הטכניון - מכון טכנולוגי לישראל
הפקולטה להנדסת חשמל

מעבדה 1

ניסוי VHDL1
חומר רקע

גרסה 1.07

אביב 2018

עורכים: דודי בר-און, אברהם קפלן
על פי חוברות של עמוס זסלבסקי

תוכן עניינים

1. מטרות הניסוי

- התנסות בתיאור תכן באופן התנהגותי בשפת תיאור חמרה VHDL
- הכרת כמה כללים וסגנונות כתיבה בשפת VHDL שמתאימים לסינתזה
- התנסות בסינתזה טכסטואלית וטכסטואלית-גרפית עם כלי הפיתוח Quartus
- שימוש בדיווחים ובכלי מעקב בכלי הסינתזה Quartus
- הורדה התכן לרכיב ובדיקתו באמצעות כרטיס התרגול

הערה חשובה:
כל הקבצים המוכנים והחצי מוכנים לניסוי זה נמצאים במוודל

2. מהי שפת VHDL ?

שפת VHDL היא תוצאה של פרוייקט מחקר גדול שנקרא VHSIC (Very High Speed Integrated Circuit) שנעשה במשרד ההגנה האמריקאי (US Department of Defense) ושהחל בשנות השמונים המוקדמות. המטרה המקורית של השפה שהוצעה כחלק מהפרויקט בשנת 1981 הייתה ליצור שפת תיעוד וסימולציה סטנדרטית לתיאור חמרה ספרתית מורכבת.
השם VHDL בא מהמלים : VHSIC Hardware Description Language.

בשנת 1986 השפה הוצגה בפני IEEE (Institute of Electrical and Electronics Engineers) (כהצעה לתקן. לאחר תוספות ושינויים, נוצר בשנת 1987 התקן שמספרו IEEE 1076, שמגדיר באופן בסיסי את השפה. במשך השנים נעשו ב – IEEE עדכונים נוספים לתקן הנ"ל) בשנים – 1993, 2002 ו 2006. בנוסף לכך נוצרו ב – IEEE במשך השנים, תקנים שנלווים לתקן המקורי של השפה, כמו למשל : תקן לסוגי מידע סטנדרטיים 1164)). בזמן הקרוב השפה עומדת לעבור שיפורים רבים וחלקם מרחיקי לכת.

למרות שהיעוד המקורי של השפה, היה כאמור לאפשר יצירה של סימולטורים ותיעוד, במשך השנים נוצרו גם כלי סינתזה רבים לשפה.

כיום שפת VHDL היא אחת משפות תיאור החמרה הנפוצות ביותר בתעשייה ובעולם האקדמי.

2.1. מקורות מידע ללימוד שפת VHDL

בניסוי זה ובניסוי הבא נכיר את שפת VHDL באופן חלקי. שפת VHDL היא שפה גדולה ומורכבת ובלימוד שלה כדאי להיעזר בספרות. קיימת ספרות ענפה בעברית ובאנגלית בנושא.

להלן ספרים מומלצים בשפה העברית ללימוד הרחבה והבהרה של שפת VHDL :

1. **עמוס זסלבסקי, "לימוד שפת *VHDL* לסימולציה וסינתזה", הוצאת שורש 2007.**
2. שאול כהן, "מדריך מקצועי לתיכון חמרה", הוצאת ארז,2005.

ספרים מומלצים נוספים בשפה האנגלית ללימוד השפה הם הספרים הבאים :

3. Jayaram Bhasker, "A VHDL Primer – Revised edition", Prentice Hall, 1995, ISBN: 0-13-181427-8
4. David Pellerin & Douglas Taylor, *"VHDL made easy"*, Prentice Hall PTR, 1997, ISBN: 0136507638.
5. Douglas Perry, *"VHDL"*, McGraw-Hill, 4th edition 2002 , ISBN: 2070041700
6. Stefan Sjohlm & Lennart Lindh, *"VHDL for Designers"*, Prentice Hall, 1997, ISBN: 0134734149

את תקן VHDL המלא והמעודכן ניתן למצוא באינטרנט (תוך הטכניון בלבד) באתר :
http://ieeexplore.ieee.org/xpl/standards.jsp
יש לחפש תקן IEEE 1076 :

IEEE Standard 1076-1993: VHDL Language Reference Manual, IEEE, 1993, ISBN: 1-55937-376-8 [SH16840].

לגבי שימוש בתקן כדאי להיזהר, מכיוון ששפת VHDL היא שפה קשה ללימוד אם מנסים ללמוד אותה ישירות מתוך התקן שלה. **תמיד כדאי להעדיף ספר לימוד על פני התקן** !

הסבר נוסף ב WIKIPEDIA–
http://en.wikipedia.org/wiki/Mealy_machine
http://en.wikipedia.org/wiki/Moore_machine

ובאתר אלטרה
http://www.altera.com/support/examples/vhdl/vhd-state-machine.html

3. שיטות שונות לתיאור המערכת – Design Entry

בניסוי הקודם השתמשנו בשיטות גרפיות על מנת לתאר את המערכת (Design Entry). כלומר השתמשנו בכלי שרטוט סכמתי (Schematic Capture) של Quartus. תיאור מערכת יכול להיעשות לחילופין באמצעות הקלדה של קובץ בשפה טכסטואלית – כמו למשל בשפת VHDL.

מהם היתרונות והחסרונות של כל אחת משיטות התיאור הנ"ל ?

אחת הבעיות בתיאורים סכמתיים היא, שקשה לכלול בהם תיאורי התנהגות (Behavioral Descriptions) כל שהם, אלא אך ורק תיאורי מבנה (Structural Descriptions). לעומת זאת בתיאורים טכסטואליים ניתן לכלול הן תיאורים התנהגותיים והן תיאורים מבניים.

מדוע אם כן קיימת בעיה כל שהיא, באי היכולת של כלי שרטוט לכלול תיאורים התנהגותיים ?

הבעיה היא שקיימות מערכות רבות כמו למשל מכונת מצבים (State Machine), שבהן קשה לתאר את המערכת באופן מבני (כלומר באמצעות חיווט של תת-רכיבים). תיאור מבני של מכונת מצבים, יכול להיעשות בעצם רק לאחר שנבצע מימוש ידני מייגע, באמצעות נייר ועיפרון ולאחר שנפרק את מימוש המכונה לפליפ-פלופים ושערים. זוהי כמובן פעולה שהיא מאוד לא כדאית, מכיוון שבעצם אנו מעוניינים שכלי הסינתזה יבצעו זאת עבורנו.

בתיאור טכסטואלי קל לתאר מכונת מצבים באופן התנהגותי באמצעות קובץ טכסט. להלן דוגמה לקטע קוד שמתאר התנהגות של מכונת מצבים באופן טכסטואלי:

```
case present_state is
when idle    =>
    if  r1 = '1' then
next_state <= grant1 ;
      elsif r2 = '1' then
next_state <= grant2 ;
      elsif r3 = '1' then
next_state <= grant3 ;
      else   next_state <= idle    ;
    end if ;
when grant1 =>
      a1 <= '1' ;
     if r1 = '1' then
         next_state <= grant1 ;
      else
        next_state <= idle    ;
     end if ;
when grant2 =>
```

.
.
.

יתרון נוסף של תיאורים התנהגותיים הוא, שקל יותר לתאר באמצעותם מערכות גדולות. כאשר רוצים לתאר מערכות גדולות באמצעות טכסט, ניתן להשתמש למשל ביכולות השכפול של חוגים.

```
for i in 1 to 100 loop
  -- this section gets multiplied 100 times
  .
end loop ;
```

תיאורים התנהגותיים נחשבים "ידידותיים" יותר לבני אדם. בתיאורים התנהגותיים אנו בדרך כלל יכולים לתאר את המערכת ברמה גבוהה יותר - כלומר אנו חושבים "במישור הבעיה" ונותנים לכלי הסינתזה שלנו להמיר את תיאור המערכת למימוש עם רכיבי חמרה – "במישור הפתרון".

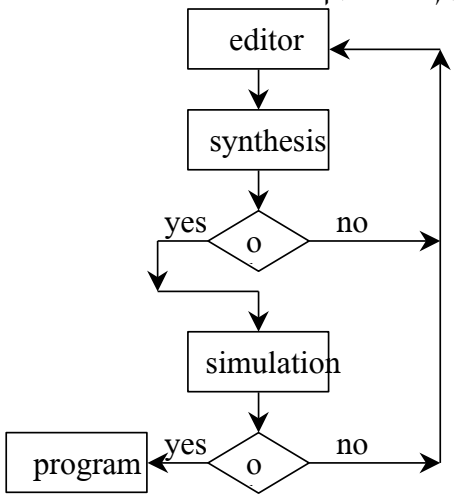
עד לרגע זה, ראינו שלתיאורים סכמתיים יש רק חסרונות. האם יש לתיאורים סכמתיים גם יתרונות כל שהם ? התשובה היא חיובית. תיאורים סכמתיים הם לעתים ידידותיים יותר מבחינה ויזואלית לבני אדם

ובעיקר כאשר מדובר באנשים שאינם שולטים טוב בשפת תיאור חמרה כל שהיא ובמיוחד כאשר מדובר בהיררכיות גבוהות של תיאור המערכת.

לכן, רוב האנשים יעדיפו לתאר את רוב החמרה באמצעות טכסט ואת החיווט של המערכת יבצעו באופן טכסטואלי או באופן סכמתי. כפי שנאמר קודם, תיאור סכמתי נפוץ יותר בהיררכיות הגבוהות של הפרויקט)קרוב להיררכית ה – Top Level(.

3.1. אפשרויות שונות לביצוע מהלך התכן – Design Flows

מהלך התכן)Design Flow(של רכיבים מיתכנתים, שנעשה בעבר, כלל בדרך כלל את השלבים הבאים :

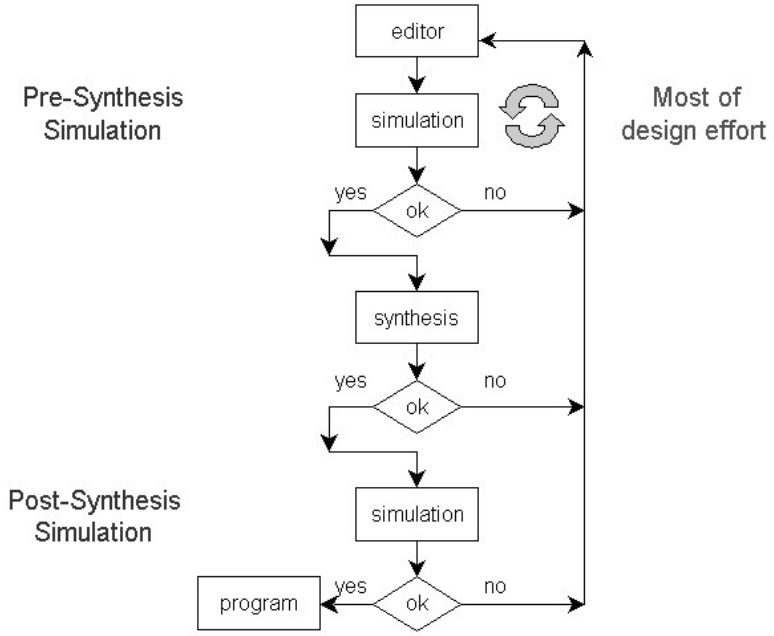


בשלב הראשון בתכן, רושמים באמצעות Text Editor קובץ טכסט בשפה כל שהיא או שמשרטטים את המערכת)כפי שאכן עשינו בניסוי הקודם(. מיד לאחר מכן מתבצעת סינתזה.
הסיבה הנפוצה לכשילון אפשרי של הסינתזה היא כמובן הימצאות של שגיאות בקובץ הטכסט או השרטוט. סיבה אפשרית נוספת היא למשל אי התאמה של הרכיב הנבחר לתכן, למשל הרכיב קטן מדי. לאחר תיקון של הקובץ או בחירה מתאימה של רכיב, חוזרים על השלבים הנ"ל, עד שהסינתזה עוברת בהצלחה. לאחר הצלחה של הסינתזה מבצעים למערכת המסונתזת סימולציה.
אם הסימולציה אינה עוברת בהצלחה, חוזרים להתחלה ומתקנים את תיאור המערכת וחוזרים שוב על כל השלבים הנ"ל, עד לקבלה של תוצאות סימולציה מוצלחות.

שים לב, זהו בדיוק מהלך התכן שבצעת בניסוי הקודם !

מהלך התכן הנ"ל התאים לתיאור של מערכות קטנות. כאשר מבצעים תכן של מערכת גדולה)בנפחים של עשרות אלפי שערים(, שלבי הסינתזה והסימולציה עשויים להיות ארוכים. היות והתהליך של תכן של מערכת הוא בדרך גם כלל תהליך איטרטיבי)מחזורי(שכולל בדרך כולל חזרות רבות על תהליך הסימולציה, ובמיוחד כאשר מדובר במערכות מורכבות, מהלך התכן של מערכת גדולה באופן הנ"ל נהפך לבלתי מעשי.

בתכן מודרני של מערכות גדולות, מעדיפים להשתמש במהלך התכן הבא:



השוני בין מהלך התכן הנ"ל ומהלך התכן הקודם הוא, שבתכן המודרני מבצעים גם סימולציה (Pre-Synthesis Simulation). לפני הסינתזה

הסימולציה שנעשית לפני הסינתזה היא סימולציה מאוד מהירה. בכדי שאפשר יהיה לבצע סימולציה כזו, יש צורך להשתמש בשפה שהקוד שלה ניתן לסמלוץ באופן ישיר כמו VHDL. אם ננסה למשל לסמלץ מונה ברוחב 32 סיביות שהקוד שלו בשפת VHDL מתואר באמצעות ההשמה הבאה:

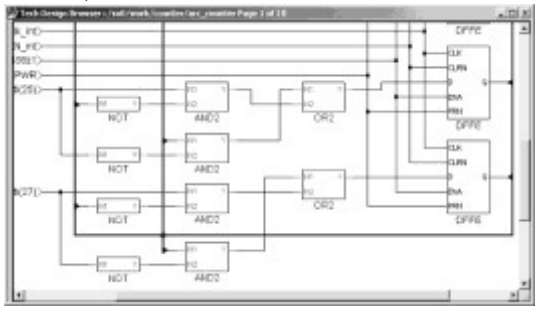
```
count <= count + 1 ;
```

סימולטור VHDL, יתרגם את הקוד כנראה לפקודה אחת בשפת מכונה של המחשב. למשל פקודה שהייצוג של בשפת אסמבלי (סף) של מעבדי אינטל (המעבד במחשבי PC) נראית למשל כך:

```
INC EAX
```

בסימולציה שנעשית לאחר הסינתזה (Post-Synthesis Simulation) מסמלצים בעצם מודל של החמרה ברמת השערים (Gate Level Simulation).

קטע מתוך התיאור הגרפי של מודל תכנה של המונה הנ"ל ברמת השערים עשוי להיראות למשל כך:



כפי שאפשר לראות באיור הנ"ל, המודל לסימולציה מכיל תיאור של שערים ופליפ-פלופים רבים והוא עשוי גם לכלול אפיון של תזמונים (Timing). כמובן שסימולציה כזו עלולה להיות הרבה יותר איטית. האיטיות של הסימולציה שנעשית לאחר הסינתזה במהלך תכן מודרני פחות מפריעה, מכיוון שהיא אינה נעשית פעמים רבות במהלך התכן.

בפיתוח חמרה מתוכנתת, המתכנן מבלה את רוב זמנו באיטרציות בחוג שכולל את השלבים של כתיבת קוד ב - Editor וביצוע סימולציה לקוד. שלבים אלו נעשים כאמור לפני הסינתזה. בשלבים אלו המתכנן מנפה את השגיאות בחשיבה שלו ובכתיבת הקוד. רק לאחר שהקוד תקין, עוברים לשלבים הבאים של סינתזה וסימולציה, שעשויים כאמור קודם, להיות שלבים איטיים אך הם אינם מתבצעים מספר רב של פעמים.

אם מבצעים סימולציה לפני הסינתזה, מדוע אם כן לבצע סימולציה גם לאחר הסינתזה ?

הסימולציה שנעשית לפני הסינתזה היא סימולציה פונקציונלית)Functional Simulation(, שאינה מתארת את ההשפעה של התזמונים של הרכיב ועל התכן. כאשר מבצעים תכן סינכרוני עם אות שעון בודד)לפי הכללים החשובים שנלמדו בניסוי הקודם(התנהגות המערכת תלויה פחות בתזמונים)כל עוד לא עוברים את מגבלות ביצועי המערכת(ולכן ניתן להסתפק לעיתים בביצוע סימולציה לפני הסינתזה בלבד.

אחד התפקידים של הסימולציה שנעשית לאחר הסינתזה, עשוי להיות בדיקה של השפעת התזמונים של הרכיב על התכן)Timing Verification(. הרכיב עלול למשל לא לעמוד בתדר השעון הנדרש או שפליפ-פלופים אחדים בתכן למשל אינם מקיימם זמן הכנה או החזקה)Setup & Hold Time(. על נושאים אלו נדון בניסוי הבא.

תפקיד נוסף של הסימולציה שנעשית לאחר הסינתזה הוא לגלות בעיות או שגיאות שנוצרו בעקבות הסינתזה. אחד המקורות האפשריים להיווצרות של בעיות מסוג זה הן טעויות בכלי הסינתזה עצמו. כמובן שתמיד קיימת אפשרות שכלי סינתזה יכיל באגים, אך כלי סינתזה מודרניים הם בדרך כלל כלים אמינים ובעיות מסוג זה הן אינן שכיחות.

מקור אפשרי נפוץ יותר להיווצרות בעיות בעקבות הסינתזה, טמון בסגנון הכתיבה של הקוד עצמו. כתיבה קוד שאינו מתאים לסינתזה של מערכת צירופית או מערכת סינכרונית קונבנציונליים) Sick Hardware Descriptions Coding(, עשויה לגרום לכלי הסינתזה ליצור חמרה שאינה מתאימה בהתנהגותה לסימולציה שנעשתה לפני הסינתזה.

למרות שכלי סינתזה איכותיים בדרך כלל מדווחים על בעיות מסוג זה באמצעות הודעות אזהרה)Warning(, המתכנן עשוי שלא לשים לב לדיווחים אלו ובעיקר באותם מקרים שבהם נוצרים דיווחים מאוד ארוכים. לצערנו כלי סינתזה פשוטים, לא תמיד מדווחים על כל סוגי הבעיות מסוג זה. ביצוע הסימולציה לאחר הסינתזה והשוואת התוצאות שלה לתוצאות הסימולציה שנעשתה לפני הסינתזה עשוי לגלות בעיות חמורות מסוג זה.

בניסוי זה נבצע אך ורק סימולציות לפני הסינתזה. רק בניסוי הבא נבצע סימולציות לאחר הסינתזה gate level simulation(.)

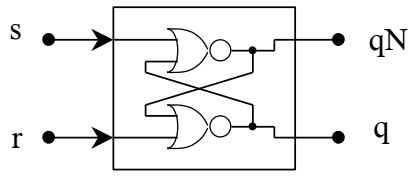
3.2. המאפיינים הבסיסיים והיכולות של השפה

שפות תכנות כמו C, PASCAL או BASIC מתארות תיאורים באופן סדרתי (Sequential Description). בשפת תכנות אלו ה – CPU מריץ את הפקודות שרשומות בתכנית לפי הסדר שלהן. אם למשל נרשום בשפת תכנות את אוסף ההשמות הבאות:

```
q = 3 ;
.
.
q = 1 ;
.
.
q = x + y ;
```

המשתנה q יקבל בכל פעם ערך אחר ולבסוף הוא יקבל את הביטוי שרשום בהשמה האחרונה x + y.

שפות תיאור חמרה מתארות חמרה באופן מקבילי (concurrently). אם למשל רוצים לתאר את רכיב ה – Latch הבא:



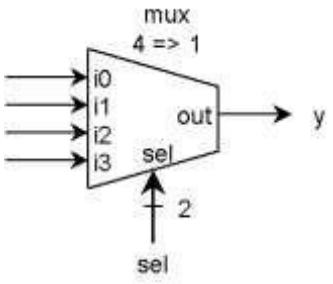
נוכל להשתמש בשתי ההשמות הבאות:

```
q    <=  not (r  or  qN) ;
qN   <=  not (s  or  q ) ;
```

הסדר של ההשמות אינו חשוב מכיוון ששתי ההשמות "חיות" ללא הפסקה ובמקביל אחת לשניה.

בשפת VHDL קיימים הן תיאורים במקביל והן תיאורים סדרתיים. התיאורים במקביל נעשים באמצעות יחידות שנקראות Process ויחידות אלו "חיות" במקביל אחת לשניה. כל process מתאר חמרה כל שהיא. בתוך ה - Process הפעולות מתבצעות באופן סדרתי – כמו בתכנית מחשב. השימוש ביחידות מסוג Process, מאפשר לתאר ולדבג מערכות מורכבות והופך את השפה לחזקה. הנושא החשוב של שימוש ב - Process יוצג בהמשך.

ניתן לתאר תיאורים בשפת VHDL הן בסגנון תיאור מבני או היררכי והן בסגנון תיאור התנהגותי והן באמצעות ערוב של שני סגנונות התיאור הנ"ל. נמחיש את שני סוגי סגנונות התיאור הנ"ל באמצעות הדוגמה הבאה, של חמרה מסוג בורר (Selector או Multiplexer) בעל ממדים 4=>1.



קטע הקוד הבא מתאר את הבורר בסגנון התנהגותי אפשרי.

```
y <= i0 when sel = "00" else
     i1 when sel = "01" else
     i2 when sel = "10" else
     i3 ;
```

שים לב שהקוד הנ"ל הוא קוד קריא וידידותי לבני אדם, והוא יכול להיות מובן גם לאנשים שאינם מכירים כלל את השפה.

את הבוררים הקטנים בתיאור המבני הנ"ל ניתן לתאר באופן התנהגותי או באופן מבני באמצעות שערים בודדים. קטע הקוד הבא מתאר את הבורר 2=>1 באופן התנהגותי.

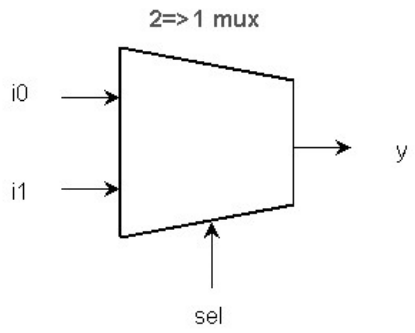
```
y <= ( not sel and i0 ) or ( sel and i1 ) ;
```

שפת VHDL מכילה תשתית מגוונת של פקודות שתומכות בסימולציה. השפה מסוגלת למשל לתאר זמני השהיית מעבר (tpd). שפת VHDL מסוגלת גם להתריע גם על אי קיום של אילוצי תזמון (Time Checker). ניתן לתאר בשפה גם גנרטורים (מחוללים). בנוסף ליכולות הנ"ל שפת VHDL מסוגלת לשלוח גם הודעות למסך, ולטפל גם בקבצי טכסט ובקבצים בעלי מבנים אחרים.

כל צורות הכתיבה הללו אינן מיועדות לסינתזה אלא לסימולציה בלבד.
בכל היכולות הללו לא נעסוק בניסוי זה. בחלקם נעסוק בניסויים שבהמשך.

3.3. דוגמה לקובץ פשוט בשפת VHDL

נתאר בשפת VHDL את המערכת הצירופית הפשוטה הבאה של בורר בעל ממדים 2=>1.



$$y = \overline{sel} \bullet i0 + sel \bullet i1$$

להלן קוד VHDL פשוט שמתאר את המערכת הנ"ל.

```
-- a very simple example
entity mux2 is
   port( i1 , i0 , sel : in  bit;
         y     : out bit ) ;
end mux2 ; architecture arc_mux2 of mux2 is
begin
  y <= ( not sel and i0 ) or ( sel and i1 ) ;
end arc_mux2 ;
```

השורה הראשונה בקטע הקוד הנ"ל היא שורת הערה (Comment).

בהמשך רשומים שני חלקים entity (ישות) ו architecture (ארכיטקטורה).

הישות מתארת את הכניסות והיציאות של המערכת. **כיווני** הכניסות מתוארים באמצעות המלים in ו out בהתאמה. סוג המידע של הכניסות הוא bit וסוג מידע זה יכול להיות בעל ערך של '0' לוגי או '1' לוגי.

הארכיטקטורה מתארת את ההתנהגות של המערכת באמצעות השמה באלגברה בוליאנית, שרשומה באמצעות אופרטורים מילוליים של and, or ו not. בשלב זה לא נתעמק יותר בתחביר של שפת VHDL. בנושא זה נעסוק בהמשך הרקע של הניסוי.

4. הישות - entity והארכיטקטורה - architecture

שני המרכיבים הבסיסיים ביותר בשפת VHDL הם ה - entity)ישות(ו ה - architecture)ארכיטקטורה(. ה - entity מתאר את חיבורי המערכת לעולם החיצוני. הארכיטקטורה מתארת את "הקרביים")החלק הפנימי(של המערכת באופן התנהגותי או מבני. בדרך כלל שני החלקים הנ"ל מאוחסנים בקובץ משותף כאשר הישות תמיד מקדימה את הארכיטקטורה. לחלקים אלו יש את המבנה התחבירי הבא:

```
entity entity_name is
  declerative_statements
        .
        .
end entity_name ; architecture architecture_name of entity_name is
  declarative_statements
        ..
begin
  operational_parallel__statements
        ..
 end architecture_name ;
```

לישות יש תפקיד דומה ל - Symbol הגרפי שהכרנו בניסוי הקודם. הישות כוללת בדרך כלל פסוקים הצהרתיים שמתארים ports)מפתחים(שמקשרים בין החלק הפנימי והחיצוני. פסוקי port מתארים:

- שמות ההדקים של הרכיב
- סוג המידע)Data Type(שלהם
- הכיוון)mode(שלהם

להלן דוגמה לתיאור ישות עם הדקים של Latch:

```
entity latch is
 port (s , r: in bit  ;
          q , qN : out bit ) ;
end latch ;
```

בניסוי זה ובאחרים נשתמש בנוהג המקובל של סימון אות N גדולה עבור אותות)כמו qN(שהם פעילים בנמוך)Active Low(. הכניסות בדוגמה הנ"ל מתוארות באמצעות הכיוון in והיציאות מתוארות באמצעות הכיוון out. קיימת בשפת VHDL גם יציאה מסוג buffer אך אנו לא נשתמש בה בניסוי זה. הדק דו כיווני מתואר בשפת VHDLבאמצעות הכיוון inout וגם בו לא נשתמש בניסוי זה.

הארכיטקטורה מורכבת משני חלקים: חלק הצהרתי)לפני המלה begin(וחלק ביצועי)אחרי המלה begin(. לסדר הפסוקים בחלק ההצהרתי עשויי להיות משמעות. בחלק הביצועי לסדר הפסוקים אין משמעות והם מתבצעים במקביל)Concurrently(.

שים לב שבשורה הראשונה של הארכיטקטורה נעשה קישור בין שם הארכיטקטורה)arc_latch(והשם של הישות)latch(באמצעות המלה of. בשפת VHDL ניתן ליצור לישות אחת כמה ארכיטקטורות, אך אנו לא נדון במקרים מסוג זה בניסוי. השם של הארכיטקטורה יכול להיות שם כל שהוא ובניסוי זה נשתמש בנוהג המקובל שלפיו השם של הארכיטקטורה יהיה מורכב מהאותיות _arc שלאחריהן מופיע שם הישות.

להלן דוגמה של ארכיטקטורה של ה - Latch שאת הישות שלו תארנו קודם :

```
architecture arc_latch of latch is
begin
  q  <= not ( r or qN ) ;
  qN <= not ( s or q  ) ;
end arc_latch ;
```

כפי שאפשר לראות, הארכיטקטורה הנ"ל כוללת בחלק הביצועי שלה שתי השמות (assignments). החלק ההצהרתי של הארכיטקטורה בדוגמה זו הוא ריק מתוכן .

הישות והארכיטקטורה הנ"ל לא יעברו קומפילציה היות ואסור לרשום בשפת VHDL אות יציאה שמוגדר בישות בכיוון(out כמו q ו)qN בצד ימין של ההשמה. ניתן לפתור את הבעיה בדרך הבאה :

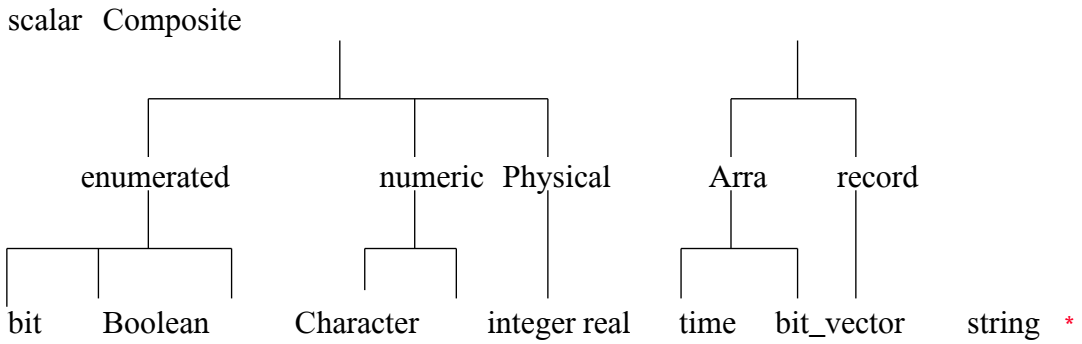
```
-- solution
entity latch is
 port (  s , r  : in  bit  ;
         q , qN : out bit ) ;
end latch ;

architecture arc_latch of latch is
   signal q_int, qN_int : bit  ;
begin
   q_int  <= not ( r or qN_int ) ;
   qN_int <= not ( s or q_int  ) ;
   q <= q_int ; qN <= qN_int ;
end arc_latch
```

בפתרון זה נעשה שימוש בהצהרה על אותות <u>פנימיים</u> q_int ו qN_int בחלק ההצהרתי של הארכיטקטורה.

אותות פנימיים יכולים להיות בכל אחד מהצדדים של פסוק ההשמה.

5. <u>הכרת חלק מסוגי המידע בשפת VHDL</u>

כמו בשפות תכנות, גם ב - VHDL קיימים סוגי מידע שונים. האיור הבא מתאר את המשפחות העיקריות של סוגי המידע שקיימים בשפה.



בחלק התחתון של האיור מופיעה רשימה של סוגי מידע הפנימיים שמובנים)Built-In(בשפה.

סוגי המידע המספריים (numeric data types), שהם חלק מסוגי המידע הסקלריים (החד ממדיים) כוללים את:

- המספרים השלמים - integer
- המספרים הממשיים בנקודה צפה - real

אנו לא נדון כאן בסוג המידע המספרי real. שלמים ב - VHDL נמצאים בתחום הסימטרי הבא סביב אפס:
$-(2^{31}-1) \ldots (2^{31}-1)$

5.1. יצוג מספרים

תת-סוגי מידע (Sub-Types) נוספים של שלמים הם:

- מספרים טבעיים - natural ($0,1,2,..(2^{31}-1)$)
- מספרים חיוביים - positive ($1,2,..(2^{31}-1)$)

להלן דוגמה להצהרה על אות מסוג integer בפורט:

```
port ( din : in integer range 0 to 255 ;
    .
    .                    ) ;
```

להלן דוגמה להצהרות על קבוע מסוג integer ואות פנימי מסוג integer:

```
constant max_count : integer := 255 ;
signal   count     : integer range 0 to max_count ;
```

ב - VHDL קיימים גם סוגי מידע שאינם מספריים (Enumerated Data Types) או סוגי מידע ממוספרים. בסוגי מידע אלו (Enumerated Data Types), הערכים הם ערך כל שהוא מתוך רשימת ערכים סופית אפשרית. שלושת סוגי המידע שעליהם מדובר הם סוגי המידע הבאים:

- סוג המידע bit - כולל ערכים: '0' ו '1'
- סוג המידע boolean - (כולל ערכים: false ו true
- סוג המידע character - כולל תווים כמו 'A', 'B' וכו..

למידע מסוג bit יש שני ערכים אפשריים: '0' ו '1'. זהו סוג המידע הבסיסי שנועד לסינתזה של '0' לוגי ו '1' לוגי של חמרה. שים לב שערכים '0' ו '1' של סוג המידע הזה מוקפים כל אחד בתווי גרש.
סימני 0 ו 1 ללא תווי גרש הם ערכים מסוג integer. ראינו קודם בדוגמאות של ה - Latch הצהרות על אות מסוג bit (אות חיצוני ב - port ואות פנימי בארכיטקטורה). אנו לא נעסוק כאן ב - character.

סוג מידע לא מספרי נוסף, שהוא בעל שני ערכים אפשריים הוא boolean, שיכול לקבל את הערכים false ו true. סוג מידע זה נועד לייצוג של תנאים לוגיים של השפה כמו למשל בתנאי השוואה בפסוקי if. תכניות סינתזה בדרך כלל ממירות אותות מסוג boolean ל - '0' לוגי ו '1' לוגי של חמרה. שלא כמו הערכים '0' ו '1' של bit, הערכים false ו true אינם מוקפים בתווי גרש. להלן דוגמה להצהרה על אות פנימי מסוג boolean.

signal flag : boolean ;

עד לשלב זה תארנו סוגי מידע פשוטים בעלי ממד אחד. לסוגי מידע אלו קוראים גם בשם "סוגי מידע סקלריים" (Scalar Data Types). אפשר לתאר בשפת VHDL גם סוגי מידע מורכבים יותר שנקראים (Composite Data Types). מדובר בשתי משפחות של סוגי מידע מורכבים:

- Arrays- מערכים
- Records - רשומות

מערך הוא אוסף של אלמנטים הומוגני, כלומר מסוג מידע אחיד. נוח להשתמש במערכים, מכיוון שקל לבצע באמצעותם פעולות באופן גלובלי על כל המערך. כמובן שניתן לבצע פעולות גם על כל אלמנט של המערך בנפרד. המשתמש יכול להצהיר בעצמו על מערכים מסוגים שונים. אנו לא נעסוק במערכים מסוג זה.

קיימים שני סוגי מערכים שכבר מובנים בשפה. מדובר בשני סוגי המערכים הבאים:

- מערך של אותות מסוג bit שנקרא bit_vector
- מערך מסוג character שנקרא string

אנו לא נעסוק כאן ב - strings. סוג המידע bit_vector הוא מערך של אלמנטים מסוג bit. להלן דוגמאות להצהרה על אותות מסוג bit_vector בישות.

```
entity bv4 is
port (  din  : in  bit_vector(7 downto 0) ;
  dout : out bit_vector(7 downto 0) ) ;
 end bv4 ;
```

בדרך כלל אנו נעדיף להגדיר תחום אינדקסים של וקטורים בכיוון יורד (downto) ולא בכיוון עולה (to) היות ונעדיף שהסיבית בעלת האינדקס הגבוה (MSB) תהייה בצד שמאל.

וקטור קבוע מורכב מאוסף תווי '0' ו '1' שמוקפים בגרשיים. להלן דוגמה להשמה לאות dout מהסוג שהוגדר קודם.

dout <= "**00110101**" ;

רשומה היא אוסף של אלמנטים הטרוגני (מסוגי מידע שונים). שלא כמו במקרה של מערכים, אין בשפת VHDL רשומות, שהם חלק מובנה בשפה. המשתמש יכול להגדיר סוגי מידע כאלו בעצמו.
אנו לא נעסוק כאן ברשומות.

לקבלת מידע נוסף על סוגי המידע של שפת VHDL והיכולת ליצור סוגי מידע חדשים בשפה תוכל להיעזר בספרות.

6. פעולות בשפת VHDL

בניגוד לשפות כמו שפת C, שפת VHDL היא שפה נוקשה מבחינת סוגי המידע והפעולות שניתן לבצע על סוגי המידע הללו. לשפה כזו קוראים Strongly Typed Language.

הנוקשות של השפה מבחינת סוגי המידע והפעולות עשויה להתבטא באופנים הבאים:

- אופרטורים ניתנים להפעלה רק על אופרנדים מסוגים מסוימים
- ברוב הפעולות אי אפשר לערבב בין שני סוגי אופרנדים בשני צדי האופרטור. שני הצדדים של פעולת ההשמה גם הם חייבים להיות בעלי אותו סוג מידע. אין המרות (Casting) אוטומטיות כאשר מחשבים ביטויים או מבצעים השמות

היתרון של שפות שהן Strongly Typed הוא, שהמשתמש מוגן יותר כנגד שגיאות שהוא כותב בתכנית. החסרון של שפות מסוג זה, הוא שהן מסורבלות יותר לכתיבה, ודורשות זמן לימוד ארוך יותר. להלן אוסף הפעולות הלוגיות שקיים בשפה:

and, or, not, xor, nand, nor, (xnor in VHDL-93)

את הפעולות הנ"ל לא ניתן לבצע על שלמים אלא על אופרנדים מסוג bit, bit_vector ו boolean.
בנוסף לכך שני האופרנדים חייבים להיות מאותו הסוג, כלומר לא ניתן לערבב למשל בין boolean ו bit או בין bit ו bit_vector או בין שני אופרנדים מסוג bit_vector ברוחב שונה. בנוסף לכך אסור לבצע בשפת VHDL רצף של פעולות לוגיות שונות כמו בשורת הקוד הבאה:

```
x <= a and b or c ;  -- error !!!
```

הפתרון של הבעיה הנ"ל הוא להשתמש בסוגריים

```
x <= ( a and b ) or c ;
```

או להשתמש באותות ביניים פנימיים. למרות מה שנאמר כאן, מותר לבצע פעולות ברצף כאשר מדובר באופרטורים זהים מסוג and, or ו xor. להלן אוסף הפעולות החשבוניות שקיים בשפה:

+ , - , * , / , rem , mod , ** , abs( )

שתי הפעולות rem ו mod הן פעולות דומות והשוני ביניהן קיים רק כאשר מדובר באופרנדים שאינם חיוביים. זוג כוכביות צמודות מסמן את אופרטור החזקה. הפונקציה abs מפיקה את הערך המוחלט של מספר. את הפעולות הנ"ל ניתן להפעיל על שלמים אך לא ניתן להפעילם על סוגי מידע אחרים כמו וקטורים ! בהמשך נראה כיצד פותרים בעיה זו.

אופרטור חשוב נוסף הוא אופרטור השרשור & (Concatenation). אופרטור זה מסוגל להדביק תווים ווקטורים ביחד לקבלת וקטורים רחבים יותר באופן הבא (האות dout בדוגמה הוא bit vector ברוחב 8)

```
dout <= "1111" & "011" & '0' ;
```

פעולות נוספות שאותן ניתן לבצע על סוגי מידע רבים הן פעולות ההשוואה (או הרלציה - relations). להלן אוסף הפעולות שקיים בשפה:

= , < , <= , > , >= , /=

הפעולה הימנית היא פעולת חוסר השוויון)not equal(. רלציות יכולות להיעשות הן על שלמים והן על ביטים או ווקטורים אך לא באופן מעורבב. רלציות תמיד מחזירות סוג מידע boolean כלומר false או true

6.1. השמות (assignments) בשפת VHDL

כבר הכרנו את אופרטור ההשמה לאות)signal(שנראה כך :

signal_name <= expression ;

השמה למשתנה)variable(- שהוא אובייקט)**שאותו עדיין לא הצגנו ויוצג בהמשך**(נראית כך :

variable_name := expression ;

כאשר מבצעים השמה כל שהיא, שני צדי ההשמה חייבים להיות מסוג מידע זהה.

נציג כמה צורות זהות לביצוע השמות לאות וקטורי בשם dout וברוחב שמונה סיביות.

```
dout <= "11110110" ;

dout(7)<= '1' ; dout(6)<= '1' ; dout(5)<= '1'; dout(4)<= '1' ;
dout(3)<= '0' ; dout(2)<= '1' ; dout(1)<= '1'; dout(0)<= '0' ;

dout <= ('1','1','1','1','0','1','1','0') ;

dout <= ( 0 => '0', 1 => '1', 2 => '1', 3 => '0',
          4 => '1', 5 => '1', 6 => '1', 7 => '1');

dout <= ( 4 => '1', 1 => '1', 6 => '1', 3 => '0',
          0 => '0', 5 => '1', 2 => '1', 7 => '1');

dout <= ( 0 => '0', 3 => '0', others => '1' ) ;
```

ההשמה הראשונה היא השמה וקטורית רגילה. בקטע הקוד הבא יצרו שמונה השמות נפרדות לכל האלמנטים של המערך dout. בקטע הקוד השלישי יצרו בצד ימין של ההשמה אגרגציה)מקבץ של אלמנטים(. הקישור בין האלמנטים משני צדי ההשמה נעשה על פי המקום Positional Association)כלומר האלמנט השמאלי בצד שמאל קיבל את האלמנט השמאלי בצד ימין וכו..(בשתי ההשמות הבאות יצרו קישור בין שני צידי ההשמה באמצעות השם של האינדקס)Named association(. שים לב שבהשמות מסוג זה ניתן לשנות את סדר ההופעה של האלמנטים בצד ימין. בהשמה האחרונה משתמשים ב - others, שמנתב את האלמנטים לפי כל האינדקסים האחרים שלא צוינו קודם.

שימוש ב - others נפוץ גם כאשר רוצים למשל לאפס וקטור. במקרה זה במקום לרשום את ההשמה הראשונה הבאה, נוכל לרשום את ההשמה השניה :

```
y <= "00000000" ;          -- a VHDL beginner
y <= ( others => '0' ) ;          -- a true VHDLnik
```

בהשמות בין וקטורים ניתן להשתמש גם בשכבה)Slice(שמעביר חלק מהווקטור .

```
y(7 downto 4) <= x(3 downto )0 ;
y(3 downto 0) <= x(7 downto )4 ;
```

6.2. סוגי מידע חשובים שהתווספו לשפה באמצעות החבילה std_logic_1164

הדלות היחסית של סוגי המידע bit ו bit_vector, שהם בעלי ערכים של '0' ואחד '1' מפתיעה. היינו מצפים שניתן יהיה לתאר באמצעות סוג מידע כמו bit גם ערכים כמו High-Z)מצב יציאה אפשרי של נתק שקיים ברכיבי Tri-State או Open-Drain(. אופציה חיונית נוספת היא אולי תוצאת סימולציה בלתי ידועה במקרה של קונפליקט בין שתי יציאות. כמו כן היינו מצפים שניתן יהיה לבצע למשל פעולות אריתמטיות על סוג המידע bit_vector. אמנם שפת VHDL הבסיסית ביותר אינה מספקת לנו יכולות מסוג זה, אך בקלות רבה ניתן להשתמש במנגנונים של השפה בכדי להרחיב אותה, כך שהיא תכלול יכולות אלו ויכולות רבות נוספות.

המנגנון שבו מרחיבים את השפה מבוסס על יחידה שנקראת חבילה)Package(ואמנם במשך השנים נוצרו חבילות שימושיות רבות.

נציג תחילה את החבילה החשובה שנקראת std_logic_1164. חבילות זו מכילה הצהרות על סוגי מידע חדשים שנקראים std_logic)שהוא הרחבה של bit(ו std_logic_vector)שהוא הרחבה של bit_vector(וסוגים נוספים שאותם לא נזכיר כאן. בניגוד לסוגי המידע bit ו bit_vector שבהם
קיימים רק שני מצבים לוגיים אפשריים של '0' ו '1' , בסוגי המידע החדשים קיימת גמישות גדולה יותר, וניתן לתאר אותם באמצעות תשעה מצבים לוגיים אפשריים. להלן המשמעות של תשעת המצבים האפשריים הנ"ל.

- מצב לוגי בלתי מאותחל 'U'
- מצב לוגי בלתי ידוע 'X'
- מצב לוגי של 0 חזק '0'
- מצב לוגי של 1 חזק '1'
- מצב של נתק חשמלי 'Z'
- מצב לוגי בלתי ידוע חלש 'W'
- מצב לוגי של 0 חלש 'L'
- מצב לוגי של 1 חלש 'H'
- צירוף ברירה)Don't care(עבור סינתזה '-'

המצבים הלוגיים '0' ו '1' הם מצבים לוגיים חזקים שנגרמים ביציאה של רכיב ספרתי כתוצאה מהולכה של טרנזיסטור Pull-Down ל - GND או הולכה של טרנזיסטור Pull-Up ל - VCC בהתאמה. המצב 'Z' מתאר נתק)High-Z(. זהו מצב שעשוי להופיע ביציאה מסוג Open-Drainאו Tri-State. המצבים הלוגיים החלשים נוצרים ביציאה מנותקת שבה קיים נגד Pull-Up שמחובר ל - VCC או נגד Pull-Down שמחומר ל - GND.

המצב 'X' הוא מצב בלתי ידוע בסימולציה. מצבים בלתי ידועים יכולים להיווצר בצורות שונות, למשל באמצעות חיווט בין יציאות שנמצאות בקונפליקט)במצבים לוגיים מנוגדים '0' ו '1'(ביחד)Wired-Logic-BUS(. אפשרות אחרת להיווצרות מצב בלתי ידוע הוא למשל ביציאה של שער שבכניסות שלו מוזנות למצבים בלתי ידועים)למשל יציאה של שער NOT שהכניסה שלו מוזנת 'Z'(. או - 'X' ב

המצב שבו כל האותות נמצאים בתחילת הסימולציה הוא 'U'. בעצם שני המצבים 'U' ו 'X' הם מצבים בלתי ידועים. ההבדל ביניהם הוא שאות שנמצא במצב 'U' הוא אות שמצבו אינו

ידוע מתחילת הסימולציה. המצב 'U' מהווה אינדיקציה חשובה לחוסר אתחול (Initialization) של אות.

המצב 'W' הוא מצב בלתי ידוע חלש. מצב כזה נוצר כאשר כמה יציאות חלשות שמחווטות ביחד ונמצאות במצב של קונפליקט (מצב 'H' ו 'L').

המצב '-' נועד לכלי סינתזה. כאשר מתארים מצב זה ביציאה, קיימת דרגת חופש למתכנן בקביעת מצב היציאה (Don't Care). למרות שמצב זה יועד לכלי סינתזה, אין וודאות שכלי סינתזה אכן תומכים בו. כלי סינתזה תמיד תומכים בשלושת המצבים '0' , '1' ו 'Z'. המצבים 'U', 'X', 'W', 'L' ו 'H' שנקראים לפעמים - Meta-Values, אינם מיועדים לסינתזה אלא לסימולציה בלבד.

כפי שכבר צוין קודם ובניגוד לסוגי המידע bit ו bit_vector, שני סוגי std_logic ו std_logic_vector הם סוגי מידע שמאפשרים ליצור גם חיווט בין יציאות (Wired Logic) ומאפשרים למצוא את המצב הלוגי של נקודת החיווט המשותפת.

החבילה std_logic_1164 מכילה גם תיאורים של פונקציות שנקראות בשמות זהים לאופרטורים שמובנים בשפה כמו "and", "or" ו "not". שים לב שאלו הם בדיוק האופרטורים או הפונקציות שבהם ניתן להשתמש עם סוגי המידע bit ו bit_vector. בשפת VHDL ניתן לתת לאופרטור כמה משמעויות (Operator overloading). זוהי תכונה מאוד חזקה של שפת VHDL, שעוזרת להרחיב את היכולת של השפה בצורה נוחה לשימוש. הגדרת הפעולות הנ"ל ("and", "or" ו "not") על סוג המידע החדשים שבחבילה std_logic_1164 מרחיבה בעצם את היכולת של השפה, כך שניתן להשתמש בסוגי המידע std_logic ו std_logic_vector בכל מה שנוגע לפעולות לוגיות בדיוק כמו ב - bit ו bit_vector בהתאמה.

מדובר לא רק בפעולות הלוגיות הנ"ל. בעצם כל הפעולות והתכונות שאותן הכרנו על סוג המידע bit_vector כמו: שרשור (&), שימוש באגרגציות בביטויים, שימוש בהשמות עם Positional, to, downto "|" ושימוש בתחומים של אינדקסים מסוג Named Association ו Association בדיוק באותה צורה כפי שהם פועלים על std_logic_vector פועלים על bit_vector. others.

כל מה שהמשתמש צריך לעשות על מנת להשתמש בחבילה החשובה הזו הוא להפוך את כל התכולה של החבילה לנראית (Visible), פעולה זו נעשית באמצעות רישום שתי השורות הבאות לפני הישות:

```
library ieee ;
use ieee.std_logic_1164.all ;
```

להלן דוגמה של שער OR שמתואר באמצעות סוג המידע std_logic.

```
library ieee ; use ieee.std_logic_1164.all ;

entity or2 is
   port (  a , b   : in  std_logic ;
           y       : out std_logic ) ;
end or2 ;

architecture arc1_or2 of or2 is
begin     y <= a or b ;
 end arc1_or2 ;
```

אם נזין למשל את אחת מהכניסות של השער במצב 'Z')נתק(או 'X')מצב בלתי ידוע(נקבל ביציאה מצב בלתי ידוע 'X' . אם למשל נזין את השער ב - 'H')אחד לוגי חלש שנובע מנגד Pull-Up שמחובר ל - VCC(היציאה תימצא במצב '1'. אלו הן התנהגויות שמתארות חמרה מעשית אמיתית של שער OR.

6.3. ביצוע פעולות חשבוניות על וקטורים

נאמר כבר קודם שלא ניתן לבצע פעולות חשבוניות על וקטורים מסוג bit_vector. שימוש בחבילות הבאות std_logic_unsigned או std_logic_signed שינה את המצב הנ"ל. החבילה הראשונה נותנת לוקטורים מסוג std_logic_vector משמעות חשבונית של מספר בינארי חסר סימן והחבילה השניה נותנת לו משמעות של מספר בעל סימן שמיוצג במשלים ל - 2.

להלן דוגמה למחבר)adder(:

```
-- overloading "+" to std_logic_vector
library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_unsigned.all ;
 entity op2_arith is
port ( a,b : in  std_logic_vector(3 downto )0 ;
       sum : out std_logic_vector(3 downto 0));
 end op2_arith ;
architecture arc_op2_arith of op2_arith is
begin
 sum <= a + b ;
end arc_op2_arith
```

הדוגמה הנ"ל מציגה פעולת חיבור בין וקטורים. ניתן לבצע גם פעולות חיבור מעורבות כמו בדוגמה הבאה:

std_logic_vector_value + integer_value

std_logic_vector_value + std_logic_vlaue

להלן דוגמה למימוש משווה)Comparator(שהכניסות הוקטוריות שלו מייצגות מספרים בעלי סימן במשלים ל - 2.

```
-- signed directional comparator
 library ieee ;
 use ieee.std_logic_1164.all ;
 use ieee.std_logic_signed.all ;
 entity op2_rel is
   port ( a,b    : in  std_logic_vector(3 downto )0 ;
          a_gt_b : out boolean                    ) ;
  end op2_rel ;
architecture arc_op2_rel of op2_rel is
begin
   a_gt_b <= (a > b);
end arc_op2_rel ;
```

ניתן להתייחס לחלק מווקטור ע"י ציון האינדקסים של הסיביות שבהן מעוניינים:
Partial_vector(5...2)

לקבלת מידע נוסף על החבילות השימושיות הרבות שמרחיבות את השפה תוכל להיעזר בספרות.

7. הכרה בסיסית של התהליך והמשתנה

עד לשלב זה רשמנו בארכיטקטורה השמות בלבד. סדר רישום ההשמות בארכיטקטורה הוא אינו חשוב היות וכל ההשמות מתבצעות במקביל אחת לשניה.

ארכיטקטורה יכולה להכיל גם יחידות ביצוע שנקראת תהליך)process(. התהליך מאפשרת ליצור תיאורים שנעשים באופן סדרתי)Sequential(. הקוד בתוך התהליך יכול להיות דומה לקוד שקיים בשפות תכנות והוא יכול לכלול למשל: התניות if ו case, חוגים וכן משתנים)Variables(.
באופן כזה ניתן בקלות ליצור בתוך תהליך תיאורים אלגוריתמיים מורכבים.

להלן המבנה התחבירי הבסיסי של סוג התהליך הנפוץ ביותר)תהליך בעל רשימת רגישות(:

```
process ( sensitivity list )
 -- declarative section
   :
   :
begin
   -- sequential statements section
   :
   :
end process ;
```

השורה הראשונה של התהליך מתארת רשימת רגישות)sensitivity list(. רשימת הרגישות מכילה אוסף של אותות. תהליך יכול להימצא באחד משני מצבים אפשריים: תהליך מופעל)Active(או תהליך מופסק)Suspended(. בכל פעם שאחד מהאותות ברשימת הרגישות משתנה, התהליך מופעל וסדרת הפסוקים שרשומים בתהליך מתבצעת. בתום הביצוע, התהליך מופסק עד להפעלתו המחודשת.

כפי שאפשר לראות, התהליך מחולק לשני חלקים. החלק שנמצא בין השורה הראשונה של התהליך והפסוק begin הוא חלק הצהרתי, שיכול להכיל למשל הצהרות על משתנים וקבועים מקומיים של התהליך.

בהמשך קטע הקוד הנ"ל, כלומר בין המלה begin והשורה end process, מצוי החלק הביצועי של התהליך. חלק זה מתאר באמצעות פסוקי השמה ופסוקים אפשריים נוספים את אוסף הפעולות שהתהליך צריך לבצע כאשר הוא מופעל. החלק הביצועי של תהליך יכול להכיל השמות רבות שיכולות להיות משולבות עם פסוקים של בקרת זרימה)flow control(כמו פסוקי if, פסוקי case פסוקי loop שונים ופסוקים נוספים.

מרגע שהתהליך מופעל באמצעות שינוי של אות כל שהוא ברשימת הרגישות התהליך מתבצע באופן סדרתי. ההשמות לאותות שרשומות בתהליך אינן נעשות באופן מידי במהלך ביצוע התהליך. ההשמות מתבצעות במקביל מיד לאחר שהתהליך מופסק. דוגמת הקוד הבא תמחיש לך מה שנאמר כאן.

```
-- intermediate signal in process ?
entity x4 is
  port ( a,b,c : in  bit ;
         y     : out bit ) ;
 end x4 ;

architecture arc_x4 of x4 is
   signal temp : bit ;
begin
  process (a,b,c)           -- bad simulation
    begin
     temp <=    a xor b ;
       y <= temp xor c ;
   end process ;
end arc_x4 ;
```

קטע הקוד הנ"ל היה אמור לבצע פעולת xor בין שלוש אותות a,b ו c באמצעות "אות ביניים" שנקרא temp. אולם מערכת זו אינה מתפקדת באופן כזה בסימולציה. להלן דוגמה לתוצאות סימולציה:

ns	a	b	c	temp	y	
0	0	0	0	0	0	
100	0	0	1	0	1	error
200	0	1	0	1	0	→
300	0	1	1	1	0	
400	1	0	0	1	1	
500	1	0	1	1	0	
600	1	1	0	0	1	→ error
700	1	1	1	0	1	

בכל פעם ש - temp משתנה מתקבלת שגיאה בסימולציה. מדוע ?

היות והשמות לאותות אינן מתבצעות באופן מידי במהלך ביצוע התהליך, האות temp אינו יכול לעדכן בהשמה השניה את האות y. ההשמה ל - temp ו y נעשות בעצם בו זמנית בהפסקת התהליך, והערך של temp הישן הוא הערך שנלקח בחשבון בהשמה ל - y. זוהי הסיבה לכך שבכל פעם שבה temp היה צריך להתעדכן בטבלה הנ"ל, קבלנו ערך שגוי של y.

שלא כמו במקרה של אות, למשתנה)Variable(, שהוא אובייקט חישובי בלתי מתוזמן, יש משמעות דומה לזו שמוכרת לנו ממשתנים בשפות תכנות. ההשמות למשתנה נעשות באופן מיידי תוך כדי הביצוע הסדרתי של התהליך. משתנים אכן מיועדים לביצוע של חישובי ביניים. להלן דוגמת קוד שמחשבת את ערך הביניים temp ומעבירה אותו להשמה לאות היציאה y באמצעות משתנה שנקרא temp.

```
- variable in process for intermediate calculations
architecture arc_x7 of x7 is
begin
 process (a,b,c)
    variable temp : bit ;  -- OK
    begin
 temp :=    a xor b ;
      y <= temp xor c ;
  end process ;
end arc_x7 ;
```

שים לב בקוד הנ"ל, למיקום של ההצהרה על המשתנה. ההצהרה נעשית בחלק ההצהרתי של התהליך. שים לב גם לאופרטור ההשמה למשתנה)=:(שהוא אופרטור שונה מאופרטור ההשמה לאות)=<(. שפת VHDL מאפשרת לבצע השמה של ערכים של אותות למשתנים ולבצע השמה של ערכים של משתנים לאותות. בדוגמת הקוד הנ"ל המשתנה temp מקבל באופן מידי את הערך של a xor b והוא יכול להעביר אותו לאות y בזמן ביצוע ההשמה שרשומה בהמשך.
בקטע הקוד הבא נעשה שילוב בין שני סגנונות תיאור. שער xor אחד מתואר באמצעות השמה)ללא תהליך(ושער xor נוסף מתואר באמצעות השמה שרשומה בתהליך.

```
-- mixing sequential & data-flow styles
architecture arc_x13 of x13 is
   signal temp : bit ;
begin
  y <= temp xor c ;
  process (a,b)
   begin
       temp <= a xor b ;
  end process ;
end arc_x13 ;
```

סדר הכתיבה של שני החלקים הנ"ל עלול להיראות במבט ראשון כבלתי הגיוני, כלומר קודם מעדכנים את האות y על סמך האות temp, ורק לאחר מכן מעדכנים את האות temp. הסדר "ההפוך")כביכול(לא יוצר בעיה כל שהיא, מכיוון ששני השערים שמתוארים בארכיטקטורה מתפקדים בעצם במקביל. במלים אחרות - התהליך אמנם מתבצע בתוכו באופן סדרתי, אך כלפי חוץ התהליך מתבצע במקביל לשאר הפסוקים בארכיטקטורה. קטע הקוד הבא מתאר את אותה המערכת שוב, אך הפעם משתמשים בשני תהליכים.

```
-- 2 processes in parallel
architecture arc_x14 of x14 is
   signal temp : bit ;
begin
process (temp,c) -- process can have a label
   begin
    y <= temp xor c ;
   end process ;
    process (a,b)
    begin
       temp <= a xor b ;
   end process ;
end arc_x14 ;
```

בקובץ הנ"ל החליפו את התיאור של שער ה - XOR שמייצר את האות y, בתיאור של השמה בארכיטקטורה, לתיאור שנעשה באמצעות תהליך. אפשר להתייחס לכל השמה שנרשמת ישירות בארכיטקטורה כאל צורת רישום מקוצרת של תהליך שרשימת הרגישות שלה מכילה את כל האותות שנמצאים בצד ימין של ההשמה.

מנועי סימולציה של שפת VHDL מריצים בסך הכל סימולציה של אוסף של תהליכים שמתבצעים במקביל אחד לשני.

שים גם לב בדוגמה האחרונה לשימוש שנעשה באות הפנימי temp. אותות מאפשרים לקשר בין תהליכים. משתנים אינם מסוגלים לבצע תפקיד כזה. התפקיד של משתנים הוא בסך הכל לאפשר ביצוע חישובי ביניים מורכבים בתוך התהליך.

7.1. <u>פסוקי if בתוך תהליך</u>

פסוקי if אינם יכולים להירשם ישירות בארכיטקטורה אלא רק בתוך תהליך שמצוי בארכיטקטורה. פסוקי if הם בעלי התחביר הבסיסי הבא:

```
 if  Boolean_condition then – sequential statements
   .
[ elsif Boolean_condition then
   . -- sequential statements
   .                    ]
[ elsif Boolean_condition then
   . -- sequential statements
   .                    ]
[ else
   . -- sequential statements
   .                    ]
   .
   .
 end if ;
```

כאשר תנאי הבוליאני (*boolean_condition*) הוא true, מתבצע גוש הקוד שמופיע מיד אחריו. התנאים הבוליאניים שנבדקים בכל אחד מהחלקים של הפסוק לא צריכים להיות קשורים אחד לשני. כאשר כמה תנאים מתקיימים בו זמנית התנאי הראשון הוא זה שקובע ולכן הסדר של החלקים השונים בתוך הפסוק הנ"ל עצמו עשוי להיות חשוב. הפסוקים הפנימיים מסוג elsif ו else הם פסוקים אופציונליים. שים לב לכתיב המיוחד של המלה elsif שאינה נרשמת עם התו "e" באופן הבא "elseif" וגם אינה נרשמת באמצעות שתי מלים נפרדות באופן הבא: "else if". אם משתמשים בפסוק else, הוא נרשם כחלק אחרון וללא תנאי בוליאני כל שהוא. פסוקים של החלק else, מתקיימים כאשר כל התנאים הבוליאניים שלפני

כן אינם מתקיימים. פסוק else הוא פסוק ברירית המחדל. מותר לבצע Nesting של פסוקי if בתוך פסוקי if אחרים.

להלן דוגמת קוד שמתארת משווה (comparator) פשוט:

```
architecture arc_ifexample of ifexample is
begin
 process (a,b)
   begin
     if a = b then
          equal <= '1' ;
      else
       equal <= '0' ;
    end if ;
   end process ;
 end arc_ifexample ;
```

7.2. פסוקי case בתוך תהליך

גם פסוקי case אינם יכולים להירשם ישירות בארכיטקטורה אלא רק בתוך תהליך שמצוי בארכיטקטורה. פסוקי case הם בעלי התחביר הבסיסי הבא:

```
case tested_expression is   when constant_val =>
  . -- sequential statements
  .
[ when constant_val =>
  . -- sequential statements
  .                 ]
[ when constant_val =>
  . -- sequential statements
  .                 ]
[when others      =>
  . -- sequential statements
  .                 ]
end case ;
```

מבחינת התחביר של פסוקי case, אפשר להסתפק בערך קבוע (*constant_val*) אפשרי אחד בלבד,
ושאר הערכים הם אופציונליים. כדי שניתן יהיה להשתמש בפסוק case, יש להקפיד לכסות את כל הערכים האפשריים שהביטוי הנבדק יכול לקבל, אחרת תתקבל הודעת שגיאה בקומפילציה.
בנוסף לכך אסור לרשום ברשימת האפשרויות, אפשרות מסוימת יותר מפעם אחת. מותר לרשום פסוקי case בתוך פסוקי case ו if אחרים (Nesting).
להלן דוגמה שמתארת שער and באמצעות פסוק case.

```
architecture arc_case_example of case_example is
begin
   process (a,b)
       variable din : bit_vector(1 downto 0) ;
   begin
      din := a & b ;
      case din is
         when "11"    => y <= '1' ;
        when others => y <= '0' ;
      end case ;
   end process ;
 end arc_case_example ;
```

היות וחייבים לרשום בפסוק case את כל הערכים האפשריים של הביטוי הנבדק בפסוק case והיות ורשימה זו עלולה להיות גדולה, פסוק case כולל מנגנונים שונים שמאפשרים לנו לכסות מספר גדול של אפשרויות בנוחות.

הנוחות מושגת באמצעות קיבוץ של כמה אפשרויות לקבוצה אחת שדורשת השמה אחידה. ניתן להשתמש בתו ההפרדה "|" או בפסוק others. כאשר משתמשים בפסוק case שמשווה ערך של ביטוי שמחזיר שלמים (integer) לרשימה של קבועים שלמים, ניתן להשתמש גם בתחומים (to ו downto).

כל האפשרויות הנ"ל מוצגות בקטע התחבירי הבא:

```
case tested_expression is
   when val1 | val2 | val3 ..    =>     .
-- sequential statements
      when integer_val1    to    integer_val2 =>
 -- only integers     . -- sequential statements
       when integer_val3 downto integer_val4 =>
 -- only integers     . -- sequential statements
     when others                              =>
   . -- sequential statements     .
end case ;
```

7.3. התניה ובחירה בארכיטקטורה - ללא תהליך

ניתן ליצור התניה גם ללא שימוש בתהליך ובפסוק if וזאת ישירות בארכיטקטורה. להלן דוגמה של משווה שמתואר ישירות בארכיטקטורה:

```
architecture arc_comp2 of comp2 is
begin
     equal <= '1' when (a = b) else '0' ;
  end arc_comp2 ;
```

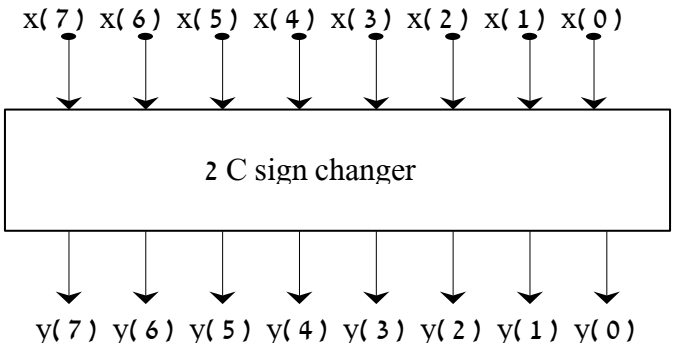
להשמה כזו קוראים השמה מותנית (conditional assignment). אסור להשתמש בהשמה מותנית בתוך תהליך.
גם לפסוק case יש שווה ערך לרישום ישירות בתהליך (מחוץ לתהליך). להלן דוגמה לתיאור מפלג (De-Multiplexer) בעל ממדים 1=>4 ישירות בתוך הארכיטקטורה.

```
architecture arc_demux1 of demux1 is
 begin
   with sel select
   dout <= ('0', '0', '0', din) when "00"    ,
           ('0', '0', din, '0') when "01"    ,
           ('0', din, '0', '0') when "10"    ,
           (din, '0', '0', '0') when others ;
 end arc_demux1 ;
```

לצורת רישום כזו קוראים השמה נבחרת (Selected assignment). אסור להשתמש בהשמה נבחרת בתוך תהליך.

7.4. יצירת חוג (Loop) בתהליך

קיימים בשפת VHDL שני סוגי חוגים בתוך תהליך: חוג for וחוג while. נדגים כאן חוג מסוג for בלבד.
הקובץ הבא מדגים שימוש בחוג for. בדוגמה זו רוצים ליצור מערכת שמוזנת באמצעות וקטור כניסה (x) ברוחב שמונה סיביות ומפיקה וקטור יציאה (y) ברוחב שמונה סיביות.



וקטור הכניסה)x(ווקטור היציאה)y(מיצגים מספרים בעלי סימן בשיטת המשלים ל - 2.
המערכת מבצעת את הפעולה החשבונית הבאה)היפוך סימן(: X = -Y
להלן הקוד :

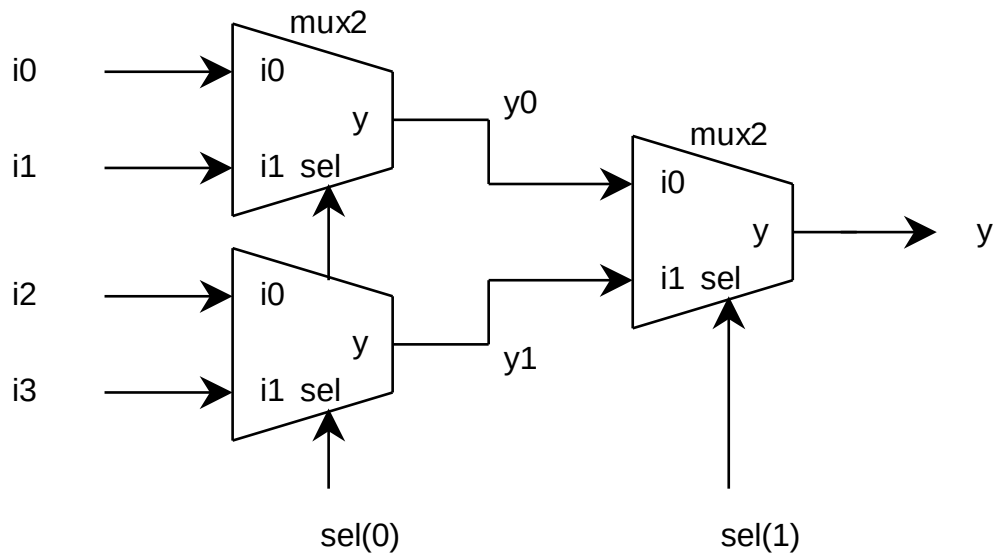
```
architecture arc_invert2c of invert2c is
begin
  process ( x )
    variable invert : boolean ;   -- inversion flag
    begin
        invert := false ;
     for i in 0 to 7 loop
         if invert then
            y(i) <= not x(i) ;       -- inverting bits
         else
            y(i) <= x(i) ;           -- passing bits
            if x(i) = '1' then
                invert := true ;
            end if ;
         end if ;
     end loop ;
  end process ;
end arc_invert2c ;
```

לקבלת מידע נוסף על תיאורים התנהגותיים אפשריים בתוך ומחוץ לתהליך תוכל להיעזר בספרות.

8. תכן הירארכי

במקרים רבים משתמשים בקוד מספר רב של פעמים, במקרה יש יש "לבנות" הירארכיה של אבני בניה ולחבר בינהם בחוטים, בדומה למה שבוצע במעבדות הקודמות בצורה סכמתית
כתיבה זו מבוצעת על ידי הפונקציה PORT MAP כמו בדוגמה הבאה המתארת בורר כהיררכיה מבנית שמורכבת מבוררים קטנים יותר בעלי ממדים 2 =>1.

```
entity mux2 is
   port( i1 , i0 , sel : in  bit;
          y      : out bit ) ;
end mux2 ;

architecture arc_mux2 of mux2 is
begin
  y <= ( not sel and i0 ) or ( sel and i1 ) ;
end arc_mux2 ;
```



קטע הקוד הבא מתאר באופן מבני את ההיררכיה הנ"ל.

```
entity mux4x1 is
   port( i3, i2, i1, i0 : in bit  ;
      sel : in  bit_vector(1 downto 0);
      y  : out bit  ) ;
end mux4x1 ;

architecture arc_mux4x1 of mux4x1 is
    component mux2 is
        port( i1 , i0 , sel : in  bit;
                y      : out bit ) ;
    end component ;

signal y0, y1 : bit ;

begin
    u0: mux2 port map (i0=>i0,i1=>i1,sel=>sel(0),y=>y0);
    u1: mux2 port map (i0=>i2,i1=>i3,sel=>sel(0),y=>y1)  ;
    u2: mux2 port map (i0=>y0,i1=>y1,sel=>sel(1),y=>y )  ;
end arc_mux4x1 ;
```

יש להגדיר את כל אבני הבניה כ COMPONENTS

בצורה דומה ניתן לייצר MUX שעובד על ווקטור ברוחב 4

```
entity mux2x4 is
   port( i1 , i0 : in bit_vector(3 downto 0) ;
      sel : in  bit;
          y     : out bit_vector(3 downto 0) ) ;
end mux2x4 ;

architecture arc_mux2x4 of mux2x4 is
    component mux2 is
        port( i1 , i0 , sel : in  bit;
          y     : out bit ) ;
    end component ;

begin
    u0: mux2 port map (i0=>i0(0),i1=>i1(0),sel=>sel,y=>y(0));
    u1: mux2 port map (i0=>i0(1),i1=>i1(1),sel=>sel,y=>y(1));
    u2: mux2 port map (i0=>i0(2),i1=>i1(2),sel=>sel,y=>y(2));
    u3: mux2 port map (i0=>i0(3),i1=>i1(3),sel=>sel,y=>y(3));
end arc_mux2x4 ;
```

8.1. תכן מבני עם פרמטרים

ניתן להגדיר בשפת VHDL רכיבים עם פרמטרים גנריים (generic parameter). להלן דוגמת קוד של בורר 2=>1 וקטורי שהרוחב שלו נקבע על ידי פרמטר שנקרא width.

```
-- vector 2=>1 mux with generic width
entity mux2v is
   generic ( width  : integer := 8 ) ;
   port    ( a , b  : in  bit_vector(width-1 downto 0) ;
             sel    : in  bit                          ;
             y      : out bit_vector(width-1 downto 0) ) ;
end mux2v ;
architecture arc_mux2v of mux2v is
begin
   y <= a when sel = '1' else b ;
end arc_mux2v ;
```

בקטע הקוד הבא מחווטים שלושה רכיבים וקטוריים (mux2v) לקבלת בורר 4=>1 וקטורי.

```
-- wiring 4=>1 generic mux
entity mux4v is
   port ( a , b , c , d : in  bit_vector(7 downto 0) ;
          sel           : in  bit_vector(1 downto 0) ;
          y             : out bit_vector(7 downto 0) ) ;
end mux4v ;
architecture arc_mux4v of mux4v  is
   component mux2v
      generic ( width  : integer := 8 ) ;
      port    ( a , b  : in  bit_vector(width-1 downto 0) ;
                sel    : in  bit                          ;
                y      : out bit_vector(width-1 downto 0) ) ;
   end component ;
   signal q , r : bit_vector(7 downto 0) ;
begin
   u0: mux2v generic map ( 8 )                    -- positional
             port map ( a , b , sel(0) , q ) ;    -- association
   u1: mux2v generic map ( width => 8 )           -- named
             port map ( a   => c     ,            -- association
                        b   => d     ,
                        sel => sel(0),
                        y   => r     ) ;

   u2: mux2v port map ( q , r , sel(1) , y ) ;    -- default width
end arc_mux4v  ;
```

בפסוק החיווט הראשון נעשה שימוש בקישור של פרמטרים ואותות על פי המקום. בפסוק החיווט השני נעשה קישור של פרמטרים ואותות על פי שם. בפסוק האחרון נעשה שימוש בפרמטר ברירת המחדל (8). השימוש בפרמטרים מהווה את התשתית של שפת VHDL לשימוש ברכיבי LPM וברכיבים גמישים אחרים (Megafunctions) שבכמותם השתמשנו בניסוי הראשון !

9. מערכות סינכרוניות - תהליך פשוט (ללא מכונת מצבים)

ישנם תהליכים רבים שהם סינכרוניים, אבל ממומשים ללא מכונת מצבים, למשל מונים.
תהליך סינכרוני מורכב משני חלקים

1. איפוס בזמן RESET אסינכרוני,
2. פעולה שמתבצעת כל עלית שעון.

הקוד נראה כך:

```
variable one_sec: integer ;
begin
    if RESETN = '0' then
        --- reset code
        one_sec := 0 ;
    elsif rising_edge(CLK) then
        -- sync Code  ;
        one_sec := one_sec + 1 ;
    end if;
end process;
```

על מנת למנוע LATCHES יש להקפיד:

1. לאתחל את כל המשתנים בחלק של ה- RESET
2. לתת ערך לכל המשתנים בקטע הסינכרוני

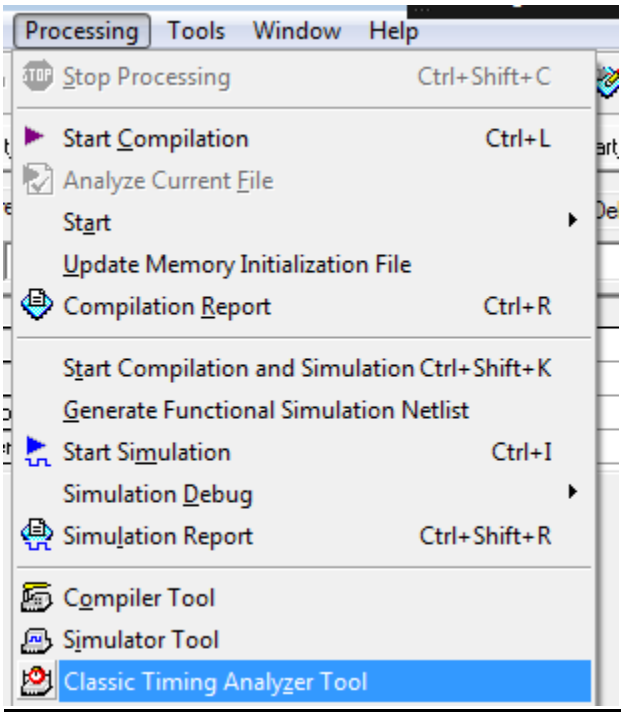
בקטע הסינכרוני ניתן לממש לוגיקה פשוטה שאינה מצדיקה מכונת מצבים
למשל מונה ציקלי, שמבצע איפוס כשמגיעים לערך מסויים:

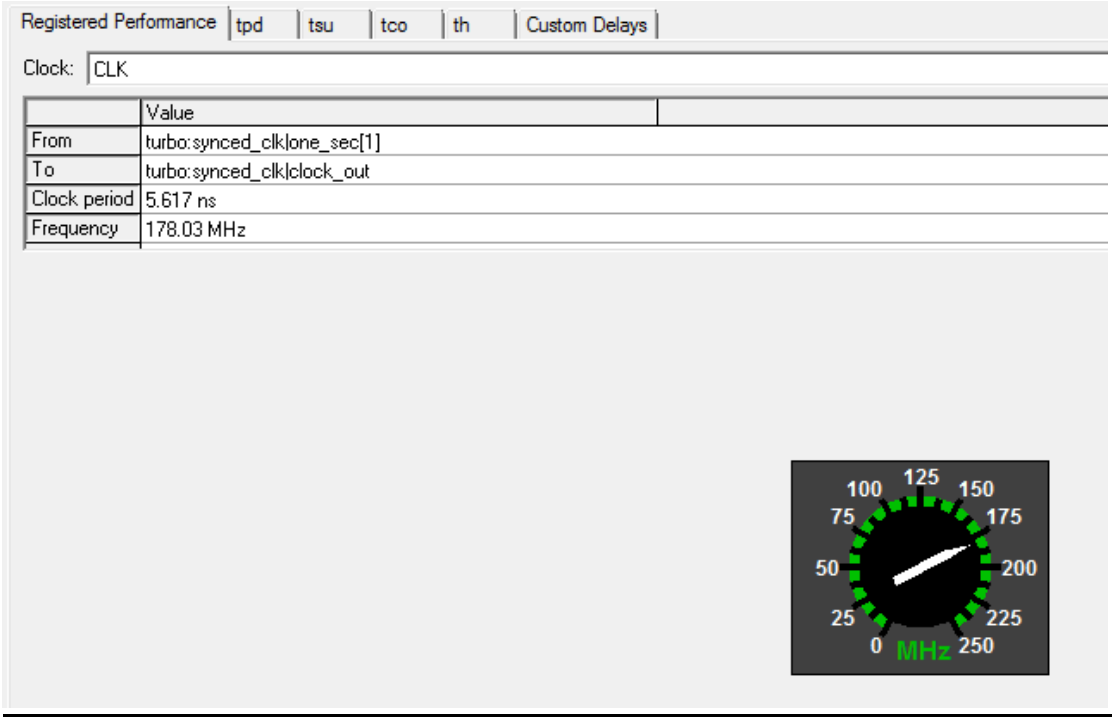
```
process(CLK,RESETN)
    variable one_sec: integer ;
--  constant sec: integer := 50000000 ; -- for Real operation
    constant sec: integer := 5 ; -- for simulation
begin
    if RESETN = '0' then
        one_sec := 0 ;
        one_sec_flag <= '0';
    elsif rising_edge(CLK) then
        one_sec := one_sec + 1;
        if (one_sec > sec) then
            one_sec_flag <= '1';
            one_sec := 0 ;
        else
            one_sec_flag <= '0';
        end if;
    end if;
end process;
```

שימו לב שלצרכי סימולציה מומלץ לעבוד עם ערכים קטנים של קבוע SEC , ובהורדה לרכיב לשנות את ערך הקבוע לגודלו האמיתי

10. מדידת מהירות מקסימלית של תכן - timing analyzer

ניתן למדוד את המהירות המקסימלית בה התכן יעבוד בעזרת פקודת timing analyzer
הפעלת הכלי תאפשר מציאת המסלול הקריטי בתכן ומכאן את תדר השעון המקסימלי





11. העתקת קוד לדו"ח מ NOTEPAD++

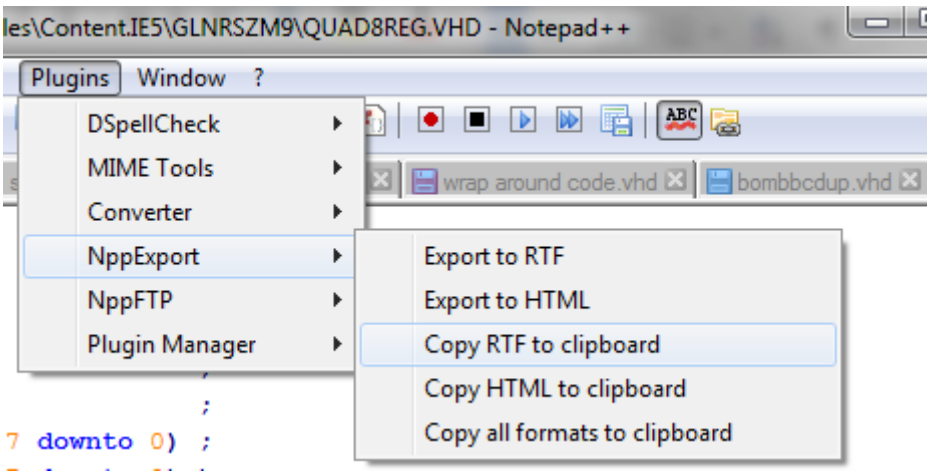
ב NOTEPAD++ הקוד מופיע ברור וצבעוני, העתקתו לWORD בצורה רגילה תבטל את הצבעים:

```
constant sec: integer := 5 ; -- for simulation
begin
if RESETN = '0' then
        one_sec := 0;
        one_sec_flag <= '0; '
elsif rising_edge(CLK) then
```

על מנת לעשות Cut And Paste מ NOTEPAD++ תוך שמירת הצבעים:

```
constant sec: integer := 5 ; -- for simulation
begin
    if RESETN = '0' then
        one_sec := 0 ;
        one_sec_flag <= '0' ;
    elsif rising_edge(CLK) then
```

יש לבצע את התהליך הבא:



את NppExport ניתן להוריד מ- http://sourceforge.net/projects/npp-plugins/files/NppExport/

Driver-ל 64bit ניתן לקחת מהמודל