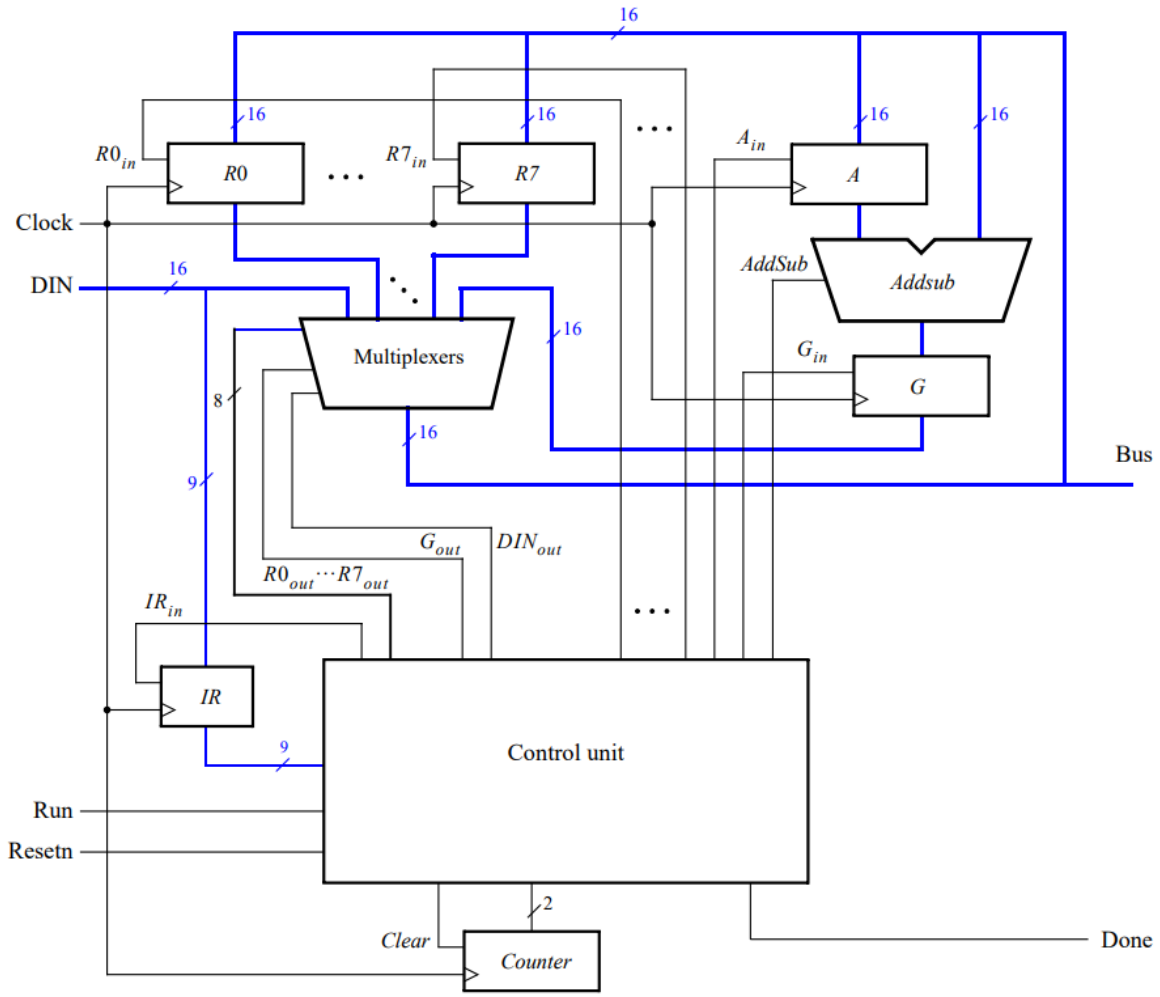
מעבדה לVHDL – תכנון מיקרופרוסור

בניסוי זה היה עלינו לתכנן מיקרופרוסור שיעבוד בשיטת RTL (Register Transfer Language) על סמך הארכיטקטורה הבאה:



שלבי עבודה לתכנון המיקרופרוסור

1. הבנת התיאוריה – כולל הבנת תהליך העבודה של המיקרופרוסור (תהליך ביצוע פקודות + מסלולי נתונים) וכולל הבנת מבנה הפקודות והסוגים שלהם.
2. תכנון יחידות – בשלב זה תכננו כל יחידה בארכיטקטורה כ- Component ובדקנו את תוצאת הסינתזה (RTL) וגם הרצנו סימולציה ב-ModelSim בשביל לראות שאנו מקבלים את התוצאות הרצויות.
3. תכנון הארכיטקטורה הכוללת של המערכת – שילוב כל היחידות בארכיטקטורה.
4. סימולציה סופית ותיקון שגיאות – בשלב זה הרצנו סימולציה לכלל המערכת בה הרצנו פקודות ובדקנו את תגובת המערכת ותיקנו את השגיאות במידת הצורך.

שלב 1 – הבנת התיאוריה

שלב זה מתחלק ל-2 חלקים.
בחלקו הראשון היה עלינו להבין את מסלול הנתונים של המיקרופרוסור + תהליך ביצוע פקודה מההתחלה ועד הסוף.
בחלק השני היה עלינו להבין כיצד מקודדת פקודה באסמבלר לשפת מכונה כך שתהיה התאמה בין קידוד הפקודה לארכיטקטורה של המיקרופרוסור.

<u>תהליך ביצוע פקודה</u>
תהליך ביצוע פקודה במיקרופרוסור הוא תהליך הכולל בדרך כלל 5 שלבים:
1. Fetch – שלב שבו שולפים את הפקודה מהזיכרון (שלב זה מתבצע בעזרת הרגיסטר IR (Instruction Register)).
2. Decode – שלב שבו המיקרופרוסור מאפשר רגלי בקרה בהתאם לפקודה (לדוגמא, על המיקרופרוסור לבצע פעולה אריתמטית כלשהיא בין הרגיסטרים R3 ו-R5, הרגלי בקרה של R3 ו-R5 יאופשרו וכל שאר הרגיסטרים יהיו במצב לא פעיל) שלב זה מבוצע על ידי חלק מסוים שנמצא ביחידה Control Unit.
3. Execute – שלב ביצוע הפקודה, מתבצע ב-ALU.
4. Memory – כתיבת התוצאה לזיכרון חיצוני (זיכרון שאינו חלק מהרגיסטרים), בניסוי זה לא היינו צריכים לממש שלב זה בתהליך.
5. Write Back – כתיבת התוצאה לרגיסטרים שלב זה מתבצע על ידי אפשור הרגיסטר אליו אנו רוצים לכתוב את המידע החדש ואפשור הוצאת מידע מהרגיסטר אליו נכנס התוצאה של ה-ALU.

כל פקודה לא חייבת לכלול את כל חמשת השלבים.

היחידה Control Unit היא היחידה העיקרית במערכת שהיא מאפשרת את הרגיסטרים המתאימים בכל שלב בתהליך ביצוע הפקודה.
במקביל ליחידה זו קיימת יחידה Counter שסופרת עליות שעון של המערכת ועוזרת ליחידה Control Unit להבין באיזה שלב בתהליך ביצוע הפקודה המיקרופרוסור נמצא.

<u>אופן קידוד פקודות באסמבלר להוראות בשפת מכונה</u>

היה עלינו לממש מיקרופרוסור שיודע לבצע בסך הכול 4 פקודות.
המיקרופרוסור מקבל פקודות אלה מSignal בשם Din (אות זה הוא ווקטור בגודל 16 סיביות)
ההוראה בשפת מכונה מקודדת לתשע סיביות הגבוהות (סיביות הMSB) שהן הולכות לרגיסטר IR.

אופן קידוד ההוראה: IIIXXXYYY
I – שלושה סיביות המייצגות את סוג ההוראה.
X – שלושה סיביות המייצגות את מספר הרגיסטר X.
Y – שלושה סיביות המייצגות את מספר הרגיסטר Y.

הפקודות שהיה עלינו לממש הייתה נתונה בטבלה: (Table 1)

Operation	Function performed
mv Rx,Ry	$Rx \leftarrow [Ry]$
mvi $Rx,\#D$	$Rx \leftarrow D$
add Rx, Ry	$Rx \leftarrow [Rx] + [Ry]$
sub Rx, Ry	$Rx \leftarrow [Rx] - [Ry]$

כפי שכבר הצגתי למעלה כל פקודה היא תהליך המיקרופרוסור כאשר כל אחת מהן דורשת מספר ספציפי של זמן לביצוע (או במילים אחרות כל פקודה דורשת כמות מחזורי שעון ספציפית המתאימה לסוג הפעולה).

לשם כל קיבלנו טבלה מוכנה המייצגת את כמות מחזורי השעון לכל הפעולות ואיזה שלב מתבצע בכל מחזור שעון בכל פעולה.

	T_1	T_2	T_3
(mv): I_0	$RY_{out}, RX_{in}, Done$		
(mvi): I_1	$DIN_{out}, RX_{in}, Done$		
(add): I_2	RX_{out}, A_{in}	RY_{out}, G_{in}	$G_{out}, RX_{in}, Done$
(sub): I_3	RX_{out}, A_{in}	$RY_{out}, G_{in}, AddSub$	$G_{out}, RX_{in}, Done$

הערה: הטבלה מתחילה מהמחזור שעון השני אך יש לקחת בחשבון כי במחזור שעון הראשון מתבצע שלב של שליפת הפקודה מהזיכרון והכנסתו לתוך רגיסטר IR (שלב Fetch).
שלב זה קורה עבור כל סוגי הפקודות, בנוסף חשוב לציין שבשלב זה כל רגלי הבקרה של כל הרגיסטרים לא מאופשרים חוץ מהרגיסטר IR.

ניתן לראות באופן ברור מספיק מה קורה בכל מחזור שעון של המערכת עבור כל פקודה וכעת וניתן להתחיל לתכנן את המיקרופרוסור.

שלב 2 - תכנון יחידות במיקרופרוסור

בשלב זה כתבנו תוכניות בVHDL לכל התת יחידות במיקרופרוסור.

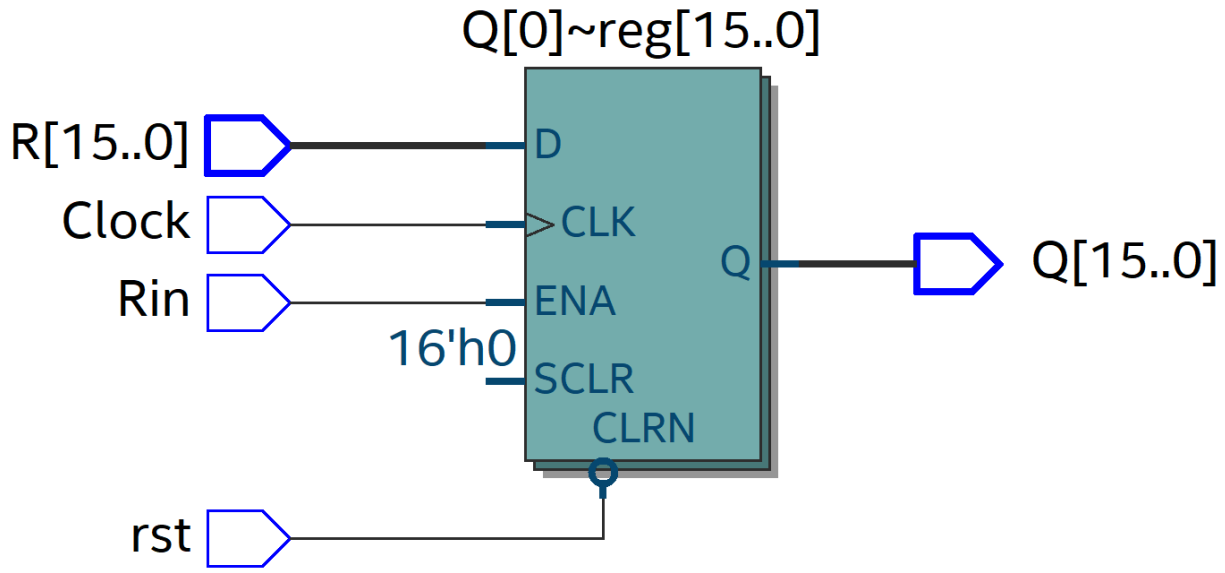
חלק מהתוכניות כבר היו נתונות אך בכל זאת ביצענו בהם קצת שינויים בכדי להתאים אותם לאופי המערכת שלנו.

היחידה הראשונה שתכננו היא האוגר (Register) יחידה זו שימשה אותנו לצורך מימוש כל האוגרים במערכת.
האוגר בתוכנית שלנו צריך להכיל מוצא בגודל 16 סיביות שתהיה פעילה תמיד, כניסה בגודל 16 סיביות שתהיה פעילה רק בעליית שעון ורק אחרי בדיקת הרגל בקרה (שתאפשר כניסה), ובנוסף הוספנו הדק Reset אסינכרוני בשביל שהערך ההתחלתי של האוגר יהיה ידוע מראש כאשר מאפסים את המיקרופרוסור.
הערה: כל האוגרים במיקרופרוסור בגודל 16 סיביות חוץ מהאוגר IR (שהוא בעל 9 סיביות) ולכן הגדרנו את האוגר באמצעות Generic מה הגודל שאנו צריכים לכל אוגר.

תוכנית לאוגר (regn):

```
library ieee;
use ieee.std_logic_1164.all;

entity regn is
generic (n : integer := 16);
port (
R: in std_logic_vector(n-1 downto 0);
rst, Rin, Clock: in std_logic;
Q: buffer std_logic_vector(n-1 downto 0));
end;

architecture Behavior of regn is
begin
process (rst,Clock)
begin
if (rst = '0') then Q <= (others =>'0');
elsif (Clock 'event and Clock = '1') then
      if (Rin = '1') then
            Q <= R;
      end if;
end if;
end process;
end;
```

סכימת RTL:



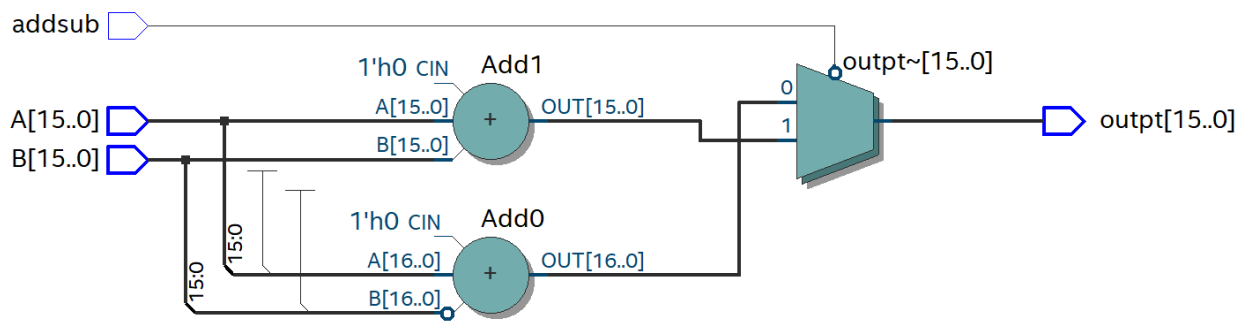
הכניסה ENA שיש לרגיסטר עובדת על ידי מרבב (mux) שדוגם את המוצא בחזרה לכניסה כאשר הselector של המרבב הוא הכניסה Rin.

לאחר מכן כתבנו את התוכנית עבור הALU.
יחידה זו אמורה לבצע חיבור/חיסור תלוי במצב של ההדק בקרה בכניסה.

תוכנית ל-ALU:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity ALU is
port(
A,B: in std_logic_vector (15 downto 0);
addsub: in std_logic;
outpt: out std_logic_vector(15 downto 0));
end;

architecture one of ALU is
begin
outpt <= A + B when addsub = '0' else A - B ;
end;
```

סכימת RTL:



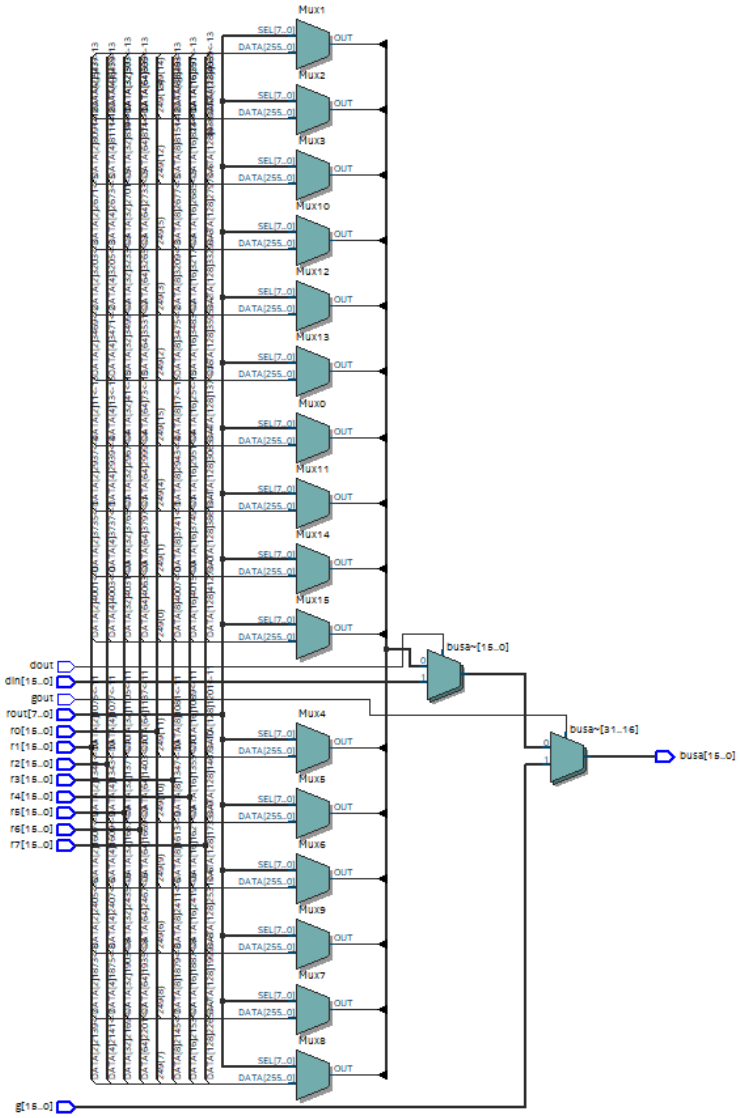
לאחר מכן תכננו את המרבב.
יחידה זו עוזרת לנו להוציא לקו הנתונים (Bus) של המיקרופרוסור את הרגיסטר הספציפי שממנו רוצים לקרוא את המידע, בכך אנו מתגברים על שגיאת Multiple Driver (שגיאה זו קוראת כאשר אנו מאלצים Signal יציאה להיות בערכים שונים בו זמנית).

תוכנית למרבב (Mux):

```
library ieee;
use ieee.std_logic_1164.all;

entity mux is
port(
r0,r1,r2,r3,r4,r5,r6,r7,din,g : in std_logic_vector (15 downto 0);
gout,dout: in std_logic;
rout: in std_logic_vector (7 downto 0);
busa: out std_logic_vector (15 downto 0));
end;
```

```
architecture bhv of mux is
begin
process (r0,r1,r2,r3,r4,r5,r6,r7,din,g,gout,dout,rout) is
begin
      if (gout = '1') then
            busa <= g;
      elsif (dout = '1') then
            busa<=din;
      else
            case (rout) is
            when "10000000" => busa <= r7;
            when "01000000" => busa <= r6;
            when "00100000" => busa <= r5;
            when "00010000" => busa <= r4;
            when "00001000" => busa <= r3;
            when "00000100" => busa <= r2;
            when "00000010" => busa <= r1;
            when others        => busa <= r0;
            end case;
            end if;
end process;
end;
```

<u>סכימת RTL:</u>

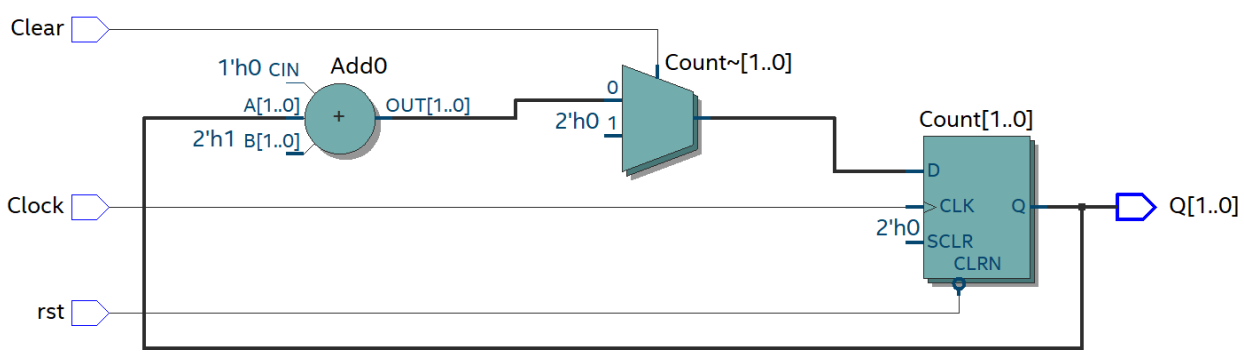


כעת עלינו לתכנן את החלק העיקרי במיקרופרוסור שהם היחידות שאחראיות לביצוע תהליך הפקודות.
היחידה הראשונה היא המונה Counter.

תוכנית למונה (upcount):

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity upcount is
port (
Clear, Clock,rst: in std_logic;
Q : out std_logic_vector(1 downto 0));
end;

architecture Behavior of upcount is

signal Count : std_logic_vector(1 downto 0);

begin
process (rst,Clock)
begin
if (rst = '0') then count <= (others => '0');
elsif (Clock 'event and Clock = '1') then
      if (Clear = '1') then
            Count <= "00";
      else
            Count <= Count + 1;
      end if;
end if;

end process;

Q <= Count;
end;
```

<u>סכימת RTL:</u>

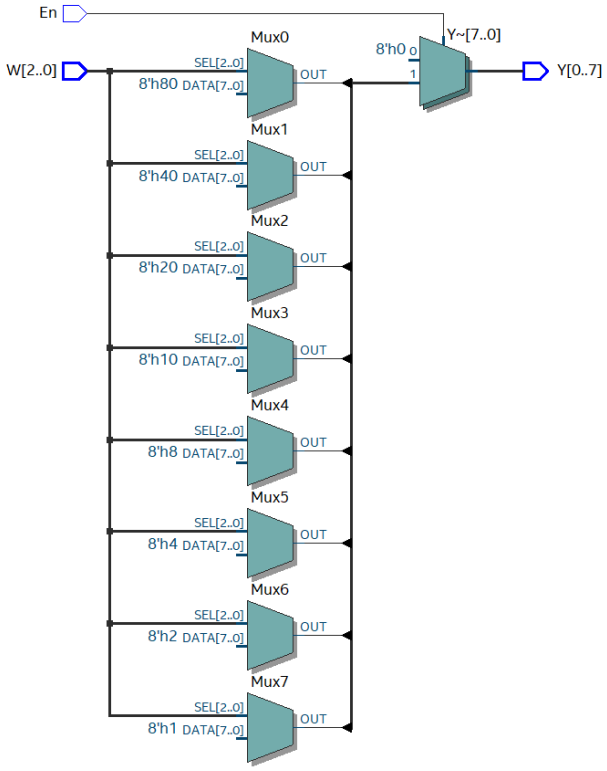


היחידה השנייה היא המפענח שעוזרת לאפשר רגלי בקרה ספציפיים עבור האוגרים.
המפענחים נמצאים בתוך ה Control Unit ומתחברים ישירות לרגל האפשור קריאת נתונים של האוגרים ולמרבב.

תוכנית למפענח (dec3to8):

```
library ieee;
use ieee.std_logic_1164.all;

entity dec3to8 is
port ( W : in std_logic_vector(2 downto 0);
En : in std_logic;
Y : out std_logic_vector(0 to 7));
end;

architecture Behavior of dec3to8 is
begin

process (W, En)
begin
if (En = '1') then
      case W is
      when "000"  => Y <= "00000001";
      when "001"  => Y <= "00000010";
      when "010"  => Y <= "00000100";
      when "011"  => Y <= "00001000";
      when "100"  => Y <= "00010000";
      when "101"  => Y <= "00100000";
      when "110"  => Y <= "01000000";
      when others => Y <= "10000000";
      end case;
else
      Y <= "00000000";
end if;
end process;
end Behavior;
```

סכימת RTL:



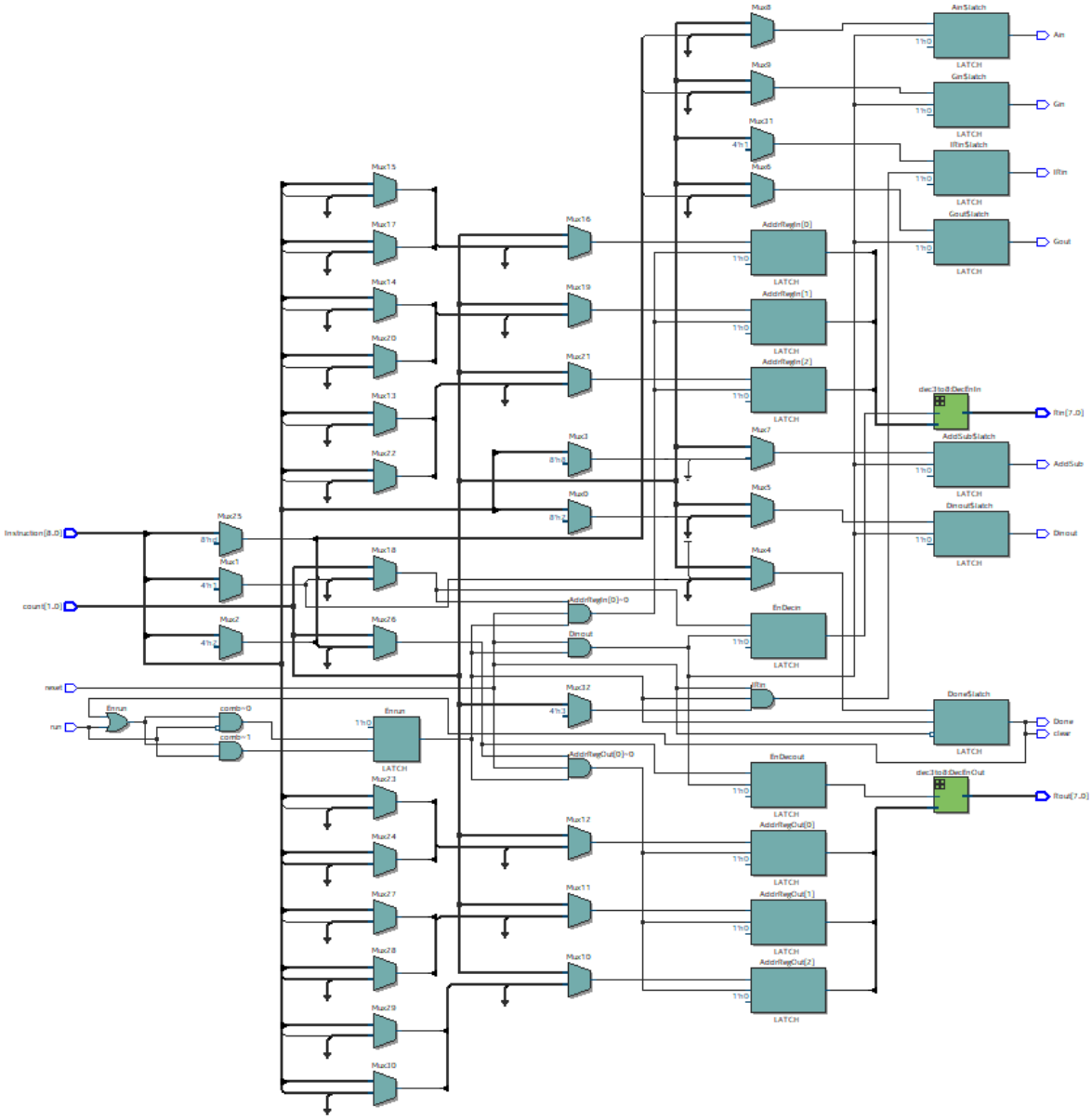
תוכנית Control Unit:

```
library ieee;
use ieee.std_logic_1164.all;

entity ControlUnit is
port(

Instruction: in std_logic_vector(8 downto 0);
run,reset: in std_logic;
count: in std_logic_vector(1 downto 0);

clear,Done: buffer std_logic;
Dinout,Gout,Gin,Ain,IRin,AddSub: out std_logic;
Rin:  out std_logic_vector(7 downto 0);
Rout: out std_logic_vector(7 downto 0));

end;

architecture one of ControlUnit is

component dec3to8
port(
w:  in std_logic_vector(2 downto 0);
En: in std_logic;
y:  out std_logic_vector(0 to 7));
end component;

signal Enrun: std_logic:='0';
signal EnDecin,EnDecout: std_logic;
signal AddrRegIn:  std_logic_vector(2 downto 0);
signal AddrRegOut: std_logic_vector(2 downto 0);

begin

DecEnIn:  dec3to8 port map (AddrRegIn,  EnDecin,  Rin);
DecEnOut: dec3to8 port map (AddrRegOut, EnDecout, Rout);

clear <= Done;

process (run,Done)
begin
      if (run = '1') then Enrun <='1';
      elsif (Done = '1') then Enrun <= '0';
      else Enrun <= Enrun; end if;
end process;



process(count,Enrun,reset)
begin

--AddrRegIn  <= Instruction(5 downto 3); -- RxIn
--AddrRegOut <= Instruction(2 downto 0); -- RyOut
--AddrRegIn  <= Instruction(2 downto 0); -- RyIn
--AddrRegOut <= Instruction(5 downto 3); -- RxOut
```

```
if (reset = '0') then Done <= '0';
elsif (Enrun = '1') then
case count is
when "00" => Done <= '0'; IRin <= '1'; -- Fetch
                    EnDecin <= '0'; EnDecout <= '0';
                    Dinout <= '0'; Gout <= '0';
                    AddSub <= '0'; Ain <= '0'; Gin <= '0';
when "01" => IRin <= '0'; -- T1
      case Instruction(8 downto 6) is
      when "000"  => EnDecin <= '1'; EnDecout <= '1';
                           Dinout <= '0'; Gout <= '0';
                           AddSub <= '0'; Ain <= '0'; Gin <= '0';
                           Done <= '1';
                           AddrRegOut <= Instruction (2 downto 0); AddrRegIn
<= Instruction (5 downto 3);
      when "001"  => EnDecin <= '1'; EnDecout <= '0';
                           Dinout <= '1'; Gout <= '0';
                           AddSub <= '0'; Ain <= '0'; Gin <= '0';
                           Done <= '1'; AddrRegIn  <= Instruction(5 downto
3);
      when "010"  => EnDecin <= '0'; EnDecout <= '1';
                           Dinout <= '0'; Gout <= '0';
                           AddSub <= '0'; Ain <= '1'; Gin <= '0';
                           Done <= '0'; AddrRegOut <= Instruction(5 downto
3);
      when "011"  => EnDecin <= '0'; EnDecout <= '1';
                           Dinout <= '0'; Gout <= '0';
                           AddSub <= '1'; Ain <= '1'; Gin <= '0';
                           Done <= '0'; AddrRegOut <= Instruction(5 downto
3);
      when others => EnDecin <= '0'; EnDecout <= '0';
                           Dinout <= '0'; Gout <= '0';
                           AddSub <= '0'; Ain <= '0'; Gin <= '0'; Done <=
'0';
      end case;
when "10" => -- T2
      case Instruction(8 downto 6) is
      when "010"  => EnDecin <= '0'; EnDecout <= '1';
                           Dinout <= '0'; Gout <= '0';
                           AddSub <= '0'; Ain <= '0'; Gin <= '1';
                           Done <= '0'; AddrRegOut <= Instruction(2 downto
0); -- Add
      when "011"  => EnDecin <= '0'; EnDecout <= '1';
                           Dinout <= '0'; Gout <= '0';
                           AddSub <= '1'; Ain <= '0'; Gin <= '1';
                           Done <= '0'; AddrRegOut <= Instruction(2 downto
0); -- Sub
      when others => EnDecin <= '0'; EnDecout <= '0';
                           Dinout <= '0'; Gout <= '0';
                           AddSub <= '0'; Ain <= '0'; Gin <= '0'; Done <=
'0';
      end case;
when others => Done <= '1'; -- T3
      case Instruction(8 downto 6) is
      when "010"  => EnDecin <= '1'; EnDecout <= '0';
                           Dinout <= '0'; Gout <= '1';
                           AddSub <= '0'; Ain <= '0'; Gin <= '0';
```

```
                        AddrRegIn <= Instruction(5 downto 3); -- Add
      when "011"  => EnDecin <= '1'; EnDecout <= '0';
                        Dinout <= '0'; Gout <= '1';
                        AddSub <= '1'; Ain <= '0'; Gin <= '0';
                        AddrRegIn <= Instruction(5 downto 3); -- Sub
      when others => EnDecin <= '0'; EnDecout <= '0';
                        Dinout <= '0'; Gout <= '0';
                        AddSub <= '0'; Ain <= '0'; Gin <= '0';
      end case;
end case;
end if;
end process;
end;
```

סכימת RTL:



שלב 3 – תכנון המיקרופרוסור על ידי חיבור כל ה-Components

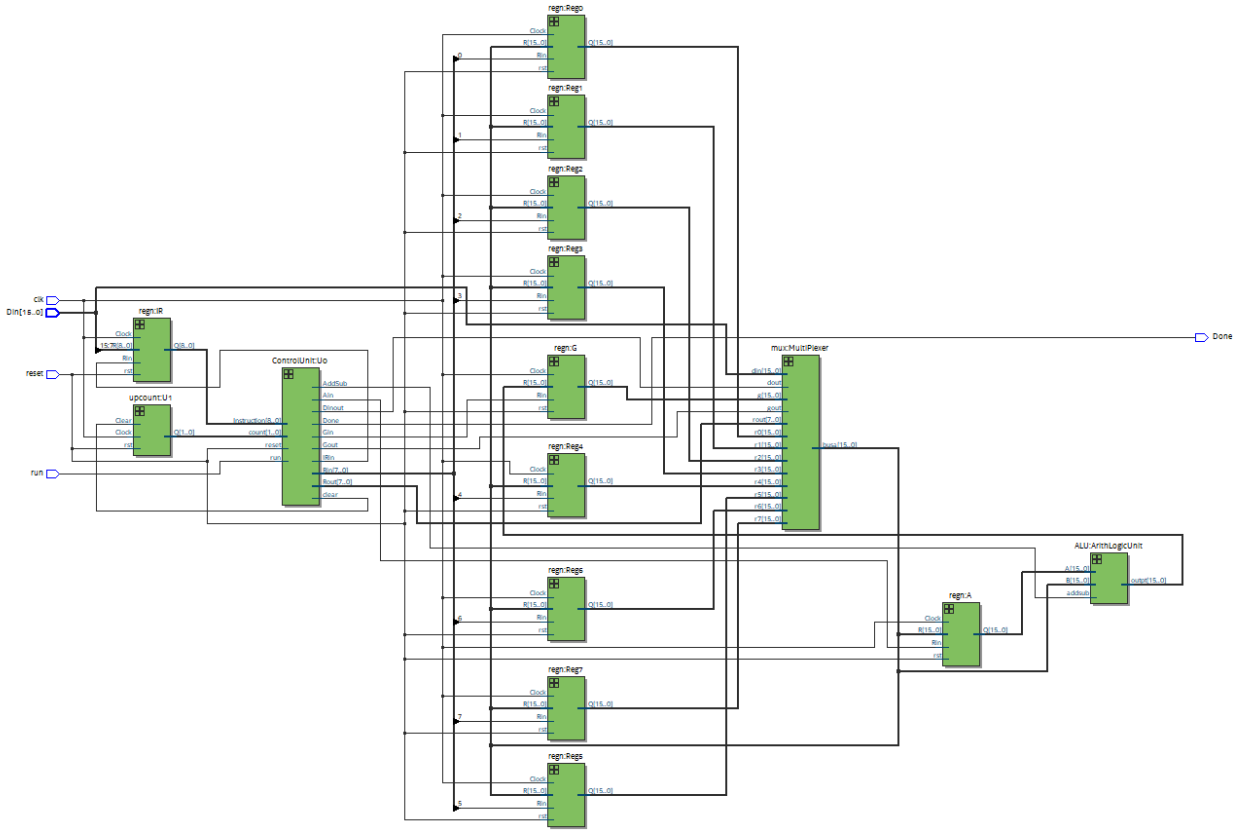
בשלב זה חיברנו את כל היחידות שפיתחנו.

תוכנית MPU:

```
library ieee;
use ieee.std_logic_1164.all;

entity MPU is
port(
clk,run,reset: in std_logic;
Din: in std_logic_vector(15 downto 0);
Done: out std_logic);
end;

architecture one of MPU is

-- / Components \ --

component regn
generic (n : INTEGER := 16);
port (
R: in STD_LOGIC_VECTOR(n-1 DOWNTO 0);
rst, Rin, Clock: in STD_LOGIC;
Q: buffer STD_LOGIC_VECTOR(n-1 DOWNTO 0));
end component;

component mux
port(
r0,r1,r2,r3,r4,r5,r6,r7,din,g : in STD_LOGIC_vector (15 downto 0);
gout,dout: in STD_LOGIC;
rout: in std_logic_vector (7 downto 0);
busa: out STD_LOGIC_vector (15 downto 0));
end component;

component ALU
port(
A,B: in std_logic_vector (15 downto 0);
addsub: in std_logic;
outpt: out std_logic_vector(15 downto 0));
end component;

component upcount
port(
Clear, Clock,rst: in std_logic;
Q: out std_Logic_vector(1 DOWNTO 0));
end component;

component ControlUnit
port(
Instruction: in std_logic_vector(8 downto 0);
run,reset: in std_logic;
count: in std_logic_vector(1 downto 0);

clear,Done: buffer std_logic;
```

```
Dinout,Gout,Gin,Ain,IRin,AddSub: out std_logic;
Rin:  out std_logic_vector(7 downto 0);
Rout: out std_logic_vector(7 downto 0));
end component;

-- / Components \ --

-- / Signals \ --
signal R0,R1,R2,R3,R4,R5,R6,R7: std_logic_vector(15 downto 0);
signal Gout: std_logic_vector(15 downto 0);
signal busa: std_logic_vector(15 downto 0);
signal EnReg: std_logic_vector(7 downto 0);
signal Ain,Gin,IRin: std_logic;
signal ALUout,ALUin: std_logic_vector(15 downto 0);
signal Instruction: std_logic_vector(8 downto 0);
signal sel: std_logic_vector(7 downto 0);
signal sg,sd: std_logic;
signal selALU: std_logic;
signal count: std_logic_vector(1 downto 0);
signal clrCount: std_logic;
-- / Signals \ --

begin
-- Rx:   regn generic map (16) port map (Datain,                    En,
clk, Q);
   Reg0: regn generic map (16) port map (busa,                      reset,
EnReg(0), clk, R0);
   Reg1: regn generic map (16) port map (busa,                      reset,
EnReg(1), clk, R1);
   Reg2: regn generic map (16) port map (busa,                      reset,
EnReg(2), clk, R2);
   Reg3: regn generic map (16) port map (busa,                      reset,
EnReg(3), clk, R3);
   Reg4: regn generic map (16) port map (busa,                      reset,
EnReg(4), clk, R4);
   Reg5: regn generic map (16) port map (busa,                      reset,
EnReg(5), clk, R5);
   Reg6: regn generic map (16) port map (busa,                      reset,
EnReg(6), clk, R6);
   Reg7: regn generic map (16) port map (busa,                      reset,
EnReg(7), clk, R7);
   G:    regn generic map (16) port map (ALUout,                    reset,
Gin,      clk, Gout);
      A:    regn generic map (16) port map (busa,            reset, Ain,
clk, ALUin);
      IR:   regn generic map (9)  port map (Din(15 downto 7), reset, IRin,
clk, Instruction);

U0: ControlUnit port map (Instruction, run,
reset,count,clrCount,Done,sd,sg,Gin,Ain,IRin,selALU,EnReg,sel);
U1: upcount port map (clrCount,clk,reset,count);

MultiPlexer: mux port map (R0,R1,R2,R3,R4,R5,R6,R7,Din,Gout,sg,sd,sel,busa);
ArithLogicUnit: ALU port map (ALUin,busa,selALU,ALUout);
end;
```

סכימת RTL:


regn:Reg0
regn:Reg1
regn:Reg2
regn:Reg3
regn:G
regn:Reg4
regn:Reg5
regn:Reg7
regn:Reg6
regn:IR
upcount:U1
ControlUnit:Uo
mux:MultiPlexer
ALU:ArithLogicUnit
regn:A
clk
Din[15..0]
reset
run
Done

שלב 4 – הרצת תוכנית לדוגמא לצורך בדיקה (סימולציה ב-ModelSim)

בשלב זה היה עלינו לכתוב תוכנית באסמבלר למיקרופרוסור הכוללת את כל הפקודות ולוודא שאכן הפקודות מתבצעות באופן תקין.

התוכנית שהיה עלינו להריץ:

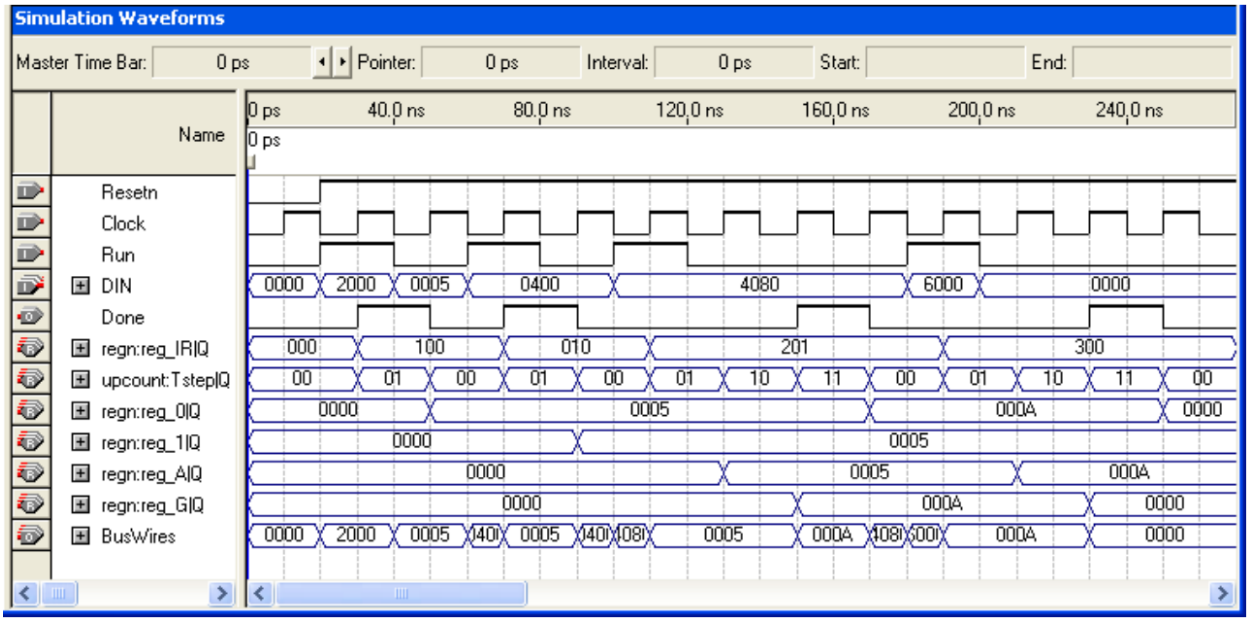
```
 # Instructions    | Encoding Data to Din
MVI R0,5            # DIN = 0x2000 , DIN = 5
MOV R1,R0           # DIN = 0x0400
ADD R0,R1           # DIN = 0x4080
SUB R0,R0           # DIN = 0x6000
```

כפי שניתן לראות התוכנית צריכה לבצע הכנסה של המספר 5 לרגיסטר R0 באופן מיידי.
לאחר מכן להכניס את אותו הערך (5) לאוגר R1 באמצעות העברה מאוגר לאוגר.
לאחר מכן לבצע פעולת חיבור ולהכניס את התוצאה לאוגר R0 (כלומר אוגר R0 יקבל את הערך 10).
ולאחר מכן לחסר את הערך של R0 עם עצמו (כלומר להכניס לתוכו את הערך 0).

כל הערכים בתיאור התוכנית הם ערכים שמיוצגים בבסיס 10.

בנוסף, יש לשים לב שהפקודה הראשונה מורכבת מ-2 נתונים שעוברים על הSignal DIN במהלך ביצוע הפקודה תחילה עובר עליו הפקודה עצמה ולאחר שלב ה- Fetch צריך לעבור הערך 5.

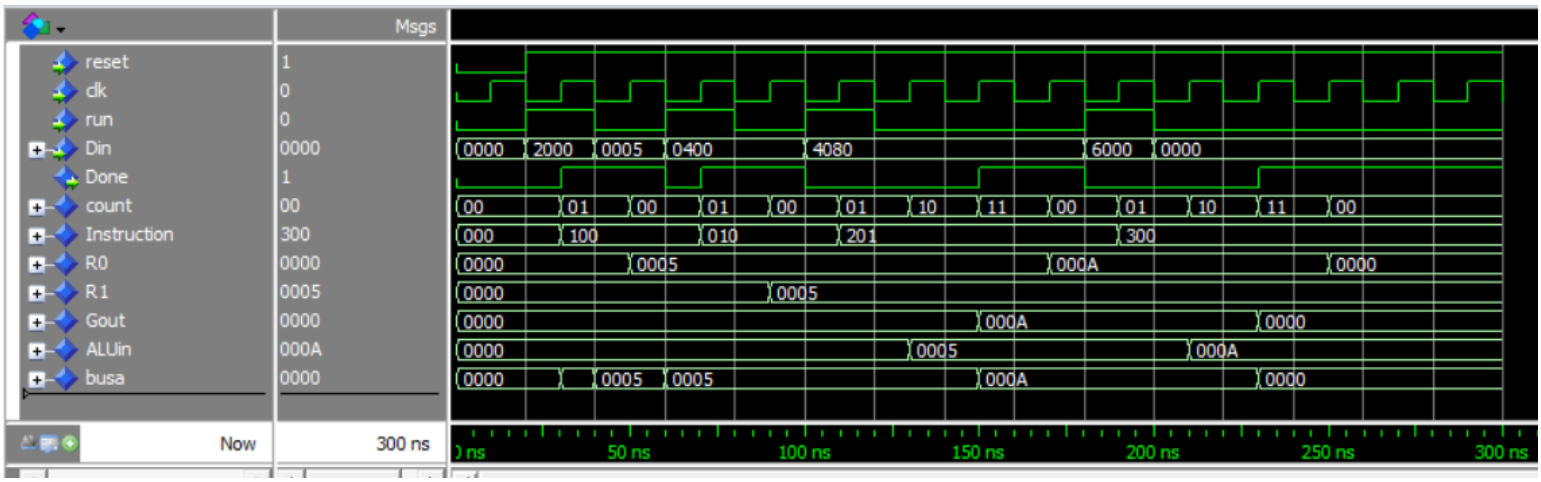
מטרתנו היא לראות שאנו מקבלים תוצאה דומה למה שהתקבל בקובץ לדוגמא (Figure 3).



לכן נכניס לScript את הפקודות הבאות:

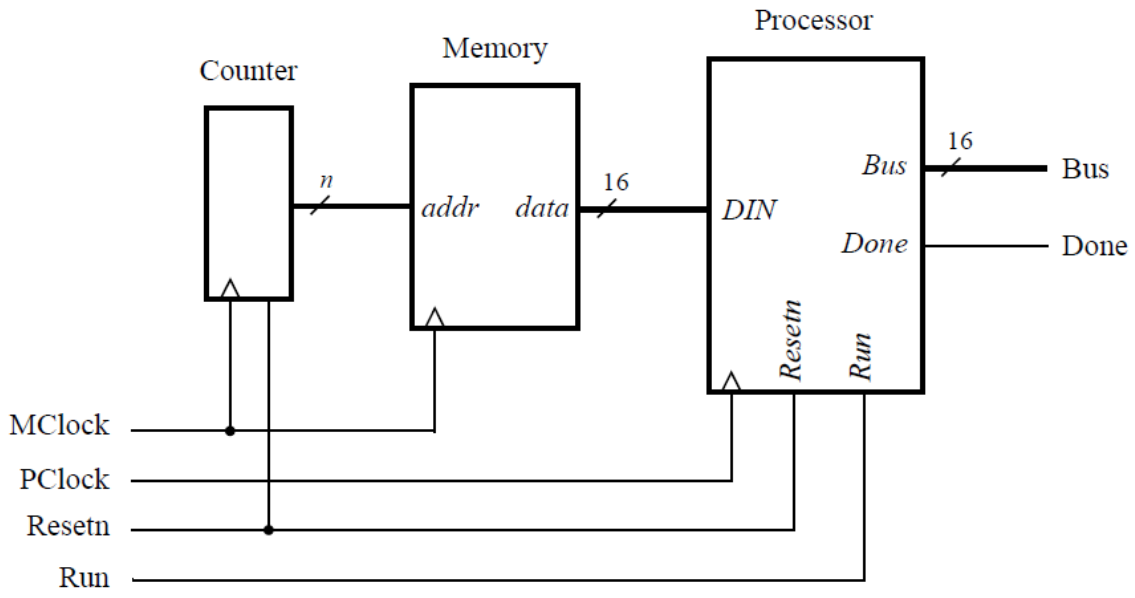
1. הגדרת clk (זמן מחזור 20[nsec] ו- DC=50[%] השינוי הראשון מסוג ירידה שעון).
2. Force reset 0, 1 20ns
3. Force run 0, 1 20ns, 0 40ns, 1 60ns, 0 80ns, 1 100ns, 0 120ns, 1 180ns, 0 200ns
4. Force Din 0000000000000000, 0010000000000000 20ns, 0000000000000101 40ns, 0000010000000000 60ns, 0100000010000000 100ns, 0110000000000000 180ns, 0000000000000000 200ns
5. Run 300ns

תמונה המציגה את התוצאות שהתקבלו:



ניתן לראות שקיבלנו את אותם תוצאות שציפינו לקבל שהם גם זהות לתוצאות שהתקבלו בסימולציה.

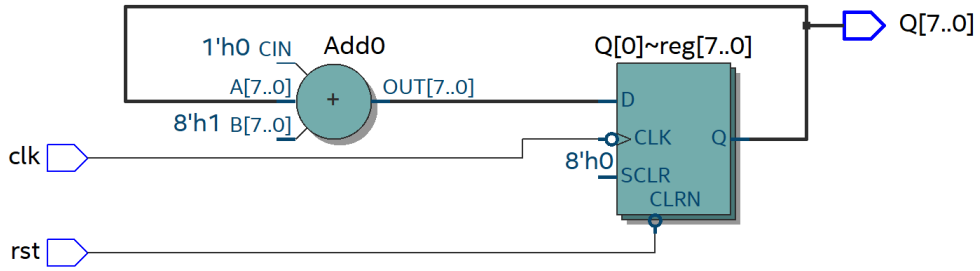
כעת נדרשנו להוסיף זיכרון שעליה צרובה התוכנית על סמך הארכיטקטורה:



תוכנית ליחידה Counter:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity counter is
generic(N:integer:=8);
port(
rst,clk: in std_logic;
Q: buffer std_logic_vector(N-1 downto 0));
end;

architecture one of counter is
begin
process(rst,clk)
begin

if (rst = '0') then Q <= (others => '0');
elsif (clk 'event and clk = '0') then Q <= Q+1; end if;

end process;
end;
```

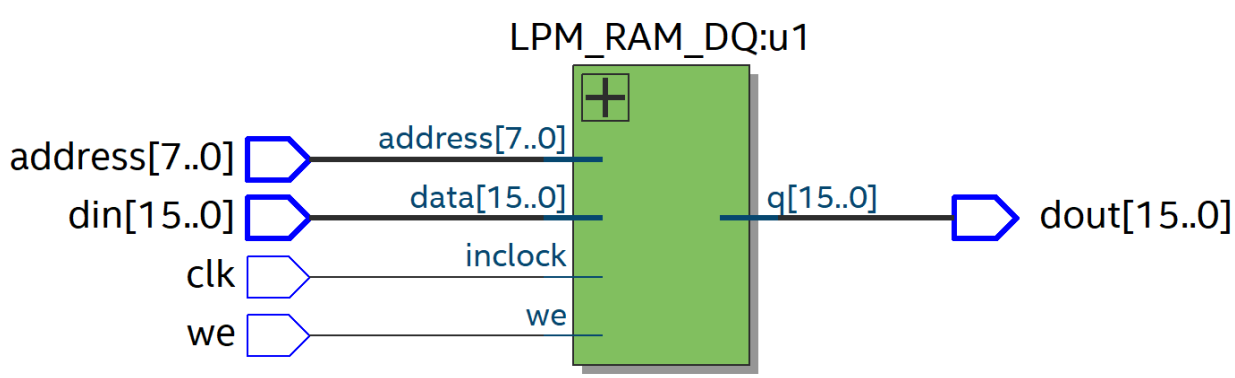
סכימת RTL:



תוכנית ליחידת הזיכרון (Memory):

```
library ieee;
library lpm;

use ieee.std_logic_1164.all;
use lpm.lpm_components.all;


entity Memory is
port( address : in std_logic_vector (7 downto 0);
we, clk : in std_logic;
din : in std_logic_vector (15 downto 0);
dout : out std_logic_vector (15 downto 0));
end;

architecture arc of Memory is begin

-- lpm_ram_dq : separate input/output data ; lpm_ram_io : common input/output
data
-- REGISTERED/UNREGISTERED : DATA+CONTROL INPUTS -
--SYNCHRONIZED (default)/NOT SYNCH to inclock
-- MEMORY FILES : hex, mif - memory initialization file

u1: lpm_ram_dq generic map (lpm_width => 16, lpm_widthad => 8, lpm_outdata =>
"UNREGISTERED", lpm_file => "Program.mif")
port map (data => din, address => address, we => we, inclock => clk, q =>
dout);
end;
```

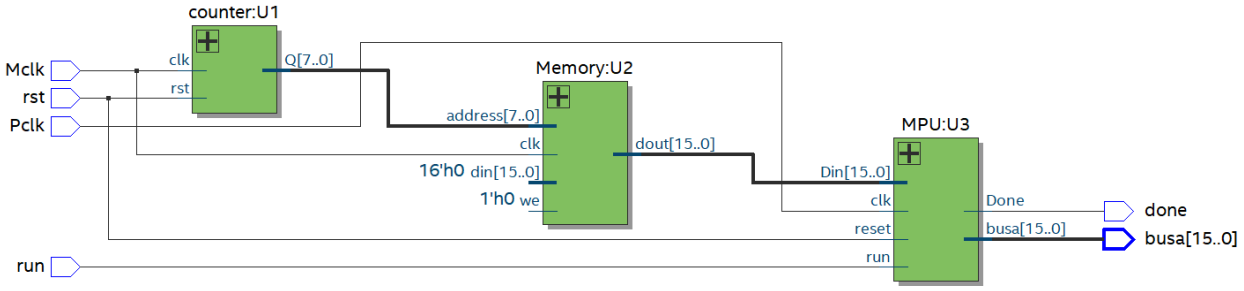
סכימת RTL:



התוכנית הכוללת של כל המערכת:

```
library ieee;
use ieee.std_logic_1164.all;

entity system1 is
port(
Mclk,Pclk,rst,run: in std_logic;
busa: out std_logic_vector(15 downto 0);
done: out std_logic);
end;

architecture one of system1 is

-- / Components \ --

component mpu
port(
clk,run,reset: in std_logic;
Din: in std_logic_vector(15 downto 0);
Done: out std_logic;
busa: out std_logic_vector(15 downto 0));
end component;

component counter
generic(N:integer:=8);
port(
rst,clk: in std_logic;
Q: buffer std_logic_vector(N-1 downto 0));
end component;

component Memory
port(
address : in std_logic_vector (7 downto 0);
we, clk : in std_logic;
din : in std_logic_vector (15 downto 0);
dout : out std_logic_vector (15 downto 0));
end component;

-- / Components \ --

-- / Signals \ --

signal Addr: std_logic_vector(7 downto 0);
signal DataIn: std_logic_vector(15 downto 0);

-- / Signals \ --

begin
U1: counter generic map (8) port map (rst,Mclk,Addr);
U2: Memory port map (Addr,'0',Mclk,(others => '0'),DataIn);
U3: MPU port map (Pclk,run,rst,DataIn,done,busa);
end;
```

סכימת RTL:


counter:U1
Mclk
rst
Pclk
clk
rst
Q[7..0]
Memory:U2
address[7..0]
clk
16'h0 din[15..0]
1'h0 we
dout[15..0]
MPU:U3
Din[15..0]
clk
reset
run
Done
busa[15..0]
done
busa[15..0]
run