Laboratoire TIMA

Équipe AMfoRS

Projet SysAx

Self-adaptative approximate SoC : HW platform development

Bref regarde sur la platforme PULP

Auteur :

Gabriel Villanova N. Magalhães

Conseiller :

Dr. Mounir BENABDENBI

Date : mars / 2018

Table des matières

Table des figures

1 PULP

1.1 Bref étude sur plataform PULP et caractéristique des hws

Dans ce section on va présenter l'objectif et recherche du projet PULP développé pour le labo ETH Zurich et Université de Bologna. En plus, on montre le type de hardware et le framework qu'ils offrent actuellement pour utiliser le projet.

Des objectifs et recherche, on souligne :

- arriver à 1GOPS/mW d'efficacité ;
- **scalabilité hw en efficacité d'energie** :
  plus hw or modif. hw pour plus d'efficacité d'energie.
- exploit le parallelism :
  multiple petit-cores dans un cluster ;
  partage de memoire avec le cluster.
- **cores simplifiés, mais efficace** ;
- optimization du hw ;
- accelerateurs dediés ;
- **extension d'isa pour des nécessités sw** :
  ajout de instructions (e.g. hw loops).
- projet *open-source*.

À partir de toutes ces idées et objectifs, un ensemble de travaux ont été développé et testés. En exploitant le repository du projet (`https://github.com/pulp-platform/pulp`), on peut voir que c'est un framework simples (essentiellement avec des ips et test benches), mais qui contient beaucoup de possibilités pour construire des hardwares différentes. La figure 1 montre toutes les ips disponibles.

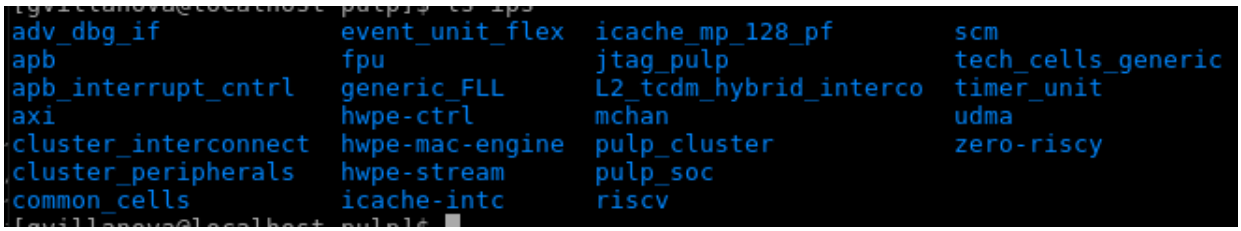

Figure 1 – IPs disponibles dans plataform PULP.

On peut voir que les ips des **cores** (riscv et zero-riscy) sont séparés, ainsi que les parties pour construir les SoCs, ça inclut : bus, memoires du type cache, périphériques, top-level niveau cluster, top-level niveau SoC, etc. Ça donne plusieurs possibilités de travailler dans ce repositoire.

Comme par exemple :

— exploiter **le single-core riscv** ;
— exploiter **le single-core zero-riscy** ;
— exploiter **le SoC avec riscv (8 cores in cluster, projet PULP)** ;
— exploiter **le SoC avec zero-riscy (8 cores incluster,projet PULP)**.

Pour comprendre la complexité des ces hardwares, on liste les caractéristiques des cores :

— **riscv or riscy** :
architeture 32 bits ;
4 niveau de pipeline ;
implementation d'ISA RVICMF.

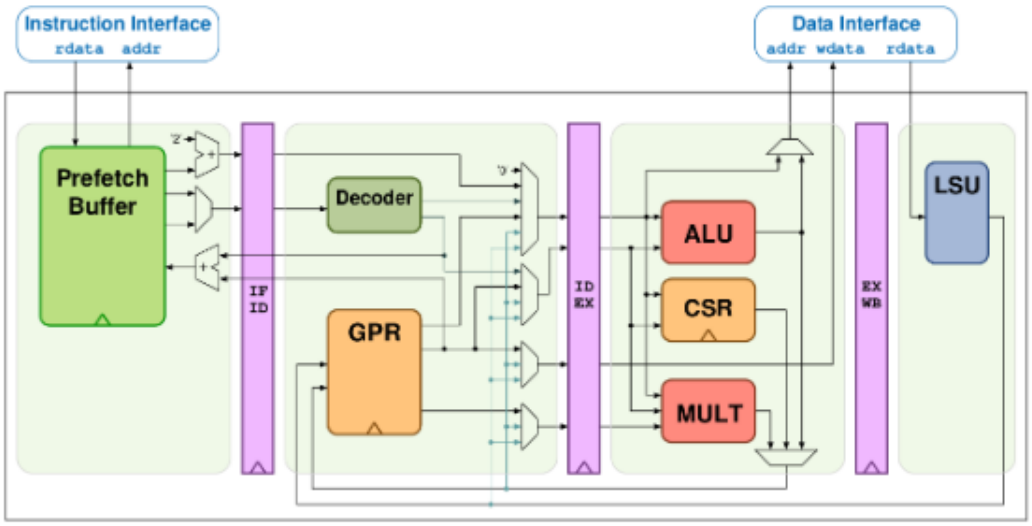

Figure 2 – Architeture générale RISCY.

— **zero-riscy** :
architeture 32 bits ;
2 niveau de pipeline ;
implementation d'ISA RVICME.

Le riscy et zero-riscy sont **cores stables** et ils ont une **toolchain espécifique** (aussi stable) pour création de **software** pour eux. La figura 3 (architeture aussi de 32 bits), montre le SoC qu'utilise ces cores, il utilise **8 cores** en travaillant ensemble dans un cluster, ça montre à-peu-près le niveau complexité du parallelism que le PULP forni. Enfin, **le SoC est stable** et **il n'y a pas de support pour le linux embarqué** (or similaire), par contre **supporte RTOS** (e.g. freeRTOS) pour être un **SoC du type microntroller**.

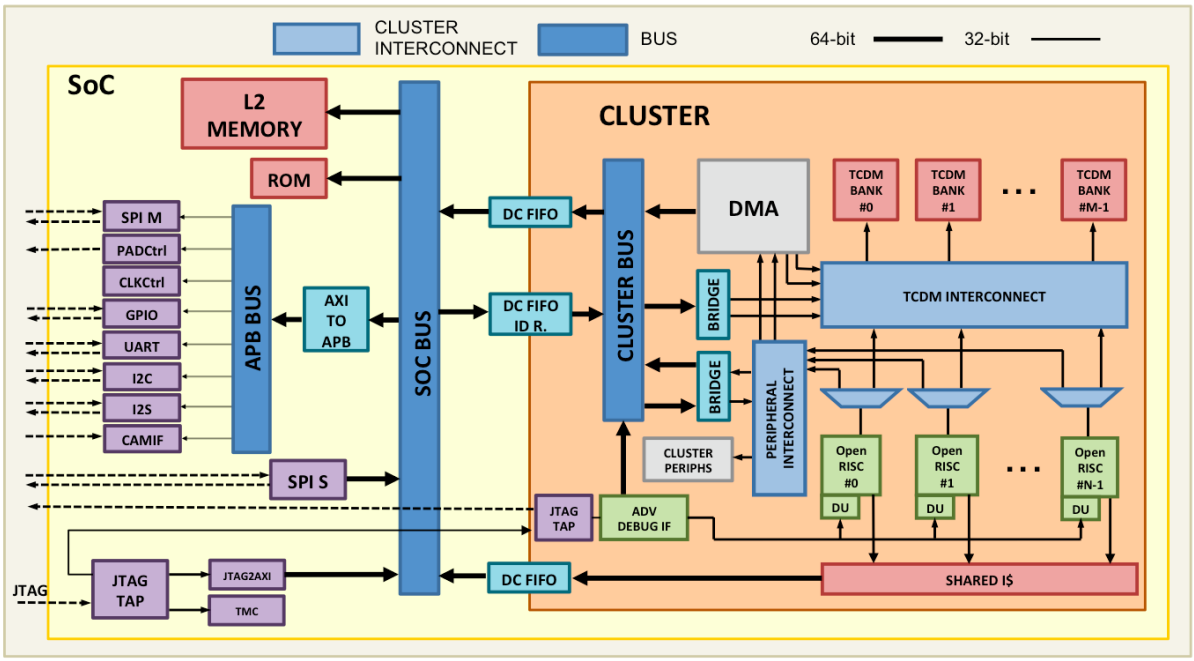

Figure 3 – Architeture générale du SoC PULP.

Il y a encore autres outils pour exploiter, e.g. **le pulp-sdk (*Software Development Kit*)** qui peut forni plus de possibilités pour exploiter cette platform.

1.2 Quelquels résultats

Ce section présenterai quelques **tests basiques avec le ip core riscv**, ça montrerait comme c'est **facile d'utiliser ce système et le porter pour l'utiliser dans un nouvelle projet**, comme par example, le projet SysAx. Une fois le core bien piloté pour nous et avec la compréhension de l'utilisation du SoC PULP, on peut avancer sur le sujet ***Approximate Computing*** autant que dans le single core comme dans le SoC complet, en permettant plusieurs types d'usage.

Dans ces tests on a utilisé les suivants softwares, sur le OS Fedora 24 :

— ModelSim Student Version 10.5b ;
— Vivado 2016.4 ;
— Verilator 3.890.

Pour des tests avec le ModelSim, une propre structure a été fait, en portent les sources su directoire /pulp/ips/riscv et en créant notre propre repository. La figure 4 montre l'organisation fait.

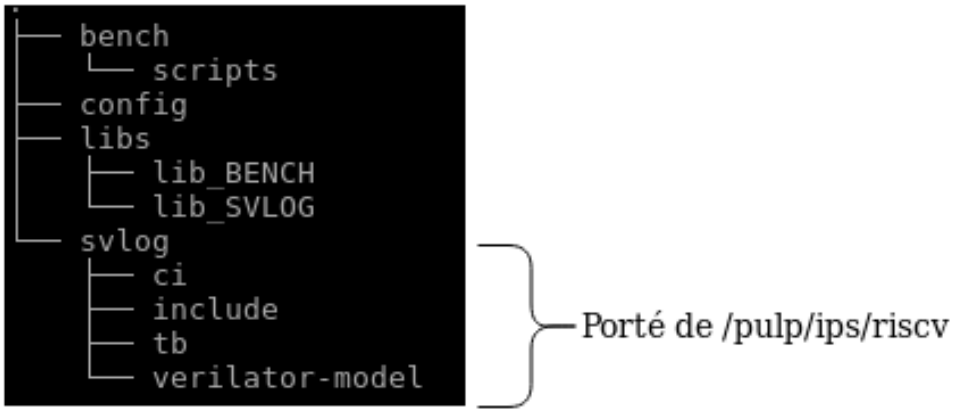

Figure 4 – Structure de simulation core riscy.

À partir de ça, on a fait un simples script pour compiler les sources du (*.sv) du ip core, en utilisant le ModelSim. On peut voir les sources et les commandes utilisés dans le code suivant :

```
vdel -lib  ${PATH_WORK}/libs/lib_SVLOG -all

vlib ${PATH_WORK}/libs/lib_SVLOG

vmap lib_SVLOG ${PATH_WORK}/libs/lib_SVLOG

vlog +incdir+./include -work lib_SVLOG ./include/apu_core_package.sv
vlog +incdir+./include -work lib_SVLOG ./include/apu_macros.sv
vlog +incdir+./include -work lib_SVLOG ./include/riscv_config.sv
vlog +incdir+./include -work lib_SVLOG ./include/riscv_defines.sv
vlog +incdir+./include -work lib_SVLOG ./include/riscv_tracer_defines.sv

vlog +incdir+./include -work lib_SVLOG riscv_L0_buffer.sv
vlog +incdir+./include -work lib_SVLOG riscv_fetch_fifo.sv
vlog +incdir+./include -work lib_SVLOG riscv_apu_disp.sv
vlog +incdir+./include -work lib_SVLOG riscv_alu.sv
vlog +incdir+./include -work lib_SVLOG riscv_alu_basic.sv
vlog +incdir+./include -work lib_SVLOG riscv_alu_div.sv
vlog +incdir+./include -work lib_SVLOG riscv_compressed_decoder.sv
vlog +incdir+./include -work lib_SVLOG riscv_controller.sv
vlog +incdir+./include -work lib_SVLOG riscv_cs_registers.sv
vlog +incdir+./include -work lib_SVLOG riscv_debug_unit.sv
vlog +incdir+./include -work lib_SVLOG riscv_decoder.sv
vlog +incdir+./include -work lib_SVLOG riscv_int_controller.sv
vlog +incdir+./include -work lib_SVLOG riscv_ex_stage.sv
vlog +incdir+./include -work lib_SVLOG riscv_hwloop_controller.sv
vlog +incdir+./include -work lib_SVLOG riscv_hwloop_regs.sv
vlog +incdir+./include -work lib_SVLOG riscv_id_stage.sv
vlog +incdir+./include -work lib_SVLOG riscv_if_stage.sv
vlog +incdir+./include -work lib_SVLOG riscv_load_store_unit.sv
vlog +incdir+./include -work lib_SVLOG riscv_mult.sv
vlog +incdir+./include -work lib_SVLOG riscv_prefetch_buffer.sv
vlog +incdir+./include -work lib_SVLOG riscv_prefetch_L0_buffer.sv
vlog +incdir+./include -work lib_SVLOG riscv_core.sv
```

Listing 1 – compil_SVLOG.sh

Le résultat de compilation n'a pas reporté des erreurs. Pour utiliser le core, on a porté le test-bench tb.sv forni, en compilant comme ça :

```
vdel -lib  ${PATH_WORK}/libs/lib_BENCH -all
vlib ${PATH_WORK}/libs/lib_BENCH
vmap lib_BENCH ${PATH_WORK}/libs/lib_BENCH

vlog -work lib_SVLOG tb.sv
```

Listing 2 – compil_BENCH.sh

Les codes du ip riscv, ainsi que du tb.sv sont recontré dans le repository pulp originale. Ce tb.sv utilise seulment l'unité riscv_alu_div.sv, comme ça, c'est une espèce de kick-off que la plataform nous donne pour exploiter et écrit des testes pour les autres unité et le core complet. Quelques formes d'onde de l'application sont montré dans la figure 6, en montrant le succès du test.

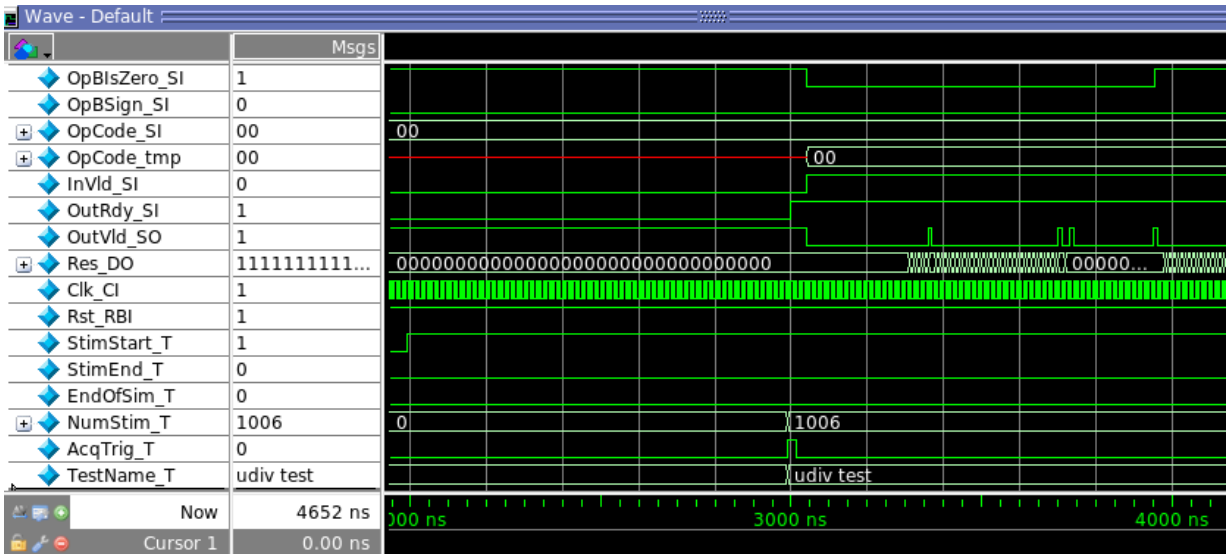

Figure 5 – Waves formes d'unité riscv_alu_div.sv.

Dans le dossier **verilator-model** est forni un test-bench pour le ip-core complèt écrit avec C++. En faisant juste un « $ make » sur ce dossier et on a des sorties de compilation imprimé sur le terminal ainsi que les waves formes.

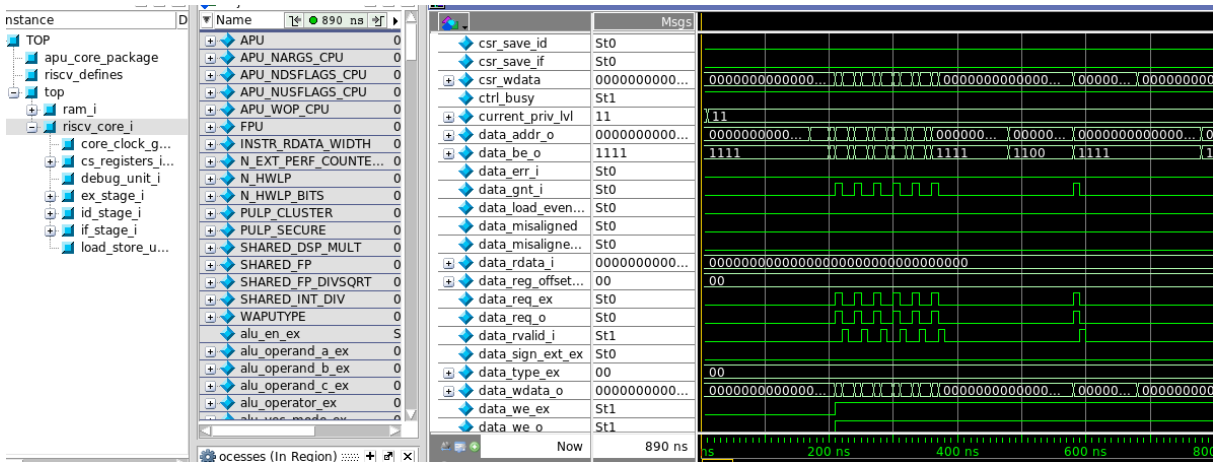

Figure 6 – Waves formes du processeur riscy.

Le « load » du program dans la memoire d'instruction est fait dans le code du test-bench.cpp. Donc, on peut le modifier pour vérifier autres functionalités du processeur.

Comme ça, on voit que le principale core de la platform pulp marche dans le niveau de simulation. Alors, on a aussi essayé de faire la synthèse, niveau RTL sur Vivado, et avec un peu de modification dans les codes, c'est possible de le synthetiser comme on peut voir dans la figure 7.

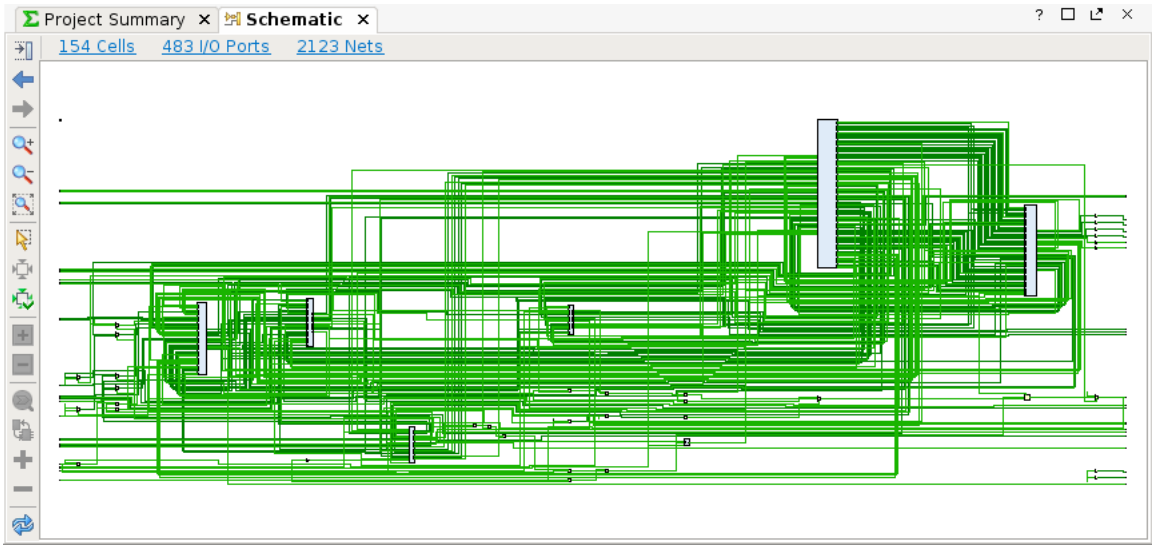

FIGURE 7 – Synthèsis niveau RTL du riscy sur Vivado.

1.2.1 Les possibilités des nouvelles tests

— Créer des sw avec la PULP-toolchain pour les cores et les re-simuler ;
— Simuler le SoC (cluster + interfaces) ;
— Créer des sw pour le SoC ;
— Synthetiser le SoC dans FPGA.

1.3 Résume des avantages et désavantages

Avantages :

- le projet PULP cherche efficacité d'energie ;
- les cores sont simples (32 bits, 4 or 2 niveau de pipeline), mais efficaces ;
- ils ont fait extension d'ISA, cette méthodologie peut-être utiliser pour mettre extension de l'approximate computing ;
- le sources du repositoire sont faciles de manipuler, comme ça c'est facile créer notre propres scripts dedans pour la simulation, tests et synthesis ;
- il y a beaucoup des ips I/O : i2c, gpio, uart, etc.
- il y a interface AMBA/APB dans le SoC, donc les projets chez PHELMA peuvent être fait avec ces interfaces pour qu'on puisse les utiliser à partir du SoC.

Desavantages :

- Le hw que on peut créer sont limités, puisque les possibilités dans la plataform Rocket est énorme ;
- Il n'y a pas de support pour le linux embarqué (ou similaire), juste RTOS (e.g. freeRTOS) pour être un SoC du type Microntroller ;
- Le repositoire n'a pas gros contribuitions, juste le basique pour la simulations des ips ;

Références

[1] https://riscv.org/wp-content/uploads/2015/06/riscv-pulp-workshop-june2015.pdf.

[2] http://iis-projects.ee.ethz.ch/index.php/PULP.