Guide d'utilisation

Protection contre les perturbations dans un FPGA : TMR

Par Tom Désesquelle, Axel Baldacchino, Pierre-Olivier Guessard
2022-2023, 3AMT

Table des matières

Table des illustrations

1. Construction de circuit tolérants aux fautes

Ce guide a pour objectif de vous aider à utiliser notre solution pour la protection contre les perturbations avec la méthode de redondance dite TMR (Triple Modular Redundancy). En effet, la méthode d'implémentation développée permet de rendre un circuit tolérant aux fautes, tout en optimisant l'espace occupé des blocs triplés dans le FPGA.
L'automatisation passe par une interface graphique (GUI) permettant à l'utilisateur de facilement utiliser cette méthode de redondance sans avoir à passer plusieurs outils séparément.

2. Hiérarchie de fichier

2.1 Scripts

2.1.1 Script TCL 1

Plusieurs scripts TCL sont exécutés lors du processus de construction de circuit tolérant aux fautes que nous avons développé. Le premier script TCL du flot permet d'importer un fichier Vivado existant à partir des fichiers fournis par l'utilisateur (projet Vivado, fichiers de contraintes et banc de test). Une synthèse est exécutée, la netlist et le rapport d'utilisation générés en sont extraits.

2.1.2 Algorithme d'estimation d'espace occupée par la redondance : *utilization.py*

Le script *utilization.py* permet de calculer la place libre dans le FPGA en prenant en compte la place occupée du circuit initial. Le but est d'estimer si l'augmentation de taille du circuit après la redondance pour savoir si l'espace libre du FPGA est suffisant pour accueillir ce nouveau circuit.

Fichier d'entrée	
	./synth/reports/utilization_report
Variables de sortie	
	Nombre d'IO restantes
	Nombre de BUF restants
	Nombre de LUT restants

Tableau 1 : Fichiers d'entrées/Varibles de sorties script utilization.py

2.1.3 Script d'extraction du pinout du FPGA choisi : *internet.py*

Le script *internet.py* permet de récupérer sur internet le pinout du FPGA renseigné par l'utilisateur.

Variable d'entrée	
	Cible FPGA choisie
Fichier de sortie	
	*./pinout/**pinout.txt***

Tableau 2 : Variable d'entrée/Fichier de sortie du script internet.py

2.1.4 Générateur de contraintes : *constraintGenerator.py*

Le script *constraitGenerator.py* permet de générer les fichiers de contraintes propres au circuit triplé, à partir des fichiers de contraintes initiaux renseignés par l'utilisateur.

Fichier d'entrée	
	pinout.txt
	*./constrs/**IO_poroperties.xdc***
	*./constrs/**pad_location.xdc***
	*./constrs/**timing.xdc***
Fichier de sortie	
	*.constrs/new/**IO_properties.xdc***
	*.constrs/new/**pad_location.xdc***
	*.constrs/new/**timing.xdc***

Tableau 3 : Fichiers d'entrées/sorties du script constraintsGenerator.py

2.1.5 Algorithme de redondance : *NMR_TOP.py*

Le script *NMR_TOP.py* est l'algorithme principal du système. Il permet de rendre un circuit tolérant aux fautes en appliquant la technique de redondance sur toutes ses fonctions. Il prend en entrée la netlist du circuit initial et génère la netlist du circuit tolérant aux fautes.

Fichier d'entrée	
	Netlist du circuit pré-redondance (*./netlist/***.edf***)
Fichier de sortie	
	Netlist du circuit post-redondance (*./netlist/***.edf***)

Tableau 4 : Fichiers d'entrées/sorties du script NMR_TOP.py

2.1.6 Script TCL 2

Ce second script TCL permet de créer un projet Vivado à partir de la netlist générée par l'algorithme de redondance. Le script extrait dans un fichier verilog les IOs du circuit qui sera utilisé ultérieurement.

2.1.7 Algorithme d'optimisation de l'espace à l'implémentation : *CellsCombining.py*

Le script *CellsCombining.py* est un algorithme qui permet de construire un fichier de contrainte (*.xdc*) optimisant l'espace occupé par le circuit dans le FPGA à l'implémentation. L'algorithme est basé sur la concaténation de LUTs dans des LUT5 (se reporter à la documentation).

Fichier d'entrée	
	****.txt***
Fichier de sortie	
	*./constrs/new/**cell_combining.xdc***

Tableau 5 : Fichiers d'entrées/sorties du script CellsCombining.py

2.1.8 Générateur de banc de test : *testbenchGenerator.py*

Le script *testbenchGenerator.py* permet de générer le banc de test du nouveau circuit, généré par la redondance. Il prend en entrée le banc de test du circuit initial ainsi que le fichier Verilog extrait à l'exécution du second script TCL.

Fichier d'entrée	
	Top module décrivant les IO du circuit (*./sim/***.v***)
	Description fonctionnelle du top module (*./sources/***.vhd***)
Fichier de sortie	
	Nouveau Banc de test du top module (*./sim/new/***.vhd***)

Tableau 6 : Fichiers d'entrées/sorties du script testbenchGenerator.py

2.1.9 Script TCL 3

Le troisième script TCL permet de charger le fichier de contrainte généré par le script d'optimisation de l'espace dans le projet Vivado précédemment créé à partir de la netlist triplée (script TCL 2). Une implémentation est exécutée.

2.1.10 Script d'injection de faute : *TclGenerator_FaultInjection.py*

Ce script permet de construire le dernier script TCL (script TCL 4) en extrayant depuis le banc de test généré précédemment, les noms des signaux du circuit triplé pour construire les commandes d'injection de fautes selon la syntaxe du fichier Vivado.

Fichier d'entrée	
	Banc de test du circuit triplé (*./sim/new/***.vhd***)
	Emplacement du projet créé à partir d'une netlist (./***.xpr**)
Fichier de sortie	
	Script TCL 4

Tableau 7 : Fichiers d'entrée/sorties du script tclGenerator_faultInjection.py

2.1.11 Script TCL 4 (pour injection de faute)

Ce quatrième script TCL est le dernier script exécuté par le processus. Il a été généré par *TclGenerator_FaultInjection.py* et permet de lancer une simulation fonctionnelle post

implémentation et de vérifier que le circuit triplé est bien tolérant aux perturbations en injectant des fautes sur plusieurs signaux.

2.2 Automatisation

2.2.1 Script donneur d'ordre : **.cmd*

Tous les scripts sont exécutés depuis un script dit "donneur d'ordre". Le processus décrit a été développé dans un environnement Windows. Le script master est un fichier de commande qui exécute tour à tour les scripts python et TCL. Pour que l'utilisation de notre logiciel soit plus agréable, nous avons développé une interface graphique permettant à l'utilisateur de renseigner les fichiers nécessaires au fonctionnement du logiciel. C'est l'interface graphique qui va appeler le script donneur d'ordre et ainsi exécuter le flot.

La figure ci-dessous correspond au flot développé. Les scripts sont exécutés dans l'ordre suivant.

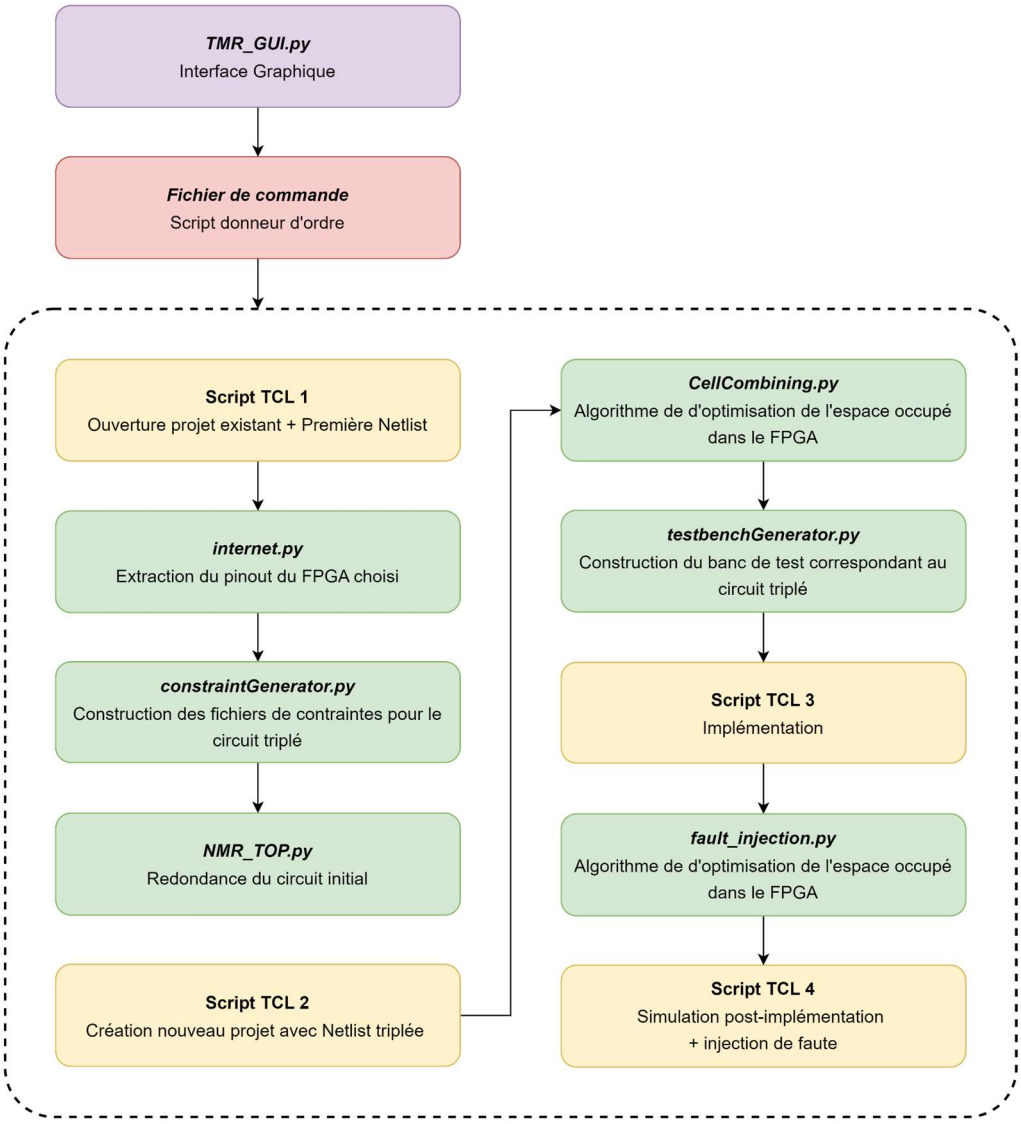

Figure 1 : Automatisation du processus

2.2.2 Interface graphique : *TMR_GUI.py*

Pour utiliser notre logiciel il faut utiliser l'interface graphique associée. L'intérêt pour l'utilisateur d'utiliser l'interface graphique est de renseigner l'emplacement de tous les fichiers nécessaires au bon fonctionnement de l'application sans ouvrir les scripts.
Pour pouvoir lancer une redondance sur un circuit il faut que l'utilisateur renseigne tous les champs demandés. Si un des champs n'est pas renseigné, un warning apparaît et la simulation n'est pas exécutée.

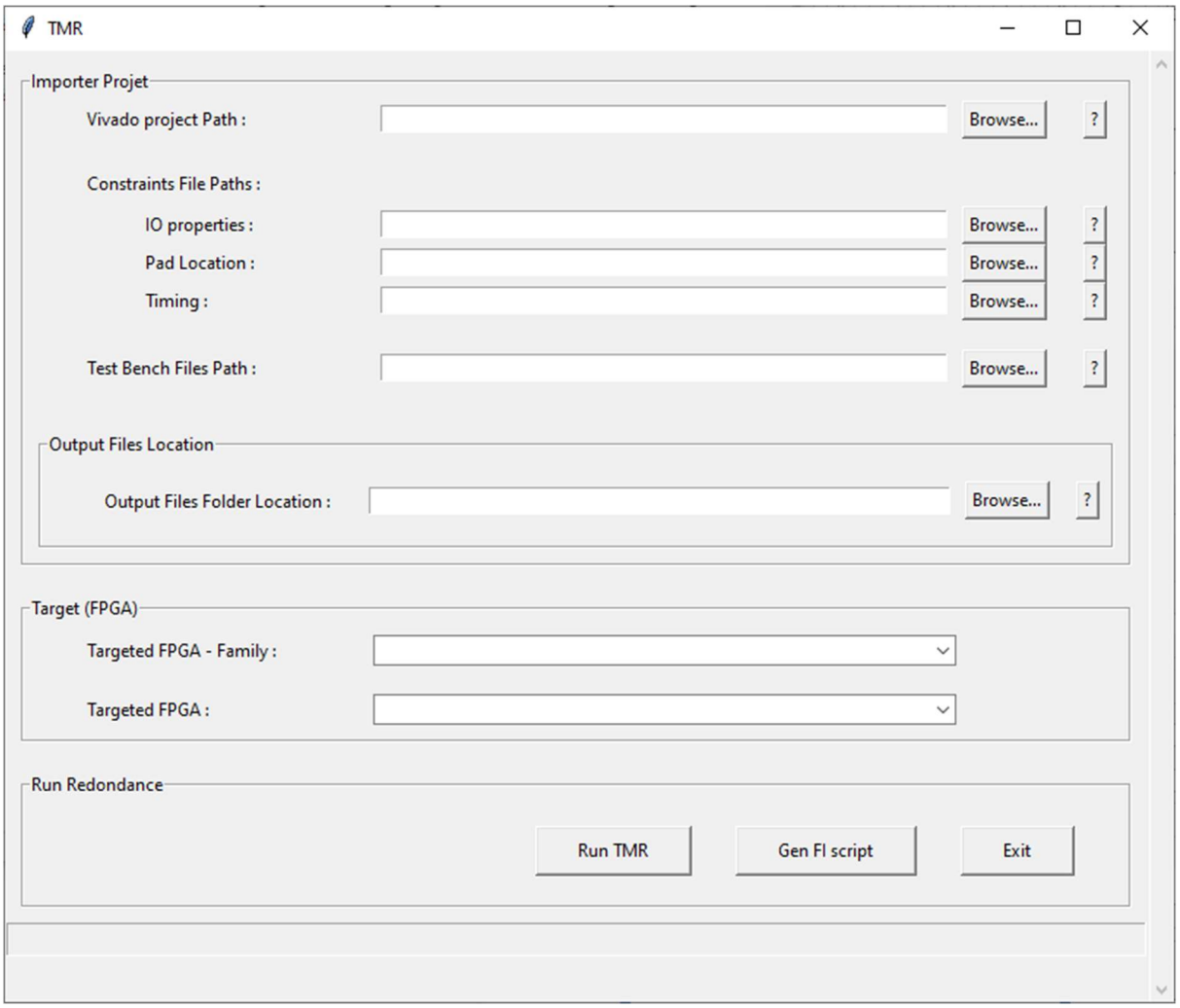

Figure 2 : Interface Graphique