

For [README english](#)

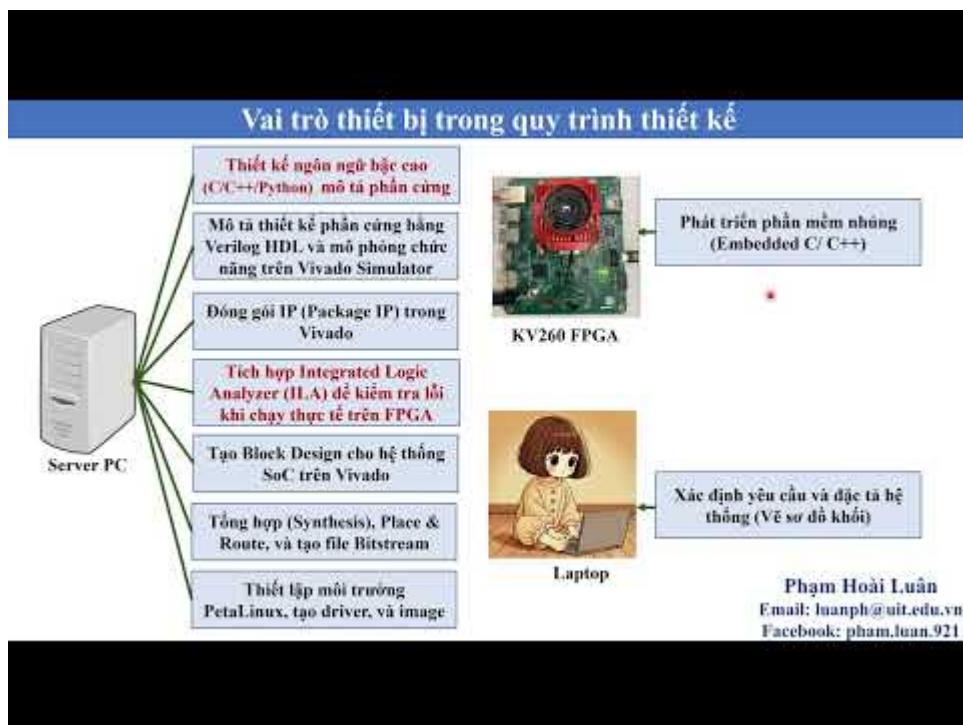
# 🎓 Thiết Kế Phần Cứng và Hệ Thống SoC trên FPGA – Level 1 (Kria KV260)

Chào mừng bạn đến với **Level 1** trong series **Thiết kế phần cứng và hệ thống SoC trên FPGA**.

Repository này chứa toàn bộ tài liệu, mã nguồn và hướng dẫn liên quan đến việc hiện thực một mô-đun phần cứng đơn giản và tích hợp vào hệ thống SoC trên bo mạch **Xilinx Kria KV260**.

## Video hướng dẫn chi tiết

Các bước sẽ được trình bày chi tiết trong video hướng dẫn tương ứng bên dưới, vui lòng bấm vào video bên dưới để xem chi tiết từng bước ⏪ ⏪ ⏪.



Hoặc truy cập link: <https://youtu.be/iHpeTRM6k9U>

### I. Yêu cầu thuật toán

Dự án này hiện thực một **bộ tăng tốc phần cứng** để thực hiện phép nhân ma trận A và vector X:

$$\mathbf{Y} = \mathbf{A} \times \mathbf{X}$$

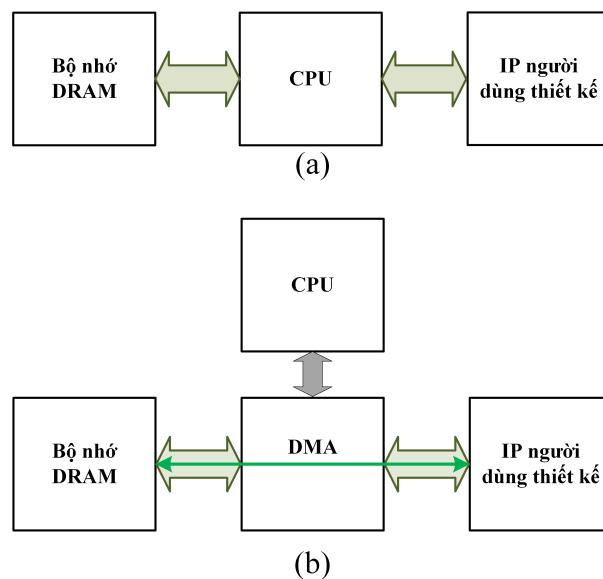
$$\begin{array}{c}
 \begin{matrix}
 & & & & 2^n \\
 & \leftarrow & \rightarrow & & \\
 \begin{matrix} 1 & -1 & 0 & \dots & 1 \\ 0 & 1 & -1 & \dots & 1 \\ \ddots & & & \vdots & \\ -1 & 1 & 1 & \dots & 0 \end{matrix} & \times & \begin{matrix} -1 \\ 0 \\ \vdots \\ 1 \end{matrix} & = & \begin{matrix} 89 \\ -12 \\ \vdots \\ 77 \end{matrix} \\
 2^n & & & & 2^n \\
 & \downarrow & & & \downarrow \\
 \text{Ma trận A} & & \text{Vector X} & & \text{Vector Y}
 \end{matrix}
 \end{array}$$

trong đó:

- **A** là ma trận vuông kích thước  $2^n \times 2^n$  ( $n \leq 14$ ),
- **X** là vector đầu vào có độ dài  $2^n$ ,
- **Y** là vector kết quả có độ dài  $2^n$ .
- Các phần tử của ma trận **A** và vector **X** chỉ nhận giá trị trong tập  $\{1, 0, -1\}$ .
- Toàn bộ dữ liệu A, X, Y được biểu diễn bằng số nguyên 16-bit có dấu.
- Hệ thống hỗ trợ cấu hình kích thước động thông qua các thanh ghi cấu hình.
- Bộ tăng tốc bao gồm:
  - **FSM (Finite State Machine)** với 4 trạng thái: **IDLE**, **LOAD**, **EXECUTE**, **DONE**.
  - **BRAM nội bộ** để lưu trữ A, X, Y.
  - Giao tiếp điều khiển sử dụng **PIO (Programmed I/O)**.
  - Truyền dữ liệu sử dụng **AXI-DMA**, băng thông **128-bit mỗi chu kỳ**. Bài học được thiết kế cho những người mới bắt đầu với phát triển hệ thống SoC trên nền FPGA.

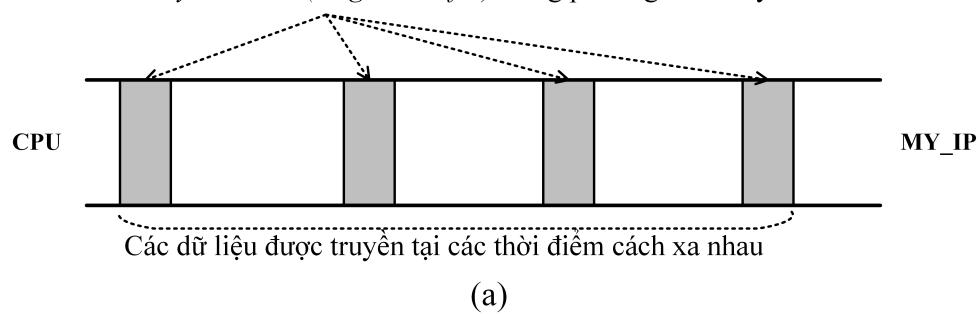
## II. Giới thiệu và Thiết cần dùng

### A. Giới thiệu về PIO và DMA:



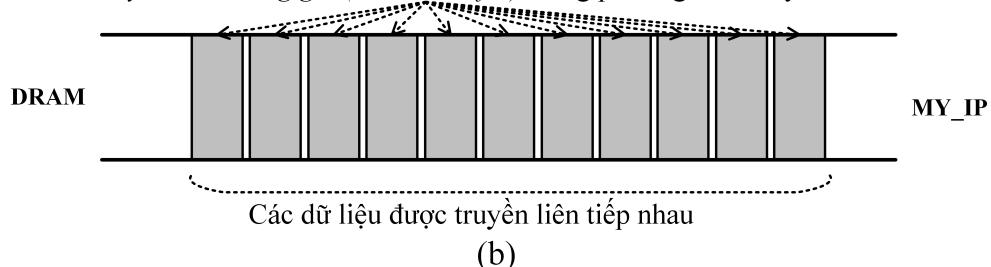
Trong các hệ thống SoC FPGA, DRAM thường được kết nối với các IP người dùng thiết kế để cung cấp dung lượng lưu trữ cao, đồng thời hỗ trợ truy xuất dữ liệu nhanh chóng cho các phép toán tính toán hoặc xử lý tín hiệu. DRAM lưu trữ các dữ liệu tạm thời cho các IP người dùng thiết kế. Cách thức sử dụng DRAM trong hệ thống SoC FPGA có thể thực hiện qua hai phương thức chính: PIO (Programmed I/O-Đầu ra vào được lập trình) và DMA (Direct Memory Access- Truy cập bộ nhớ trực tiếp). Trong phương thức PIO, CPU trực tiếp điều khiển việc truyền tải dữ liệu giữa DRAM và các IP thông qua các lệnh I/O, như được mô tả trong Hình (a) bên trên. Phương thức này dễ triển khai nhưng có thể gây tải nặng cho CPU vì CPU phải xử lý tất cả các thao tác truyền nhận dữ liệu. Ngược lại, DMA cho phép truyền tải dữ liệu giữa DRAM và các IP mà không cần can thiệp của CPU, giúp giảm tải cho CPU và tối ưu hóa băng thông, như mô tả trong Hình (b) bên trên.

*Truyền đơn lẻ (single transfer) trong phương thức truyền PIO*



(a)

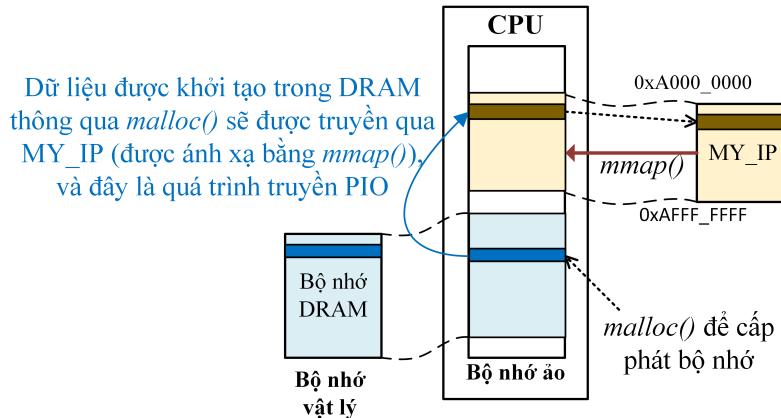
*Truyền theo dạng gói (burst transfer) trong phương thức truyền DMA*



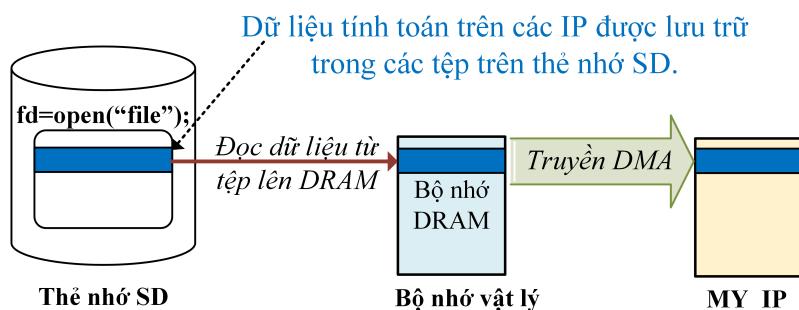
(b)

Hình trên minh họa sự khác biệt giữa phương thức truyền PIO và DMA. Trong phương thức truyền PIO, như thể hiện ở Hình (a) bên trên, truyền đơn lẻ (single transfer) yêu cầu CPU điều khiển từng lần truyền dữ liệu giữa DRAM và MY\_IP, với mỗi đơn vị dữ liệu được xử lý riêng biệt. Phương thức này cần sự can thiệp của CPU vào mỗi thao tác truyền tải, với CPU phải gửi lệnh và xử lý từng đơn vị dữ liệu, điều này làm giảm hiệu suất và hiệu quả băng thông vì CPU phải thực hiện quá nhiều thao tác cho mỗi đơn vị dữ liệu. Ngược lại, trong phương thức truyền DMA, như thể hiện ở Hình (b) bên trên, dữ liệu được truyền theo dạng gói (burst

transfer), cho phép dữ liệu được truyền liên tục từ DRAM vào MY\_IP mà không cần sự can thiệp của CPU. Dữ liệu trong DMA có thể được truyền liên tục hoặc cách xa nhau tùy theo yêu cầu của ứng dụng. Việc truyền liên tục dữ liệu trong các gói giúp tận dụng tối đa băng thông và giảm thiểu độ trễ, vì DMA có khả năng truyền tải dữ liệu từ các vùng bộ nhớ liền kề mà không cần sự gián đoạn.



Hình trên minh họa quá trình truyền dữ liệu được khởi tạo trong DRAM thông qua `malloc()` và sau đó truyền vào MY\_IP, nơi MY\_IP được ánh xạ vào bộ nhớ ảo thông qua `mmap()`. Quá trình này thể hiện việc sử dụng bộ nhớ ảo để cấp phát bộ nhớ, sau đó truyền dữ liệu từ bộ nhớ DRAM vào MY\_IP. Đây là một ví dụ về quá trình truyền PIO (Programmed I/O), trong đó CPU điều khiển việc truyền tải dữ liệu giữa DRAM và MY\_IP thông qua các lệnh I/O. Trong quá trình này, MY\_IP sử dụng bộ nhớ ảo đã được ánh xạ từ vùng bộ nhớ vật lý của DRAM, giúp dữ liệu được truy cập và truyền gián tiếp từ DRAM vào MY\_IP.

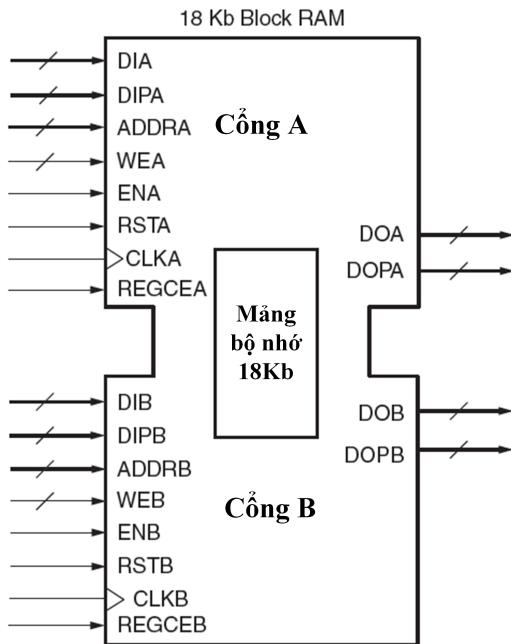


Hình trên mô tả truyền dữ liệu từ DRAM đến IP thiết kế thông qua phương pháp truyền DMA. Dữ liệu được lưu trữ trong các tệp trên thẻ nhớ SD và sau đó được đọc vào bộ nhớ DRAM thông qua các lệnh truy xuất tiêu chuẩn như `open("file")`. Sau khi dữ liệu đã có mặt trong DRAM, quá trình truyền dữ liệu từ DRAM đến MY\_IP được thực hiện thông qua DMA, giúp giảm tải cho CPU và tối ưu hóa băng thông. Quá trình này không yêu cầu sự can thiệp của CPU và đảm bảo hiệu suất truyền tải dữ liệu nhanh chóng và hiệu quả giữa bộ nhớ và các IP xử lý.

## B. Giới thiệu về BRAM:

Bộ nhớ Block RAM (BRAM) là một thành phần quan trọng trong FPGA, cung cấp khả năng lưu trữ dữ liệu trực tiếp trên chip với độ trễ thấp và băng thông cao. BRAM có kích thước lớn hơn so với LUTRAM (RAM tạo thành từ các LUT), vì vậy nó thường được sử dụng để lưu trữ các khối dữ liệu lớn hơn, hỗ trợ các ứng dụng đòi hỏi dung lượng bộ nhớ cao hơn. Được cấu trúc dưới dạng các khối bộ nhớ riêng biệt và độc lập, BRAM cho phép thiết kế linh hoạt trong việc lưu trữ và truy xuất dữ liệu, đặc biệt hữu ích cho các ứng dụng cần tốc độ xử lý cao như xử lý tín hiệu số, điều khiển hệ thống, và xử lý dữ liệu trong các hệ thống SoC. Trong thiết kế IP, BRAM thường được sử dụng như bộ nhớ toàn cục (global memory) để lưu trữ các dữ liệu lớn cần được truy cập bởi nhiều thành phần xử lý khác nhau, hoặc như bộ nhớ cục bộ (local memory) để lưu trữ dữ liệu tạm thời

và trạng thái trong các lõi xử lý cụ thể. Sự đa dụng này giúp BRAM trở thành một lựa chọn phổ biến trong các thiết kế FPGA phức tạp.



Hình trên mô tả giao diện của một khối BRAM có dung lượng 18 Kb. Bộ nhớ BRAM hỗ trợ giao diện hai cổng (cổng A và cổng B), cho phép đọc và ghi độc lập. Mỗi cổng bao gồm các tín hiệu dữ liệu đầu vào (DIA/DIB), địa chỉ (ADDRA/ADDRB), xung nhịp (CLKA/CLKB), tín hiệu điều khiển ghi (WEA/WEB), và tín hiệu điều khiển đầu ra (DOA/DOB). Mỗi cổng của BRAM, A và B, đều có các tín hiệu riêng để điều khiển và truyền dữ liệu. Tín hiệu đầu vào dữ liệu (DIA/B) và tín hiệu đầu vào kiểm tra chẵn lẻ (DIPA/B) cung cấp dữ liệu và kiểm tra tính toàn vẹn của dữ liệu được ghi vào bộ nhớ. Tín hiệu địa chỉ (ADDRA/B) xác định vị trí dữ liệu cần truy xuất hoặc ghi vào. Tín hiệu WEA/B là tín hiệu cho phép ghi theo chiều rộng byte, quyết định việc ghi dữ liệu vào bộ nhớ. ENA/B là tín hiệu cho phép hoạt động của cổng; khi không hoạt động, không có dữ liệu nào được ghi vào BRAM và tín hiệu đầu ra vẫn giữ nguyên trạng thái trước đó. Tín hiệu RSTA/B được sử dụng để thiết lập lại hoặc đặt lại đồng bộ các thanh ghi đầu ra khi DO\_REG = 1. CLKA/B là tín hiệu đầu vào xung nhịp cho cổng A hoặc B, điều khiển tốc độ hoạt động của bộ nhớ. Tín hiệu đầu ra dữ liệu (DOA/B) và tín hiệu đầu ra kiểm tra chẵn lẻ (DOPA/B) cung cấp dữ liệu và thông tin kiểm tra chẵn lẻ từ bộ nhớ. Cuối cùng, REGCEA/B là tín hiệu cho phép xung nhịp thanh ghi đầu ra, điều khiển hoạt động của các thanh ghi này.

### C. Danh sách thiết bị:

Dưới đây là danh sách các thiết bị phần cứng cần chuẩn bị để thực hành Level 0 trên bo mạch **Kria KV260 FPGA**.

**Kria KV260 FPGA****Dây cáp mạng**  
(Kết nối FPGA với internet)**Dây JTAG**  
(Kết nối KV260  
FPGA với Server PC)**Thẻ nhớ Micro và đầu  
đọc thẻ nhớ**  
(Để cài Linux cho FPGA)**1 Server PC**  
(chạy hệ điều hành Linux)**1 Laptop/PC cá nhân**  
(chạy hệ điều hành Windows)**1 Laptop/PC cá  
nhân**  
(chạy hệ điều  
hành Windows và  
hệ điều hành ảo  
Linux bằng  
VMware)

Có thể thay thế cho nhau

- **Kria KV260 FPGA:** Bo mạch chính dùng để triển khai hệ thống SoC và chạy ứng dụng nhúng.
- **Dây cáp mạng (LAN):** Dùng để kết nối FPGA với Internet thông qua router/switch, hỗ trợ cập nhật và debug qua SSH.
- **Dây JTAG:** Kết nối từ FPGA đến Server PC để nạp bitstream, debug hoặc hoạt động như dây UART để hiện thị console của Linux trên FPGA.
- **Thẻ nhớ MicroSD và đầu đọc thẻ:** Dùng để tạo image khởi động (BOOT.BIN + Linux kernel + rootfs) và cài hệ điều hành cho FPGA.
- **Server PC (Linux):** Cài đặt công cụ thiết kế phần cứng (Vivado), công cụ PetaLinux, và thực hiện build toàn bộ hệ thống.
- **Laptop/PC cá nhân (Windows hoặc Linux):** Dùng để kết nối SSH đến Server, hoặc truyền file (WinSCP). Nếu dùng Windows, cần cài **VMware** để chạy Linux.

⚠ **Lưu ý:** Bạn có thể thay thế **1 Server PC** và **1 Laptop/PC** thành **1 Laptop/PC duy nhất**, miễn là máy có cài đặt Linux để cài PetaLinux.

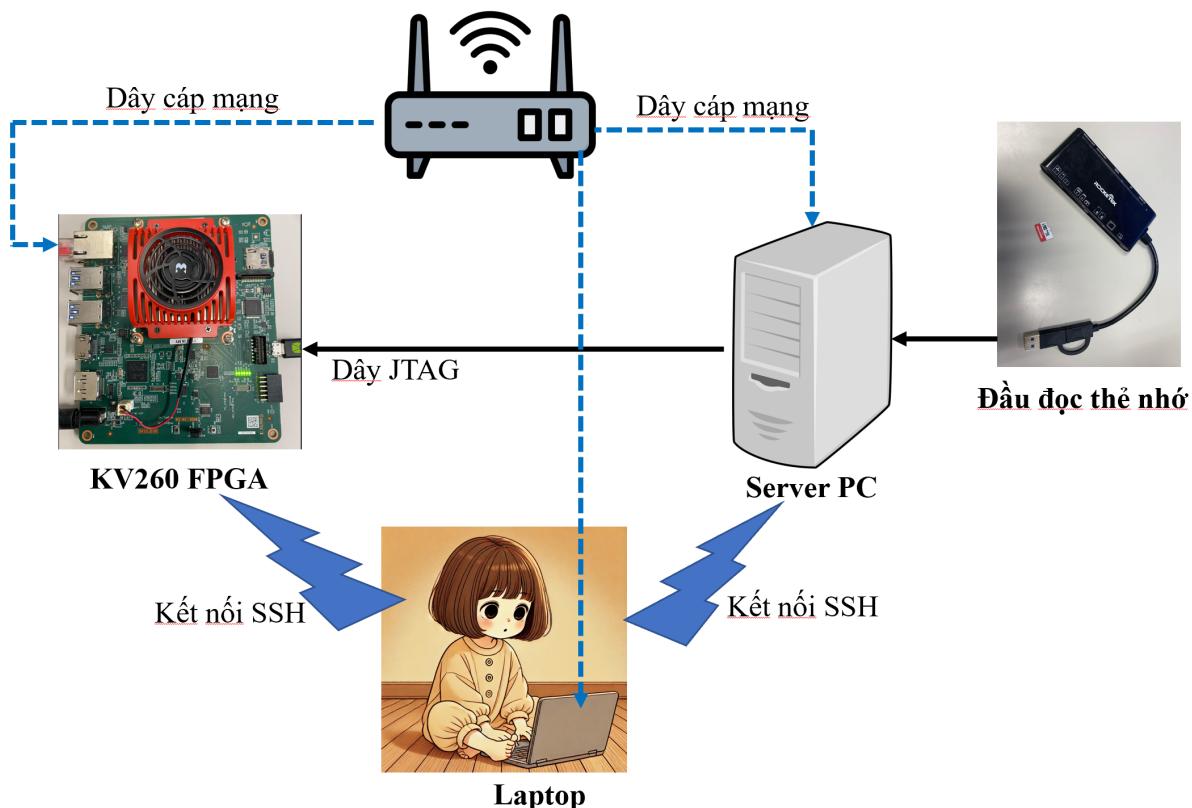
#### D. Kết nối thiết bị

Trước khi bắt đầu quy trình thiết kế phần cứng, cần kết nối và thiết lập các thiết bị như sau:

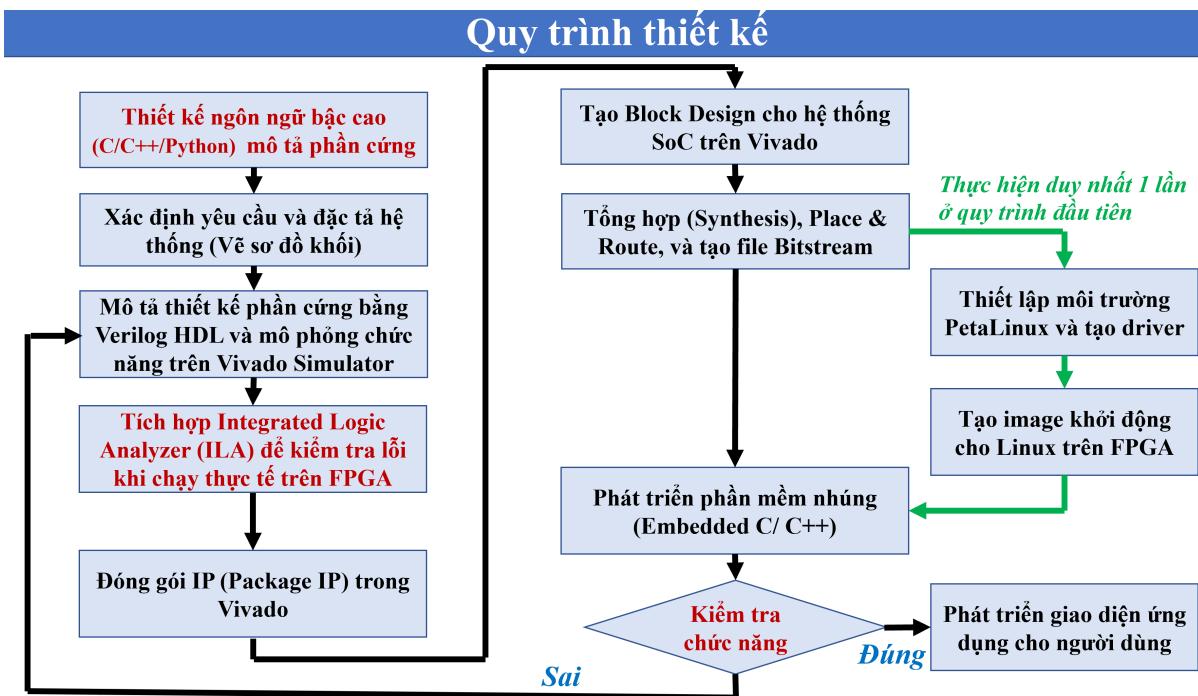
- **KV260 FPGA:** kết nối với router qua **dây mạng** để có internet, và kết nối với Server PC qua **dây JTAG** để nạp bitstream, debug.
- **Server PC:** dùng để cài **Vivado** và **Petalinux**, kết nối mạng và đầu đọc thẻ nhớ để chuẩn bị Linux cho FPGA.
- **Laptop:** sử dụng để điều khiển Server PC và KV260 thông qua **kết nối SSH** (qua MobaXterm, VSCode, hoặc Terminal).

⚠ **Lưu ý:**

- Server PC và Laptop cần nằm chung mạng nội bộ (LAN/WiFi).
- Thẻ nhớ microSD sẽ được dùng để nạp hệ điều hành Linux vào FPGA.



### III. Chi tiết từng bước trong quy trình thiết kế



Quy trình thiết kế hệ thống SoC trên FPGA gồm 8 bước tuần tự, bắt đầu từ việc xác định yêu cầu và mô tả phần cứng bằng Verilog, đến đóng gói IP, thiết kế hệ thống trên Vivado, thiết lập PetaLinux, và cuối cùng là phát triển phần mềm nhúng để điều khiển phần cứng đã thiết kế.

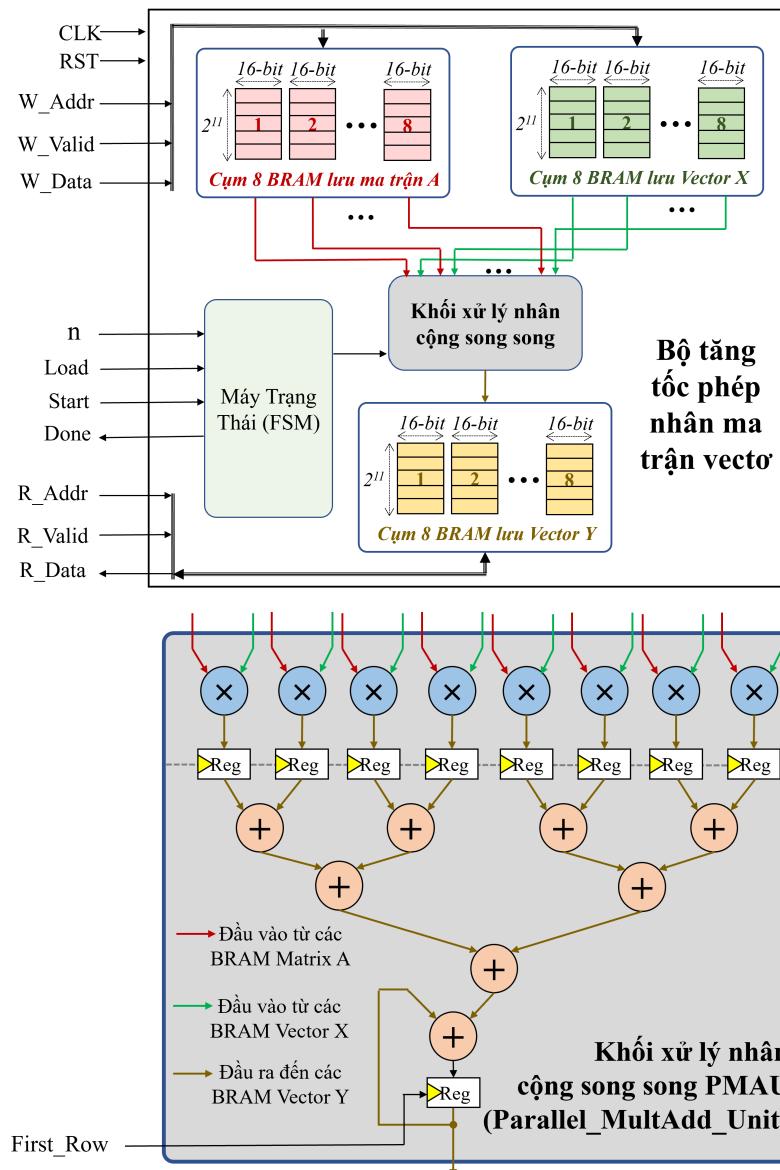
Kế tiếp tôi sẽ trình bày chi tiết 8 bước trên.

## A. Bước 1: Xác định yêu cầu và đặc tả hệ thống (vẽ sơ đồ khối)

- Hàm cần hiện thực:  $Y = A \times X + B$ , dùng chuẩn số **fixed point Q15.16** (1 bit dấu, 15 bit số nguyên, 16 bit thập phân).
- Xây dựng sơ đồ khối gồm các khối nhân, cộng, thanh ghi và điều khiển bởi **FSM (Finite State Machine)**.
- FSM gồm 3 trạng thái: **IDLE**, **EXECUTE**, **WAIT\_DONE**, điều khiển thông qua tín hiệu **Start\_in** và **Done\_in**.

❖ Tín hiệu chính:

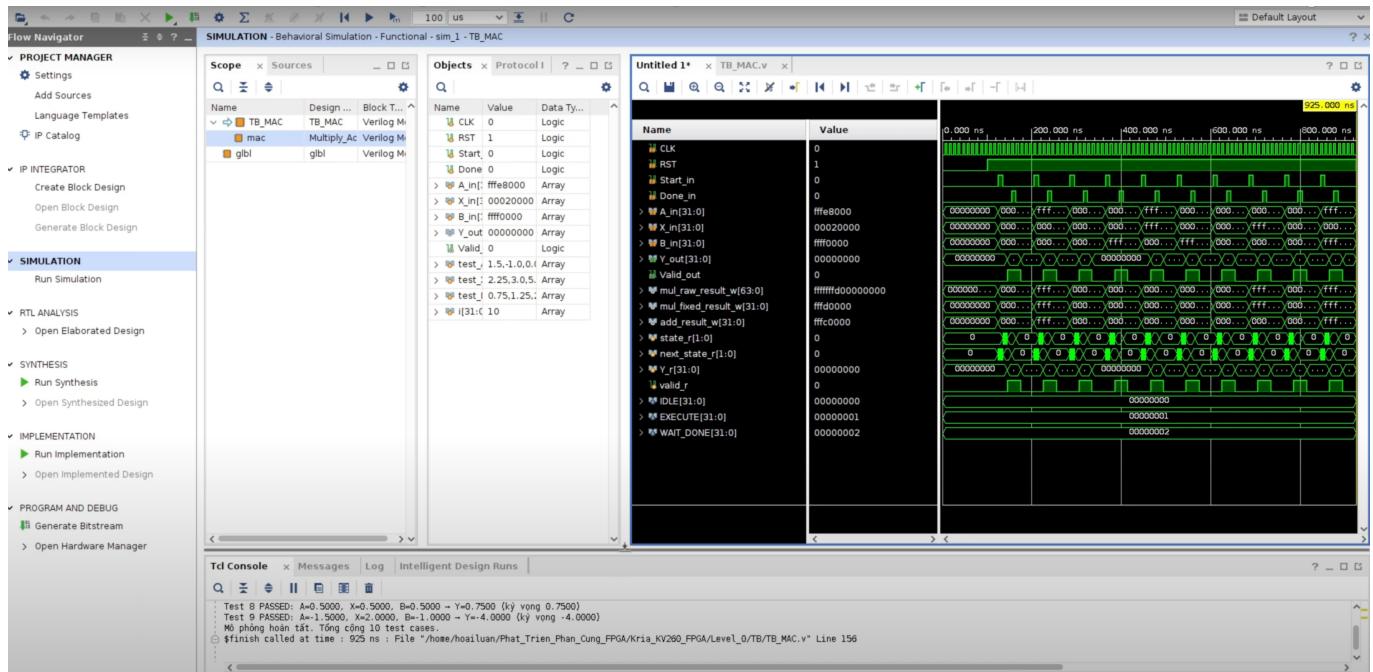
**A\_in**, **X\_in**, **B\_in** (đầu vào), **Y\_out**, **Valid\_out** (đầu ra), **Start\_in**, **Done\_in** (điều khiển)



## B. Bước 2: Mô tả thiết kế phần cứng và mô phỏng chức năng

- Viết mã **Verilog HDL** mô tả mạch số thực hiện phép tính  $Y = A \times X + B$  với chuẩn **fixed-point Q15.16** cho các toán hạng.
- Mã nguồn RTL Verilog** được đặt trong thư mục:
  - RTL/MAC.v

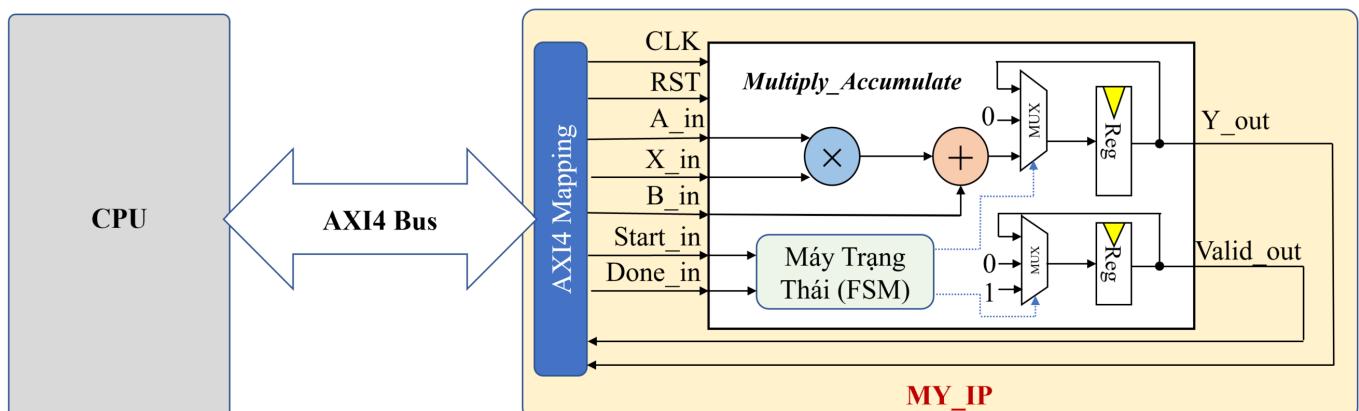
- Viết **testbench** để mô phỏng **10 test case** với các giá trị thực (real), kiểm tra đầu ra **Y\_out** có khớp với giá trị mong đợi. Chạy mô phỏng bằng **Vivado Simulator**, quan sát:
  - Dạng sóng tín hiệu trên waveform
  - Kết quả tính toán in ra cửa sổ console (PASS/FAIL từng test case)
- Mã nguồn testbench** được đặt trong thư mục:
  - TB/TB\_MAC.v**
- Project Vivado (2022.2)** đã cấu hình sẵn cho mô phỏng nằm trong thư mục:
  - Simulation**



### C. Bước 3: Đóng gói IP (Package IP) trong Vivado

Sau khi mô tả phần cứng bằng **Verilog HDL** và mô phỏng thành công, chúng ta tiến hành **đóng gói thiết kế thành một IP** để có thể tái sử dụng và tích hợp vào hệ thống SoC trong các bước tiếp theo.

Hình dưới minh họa cách **IP tự thiết kế (MY\_IP)** được tích hợp vào hệ thống SoC và kết nối với CPU thông qua **AXI4 Bus**. Tín hiệu đầu vào/ra của mạch (**A\_in**, **X\_in**, **B\_in**, **Start\_in**, **Done\_in**) được ánh xạ qua giao diện AXI4-Full thông qua lớp **AXI4 Mapping**.

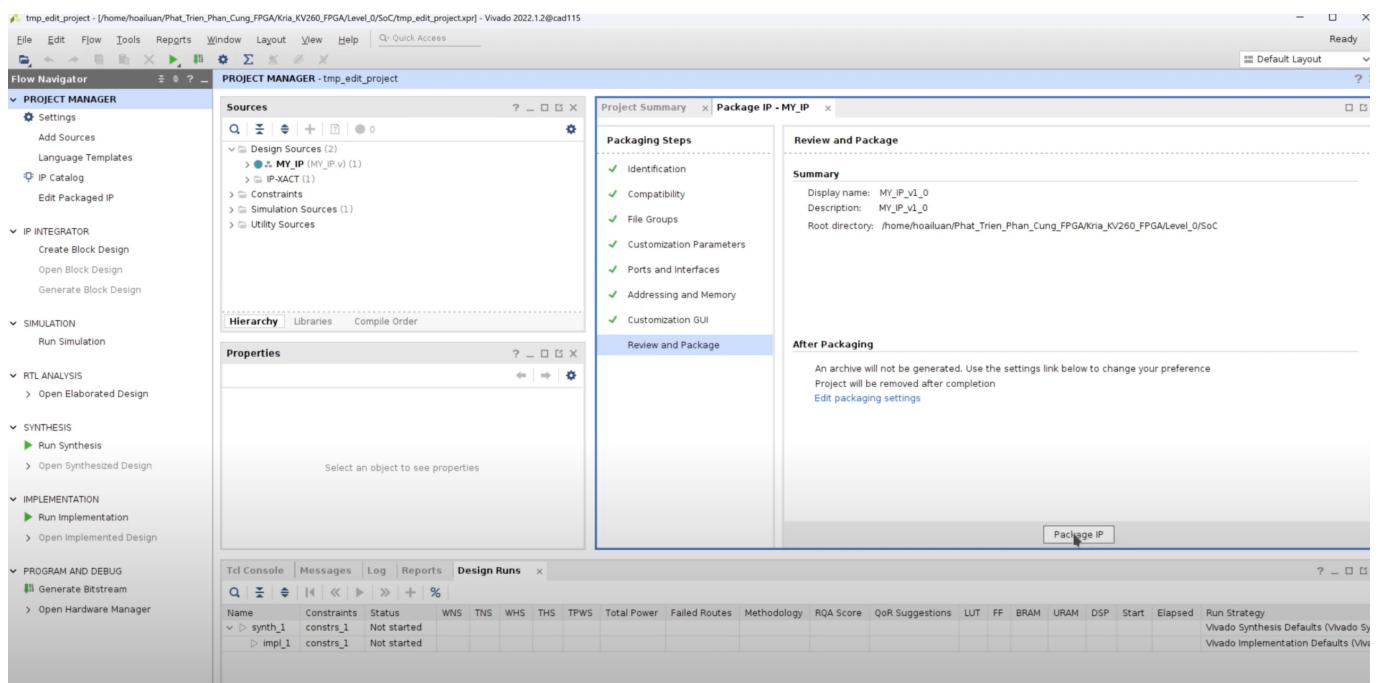


- Tham khảo nội dung về **hệ thống bus bao gồm AXI4-Full** ở thư mục :
  - [Tai\\_Lieu\\_Tham\\_Khao/Hệ\\_Thống\\_Bus.pdf](#)

Các bước thực hiện:

1. Mở Vivado, chọn menu **Tools → Create and Package New IP**
2. Chọn kiểu IP: từ mã RTL có sẵn ([Package your current project](#))
3. Điền thông tin định danh cho IP:
  - Tên IP ([MY\\_IP](#))
  - Phiên bản (ví dụ: [1.0](#))
  - Mô tả chức năng (Multiply-Accumulate core with FSM control)
4. Cấu hình các cổng tín hiệu I/O và địa chỉ giao tiếp:
  - Mapping tín hiệu qua chuẩn **AXI4-Full** nếu dùng giao tiếp với CPU
5. Kiểm tra lại toàn bộ cấu hình
6. Nhấn **Package IP** để đóng gói và thêm IP này vào Vivado IP Catalog

Đây là bước cần thiết để có thể sử dụng lại IP trong các Block Design.



## D. Bước 4: Tạo Block Design cho hệ thống SoC trên Vivado

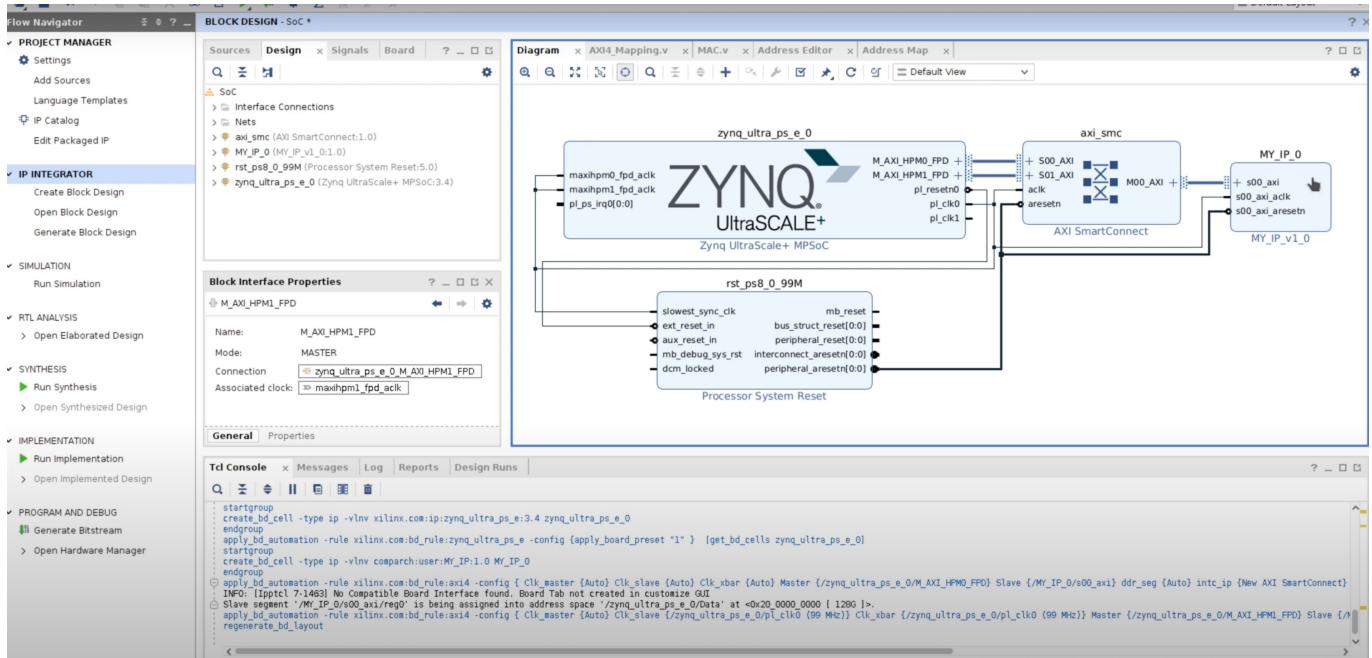
Sau khi đóng gói IP thành công, ta tiến hành tạo hệ thống SoC bằng cách sử dụng **Block Design** trong Vivado.

Các thành phần chính trong sơ đồ Block Design:

- **ZYNQ MPSoC**: bộ xử lý chính điều khiển hệ thống, cấu hình chân và kết nối AXI.
- **IP tự thiết kế (MY\_IP\_v1\_0)**: chứa hàm MAC  $Y = A * X + B$ , được kết nối thông qua chuẩn **AXI4-Full**.
- **AXI SmartConnect**: cầu nối giữa các master/slave sử dụng giao thức AXI.
- **Reset module**: đồng bộ hóa tín hiệu reset giữa phần xử lý và phần lập trình.

### Các thao tác cần thực hiện trong Vivado:

1. Tạo **Block Design mới** từ menu **IP Integrator**.
2. Thêm các thành phần chính vào sơ đồ (ZYNQ MPSoC, MY\_IP\_v1\_0, AXI SmartConnect, Reset).
3. Dùng **Run Block Automation** để tự động cấu hình ZYNQ.
4. Kết nối các cổng AXI và Reset đúng cách.



## E. Bước 5: Tổng hợp (Synthesis), Place & Route, và tạo file Bitstream

Sau khi hoàn tất sơ đồ kết nối:

1. Chuột phải vào **Block Design** → chọn "**Generate Output Products**".
2. Chuột phải lần nữa → chọn "**Create HDL Wrapper**" để sinh mã top-level cho thiết kế.
3. Cuối cùng, nhấn "**Generate Bitstream**" để chạy toàn bộ các bước:
  - Synthesis (tổng hợp)
  - Implementation (triển khai)
  - Bitstream Generation (tạo file cấu hình FPGA)

Đây là bước quan trọng để chuyển thiết kế thành file cấu hình **.bit** có thể nạp lên FPGA và file **.xsa** để cài đặt Petalinux cho FPGA.

## F. Bước 6: Thiết lập môi trường PetaLinux và tạo driver

Sau khi hoàn tất thiết kế phần cứng và tạo Block Design trong Vivado, bước tiếp theo là **xuất file phần cứng (.xsa)** để sử dụng trong PetaLinux nhằm tạo hệ điều hành và driver phù hợp cho hệ thống.

### 1. Xuất file phần cứng (.xsa) từ Vivado

- Trong Vivado, sau khi **Generate Bitstream** thành công:
  - Vào menu: **File → Export → Export Hardware**
  - Chọn: Include bitstream
  - File **.xsa** sẽ được tạo ra (ví dụ: **SoC\_wrapper.xsa**)

### 2. Cài đặt PetaLinux

- Tải bộ cài **PetaLinux 2022.2** từ trang chính thức Xilinx:   
<https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-design-tools/archive.html>

#### Cài đặt các gói phụ thuộc (Ubuntu/Debian)

```
sudo apt-get install tofrodos gawk xvfb git libncurses5-dev tftpd zlib1g-dev  
zlib1g-dev:i386 \  
libssl-dev flex bison chrpath socat autoconf libtool texinfo gcc-multilib \  
libsdl1.2-dev libglib2.0-dev screen pax libtinfo5 xterm build-essential net-tools
```

#### Cấp quyền thực thi cho file .run

```
chmod +x petalinux-v2022.2-* .run
```

#### Chạy trình cài đặt

```
./petalinux-v2022.2-* .run
```

- Trong quá trình cài đặt, trình cài đặt sẽ hiển thị các thỏa thuận bản quyền:
  - Dùng PgUp / PgDn để đọc
  - Nhấn q để thoát khỏi phần hiển thị
  - Nhấn y để đồng ý và tiếp tục

### 3. Xây dựng môi trường phần cứng

#### Thiết lập môi trường làm việc Petalinux

Source đến thư mục cài đặt Petalinux để sử dụng được các lệnh **petalinux-\***:

```
source <đường_dẫn_cài_petalinux>/2022.2/settings.sh
```

#### Tải bộ cài BSP cho KV260 FPGA từ trang chính thức Xilinx:

 <https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-design-tools/archive.html>

#### Tạo project PetaLinux từ BSP

```
petalinux-create -t project -s <đường_dẫn_tới_file_BSP>.bsp --name KV260_Linux
cd KV260_Linux
```

**Import phần cứng (.xsa) vào project Sau khi bạn export file .xsa từ Vivado (có chứa bitstream), hãy dùng lệnh sau để tích hợp phần cứng vào project:**

```
petalinux-config --get-hw-description=<path_to_the_hw_description_file>
```

**Cấu hình kernel bootargs thủ công Sau khi chạy petalinux-config, hệ thống sẽ mở giao diện curses để bạn cấu hình sâu hơn.**

**Điều chỉnh cấu hình kernel bootargs Trong cửa sổ cấu hình, thực hiện các bước sau:**

```
Subsystem AUTO Hardware Settings --->
    DTG Settings --->
        Kernel Bootargs --->
            [ ] generate boot args automatically
            (user-defined) user set kernel bootargs
```

Dán đoạn bootargs dưới đây vào phần user set kernel bootargs:

```
earlycon console=ttyPS1,115200 root=/dev/mmcblk1p2 rw rootwait cpuidle.off=1
uio_pdrv_genirq.of_id=generic-uio clk_ignore_unused init_fatal_sh=1 cma=256M
```

❖ Cấu hình này giúp khởi động đúng thiết bị, bật driver UIO, cấp vùng bộ nhớ CMA, và giữ clock cho các IP tự thiết kế trong PL.

#### Chỉnh sửa Device Tree (system-user.dtsi)

Để hệ điều hành Linux có thể sử dụng **IP tự thiết kế trong PL** thông qua driver **uio**, bạn cần chỉnh sửa file **Device Tree Overlay**. Trong file ở đường dẫn **KV260\_Linux/project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi**, chỉnh lại file thành:

```
/include/ "system-conf.dtsi"
{
    reserved-memory {
        #address-cells = <2>;
        #size-cells = <2>;
        ranges;
        reserved: buffer@0 {
            no-map;
            reg = <0x8 0x0 0x0 0x80000000>;
        };
    };
}
```

```
};

amba: axi {
    /* GDMA */
    fpd_dma_chan1: dma-controller@fd500000 {
        compatible = "generic-uio";
    };

    fpd_dma_chan2: dma-controller@fd510000 {
        compatible = "generic-uio";
    };

    fpd_dma_chan3: dma-controller@fd520000 {
        compatible = "generic-uio";
    };

    fpd_dma_chan4: dma-controller@fd530000 {
        compatible = "generic-uio";
    };

    fpd_dma_chan5: dma-controller@fd540000 {
        compatible = "generic-uio";
    };

    fpd_dma_chan6: dma-controller@fd550000 {
        compatible = "generic-uio";
    };

    fpd_dma_chan7: dma-controller@fd560000 {
        compatible = "generic-uio";
    };

    fpd_dma_chan8: dma-controller@fd570000 {
        compatible = "generic-uio";
    };

};

amba_pl@0 {
    MY_IP@a0000000 {
        compatible = "generic-uio";
    };
};

ddr_high@000800000000 {
    compatible = "generic-uio";
    reg = <0x8 0x0 0x0 0x8000000>;
};

};
```

File [system-user.dtsi](#) mẫu được lưu trong thư mục KV260\_Linux ở github.

### Sau đó tiến hành build project

```
petalinux-build
```

### G. Bước 7: Tạo image khởi động và rootfs cho Linux trên SoC FPGA

Sau khi build project thành công, gõ lệnh này để đóng gói file khởi động BOOT.BIN cùng với U-Boot phù hợp cho hệ thống.

```
petalinux-package --boot --force --u-boot
```

Sau đó cắm SD card vào PC, tiến hàn phân vùng và định dạng thẻ nhớ SD. \*\*Bạn có thể làm theo hướng dẫn chi tiết trong Video hướng dẫn bên trên \*\* từ phút **53:40 đến 1:03:18** tại link bên dưới:

 [Tải file Debian rootfs tại đây](#)

File rootfs này chứa hệ điều hành Debian đã được cấu hình sẵn cho kiến trúc ARM64, hỗ trợ giao diện XFCE và dễ dàng cài đặt thêm ứng dụng bằng [apt](#).

### H. Bước 8: Phát triển phần mềm nhúng (Embedded C/ C++)

Sau khi đã chuẩn bị đầy đủ hệ điều hành Linux trên FPGA, chúng ta tiến hành chạy chương trình nhúng điều khiển IP tự thiết kế bằng ngôn ngữ **C/C++**.

#### Thư mục code

Trong repo GitHub này, thư mục [Embedded\\_C\\_Code](#) chứa toàn bộ mã nguồn C điều khiển IP MAC thông qua giao tiếp PIO.

#### Cách chạy

1. Mở phần mềm **WinSCP** để kết nối từ máy tính cá nhân đến board **KV260 FPGA** (qua SSH).
2. **Copy toàn bộ thư mục Embedded\_C\_Code** từ repo này vào thư mục [/home/debian/](#) trên KV260.
3. Trên terminal (hoặc qua MobaXterm), truy cập vào thư mục đã copy:

```
ssh debiang@<địa chỉ IP của KV260 FPGA> (Ví dụ 192.168.1.10)
cd Embedded_C_Code
sh run.sh
```

---

 Mọi góp ý hoặc liên hệ để giải đáp lỗi khi thực hiện project này vui lòng liên hệ:

- Facebook: <https://www.facebook.com/pham.luan.921/>
- Email: [luanph@uit.edu.vn](mailto:luanph@uit.edu.vn)

Rất cảm ơn sự đồng hành và ủng hộ của bạn 🙏

**Chúc bạn học FPGA thật vui và hiệu quả!**

Nếu bạn thấy nội dung mình chia sẻ **hữu ích, thực tế và có giá trị học tập hoặc nghiên cứu**, bạn có thể **ủng hộ** mình một chút để tiếp thêm động lực ra những phần tiếp theo chất lượng hơn.

☞ **Lưu ý:** Nếu bạn là **sinh viên**, mình **không mong chờ sự ủng hộ tài chính** từ bạn đâu.

Chỉ cần bạn học tốt, hiểu bài và lan tỏa kiến thức đến những người cần là mình đã rất vui rồi! ❤️

