**VHDL Coding Guidelines**

# שם הקובץ

## שם הקובץ צריך להיות זהה לשם ה- entity.

## שם הקובץ צריך להיות בעל משמעות כך שיובן באופן כללי במה מטפל הקובץ.

## שם הקובץ יכול להכיל רק תווים אלפא-נומריים ו- \_ (underscore), והתו הראשון לא יכול להיות ספרה.

## רצוי שהאותיות בשם הקובץ יהיו אותיות קטנות ולא גדולות.

## אין להשתמש במילים שמורות עבור שם קובץ (שם entity).

## היררכיה בקובץ בודד

## יש להקפיד שבכל קובץ יהיה רק entity אחד.

## יש להקפיד שלכל entity תהיה רק ארכיטקטורה אחת.

# שמות

## שמות (סיגנלים, entity, ארכיטקטורה, קבועים...) צריכים להיות בעלי משמעות.

## יש להשתמש ב- \_ כדי להפריד בין מילים (ולא ע"י ערבוב בין מילים באותיות גדולות וקטנות).

## רצוי שהשמות יהיו באותיות קטנות ולא גדולות.

## אין להשתמש במילים שמורות עבור שמות.

## השמות צריכים לא להיות ארוכים מדי.

# סיומות

## שמם של סיגנלים שפעילים בנמוך יסתיים ב- \_n. לדוגמא: reset\_n פירושו קו RESET שפעיל בנמוך.

## שמות של ports שהם מסוג inout יסתיימו עם סיומת \_io. ה- Output Enable שלהם יסתיים עם סיומת \_oe.

## שמות של generics יסתיימו עם סיומת של \_g.

## שמות של constants יסתיימו עם סיומת \_c.

## שמות של types יסתיימו עם סיומת \_t.

## שמות של סיגנלים שלState Machine יסתיימו עם \_sm. שם של מצב במכונת מצבים יסתיים ב- \_st.

## סיומת של variables יסתיימו עם \_v.

## ניתן גם להשתמש בתחילית במקום בסיומת, אך יש לשמור על עקביות ולא לערבב.

# Header

## יש להשתמש ב- Header לכל קובץ.

## ה- Header כולל (בין היתר):

* כותרת
* שם מחבר הקובץ
* שם הקובץ
* תאריך יצירת הקובץ
* תיאור
* רשימת Revision History, ששיטת קידום הורסיות שלו צריכה להיות – לכל שינוי קטן הספרה מצד ימין לנקודה תקודם, ולכל שינוי מהותי (שינוי ב- ports או שינוי מהותי בפונקציונאליות) הספרה מצד שמאל לנקודה תקודם, והספרה מצד ימין לנקודה תאופס.

# ארכיטקטורה

## מומלץ להשתמש רק ב- 3 שמות לארכיטקטורה:

* rtl – עבור קוד המיועד לסינתזה.
* beh – עבור קוד התנהגותי המחליף את הקוד המיועד לסינתזה.
* sim – עבור קוד המשמש את סביבת הסימולציה.

# קוד

## רצוי לרשום את כל הקוד באותיות קטנות. יש כאלו שרושמים types ו- constants בלבד באותיות גדולות (גם מקובל).

## אין לערבב במילה אחת בין אותיות גדולות לקטנות.

## יש להשתמש בהזחות ורווחים.

## יש להוסיף Comments לכל port, signal, process, עם הסבר מועיל לטובת קריאות, הבנה ותחזוקה של הקוד.

## אין לבצע יותר מפעולה אחת (השמה, הגדרה...) בכל שורת קוד.

* לדוגמא: הגדרת port, הגדרת signal, הגדרת state של State Machine ועוד...
* דוגמא לכתיבה לא תקינה:

signal a1, a2, a3 : integer;

דוגמא לכתיבה תקינה:

signal a1 : integer;

signal a2 : integer;

signal a3 : integer;

## כל קובץ לא יכיל יותר מדי שורות קוד. אם יש בקובץ יותר מ- 600 שורות, כנראה שיש לחלק לבלוקים את הבלוק הנוכחי.

## יש להשתמש ככל הניתן ב- generics וב- constants.

# Process

## יש לתת לכל process שם (label): רצוי עם סיומת \_proc.

## יש לסיים את ה- process עם שם ה-label אשר ניתן ל- process בתחילתו.

## יש להשתדל שכל process יטפל בסיגנל בודד, או בקבוצה משותפת (מבחינה פונקציונאלית) של סיגנלים.

# Label

## יש לסגור כל label באמצעות end עם שם ה- label. רלוונטי ל-

* entity
* architecture
* case
* component
* process
* for loop / while
* function
* procedure
* Package

# משפט If

## יש לתחום משפטי התניה באמצעות סוגריים.

## יש לתחום כל אפשרות קיום, המכילה יותר מתנאי בודד, במשפט ההתניה עצמו, באמצעות סוגריים.

## רצוי לתחום את רמת הקינון של משפט ההתניה ל- 5 רמות.

## ב- process אסינכרוני יש לסיים משפט if עם else, ולא עם elsif, כדי למנוע מימוש LATCH.

# משפט case

## יש להוסיף לכל משפטcase מצב של when others.

## למצב when others יש משמעות אמיתית במשפט case, בו הסיגנל הנשאל הוא לא מסוג enumerated type.

## במשפט case בו הסיגנל הנשאל הוא מסוג enumerated type, יוגדר טיפול (משפט when) לכל מצב.

# שם של Instance

## שם של Instance צריך להיות שווה לשם ה- entity, \_ אחריו, ו- inst אחריו.

* לדוגמא: אם שם ה- entity הוא mult\_8x8 אז Instance שלו יהיה בעל שם: mult\_8x8\_inst.

## אם יש יותר מ- Instance אחד ל-entity מסוים, אז יש להוסיף מספור.

* לדוגמא: mult\_8x8\_1\_inst.

## יוצא מין הכלל יכול להיות שם של Instance ב- TB המיועד לבדיקה: רצוי ששמו יהיה dut.

# מיפוי

## מומלץ לתת שמות לסיגנלים, בעת ההשמה של ה- Instance, הזהים לשמות של ה-entity (עד כמה שניתן).

## חיבור ה-ports , כלומר ה- port map, יעשה באמצעות שמות ולא ע"פ סדר.

## גם generic map יעשה באמצעות שמות ולא ע"פ סדר.

# Package

## במקרה שקיימים constants, procedures, function שמשותפים לכל הפרויקט, אז רצוי להכין package ובכל entity שזקוק להם יש להשתמש ב- use work.xxx\_pkg.all.

# Ports

## Ports יהיו רק מסוג: in, out, inout (ולא buffer).

## Ports חיצוניים של ה- FPGA (או IP האמור להיות בשימוש חוזר) יהיו רק מסוג:

* std\_logic
* std\_logic\_vector

## שימוש ב-

* integer
* boolean
* unsigned
* signed
* user defined type

יעשה רק בארכיטקטורה הפנימית של ה- FPGA.

## כל port יוגדר בצורה מלאה וברורה עם הכיוון שלו והסוג שלו.

* ניתן ב- VHDL להגדיר לקבוצה שלמה את הכיוון ו/או הסוג – הדבר אינו מומלץ עקב פגימה בקריאות.

## אין לאתחלports בהגדרת ה- ports ב-entity , בקוד המיועד לסינתזה.

* ניתן לעשות זאת בקוד המיועד לסימולציה, ואף מומלץ.

# הגדרת טווח

## שימוש ב- integer יעשה רק עם הגדרת טווח.

## שימוש ב- std\_logic\_vector יעשה עם הגדרת טווח.

* מלבד עבור הגדרת פונקציה או פרוצדורה.
* רצוי להשתמש ב- downto (ולא to).
* רצוי שה- LSB יהיה 0 (למשל: 7 downto 0 ולא 8 downto 1).

# Shared variables

## אין להשתמש ב-shared variable בקוד המיועד לסינתזה.

## כן ניתן להשתמש ב-shared variable בקוד המיועד לסימולציה.

## מומלץ להשתמש ב- protected type עבור shared variables.

# לולאות

## אין להשתמש ב- while loopבקוד המיועד לסינתזה.

## כן ניתן להשתמש ב- for loop בקוד המיועד לסינתזה.

## ניתן להשתמש בשתיהן בקוד המיועד לסימולציה.

# רשימת רגישויות

## ברשימת הרגישויות של process סינכרוני ימצאו רק השעון וה-RESET .

* מומלץ לעבוד עם RESET שעבר סינון וסונכרן לשעון של אותו FF. לא מומלץ להשתמש ב- RESET (אסינכרוני) שמגיע ישירות ככניסה ל- FPGA.
* מומלץ להשתמש בפונקצית rising\_edge (או falling\_edge, אך רק אחת מהן לאותו FF).

## ברשימת הרגישויות של process אסינכרוני ימצאו כל הסיגנלים המשפיעים (ורק הם).

# Reset

## לא יעשה שימוש ב- Global Reset מלבד הכנסתו לקו ה- RESET/SET של ה- FF. כלומר, אין להשתמש בו כלוגיקה כלשהי.

## רק ה- Global Reset יכנס לקו ה- RESET/ SETשל ה- FF.

## מספר ה- RESET-ים הסינכרוניים יהיה כמספר השעונים בתכן.

## יש לסנן את כניסת ה- RESET האסינכרונית ל- FPGA.

## יש להחזיק את ה- FPGA במצב RESET כל עוד ה- LOCK של ה- PLL לא מחווה על נעילה.

## יש להקפיד שכלsignal סינכרוני יאותחל במצב RESET.

## רצוי להשתמש ב- generic בכל entity (שיש בו (RESET, הקובע את הפולאריות של ה- Reset. שם ה- generic יכול להיות reset\_polarity\_g.

# Clock

## שעון שבשימוש ב-FPGA יהיה רק:

* משעון המיוצר מבחוץ ל-FPGA .
* משעון יציאה שלPLL פנימי.

ולא ע"י חלוקה של שעון אחר באמצעות לוגיקה.

## אין להכניס מוצא של FF אחד לכניסת שעון של FF אחר.

## יש להשתמש בשעון רק ככניסת שעון ל- FF.

* ולא, לדוגמא, ככניסת ה- DATA (או כניסת ה- ENABLE) של FF.

## יש להימנע, ככל הניתן, מהשמה של שעון אחד לשעון אחר (rename) עקב בעיית Tick-ים (דלתאות) של סימולציה.

# Latch

## אין לכתוב קוד המממש latch.

## ב- process אסינכרוני חייבת להיות השמה ברורה, עם ערך ברור, לכל מצב.

* אין else בלי השמה ויש else לכל if.
* יש לסיים משפט if עם else (ולא עם elsif).

## בצורה דומה: ל- process חבוי, חייבת להיות השמה ברורה, עם ערך ברור, לכל מצב (משפט when ולא if).

# Feedback

## אין להשתמש ב- FEEDBACK בלוגיקה אסינכרונית.

# Clock Domain Crossing

## כל כניסה אסינכרונית ל- FPGA צריכה להידגם פעמיים לפני השימוש בה.

* אלא אם מדובר ב- BUS אסינכרוני של DATA, שמגיע עם קו בקרה של VALID, ואז מספיק לדגום פעמיים רק את קו הבקרה.

## אין למקם לוגיקה במעבר DATA בין CLOCK DOMAIN אחד ל-CLOCK DOMAIN שני.

## אין למקם לוגיקה במעבר ה- DATA בין ה- FF הראשון של ה- SYNCHRONIZER לבין ה- FF השני של ה- SYNCHRONIZER, ואין להשתמש ב- FEEDBACK של אותו DATA בין שני ה- FF-ים.

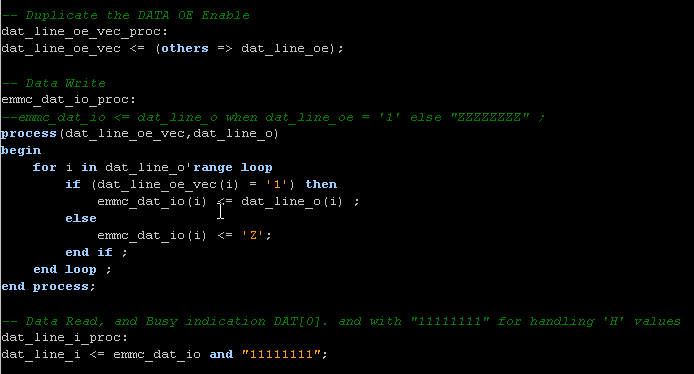
# Outputs sampling

## רצוי שכל יציאה מה-FPGA תהיה דגומה.

## רצוי שכל יציאה מבלוק פונקציונאלי תהיה דגומה.

# Inout

## יש לטפל ב- port מסוג INOUT בדומה לדוגמא הבאה:



ה- OE משוכפל כרוחב ה- DATA. ה- IO מקבל את ההשמה שמגיעה מה- FPGA כאשר ה- OE מאופשר, אחרת מקבלת 'Z'. הערך שנכנס ל- FPGA שווה תמיד ל- IO. בנוסף, בדוגמה המצורפת, יש PU ב- TOP של הסימולציה, כך שכאשר שיש ערך על הקו של '0' או 'H', ולכן יש הכפלה (and) עם '1' כדי לזהות ערך של '1' ('H' and '1' = '1').

הסיגנלים שנדחפים ע"י ה- FPGA, לדוגמא:data\_line\_o ו- dat\_line\_oe\_vec צריכים להיות ברגיסטרים של ה- PAD, וגם הדגימה של dat\_line\_i צריכה להיות ברגיסטר של ה- PAD.

# Pipeline

## מומלץ להימנע משימוש במשפט if, בו מופיעים תנאים רבים, בכדי לעבוד בתדר גבוה.

דוגמא לקידוד לא יעיל מבחינת ביצועי תדר (יעמוד בתדר נמוך):

**If** (hcnt > h\_min) **and** (hcnt < h\_max) **and** (vcnt > v\_min) **and** (vcnt < vmax)

**and** (hcnt < frame\_right) **and** (hcnt > frame\_left) **and** (vcnt < frame\_low) **and** (vcnt > frame\_high) then

…

end if;

דוגמא לכתיבה יעילה יותר מבחינת ביצועי תדר (יעמוד בתדר גבוה יותר בזכות הוספת ה- pipeline):

**If** hcnt\_cond\_hmin **and** hcnt\_cond\_hmax **and** hcnt\_cond\_l\_frame **and** hcnt\_cond\_r\_frame **and** vcnt\_cond\_vmin **and** vcnt\_cond\_vmax **and** vcnt\_cond\_u\_frame **and** vcnt\_cond\_l\_frame then

…

end if;

כאשר כל תנאי ייכתב ב-process נפרד, לדוגמא:

hcnt\_max\_proc : process (clk, rst)

begin

if (rst = reset\_polarity\_g) then

hcnt\_cond\_hmax <= ‘0’;

elsif rising\_edge (clk) then

if (hcnt < frame\_right) then

hcnt\_cond\_hmax <= ‘1’;

else

hcnt\_cond\_hmax <= ‘0’;

end if;

end if

end process hcnt\_max\_proc;

# State Machine

## מצורפת דוגמא לאופן כתיבת State Machine.

# מצביע מסוג סיגנל

## רצוי לא להשתמש במצביע שהוא סיגנל, בקוד המיועד לסינתזה, אלא אם מדובר במימוש של זיכרון.

לדוגמא: במקום לקלוט מידע טורי לרגיסטר ע"פ מיקום של counter, כלומר:

val\_arr (counter) <= serial\_in;

יש לקלוט את המידע באמצעות Shift Register:

val\_arr (7 downto 0) <= val\_arr (6 downto 0) & serial\_in;

