

המעבדה ל - VLSI

חוברת הדרכה על כלי SYNOPSIS

הדרכה על ביצוע סימולציה וסינתזה עם Design Kit TOWER TSL 018

גואל סמואל, אמנון סטניסלבסקי

התחברות ממחשבי Linux-PC

זאת הדרך המומלצת לעבודה. פשוט יש לבצע login לתחילת העבודה. ניתן לפתוח חלון טרמינל באמצעות Applications→System Tools→Terminal

User Manual : ניתן לראות את כל הדוקומנטציה של כלי Synopsys ע"י הרצת הפקודה:

vcs:

```
acroread /eda/synopsys/2020-21/RHELx86/VCS_2020.12/vcs/R-2020.12//doc/UserGuide/vcs_docs_2020.12/PDF/vcs.pdf
```

designWare:

```
acroread /eda/synopsys/2020-21/RHELx86/SYN_2020.09-SP2/dw/doc/manuals/dwbb_userguide.pdf
```

כתיבה ב Systemverilog

ראשית יש לעבור לספריית העבודה :

```
cd Project
```

ולהפעיל את סביבת העבודה :

```
start_veride
```

ולהתחיל ברישום הקוד.

אם הכלי לא עולה (למשל אחרי יציאה לא מסודרת מהכלי), יש לבצע את הפעולות הבאות :

```
/bin/rm -r ../WS
```

```
mkdir ../WS
```

ואז להפעיל את הכלי מחדש : start_veride.

כתיבה ב - VHDL

לפני תחילת העבודה יש לבנות את הסביבה הנכונה. העתק את קבצי האתחול:

```
cp -r /users/iit/synopsys/tsl018_sim_2021 tsl018_sim_2021
```

עבור לספריית tsl018_sim_2021 ורשום קבצי ה-VHDL בספרייה זאת:

```
cd tsl018_sim_2021
```

1. סימולציה vcs

בעזרת הסימולטור vcs ניתן לבצע סימולציות של VHDL, verilog ו-systemverilog. בין היתר, הכלי מאפשר ביצוע הפעולות הבאות :

- הצגת והשוואה של צורות גל
- הצגת drivers
- הצגת סכמות ומסלולים
- ביצוע של פקודות UCLl/Tcl
- קביעת breakpoints
- ביצוע סימולציה בצעד בודד

1.1 סימולציית - SystemVerilog / Verilog

ניתן לראות tutorial בסיסי ב:

<https://www.youtube.com/watch?v=UgtC9S1-yQM>

הרצת הסימולציה מורכבת משלשה שלבים : elaborate ,analyze ופעלת ה-gui כדלקמן:

עבור שפת SystemVerilog יש להוסיף את האופציה -sverilog לפקודה הקומפילציה.

```
vlogan -kdb -sverilog -full64 file1.v file2.v -- Analyze
vcs -kdb -debug_access+all -full64 top_module -- Elaborate
simv -gui
```

בשורה הראשונה יפורטו שמות קבצי התכנון (כל שמות הקבצים המכילים את הבלוקים ותתי הבלוקים). פקודת ה-vcs תבצע אלבורציה, יש לפרט את שם ה-`top_module` כפי שמופיעה בקובץ ה-`top level` של התכנון.

יש להוסיף +v2k לשורת vlogan במידה וקוד ה-Verilog נכתב בתקן 2001.

(עבור עבודה עם 64 ביט יש לפנות לצוות המעבדה).

במידה והקוד הוא כולו Verilog, ניתן להריץ את שלשת השלבים בעזרת שתי הפקודות:

```
vcs -kdb -sverilog -debug_access+all -full64 file1.v file2.v
simv -gui
```

הסבר על הממשק הגרפי מופיע בסעיף הקודם.

Testbench

ב-systemverilog מקובל לבצע סימולציות באמצעות רכיב עזר (testbench) שמספק את אותות הכניסה לתכנון. בדוגמה הבאה, flop הוא התכנון אותו יש לבדוק:

```
module flop_test;
  logic data, clock;
  flop f1 (clock, data, qa, qb);

  initial
  begin
    clock = 0; data = 0;
    #10000 $finish;
  end

  always #100 clock = ~clock;
  always #300 data = ~data;

endmodule
```

```
module flop (clock, data, qa, qb);
  input clock,data;
  output qa, qb;

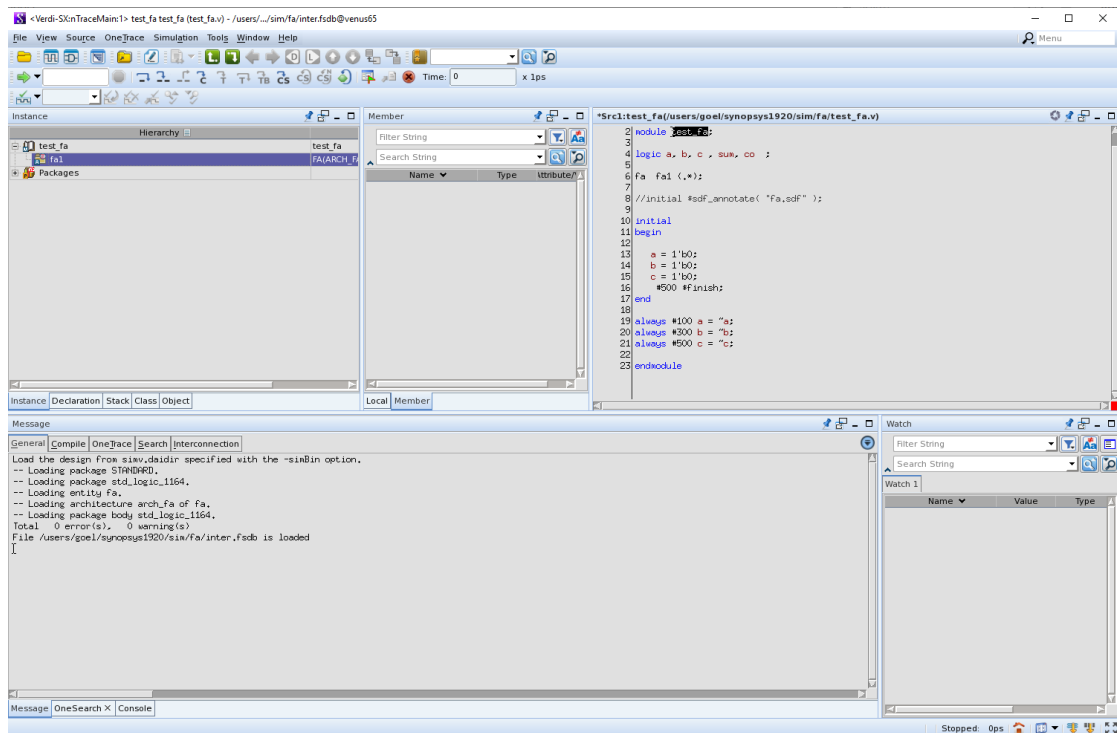
  nand #10 nd1 (a, data, clock),
  nd2 (b, ndata,clock),
  nd3 (qa, a, qb),
  nd4 (qb, b,qa);
  mynot nt1 (ndata, data);
endmodule

module mynot (out, in);
  output out;
  input in;
  not(out,in);
endmodule
```

ניתן להפעיל את הסימולציה באמצעות הפקודה:

```
vcs -kdb -sverilog -debug_access+all -full64 flop_test.v flop.v
simv -gui
```

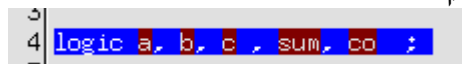
תיאור הממשק הגרפי



- קליק כפול על שם של בלוק או תת בלוק יציג את הקוד שלו בחלון הימני.

הצגת צורות גל :

- א. בחר את המודול בחלון השמאלי ובחר ב: Add To Waveform או
- ב. בחר את הסיגנלים על התיאור של הקוד כדלקמן:



- אין חשיבות לעובדה שבחורים גם מילים שאינן שמות סיגנלים.
- לחץ על הכפתור הימני ובחר ב: Add To Waveform (new או חלון שכבר קיים).
- ניתן להוסיף סיגנלים נוספים גם עם Drag and Drop.
- שים לב, נפתח חלון של צורות גל מעל חלון הקונסול. על מנת לראות את הקונסול יש ללחוץ על 'console' בתחתית החלון. ניתן גם להזיז אותו עם הכפתור השמאלי של העכבר או עם הפקודה: Window->Dock To

שינוי יחידות הזמן :



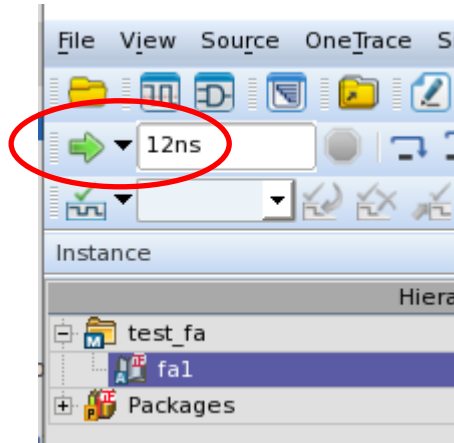
- ניתן לשנות את יחידות הזמן בעזרת :
- לחץ על 'ps' ובחר את היחידות הרצויות.

שמירת הקונפיגורציה של ה - session

- File-Save Session שומר מצב הריצה. ניתן לשחזר אותה עם File - Restore Session

הרצת הסימולציה :

- להרצת הסימולציה יש ללחוץ על החץ הירוק. אם רושמים זמן בחלון שליד אז הסימולציה תתקדם בזמן שצוין. לחיצה על החץ ללא רישום של זמן מריצה סימולציה עד הסוף.



- ניתן לקדם את הסימולציה גן ע"י רישום **run Nns** בקונסול כאשר Nns מסמן את מספר יחידות הזמן שברצונך להריץ.

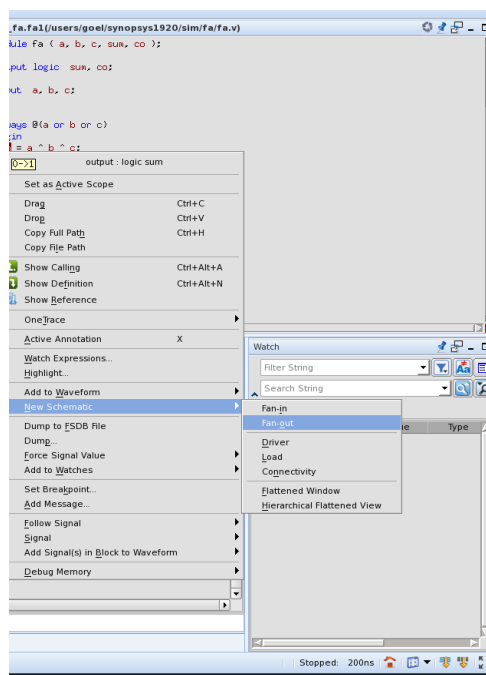
ביצוע Zoom :



- ניתן לבצע zoom באמצעות לחצני

פעולות נספות :

- בחירת סיגל, לחיצה על הכפתור הימני של העכבר (RMB) ובחירת New Schematic תאפשר ביצוע מספר מפעולות "דיבוג" כגון:



- **הוספת breakpoint :** בחירת שורה, לחיצה על הכפתור הימני של העכבר (RMB) ובחירת Set Breakpoint וללחוץ על Create בחלון שנפתח.

- **ביצוע Single Step :** ניתן לבצע את הסימולציה בצעדים בודדים בעזרת



או Step

- **ביצוע watch על סיגנל** : ניתן לקבל הדפסת של הערכים של סיגנל ע"י בחירת הסיגנל, לחיצה על הכפתור הימני של העכבר (RMB) ובחירת Add to Watches.

ייצוג הערכים בסיסים שונים

- ניתן להציג ערכים של Bus בחלון ה-waveform בסיסים שונים. לשם כך, בחר בחלון זה : Waveform->Set Radix ובחר בבסיס הרצוי.

Menu - החזרת ה-

- לעתים ה-menu של חלון מסוים נעלם. לחיצה על ה-RMB בחלק העליון של החלון ובחירת menu תחזיר את ה-menu.

1.2 סימולציית VHDL

הרצת הסימולציה מורכבת משלשה שלבים : elaborate ,analyze והפעלת ה-gui כדלקמן:

```
vhdlan -kdb file1.vhd file2.vhd --Analyze
vcs -kdb -debug_access+all -full64 cfg_design --Elaborate
simv -gui
```

בשורה הראשונה יפורטו שמות קבצי התכנון(כל שמות הקבצים המכילים את הבלוקים ותתי הבלוקים). פקודת ה-vcs תבצע אלבורציה, יש לפרט את שם הקונפיגורציה cfg_design כפי שמופיעה בקובץ ה-top level של התכנון. הערה- ניתן לבצע אלבורציה גם על top_entity

```
vcs -debug_all -debug -full64 top_entity -- Elaborate
TestBench
```

מקובל לבצע סימולציות באמצעות רכיב עזר (testbench) שמספק את אותות הכניסה לתכנון. בדוגמא הבאה, jkff הוא התכנון אותו יש לבדוק :

```
Library IEEE;
USE IEEE.std_logic_1164.ALL;

entity jkff is
port (clk, j, k :in std_logic;
      q, qn :inout std_logic);
end jkff;

architecture arc_jkff of jkff is
signal q_tmp, qn_tmp : std_logic;
begin
  process(j, k, clk)
  begin
    if (j = '0') then
      if (k = '0') then
        q_tmp <= q;
        qn_tmp <= qn;
      else
        q_tmp <= '0';
        qn_tmp <= '1';
      end if;
    elsif (k = '0') then
      q_tmp <= '1';
      qn_tmp <= '0';
    else
      q_tmp <= qn;
      qn_tmp <= q;
    end if;
  end process;
  process
  begin
    wait until clk'event and clk = '1' ;
    q <= q_tmp;
    qn <= qn_tmp;
  end process;
end arc_jkff;

configuration cfg_jkff of jkff is
  for arc_jkff
  end for;
end cfg_jkff;
```

jk_test מספק אותות כניסה ל- jkff ו- jk_test_top מחבר ביניהם :

```
library IEEE;
use IEEE.std_logic_1164.all;

entity jk_test is
  PORT (j, k : OUT STD_LOGIC;
        clk : INOUT STD_LOGIC);
end jk_test;

architecture arc_jk_test of jk_test is
begin
  PROCESS
  BEGIN
    WAIT FOR 5 ns;
    IF clk='0' THEN
      clk <= '1';
    ELSE
      clk <= '0';
    END IF;
  END PROCESS;
  PROCESS
  BEGIN
    WAIT FOR 20 ns;
    j <= '0';
    k <= '1';
    WAIT FOR 40 ns;
    j <= '1';
    k <= '0';
    WAIT FOR 40 ns;
    k <= '1';
    WAIT FOR 40 ns;
    j <= '0';
    k <= '0';
    WAIT FOR 60 ns;
  END PROCESS;
end arc_jk_test;

configuration cfg_jk_test of jk_test is
  for arc_jk_test
  end for;
end cfg jk test;
```

```
library IEEE;
use IEEE.std_logic_1164.all;

entity jk_test_top is
  PORT (q, qn : INOUT STD_LOGIC);
end jk_test_top;

architecture arc_jk_test_top of
jk_test_top is
  SIGNAL clk, j, k : STD_LOGIC;
  COMPONENT jk_test
    PORT (j, k : OUT STD_LOGIC;
          clk : INOUT STD_LOGIC);
  END COMPONENT;
  COMPONENT jkff
    PORT (clk, j, k : IN STD_LOGIC;
          q, qn : INOUT STD_LOGIC);
  END COMPONENT;
  BEGIN
    u1 : jk_test
      PORT MAP(j, k, clk);
    u2 : jkff
      PORT MAP(clk, j, k, q, qn);
  end arc_jk_test_top;

configuration cfg_jk_test_top of jk_test_top is
  FOR arc_jk_test_top
    FOR u1 : jk_test USE ENTITY
      WORK.jk_test(arc_jk_test);
    END FOR;
    FOR u2 : jkff USE ENTITY
      WORK.jkff(arc_jkff);
    END FOR;
  END FOR;
end cfg_jk_test_top;
```

הערה : הסברים על הממשק הגרפי מופיעים בסעיף 1.1 לעיל.

חשוב : אם אין לך משפט עצירה ב- Testbench יש לקדם את הסימולציה גם פקודת run בקונסול ולא עם החץ הירוק.

1.3 סימולציה מעורבת

ניתן לבצע סימולציות המערבות קוד VHDL ו-verilog ראשית מבצעים vlogan או vhdlan לכל הקבצים בהתאם לשפה. לאחר מכן, אם ה- top_level הוא Verilog יש להשתמש בפקודה vcs הבאה :

```
vcs -debug_all -debug -full64 top_module
```

ואם ה- top_level הוא VHDL יש להשתמש בפקודה vcs הבאה :

```
vcs -debug_all -debug -full64 cfg_design
```

2. סינתזה – Design Vision

חשוב : עליך לבקש מאחראי המעבדה גישה לקבצי tower לפני תחילת הסינתזה.

להכנת סביבת הסינתזה יש ליצור ספרייה חדשה בשם design_syn בעזרת הפקודה הבאה :

```
cp -r /users/iit/synopsys/tsl_syn_2021 tsl_syn_2021
```

עליך להפעיל את כלי הסינתזה מתוך ספרייה זאת. קבצים המיוצרים על ידי הכלי מאוחסנים בספרייה ששמה WORK שנוצרת בתוך design_syn. לאחר יצירת הספרייה עבור אליה עם :

```
cd tsl_syn_2021
```

ה-Design Compiler (או Design Vision בגרסתו הגרפית) הוא כלי הסינתזה של חברת Synopsys. הכלי מקבל כקלט תיאור SystemVerilog או VHDL ויוצר כפלט מעגל ברמת השערים המממש את ה-SystemVerilog או VHDL. הכלי מופעל באמצעות הפקודה design_vision שגורם לפתיחה של חלון גרפי המאפשר הרצת פקודות מתוך תפריטים.

כל פקודה מופעלת בעזרת הממשק הגרפי תופיע גם בחלון ה-terminal ממנו הרצנו את הכלי. ניתן להריץ את פקודות גם על ידי כתיבתם בחלון ה-terminal. ניתן כמובן לרשום את כל הפקודות בקובץ (בשם script.tcl למשל) ולהריץ את כל הקובץ בבת אחת על ידי File→Execute Script. ניתן לקבל את קובץ כל הפקודות שהרצנו בסוף של קובץ בשם command.log שנוצר במהלך העבודה.

help : בצד ימין של התפריט הראשי, מופיע כפתור help. בעזרתו ניתן לקבל הסברים של כל הפקודות וכל המשתנים של design_vision.

חשוב: שילוב זיכרונות RAM: אם התכנון מכיל זיכרונות RAM ראה המשך (עמ' 10) לפני תחילת הסינתזה!

סינתזה - שלבי הביצוע:

1. קריאת הקבצים

קיימות שתי דרכים לקריאת הקבצים :

א. כאשר התכנון אינו מכיל פרמטרים, נתן לקרוא את קבצי ה-VHDL/systemverilog, בעזרת File→Read. בחלון שנפתח יש לבחור את קובץ התכנון (לא כולל קובצי הזיכרון) וללחוץ OK. אם מדובר על קובץ systemverilog יש לבחור בפורמט sverilog. יש לחזור על פעולות אלו עד שנקראו כל קבצי התכנון (יש להתחיל עם המודולים הקטנים ביותר).

ב. אם התכנון ממומש בצורה פרמטרי, לקריאת קבצי ה-VHDL/systemverilog לחץ על File→Analyze. בחלון שנפתח יש ללחוץ add. בחלון שנפתח יש לבחור את כל קבצי התכנון (לא כולל קובצי הזיכרון) וללחוץ OK. יש לחזור על פעולות אלו עד שנקראו כל קבצי התכנון (אין חשיבות לסדר בו קוראים את הקבצים). בחר בפורמט AUTO. **הערה :** לעתים, למרות שבוחרים בפורמט sverilog הכלי לא מבצע זאת כהלכה. במקרה כזה, יש לקרוא את הקבצים בעזרת פקודה בקונסול. לדוגמא :

```
analyze -format sverilog counter.sv
```

על מנת למפות את התכנון לשערים לוגיים (סינתזה ללא תלות בטכנולוגיה). לחץ על File→Elaborate, בחלון שנפתח רשום WORK בשדה של ה-Library ובחר את ה-top level (התכנון הראשי שמכיל את שאר התכנונים), סמן ב-V את הריבוע Re-analyze out-of-date libraries. רשום ערכים עבור כל הפרמטרים ולחץ על OK. בתום

שלב זה יופיע שם התכנון שבחרנו וכל היחידות אותו הוא מכיל בחלון ה- Logical Hierarchy. על ידי לחיצה על הכפתור הימני על שם התכנון יפתח תפריט בו ניתן לבחור לראות את הסכמה שנוצרה ע"י schematic view ולראות את תכונות היחידה על ידי Properties. (חלון זה מכיל מידע על היחידה ואפשר דרכו לסמן אותה כ- dont touch) לדוגמא.

2. הגדרת אילוצים

לפני ביצוע הסנתזה יש להגדיר את אילוצי התכנון /האילוץ הבסיסי שעלינו להגדיר הוא מחזור השעון.

פקודה להגדרת שעון `create_clock`, דוגמא:

```
create_clock -name "CLK" -period 10 -waveform {0 5} clk
```

מגדירה את שעון על ההדק CLK בעל מחזור 10ns, המתחיל ב- 0 ועולה ל- 1 בזמן 5ns.

3. אופטימיזציה ומיפוי לשערי ספרייה (סינתזה תלוית טכנולוגיה)

ראשית, חשוב שבחלון ה- Logical Hierarchy יופיע השם של ה- `top_level` של התכנון. אם זה לא המצב, אפשר להריץ את הפקודה הבא:

```
current_design top_level
```

כשאר `top_level` זה השם שהשתמשת בתכנון שלך.

בחלון ה- Logical Hierarchy בחר בתא ה- `top_level`. לחץ על `Design` → `Compile` Design. בחלון שנפתח ניתן לבחור את כמות העבודה שהכלי ישקיע בכל שלב על ידי שינוי הבחירה באפשרויות ה- `effort`, השארת האפשרויות הבסיסיות מאפשרת קבלת תוצאות טובות, לסיום לחץ OK.

אם חלון ה- Logical Hierarchy ריק, יש לסגור אותו ולפתוח חדש עם:

```
Hierarchy->New Logical Hierarchy View
```

צפיה ב- schematic (עלול לקחת זמן רב)



בחלון ה- Logical Hierarchy בחר בתא ה- `top_level`. לחת על `Design` → `Check Design` בצע לחיצה כפולה על התא שמופיע.

4. חשוב: כעת עליך לבצע בדיקה של התכנון כדי לאתר תקלות באמצעות `Design` → `Check Design`. עליך לעבור בקפידה על הפלט של הפקודה. חשוב לקרוא את כל ה- `warnings` ולתקן את כל התקלות בתכנון.

הערה: ניתן לשמור את הפלט של ה- `design_vision` בקובץ ע"י בחירת `save content as` שמופיע בתפריט `options` בצד ימיני התחתון של החלון כדי להקל מציאת ה- `warnings`.

שמירת התוצאות

לחץ על `File` → `Save` ובחר בשדה `Format` את הפורמט הרצוי. וודא כי `Save All designs in hierarchy` מסומן ולחץ `Save`.

הפקת דוחות

- מסלול קריטי: `Timing` → `Report Timing Path`, לחץ OK. בתחילת הדו"ח שנוצר ניתן לראות פרטים על הטכנולוגיה שבה נעשה השימוש, בטבלה ניתן לראות את זמני ההשהיה של כל תא (ב-ns) שה"כ הזמן עבור המסלול הקריטי מופיע בתחתית הדוח ב- `data arrival time` (ב-ns), במידה והוגדרו אילוצים על זמן קריטי/מקסימלי תופיע שורה המציינת אם התכנון עמד באילוץ (`slack met/violated`):

data required time	3.73
data arrival time	-3.73
slack (MET)	0.00

- שטח : באמצעות Design→Report area, לחץ OK. בדו"ח ניתן לראות את השטח שמאוכלס על ידי הלוגיקה הצרופית ושאינה צרופית והרשת (חוטים), היחידות הן מספרי שערי NAND בעלי שתי כניסות. זה מספר השוואתי בלבד. רצוי לחלץ שטח מ- layout.
- צריכת הספק : באמצעות Design→Report power

עד כאן כל הפעולות הדרושות לביצוע סינתזה בסיסית. אם ברצוננו לבצע סינתזה מורכבת יותר, ניתן להיעזר בפעולות הבאות :

הגדרת אילוצים נוספים

ניתן להגדיר אילוץ אופטימיזציה נוספים (מעבר להגדרת השעון) על תהליך הסינתזה, כמו הגבלות זמן על המסלול הארוך ביותר, שטח התכנון וכד', הכלי יסנתז את התכנון בהתאם לאילוצים אלו. את האילוצים ניתן לכתוב בקובץ סקריפט יעודי עם סיומת tcl. יש להריץ את הסקריפט לאחר שלב הטעינה ולפני שלב הסינתזה באמצעות File→Execute script.

set_max_delay

קובע אילוץ של השהיה המרבית בין צמתים שונים במעגל. דוגמא :

```
set_max_delay 10 -to D[*]
```

הכלי ינסה לבנות מעגל כך שההשהיה בין כל צומת במעגל לצמתים ששם מתאים ל- D[*] לא תעלה על 10ns. אין ודאות שהכלי יצליח לסנתז מעגל כזה.

set_max_area

קובע אילוץ לגבי השטח הכולל של המעגל.

set_input_delay

מאפשר למשתמש להגדיר מתי (ביחס לעליית השעון) אותות הכניסה מוכנים.

set_output_delay

מאפשר למשתמש להגדיר מתי (ביחס לעליית השעון) אותות היציאה צריכים להיות מוכנים.

set_dont_touch block_name

מורה לכלי לא לסנתז בלוק מסוים.

*שילוב זיכרונות RAM

השלב הבא אינו חיוני לכל פרויקט ולכן יש לבצע אותו רק בהנחית המנחה

אם קיים זיכרון, הכלי דורש את קובץ ה- db של הזיכרון. קובץ זה מתאר את המאפיינים של הזיכרון. לרב מגיע קובץ lib. ולא קובץ db. ניתן לתרגם את ה- lib ל- db. באופן הבא : הפעל את ה- design_vision. בצע את הפעולות הבאות :

```
read_lib memory_typ.lib
```

```
write_lib memory_typ -format db
```

כאשר memory_typ.lib הוא קובץ ה- lib* של הזיכרון. סגור את ה- design_vision.

השלב הבא אינו חיוני לכל פרויקט ולכן יש לבצע אותו רק בהנחית המנחה

שילוב RAM בתכנון בסינתזה

אם התכנון מכיל זיכרון RAM, יש לקרוא את קובצי המימוש (Systemverilog או VHDL) ל- Design Vision כפי שהוסבר, אבל אין לקרוא את הקובץ שמכיל את תיאור ה- RAM עצמו.

כלומר, קוראים את הרמה שמכילה את התיאור המבני של ה-RAM בלבד. בצע סינתזה כפי שהוסבר ושמור את התכנון לקובץ Verilog.

יש לערוך את קובץ ה-Verilog ולהוסיף (בסוף הקובץ) תיאור ריק של ה-RAM, לדוגמא:

```
module dpram1024x16 ( CSA, CSB, NRSTA, NRSTB, ENA, ENB, RDA, RDB,
    WRA, WRB, ADA, ADB, DIA , DIB, DOA, DOB);

input  CSA;
input  CSB;
input  NRSTA;
input  NRSTB;
input  ENA;
input  ENB;
input  RDA;
input  RDB;
input  WRA;
input  WRB;
input [9:0]  ADA;
input [9:0]  ADB;
input [15:0] DIA;
input [15:0] DIB;
output [15:0] DOA;
output [15:0] DOB;

endmodule
```

ממשיכים אם הקובץ המתוקן ל- **Innovus**.

שימוש ברכיבי DesignWare

DesignWare (DW) היא ערכה של Synopsys. ערכה זו מכילה מימושי VHDL ו-Verilog של רכיבים ומאפשרת למשתמש לשלבם בתהליך כתיבת הקוד (למשל שימוש ב- יחידות אריטמיות מסוגים שונים כגון מחלק). את כל ההסברים ניתן למצוא ב- manual של הכלי.

דוגמא לשילוב מחלק במימוש Systemverilog

```
module DW_div_inst (a, b, quotient, remainder, divide_by_0);
parameter width = 8;
parameter tc_mode = 0;
parameter rem_mode = 1; // corresponds to "%" in Verilog
input [width-1 : 0 ] a;
input [width-1 : 0 ] b;
output [width-1 : 0 ] quotient ;
output [width-1 : 0 ] remainder;
output divide_by_0;
DW_div #(width, width, tc_mode, rem_mode)
    U1 (.a(a), .b(b),
        .quotient(quotient), .remainder(remainder),
        .divide_by_0(divide_by_0));
endmodule
```

דוגמא לשילוב מחלק במימוש VHDL :

```
library IEEE,DWARE,DW01,DW02,DW03;
use IEEE.std_logic_1164.all;
use ieee.std_logic_arith.all;
use DW01.DW01_components.all;
use DW02.DW02_components.all;
use DW03.DW03_components.all;
```

ניתן להשתמש במספר שיטות :

inference .א

בדרך זאת קוראים לפונקציה שברצוננו להשתמש כגון כפי שמופיע בדוגמא הבאה :

```
1 library IEEE,DWARE,DW01,DW02;
2 use IEEE.std_logic_1164.all;
3 use ieee.std_logic_arith.all;
4 use DW01.DW01_components.all;
5 use DW02.DW02_components.all;
6
7 entity mymult is
8   port (
9     a, b      : in  std_logic_vector(15 downto 0);
10    CLK       : in  std_logic;
11    product   : out std_logic_vector(31 downto 0));
12 end mymult;
13
14 architecture arc_mult of mymult is
15
16 begin
17
18 --product <= a * b;
19 product <= std_logic_vector(DWF_mult_4_s(signed(a),signed(b),CLK));
20
21 end arc_mult;
```

שים לב ששורה 19 גורמת לשימוש במכפל מצונר בעל 4 דרגות. ניתן היה גם להשתמש בקריאה כפי שמופיעה בשורה 18 (שנמצא ב- comment), במקרה הזה ה-DW היה משתמש במכפל מקבילי.

Instantiation .ב

בשיטה זאת משתמשים בתיאור מבני של ארכיטקטורה ופשוט מציבים בלוקים של ה-DW. דוגמא :

```
library IEEE,DWARE,DW01;
2 use IEEE.std_logic_1164.all;
3 use DW01.DW01_components.all;
4 use DWARE.DWpackages.all;
5
6 entity DW01_add_ci_co_test is
7   port (
```

```

8   a, b : in std_logic_vector(31 downto 0);
9   cin : in std_logic;
10  sum : out std_logic_vector(31 downto 0);
11  cout : out std_logic;
12 end DW01_add_ci_co_test;
13
14 architecture inst of DW01_add_ci_co_test is
15
16 begin
17
18 U1 : DW01_add
19   generic map ( width => 32 )
20   port map ( a , b , cin , sum , cout ) ;
21
22 end inst;

```

הערה : משפט ה- generic map שבשורה 19 בקוד הנ"ל מאפשר למשתמש לקבוע את רוחב המסכם.

השלבים הבאים אינם חיוניים לכל פרויקט ולכן יש לבצע אותו רק בהנחית המנחה יצירת קובץ SDF (Standard Delay Format)

קובץ ה-sdf מכיל נתוני השהיה של התכנון המסונתז. קיומם של שערים וקווים (חוטים) "פיסיקליים" בתכנון המסונתז גורם להשהיות. קובץ ה-sdf מכיל נתוני השהיות אלו ובעזרתו ניתן לבצע סימולציות דיגיטליות (באמצעות VCS, NCSIM) ואנליזות (Design Vision, Prime) Time מדויקות יותר ותואמות למציאות הפיסיקלית של התכנון.

בחר ביחידה העליונה ורשום ב- command window :

```
write_sdf your_design_name.sdf
```

סימולציה של מעגל מסונתז ושימוש בקבצי SDF

סימולציה של מעגל מסונתז שונה מסימולציה רגילה משום שהיא מתחשבת בהשהיותיהם של השערים והקווים (באמצעות קובץ ה-sdf) ולכן סימולציה מסוג זה מציגה גם השהיות.

כדי לבצע סימולציה כזאת, יש לבצע את הפעולות הבאות :

א. ליצור תיאור VHDL של המעגל המסונתז. ב- Design Vision בצע File → Save As לשמור בפורמט הרצוי.

ב. בקוד ה-VHDL המסונתז, עליך להוסיף את השורות הבאות לפני כל entity :

```
library FS120;
use FS120.TSL18FS120_COMPONENTS.all;
```

ג. את השלב הבא ניתן לעשות בשתי דרכים, בדרך הראשונה יש להוסיף קונפיגורציה בצורה הבאה :

```
configuration CFG of DESIGN is
  for SYN_BEH
  end for;
end CFG;
```

כאשר Design הוא שם התכנון ו-SYN_BEH הוא שם הארכיטקטורה, ויש לבצע את פקודת ה-vcs להלן על הקונפיגורציה CFG. דרך שנייה- על ידי ביצוע פקודת vcs להלן על ה-top_entity במקום על הקונפיגורציה. בצע את הפקודות הבאות :

```
vlogan vlog_files_names
vhdlan vhd_files_names
vcs -debug configuration_name --or top_entity instead
simv -sdf testbench_entity_name/design_portmap_name:sdf_file.sdf -gui
```

דוגמא

עבור תכנון של MUX2 בוצע סינתזו לקובץ mux2_mapped.vhdl כמו כן יוצר קובץ .mux2.sdf קובץ ה-Testbench מכיל top_entity בשם MUX2_TB, שם הארכיטקטורה בקובץ זה הוא MUX2_TB_arch ושם האינסטנס (port map) של MUX2 הוא MUX2_PM, לכן יתבצע:

```
vhdlan mux2_mapped.vhdl testbench.vhdl
vcs -debug MUX2_TB
simv -sdf MUX2_TB/MUX2_PM:mux2.sdf -gui
```

אם הפקודות מתבצעות בהצלחה, ניתן להריץ סימולציה כפי שעשית על הקוד ההתנהגותי. חשוב לציין שמתבצעת כאן סימולציה ברמת השערים הלוגיים וההשהיות האמיתיות נלקחות בחשבון.