Infineon Summer School
Functional Verification

Anul
2025

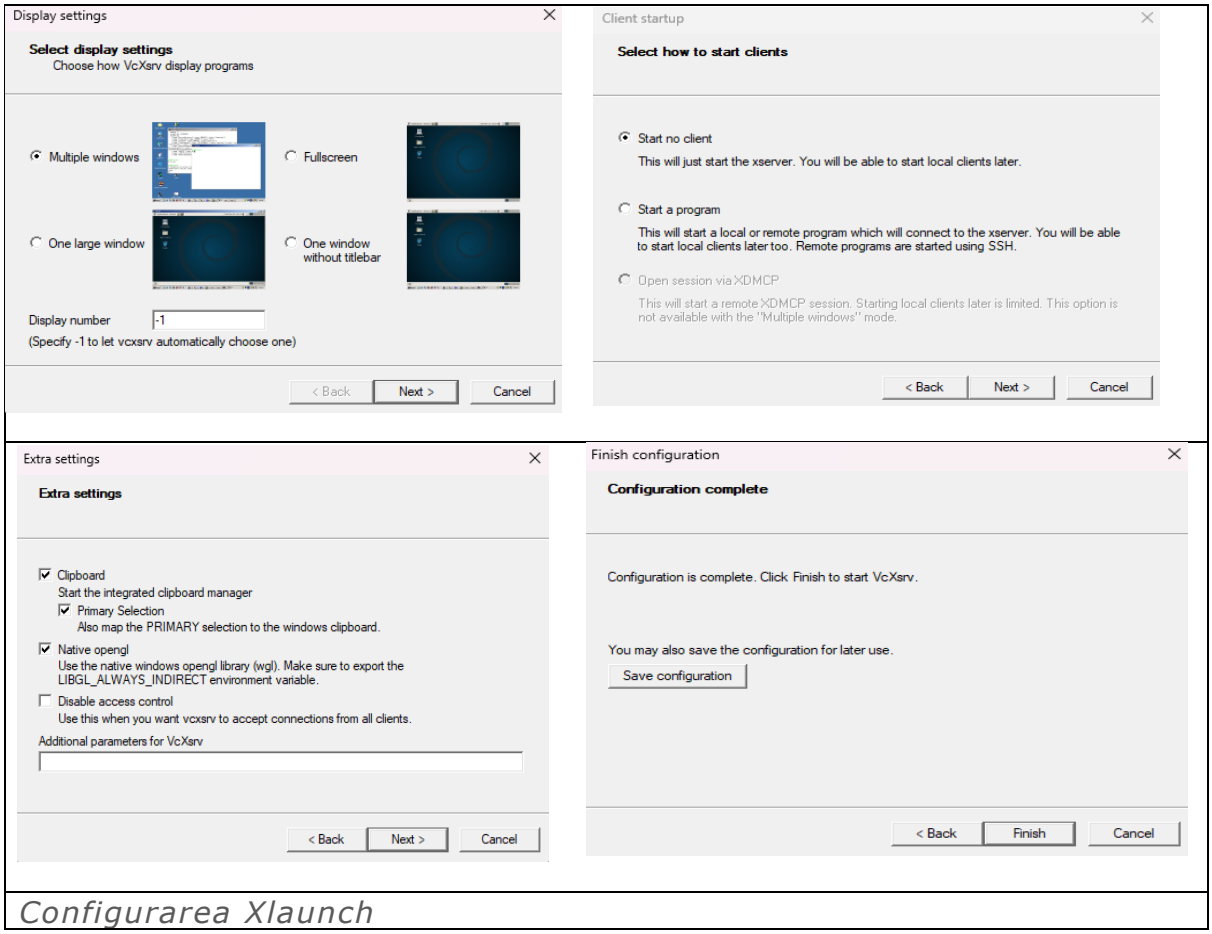
Cuprins

Introducere

Conectare la server

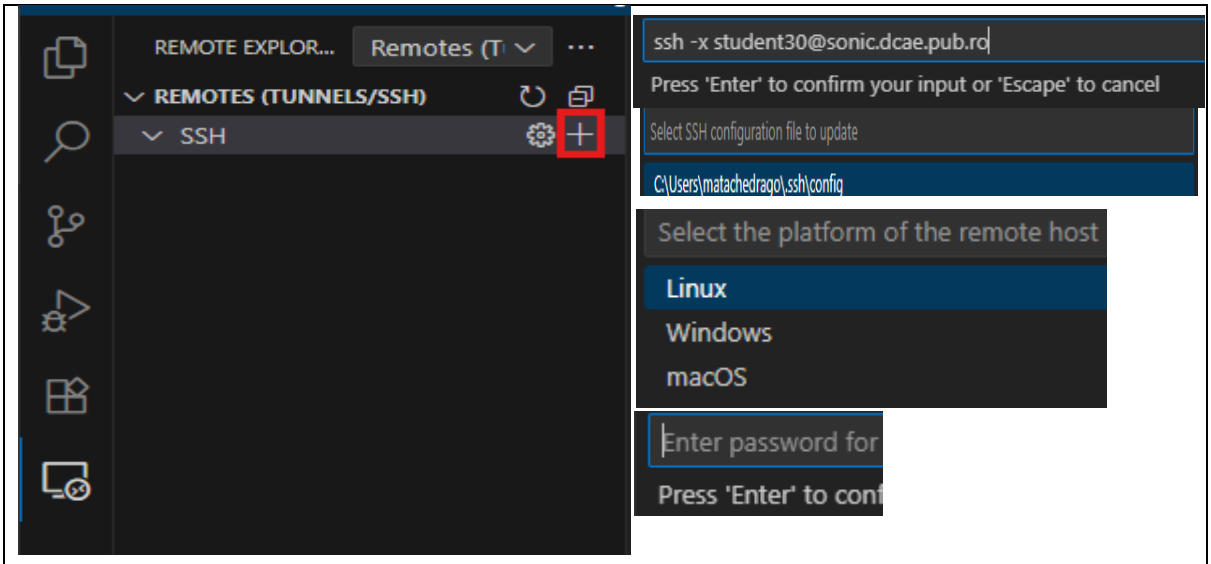
Acest document conține informațiile necesare pentru implementarea corectă și completă a aplicației de verificare funcțională din cadrul Infineon Summer School ediția 2025.

Pe toată durata desfășurării cursurilor se va folosi mediul de lucru disponibil la adresa ***sonic2.dcae.pub.ro*** sau ***sonic3.dcae.pub.ro*** în funcție de unde vi s-a spus ca este arondat userul primit. Pentru conectarea la server se va folosi o conexiune SSH. Pentru o experiență mai plăcută și intuitivă de lucru se recomandă instalarea VsCode impreună cu extensiile SystemVerilog - Language Support, EditorConfig, ToDo Tree și Remote - SSH disponibile in cadrul magazinului de extensii integrate in aplicație. De asemenea, dacă utilizați un sistem de operare Microsoft Windows este necesară și descărcarea aplicației Windows X Server din cadrul căreia veți porni programul XLaunch înainte de a porni conexiunea SSH.



Configurarea Xlaunch

Odată deschis VsCode va fi nevoie sa stabilim conexiunea SSH cu serverul precum in imaginile de mai jos.



Configurare conexiune SSH

Pentru numele de utilizator și parolă folosiți datele ce v-au fost transmise individual fiecăruia dintre voi.

Mediul de lucru și simulatorul

Pe durata acestui proiect se va utiliza IDE-ul VsCode cu conexiune SSH așa cum s-a arăta în capitolul de mai sus sau in cazuri excepționale nedit pentru a menține o performanță cât mai bună a serverului.

Pentru pornirea simularilor, în fereastra de terminal integrată in VsCode, se va executa următoarea comandă ***cd ~/ifro_summer_school_2025/simulation/sv_sim***(pentru verificare) sau ***cd ~/ifro_summer_school_2024/simulation/rtl_sim*** (pentru design). Odată executată această comandă consola se va afla in folderul destinat simulării si puteți porni simulatorul prin scrierea in terminal a următoarelor comenzi:

- Verificare:
  - ***./run.py -mode sim -test <nume_test> -seed <numar>*** .Scriptul run.py dispune si de alte posibilități ce le puteți afla prin apelarea: ***./run.py -help.***
  - **./run.sh <nume_test> <seed>**
- Design:
  - ***./run.sh <nume_testbench>***

Atenție execuția comenzilor de pornire a simulatorului trebuie făcută din folderele corecte: ***rtl_sim*** pentru ***design*** respectiv ***sv_sim*** pentru ***verificare.***

Simulatorul ce va fi utilizat este simulatorul comercial **CADENCE XCELIUM** pentru care exista un număr de 30 de licențe disponibile. Din acest motiv la încetarea sesiunii de lucru ***ESTE NECESARĂ INCHIDEREA SIMULATORULUI*** și utilizarea opțiunii de ***LOG-OFF***.

Coding style și debuging

Pentru o utilizare judicioasă a spațiului de lucru oferit de către fișierul în care scriem se recomandă setarea caracterului TAB la 4 pct și, dacă este disponibilă opțiunea, înlocuirea automată a caracterului TAB cu SPACE De asemenea, pentru a ușura munca de debug se recomandă utilizarea de mesaje în punctele cheie sau ori de câte ori se consideră ca fiind necesar prin utilizarea următoarei sintaxe:

`uvm_info(get_name(), $sformatf(„pattern_ca_in_c”), UVM_LOW)

E.X: `uvm_info(get_name(), $sformatf(„A=%0d”, a), UVM_LOW)

Pentru a crește gradul de productivitate și ușurința de debug se recomandă respectarea următoarelor reguli:

- Numele de variabile, funcții și taskuri vor fi semnificative și vor descrie modul de utilizare al acestora (ex: get_data_pkg , interrupt_cnt, etc)
- Pentru a crește lizibilitatea se recomandă utilizarea snake case pentru scrierea numelor de variabile si funcții (ex. get_data_pkg)
- Fiecare instrucțiune aflată in interiorul unui bloc ***begin end*** va fi indentată astfel încât codul să se afle în interiorul acestuia

```
E.X: if (interrupt_cnt == 2) begin
  execute_interrupt_chk();
  ->check_e_trigger;
end
```

Referințe teoretice

Pentru a ajuta în procesul de înțelegere al sarcinilor de lucru și în general al procesului de verificare se sugerează consultarea următoarelor linkuri:

- https://www.asic-world.com/systemverilog/index.html
- https://www.chipverify.com/tutorials/uvm
- https://verificationacademy.com/forums/all-topics
- https://cluelogic.com/2011/07/uvm-tutorial-for-candy-lovers-overview/

De asemenea, se recomanda si consultarea fișierelor din directorul ref_cards disponibil împreună cu mediul de verificare în care se vor găsi manualele de referință atât pentru UVM cât și pentru limbajul System Verilog.

Utilizarea chatbotilor (i.e. ChatGpt / Gemini / DeepSeek) este de asemenea o practică încurajată pentru lămuriri suplimentare însă este descurajată pentru rezolvarea cerințelor efective. Rețineți faptul ca orice model AI poate inventa explicații/ detalii legate de întrebarea voastră motiv pentru care în timpul laboratoarelor este mai bine să întrebați instructorul.

Descriera templateului

Directorul *ifx_summer_school_2025* conține următoarea structură:

1) **DOC**:
   a) ***Uarch***: conține documentul necesar pentru faza de design, el explicând modul de funcționare al acestuia și cerințele sale de implementare
   b) ***Ref_cards***: conține materiale conexe ce vă pot ajuta în înțelegerea linux, uvm și SystemVerilog
2) **Sim**:
   a) ***sv_sim***: director dedicat pentru scriptul de rulare al mediului de verificare, toate simulările de verificare funcțională vor fi pornite de aici
   b) ***rtl_sim***: director dedicat pentru scriptul de rulare al testbench-urilor, toate simulările din timpul fazei de design vor fi pornite de aici.
3) **Source**:
   a) SV: director ce conține toate sursele scrise in limbaj SystemVerilog

      i) rtl: director în care se vor crea toate fișierele sursă ale designului
         (1) Tb: director în care se vor crea toate fișierele de testbench

      ii) filter_tb: directorul rădăcină al mediului de verificare
         (1) Include: director în care se găsesc toate fișierele proprii mediului de verificare
            (a) Env: director ce conține fișierele sursă ale mediului de verificare
               (i) ifx_dig_checkers: fișier ce va conține verificările asupra semnalelor și valorilor ce se calculează
               (ii) ifx_dig_config: fișier folosit pentru configurarea mediului de verificare
               (iii) ifx_dig_coverage: fișier în care se vor scrie grupurile de acoperire funcțională
               (iv) ifx_dig_defines: fișier ce conține defineuri
               (v) ifx_dig_env: fișier în care vom crea componentele mediului de verificare
               (vi) ifx_dig_golden_model: fișier în care vom descrie funcțional comportamentul DUT-ului
               (vii) ifx_dig_pkg: fișier în care se vor adăuga fișierele mediului de verificare pentru a fi compilate
               (viii) ifx_dig_scoreboard: fișier în care se vor implementa comparațiile dintre valorile DUT-ului și cele ale modelului
               (ix) ifx_dig_testbase: fișier ce conține clasa test de bază

            (b) tests: director în care se află toate testele de verificare funcțională împreună cu pachetul de teste
               (i) ifx_dig_test_pkg: fișier în care vor fi adăugate toate testele ce se doresc a fi compilate

(c) seq_lib: director în care se află sequncerul mediului de verificare și în care se vor plasa dacă este necesar biblioteca de secvențe a mediului de verificare.

(2) Tb: director ce conține testbenchul utilizat pentru etapa de verificare funcțională și interfețele
   (a) ifx_dig_interface: fișier ce conține interfața ce se va folosi la comunicarea dintre DUT și mediul de verificare
   (b) ifx_dig_top: fișier ce conține testbenchul aferent modulului

Ziua 1

Discuții:

- Prezentarea domeniului verificării funcționale
- Prezentarea unui proces complet de verificare funcțională
- Demonstrație a unui mediu complet de verificare
- Scurtă recapitulare concepte SystemVerilog (porturi, instanțiere, tipuri de atribuire)
- Recapitulare sumară concepte POO(exemple folosind SystemVerilog)
- Prezentare sumară a sistemului de versionare Git

Sarcini de lucru:

i) În interiorul directorului *ifx_summer_school_2025* să se inițializeze un repository local și configurați repositoriul astfel încât sa permită realizareade commituri.

ii) Să se configureze/creeze fișierul *.gitignore* astfel încât directorul *simulation* și tot ceea ce include sa nu fie versionat.

iii) Creați un commit care sa includă toate fișierele. La mesajul de commit scrieți „Add initial project version”.

iv) Creati un branch nou pe care să îl numiți „Ziua-1” și mutați-vă pe acesta.

v) În interiorul testbench-ului(top) să se instanțieze o interfață de tip *”ifx_dig_interface”* numită dig_if și să se realizeze conexiunile dintre semnalele disponibile în interfață și cele disponibile în top.

vi) Să se instanțieze DUT-ul și să se realizeze conexiunile acestuia cu semnalele disponibile la nivelul top-ului.

vii) Să se scrie un task care permite configurarea modului de generare al ceasului de sistem în funcție de perioadă și unitatea de timp. Să se folosească acest task pentru a genera un ceas cu frecventa de 100 MHz.

viii) În directorul de teste să se creeze un fișier nou numit ifx_dig_hello_world.svh și să se includă acest fișier în interiorul pachetului *„ifx_dig_test_pkg.sv”*

ix) În fișierul nou creat *„ifx_dig_hello_world.svh”* să se scrie o clasă ce extinde clasa *uvm_test* și care implementează următoarea funcționalitate:

   (a) afișează mesajul „Start of test”,
   (b) realizează un reset ce ține 2 pulsuri de ceas,
   (c) așteaptă 500 us,
   (d) afișează mesajul „Hello UVM World!”,
   (e) așteaptă 500 us și încheie simularea.

Pentru implementarea cerințelor, la run_phase se va aduaga următoarea linie: **uvm_config_db#(virtual ifx_dig_interface)::get(this, "", "dig_if", dig_vif)** și se va folosi direct interfața, nu uitați că dig_vif este o variabilă ce trebuie declarată. Pentru scrierea acestui test puteți sa va inspirați din structura celorlalte teste disponibile în directorul de teste.

După finalizarea tuturor sarcinilor de lucru rulați o simulare urmând următorii pași:

a) Asigurați-vă că terminalul de comandă deschis se află în directorul *sv_sim.*

b) Rulați comanda: **./run.py -mode sim -test ifx_dig_hello_world -seed 1234**

c) Realizați probe pe toate semnalele disponibile la nivelul testbenchului și adăugați semnalele din interfețele disponibile pe forma de undă(waveform window) și semnalele din interfața DUT-ului.

d) Rulați testul

e) Observați textul din consola simulatorului și forma de undă. Discutați aceste observații.

f) Realizați operațiunea merge a branchului Ziua-1

Ziua 2

Discuții

- Structura de pachete (includes și imports)
- Instanțierea claselor
- Arhitectura generală a unui mediu de verificare
- Fazele de rulare UVM
- Agentul și componentele sale – privire de ansamblu

Sarcini de lucru:

i. Creați un nou branch in proiectul git numit Ziua-2
ii. În fișierul „*ifx_dig_regblock_pkg.svh*” să se realizeze includerea tuturor fișierelor necesare pentru blocul de registre.
iii. În fișierul „*ifx_dig_data_bus_uvc_pkg.sv*” să se includă componentele aferente UVC-ului.
iv. În fișierul „*ifx_dig_pkg.sv*” să se realizeze importul pachetelor aferente UVC-urilor.
v. În fișierul „*ifx_dig_pkg.sv*” să se realizez includerea tuturor componentelor mediului de verificare.
vi. În fișierul „*ifx_dig_registers.svh*” să se implementeze clasa aferentă registrului FILTER_CONTROL. Folosiți ca exemplu clasa de mai sus ce implementeaza registrele INT_STATUS.
vii. În fișierul „*ifx_dig_regblock.svh*” să se implementeze funcția *get_reg_by_name()* care primește ca parametru numele registrului căutat și întoarce obiectul aferent registrului. În cazul în care registrul nu poate fi găsit, funcția va afișa pe consolă un mesaj de avertizare și va întoarce null.
viii. Adăugați în cadrul tuturor fazelor de rulare din cadrul „*ifx_dig_testbase.svh*”, „*ifx_dig_env.svh*”, „*ifx_dig_scoreboard.svh”* mesaje care să permită urmărirea acestora.
Ex. `uvm_info(get_name(), „ENV build_phase done”, UVM_LOW)
ix. În fișierul „*ifx_dig_hello_world.svh*” modificați testul astfel încât să extindă „*ifx_dig_testbase*” și rulați testul. Observați noua ierarhie și log-ul de simulare.
x. Realizați operația de merge dintre branchul zilei curente si master/main

Ziua 3

Discuții:

- Agentul și componentele sale (aprofundare, folosind codul din ifx_dig_data_bus_uvc ca exemplu)
- Secvențe
- Teste

Sarcini de lucru:

i. Creați un nou branch numit Ziua-3
ii. În interiorul fișierului *„ifx_dig_testbase.svh”* să se implementeze taskul *drive_reset()* capabil să genereze intrare asincronă în reset și ieșirea sincronă din acesta. Durata resetului trebuie să fie configurabilă atât în timp cât și în ciclii de ceas.Lungimea minimă a unui reset trebuie să fie 3 pulsuri de ceas.
iii. În fișierul *„ifx_dig_pin_filter_uvc_sequence_lib.svh”* să se implementeze logica necesară taskului body pentru a genera un puls valid de semnal pentru a fi filtrat.
iv. În interiorul fișierului *„ifx_dig_data_bus_uvc_sequence_lib.svh”* să se scrie o secvență care să permită realizarea unui acces de citire de la o adresă specificată.
v. În directorul de teste să se creeze un fișier nou denumit *„ifx_dig_sfr_test.svh”* care să conțină o clasă cu același nume ce se extinde din ifx_dig_testbase.svh și implementează următorul scenariu:
    a. realizează un reset inițial al DUT-ului;
    b. scrie la fiecare registru valoarea patternului;
    c. citește valoarea tuturor registrelor; dă reset; schimbă patternul;
    d. reia scenariul de la pasul 2 până ce s-au folosit toate patternurile.

    Discutați între voi care ar fi patternurile necesare și argumentați de ce patternul propus ar fi necesar. Observați tranzacțiile pe de bus utilizând atât consola de simulare cât și forma de undă.
vi. În fișierul „ifx_dig_test_filter_rising.svh” să se modifice main phase astfel încât:
    a. Să facă drive la un reset de 5 pulsuri de ceas.
    b. Să genereze un puls valid pe intrarea unui filtru.
vii. Realizați operația de merge dintre branchul zilei curente si master/main

Ziua 4

Discuții:

- Secvențe
- Constrângeri și randomizare

Sarcini de lucru:

i. Creați un nou branch numit Ziua-4
ii. În interiorul *„ifx_dig_testbase.svh”* să se completeze taskul read_reg() care să permită utilizarea mai simplă a secvenței de read scrise anterior

iii. În interiorul „*ifx_dig_data_bus_uvc_sequence_lib.svh*” să se implementeze o secvența generică „*ifx_dig_data_bus_uvc_op_sequence*” capabilă să realizeze atât un acces de citire cât și unul de scriere. Această secvență va conține o constrângere soft in care dacă nu se specifică explicit accesul ca fiind de scriere se va face o citire de la o adresa specificată.

iv. Să se rescrie testul „*ifx_dig_sfr_test.svh*” astfel încât acesta să utilizeze funcțiile read_reg() și write_reg_fields(). Să se implementeze posibilitatea ca datele ce se vor scrie in registre sa fie aleatoare.

v. În fișierul „*ifx_dig_test_filter_toggle.svh*” implementați urmatorul scenariu:
   a. Anunță printr-un mesaj filtrul ce urmează a fi testat
   b. Configurează în mod aleator un filtru (întreruperea va rămâne dezactivată)
   c. Generează un puls valid pe intrarea acestuia
   d. Citește statusul
   e. Așteaptă 100ns după care trece să testeze următorul filtru până ce au fost testate toate.

vi. Realizați operația de merge dintre branchul zilei curente si master/main

Ziua 5

Discuții

- Porturi de analiză
- Golden model
- Scoreboarding
- Execuția paralelă a taskurilor

Sarcini de lucru:

i. Creați un nou branch numit Ziua-5

ii. În fișierul „*ifx_dig_env.svh* ” să se realizeze conectarea portului de analiză al monitorului agentului de date la cel disponibil în scoreboard. Rulați unul dintre testele funcționale și observați ce modificare apare în log-ul de pe consolă.

iii. În fișierul „*ifx_dig_scoreboard.svh*” adăugați în faza de build un apel la uvm_config_db astfel încât variabila dig_vif să fie populată.

iv. Tot in fișierul de la punctul iii să se scrie un cod ce să permită execuția în paralel a taskurilor golden_model , do_checkers, collect_coverage

v. În fișierul „*ifx_dig_golden model*” completați taskurile *model_data_out* și *model_interrupt* astfel încât acestea sa emuleze comportamentul descris in specificația modulului.

vi. Rulați iarăși testele disponibile. Se observă vreo diferență ?

vii. Realizați operația de merge dintre branchul zilei curent si master/main

Ziua 6

Discuții:

- Checkere
- Acoperire funcțională

Sarcini de lucru:

i. Creați un nou branch numit Ziua-6
ii. În fișierul *„ifx_dig_interface.svh”* implementați asserții care să permită verificarea faptului că semnalele de intrare au valoare de reset corectă atunci când se încheie resetul.
iii. Completați funcția check_read_data() astfel încât aceasta să verifice valoarea citită din DUT cu cea existentă in modelul de registre. În cazul unui acces la o adresă la care nu se află niciun registru se va verifica ca s-a citit valoarea 0.
iv. În cadrul fiecărui checker existent in fișier adăugați un mesaj care să indice faptul că acesta a pornit/repornit.
v. Rulați iarăși unul dintre teste. Ce modificări observați ? Cum rulează checkerele ? Există vreo problemă ? Dacă da, propuneți o soluție pentru aceasta.
vi. În fișierul *„ifx_dig_coverage.svh”* scrieți un covergroup care să demonstreze că s-a citit statusul de întrerupere 1 pentru fiecare filtru indiferent de tipul de filtru ales.
vii. Adăugați alte covergroupuri suplimentare la care vă gandiți.
viii. Completați covergroupurile deja existente cu alte crossuri pe care le considerați necesare.
ix. Adăugați alte checkere pe care le considerați ca fiind necesare.

Ex:

```
covergroup my_covergroup@(coverage_e);
        data_val_label: coverpoint data { bins cifre= {[0:9]};
                                          bins numere= {[10:99]}; }

        addr_val_label: coverpoint addr { bins low_addr= {[0:25]};

                                          bins high_adrr= {[26:99]}; }


        addr_x_data: cross data_val_label, addr_val_label;


endcovergroup
```

Ziua 7 -> 9

Discuții:

- Debug pe cod
- Clarificări adiționale
- Implementare teste/functionalități adiționale