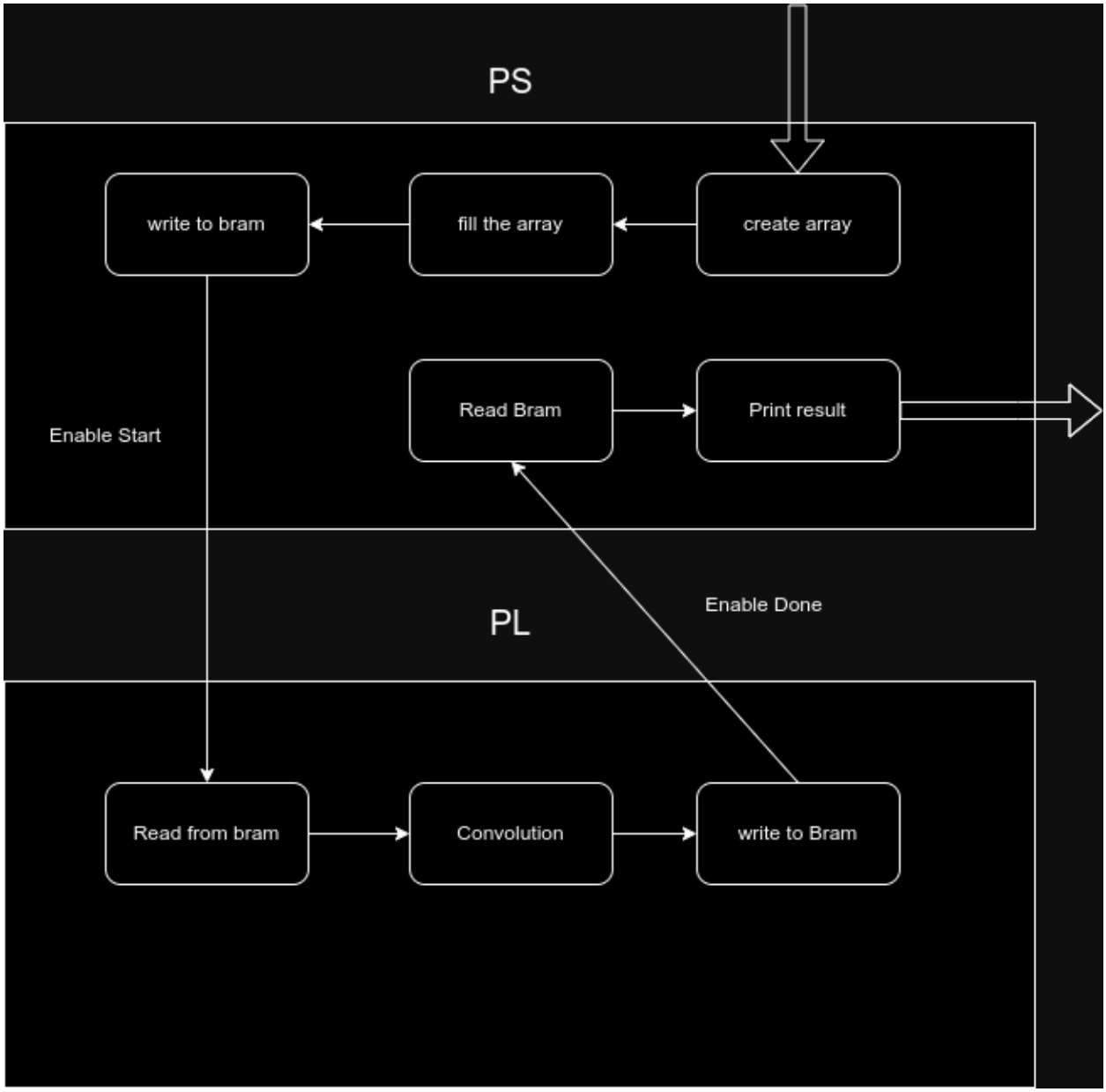
به نام خدا

تهیه کننده: ابراهیم صدیقی شماره دانشجویی: ۹۹۳۱۰۹۸

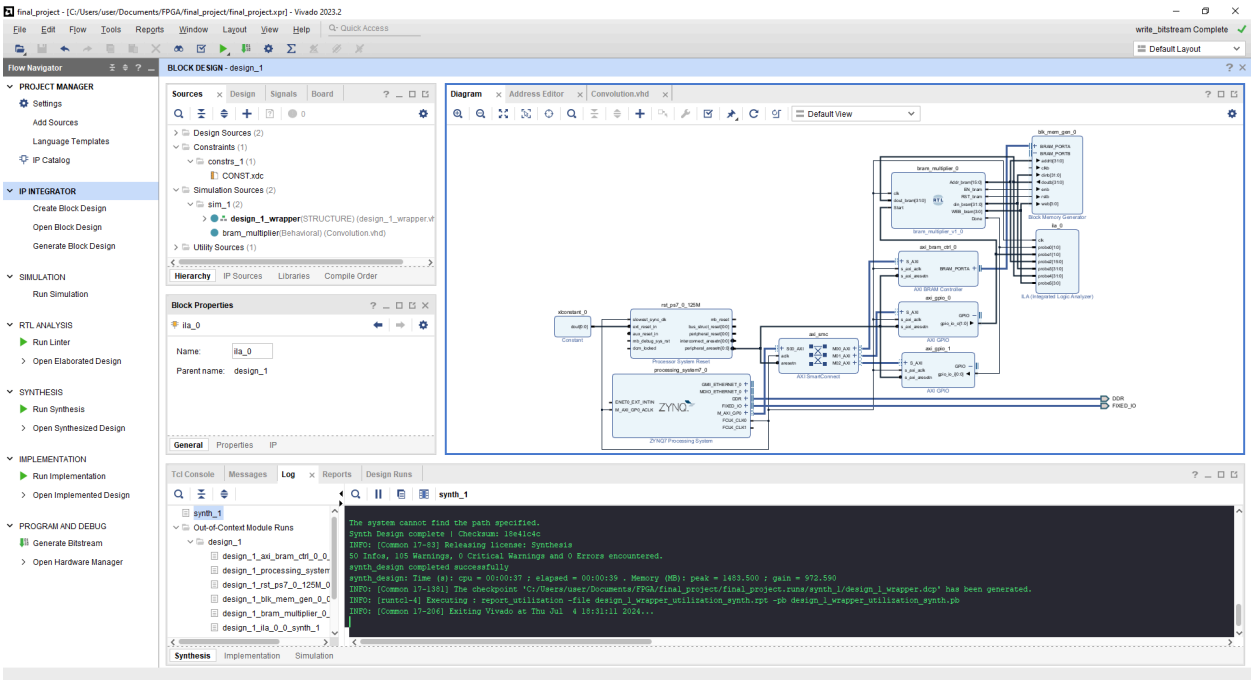
در ابتدا برای پیاده سازی یک شتاب دهنده ابتدا باید با استفاده از co-design قسمت های مختلف PL و PS را طراحی کنیم.
نمودار کلی این طراحی به شکل زیر است:



شکل بالا دو قسمت PL و PS را این‌گونه پیاده‌سازی کرده است که ابتدا یک آرایه اصلی و یک آرایه kernel در قسمت PS ساخته می‌شود، سپس این آرایه‌ها مقداردهی می‌شوند و بعد از آن، آرایه‌ها به داخل BRAM نوشته می‌شوند. در قسمت PL نیز برنامه در صورت فعال شدن پین start شروع به خواندن BRAM میکند، آرایه‌ها در قسمت PL خوانده می‌شوند و عملیات

convolution بر روی آرایه اصلی اجرا می‌شود. خروجی به BRAM نوشته می‌شود و پین done را فعال میکندو سپس PS آن را می‌خواند و خروجی را بازمی‌گرداند.

برای پیاده‌سازی این سیستم، ابتدا یک پروژه جدید در Vivado ایجاد می‌کنیم. برد EBAZ4205 Development Board را به پروژه خود اضافه می‌کنیم. سپس Block Design را باز می‌کنیم و سه IP با نام‌های AXI BRAM Controller، ZYNQ 7، و Block Memory Generator را اضافه می‌کنیم. حال یک فایل source جدید باز می‌کنیم و کد VHDL مربوط به پیاده‌سازی Convolution را در آن قرار می‌دهیم. سپس با زدن دکمه Auto Generate مشاهده می‌کنیم که IPها به هم متصل می‌شوند.



حال به کد vhdl می پردازیم :

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.numeric_std.all;

entity bram_multiplier is
    Port (
        clk : in  STD_LOGIC;
        Addr_bram   : out std_logic_vector(15 downto 0);
        EN_bram     : out STD_LOGIC := '0';
        dout_bram   : in  std_logic_vector(31 downto 0);
        RST_bram    : out STD_LOGIC := '1';
        din_bram    : out std_logic_vector(31 downto 0);
        WEB_bram    : out std_logic_vector(3 downto 0);
        Done        : out STD_LOGIC := '0';
        Start       : in  STD_LOGIC
    );
end bram_multiplier;

architecture Behavioral of bram_multiplier is
    constant MATRIX_SIZE : integer := 28;
    constant KERNEL_SIZE : integer := 3;

    signal adr_s  : std_logic_vector(15 downto 0) := (others => '0');
    signal RData1     : std_logic_vector(31 downto 0) := (others => '0');
    signal RData2     : std_logic_vector(31 downto 0) := (others => '0');
    signal WEB_S : std_logic_vector(3 downto 0) := (others => '0');
    signal product : std_logic_vector(31 downto 0) := (others => '0');
    signal Is_Run : STD_LOGIC := '0';
    signal cnt : integer := 0;

    type matrix_28x28 is array (0 to MATRIX_SIZE-1, 0 to MATRIX_SIZE-1) of std_logic_vector(31 downto 0);
    type matrix_3x3 is array (0 to KERNEL_SIZE-1, 0 to KERNEL_SIZE-1) of std_logic_vector(31 downto 0);

    signal input_matrix : matrix_28x28 := (others => (others => (others => '0')));
    signal kernel_matrix : matrix_3x3 := (others => (others => (others => '0')));
```

```
    type matrix_28x28 is array (0 to MATRIX_SIZE-1, 0 to MATRIX_SIZE-1) of std_logic_vector(31 downto 0);
    type matrix_3x3 is array (0 to KERNEL_SIZE-1, 0 to KERNEL_SIZE-1) of std_logic_vector(31 downto 0);

    signal input_matrix : matrix_28x28 := (others => (others => (others => '0')));
    signal kernel_matrix : matrix_3x3 := (others => (others => (others => '0')));
    signal result_matrix : matrix_28x28 := (others => (others => (others => '0')));

    signal current_row : integer := 0;
    signal current_col : integer := 0;
    signal kernel_row : integer := 0;
    signal kernel_col : integer := 0;
begin

    Addr_bram <= adr_s;
    RST_bram <= '0';
    EN_bram <= '1';
    WEB_bram <= WEB_S;

    process(clk)
    begin
        if rising_edge(clk) then
            if (Start='1') and (Is_Run = '0') then
                Is_Run <= '1';
                Done <= '0';
                cnt <= 0;
                WEB_S <= "0000";
                adr_s <= (others => '0');
                current_row <= 0;
                current_col <= 0;
                kernel_row <= 0;
                kernel_col <= 0;
            elsif (Is_Run = '1') then
                if cnt < (MATRIX_SIZE * MATRIX_SIZE) then
```

```
                if cnt < (MATRIX_SIZE * MATRIX_SIZE) then
                    input_matrix(current_row, current_col) <= dout_bram;
                    adr_s <= std_logic_vector(to_unsigned(cnt + 1, 16));
                    cnt <= cnt + 1;
                    current_col <= (current_col + 1) mod MATRIX_SIZE;
                    if current_col = 0 then
                        current_row <= current_row + 1;
                    end if;
                elsif cnt < (MATRIX_SIZE * MATRIX_SIZE + KERNEL_SIZE * KERNEL_SIZE) then
                    kernel_matrix(kernel_row, kernel_col) <= dout_bram;
                    adr_s <= std_logic_vector(to_unsigned(cnt + 1, 16));
                    cnt <= cnt + 1;
                    kernel_col <= (kernel_col + 1) mod KERNEL_SIZE;
                    if kernel_col = 0 then
                        kernel_row <= kernel_row + 1;
                    end if;
                else
                    -- Perform convolution
                    for i in 0 to MATRIX_SIZE-KERNEL_SIZE loop
                        for j in 0 to MATRIX_SIZE-KERNEL_SIZE loop
                            product <= (others => '0');
                            for ki in 0 to KERNEL_SIZE-1 loop
                                for kj in 0 to KERNEL_SIZE-1 loop
                                    product <= std_logic_vector(
                                        resize(
                                            unsigned(product) + unsigned(input_matrix(i+ki, j+kj)) * unsigned(kernel_matrix(ki, kj)), 32
                                        )
                                    );
                                end loop;
                            end loop;
                            result_matrix(i, j) <= product;
                        end loop;
                    end loop;
```

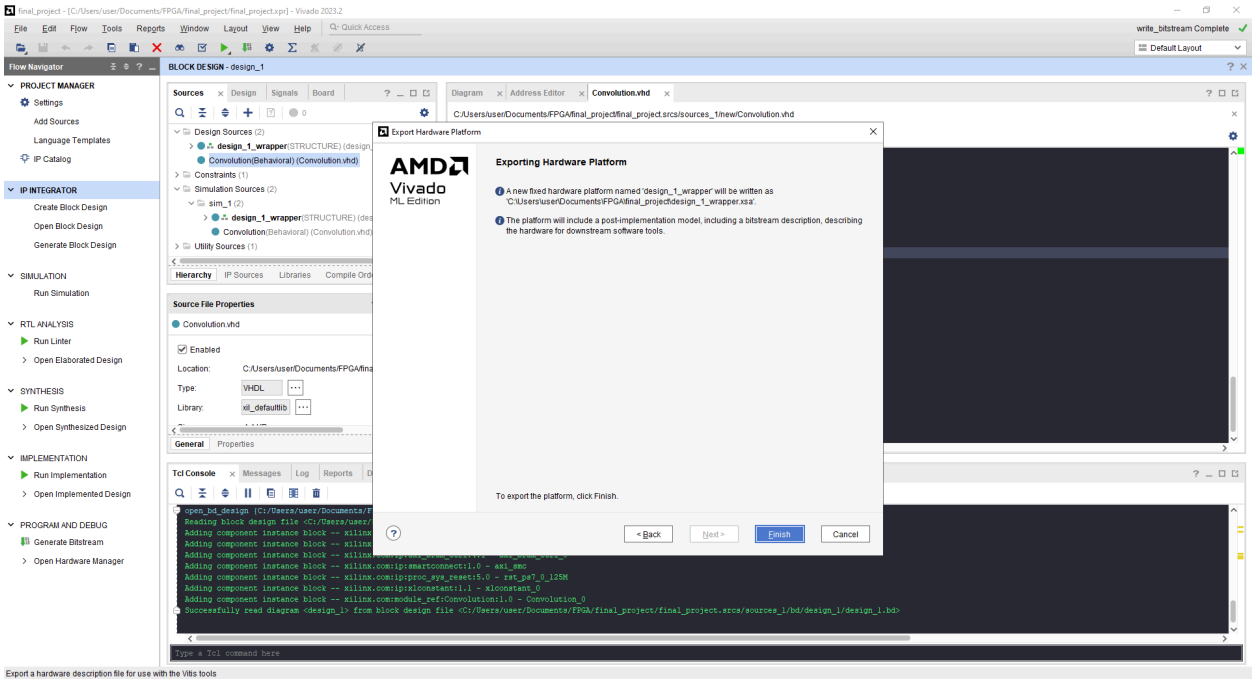
```
                    end loop;

                    -- Write result back to BRAM
                    if cnt < (MATRIX_SIZE * MATRIX_SIZE + KERNEL_SIZE * KERNEL_SIZE + MATRIX_SIZE * MATRIX_SIZE) then
                        adr_s <= std_logic_vector(to_unsigned(cnt + 1, 16));
                        din_bram <= result_matrix(current_row, current_col);
                        WEB_S <= "1111";
                        cnt <= cnt + 1;
                        current_col <= (current_col + 1) mod MATRIX_SIZE;
                        if current_col = 0 then
                            current_row <= current_row + 1;
                        end if;
                    else
                        Done <= '1';
                        if Start='0' then
                            Is_Run <= '0';
                        end if;
                    end if;
                end if;
            end if;
        end if;
    end process;

end Behavioral;
```

این کد VHDL یک ماژول به نام `bram_multiplier` را پیاده‌سازی می‌کند که وظیفه انجام ضرب ماتریسی (کانولوشن) را با استفاده از حافظه بلوکی (BRAM) بر عهده دارد. این ماژول شامل پورت‌هایی برای سیگنال ساعت (clk)، آدرس BRAM، سیگنال فعال‌سازی (EN_bram)، داده‌های خروجی از BRAM (dout_bram)، سیگنال بازنشانی (RST_bram)، داده‌های ورودی به BRAM (din_bram)، سیگنال‌های نوشتن به BRAM (WEB_bram)، سیگنال اتمام عملیات (Done) و سیگنال شروع عملیات (Start) است.

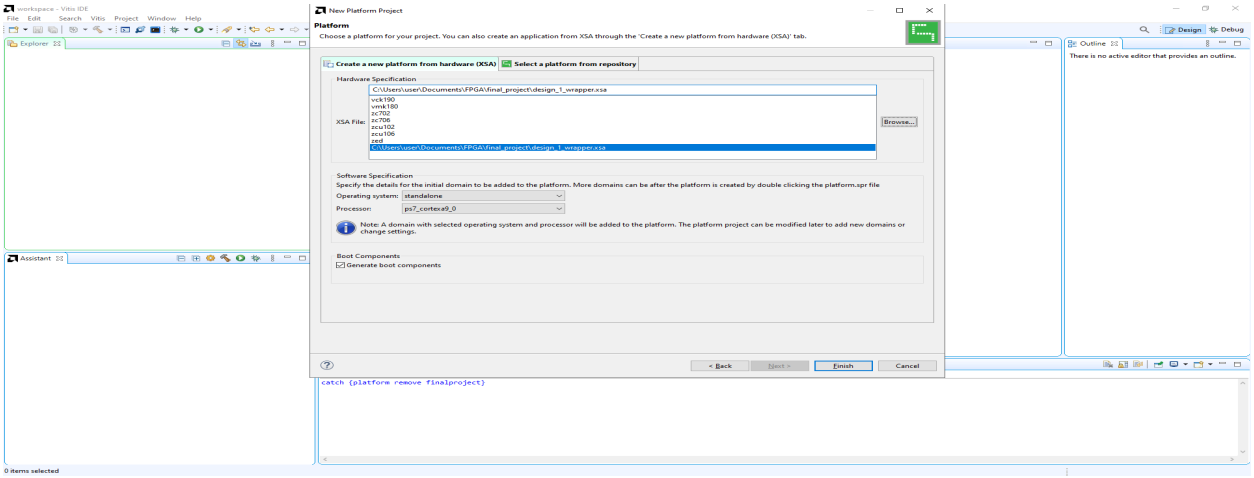
در معماری Behavioral این ماژول، ثابت‌هایی برای اندازه ماتریس ورودی (۲۸x۲۸) و اندازه کرنل (۳x۳) تعریف شده‌اند. سیگنال‌های داخلی برای آدرس‌دهی، داده‌های خوانده شده، داده‌های نوشته شده، سیگنال‌های نوشتن، محصول نهایی، وضعیت اجرای عملیات و شمارنده تعریف شده‌اند. همچنین، انواع داده‌های ماتریسی برای ذخیره ماتریس ورودی، کرنل و نتیجه عملیات نیز تعریف شده‌اند.

فرآیند اصلی با فعال شدن سیگنال `Start` آغاز می‌شود. در این مرحله، داده‌های ماتریس ورودی و کرنل از BRAM خوانده می‌شوند. سپس، عملیات کانولوشن بر روی ماتریس ورودی با استفاده از کرنل انجام می‌شود و نتیجه در ماتریس نتیجه ذخیره می‌گردد. این عملیات شامل جمع حاصلضرب عناصر ماتریس ورودی با کرنل است. پس از اتمام کانولوشن، نتایج به BRAM نوشته می‌شوند و سیگنال Done فعال می‌شود تا نشان دهد که عملیات به پایان رسیده است.

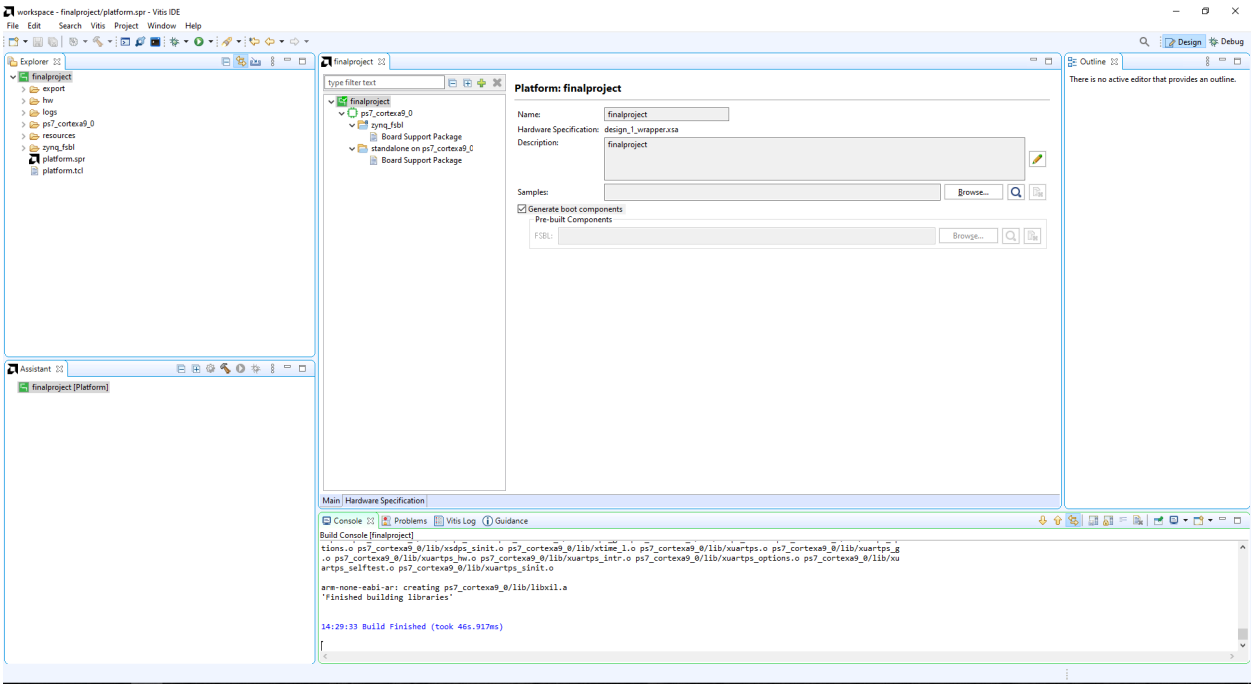
پروژه را سنتز میکنیم و سخت افزار را export میکنیم.



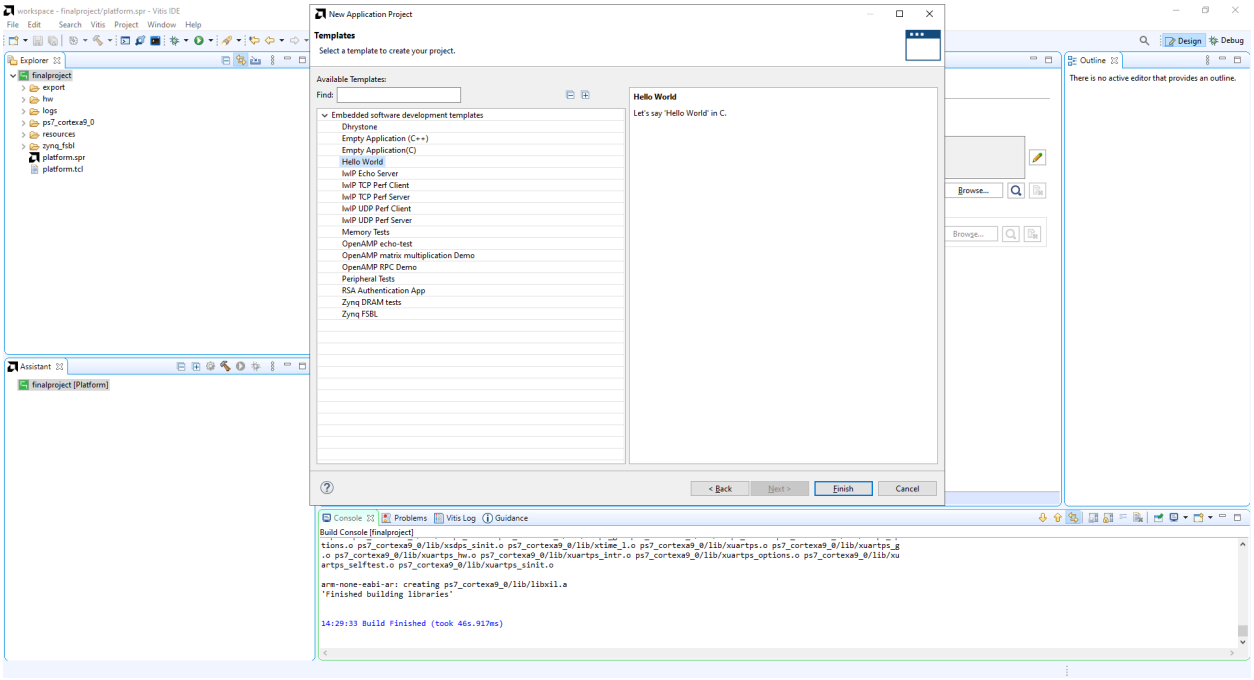
در مرحله بعد vitis classic را باز میکنیم. در بخش new یک platform project را ایجاد میکنیم و فایل سخت افزار export شده را به نرم افزار میدهیم.

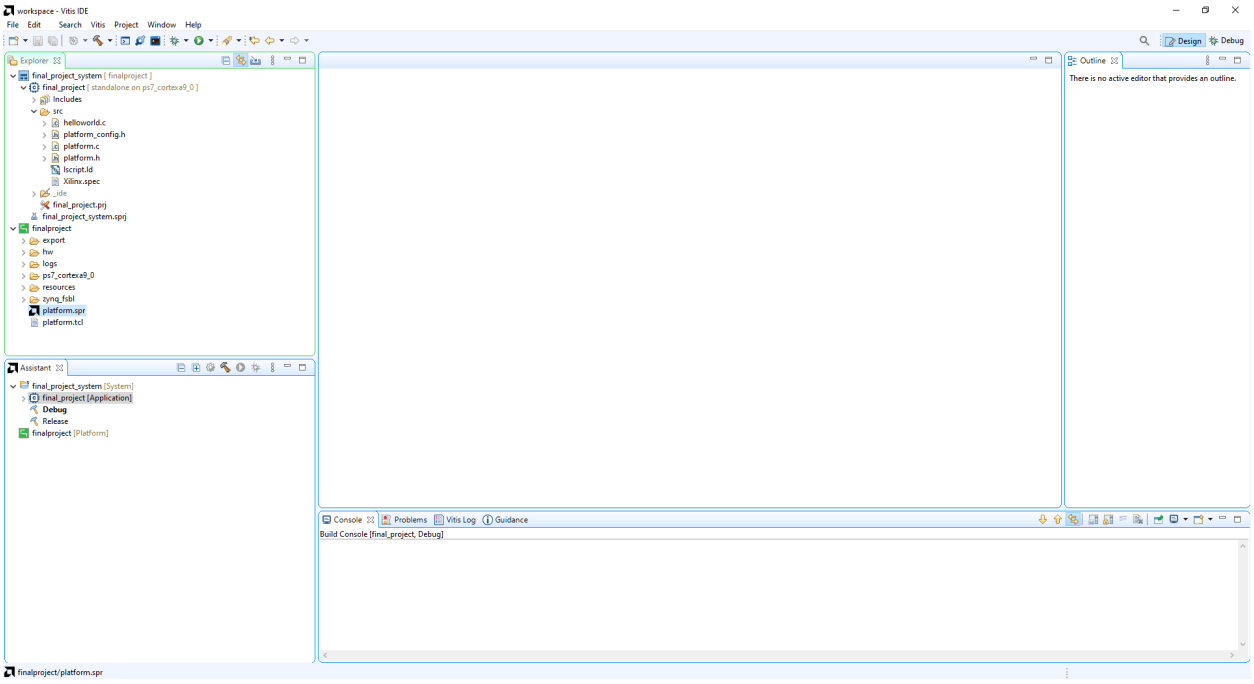


پروژه را build میکنیم.

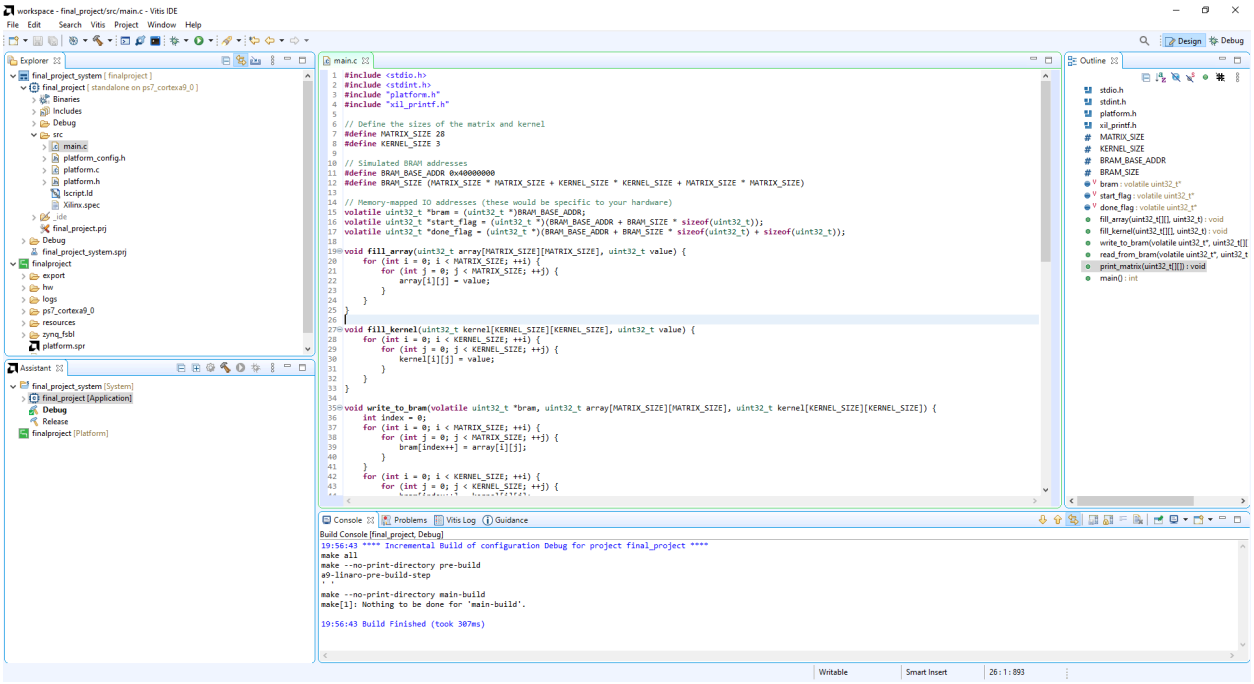


حال یک application project میسازیم. سخت افزار را انتخاب میکنیم و به عنوان template از hello world استفاده میکنیم.





درون پوشه src فایل helloworld.c را به main.c تغییر می دهیم و درون فایل main.c کد مربوط به ps خود را اضافه میکنیم. حال پروژه نرم افزاری را build میکنیم.



حال به کد c می پردازیم:

```c
#include <stdio.h>
#include <stdint.h>
#include "platform.h"
#include "xil_printf.h"

// Define the sizes of the matrix and kernel
#define MATRIX_SIZE 28
#define KERNEL_SIZE 3

// Simulated BRAM addresses
#define BRAM_BASE_ADDR 0x40000000
#define BRAM_SIZE (MATRIX_SIZE * MATRIX_SIZE + KERNEL_SIZE * KERNEL_SIZE + MATRIX_SIZE * MATRIX_SIZE)

// Memory-mapped IO addresses (these would be specific to your hardware)
volatile uint32_t *bram = (uint32_t *)BRAM_BASE_ADDR;
volatile uint32_t *start_flag = (uint32_t *)(BRAM_BASE_ADDR + BRAM_SIZE * sizeof(uint32_t));
volatile uint32_t *done_flag = (uint32_t *)(BRAM_BASE_ADDR + BRAM_SIZE * sizeof(uint32_t) + sizeof(uint32_t));

void fill_array(uint32_t array[MATRIX_SIZE][MATRIX_SIZE], uint32_t value) {
    for (int i = 0; i < MATRIX_SIZE; ++i) {
        for (int j = 0; j < MATRIX_SIZE; ++j) {
            array[i][j] = value;
        }
    }
}

void fill_kernel(uint32_t kernel[KERNEL_SIZE][KERNEL_SIZE], uint32_t value) {
    for (int i = 0; i < KERNEL_SIZE; ++i) {
        for (int j = 0; j < KERNEL_SIZE; ++j) {
            kernel[i][j] = value;
        }
    }
}

void write_to_bram(volatile uint32_t *bram, uint32_t array[MATRIX_SIZE][MATRIX_SIZE], uint32_t kernel[KERNEL_SIZE][KERNEL_SIZE]) {
    int index = 0;
    for (int i = 0; i < MATRIX_SIZE; ++i) {
        for (int j = 0; j < MATRIX_SIZE; ++j) {
            bram[index++] = array[i][j];
        }
    }
    for (int i = 0; i < KERNEL_SIZE; ++i) {
        for (int j = 0; j < KERNEL_SIZE; ++j) {
            bram[index++] = kernel[i][j];
        }
    }
}

void read_from_bram(volatile uint32_t *bram, uint32_t result[MATRIX_SIZE][MATRIX_SIZE]) {
    int index = MATRIX_SIZE * MATRIX_SIZE + KERNEL_SIZE * KERNEL_SIZE;
    for (int i = 0; i < MATRIX_SIZE; ++i) {
        for (int j = 0; j < MATRIX_SIZE; ++j) {
            result[i][j] = bram[index++];
        }
    }
}
```

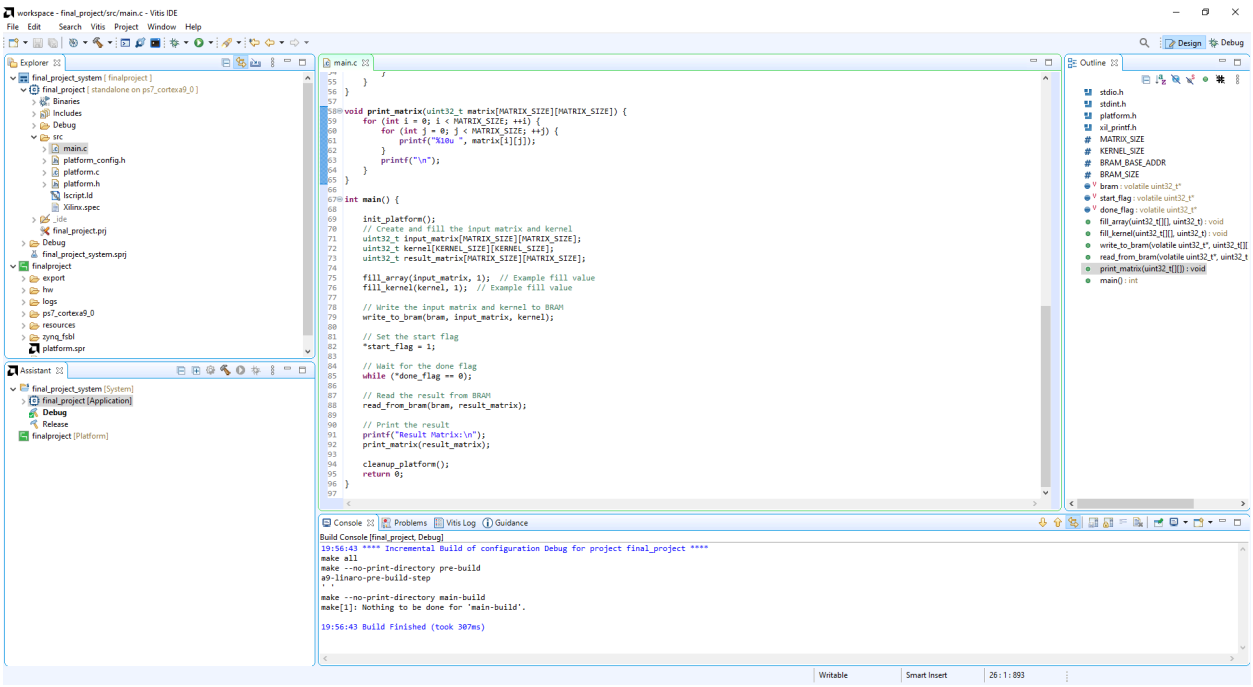
```c
void print_matrix(uint32_t matrix[MATRIX_SIZE][MATRIX_SIZE]) {
    for (int i = 0; i < MATRIX_SIZE; ++i) {
        for (int j = 0; j < MATRIX_SIZE; ++j) {
            printf("%10u ", matrix[i][j]);
        }
        printf("\n");
    }
}

int main() {

    init_platform();
    // Create and fill the input matrix and kernel
    uint32_t input_matrix[MATRIX_SIZE][MATRIX_SIZE];
    uint32_t kernel[KERNEL_SIZE][KERNEL_SIZE];
    uint32_t result_matrix[MATRIX_SIZE][MATRIX_SIZE];

    fill_array(input_matrix, 1);  // Example fill value
    fill_kernel(kernel, 1);  // Example fill value

    // Write the input matrix and kernel to BRAM
    write_to_bram(bram, input_matrix, kernel);

    // Set the start flag
    *start_flag = 1;

    // Wait for the done flag
    while (*done_flag == 0);

    // Read the result from BRAM
    read_from_bram(bram, result_matrix);

    // Print the result
    printf("Result Matrix:\n");
    print_matrix(result_matrix);

    cleanup_platform();
    return 0;
}
```

این کد یک مثال ساده از استفاده از حافظه‌های موری مپ شده در برنامه‌نویسی سخت‌افزاری است. در این برنامه، از حافظه BRAM (Block RAM) برای ارتباط بین نرم‌افزار و سخت‌افزار استفاده می‌شود. ابتدا، اندازه و آدرس پایه حافظه BRAM تعریف می‌شوند تا بتوان اطلاعات را به آن نوشته و از آن خواند. سپس، توابع کمکی برای پر کردن ماتریس و کرنل با مقادیر مشخص و نوشتن این اطلاعات به حافظه BRAM تعریف می‌شوند.

در `main`، ابتدا ماتریس و کرنل با مقادیر مورد نظر پر می‌شوند. سپس، این داده‌ها به حافظه BRAM با استفاده از تابع `write_to_bram` نوشته می‌شوند. پس از آن، flag شروع (`start_flag`) به 1 تنظیم می‌شود تا سخت‌افزار بفهمد که پردازش باید شروع شود. برنامه در حالی که flag انجام (`done_flag`) به 0 است، منتظر اتمام پردازش در سخت‌افزار می‌ماند. وقتی که این flag به 1 تغییر کند، نتیجه از حافظه BRAM خوانده شده و در ماتریس `result_matrix` قرار داده می‌شود. در نهایت، ماتریس نتیجه چاپ می‌شود.

این نمونه نشان می‌دهد که چگونه از حافظه‌های موری مپ شده برای انتقال داده‌ها بین نرم‌افزار و سخت‌افزار استفاده کرد تا فرآیندهای پردازشی مانند کانولوشن به صورت کارآمد اجرا شوند.

به عنوان جمع بندی ما یک شتابدهنده ضرب convolution برای یک ماتریس و یک kernel را به کمک co-design طراحی و پیاده سازی کردیم.



با تشکر