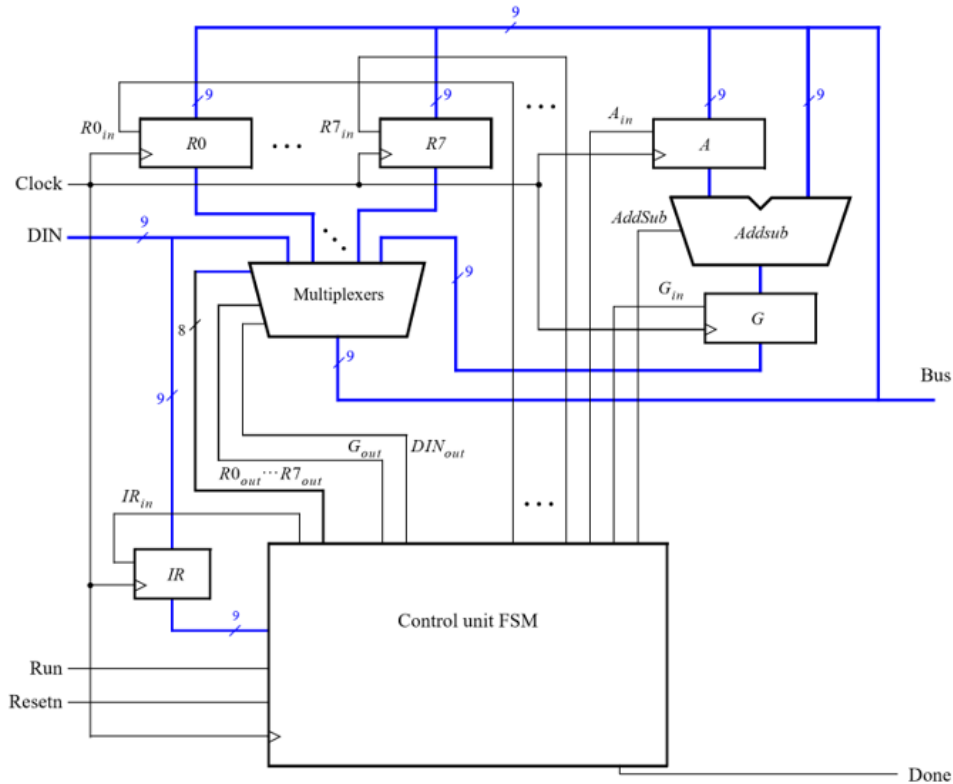


דו"ח מעבדה

מעבדה 3 – מעבד multi-cycle פשוט

מגישים : 342851284 איליה פורטוס

206698433 ארד שפירא



במעבדה הזאת ממשנו מעבד ה-multi-cycle אשר תומך בפעולות: move, move immediate, add, subtract, specialMult. בפרויקט שלנו קראנו ל-Smult - specialMult.

מבנה המעבד:

המערכת מורכבת מ-4 חלקים עיקריים:

1 - זיכרון:

במעבד קיימים 11 רגיסטרים של 9 ביטים (R0-R7, A, G, IR). הרכיבים הללו סינכרוניים ופועלים בעליית שעון.

רגיסטרים R0-R7: רגיסטרים המכילים את "המידע ממש" של המעבד.

כניסות:

- clk - אות שעון
- Resetn – ריסט לאיפוס
- BusWires – המידע שנכנס לרגיסטר ה-i
- Rin[i] – ביט enable שמגיע מהמולטיפלקסר.

יציאה:

- $R[i]$ – הרגיסטר עצמו – מחובר למולטיפלקסר כדי להעביר את הערך השמור הלאה.

רגיסטרים A,G: מיועדים לביצוע פעולות אריתמטיות. ערכיהם נכנסים בפעולות החיבור חיסור והמכפלה ב-3.5 ALU

כניסות:

- clk – אות שעון
- Resetn – ריסט לאיפוס
- BusWires – המידע שנכנס לרגיסטר
- Ain/Gin – ביט enable שמגיע מהמולטיפלקסר.

יציאה:

- Aout_bus/Gout_bus – הרגיסטר עצמו. היציאה של A מחוברת ל-3.5 ALU לצורך ביצוע הפעולה האריתמטית. היציאה של G מחוברת למולטיפלקסר לצורך העברת הערך הנדרש לרגיסטרים R0-R7.

רגיסטר IR: שומר את הפקודה הנוכחית שיש לבצע. קבלת הפקודה היא מהכניסה DIN.

כניסות:

- clk – אות שעון
- Resetn – ריסט לאיפוס
- DIN – המידע שנכנס לרגיסטר
- IRin – ביט enable שמגיע מהמולטיפלקסר

יציאה:

- IR – מחובר ליחידת הבקרה לצורך פיענוח ומימוש הפקודה

להלן מופיע מודול הרגיסטר שניתן לנו, בו השתמשנו לטובת מימוש 11 הרגיסטרים:

```
1 module regn(R, Rin, Clock, Resetn, Q); //with this
2 //module we have implemented the 11 registers
3
4 parameter n = 9;
5 input wire [n-1:0] R; //wires input
6 input wire Rin, //enables the reg
7           Clock,
8           Resetn;
9 output reg [n-1:0] Q; //the reg itself
10
11 always @(posedge Clock or negedge Resetn)
12     if (~Resetn)
13         Q <= 0;
14     else if (Rin)
15         Q <= R;
16 endmodule
17
```

2 – מולטיפלקסר :

יחידה א-סינכרונית שאחראית על העברת המידע בין החלקים השונים במעבד לפי אותות הבקרה המתקבלים. סה"כ בוררת בין 10 ערוצים.

כניסות:

- רגיסטרים R0-R7
- הפקודה DIN
- מוצא רגיסטר G – Gout_bus
- Rout – אות בקרה – ערך של איזה רגיסטר עלינו להעלות על BusWiresn.
- Gout - אות בקרה – להעלאת הערך שברגיסטר G על ה BusWires לטובת הכנסה לאחד הרגיסטרים.
- DINout - אות בקרה – להעלאת הערך שבכניסה DIN לאחד הרגיסטרים.

יציאה:

- mux_reg - הערך שעולה בפועל מכל הערוצים BusWiresn.

להלן המימוש שלנו ליחידת המולטיפלקסר:

```
1 module mux10to1 (
2     //registers:
3     input wire [8:0] R0,
4     input wire [8:0] R1,
5     input wire [8:0] R2,
6     input wire [8:0] R3,
7     input wire [8:0] R4,
8     input wire [8:0] R5,
9     input wire [8:0] R6,
10    input wire [8:0] R7,
11
12    input wire [8:0] DIN, //input command
13    input wire [8:0] Gout_bus, //output of G reg
14    //control signals
15    input wire [7:0] Rout,
16    input wire Gout,
17    input wire DINout,
18
19    output reg [8:0] mux_reg //what goes out form the mux on the Buswires
20 );
21
22 reg [9:0] mux_sel;
23 // assign mux_sel={Rout,Gout,DINout};
24
25 always @(*)
26 begin
27     //here are all the possibilities to choose depended on the control signals
28     if (Gout) begin
29         mux_reg <= Gout_bus;
30     end else if (DINout) begin
31         mux_reg <= DIN;
32     end else if (Rout == 8'b00000001) begin
33         mux_reg <= R0;
34     end else if (Rout == 8'b00000010) begin
35         mux_reg <= R1;
36     end else if (Rout == 8'b00000100) begin
37         mux_reg <= R2;
38     end else if (Rout == 8'b00001000) begin
39         mux_reg <= R3;
40     end else if (Rout == 8'b00010000) begin
41         mux_reg <= R4;
42     end else if (Rout == 8'b00100000) begin
43         mux_reg <= R5;
44     end else if (Rout == 8'b01000000) begin
45         mux_reg <= R6;
46     end else if (Rout == 8'b10000000) begin
47         mux_reg <= R7;
48     end
49 end
50
51
52 endmodule
```

3 - ALU:

מימשנו במקום יחידת הaddsub. זוהי יחידה אסינכרונית שאחראית על ביצוע הפעולות האריתמטיות. הפעולות הן חיבור וחסור בין רגיסטרים והן מכפלה של רגיסטר ב3.5 (תחת הנחה שהרגיסטר בעל כפולה שלמה של 10)

כניסות:

- ALUin1 - הכניסה – הערך שמגיע מרגיסטר A
- ALUin2 - הכניסה – הערך שמגיע מרגיסטר G
- sel – סלקטור שמקבל איזה פקודה צריך לבצע.

יציאה:

- ALUout - המוצא של הALU.

להלן המימוש:

```
1 module ALU(  
2     input wire [8:0] ALUin1,    // reg A  
3     input wire [8:0] ALUin2,    // reg G  
4     input wire [1:0] sel,       // selector what operation to do  
5     output reg [8:0] ALUout);   // output  
6     //operations:  
7     parameter ADD = 2'b01;  
8     parameter SUB = 2'b00;  
9     parameter MULT = 2'b10;  
10  
11     //implemented with blocking assignment because of combinatoric logic  
12     always @(*)  
13     case (sel)  
14         ADD:  
15         begin  
16             ALUout = ALUin1 + ALUin2;  
17         end  
18  
19         SUB:  
20         begin  
21             ALUout = ALUin1 - ALUin2;  
22         end  
23         MULT:  
24         begin  
25             ALUout = (ALUin2 << 1) + ALUin2 + (ALUin2 >> 1);  
26         end  
27     endcase  
28 endmodule
```

4- יחידת בקרה (FSM):

מדובר בלב של המעבד. ממומש כמכונת מצבים שמחליפה בין מצבים בעליית שעון. בכל מחזור שעון מתבצע חלק מסוים של פקודה (מספר מחזורי השעון משתנה בין פקודות, בין השאר מכיוון שיש BusWires אחד) ונשלחים אותות בקרה הרלוונטים למחזור שעון העכשווי.

חלקי הFSM מומשו בקובץ proc.v – להלן חלקי מכונת המצבים:

הגדרת רגיסטרים:

```
35 // parameter regs
36 reg [1:0] Tstep_Q; // cs
37 reg [1:0] Tstep_D; // ns
```

עדכון הצמב:

```
176 // FSM - Resetn controlling
177 always @(posedge Clock, negedge Resetn)
178 begin
179     if (!Resetn)
180         Tstep_Q <= T0;
181     else
182         Tstep_Q <= Tstep_D;
183 end
184
```

מעבר בין מחזורי שעון:

```
53 // Control FSM- cycles depented on commands
54 always @(*)
55 begin
56     case (Tstep_Q)
57     T0: begin
58         if (!Run)
59             Tstep_D <= T0; // If not given Run prompt
60         else
61             Tstep_D <= T1;
62     end
63     T1: begin
64         if (Done)
65             Tstep_D <= T0; // return if command is 2 cycles long
66         else
67             Tstep_D <= T2;
68     end
69     T2: begin
70         if (Done)
71             Tstep_D <= T0; //happens if smult happens
72         else
73             Tstep_D <= T3;
74     end
75     T3: begin
76         Tstep_D <= T0;
77     end
78     default: Tstep_D <= T0;
79 endcase
80 end
```

שליטה בלוגיקה:

```
82 // Control FSM
83 always @(*)
84 begin
85     // instance for all the regs
86     IRin <= 1'b0;
87     Gin <= 1'b0;
88     Ain <= 1'b0;
89     Done <= 1'b0;
90     Rin <= 8'b0;
91     Gout <= 1'b0;
92     DINout <= 1'b0;
93     Rout <= 8'b0;
94
95     case (Tstep_Q) // CURRENT STATE CASE:
96     T0: begin
97         IRin <= 1'b1; // enable reg IR
98     end
99
```

```

100 T1: // first cycle
101 case(I)
102 MV:
103 begin
104 Rin <= Xreg; // read reg X
105 Rout <= Yreg; // write reg Y
106 Done <= 1'b1; // enable Done
107 end
108 MVI:
109 begin
110 Rin <= Xreg; // write reg X
111 DINout <= 1'b1; // enable DINout
112 Done <= 1'b1; // enable Done
113 end
114 ADD:
115 begin
116 Rout <= Xreg; // read reg X
117 Ain <= 1'b1; // enable reg A
118 end
119 SUB:
120 begin
121 Rout <= Xreg; // read reg X
122 Ain <= 1'b1; // enable reg A
123 end
124 SMULT:
125 begin
126 Rout <= Xreg; // read reg X
127 Gin <= 1'b1; // enable reg G
128 AddSub <= 2'b10; //enable smult operation in alu
129 end
130 endcase
131 T2: // second cycle
132 case(I)
133 ADD:
134 begin
135 Rout <= Yreg; // read from reg Y
136 Gin <= 1'b1; // enable reg G
137 AddSub <= 2'b01; // selector for add
138 end
139 SUB:
140 begin
141 Rout <= Yreg; // read from reg Y
142 Gin <= 1'b1; // enable reg G in
143 AddSub <= 2'b00; // selector for subtract
144 end
145 SMULT:
146 begin
147 Gout <= 1'b1; // enable reg G out
148 Rin <= Yreg; // // write to reg Y
149 Done <= 1'b1;
150 end
151 endcase
152 T3:
153 case(I)
154 ADD:
155 begin
156 Gout <= 1'b1; // enable reg G out
157 Rin <= Xreg; // write to reg X
158 Done <= 1'b1; // enable Done
159 end
160 SUB:
161 begin
162 Gout <= 1'b1;
163 Rin <= Xreg;
164 Done <= 1'b1;
165 end
166 endcase
167 endcase
168 end

```

כאשר I בcase שבקוד לעיל הוא שלושת הביטים MSB שמציינים איזו פקודה מבוצעת (יוסבר בהמשך),

Xreg ו Yreg הם אותות הבקרה שמציינים לאיזה רגיסטר עלינו לפנות (יוסבר בהמשך)

הקלט למעבד נכנס ב DIN כקלט של 9 ביטים כIIIXXXYYY. כאשר שלושת הביטים MSB מייצגים את הפקודה. האמצעיים מייצגים את RX מתוך 8 הרגיסטרים, הLSB מייצגים את RY מתוך 8 הרגיסטרים. בלחיצה על RUN ובעליית שעון מתבצעת פקודה חדשה (במעבדה השתמשנו בKEY, כך שדגימת השעון קוראת בnegedge כי כך עובדים הKEYS). ביט Done נשלח במחזור שעון האחרון שלכל פקודה.

במחזור שעון הראשון של כל פקודה, הפקודה עצמה נשמרת ברגיסטר IR. ואנחנו מפרקים אותה לפי הפירוק שלעיל: נראה את החלק הרלוונטי בקוד:

```
51 // decode input of IR
52 assign I = IR[8:6];
53 dec3to8 decX (IR[5:3], 1'b1, Xreg);
54 dec3to8 decY (IR[2:0], 1'b1, Yreg);
55
```

וכך מקבלים את Xreg, Yreg, I במכונת מצבים שלעיל.

קבלת Xreg ו Yreg קוראת ע"י שימוש בדיקודר 3 ל8, שמעביר ייצוג בינארי לייצוג של 8 ספרות:

```
1 module dec3to8(w, En, Y);
2 input wire [2:0] w;
3 input wire En;
4 output reg [7:0] Y;
5
6 always @(*)
7 begin
8 if (En == 1)
9 case (w)
10 3'b111: Y = 8'b10000000;
11 3'b110: Y = 8'b01000000;
12 3'b101: Y = 8'b00100000;
13 3'b100: Y = 8'b00010000;
14 3'b011: Y = 8'b00001000;
15 3'b010: Y = 8'b00000100;
16 3'b001: Y = 8'b00000010;
17 3'b000: Y = 8'b00000001;
18 endcase
19 else
20 Y = 8'b00000000;
21 end
22 endmodule
23
```

בקובץ proc.v המצורף לקובץ זה מופיעים בנוסף כל הכניסות והיציאות של המעבד, מימוש הFSM (כפי שצורף כאן), האינסטנציאציות לALU, לרגיסטרים ולמולטיפלקסר.

טבלת תיאור הפקודות:

בתאים של T0-T3 מתוארים אותות הבקרה שנדלקים במחזורי השעון הללו.

Operation	Instruction (MSB)	What happens	T0	T1	T2	T3
mv	000	$RX \leftarrow [RY]$	IRin	Ryout, Rxin Done		
mvi	001	$RX \leftarrow D$	IRin	DINout, Rxin, Done		
add	010	$RX \leftarrow [RX] + [RY]$	IRin	Rxout, Ain	Ryout, Gin, addsub=01	Gout, Rxin, Done
sub	011	$RX \leftarrow [RX] - [RY]$	IRin	Rxout, Ain	Ryout, Gin, addsub=00	Gout, Rxin, Done
smult	100	$RY \leftarrow 3.5 * RX$	IRin	Rxout, Gin, addsub=10	Gout, Ryin, Done	

הסבר לפקודות:

נסביר את מה שקורה בכל פקודה, כל מחזור שעון, ועל כן אותות הבקרה שבטבלה הם הללו שצריכים לדלוק (חשוב היה לדאוג שהם כבויים כל זמן שהם אינם נדרשים להיות דלוקים)

MV - ראשית שומרים את הפקודה ב IRin ב T0. במחזור שעון T1 מוציאים את הערך שיש ברגיסטר Y ומכניסים אותו לרגיסטר X, מוציאים ביט Done.

MVI - ראשית שומרים את הפקודה ב IRin ב T0. במחזור שעון T1 מכניסים ב DIN את הערך אותו רוצים לכתוב לרגיסטר X ומכניסים אותו לרגיסטר X ומוציאים ביט Done.

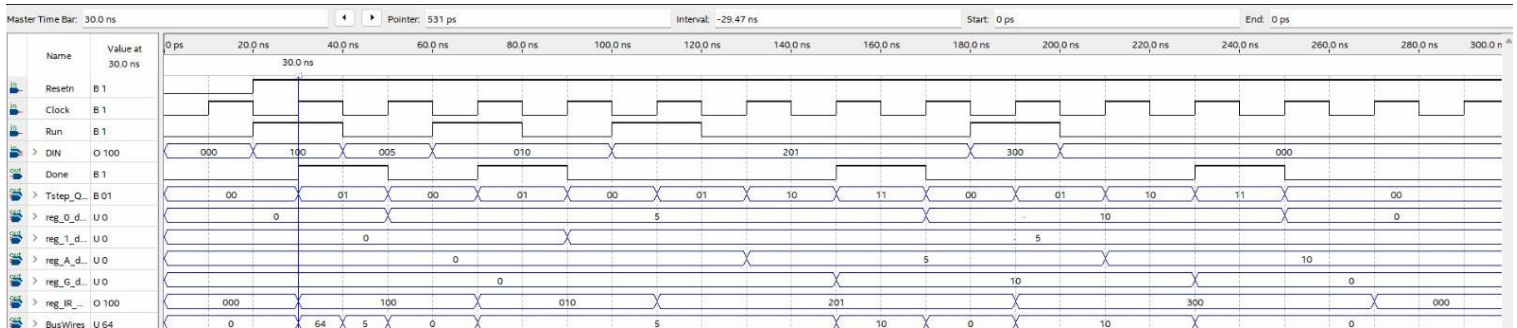
ADD – ראשית שומרים את הפקודה ב IRin ב T0. במחזור שעון T1 מכניסים את הערך של רגיסטר X לרגיסטר A. במחזור שעון T2 מכניסים את הערך שברגיסטר Y ל ALU ביחד עם הערך שברגיסטר A ומתבצעת פעולת החיבור והערך מוכנס לרגיסטר G. ב T3 הערך יוצא מרגיסטר G ונכנס לרגיסטר X ומוציאים ביט Done.

SUB - זהה ל ADD עד כדי חיסור בין הרגיסטרים.

Smult – ראשית שומרים את הפקודה ב IRin ב T0. ב T1 אנחנו מוציאים את הערך של רגיסטר X, מבצעים את פעולת ההכפלה עליו ושומרים את התוצאה ברגיסטר G. ב T2 מוציאים את הערך שיש ברגיסטר G, מכניסים אותו לרגיסטר Y ומוציאים ביט Done.

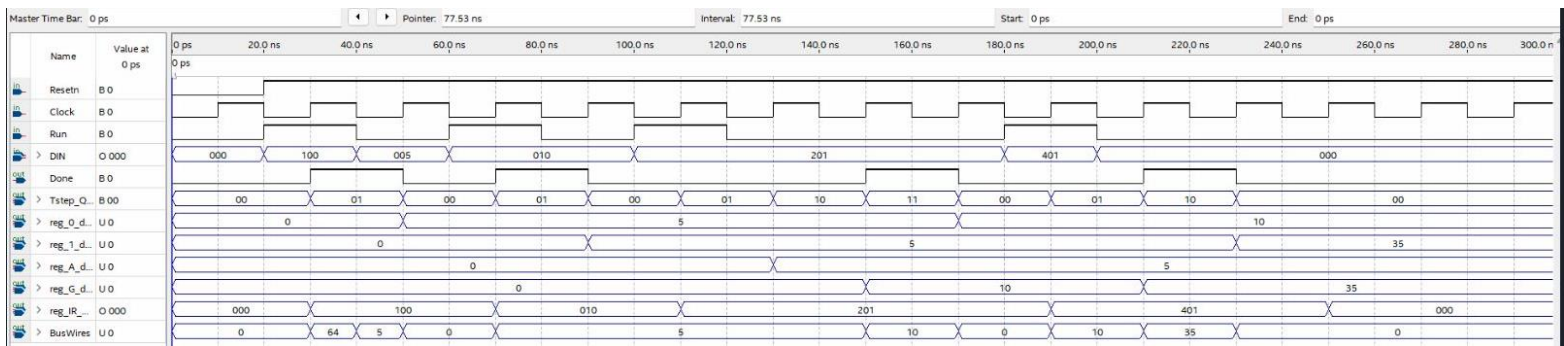
סימולציות-

נראה כעת סימולציות המראות את נכונות מימוש המעבד :



תוצאת הסימולציה זהה לזו שניתנה לנו בתרגיל

נראה סימולציה נוספת עם מימוש של הפקודה הנוספת (התחלה בזמן 180 ns) ניתן לראות שרגיסטר 1 מקבל את הערך של רגיסטר 0 מוכפל ב 3.5 (ברגיסטר 0 היה 10 – על כן נכנס לרגיסטר 1 הערך 35). כמו כן ניתן לראות שהפקודה שמימשנו לוקחת 3 מחזורי שעון, כפי שרצינו לממש.



לפי מה שביקשו בחלק א' עשינו מודול top למעבד, והצריבה ל FPGA קרתה דרכו.

נראה את המודול כאן :

```

1 module proc_top(DIN, Resetn, Clock, Run, Done, Buswires);
2
3   input wire [8:0] DIN; //data into the processor
4   input wire Resetn, Clock, Run;
5   output wire Done; // sent in the last cycle of each command
6   output wire [8:0] Buswires; //entry to regs & output of mux
7
8   //instance to the processor:S
9   proc proc_inst(.DIN(DIN), .Resetn(Resetn), .Clock(Clock),
10  |       .Run(Run), .Done(Done), .Buswires(Buswires));
11
12 endmodule

```

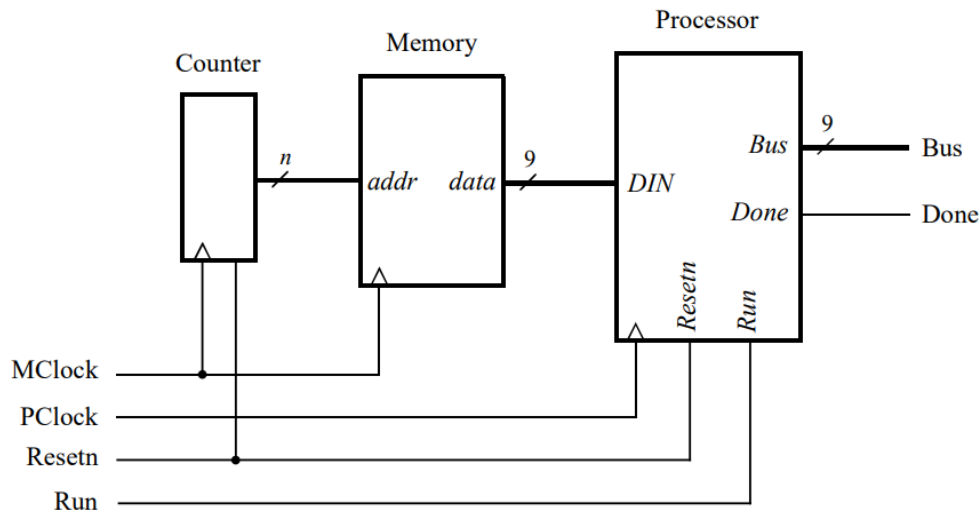
מציאת תדר מקסימלי - מצאנו את הנתונים מהפלט של הקומפליציה :

Slow 1100mV 0C Model Fmax Summary			
	Fmax	Restricted Fmax	Clock Name
1	52.04 MHz	52.04 MHz	proc:proc_inst Tstep_Q.T1
2	79.18 MHz	79.18 MHz	Clock
3	137.06 MHz	137.06 MHz	proc:proc_inst AddSub[0]

כלומר בהינתן הנתונים בשורה הכחולה השעון (Clock) המקסימלי שהמערכת תעבוד בו יהיה 79.18 MHz .
מצורף כאן קישור לסרטון להראות את נכונות המימוש לאחר הצריכה:

<https://drive.google.com/file/d/1VgN2lJRZ8xGvc64mTOYu7wc-Lt9yze-t/view?usp=sharing>

בחלק השני של המעבדה נדרשנו לממש את המערכת הבאה :



כאשר מדובר במעבד אותו יצרנו כבר בחלק הקודם (עם כל הכניסות היציאות) – כעת הDIN מגיע מרכיב הזיכרון (SRAM – synchronous read only memory) – על כן זהו זיכרון ממנו ניתן רק לקרוא). את הפקודות בזיכרון ניתן לקדם ע"י שימוש בcounter ובכך להביא בעצם פקודות חדשות לפי מחזור שעון. נשים לב כי כעת יש שעון נפרד למעבד (PClock) ושעון נפרד לSRAM ולcounter (Mclock). הריסט משותף לרכיבים.

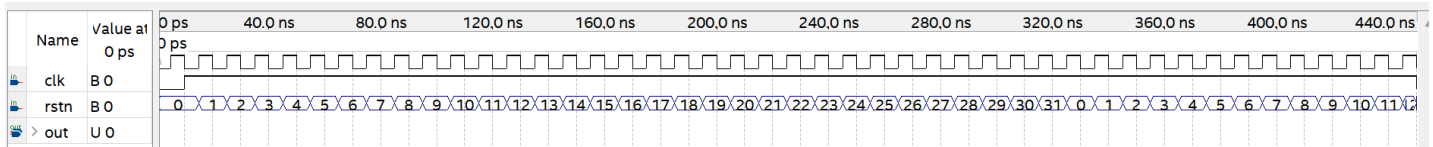
הסבר איך המערכת עובדת: counter סופר מ0 עד 31 (יש סה"כ 32 פקודות- יוסבר בהמשך) – ומוציא כל מחזור שעון של Mclock את הספירה הנוכחית (בציר מדובר ב n). n שבציר נכנס לזיכרון – הכניסה הזאת מתורגמת לאיזה שורה (כל שורה היא פקודה של 9 ביטים) צריך להוציא outputs, שייכנס לDIN ומשם למעבד.

מימוש של counter:

```

1 module counter ( //counter 0 to 31 included
2   input wire clk,
3   input wire rstn,
4   output wire [4:0] out //this enters the memory
5 );
6 parameter COUNT_TO = 5'b11111; //31 in binary
7 reg [4:0] count; // saves the current counting
8
9 always @(posedge clk or negedge rstn)
10 begin
11   if (~rstn)
12   begin
13     count <= 0; //no need to count
14   end
15   else if (count > COUNT_TO) //need to initialize
16   begin
17     count <= 5'b0;
18   end
19   else
20   begin
21     count <= count + 5'b1; //else: need to add 1
22   end
23 end
24 assign out = count; //sends as output current counting
25 endmodule
  
```

נראה דיאגרמת גלים לנכונות של המימוש :



ניתן לראות שהרכיב סופר עד הנדרש, ומתאפס אח"כ ומתחיל ספירה מחדש.

מימוש רכיב הזיכרון:

לצורך המימוש עקבנו אחר ההסברים שניתנו. השתמשו בIP Catalog ליצירת זיכרון של 32 מילים, כאשר כל מילה של 9 ביטים. יצרנו זיכרון ROM-1PORT, צרבנו לתוכו את הערכים של קובץ inst_mem.mif שיצרנו (נראה תכף) ושמרנו את הזיכרון כקובץ בשם inst_mem.v

קובץ mifn (memory initialization file) שיצרנו בעל 32 שורות, כל שורה של 9 ביט. כך שכל שורה מתאימה לכניסה לDIN (וכמות השורות מתאימה למה שדרשו מאיתנו). הכנסנו לקובץ mif את הפקודות הנדרשות. יצרנו את הזיכרון פעמיים - פעם אחת עבור סימולציה שתהיה תואמת לסימולציה שהראו לנו בכיתה, ופעם נוספת עבור סימולציה שמחליפה את הפקודה 300 שהופיעה (פעולת חיסור - בין רגיסטר 0 לעצמו) לפקודה 401 שהיא הפקודה שמדגימה את נכונות הפקודה smult שקיבלנו לממש (פעולת smult - הכנסת ערך של רגיסטר 0 כפול 3.5 לרגיסטר 1)

להלן קבצי mifn :

- ללא פקודה נוספת :

```

1  DEPTH = 32;
2  WIDTH = 9;
3  ADDRESS_RADIX = HEX;
4  DATA_RADIX = BIN;
5  CONTENT
6  BEGIN
7
8      00: 001000000;
9      01: 000000101;
10     02: 000001000;
11     03: 010000001;
12     04: 011000000;
13     05: 000000000;
14     06: 000000000;
15     07: 000000000;
16     08: 000000000;
17     09: 000000000;
18     0A: 000000000;
19     0B: 000000000;
20     0C: 000000000;
21     0D: 000000000;
22     0E: 000000000;
23     0F: 000000000;
24     10: 000000000;
25     11: 000000000;
26     12: 000000000;
27     13: 000000000;
28     14: 000000000;
29     15: 000000000;
30     16: 000000000;
31     17: 000000000;
32     18: 000000000;
33     19: 000000000;
34     1A: 000000000;
35     1B: 000000000;
36     1C: 000000000;
37     1D: 000000000;
38     1E: 000000000;
39     1F: 000000000;
40
41  END;
```

- בתוספת פקודת smult :

```

1  DEPTH = 32;
2  WIDTH = 9;
3  ADDRESS_RADIX = HEX;
4  DATA_RADIX = BIN;
5  CONTENT
6  BEGIN
7
8      00: 001000000;
9      01: 000000101;
10     02: 000001000;
11     03: 010000001;
12     04: 100000001;
13     05: 000000000;
14     06: 000000000;
15     07: 000000000;
16     08: 000000000;
17     09: 000000000;
18     0A: 000000000;
19     0B: 000000000;
20     0C: 000000000;
21     0D: 000000000;
22     0E: 000000000;
23     0F: 000000000;
24     10: 000000000;
25     11: 000000000;
26     12: 000000000;
27     13: 000000000;
28     14: 000000000;
29     15: 000000000;
30     16: 000000000;
31     17: 000000000;
32     18: 000000000;
33     19: 000000000;
34     1A: 000000000;
35     1B: 000000000;
36     1C: 000000000;
37     1D: 000000000;
38     1E: 000000000;
39     1F: 000000000;
40
41  END;

```

מודול ה mem_inst.v נוצר באופן אוטומטי, נראה את החתימה שלו (ניתן לצפות בקובץ כולו שמצורף):

```

39 module inst_mem (
40     address,
41     clock,
42     q);
43
44     input [4:0] address;
45     input      clock;
46     output [8:0] q;

```

כעת נראה את קובץ ה top שמאחד את המעבד עם הזיכרון ועם counter :

```

1 module part2_top(Resetn, MClk, PClock, Run, Done, BusWires);
2
3   input wire Resetn, Run;
4   input wire MClk, PClock; //clocks for memory and processor
5   output wire Done; // sent in the last cycle of command in PClock
6   output wire [8:0] BusWires;
7
8   wire [8:0] q; //output of the Srom
9   wire [4:0] count_out; //output of counter
10
11   // counter instance
12   counter counter_inst(.clk(MClk), .rstn(Resetn), .out(count_out));
13
14   // memory instance
15   inst_mem inst_mem(.address(count_out), .clock(MClk), .q(q));
16
17   // processor instance
18   proc proc_inst(.DIN(q), .Resetn(Resetn), .clock(PClock),
19   |   .Run(Run), .Done(Done), .BusWires(BusWires));
20
21 endmodule
22

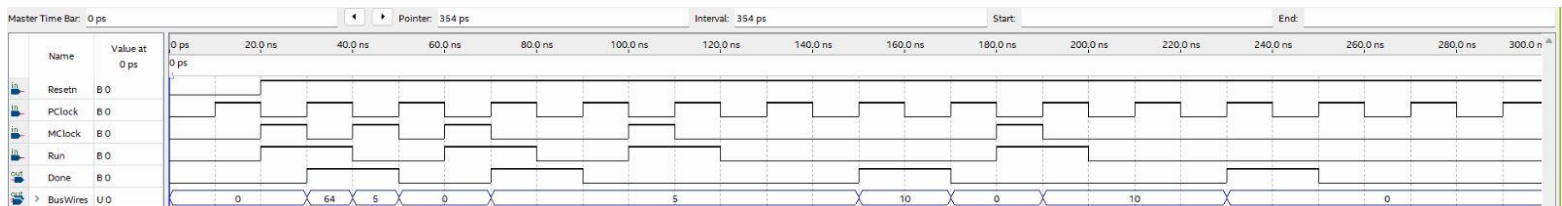
```

נראה כעת סימולציות לנכונות ה design. נציין כי היה צורך לשים את ה clocks בהיסט אחד מהשני, בנוסף השעונים הללו בהכרח לא חופפים, מכיוון שיש פקודות שצריכות פנייה אחת לזיכרון (לשליפת הפקודה), ושיש את פקודת movi שצריכה פנייה כפולה לזיכרון.

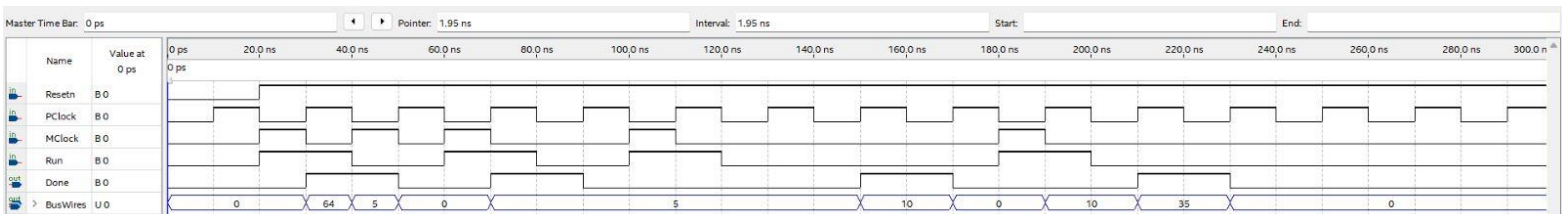
סימולציות להראת נכונות ה design :

ב2 הסימולציות ניתן לראות שה BusWires וה Done זהים למה שנראה לפני חיבור המעבד לזיכרון ול counter, והם משתנים בזמנים הנכונים

• ללא פקודה נוספת :



• עם פקודה נוספת :



צרבנו את ה design לכרטיס לפי ההנחיות.

מצורף קישור סרטון בהינתן הפקודה הנוספת :

<https://drive.google.com/file/d/1WIm2kdUMirYlhQVq6mn-fNzLsLylDfCv/view?usp=sharing>