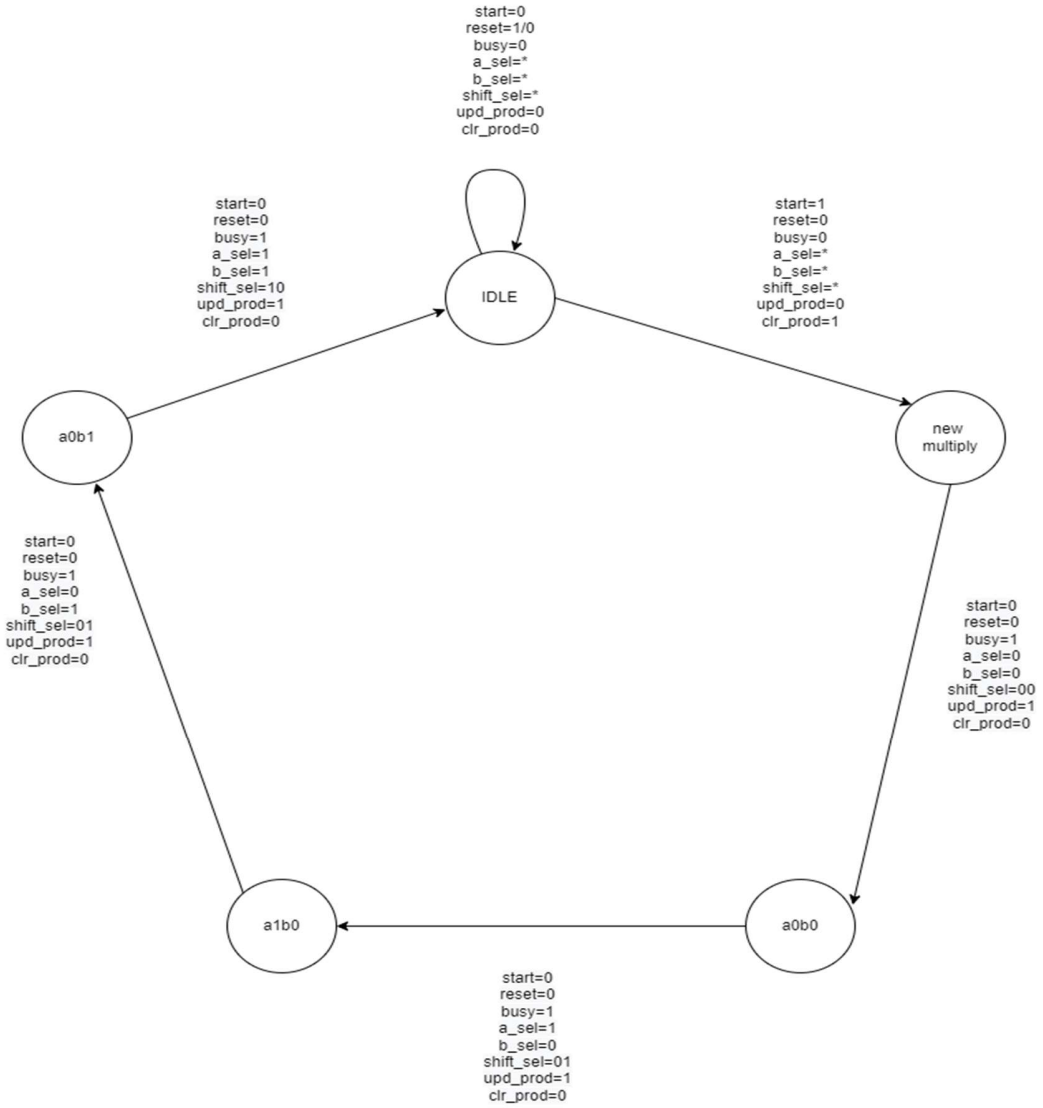
Simulation 2:

[redacted]	דניאל טרדלר
[redacted]	[redacted]

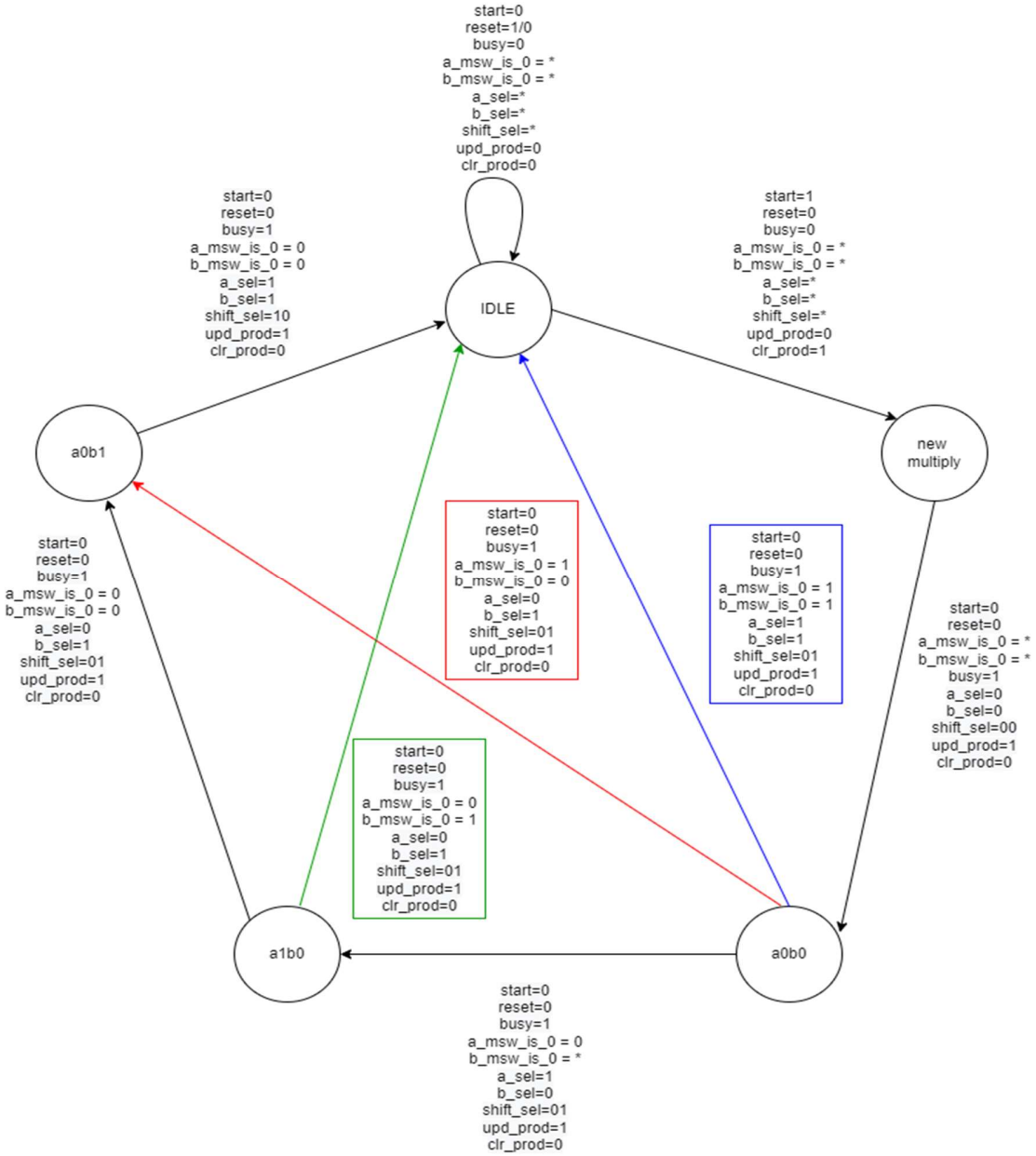
2.1. **דיאגרמת מצבים :**



<u>הערה</u>: הסימן " * " מייצג don’t care , כלומר אין תלות אם במשתנה יהיה 0/1.

מהסתכלות בדיאגרמת הגל על הגל שמייצג את busy, ניתן לראות כי פעולת הכפל מתרחשת תוך 4 מחזורי שעון.

2.2. **דיאגרמת מצבים :**



<u>הערה</u>: הסימן " * " מייצג don’t care , כלומר אין תלות אם במשתנה יהיה 0/1.

כעת לפי דיאגרמת המצבים הנ"ל , פעולת הכפל תתבצע תוך 2-4 מחזורי שעון .
עבור המקרה בו a_msw_is_0=1 וגם b_msw_is_0=1 פעולת הכפל תתבצע במהירות המרבית:
2 מחזורי שעון.

2.3. אלגוריתם

```
sum=0
result=0
a_bit=0
b_bit=0
for a_bit from 0 to N/2 (N/2 not include)
    for b_bit from 0 to N (N not include)
        result= a[a_bit*8+15 : a_bit*8] * b[b_bit*8+7 : b_bit*8]
        result.shift(16*a_bit + 8*b_bit)
        sum+=result
output sum
```

באלגוריתם שיצרנו התוכנה תרוץ לולאה בת N/2 איטרציות שכל איטרציה בה מציינת מספר בן 16 ביטים , ובנוסף תרוץ לולאה בת N איטרציות על מספר בן 8 ביטים , ותבצע פעולות אריתמטיות O(1) על מנת לקבל את תוצאת הכפל בין 2 המספרים.
לאחר מכן התבצע shift שגם עבורו סיבוכיות הזמן היא O(1) ,ושוב פעולה אריתמטית.
ולכן , סיבוכיות התוכנה: $O(N^2)$.

2.4 – סה"כ ביצענו 13 שורות קוד => סה"כ 13 מחזורי שעון.

ra (x1)	0x00000000
sp (x2)	0x7ffffff0
gp (x3)	0x10000000
tp (x4)	0x00000000
t0 (x5)	0x00ffffff
t1 (x6)	0x00000000
t2 (x7)	0x00000000
s0 (x8)	0x00000000
s1 (x9)	0x00000000
a0 (x10)	0x0000000a
a1 (x11)	0x0ba07529
a2 (x12)	0x0000000b
a3 (x13)	0x00000000
a4 (x14)	0x000000ad
a5 (x15)	0x00000000
s10 (x26)	0x00000000
s11 (x27)	0x00000000
t3 (x28)	0x0000000b
t4 (x29)	0x0000feed
t5 (x30)	0x00ac4629
t6 (x31)	0x0ba07529

Machine Code	Basic Code	Original Code
0x10000e17	auipc x28 65536	la t3, a
0x000e0e13	addi x28 x28 0	la t3, a
0x000e2e03	lw x28 0(x28)	lw t3, 0(t3)
0x10000e97	auipc x29 65536	la t4, b
0xff8e8e93	addi x29 x29 -8	la t4, b
0x000eae83	lw x29 0(x29)	lw t4, 0(t4)
0x00000fb3	add x31 x0 x0	add t6, x0, x0
0x0ff06293	ori x5 x0 255	ori t0, x0, 0xff
0x00829293	slli x5 x5 8	slli t0, t0, 8
0x0ff2e293	ori x5 x5 255	ori t0, t0, 0xff
0x00829293	slli x5 x5 8	slli t0, t0, 8
0x0ff2e293	ori x5 x5 255	ori t0, t0, 0xff
0x0ffe7713	andi x14 x28 255	andi a4,t3,0xff#First 8 bits
0x008e5e13	srli x28 x28 8	srli t3,t3,8 #move number to right
0x0ffe7613	andi x12 x28 255	andi a2,t3,0xff #Second 8 bits
0x00000f33	add x30 x0 x0	add t5,x0,x0 #helper-sum
0x00000fb3	add x31 x0 x0	add t6,x0,x0 #result-sum
0x03d60f33	mul x30 x12 x29	mul t5, a2, t4
0x005f7f33	and x30 x30 x5	and t5, t5, t0
0x01ef8fb3	add x31 x31 x30	add t6,t6,t5
0x00100513	addi x10 x0 1	finish: addi a0, x0, 1
0x000f8593	addi x11 x31 0	addi a1, t6, 0
0x00000073	ecall	ecall # print integer ecall
0x00a00513	addi x10 x0 10	addi a0, x0, 10
0x00000073	ecall	ecall # terminate ecall

195065129

2.5. ישנם 3 מצבים:

מצב 1 – מקרה בו a_msb_is_0=1 , יש צורך לבצע מכפלה אחת ולא שתי מכפלות , ולכן סיבוכיות הזמן מתקצרת בתהליך זה ל-9 מחזורי שעון.

מצב 2 – מקרה בו a_msb_is_0=0 , ו- b_msb_is_0=1 , במקרה זה נצטרך לבצע מכפלה אחת אך יותר בדיקות על מנת לאמת את סטטוס b_msb_is_0. סיבוכיות הזמן עבור תהליך זה יהיה 13 מחזורי שעון כמו סעיף 2.4.

מצב 3 - מקרה בו גם a_msb_is_0=0 וגם b_msb_is_0=0 , בגלל הבדיקות הרבות שהכנסנו לקוד העלנו את מחזורי השעון וכעת התהליך הכללי מתארך ל-20 מחזורי שעון.

```
####################
# Start of your code
    add s11,x0,t3
    andi a4,t3,0xff#First 8 bits
    srli t3,t3,8 #move number to right
    andi a2,t3,0xff #Second 8 bits
    beq a2,x0,COND1

    andi a5,t4,0xff#First 8 bits
    srli t4,t4,8 #move number to right
    andi s10,t4,0xff #Second 8 bits
    beq s10,x0,COND2


    add t5,x0,x0 #helper-sum
    add t6,x0,x0 #result-sum

# mul second half from first number with all other number
   mul       t5, a2, t4
   and       t5, t5, t0

   add t6,x0,t5 #add result to sum
   add t5,x0,x0 #zero helper

   slli t6,t6,8 #move second half 8 bits

   # mul first half from first number with all other number
   COND1:   mul      t5, a4, t4
            and      t5, t5, t0

    add t6,t6,t5

    j finish
    COND2: add t6,x0,x0 #result-sum

        # mul second half from first number with all other number
        mul      t5, a5, s11
        and      t5, t5, t0

   add t6,x0,t5 #add result to sum
# End of your code
####################
```

לסיכום , תהליך השינוי גרם להאטת התהליך במקרה הגרוע ביותר (מצב 3), ולכן שינוי זה אינו משתלם.

3.4. דיאגרמת הגלים(הכללית):

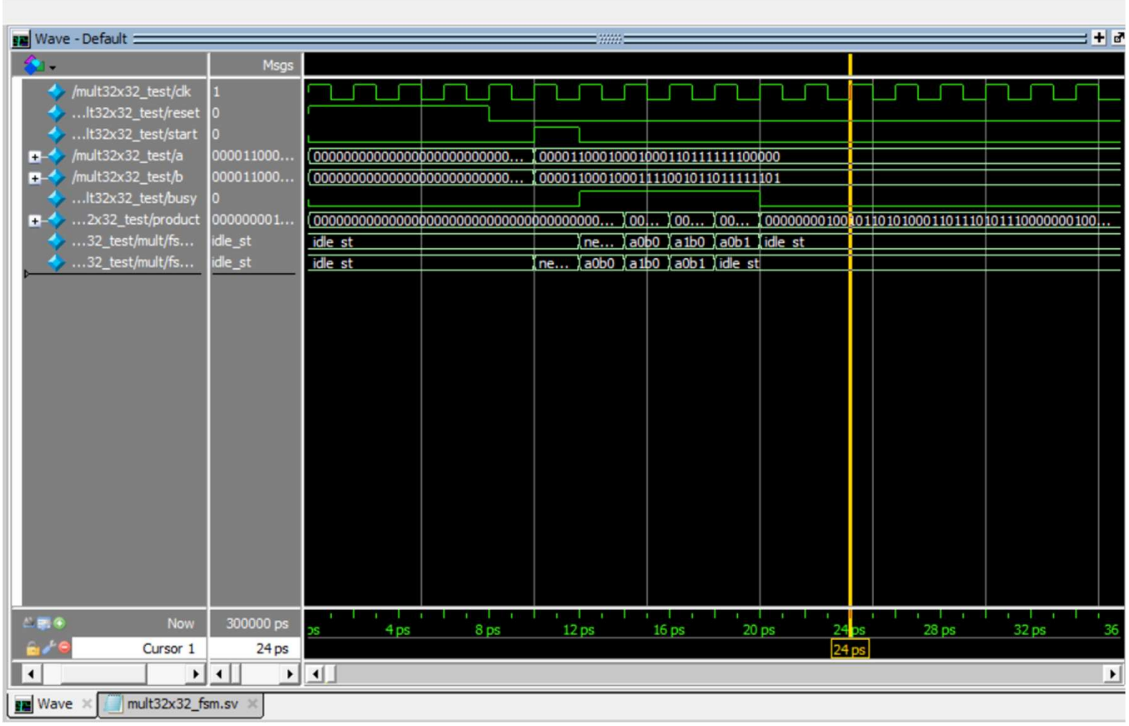
תעודת זהות במשתנה a = 205811680 , 00001100010001000110111111100000

תעודת זהות במשתנה b = 206018301 , 00001100010001111001011011111101

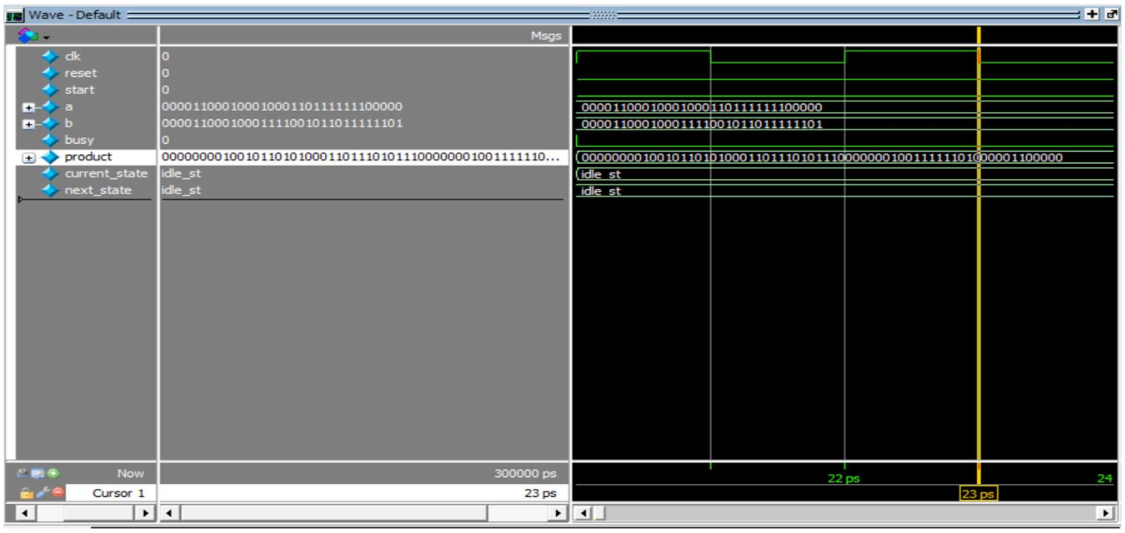
תוצאת הכפלת שני המספרים: 00000000100101101010001101110101110000000100111111010000011000000

ניתן לראות בדיאגרמה :

- Start עולה ולאחר מחזור שעון אחד הוא יורד ו- busy עולה ל-4 מחזורי שעון כמתוכנן.
- ניתן לראות שלאחר מכן לאחר כל מחזור שעון מתבצע שינוי מצב שגורר לשינוי התוצאה ב-product.
- בסוף התהליך המערכת חוזר למצב idle כמצופה.

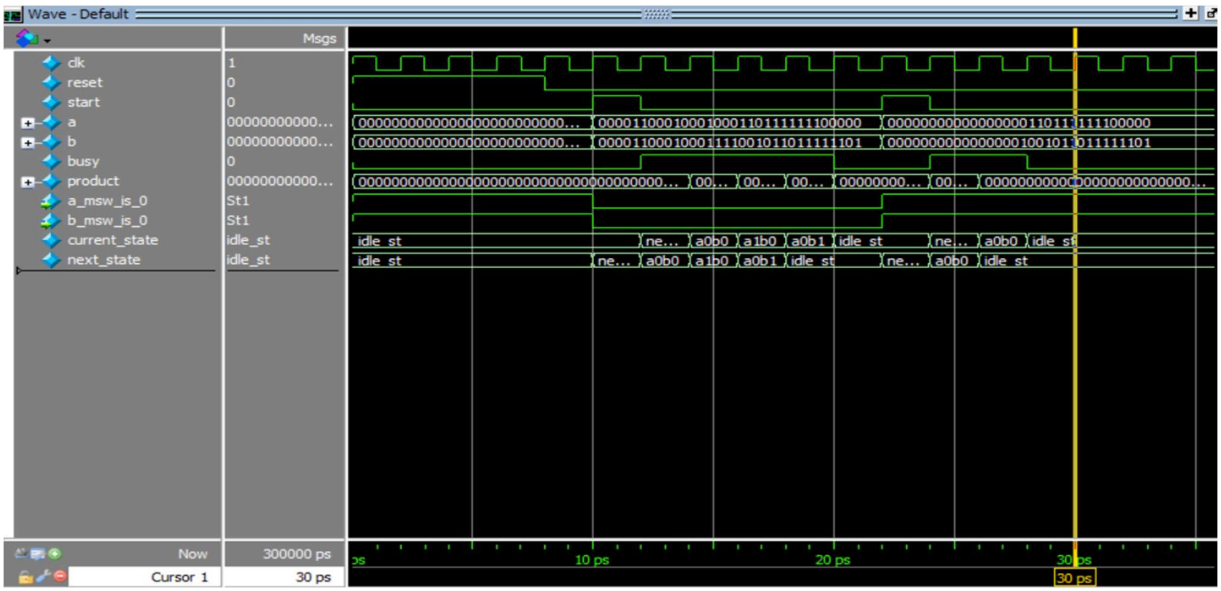


רואים את מכפלת המספרים בתמונה הבאה:

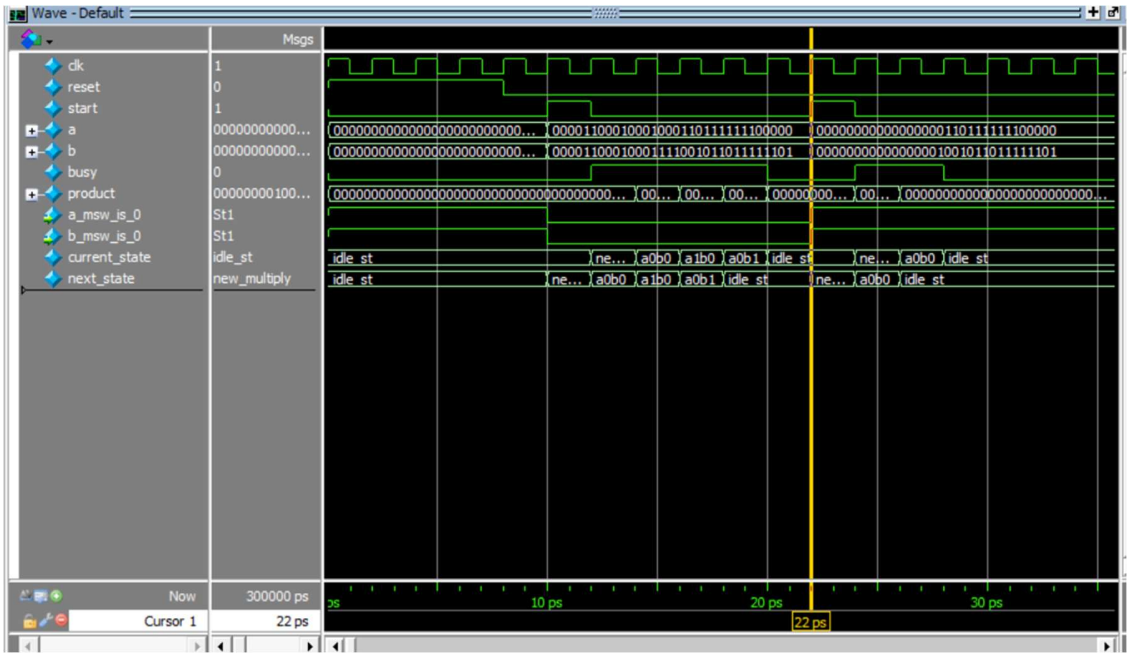


3.5. <u>דיאגרמת גלים עבור סעיף 2.2:</u>

תמונה בסוף הטסט:



תמונה באמצע הטסט לאחר חישוב תעודות הזהות המלאות:



- ניתן לראות כי החישוב הראשון ארך 4 מחזורי שעון כמצופה (Busy) , ולאחר מכן כאשר הכנסנו את תעודות הזהות בהן הביטים העליונים הם אפסים החישוב ארך רק 2 מחזורי שעון כמצופה לפי סעיף 2.2.

- באותה צורה כמו הסעיף הנ"ל ניתן לראות שאחרי ששינינו את הקלט למספרים שMSW שלהם הוא אפסים, אז מתבצע רק חישוב אחד וישר המערכת חוזרת לIDLE

- תעודת זהות עם אפסים בביטים העליונים במשתנה a = 205811680 :
00000000000000000110111111100000
תעודת זהות עם אפסים בביטים העליונים במשתנה b = 206018301 :
00000000000000001001011011111101
תוצאת הכפלת שני המספרים:
0000000000000000000000000000000100000111111011110100000110000 0

תמונה המראה שתוצאת המכפלה זהה לערך הנ"ל: