שאלות הכנה ב-SystemVerilog

שאלה 1

נתון קוד ה-SystemVerilog הבא:

```
module my_module(
    input logic [1:0] w,
    input logic a,
    output logic z1,
    output logic z2
);
    always_comb begin
        if (a == 1'b1) begin
            z1 = ~a;
        end
    end

    always_comb begin
        z2 = 1'b1;
        case (w)
           2'b00: z2 = 1'b0;
           2'b01: z2 = 1'b1;
           2'b10: z2 = 1'b0;
        endcase
    end

endmodule
```

כמה רכיבי זיכרון ייווצרו בתהליך הסינתזה של קוד זה?

א. 0
ב. 1
ג. 2
ד. 3
ה. 4
ו. לא ניתן לדעת

שאלה 2

נתונים שני קטעי הקוד הבאים:

```
always_ff @(posedge clk, posedge rst) begin
    if (rst == 1'b1) begin
        z = 1'b0;
    end
    else begin
        z = a;
    end
end
```

```
always_ff @(posedge clk) begin
    if (rst == 1'b1) begin
        z = 1'b0;
    end
    else begin
        z = a;
    end
end
```

בחר את התשובה הנכונה:

א. בקטע הקוד הראשון ה-reset הוא סינכרוני ובקטע הקוד השני ה-reset הוא סינכרוני.
ב. בקטע הקוד הראשון ה-reset הוא אסינכרוני ובקטע הקוד השני ה-reset הוא אסינכרוני.
ג. בקטע הקוד הראשון ה-reset הוא סינכרוני ובקטע הקוד השני ה-reset הוא אסינכרוני.
ד. בקטע הקוד הראשון ה-reset הוא אסינכרוני ובקטע הקוד השני ה-reset הוא סינכרוני.

שאלה 3 (מתוך מועד א' אביב תשע"ח)

להלן קוד SystemVerilog המתאר מכונת מצבים:

```
module simple_fsm (
    output logic z,
    input logic clk,
    input logic rst,
    input logic x
);

    // State declarations
    localparam S0 = 3'd0;
    localparam S1E = 3'd1;
    localparam S1O = 3'd2;
    localparam S2E = 3'd3;
    localparam S2O = 3'd4;

    logic [2:0] current, next;

    // Next state sampling
    always_ff @(posedge clk, posedge rst) begin
        if (rst) begin
            current <= S0;
        end
        else begin
            current <= next;
        end
    end

    // State transitions
    always_comb begin
        case (current)
            S0: if (x == 1'b1) begin
                    next = S1O;
                else
                    next = S1E;
                end
            S1E: if (x == 1'b1) begin
                    next = S2O;
                else
                    next = S2E;
                end
            S1O: if (x == 1'b1) begin
                    next = S2E;
                else
                    next = S2O;
                end
            S2E: next = S0;
            S2O: next = S0;
            default: // Should never reach this
                next = S0;
        endcase
    end

    // State machine outputs
    always_comb begin
```

```
        z = 1'b0;
        case (current)
            S0: z = 1'b0;
            S1E: z = 1'b0;
            S1O: z = 1'b0;
            S2E: z = 1'b1;
            S2O: z = 1'b0;
        endcase
    end

endmodule
```

ציירו את דיאגרמת המצבים של המכונה ורשמו האם המכונה הינה Moore או Mealy.

שאלה 4 (מתוך מועד ב' חורף תשע"ט)

נתון כי הסיגנל a הכיל את הערך 0x1234 לפני עליית השעון. עבור אילו מקטעי הקוד הבאים הסיגנל a יכיל בהכרח את הערך 0x3412 לאחר עליית השעון? בחר/י את התשובה הנכונה:

א.

```
always_ff @(posedge clk) begin
   a2 <= a;
   a[7:0] <= a[15:8];
   a[15:8] <= a2[7:0];
end
```

ב.

```
always_ff @(posedge clk) begin
   a[7:0] <= a[15:8];
   a[15:8] <= a[7:0];
end
```

ג.

```
always_ff @(posedge clk) begin
   a2 = a;
   a[7:0] = a[15:8];
   a[15:8] = a2[7:0];
end
```

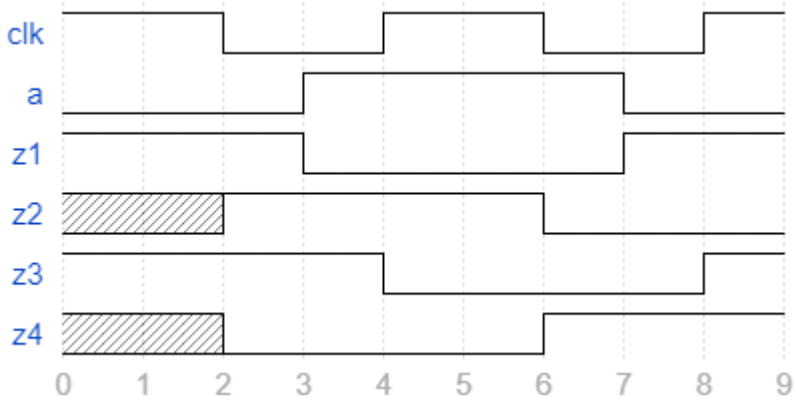
ד. תשובות א', ב' נכונות
ה. תשובות ב', ג' נכונות

שאלה 5 (מתוך מועד מילואים חורף תשע"ט)

נתון קוד ה-SystemVerilog הבא:

```
module test(
    input logic clk,
    input logic a,
    output logic z
);

    always_ff @(posedge ~clk) begin
        z <= ~a;
    end

endmodule
```

נתונות ארבע דיאגרמות אפשריות עבור הפלט z. שימו לב כי בכל הדיאגרמות הקלטים a ו-clk זהים, ולפני t=0 הסיגנל clk היה '0' והסיגנל a היה '1':



הערה: הסימון המפוספס בדיאגרמות z2, z4 מסמן את הערך 'x'.

בחר/י את הטענה הנכונה:

א. הפלט z ייראה כמו דיאגרמה z1 בסימולציה
ב. הפלט z ייראה כמו דיאגרמה z2 בסימולציה
ג. הפלט z ייראה כמו דיאגרמה z3 בסימולציה
ד. הפלט z ייראה כמו דיאגרמה z4 בסימולציה
ה. אף תשובה לא נכונה

שאלה 6 (מתוך מועד א' אביב תשע"ט)

נתון קוד ה-SystemVerilog הבא:

```
module test(
    input logic clk,
    input logic [3:0] a,
    output logic [3:0] z
);
    always_comb begin
        z = (a << 2) + 1;
    end
endmodule
```

איזה קטע קוד ייצור חומרה בעלת פונקציונליות זהה לחומרה שתיווצר כתוצאה מהקוד הנ"ל?

א.

```
module test(
    input logic clk,
    input logic [3:0] a,
    output logic [3:0] z
);
    always_ff @(posedge clk) begin
        z <= (a * 4) + 1;
    end
endmodule
```

ב.

```
module test(
    input logic [3:0] clk,
    input logic [3:0] a,
    output logic [3:0] z
);
    assign z = {a[1:0], {2{1'b1}}};
endmodule
```

ג.

```
module test(
    input logic clk,
    input logic [3:0] a,
    output logic [3:0] z
);
    assign z[3] = a[1];
    assign z[2] = a[0];
    assign z[1:0] = 2'b01;
endmodule
```

ד. תשובות א' ו-ב' נכונות.
ה. תשובות ב' ו-ג' נכונות.

שאלה 7 (מתוך מועד ב' אביב תשע"ט)

נתונים שני ה-Modules הבאים:

```
module mymodule (
    input logic clk,
    input logic rst,
    output logic done
);

    parameter N = 3;

    logic [31:0] w;

    always_ff @(posedge clk, posedge rst) begin
        if (rst == 1'b1) begin
            w <= 32'd0;
        end
        else begin
            if (w < N - 1) begin
                w <= w + 1;
            end
            else if (w == N - 1) begin
                w <= 32'd0;
            end
        end
    end

    always_comb begin
        if (w == N - 1) begin
            done = 1'b1;
        end
        else begin
            done = 1'b0;
        end
    end

endmodule
```

```
module mymodule2 (
    input logic clk,
    input logic rst,
    output logic out
);
    logic w1;
    logic w2;
    mymodule uut1 (clk, rst, w1);
    mymodule #(.N(2)) uut2 (clk, rst, w2);

    always_comb begin
        if ((w1 == 1'b1) && (w2 == 1'b1)) begin
            out = 1'b1;
        end
        else begin
            out = 1'b0;
        end
    end

endmodule
```

אל כניסת ה-clk של mymodule2 מחברים שעון ואל כניסת ה-rst אות reset. לאחר reset, כל כמה מחזורי שעון יקבל הסיגנל out את הערך 1'b1?

א. 2
ב. 3
ג. 4
ד. 5
ה. 6

שאלה 8 (מתוך מועד א' חורף תש"פ)

נתון קוד ה-SystemVerilog הבא:

```
module my_module (
    input logic  clk,
    input logic  rst,
    input logic  a,
    output logic out
);

    typedef enum {first_st, second_st} sm_type;

    sm_type current_state;
    sm_type next_state;
    logic w;

    always_ff @(posedge clk, posedge rst)
    begin
        if (rst == 1'b1) begin
            current_state <= first_st;
        end
        else begin
            current_state <= next_state;
        end
    end

    always_comb begin
        next_state = current_state;
        w = 1'b0;
        case (current_state)
            first_st:
                if (a == 1'b1) begin
                    next_state = second_st;
                    w = 1'b1;
                end
            second_st:
                if (a == 1'b0) begin
                    next_state = first_st;
                    w = 1'b1;
                end
        endcase
    end

    assign out = w & (~a);

endmodule
```

ניתן להניח כי הכניסה a מסונכרנת עם עליית השעון וכי כל הרכיבים אידיאליים (וכן כי הכניסות עומדות במשטר הזמנים). מתי הסיגנל out בעל ערך '1'?

א. בכל זמן שבו הסיגנל a בעל ערך '1'
ב. תמיד. הסיגנל out הוא הערך הקבוע '1'
ג. במחזור שעון בו ישנה עליה של הסיגנל a
ד. במחזור שעון בו ישנה ירידה של הסיגנל a
ה. במחזור שעון בו ישנו שינוי כלשהו של הסיגנל a (עליה או ירידה)

שאלה 9 (מתוך מועד ב' חורף תש"פ)

נתון קוד ה-SystemVerilog הבא, כאשר counter הוא module הממש מונה בעל 2 ביטים (סופר מ-0 עד 3):

```
module my_module (
    input logic clk,
    input logic rst,
    input logic a,
    output logic q
);
    typedef enum { S0_st, S1_st, S2_st } sm_type;

    sm_type current_state;
    sm_type next_state;

    always_ff @(posedge clk, posedge rst) begin
        if (rst == 1'b1) begin
            current_state <= S0_st;
        end
        else begin
            current_state <= next_state;
        end
    end

    always_comb begin
        case (current_state)
            S0_st: begin
                next_state = S1_st;
                q = 1'b0;
            end
            S1_st: begin
                next_state = S2_st;
                q = 1'b0;
            end
            S2_st: begin
                next_state = S0_st;
                q = a;
            end
            default: begin
                next_state = S0_st;
                q = 1'b0;
            end
        endcase
    end

endmodule
```

```
module my_module2 (
    input logic clk,
    input logic rst,
    output logic out
);
    logic [1:0] cnt;
    counter cnt_inst(.clk(clk), .rst(rst), .cnt(cnt));
    my_module inst (.clk(clk), .rst(rst), .a(cnt[0] & cnt[1]),
.q(out));

endmodule
```

```
module counter (
    input logic  clk,
    input logic  rst,
    output logic [1:0] cnt
);
    always_ff @(posedge clk, posedge rst)
    begin
        if (rst == 1'b1) begin
            cnt <= 2'b00;
        end
        else begin
            cnt <= cnt + 1;
        end
    end
endmodule
```

Duty cycle מוגדר עבור אות מחזורי בתור החלק היחסי מתוך זמן מחזור האות שבו האות בעל ערך '1' (למשל, duty cycle של אות שעון סטנדרטי הוא 1/2). מהו ה-duty cycle של הסיגנל out (היציאה של my_module2)?

א. 1/12
ב. 1/6
ג. 1/4
ד. 1/3
ה. 1/2

תשובה לשאלה 1

התשובה הנכונה: **ב. 1**

הסבר

במשפט ה-always הראשון יש תנאי על a. כל עוד a הוא 1, היציאה z1 מקבלת את ~a. ניתן לממש זאת בחומרה ע"י לוגיקה קומבינטורית בלבד (שער NOT). אך במידה ו-a הוא 0, אין השמה כלשהי ו-z1 צריך לשמור על ערכו הקודם. על מנת לממש זאת בחומרה, נוצר רכיב זיכרון (או יותר בפירוט, נוצר latch שמטרתו לשמור את הערך האחרון שנכתב ל-z1). זה תמיד המצב כאשר אין השמה דיפולטית במשפט always אסינכרוני. אם, למשל, היינו משנים את הקוד לקוד הבא:

```
always_comb begin
    if (a == 1'b1) begin
        z1 = ~a;
    end
    else begin
        z1 = a;
    end
end
```

בזכות ההשמה הדיפולטית (else) לא היה נוצר רכיב זיכרון (כי בכל מצב שהוא, לא משנה מה ערך a, ישנה לוגיקה קומבינטורית שנכתבת ל-z1 ואין צורך לשמור את ערכו הקודם).

במשפט ה-always השני יש משפט case שבודק את ערכו של w ובהתאם מציב ערך ביציאה z2. נשים לב שעבור המקרה w=1'b11 לא מציבים אף ערך ביציאה z2 בתוך ה-case, אך בזכות השמה דיפולטית שמתבצעת לפני משפט ה-case, גם במקרה זה מציבים ערך כלשהו ביציאה ולמעשה בכל מצב מתבצעת השמה כלשהי ליציאה. לכן, כאן לא ייווצר רכיב זיכרון. הערה: ניתן היה גם להשתמש ב-default:... בתוך משפט ה-case על מנת למנוע מצב של יצירת רכיב זיכרון בסינתזה. במידה ולא הייתה השמה דיפולטית מסוג כלשהו, היה נוצר רכיב זיכרון נוסף.

תשובה לשאלה 2

התשובה הנכונה: **ד. בקטע הקוד הראשון ה-reset הוא אסינכרוני ובקטע הקוד השני ה-reset הוא סינכרוני.**

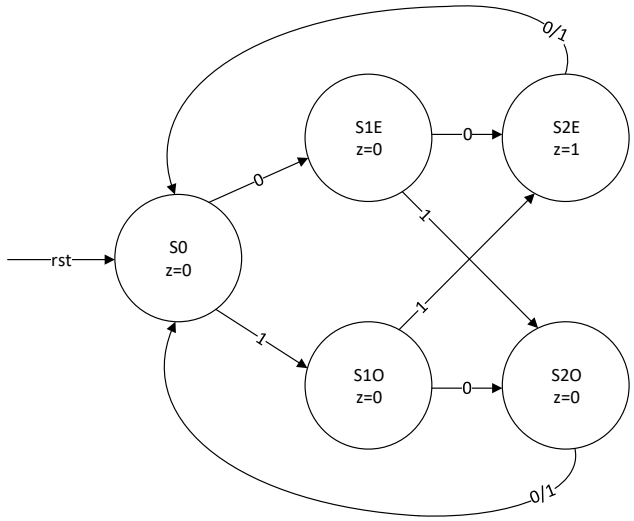
הסבר

Reset אסינכרוני הוא reset שמשפיע על ערך רגיסטר באופן מידי. על מנת לתאר זאת ע"י משפט always, על ה-reset להופיע ברשימת הרגישויות.

לעומת זאת, reset סינכרוני הוא reset שמשפיע על ערך רגיסטר רק בעליית השעון, בדומה לכל סיגנל אחר במעגל. לכן, במקרה זה ה-reset אינו מופיע ברשימת הרגישויות.

תשובה לשאלה 3

<u>תשובה: מכונת MOORE</u>



תשובה לשאלה 4

ה': תשובות ב', ג' נכונות.

הסבר:

נשים לב שבתשובה א' ההשמה מסוג non-blocking, לכן צד ימין של כל המשוואות יחושב שורה אחרי שורה, אך רק בסוף ה-procedural block ההשמות יתבצעו (במקביל). בחישוב הביטוי של ההשמה השלישית לא ניתן לדעת מה יש ב-a2 (לא נתון) ולכן התשובה אינה נכונה.

בתשובה ב' ההשמה גם היא מסוג non-blocking, אך כאן משתמשים בערך של a ישירות בכל ההשמות. מכיוון שהביטויים מחושבים לפני שההשמות מתבצעות, הביטים התחתונים יתחלפו עם העליונים ולהפך ללא דריסת המידע, כדרוש. לכן תשובה ב' נכונה.

בתשובה ג' ההשמה היא blocking. במקרה זה, עוברים לשורה הבאה רק לאחר שההשמה התבצעה. לאחר ביצוע השורה הראשונה a2 מקבל את הערך 0x1234. לאחר מכן, הביטים התחתונים של a נדרסים ע"י הביטים העליונים. לאחר ביצוע ההשמה הזו הערך של a יהיה 0x1212. לבסוף, הביטים העליונים של a נדרסים ע"י הביטים התחתונים המקוריים, שאוחסנו ב-a2 (שכבר מכיל 0x1234 בזכות סוג ההשמה). סה"כ a יקבל את הערך 0x3412 וגם תשובה זו נכונה.

תשובה לשאלה 5

ב': פלט z ייראה כמו דיאגרמה z2 בסימולציה

הסבר:

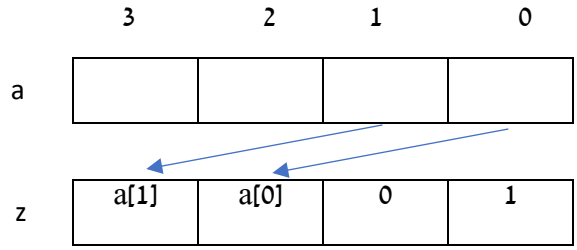
נשים לב כי ברשימת הרגישויות מופיע posedge ~clk, ולכן היציאה z תתעדכן בירידת השעון עם הערך של פעולת NOT על הכניסה a.

תשובה לשאלה 6

התשובה הנכונה: **ג**

הסבר

קטע הקוד הנתון מבצע shift left בשני מקומות ומוסיף 1, ואת התוצאה מוציא ל-output.
כלומר:



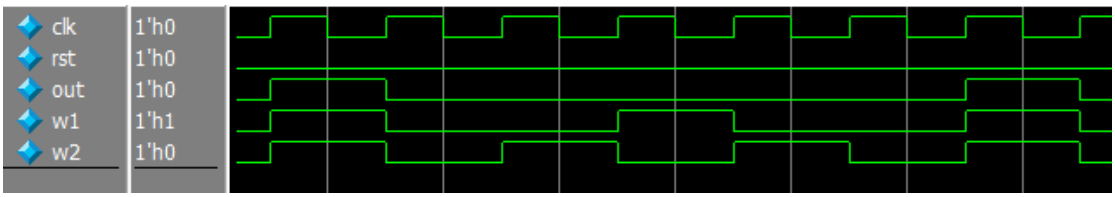
לכן תשובה ג' נכונה.

נשים לב כי אחד מה-input-ים הוא שעון, אבל מאחר והמימוש הפנימי של ה-module כולל רק לוגיקה צירופית, אין לו משמעות. קטע הקוד בתשובה א' מבצע לוגיקה דומה (כי shift left בשני מקומות שקול להכפלה ב-4), אך מאחר והלוגיקה נמצאת בתוך משפט always_ff, היא מחושבת רק בעליית שעון ולכן יוצרת גם Flip flops, בניגוד ללוגיקה המקורית. בתשובה ב', שני '1' משורשרים אל הביטים התחתונים של a במקום '01', כדרוש.

תשובה לשאלה 7

התשובה הנכונה היא תשובה ה'.

ה-module הראשון שנתון בשאלה, mymodule, מממש counter שכאשר הוא מסיים לספור עד N, מוציא ביציאה done את הערך הלוגי 1. ב-mymodule2 יוצרים שני instances של ה-counter, כאשר הראשון מביניהם סופר עד 3 והשני סופר עד 2 (כי בראשון משתמשים בערך ברירת-המחדל של N, המוגדר בתוך mymodule, ובשני מגדירים ערך שונה ל-N כאשר מבצעים module instantiation). היציאה out תכיל ערך לוגי 1 כאשר סיגנלי ה-done של שני ה-instances יהיו בעלי ערך לוגי 1. זה יקרה כל 6 מחזורי שעון (המכפלה המשותפת המינימלית של שני המספרים), כפי שניתן לראות בסימולציה המצורפת:



תשובה לשאלה 8

תשובה: ד'

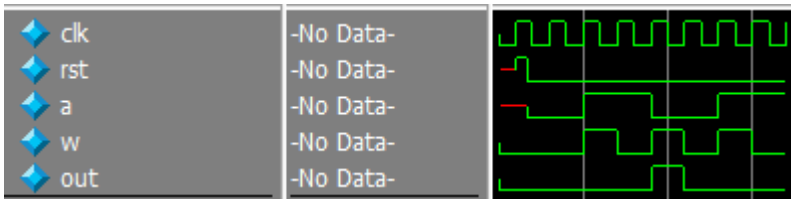
פתרון:

הקוד הנתון דומה לקוד של EdgeDetector שראינו בסדנה, עם השינויים הבאים:

```
module my_module (
    input logic  clk,
    input logic  rst,
    input logic  a,
    output logic out
);

    typedef enum {first_st, second_st} sm_type;

    sm_type current_state;
    sm_type next_state;
    logic w;

    always_ff @(posedge clk, posedge rst)
    begin
        if (rst == 1'b1) begin
            current_state <= first_st;
        end
        else begin
            current_state <= next_state;
        end
    end

    always_comb begin
        next_state = current_state;
        w = 1'b0;
        case (current_state)
            first_st:
                if (a == 1'b1) begin
                    next_state = second_st;
                    w = 1'b1;
                end
            second_st:
                if (a == 1'b0) begin
                    next_state = first_st;
                    w = 1'b1;
                end
        endcase
    end

    assign out = w & (~a);

endmodule
```

הסיגנל w מזהה עליה וירידה של הסיגנל a משום שכל פעם שעוברים בין שני המצבים, הוא מקבל את הערך '1'. לעומת זאת, הסיגנל out הוא תוצאת ה-AND של w ושל הערך ההפוך (NOT) של a. לכן, רק כאשר a יהיה בעל ערך '0' ויש מעבר בין המצבים, שזה שקול לירידה של הסיגנל a, הסיגנל out יקבל את הערך '1'.

להלן תוצאות סימולציה של הקוד:


clk
rst
a
w
out
-No Data-
-No Data-
-No Data-
-No Data-
-No Data-

תשובה לשאלה 9

התשובה הנכונה היא תשובה א'.

הקוד של my_module דומה לקוד של Clock Divider שראינו בסדנה, עם השינוי הבא:

```
module my_module (
    input logic clk,
    input logic rst,
    input logic a,
    output logic q
);
    typedef enum { S0_st, S1_st, S2_st } sm_type;

    sm_type current_state;
    sm_type next_state;

    always_ff @(posedge clk, posedge rst) begin
        if (rst == 1'b1) begin
            current_state <= S0_st;
        end
        else begin
            current_state <= next_state;
        end
    end

    always_comb begin
        case (current_state)
            S0_st: begin
                next_state = S1_st;
                q = 1'b0;
            end
            S1_st: begin
                next_state = S2_st;
                q = 1'b0;
            end
            S2_st: begin
                next_segment = S0_st;
                q = a;
            end
            default: begin
                next_state = S0_st;
                q = 1'b0;
            end
        endcase
    end

endmodule
```

א.

כלומר, כדי שהיציאה תהיה '1', ה-FSM צריך להיות במצב S2_st וגם a צריך להיות '1'. ה-FSM מגיע למצב S2_st כל שלושה מחזורי שעון ותוצאת ה-AND של ביטי ה-counter היא '1' כל ארבעה מחזורי שעון (כאשר ה-counter בעל ערך '11') ולכן היציאה תהיה '1' בכל מכפלה משותפת של 3 ו-4, כלומר כל 12 מחזורי שעון.

להלן תוצאות סימולציה של הקוד: