מערכות ספרתיות ומבנה המחשב

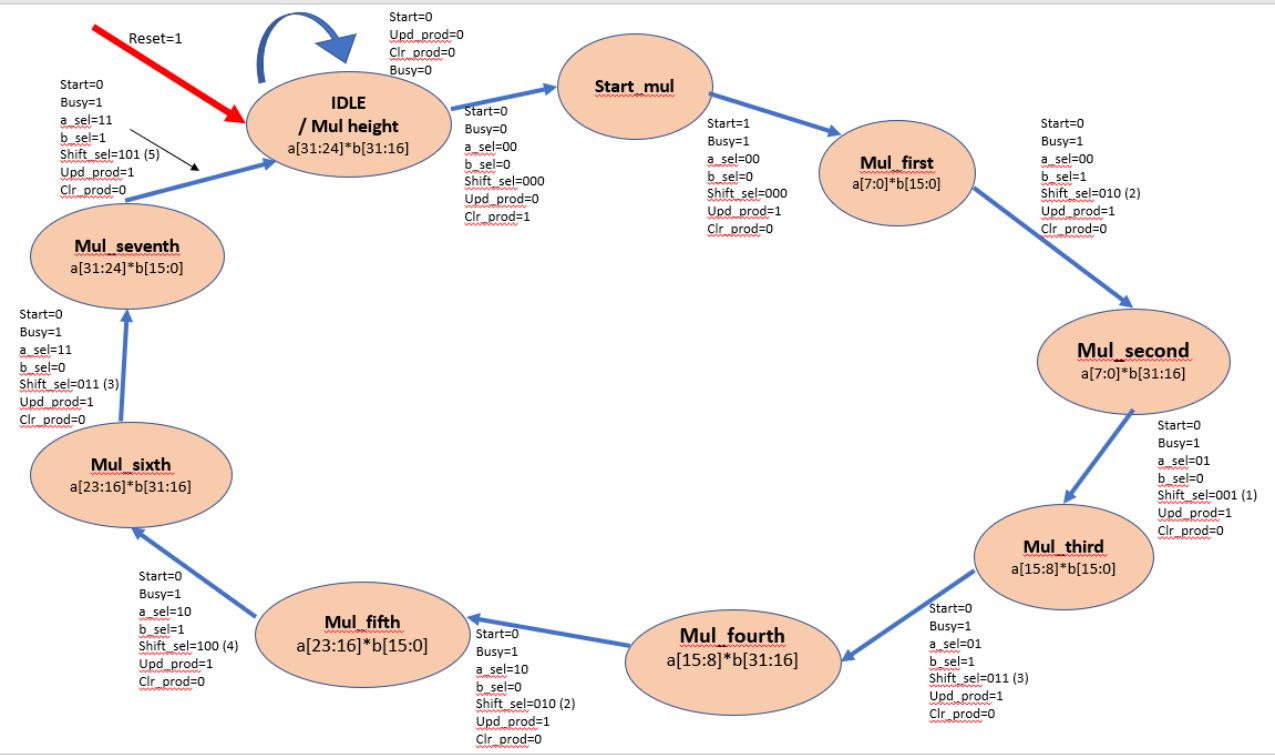
סומולציה 2

חלק יבש

אילת-השחר ברקוביץ	209947514
שלמה-דניאל אבנון	322677063

.2.1

מכונת מצבים של מכפל 32*32 באמצעות מכפל 8*16



Some explanation about the machine:

IDLE state:

Each time, we reset the machine, (which means each time reset=1) the machine comes back to this state.

It's also a waiting state for the machine until it gets the start value and start to run (start value=1).

clr_prod=1 and upd_prod=0 for the product register not to get any value.

a_sel=0, b_sel=0 and shift_sel=0 (by default)

busy=0 because the machine isn't busy, she's not in work yet.

We stay in this waiting state until start=1 and then we pass to the next state.

Start_mul:

When start=1, it means the multiplication is going to begin. We change for example the value of start so it's a mealy machine because there's a result of a direct entry in one of the state.

That state is a transition to prepare the machine to start working, we start performing the multiplication by changing the default values of the exits.

Of course that as we can see in the diagram the value of busy pass from 0 to 1 because we are working.

We have to multiplicate a number of 32 bits by a number of 32 bits and we can do just multipplications of 8 bits by 16 bits. So we will separate the number a to 4 vectors of 8 bits and the number b to two words of 16 bits. And we will multiply each part of a by each part of b (8 multiplications) and will shift the number by the relevant number of bits. At each state we will do one multiplication and add his shifted result to the sum of all the results pf the other multiplications.

(For example when we want to multiply the third vector of a by the second vector of b we want to do: a[23:16]*b[31:16] so we will send the signals: a_sel=10 (for the third vector), b_sel=1 (for the second vector), shift_sel=100 because we want to shift by 4 bits (2 because we are in the third vector (of 8 bits) of a and 2 because we are in the second vector (of 16 bits) of a).

In all the states of the multiplications, upd_prod=1 and clr_prod=0 because we don't want to clear the precedent product (we want to keep the precedent result) and add the new result to the last.

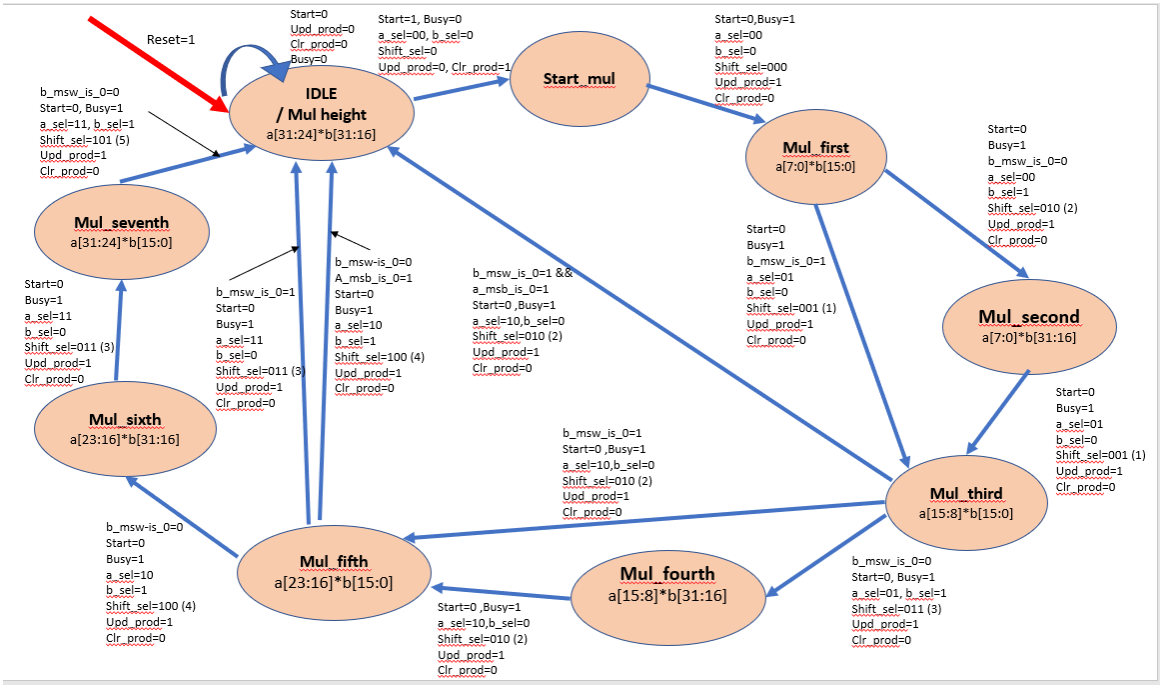
After that we pass from Mul_seventh to the Idle state and we wait again for the start to be 1 for the machine to receive new values and multiply them.

When getting to one of Idle State we have multiplied the 16 msb of b by the 8 msb of a (that is the last multiplication) and we shifted the result of the multiplication by 40 bits as the MSBs of both a and b as it should be. In fact we pass from the last mul to Idle state with the values a_sel=11, b_sel=1, shift_sel=5. The shifted result of the multiplication gets added in the 64-bit adder with the value getting out of the product register (which is the value of the previous addition between the seven products of the previous multiplications that have been made).

clr_prod=0 and upd_prod=1 so the product register will get out as the product value the final result of the multiplication.

אנחנו מצבעים סה"ך 8 פעולות כפל ושמן החזור הוא אכן 9 כי יש 9 מתבים במכונת מצבים (כולל ה-IDLE וה-Start_mul).

2.2.



אם מתקיים שה-msb 16 של b מתאפסות או שה-msb8 של a מתאפסות נוכל לעשות קיצורי דרך במכונת מצבים כדי לא לבצע פעולות כפל שתוצאתם תהיה 0 בוודאות.

נפריד למקרים:

9 מחזורי שעון	שום דבר לא מתאפס ולכן יש לבצע את 8 הפעולות כפל הנדרשוצ מה שייקח לנו 9 מחזורי שעון כפי שמוספר בסעיף הקודם.	a_msb_is_0=0 && b_smw_is_0=0
5 מחזורי שעון	ה-16 ביטים העליונים של b מאופסים ולכן לא צריך לכפול בהם בכלל כי המכפלה בהם תיתן 0 לכן אנחנו חוסכים 4 פעולות כפל (יש לבצע רק פעולות כפל בין כל הווקטורים באורך 8 של a עם ה-16 lsb של b.)	a_msb_is_0=0 && b_smw_is_0=1
7 מחזורי שעון	ה-8 ביטים העליונים של a מאופסים ולכן לא צריך לכפול בהם בכלל כי המכפלה בהם תיתן 0 לכן אנחנו חוסכים 2 פעולות כפל (יש לבצע רק פעולות כפל בין כל הווקטורים באורך 16 של b עם 3 ווקטורים באורך 8 של a (סה"ך 6 פעולות כפל)	a_msb_is_0=1 && b_smw_is_0=0

4 מחזורי שעון	עכשיו אנחנו חוסכים 4 פעולות כפל (פעולות שבהן משתתף ה-16 msb של b כמו מקודם) ובנוסף נחסוך עוד פעולה אחת (הכפל בין ה-16 lsb של b וה-msb16 של a כי מכפלתן תיתן 0. לכן נשאר לנו לעשות רק 3 פעולות כפל מה שייתן 4 מחזורי שעון (מוסיפים 1)	a_msb_is_0=1 && b_smw_is_0=1

לכן המכונה תעבוד הכי מהר אם a_msb_is_0==1 וגם b_msw_is_0=1.

2.3. מימוש פעולת כפל 16*16 באמצעות פעולת כפל 8*16:

בשאלה הזאת אנחנו צריכים להניח שיש לנו פקודה שיכולה לכפול מספר באורך 16 ביט במספר באורך 8 ביט בניהם ולהחזיר מספר באורך 24 ביט.

באמצעותה אנחנו יכולים לכפול שני מספרים באורך 8N כאשר N חיובי וזוגי.

נעשה את זה באמצעות האלגוריתם הבא: נניח שאנחנו רוצים לכפול ביחד שני מספרים באורך הזה: A ו B.

הכנת הכפל:

נפריד את A לווקטורים של 8 ביטים רציפים ו B לווקטורים של 16 ביטים רציפים.

A הוא באורך 8N והפרדנו אותו לווקטורים בני 8 ביטים לכן נקבל בסה"כ N ווקטורים. באותו אופן, עבור B נקבל 2/N ווקטורים. (לא עושים הווקטורים באופן אקראי זה אומר שעבור A הווקטור ה-i מתחיל בביט ה- i*8 ותסתיים בביט ה- 8*i+7 ועבור B, הווקטור ה-i מתחיל בביט ה-16*i ותסתיים בביט ה-16*i+15).

פעולת הכפל:

ניקח את ה-8 lsb של A וה- 16 lsb של B ונכפול אותם באמתעות הפקודת כפל 8*16 הנתונה ונאחסן את התוצאה בצד.

עכשיו ניקח שוב את ה-8 lsb של A והווקטור השני באורך 16 ביט של B ונכפול בניהם באמצעות אותה פעולה. נעשה shift left לתוצאה מ16 ביטים ונוסיף אותה לערך ששמרנו בצד, זה הערך החדש ששומרים בצד.

עכשיו ניקח שוב את ה-8 lsb של A והווקטור השלישי באורך 16 ביט של B ונכפול בניהם באמצעות אותה פעולה. נעשה shift left לתוצאה מ32 ביטים ונוסיף אותה לערך ששמרנו בצד, זה הערך החדש ששומרים בצד.

ונמשיך כך (כל פעם שמכפילים עם הווקטור ה-i של B עושים לתוצאה shift left מ- (i-1)*16 ביטים).

כשסיימנו לטפל ב-8 lsb של A, ניקח את הווקטור השני באורך 8 ביט של A וה-16 lsb של B, נכפול בניהם באמצעות אותה פעולה, ומוסיפים את זה לערך הזמור בצד.

אחרי ניקח עדיין את אותו ווקטור של A והווקטור באורך 16 השני של B ונכפול בניהם. רק שכשהקבל את התוצאה נעשה לה shift left של 16 ביטים (כי זה הווקטור השני של B + 8 כי זה הווקטור השני של A) לפני שמוסיפים אותה לתוצאה ששומרים בצד.

נמשיך באותו אופן (עם אותו היגיון של shift left עד שסיימנו לכפול את הווקטור השני של A עם כל הווקטורים ה-16 ביט של B.

ואז נעבור לווקטור השלישי של A שנפול עם כל הווקאורים באורך 16 של B. (הפעם יהיה לנו בכל כפל shift left של 16 ביטים (כי זה הווקטורים שלישי של A) ועוד ה shift המגיע מווקטור B.

נמשיך את התהליך באותו אופן עד שסיימנו לכפול את כל הווקטורים של A עם הווקטורים של B וביצענו את ה shift left המתאימים והוספנו הכל לערך השמור בצד שיהיה בסוף התוצאה הסופית של כפל A ב-B.

נתאר את האלגוריתם הזה בעזרת פסאדו קוד:

-הפונקציה 8By16(x,y) מקבלת מספר באורך 8 ביט ומספר באורך 16 ומחזירה מספר באורך 24 ביט שהוא תוצאת מהכפל בניהם.

- הפונקצייה shifleft(x, number) עושה למספר x פעולה של shiftleft מ- number ביטים.

```
8NByN (x, y)
{
        temp=0, total_result=0;

        for( i=0; i< N; i++)
        {
                for(j=0; j<N/2; j++)
                {
                        temp= 8By16( x[8i+7:8i], y[16j+15:16i];
                        sum+=shiftleft(temp, 8i+16j);
                }
        }
        return sum;
}

// x[8i+7:8i] is the (i+1)th vector of 8 bit from x.
```

לפי חישובי סיבוכיות קלים בפסאודו קוד, רואים שסיבוכיות הזמן של האלגוריתם היא N*N/2 ולכן היא: O(N^2).

2.4.

המימוש עצמו בקובץ mul16x16.s

s11 (x27)	0x00000000
t3 (x28)	0x00000bad
t4 (x29)	0x0000feed
t5 (x30)	0x00000000
t6 (x31)	0x0ba07529

s2 (x18)	0x000000fe
s3 (x19)	0x000acf29
s4 (x20)	0x0b95a600

t2 (x7)	0x00000000
s0 (x8)	0x00000000
s1 (x9)	0x000000ed
a0	0x0000000a

אנחנו מפרידים את המספר b לשני חלקים באורך 8 ביט.

כופלים את ה-8 lsb של b עם המספר a (באורך 16 ביט) בעזרת פעולת הכפל 8*16 שנתונה לנו.

כופלים את ה-8 msb של b עם המספר a באמצעות אותה פעולה ועושים shift לתוצאה מ-8 ביט שמאלה (כי זה ה-msb של התוצאה).

לבסוף מחברים את שניהם ושמים את התוצאה הסופית ב-t6 כנדרש.

בסימולציה הזאת אנחנו מכפילים שני מספרים:

00000bad

0000feed

זאת אומרת: 2989 כפול 65261. אנחנו אמורים לקבל 195065129 ששווה לערך הרשום ב-t6 בייצוג bit-16.

.2.5

In this question, like the question 2.2 we'd like to improve the code by skipping steps that are unnecessary because it'll come out as 0 anyway. We will see 3 special cases that allow us to skip steps:

- **If 8 msb of a are 0:**
  In 2.4, we separate b into two parts: 8Msbs and 8Lsb. And then we multiply the Msbs of b with a and the Lsbs of b with a.
  The multiplication with the Msbs of a in those 2 multiplications will give 0 because the 9 Msbs of a are 0.
  So we'll change the code by separating a instead of b into 8 Lsbs and 8 Msbs. And we multiply b only with the 8 Lsbs of a because the multiplication with the Msbs of a will give 0.

- **If 8 msb of b are 0:**
  In 2.4, we separate b into two parts: 8Msbs and 8Lsb. And then we multiply the Msbs of b with a and the Lsbs of b with a.
  We skip here the multiplication of a with the Msbs of b because it'll give 0 anyway and we don't need also to sum up the two products because there's only one.

- **If both are 0:**
  We have to multiply 16bits of a or b because our multiplier multiplies 16bitsx8bits so, like in the previous case, we'll multiply a with the 16 Lsbs of b and we'll skip the second multiplication with the Msbs and also the sum between the two products.

- **If none are 0:**
  We'll use the same algorithm as in 2.

```
main:   # Load data from memory
        la      t3, a
        lw      t3, 0(t3)
        la      t4, b
        lw      t4, 0(t4)

        # t6 will contain the result
        add         t6, x0, x0

        # Mask for 16x8=24 multiply
        ori         t0, x0, 0xff
        slli  t0, t0, 8
        ori         t0, t0, 0xff
        slli  t0, t0, 8
        ori         t0, t0, 0xff
```

```
 # FROM HERE THE CODE IS DIFFERENT

 srli s2, t4, 8     #save in s2 the 8 msb of b
 srli s6, t3, 8     #save in s4 the 8 msb of a

 beq     s2,x0,BisZero  #If msbs of b are 0 then a doesn't matter.
 beq     s6,x0,AisZero  #If msvs of a are 0 then it's another code.


#Here the Msbs are both different from 0.
 slli s1, t4, 24
 srli s1, s1, 24    #save in s1 the 8 lsb of b

 mul s3, s1, t3     #multiplication between a and 8 lsb of b
 and s3, s3, t0

 mul s4, s2, t3    #multiplication between a and 8 msb of b and
 and s4, s4, t0
 slli s4, s4, 8    #shift 8 bits the MSbs

add t6, s4,s3
 j finish


BisZero:
     slli s1, t4, 24
     srli s1, s1, 24    #save in s1 the 8 lsb of b

     mul s3, s1, t3     #multiplication between a and 8 lsb of b
     and s3, s3, t0

     add t6, x0, s3
     j finish


 AisZero:
     slli s5, t3, 24
     srli s5, s5, 24    #save in s5 the 8 lsb of a

     mul s7, s5, t4     #multiplication between b and 8 lsb of a
     and s7, s7, t0

    add t6, x0, s7
     j finish


finish: addi    a0, x0, 1
   addi    a1, t6, 0
   ecall # print integer ecall
   addi    a0, x0, 10
   ecall # terminate ecall
```

In code 2.4:

- load the 8 Msbs of b and the 9 Lsbs of b = 1+2= 3 commands.
- Multiply between a and the LSbs of b = 2 commands

- Multiply and shift the product btw a and the Msbs of b = 3 commands
- Put the result in t6 = 1 command
  - ⇨ 9 commands = 9 time cycles.

In code 2.5:

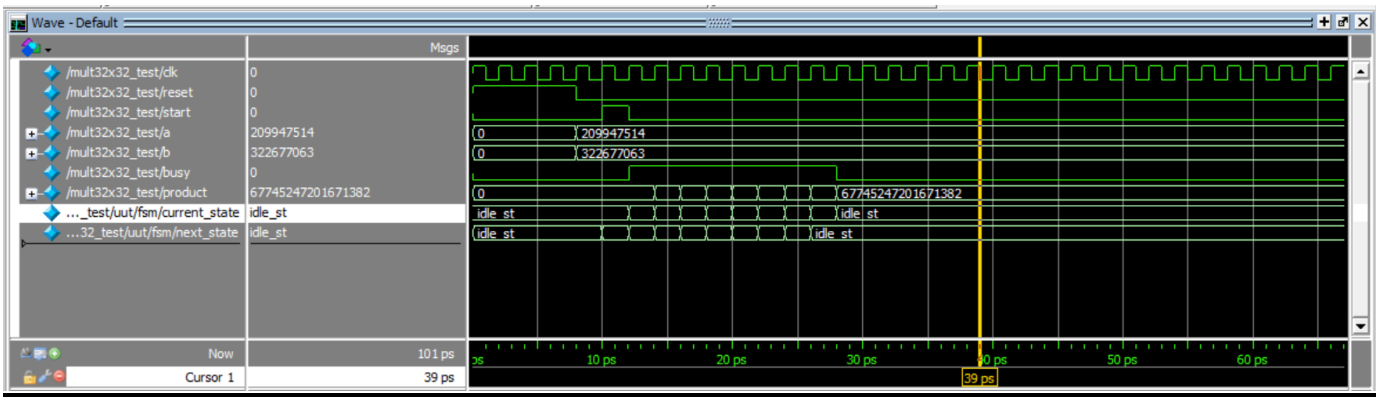
- We load Msbs of a and b to check if they are 0 = 2 commands.
- We check if one of them is 0 = 2 commands.
  - If they're both different from 0:
    then we have to return to the code of 2.4 except the Msb of b are already loaded so 8 commands and the command jump to skip the other cases = 1 command so in total
  - ⇨ 9 commands+4commands=13 commands.
  - If the Msbs of a is 0:
    Then we jump to AisZero, we load the Lsbs of a (2 commands), we multiply b with the Lsbs of a (2 commands), we put the result in t6 and then we jump to finish to skip the other cases:
  - ⇨ 6 commands+4 commands=10 commands.
  - If the Msbs of b is 0 or both are 0:
    Then we jump to BisZero, we load the Lsbs of b (2 commands), we multiply a with the Lsbs of b (2 commands), we put the result in t6 and then we jump to finish to skip the other cases:
  - ⇨ 6 commands+4 commands=10 commands.

In total:

- We added 4 time cycles if they're not 0.
- We added 1 cycle time is one or both are 0.

Which means, in all cases we increased the cycle time and the code is way longer and difficult to understand. The change is not worth it.

3.4



הסבר:

-Reset אפשר לראות שהוא מתאפס אחרי 4 מחזורי שעון כמבוקש

-Start מחזור אחד אחרי שRESET אופס הוא עולה למשך מחזור אחד

-a תז של איילת

-b תז של שלמה

-Busy מחזור אחד אחרי שstart עולה הוא עולה, כי המכונה התחילה לעבוד

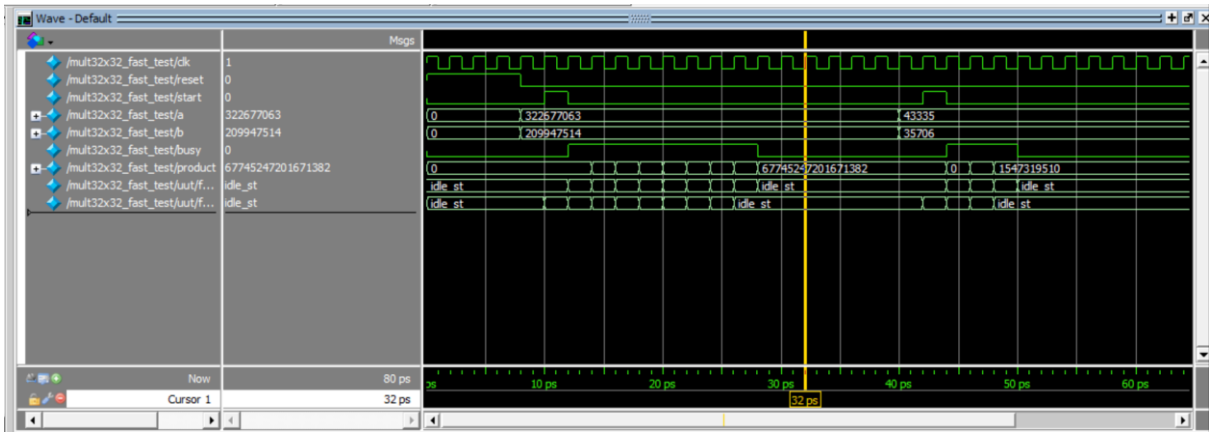
-Product תוצאת המכפלה, אפשר לראות שהוא משתנה כל מחזור שעון, החל מתחילת פעולת המכונה, כי הוא שומר את התוצאות הזמניות של הכפלים החלקיים

-Current state אנו רואים שהוא משתנה בעליית שעון, החל ממחזור שעון אחד אחרי שstart עולה, לפני ואחרי פעולת המכונה הוא על מצב ניטרלי

-Next state מיד כש start עולה הוא מתעדכן להיות המצב הרצוי הראשון, ואחר כך אנו מעדכנים אותו לפי המצב של current state, ולכן הוא יתעדכן גם בכל עליית שעון, לפני פעולת המכונה הוא על מצב ניטרלי, ומחזור אחד לפני הסוף הוא חוזר למצב זה.

המצבים עוברים מאחד לשני בסדר הבא start_st-mul_first-mul_second-mul_third...

3.7



הסבר:

-Reset אפשר לראות שהוא מתאפס אחרי 4 מחזורי שעון כמבוקש

-Start מחזור אחד אחרי שRESET אופס הוא עולה למשך מחזור אחד

-a תז של איילת 11001000001110001011011110 10

-b תז של שלמה 10011001110111010100101000111 1

-Busy מחזור אחד אחרי שstart עולה הוא עולה, כי המכונה התחילה לעבוד

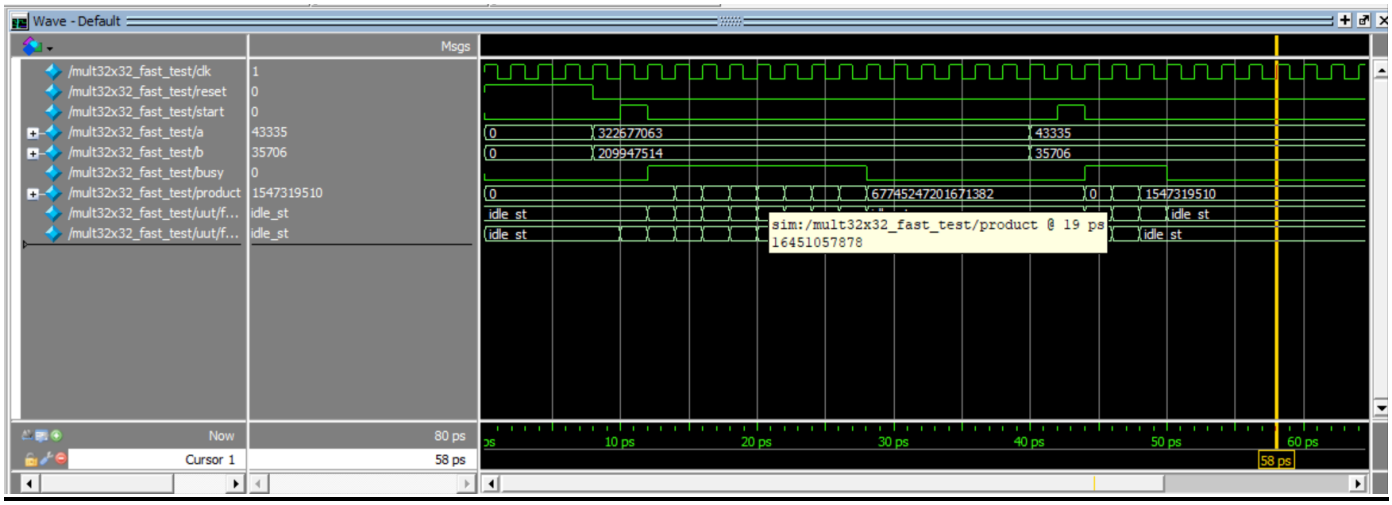
-Product תוצאת המכפלה, אפשר לראות שהוא משתנה כל מחזור שעון, החל מתחילת פעולת המכונה, כי הוא שומר את התוצאות הזמניות של הכפלים החלקיים

-Current state אנו רואים שהוא משתנה בעליית שעון, החל ממחזור שעון אחד אחרי שstart עולה, לפני ואחרי פעולת המכונה הוא על מצב ניטרלי

-Next state מיד כש start עולה הוא מתעדכן להיות המצב הרצוי הראשון, ואחר כך אנו מעדכנים אותו לפי המצב של current state, ולכן הוא יתעדכן גם בכל עליית שעון, לפני פעולת המכונה הוא על מצב ניטרלי, ומחזור אחד לפני הסוף הוא חוזר למצב זה.

בפעולה הזאת אין הבדל עם המכפל הלא מקוצר, כי אין אף בית שכולו אפסים.

<u>פעולה עם שתי בתים מאופסים</u>



<u>הסבר:</u>

-Reset אפשר לראות שהוא מתאפס אחרי 4 מחזורי שעון כמבוקש

-Start מחזור אחד אחרי שRESET אופס הוא עולה למשך מחזור אחד

-a תז של איילת 11001000001110001011011110 10

-b תז של שלמה 10011001110111010100101000111 1

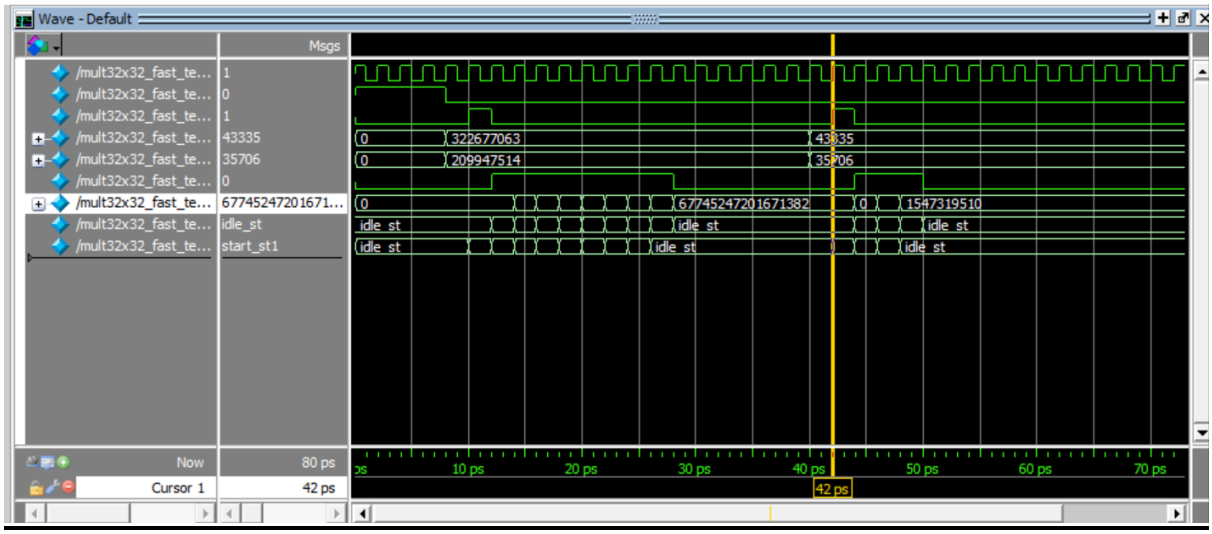
-Busy מחזור אחד אחרי שstart עולה הוא עולה, כי המכונה התחילה לעבוד

-Product תוצאת המכפלה, אפשר לראות שהוא משתנה כל מחזור שעון, החל מתחילת פעולת המכונה, כי הוא שומר את התוצאות הזמניות של הכפלים החלקיים

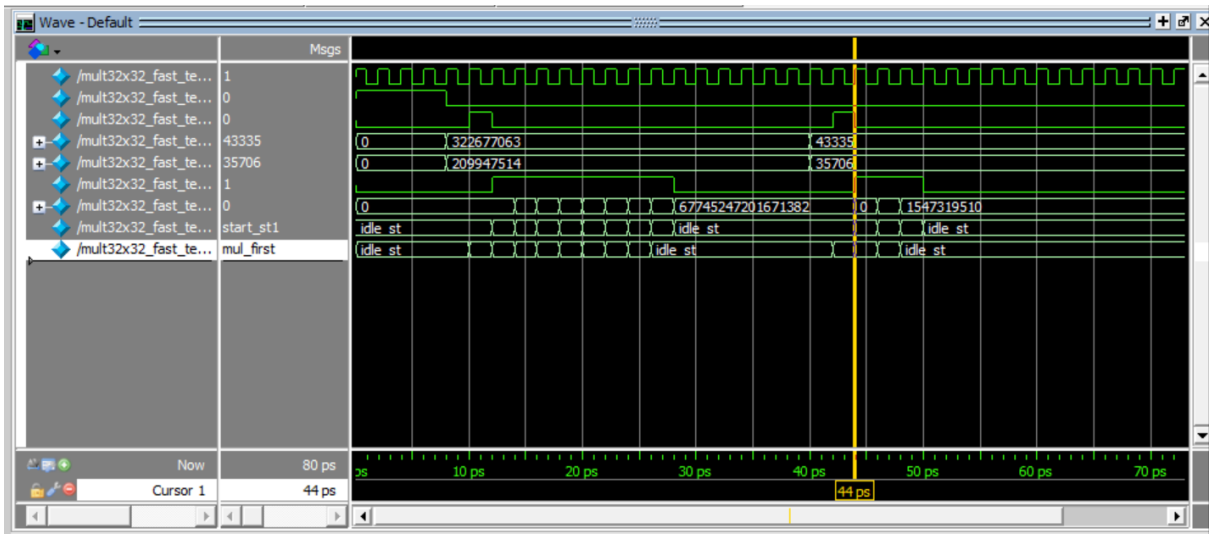
-Current state אנו רואים שהוא משתנה בעליית שעון, החל ממחזור שעון אחד אחרי שstart עולה, לפני ואחרי פעולת המכונה הוא על מצב ניטרלי

-Next state מיד כש start עולה הוא מתעדכן להיות המצב הרצוי הראשון, ואחר כך אנו מעדכנים אותו לפי המצב של current state, ולכן הוא יתעדכן גם בכל עליית שעון, לפני פעולת המכונה הוא על מצב ניטרלי, ומחזור אחד לפני הסוף הוא חוזר למצב זה.

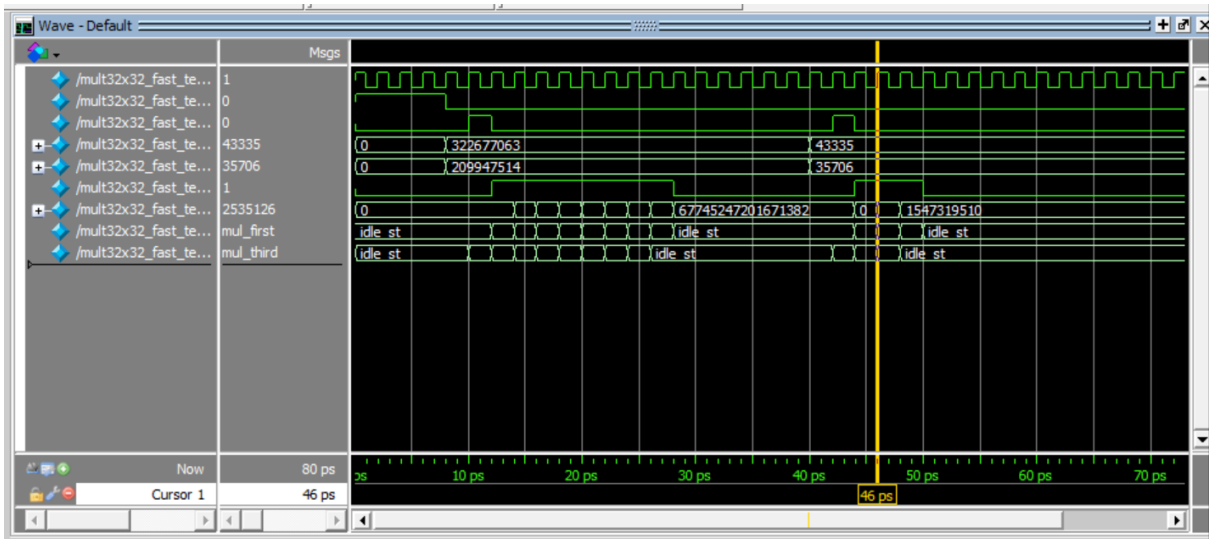
<u>מעקב אחר המצבים:</u>



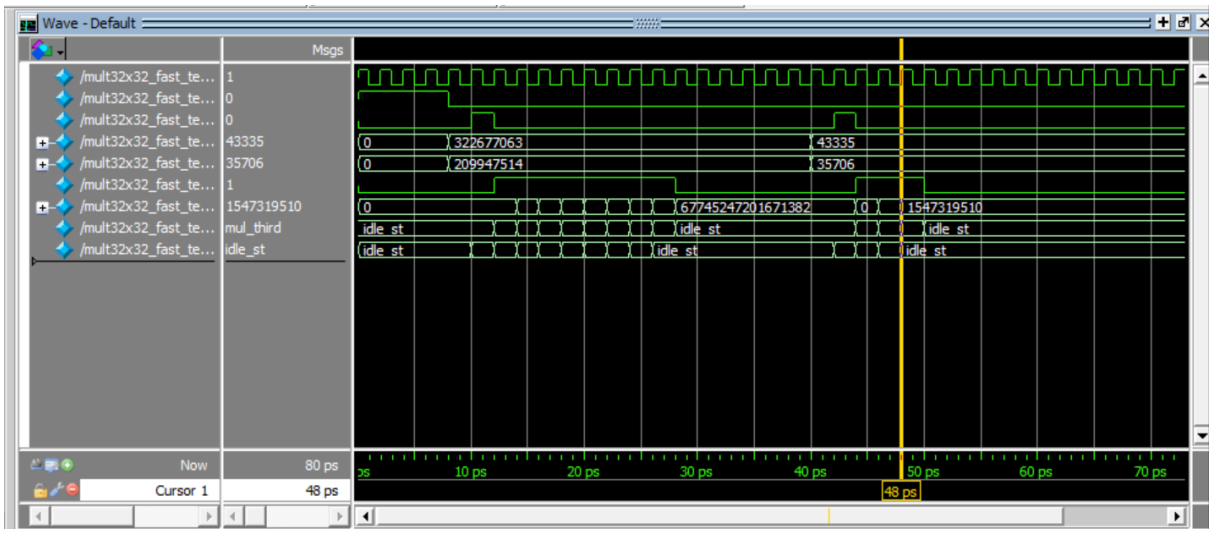
המצב הראשון תמיד-start_st



המצב השני תמיד- mul_first



בגלל ש b_msw==0 מדלגים לשלב השלישי, ו next_stיהיה mul_third



בגלל ש b_msw_is_0 == 1'b1 && a_msb_is_0 == 1'b1 השלב הבא יהיה הidle_st