The Andrew and Erna Viterbi Faculty of ELECTRICAL & COMPUTER ENGINEERING

Technion Israel Institute of Technology

המעבדה ל- VLSI Laboratory

מעבדה ב- VLSI ספרתי - 0450111

חוברת הדרכה
SystemVerilog

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

הערות נא לשלוח ל-goel@ee

כל הערה תתקבל בברכה!

עדכון אחרון - 19/09/2024 11:04

מסמך זה כתוב בלשון זכר ע"מ להקל על הכתיבה אך מתייחס לנשים ולגברים כאחד. עמכם הסליחה.

תוכן העניינים

הקדמה 2
מבוא קצר לשפת SystemVerilog 2
אינסטנסיאציה –instantiation (הצבה) 2
כתיבת testbench 2
משפט initial 2
דוגמא של מודול וה- testbench שלו 2

הקדמה

חוברת זו מהווה הסבר קצר על שפת **SystemVerilog.**

רקע

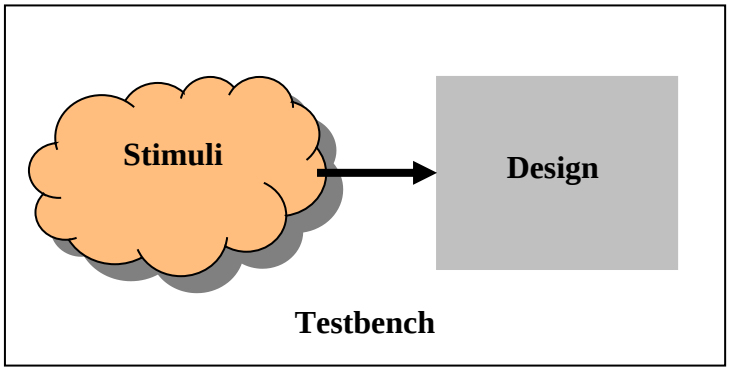
שפות תיאור החומרה (**HDL-Hardware Description Language**) למיניהן דוגמת **SystemVerilog** ו-**Verilog** פותחו על מנת לעזור למתכנתים לוגיים ליצור ולממש מעגלים במהירות.

כאמור שפת **SystemVerilog** היא שפה לתיאור חומרה שבעזרתה ניתן לתכנן, לממש ולבצע סימולציה של מערכות מורכבות. במגבלות מסוימות ניתן להפוך מימוש **SystemVerilog** למעגל חשמלי באמצעות כלי סינתזה. שפת **SystemVerilog** מאפשרת מימוש ברמת הפשטה גבוהה יותר לעומת **Verilog**. מבנים ופקודות כגון טיפוסי **data** חדשים **(logic, int)**, **enumerated types**, מערכים, משפטי **always** ייעודיים **(always_ff, always_comb)** מאפשרים מימוש בקוד יותר קומפקטי. בשפה גם ממשקים

מבוא קצר לשפת SystemVerilog

ישירים לשפות נוספות כגון **C**, **C++**, **systemC** ועוד.

בפרק זה נתאר רק את תכונות השפה שדרושות לביצוע הניסוי. המטרה העיקרית של העבודה עם שפת **HDL** היא מימוש התכנון וסביבת הסימולציה. הסברים נוספים ניתן למצוא ב-
https://vlsi.eelabs.technion.ac.il/wp-content/uploads/sites/18/2018/05/System-Verilog.pdf



איור מס' 1

בשפת **SystemVerilog** תכנון רכיב (מודול) ממומש באמצעות משפט **module** שניתן לתאר באופן הבא :

```
module name (  )
  interface // input and output definition
  variables //declare variables and type
  functional behaviour :
      // assign statements
      // always statements
      // instantiation of other module
endmodule
```

המודול מכיל את הגדרת הממשק (כניסות ויציאות), הגדרת משתנים ואת תיאור ההתנהגות הלוגית של הרכיב. ה- **testbench** (איור 1) הוא רכיב עזר ללא כניסות ויציאות המדמה את העולם החיצון של הרכיב הנבדק המהווה תת-רכיב שלו. לכן הוא מכיל הצבה של רכיב התכנון עצמו וקוד

(**stimuli**) שיוצר את אותות הכניסה עבור התכנון. הסברים מפורטים יובאו בהמשך. לעתים (גם

אינסטנסיאציה – instantiation (הצבה)

בניסוי זה) ליחידה **stimuli** בלבד קוראים **testbench**.

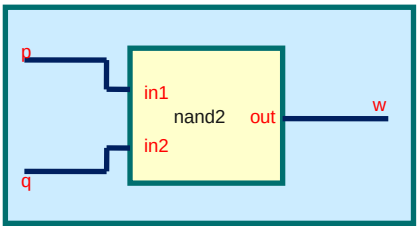
אינסטנסיאציה היא הצבת עותק של מודול קיים וחיבורו ללוגיקה הנוספת במודול. לדוגמא שער **and** יכול להיות בנוי משער **nand** ומשער not המחוברים ביניהם. להלן דוגמא של משפט אינסטנסיאציה:

```
nand nand_U1 (.out(w), .in1(p), .in2(q));
```

המשפט הנ"ל יוצר עותק של יחידה מסוג **nand** בשם **nand_U1** ומחבר את הכניסות והיציאות שלו (**out,in1,in2**) לקווים של היחידה המכילה אותו (**w,p,q**). התוספת **U1** הינה קיצור של **Unit-1** כיוון שעקרונית ניתן להציב יותר ממודול אחד מאותו סוג בתוך מודול אחר. לא חובה לציין את שמות הכניסות והיציאות של ה- **nand**, לדוגמא :

```
nand nand_U1 (w,p,q);
```

במקרה זה החיבור יתבצע לפי סדר רישום האותות, כלומר **w** יחובר לכניסה או ליציאה הראשונה בהגדרה של ה- **nand** וכן הלאה כפי שמופיע באיור הבא :



איור מס' 2

אם במודול שבו מבצעים את האינסטנציאציה מופיעים סיגנלים בשמות זהים לכניסות וליציאות של הרכיב (במקרה הזה **nand**) אז ניתן לבצע אינסטנציאציה מבלי לציין אף שם. לדוגמא:

```
nand U1 (.*) ;
```

ה "*" מציין שה- pins של ה- **nand** יחוברו לרשתות (חוטים) בעלי שם זהה.

כתיבת testbench

לאחר מימוש מודול צריך לבדוק אותו באמצעות סימולציות. לשם כך יש להכין סביבה פשוטה המכונה testbench. הסביבה היא למעשה מודול שמכיל הצבה של המעגל הנבדק בתוספת קוד שיוצר ערכים שמשתנים עם הזמן עבור הכניסות. מקובל לממש את הקוד באמצעות משפטי always ו/או initial לפי נוחיות המשתמש.

משפט initial

בניגוד למשפט **always** שמתבצע שוב ושוב, משפט **initial** מתבצע רק פעם אחת עם תחילת הסימולציה. משפט **initial** מתחיל בתחילת הסימולציה ויכול להמשיך לאורך כל הסימולציה ועד סופה. בעזרת משפט זה ניתן לקבוע ערכים התחלתיים למשתנים. לדוגמא, בעזרת משפט **always** מחליפים את הערך של אות השעון כל מחזור ובמשפט **initial** קובעים את ערכו ההתחלתי :

ניתן לבצע בלוקי initial במקביל ורצוי שלפחות אחד הבלוקים יכיל את המילה $finish אשר גורמת לסיום הסימולציה.

```
initial
begin
    clock = 0;
end
```

```
always
begin
  #5 clock =  ~ clock;
end
```

- מסמן השהייה של מספר יחידות זמן. במקרה זה הכלי ימתין **5ns** עד לביצוע ההשמה

~ - מסמן היפוך ערך הסיגנל

דוגמא של מודול וה- testbench שלו

```
module arbiter (clock, reset, req_0,req_1, gnt_0,gnt_1);
input clock, reset, req_0, req_1;
output logic gnt_0, gnt_1;
logic gnt_0, gnt_1;
always @ (posedge clock)
   if (reset) begin
   gnt_0 <= 0;
   gnt_1 <= 0;
  end else if (req_0) begin
    gnt_0 <= 1;
    gnt_1 <= 0;
  end else if (req_1) begin
    gnt_0 <= 0;
    gnt_1 <= 1;
  end
endmodule
```

משפט זה יוצר **flip flop** עבור כל משפט השמה.

להלן מימוש ה- testbench של הבורר :

```
module arbiter_test;
  logic clock, reset, req0,req1;
  logic gnt0,gnt1;
initial
begin
     reset = 0; clock = 0;
     req0 = 0; req1 = 0;
     #5  reset = 1;
     #15  reset = 0;
     @(posedge clk);
     req0 = 1;
     @(negedge clk);
     {req0,req1} = 2'b11;
     #10  $finish;
end

always
begin
    #5  clock =  ~ clock;
end

arbiter arb_U1 (.clock (clock),.reset (reset),.req_0 (req0),.req_1
(req1),.gnt_0 (gnt0),.gnt_1 (gnt1));
endmodule
```

קביעת ערכים התחלתיים

לאחר השהייה של **5ns**

בעלית השעון הבאה...

בירידת השעון הבאה...

יצירת שעון :

אינסטסיאציה של הבורר :

חשוב : שים לב לשתי השיטות לקדם את הזמן ב- **testbench**:

א. **#N** – מציין שההשמה הבאה מתבצעת אחרי **N** ננו שניות.

ב. `@(posedge clk)` -ו `@(negedge clk)` מציין ההשמה הבאה מתבצעת מיד אחרי ירידת או עלית השעון הבאה.