הטכניון, מכון טכנולוגי לישראל
הפקולטה להנדסת חשמל
מעבדה ל-VLSI

תיאור מקוצר של שפת VHDL

עדכון אחרון - 16:01 01/03/2005

http://www.ee.technion.ac.il/vlsi/

תוכן עניינים

פרק 1 - הקדמה 3
שפת VHDL 3

פרק 2 - מבואות 3
המרכיבים הבסיסיים של השפה 3
תיאור מקבילי 5
תיאור סדרתי 6
יחידות סינכרוניות 8
הערות נוספות לתכנות נכון 9
שילוב תהליכים 9
משפט generate : 10
שימוש ב- generic : 10
מכונות מצבים 11
הגדרות תצורה 12
פעולות על std_logic_vectors 13

פרק 3 – VHDL למטרות סינתזה 13
כללים לסינתזה 13
א. משפטי CLK 13
ב. יצירה של Latches - לא רצויים : 14
ג. בחירת types 14
ד. אלמנטים שאינם מוכרים לצורך סינתזה : 14
ה. שימוש ב-undefined 14
ו. ביצוע אתחול רגיסטרים 15
ז. שימוש בערכים התחלתיים לסיגנלים 15
ח. תכנון היררכי מול תכנון שטוח 15

פרק 4 - דוגמאות ב- VHDL 16
מימוש מעגלים פשוטים 16
קריאה וכתיבה לקבצים 18

פרק 5 - סימולציות ובדיקות תזמון 19
סימולציה 19
סימולציה בעזרת רכיב מעורר 21
סינתזה בכלים של SYNOPSYS 23

פרק 6 - רקע בסיסי למערכת ההפעלה Unix - Solaris 24
מנשק המשתמש של Solaris 24
מערכת הקבצים 24
עורכי טקסט 25
הדפסות 25

מידע נוסף 26

פרק 1 - הקדמה

הערה : לפני כתיבת קוד VHDL **חובה לקרוא את כל החוברת הזאת וגם את החוברת שב:**
http://www.ee.technion.ac.il/vlsi/manuals/synopsys_ams351_2004.pdf

שפת VHDL

VHDL היא שפה לתיאור חומרה[1], המהווה תקן רשמי בתעשייה (IEEE 1076). בעזרת VHDL אפשר לתכנן לתאר ולבצע סימולציה על מערכות מורכבות. ניתן לתאר מערכת בכל רמת הפשטה רצויה מרמת הטרנזיסטור עד לרמת המערכת. במגבלות מסוימות ניתן להפוך תאור VHDL של מערכת למעגל חשמלי באמצעות כלי סינתזה.
שפת VHDL משתייכת למשפחת השפות התיאוריות. משפחה זו נבדלת באופן משמעותי מהשפות הנוהליות (Procedural), בהן Microcode, שפת מכונה, Assembly, Fortran, C, פסקל, BASIC, C++, Java וכיו"ב. בשפות הנוהליות מורה התוכנית למחשב מה לעשות בכל שלב נתון וכיצד בדיוק לעשות זאת. לעומת זאת, בשפות תיאוריות מתארת התוכנית את התוצאות הרצויות, ומשאירה לכלי תוכנה שונים (מהדר - Compiler ; כלי ניתוח - Analyzer ; כלי סינתזה - Synthesizer), להחליט מה וכיצד לבצע, כלומר - איך לממש, על-מנת להגיע לתוצאות אלו.
בעת כתיבת קוד VHDL, חשוב לבחון את דרכי המימוש האפשריות בחומרה, על מנת לוודא שהקוד שנכתב הגיוני. לפני שמתחלים לכתוב ב- VHDL מומלץ להכין דיאגרמת בלוקים מפורטת של התכנון עם הגדרה מדויקת של כל הכניסות והיציאות של כל בלוק. בנוסף לכך רצוי לצייר סכמה של כל בלוק. אין צורך להעמיק מדי בפירוט התכנון, יש להגיע עד לרמת פירוט של מונים, רגיסטרים, יחידות אריתמטיות, מכונות מצבים וכדומה. עבור כל מכונת מצבים יש להכין דיאגרמת מצבים מפורטת.

פרק 2 - מבואות

המרכיבים הבסיסיים של השפה

לשפה מילות מפתח המתחלקות למספר קבוצות: מילות מפתח פונקציונאליות בדומה למילות המפתח בשפה פונקציונאלית (FOR, BEGIN, IF,...); מילות מפתח תיאוריות, המתארות מרכיבים ותכונות של מערכות (כגון: ARCHITECTURE, PROCESS); אופרטורים מסוגים שונים; וספריה תקנית של סוגי נתונים ופונקציות. השפה חופשית מרישיות (Case-Insensitive), אולם מדיניות המעבדה היא לרשום את מילות המפתח השונות באותיות גדולות, ואילו את השמות הניתנים על-ידי התוכניתן באותיות קטנות. מניסיוננו, זה מקל על האבחנה בין מילות המפתח לשאר המזהים.
לא ניתן "לקרוא" Port שמוגדרת כיציאה ולא ניתן "לכתוב" ל- Port שמוגדרת ככניסה. התנהגות של `Port` זהה להתנהגות הסיגנלים (ראה המשך), כלומר ערכם מתעדכן בסוף התהליך.

מילות המפתח התיאוריות:

יישות (ENTITY): היא אבן הבניין הבסיסית בתכן כל מערכת. רמת התכן העליונה מכונה top-level entity, ובתכנון היררכי היא תכיל ישויות נוספות המתארות רמות נמוכות יותר. ישות VHDL מקבילה ל-Symbol בתיאור סכמטי. לכל ישות מוגדר מנשק מול שאר חלקי המערכת, המאופיין בכניסות ויציאות, או באופן כללי יותר "פתחים" (PORTs). ניתן להגדיר את הפתחים כ-IN, OUT או INOUT.

[1] השם VHDL מהווה ראשי תיבות של VHSIC Hardware Description Language. השם VHSIC מהווה ראשי תיבות של Very High Scale Integrated Circuit. במקום VHSIC אפשר גם VLSI, כלומר Very Large Scale Integration.

ארכיטקטורה (ARCHITECTURE): מתארת את התנהגותה של יישות אחת מסויימת, כלומר, כיצד משפיעים הנתונים המתקבלים בכניסות והמצב הנוכחי של המערכת, על המצב העתידי והנתונים המועברים ביציאות. ליישות בודדת ניתן להגדיר מספר ארכיטקטורות (למשל: ארכיטקטורה מבנית, המגדירה את היישות כהרכבה של מספר ישויות בסיסיות יותר; ארכיטקטורה התנהגותית-מקבילית, המתארת את היחס הלוגי בין כניסות, יציאות, משתנים ואותות; ארכיטקטורה התנהגותית-סדרתית, המתארת השינויים באופן סדרתי-אלגוריתמי). ארכיטקטורת VHDL מקבילה לסכמה בתיאור הסכמטי.

תהליך (PROCESS): יחידת ביצוע בסיסית בשפה, המאפשרת תיאור תפקודי של פעילות רכיב או מערכת, באופן סדרתי-אלגוריתמי. כל פעולות הסימולציה מחולקות לתהליך אחד או יותר.

רכיב (COMPONENT): אבן בניין, המאפשרת תיאור מבני של פעילות התקן מורכב יותר. רכיב מגדיר מנשק בין ההתקן הנבנה, לבין ישות בה מעוניינים להשתמש עבורו.

תצורה (CONFIGURATION): מגדירה באיזו ארכיטקטורה להשתמש במקרה מסוים, מבין הקיימות עבור ישות נתונה. חובה להגדיר תצורות מתאימות לכל הישויות המשתתפות, לפני ביצוע סימולציה.

ערכה (PACKAGE): מגדירה בצורה מרוכזת, אוסף של סוגי נתונים (Data Types) ופונקציות פרטיות, לצורך שימוש ביתר חלקי המערכת המתוכננת. קיימות גם ערכות תקניות בהן ניתן להשתמש, לדוגמא:

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

שתי ההצהרות שלעיל מאפשרות למערכת המתוכננת להשתמש בערכה המכונה std_logic_1164, הנמצאת בספריה IEEE. ערכה זו כוללת בין היתר, את הגדרת סוג הנתונים STD_LOGIC, וכל הפונקציות המטפלות בו. בהמשך מופיעה דוגמא לדרך הגדרת ערכה.

סוגי נתונים (Data Types)

השפה כוללת סוגי נתונים רבים. במסגרת הניסוי נשתמש במספר מצומצם של סוגים:

INTEGER: מספר שלם שערכו בתחום 2147483647± (סימטרי סביב אפס).

STD_LOGIC: סוג נתונים שיכול לקבל 9 ערכים שונים, ביניהם:
'0': אפס לוגי.
'1': אחד לוגי.
'X': לא ידוע.
'U': לא מאותחל.
'Z': עכבה גבוהה (High Impedance או HIGHZ).
'-': חסר משמעות (Don't Care).

STD_LOGIC_VECTOR: וקטור של STD_LOGIC. דוגמאות לשימוש בסוג זה:

```
vec :  STD_LOGIC_VECTOR(7 downto 0)
vec2 : STD_LOGIC_VECTOR(0 to 15)
```

רשימות ערכים (Enumerated Data Types):
סוג הנתונים STD_LOGIC הוא דוגמא לרעיון כללי יותר. התוכניתן יכול להגדיר סוגי נתונים ייחודיים במתכונת של רשימות ערכים, לייצוג ערכים ייחודיים הדרושים לפעולה מסוימת. למשל, אם יש לבצע פעולות על ימי השבוע ניתן להגדיר:

```
PACKAGE days_package IS
     TYPE day_t IS (Sunday, Monday, Tuesday, Wednesday,
                    Thursday, Friday, Saturday);
END days_package;
```

ערכה כזו רצוי להגדיר בקובץ נפרד, ואז כדי שישות תכיר את הערכה מוסיפים לפני הגדרתה:

```
USE WORK.days_package.all;
```

אופרטורים :
מוגדרים בשפה אוסף של אופרטורים תקניים. חלק מהאופרטורים משמשים לשימושים שונים בהתאם להקשר. להלן הנפוצים שביניהם :

אופרטורים לוגיים :	AND	OR	NAND	NOR	XOR	NOT
אופרטורי יחס :	=	/=	<	<=	>	>=
אופרטורים אריתמטיים :	+	-	*	/		
אופרטורי השמה :	:= (למשתנה - Variable)	<= (לאות - Signal או פתח - Port)				
אופרטור הפעלה :	=>					

תכונות (Attributes) :
מרכיב מעניין בשפה הוא היכולת לקבל נתונים שונים לגבי מצבם של אותות, פתחים ומשתנים, מעבר לערכם. נתונים אלו מכונים "תכונות" (Attributes), ולמרכיבים שוני של השפה יש תכונות רבות ושונות. למשל, תכונה של מערך יכולה להיות אורכו, תכונות של משתנה שלם יכולות להיות הערך המזערי והמירבי שלו. כדי לקבל את ערך התכונה, רושמים תג (Tick) מיד לאחר הרכיב אליו מתייחסים, ולאחריו את שם התכונה.
לדוגמא, הביטוי clock'EVENT, כאשר clock הוא מסוג std_logic, מחזיר true אם התרחש שינוי כלשהו בערך של clock, בפרק הזמן הנוכחי (Current Delta). הביטוי clock'LAST_VALUE ייתן במקרה זה את הערך הקודם של clock ; ערך חשוב כשלעצמו, כיוון שלעתים עצם העובדה שהערך השתנה והוא למשל '1' כרגע, אין בה כדי לומר שקודם היה '0', ייתכן שלפני-כן היה הערך בלתי מוגדר, עובדה העשויה לדרוש התייחסות שונה.
שימוש טיפוסי נוסף בתכונות, הוא על-מנת לעבור מערך פיסיקלי נתון (כמו 10 ns למשתנה זמן או '1' ל-std_logic) לערך מספרי פשוט (כמו 10 או 4). זאת באמצעות התכונה VAL בעלת הצורה הכללית : sometype'VAL(some_val).

הערות תיעוד (Comments) :
הערות תיעוד פנימי בקבצי שפת VHDL רושמים אחרי זוג מקפים (--). כל מה שמופיע אחרי שני המקפים ועד סוף השורה נחשב כתיעוד ואינו נבדק על-ידי המהדר. סיום השורה מסיים המחרוזת שאינה נבדקת, ואין צורך בסימנים מיוחדים נוספים. שיטה זו זהה לשיטת התיעוד הפנימי של C++ (שם התווים הפותחים הם //).

תיאור מקבילי

נראה להלן בעזרת דוגמא מוסברת, כיצד יש לבצע תיאור התקנים באופן מקבילי.

דוגמא א : **הגדרת ישות המייצגת RSFF (Reset-Set Flip-Flop)**

בהגדרת הישות של רכיב מופיעים שם הרכיב, הכניסות והיציאות שלו כדלקמן :

```
ENTITY rsff IS
     PORT (set, reset: IN STD_LOGIC;
               q, qb: INOUT STD_LOGIC);
END rsff;
```

הגדרת הארכיטקטורה של ה- RSFF בתיאור התנהגותי מקבילי :

```
ARCHITECTURE arc_rsff OF rsff IS
BEGIN
     q<= NOT (qb AND set);
     qb<= NOT (q AND reset);
END arc_rsff;
```

ארכיטקטורה התנהגותית-מקבילית זו מממשת RSFF, על-ידי הגדרת ערכי היציאות שלו, בהתאם לערכי הכניסות. כיוון שארכיטקטורה זו מקבילית, אין חשיבות לסדר ההשמות בתוך הבלוק. בכל

מקרה, השמות אלו מופעלות במקרה שאחד הערכים בצידן הימני משתנה, ללא קשר לסדר כלשהו.
השתנות הערכים הימניים, מתורגמת לשינוי ערך הצד השמאלי. למשל, אם q = 1 והערך reset הופך מ-'0' ל-'1', אזי qb מקבל ערך '0', בהתאם להשמה השניה. כעת, עשויה להיכנס לפעולה ההשמה הראשונה.
כאן נתקלנו במקרה של **משפטים מקביליים (Concurrent Statements)**. זהו המצב הרגיל בהגדרות ארכיטקטורה, כאשר כל ההשמות המופיעות בבלוק הארכיטקטורה (בין ה-BEGIN ל-END) מתבצעות במקביל. **עובדה זו מהותית מאוד, ומומלץ לזכור אותה בכל עת!**

הגדרת הארכיטקטורה של ה- RSFF בתיאור מבני:

```
ARCHITECTURE arc2_rsff OF rsff IS
COMPONENT nand2
     PORT (a,b :IN STD_LOGIC;
          c :OUT STD_LOGIC);
END COMPONENT;

BEGIN
     U1: nand2
          PORT MAP (set, qb, q);
     U2: nand2
          PORT MAP (reset, q, qb);
END arc2_rsff;
```

ארכיטקטורה מבנית זו מממשת RSFF בעזרת שערי nand2. ההנחה היא ששערים אלו הוגדרו מראש כישויות בעלות ארכיטקטורה ותצורה מתאימה. כאן, הבלוק COMPONENT מגדיר את המנשק לשערי nand2, ואילו הבלוק הראשי מגדיר את הרכיבים השונים השותפים במבנה rsff, בארכיטקטורה זו. הרכיבים מוגדרים על-ידי מיפוי מהפתחים (Ports) השונים של rsff, לבין הפתחים המתאימים בכל אחד מרכיבי nand2 השותפים במבנה.
יש לציין כי למרות השוני בתיאור המבני כאן, גם כאן כל ההשמות המופיעות בבלוק הארכיטקטורה מתבצעות במקביל; כלומר, גם כאן מדובר בתיאור מקבילי.

תיאור סדרתי

תיאור סדרתי של התקנים הוא למעשה תיאור של תהליך פעולתם. בהתאם, הוא מוגדר כתהליך (Process), בעזרת מילת המפתח המתאימה. התהליך מתואר על-ידי משפטיים סדרתיים (Sequential Statement) המתבצעים זה אחר זה לפי סדרם, באופן זהה רעיונית לדרך ביצוען של תוכניות אלגוריתמיות בשפות נוהליות (Procedural Languages). בהתאם, קיימות בשפת VHDL מילות מפתח האופייניות לשפות כאלו, כגון: CASE, IF, FOR ו- LOOP.
מבחינת התוכניתן, לרוב תיאור סדרתי הוא טבעי יותר מתיאור מקבילי. לכן הוא נפוץ יותר בתיאור התנהגותם של רכיבים בסיסיים. התיאור המקבילי ישמש לרוב במקרים של תיאור מבני, וכן בהגדרה של רכיבים פשוטים מאוד, שקל לתארם באמצעות ביטוי בוליאני.

דוגמא ב: **משפט IF**

```
IF (day = Sunday) THEN
     weekend := TRUE;
ELSIF (day = Saturday) THEN
     weekend := TRUE;
ELSE
     weekend := FALSE;
END IF;
```

דוגמא ג: **משפט CASE**

```
CASE vector IS
     WHEN “10” =>
          a := 1;
```

```
     WHEN “01” =>
          a := 2;
     WHEN OTHERS =>
          a := 0;
END CASE;
```

דוגמא ד: **תיאור סדרתי של התנהגות ה-RSFF**

```
ARCHITECTURE   arc3_rsff OF rsff   IS
BEGIN
     PROCESS(set , reset)
     BEGIN
          IF set = ‘1’ AND reset = ‘0’ THEN
               q <= ‘0’;
               qb <=’1’;
          ELSIF set = ‘0’ AND reset = ‘1’ THEN
               q <=’1’;
               qb <=’0’ ;
          ELSIF set = ‘0’ AND reset = ‘0’ THEN
               q <=’1’;
               qb <=’1’;
          END IF;
     END PROCESS;
END arc3_rsff;
```

הארכיטקטורה שלעיל מוגדרת על-ידי בלוק תהליך בודד, שתחילתו במילה PROCESS וסופו END PROCESS. כל המשפטים בין שתי שורות אלה הם חלק מהבלוק (תהליך).

משתנים אותות ופתחים

בשפת VHDL ניתן להגדיר משתנים (Variables), אותות (Signals) ופתחים (Ports).
משתני VHDL דומים למשתנים מקומיים של שגרות (Procedures) בתכנות מבני. הגדרות המשתנים מופיעות בתוך בלוק התהליך; ורק התהליך שבו מוגדר המשתנה, מכיר אותו. משפטי השמה של משתנים (מהצורה num := 1) מתבצעים מיד.
אותות VHDL שקולים לרוב לחוטים בסכמה. כאשר רוצים לחבר שתי רכיבים בהתקן נעזרים באותות (ראה דוגמא ה' בהמשך). הגדרת האותות מופיעה אחרי כותרת הבלוק "ARCHITECTURE", והם מופרים בארכיטקטורה כולה. כל משפטי השמה של אותות (מהצורה sig <= ’1’) מתבצעים בו זמנית כאשר התהליך עוצר; למשל בסוף התהליך או כאשר מגיעים למשפט "WAIT", כמו WAIT UNTIL clock`EVENT. אם בתהליך מסוים קיים עבור אות כלשהו יותר ממשפט השמה אחד בין עצירות, המערכת תתעלם מכולם חוץ מהאחרון. כיוון שמשפטי ההשמה לאותות מתבצעים רק עם עצירת התהליך, אין כל אפשרות להשתמש בהם כמונים או מעבירי ערך בתוך התהליך (ללולאות למשל), תפקיד זה שמור למשתנים בלבד.
כמו בכל חלקי השפה, גם במסגרת תהליך עשויה להיות התייחסות ל**פתחים** (**PORTs**). התנהגות הפתחים דומה מאוד להתנהגות אותות, כלומר ערכם מתעדכן בסוף התהליך, ונשאר קבוע לאורך ביצועו; ההשמה מבוצע בעזרת אותו אופרטור; וההשמה האחרונה היא הקובעת. אולם, פתחים מוגדרים גם ככניסה (IN), יציאה (OUT) או שילוב (INOUT). פתח המוגדר ככניסה, ניתן רק לקרוא ממנו, ולא לכתוב לתוכו; בדומה, ליציאה ניתן רק לכתוב, ולא ניתן לקרוא ממנה.
חשוב: מומלץ תמיד להשתמש רק באותות (ולא במשתנה) במימוש של לוגיקה. רצוי להשתמש במשתנים למטרות כגון האינדקס של לולאה.

דוגמא ה: **חיבור שני מהפכים בעזרת אותות**

להבהרה: buf_in ו- buf_out הן כניסות המעגל (מוגדרות בהגדרת הישות של buf). inv_in_out הוא אות המחבר בין הרכיבים U1 ו-U2 של buf.

```
ARCHITECTURE arc_buf OF buf IS
SIGNAL    inv_in_out : STD_LOGIC;

COMPONENT inv
```

```
     PORT (inv_in :IN STD_LOGIC;
          inv_out :OUT STD_LOGIC);
END COMPONENT;

BEGIN
     U1: inv
          PORT MAP (buf_in, inv_in_out);
     U2: inv
          PORT MAP (inv_in_out, buf_out);
END arc_buf;
```

דוגמא ו : **לולאות ושימוש במשתנים במסגרת תהליך**

```
ENTITY add IS
     PORT (a, b  : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
                s : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END add;

ARCHITECTURE arc_add OF add IS
BEGIN
     PROCESS(a,b)
          VARIABLE  i : INTEGER;
     BEGIN
          FOR i IN 0 TO 3 LOOP
               s(i) <= a(i) xor b(i) ;
          END LOOP;
     END PROCESS;
END arc_add;
```

יחידות סינכרוניות

במסגרת תהליך, ניתן לבנות יחידות סינכרוניות כגון Flip-Flop בעזרת משפט מהסוג :

```
IF clock'EVENT AND clock = 1 THEN
```

משמעות המשפט הוא : אם השעון השתנה וכעת הוא שווה ל- '1' (כלומר הייתה עליית שעון).

דוגמא 7 : מימוש DFF בעזרת תכונות (Attributes)

```
ENTITY dff_asynch IS
     PORT( clock, reset, din : IN std_logic;
                         dout : OUT std_logic );
END dff_asynch;

ARCHITECTURE arc_dff_asynch OF dff_asynch IS
BEGIN
     PROCESS(reset, clock)
     BEGIN
          IF (reset = '1') THEN
               dout <= '1';
          ELSIF ( clock'EVENT ) AND ( clock = '1') THEN
               dout <= din;
          END IF;
     END PROCESS;
```

```
END arc_dff_asynch;
```

הערות נוספות לתכנות נכון

- ככלל תהליכים פועלים בלולאה אינסופית, אם אין מגדירים להם סיבות לעצירה. קיימות שתי דרכים עיקריות להגדרת עצירה:
הראשונה היא **רשימת רגישויות** (Sensitivity List). זו רשימה של אותות המופיע מיד לאחר מילת המפתח PROCESS. כאשר מוגדרת כזו היא גורמת לעצירת התהליך לפני תחילתו, עד שאחד מערכיה משתנה. שינוי כזה גורר ביצוע של התהליך. רשימת הרגישויות נדרשת כאן, בניגוד לתיאור המקבילי, כיוון שכאן אין המערכת יכולה לצפות מה משפיע על מה, מבלי לבצע החישובים בפועל. הגדרה נכונה של רשימת הרגישויות היא באחריות התוכניתן. באופן כללי, אם בתהליך קיים אות (Signal) המופיע בצד ימין של משפט השמה, האות יופיע ברשימה. בדוגמא ד' שלעיל רשימת הרגישויות מורכבת מהאותות set ו-reset. כדאי להביט גם בשאר הדוגמאות ולזהות את רשימת הרגישויות ואת תפקידה וסיבות הגדרתה בכל מקרה.

  הדרך השניה היא שימוש במשפטי WAIT, וגם זאת ניתן לבצע בשתי דרכים: שימוש בהמתנה לזמן קבוע (WAIT FOR 10 ns למשל), או שימוש בהמתנה עד לקיום צירוף קומבינטורי מסויים (WAIT UNTIL clk'EVENT AND clk = '1' למשל). המקרה האחרון דומה במידה מסויימת לרשימת רגישויות, בעיקר עבור צירופים קומבינטוריים פשוטים.
- כאשר אנו משתמשים במשפטי IF, כמעט תמיד נדרש בסוף הסדרה משפט ELSE (בניגוד ל-ELSIF), על-מנת לכסות מצבים שאינם מכוסים אחרת. החסרת חלק זה כשהוא נדרש תביא ליצירת Latches במימוש; יצירה שהיא לרוב בלתי-רצויה ומיותרת.
באופן כללי יותר, כאשר תהליך (Process) מסויים נותן ערך לאות (Signal) במצב קומבינטורי מסויים, עליו לתת לו ערך גם בכל מצב אחר. אם לא ינתן ערך כזה באופן מפורש, ישלים הסימולטור ערך ברירת מחדל מטעמו. ערך כזה עשוי ליצור התנגשות עם ערכים אחרים הנדחפים לאותו אות על-ידי תהליכים אחרים. עם-זאת, יתכנו מקרים בהם מספר תהליכים נדרשים לתת לאות ערך באופן מקביל, כאשר רק ערך אחד בעל משמעות. במקרה כזה, על כל התהליכים הנותנים ערך חסר משמעות להציב לאות 'Z' (HIGHZ) (לאות מסוג std_logic).
- באופן כללי, אין להכניס לרשימת הרגישויות של תהליך קומבינטורי אותות שאינם שייכים לאותו תהליך, כגון אותות שעון.

שילוב תהליכים

במקרים רבים מומלץ ורצוי לשלב מספר תהליכים שונים במימוש ארכיטקטורה אחת. דוגמא חשובה ומומלצת היא הפרדת ארכיטקטורה לשני תהליכים מקבילים, האחד מתוזמן (סינכרוני - Synchronous) והשני הכולל את הלוגיקה הצירופית (קומבינטורי - Combinational, Combinatorial). בדוגמא ח' למשל, ניתן לראות כי הלוגיקה הצירופית פועלת בתהליך הראשון ללא תלות בשעון, אך הערך ביציאה (dout) משתנה רק עם עלית השעון. זוהי דרך מימוש נכונה ורצויה.

דוגמא ח: **מימוש מונה בעזרת שני תהליכים – צירופי ומתוזמן**

```
PACKAGE count_types IS
     TYPE bit4 IS range 0 to 15;
END count_types;

Library IEEE;
USE IEEE.std_logic_1164.ALL;
USE WORK.count_types.ALL;

ENTITY count IS
     PORT (clock, load, clear : IN std_logic;
          din : IN bit4;
          dout : INOUT bit4);
END count;

ARCHITECTURE arc_count OF count IS
```

```
     SIGNAL count_val : bit4;
BEGIN
     PROCESS (load, clear, din, dout)   -- Combinational
     BEGIN
          IF (load = '1') THEN
               count_val <= din;
          ELSIF (clear = '1') THEN
               count_val <= 0;
          ELSIF (dout >= 15) THEN
               count_val <= 0;
          ELSE
               count_val <= dout + 1;
          END IF;
     END PROCESS;

     PROCESS                                -- Synchronous
     BEGIN
          WAIT UNTIL clock'EVENT and clock = '1';
          dout <= count_val;
     END PROCESS;
END arc_count;
```

משפט generate :

לעתים ניתן לקצר את אורך הקוד באמצעות משפט generate. להלן דוגמא של מימוש מסכם 4 סיביות בעזרת משפט generate"

```
  add_label:        -- Note that a label is required here
   for i in 4 downto 1 generate
    FA: full_adder port map(C(i-1), A(i), B(i), C(i), Sum(i));
   end generate;
```

קוד מתורגם ל:

```
    FA4: full_adder port map(C(3), A(4), B(4), C(4), Sum(4));
    FA3: full_adder port map(C(2), A(3), B(3), C(3), Sum(3));
    FA2: full_adder port map(C(1), A(2), B(2), C(2), Sum(2));
    FA1: full_adder port map(C(0), A(1), B(1), C(1), Sum(1));
```

ניתן לשים כל משפט מקבילי (כולל process שלם) בלולאה של ה- generate.

שימוש ב- generic:

מאפשר הגדרת יחידות בעלי פרמטרים משתנים. לדוגמא:

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

entity reg_g is
  generic(left   : natural := 31;     -- top bit
          prop   : time := 100 ps);     -- delay
  port   (clk    : in  std_logic;
          input  : in  std_logic_vector (left downto 0);
          output : out std_logic_vector (left downto 0)
);
end entity reg_g;
```

```
architecture behavior of reg_g is
begin  -- behavior
  reg: process(clk)
       begin
         if clk='1' then               -- rising edge
           output <= input after prop;
         end if;
       end process reg;
end architecture behavior;  -- of reg_g
```

הפרמטרים left ו- prop יכולים לקבל ערך משתנה בזמן הצבת הרכיב בארכיטקטורה. הערכים המוגדים ב- entity הם ערכי בררת המחדל. דוגמא של שימוש היחידה:

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

architecture test of test_g is
  constant prop : time := 50 ps;
  constant left : natural := 7;        -- top bit number
  …….

begin  -- test
  load <= '0' after 1 ps;         -- one shot
  clk <= not clk after 5 ns;      -- 10 ns period
  …….

  reg:entity WORK.reg_g  generic map(left=>left,
prop=>prop)
                         port map(clk=>clk, input=>inp,
output=>outp);

end test; -- of test_g
```

מכונות מצבים

בדומה לאמור לעיל, מקובל לממש מכונת מצבים בעזרת שני תהליכים - מתוזמן וצירופי. התהליך הצירופי קובע מה יהיה המצב הבא ואת ערכי היציאות (עבור מכונת mealy), והתהליך המתוזמן משנה בפועל את המצב, בזמן הנכון. ערכי היציאות יכולים להשתנות באופן מתוזמן עם השתנות המצב, או באופן בלתי-מתוזמן, בהתאם למצב ולערכי הכניסות בזמן נתון; הכל בהתאם לסוג המכונה.

דוגמא ט : **מימוש מכונת מצבים בעזרת שני תהליכים – מתוזמן וצירופי**

אין צורך לנסות להבין את מטרת המכונה. הדוגמא מובאת כאן רק על מנת להציג את צורת המימוש של מכונות מצבים בשפת VHDL.

```
ENTITY fsm IS
     PORT (clk :IN STD_LOGIC;
           reset,i1: IN STD_LOGIC;
           z1: OUT STD_LOGIC);
END fsm;
```

```
ARCHITECHTURE arc_fsm OF fsm IS
     SIGNAL next_state, present_state: STD_LOGIC;
BEGIN
     PROCESS (present_state, i1)        -- Combinational
     BEGIN
          CASE present_state IS
               WHEN '1' =>
                    IF i1 = '0' THEN
                         next_state <= '1';
                         z1 <= '0';
                    ELSE
                         next_state <= '0';
                         z1 <= '1';
                    END IF;
               WHEN '0' =>
                    IF i1 = '0' THEN
                         next_state <= '0';
                         z1 <= '1';
                    ELSE
                         next_state <= '1';
                         z1 <= '1';
                    END IF;
          END CASE;
     END PROCESS;

     PROCESS (clk, reset)               -- Synchronous
     BEGIN
          IF reset = '1' THEN
               present_state <= '0';
          ELSIF  clk'EVENT AND clk = '1' THEN
               present_state <= next_state;
          END IF;
     END PROCESS;
END arc_elev;
```

הגדרות תצורה

כאמור, לפני ביצוע סימולציה, חובה להגדיר תצורות מתאימות לכל הישויות המשתתפות. הגדרה זו מבוצעת בעזרת מילת המפתח CONFIGURATION המגדירה באיזו ארכיטקטורה להשתמש במקרה מסוים, מבין כל הקיימות עבור ישות נתונה.

דוגמא י : **תיאור תצורה של RSFF**

```
CONFIGURATION cfg_rsff OF rsff IS
FOR arc2_rsff
  FOR U1, U2 : nand2 USE ENTITY WORK.mynand(arc_mynand);
  END FOR;
END FOR;
END cfg_rsff;
```

משמעות הגדרת התצורה שבדוגמא היא שבארכיטקטורה arc2_rsff של rsff עבור שני שערי ה-NAND : U1 ו-U2, בחר את הישות mynand, עם ארכיטקטורה arc_mynand. הארכיטקטורה arc_mynand נמצא בספריה WORK שהיא ספרית העבודה.

לעומת זאת, הארכיטקטורה arc_rsff לא מכילה שימוש בתת בלוקים אלה תאור התנהגותי בלבד. תיאור התצורה שלה יראה כך:

```
CONFIGURATION cfg2_rsff OF rsff IS
     FOR arc_rsff
     END FOR;
END cfg2_rsff;
```

פעולות על std_logic_vectors

$SYNOPSYS/packages/IEEE/src/std_logic_1164.vhd מגדיר את ה- type הנפוץ std_logic ו- std_logic_vector. במקרים רבים יש צורך בביצוע פעולות חשבוניות או לוגיות על וקטורים אלה. ספריה $SYNOPSYS/packages/IEEE/src/ קיימים קבצים של packages שימושיים בדיוק למטרות אלה. כמו כן ב- packages אלה מוגדרות פונקציות הממירות type אחד לאחר (לדוגמא std_logic_vector ל- integer). חשוב לעבור על ה- packages בשלב מוקדם של הפרויקט.

פרק 3 – VHDL למטרות סינתזה

לאחר שתאור בשפת VHDL נבדק סימולטיבית, עוברים לשלב הבא בתהליך התכנון. בשלב זה הופכים את התיאור האמור למעגל חשמלי. תהליך זה מכונה סינתזה.
על המימוש המתקבל ניתן לבצע סדרה של בדיקות, כגון:

א. מדידת מספר השערים (שטח כולל).
ב. מציאת מסלולים קריטיים במעגל.

ניתן גם ליצור קובץ VHDL חדש המתאר במדויק את המימוש שהתקבל, ולהחזירו לסימולטור. הרצת סימולציה על קובץ זה, תיתן את ההתנהגות הלוגית עם כל ההשהיות הממשיות, הנובעות מהטכנולוגיה שבשימוש.

VHDL היא שפה המאפשרת חופש רב למשתמשים בה, אבל בעת כתיבת תוכנית המשמשת לצורך תכנון רכיב VLSI , יש להקפיד על מספר כללים על-מנת לעשות את החיים קלים יותר ולהימנע מבעיות בזמן סינתזה וסימולציה ברמת שערים.

כללים לסינתזה

א. משפטי CLK

- יש להשתמש במשפטים מסוג: if (clock'event and clock='1') then עבור אותות שעון בלבד!
- בכל PROCESS מותר להשתמש במשפט WAIT או במשפט "IF CLK'EVENT" אחד לכל היותר.
- אין להשתמש במשפטי "WHILE" כי הם לא עוברים סינתזה.
- אין להשתמש במשפט מסוג:

```
if (clock'event and clock='1' and en='1') then
```

במקום זה יש לרשום:

```
if (clock'event and clock='1') then
    if (en='1') then
```

- יש להשתמש רק במערכים חד מימדיים , מערכים בעלי מימד גדול מאחד לא יעברו סינתזה.

- פעולות אריתמטיות או לוגיות על STD-LOGIC-VECTOR ניתן לבצע בעזרת packages מוכנים. בכלי של Synopsys יש להשתמש packages כגון STD_LOGIC_ARITH או STD_LOGIC_UNSIGNED. ה- packages הנ"ל מוגדרים בקבצים שנמצאים ב:
$SYNOPSYS/packages/IEEE/src

ב. יצירה של - Latches לא רצויים:

Latch נוצר למשל כאשר לא מגדירים באופן מלא את פעולת המעגל. לדוגמא:

```
sample_proc : process(ENABLE,A)
begin
    if (ENABLE='1') then
        Z <= not A;
       end if;
    end if;
end process sample_proc;
```

ה- latch יופיע כי המערכת לא יודעת מה לעשות כאשר ה- ENABLE=0 ולכן היא שומרת על המצב הקודם עם latch. אין להשלים עם הופעה של Latch שלא הוכנס בכוונה ע"י המתכנן. צריך למנוע היוצרות של Latches לא רצויים ע"י כתיבת ה- VHDL בצורה מלאה. למשל להלן הדרך הנכונה לכתוב את הדוגמא הקודמת:

```
sample_proc : process (ENABLE,A)
begin
        if (ENABLE='1') then Z <= not A;
        else Z <= A;
        end if;
    end if;
end process sample_proc;
```

הערה: יש להגדיר את ההתנהגות עבור **כל** ערך אפשרי של ENABLE.

ג. בחירת types

- יש להשתמש ב- types בגודל מוגבל. לדוגמא, אם נדרש להשתמש במשתנה מסוג integer וידוע שערכו אינו עולה על 200 למשל, צריך להגדיר subtype של integer מוגבל למספרים בין 0 ל- 255.

ד. אלמנטים שאינם מוכרים לצורך סינתזה:

האלמנטים הבאים אינם מוכרים ע"י הסינתיסייזר:
- כתיבה לקבצים (שימוש ב- textio).
- שימוש ב- Assertions.
- מימוש השהיות ב- VHDL (ע"י שימוש בפקודה after).
- שימוש במשתנה time.
- שימוש ב- 'wait for...'.

ה. שימוש ב- undefined

יש להימנע בכל מחיר מנתינת ערך undefined למשתנים וסינגלים. תוצאות הסינתזה במקרה זה יהיו בלתי מוגדרות.

ו. ביצוע אתחול רגיסטרים

יש לוודא שניתן יהיה לאתחל או לאפס (בחומרה) רגיסטרים לערכם ההתחלתי, על-מנת למנוע מצב בו הערך ההתחלתי של הרגיסטרים לא ידוע או לא מוגדר. ביצוע איפוס אסינכרוני עדיף על סינכרוני מאחר והוא פחות בעייתי וחסכוני יותר בחומרה.

ז. שימוש בערכים התחלתיים לסיגנלים

ב- VHDL קיימת האפשרות בזמן הגדרת signal לתת לו ערך התחלתי כלשהו הבא לידי ביטוי בזמן התחלת הרצת הסימולציה. הסינטיסייזר מתעלם מהערך הזה בזמן ביצוע הסינטזה. כתוצאה מכך יתכנו אי התאמות בין תוצאת הסימולציה ברמת VHDL וזו שברמת שערים. מסיבה זו מומלץ לא להשתמש במתן ערכים התחלתיים ל-signals. במקום זאת יש לבצע אתחול ע״י מנגנון שניתן למימוש בחומרה.

ח. תכנון היררכי מול תכנון שטוח

באופן כללי תכנון היררכי עדיף וזאת ממספר סיבות:
- ניתן לבצע סימולציה וסינטזה על כל חלק בנפרד דבר המאפשר גמישות רבה יותר.
- ניתן במהלך הסינטזה לוותר על חלק מהיררכיה (או כולה) לצורך אופטימיזציה.
- הסינטזה מתבצעת בצורה קלה ומהירה יותר.
- שמות הרכיבים המתקבלים מהסינטיסייזר מכילים את כל ההיררכיה כך שניתן לזהות את מיקום הרכיב ע״פ שמו.
- ניתן לקבל בצורה קלה נתונים לגבי כל אחת מהיחידות.
- בזמן העריכה מתקבל חופש גדול יותר לצורך סידור הבלוקים השונים.

פרק 4 - דוגמאות ב- VHDL

מימוש מעגלים פשוטים

א. שער OR בעל שלוש כניסות :

```
Library IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY or3 IS
 PORT (a, b, c : IN std_logic;
          d : OUT std_logic);
END or3;

ARCHITECTURE arc_or3 OF or3 IS
BEGIN
   d <= a OR b OR c;
END arc_or3;
```

ב. DFF בעל RESET אסינכרוני:

```
Library IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY dff_asynch IS
  PORT( clock, reset, din : IN std_logic;
           dout : OUT std_logic);
END dff_asynch;

ARCHITECTURE arc_dff_asynch OF dff_asynch IS
BEGIN
   PROCESS(reset, clock)
  BEGIN
     IF (reset = '1') THEN
       dout <= '1';
      ELSIF (clock'EVENT) AND (clock = '1') THEN
       dout <= din;
     END IF;
   END PROCESS;
END arc_dff_asynch;
```

ג. DFF בעל CLEAR ו- PRESET :

```
Library IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY dff_pc IS
   PORT( preset, clear, clock, din : IN std_logic;
       dout : OUT std_logic);
END dff_pc;
```

```
ARCHITECTURE arc_dff_pc OF dff_pc IS
BEGIN
  PROCESS(preset, clear, clock)
 BEGIN
    IF (preset = ‘1’) THEN
     dout <= ‘1’;
    ELSIF (clear = ‘1’) THEN
     dout <= ‘0’;
    ELSIF (clock’EVENT) AND (clock = ‘1’) THEN
     dout <= din;
   END IF;
  END PROCESS;
END arc_dff_pc;
```

٦. 4-bit shifter

```
Library IEEE;
USE IEEE.std_logic_1164.ALL;
PACKAGE shift_types IS
  SUBTYPE bit4 IS std_logic_vector(3 downto 0);
 END shift_types;

USE WORK.shift_types.ALL;
Library IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY shifter IS
 PORT( din : IN bit4;
       clk, load, left_right : IN std_logic;
       dout : INOUT bit4);
END shifter;


ARCHITECTURE arc_shifter OF shifter IS
  SIGNAL shift_val :bit4;
BEGIN
  nxt: PROCESS(load, left_right, din, dout)
 BEGIN
   IF (load = ’1’) THEN
     shift_val <= din;
        ELSIF (left_right = ’0’) THEN
          shift_val(2 downto 0) <= dout(3 downto 1);
          shift_val(3) <= ’0’;
        ELSE
          shift_val(3 downto 1) <= dout(2 downto 0);
     shift_val(0) <= ’0’;
        END IF;
 END PROCESS;
```

```
  current: PROCESS
  BEGIN
    WAIT UNTIL clock'EVENT and clock = '1';
    dout <= shift_val;
  END PROCESS;
END arc_shifter;
```

קריאה וכתיבה לקבצים

בזמן סימולציה ניתן לקרוא או לכתוב נתונים לקבצי TEXT. להלן דוגמא של תהליך הקורא מספר מקובץ, מעלה אותו בריבוע ורושם את התוצאה בקובץ פלט. תפקיד האות go הוא רק לתת למשתמש שליטה על הפעלת התהליך:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library std;
use std.textio.all;
entity square is
  port (go : in std_logic);
end square;

architecture simple of square is
begin
     process(go)
  file infile : text is in "example1";
  file outfile : text is out "outfile1";
  variable out_line, my_line : line;
  variable int_val : integer;
begin
  while not (endfile(infile)) loop
   readline(infile,my_line);
   read(my_line,int_val);
   int_val := int_val ** 2;
   write(out_line, int_val);
   writeline(outfile,out_line);
  end loop;
end process;
end simple;
```

פרק 5 - סימולציות ובדיקות תזמון

השלב לאחר תיאור המעגל בשפת VHDL הוא שלב הסימולציה. מקובל לא לערב את בעיות התזמון עם הבדיקה של הפונקציונליות הלוגית. ההנחה היא שכל הפעולות מתבצעות באפס זמן. לדוגמא, סיגנל מתפשט דרך לוגיקה באפס זמן. החשוב הוא לוודא שכל הפעולות מתבצעות בסדר הנכון והשעון (יחד עם ה- FFs) עוזר בקביעת הסדר.

כאשר המעגל עובד מבחינה לוגית מסנתזים אותו, כלומר הופכים אותו לשערים אמיתיים. רק אחרי שלב זה ניתן לראות ולבדוק את השהייה האמיתיות שבמעגל. אם קיימות בעיות תזמון או שמנסים לבקש מכלי הסינתזה ליצור מעגל מהיר יותר או שחוזרים ל- VHDL ומנסים לבנות מימוש יעיל יותר.

סימולציה

לרב, הסימולציות והסינתזות של תכנוני VHDL במעבדה ל- VLSI מתבצעות באמצעות כלים של חברת Synopsys. הסברים על הפעלת הכלים מובאים החוברת :
http://www.ee.technion.ac.il/vlsi/manuals/synopsys_ams351_2004.pdf

חובה לקרוא את החוברת הנ"ל לפני תחילת כתיבת קוד VHDL. בחוברת (בסעיף המסביר על סינתזה) קיימים הסברים על יכולות הכלים העלולים לעזור מאד בשלב כתיבת הקוד.

למען הנוחיות, מובא כאן הסבר קצר על סימולציות. הסימולציה מתבצעת בעזרת כלי סימולציה המכונה scirocco ("שׁירוֹקוֹ") .

להפעלת הסימולטור יש להריץ run_scirocco. הפקודה "run_scirocco" פועלת כקובץ script המבצע את כל השלבים הנחוצים להפעלת הסימולטור.
יש לבצע את הפקודה באופן הבא :

```
scirocco_run  filename configuration_name
```

כאשר filename הוא שם קובץ ה-VHDL (ולרוב יכלול סיומת vhd) ו-configuration_name הוא שם תצורת התכנון. אם התהליך נכשל (למשל מתקבלת הודעת : Simulation Terminated), ניתן להריץ את השלבים בנפרד על מנת למצוא את מקור השגיאה :

```
vhdlan -noevent filename

scs -ccpath /usr/local/bin/gcc -exe /tmp/scsim$$ configuration_name

scirocco +sim+/tmp/scsim$$ +simargs+-debug_all
```

הערות :
הפקודה vhdlan מפעילה נתח (VHDL Analyzer), המנתח את קבצי המקור ויוצר קבצים להפעלת הסימולציה במספר צורות. בין אלו נמצא קבצים מהסוגים *.sim ו-*.mra בספריה WORK. הסימולטור משתמש בקבצים אלה. האופציה -noevent גורמת לכך שלא יווצרו קבצים עבור סימולציה תלוית אירועים (Event Simulation), והקבצים הנוצרים הם עבור סימולציה סינכרונית (Cycle Simulation).
הפקודה scs מפעילה מהדר (Compiler), המעבד את קובץ התצורה, על-מנת ליצור קובץ סימולציה ניתן להרצה. המהדר משתמש במהדר C, שמיקומו נקבע בהתאם לאופציה -ccpath. האופציה -exe קובעת מה יהיה שמו ומיקומו של קובץ ההרצה המתאים.
הפקודה scirocco מפעילה את המנשק הגראפי של VirSim, המאפשר הרצה אינטראקטיבית של הסימולציה.
אם התכנון מכיל מימוש Package בקובץ נפרד, לפני הפעלת הסימולטור, יש לבצע :

```
vhdlan -noevent package_filename
```

בחלון שנפתח לחץ על :

- window->waveform
- window->hierarchy

- window->source
יתקבלו 3 חלונות חדשים.
בחלון Source לחץ על הצלמית השמאלית העליונה, יופיע בחלון All Group - יש לבחור אותו על-ידי לחיצת העכבר. כתוצאה מכך "ידלקו" שני חיצים צהובים. לחץ פעם אחת על החץ הימני. כעת יופיע הקוד בחלון.

בחלון Hierarchy ניתן לעבור בין היחידות השונות של התכנון. עבור ליחידה מסוימת וסמן את כל אותות הכניסה והיציאה שברצונך להציג. לחץ על כפתור Add שבצד ימין למטה, כתוצאה מכך יוצגו האותות שנבחרו בחלון Waveform.

כדי להריץ את הסימולציה ב-1000nSec למשל, יש לרשום 1000000 בחלון Interactive, בשדה הסמוך ל- Step Time שבתחתית החלון, ולאשר על-ידי לחיצה על הכפתור OK הסמוך לשדה.

כדי להציג את צורות הגל באופן נוח יש לבחור בחלון Waveform:
Zoom → Zoom percent → Zoom 100%.

כעת בדוק את תוצאות הסימולציה.

להדפסת צורות הגל, לחץ על File → Print ורשום lpr –Pbp בשדה Print Command.

שמירת תצורת חלונות: על מנת למנוע את הצורך להגדיר את צורת החלונות עם כל כניסה לסימולטור רצוי לשמור את תצורת החלונות בקובץ בעזרת File → Save Configuration. ניתן להעלות את הקובץ מיד לאחר כניסה לסימולטור בעזרת File → Load Configuration.

הזנת ערכים לכניסות: ניתן להזין ערכים לכניסות בעזרת הפקודה assign. לדוגמא:
```
assign '1' /decoder/A0
assign x"F3" /decoder/P
assign b"0001" /decoder/A
```
בדוגמא זו הפקודה השנייה מזינה ערך הקסהדצימלי, והשלישית ערך בינארי.
אם הכניסה היא וקטור יש לרשום את הפקודה באופן הבא:
```
assign b"1010" /e/data
```

במקום להזין את הפקודות ידנית שוב ושוב, ניתן להגדיר קובץ סימולציה, בו ירשמו הפקודות לפי סדרן. בהנתן קובץ כזה בשם decoder למשל, נריץ אותו בעזרת הפקודה:
```
source decoder
```

דוגמא : **קובץ סימולציה**
```
assign '1' /e/EN
assign '0' /e/A0
assign '1' /e/A1
run 4
assign '1' /e/A0
run 4
assign '0' /e/A1
run 4
assign '0' /e/A0
run 4
```

הצגת ערכים: בכל רגע ניתן לראות את הערך של כל אות או משתנה בעזרת הפקודה "ls –v".
הפקודה ls –v /e/A0 מדפיסה את הערך של /e/A0.
לחיצה על הכפתור ls –v מדפיסה את הערכים של כל האותות, השייכים לרמת ההיררכיה המצויינת על יד scope בחלון Interactive. ניתן לעבור רמה בעזרת הפקודה cd /e לדוגמא.
עם-זאת, בקבצי פקודות הסימולציה אותם יש להכין לניסויים, **<u>אין צורך</u>** להשתמש בפקודות להצגת הערכים. את הערכים נציג במהלך הסימולציה בעזרת החלונות Waveform ו-Hierarchy.

בפקודות האמורות נשתמש רק במקרים בהם נרצה להציג ערך מספרי ספציפי בזמן נתון, לצורך מסוים.

Drivers : במקרה של קבלת X כערך של אות, ניתן לבדוק איזה drivers מאלצים ערך על האות על-ידי סימון האות בחלון ה-Source ולחיצה על כפתור ה-[sig] drivers.

יציאה מהסימולציה נעשה עם File → Quit.

ביציאה, קיימת אפשרות לשמור את קובץ תצורת המערכת (Configuration). זו פעולה מומלצת בעיקר אם אתם נדרשים לצאת ולהכנס מספר פעמים לגבי אותו קובץ סימולציה (ככל שהסימולציה מורכבת יותר, זו פעולה שכיחה יותר - ניפוי שגיאות לוגיות בתוכנית ה-VHDL). שמירת קובץ התצורה מאפשרת לחזור במהירות לאותה תמונה של הסימולטור, מבלי צורך להגדיר מחדש את כל צורת החלונות בכל פעם. על-מנת להעלות מחדש את התצורה השמורה, יש להיכנס ל- File → Load Configuration, עם הכניסה לסימולטור, ולבחור את קובץ התצורה שנשמר לפני-כן.

סימולציה בעזרת רכיב מעורר

קיימת שיטה נוספת (ואולי עדיפה) לביצוע סימולציה, המכונה **TestBench 1.2**. במקום לאלץ ערכים בכניסות של הרכיב בעזרת פקודות assign, ניתן לבנות רכיב VHDL נוסף, שתפקידו לייצר את הערכים הרצויים. לאחר מכן מחברים רכיב זה לרכיב הנבדק.

דוגמא : **רכיב מעורר סימולציה**
הארכיטקטורה שלהלן מורכבת משני תהליכים, אחד היוצר אות שעון, והשני היוצר את האותות "a" ו-"b" שיחוברו לכניסות "j" ו-"k" של ה-JKFF.

```
Library IEEE;
USE IEEE.std_logic_1164.all;

ENTITY jktest IS
     PORT (clk : INOUT std_logic;
           a,b : OUT std_logic);
END jktest;

ARCHITECTURE arc_jktest OF jktest IS
BEGIN
     PROCESS
     BEGIN
          WAIT FOR 50 ns;
          IF (clk = '1') THEN
               clk <= '0';
          ELSE
               clk <= '1';
          END IF;
     END PROCESS;

     PROCESS
     BEGIN
          a <= '0';
          b <= '1';
          WAIT FOR 90 ns;
          a <= '1';
          b <= '0';
          WAIT FOR 100 ns;
```

```
    END PROCESS;
END arc_jktest;
```

כעת ניתן לקלוט לסימולטור את תצורת היישות המכילה את הרכיב הנבדק ואת הרכיב המעורר, היוצר את האותות הרצויים. ליד Step Time (בחלון Interactive) יש להקליד את הזמן הרצוי להרצה, ולאשר על-ידי לחיצה על OK. גם כאן עדיין יש צורך לבקש הצגה של האותות המעניינים. אחד היתרון המשמעותיים בשיטה זו הוא בכך שהקובץ מעורר הסימולציה כתוב בשפה תקנית ומוכרת; במקרה זה VHDL. בניגוד לקוד שמיועד לסינתזה (ראה בהמשך), כאן מותר להשתמש בכל משפט חוקי של VHDL (WAIT למשל).

סינתזה בכלים של SYNOPSYS

לאחר שתיאור ה - VHDL נבדק בעזרת סימולציות עוברים לשלב הבא בתהליך התכנון. בשלב זה הופכים את התיאור הנ"ל למעגל חשמלי. תהליך זה נקרא סינתזה. הסינתזה מבוצעת ע"י כלי בשם design analyser שמופעל בעזרת הפקודה "da". בחלון שנפתח מפעילים את הפקודה "read" על קובץ ה - VHDL . פעולה זו מבצעת את השלב הראשון של הסינתזה כלומר נוצר תיאור של מעגל שמתנהג מבחינה לוגית כמו תיאור ה - VHDL . האלמנטים שמרכיבים את המעגל הם אלמנטים גנריים (שערים כללים) שאינם קשורים לאף ספריה או טכנולוגיה.

בשלב שני יש צורך למפות את התיאור הגנרי על ספריה אמיתית עם טכנולוגיה מוגדרת. זה נעשה בעזרת הפקודה design optimization. התוצאה של ביצוע design optimization היא המימוש האופטימלי שניתן לקבל עבור קובץ ה- VHDL בעזרת הספרייה הנתונה שבמקרה שלנו היא ספרית AMS בטכנולוגית 0.35 מיקרון. על המימוש המתקבל ניתן לבצע סדרה של בדיקות כגון:

a. מדידת מספר השערים (שטח כולל).
b. מציאת מסלולים קריטיים במעגל.

ניתן גם ליצור קובץ VHDL חדש שמתאר במדויק את המימוש שהתקבל ולהחזיר אותו לסימולטור. הרצת סימולציה על קובץ זה תיתן את ההתנהגות הלוגית עם כל ההשהיות המדויקות. פרטים נוספים על כלי זה בחוברת:
http://www.ee.technion.ac.il/vlsi/manuals/synopsys_ams351_2004.pdf
של המעבדה.

פרק 6 - רקע בסיסי למערכת ההפעלה Unix - Solaris

מנשק המשתמש של Solaris

במעבדה מחשבי Sun עליהם מותקנת מערכת ההפעלה Solaris 7 (SunOS 5.7). זו מערכת חלונאית בסביבת Unix, המאפשרת לבצע את מרבית הפעולות הנדרשות הן מתוך שורת פקודה כמקובל ב-Unix והן דרך תפריטים וצלמיות (Icons) כמקובל במערכות חלונאיות.
עם הכניסה למערכת, מופיע על המסך שולחן עבודה (Desktop) ובתחתיתו לוח הפעלה (Front Panel). לוח ההפעלה מאפשר הפעלת תוכניות נפוצות רבות, מעבר בין 4 שולחנות עבודה שונים, קבלת מידע בסיסי על מצב המערכת ועוד. אולם, במעבדה זו כמעט אין צורך בו. זאת, כיוון שאת כל הפעולות הבסיסיות הנדרשות לנו, ניתן להפעיל מ**תפריט מהיר** (Workspace Menu), הנפתח בלחיצת עכבר ימני על שטח ריק בשולחן העבודה. כדי להשלים ההכרות עם שולחן העבודה, נוסיף כי על משטח זה נפתחים חלונות כבכל מערכת חלונאית, ומיזעורם (Minimize) מביא להצגתם כצלמיות בעמודה המתחילה בפינה השמאלית העליונה של המסך.
פקודות עיקריות בתפריט המהיר (נפתח בלחיצת עכבר ימני על שטח ריק בשולחן העבודה):

מפעיל	**פעולה**
חלון לשורת פקודה (Unix Terminal)	Tools → Terminal
מנהל קבצים (File Manager)	Files → File Manager
עורך טקסט בסיסי (דומה ל-Notepad של MS-Windows)	Applications → Text Editor
הפעלת דפדפן אינטרנט (Web Browser)	Links → Web Browser
יציאה מהמערכת (בסיום העבודה)	Log out…

מערכת הקבצים

להלן מספר פקודות בסיסיות לניהול מערכת הקבצים ב-Unix, דרך שורת פקודה
(פעולות דומות ניתן לבצע גם ממנהל הקבצים - File Manager):

פירוש	**פקודה**
שם הספריה הנוכחית	pwd
שינוי הספריה הנוכחית	cd [שם ספריה]
מעבר לספרית האב של ההנוכחית	cd ..
העתקת קובץ	cp [קובץ מקור] [קובץ יעד]
העתקת קובץ לספרית בת של הנוכחית	cp [שם קובץ] [שם ספריה]
העתקת קובץ לספרית האב	cp [שם קובץ] ..
העברת קובץ לספרית בת של הנוכחית	mv [שם קובץ] [שם ספריה]
שינוי שם קובץ	mv [שם ישן] [שם חדש]
העברת קובץ לספרית האב	mv [שם קובץ] ..
מחיקת קובץ	rm [שם קובץ]
יצירת ספרית בת לספריה הנוכחית	mkdir [שם ספריה]
מחיקת ספרית בת של הנוכחית	rmdir [שם ספריה]
רשימת שמות הקבצים הגלויים בספריה הנוכחית	ls
רשימת פרטי כל הקבצים בספריה הנוכחית	ls -la

כפי שניתן לראות, שם ספרית האב של הנוכחית הוא תמיד ״..״ (נקודה-נקודה).

כשברצוננו להתייחס לספריה אחות (הנמצאת תחת אותו אב), ניתן לרשום: [שם ספריה]/..

עורכי טקסט

במהלך הניסויים נדרש להשתמש בעורך טקסט (Text Editor), כדי ליצור הקבצים. קיימים מספר עורכים שונים במערכת, כגון: vi, vim, nedit, textedit, xemacs ועוד. ניתן להשתמש בכל עורך לפי שיקול המשתתפים בניסוי. העורכים xemacs, nedit ו-vim "מכירים" שפת VHDL.

למי שאינו מכיר היטב עורכי טקסט ב-Unix ורגיל דווקא לסביבת חלונות, מומלץ על Text Editor, המופעל מתוך תת-התפריט Applications של התפריט המהיר. עורך זה עובד באופן דומה מאוד ל-Notepad של חלונות.

עורך חלונאי דומה, אך חזק מעט יותר הוא **NEdit**, המופעל בעזרת הפקודה (בשורת פקודה):

```
nedit filename.vhd
```

עורך זה גם מראה בהדגשה את המלים השמורות של השפה.

עורך חלונאי נוסף, חזק ומורכב יותר, הוא **xemacs** מופעל בעזרת הפקודה:

```
xemacs filename.vhd
```

עורך זה מסוגל להשלים את הקטעים הנפוצים בקוד VHDL, על-ידי לחיצה על TAB.

עורך נוסף, לא חלונאי אך חזק במיוחד, הוא **vim**. ניתן להפעילו ב-UNIX Terminal בעזרת הפקודה:

```
vim -g filename.vhd
```

בעורך זה שני מצבים (Modes): מצב הכנסת מלל חדש, ומצב של עריכה.
מצב עריכה הוא מצב ברירת המחדל, והפקודות החשובות בו:

SPACE	מקדם את הסמן תו אחד
BACKSPACE	מחזיר את הסמן תו אחד
ENTER	מעביר את הסמן לשורה הבאה
-	מעביר את הסמן לשורה הקודמת
:w	שומר את הקובץ בדיסק
:q!	יוצא מהעורך בלי לשמור
:wq	שומר ויוצא
x	מוחק את התו מתחת לסמן
dd	מוחק את השורה עליה נמצא הסמן
D	מוחק מהתו שעליו הסמן עד סוף השורה
u	מבטל את הפקודה האחרונה
.	חוזר שוב על הפעולה האחרונה
פקודות כניסה למצב הכנסת מלל חדש (ליציאה מהמצב - Esc):	
o	פותח שורה חדשה אחרי השורה שעליה הסמן
O	פותח שורה חדשה לפני השורה שעליה הסמן
a	מאפשר הכנסת text אחרי התו שעליו הסמן
i	מאפשר הכנסת text לפני התו שעליו הסמן

הדפסות

ברוב עורכי הטקסט והחלונות בהם נעשה שימוש, ניתן להדפיס בדומה למערכת חלונות, בעזרת File → Print, ולחיצה על <Enter> ללא שום שינויים או תוספות.
כמו-כן, ניתן להדפיס קובץ המוכר על-ידי המדפסת (טקסט או PostScript) בעזרת הפקודה:

```
lpr -Pbp filename
```

מידע נוסף

ספר VHDL מומלץ להרחבה ולהבהרה:
Douglas L. Perry, "**VHDL**", McGraw-Hill. Third Ed. - 1998; Fourth Ed. - 2002.
לספר זה מצורף דיסק (CD) עם תוכניות, המושאל בנפרד בספריה.

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

ספר מומלץ לחזרה על נושאי "מערכות ספרתיות" \ "תכן לוגי":
Stephan A. Ward & Robert H. Halstead, Jr., "**Computation Structures**", McGraw-Hill, 1989.

סימולטור VHDL חופשי לסביבת חלונות:
http://www.symphonyeda.com/proddownloads.htm

סימולטור חופשי נוסף לסביבת חלונות, שבעת כתיבת שורות אלו טוב עבור Windows98/NT בלבד: http://www.bluepc.com/download.html

האתר הראשי של התחום:
http://www.vhdl.org

הסברים נוספים:
http://www.vhdl-online.de/~vhdl/tutorial/englisch/inhalt.htm