

הטכניון - מכון טכנולוגי לישראל
הפקולטה להנדסת חשמל



מעבדה 1,1 ח'

ניסוי HDL 2 - חומר רקע
הכרת תכן VHDL וכתיבה לסינתזה

גרסה 1.2

אביב 2017

מסכמים: אברהם קפלן, דודי בר-און
על פי חוברת של עמוס זסלבסקי מ 2009
וחומר מהרצאות קורס תכן לוגי 044262

תוכן עניינים

4	מבוא לכתיבת קוד לסינתזה צירופית	1
4	1.1 הקפדה על רשימת רגישות מלאה	
5	1.2 הקפדה על שימוש בהשמות מלאות	
8	1.3 default output assignment(s) אתחול יציאות	
9	1.4 Variables שימוש נכון ב- Variables	
10	2 כללי כתיבת קוד בסיסיים לסינתזה סינכרונית	
12	2.1 כניסות אסינכרוניות בקוד של מערכת סינכרונית	
12	2.2 השמות סינכרוניות של אותות וביטויים	
14	2.3 התניות באזור השמות סינכרוני	
15	3 מכונות מצבים:	
17	3.1 כתיבה מכונה סינכרונית, יציאות אסינכרוניות –	
17	3.2 מכונת MOORE	
18	3.3 מימוש מכונת בחירת המצב הבא:	
19	3.4 מימוש יציאה שאינה תלויה בכניסה מכונת – MOORE	
20	3.5 כתיבה סינכרונית טהורה	
21	3.6 הגדרת שמות המצבים של מכונה	
22	4 הירארכיה בVHDL	
23	5 המלצות לתכן נכון:	
23	6 חומר עזר	

1 מבוא לכתיבת קוד לסינתזה צירופית

לא כל קוד שנכתב בשפת VHDL מסוגל לעבור סינתזה. קיימים כללים רבים לכתיבה תיאורים שמתאימים לסינתזה. בכמה כללים חשובים נעסוק בניסוי זה. לקבלת תמונה רחבה יותר על הנושא, תוכל להיעזר בספרות (ראה רשימת ספרות שמצויה בסוף חומר הרקע של מפגש זה) ובתיעוד של יצרני כלי הסינתזה הספציפי שאתו אתה עובד.

בתת-הפרקים הראשונים נתמקד בכמה כללים בסיסיים לכתיבת תיאורים התנהגותיים צירופיים שמתאימים לסינתזה. בתת-פרקים שבהמשך נתמקד גם בכמה כללים בסיסיים לכתיבת תיאורים התנהגותיים סינכרוניים.

בפרקים אלו תכיר שבלונות (Templates) לכתיבה, שמבטיחות שכל כלי הסינתזה יצליחו לפרש את הקוד שלך, כחמרה של מערכת צירופית או סינכרונית ולא כחמרה מופרעת ובלתי הגיונית (Sick Hardware). הקפדה על כללי הכתיבה שנלמדים בפרקים אלו תגרום לכך שהקוד יפורש באופן אחיד ועקבי על ידי כל כלי התכנה שבהם נשתמש. מדובר בפירוש עקבי ואחיד שינתן לקוד על ידי הכלים הבאים:

- כלי הסימולציה וכלי הסינתזה
- כלי סינתזה שונים
- כלי הסימולציה לפני הסינתזה וכלי הסימולציה אחרי הסינתזה

כדאי לדעת שכתובת קוד שהיא "ידידותית לכלי סינתזה", חופפת ברוב המקרים גם לכתיבת קוד שהיא "קריאה וידידותית לבני אדם"!

1.1 הקפדה על רשימת רגישות מלאה

קיימים כמה כללים חשובים בכתיבת קוד של מערכת צירופית

- הקפדה על רישום רשימות רגישות מלאה (Complete sensitivity list)
- הקפדה על שימוש בהשמות מלאות (Complete Assignments)
- הקפדה על שימוש נכון במשתנים (Variables)

בחלק זה נעסוק בכלל הראשון מבין השלושה והוא רלוונטי רק כאשר מדובר בתיאור מערכת צירופית על יד תהליך (process). נדגים את הבעיה באמצעות דוגמת הקוד הבאה.

```
- bad code for a three input OR gate
library ieee ;
use ieee.std_logic_1164.all ;
entity comrule1 is
    port ( a,b,c : in  std_logic ;
          y      : out std_logic ) ;
end comrule1 ;
architecture arc_comrule1 of comrule1 is
begin
    process (b ,c) -- a is missing from sensitivity list
begin
        y <= a or b or c ;
    end process ;
end arc_comrule1 ;
```

בדוגמת הקוד הנ"ל רצינו לתאר שער or בעל שלוש כניסות (a, b ו c), אך בתהליך שבו השתמשנו לתיאור המערכת "הצירופית" הנ"ל לא הקפדנו שרשימת הרגישות תכלול את כל

הכניסות של המערכת. כפי שאפשר לראות מדוגמת הקוד הנ"ל, הכניסה a נשכחה מרשימת הרגישות.

נבדוק מהי ההתנהגות של המערכת. כל שינוי באות b או c מפעיל את התהליך באופן מידי. הפעלה זו עשויה לשנות את מצב היציאה y. לעומת זאת, כאשר מתרחש שינוי ב - a, לא קורה דבר באופן מידי, מכיוון שהתהליך אינו מופעל. קטע הקוד הנ"ל בוודאי לא מתנהג בסימולציה כשער OR בעל שלוש כניסות !

להלן דוגמה לתוצאות סימולציה שהתקבלו.

ns	a	b	c	y	0	0	1	0	1
100	0	0	0	0					
200	1	0	0	0	←	bad	result		
300	0	0	0	0					
400	0	1	0	1					
500	1	0	0	1	←		?		

כפי שאפשר לראות בתוצאות הסימולציה, המערכת אינה מתנהגת כשער OR (התבונן בתוצאות של השורה השלישית). בעצם המערכת אינה מתנהגת כלל כמערכת צירופית (השווה בין השורה השלישית והשורה האחרונה).

על כתיבת קוד מסוג זה כלי סינתזה עשויים להודיע הודעות אזהרה שנראות למשל כך :

Warning, **a** should be declared on the sensitivity list of the process.

כלי סינתזה אחרים, שאינם בודקים את התכולה של רשימת הרגישות, עלולים בכל זאת לסנתז את הקוד כשער OR רגיל וללא הודעות אזהרה כל שהן. הקורא עלול לחשוב שזוהי תוצאה רצויה, היות וכלי הסינתזה מתקן לכאורה את הליקוי בסגנון הכתיבה שלנו ובכל זאת מסנתז את החמרה שככל הנראה לה אנו מצפים. אין הדבר כך ! התוצאה הזו בהחלט אינה רצויה, מכיוון שאנו מקבלים חמרה שמתנהגת באופן שונה מהאופן שבו התנהגה הסימולציה שנעשתה לפני הסינתזה.

בהחלט יכול להיות מצב שבו הסימולציה שנעשתה לפני הסינתזה עברה בהצלחה דווקא מכיוון שהמערכת התנהגה באופן המוזר שבה מתנהגות מערכות שבהן משמיטים כניסות מרשימת הרגישות !

הדוגמה הפשוטה הנ"ל, ממחישה לנו מדוע חשוב להקפיד לרשום את כל הכניסות של מערכת צירופית ברשימת הרגישות. כניסות הן כל האותות שרשומים בצד ימין של השמות בתהליך.

כניסות הן לא רק כל האותות שרשומים בצד ימין של פסוקי השמה, אלא גם כל האותות שרשומים בתנאים של פסוקי if וכל האותות שרשומים בביטויים הנבדקים בפסוקי case. קל מאוד לשכוח לרשום כניסות ברשימת הרגישות ובעיקר כשמדובר בתהליך מאוד מסובך, שעובר שינויים ושיפורים רבים במהלך התכנון שלו. לרוע המזל לא תמיד קל לעלות על בעיה זו בזמן הסימולציה. ראו הוזהרתם !

1.2 הקפדה על שימוש בהשמות מלאות

כלל חשוב נוסף בתיאור של מערכת צירופית באמצעות תהליך הוא, להקפיד לרשום השמות מלאות. השמות מלאות הן השמות שבהן כל הפעלה של התהליך מבצעת השמה כל שהיא לכל היציאות של המערכת שאותה מתאר התהליך. מה יקרה אם לא נקפיד על כלל חשוב זה ? קטע הקוד הבא מנסה לתאר משווה שמפיק '1' לוגי ביציאה שלו (שנקראת a_gt_b) כאשר הערך של הכניסה a גדול מהערך של הכניסה b.

```
-- bad code for comparator (with process)
entity comrule2 is
    port (a , b : in integer range 0 to 7 ;
          a_gt_b : out bit ) ;
end comrule2 ;

architecture arc_comrule2 of comrule2 is
begin
    process ( a , b )
    begin
        if a > b then
            a_gt_b <= '1';
        end if ;
    end process ;
end arc_comrule2 ;
```

להלן דוגמה לתוצאות סימולציה של המערכת הנ"ל :

ns	a	b	a_gt_b
0	1	2	0
100	4	2	1
200	4	6	1
300	1	2	1

← error

כפי שאפשר לראות מתוצאות הסימולציה, המערכת אינה מתנהגת כמשווה (התבונן בתוצאות של השורה השלישית). בעצם המערכת אינה מתנהגת כלל כמערכת צירופית (השווה בין השורה הראשונה והשורה האחרונה).

מדוע המערכת אינה פועלת כנדרש ? בצירוף הנבדק הראשון, האות a אינו גדול יותר מהאות b.

אמנם התהליך מופעל אך התנאי של הפסוק if אינו מתקיים ולכן ההשמה לאות a_gt_b אינה מתבצעת. המצב '0' לוגי שמפיק האות a_gt_b הוא בעצם המצב של ברירת המחדל של סוג המידע bit.

בצירוף הנבדק השני, האות a נהפך לגדול מהאות b. התהליך מופעל והתנאי של הפסוק if מתקיים ולכן ההשמה של '1' לוגי לאות a_gt_b אכן מתבצעת. בצירוף הנבדק השלישי האות a נהפך שוב לקטן יותר מהאות b. אמנם התהליך מופעל, אך התנאי של הפסוק if אינו מתקיים ולכן ההשמה לאות a_gt_b אינה מתבצעת. היות ואותות (כמו גם משתנים בתהליך) בשפת VHDL הם

אובייקטים סטטיים (כלומר שומרים על מצבם כאשר לא מציבים אליהם ערך חדש), המצב של האות a_gt_b נשאר כפי שהוא היה קודם - כלומר '1' לוגי.

בכל הצירופים שבהמשך האות a_gt_b יישאר תמיד ב - '1' לוגי וזאת ללא תלות במצבן של הכניסות a ו b. בכדי שתופעה כזו תקרה בחמרה אמיתית, החמרה חייבת להכיל אלמנט של זיכרון (Storage או Memory), כלומר המערכת המתוארת אינה מערכת צירופית. המערכת הנ"ל מורכבת בעצם ממשווה שהיציאה שלו מוזנת ל - Parasitic Latch. זהו רכיב, שמהרגע שהוא מוזן ב - '1' לוגי, הוא מפיק ביציאה שלו '1' לוגי לעולמים. כמובן שזהו רכיב לא שימושי וכל מערכת החמרה שנוצרה יכולה להיחשב כ - Sick Hardware.

יצירת רכיבי Latch שלא במתכוון בכלל ויצירה של רכיבי Latch שנתקעים במצב מסוים בפרט (כמו בדוגמה הנ"ל), הם בהחלט תוצרים בלתי רצויים.

כלי סינתזה בהחלט עלולים לנסות לסנתז רכיבי Latch שלהם כלל לא התכוונו מלכתחילה. כלי סינתזה טובים עשויים להודיע לנו הודעות אזהרה כל שהן, במקרה של ביצוע פעולה מסוג זה. להלן דוגמאות להודעה מסוג זה.

```
Warning, a_gt_b is not always assigned. Storage may be needed..
```

כלי סינתזה עשויים להודיע אף הודעות שמצביעות על הבעיה באופן עקיף, כמו למשל הודעות מהסוג הבא:

```
Warning: Pin "a_gt_b" stuck at VCC
```

```
Warning: Design contains input pin(s) that do not drive logic
```

כאשר מתארים מערכת צירופית באמצעות תהליך, חייבים לרשום השמות מלאות. השמות מלאות הן השמות שבהן כל הפעלה של התהליך, מבצעת השמה כל שהיא לכל היציאות של המערכת שאותה מתאר התהליך. כיצד מיישמים את הכלל הנ"ל באופן מעשי? נשתמש בדוגמה הנ"ל של המשווה.

קטע הקוד הבא מציג פתרון אפשרי של הבעיה עבור המקרה הפשוט של המשווה.

```
-- avoiding lathes through covering all possibilities
entity solution1 is
    port (a , b : in integer range 0 to 7 ;
          a_gt_b : out bit
          ) ;
end solution1 ;

architecture arc_solution1 of solution1 is
begin
    process ( a , b )
    begin
        if a > b then
            a_gt_b <= '1';
        else
            a_gt_b <= '0';
        end if ;
    end process ;
end arc_solution1 ;
```

הפתרון מושג על ידי כיסוי כל האפשרויות באמצעות שימוש בפסוק else ורישום השמה גם עבור המקרים שבהם האות a קטן מהאות b.

בתהליכים שבהם קיימים פסוקי if ו case מורכבים שרשומים אחד בתוך השני (Nesting) וקיימות הרבה יציאות, השיטה הנ"ל אינה מעשית. קטע הקוד הבא מציג פתרון מעשי נוח יותר כאשר מדובר בתהליכים מורכבים כנ"ל.

1.3 אתחול יציאות (default output assignment(s))

```
entity solution2 is
    port (a , b : in integer range 0 to 7 ;
          a_gt_b: out bit
          );
end solution2 ;

architecture arc_solution2 of solution2 is
begin process ( a , b )
begin
    a_gt_b <= '0'; -- default output assignment(s)
    if a > b then
        a_gt_b <= '1';
    end if ;
end process ;
end arc_solution2 ;
```

בשיטה זו רושמים בתחילת התהליך רשימת של השמות ברירת מחדל (default assignments) לכל היציאות של המערכת. באופן כזה, אם בהמשך התהליך אין השמה לאות יציאה מסוים באחד התרחישים האפשריים, ההשמה שרשומה בתחילת התהליך מתקיימת.

בשפת VHDL לא נוצרת בעיה כל שהיא אם אות יציאה מסוים מקבל בתהליך השמה שהיא שונה מההשמה שרשומה בתחילת התהליך. לא נוצר Spike כל שהוא ! במקרה של ריבוי השמות, לסדר של ההשמות יש משמעות ורק ההשמה האחרונה קובעת.

1.4 שימוש נכון ב - Variables

ראינו במפגש הראשון של הניסוי שניתן להשתמש במשתנים בתוך תהליך. משתנים מסוגלים כמובן לאחסן תוצאות ביניים של חישובים, ומאפשרים לנו לתאר מערכות צירופיות מורכבות.

שימוש בלתי זהיר במשתנים, עלול לגרום לקוד לא להתנהג כמערכת צירופית, אלא כ - Sick Hardware. הקוד הבא מציג דוגמה מאוד פשוטה.

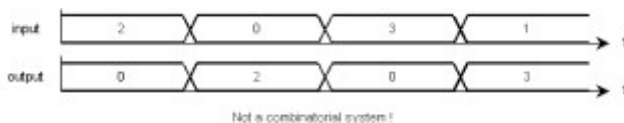
```
entity badvar is
  port ( input: in integer range 0 to 3 ;
         output: out integer range 0 to 3 );
end badvar ;

architecture arc_badvar of badvar is
begin
  -- input & output are integer (range 0 to 3) signals
  process ( input )
    variable var: integer range 0 to 3 := 0 ; -- init to 0
  begin
    -- reading variable before it's assigned a value
    output <= var ;
    var := input ;
  end process ;
end arc_badvar ;
```

במבט ראשון נראה שהמערכת הנ"ל מתנהגת פשוט כאוסף חוטים. נבדוק האם אכן כך הוא הדבר. להלן דוגמה לתוצאות סימולציה :

ns	input	output
0	2	0
100	0	2
200	3	0
300	1	3

התוצאות תיראינה כך בחלון wave.

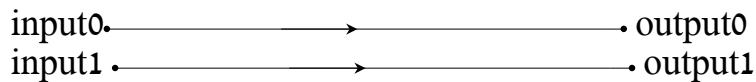


זוהי בוודאי אינה התנהגות של מערכת צירופית ! מדוע המערכת הנ"ל מתנהגת כך ? בהפעלה הראשונה של התהליך אות הכניסה input הוא 2. ההשמה הראשונה ל - output אינה מתבצעת במהלך הביצוע של התהליך, אלא נרשמת לתור (Queue) וההשמה תתבצע בהמשך. הערך שנרשם לביצוע הוא הערך המקורי של המשתנה var שהוא ברירת המחדל 0. בהמשך ביצוע התהליך, האות var מתעדכן בערך החדש 2, אך העדכון כבר מאוחר מדי עבור חישוב ערכו של output. כאשר ביצוע התהליך מסתיים ולאחר זמן השהייה של delta, ההשמה ל - output מתבצעת עם ערכו הישן של var שהוא 0.

בהפעלה השנייה של התהליך אות הכניסה input הוא 0. ההשמה הראשונה ל - output אינה מתבצעת במהלך הביצוע של התהליך, אלא נרשמת לתור וההשמה תתבצע בהמשך. הערך

שנרשם לביצוע הוא הערך הקודם של המשתנה var, שהוא 2. בהמשך ביצוע התהליך האות var מתעדכן בערך החדש 0, אך העדכון כבר מאוחר מדי עבור חישוב ערכו של output. כאשר ביצוע התהליך מסתיים ולאחר זמן השהייה של delta, ההשמה ל output מתבצעת עם ערכו הישן של var שהוא 2.

כיצד יסנתזו כלי סינתזה את הקוד הנ"ל ? רוב כלי הסינתזה יסנתזו את הקוד הנ"ל כמערכת חוטים.



ישנם כלים שעשויים לשגר גם הודעת אזהרה שנראית למשל כך :

Warning: Local variable 'var' is being read before its value is assigned, This may cause simulation not to match synthesis.

בכדי להימנע מבעיות דומות בשימוש במשתנים בתיאור של מערכת צירופית, צריך להקפיד על הכלל הבא :

אל תקרא משתנה לפני שתציב למשתנה ערך חדש !

קיום של דרישה זו מבטיח שהשימוש שייעשה במשתנה יהיה כבאובייקט חישובי ולא כבזיכרון (Storage או Memory).

קל להקפיד באופן מעשי על השימוש בכלל זה אם מפרקים את התהליך לשני חלקים. בחלק העליון של התהליך מרשים לעדכן משתנים מסיגנלים. בחלק התחתון של התהליך מרשים לעדכן סיגנלים ממשתנים. באופן כזה לעולם לא מתבצע ערבוב מסוכן שאינו מקיים את הכלל הנ"ל.

שים לב: הכלל הנ"ל של שימוש נכון במשתנים תקף גם במקרה של תיאור מערכת סינכרונית !

ובקיצור :

הצבה SIGNAL WIRE	<code>signal_name <= expression ;</code> <code>variable_name := expression ;</code>
VARIABLE מגיב מידית ללא זיכרון SIGNAL מגיב רק בשעון הבא	<pre> signal a,b,c : std_logic_vector(0 to 3); variable var : std_logic_vector(0 to 3); a <= "0000"; var := "0000"; ... if (rising_edge(clk)) then var := "1111"; a <= "0001"; b <= var; c <= a ; end if; </pre>
והתוצאה תהיה :	<code>b="1111" c= "0000" a = "0001" var ="1111"</code>

2 כללי כתיבת קוד בסיסיים לסינתזה סינכרונית

תיאור סינכרוני נעשה בדרך כלל באמצעות תהליך שכולל תנאי if בעל המבנה הבא :

```
clk'event and clk = '1'
```

הביטוי clk'event מחזיר ערך בוליאני true כאשר הערך של האות clk משתנה. שילוב של ביטוי זה עם התנאי clk = '1' גורם לכך שכל הביטוי הזה מחזיר true כאשר האות clk עולה (rising).

להלן דוגמה לתיאור של Flip-Flop מסוג D.

```
-- a simple D flip-flop
entity dff4 is
  port ( clk : in bit ;
        d   : in bit ;
        q   : out bit ) ;
end dff4 ;

architecture arc_dff4 of dff4 is
begin
  process(clk)
  begin
    if clk'event and clk = '1' then
      q <= d;
    end if ;
  end process ;
end arc_dff4 ;
```

No else or elsif

שים לב שההתניה לא נכתבה באופן הבא :

```
if clk'event and clk = '1' then
  q <= d;
else
  q <= q;
end if ;
```

הרישום האחרון (הכולל else) הוא מיותר היות ואותות הם סטטיים (כלומר שומרים על מצבם כל עוד לא מבצעים עליהם השמה). חשוב להבין שלא מדובר כאן רק ברישום מקוצר אלא קיימת בעצם המלצה מפורשת להימנע משימוש בתנאי else (או elsif) אחרי התנאי הסינכרוני הראשי (כלומר אחרי התנאי שמכיל את הביטוי clk'event and clk = '1'). הסיבה לכך היא שכל השמה אחרת ששונה מההשמה q <= q; (למשל השמה q <= p) יוצרת Sick Hardware שאינו מתאר חמרה אמיתית כל שהיא ובכלי סינתזה אחדים קיים איסור גורף מלרשום רישום כזה ואפילו כשמדובר בהשמה q <= q.

כאשר האות clk הוא מסוג std_logic ניתן להחליף את התנאי הסינכרוני הנ"ל בתנאי הבא :
rising_edge(clk)

תיאור מערכת שרגישה לירידה נעשה באופן דומה באחד משני האופנים הבאים :

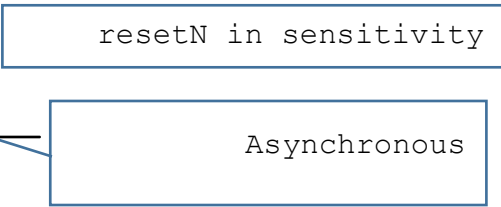
```
faling_edge(clk)
clk'event and clk = '0'
```

2.1 כניסות אסינכרוניות בקוד של מערכת סינכרונית

במערכת הקודמת לא היה אתחול חד משמעי לFF. בקטע הקוד הבא מתארים פליפ-פלופ עם כניסת איפוס אסינכרונית פעילה בנמוך שנקראת .resetN

```
-- sync system with async resetN
library ieee ; use ieee.std_logic_1164.all ;
entity dff12 is
    port ( clk      : in  std_logic ;
          resetN    : in  std_logic ;
          d         : in  std_logic ;
          q         : out std_logic ) ;
end dff12 ;

architecture arc_dff12 of dff12 is
begin
    process ( clk , resetN )
    begin
        if resetN = '0' then
            q <= '0';
        elsif rising_edge(clk) then
            q <= d;
        end if ;
    end process ;
end arc_dff12;
```



שים לב שהתנאי הא-סינכרוני מקדים את התנאי הסינכרוני, כלומר הוא בעל עדיפות גבוהה יותר.

זהו כמובן המצב גם בחמרה סינכרונית אמיתית שבה הכניסות הא-סינכרוניות חזקות יותר מהכניסות הסינכרוניות.

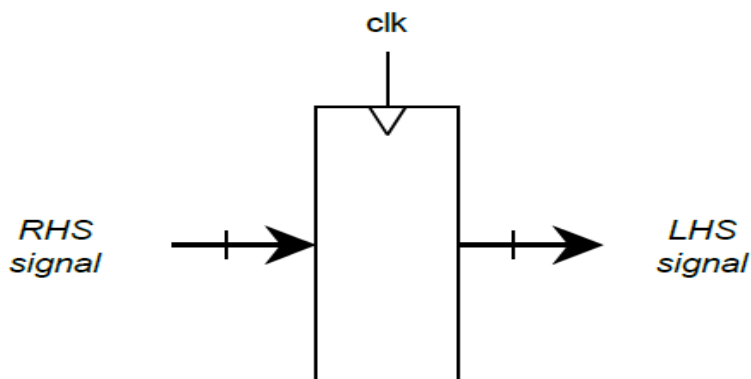
שים גם לב לכך שרשימת הרגישות אינה צריכה להיות מלאה והיא מכילה את אות השעון ואות האיפוס הא-סינכרוני (בדוגמה זו). רשימת הרגישות אינה צריכה לכלול כניסות סינכרוניות כמו הכניסה D מכיוון שגם בחמרה אמיתית, שינוי בכניסה D אינו יוצר פעילות כל שהיא במערכת (רק שינויים באות השעון ו resetN יוצרים פעילות).

באופן דומה ניתן היה ליצור מערכת בעלת כניסת preset (על ידי שינוי ההשמה באזור הא-סינכרוני ל - 'q <= '1';) או מערכת בעלת כניסת טעינה א-סינכרונית (על ידי שינוי ההשמה באזור הא-סינכרוני ל - q <= din; והוספת din לרשימת הרגישות).

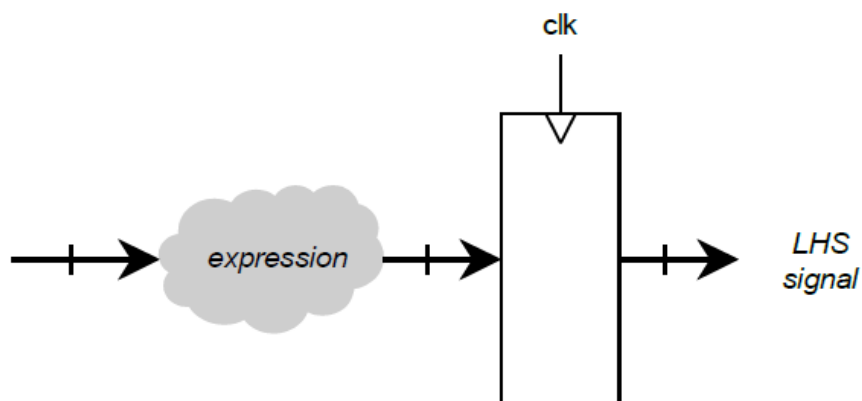
2.2 השמות סינכרוניות של אותות וביטויים

שים לב לכללים הבאים :

- כל השמה באזור הסינכרוני יוצרת פליפ-פלופ (או רגיסטר במקרה של אות וקטורי)
 - צד שמאל של ההשמה מהווה את היציאות של הפלי-פלופ (רגיסטר)
 - אות בצד ימין של ההשמה מהווה את כניסת הפליפ-פלופ (רגיסטר)
- (Right Hand Side = RHS) : ו (Left Hand Side = LHS) האזור הבא מדגים זאת

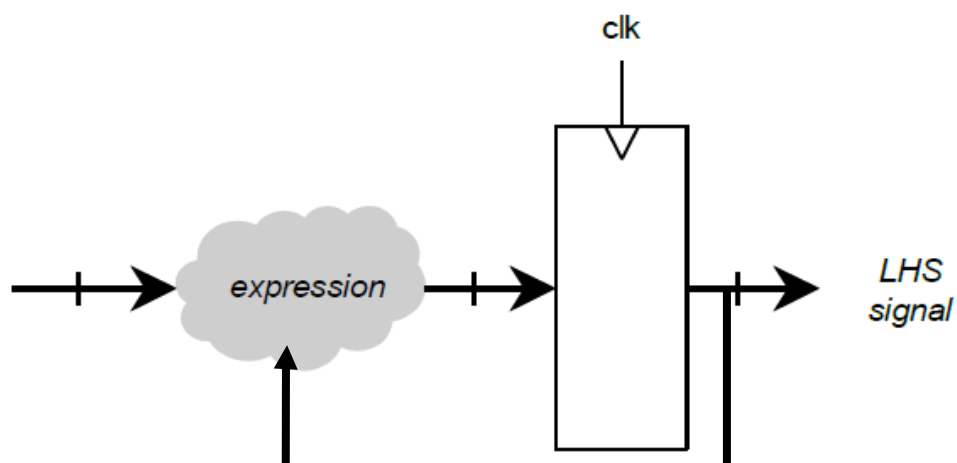


כל ביטוי (expression) בצד ימין של ההשמה מהווה מערכת צירופית שנמצאת לפני הכניסות של הפליפ-פלופ (רגיסטר) - האיור הבא מתאר מקרה כזה.



צד ימין שמכיל אותות שמופיעים גם בצד שמאל של ההשמה מתאר משוב צירופי בחמרה.

השמות מסוג זה מתארת משוב (feedback), שמגיע מהיציאות של הפליפ-פלופים, כפי שהוא מתואר באיור הבא:



למערכת הצירופית שמזינה באיור האחרון את הכניסות של הפליפ-פלופים קוראים לעתים גם בשם: מערכת העירור (Excitation) למצב הבא (Next State) של המכונה. להלן דוגמת קוד שמתארת מונה בעל יחס חלוקה של 256 (מונה ברוחב 8 סיביות).

```
-- expression on the right side of an assignment
```

```

use ieee.std_logic_1164.all ;
use ieee.std_logic_unsigned.all ;
entity counter4 is
    port ( clk : in std_logic ;
          count: out std_logic_vector(7 downto 0));
end counter4 ;

architecture arc_counter4 of counter4 is
    signal cnt : std_logic_vector(7 downto 0);
begin
    process ( clk )
    begin
        if rising_edge( clk ) then
            cnt <= cnt + 1 ;
        end if ;
    end process ;
    count <= cnt ;
end arc_counter4 ;

```

שים לב לביטוי בצד ימין שכולל את האות cnt ופעולת חיבור. לשם כך דרושה הספרייה `ieee.std_logic_unsigned`

- בתרגיל זה לא בוצע איפוס למונה, דבר שאינו מומלץ כלל

2.3 התניות באזור השמות סינכרוני

גם התניות פנימיות שנמצאות מתחת ל - if הראשי מתארות בחמרה מערכות צירופיות שמזינות את כניסות הפליפ-פלופים (מותר להשתמש שם גם ב - else ו elif). להלן דוגמה לתיאור פליפ-פלופ עם אפשרות סינכרוני:

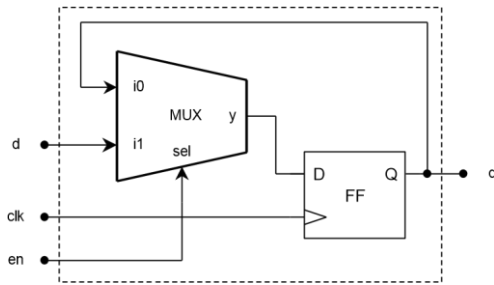
```

-- the good way to describe an enabled flip-flop
entity dff18 is
    port ( clk : in bit ;
          en  : in bit ;
          d   : in bit ;
          q   : out bit ) ;
end dff18 ;

architecture arc_dff18 of dff18 is
begin
    process ( clk )
    begin
        if clk'event and clk = '1'
        then
            if en = '1' then
                q <= d ;
            end if ;
        end if ;
    end process ;
end arc_dff18 ;

```

האיור הבא מציג את מבנה החמרה של המערכת שאותה מימשנו בדוגמה זו.



כניסת האפשר נקראת en והיא מחוברת למערכת צירופית שמייצרת את לוגיקת העירור לכניסה D של הפליפ-פלופ. הקוד הבא הוא קוד בלתי תקין שאינו מתאים ל - template הסינכרוני שמנסה לממש את המערכת הנ"ל.

```
process ( clk )
begin
    if clk'event and clk = '1' and en = '1' then
        q <= d ;
    end if ;
end process ;
```

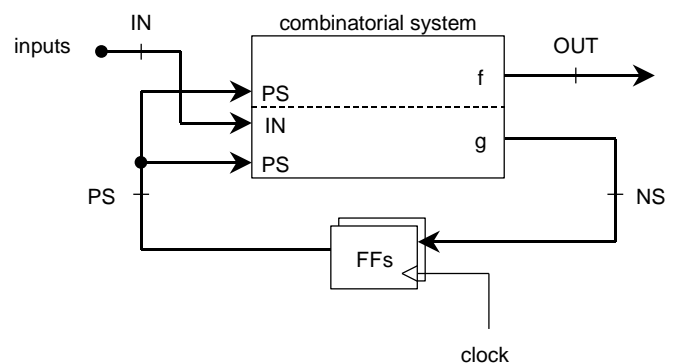
מומלץ לא להשתמש בסגנון הכתיבה האחרון בסינתזה !

3 מכונות מצבים:

מכונות מצבים משמשות לבנות אלגוריתם מסובך בו מצב המכונה והיציאות משנות מצב כתלות בכניסות ובמצב הקודם.

מונים ורגיסטרים שאותם הזכרנו קודם, הם מקרים פרטיים חשובים ונפוצים של מכונות מצבים סינכרוניות. נזכיר לקורא כמה מושגים חשובים במכונות מצבים כלליות.

ניתן לתאר מכונות מצבים כאוסף של פליפ-פלופים ומערכת צירופית שמחברים במבנה של משוב הבא. מכונה מסוג Moore.

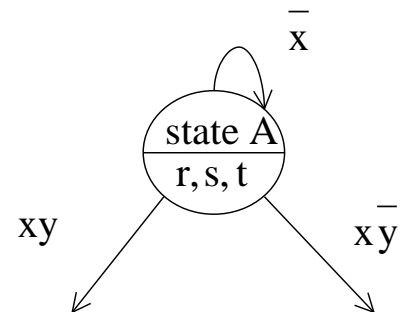


במכונות מסוג זה אין קשר צירופי בין הכניסות והיציאות של המכונה.

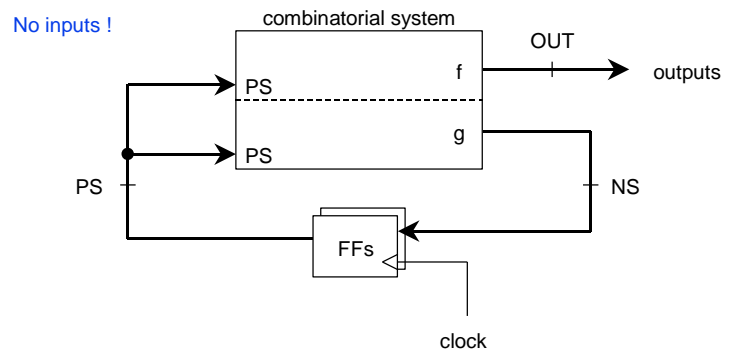
$$OUT = f(PS)$$

$$NS = g(IN, PS)$$

בכדי לגרום ליציאה להשתנות במכונות מסוג זה, יש לשנות את המצב הנוכחי של המכונה. בדיאגרמת מצבים של מכונות מסוג זה על כל החצים שיוצאים ממצב כלשהו יש יציאות זהות. בדרך כלל במקרה כזה רושמים את מצב היציאה בתוך העיגול של המצב עצמו כפי שהדבר רשום בדיאגרמת המצבים החלקית הבאה:



מכונה אוטונומית היא מכונה חסרת כניסות:

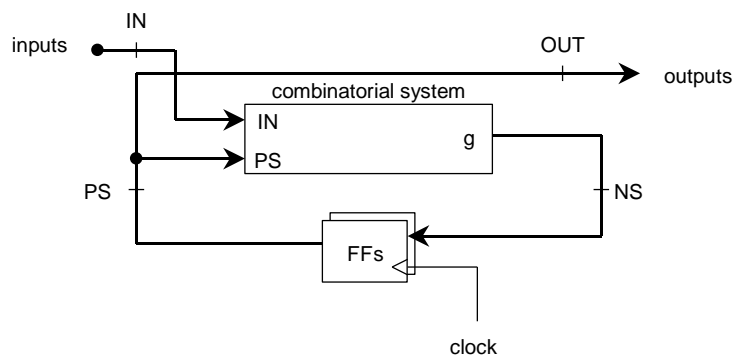


$$OUT = f(PS)$$

$$NS = g(PS)$$

זוהי מכונה ששימושית בדרך כלל בגנרטורים או במחוללי שעון.

מקרה פרטי חשוב מאוד של מכונות מצבים סינכרונית היא מכונת Moore ישירה (Direct Moore).



במכונה זו היציאות של המכונה מגיעות ישירות מהיציאות של הפליפ-פלופים של המכונה.

$$OUT = PS$$

$$NS = g(IN, PS)$$

מונים ורגיסטרים רבים שייכים לסוג זה של מכונות. כאשר מתכננים מכונת מצבים מסוג זה המתכנן חייב להתערב בהקצאת המצבים של המכונה בתהליך הסינתזה (מה שלא נדרש בדרך כלל בתכן של מכונות מסוגים אחרים). היתרונות של מכונה מסוג זה היא שהיא מפיקה ביציאה אותות נקיים ובעלי תזמונים קצרים (נדבר על נושא זה בהמשך).

ישנן מספר גישות לכתיבת מכונת מצבים :

- מכונת מצבים סינכרונית ואסינכרונית – לא בקורס שלנו !
- מכונת מצבים סינכרונית בלבד

3.1 כתיבה מכונה סינכרונית, יציאות אסינכרוניות –

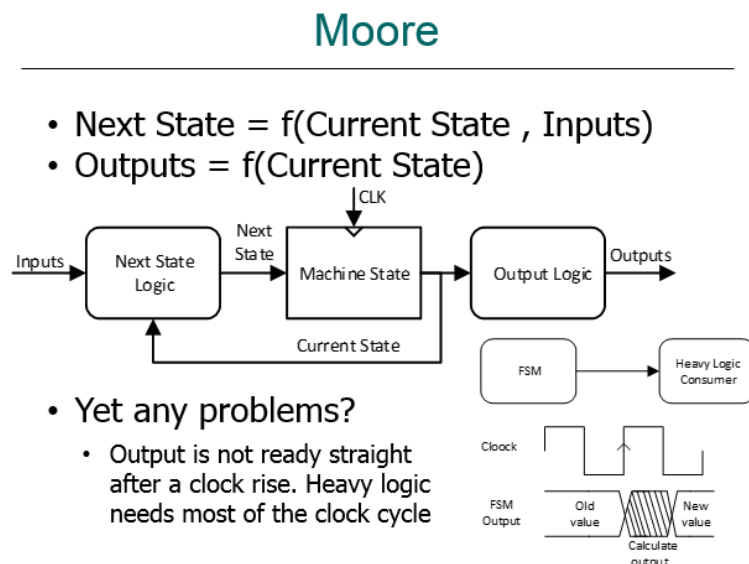
בשיטה זו, כל הקוד לבחירת מצב המכונה הבא הינו סינכרוני, אבל ישנו קטע קוד אסינכרוני הממיר ממצב המכונה הסינכרונית ליציאות

ישנן שתי שיטות מקובלות למכונת מצבים :

- מכונת MOORE
- מכונת MEALY - לא בקורס שלנו !

3.2 מכונת MOORE

במכונת MOORE היציאה תלויה רק במצב המכונה, מכונה זו מכילה יותר מצבים ממכונת MEALY



3.3 מימוש מכונת בחירת המצב הבא:

בצורה זו המימוש הוא הפשוט ביותר, יש משתנה אחד state שמתעדכן כל עלית שעון.

```
-- A Mealy machine has outputs that depend on both the state and
-- the inputs. When the inputs change, the outputs are updated
-- immediately, without waiting for a clock edge. The outputs
-- can be written more than once per state or per clock cycle.
```

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

entity mealy_4s is
  port
  (
    clk      : in    std_logic;
    data_in  : in    std_logic;
    reset    : in    std_logic;
    data_out : out   std_logic_vector(1 downto 0)
  );
end entity;
```

```
architecture rtl of mealy_4s is
  -- Build an enumerated type for the state machine
  type state_type is (s0, s1, s2, s3);

  -- Register to hold the current state
  signal state : state_type;

begin
  process (clk, reset)
  begin
    if reset = '1' then
      state <= s0;
    elsif (rising_edge(clk)) then
      -- Determine the next state synchronously, based on
      -- the current state and the input
      case state is
        when s0=>
          if data_in = '1' then
            state <= s1;
          end if;
        when s1=>
          if data_in = '1' then
            state <= s2;
          end if;
        when s2=>
          if data_in = '1' then
            state <= s3;
          end if;
        when s3=>
          if data_in = '0' then
            state <= s1;
          end if;
      end case;
    end if;
  end process;
```

3.4 מימוש יציאה שאינה תלויה בכניסה מכונת – MOORE

בנוסף למכונה ניתן להגדית תהליך אסינכרוני שמגדיר את היציאות :

```
process (state)
begin
  case state is
    when s0=>
      data_out <= "00";
    when s1=>
      data_out <= "11";
    when s2=>
      data_out <= "10";
    when s3=>
      data_out <= "10";
  end case;
end process;

end rtl;
```

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>

3.5 כתיבה סינכרונית טהורה

בשיטה זו, כל הקוד הינו סינכרוני, כאשר יחד עם המעבר למצב הבא יש לשנות את היציאות מבחינת תזמונים זו המכונה היעילה ביותר, רק השהיה אחת מעלית השעון וכל היציאות סינכרוניות. יחד עם המעבר למצב הבא יש להגדיר את היציאות, ויש לממש זאת בכל המעברים לכל מצב

```
architecture arc_SyncStateMachine of SyncStateMachine is
    type state_type is (s0, s1, s2, s3); --enumerated type for state machine
    signal state : state_type; -- Register to hold the current state

begin
    process (clk, reset)
    begin
        if reset = '1' then
            state <= s0;
            data_out <= "00";
        elsif (rising_edge(clk)) then
            -- Determine the next state synchronously, based on
            -- the current state and the input
            data_out <= "00"; -- default
            case state is
                when s0=>
                    if data_in = '1' then
                        state <= s1;
                        data_out <= "01";
                    else
                        data_out <= "00";
                    end if;
                when s1=>
                    if data_in = '1' then
                        state <= s2;
                        data_out <= "10";
                    else
                        data_out <= "01";
                    end if;
                when s2=>
                    if data_in = '1' then
                        state <= s3;
                        data_out <= "11";
                    else
                        data_out <= "10";
                    end if;
                when s3=>
                    if data_in = '1' then
                        data_out <= "11";
                    else
                        state <= s1;
                        data_out <= "01";
                    end if;
            end case;

            end if;
        end process;
    end arc_SyncStateMachine ;
```

3.6 הגדרת שמות המצבים של מכונה

להגדרת מצבי המכונה משתמשים בשני שלבים : הגדרת המצבים והגדרת המשתנים :

```
-- Build an enumerated type for the state machine
type state_type is (s0, s1, s2, s3);

-- Register to hold the current state
signal state : state_type;
```

על מנת לחסוך את ההמרה מ- STATE למצב היציאה ניתן לקודד את מצב היציאה ישירות מה STATE :

קיימת שיטה הנקראת ONE_HOT שבה יש FLIP_FLOP לכל STATE בצורה הבאה :
בשיטה זו אין מצבי מעבר ביציאות כתוצאה משינוי בו זמנית של שני FLIP-FLOPS

```
-- declare the enumeration type
TYPE states IS (idle, state1, state2, state3, state4, state5);

-- assign the states
ATTRIBUTE enum_encoding : string;
ATTRIBUTE enum_encoding OF states: TYPE IS "000001 000010 000100
                                             001000 010000 100000";
```

ניתן לממש את שמות המצבים כך שהם יתאימו ישירות ליציאות ובכך לחסוך גם לוגיקה וגם את ההשהיה מעלית שעון ליציאה, אולם היא מסובכת יותר למימוש.
במידה וישנם שני מצבים בעלי אותן יציאות, יש להגדיר יציאת "דמי" שמבדילה ביניהן.

```
-- declare the enumeration type
TYPE states IS (idle, state1, state2, state3, state4, state5);

-- assign the states
ATTRIBUTE enum_encoding : string;
ATTRIBUTE enum_encoding OF states: TYPE IS "00 01 11 10";
```

ראה למשל :

<http://www.oocities.org/siliconvalley/screen/2257/vhdl/state/one-hot.html>

4 הירארכיה ב VHDL

עד היום חיברנו הירארכיה של מספר קבצים בצורה גרפית, ניתן לבצע הירארכיה גם בשפת VHDL, יש לפתוח קובץ VHDL בו יופיעו כל ההגדרות של ה ENTITY ואחריו מחברים אותן למערכת בעזרת חוטים – SIGNALS

הגדרת ENTITY

```
library ieee ;
use ieee.std_logic_1164.all ;
entity reg8 is --Register
    port ( resetN,clk : in std_logic          ;
          ena       : in std_logic          ;
          din       : in std_logic_vector(7 downto 0) ;
          dout      : out std_logic_vector(7 downto 0) ) ;
end reg8 ;
```

חיבור מספר ENTITIES בתוך ארכיטקטורה של VHDL

```
architecture arc_reg8x4 of reg8x4 is
    signal d0_out: std_logic_vector(7 downto 0) ;
    signal d1_out: std_logic_vector(7 downto 0) ;
    signal d2_out: std_logic_vector(7 downto 0) ;
    signal d3_out: std_logic_vector(7 downto 0) ;
begin
    reg0 : reg8 port map (resetN, clk, ena, din, d0_out);
    reg1 : reg8 port map (resetN, clk, ena, d0_out, d1_out);
    reg2 : reg8 port map (resetN, clk, ena,d1_out, d2_out);
    reg3 : reg8 port map (resetN, clk, ena, d2_out, d3_out);

    da <= d0_out;
```

ניתן גם לשלוח פרמטרים גנריים בזמן המיפוי בעזרת הפקודה generic map

5 המלצות לתכן נכון:

כאמור כל מכונה מכילה שני חלקים אסינכרוני ב RESET וסינכרוני ב-CLK

```
process (clk, reset)
begin
    if reset = '1' then
1. Initialization
        elsif (rising_edge(clk)) then

2. Default variables values
3. Math (case etc)

        end if;
4. Copy variables to output
    end process;
```

1. יש להקפיד שהאתחולים ב reset לא יהיו תלויים בכניסות – כך נוודא שתמיד המכונה מתחילה מאותו מצב, אם reset פעיל בנמוך שמו resetN
2. בפעולות תחת CLK מומלץ קודם לאתחל את כל היציאות ואז לשנות רק את מה שצריך ע"י CASE
3. לאחר המכונה יש להעתיק את המשתנים הפנימיים ליציאות – בצורה אסינכרונית

6 חומר עזר

קיימת ספרות ענפה בעברית ובאנגלית בנושא.

שני הספרים הבאים (בשפה העברית) עוסקים בהרחבה בנושא של כתיבה לסינתזה:

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

ספרים נוספים באנגלית שעוסקים בהרחבה בנושא הסינתזה הם הספרים הבאים:

1. Jayaram Bhasker, A VHDL Synthesis Primer, Star Galaxy Publishing
2. Pong P. CHU, RTL Hardware design using VHDL, Wiley-IEEE Press
3. Sundar Rajan, Essential VHDL RTL Synthesis Done Right, S & G Publishing
4. Andrew Rushton, VHDL for Logic Synthesis, Wiley
5. Kevin Skahill, VHDL for programmable logic, Addison Wesley

וכמובן ניתן לפנות לויקפדיה ולאתרים של ALTERA ו XILINX