המכללה האקדמית להנדסה בראודה בכרמיאל

המחלקה להנדסת חשמל ואלקטרוניקה

התקנים לוגיים מתוכנתים (31551)

פרויקט מסכם
Matrics Muliply

מנחה: ד"ר פאדל טריף

תאריך : 30/08/2024

תוכן ענינים

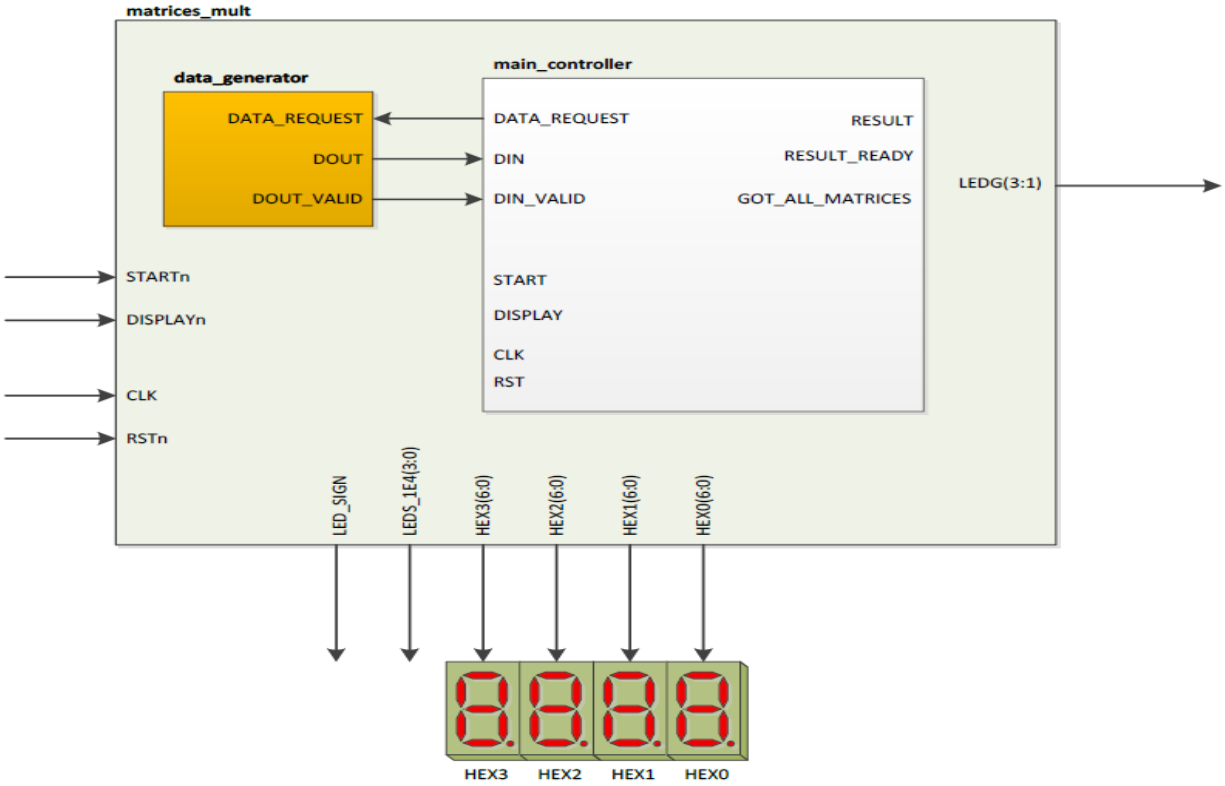
1) תיאור כללי

יש לתכנן מערכת שתקלוט שתי מטריצות בגודל 4×4 כל אחת, תחבר את תוצאת המכפלה שלהן, ותציג את התוצאה על גבי תצוגת segment-7. המערכת תשתמש במעבד Cyclone V Starter Kit ותקבל קלטים מהמשתמש באמצעות כפתורי לחיצה. המערכת תתחיל במצב המתנה, ותפעל כאשר מתקבלות שתי המטריצות. בסיום חישוב המכפלה, התוצאה תוצג על גבי התצוגה.

2) סכימת בלוקים כללית

המערכת מורכבת ממספר בלוקים עיקריים:

1. **data_generator:** אחראי על יצירת המטריצות לכניסה. בלוק זה יספק את המטריצות לחישוב.
2. **main_controller:** הבלוק הראשי של המערכת שמנהל את כל התהליך – קבלת המטריצות, חישוב התוצאה, ורישום התוצאה בזיכרון.
3. **matrices_mult:** אחראי על חישוב המכפלה בין שתי המטריצות.
4. **DISPLAY:** תציג את התוצאה על גבי תצוגת ה-segment-7.



איור 1: סכימת בלוקים כללית

הבלוק הכתום יספק את המטריצות עליהן יבוצע החישוב. בלוק זה יסופק על ידינו ויהיה חלק מהקושחה הכוללת.

שימו לב: הבלוק main_controller הוא הבלוק הראשי אך לא היחיד. כל השאר לא מופיעים באיור זה.

3) הגדרת הדרישות

- **שעון המערכת:** המערכת תפעל על שעון יחיד של MHz50.
- **RESET:** הכניסה היא active low, כלומר היא מופעלת כאשר הערך בכניסה הוא '0'.
- **מצב המתנה:** במצב המתנה כל ה-LEDs ותצוגת ה-seg7 יהיו כבויים למעט LEDG(1) שיהיה דולק לאורך כל זמן פעולת המערכת.
- **קבלת מטריצות:** לחיצה ראשונה על לחצן STARTn תגרום ל-data_generator להוציא את שתי המטריצות. סדר קבלת האיברים לכל מטריצה הוא משמאל לימין ומלמעלה למטה.
- **רישום בזיכרון:** המטריצות ירשמו לזיכרון בסדר שנקבע ע"י המשתמש.
- **חישוב מכפלה:** לאחר קבלת המטריצות, המערכת תבצע חישוב מכפלת המטריצות ותשמור את התוצאה בזיכרון.
- **מצב הצגה:** לאחר חישוב התוצאה, המערכת תעבור למצב הצגה. תוצג כל פעם ספרה אחת של התוצאה על גבי תצוגת ה-segment-7.
- **הצגה בלולאה:** לאחר הצגת כל האיברים של התוצאה, לחיצה נוספת על לחצן DISPLAYn תציג שוב את האיבר הראשון בלולאה.
- **התחלה מחדש:** לחיצה על הלחצן STARTn תחזיר את המערכת למצב המתנה.
- **יעילות ואופטימיזציה:** ינתן דגש על זמן החישוב של מטריצת היעד וכמות המכפלים בשימוש. יש למצוא את האיזון הנכון בין השניים.

4) ציוד נדרש

- **ערכת Cyclone V Starter Kit.**
- **מחשב הכולל תוכנות modelsim, notepad++ ו-quartus.**

5) הנחיות כלליות

- **כתיבת הקוד:** הקוד יכתב ב-VHDL ויש לכלול testbench לכל בלוק.
- **סימולציה מלאה:** יש לבצע סימולציה מלאה לכל המערכת.
- **הגשת קוד אישי:** כל הקוד שיוגש חייב להיות אישי וללא שימוש בקוד מוכן ממקורות אחרים.
- **אופטימיזציה:** יש להקפיד על כתיבה מבנית, שימוש בסינכרוניזרים, ושמירה על כללי הכתיבה הנכונה לסינתיזה.
- **שמות קבועים:** חובה לשמור על שמות הקבצים, שמות ה-entities, שמות ה-ports וה-generics המוגדרים במסמך זה.

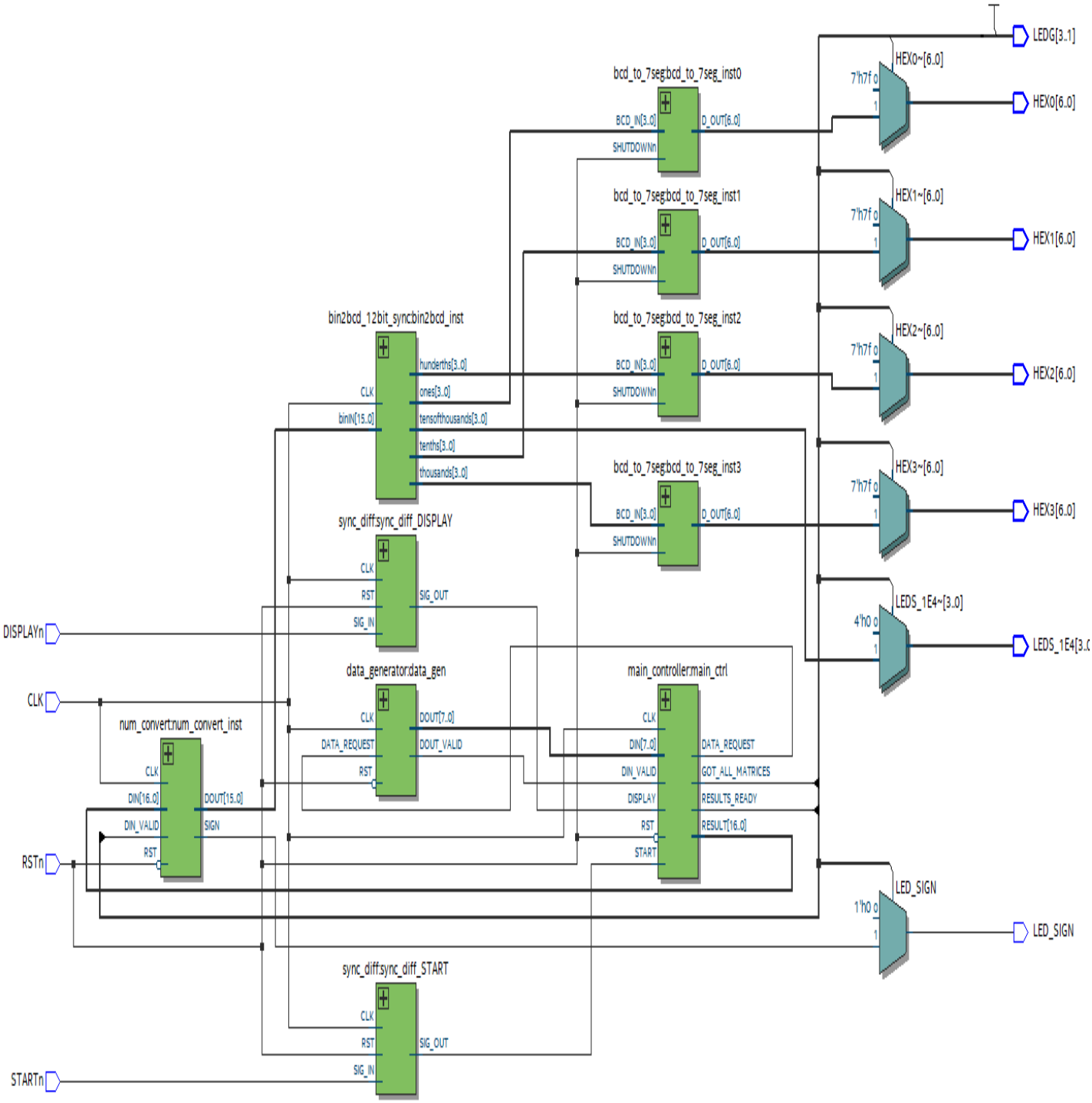
6) תכן מפורט

1. **תיאור הבלוק matrices_mult:**
   - זהו הבלוק העליון של המערכת, שמחבר בין כל הבלוקים האחרים למטרת הכפלת מטריצות והצגת התוצאות. הבלוק מקבל אותות שעון ואיפוס, ומנהל את זרימת הנתונים בין הבלוקים השונים כמו data_generator, main_controller, ו-num_convert. כמו כן, הוא דואג להמרת התוצאה הבינארית לפורמט BCD ולהצגת התוצאה על גבי תצוגות segment-7 ו-LEDים. הבלוק הזה מתאם את כל התהליכים במערכת כדי להבטיח ביצועים נכונים ותוצאות מדויקות.
2. **תיאור הבלוק main_controller:**
   - בלוק זה הוא הבלוק הראשי של המערכת שמנהל את כל התהליך – קבלת המטריצות, חישוב התוצאה, ורישום התוצאה בזיכרון.
3. **תיאור הבלוק data_generator:**
   - בלוק זה אחראי על ייצור המטריצות לכניסה. הבלוק יוציא את המטריצות לפי דרישת המשתמש ויתעלם מפקודות בזמן הוצאת נתונים.
4. **תיאור הבלוק my_multiplier:**
   - בלוק זה מכפיל שני מספרים (signed/unsigned) ומציג את התוצאה לאחר מספר מחזורי שעון שנקבע מראש.
5. **תיאור הבלוק bcd_to_7seg:**
   - הבלוק אחראי על המרת BCD לפורמט המתאים לתצוגת segment-7.
6. **תיאור הבלוק bin2bcd_12bit_sync:**
   - בלוק זה ממיר מספר בינארי למספר BCD באורך של עד 12 ביטים.
7. **תיאור הבלוק matrix_ram:**
   - בלוק זה אחראי על רישום וקריאה של המטריצות והתוצאה מהזיכרון.
8. **תיאור הבלוק num_convert:**
   - הבלוק אחראי על המרת מספרים לא חתומים (unsigned) למספרים חתומים (signed) ולהיפך.
9. **תיאור הבלוק sync_diff :**
   - בלוק זה אחראי על סנכרון אותות אסינכרוניים עם השעון, על מנת למנוע בעיות של metastability במערכת. הבלוק מבטיח שהאות הנכנס יעבור סינכרון ויהיה זמין בצורה יציבה לשימוש בשאר חלקי המערכת.

7) תיאור משימות

משימה 1: מימוש הרמה העליונה matrices_mult

2.1.1 תכנון הבלוק matrices_mult



6.1.20 מימוש הרמה העליונה

איור 8 : הבלוק של הרמה העליונה

טבלה 8: תיאור הבלוק matrices_mult

Generics			
Name	**Dir.**	**Type**	**Description**
Ports			
Name	**Dir.**	**type**	**Description**
System Signals			
CLK	I	std_logic	System clock
RSTn	I	std_logic	Active low system reset. Connect to KEY0 on EVB.
STARTn	I	std_logic	0 – KEY pressed 1 – KEY not pressed
DISPLAYn	I	std_logic	0 – KEY pressed 1 – KEY not pressed
HEX0	O	std_logic_vector (6:0)	0 will turn on the relevant segment ספרת האחדות
HEX1	O	std_logic_vector (6:0)	0 will turn on the relevant segment ספרת העשרות
HEX2	O	std_logic_vector (6:0)	0 will turn on the relevant segment ספרת המאות
HEX3	O	std_logic_vector (6:0)	0 will turn on the relevant segment ספרת האלפים
LEDS_1E4	O	std_logic_vector (3:0)	יצוג בינארי של ספרת עשרות האלפים 1 will turn on the relevant LED
LED_SIGN	O	std_logic	1 will turn on the relevant LED
LEDG	O	std_logic_vector (3:1)	1 will turn on the relevant LED

מבנה כללי:

היישות matrices_mult מקבלת מספר אותות כניסה, כולל שעון (**CLK**), אות איפוס (**RSTn**), אות התחלה (**STARTn**), ואות תצוגה (**DISPLAYn**). היא מבצעת את הכפל של שתי מטריצות ומציגה את התוצאה באמצעות תצוגות **7-segment** ונורות **LED**.

2.1.2 סימולציה פונקציונאלית של הבלוק matrices_mult

❖ matrices_mult code:

```
library ieee; -- Import the IEEE library for standard logic and arithmetic operations
use ieee.std_logic_1164.all; -- Use the IEEE standard logic package for logic operations
use ieee.numeric_std.all; -- Use the IEEE numeric standard package for arithmetic operations

entity matrices_mult is
    port (
        CLK          : in std_logic; -- Input clock signal
        RSTn         : in std_logic; -- Active-low reset signal
        STARTn       : in std_logic; -- Active-low start signal
        DISPLAYn     : in std_logic; -- Active-low display signal
        HEX0         : out std_logic_vector(6 downto 0); -- 7-segment display output for lower digit
        HEX1         : out std_logic_vector(6 downto 0); -- 7-segment display output for the next digit
        HEX2         : out std_logic_vector(6 downto 0); -- 7-segment display output for the third digit
        HEX3         : out std_logic_vector(6 downto 0); -- 7-segment display output for the fourth digit
        LEDS_1E4     : out std_logic_vector(3 downto 0); -- Output for LEDs showing thousands place
        LED_SIGN     : out std_logic; -- Output for the sign LED
        LEDG         : out std_logic_vector(3 downto 1) -- Output for additional green LEDs
    );
end entity matrices_mult;

architecture structural of matrices_mult is

    -- Internal signals for communication between data_generator and main_controller components
    signal data_request : std_logic; -- Signal to request new data
    signal din          : std_logic_vector(7 downto 0); -- Data input from data_generator to main_controller
    signal din_valid    : std_logic; -- Validity flag for the data input
    signal result       : std_logic_vector(16 downto 0); -- Result output from main_controller
    signal result_ready : std_logic; -- Flag indicating the result is ready
    signal got_all_matrices : std_logic; -- Flag indicating all matrices have been processed

    -- Signals for converting binary result to BCD and driving 7-segment displays
    signal bcd_ones          : std_logic_vector(3 downto 0) := (others => '0'); -- BCD value for ones place
    signal bcd_tenths        : std_logic_vector(3 downto 0) := (others => '0'); -- BCD value for tens place
    signal bcd_hundreds      : std_logic_vector(3 downto 0) := (others => '0'); -- BCD value for hundreds place
    signal bcd_thousands     : std_logic_vector(3 downto 0) := (others => '0'); -- BCD value for thousands place
    signal bcd_tens_of_thousands : std_logic_vector(3 downto 0) := (others => '0'); -- BCD value for tens of thousands place
    signal display_hex0      : std_logic_vector(6 downto 0) := (others => '1'); -- Default off state for HEX0
    signal display_hex1      : std_logic_vector(6 downto 0) := (others => '1'); -- Default off state for HEX1
    signal display_hex2      : std_logic_vector(6 downto 0) := (others => '1'); -- Default off state for HEX2
    signal display_hex3      : std_logic_vector(6 downto 0) := (others => '1'); -- Default off state for HEX3

    -- Additional internal signals
    signal sign_out          : std_logic; -- Signal indicating the sign of the result
    signal dout              : std_logic_vector(15 downto 0);    Processed data output

    -- Synchronization signals for START and DISPLAY
    signal START_SIG         : std_logic; -- Synchronized start signal
    signal DISPLAY_SIG       : std_logic; -- Synchronized display signal

    -- Component declarations

    component data_generator
        port (
            CLK          : in std_logic; -- Input clock signal
            RST          : in std_logic; -- Reset signal
            DATA_REQUEST: in std_logic; -- Signal to request data generation
            DOUT         : out std_logic_vector(7 downto 0); -- Generated data output
            DOUT_VALID   : out std_logic -- Validity flag for the generated data
        );
    end component;

    component main_controller
        port (
            CLK               : in std_logic; -- Input clock signal
            RST               : in std_logic; -- Reset signal
            START             : in std_logic; -- Start signal
            DISPLAY           : in std_logic; -- Display control signal
            DIN               : in std_logic_vector(7 downto 0); -- Data input
            DIN_VALID         : in std_logic; -- Validity flag for the data input
            DATA_REQUEST      : out std_logic; -- Signal to request more data
            RESULT            : out std_logic_vector(16 downto 0); -- Computed result output
            RESULTS_READY     : out std_logic; -- Signal indicating the result is ready
            GOT_ALL_MATRICES: out std_logic -- Signal indicating all matrices have been processed
        );
    end component;

    component bin2bcd_12bit_sync
        port (
            CLK               : in std_logic; -- Input clock signal
            binIN             : in std_logic_vector(15 downto 0); -- Binary input value
            ones              : out std_logic_vector(3 downto 0); -- BCD output for ones place
            tenths            : out std_logic_vector(3 downto 0); -- BCD output for tens place
            hunderths         : out std_logic_vector(3 downto 0); -- BCD output for hundreds place
            thousands         : out std_logic_vector(3 downto 0); -- BCD output for thousands place
            tensofthousands : out std_logic_vector(3 downto 0) -- BCD output for tens of thousands place
        );
    end component;

    component bcd_to_7seg
        port (
            BCD_IN      : in std_logic_vector(3 downto 0); -- BCD input value
            SHUTDOWNn   : in std_logic; -- Shutdown control signal, active low
            D_OUT       : out std_logic_vector(6 downto 0) -- 7-segment display output
        );
    end component;

    component sync_diff
        generic (
            G_DERIVATE_RISING_EDGE   : boolean := true; -- Enable/disable edge detection
            G_SIG_IN_INIT_VALUE      : std_logic := '0'; -- Initial value of the input signal
            G_RESET_ACTIVE_VALUE     : std_logic := '0' -- Reset active value
        );
        port (
            CLK      : in std_logic; -- Input clock signal
            RST      : in std_logic; -- Reset signal
            SIG_IN   : in std_logic; -- Input signal to be synchronized
            SIG_OUT  : out std_logic -- Synchronized output signal
        );
    end component;

    component num_convert
        port (
            CLK        : in std_logic; -- Input clock signal
            RST        : in std_logic; -- Reset signal
            DIN        : in std_logic_vector(16 downto 0); -- Input data value
            DIN_VALID  : in std_logic; -- Validity flag for input data
            DOUT       : out std_logic_vector(15 downto 0); -- Output data value
            SIGN       : out std_logic -- Output sign indicator
        );
    end component;

begin
```

```vhdl
    -- Instantiate the data_generator component
    data_gen: data_generator
        port map (
            CLK          => CLK, -- Connect clock signal
            RST          => not RSTn, -- Connect inverted reset signal
            DATA_REQUEST=> data_request, -- Connect data request signal
            DOUT         => din, -- Connect data output to din signal
            DOUT_VALID   => din_valid -- Connect data valid output to din_valid signal
        );

    -- Instantiate the main_controller component
    main_ctrl: main_controller
        port map (
            CLK              => CLK, -- Connect clock signal
            RST              => not RSTn, -- Connect inverted reset signal
            START            => START_SIG, -- Connect synchronized start signal
            DISPLAY          => DISPLAY_SIG, -- Connect synchronized display signal
            DATA_REQUEST     => data_request, -- Connect data request signal
            DIN              => din, -- Connect data input
            DIN_VALID        => din_valid, -- Connect data valid input
            RESULT           => result, -- Connect result output
            RESULTS_READY    => result_ready, -- Connect result ready flag
            GOT_ALL_MATRICES=> got_all_matrices -- Connect all matrices processed flag
        );

    -- Instantiate the num_convert component to get the absolute value of the result
    num_convert_inst: num_convert
        port map (
            CLK          => CLK, -- Connect clock signal
            RST          => not RSTn, -- Connect inverted reset signal
            DIN          => result, -- Pass the result to num_convert for processing
            DIN_VALID    => result_ready, -- Connect result ready flag
            DOUT         => dout, -- Connect processed data output
            SIGN         => sign_out -- Connect sign output
        );

    -- Instantiate the binary to BCD converter
    bin2bcd_inst: bin2bcd_12bit_sync
        port map (
            binIN            => dout, -- Use the positive result for BCD conversion
            ones             => bcd_ones, -- Connect BCD ones output
            tenths           => bcd_tenths, -- Connect BCD tens output
            hunderths        => bcd_hundreds, -- Connect BCD hundreds output
            thousands        => bcd_thousands, -- Connect BCD thousands output
            tensofthousands => bcd_tens_of_thousands, -- Connect BCD tens of thousands output
            CLK              => CLK -- Connect clock signal
        );

    -- Instantiate the BCD to 7-segment display converters
    bcd_to_7seg_inst0: bcd_to_7seg
        port map (
            BCD_IN => bcd_ones, -- Connect BCD ones input
            SHUTDOWNn => RSTn, -- Connect reset signal
            D_OUT => display_hex0 -- Connect 7-segment display output for HEX0
        );

    bcd_to_7seg_inst1: bcd_to_7seg
        port map (
            BCD_IN => bcd_tenths, -- Connect BCD tens input
            SHUTDOWNn => RSTn, -- Connect reset signal
            D_OUT => display_hex1 -- Connect 7-segment display output for HEX1
        );

    bcd_to_7seg_inst2: bcd_to_7seg
        port map (
            BCD_IN => bcd_hundreds, -- Connect BCD hundreds input
            SHUTDOWNn => RSTn, -- Connect reset signal
            D_OUT => display_hex2 -- Connect 7-segment display output for HEX2
        );

    bcd_to_7seg_inst3: bcd_to_7seg
        port map (
            BCD_IN => bcd_thousands, -- Connect BCD thousands input
            SHUTDOWNn => RSTn, -- Connect reset signal
            D_OUT => display_hex3 -- Connect 7-segment display output for HEX3
        );

    -- Synchronize the START signal using sync_diff component
    sync_diff_START: sync_diff
        generic map (
            G_DERIVATE_RISING_EDGE   => false, -- Disable edge detection
            G_SIG_IN_INIT_VALUE      => '0', -- Initial value for the input signal
            G_RESET_ACTIVE_VALUE     => '0' -- Reset active value
        )
        port map (
            CLK      => CLK, -- Connect clock signal
            RST      => RSTn, -- Connect reset signal
            SIG_IN   => STARTn, -- Connect raw STARTn input signal
            SIG_OUT  => START_SIG -- Output the synchronized START signal
        );

    -- Synchronize the DISPLAY signal using sync_diff component
    sync_diff_DISPLAY: sync_diff
        generic map (
            G_DERIVATE_RISING_EDGE   => false, -- Disable edge detection
            G_SIG_IN_INIT_VALUE      => '0', -- Initial value for the input signal
            G_RESET_ACTIVE_VALUE     => '0' -- Reset active value
        )
        port map (
            CLK      => CLK, -- Connect clock signal
            RST      => RSTn, -- Connect reset signal
            SIG_IN   => DISPLAYn, -- Connect raw DISPLAYn input signal
            SIG_OUT  => DISPLAY_SIG -- Output the synchronized DISPLAY signal
        );

    -- Connect the 7-segment display outputs, enabling them when result is ready
    HEX0 <= display_hex0 when result_ready else (others=>'1'); -- Show HEX0 when result is ready, otherwise turn off
    HEX1 <= display_hex1 when result_ready else (others=>'1'); -- Show HEX1 when result is ready, otherwise turn off
    HEX2 <= display_hex2 when result_ready else (others=>'1'); -- Show HEX2 when result is ready, otherwise turn off
    HEX3 <= display_hex3 when result_ready else (others=>'1'); -- Show HEX3 when result is ready, otherwise turn off

    -- Control the LEDs based on the result readiness and sign
    LEDS_1E4 <= bcd_tens_of_thousands when result_ready else (others=>'0'); -- Show the BCD tens of thousands on LEDs
    LED_SIGN <= sign_out when result_ready else '0'; -- Show the sign of the result on the sign LED
    LEDG(3) <= result_ready; -- Light up the third green LED when result is ready
    LEDG(2) <= got_all_matrices; -- Light up the second green LED when all matrices are processed
    LEDG(1) <= '1'; -- Constantly light up the first green LED

end architecture structural;
```

❖ אותות ומסרים פנימיים:

1. **data_request, din, din_valid:** משמשים לתקשורת בין היישות המרכזית (**main_controller**) ליחידת ייצור הנתונים (**data_generator**). היישות המרכזית מבקשת נתונים, מקבלת אותם, ומסמנת האם הם תקפים.
2. **result, result_ready, got_all_matrices:** אותות שמעידים על קבלת התוצאה, מוכנות התוצאה וסיום תהליך קבלת המטריצות בהתאמה.

❖ הסבר על רכיבי המערכת:

1. **data_generator:** אחראי על יצירת הנתונים לשימוש ביישות המרכזית. הוא יוצר את המטריצות שצריך להכפיל.
2. **main_controller**: היישות המרכזית שמנהלת את כל תהליך הכפל. היא מקבלת את הנתונים מהמטריצות, מבצעת את החישובים, ושומרת את התוצאה.
3. **num_convert:** ממיר את התוצאה הבינארית לפורמט שבו ניתן להציג על תצוגת segment-7, כולל קבלת הערך המוחלט במקרה של מספרים שליליים.
4. **bin2bcd_12bit_sync:** ממיר את התוצאה הבינארית לפורמט **BCD** כדי שניתן יהיה להציג אותה על תצוגות **segment-7**.
5. **bcd_to_7seg**: ממיר כל ספרת **BCD** לקוד תצוגת **segment-7** המתאים.
6. **sync_diff**: רכיב שמשתמש לסנכרון האותות האסינכרוניים כמו **START** ו-**DISPLAY**, על מנת למנוע בעיות של אי יציבות במערכת (metastability).

❖ תהליך הפעולה:

1. **סנכרון אותות:** האותות **STARTn** ו-**DISPLAYn** מסונכרנים באמצעות הרכיב **sync_diff** כדי להבטיח שהמערכת תקבל את האותות בצורה יציבה.
2. **יצירת נתונים:** היישות המרכזית שולחת בקשה ליחידת ייצור הנתונים (**data_generato**r) כדי לקבל את המטריצות שהיא צריכה להכפיל.
3. **כפל מטריצות:** היישות המרכזית (**main_controller**) מבצעת את הכפל של המטריצות שהתקבלו ושומרת את התוצאה.
4. **המרת תוצאה לפורמט תצוגה:** לאחר קבלת התוצאה, היא מומרת לפורמט **BCD** באמצעות **bin2bcd_12bit_sync**, ולאחר מכן מומרת לקוד תצוגת **segment-7** באמצעות **bcd_to_7seg**.
5. **הצגת התוצאה:** התוצאה מוצגת על ארבע תצוגות **segment-7** ועל נורות **LED** שמציגות את הספרה הראשונה (אלפי) ואת סימן המספר (חיובי או שלילי).

❖ שליטה בתצוגה ובנורות LED:

1. **HEX0-HEX3:** מציגים את הספרות השונות של התוצאה על תצוגות **segment-7**.
2. **LEDS_1E4:** מציג את הספרה במיקום האלפי.
3. **LED_SIGN:** מציג את הסימן של התוצאה (חיובי או שלילי).
4. **LEDG:** נורות ירוקות שמסמנות שהמערכת מוכנה עם התוצאה, ושהמטריצות התקבלו.

❖ סיכום:

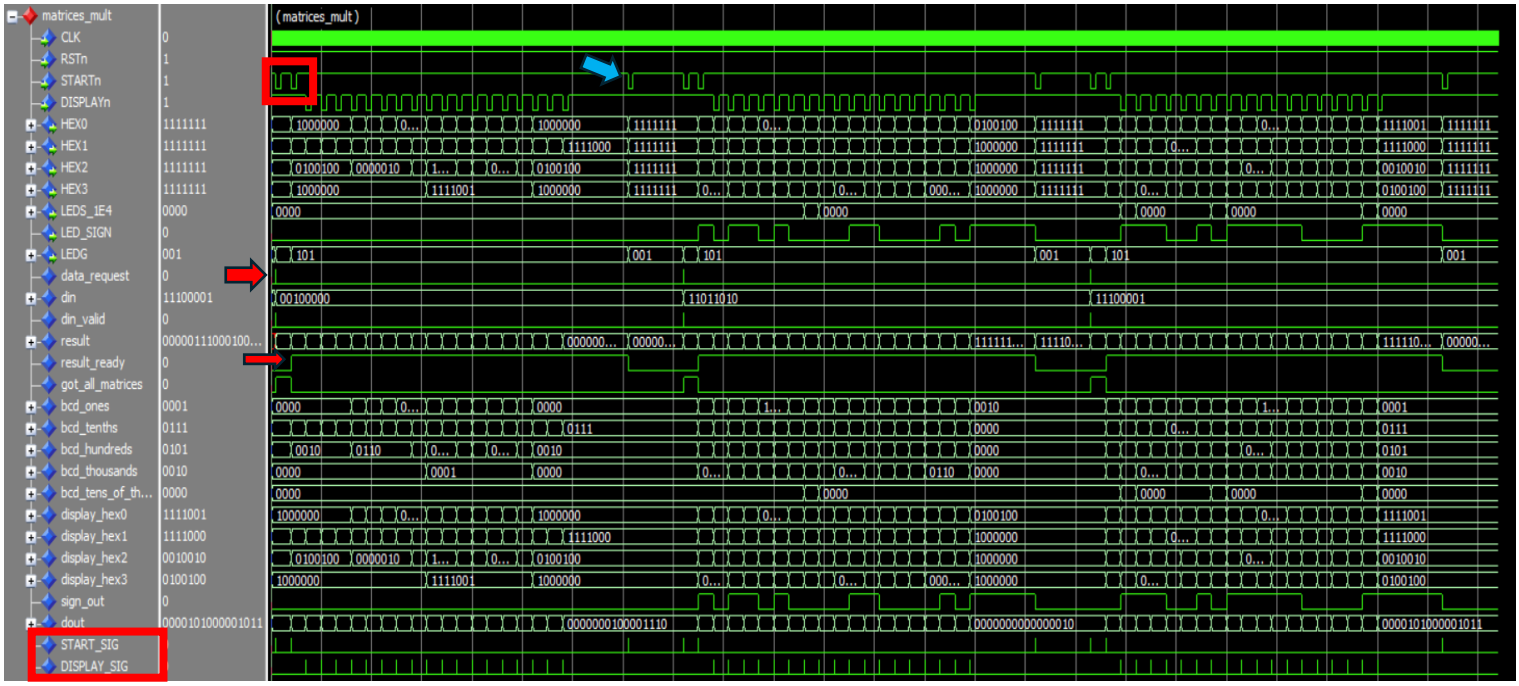
הקוד הזה מגדיר מערכת מורכבת שמשלבת כמה רכיבים לשם ביצוע כפל מטריצות והצגת התוצאה בצורה ברורה. הוא משתמש במכונת מצבים ובאותות סנכרון כדי לוודא שכל שלב בתהליך מתבצע בצורה תקינה ולוגית.

❖ matrices_mult_TB:

```vhdl
library ieee; -- Import IEEE standard logic library for common logic operations
use ieee.std_logic_1164.all; -- Use the IEEE standard logic package for logic operations
use ieee.std_logic_unsigned.all; -- Use the IEEE standard logic package for unsigned arithmetic operations
use ieee.std_logic_arith.all; -- Use the IEEE standard logic package for arithmetic operations
library std; -- Import standard library
use std.textio.all; -- Use standard text I/O package for file handling

entity matrices_mult_tb is
end entity; -- Testbench entity for matrices_mult

architecture behave of matrices_mult_tb is

    constant C_CLK_PRD       : time := 20 ns; -- Clock period for the simulation
  constant NUM_OF_TESTS : integer := 3; -- Number of test cases to be simulated

    type int_array is array(integer range <>) of integer; -- Define a type for an array of integers

    component matrices_mult is
    port (
  CLK                     : in    std_logic;   -- System clock input
  RSTn                    : in    std_logic;   -- Asynchronous, active-low reset input

  STARTn                  : in    std_logic; -- Active-low start signal
  DISPLAYn     : in    std_logic; -- Active-low display signal
  HEX0                    : out   std_logic_vector(6 downto 0); -- Output to 7-segment display 0
  HEX1                    : out   std_logic_vector(6 downto 0); -- Output to 7-segment display 1
  HEX2                    : out   std_logic_vector(6 downto 0); -- Output to 7-segment display 2
  HEX3                    : out   std_logic_vector(6 downto 0); -- Output to 7-segment display 3
  LEDS_1E4     : out   std_logic_vector(3 downto 0); -- Output for LED display (thousands place)
  LED_SIGN                : out   std_logic; -- Output for sign LED
  LEDG                    : out   std_logic_vector(3 downto 1) -- Output for additional green LEDs
    );
    end component; -- End of matrices_mult component declaration

    function seg7_to_bcd(val_in: std_logic_vector(6 downto 0)) return integer is
    begin
        -- Convert 7-segment display code to corresponding BCD integer value
        case val_in is
            when "1000000" =>
                return 0;
            when "1111001" =>
                return 1;
            when "0100100" =>
                return 2;
            when "0110000" =>
                return 3;
            when "0011001" =>
                return 4;
            when "0010010" =>
                return 5;
            when "0000010" =>
                return 6;
            when "1111000" =>
                return 7;
            when "0000000" =>
                return 8;
            when "0010000" =>
                return 9;
            when others =>
                return -1; -- Return -1 if input does not match any valid 7-segment code
        end case;
    end function; -- End of function seg7_to_bcd
    -- Signals for testbench stimulus and response
    signal clk           : std_logic := '0'; -- Clock signal
    signal rstn          : std_logic := '0'; -- Active-low reset signal
    signal start         : std_logic := '1'; -- Start signal for matrices_mult
    signal display       : std_logic := '1'; -- Display signal for matrices_mult
    signal hex0          : std_logic_vector(6 downto 0); -- Output signal for 7-segment display 0
    signal hex1          : std_logic_vector(6 downto 0); -- Output signal for 7-segment display 1
    signal hex2          : std_logic_vector(6 downto 0); -- Output signal for 7-segment display 2
  signal hex3          : std_logic_vector(6 downto 0); -- Output signal for 7-segment display 3
    signal led_sign      : std_logic; -- Output signal for sign LED
    signal ledg          : std_logic_vector(3 downto 1); -- Output signal for additional green LEDs
  signal leds_1e4   : std_logic_vector(3 downto 0); -- Output signal for LED display (thousands place)

begin

    -- Instantiate the design under test (DUT)
    dut: matrices_mult
    port map (
        CLK                     => clk, -- Connect testbench clock signal
        RSTn                     => rstn, -- Connect testbench reset signal
        STARTn                   => start, -- Connect testbench start signal
        DISPLAYn                 => display, -- Connect testbench display signal
        HEX0                     => hex0, -- Connect to testbench HEX0 output
        HEX1                     => hex1, -- Connect to testbench HEX1 output
        HEX2                     => hex2, -- Connect to testbench HEX2 output
        HEX3                     => hex3, -- Connect to testbench HEX3 output
  LEDS_1E4     => leds_1e4, -- Connect to testbench LEDS_1E4 output
  LED_SIGN                => led_sign, -- Connect to testbench LED_SIGN output
  LEDG                    => ledg -- Connect to testbench LEDG output
    );

    clk <= not clk after C_CLK_PRD / 2; -- Generate clock signal with specified period
    rstn <= '0', '1' after 100 ns; -- Apply reset pulse at the beginning of the simulation

    process
    begin
        start <= '1'; -- Initialize start signal to idle state
        display <= '1'; -- Initialize display signal to idle state
        wait for 100 us; -- Wait for 100 microseconds

        for i in 0 to 2 loop
            start <= '0'; -- Start process to get matrices
            wait for 100 us; -- Wait for 100 microseconds
            start <= '1'; -- Return start signal to idle

            wait for 200 us; -- Wait for 200 microseconds

            start <= '0'; -- Start calculation process
            wait for 100 us; -- Wait for 100 microseconds
            start <= '1'; -- Return start signal to idle

            wait for 200 us; -- Wait for 200 microseconds

            for j in 0 to 17 loop -- Loop to simulate display process
                display <= '0'; -- Activate display
                wait for 100 us; -- Wait for 100 microseconds
                display <= '1'; -- Deactivate display
                wait for 200 us; -- Wait for 200 microseconds
            end loop;

            wait for 1 ms; -- Wait for 1 millisecond

            start <= '0'; -- Return to idle state
            wait for 100 us; -- Wait for 100 microseconds
            start <= '1'; -- Return start signal to idle
            wait for 1 ms; -- Wait for 1 millisecond

        end loop;
```

```
        report "End of Simulation"
        severity failure; -- End the simulation with a failure report to stop the simulation

    end process; -- End of test process


    verify_results: process
        variable expected_values    : int_array(0 to 15); -- Array to store expected values
    variable expected_sign      : int_array(0 to 15); -- Array to store expected sign values
        file infile                 : text open read_mode is "expected_results.dat"; -- Open file for reading expected results
        variable inline             : line; -- Line variable to hold the current line from the file
        variable errors_counter     : integer := 0; -- Counter for the number of errors
        variable param_num          : integer := 0; -- Parameter number for indexing
        variable ones, tens, hunds  : integer := 0; -- Variables to hold decoded 7-segment values
        variable dut_val            : integer := 0; -- Variable to hold the calculated DUT value
    variable thousands     : integer := 0; -- Variable to hold the thousands place value
    variable tensofthousands    : integer := 0; -- Variable to hold the tens of thousands place value

    begin

        readline(infile, inline); -- Skip the first line of the file

        for i in 1 to NUM_OF_TESTS loop -- Loop through the number of tests
            wait until falling_edge(start); -- Wait until the falling edge of the start signal

            for j in 0 to 15 loop -- Loop to read expected values from the file
    readline(infile, inline); -- Read a line from the file
    read(inline, expected_sign(j)); -- Read the expected sign
    read(inline, expected_values(j)); -- Read the expected value
            end loop;

            param_num := 0; -- Initialize parameter number

            for k in 0 to 17 loop -- Loop through the display segments
                wait until falling_edge(display); -- Wait until the falling edge of the display signal
                ones := seg7_to_bcd(hex0); -- Decode 7-segment HEX0 to BCD
                tens := seg7_to_bcd(hex1); -- Decode 7-segment HEX1 to BCD
                hunds := seg7_to_bcd(hex2); -- Decode 7-segment HEX2 to BCD
                thousands := seg7_to_bcd(hex3); -- Decode 7-segment HEX3 to BCD
    tensofthousands := conv_integer(leds_1e4); -- Convert LEDS_1E4 to integer
    dut_val := ones + tens*10 + hunds*100 + thousands*1E3 + tensofthousands*1E4; -- Calculate the full value from decoded digits

                if (dut_val = expected_values(param_num)) then -- Check if the DUT value matches the expected value
                    report "Value Pass" & LF; -- Report success if values match
                else
                    report "Value Fail!  " & "Expected=" & integer'image(expected_values(param_num)) & "    Actual=" & integer'image(dut_val) & LF; -- Report failure if values do not match
                    errors_counter := errors_counter + 1; -- Increment error counter
                end if;

    if (conv_integer(led_sign) = expected_sign(param_num)) then -- Check if the DUT sign matches the expected sign
                    report "Sign Pass" & LF; -- Report success if signs match
                else
                    report "Sign Fail!  " & "Expected=" & integer'image(expected_sign(param_num)) & "    Actual=" & integer'image(conv_integer(led_sign)) & LF; -- Report failure if signs do not match
                    errors_counter := errors_counter + 1; -- Increment error counter
                end if;

                if param_num = 15 then -- Check if the parameter number has reached its limit
                    param_num := 0; -- Reset the parameter number
                else
                    param_num := param_num + 1; -- Increment the parameter number
                end if;
            end loop;

        end loop;

        report "Total errors: " & integer'image(errors_counter) & LF; -- Report the total number of errors

        wait; -- Wait indefinitely, effectively ending the process

    end process; -- End of verify_results process

end architecture; -- End of architecture behave
```

❖ matrices_mult simulation:



הסבר סימולציה:

קבלת מטריצות והתחלת חישוב:

- תחילת מצב :כאשר האות STARTn יורד לאפס.
- לחיצה ראשונה על START,המערכת מתחילה לקבל את נתוני המטריצות דרך אות ה .din-כל האותות המעידים על קבלת הנתונים (data_request, din_valid) =1 פעילים ומסמנים שהנתונים נקלטו אחרי ש got_all matrices עלה ל1.
- לחיצה שניה על START, מתחילה חישוב המכפלה בין המטריצות. תהליך זה נמשך עד לסיום החישוב ואז רואים ש result_ready עולה ל1.

הצגת התוצאה:

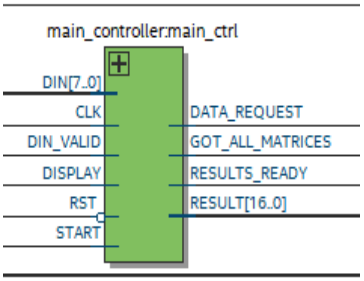
- תחילת מצב :כאשר האות DISPLAYnמופעל.
- לחיצה עלDISPLAYn מציגה את האברים , התוצאה הסופית מוצגת על תצוגות ה-7-segment (HEX0-HEX3), כאשר אותות ה LED-מסמנים את מצב התוצאה (חיובי/שלילי). המערכת ממשיכה במצב זה עד לקבלת פקודת איפוס חדשה.

לחיצת בפעם שלישית ל START :

- לחיצה שלישית על START, המערכת מתחילה לקבל את נתוני המטריצות
- השניות ועוד שתי לחיצות מתילים לראות תוצאות המטריצה השניה.

START_SIG DISPLAY_SIG :

- אלה ה signals של SRART וDISPLAY אחרי שהכנסנו אותם ל SYNC כדי שיהיו באורך שעון אחד בלבד והן סנכרונים



משימה 2: ייצור הבלוק main_controller

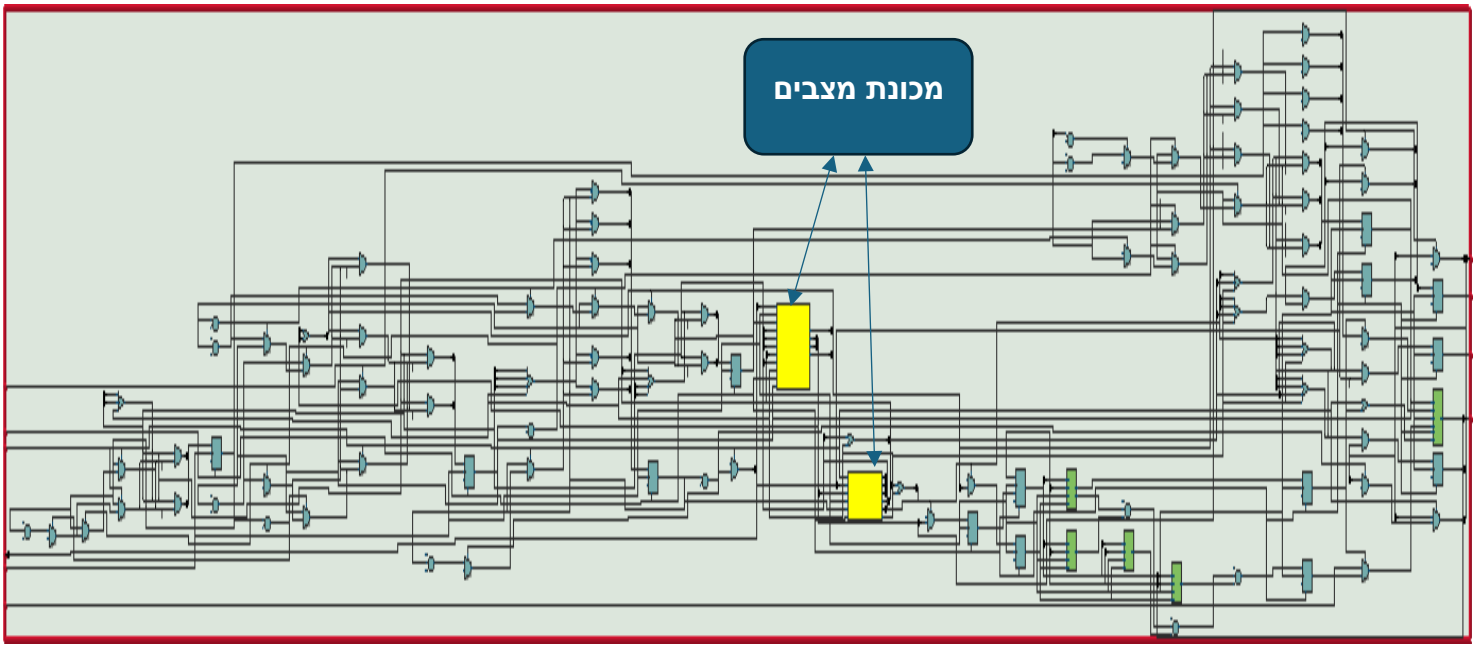
2.2 הבלוק main_controller

6.1.19 מימוש הבלוק main_controller

איור 7 : הבלוק main_controller

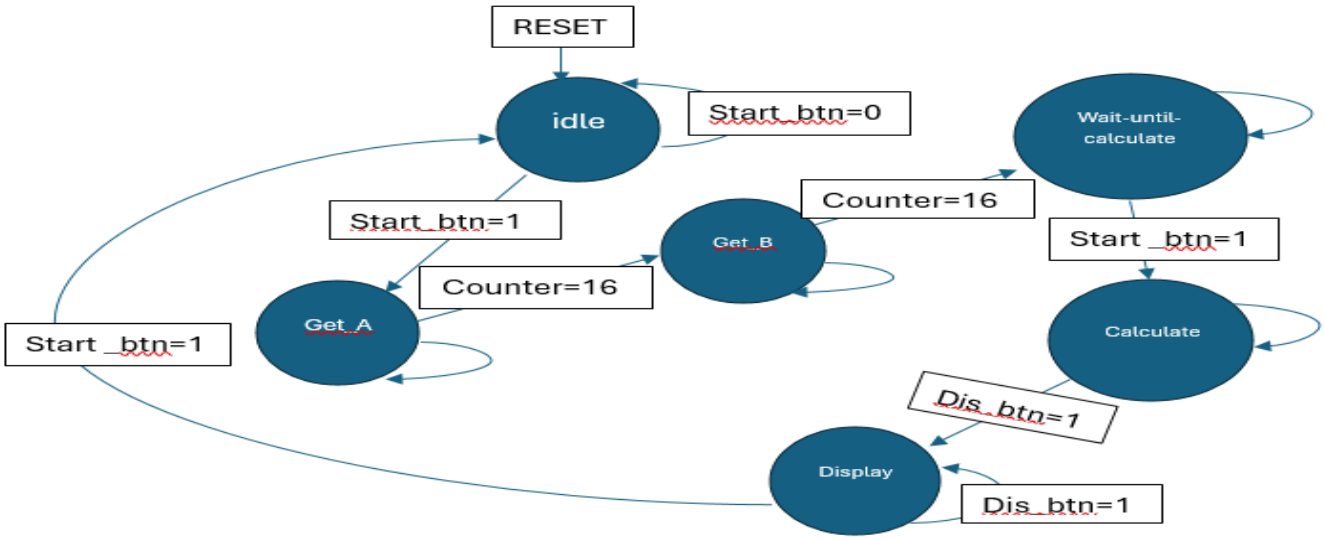
טבלה 7: תיאור הבלוק main_controller

Generics			
Name	**Dir.**	**Type**	**Description**
Ports			
Name	**Dir.**	**type**	**Description**
System Signals			
CLK	I	std_logic	System clock
RST	I	std_logic	Active high asynchronous reset
START	I	std_logic	פולס חיובי ברוחב מחזור שעון אחד שמתקבל בעקבות **לחיצה** (ולא שחרור) על הלחצן STARTn
DISPLAY	I	std_logic	פולס חיובי ברוחב מחזור שעון אחד שמתקבל בעקבות **לחיצה** (ולא שחרור) על הלחצן DISPLAYn
DATA_REQUEST	O	std_logic	פולס חיובי ברוחב מחזור שעון אחד שבעקבותיו יתקבלו 16 איברים של מטריצה אחת.
DIN	I	std_logic_vector (7:0)	כניסת המידע של איברי המטריצה המתקבלת
DIN_VALID	I	std_logic	המידע בכניסה DIN תקף בכל עלית שעון שבה כניסה זו ב-'1'
RESULT	O	std_logic_vector (16:0)	ביציאה זו יצאו הערכים של מטריצת היעד
RESULT_READY	O	std_logic	המידע ביציאה RESULT תקף בכל עלית שעון שבה יציאה זו ב-'1'.
GOT_ALL_MATRICES	O	std_logic	יציאה זו תעלה ל-'1' לאחר שהתבלו שתי מטריצות המקור ונרשמו לזיכרון.



מכונות מצבים:

1. Main state Machine



תיאור המצבים במכונת המצבים הראשית (main_sm):

1. st_idle:

במצב זה, המערכת מחכה לאתחול (באמצעות אות START). כאשר מתקבל האות, המערכת עוברת למצב **st_receive_first_mat** שבו היא מתחילה לקבל את המטריצה הראשונה.

2. st_receive_first_mat:

המערכת נמצאת במצב קבלת המטריצה הראשונה. היא מקבלת נתונים דרך האות **DIN** ומאמתת אותם באמצעות **DIN_VALID**. לאחר קבלת כל הנתונים למטריצה הראשונה, המערכת עוברת למצב קבלת המטריצה השנייה **st_receive_second_mat**.

3. st_receive_second_mat:

במצב זה, המערכת מקבלת את המטריצה השנייה. תהליך זה דומה לתהליך של קבלת המטריצה הראשונה. לאחר קבלת כל הנתונים, המערכת עוברת למצב המתנה לחישוב **st_wait_for_calculate.**

4. st_wait_for_calculate:

במצב זה, המערכת מחכה לקבלת אישור נוסף (אות **START**) כדי להתחיל את תהליך החישוב. כאשר מתקבל האות, המערכת עוברת למצב **st_calculate.**

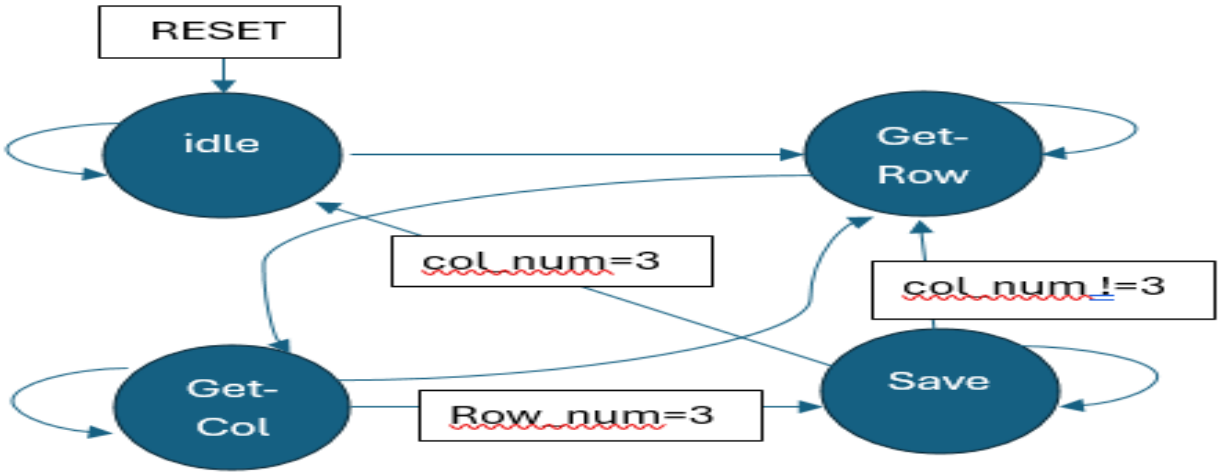
5. st_calculate:

זהו מצב שבו מתבצע החישוב של כפל המטריצות. כאן מכונת המצבים המשנית (**calc_sm**) נכנסת לפעולה ומבצעת את החישוב עבור כל שורה ועמודה במטריצות.

6. st_display:

לאחר שהחישוב הושלם, המערכת עוברת למצב התצוגה שבו התוצאה מוצגת. כאשר מתקבל האות **DISPLAY**, המערכת מציגה את התוצאה על המסך או תצוגה אחרת. לאחר הצגה של כל התוצאה, המערכת חוזרת למצב **st_idle**.

2. Multiply State Machine



תיאור המצבים במכונת המצבים של החישוב (calc_sm):

st_idle:

מצב מנוחה שבו מחכה המערכת להתחלת תהליך החישוב.

st_get_row:

במצב זה, המערכת משיגה את השורה מהמטריצה הראשונה לצורך חישוב.

st_get_col:

במצב זה, המערכת משיגה את העמודה מהמטריצה השנייה לצורך חישוב כפל של השורה והעמודה. לאחר מכן, היא עוברת למצב **st_save**.

st_save:

במצב זה, המערכת שומרת את התוצאה של כפל השורה והעמודה במשתנה תוצאה. אם כל השורות והעמודות עובדו, המערכת חוזרת למצב **st_idle**.

❖ main_controller code:

```
library ieee; -- Import the IEEE library for standard logic and arithmetic operations
use ieee.std_logic_1164.all; -- Use the IEEE standard logic package for logic operations
use ieee.numeric_std.all; -- Use the IEEE numeric standard package for arithmetic operations

entity main_controller is
    port (
        CLK              : in    std_logic; -- System clock input
        RST              : in    std_logic; -- System reset input
        START            : in    std_logic; -- Start signal input
        DISPLAY          : in    std_logic; -- Display signal input
        DIN              : in    std_logic_vector(7 downto 0); -- Data input vector
        DIN_VALID        : in    std_logic; -- Data valid flag input
        DATA_REQUEST     : out   std_logic; -- Data request output signal
        RESULT           : out   std_logic_vector(16 downto 0); -- Result output vector
        RESULTS_READY    : out   std_logic; -- Result ready output flag
        GOT_ALL_MATRICES : out   std_logic -- All matrices received output flag
    );
end entity;

architecture behave of main_controller is
    constant C_NUM_OF_ELEMENTS : integer := 16; -- Number of elements in matrices
    constant N                 : integer := 8; -- Width of data input/output for multiplier
    constant LATENCY           : integer := 2; -- Latency for multiplier
    constant IS_SIGNED         : boolean := true; -- Flag for signed or unsigned operation

    -- State machine definitions for main control and calculation processes
    type main_sm_states is (
        st_idle, -- Idle state
        st_receive_first_mat, -- Receiving first matrix
        st_receive_second_mat, -- Receiving second matrix
        st_wait_for_calculate, -- Waiting to start calculation
        st_calculate, -- Calculating matrix multiplication
        st_display -- Displaying results
    );

    type calc_sm_states is (
        st_idle, -- Idle state
        st_get_row, -- Getting row for calculation
        st_get_col, -- Getting column for calculation
        st_save -- Saving calculated results
    );

    -- Component declaration for matrix RAM
    component matrix_ram is
        generic (
            DATA_WIDTH    : integer := 32; -- Data width of RAM
            ADDRESS_BITS  : integer := 5 -- Number of address bits for RAM
        );
        port (
            CLK       : in  std_logic; -- Clock input
            RST       : in  std_logic; -- Reset input
            DATA      : in  std_logic_vector(DATA_WIDTH-1 downto 0); -- Data input
            WREN      : in  std_logic; -- Write enable input
            ADDRESS   : in  std_logic_vector(ADDRESS_BITS-1 downto 0); -- Address input
            BYTEENA   : in  std_logic_vector(DATA_WIDTH/8-1 downto 0); -- Byte enable input
            Q         : out std_logic_vector(DATA_WIDTH-1 downto 0) -- Data output
        );
    end component;

    -- Component declaration for multiplier
    component my_multiplier is
        generic (
            N         : integer := 8; -- Width of data inputs/outputs
            LATENCY   : integer range 1 to 8 := 2; -- Latency of multiplier
            IS_SIGNED : boolean := false -- Flag for signed/unsigned operation
        );
        port (
            CLK       : in  std_logic; -- Clock input
            DIN_VALID : in  std_logic; -- Data valid input
            A         : in  std_logic_vector(N-1 downto 0); -- Multiplier input A
            B         : in  std_logic_vector(N-1 downto 0); -- Multiplier input B
            Q         : out std_logic_vector(N*2-1 downto 0); -- Multiplication result output
            DOUT_VALID : out std_logic -- Data output valid flag
        );
    end component;
    -- Signal declarations
    signal main_sm                  : main_sm_states; -- Main state machine signal
    signal calc_sm                  : calc_sm_states; -- Calculation state machine signal
    signal data_count               : integer range 0 to C_NUM_OF_ELEMENTS; -- Counter for received data elements
    signal row_num                  : integer range 0 to 3; -- Row index signal
    signal col_num                  : integer range 0 to 3; -- Column index signal
    signal byte_enable              : std_logic_vector(3 downto 0); -- Byte enable signal for RAM
    signal mem_byte_enable          : std_logic_vector(3 downto 0); -- Byte enable signal for memory
    signal mem_address              : std_logic_vector(4 downto 0); -- Memory address signal
    signal mem_wr_data              : std_logic_vector(31 downto 0); -- Memory write data signal
    signal mem_wr                   : std_logic; -- Memory write enable signal
    signal high_address             : std_logic_vector(2 downto 0) := "000"; -- High bits of the address
    signal iteration_num            : integer range 0 to 3; -- Iteration number for calculation
    signal store_mat1_row_data      : std_logic := '0'; -- Signal to store row data from matrix 1
    signal store_mat2_col_data      : std_logic := '0'; -- Signal to store column data from matrix 2
    signal mat1_row_data            : std_logic_vector(31 downto 0); -- Data for row from matrix 1
    signal mem_dout                 : std_logic_vector(31 downto 0); -- Data output from memory

    signal mult1_q                  : std_logic_vector(15 downto 0); -- Output of first multiplier
    signal mult2_q                  : std_logic_vector(15 downto 0); -- Output of second multiplier
    signal mult3_q                  : std_logic_vector(15 downto 0); -- Output of third multiplier
    signal mult4_q                  : std_logic_vector(15 downto 0); -- Output of fourth multiplier

    signal mult1_q_valid            : std_logic; -- Valid flag for first multiplier output
    signal mult2_q_valid            : std_logic; -- Valid flag for second multiplier output
    signal mult3_q_valid            : std_logic; -- Valid flag for third multiplier output
    signal mult4_q_valid            : std_logic; -- Valid flag for fourth multiplier output

    signal res_mat_element_valid    : std_logic := '0'; -- Valid flag for result matrix element
    signal res_mat_element          : std_logic_vector(31 downto 0) := (others => '0'); -- Result matrix element
    signal result_ready             : std_logic := '0'; -- Flag to indicate result is ready

begin

    -- Main process handling the state machines and control logic
    process(CLK, RST)
    begin
        if RST = '1' then -- Reset condition
            main_sm <= st_idle; -- Set main state machine to idle
            calc_sm <= st_idle; -- Set calculation state machine to idle
            DATA_REQUEST <= '0'; -- Clear data request signal
            data_count <= 0; -- Reset data count
            row_num <= 0; -- Reset row number
            col_num <= 0; -- Reset column number
            high_address <= "000"; -- Reset high address bits
            iteration_num <= 0; -- Reset iteration number
            store_mat1_row_data <= '0'; -- Clear store matrix 1 row data signal
            store_mat2_col_data <= '0'; -- Clear store matrix 2 column data signal
            result_ready <= '0'; -- Clear result ready flag
            GOT_ALL_MATRICES <= '0'; -- Clear all matrices received flag
            RESULTS_READY <= '0'; -- Clear results ready flag
        elsif rising_edge(CLK) then -- On rising edge of the clock
            DATA_REQUEST <= '0'; -- Clear data request signal
            store_mat1_row_data <= '0'; -- Clear store matrix 1 row data signal
            store_mat2_col_data <= '0'; -- Clear store matrix 2 column data signal

            -- Main state machine
            case main_sm is
                when st_idle =>
                    if start = '1' then -- Check if start signal is active
                        main_sm <= st_receive_first_mat; -- Transition to receive first matrix state
                        DATA_REQUEST <= '1'; -- Set data request signal
                    end if;
                    row_num <= 0; -- Reset row number
                    col_num <= 0; -- Reset column number
                    iteration_num <= 0; -- Reset iteration number

                when st_receive_first_mat =>
                    if DIN_VALID = '1' then -- Check if data is valid
                        if data_count = C_NUM_OF_ELEMENTS -1 then -- If all elements received
                            main_sm <= st_receive_second_mat; -- Transition to receive second matrix state
                            data_count <= 0; -- Reset data count
                            DATA_REQUEST <= '1'; -- Set data request signal
                        else
                            data_count <= data_count + 1; -- Increment data count
                        end if;
```

```
                        if col_num = 3 then -- Check if last column is received
                            col_num <= 0; -- Reset column number
                            if row_num = 3 then -- Check if last row is received
                                row_num <= 0; -- Reset row number
                            else
                                row_num <= row_num + 1; -- Increment row number
                            end if;
                        else
                            col_num <= col_num + 1; -- Increment column number
                        end if;

                    end if;
                    high_address <= "000"; -- Reset high address bits

                when st_receive_second_mat =>
                    if DIN_VALID = '1' then -- Check if data is valid
                        if data_count = C_NUM_OF_ELEMENTS -1 then -- If all elements received
                            main_sm <= st_wait_for_calculate; -- Transition to wait for calculation state
                            GOT_ALL_MATRICES <= '1'; -- Set all matrices received flag
                            data_count <= 0; -- Reset data count
                        else
                            data_count <= data_count + 1; -- Increment data count
                        end if;

                        if row_num = 3 then -- Check if last row is received
                            row_num <= 0; -- Reset row number
                            if col_num = 3 then -- Check if last column is received
                                col_num <= 0; -- Reset column number
                            else
                                col_num <= col_num + 1; -- Increment column number
                            end if;
                        else
                            row_num <= row_num + 1; -- Increment row number
                        end if;
                    end if;
                    high_address <= "001"; -- Set high address bits for second matrix

                when st_wait_for_calculate =>
                    row_num <= 0; -- Reset row number
                    high_address <= "000"; -- Reset high address bits
                    if start = '1' then -- Check if start signal is active
                        main_sm <= st_calculate; -- Transition to calculate state
                        calc_sm <= st_get_row; -- Set calculation state machine to get row state
                    end if;

                when st_calculate =>
                    -- Calculation state machine
                    case calc_sm is
                        when st_get_row =>
                            calc_sm <= st_get_col; -- Transition to get column state
                            high_address <= "001"; -- Set high address bits for second matrix
                            row_num <= 0; -- Reset row number
                            store_mat1_row_data <= '1'; -- Set store matrix 1 row data signal


                        when st_get_col =>
                            if row_num = 3 then -- Check if last row is reached
                                calc_sm <= st_save; -- Transition to save state
                                row_num <= 0; -- Reset row number
                                high_address <= '1' & std_logic_vector(to_unsigned(iteration_num, 2)); -- Set high address bits for result storage
                            else
                                row_num <= row_num + 1; -- Increment row number
                            end if;
                            store_mat2_col_data <= '1'; -- Set store matrix 2 column data signal


                        when st_save =>
                            if row_num = 3 then -- Check if last row is reached
                                if iteration_num = 3 then -- Check if last iteration is done
                                    calc_sm <= st_idle; -- Transition to idle state
                                else
                                    iteration_num <= iteration_num + 1; -- Increment iteration number
                                    calc_sm <= st_get_row; -- Transition to get row state
                                    high_address <= "000"; -- Reset high address bits
                                    row_num <= iteration_num + 1; -- Set row number for next iteration
                                end if;
                            else
                                row_num <= row_num + 1; -- Increment row number
                            end if;

                          when st_idle =>
                              main_sm <= st_display; -- Transition to display state
                              result_ready <= '1'; -- Set result ready flag
                              GOT_ALL_MATRICES <= '0'; -- Clear all matrices received flag
                              row_num <= 0; -- Reset row number
                              high_address <= "100"; -- Set high address bits for result display
                        end case;

                 when st_display =>
                      if display = '1' then -- Check if display signal is active
                          if row_num = 3 then -- Check if last row is reached
                              if unsigned(high_address(1 downto 0)) = 3 then -- Check if last address is reached
                                  high_address <= "100"; -- Set high address bits to result display state
                              else
                                  high_address <= std_logic_vector(unsigned(high_address) + 1); -- Increment high address bits
                              end if;
                              row_num <= 0; -- Reset row number
                          else
                              row_num <= row_num + 1; -- Increment row number
                          end if;
                      end if;
                      if start = '1' then -- Check if start signal is active
                          main_sm <= st_idle; -- Transition to idle state
                          result_ready <= '0'; -- Clear result ready flag
                      end if;

            end case;

            RESULTS_READY <= result_ready; -- Update results ready flag

        end if;
    end process;

    -- Process to store row data from matrix 1 into memory
    process(CLK, RST)
    begin
        if RST = '1' then -- Reset condition
            mat1_row_data <= (others => '0'); -- Clear matrix 1 row data
        elsif rising_edge(CLK) then -- On rising edge of the clock
            if store_mat1_row_data = '1' then -- Check if store signal is active
                mat1_row_data <= mem_dout; -- Store memory output into matrix 1 row data
            end if;
        end if;
    end process;

    -- Memory address and write control logic
    mem_address <= high_address & std_logic_vector(to_unsigned(row_num, 2)); -- Combine high address and row number for memory address
    mem_wr_data <= res_mat_element when res_mat_element_valid = '1' else DIN & DIN & DIN & DIN; -- Select write data for memory
    mem_wr <= DIN_VALID or res_mat_element_valid; -- Enable memory write if data input is valid or result element is valid
    mem_byte_enable <= "1111" when res_mat_element_valid = '1' else byte_enable; -- Select byte enable for memory

    -- Process to generate byte enable signals based on column number
    gen_byte_enable : process(col_num)
    begin
        case col_num is
            when 0 =>
                byte_enable <= "0001"; -- Enable first byte
            when 1 =>
                byte_enable <= "0010"; -- Enable second byte
            when 2 =>
                byte_enable <= "0100"; -- Enable third byte
            when 3 =>
                byte_enable <= "1000"; -- Enable fourth byte
        end case;
    end process;
```

```vhdl
    -- Instantiate matrix RAM component
    matrix_ram_inst : matrix_ram
        generic map (
            DATA_WIDTH => 32, -- Data width of RAM
            ADDRESS_BITS => 5 -- Number of address bits for RAM
        )
        port map (
            CLK        => CLK, -- Connect clock signal
            RST        => RST, -- Connect reset signal
            DATA       => mem_wr_data, -- Connect memory write data
            WREN       => mem_wr, -- Connect memory write enable
            ADDRESS    => mem_address, -- Connect memory address
            BYTEENA    => mem_byte_enable, -- Connect byte enable signals
            Q          => mem_dout -- Connect memory output data
        );

    -- Instantiate first multiplier component
    mult1 : my_multiplier
        generic map (
            N          => N, -- Width of data inputs/outputs
            LATENCY    => LATENCY, -- Latency of multiplier
            IS_SIGNED  => IS_SIGNED -- Signed/unsigned operation flag
        )
        port map (
            CLK        => CLK, -- Connect clock signal
            DIN_VALID  => store_mat2_col_data, -- Connect data valid signal
            A          => mat1_row_data(7 downto 0), -- Connect lower 8 bits of matrix 1 row data
            B          => mem_dout(7 downto 0), -- Connect lower 8 bits of memory output data
            Q          => mult1_q, -- Connect multiplier output
            DOUT_VALID => mult1_q_valid -- Connect multiplier output valid flag
        );

    -- Instantiate second multiplier component
    mult2 : my_multiplier
        generic map (
            N          => N, -- Width of data inputs/outputs
            LATENCY    => LATENCY, -- Latency of multiplier
            IS_SIGNED  => IS_SIGNED -- Signed/unsigned operation flag
        )
        port map (
            CLK        => CLK, -- Connect clock signal
            DIN_VALID  => store_mat2_col_data, -- Connect data valid signal
            A          => mat1_row_data(15 downto 8), -- Connect next 8 bits of matrix 1 row data
            B          => mem_dout(15 downto 8), -- Connect next 8 bits of memory output data
            Q          => mult2_q, -- Connect multiplier output
            DOUT_VALID => mult2_q_valid -- Connect multiplier output valid flag
        );

    -- Instantiate third multiplier component
    mult3 : my_multiplier
        generic map (
            N          => N, -- Width of data inputs/outputs
            LATENCY    => LATENCY, -- Latency of multiplier
            IS_SIGNED  => IS_SIGNED -- Signed/unsigned operation flag
        )
        port map (
            CLK        => CLK, -- Connect clock signal
            DIN_VALID  => store_mat2_col_data, -- Connect data valid signal
            A          => mat1_row_data(23 downto 16), -- Connect next 8 bits of matrix 1 row data
            B          => mem_dout(23 downto 16), -- Connect next 8 bits of memory output data
            Q          => mult3_q, -- Connect multiplier output
            DOUT_VALID => mult3_q_valid -- Connect multiplier output valid flag
        );

    -- Instantiate fourth multiplier component
    mult4 : my_multiplier
        generic map (
            N          => N, -- Width of data inputs/outputs
            LATENCY    => LATENCY, -- Latency of multiplier
            IS_SIGNED  => IS_SIGNED -- Signed/unsigned operation flag
        )
        port map (
            CLK        => CLK, -- Connect clock signal
            DIN_VALID  => store_mat2_col_data, -- Connect data valid signal
            A          => mat1_row_data(31 downto 24), -- Connect upper 8 bits of matrix 1 row data
            B          => mem_dout(31 downto 24), -- Connect upper 8 bits of memory output data
            Q          => mult4_q, -- Connect multiplier output
            DOUT_VALID => mult4_q_valid -- Connect multiplier output valid flag
        );
    -- Process to sum the products of the multiplications and store the result
    sum_of_products : process(CLK, RST)
    begin
        if RST = '1' then -- Reset condition
            res_mat_element        <= (others => '0'); -- Clear result matrix element
            res_mat_element_valid  <= '0'; -- Clear result element valid flag
        elsif rising_edge(CLK) then -- On rising edge of the clock
            res_mat_element <= std_logic_vector(
                resize(signed(mult1_q), 32) +
                resize(signed(mult2_q), 32) +
                resize(signed(mult3_q), 32) +
                resize(signed(mult4_q), 32)); -- Sum the resized multiplier outputs
            res_mat_element_valid <= mult1_q_valid; -- Set result element valid flag based on first multiplier valid flag
        end if;
    end process;

    RESULT <= mem_dout(16 downto 0); -- Output the lower 17 bits of the memory output as the final result

end architecture;
```

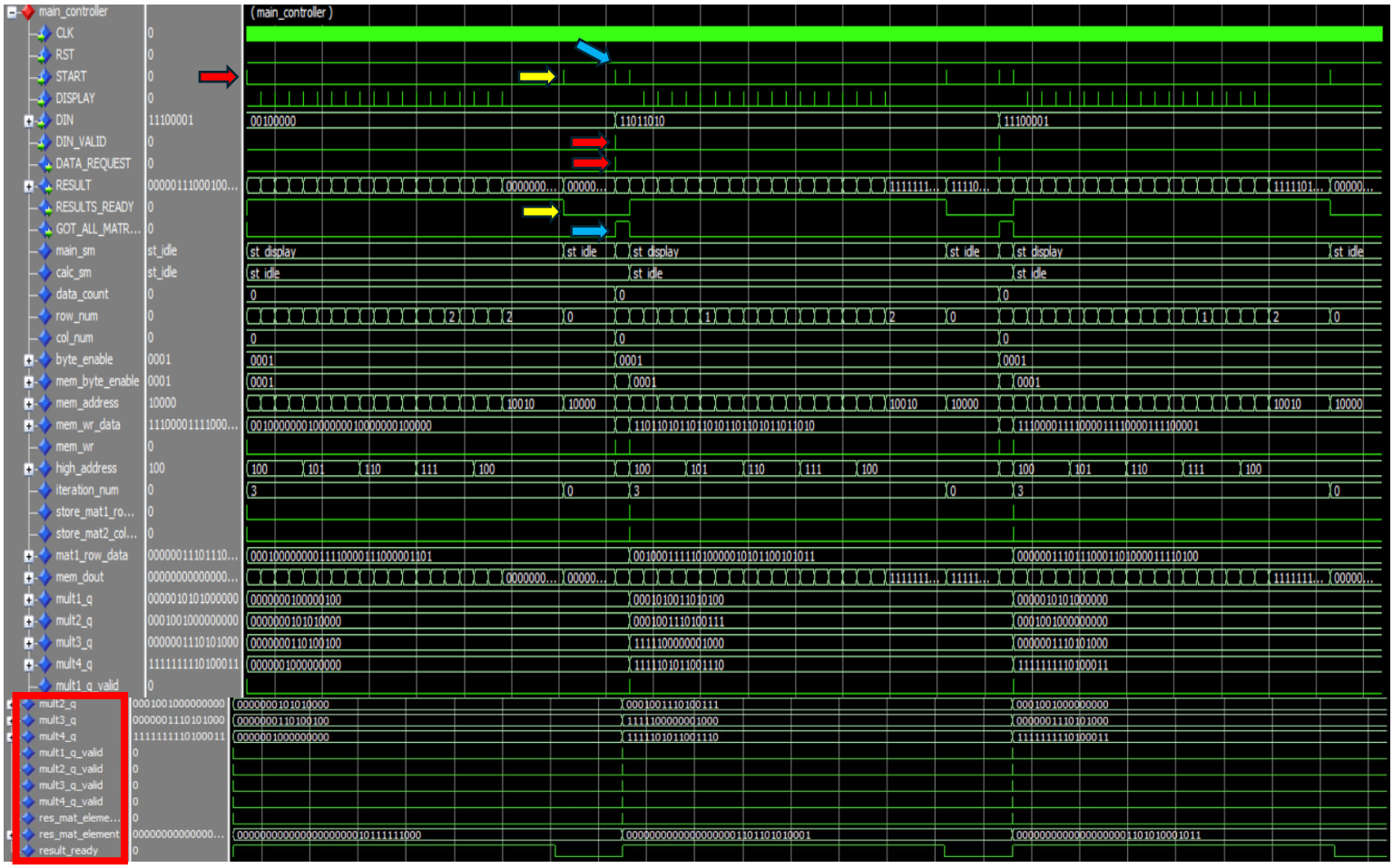
הסברים כלליים:

- **הגדרות ראשוניות:** הקוד כולל קבועים (C_NUM_OF_ELEMENTS, N, LATENCY) המשמשים להגדרת מאפיינים שונים של המערכת, כמו מספר האיברים במטריצה, רוחב הנתונים, וזמן ההשהיה של הכפל.
- **מכונת מצבים:** (State Machine) הקוד משתמש בשתי מכונות מצבים - אחת לתהליך הראשי (main_sm) ואחת לתהליך החישוב. (calc_sm) כל אחת מהן עוברת בין מצבים שונים כדי לבצע את המשימות הנדרשות.

סיכום כללי:

הקוד הזה מגדיר יחידה מרכזית שתפקידה לקבל נתונים של מטריצות, לבצע כפל מטריצות בצורה סינכרונית באמצעות מכפלים, ולשמור את התוצאה בזיכרון לשימוש בהמשך או להצגה. מכונות המצבים מבטיחות שהמעבר בין השלבים יתבצע בצורה נכונה ורציפה, תוך שמירה על הסדר הלוגי של הפעולות.

❖ **main_controller simulation:**



הסבר סימולציה:

קבלת המטריצות:

- **תחילת מצב :** כאשר האות **STARTn** עולה לגבוה.
- **לחיצה ראשונה על STARTn :** המערכת מתחילה לקבל את נתוני המטריצה הראשונה דרך אות ה- **DIN**. כל האותות המעידים על קבלת הנתונים (**DATA_REQUEST, DIN_VALID**) פעילים ומסמנים שהנתונים נקלטו כראוי. האות **GOT_ALL_MATRICES** עולה ל-1 כדי להראות שכל המטריצות התקבלו.
- **לחיצה שניה על STARTn :** המערכת מתחילה לקבל את נתוני המטריצה השנייה באותו תהליך כמו בקבלת המטריצה הראשונה. כאשר הנתונים התקבלו, המערכת מוכנה להתחיל את תהליך החישוב.

חישוב המטריצות:

- **תחילת מצב** : כאשר האות **STARTn** מופעל פעם נוספת לאחר קבלת שתי המטריצות.
- **תהליך :** המערכת מבצעת את חישוב המכפלה בין המטריצות. תהליך זה נמשך עד לסיום החישוב, שאז ניתן לראות שהאות **RESULTS_READY** עולה ל-1 כדי לסמן שהתוצאה מוכנה.

הצגת התוצאה:

- **תחילת מצב :** כאשר האות **DISPLAYn** מופעל.
- **לחיצה על DISPLAYn :** התוצאה הסופית מוצגת על תצוגות ה (7-segment HEX0-HEX3) כאשר האותות **LEDG** ו- **LED_SIGN** מסמנים את מצב התוצאה (חיובי/שלילי). המערכת ממשיכה במצב הצגה זה עד לקבלת פקודת איפוס של **STARTn** חדשה.

לחיצה נוספת על STARTn:

- **תחילת מצב :** לחיצה שלישית על **START** מתחילה את התהליך מחדש, המערכת שוב מוכנה לקבל נתוני מטריצה חדשה ולהציג את התוצאות במידה והתקבלו שתי מטריצות חדשות.

קבלת נתוני המטריצות והתחלת החישוב:

- **תחילת מצב** : כאשר האותות **mult1_q_valid** עד **mult4_q_valid** הופכים לגבוהים.
- **תהליך :** המערכת מקבלת את תוצאות החישובים הבינאריים מהחישובים של כל שורה ועמודה במטריצות (דרך **mult1_q** עד **mult4_q**). כל אחד מהאותות האלה מסמן שהערך התקבל ומאומת. ברגע שהאות **res_mat_elements_valid** עולה ל-1, המערכת מסיימת את קבלת הנתונים ועוברת לשלב הבא.

חישוב תוצאת המטריצה:

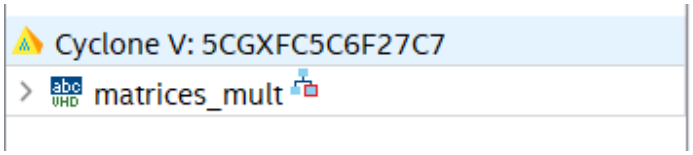
- **תחילת מצב :** כאשר האות **res_mat_elements_valid** במצב גבוה.
- **תהליך :** המערכת מתחילה לחשב את התוצאה של כפל המטריצות, כאשר האות **res_mat_elements_q** מציג את התוצאה של כל אלמנט במטריצה. תהליך זה ממשיך עד לסיום חישוב כל האלמנטים של המטריצה.

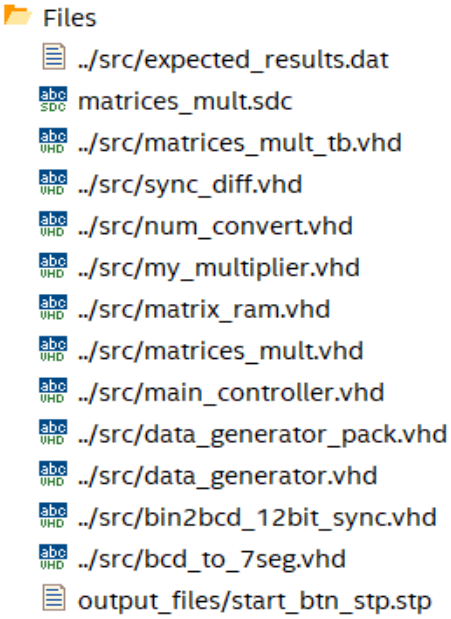
סיום החישוב והצגת התוצאה:

- **תחילת מצב :** כאשר האות **result_ready** עולה ל-1.
- **תהליך :** בסיום החישוב, האות **result_ready** עולה כדי לסמן שהתוצאה הסופית מוכנה להצגה או לשימוש נוסף במערכת. בשלב זה, המערכת סיימה את החישוב ומוכנה לשלב הבא של הצגת התוצאה או קבלת פקודת איפוס חדשה.

3. שלבי הסינתיזה

3.1 הרמה העליונה והקבצים





3.2 קומפילציה (ניצול משאבים)

Flow Summary

Flow Status	Successful - Wed Aug 28 10:26:05 2024
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	matrices_mult
Top-level Entity Name	matrices_mult
Family	Cyclone V
Device	5CGXFC5C6F27C7
Timing Models	Final
Logic utilization (in ALMs)	1,041 / 29,080 (4 %)
Total registers	2106
Total pins	40 / 364 (11 %)
Total virtual pins	0
Total block memory bits	4,736 / 4,567,040 (< 1 %)
Total DSP Blocks	4 / 150 (3 %)
Total HSSI RX PCSs	0 / 6 (0 %)
Total HSSI PMA RX Deserializers	0 / 6 (0 %)
Total HSSI TX PCSs	0 / 6 (0 %)
Total HSSI PMA TX Serializers	0 / 6 (0 %)
Total PLLs	0 / 12 (0 %)
Total DLLs	0 / 4 (0 %)

הסבר למה השתמשנו ביותר DSP בלקים:

בגלל שעשינו Signal tap היא צורכת יותר משאבים.

3.3 ניצול משאבים של כל בלוק

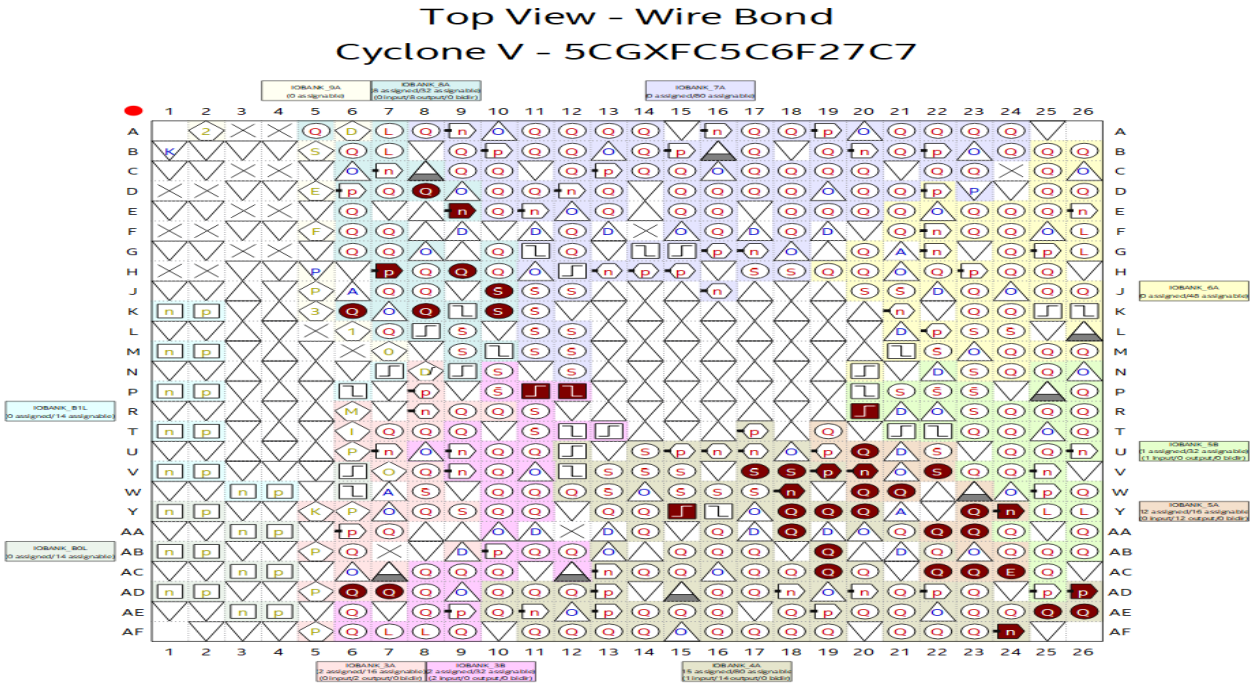
Entity:Instance	ALMs needed [=A-B+C]	[A] ALMs used in final placement	[B] Estimate of ALMs recoverable by dense packing	[C] Estimate of ALMs unavailable	ALMs used for memory
Cyclone V: 5CGXFC5C6F27C7					
matrices_mult	1041.0 (3.2)	1365.0 (3.2)	325.5 (0.0)	1.5 (0.0)	0.0 (0.0)
sld_hub:auto_hub	61.0 (0.5)	69.5 (0.5)	8.5 (0.0)	0.0 (0.0)	0.0 (0.0)
sld_signaltap:auto_signaltap_0	276.5 (6.8)	396.0 (28.7)	119.5 (21.9)	0.0 (0.0)	0.0 (0.0)
bcd_to_7seg:bcd_to_7seg_inst0	4.5 (4.5)	6.0 (6.0)	1.5 (1.5)	0.0 (0.0)	0.0 (0.0)
bcd_to_7seg:bcd_to_7seg_inst1	4.7 (4.7)	5.0 (5.0)	0.3 (0.3)	0.0 (0.0)	0.0 (0.0)
bcd_to_7seg:bcd_to_7seg_inst2	5.0 (5.0)	5.3 (5.3)	0.3 (0.3)	0.0 (0.0)	0.0 (0.0)
bcd_to_7seg:bcd_to_7seg_inst3	7.0 (7.0)	7.0 (7.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
bin2bcd_12bit_sync:bin2bcd_inst	44.7 (44.7)	54.3 (54.3)	9.7 (9.7)	0.0 (0.0)	0.0 (0.0)
data_generator:data_gen	23.2 (23.2)	24.6 (24.6)	1.4 (1.4)	0.0 (0.0)	0.0 (0.0)
main_controller:main_ctrl	600.3 (43.8)	782.3 (43.8)	183.4 (0.5)	1.5 (0.5)	0.0 (0.0)
num_convert:num_convert_inst	8.3 (8.3)	8.5 (8.5)	0.2 (0.2)	0.0 (0.0)	0.0 (0.0)
sync_diff:sync_diff_DISPLAY	1.3 (1.3)	2.0 (2.0)	0.7 (0.7)	0.0 (0.0)	0.0 (0.0)
sync_diff:sync_diff_START	1.3 (1.3)	1.3 (1.3)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)

Combinational ALUTs	Dedicated Logic Registers	I/O Registers	Block Memory Bits	M10Ks	DSP Blocks	Pins	Virtual Pins	Full Hierarchy Name
1074 (7)	2106 (0)	0 (0)	4736	1	4	40	0	|matrices_mult
91 (1)	92 (0)	0 (0)	0	0	0	0	0	|matrices_mult|sld_hub:auto_hub
255 (2)	745 (74)	0 (0)	4736	1	0	0	0	|matrices_mult|sld_signaltap:auto_signaltap_0
7 (7)	0 (0)	0 (0)	0	0	0	0	0	|matrices_mult|bcd_to_7seg:bcd_to_7seg_inst0
7 (7)	0 (0)	0 (0)	0	0	0	0	0	|matrices_mult|bcd_to_7seg:bcd_to_7seg_inst1
7 (7)	0 (0)	0 (0)	0	0	0	0	0	|matrices_mult|bcd_to_7seg:bcd_to_7seg_inst2
7 (7)	0 (0)	0 (0)	0	0	0	0	0	|matrices_mult|bcd_to_7seg:bcd_to_7seg_inst3
66 (66)	38 (38)	0 (0)	0	0	0	0	0	|matrices_mult|bin2bcd_12bit_sync:bin2bcd_inst
33 (33)	20 (20)	0 (0)	0	0	0	0	0	|matrices_mult|data_generator:data_gen
575 (88)	1186 (57)	0 (0)	0	0	4	0	0	|matrices_mult|main_controller:main_ctrl
17 (17)	17 (17)	0 (0)	0	0	0	0	0	|matrices_mult|num_convert:num_convert_inst
1 (1)	4 (4)	0 (0)	0	0	0	0	0	|matrices_mult|sync_diff:sync_diff_DISPLAY
1 (1)	4 (4)	0 (0)	0	0	0	0	0	|matrices_mult|sync_diff:sync_diff_START

3.4 הקצאת פינים.

הקצאת הפינים נעשתה דרך קובץ שהורדנו מהמודל והוספנו אותו לפרויקט

CLK	Input	PIN_R20	5B	B5B_N0	3.3-V LVTTL		16mA (default)	
DISPLAYn	Input	PIN_Y15	4A	B4A_N0	1.2 V		8mA (default)	
HEX0[6]	Output	PIN_Y18	4A	B4A_N0	1.2 V		8mA (default)	1 (default)
HEX0[5]	Output	PIN_Y19	4A	B4A_N0	1.2 V		8mA (default)	1 (default)
HEX0[4]	Output	PIN_Y20	4A	B4A_N0	1.2 V		8mA (default)	1 (default)
HEX0[3]	Output	PIN_W18	4A	B4A_N0	1.2 V		8mA (default)	1 (default)
HEX0[2]	Output	PIN_V17	4A	B4A_N0	1.2 V		8mA (default)	1 (default)
HEX0[1]	Output	PIN_V18	4A	B4A_N0	1.2 V		8mA (default)	1 (default)
HEX0[0]	Output	PIN_V19	4A	B4A_N0	1.2 V		8mA (default)	1 (default)
HEX1[6]	Output	PIN_AF24	4A	B4A_N0	1.2 V		8mA (default)	1 (default)
HEX1[5]	Output	PIN_AC19	4A	B4A_N0	1.2 V		8mA (default)	1 (default)
HEX1[4]	Output	PIN_AE25	4A	B4A_N0	1.2 V		8mA (default)	1 (default)
HEX1[3]	Output	PIN_AE26	4A	B4A_N0	1.2 V		8mA (default)	1 (default)
HEX1[2]	Output	PIN_AB19	4A	B4A_N0	1.2 V		8mA (default)	1 (default)
HEX1[1]	Output	PIN_AD26	4A	B4A_N0	1.2 V		8mA (default)	1 (default)
HEX1[0]	Output	PIN_AA18	4A	B4A_N0	1.2 V		8mA (default)	1 (default)
HEX2[6]	Output	PIN_W20	5A	B5A_N0	3.3-V LVTTL		16mA (default)	1 (default)
HEX2[5]	Output	PIN_W21	5A	B5A_N0	3.3-V LVTTL		16mA (default)	1 (default)
HEX2[4]	Output	PIN_V20	5A	B5A_N0	3.3-V LVTTL		16mA (default)	1 (default)
HEX2[3]	Output	PIN_V22	5A	B5A_N0	3.3-V LVTTL		16mA (default)	1 (default)
HEX2[2]	Output	PIN_U20	5A	B5A_N0	3.3-V LVTTL		16mA (default)	1 (default)
HEX2[1]	Output	PIN_AD6	3A	B3A_N0	3.3-V LVTTL		16mA (default)	1 (default)
HEX2[0]	Output	PIN_AD7	3A	B3A_N0	3.3-V LVTTL		16mA (default)	1 (default)
HEX3[6]	Output	PIN_AC22	5A	B5A_N0	3.3-V LVTTL		16mA (default)	1 (default)
HEX3[5]	Output	PIN_AC23	5A	B5A_N0	3.3-V LVTTL		16mA (default)	1 (default)
HEX3[4]	Output	PIN_AC24	5A	B5A_N0	3.3-V LVTTL		16mA (default)	1 (default)
HEX3[3]	Output	PIN_AA22	5A	B5A_N0	3.3-V LVTTL		16mA (default)	1 (default)
HEX3[2]	Output	PIN_AA23	5A	B5A_N0	3.3-V LVTTL		16mA (default)	1 (default)
HEX3[1]	Output	PIN_Y23	5A	B5A_N0	3.3-V LVTTL		16mA (default)	1 (default)
HEX3[0]	Output	PIN_Y24	5A	B5A_N0	3.3-V LVTTL		16mA (default)	1 (default)
LED_SIGN	Output	PIN_H9	8A	B8A_N0	2.5 V		12mA (default)	1 (default)
LEDG[3]	Output	PIN_E9	8A	B8A_N0	2.5 V		12mA (default)	1 (default)
LEDG[2]	Output	PIN_D8	8A	B8A_N0	2.5 V		12mA (default)	1 (default)
LEDG[1]	Output	PIN_K6	8A	B8A_N0	2.5 V		12mA (default)	1 (default)
LEDS_1E4[3]	Output	PIN_J10	8A	B8A_N0	2.5 V		12mA (default)	1 (default)
LEDS_1E4[2]	Output	PIN_H7	8A	B8A_N0	2.5 V		12mA (default)	1 (default)
LEDS_1E4[1]	Output	PIN_K8	8A	B8A_N0	2.5 V		12mA (default)	1 (default)
LEDS_1E4[0]	Output	PIN_K10	8A	B8A_N0	2.5 V		12mA (default)	1 (default)
RSTn	Input	PIN_P11	3B	B3B_N0	1.2 V		8mA (default)	

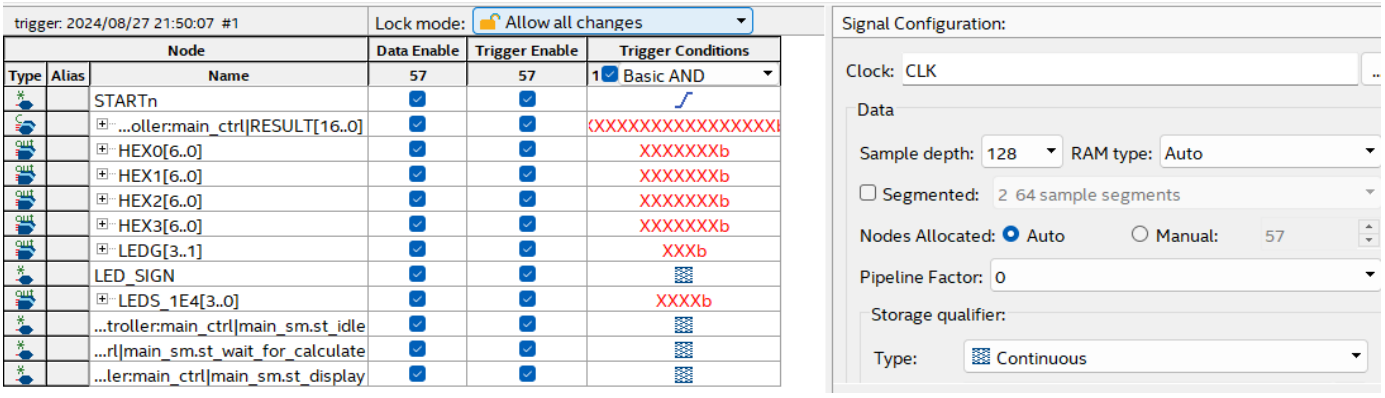


קובץ Do:

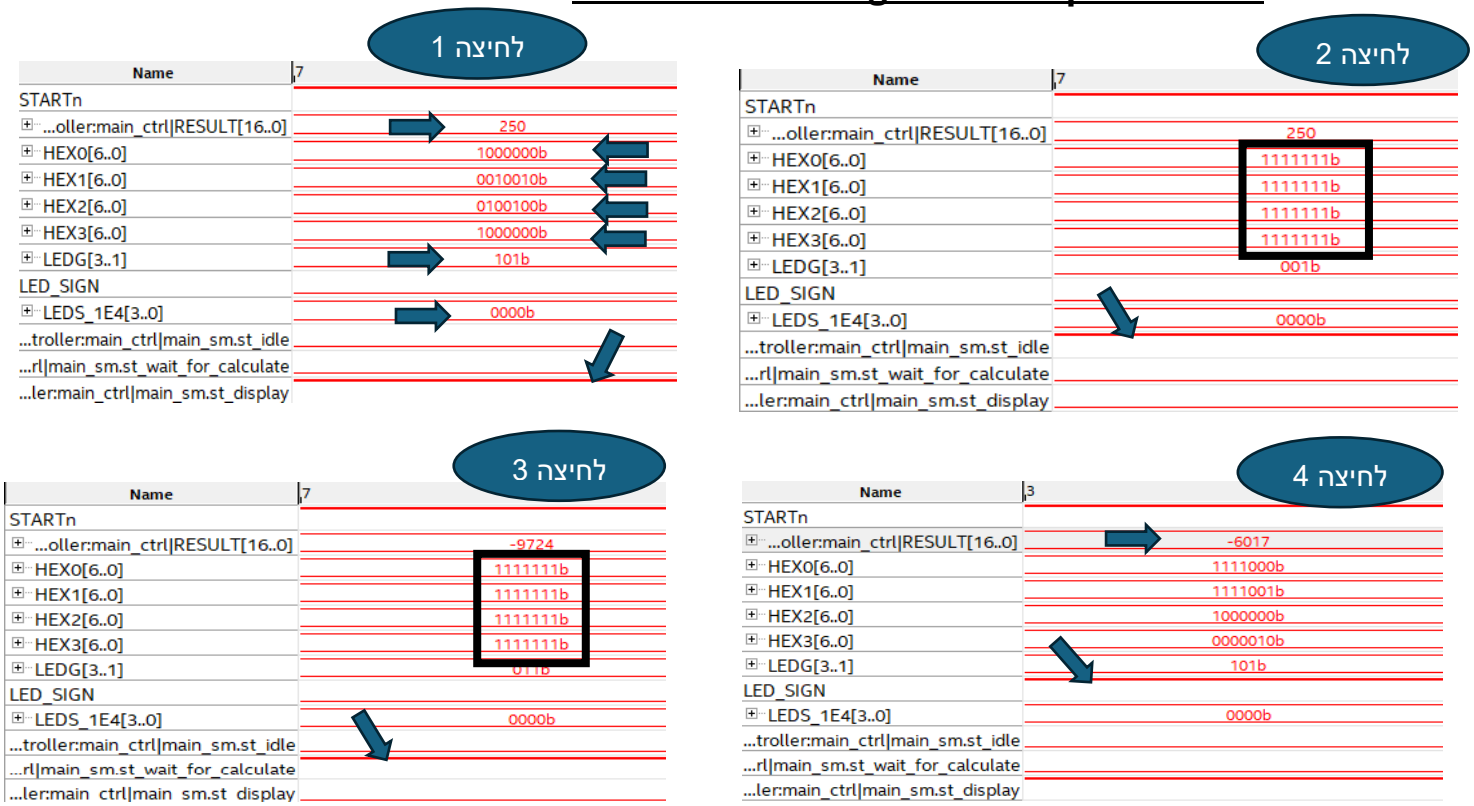
```
# Turn on transcript to log all console output to a file
transcript on

# Check if the library 'rtl_work' exists and delete it if it does
if {[file exists rtl_work]} {
    vdel -lib rtl_work -all
}

# Create a new library 'rtl_work' for the simulation
vlib rtl_work

# Map the logical library 'work' to the 'rtl_work' physical library
vmap work rtl_work

# Compile all the VHDL source files into the 'work' library
vcom -2008 -work work {C:/Users/aboud/VHDL2024B/FinalProject/src/sync_diff.vhd}
vcom -2008 -work work {C:/Users/aboud/VHDL2024B/FinalProject/src/num_convert.vhd}
vcom -2008 -work work {C:/Users/aboud/VHDL2024B/FinalProject/src/my_multiplier.vhd}
vcom -2008 -work work {C:/Users/aboud/VHDL2024B/FinalProject/src/matrix_ram.vhd}
vcom -2008 -work work {C:/Users/aboud/VHDL2024B/FinalProject/src/matrices_mult.vhd}
vcom -2008 -work work {C:/Users/aboud/VHDL2024B/FinalProject/src/main_controller.vhd}
vcom -2008 -work work {C:/Users/aboud/VHDL2024B/FinalProject/src/data_generator_pack.vhd}
vcom -2008 -work work {C:/Users/aboud/VHDL2024B/FinalProject/src/bin2bcd_12bit_sync.vhd}
vcom -2008 -work work {C:/Users/aboud/VHDL2024B/FinalProject/src/bcd_to_7seg.vhd}
vcom -2008 -work work {C:/Users/aboud/VHDL2024B/FinalProject/src/data_generator.vhd}

# Compile the testbench file
vcom -2008 -work work {C:/Users/aboud/VHDL2024B/FinalProject/par/../src/matrices_mult_tb.vhd}

# Simulate the testbench with specific libraries and options
vsim -t 1ps -L altera -L lpm -L sgate -L altera_mf -L altera_lnsim -L cyclonev -L rtl_work -L work -voptargs="+acc"  matrices_mult_tb

# Add all signals to the waveform view, grouped by their hierarchy
add wave -group matrices_mult_tb/* -- Add all top-level testbench signals
add wave -group matrices_mult matrices_mult_tb/dut/* -- Add all signals from the DUT

# Add signals from specific components in the design under test (DUT)
add wave -group data_generator matrices_mult_tb/dut/data_gen/*
add wave -group main_controller matrices_mult_tb/dut/main_ctrl/*
add wave -group matrix_ram_inst matrices_mult_tb/dut/main_ctrl/matrix_ram_inst/*
add wave -group mult1 matrices_mult_tb/dut/main_ctrl/mult1/*
add wave -group mult2 matrices_mult_tb/dut/main_ctrl/mult2/*
add wave -group mult3 matrices_mult_tb/dut/main_ctrl/mult3/*
add wave -group mult4 matrices_mult_tb/dut/main_ctrl/mult4/*

add wave -group bin2bcd_12bit_sync matrices_mult_tb/dut/bin2bcd_inst/*
add wave -group num_convert matrices_mult_tb/dut/num_convert_inst/*

add wave -group bcd_to_7seg_inst0 matrices_mult_tb/dut/bcd_to_7seg_inst0/*
add wave -group bcd_to_7seg_inst1 matrices_mult_tb/dut/bcd_to_7seg_inst1/*
add wave -group bcd_to_7seg_inst2 matrices_mult_tb/dut/bcd_to_7seg_inst2/*
add wave -group bcd_to_7seg_inst3 matrices_mult_tb/dut/bcd_to_7seg_inst3/*

add wave -group sync_diff_START matrices_mult_tb/dut/sync_diff_START/*
add wave -group sync_diff_DISPLAY matrices_mult_tb/dut/sync_diff_DISPLAY/*

# Open the structure and signals views in the simulator GUI
view structure
view signals

# Run the simulation indefinitely or until manually stopped
run -all
```

8) בדיקת המערכת בעזרת Signal TAP

❖ יצירת קובץ Signal TAP לבדיקת המערכת ללחיצת STARTn



❖ סימולציה דרך ה Signal TAP ללחיצת STARTn



הסבר איטרציות (העברה ממטריציה A ל B):

❖ בלחיצה STARTn 1:

אנחנו נמצאים במטריציה A, התוצאה 250:

הערך 250 הוא תוצאת החישוב הראשון במטריציה A שמערכת הפיקה, והוא זה שאמור להיות מוצג על תצוגות ה-segment-7.

תצוגות ה-segment-7 (HEX0 עד HEX3):

כל HEX מייצג תצוגת segment-7 אחת:

HEX0: מראה b1000000 שזהו הקוד להצגת הספרה 0 על תצוגת segment-7.

HEX1: מראה b0100000 שזהו הקוד להצגת הספרה 2.

HEX2: מראה b0010010 שזהו הקוד להצגת הספרה 5.

HEX3: מראה b1000000 שזהו הקוד להצגת הספרה 0.

נורות ה-LED (LEDG ו-LED_SIGN):

LEDG[3:1] מציגים את מצב ה-LED הירוק: b101 מציין שה-LED הראשון והשלישי דולקים והשני כבוי.

LED_SIGN: ערך 0 מציין שהתוצאה חיובית (ללא סימן מינוס).

מצבי מכונת המצבים:

מכונת המצבים נמצאת במעבר בין המצבים **st_idle, st_wait_for_calculate**, ו-**st_display**, מה שמראה שהמערכת בהתחלה מחכה לפעולה (**idle**), ואז ממתינה להתחלת החישוב (**wait_for_calculate**), ולאחר מכן מציגה את התוצאה (**display**).

❖ בלחיצה STARTn 2 ו- 3:

כל HEX מייצג תצוגת segment-7 אחת שהי b1111111 שזהו הקוד לכבוי התצוגה

מכונת המצבים עברה למצב **idle** ואחר כך למצב **wait_for_calculate**

❖ בלחיצה STARTn 4:

אנחנו עברנו למטריציה B, התוצאה 6017-:

הערך 6017- הוא תוצאת החישוב הראשון במטריציה B שמערכת הפיקה.

LED_SIGN: ערך 1 מציין שהתוצאה שלילי (עם סימן מינוס).

❖ יצירת קובץ Signal TAP לבדיקת המערכת ללחיצת DISPALYn

trigger: 2024/08/27 23:05:05 #1 | Lock mode: Allow all changes

Node			Data Enable	Trigger Enable	Trigger Conditions
Type	Alias	Name	55	55	1 Basic AND
		DISPLAYn			
		...oller:main_ctrl|RESULT[16..0]			XXXXXXXXXXXXXXXXX
		HEX0[6..0]			XXXXXXXb
		HEX1[6..0]			XXXXXXXb
		HEX2[6..0]			XXXXXXXb
		HEX3[6..0]			XXXXXXXb
		LEDG[3..1]			XXXb
		LED_SIGN			
		LEDS_1E4[3..0]			XXXXb
		...ler:main_ctrl|main_sm.st_display			

Signal Configuration:

Clock: CLK

Data

Sample depth: 128 RAM type: Auto

Segmented: 2 64 sample segments

Nodes Allocated: Auto Manual: 55

Pipeline Factor: 0

Storage qualifier:

❖ סימולציה דרך ה Signal TAP ללחיצת DISPLAYn

Sign	Value
0	250
0	260
0	270
0	280
0	618
0	644
0	670
0	696
0	986
0	1028
0	1070
0	1112
0	1354
0	1412
0	1470
0	1528

לחיצה 1

Name	
DISPLAYn	
...oller:main_ctrl|RESULT[16..0]	260
HEX0[6..0]	1000000b
HEX1[6..0]	0000010b
HEX2[6..0]	0100100b
HEX3[6..0]	1000000b
LEDG[3..1]	101b
LED_SIGN	
LEDS_1E4[3..0]	0000b
...ler:main_ctrl|main_sm.st_display	

לחיצה 2

Name	
DISPLAYn	
...oller:main_ctrl|RESULT[16..0]	270
HEX0[6..0]	1000000b
HEX1[6..0]	1111000b
HEX2[6..0]	0100100b
HEX3[6..0]	1000000b
LEDG[3..1]	101b
LED_SIGN	
LEDS_1E4[3..0]	0000b
...ler:main_ctrl|main_sm.st_display	

לחיצה 3

Name	
DISPLAYn	
...oller:main_ctrl|RESULT[16..0]	280
HEX0[6..0]	1000000b
HEX1[6..0]	0000000b
HEX2[6..0]	0100100b
HEX3[6..0]	1000000b
LEDG[3..1]	101b
LED_SIGN	
LEDS_1E4[3..0]	0000b
...ler:main_ctrl|main_sm.st_display	

לחיצה 4

DISPLAYn	
...oller:main_ctrl|RESULT[16..0]	618
HEX0[6..0]	0000000b
HEX1[6..0]	1111001b
HEX2[6..0]	0000010b
HEX3[6..0]	1000000b
LEDG[3..1]	101b
LED_SIGN	
LEDS_1E4[3..0]	0000b
...ler:main_ctrl|main_sm.st_display	

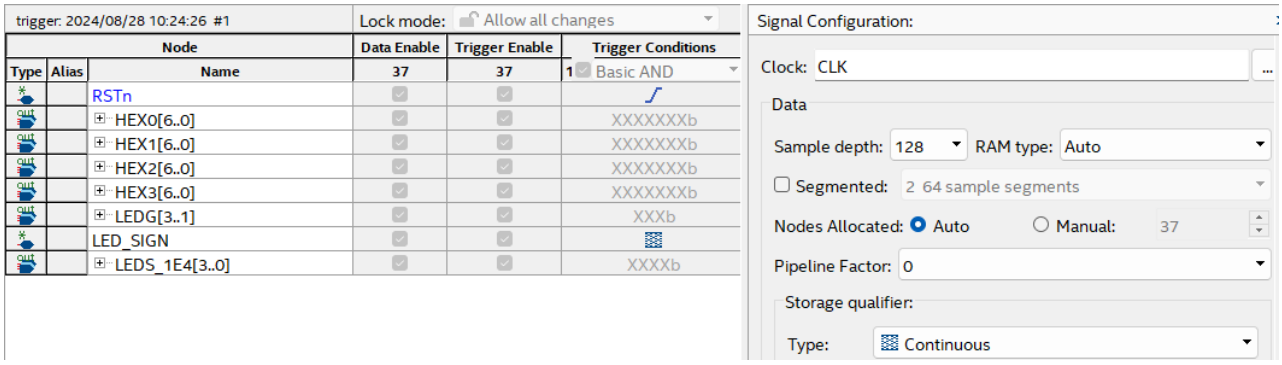
הסבר איטרציות (העברה במטריצה A):

❖ בלחיצה DISPLAYn:

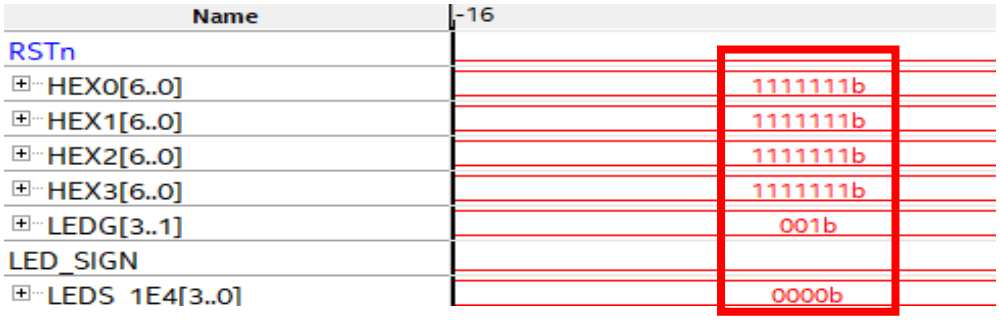
התוצאה עברה לאיבר השני 260

ובכל לחיצה על DISPLAYn התוצאה תעבור להאיבר הבא

❖ יצירת קובץ Signal TAP לבדיקת המערכת ללחיצת RSTn



סימולציה דרך ה Signal TAP ללחיצת RSTn



❖ בלחיצה RSTn:

כל HEX מייצג תצוגת segment-7 אחת שהי b1111111 שזהו הקוד לכבוי התצוגה

ובכל לחיצה על DISPLAYn התוצאה תעבור להאיבר הבא

LEDG[3:1] מציגים את מצב ה-LED הירוק: b000 מציין שכל ה-LEDs כבוים.

LED_SIGN: ערך 0 מציין שה-LED כבויה.

9) סרטון הדגמה:

https://www.youtube.com/watch?v=htl3920-ROM