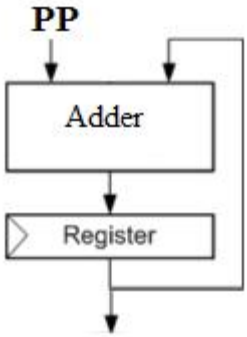
מכפל סדרתי



איור 1.

במשימה זאת יש לתכנן מכפל בינארי, סידרתי וסינכרוני . בכל מחזור מחשבים תוצאות ביניים (PP- partial products) הנצברת בצובר (accumulator איור 1) ומבצעים הזזה, שמאלה של הנכפל וימינה של הכופל, בביט 1. שיטה זאת דורשת ביצוע של n מחזורי צבירה/הזזה בעבור אופרנדים של- n ביט. פעולת כפל בעבור נכפל $A = 1011$ והכופל $B = 1011$, כל אחד בן 4 ביט מודגמת להלן.

תחילה: טוענים את הנכפל לאוגר בגודל של התוצאה הסופית (8 ביט בדוגמה זו), והכופל לאוגר בגודל הכופל: $RA = 00001011, RB = 1011$. מאתחלים את הצובר ל-'0'.

מחזור ראשון: הביט הימני של RB הוא $'1'$. מחשבים את ה-PP: $PP = 00001011$. צוברים את PP, והתוצאה היא 00001011. מזיזים שמאלה את RA ואז $RA = 00010110$, מזיזים את RB ימינה ואז $RB = 0101$.

מחזור שני: הביט הימני של RB הוא $'1'$. מחשבים את ה-PP: $PP = 00010110$. צוברים את PP, והתוצאה היא 00100001. מזיזים שמאלה את RA ואז $RA = 00101100$, מזיזים את RB ימינה ואז $RB = 0010$.

מחזור שלישי: הביט הימני של RB הוא $'0'$. מחשבים את ה-PP: $PP = 00000000$. צוברים את PP, והתוצאה היא 00100001. מזיזים שמאלה את RA ואז $RA = 01011000$, מזיזים את RB ימינה ואז $RB = 0001$.

מחזור רביעי: הוא המחזור האחרון. הביט הימני של RB הוא $'1'$. $PP = 01011000$. צוברים את PP. התוצאה הסופית היא 01111001.

כל הפעולות:

00001011_b (RA)
*
1011_b (RB)

00001011_b (PP=RA) 11_d
+
00010110_b (PP=RA *2) 22_d
+
00000000_b (PP=0)
+
01011000_b (PP=RA *8) 88_d

$01111001_b = 121_d$

<u>המשימה:</u> לממש מכפל **סינכרוני** סידרתי **קביל לסינתזה** לפי האלגוריתם המתואר מעלה, בעל שתי כניסות של 16 ביט (ללא סימן) כל אחת. פעולת המערכת מתחילה ע"י אות כניסה **start** העולה ל- '1' **למחזור שעון אחד**. המערכת מודיעה על סיום הפעולה ע"י אות מוצא **done** העולה ל-'1' **למחזור שעון אחד**. ממש את מערכת ע"י חלוקה לתת מבנים: יחידת בקרה ויחידות הביצוע. את יחידת הבקרה יש לממש כמכונת מצבים (FSM) ויחידות הביצוע כגון אוגרי הזזה לאופרנדים, מונה וצובר, ב-processes **נפרדים**.

שלבי העבודה:

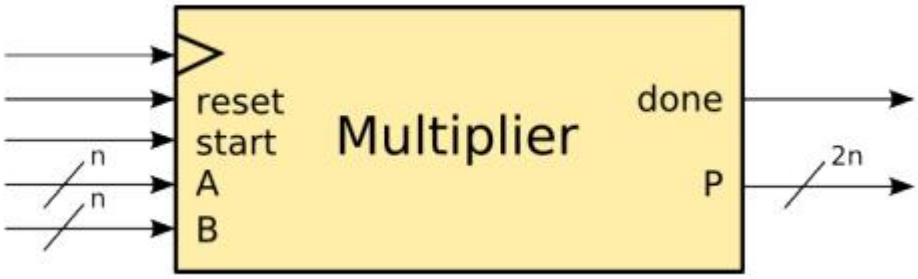
א) חלוקה לתת מבנים וסרטוט של המערכת כולה ברמת תת מבנים .

ב) תיאור דיאגרמאת מצבים של יחידת הבקרה (FSM)

ג) כתיבת קוד הכולל entity, architecture וה-processes המתארים את המערכת

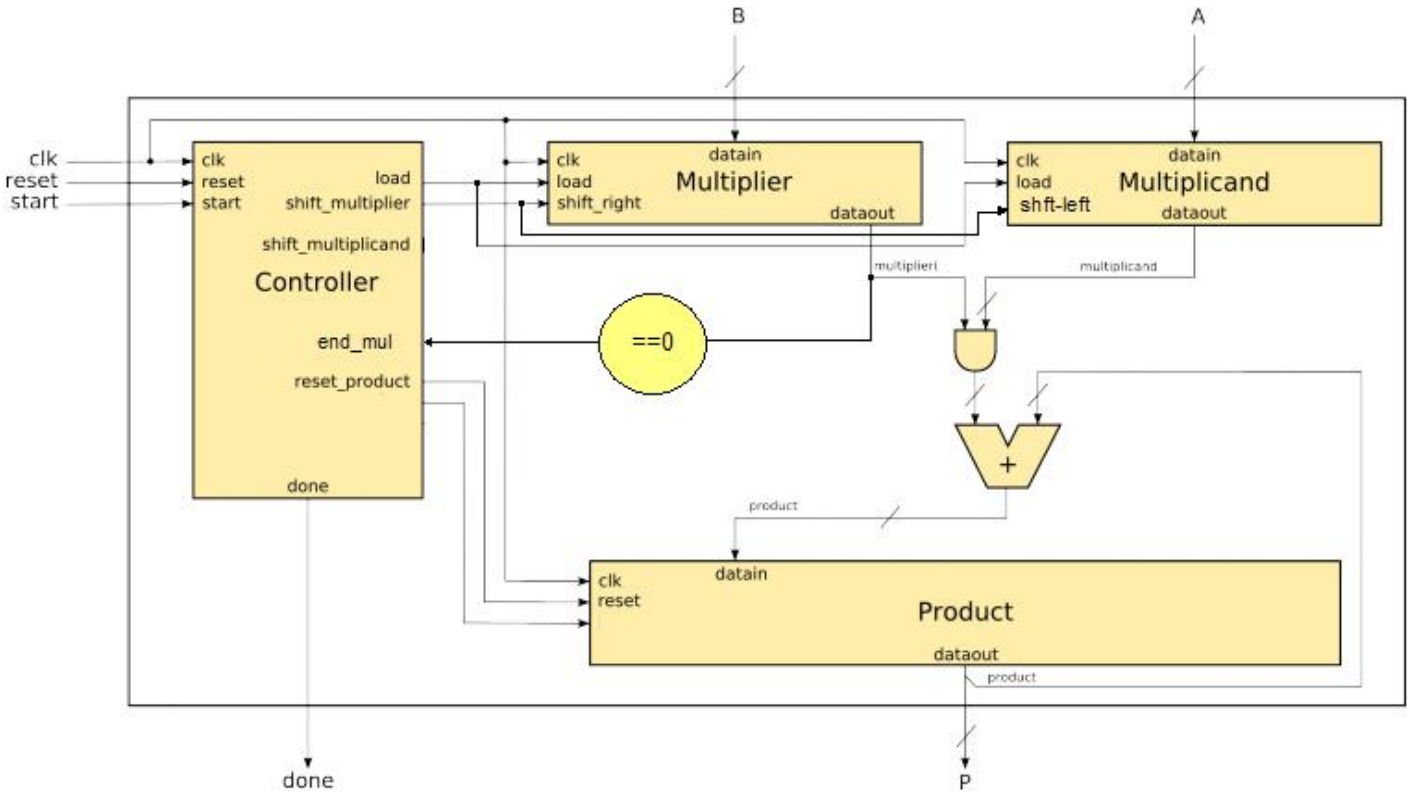
הפתרון

1. תחילה נגדיר מסגרת החיצונית (entity):



איור 2

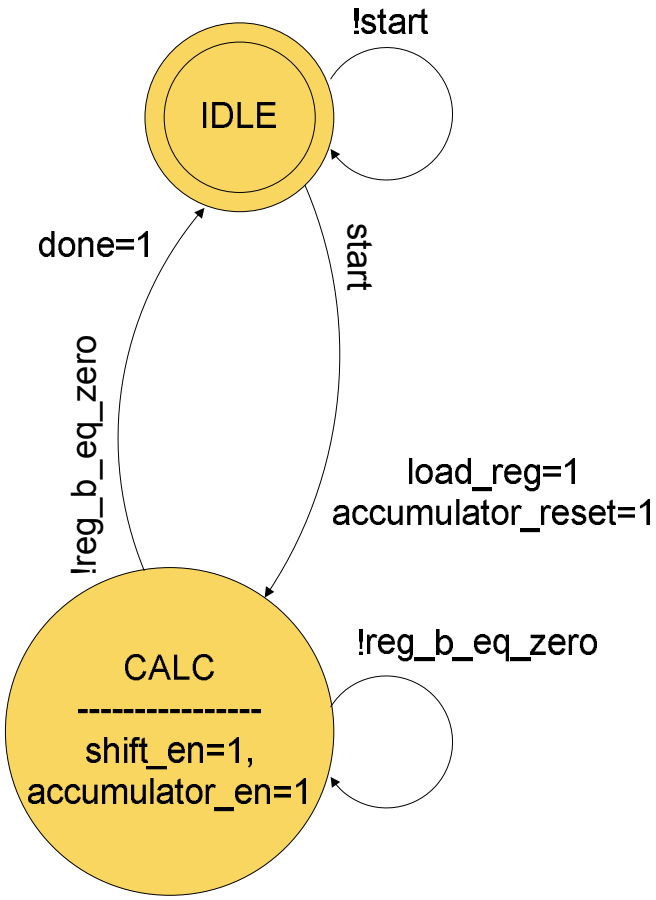
2. נפרט מבנים פנימיים של המערכת ונגדיר קשרים בניהם:



איור 3

3. נתאר תיפקוד של ה-Controller כמכונת Mealy

יש לזכור שבדיאגרמת מצבים של מודל Mealy בקשת המחברת בין המצבים בתחילתה מציינים תנאי מעבר, ובסופה פעולות אותן מבצעים במעבר בין המצבים. כאשר פעולה מסויימת אינה תלויה באותות כניסה למכונה והיא נעשת במצב כל שהוא, פעולה זאת מצויינת בתוך המצב הרלוונטי.



איור 4

4. עתה נתן לתאר מערכת בקוד VHDL:

להלן הגדרת ה-entity

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_arith.all;
4  use ieee.std_logic_unsigned.all;
5
6  entity seq_mul is
7     port (
8        clk   : in  std_logic;
9        rst   : in  std_logic;
10       a     : in  std_logic_vector(15 downto 0);
11       b     : in  std_logic_vector(15 downto 0);
12       start : in  std_logic;
13       res   : out std_logic_vector(31 downto 0);
14       done  : out std_logic);
15 end entity seq_mul;
```

הגדרת הארכיטקטורה והמשאבים הפנימיים.

```
17  architecture arc_seq_mul of seq_mul is
18     signal reg_a, acc, pp    : std_logic_vector(res'range);
19     signal reg_b             : std_logic_vector(b'range);
20     signal load, shift_en    : std_logic;
21     alias acc_sync_reset     : std_logic is load;
22     alias acc_en             : std_logic is shift_en;
23     signal reg_b_eq_zero     : boolean;
24     type   fsm_state is (IDLE, CALC);
25     signal curr_st, next_st : fsm_state;
26  begin
```

אוגרי הזזה לטעינה והזזה של הנכפל והכופל.

```
27     register_a : process (clk, rst) is
28     begin
29        if (rst = '0') then
30           reg_a <= (others => '0');
31        elsif rising_edge(clk) then
32           if (load = '1') then
33              reg_a <= x"0000" & a;
34           elsif (shift_en = '1') then
35              --shift left
36              reg_a <= reg_a(30 downto 0) & '0';
37           end if;
38        end if;
39     end process register_a;
40
41     register_b : process (clk, rst) is
42     begin
43        if (rst = '0') then
44           reg_b <= (others => '0');
45        elsif rising_edge(clk) then
46           if (load = '1') then
47              reg_b <= b;
48           elsif (shift_en = '1') then
49              --shift right
50              reg_b <= '0' &  reg_a(15 downto 1);
51           end if;
52        end if;
53     end process register_b;
```

חישוב של ה-PP והצובר.

```
55     pp <= reg_a when (reg_b(0) = '1') else (others => '0');
56
57     accumulator : process (clk, rst) is
58     begin
59        if (rst = '0') then
60           acc <= (others => '0');
61        elsif rising_edge(clk) then
62           if (acc_sync_reset = '1') then
63              acc <= (others => '0');
64           elsif (acc_en = '1') then
65              acc <= acc + pp;
66           end if;
67        end if;
68     end process accumulator;
69
70     res <= acc;
```

תיאור של מכונת מצבים.

```
72    --control state machine
73    curr_st_register : process (clk, rst) is
74    begin
75       if (rst = '0') then
76          curr_st <= IDLE;
77       elsif rising_edge(clk) then
78          curr_st <= next_st;
79       end if;
80    end process curr_st_register;
81
82    reg_b_eq_zero <= (reg_b = 0);
83
84    control_logic : process (curr_st, start, reg_b_eq_zero) is
85    begin
86       done     <= '0';
87       shift_en <= '0';
88       load     <= '0';
89       next_st  <= curr_st;
90       case curr_st is
91          when IDLE => if (start = '1') then
92                          load    <= '1';
93                          next_st <= CALC;
94                       end if;
95          when CALC => shift_en <= '1';
96                       if (reg_b_eq_zero) then
97                          done    <= '1';
98                          next_st <= IDLE;
99                       end if;
100         when others => null;
101      end case;
102   end process control_logic;
103 end architecture arc_seq_mul;
```