FPGA avec de l'IA:
Un exemple d'outils multi-fonctions pour le temps fréquence

Yann Le Coq (lecoqya@univ-grenoble-alpes.fr)

Carte Koheron Alpha 250



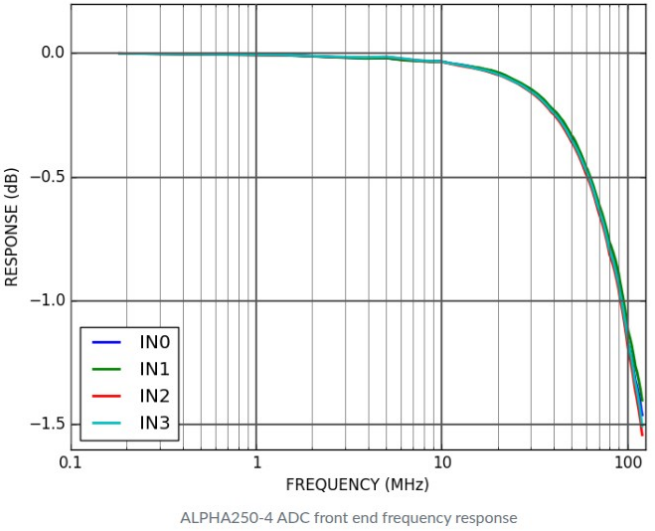
2 ADC rapide 14 bits + 2 DAC rapides 16 bits 250MSPS 2
+ SlowDAC 4 channel 16 bits (+Slow ADC 4 channels 24 bits) +...

Zynq 7020 (usage vivado illimité)
- dual core ARM A9
- 85K cells, 4.9Mb BRAM, 220 DSP48E1



ALPHA250 input to output latency

RF in→ out 90ns



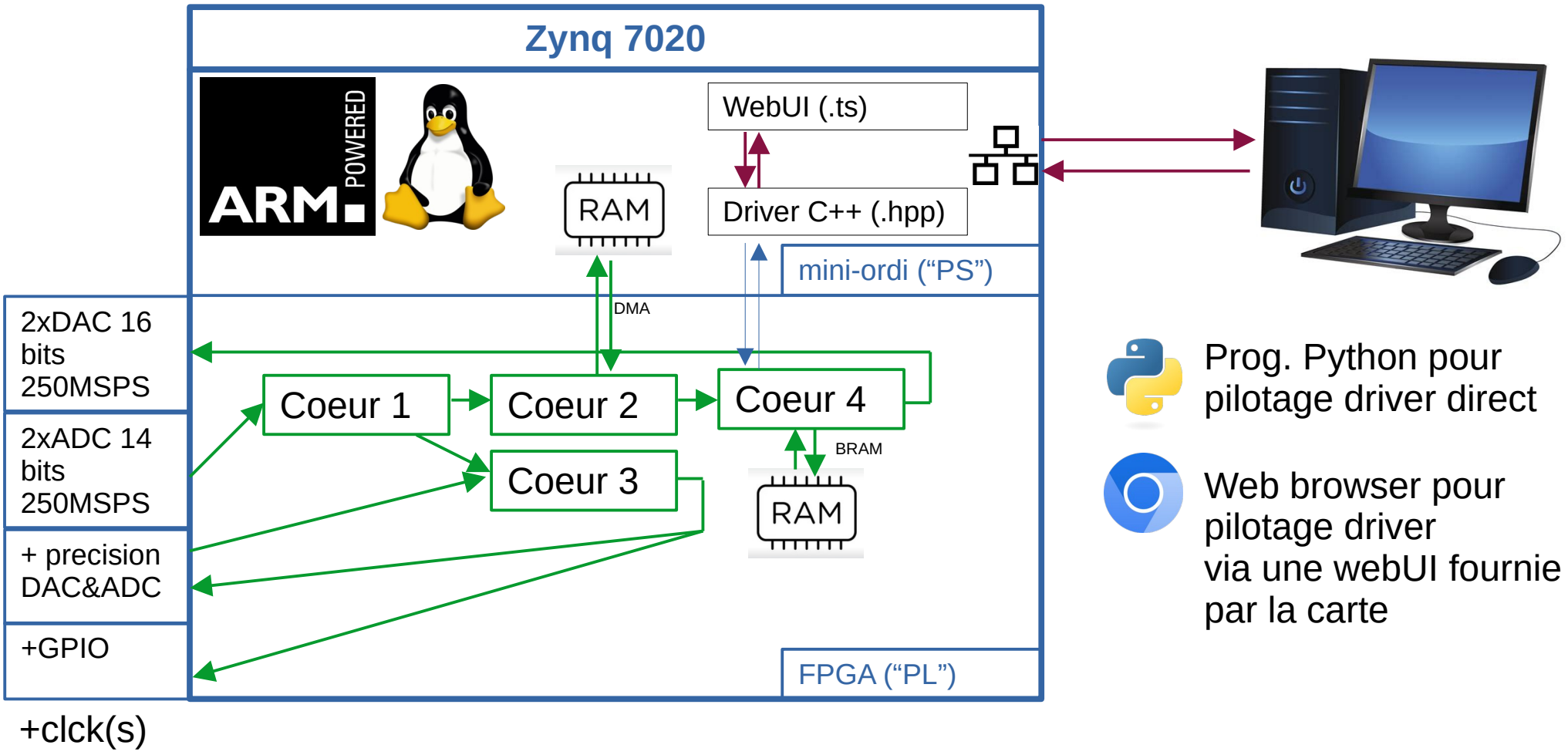
ALPHA250-4 ADC front end frequency response

>10MHz bandpass

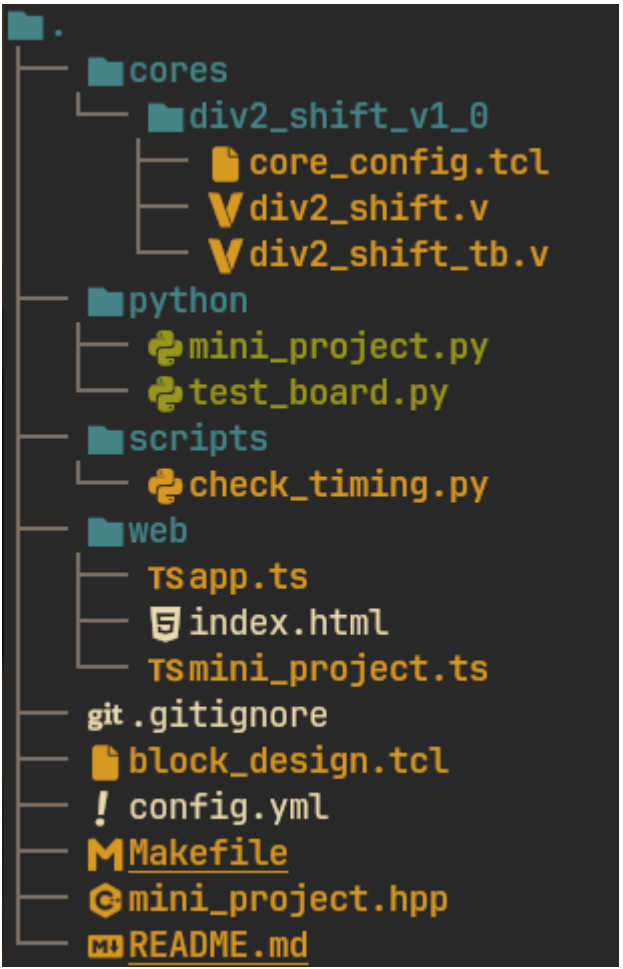
- Clock 10MHz int ou ext

- SDK Koheron, basé sur:
  . scripts .tcl pour piloter vivado
  . intégration “simple” de .v
  . contrôle en python et/ou web
  . ref design complet
  . exemples loopback,
    DMA
    BRAM
    FFT

Organisation d'un projet dans le paradigme du SDK Koheron



Organisation d’un projet dans le paradigme du SDK Koheron



File: block_design.tcl

```
# Block design mini_project
#
#   ADC0 --> div2_shift_0 --> DAC0     (identité ou division par 2, selon ctl div2)

# PS, horloges, ADC/DAC, registres de contrôle/status, SPI, XADC
source ${board_path}/starting_point.tcl

cell YLC:user:div2_shift:1.0 div2_shift_0 {
} {
  clk  adc_dac/adc_clk
  div2 [get_slice_pin [ctl_pin div2] 0 0]
  din  adc_dac/adc0
  dout adc_dac/dac0
}
```

```
module div2_shift (
    input  wire              clk,
    input  wire              div2,   // 0 : identité, 1 : division p
    input  wire signed [15:0] din,
    output wire signed [15:0] dout
);

    reg signed [15:0] din_r  = 16'sd0;
    reg               div2_r = 1'b0;
    reg signed [15:0] dout_r = 16'sd0;

    always @(posedge clk) begin
        din_r  <= din;
        div2_r <= div2;
        dout_r <= div2_r ? (din_r >>> 1) : din_r;
    end

    assign dout = dout_r;

endmodule
```

File: config.yml

```
---
name: mini_project
board: boards/alpha250
version: 0.1.0

cores:
  - fpga/cores/axi_ctl_register_v1_0
  - fpga/cores/axi_sts_register_v1_0
  - fpga/cores/edge_detector_v1_0
  - fpga/cores/comparator_v1_0
  - fpga/cores/unrandomizer_v1_0
  - boards/alpha250/cores/precision_dac_v1_0
  - boards/alpha250/cores/spi_cfg_v1_0
  - ./cores/div2_shift_v1_0

memory:
  - name: control
    offset: '0x60000000'
    range: 4K
  - name: ps_control
    offset: '0x64000000'
    range: 4K
  - name: status
    offset: '0x50000000'
    range: 4K
  - name: ps_status
    offset: '0x54000000'
    range: 4K
  - name: xadc
    offset: '0x43C00000'
    range: 64K
```

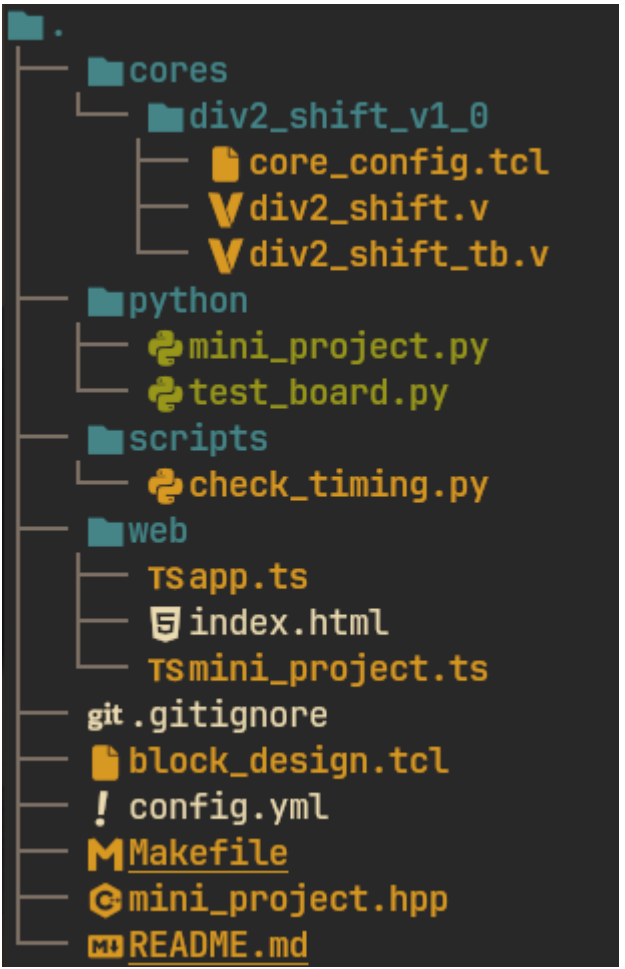
File: Makefile

```
# mini_project — build autonome qui délègue au SDK Koheron
#
#   make            : FPGA + serveur + web (équivalent à make all)
#   make sim        : simulation batch du core div2_shift (xsim), PASS/FAIL
#   make test_core  : même testbench dans le simulateur Vivado (GUI, cible du SDK)
#   make timing     : vérifie la fermeture de timing du bitstream routé
#   make run        : déploie sur la carte (HOST=...)
SHELL       := /bin/bash
SDK_DIR     := /home/yann/src/koheron-sdk
PROJECT_DIR := $(CURDIR)
BUILD_DIR   := $(PROJECT_DIR)/build
HOST        ?= 192.168.2.13

# make.py du SDK a besoin de jinja2, pyyaml et CppHeaderParser : on utilise l'env
# conda "koheron". Le SDK compose lui-même $(VENV)/bin/$(PYTHON), donc PYTHON doit
# rester un simple nom d'interpréteur.
PYTHON := python
VENV   := $(HOME)/miniconda3/envs/koheron

SDK_MAKE = $(MAKE) -C $(SDK_DIR) \
    CONFIG=$(PROJECT_DIR)/config.yml \
    TMP_PROJECT_PATH=$(BUILD_DIR) \
    PYTHON=$(PYTHON) \
    VENV=$(VENV) \
    VIVADO_PATH=/tools/Xilinx/Vivado \
    VIVADO_VERSION=2024.2
```

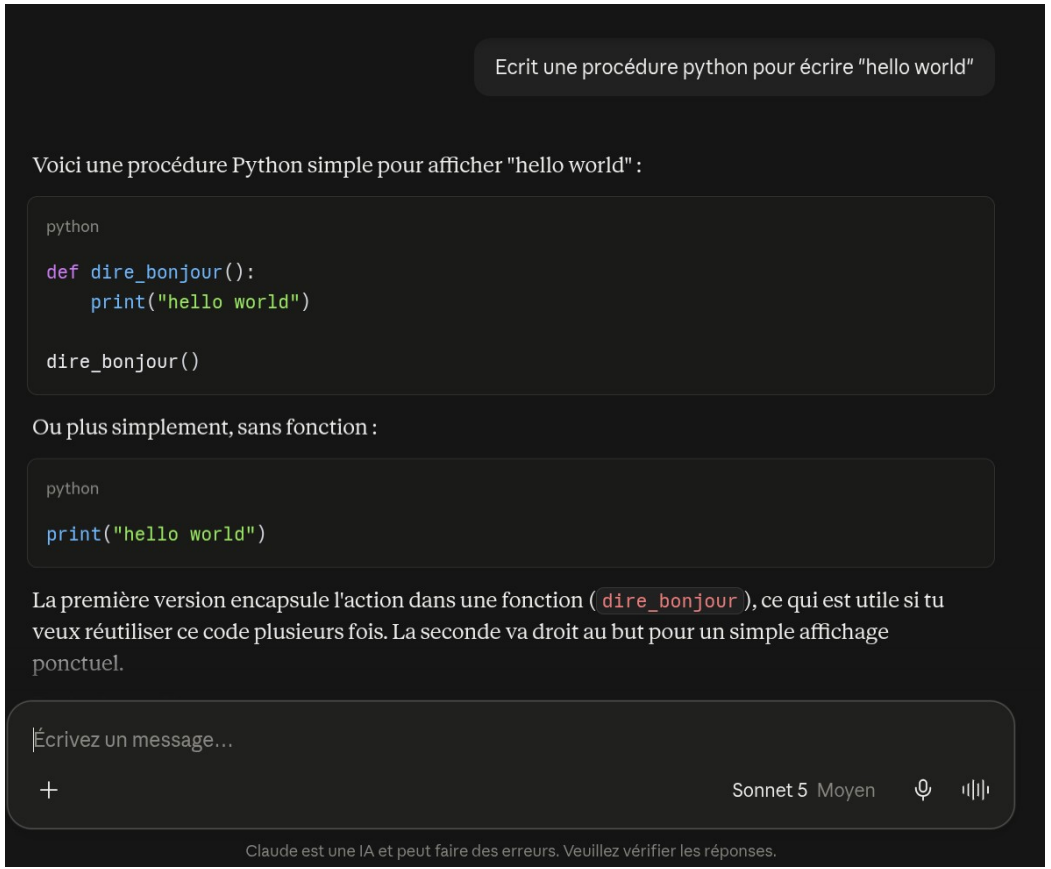
Workflow complet (recommandé)



1) écrire coeur en verilog (.v) + test unitaire (_tb.v)
2) simuler (xsim) et vérifier que ca fait ce qu'on veut
3) synthèse "hors contexte" → vérifier timing du coeur
   (corriger jusqu'à OK)
4) intégrer le coeur dans le projet,
   via config.yml et block_design.tcl
5) synthèse complète → vérifier timing
   (si nécessaire, corriger jusqu'à OK)
6) écrire/modifier le driver C++ (.hpp)
   pour accéder aux contrôles/status reg du design
7) écrire/modifier prog python
   pour commander/sortir les données via le driver
   → vérifier les bugs éventuels
8) écrire/modifier le webUI (.ts) et tests
9) profiter du fruit de son travail !

Note: interface graphique (IDE) Vivado n'est jamais lancée !
(on lui dit de faire des choses via scripts .tcl)

IA en chat bot vs codage agentique



modèle+ Harnais chat bot et outils intégrés
(cc/cv de texte, fichiers, peut etre container python pour tests,...)

Harnais sur le pc avec accès direct au pc (selon autorisations):
- explorer un répertoire et ses sous-repertoires
- lire un fichier (code, .md, .pdf,…)
- écrire un fichier (code, .md., docx,…)
- executer commande (prog, script,…)
- consulter le web (curl,…)
- avec extensions, pilotage direct de certains programes (ex.: chrome…)

exemple d'intérêt:
- aller chercher sois même la doc/exemples nécessaire (web et/ou pc)
- manipuler directement tous les fichiers nécessaires pour implémenter/modifier une fonctionnalité
- lire directement les erreurs de compilation/execution, et en déduire quoi corriger (y compris log synthèses vivado et problèmes de timing!)
- lancer des sous-agents dédiés à certaines taches

exemple de harnais: claude code, codex, vibe cli opencode, cline, hermess, openfox, pi.dev, …

exemple de modèles:
Claude Opus 5.5, Sonnet 5, GPT Sol, Luna
GLM 5.3, Kimi K4, DeepSeek v4, Qwen 3.8, Mistral,...

Quelques points

Points centraux :

- le LLM nu est sans mémoire: CLAUDE.md, AGENT.md (lus systématiquement en début de contexte) permettent mémorisation et facilitent grandement le travail de l'IA
  (les autres .md et les commentaires dans le codes sont aussi très utilisés quand l'IA tombe dessus)

- le contexte (conversation) est limité à 1M tokens (ou moins). Les CLAUDE.md et autres instructions ou outils du harnais sont également inclus dans ces 1Mt. Utiliser /compact ou /clear au moments opportuns. Garder les .md clairs, concis et cohérents...

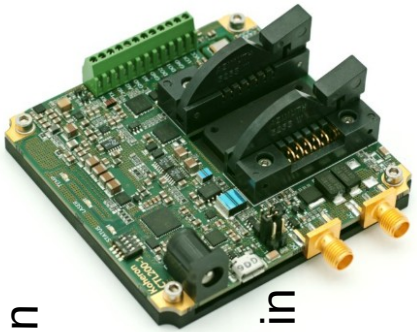
- les skills (.md et outils spécifiques à certaines taches) facilitent le travail de l'IA, surtout des modèles faibles (voire plus loin pour l'utilisation de modèles locaux)

Petites astuces :

- utiliser ~ 10 à 20% pour "faire le ménage" régulièrement. Exemple:
  "vérifie que CLAUDE.md est bien à jour, concis et sans duplication",
  "analyse en profondeur avec un œil neuf le code de xxx pour vérifier la cohérence (y compris noms de variables, noms de méthodes et commentaires), l'absence de code mort, l'emploi des bonnes pratiques de codage, la maintenabilité et l'absence de bug a priori". Etc.

- ne pas décrire un point problématique d'un interface graphique: faire plutôt un screen shot de l'endroit concerné dans le prompt (modèle multi-modal...)

Exemple: lock diode DBR Eblana sur REFIMEVE +compteur de fréquence, PSD et fonctions de xfer



+ Eblana
1.542µm
linewidth<100kHz

REFIMEVE

PD

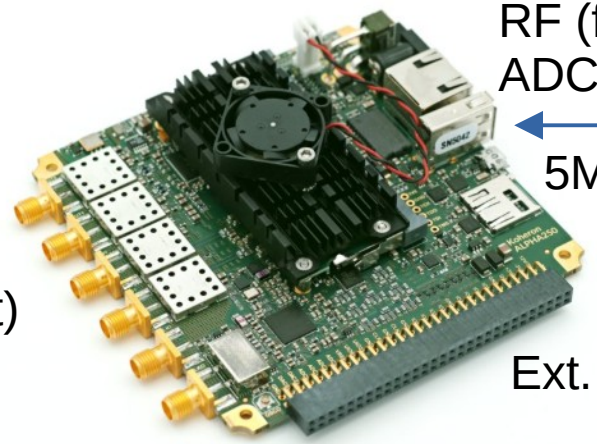
(Note: beaucoup d'inspiration/récupérations de https://github.com/jddes/Frequency-comb-DPLL du NIST, sur redpitaya il y a 9 ans...)

Temp mod in

Current mod in

RF (fast)
ADC

5MHz<Beatnote<120MHz



RF (fast)
DAC

10MHz

Ext. clock in

NTP

slowDAC

Loop Controller ADC0 ⇒ DAC0(P+I1+I2+I3) + SlowDAC0(I4)

Sign: +1 -1

Setpoint: 62,500000000 MHz Frac

Proportional Gain Shift: -2

1st Integrator Shift: -8

2nd Integrator Shift: -18

3rd Integrator Shift: -26

Phase Advance & Fine Gain G: 1.0

fz: 1000 kHz fp: 1000 kHz

4th Integrator Shift: -56 Inv

All Locks ON | All Locks OFF

ADC0 Range

12.4/14 bits

ADC0 FFT Spectrum

Avg: 1 N: 16K

δf ADC0 (wrt 62.500000 MHz)

Auto Y | Clear | Clear both

δf = -35.22 nHz RMS: 66.506 nHz (81)

DAC0 Corr

pp: 0.12
avg: -0.330

SlowDAC0

pp: 0.000
avg: 0.500

Loop Controller ADC1 ⇒ DAC1(P+I1+I2+I3) + SlowDAC1(I4)

Sign: +1 -1

Setpoint: 62,500000000 MHz Frac

Proportional Gain Shift: 0

1st Integrator Shift: -10

2nd Integrator Shift: -18

3rd Integrator Shift: -26

Phase Advance & Fine Gain G: 1.0

fz: 1000 kHz fp: 1000 kHz

4th Integrator Shift: -56 Inv

All Locks ON | All Locks OFF

ADC1 Range

4.5/14 bits

ADC1 FFT Spectrum

Avg: 1 N: 16K

δf ADC1 (wrt 62.500000 MHz)

Auto Y | Clear

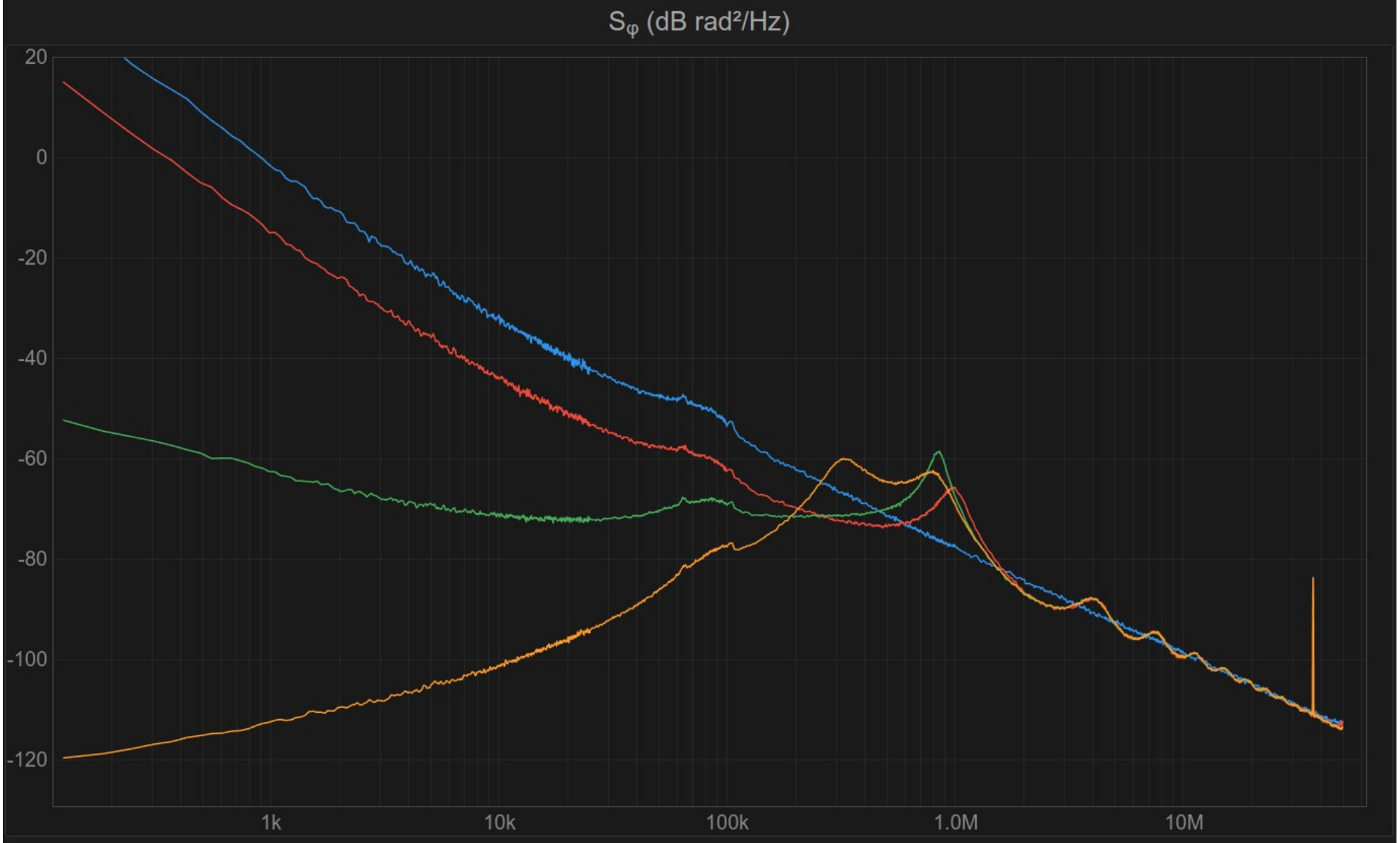
δf = -441.472 kHz RMS: 5.007 kHz (81)

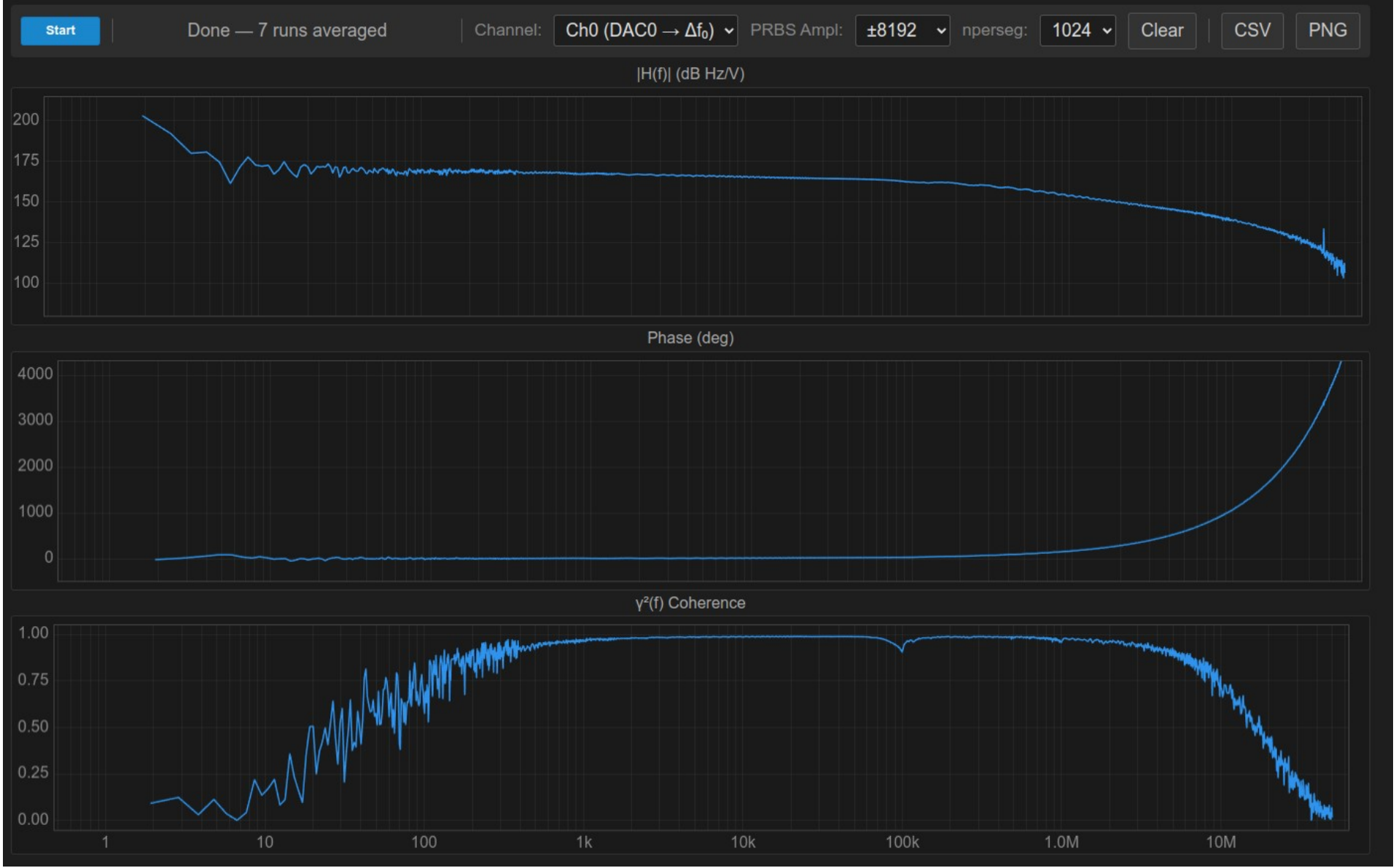
DAC1 Corr

pp: 0.00
avg: -0.000

SlowDAC1

pp: 0.000
avg: 0.500


S_φ (dB rad²/Hz)
20
0
-20
-40
-60
-80
-100
-120
1k
10k
100k
1.0M
10M


Start
Done — 7 runs averaged
Channel:
Ch0 (DAC0 → Δf0)
PRBS Ampl:
±8192
nperseg:
1024
Clear
CSV
PNG
|H(f)| (dB Hz/V)
200
175
150
125
100
Phase (deg)
4000
3000
2000
1000
0
γ²(f) Coherence
1.00
0.75
0.50
0.25
0.00
1
10
100
1k
10k
100k
1.0M
10M

Exemple: mesure timing PPS, correction sawtooth, lock OCXO sur GNSS

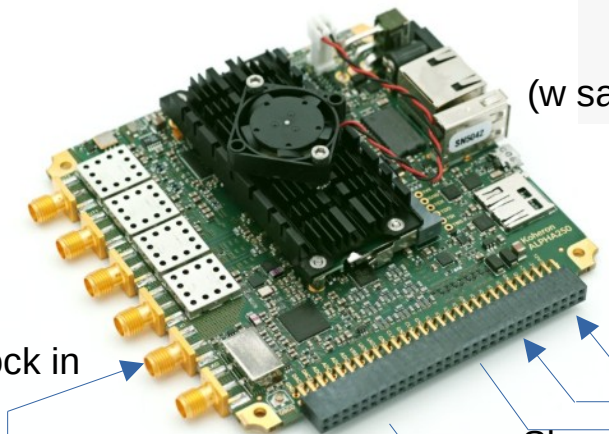




GNSS in

PPS
(w sawtooth)

XPPSOffset
(UART)



Clock in

(choix TCXO interne
ou OCXO externe)

GPIO

Slow DAC

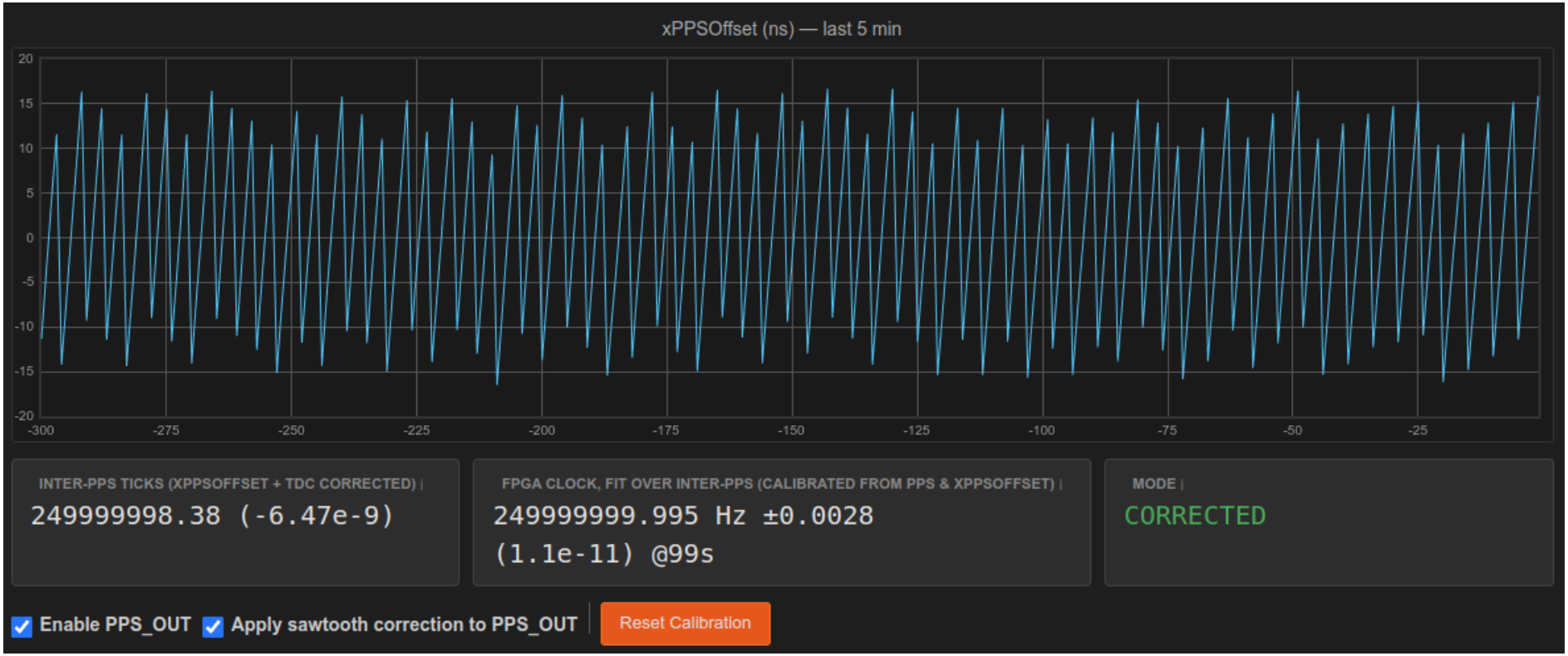
tune



10MHz out

PPS out
(corrected)

Exemple: mesure timing PPS, correction sawtooth, lock OCXO sur GNSS



- ✅ Mesure temps d'arrivée des PPS (wrt FPGA clock) avec TDC (~20ps résolution, auto-calibrée par densité de code)
- ✅ Génération pps corrigés
- ✅ Asservissement OCXO sur PPS_IN (au choix, PI3 ou Kalman)
- ✅ 2 canaux TDC input (pour comparaison horloges)

Pilotage intégral de campagnes de mesures

Prompt: « j’ai branché OCXO 10MHz libre sur clock in alpha250, horloge PRS10 10MHz sur ADC0, et PPS du mosaic X5 sur PPS_in. Il y a ce qu’il faut pour faire un three-cornered hat. Lances une telle campagne sur 12h, rapport intermédiaire toutes les heures. Pour le three cornered hat avec barres d’erreurs correctes, télécharge et installe SigmaTheta »

Prompt: « Les paramètres du filtre de Kalman pour l’asservissement du TCXO interne à l’alpha250 sur PPS_IN (corrigés de xPPSOffset) sont actuellement fixés via des hypothèses physiques relativement incertaines. Lance une campagne permettant d’estimer ceux-ci par Maximum Likelyhood Estimation. Donne le temps de mesure nécessaire estimé, si il dépasse 48h, reviens vers moi pour confirmation avant de lancer. Ensuite recommence un processus similaire pour l’asservissement Kalman de l’OCXO MV89A sur PPS_IN (toujours corrigé de xPPSOffset) »

Essais en IA locale

Problèmes de l'IA SOTA fermée, en cloud:
- accès, sécurité informatique
- propriété intellectuelle (comment être certain que son travail ne servira pas de données d'entraînement à des IA futures ?)
- prix
- souveraineté

Modèle open weight, quantifié pour exécution locale + harnais opensource
- exemple avec GPU 32Go VRAM :
  modèle : Qwen-3.8-27B (Q4 via NVFP4, KV Q8, 262kt contexte, DFlash2 + visio)
  harnais : pi.dev (open source, minimaliste)
  skill optimisé pour développement alpha250 avec koheron-sdk

"Dans le cwd, écrit un projet complet pour koheron alpha250 (utilise le skill koheron-sdk), tournant à 250 MSPS et générant sur DAC0 (16 bits) un signal sinusoidal de type NCO : un accumulateur de phase, et une lookup table qui transforme la phase instantanée en valeur sinusoidale de sortie. Quatre contrôles en temps réel, exposés à la fois en RPC standard et dans une interface graphique web : amplitude, fréquence, dérive de fréquence, et marche/arrêt.
- Amplitude : de 0 à la pleine échelle du DAC 16 bits (slider pour UI web).
- Fréquence : de 0 à la fréquence de Nyquist (125 MHz), commandée en Hz, résolution meilleure que 1Hz (contrôle numérique pour UI web).
- Dérive : linéaire, commandée en Hz/s. De -100Hz/s à +100Hz/s, quantum inférieur ou égal à 1mHz/s (contrôle numérique pour UI web)
La phase instantanée prend un incrément à chaque échantillon d'horloge : phase_new = phase_prev + inc ; inc = inc + inc_inc où inc est déduit de la fréquence en Hz et inc_inc (qui contiendra des bits fractionnaires sous le bit de poids faible de inc) de la dérive en Hz/s (coefficient de proportionnalité approprié). Justifie tes largeurs par la résolution et les gammes qu'elles procurent, documente et vérifie l'arithmétique fractionnaire.
Dimensionne au plus juste."

➡ Sur 10 runs, 100% de succès fonctionnel, après entre 1h50 et 4h20 en autonomie
(pour comparaison, ClaudeCode + Opus5: 75' sans skill, 40' avec)

Conclusion

1) Take-away: codage agentique complet avec sdk-koheron (probablement aussi tout autre approche basée sur des fichiers textes) mets le développement FPGA à la portée de tous.tes

2) on se concentre sa reflexion sur l'architecture globale, l'identification des vrais besoins utilisateurs et éventuellement de l'optimisation verilog (mais toujours avec l'aide de l'IA) des seuls (petites) parties archi-critiques

3) pilotage (+ interprétation de niveau très correcte) de campagnes de mesures complexes via IA est possible

3) problèmes à l'utilisation des modèles+harnais propriétaires fermés,
mais openweight+opensource ne sont pas très loin derrière et semblent utilisable à ce stade "dans la vrai vie" (à confirmer).
Auto-hébergement et/ou cloud souverain/employeur pour modèles les plus performants.
(note : ce type de service n'est pas fourni, aujourd'hui, par emmy/CNRS)

4) accessoirement: l'outils multifonctions présenté a vocation, à terme, à être partagé
(à discuter du comment avec FIRST-TF)