pour banc tests
- Pilotage distant du moteur GHDL
- Conception matérielle, proche du hw
- Sw simulé au même temps que le hw
- Interface bidirectionnelle vers une DUT en VHDL
- Notion de temps, d'événements, d'horloge, de latence, introspection de signaux, etc.

Co-simulation RTl-sw/hw


Cocotb: a Python-based digital logic verification framework

Ben Rosser

University of Pennsylvania

December 10, 2019

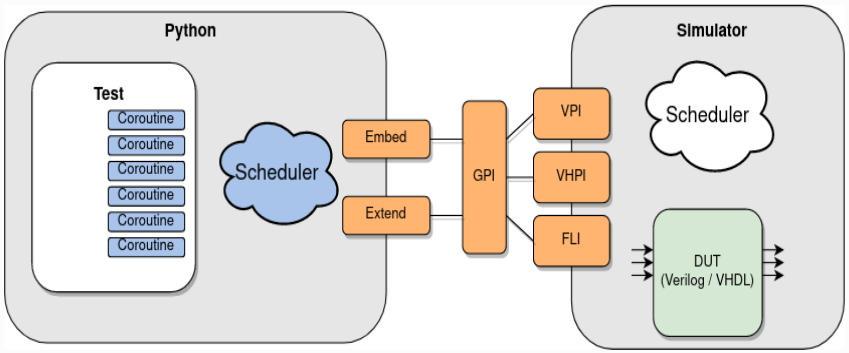
Penn
University of Pennsylvania

ATLAS
EXPERIMENT

- Développement algorithmique simplifié
- Vérification de résultats simplifié
- Chronogrammes pour développement du banc de test seulement
- Tests plus significatifs, en profondeur et pendant plus longtemps
- Possibilité de réjoindre des environments de test logiciels (tests unitaires)
- Plus besoin de vérification visuelle
- Capacités avancés de tests, assertions

Testbench haut niveau

Bibliothèque Cocotb



- Python - haut niveau, simple et puissant
- Ecosystème complet pour banc tests
- Pilotage distant du moteur GHDL
- Conception matérielle, proche du hw
- Sw simulé au même temps que le hw
- Interface bidirectionnelle vers une DUT en VHDL
- Notion de temps, d'événements, d'horloge, de latence, introspection de signaux, etc.

Co-simulation RTl-sw/hw


Cocotb: a Python-based digital logic verification framework

Ben Rosser

University of Pennsylvania

December 10, 2019

Penn
University of Pennsylvania

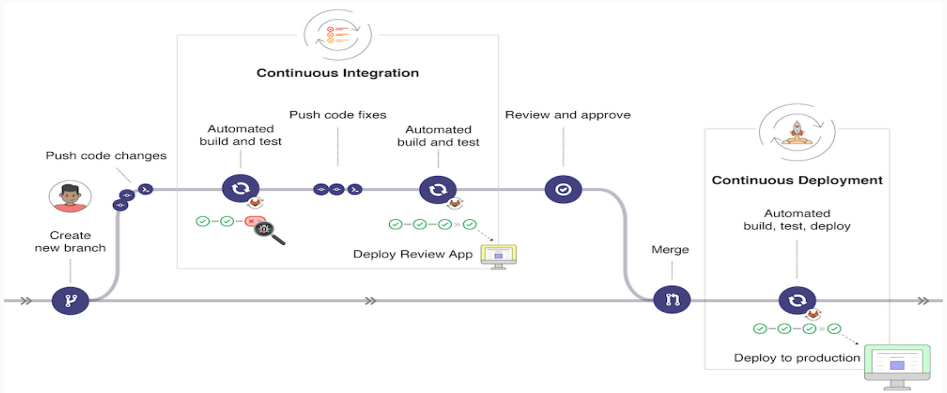
ATLAS
EXPERIMENT

- Développement algorithmique simplifié
- Vérification de résultats simplifié
- Chronogrammes pour développement du banc de test seulement
- Tests plus significatifs, en profondeur et pendant plus longtemps
- Possibilité de réjoindre des environments de test logiciels (tests unitaires)
- Plus besoin de vérification visuelle
- Capacités avancés de tests, assertions

L2IT

Intégration continue

Gitlab / Gitlab-CI

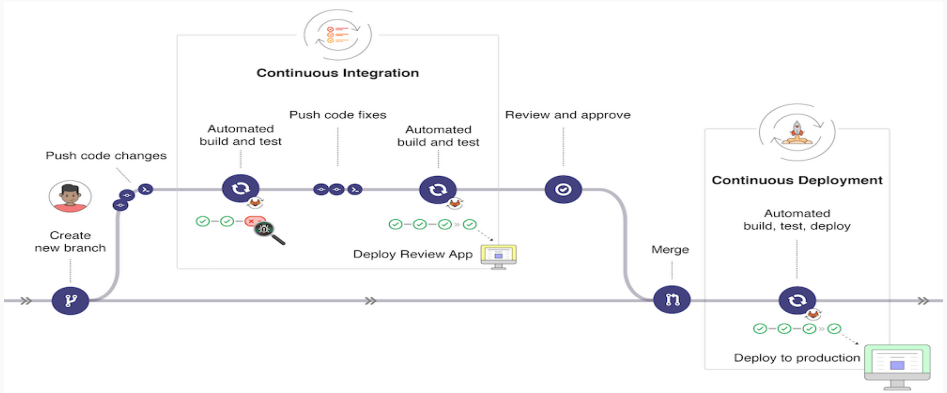


- Suivi de version du code avec git
- Possibilité de collaborer, participer dans des gros projects, etc.
- Modularité avec submodules (composants matériels)
- Gestion simplifié de bibliothèques de composants distantes (`fusesoc`)
- Suivi des dépendances et des versions (tags, releases)
- Copie de sécurité !

- Automatisation des tests avec la CI Gitlab
- (Presque) déterminisme : tests exécutés dans des containeurs `docker`
- Plus besoin de bloquer la CPU de son poste de travail
- Documentation integré au projet avec `doxygen` + deploiement sur `Gitlab pages`
- Génération automatisée du bitstream
- Serveur maison sur ssh dedié
- Sorties : logs, binaires, histograms, . . . tout

Intégration continue

Gitlab / Gitlab-CI

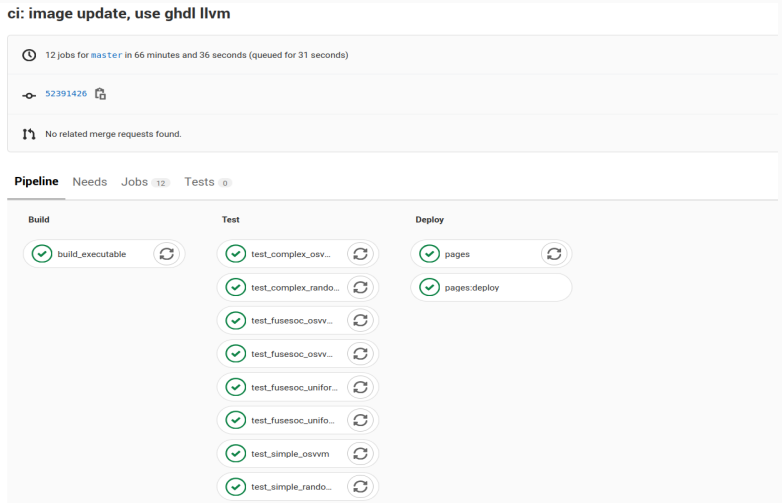


- Suivi de version du code avec git
- Possibilité de collaborer, participer dans des gros projects, etc.
- Modularité avec submodules (composants matériels)
- Gestion simplifié de bibliothèques de composants distantes (`fusesoc`)
- Suivi des dépendances et des versions (tags, releases)
- Copie de sécurité !

- Automatisation des tests avec la CI Gitlab
- (Presque) déterminisme : tests exécutés dans des containeurs `docker`
- Plus besoin de bloquer la CPU de son poste de travail
- Documentation integré au projet avec `doxygen` + deploiement sur `Gitlab pages`
- Génération automatisée du bitstream
- Serveur maison sur ssh dedié
- Sorties : logs, binaires, histograms, ... tout

Projects d'example

Mux fifo : https ://gitlab.com/ip-vhdl/mux_fifo



```
#!/bin/sh

# setup fusesoc
export fusesoc_tmp=/tmp/fusesoc
[ -e $fusesoc_tmp ] && rm -rf $fusesoc_tmp
mkdir -p $fusesoc_tmp/fusesoc_libraries
touch $fusesoc_tmp/fusesoc.conf
alias fs='fusesoc --config $fusesoc_tmp/fusesoc.conf'

# add ip_lib library
fs library add --location $fusesoc_tmp/fusesoc_libraries/ip_lib --sync-type git ip_lib https://gitlab.com/ip-vhdl
/ip_lib
fs library update
fs library list

# check ip core
fs core list
fs core show mux_fifo

# run

# necessary, as hard coded in Makefile
cd /tmp

# Reproduce here tests in ci
fs run --target=simulate.run.osvvm.simple --tool=generic mux_fifo
```

```
4: 0 - -[mux_fifo]_/test.fusesoc.osvvm.simple.sh     All -master Shell-script[sh] company-box + [mux_fifo.csantos] 5
#!/bin/sh

# restart from scratch, just in case
# make clean

# data producer is vhdl osvvm library
MIN_INTERVAL=50 MAX_INTERVAL=300 NB_CHANNELS=60 TIMEOUT=1000 COVERAGE_SAMPLES_OSVVM=1 make -f Makefile.mux_fifo
```

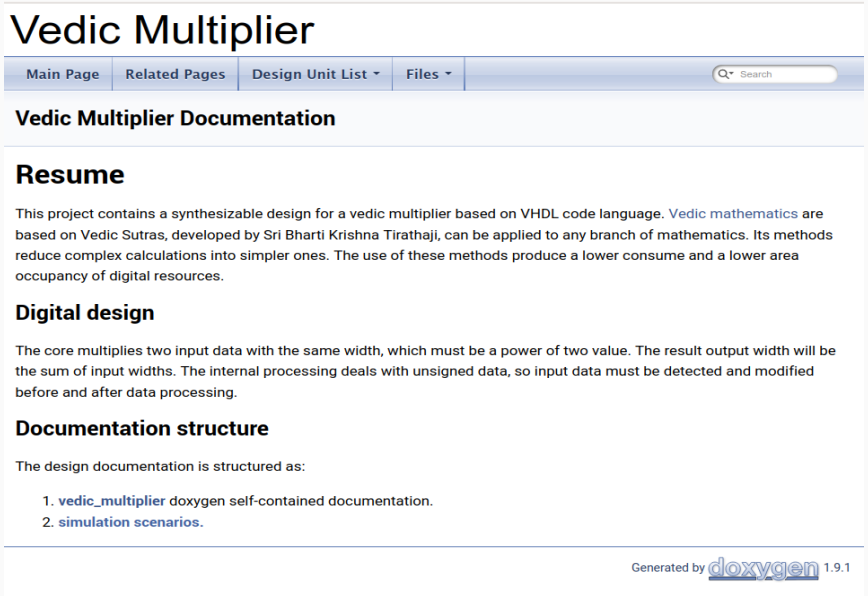
- NxM - fifos en parallel vers 1 fifo
- Controlleur USB ft2232h
- Evenements random dans le temps / par canal
- Numéro d'événement, time stamp
- Gestion periphériques avec Fusesoc
- TB en Python
- Vérification / histogrammes avec Osvvm

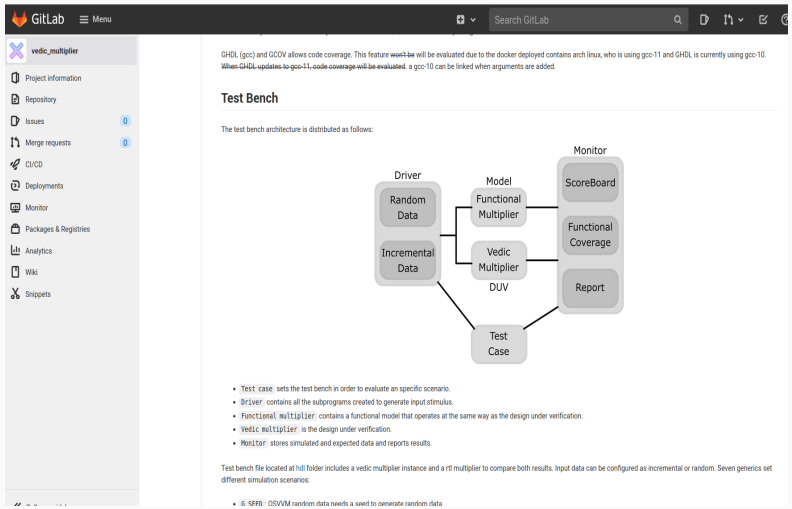
- CI dans container docker sur mesure
- ArchLinux image comprend custom ghdl, osvvm, cocotb, fusesoc, etc.
- Git clone & make
- Simulations : makefile parametrable
- Project commun avec code support

Tests complexes : injection d'événements sw random pendant des jours + tests en sortie à la volée.

Vedic : https ://gitlab.com/jcorrecher/vedic_multiplier

https ://jcorrecher.gitlab.io/vedic_multiplier/doc/html/index.html





La documentation doit évoluer avec le code.

- Structuré, git-submodules
- Hierarchie de dossiers
- GHDL+Cocotb+Osvvm
- Simulation fonctionnelle
- Coverture de code

- Parfaitement documenté
- Readme.org par dossier
- References
- Code commenté
- ci -> doxygen -> gitlab pages

Vedic : https ://gitlab.com/jcorrecher/vedic_multiplier

Coverture fonctionnelle et de code avec Osvvm

LCOV - code coverage report

Current view: top level
Test: vedic_multiplier_tb.info
Date: 2021-09-14 16:12:20

	Hit	Total	Coverage
Lines:	315	444	70.9 %
Functions:	83	100	83.0 %

Directory	Line Coverage		Functions	
/builds/jcorrecher/vedic_multiplier/common/hdl	16.0 %	4 / 25	42.9 %	3 / 7
/builds/jcorrecher/vedic_multiplier/rtl	100.0 %	81 / 81	84.0 %	21 / 25
hdl	68.0 %	230 / 338	86.8 %	59 / 68

Generated by: LCOV version 1.15-2-gb9d6727

LCOV - code coverage report

Current view: top level - builds/jcorrecher/vedic_multiplier/rtl - vedic_multiplier.vhd (source / functions)
Test: vedic_multiplier_tb.info
Date: 2021-09-14 16:12:20

	Hit	Total	Coverage
Lines:	81	81	100.0 %
Functions:	21	25	84.0 %

Function Name	Hit count
work__vedic_multiplier__ARCH__rtl__t_split_BUILDER	0
work__vedic_multiplier__ARCH__rtl__t_split_EQ	0
work__vedic_multiplier__ARCH__rtl__t_split_SIGBUILDER	0
work__vedic_multiplier__STMT_ELAB	0
work__vedic_multiplier__ARCH__rtl__DECL_ELAB	1
work__vedic_multiplier__ARCH__rtl__P0__PROC	1
work__vedic_multiplier__ARCH__rtl__P1__PROC	1
work__vedic_multiplier__ARCH__rtl__STMT_ELAB	1
work__vedic_multiplier__DECL_ELAB	1
work__vedic_multiplier__PKG_ELAB	1
work__vedic_multiplier__ARCH__rtl__P7__PROC	4
work__vedic_multiplier__ARCH__rtl__P3__PROC	259
work__vedic_multiplier__ARCH__rtl__u_split__P1__PROC	278
work__vedic_multiplier__ARCH__rtl__P5__PROC	18483
work__vedic_multiplier__ARCH__rtl__P4__PROC	30467
work__vedic_multiplier__ARCH__rtl__P6__PROC	65283
work__vedic_multiplier__ARCH__rtl__P2__PROC	65544
work__vedic_multiplier__ARCH__rtl__P8__PROC	65795
work__vedic_multiplier__ARCH__rtl__u_split__P0__PROC	69642
work__vedic_multiplier__ARCH__rtl__sign_detect__PROC	131324
work__vedic_multiplier__ARCH__rtl__srl_ctrl__PROC	131324
work__vedic_multiplier__ARCH__rtl__stage1__PROC	131324
work__vedic_multiplier__ARCH__rtl__stage2__PROC	131324
work__vedic_multiplier__ARCH__rtl__stage3__PROC	131324
work__vedic_multiplier__ARCH__rtl__stage4__PROC	131324

Generated by: LCOV version 1.15-2-gb9d6727

- Quels lignes de code ont été executés ?
- Combien de fois ? Est-ce normal ?
- Il y a du code non executé ? Porquoi ?

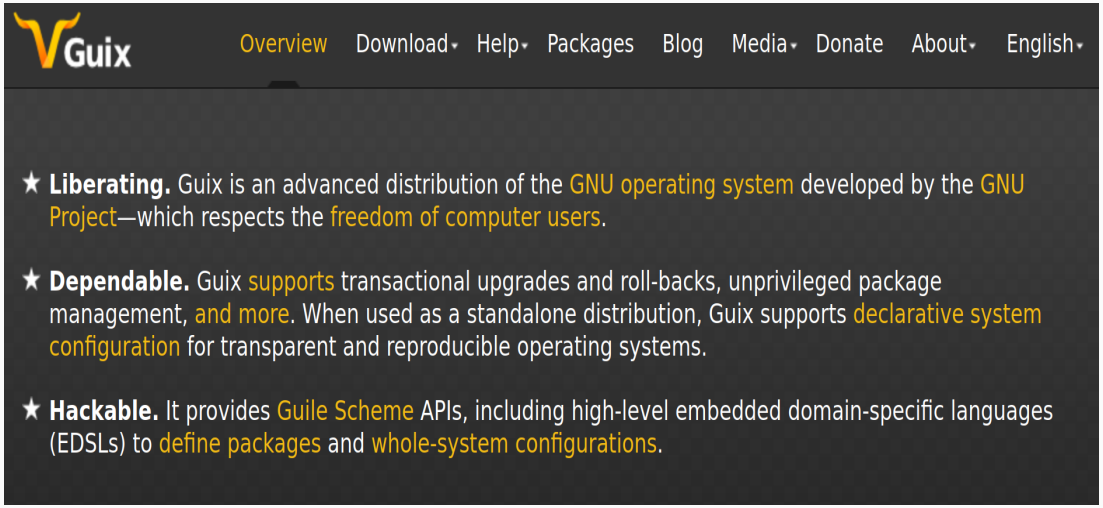
```
%% Bin:(144)    Count = 29   AtLeast = 1
%% Bin:(145)    Count = 24   AtLeast = 1
%% Bin:(146)    Count = 25   AtLeast = 1
%% Bin:(147)    Count = 18   AtLeast = 1
%% Bin:(148)    Count = 12   AtLeast = 1
%% Bin:(149)    Count = 13   AtLeast = 1
%% Bin:(150)    Count = 16   AtLeast = 1
%% Bin:(151)    Count = 20   AtLeast = 1
%% Bin:(152)    Count = 16   AtLeast = 1
%% Bin:(153)    Count = 17   AtLeast = 1
%% Bin:(154)    Count = 10   AtLeast = 1
%% Bin:(155)    Count = 14   AtLeast = 1
%% Bin:(156)    Count = 14   AtLeast = 1
%% Bin:(157)    Count = 22   AtLeast = 1
%% Bin:(158)    Count = 21   AtLeast = 1
%% Bin:(159)    Count = 22   AtLeast = 1
%% Bin:(160)    Count = 29   AtLeast = 1
%% Bin:(161)    Count = 11   AtLeast = 1
%% Bin:(162)    Count = 12   AtLeast = 1
```

- Injection de mots de test
- Combien ? Lesquels ?
- Histogramme des entrées / sorties
- Histogramme croisé
- Critère d'arret des tests ?

Conclusions

Aller au-délà

Prochaine étape : faire appel à des outils modernes fournissant un contexte reproductible

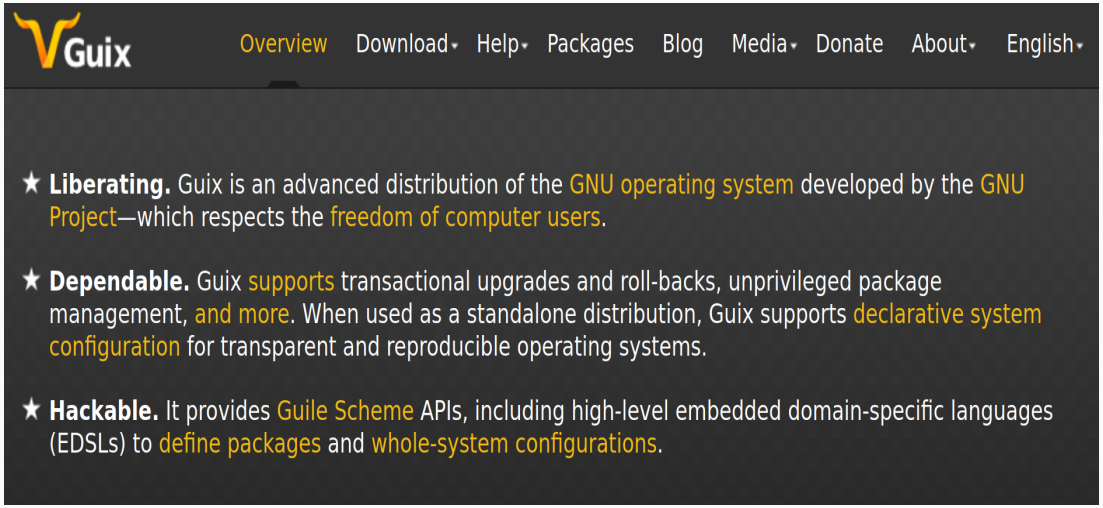


hpc.guix.info/events/2021/atelier-reproductibilité-environnements

- Environment connu et maitrissé où faire tourner des tests
- Env. indépendant d'où et de quand : résultats prédécibles, toujours identiques
- Contexte contenu et figé dans des fichiers en texte plat (fichiers manifest)
- Manifest faisant partie du project, tracés avec git
- Quelques points de recherche . . .

Aller au-délà

Prochaine étape : faire appel à des outils modernes fournissant un contexte reproductible

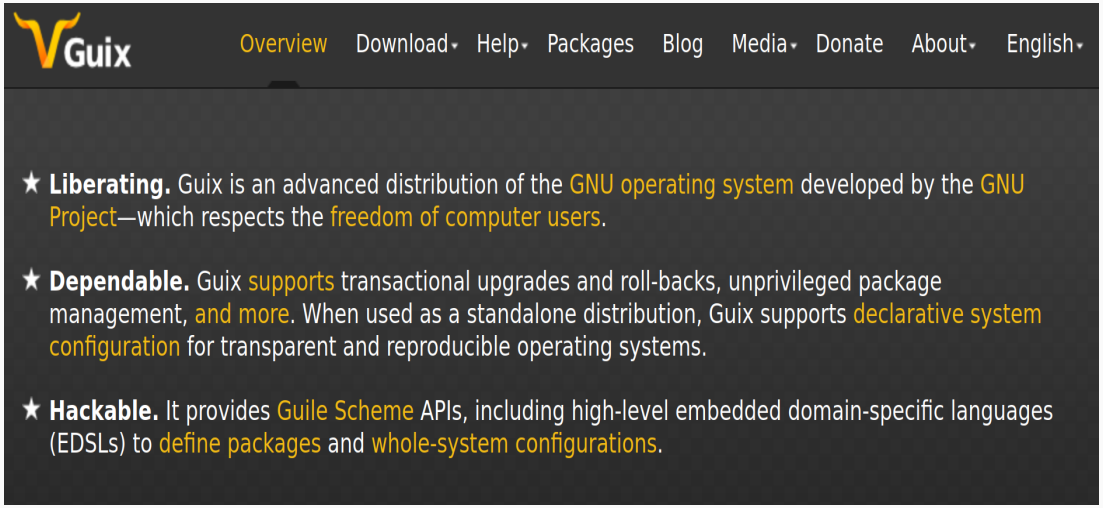


hpc.guix.info/events/2021/atelier-reproductibilité-environnements

- Environment connu et maitrissé où faire tourner des tests
- Env. indépendant d'où et de quand : résultats prédécibles, toujours identiques
- Contexte contenu et figé dans des fichiers en texte plat (fichiers manifest)
- Manifest faisant partie du project, tracés avec git
- Quelques points de recherche . . .

Aller au-délà

Prochaine étape : faire appel à des outils modernes fournissant un contexte reproductible

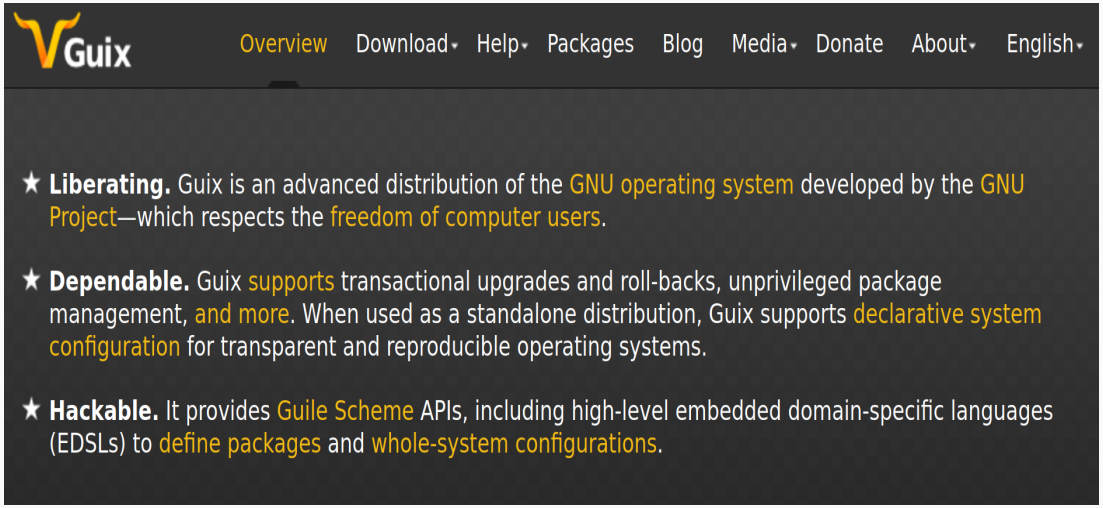


hpc.guix.info/events/2021/atelier-reproductibilité-environnements

- Environment connu et maitrissé où faire tourner des tests
- Env. indépendant d'où et de quand : résultats prédécibles, toujours identiques
- Contexte contenu et figé dans des fichiers en texte plat (fichiers manifest)
- Manifest faisant partie du project, tracés avec git
- Quelques points de recherche . . .

Aller au-délà

Prochaine étape : faire appel à des outils modernes fournissant un contexte reproductible

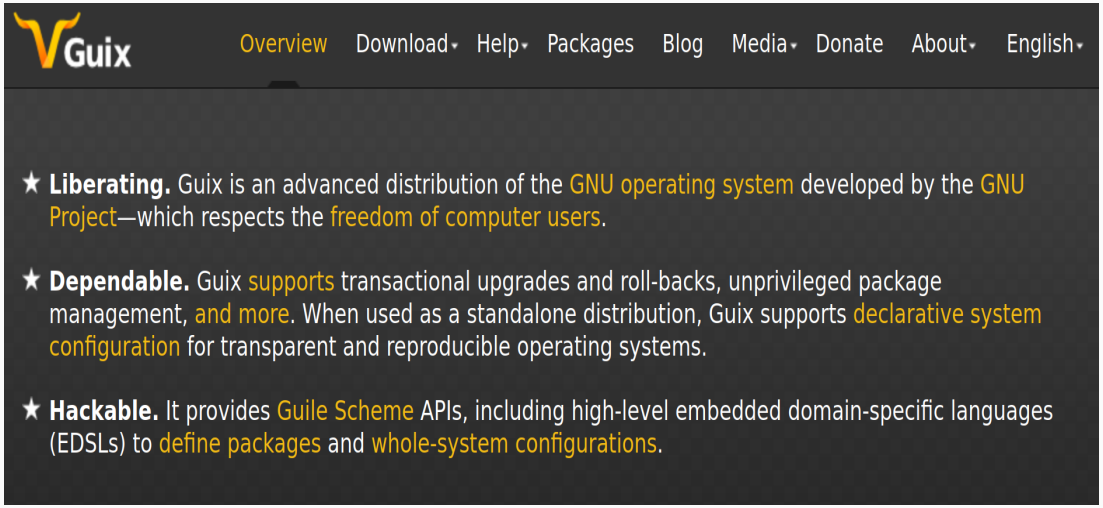


hpc.guix.info/events/2021/atelier-reproductibilité-environnements

- Environment connu et maitrissé où faire tourner des tests
- Env. indépendant d'où et de quand : résultats prédécibles, toujours identiques
- Contexte contenu et figé dans des fichiers en texte plat (fichiers manifest)
- Manifest faisant partie du project, tracés avec git
- Quelques points de recherche . . .

Aller au-délà

Prochaine étape : faire appel à des outils modernes fournissant un contexte reproductible

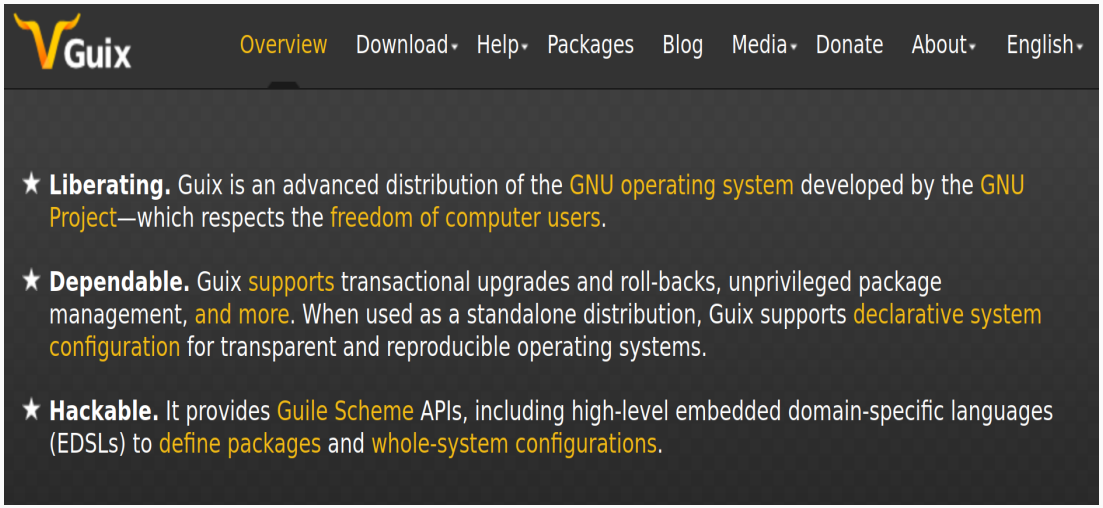


hpc.guix.info/events/2021/atelier-reproductibilité-environnements

- Environment connu et maitrissé où faire tourner des tests
- Env. indépendant d'où et de quand : résultats prédécibles, toujours identiques
- Contexte contenu et figé dans des fichiers en texte plat (fichiers manifest)
- Manifest faisant partie du project, tracés avec git
- Quelques points de recherche . . .

Aller au-délà

Prochaine étape : faire appel à des outils modernes fournissant un contexte reproductible



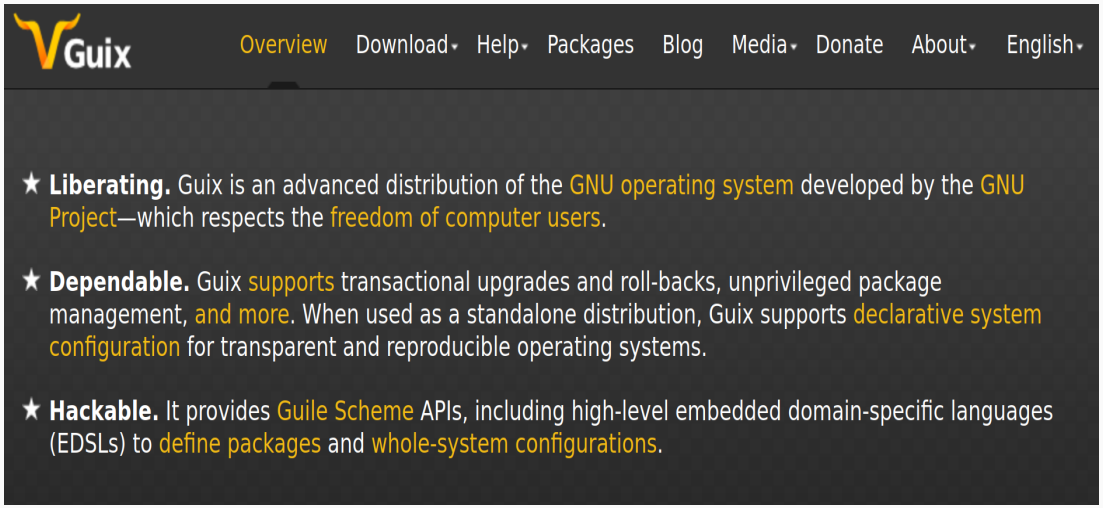
hpc.guix.info/events/2021/atelier-reproductibilité-environnements

- Environment connu et maitrissé où faire tourner des tests
- Env. indépendant d'où et de quand : résultats prédécibles, toujours identiques
- Contexte contenu et figé dans des fichiers en texte plat (fichiers manifest)
- Manifest faisant partie du project, tracés avec git
- Quelques points de recherche . . .

Comment coupler Guix avec CI Gitlab ?

Aller au-délà

Prochaine étape : faire appel à des outils modernes fournissant un contexte reproductible



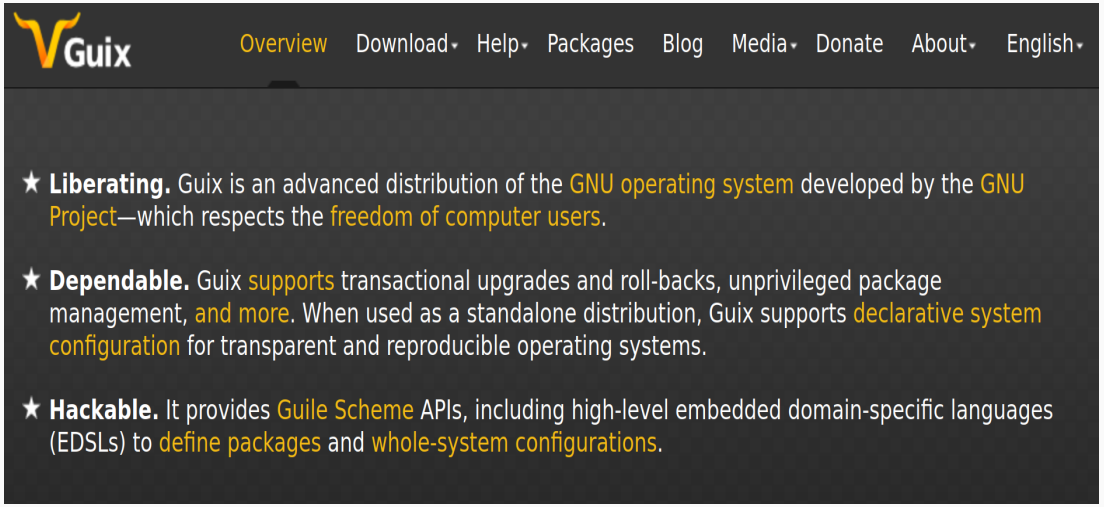
hpc.guix.info/events/2021/atelier-reproductibilité-environnements

- Environment connu et maitrissé où faire tourner des tests
- Env. indépendant d'où et de quand : résultats prédécibles, toujours identiques
- Contexte contenu et figé dans des fichiers en texte plat (fichiers manifest)
- Manifest faisant partie du project, tracés avec git
- Quelques points de recherche . . .

Méthodes HPC/GPU pour accelerer la simulation ?

Aller au-délà

Prochaine étape : faire appel à des outils modernes fournissant un contexte reproductible



hpc.guix.info/events/2021/atelier-reproductibilité-environnements

- Environment connu et maitrissé où faire tourner des tests
- Env. indépendant d'où et de quand : résultats prédécibles, toujours identiques
- Contexte contenu et figé dans des fichiers en texte plat (fichiers manifest)
- Manifest faisant partie du project, tracés avec git
- Quelques points de recherche . . .

Génération automatisée du bitstream dans serveur ssh ?

Sommaire

- Pourquoi ne pas profiter des méthodologies existantes ?
- Des technologies libres et puissantes existent. Autant en profiter.
- La simulation et la vérification matérielles ont beaucoup à y gagner
- VHDL / Verilog limités en capacité de co-simulation
- La vérification de code est très complèxe : on dot automatiser autant que possible

Sommaire

- Pourquoi ne pas profiter des méthodologies existantes ?
- Des technologies libres et puissantes existent. Autant en profiter.
- La simulation et la vérification matérielles ont beaucoup à y gagner
- VHDL / Verilog limités en capacité de co-simulation
- La vérification de code est très complèxe : on dot automatiser autant que possible

Sommaire

- Pourquoi ne pas profiter des méthodologies existantes ?
- Des technologies libres et puissantes existent. Autant en profiter.
- La simulation et la vérification matérielles ont beaucoup à y gagner
- VHDL / Verilog limités en capacité de co-simulation
- La vérification de code est très complèxe : on dot automatiser autant que possible

Sommaire

- Pourquoi ne pas profiter des méthodologies existantes ?
- Des technologies libres et puissantes existent. Autant en profiter.
- La simulation et la vérification matérielles ont beaucoup à y gagner
- VHDL / Verilog limités en capacité de co-simulation
- La vérification de code est très complèxe : on dot automatiser autant que possible

Sommaire

- Pourquoi ne pas profiter des méthodologies existantes ?
- Des technologies libres et puissantes existent. Autant en profiter.
- La simulation et la vérification matérielles ont beaucoup à y gagner
- VHDL / Verilog limités en capacité de co-simulation
- La vérification de code est très complèxe : on dot automatiser autant que possible

Sommaire

- Pourquoi ne pas profiter des méthodologies existantes ?
- Des technologies libres et puissantes existent. Autant en profiter.
- La simulation et la vérification matérielles ont beaucoup à y gagner
- VHDL / Verilog limités en capacité de co-simulation
- La vérification de code est très complèxe : on dot automatiser autant que possible

Questions ?