IN3160: HDL verifikasjon med Cocotb

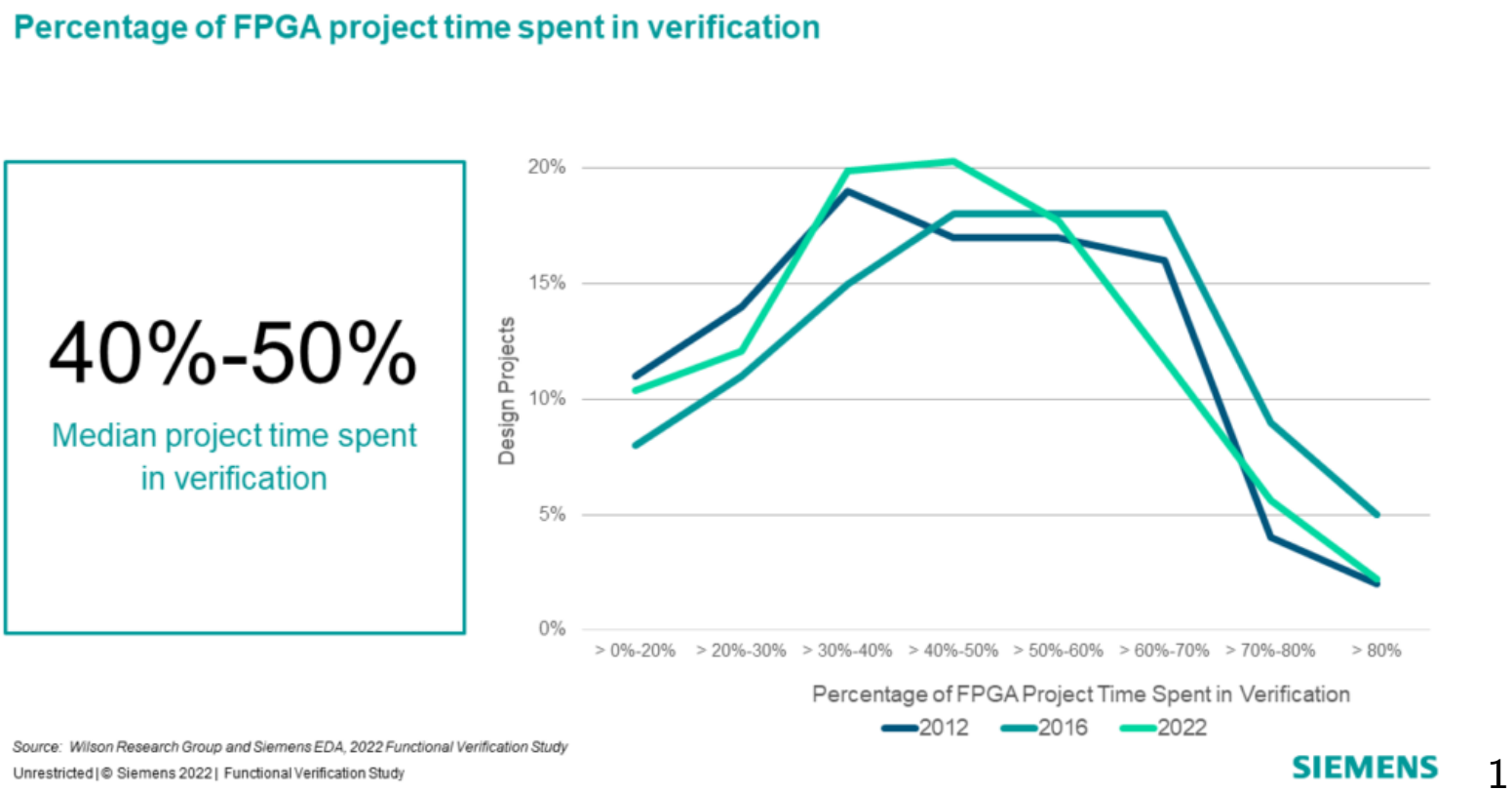
Alexander Wold

March 14, 2023

Agenda

- Verifikasjon og trender
- Hvorfor Python?
- Cocotb
- Live coding

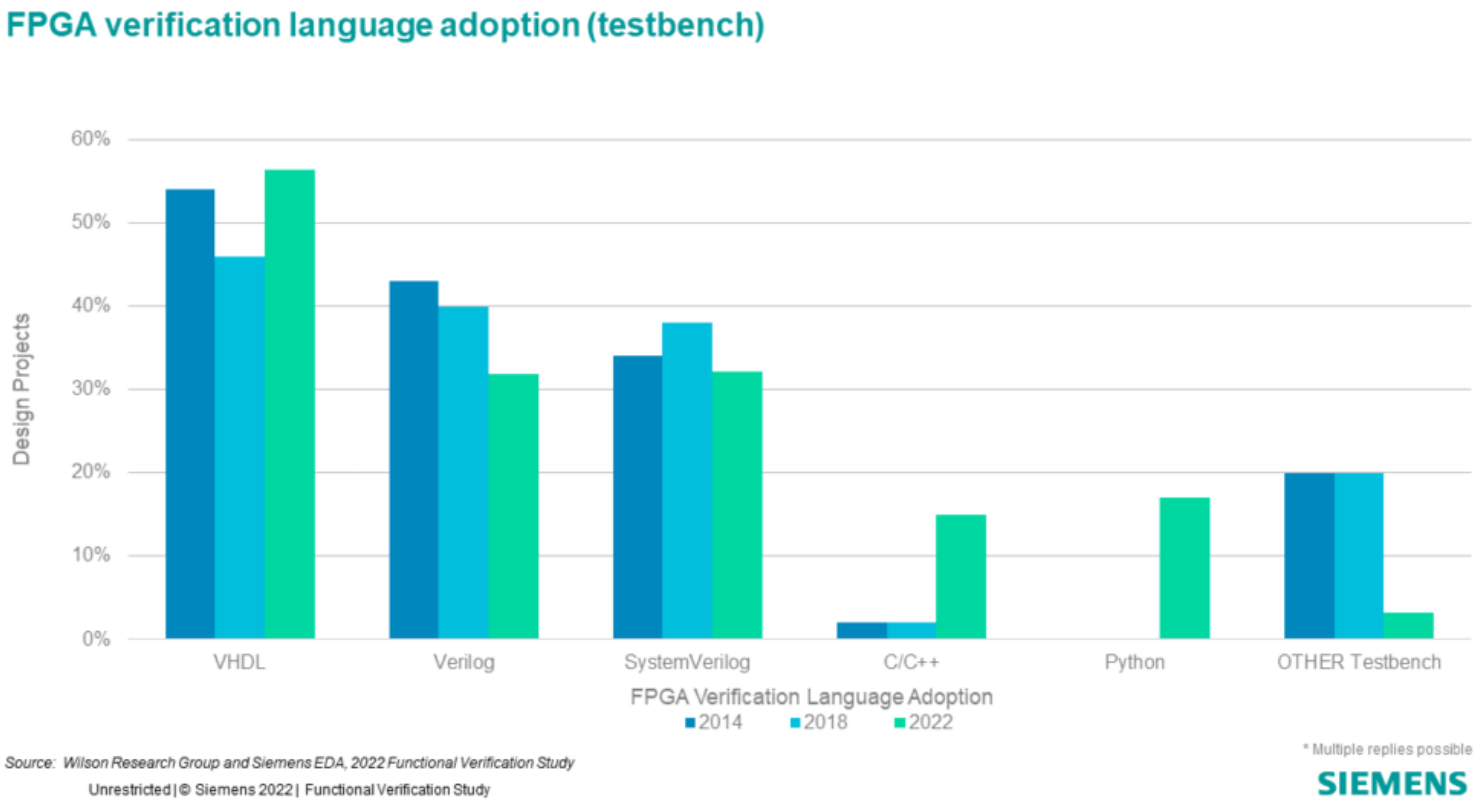
Hvorfor verifikasjon?



1

[1] https://blogs.sw.siemens.com/verificationhorizons/2022/10/30/part-3-the-2022-wilson-research-group-functional-verification-study/

Verifikasjon - språk?



2

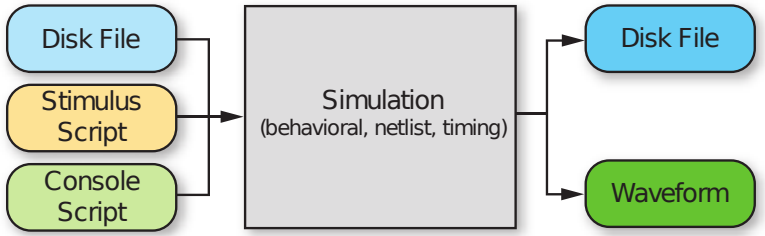
[2] https://blogs.sw.siemens.com/verificationhorizons/2022/11/21/part-6-the-2022-wilson-research-group-functional-verification-study/

Motivasjon for å skrive testbenker i Python

- Raskt å skrive kode
  - Python har god strenghåndtering
- Lett å interface med
- Mange eksisterende Python-biblioteker
- Enkelt å skille mellom software og hardware

Generell testbenk

- En testbenk er kode som utfører funksjonell verifikasjon av et HDL design.
- En testbenk:
  - Genererer stimulus til simuleringen
  - Påtrykker stimulus to DUT/UUT (device/unit under test)
  - Leser output fra DUT/UUT
  - Sammenlignet output med forventete verdier
  - Genererer en rapport
- Bruker modeller som emulerer oppførsel, for eksempel en oppførselsmodell av en krets.

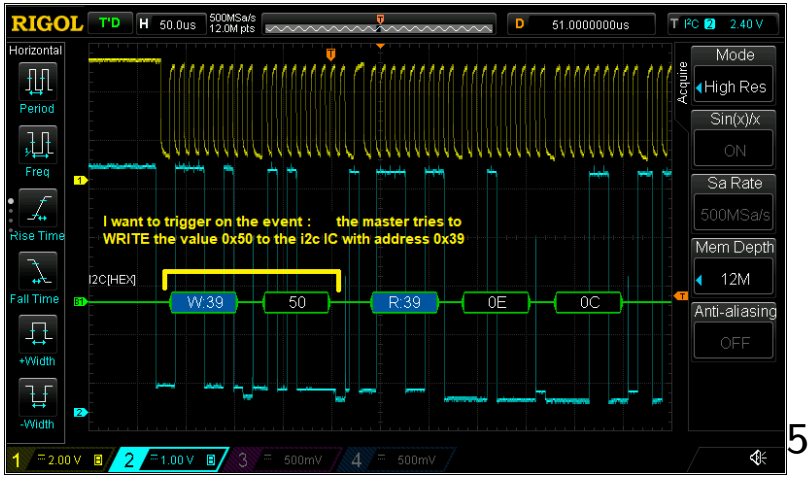


[3]

[2] Xilinx Xcell78

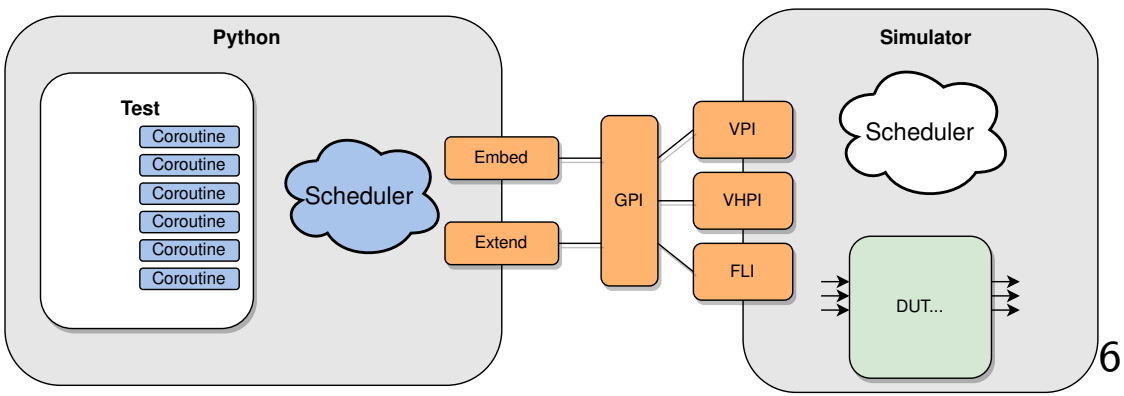
Validering av designet

- *Validering* av verifisert design implementert på FPGA
- Gjøres med scop eller logikkanalysator
- Sammenligner med beskrivelse i datablad (data, elektriske egenskaper, timing)
- Bruker scopets protokolldekoder til å dekode data og trigge på sekvenser
- Kan automatiseres ved å programmere scopet med SCPI[4](PyVISA)



Cocotb interface til simulator

- Simulatoren (GHDL/Modelsim) eksekverer Python som en plugin
- Dette skjer gjennom VPI/VHPI/FLI interfacet
- Cocotb Corutiner utføres uavhengig av hverandre
- Cocotb gir innsikt til signaler i designet under simulering



[3] https://docs.cocotb.org/en/stable/

Et enkel Cocotb-prosjekt

- Makefil
  - Valg av simulator, simulatorargumenter, generics til DUT
- Testbenk
  - Alle funksjoner med *@cocotb.test()*-decorator kjøres
  - Signaler aksesseres med *dut.navn.value*. Følger hierarket implementert i VHDL
  - *await* gir kontrollen tilbake til simulator
  - Har funksjoner som *Timer*, *RisingEdge*, *FallingEdge*
  - Logging: *debug*, *info*, *warning*, *error*, *critical*
    - *dut._log.info("Running test")*
- VHDL-kode

Eksempel på testbenk med python-genererte data

- Testbenk genererer og påtrykker sinus

Eksempel på selvtestende testbenk

- Kompleks multiplikator implementert i VHDL
- Testbenk genererer og påtrykker tilfeldige verdier
- Testbenk regner ut resultatet, og sammenligned med verdier fra DUT

Eksempel på testbenk med bilde

- Testbenken leser inn et bilde og sender til simulatoren
- VHDL-koden fjerner rødfargen
- Resultatet blir lagret til fil

Linker

2. SCPI
   https://www.rohde-schwarz.com/us/driver-pages/remote-control/remote-programming-environments_231250.html
- VHDL-kode

End

Test plan and coverage

- Test plan:
  - A document which describes required tests.
  - Testing is complete when the test plan is executed and all code executed
- Coverage is used to control and report verification progress and results
- Coverage types:
  - Code coverage
  - Functional coverage
  - Property coverage