

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA ĐIỆN TỬ - VIỄN THÔNG

MSSV 21207001

Bùi Thành Đạt

BÁO CÁO THỰC TẬP THỰC TẾ

Thiết kế và Tích hợp Bộ điều khiển DMA Tùy chỉnh trên FPGA
trong Hệ thống SoC Nios V/m

NGÀNH KỸ THUẬT ĐIỆN TỬ - VIỄN THÔNG
CHƯƠNG TRÌNH CHẤT LƯỢNG CAO

Tp. Hồ Chí Minh, tháng 04/2025

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA ĐIỆN TỬ - VIỄN THÔNG

MSSV 21207001
Bùi Thành Đạt

BÁO CÁO THỰC TẬP THỰC TẾ

Thiết kế và Tích hợp Bộ điều khiển DMA Tùy chỉnh trên FPGA
trong Hệ thống SoC Nios V/m

NGÀNH KỸ THUẬT ĐIỆN TỬ - VIỄN THÔNG
CHƯƠNG TRÌNH CHẤT LƯỢNG CAO

GIÁO VIÊN HƯỚNG DẪN
TS. Huỳnh Hữu Thuận

Tp. Hồ Chí Minh, tháng 04/2025

MỤC LỤC

MỤC LỤC	i
DANH SÁCH CHỮ VIẾT TẮT	iii
DANH SÁCH CÁC HÌNH	v
DANH SÁCH CÁC BẢNG	ix
1 GIỚI THIỆU VỀ ĐƠN VỊ THỰC TẬP	1
1.1 Giới thiệu chung	1
1.2 Lĩnh vực hoạt động và Nghiên cứu	1
2 KIẾN TRÚC HỆ THỐNG VÀ LÝ THUYẾT CƠ SỞ	3
2.1 Tổng quan về Hệ thống trên Chip (SoC)	3
2.2 Bộ xử lý mềm Intel Nios V/m	3
2.3 Truy cập Bộ nhớ Trực tiếp (DMA)	4
2.4 Giao tiếp Hệ thống: Bus Avalon	5
2.4.1 Giao tiếp Bus Avalon Memory Mapped Interface (Avalon-MM)	5
2.5 Thiết kế và Kiến trúc Bộ điều khiển DMA Tùy chỉnh	8
2.5.1 Mục tiêu và Bối cảnh: Truyền dữ liệu giữa Bộ nhớ On-Chip .	8
2.5.2 Kiến trúc Tổng thể và Luồng Dữ liệu	9
2.5.3 Thiết kế Module Điều khiển ('CONTROL_SLAVE')	11
2.5.4 Thiết kế Module Master Đọc (READ_MASTER)	12
2.5.5 Thiết kế Module Master Ghi (WRITE_MASTER)	13
2.5.6 Tích hợp và Tín hiệu Giao tiếp	15
3 TRIỂN KHAI VÀ KIỂM THỬ HỆ THỐNG NIOS V VỚI DMA	17
3.1 Xây dựng Hệ thống Phần cứng trên Quartus Prime	17
3.2 Phát triển Ứng dụng Phần mềm Nios V	31
4 Mô phỏng và kiểm thử hệ thống Nios V và DMA	45
4.1 Công cụ Mô phỏng và Yêu cầu Môi trường	45
4.2 Kết quả mô phỏng và kiểm thử chức năng	46
4.3 Kết luận Chương 4	54

5 KẾT QUẢ ĐẠT ĐƯỢC VÀ THÁI ĐỘ	55
5.1 Về kiến thức	55
5.2 Về kỹ năng	55
5.3 Về thái độ	56
TÀI LIỆU THAM KHẢO	57
A MÃ NGUỒN CỦA THIẾT KẾ	59
A.1 Mã Verilog Top-Level cho DMANiosV	59
A.2 Mã Verilog DMA Controller	59
A.3 Mã Verilog Control Slave	61
A.4 Mã Verilog Read Master	64
A.5 Mã Verilog Write Master	67
A.6 Mã nguồn C kiểm thử DMA	72
B MỘT SỐ LỖI THƯỜNG GẶP	81
B.1 Lỗi Tạo BSP hoặc Biên dịch Ứng dụng	81
B.2 Sự cố Kết nối JTAG và Debug	81
B.3 Không có file ‘.sof’ để nạp xuống board	82
B.4 Nên sử dụng bản Quartus nào khi xây dựng hệ thống Nios V	83
B.5 Lấy Giấy phép (License) Nios V/m	83
B.6 Quy trình Generate Testbench System trên Linux và Chạy Mô phỏng Nios V dùng Questa Sim trên Windows	91
B.7 Signal Tap chỉ thu thập được một phần nhỏ tín hiệu	94

DANH SÁCH CHỮ VIẾT TẮT

AI Artificial Intelligence. 2

ASIC Application-Specific Integrated Circuit. 1

BSP Board Support Package. 34, 35

CESLAB Computer Embedded Systems Laboratory. 1, 2, 59

DMA Direct Memory Access. 3, 4, 14, 16, 19, 49–51, 58, 59

DSP Digital Signal Processing. 2, 3

ELF Executable and Linkable Format. 85

FETEL Faculty of Electronics and Telecommunications. 1

FPGA Field-Programmable Gate Array. 1–3, 19, 21, 22, 49, 58, 60

HCMUS University of Science. 1

IC Integrated Circuit. 1, 3

IDE Integrated Development Environment. 19, 34, 59

IoT Internet of Things. 2

IP Intellectual Property. 60, 88

JTAG Joint Test Action Group. 20, 36, 85

MAC Media Access Control. 88

NIC Network Interface Card. 88

Nios II Nios® II. 49, 59

Nios V Nios V/m. 4, 14, 49–51, 58, 59, 87

RAM Random Access Memory. 3, 4, 20, 49

ROM Read-Only Memory. 3, 20

RTL Register Transfer Level. 58

RTOS Real-Time Operating System. 1, 2

SoC System-on-Chip. 1, 3, 14, 19–21, 49, 58, 60

SSLC Self-Service Licensing Center. 87

UART Universal Asynchronous Receiver/Transmitter. 3, 20

USB Universal Serial Bus. 3, 36

VNU-HCM Vietnam National University Ho Chi Minh City. 1

DANH SÁCH CÁC HÌNH

2.1	Kiến trúc Nios V/m [10].	4
2.2	Các giao dịch đọc và ghi Avalon-MM điển hình với tín hiệu waitrequest [3].	5
2.3	Giản đồ sóng đọc và ghi bộ nhớ Memory qua giao tiếp Avalon-MM. . .	6
2.4	Giản đồ sóng đọc và ghi của giao tiếp Avalon-MM Master.	7
2.5	Giản đồ sóng chức năng của Byte Enable. Hình này minh họa ảnh hưởng của byte enable lên dữ liệu được ghi vào và đọc ra từ bộ nhớ [4].	8
2.6	Sơ đồ khái tổng thể của bộ điều khiển DMA.	10
2.7	Giản đồ sóng đọc và ghi của giao tiếp Avalon-MM Slave.	11
2.8	Sơ đồ trạng thái của module CONTROL_SLAVE.	12
2.9	Sơ đồ trạng thái các module Master.	14
2.10	Sơ đồ tuần tự hoạt động của DMA.	15
3.1	Quartus: New Project Wizard: Chỉ định đường dẫn thư mục và tên Project.	20
3.2	Quartus: New Project Wizard: Device Selector	20
3.3	Thư mục dự án hiển thị các tệp Verilog DMA tùy chỉnh đã sao chép. .	21
3.4	Cấu hình IP Nios V/m: Bật "Enable Reset from Debug Module". . .	21
3.5	Cấu hình IP Bộ nhớ Trên Chip: Đặt kích thước thành 128 KB. . . .	22
3.6	Cấu hình IP JTAG UART.	23
3.7	Component Editor: Các tệp sau khi Phân tích và Sao chép từ Tổng hợp.	24
3.8	Component Editor: Các loại tín hiệu clock và reset đã được sửa. . . .	25
3.9	Component Editor: Chế độ xem sơ đồ khái giao diện Avalon Slave. .	26
3.10	Component Editor: Chế độ xem giao diện Reset và Clock sink. . . .	26
3.11	Platform Designer: Hệ thống có kết nối giữa instruction_master với s1 (của onchip_memory2_1).	27
3.12	Platform Designer: Gán Địa chỉ Cơ sở (Assign Base Addresses). . . .	27
3.13	Platform Designer: Mô hình hệ thống hoàn chỉnh (sau khi ngắt kết nối instruction_master với s1 trên onchip_memory2_1).	28
3.14	Cấu hình IP Nios V/m: Đặt Bộ nhớ Vector Reset (Reset Vector Memory).	28
3.15	Platform Designer: Hộp thoại Tạo HDL (Generate HDL).	29
3.16	Quartus: Menu Project -> Add/Remove Files in Project...	29
3.17	Quartus: Cài đặt Dự án - Cửa sổ Tệp hiển thị tệp .qip đã thêm. . . .	30

3.18	Quartus: Menu Assignments -> Import Assignments...	30
3.19	Quartus: Hộp thoại Nhập Gán chân (Import Assignments) để chọn tệp .qsf.	30
3.20	Quartus: Compile Design thành công.	31
3.21	Quartus: Trình chỉnh sửa Gán chân (Assignment Editor) hiển thị các gán chân đã nhập.	31
3.22	Cấu trúc thư mục Software gồm các thư mục con app và bsp.	34
3.23	Thư mục software/app chứa tệp mã nguồn source.c.	34
3.24	Vị trí của niosv-shell.exe.	34
3.25	Cửa sổ Nios V Shell.	34
3.26	Đầu ra từ việc thực thi lệnh niosv-bsp.	35
3.27	Đầu ra từ việc thực thi lệnh niosv-app.	35
3.28	Khởi chạy Ashling RiscFree™ IDE từ Nios V Shell, chọn đường dẫn Workspace.	35
3.29	Màn hình chào mừng Ashling RiscFree™ IDE: Chọn "Create a project...".	36
3.30	Ashling IDE: Chọn C/C++ → C Project.	36
3.31	Ashling IDE: Chọn CMake driven → Empty Project.	37
3.32	Ashling IDE: Project Explorer hiển thị dự án 'app'.	37
3.33	Ashling IDE: nhấp chuột phải 'app' → Build Project.	38
3.34	Ashling IDE: Console thông báo Build thành công.	38
3.35	Quartus Programmer: Nạp tệp .sof xuống board thành công.	39
3.36	Ashling IDE: Nạp Firmware xuống lõi Nios V (Run As → Ashling RISC-V Hardware Debugging).	39
3.37	Ashling IDE: Cấu hình Gõ lỗi - Chỉ định ứng dụng C/C++ (.elf).	40
3.38	Ashling IDE: Cấu hình Gõ lỗi - Cài đặt tab Debugger.	41
3.39	Ashling IDE: Debug Console cho biết đang chờ kết nối (Nhấn nút đỏ để ngừng chạy).	41
3.40	Đầu ra JTAG UART Terminal: Thông báo ban đầu và reset DMA.	42
3.41	Đầu ra JTAG UART Terminal: Trạng thái bộ đệm và cấu hình DMA.	43
3.42	Đầu ra JTAG UART Terminal: Xác minh truyền DMA thành công.	44
4.1	Generate Testbench System trong Platform Designer trên Ubuntu 22.04.5 thành công.	46
4.2	Mô phỏng: Ghi địa chỉ Read Master Start Address vào giao diện DMA Control Slave.	48

4.3	Mô phỏng: Ghi địa chỉ Write Master Start Address vào giao diện DMA Control Slave.	48
4.4	Mô phỏng: Thiết lập độ dài truyền (Transfer Length) qua giao diện DMA Control Slave.	49
4.5	Mô phỏng: Khởi tạo truyền DMA bằng cách đặt bit Control GO.	50
4.6	Mô phỏng: Hoàn thành các hoạt động của Read Master trong quá trình truyền DMA.	51
4.7	Mô phỏng: Tín hiệu hoàn thành (WM_done và status_done) cho biết truyền DMA thành công.	52
4.8	Mô phỏng: Transcript hiển thị quá trình và kết quả truyền DMA.	53
B.1	Lỗi niosv-app báo đường dẫn hoặc tệp source code không tồn tại.	81
B.2	Khởi động lại Altera JTAG Server trong Services của Windows.	82
B.3	Trang đăng nhập Công Intel Licensing Portal.	85
B.4	Email thông báo tài khoản Intel FPGA Self-Service Licensing đã được tạo thành công.	86
B.5	Chọn "Sign up for Evaluation or No-Cost Licenses".	87
B.6	Chọn bộ xử lý Nios V/m để cấp phép License.	87
B.7	Tìm ID Card Giao diện Mạng (Network Interface Card - NIC ID) (Địa chỉ Vật lý).	88
B.8	Nhập thông tin máy tính (Tên, Loại, NIC ID) cho giấy phép.	88
B.9	Tạo giấy phép cố định (fixed license) dựa trên chi tiết máy tính.	88
B.10	Email xác nhận chứa tệp giấy phép đính kèm.	89
B.11	Truy cập Cài đặt Giấy phép (License Setup) từ menu Công cụ (Tools) của Quartus.	90
B.12	Cửa sổ Cài đặt Giấy phép Quartus (Quartus License Setup) hiển thị tệp giấy phép Nios V/m đã thêm.	90

DANH SÁCH CÁC BẢNG

CHƯƠNG 1: GIỚI THIỆU VỀ ĐƠN VỊ THỰC TẬP

1.1 Giới thiệu chung

Tên đơn vị: Phòng thí nghiệm Máy Tính và Hệ Thống Nhúng (CESLAB).

Địa chỉ: Phòng E103A, Khu nhà E, Trường Đại học Khoa học Tự nhiên, VNU-HCM, 227 Nguyễn Văn Cừ, Phường 4, Quận 5, TP. Hồ Chí Minh.

Trực thuộc: Bộ môn Máy tính - Hệ thống Nhúng, Khoa Điện tử - Viễn thông (FETEL), Trường Đại học Khoa học Tự nhiên (HCMUS), VNU-HCM.

Website (tham khảo): <https://www.ceslab.id.vn/>

Email (tham khảo): ceslab.fetel@gmail.com

CESLAB được thành lập với mục tiêu trở thành trung tâm nghiên cứu và đào tạo chuyên sâu, uy tín trong lĩnh vực hệ thống nhúng (embedded systems), thiết kế vi mạch (IC Design), và các công nghệ liên quan. Phòng thí nghiệm tạo môi trường học tập, thực hành và nghiên cứu hiện đại cho sinh viên, học viên cao học và nghiên cứu sinh, đồng thời thúc đẩy hợp tác với các doanh nghiệp và tổ chức trong và ngoài nước.

1.2 Linh vực hoạt động và Nghiên cứu

CESLAB tập trung vào các hướng nghiên cứu, phát triển và đào tạo chính trong các lĩnh vực công nghệ cao, bao gồm:

- Thiết kế Hệ thống nhúng (*Embedded Systems Design*):** Nghiên cứu, thiết kế và phát triển các hệ thống nhúng hoàn chỉnh, từ phần cứng đến phần mềm, ứng dụng trong nhiều lĩnh vực như công nghiệp, y tế, tiêu dùng.
- Thiết kế Vi mạch và Hệ thống trên chip (*IC Design & System-on-Chip - SoC*):** Tập trung vào thiết kế vi mạch số (digital IC design), vi mạch tương tự (analog IC design), tín hiệu hỗn hợp (mixed-signal) và tích hợp các hệ thống phức tạp trên một vi mạch duy nhất (SoC) sử dụng các công nghệ FPGA và ASIC.
- Lập trình Hệ thống nhúng và Hệ điều hành Thời gian thực (*Embedded Software & RTOS*):** Phát triển phần mềm điều khiển cấp thấp (low-level

control software), trình điều khiển thiết bị (device drivers), và ứng dụng trên các nền tảng nhúng, bao gồm cả việc sử dụng hệ điều hành thời gian thực (RTOS).

4. **Internet of Things (IoT):** Nghiên cứu và triển khai các giải pháp IoT, bao gồm thiết kế thiết bị biên (edge devices), mạng cảm biến không dây (wireless sensor networks), và các nền tảng thu thập, xử lý dữ liệu.
5. **Xử lý Tín hiệu Số trên Phần cứng (Hardware-based Digital Signal Processing - DSP):** Tối ưu và triển khai các thuật toán xử lý tín hiệu số phức tạp trên các nền tảng phần cứng như FPGA và bộ xử lý DSP.
6. **Trí tuệ Nhân tạo cho Hệ thống nhúng (AI for Embedded Systems / Edge AI):** Nghiên cứu và ứng dụng các mô hình học máy (machine learning), học sâu (deep learning) trên các thiết bị nhúng có tài nguyên hạn chế, phục vụ các bài toán như nhận dạng hình ảnh (image recognition).

Trong quá trình thực tập tại CESLAB, sinh viên được tiếp cận với các công cụ thiết kế, mô phỏng và kiểm thử tiên tiến, làm việc trên các bo mạch phát triển FPGA hiện đại và được hướng dẫn bởi các giảng viên, nghiên cứu viên có kinh nghiệm. Đây là môi trường lý tưởng để sinh viên trau dồi kiến thức chuyên môn, rèn luyện kỹ năng thực hành và định hướng cho sự nghiệp tương lai trong ngành Điện tử - Viễn thông.

Danh sách các nghiên cứu viên chính tại CESLAB:

- Trần Tuấn Kiệt, MSc. (trtkiet@hcmus.edu.vn)
- Đặng Tấn Phát, MSc. (dtphat@hcmus.edu.vn)
- Nguyễn Như Hoàng, MSc. (ninhoang@hcmus.edu.vn)
- Huỳnh Thị Minh Tuyền, M2. (htmtuyen@hcmus.edu.vn)
- Hồ Thanh Bảo, BSc. (htbao@hcmus.edu.vn)

CHƯƠNG 2: KIẾN TRÚC HỆ THỐNG VÀ LÝ THUYẾT CƠ SỞ

Chương 2 trình bày các khái niệm lý thuyết cơ bản liên quan đến hệ thống được triển khai trong báo cáo này, bao gồm Hệ thống trên Chip (SoC), bộ xử lý mềm Nios V, Truy cập Bộ nhớ Trực tiếp (DMA), chuẩn giao tiếp Avalon, và cuối cùng là thiết kế và kiến trúc cụ thể của bộ điều khiển DMA tùy chỉnh được sử dụng.

2.1 Tổng quan về Hệ thống trên Chip (SoC)

System-on-Chip (SoC) là một vi mạch (IC) tích hợp hầu hết hoặc tất cả các thành phần của một máy tính hoặc hệ thống điện tử khác vào một chip duy nhất. Nó có thể chứa các thành phần kỹ thuật số, tương tự, tín hiệu hỗn hợp và thường cả tần số vô tuyến - tất cả trên một đế chip. Một SoC điển hình bao gồm:

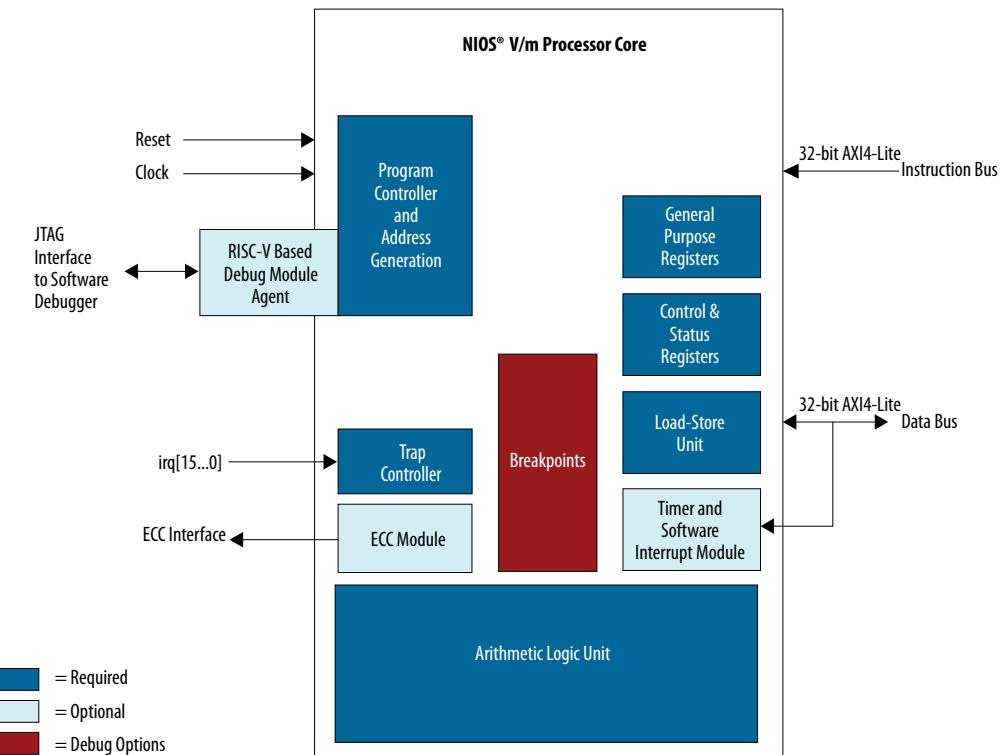
- Một hoặc nhiều lõi vi xử lý (Microprocessor cores) hoặc vi điều khiển (Microcontroller cores).
- Các khối bộ nhớ như RAM, ROM, EEPROM hoặc Flash.
- Các bộ định thời (Timers), bộ đếm (Counters).
- Các giao tiếp ngoại vi như UART, SPI, I2C, USB.
- Các khối xử lý tín hiệu số (DSP).
- Các bộ chuyển đổi tương tự-số (ADC) và số-tương tự (DAC).
- Hệ thống kết nối nội bộ (interconnect) như bus (ví dụ: AMBA AXI, Avalon).

Trong bối cảnh FPGA, SoC thường đề cập đến việc tích hợp một hoặc nhiều lõi xử lý mềm (soft-core) hoặc cứng (hard-core) cùng với các ngoại vi và bộ nhớ trên cùng một cấu trúc FPGA, sử dụng các công cụ như Intel Platform Designer.

2.2 Bộ xử lý mềm Intel Nios V/m

Nios V là thế hệ bộ xử lý mềm (soft-core processor) mới nhất của Intel dựa trên kiến trúc tập lệnh (ISA) RISC-V mã nguồn mở [9, 10, 1]. Là một bộ xử lý mềm, Nios V được triển khai hoàn toàn bằng logic FPGA, mang lại sự linh hoạt cao trong việc cấu hình và tích hợp vào các thiết kế SoC.

Phiên bản được sử dụng trong báo cáo này là Nios V/m (RV32IMAZicsr), được tối ưu hóa cho các ứng dụng vi điều khiển. Nó phù hợp cho các nhiệm vụ điều khiển, quản lý ngoại vi và xử lý cũng như nhu cầu debug cơ bản trong hệ thống nhúng, đáp ứng nhu cầu của một bộ vi xử lý cơ bản. Nios V/m có thể được cấu hình với pipeline 5-stage kinh điển (Fetch, Decode, Execute, Memory, Write back) để tăng hiệu suất (hoặc non-pipelined để tối ưu diện tích) [10].



Hình 2.1: Kiến trúc Nios V/m [10].

2.3 Truy cập Bộ nhớ Trực tiếp (DMA)

Direct Memory Access (DMA) thường được biết đến là một phần cứng của hệ thống máy tính cho phép một số hệ thống con phần cứng nhất định truy cập bộ nhớ hệ thống chính (RAM) để đọc và/hoặc ghi một cách độc lập với bộ xử lý trung tâm (CPU).

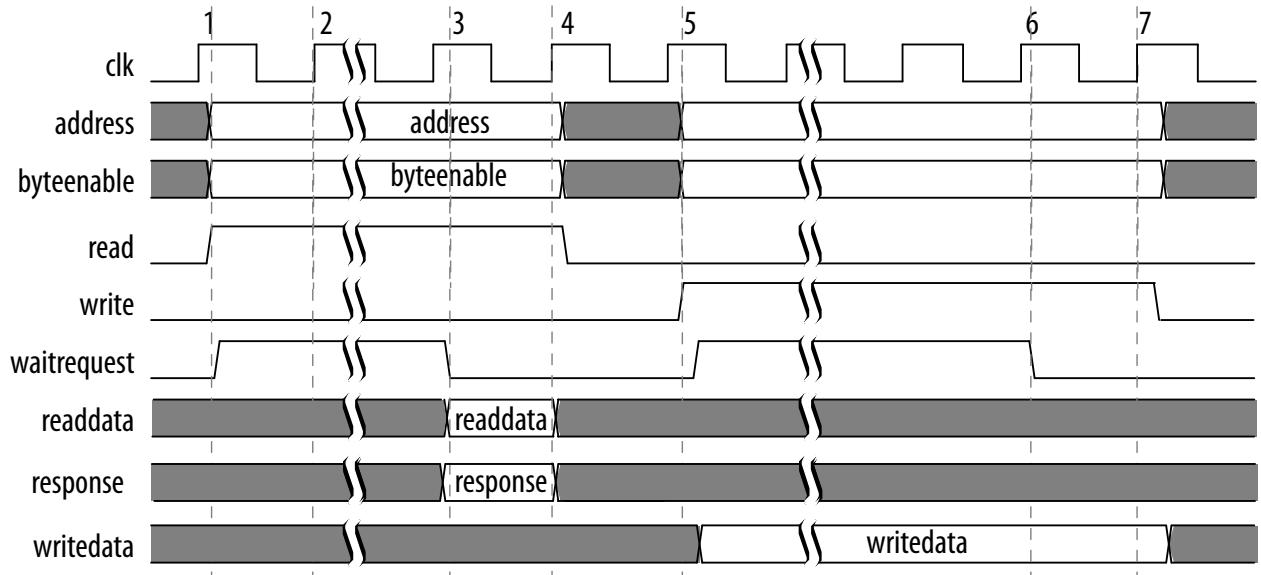
- Mục đích:** Giảm tải cho CPU trong các tác vụ truyền dữ liệu khối lượng lớn giữa bộ nhớ và các thiết bị ngoại vi (hoặc giữa các vùng nhớ khác nhau). Khi CPU khởi tạo một hoạt động DMA, nó có thể thực hiện các công việc khác trong khi bộ điều khiển DMA (DMA Controller - DMAC) thực hiện việc truyền dữ liệu.

- **Hoạt động:** CPU cấu hình DMAC với địa chỉ nguồn, địa chỉ đích, và số lượng dữ liệu cần truyền. Sau đó, CPU ra lệnh cho DMAC bắt đầu. DMAC chiếm quyền điều khiển bus hệ thống (hoặc sử dụng một bus riêng) để thực hiện việc truyền dữ liệu trực tiếp. Khi hoàn tất, DMAC thường thông báo cho CPU thông qua một ngắt (interrupt) hoặc bằng cách đặt một cờ trạng thái.
- **Lợi ích:** Tăng throughput dữ liệu, giải phóng CPU cho các tác vụ tính toán khác, cải thiện hiệu suất hệ thống tổng thể, đặc biệt quan trọng trong các hệ thống xử lý dữ liệu lớn hoặc thời gian thực.

2.4 Giao tiếp Hệ thống: Bus Avalon

Trong các hệ thống SoC được thiết kế trên nền tảng FPGA của Intel bằng công cụ Platform Designer, Giao tiếp Bus Avalon đóng vai trò là chuẩn kết nối (interconnect) nền tảng [3].

Đặc tả Avalon bao gồm nhiều loại giao tiếp con, trong đó Avalon Memory-Mapped (Avalon-MM) là giao tiếp được sử dụng chủ yếu trong thiết kế bộ điều khiển DMA này để truy cập bộ nhớ và thanh ghi điều khiển [3].



Hình 2.2: Các giao dịch đọc và ghi Avalon-MM diễn hình với tín hiệu waitrequest [3].

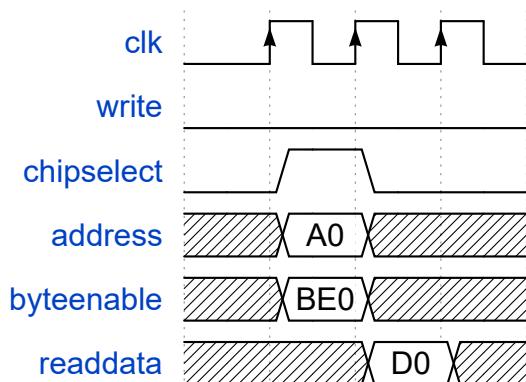
2.4.1 Giao tiếp Bus Avalon Memory Mapped Interface (Avalon-MM)

Avalon-MM là một giao diện dựa trên địa chỉ, được thiết kế theo kiến trúc Master-Slave cho các giao dịch truy cập vào không gian bộ nhớ hoặc thanh ghi [3].

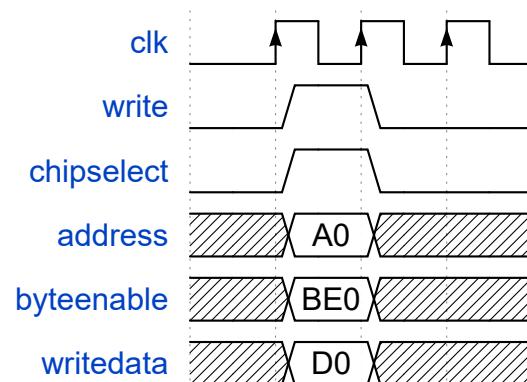
2.4.1.1 Đặc tả và Yêu cầu Giao thức

- Vai trò Master/Slave:

- Một giao dịch luôn được khởi tạo bởi Master (ví dụ: CPU, DMA Master) và được phản hồi bởi Slave (ví dụ: Bộ nhớ, Thanh ghi DMA).
- Master chịu trách nhiệm cung cấp địa chỉ và tín hiệu điều khiển (**read**, **write**, **byteenable**), trong khi Slave giải mã địa chỉ, phản hồi bằng dữ liệu (**readdata**) hoặc chấp nhận ghi dữ liệu (**writedata**), và có thể sử dụng tín hiệu điều khiển luồng (**waitrequest**, **readdatavalid**).



(a) Giản đồ sóng đọc Memory.

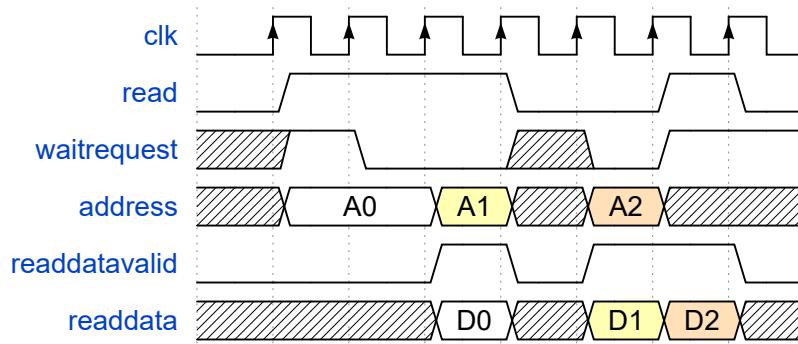


(b) Giản đồ sóng ghi Memory.

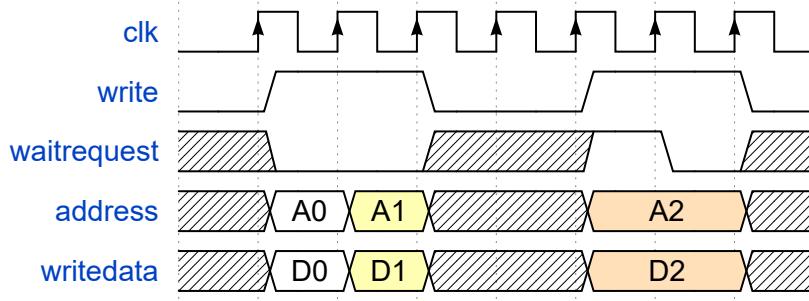
Hình 2.3: Giản đồ sóng đọc và ghi bộ nhớ Memory qua giao tiếp Avalon-MM.

- Giao dịch Đọc (Read Transaction):

- Master kích hoạt tín hiệu **read=1** và đưa ra **address**.
- Slave, sau khi giải mã địa chỉ và sẵn sàng dữ liệu, sẽ đặt dữ liệu lên **readdata**.
- **Điều khiển luồng (waitrequest)**: Nếu Slave cần thêm thời gian (ví dụ: bộ nhớ chậm), nó sẽ kích hoạt **waitrequest=1**. Master *phải* giữ nguyên **address** và **read** cho đến khi **waitrequest=0**.
- **Tính hợp lệ dữ liệu (readdatavalid)**: Trong các giao dịch đọc có độ trễ thay đổi (variable-latency reads, thường là pipelined), Slave sẽ kích hoạt **readdatavalid=1** cùng lúc với **readdata** để báo hiệu dữ liệu hợp lệ. Master *phải* chỉ lấy dữ liệu khi **readdatavalid=1** (và **waitrequest=0**). Giao diện Read Master của DMA này (**READ_MASTER**) được thiết kế để hỗ trợ **readdatavalid**.



(a) Giản đồ sóng đọc.



(b) Giản đồ sóng ghi.

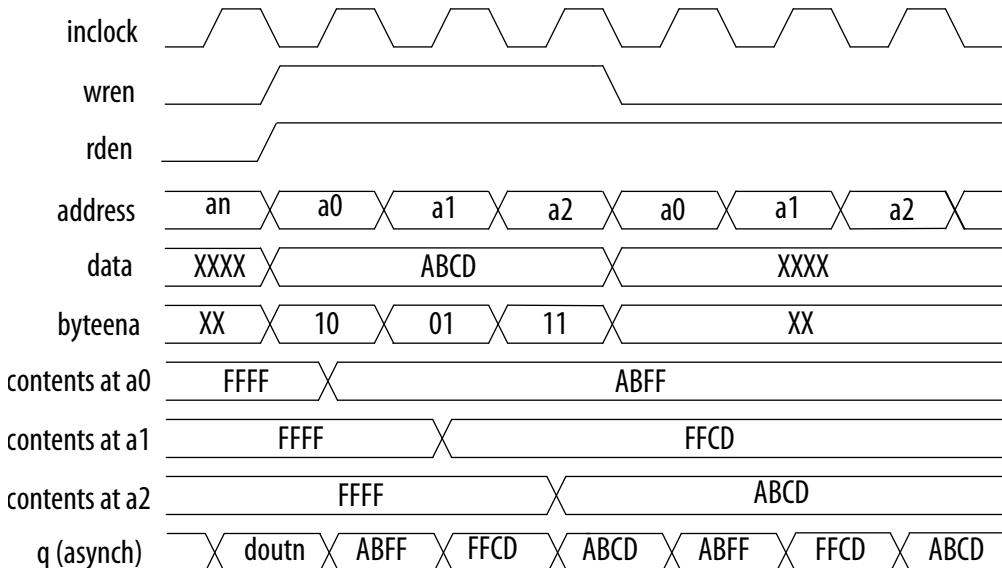
Hình 2.4: Giản đồ sóng đọc và ghi của giao tiếp Avalon-MM Master.

- Giản đồ sóng đọc điển hình được minh họa trong Hình 2.2 (phần Read), Hình 2.7(a), 2.4a, và 2.3(a).

- **Giao dịch Ghi (Write Transaction):**

- Master kích hoạt `write=1`, đưa ra `address`, `writedata`, và `byteenable`.
- **Điều khiển luồng (waitrequest):** Nếu Slave cần thêm thời gian để chuẩn bị nhận dữ liệu, nó sẽ kích hoạt `waitrequest=1`. Master **phải** giữ nguyên tất cả tín hiệu (`address`, `write`, `writedata`, `byteenable`) cho đến khi `waitrequest` bị hủy kích hoạt (`=0`). Slave sẽ ghi dữ liệu vào chu kỳ clock mà `write=1` và `waitrequest=0`.
- Tín hiệu `byteenable` cho phép ghi vào các byte cụ thể trong word dữ liệu (ví dụ: `4'b0011` chỉ ghi 2 byte thấp). Thiết kế DMA này sử dụng `byteenable=4'b1111` để ghi toàn bộ word 32-bit (Hình 2.5).
- Giản đồ sóng ghi điển hình được minh họa trong Hình 2.2 (phần Write), Hình 2.7(b), 2.4b, và 2.3(b).

Việc hiểu rõ và tuân thủ các yêu cầu về thời gian và hành vi của các tín hiệu này là cực kỳ quan trọng khi thiết kế một thành phần IP tương thích Avalon-MM.



For this functional waveform, New Data Mode is selected.

Hình 2.5: Giản đồ sóng chức năng của Byte Enable. Hình này minh họa ảnh hưởng của byte enable lên dữ liệu được ghi vào và đọc ra từ bộ nhớ [4].

2.5 Thiết kế và Kiến trúc Bộ điều khiển DMA Tùy chỉnh

Mục tiêu chính của việc thiết kế bộ điều khiển DMA này là tạo ra một thành phần phần cứng có khả năng truyền dữ liệu hiệu quả giữa các vùng nhớ trong hệ thống SoC trên FPGA, qua đó giảm tải cho bộ xử lý trung tâm Nios V/m. Để đạt được mục tiêu này, việc thiết kế phải tuân thủ các đặc tả giao tiếp của hệ thống bus Avalon-MM đã được trình bày ở Mục 2.4, đồng thời đảm bảo tính đúng đắn và hiệu quả của quá trình truyền dữ liệu.

Dựa trên yêu cầu về khả năng truyền dữ liệu tốc độ cao và độc lập với CPU, cũng như đặc tả của bus Avalon-MM, bộ điều khiển DMA tùy chỉnh được thiết kế với kiến trúc và các lựa chọn sau:

2.5.1 Mục tiêu và Bối cảnh: Truyền dữ liệu giữa Bộ nhớ On-Chip

Một trong những ứng dụng ban đầu và quan trọng thúc đẩy việc thiết kế bộ điều khiển DMA này là nhu cầu truyền dữ liệu nhanh chóng và hiệu quả giữa các khối bộ nhớ trong chip (On-chip Memory - OCM) trên FPGA. Bộ nhớ on-chip, thường được hiện thực hóa bằng các khối RAM tích hợp sẵn trong cấu trúc FPGA [4], cung cấp tốc độ truy cập rất cao so với bộ nhớ ngoài (off-chip). Việc sử dụng DMA để di chuyển dữ liệu giữa các khối OCM (ví dụ: từ một vùng RAM chứa dữ liệu thô sang

một vùng RAM khác để xử lý) giúp giải phóng CPU Nios V/m khỏi các vòng lặp sao chép bộ nhớ (`memcpy`) tốn thời gian, cho phép CPU tập trung vào các nhiệm vụ tính toán hoặc điều khiển khác.

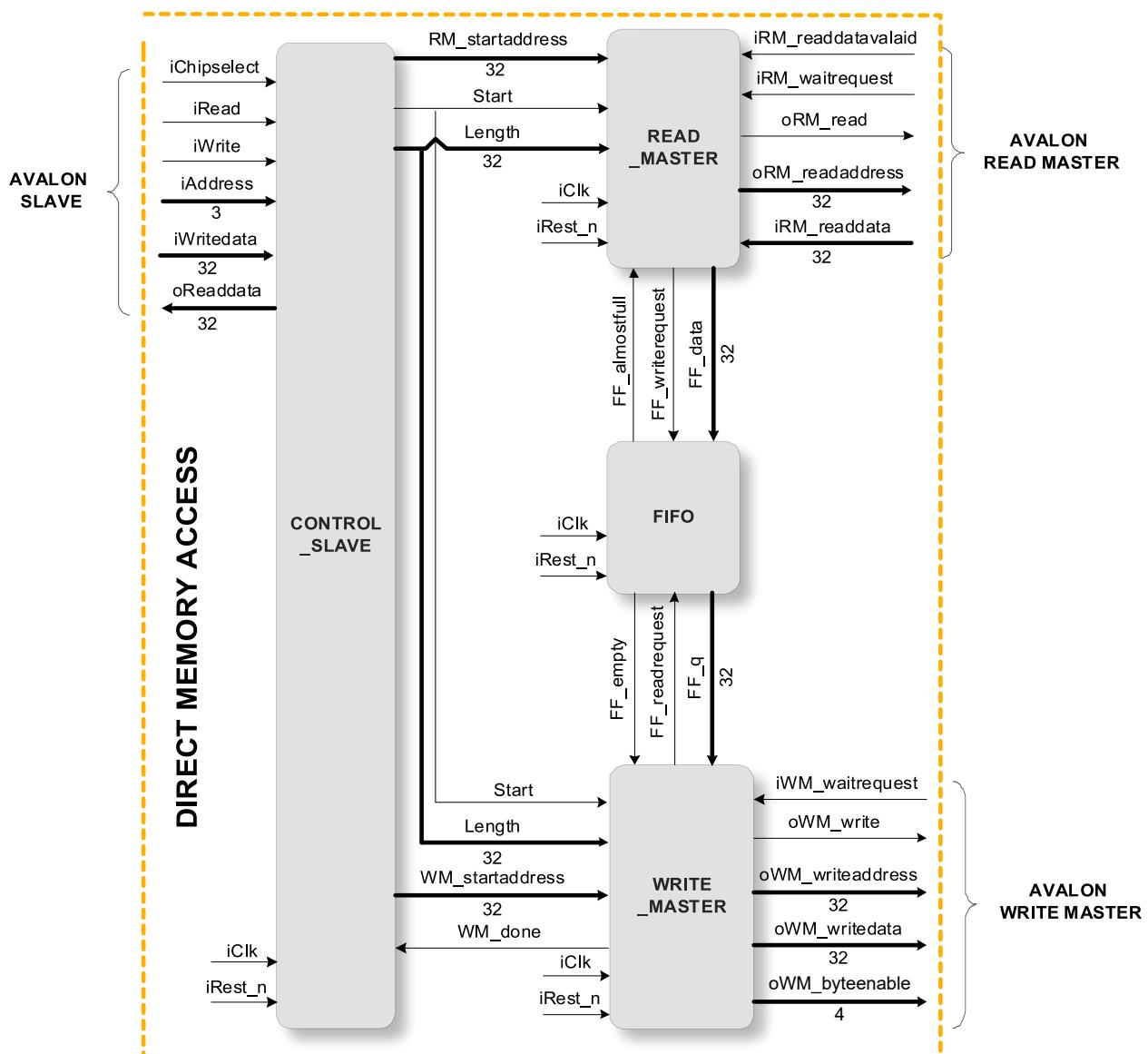
Do đó, các giao diện Avalon-MM Master của `READ_MASTER` và `WRITE_MASTER` được thiết kế với giả định ban đầu là tương tác với các Slave là bộ nhớ on-chip. Điều này có nghĩa là chúng được tối ưu cho các giao dịch truy cập bộ nhớ có độ trễ tương đối thấp và ổn định [4], như được minh họa trong các giản đồ sóng đọc/ghi bộ nhớ (Hình 2.3). Mặc dù vậy, thiết kế vẫn tuân thủ đầy đủ đặc tả Avalon-MM [3], cho phép DMA hoạt động với các loại bộ nhớ hoặc ngoại vi khác có hành vi thời gian khác nhau, miễn là chúng cũng tuân thủ chuẩn Avalon-MM.

2.5.2 Kiến trúc Tổng thể và Luồng Dữ liệu

Kiến trúc DMA được chọn dựa trên mô hình tách biệt luồng đọc và ghi dữ liệu để tối đa hóa thông lượng, bao gồm các khôi chính sau:

- **CONTROL_SLAVE** ('`CONTROL_SLAVE.v`'): Hoạt động như giao diện điều khiển Avalon-MM Slave cho CPU.
- **READ_MASTER** ('`READ_MASTER.v`'): Hoạt động như Avalon-MM Master để đọc dữ liệu từ bộ nhớ nguồn.
- **WRITE_MASTER** ('`WRITE_MASTER.v`'): Hoạt động như Avalon-MM Master để ghi dữ liệu vào bộ nhớ đích.
- **FIFO** ('`FIFO_IP.v`'): Bộ đệm dữ liệu giữa Read và Write Master, được tạo từ IP Catalog của Quartus.

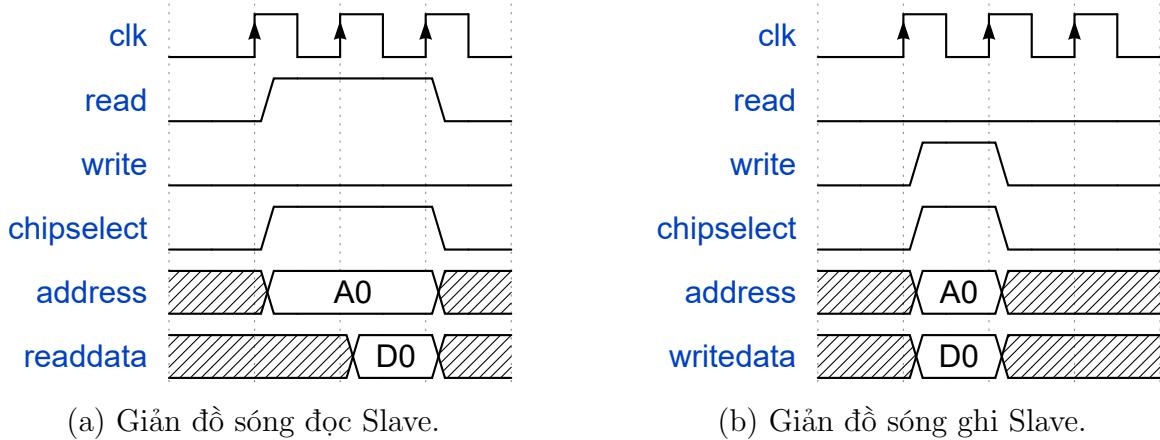
Luồng dữ liệu cơ bản diễn ra như sau: CPU cấu hình DMA thông qua `CONTROL_SLAVE` (ghi địa chỉ nguồn, đích, độ dài). CPU kích hoạt DMA bằng cách ghi vào thanh ghi điều khiển. Khi được kích hoạt (`Start=1`), `READ_MASTER` bắt đầu đọc dữ liệu từ bộ nhớ nguồn theo địa chỉ và độ dài đã cấu hình, đẩy dữ liệu vào FIFO. Đồng thời, `WRITE_MASTER` đọc dữ liệu từ FIFO (khi FIFO không rỗng - `FF_empty=0`) và ghi vào bộ nhớ đích theo cấu hình. `WRITE_MASTER` báo hiệu hoàn thành (`WM_done=1`) khi ghi xong từ cuối cùng, tín hiệu này được `CONTROL_SLAVE` sử dụng để cập nhật thanh ghi trạng thái và tự động xóa bit Go.



Hình 2.6: Sơ đồ khái tổng thể của bộ điều khiển DMA.

2.5.3 Thiết kế Module Điều khiển ('CONTROL_SLAVE')

- Yêu cầu:** Cung cấp giao diện để CPU cấu hình (địa chỉ nguồn/dích, độ dài), khởi động và kiểm tra trạng thái DMA.

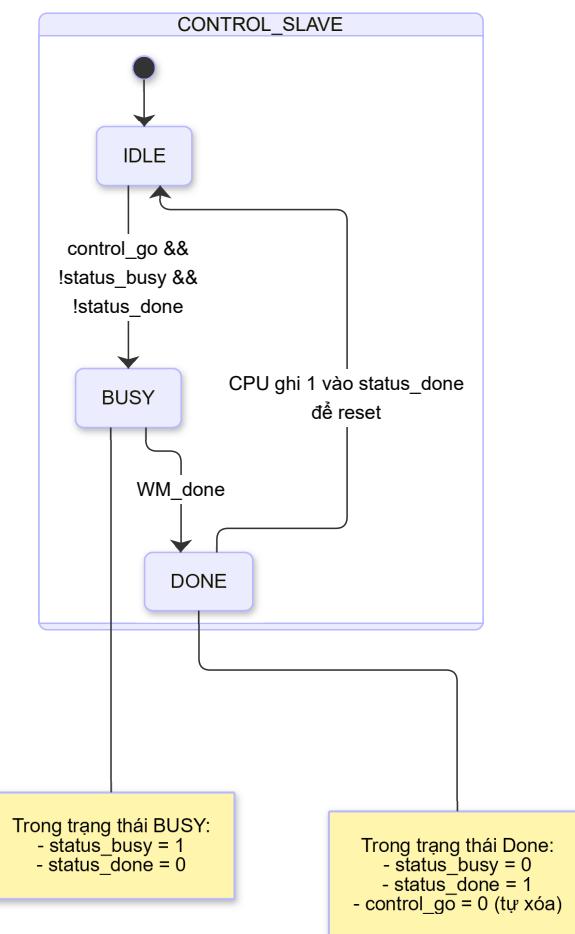


Hình 2.7: Giản đồ sóng đọc và ghi của giao tiếp Avalon-MM Slave.

- Giải pháp thiết kế (Dựa trên Avalon-MM Slave):**

- **Thanh ghi ánh xạ bộ nhớ:** Định nghĩa các thanh ghi nội bộ cho địa chỉ đọc (`readaddress_reg`), địa chỉ ghi (`writeaddress_reg`), độ dài (`length_reg`), điều khiển (`control_go`), và trạng thái (`status_busy`, `status_done`). Các thanh ghi này được gán các địa chỉ offset cụ thể (0, 1, 2, 4, 5) trong không gian địa chỉ 3-bit của module.
- **Logic giải mã địa chỉ:** Module này hoạt động như một Slave đơn giản, không sử dụng `waitrequest`. Nó giải mã `iAddress` khi `iChipselect=1` để đọc/ghi các thanh ghi nội bộ.
- **Logic đọc/ghi:**
 - * **Khi đọc (`iRead=1`):** Dữ liệu từ thanh ghi tương ứng (địa chỉ nguồn/dích, độ dài, control, status) được đưa ra `oReaddata`.
 - * **Khi ghi (`iWrite=1`):** Dữ liệu `iWritedata` được ghi vào thanh ghi cấu hình (địa chỉ nguồn/dích, độ dài) hoặc bit GO của thanh ghi điều khiển. Ghi 1 vào bit 0 của thanh ghi trạng thái sẽ xóa cờ Done.
- **Logic điều khiển/trạng thái:**
 - * Bit GO (bit 0 của thanh ghi điều khiển `iWritedata[0]` tại địa chỉ 4) được CPU ghi (`iWrite=1`) để khởi động DMA; giá trị này được chốt vào thanh ghi nội bộ `control_go`.

- * Tín hiệu Start được gán bằng control_go. Module cập nhật status_busy và status_done dựa trên Start và tín hiệu WM_done từ WRITE_MASTER.
 - * Khi WM_done tích cực, status_busy bị xóa, status_done được đặt, và quan trọng là control_go cũng tự động bị xóa để ngăn DMA tự khởi động lại. Bit status_done có thể được CPU xóa bằng cách ghi 1 vào bit 0 của thanh ghi trạng thái (địa chỉ 5).
- **Máy trạng thái hữu hạn (FSM):** Tham khảo hình 2.8.
 - **Mã nguồn tham khảo:** Phụ lục A.3.



Hình 2.8: Sơ đồ trạng thái của module CONTROL_SLAVE.

2.5.4 Thiết kế Module Master Đọc (READ_MASTER)

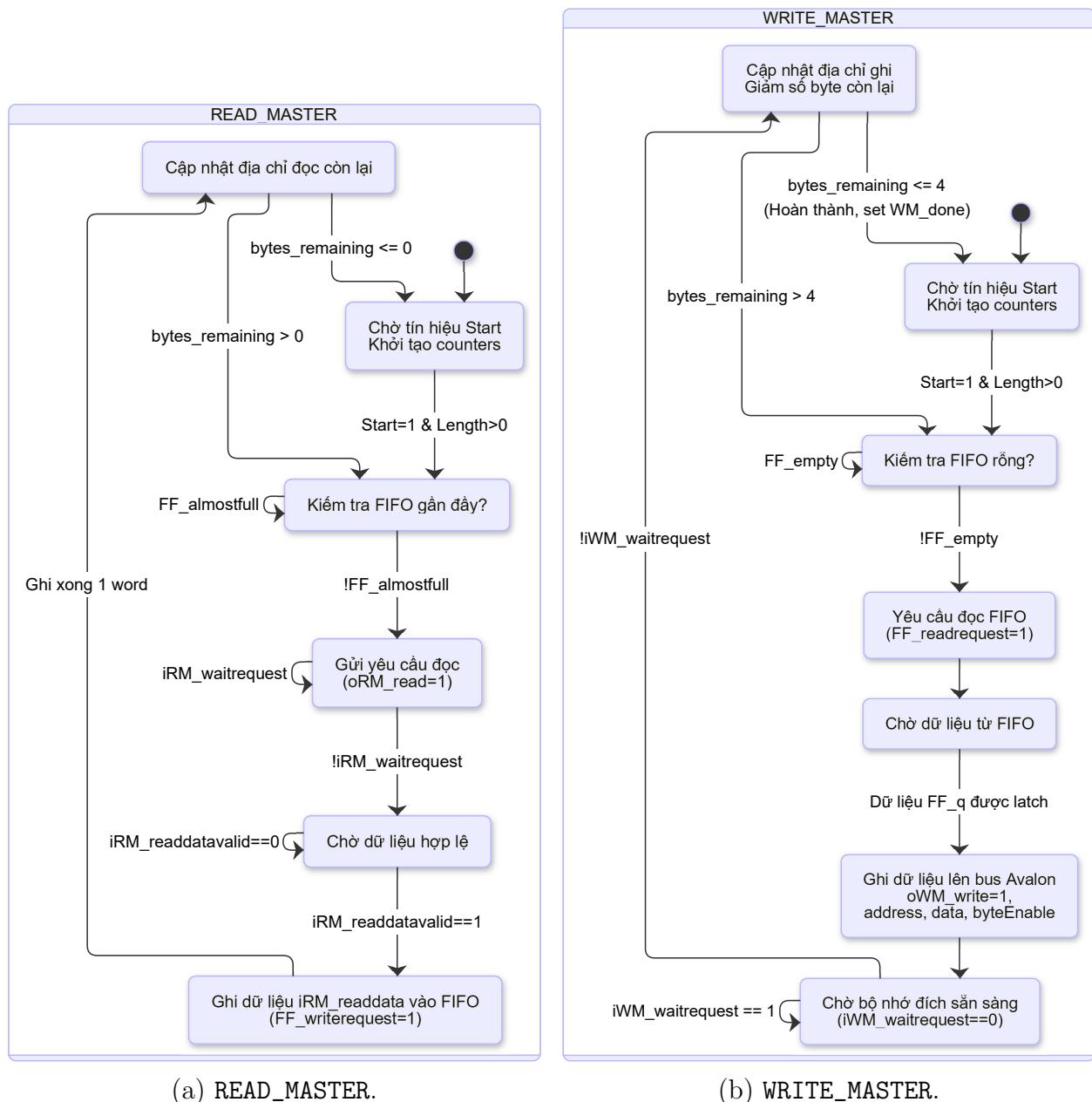
- **Yêu cầu:** Đọc tuần tự dữ liệu từ bộ nhớ nguồn theo cấu hình và đẩy vào FIFO, tuân thủ giao thức Avalon-MM Read Master và xử lý điều khiển luồng từ FIFO.

- **Giải pháp thiết kế (Dựa trên Avalon-MM Master và FSM):**
 - **Giao diện Avalon-MM Master:** Module chủ động tạo giao dịch đọc, xử lý `iRM_waitrequest` và `readdatavalid`.
 - * Kích hoạt `oRM_read=1` và đưa ra `oRM_readaddress`.
 - * Nếu `iRM_waitrequest=1`, giữ nguyên tín hiệu và trạng thái.
 - * Khi `iRM_waitrequest=0`, chờ `iRM_readdatavalid=1` mới lấy dữ liệu `iRM_readdata` và ghi vào FIFO. Hình 2.4a.
 - **Giao diện FIFO:** Module tạo tín hiệu `FF_writerequest=1` và đưa `FF_data` khi có dữ liệu hợp lệ từ Avalon bus và sẵn sàng ghi vào FIFO. Nó phản hồi với `FF_almostfull` bằng cách tạm dừng gửi yêu cầu đọc Avalon mới để tránh làm đầy FIFO.
 - **Máy trạng thái hữu hạn (FSM):** Tham khảo hình 2.9a.
 - **Mã nguồn tham khảo:** Phụ lục A.4.
- ### 2.5.5 Thiết kế Module Master Ghi (WRITE_MASTER)
- **Yêu cầu:** Đọc dữ liệu từ FIFO (khi `FF_empty=0`) và ghi tuần tự vào bộ nhớ đích theo cấu hình (`WM_startaddress`, `Length`) khi `Start=1`, tuân thủ giao thức Avalon-MM Write Master (xử lý `iWM_waitrequest`) và báo hiệu hoàn thành (`WM_done=1`).
 - **Giải pháp thiết kế (Dựa trên Avalon-MM Master và FSM):**
 - **Giao diện FIFO:** Module kiểm tra `FF_empty`. Nếu không rỗng, nó tạo tín hiệu `FF_readrequest=1` để lấy dữ liệu `FF_q` trong chu kỳ tiếp theo.
 - **Giao diện Avalon-MM Master:** Module chủ động tạo giao dịch ghi, xử lý `waitrequest`.
 - * Kích hoạt `oWM_write=1`, đưa ra `oWM_writeaddress`, `oWM_writedata` (lấy từ thanh ghi nội bộ chứa dữ liệu đọc từ FIFO), và `oWM_bytelenable = 4'b1111`.
 - * Nếu `iWM_waitrequest=1`, giữ nguyên tất cả tín hiệu đầu ra và trạng thái.
 - * Khi `iWM_waitrequest=0`, giao dịch ghi được xem là hoàn thành, FSM chuyển trạng thái cập nhật bộ đếm (`UPDATE_CNT`)..

- Máy trạng thái hữu hạn (FSM): Tham khảo hình 2.9b.

- Mã nguồn tham khảo: Phụ lục A.5.

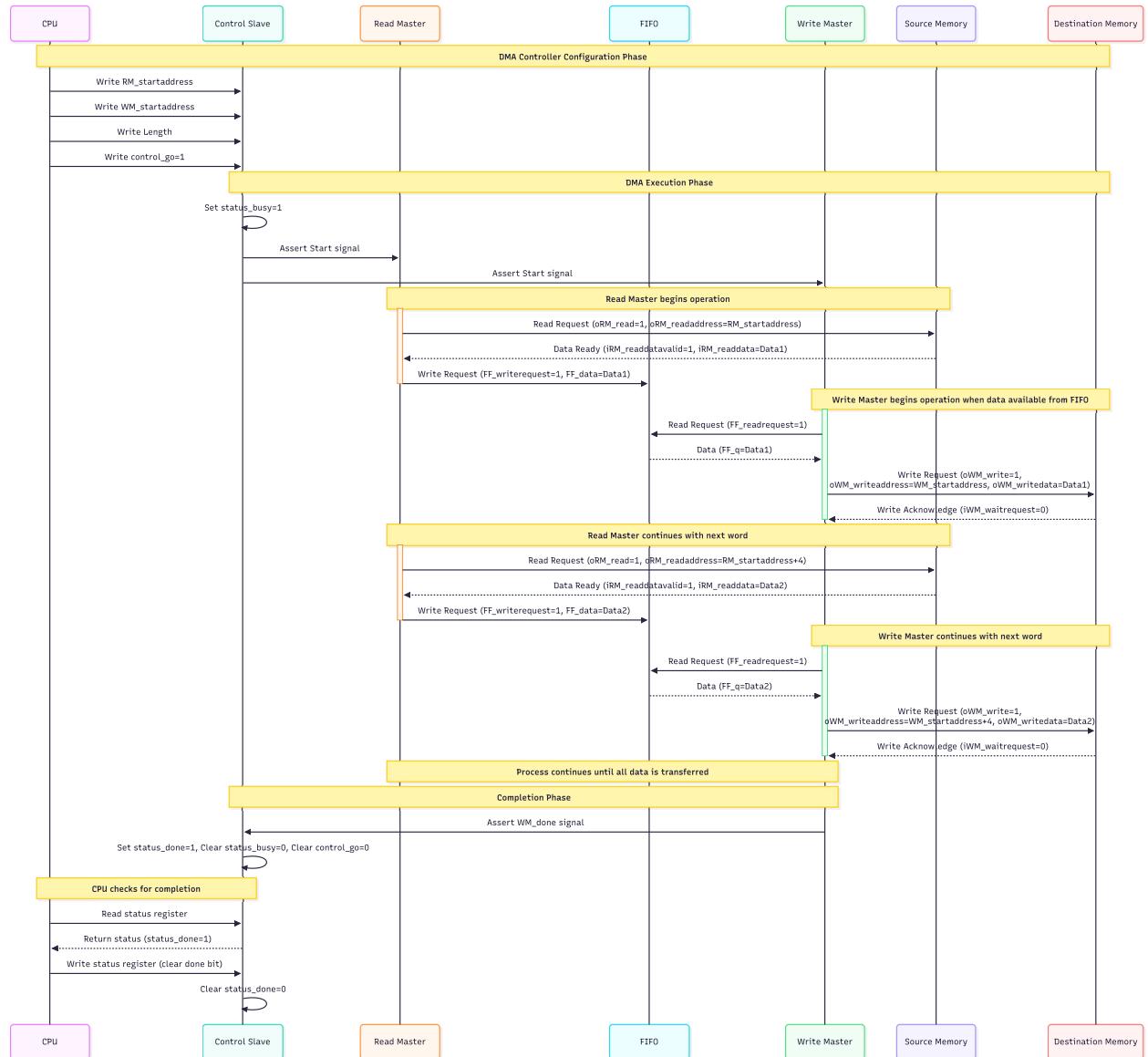
Việc thiết kế các giao diện DMA dựa trên và tuân thủ chặt chẽ đặc tả Avalon-MM [3] là yếu tố then chốt đảm bảo bộ điều khiển DMA tùy chỉnh có thể hoạt động chính xác và tích hợp liền mạch vào hệ thống SoC Nios V trên Platform Designer.



Hình 2.9: Sơ đồ trạng thái các module Master.

2.5.6 Tích hợp và Tín hiệu Giao tiếp

Module DMAController.v (Phụ lục A.2) đóng vai trò kết nối các module con (CONTROL_SLAVE, READ_MASTER, WRITE_MASTER, FIFO_IP) lại với nhau và cung cấp giao diện mức cao cho hệ thống. Bảng 2.1 liệt kê chi tiết các tín hiệu vào/ra của module này, thể hiện giao tiếp với CPU (qua Avalon Slave) và với bộ nhớ (qua Avalon-MM Master). Hình 2.10 mô tả luồng hoạt động tuần tự của DMA.



Hình 2.10: Sơ đồ tuần tự hoạt động của DMA.

Bảng 2.1: Bảng mô tả tín hiệu chính của DMA Controller.

STT	Tên tín hiệu	Ngõ vào/ra	Số bit	Mô tả
1	iClk	In	1	Cấp xung clock 50 MHz cho DMA hoạt động
2	iReset_n	In	1	iReset_n = 0: reset lại DMA

Giao tiếp Bus Avalon Slave (Từ CPU tới DMA)

3	iChipselect	In	1	iChipselect = 1: cho phép truy xuất vào các thanh ghi trạng thái và điều khiển của DMA
4	iRead	In	1	iRead = 1: cho phép đọc giá trị của các thanh ghi trong DMA
5	iWrite	In	1	iWrite = 1: cho phép ghi giá trị vào các thanh ghi của DMA
6	iAddress	In	3	Chọn địa chỉ của thanh ghi cần đọc/ghi
7	iWritedata	In	32	Truyền giá trị cần ghi vào thanh ghi từ Avalon bus
8	oReaddata	Out	32	Xuất giá trị cần đọc từ thanh ghi ra Avalon bus

Giao tiếp Bus Avalon Read Master (Từ DMA tới Bộ nhớ Nguồn)

9	iRM_readdatavalid	In	1	=1: báo hiệu iRM_readdata hợp lệ
10	iRM_waitrequest	In	1	=1: yêu cầu Read Master chờ
11	oRM_read	Out	1	=1: yêu cầu đọc dữ liệu từ oRM_readaddress
12	oRM_readaddress	Out	32	Địa chỉ byte của vùng nhớ cần đọc
13	iRM_readdata	In	32	Dữ liệu đọc được từ Avalon bus

Giao tiếp Bus Avalon Write Master (Từ DMA tới Bộ nhớ Đích)

14	iWM_waitrequest	In	1	=1: yêu cầu Write Master chờ
15	oWM_write	Out	1	=1: yêu cầu ghi dữ liệu vào oWM_writeaddress
16	oWM_writeaddress	Out	32	Địa chỉ byte của vùng nhớ cần ghi
17	oWM_writedata	Out	32	Dữ liệu cần ghi ra Avalon bus
18	oWM_bytenable	Out	4	Cho phép ghi từng byte (luôn là 4'b1111)

CHƯƠNG 3: TRIỂN KHAI VÀ KIỂM THỬ HỆ THỐNG NIOS V VỚI DMA

Chương này trình bày chi tiết quy trình từng bước được thực hiện để triển khai và kiểm thử một Hệ thống trên Chip (SoC) dựa trên bộ xử lý mềm (soft-core processor) Nios V/m của Intel, tích hợp một bộ điều khiển Truy cập Bộ nhớ Trực tiếp (Direct Memory Access (DMA)) tùy chỉnh. Nền tảng phần cứng mục tiêu là bo mạch phát triển (development board) Terasic DE10-Standard [12] trang bị FPGA Intel Cyclone V SoC. Việc triển khai sử dụng Phần mềm Quartus Prime Lite Edition của Intel, Platform Designer (trước đây là Qsys), và IDE Ashling RiscFree™ [2] cho việc phát triển và gỡ lỗi (debug) phần mềm.

3.1 Xây dựng Hệ thống Phần cứng trên Quartus Prime

Phần này bao gồm việc tạo dự án Quartus và thiết kế hệ thống phần cứng bằng Platform Designer.

1. Tạo Dự án Quartus:

- Khởi động Quartus Prime Lite Edition.
- Tạo một dự án mới bằng Trình hướng dẫn Dự án Mới (New Project Wizard) (File -> New Project Wizard...).
- Chỉ định thư mục làm việc (working directory) và tên dự án (project name) (ví dụ: D:\FETEL_WorkDir\Y4S2_DMANiosVIntern, DMANiosV) (Hình 3.1).
- Chọn thiết bị mục tiêu (target device) tương ứng với bo mạch DE10-Standard: Họ (Family) ‘Cyclone V’, Thiết bị (Device) 5CSXFC6D6F31C6 (Hình 3.2).

2. Chuẩn bị các tệp Verilog DMA Controller:

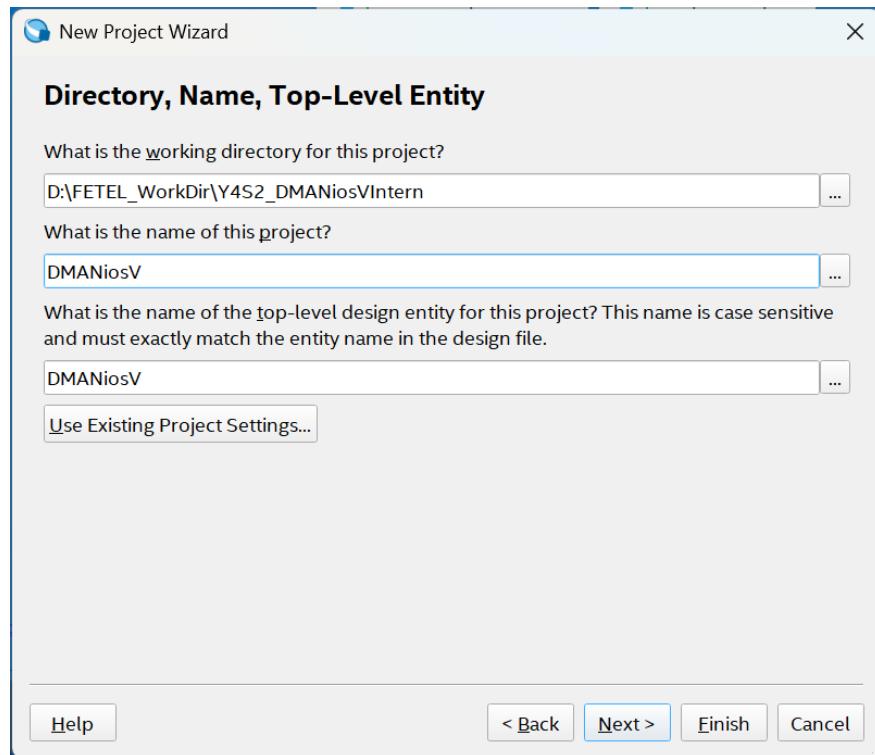
Sao chép các tệp nguồn Verilog được cung cấp cho bộ điều khiển DMA (DMAController.v (A.2), CONTROL_SLAVE.v (A.3), FIFO_IP.v, READ_MASTER.v (A.4), WRITE_MASTER.v (A.5)) vào thư mục dự án Quartus (Hình 3.3).

3. Tạo Hệ thống Platform Designer:

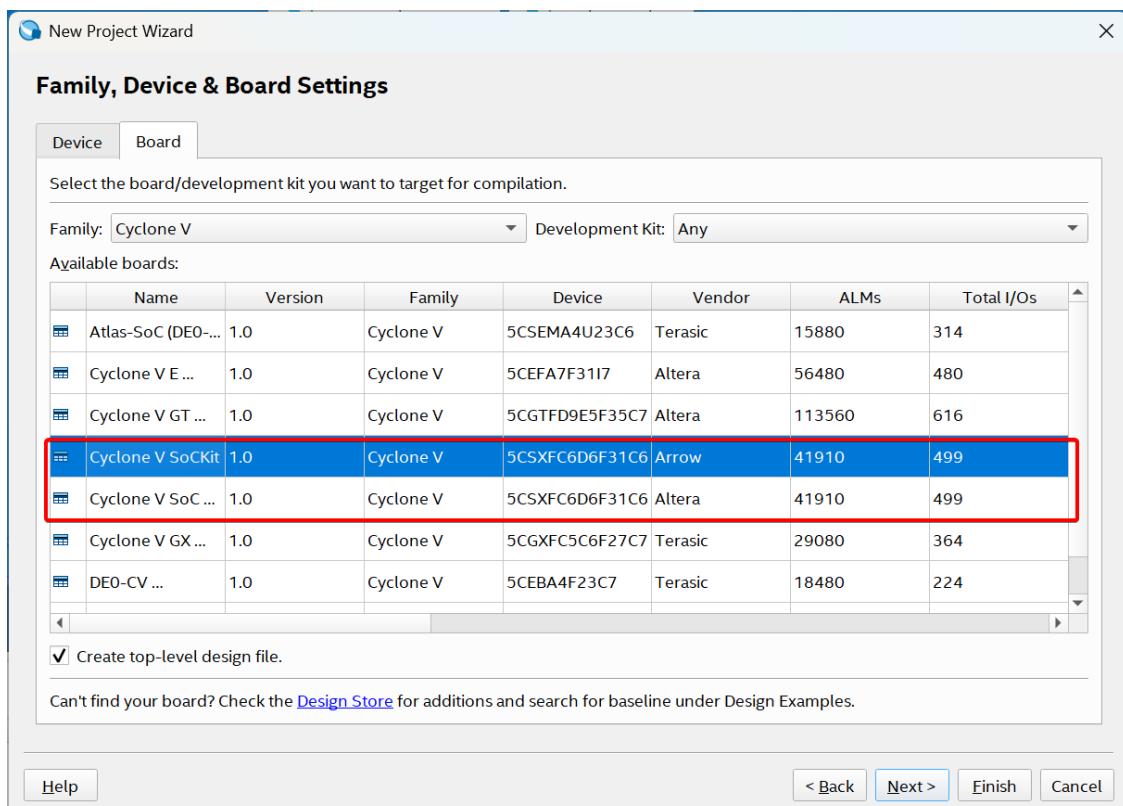
- Khởi chạy Platform Designer (Tools -> Platform Designer).

- **Thêm Bộ xử lý Nios V/m:** Trong IP Catalog, tìm và thêm "Nios V/m Microcontroller Intel FPGA IP". Trong cấu hình của nó, bật tùy chọn "Enable Reset from Debug Module" để dễ dàng reset phần mềm khi gỡ lỗi (Hình 3.4).
- **Thêm Bộ nhớ Trên Chip (On-Chip Memory):** Thêm instances của IP "On-Chip Memory (RAM or ROM)". Cấu hình cả hai là RAM. Đặt "Tổng dung lượng bộ nhớ (Total memory size)" cho mỗi bộ nhớ là 131072 byte (128 KB) (Cấu hình ví dụ Hình 3.5). Một bộ nhớ (`onchip_memory2_0`) cho lệnh (instructions), và bộ nhớ còn lại (`onchip_memory2_1`) sẽ được sử dụng cho dữ liệu (data).
- **Thêm JTAG UART:** Thêm "JTAG UART Intel FPGA IP" để giao tiếp (communication) giữa bộ xử lý Nios V và máy tính chủ (host PC) thông qua kết nối JTAG (cho việc in ra terminal `printf`).
- **Tạo IP DMA Controller:**
 - Trong IP Catalog, chọn "New Component...".
 - Trong Component Editor, đặt tên (ví dụ: `DMA_Controller`).
 - Chuyển đến tab "Files". Thêm tất cả các tệp Verilog DMA tùy chỉnh ('.v'). Đặt `DMAController.v` làm Top-Level File bằng cách nhấp đúp vào "Attributes".
 - Nhấp vào "Analyze Synthesis Files" để Quartus tổng hợp các tệp Verilog. Nhấn vào "Copy from Synthesis Files" để điền các tệp mô phỏng (simulation files) (Hình 3.7).
 - Chuyển đến tab "Signals". Sau đó cấu hình các tín hiệu như hình 3.8.
 - Chuyển đến tab "Signals & Interfaces". Dảm bảo các interface được cấu hình như sau (Hình 3.9, 3.10):
 - * Ngõ vào Đồng hồ (Clock input - `clk`): `clock_sink`.
 - * Ngõ vào Reset (`reset_n`): `reset_sink`.
 - * Associated Clock: `clk`.
 - * Associated Reset: `reset_n`.
 - Nhấp vào "Finish..." để lưu IP DMA Controller (tệp '.tcl').
- **Thêm IP DMA Controller vào Hệ thống SoC.**
- **Kết nối các Thành phần:** Thực hiện kết nối như hình 3.11.

- **Gán Địa chỉ Cơ sở (Assign Base Addresses):** Di đến menu "System" và chọn "Assign Base Addresses" (Hình 3.12). Công cụ sẽ tự động gán các địa chỉ không trùng lặp cho tất cả các giao diện slave. Các địa chỉ này được lưu trong thư viện BSP dưới file **system.h**.
 - **Thay Nios V (Vector Reset):** Vector reset (reset vector) của Nios V cần trỏ đến bộ nhớ chứa mã lệnh ban đầu (initial program code memory).
 - Ngắt kết nối `instruction_master` khỏi `onchip_memory2_1` (hình 3.13).
 - Mở lại cấu hình Nios V/m (nhấp đúp). Đặt "Bộ nhớ Vector Reset (Reset Agent)" thành `onchip_memory2_0.s1` (hình 3.14).
 - **Lưu Hệ thống:** Lưu hệ thống Platform Designer (**system.qsys**).
4. **Generate HDL:** Trong Platform Designer, nhấp vào nút "Generate HDL..." và nhấp vào "Generate" (Hình 3.15). Thao tác này tạo ra các tệp Verilog đại diện cho hệ thống SoC được kết nối với nhau. Sau khi tổng hợp thành công, ta tích hợp hệ thống vào thiết kế bằng việc thêm tệp '**.qip**' vào Project Quartus.
5. **Tích hợp Hệ thống vào Project Quartus:**
- Trong Quartus, thêm tệp '**.qip**' được tạo (**system/synthesis/system.qip**) vào project (Project -> Add/Remove Files in Project...) (Hình 3.16, Hình 3.17).
 - Tạo một tệp Verilog top-level (**DMANiosV.v** A.1) để khởi tạo hệ thống được tạo ra, kết nối các ngõ vào đồng hồ (clock) và reset của hệ thống tương ứng với ngõ vào đồng hồ của FPGA (**CLOCK_50**) và một nút reset (**KEY[0]**) (Hình 3.21).
6. **Import cấu hình chân (Import Pin Assignments):** Sử dụng tệp gán chân DE10-Standard ('.qsf') được cung cấp từ FPGAcademy [5], [6]. Trong Quartus, mở menu Assignments -> Import Assignments... (Hình 3.18). Duyệt đến tệp '**.qsf**' đã tải xuống và import nó (Hình 3.19).
7. **Biên dịch thiết kế (Compile Design):** Chạy quy trình biên dịch toàn bộ trong Quartus (Processing -> Start Compilation) (Hình 3.20). Xác minh biên dịch thành công và kiểm tra các gán chân trong Assignment Editor (Assignments -> Assignment Editor) (Hình 3.21). Thao tác này tạo ra tệp '**.sof**' cần thiết để lập trình FPGA.



Hình 3.1: Quartus: New Project Wizard: Chỉ định đường dẫn thư mục và tên Project.



Hình 3.2: Quartus: New Project Wizard: Device Selector

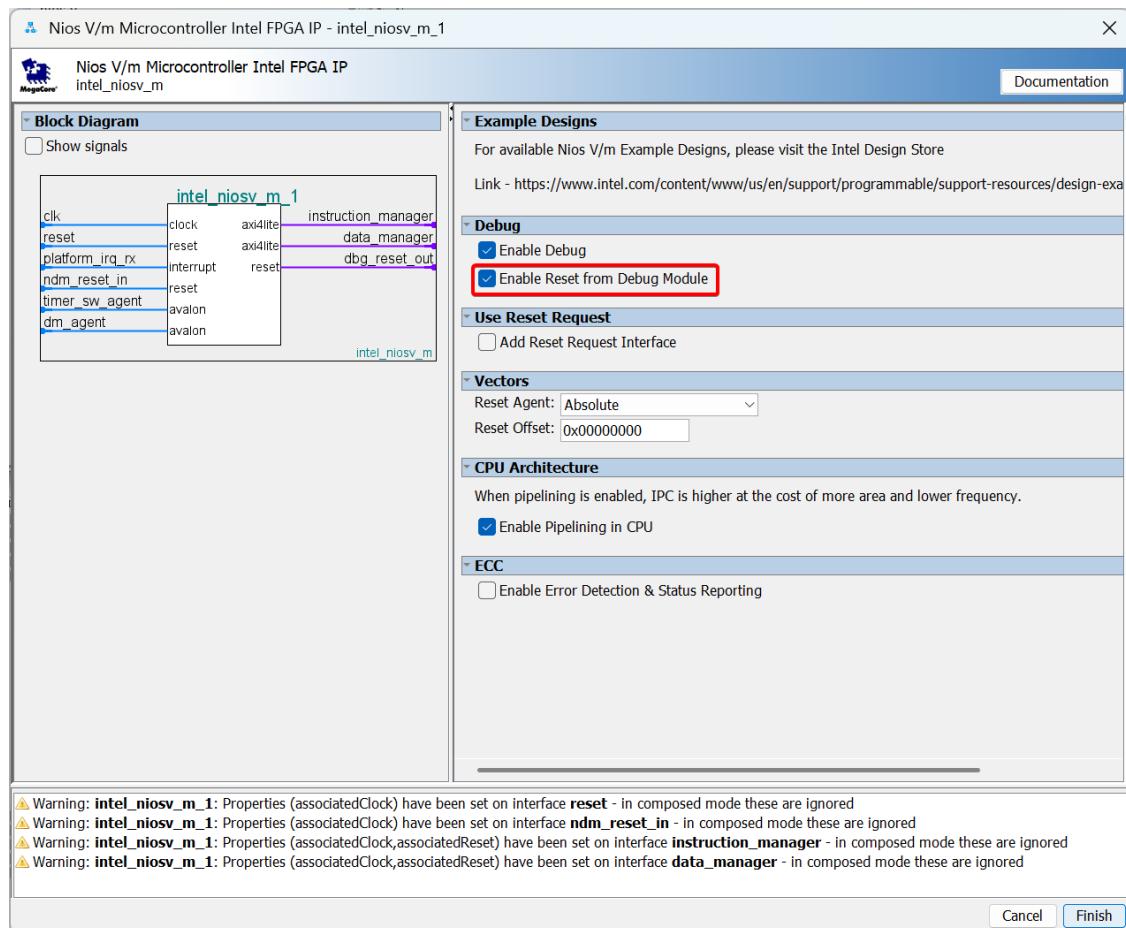
Y4S2_DMAniosVIntern > DMAC

Search DMAC

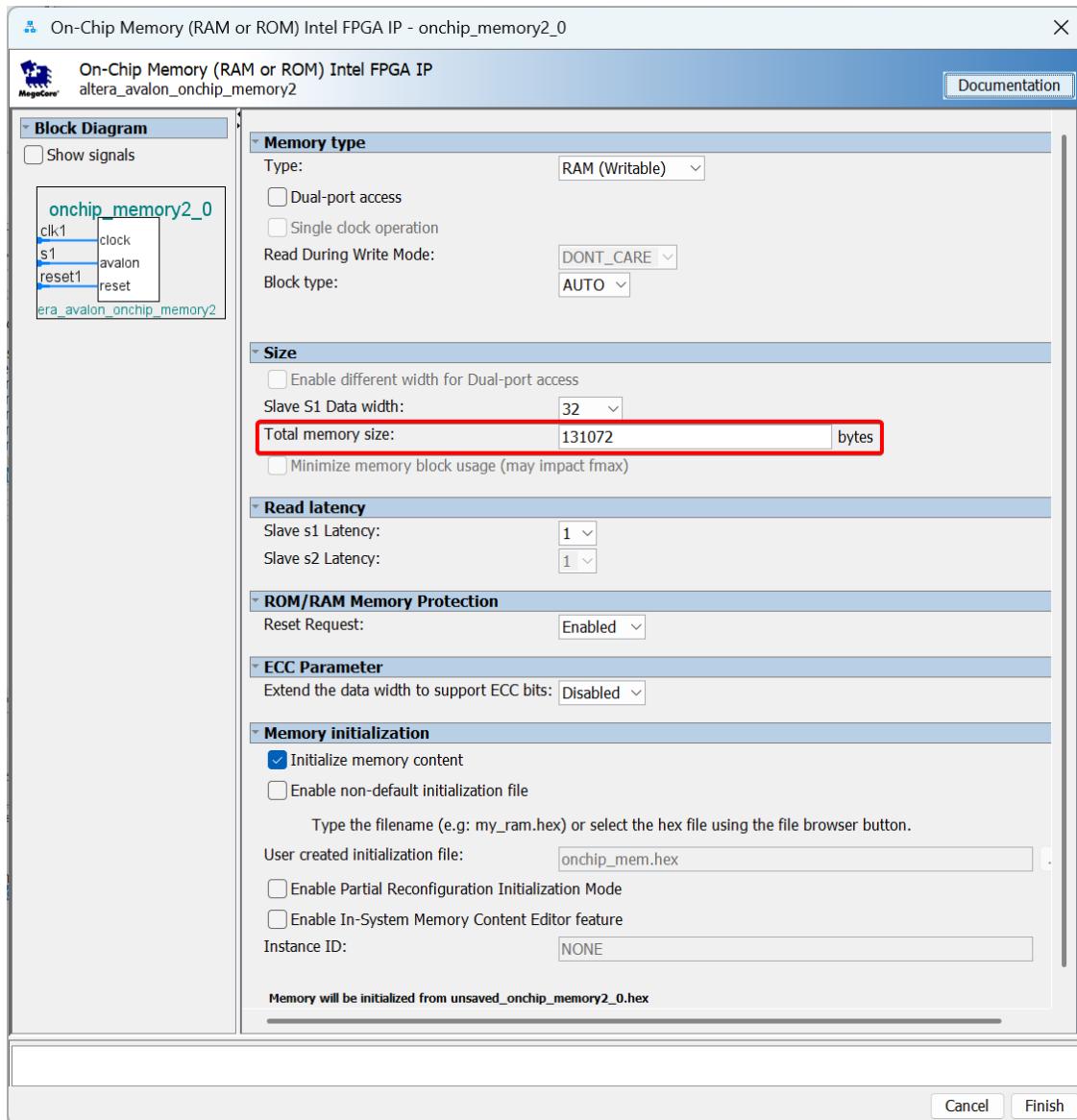
Sort View Details

Name	Date modified	Type	Size
CONTROL_SLAVE.v	4/9/2025 6:11 PM	V File	5 KB
DMAController.v	4/8/2025 7:31 PM	V File	4 KB
FIFO_IP.qip	3/17/2025 3:04 PM	QIP File	1 KB
FIFO_IP.v	3/17/2025 3:04 PM	V File	7 KB
FIFO_IP_bb.v	3/17/2025 3:04 PM	V File	6 KB
READ_MASTER.v	4/8/2025 7:25 PM	V File	5 KB
WRITE_MASTER.v	4/9/2025 6:31 PM	V File	7 KB

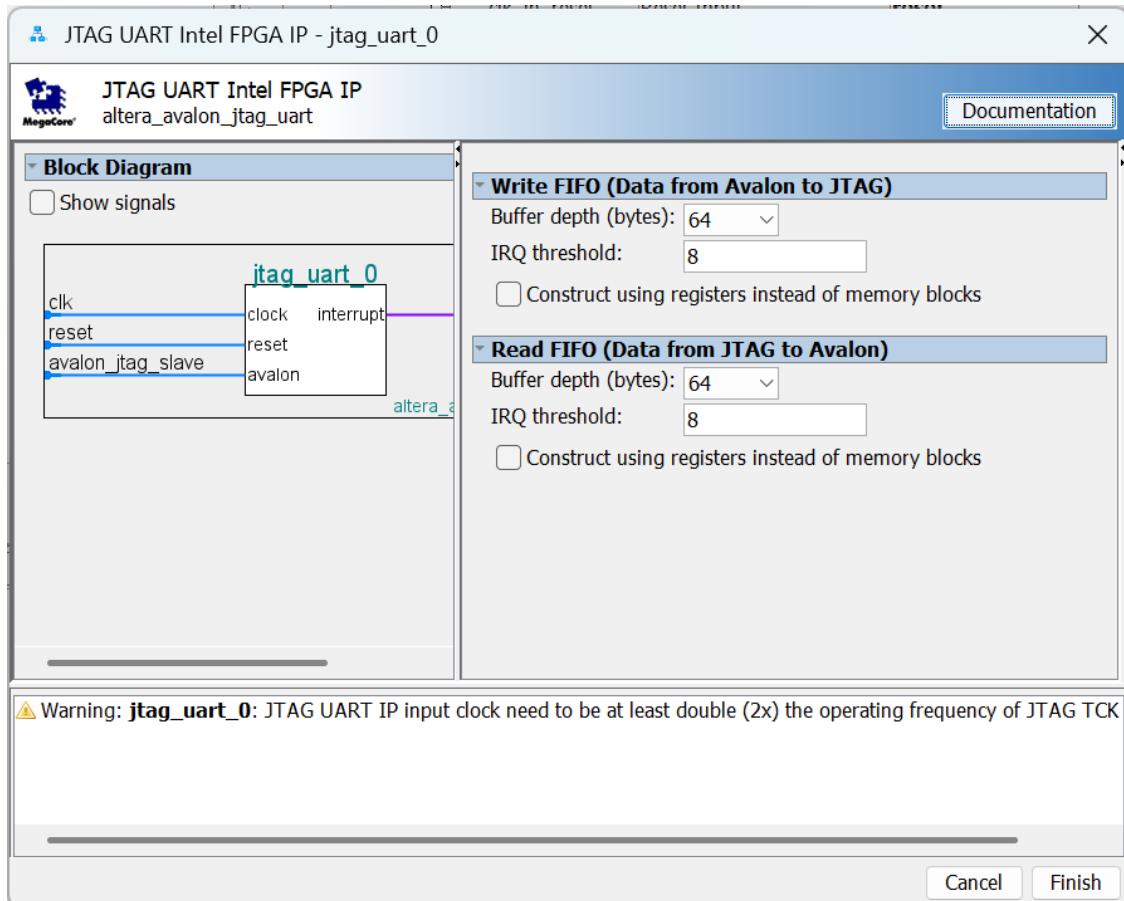
Hình 3.3: Thư mục dự án hiển thị các tệp Verilog DMA tùy chỉnh đã sao chép.



Hình 3.4: Cấu hình IP Nios V/m: Bật "Enable Reset from Debug Module".



Hình 3.5: Cấu hình IP Bộ nhớ Trên Chip: Đặt kích thước thành 128 KB.



Hình 3.6: Cấu hình IP JTAG UART.

Component Editor - DMA_Controller_hw.tcl*

File Templates Beta View

Component Type | Block Symbol | **Files** | Parameters | Signals & Interfaces

>About Files

Synthesis Files

These files describe this component's implementation, and will be created when a Quartus synthesis model is generated.

The parameters and signals found in the top-level module will be used for this component's parameters and signals.

Output Path	Source File	Type	Attributes
CONTROL_SLAVE.v	DMAC/CONTROL_SLA...	Verilog HDL	<i>no attributes</i>
DMAController.v	DMAC/DMAController.v	Verilog HDL	Top-level File
FIFO_IP.qip	DMAC/FIFO_IP.qip	Other	<i>no attributes</i>
FIFO_IP.v	DMAC/FIFO_IP.v	Verilog HDL	<i>no attributes</i>
FIFO IP bb.v	DMAC/FIFO_IP_bb.v	Verilog HDL	<i>no attributes</i>

Add File... Remove File **Analyze Synthesis Files** Create Synthesis File from Signals

Top-level Module: DMAController

Verilog Simulation Files

These files will be produced when a Verilog simulation model is generated.

Output Path	Source File	Type	Attributes
CONTROL_SLAVE.v	DMAC/CONTROL_SLA...	Verilog HDL	<i>no attributes</i>
DMAController.v	DMAC/DMAController.v	Verilog HDL	<i>no attributes</i>
FIFO_IP.qip	DMAC/FIFO_IP.qip	Other	<i>no attributes</i>
FIFO_IP.v	DMAC/FIFO_IP.v	Verilog HDL	<i>no attributes</i>
FIFO IP bb.v	DMAC/FIFO_IP_bb.v	Verilog HDL	<i>no attributes</i>

Add File... Remove File **Copy from Synthesis Files**

VHDL Simulation Files

These files will be produced when a VHDL simulation model is generated.

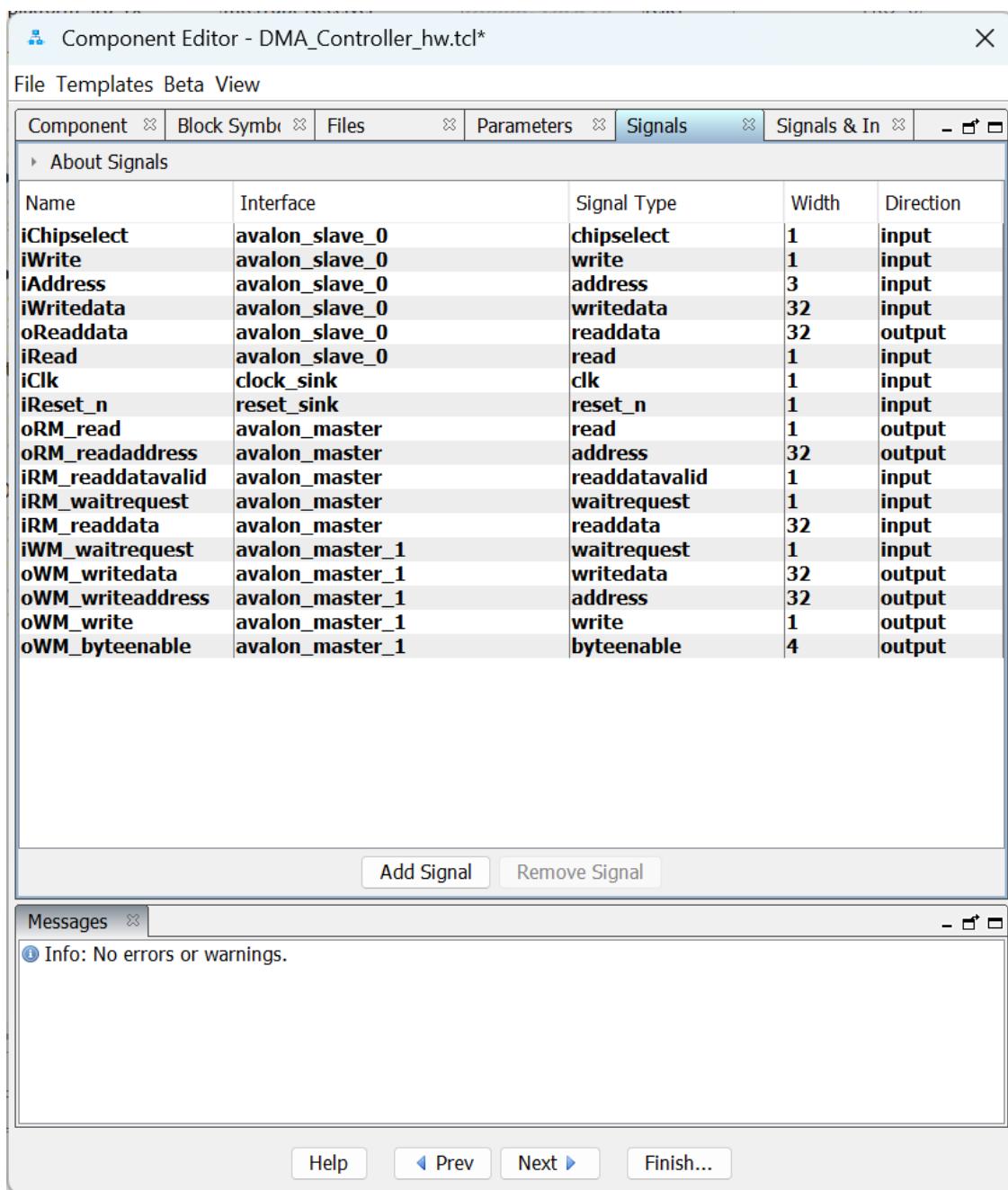
Output Path	Source File	Type	Attributes
(No files)			

Messages

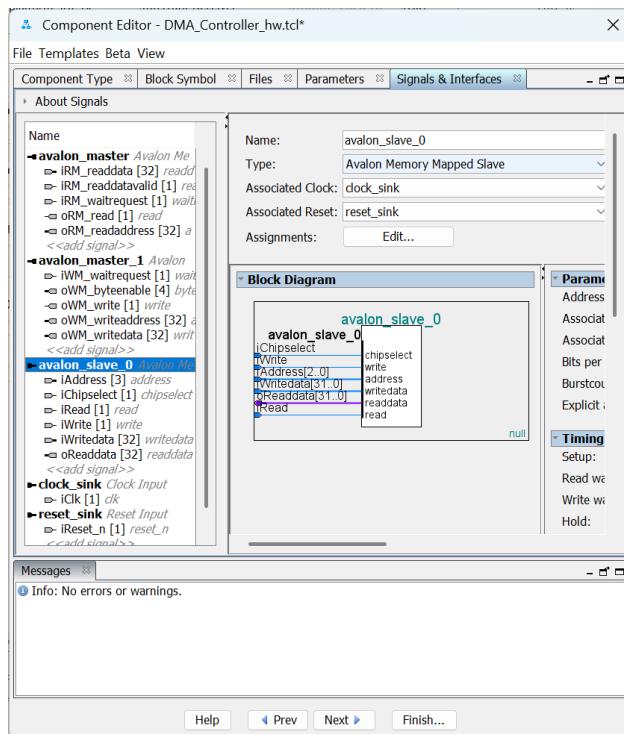
- ⚠ Warning: **avalon_slave_0**: Signal **beginbursttransfer** appears 8 times (only once is allowed)
- ⚠ Warning: **avalon_slave_0**: Signal **readdata** appears 5 times (only once is allowed)
- ⚠ Warning: **avalon_slave_0**: Signal **writebyteenable_n** appears 3 times (only once is allowed)
- ⚠ Warning: **avalon_slave_0**: Signal **writeresponsevalid_n** appears 2 times (only once is allowed)
- ⚠ Warning: **avalon_slave_0**: Slave with **beginbursttransfer** also needs **burstcount** for burst transfers
- ✖ Error: **avalon_slave_0**: Should have **readdatavalid** signal for read burst transfers
- ✖ Error: **avalon_slave_0**: Should have **waitrequest** signal for read burst transfers
- ✖ Error: **avalon_slave_0**: Signal **oWM_byteenable[4]** must be a multiple of the symbol width 8
- ✖ Error: **avalon_slave_0**: Interface must have an associated clock

Help Prev Next Finish...

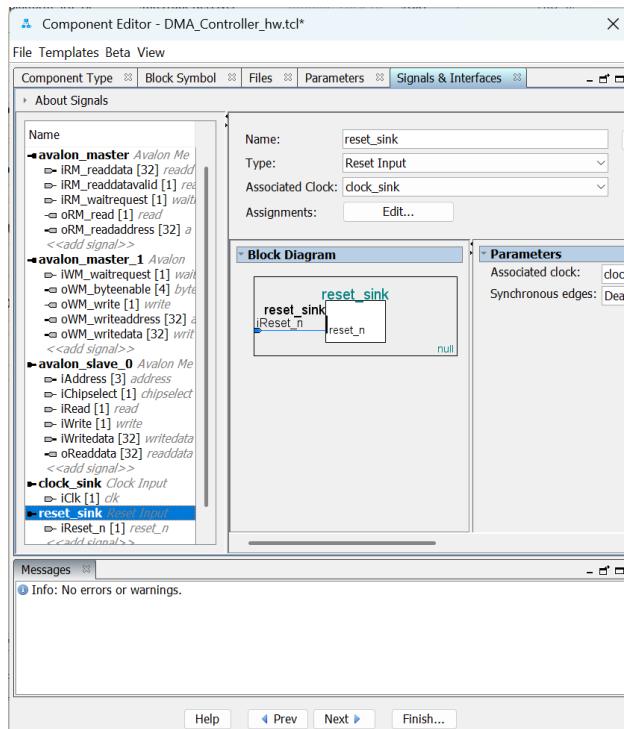
Hình 3.7: Component Editor: Các tệp sau khi Phân tích và Sao chép từ Tổng hợp.



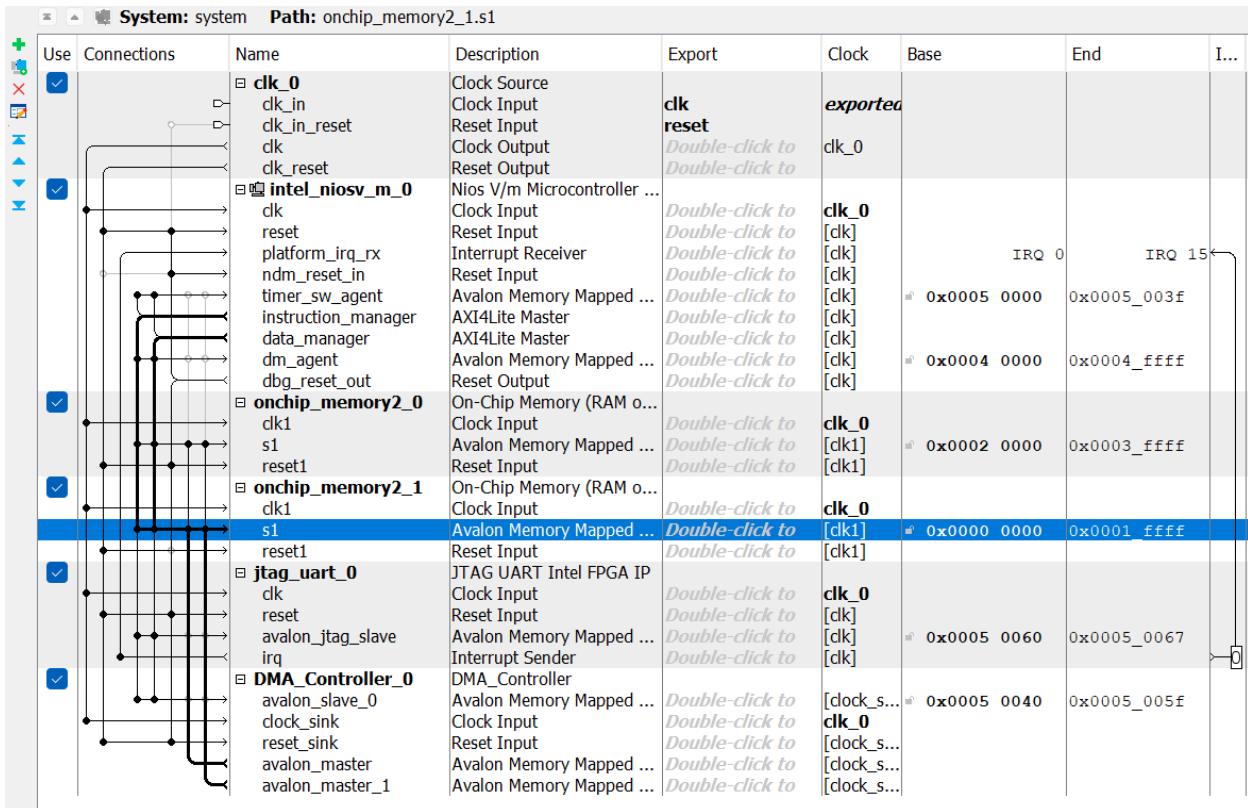
Hình 3.8: Component Editor: Các loại tín hiệu clock và reset đã được sửa.



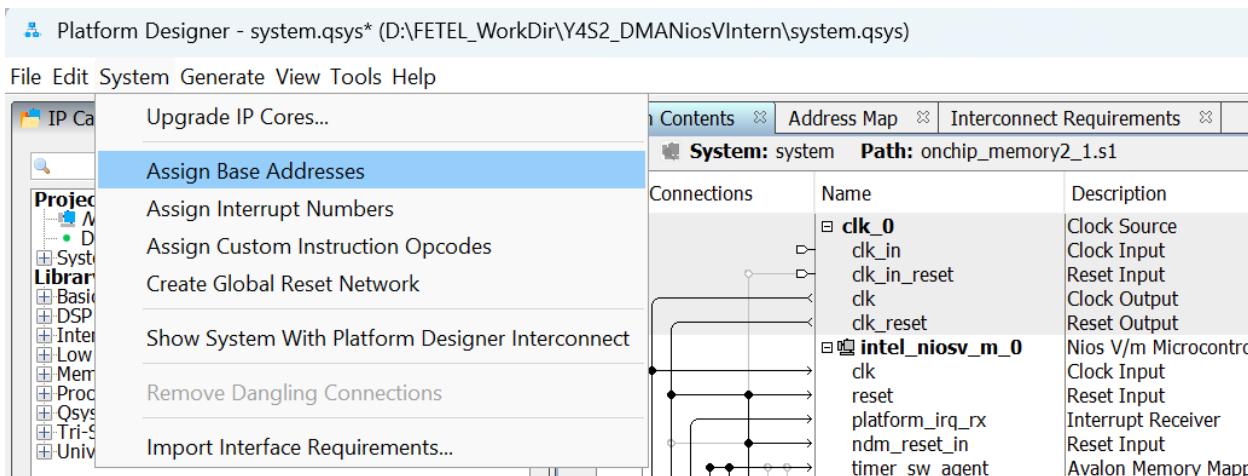
Hình 3.9: Component Editor: Ché độ xem sơ đồ khối giao diện Avalon Slave.



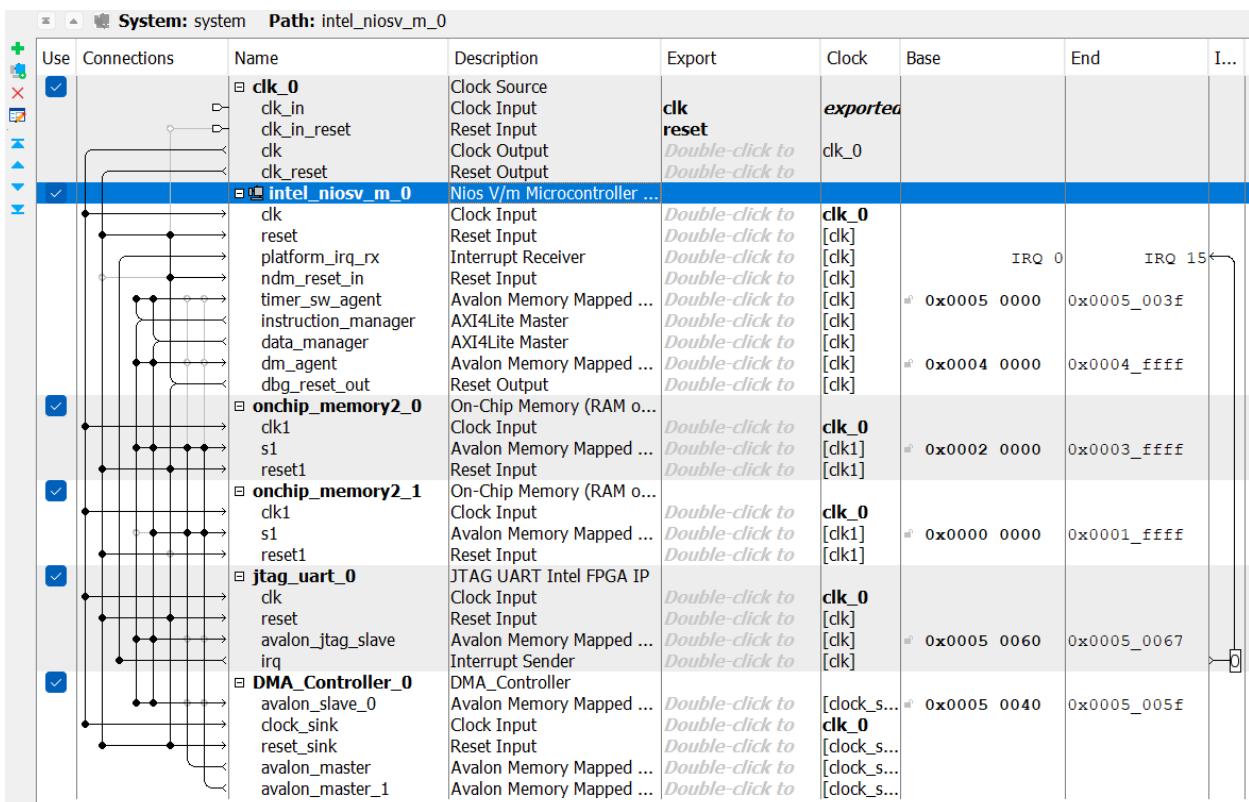
Hình 3.10: Component Editor: Ché độ xem giao diện Reset và Clock sink.



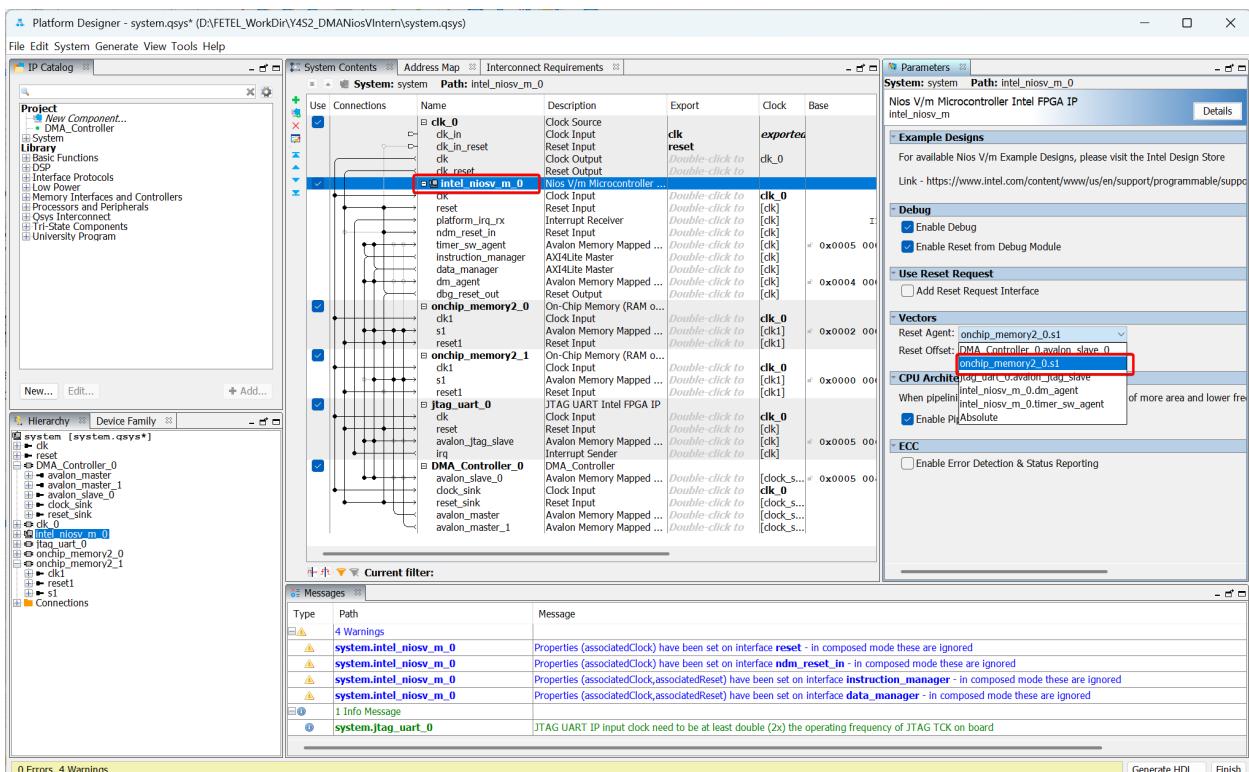
Hình 3.11: Platform Designer: Hệ thống có kết nối giữa instruction_master với s1 (của onchip_memory2_1).



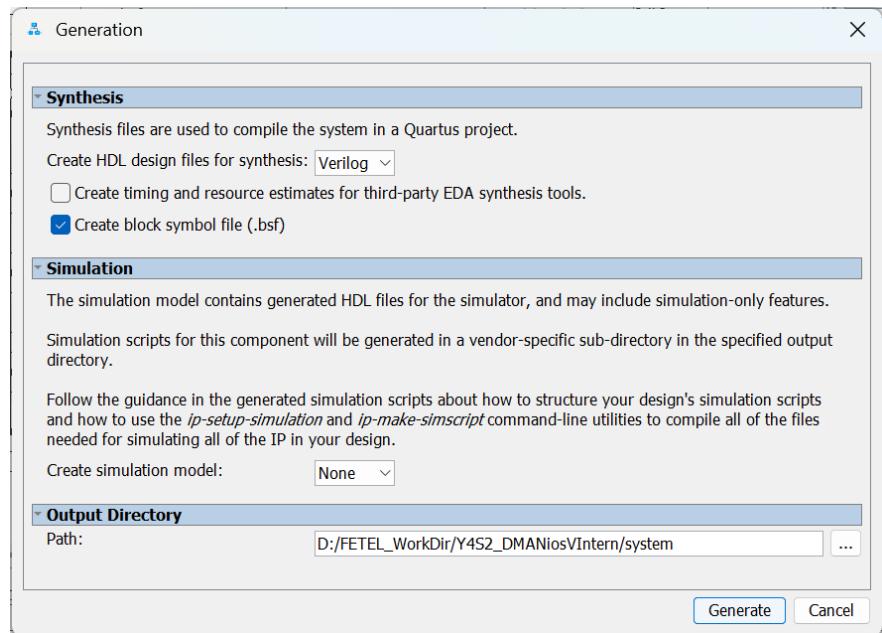
Hình 3.12: Platform Designer: Gán Địa chỉ Cơ sở (Assign Base Addresses).



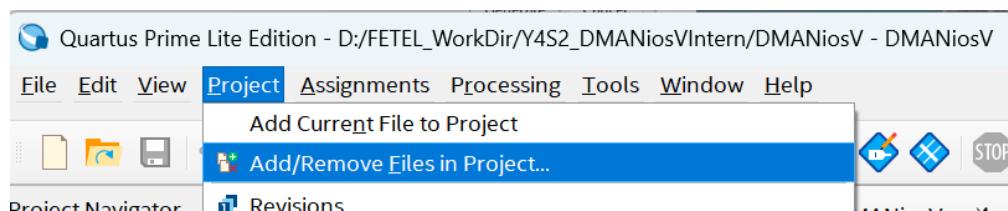
Hình 3.13: Platform Designer: Mô hình hệ thống hoàn chỉnh (sau khi ngắt kết nối instruction_master với s1 trên onchip_memory2_1).



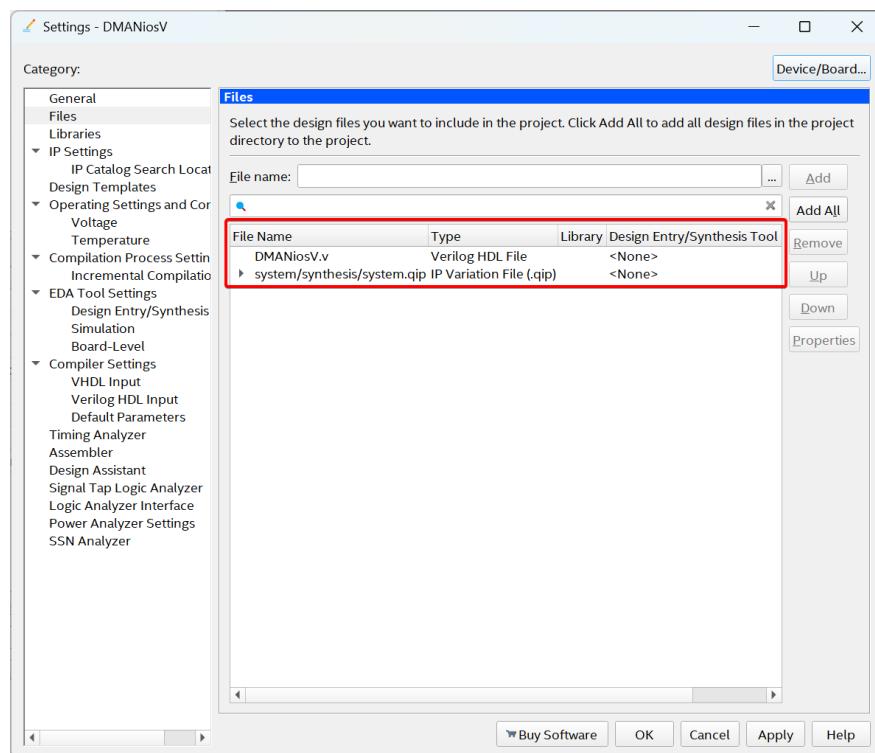
Hình 3.14: Cấu hình IP Nios V/m: Đặt Bộ nhớ Vector Reset (Reset Vector Memory).



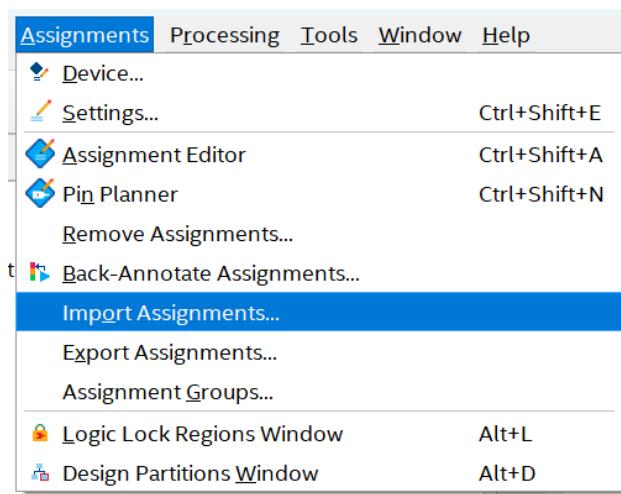
Hình 3.15: Platform Designer: Hộp thoại Tạo HDL (Generate HDL).



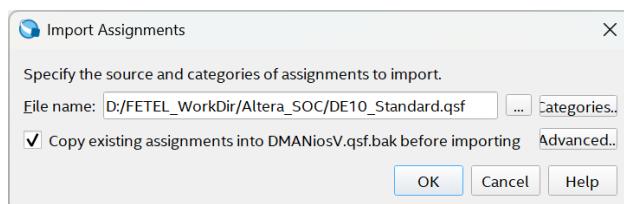
Hình 3.16: Quartus: Menu Project -> Add/Remove Files in Project...



Hình 3.17: Quartus: Cài đặt Dự án - Cửa sổ Tệp hiển thị tệp .qip đã thêm.



Hình 3.18: Quartus: Menu Assignments -> Import Assignments...



Hình 3.19: Quartus: Hộp thoại Nhập Gán chân (Import Assignments) để chọn tệp .qsf.

Tasks		Compilation	Time
	Task		
✓	Compile Design	00:03:31	
✓	▶ Analysis & Synthesis	00:01:02	
✓	▶ Fitter (Place & Route)	00:01:55	
✓	▶ Assembler (Generate programming files)	00:00:12	
✓	▶ Timing Analysis	00:00:22	
	▶ EDA Netlist Writer		
	Edit Settings		
	Program Device (Open Programmer)		

Hình 3.20: Quartus: Compile Design thành công.

Assignment Editor									
at ▾	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag	
1 ✓		in CLOCK_50	Location	PIN_AF14	Yes				
2 ✓		in KEY[0]	Location	PIN_AJ4	Yes				
3 ?		ADC_CONVST	Location	PIN_Y21	Yes				
4 ?		ADC_SCLK	Location	PIN_W24	Yes				
5 ?		ADC_SDI	Location	PIN_W22	Yes				
6 ?		ADC_SDO	Location	PIN_V23	Yes				

Hình 3.21: Quartus: Trình chỉnh sửa Gán chân (Assignment Editor) hiển thị các gán chân đã nhập.

3.2 Phát triển Ứng dụng Phần mềm Nios V

Phần này mô tả việc tạo Gói Hỗ trợ Bo mạch (Board Support Package - BSP) và mã ứng dụng C, sau đó biên dịch và gỡ lỗi (debug) nó bằng IDE Ashling RiscFree.

1. **Chuẩn bị Cấu trúc Thư mục Phần mềm:** Tạo một thư mục `software` trong thư mục dự án chính (`DMANiosVIntern`). Bên trong `software`, tạo hai thư mục con: `app` và `bsp` (Hình 3.22). Đặt mã nguồn ứng dụng C của bạn (ví dụ: `source.c` chứa logic kiểm thử DMA) vào bên trong thư mục `app` (Hình 3.23).

2. Tạo BSP và Tệp CMake:

- Mở Nios V command shell (`niosv-shell.exe`) nằm trong thư mục cài đặt Intel FPGA (ví dụ: `intelFPGA_lite/23.1std/niosv/bin`) (Hình 3.24).
- Điều hướng Nios V Shell đến thư mục dự án (Working Directory, 3.1):

```
1 cd D:\FETEL_WorkDir\Y4S2_DMANiosVIntern
```

Listing 3.1: Điều hướng trong Nios V Shell

- Tạo BSP bằng lệnh `niosv-bsp`. Lệnh này sử dụng tệp thông tin phần cứng (`.sopcinfo`) được tạo bởi Platform Designer để tạo các trình điều khiển (drivers) và các tệp tiêu đề hệ thống (system headers) [11]:

```
1 # -c: create BSP, -t=hal: type HAL, --sopcinfo: path to
      hardware info
2 niosv-bsp -c -t=hal --sopcinfo=system.sopcinfo software/bsp/
      settings.bsp
```

Listing 3.2: Lệnh tạo BSP Nios V

- Tạo các tệp xây dựng CMake cho ứng dụng bằng `niosv-app`. Lệnh này liên kết nguồn ứng dụng (-a) với BSP đã tạo (-b) và chỉ định thư mục nguồn (-s) [11]:

```
1 # -a: app dir, -b: bsp dir, -s: source dir (relative to app
      dir)
2 niosv-app -a=software/app -b=software/bsp -s=software/app
```

Listing 3.3: Lệnh tạo tệp CMake ứng dụng Nios V

(Xem Hình 3.25 cho việc thực thi lệnh và Hình 3.26 cho đầu ra tạo BSP). Đảm bảo không có lỗi xảy ra. Một tệp `CMakeLists.txt` bây giờ sẽ tồn tại trong thư mục `software/app`.

3. Xây dựng Ứng dụng bằng Ashling RiscFree™ IDE [2]:

- **Khởi chạy IDE:** Quan trọng là khởi chạy Ashling RiscFree™ IDE từ `niosv-shell.exe` bằng cách gõ `riscfree`. Điều này đảm bảo các biến môi trường (environment variables) được thiết lập chính xác (Hình 3.27).
- **Chọn Workspace:** Chọn thư mục `software` (Hình 3.28).
- **Import Project:** Trong IDE (Hình 3.29), đi đến File -> New -> C/C++ Project (Hình 3.30). Chọn "C Project". Đặt tên dự án là `app` (trùng với tên thư mục). Chọn "Empty Project" dưới loại dự án "CMake driven" (Hình 3.31). Nhấp Finish. IDE sẽ phát hiện `CMakeLists.txt` và cấu hình dự án.
- **Xây dựng Dự án (Build Project):** Nhấp chuột phải vào dự án `app` trong Trình khám phá Dự án (Project Explorer) (Hình 3.32) và chọn

"Build Project" (hoặc sử dụng Ctrl+B) (Hình 3.33). Kiểm tra cửa sổ Console để xem tiến trình xây dựng và đảm bảo nó hoàn thành mà không có lỗi (Hình 3.34). Thao tác này tạo ra tệp thực thi (app.elf) bên trong một thư mục xây dựng (ví dụ: software/app/build/Debug).

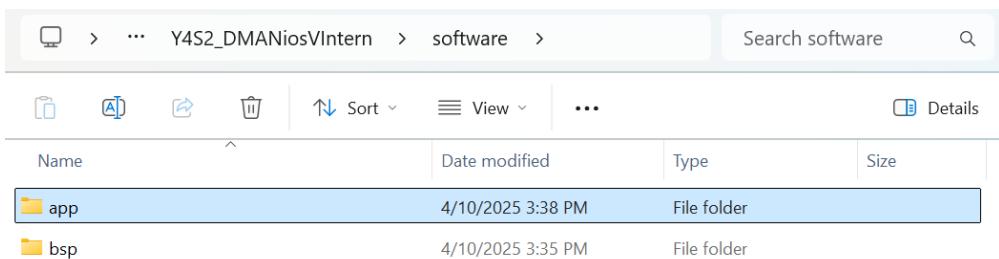
4. Nạp chương trình cho FPGA và Chạy/Gỡ lỗi Phần mềm:

- **Nạp chương trình cho FPGA (Program FPGA):** Sử dụng Quartus Programmer (Tools -> Programmer) để tải thiết kế phần cứng đã biên dịch (tệp .sof từ mục 3.1) lên bo mạch DE10-Standard thông qua kết nối USB-Blaster. Đảm bảo phần cứng đã được kết nối và bắt đầu quá trình nạp (Hình 3.35).
- **Cấu hình Trình gỡ lỗi (Configure Debugger):** Trong Ashling IDE, nhấp chuột phải vào dự án app -> Run As -> Ashling RISC-V Hardware Debugging... (Hình 3.36). Chọn app.elf (Hình 3.37).
 - Chuyển đến tab "Debugger". Chọn Đầu dò Gỡ lỗi (Debug Probe) chính xác (ví dụ: "DE-SoC [USB-1]"). Đảm bảo "Lựa chọn Thiết bị/TAP (Device/TAP selection)" khớp với lõi Nios V được xác định trên chuỗi JTAG (sử dụng "Auto-detect Scan Chain" nếu không chắc chắn). Loại Vận chuyển (Transport type) phải là JTAG (Hình 3.38). Áp dụng (Apply) và nhấp Debug.
- **Chạy và Quan sát (Run and Observe):** IDE sẽ kết nối với lõi Nios V, tải xuống tệp .elf, và dừng tại đầu hàm main() (Hình 3.39). Mở một juart-terminal trong Nios V shell để xem đầu ra (output) của chương trình:

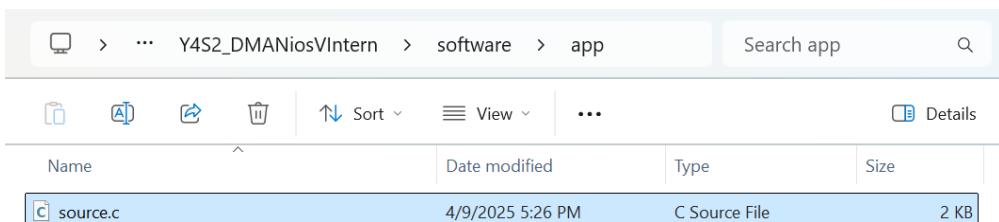
```
1 juart-terminal
```

Listing 3.4: Khởi chạy JTAG UART Terminal

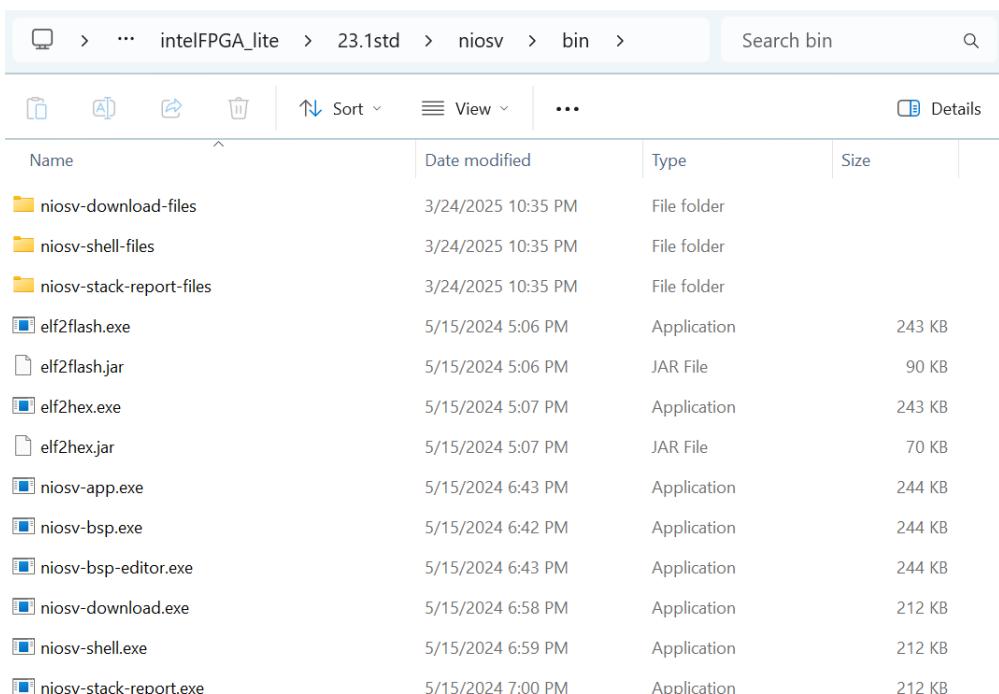
- Quan sát đầu ra trong cửa sổ juart-terminal (Hình 3.40, 3.41, 3.42). Đầu ra ví dụ cho thấy các thanh ghi DMA đang được cấu hình, trạng thái đang được thăm dò (polling), bộ đệm bộ nhớ đang được xóa/xả (cleared/flushed), và cuối cùng, xác minh rằng việc truyền dữ liệu qua DMA đã diễn ra chính xác.



Hình 3.22: Cấu trúc thư mục Software gồm các thư mục con app và bsp.



Hình 3.23: Thư mục software/app chứa tệp mã nguồn source.c.



Hình 3.24: Vị trí của niosv-shell.exe.

```

D:\FETEL_WorkSoft\intelFPGA < + <
TBBmalloc: skip allocation functions replacement in ucrtbase.dll: unknown prologue for function _msize
Entering Nios V shell
Microsoft Windows [Version 10.0.26100.3775]
(c) Microsoft Corporation. All rights reserved.

[niosv-shell] D:\FETEL_WorkSoft\intelFPGA_lite\23.1std\niosv\bin> |

```

Hình 3.25: Cửa sổ Nios V Shell.

```

[niosv-shell] D:\FETEL_WorkDir\Y4S2_DMANiosVIntern> niosv-bsp -c -t=hal --sopcinfo=system.sopcinfo software/bsp/settings.bsp
2025.04.10.15:35:36 Info: Searching for BSP components with category: os_software_element
2025.04.10.15:35:37 Info: Creating BSP settings.
2025.04.10.15:35:37 Info: Initializing SOPC project local software IP
2025.04.10.15:35:37 Info: Finished initializing SOPC project local software IP. Total time taken = 2 seconds
2025.04.10.15:35:38 Info: Searching for BSP components with category: driver_element
2025.04.10.15:35:38 Info: Searching for BSP components with category: software_package_element
2025.04.10.15:35:38 Info: Loading drivers from ensemble report.
2025.04.10.15:35:38 Info: Finished loading drivers from ensemble report.
2025.04.10.15:35:38 Info: Evaluating default script "d:\fetel_worksoft\intelfpga_lite\23.1std\quartus..\niosv\scripts\bsp-defaults.tcl".
2025.04.10.15:35:38 Info: Tcl message: "STDIO character device is jtag_uart_0"
2025.04.10.15:35:38 Info: Tcl message: "System timer device is intel_miosv_m_0"
2025.04.10.15:35:38 Info: Tcl message: "Default instruction linker sections mapped to onchip_memory2_0"
2025.04.10.15:35:38 Info: Tcl message: "Default data linker sections mapped to onchip_memory2_1"
2025.04.10.15:35:38 Info: Tcl message: "No bootloader located at the reset address."
2025.04.10.15:35:38 Info: Tcl message: "Application ELF allowed to contain code at the reset address."
2025.04.10.15:35:38 Info: Tcl message: "The alt_load() facility is enabled."
2025.04.10.15:35:38 Info: Tcl message: "The .rodata section is copied into RAM by alt_load()."
2025.04.10.15:35:38 Info: Tcl message: "The .rwdtata section is copied into RAM by alt_load()."
2025.04.10.15:35:38 Info: Saving BSP settings file.
2025.04.10.15:35:38 Info: Default memory regions will not be persisted in BSP Settings File.
2025.04.10.15:35:38 Info: Generated file "D:\FETEL_WorkDir\Y4S2_DMANiosVIntern\software\bsp\settings.bsp"
2025.04.10.15:35:38 Info: Generating BSP files in "D:\FETEL_WorkDir\Y4S2_DMANiosVIntern\software\bsp"
2025.04.10.15:35:38 Info: Default memory regions will not be persisted in BSP Settings File.
2025.04.10.15:35:38 Info: Generated file "D:\FETEL_WorkDir\Y4S2_DMANiosVIntern\software\bsp\settings.bsp"
2025.04.10.15:35:39 Info: Finished generating BSP files. Total time taken = 2 seconds

```

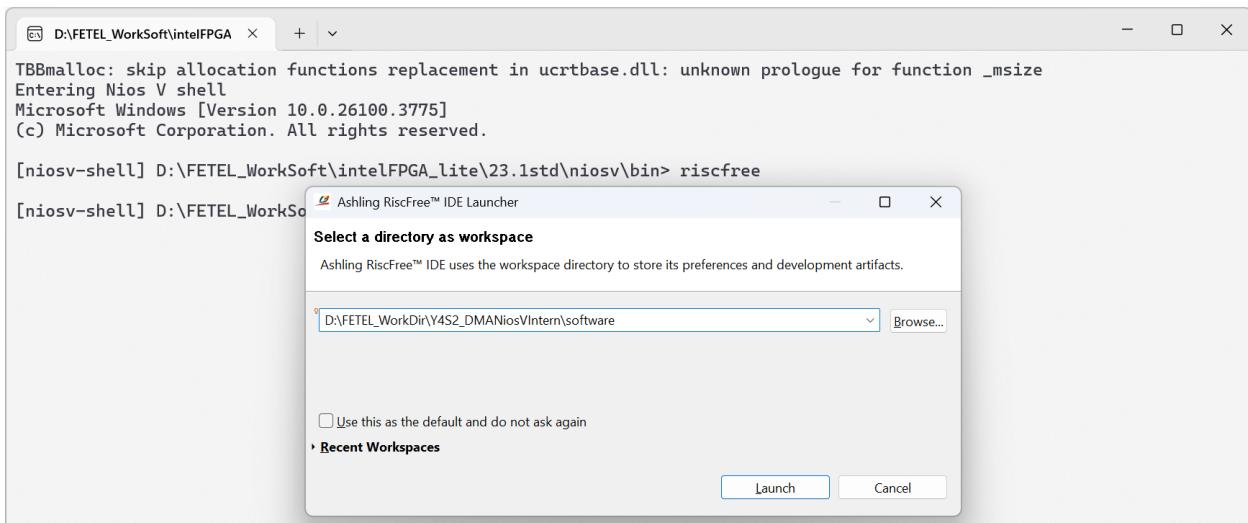
Hình 3.26: Đầu ra từ việc thực thi lệnh niosv-bsp.

```

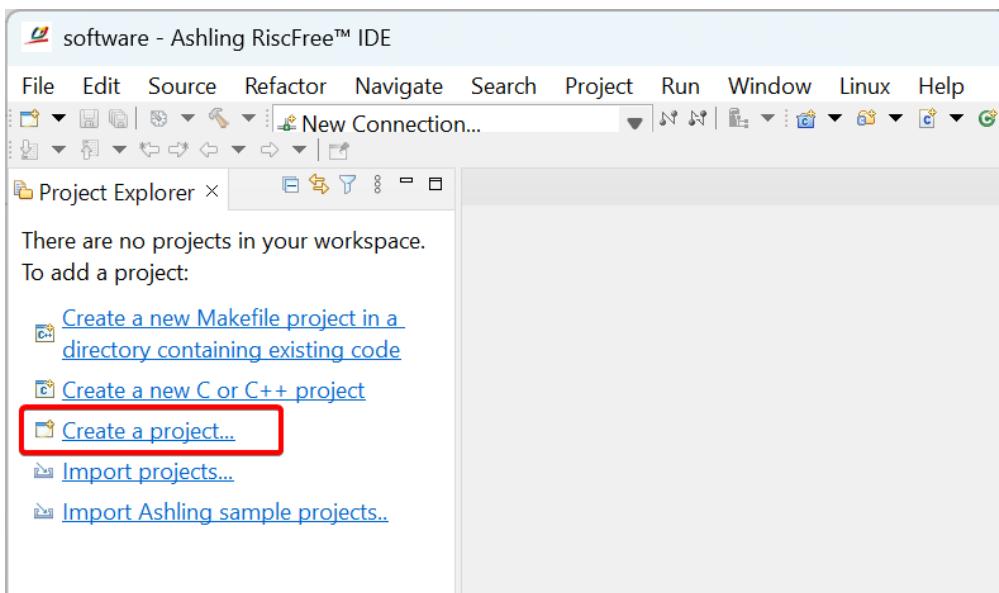
[niosv-shell] D:\FETEL_WorkDir\Y4S2_DMANiosVIntern> niosv-app -a=software/app -b=software/bsp -s=software/app
2025.04.10.15:48:16 Info: Source file "source.c" added from directory "software\app".
2025.04.10.15:48:16 Info: "software\app\CMakeLists.txt" was generated.

```

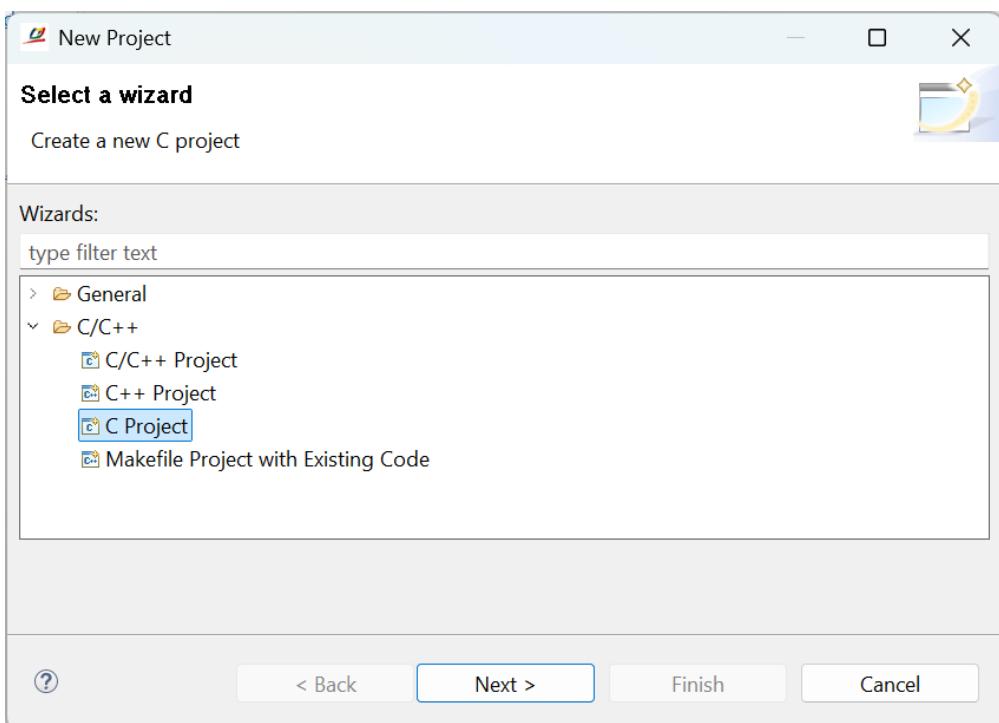
Hình 3.27: Đầu ra từ việc thực thi lệnh niosv-app.



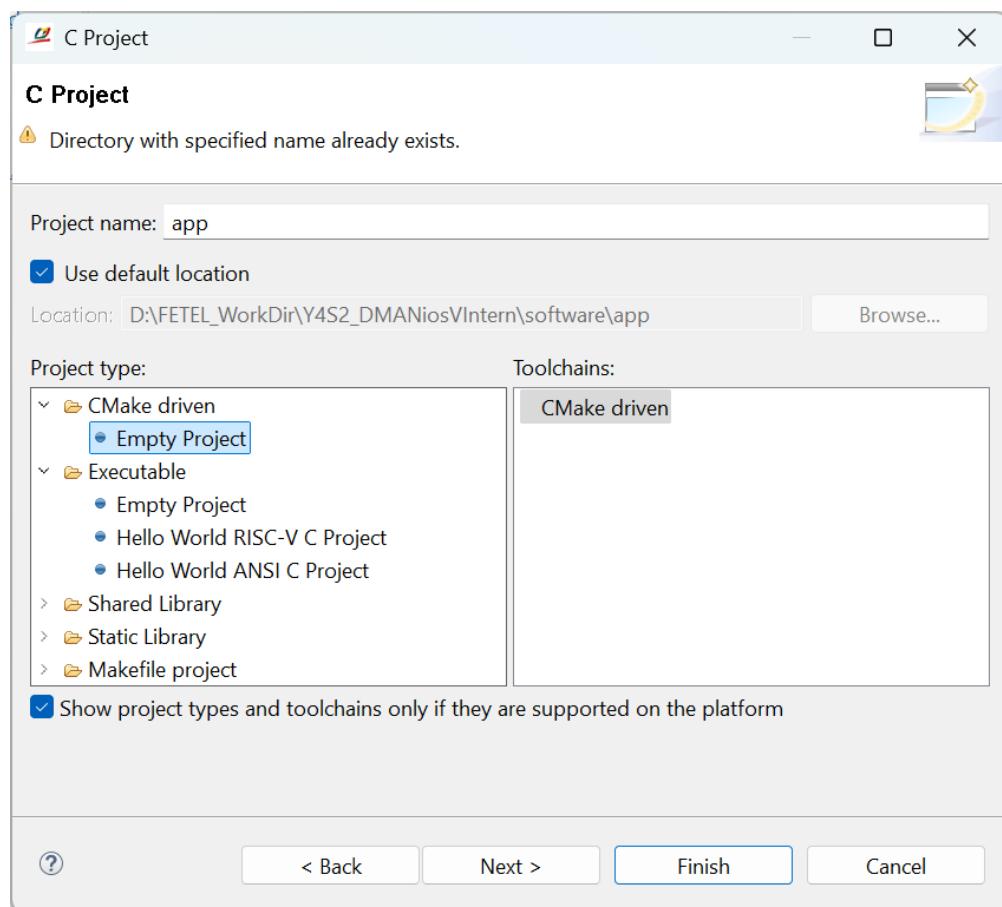
Hình 3.28: Khởi chạy Ashling RiscFree™ IDE từ Nios V Shell, chọn đường dẫn Workspace.



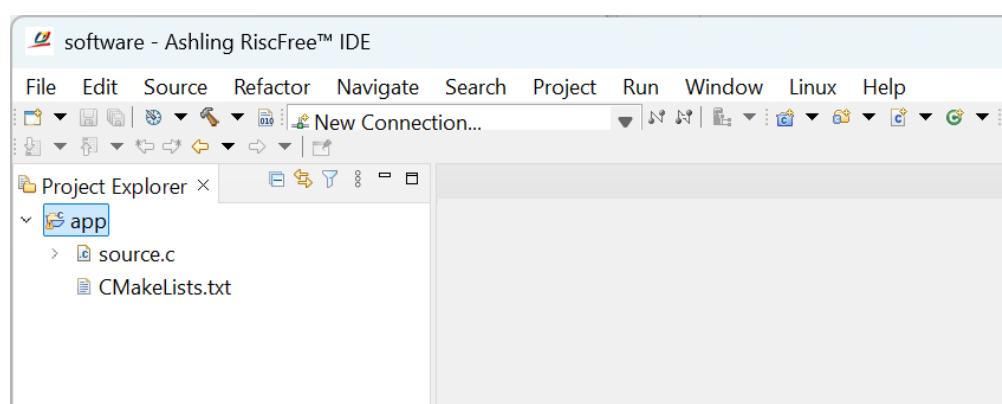
Hình 3.29: Màn hình chào mừng Ashling RiscFree™ IDE: Chọn "Create a project...".



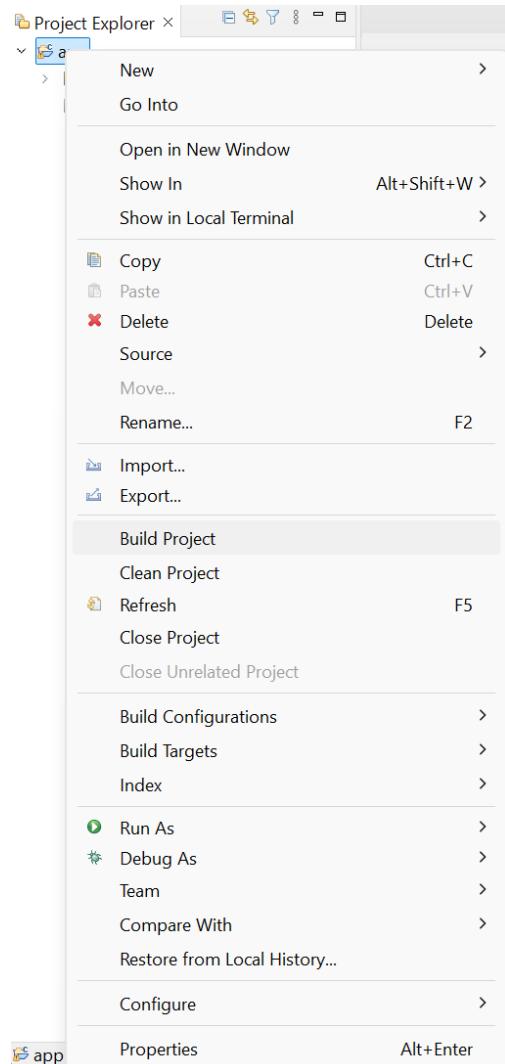
Hình 3.30: Ashling IDE: Chọn C/C++ → C Project.



Hình 3.31: Ashling IDE: Chọn CMake driven → Empty Project.



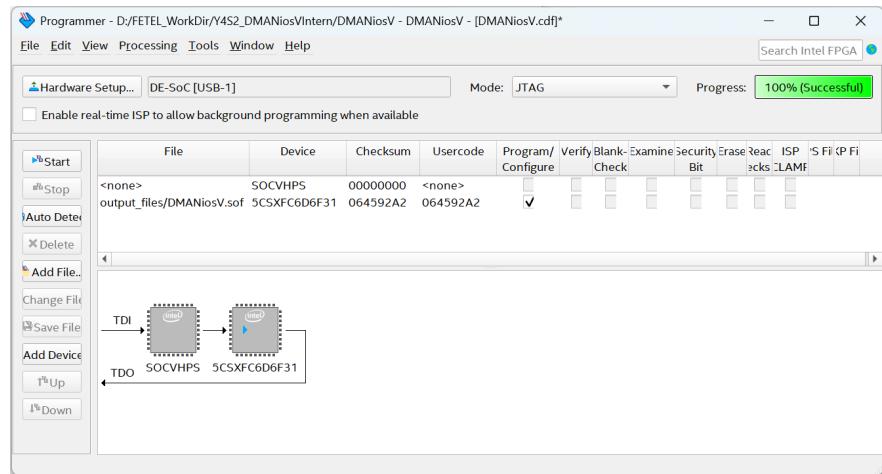
Hình 3.32: Ashling IDE: Project Explorer hiển thị dự án 'app'.



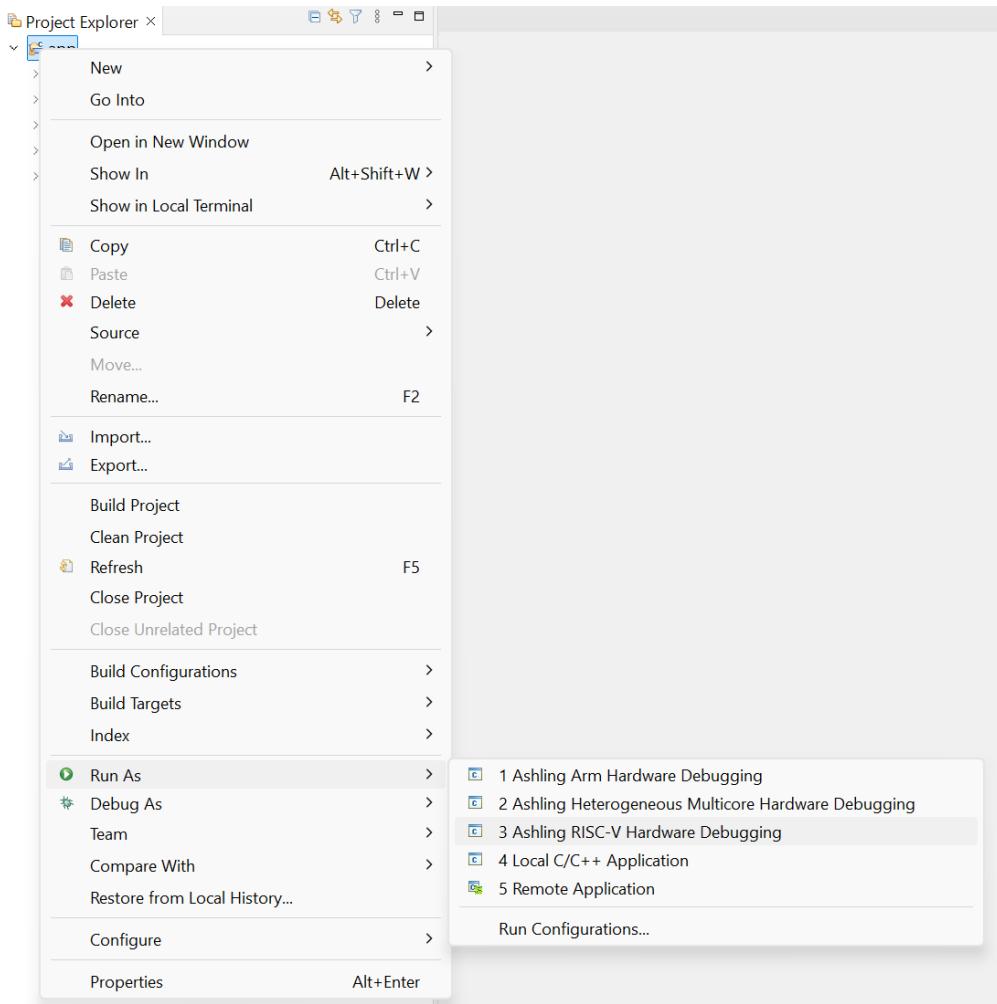
Hình 3.33: Ashling IDE: nhấp chuột phải 'app' → Build Project.

```
Problems Tasks Console × Properties
CMake Console [app]
-- Detecting C compile features - done
-- The CXX compiler identification is GNU 12.1.0
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: D:/FETEL_WorkSoft/intelFPGA_lite/23.1std/riscfree/toolchain/riscv32-unknown-elf-gcc
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (2.4s)
-- Generating done (0.1s)
-- Build files have been written to: D:/FETEL_WorkDir/Y4S2_DMAniosVIntern/software/app/build/Debug
15:56:22 Buildscript generation finished (took 2644 ms)
```

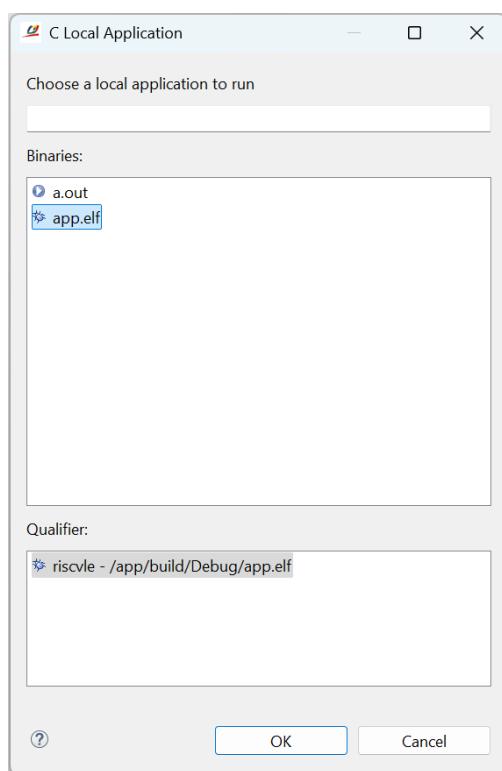
Hình 3.34: Ashling IDE: Console thông báo Build thành công.



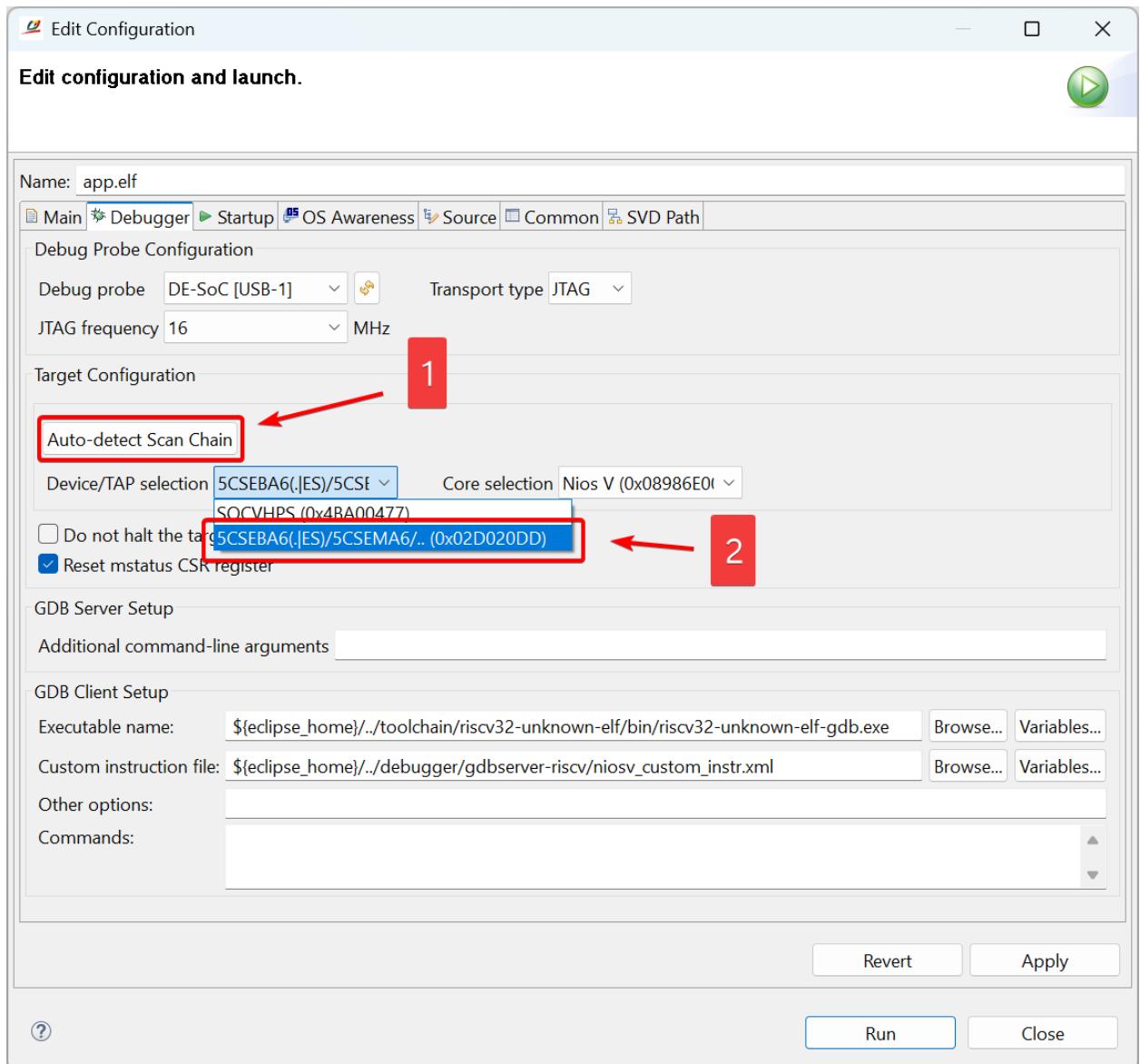
Hình 3.35: Quartus Programmer: Nạp tệp .sof xuống board thành công.



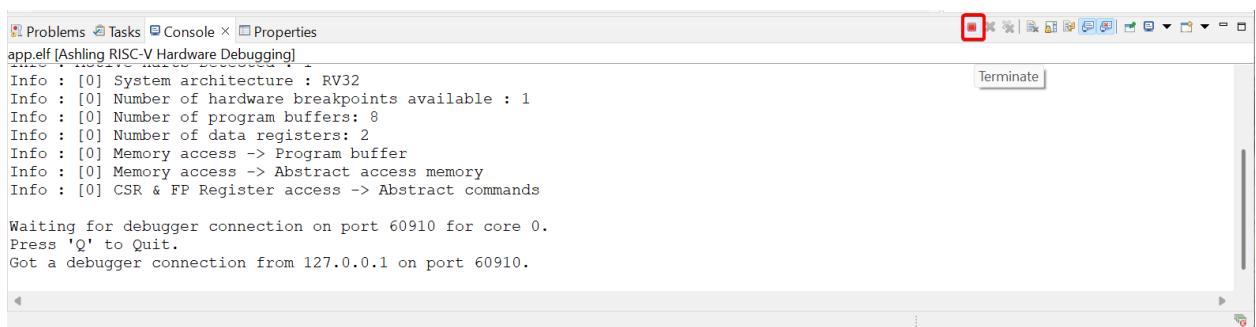
Hình 3.36: Ashling IDE: Nạp Firmware xuống lõi Nios V (Run As → Ashling RISC-V Hardware Debugging).



Hình 3.37: Ashling IDE: Câu hình Gõ lỗi - Chỉ định ứng dụng C/C++ (.elf).



Hình 3.38: Ashling IDE: Cấu hình Gõ lỗi - Cài đặt tab Debugger.



Hình 3.39: Ashling IDE: Debug Console cho biết đang chờ kết nối (Nhấn nút đỏ để ngừng chạy).

```

[niosv-shell] D:\FETEL_WorkSoft\intelFPGA_lite\23.1std\niosv\bin> juart-terminal
juart-terminal: connected to hardware target using JTAG UART on cable
juart-terminal: "DE-SoC [USB-1]", device 2, instance 0
juart-terminal: (Use the IDE stop button or Ctrl-C to terminate)

=====
= 21207001 - Bui Thanh Dat - HCMUS - FETEL - CESLAB =
= Nios V DMA Example =
=====

Resetting DMA Controller...
    Clearing GO bit (Reg 4)...
    Clearing Address/Length Regs (0, 1, 2)...
    Waiting for BUSY bit to clear (Reg 5)...
DMA Reset complete. Final Status: 0x0

System Base Addresses:
    ONCHIP_MEMORY2_0_BASE: 0x00020000
    ONCHIP_MEMORY2_1_BASE: 0x00000000
    DMA_CONTROLLER_0_BASE: 0x00050040
Actual pdata0 address: 0x1000
Actual pdata1 address: 0x200

Starting DMA transfer...
Source Address: 0x00001000 (pdata0)
Destination Address: 0x00000200 (pdata1)
Length: 128 bytes (32 words)

Destination Buffer Before Clearing (potentially cached):
index 0: 0x74656420
index 1: 0x65746365
index 2: 0x00002e64
index 3: 0x6f727245
index 4: 0x54203a72
index 5: 0x6f656d69
index 6: 0x77207475
index 7: 0x69746961

index 8: 0x6620676e
index 9: 0x4420726f
index 10: 0x4420414d
index 11: 0x20454e4f
index 12: 0x2e746962
index 13: 0x6e694620
index 14: 0x53206c61
index 15: 0x75746174

index 16: 0x30203a73
index 17: 0x786c2578
index 18: 0x0000000a
index 19: 0x69726556
index 20: 0x6e697966
index 21: 0x6c252067
index 22: 0x6f772075
index 23: 0x2e736472

index 24: 0x000a2e2e
index 25: 0x6d73694d
index 26: 0x68637461
index 27: 0x20746120
index 28: 0x65646e69
index 29: 0x64252078
index 30: 0x7865203a
index 31: 0x74636570

Resetting destination buffer via CPU...
Destination buffer cleared and cache flushed.

Destination Buffer After Reset & Flush (read back):
index 0: 0x00000000

```

Hình 3.40: Đầu ra JTAG UART Terminal: Thông báo ban đầu và reset DMA.

```
D:\FTEL_WorkSoft\intelFPGA X + ▾ - ▾ X

Resetting destination buffer via CPU...
Destination buffer cleared and cache flushed.

Destination Buffer After Reset & Flush (read back):
index 0: 0x00000000
index 1: 0x00000000
index 2: 0x00000000
index 3: 0x00000000
index 4: 0x00000000
index 5: 0x00000000
index 6: 0x00000000
index 7: 0x00000000

index 8: 0x00000000
index 9: 0x00000000
index 10: 0x00000000
index 11: 0x00000000
index 12: 0x00000000
index 13: 0x00000000
index 14: 0x00000000
index 15: 0x00000000

index 16: 0x00000000
index 17: 0x00000000
index 18: 0x00000000
index 19: 0x00000000
index 20: 0x00000000
index 21: 0x00000000
index 22: 0x00000000
index 23: 0x00000000

index 24: 0x00000000
index 25: 0x00000000
index 26: 0x00000000
index 27: 0x00000000
index 28: 0x00000000
index 29: 0x00000000
index 30: 0x00000000
index 31: 0x00000000

Configuring DMA Regs:
Reg 0 (Read Addr) : 0x00001000
Reg 1 (Write Addr): 0x00000200
Reg 2 (Length)   : 128 bytes
Reg 4 (Control)  : 0x00000001
DMA GO bit set.
Polling DMA Status (Reg 5)...
DMA DONE bit

Flushing D-Cache for destination buffer...
D-Cache flushed.

Verifying DMA transfer...
index 0 = 2
index 1 = 3
index 2 = 4
index 3 = 5
index 4 = 6
index 5 = 7
index 6 = 8
index 7 = 9
index 8 = 10
index 9 = 11
index 10 = 12
index 11 = 13
index 12 = 14
index 13 = 15
index 14 = 16
index 15 = 17
index 16 = 18
index 17 = 19
```

Hình 3.41: Đầu ra JTAG UART Terminal: Trạng thái bộ đệm và cấu hình DMA.

```
D:\FETEL_WorkSoft\intelFPGA < + > - X

index 15: 0x00000000
index 16: 0x00000000
index 17: 0x00000000
index 18: 0x00000000
index 19: 0x00000000
index 20: 0x00000000
index 21: 0x00000000
index 22: 0x00000000
index 23: 0x00000000

index 24: 0x00000000
index 25: 0x00000000
index 26: 0x00000000
index 27: 0x00000000
index 28: 0x00000000
index 29: 0x00000000
index 30: 0x00000000
index 31: 0x00000000

Configuring DMA Regs:
Reg 0 (Read Addr) : 0x00001000
Reg 1 (Write Addr): 0x00000200
Reg 2 (Length)    : 128 bytes
Reg 4 (Control)   : 0x00000001
DMA GO bit set.
Polling DMA Status (Reg 5)...
DMA DONE bit

Flushing D-Cache for destination buffer...
D-Cache flushed.

Verifying DMA transfer...
index 0 = 2
index 1 = 3
index 2 = 4
index 3 = 5
index 4 = 6
index 5 = 7
index 6 = 8
index 7 = 9
index 8 = 10
index 9 = 11
index 10 = 12
index 11 = 13
index 12 = 14
index 13 = 15
index 14 = 16
index 15 = 17
index 16 = 18
index 17 = 19
index 18 = 20
index 19 = 21
index 20 = 22
index 21 = 23
index 22 = 24
index 23 = 25
index 24 = 26
index 25 = 27
index 26 = 28
index 27 = 29
index 28 = 30
index 29 = 31
index 30 = 32
index 31 = 33
DMA Transfer Verification Successful!
```

Hình 3.42: Đầu ra JTAG UART Terminal: Xác minh truyền DMA thành công.

CHƯƠNG 4: Mô phỏng và kiểm thử hệ thống Nios V và DMA

Mô phỏng (Simulation) là một bước quan trọng trong quy trình thiết kế FPGA, cho phép xác minh chức năng của thiết kế phần cứng và sự tương tác giữa các thành phần trước khi triển khai lên phần cứng vật lý. Việc này giúp phát hiện lỗi sớm, tiết kiệm thời gian và công sức gỡ lỗi trên bo mạch thực tế. Chương này trình bày quy trình mô phỏng, công cụ sử dụng, và kết quả kiểm thử chức năng của bộ điều khiển DMA tùy chỉnh được tích hợp trong hệ thống SoC Nios V/m.

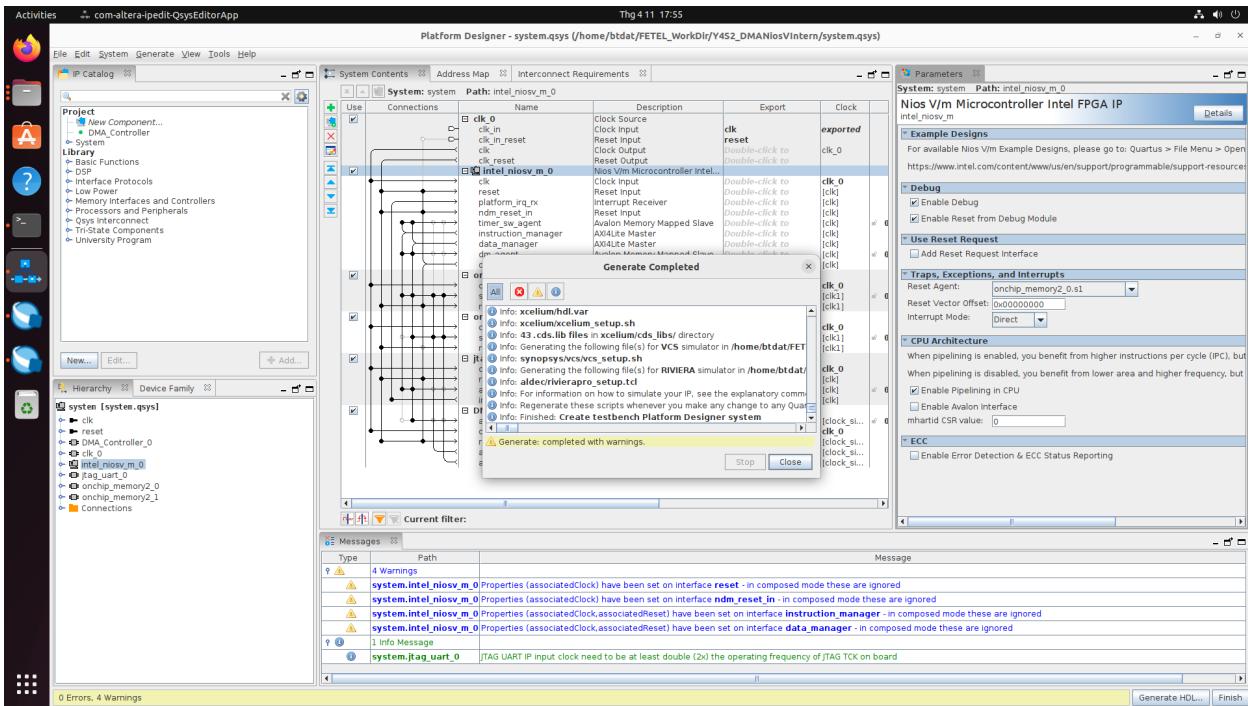
4.1 Công cụ Mô phỏng và Yêu cầu Môi trường

Công cụ chính được sử dụng để mô phỏng hệ thống trong dự án này là **Questa Advanced Simulator** (trước đây là ModelSim-Intel FPGA Edition), một trình mô phỏng HDL mạnh mẽ được cung cấp bởi Siemens.

Khác với Nios II, hệ thống Nios V không hỗ trợ Generate Testbench System với các phiên bản Quartus Prime Lite và Standard trên hệ điều hành Windows (và Intel cũng đã thông báo sẽ không hỗ trợ cho việc này trên hệ điều hành Windows [7]). Do đó, có hai phương pháp chính để tạo testbench mô phỏng cho hệ thống Nios V:

- Sử dụng Quartus Prime Pro:** Phiên bản Pro hỗ trợ đầy đủ việc tạo testbench cho Nios V trên Windows, tuy nhiên phiên bản này yêu cầu giấy phép (license) khác, và tốn dung lượng ổ cứng từ 40-145 GB.
- Generate Testbench System trên Linux:** Thực hiện cài đặt Quartus Prime (Lite hoặc Standard) trên một môi trường Linux. Sử dụng Platform Designer trên Linux để "Generate Testbench System" cho tệp `system.qsys`. Sau khi testbench được tạo thành công trên Linux (như minh họa trong Hình 4.1), toàn bộ thư mục project (bao gồm cả thư mục testbench đã tạo) có thể được sao chép sang môi trường Windows để thực hiện các bước mô phỏng tiếp theo bằng QuestaSim trên Windows. Quy trình chi tiết cho việc này được mô tả trong Phụ lục B.6.

Ngoài ra, để mô phỏng hoạt động của bộ xử lý Nios V thực thi mã phần mềm kiểm thử DMA, tệp chương trình thực thi (`.elf`) cần được biên dịch (như mô tả ở Chương 3) và sau đó chuyển đổi thành định dạng bộ nhớ (`.hex`) [11]. Các tệp `.hex` này sẽ được testbench tự động nạp vào mô hình bộ nhớ RAM on-chip trong quá trình



Hình 4.1: Generate Testbench System trong Platform Designer trên Ubuntu 22.04.5 thành công.

khởi tạo mô phỏng, cho phép bộ xử lý Nios V mô phỏng thực thi chương trình và tương tác với bộ điều khiển DMA [9].

4.2 Kết quả mô phỏng và kiểm thử chức năng

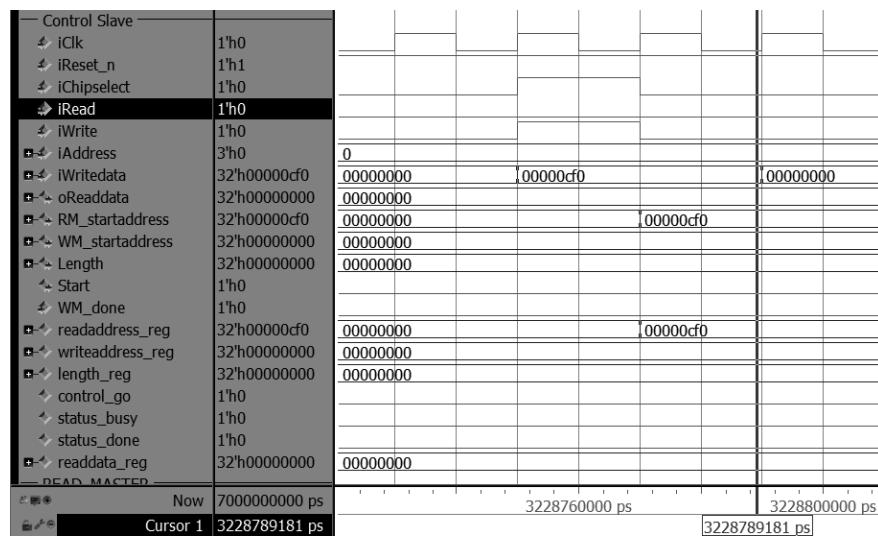
Mục tiêu của mô phỏng là xác minh rằng bộ điều khiển DMA tùy chỉnh hoạt động đúng theo thiết kế: nhận cấu hình từ Nios V, thực hiện truyền dữ liệu giữa hai vùng nhớ on-chip một cách độc lập, và báo hiệu hoàn thành chính xác. Mô phỏng được thực hiện bằng Questa Sim [9], sử dụng testbench được tạo ra theo quy trình ở Mục B.6 và Phụ lục B.6.

Quá trình mô phỏng bao gồm các bước chính sau, được minh họa bằng các dạng sóng tín hiệu Avalon-MM Slave và Master của bộ điều khiển DMA:

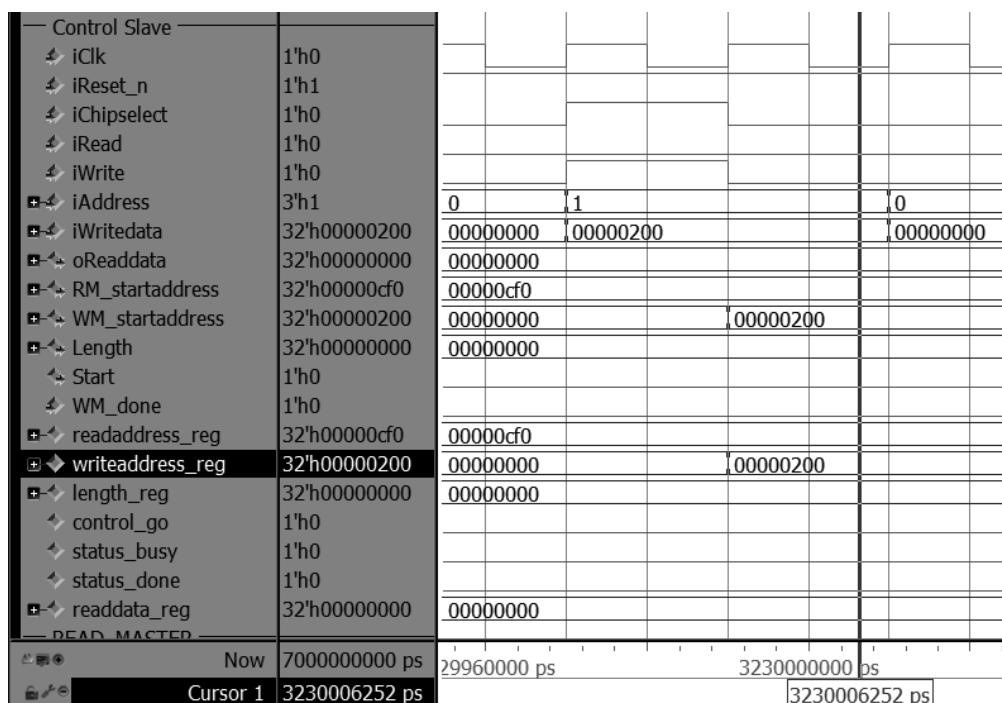
- Cấu hình DMA qua giao diện Slave:** Bộ xử lý Nios V (được mô phỏng thực thi mã trong tệp .hex) ghi các thông số cần thiết vào các thanh ghi của module CONTROL_SLAVE thông qua giao diện Avalon-MM Slave.

- Ghi địa chỉ bắt đầu đọc (Source Address) vào thanh ghi tại offset 0 (Hình 4.2).

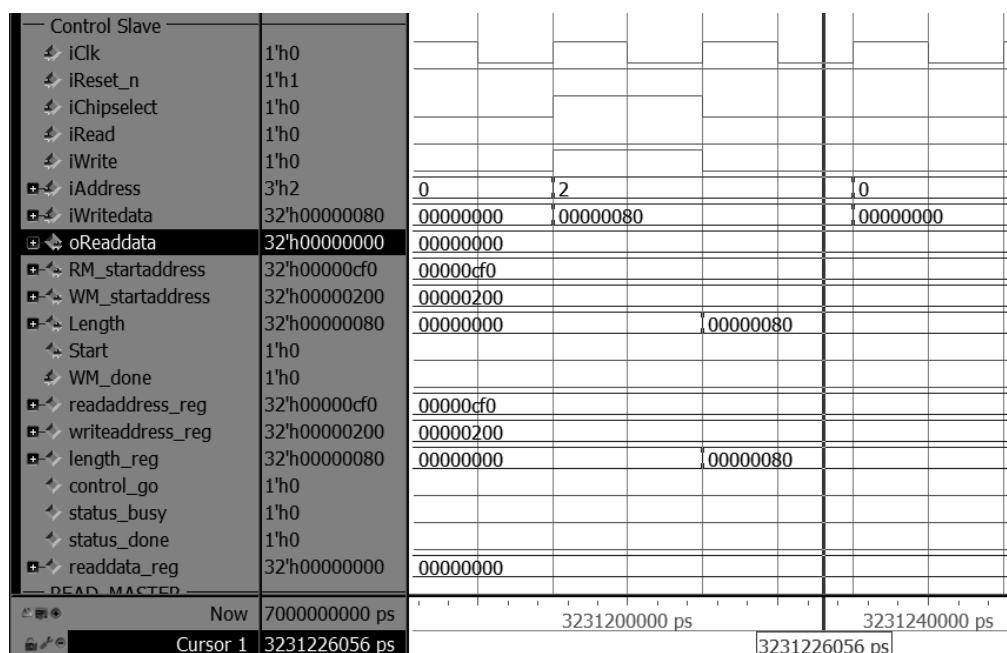
- Ghi địa chỉ bắt đầu ghi (Destination Address) vào thanh ghi tại offset 1 (Hình 4.3).
 - Ghi độ dài dữ liệu cần truyền (tính bằng byte) vào thanh ghi tại offset 2 (Hình 4.4).
2. **Khởi động DMA Transfer:** Sau khi cấu hình xong, Nios V ghi giá trị 1 vào bit 0 (bit GO) của thanh ghi điều khiển (Control Register) tại offset 4. Thao tác ghi này (tín hiệu `iWrite=1`, `iAddress=4`, `iWritedata[0]=1`) được `CONTROL_SLAVE` phát hiện, và nó sẽ kích hoạt tín hiệu `Start` nội bộ, báo hiệu cho các module Master bắt đầu hoạt động (Hình 4.5).
3. **Quá trình truyền dữ liệu (Read Master và Write Master):**
- Module `READ_MASTER` bắt đầu tạo các giao dịch đọc Avalon-MM từ địa chỉ nguồn đã cấu hình, đọc dữ liệu và đẩy vào FIFO.
 - Khi FIFO có dữ liệu (`FF_empty=0`), module `WRITE_MASTER` bắt đầu đọc dữ liệu từ FIFO và tạo các giao dịch ghi Avalon-MM đến địa chỉ đích đã cấu hình.
 - Quá trình đọc và ghi này diễn ra song song (pipelined) cho đến khi đủ số byte yêu cầu đã được truyền. Hình 4.6 minh họa thời điểm module `READ_MASTER` hoàn thành việc đọc toàn bộ dữ liệu từ nguồn và ghi vào FIFO.
4. **Hoàn thành DMA Transfer:** Sau khi `WRITE_MASTER` ghi xong word dữ liệu cuối cùng vào bộ nhớ đích, nó sẽ kích hoạt tín hiệu `WM_done=1`. Module `CONTROL_SLAVE` nhận tín hiệu này, cập nhật thanh ghi trạng thái (Status Register) bằng cách đặt bit `DONE` (`status_done=1`) và xóa bit `BUSY` (`status_busy=0`), đồng thời tự động xóa bit `GO` (`control_go=0`) để kết thúc phiên truyền (Hình 4.7). Bộ xử lý Nios V có thể đọc thanh ghi trạng thái để biết khi nào quá trình DMA hoàn tất.
5. **Kết quả tổng thể:** Cửa sổ Transcript của Questa Sim (Hình 4.8) ghi lại log của quá trình mô phỏng, bao gồm các thông báo từ testbench hoặc mã phần mềm Nios V (nếu có `printf`), xác nhận các bước thực hiện và kết quả cuối cùng của việc truyền dữ liệu.



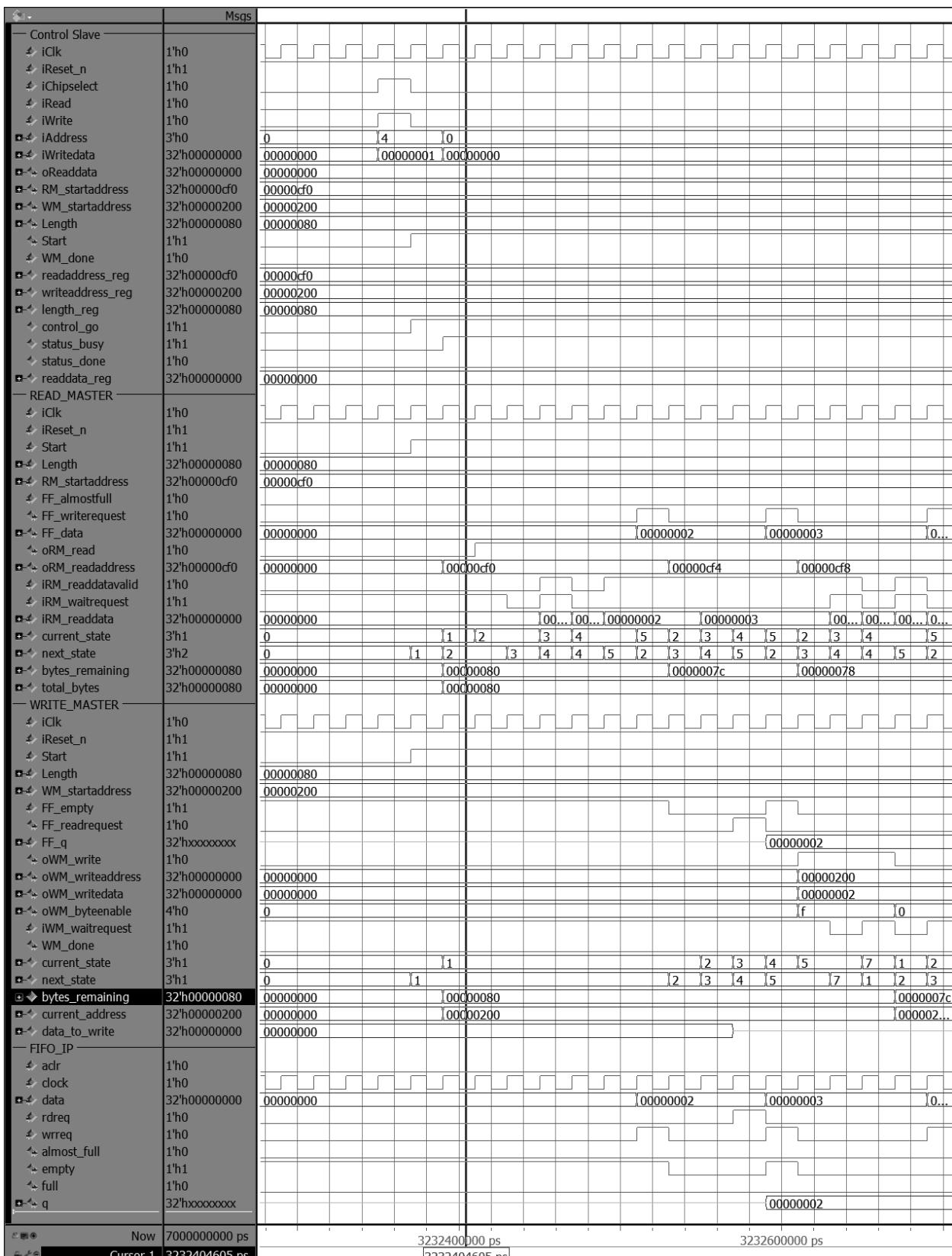
Hình 4.2: Mô phỏng: Ghi địa chỉ Read Master Start Address vào giao diện DMA Control Slave.



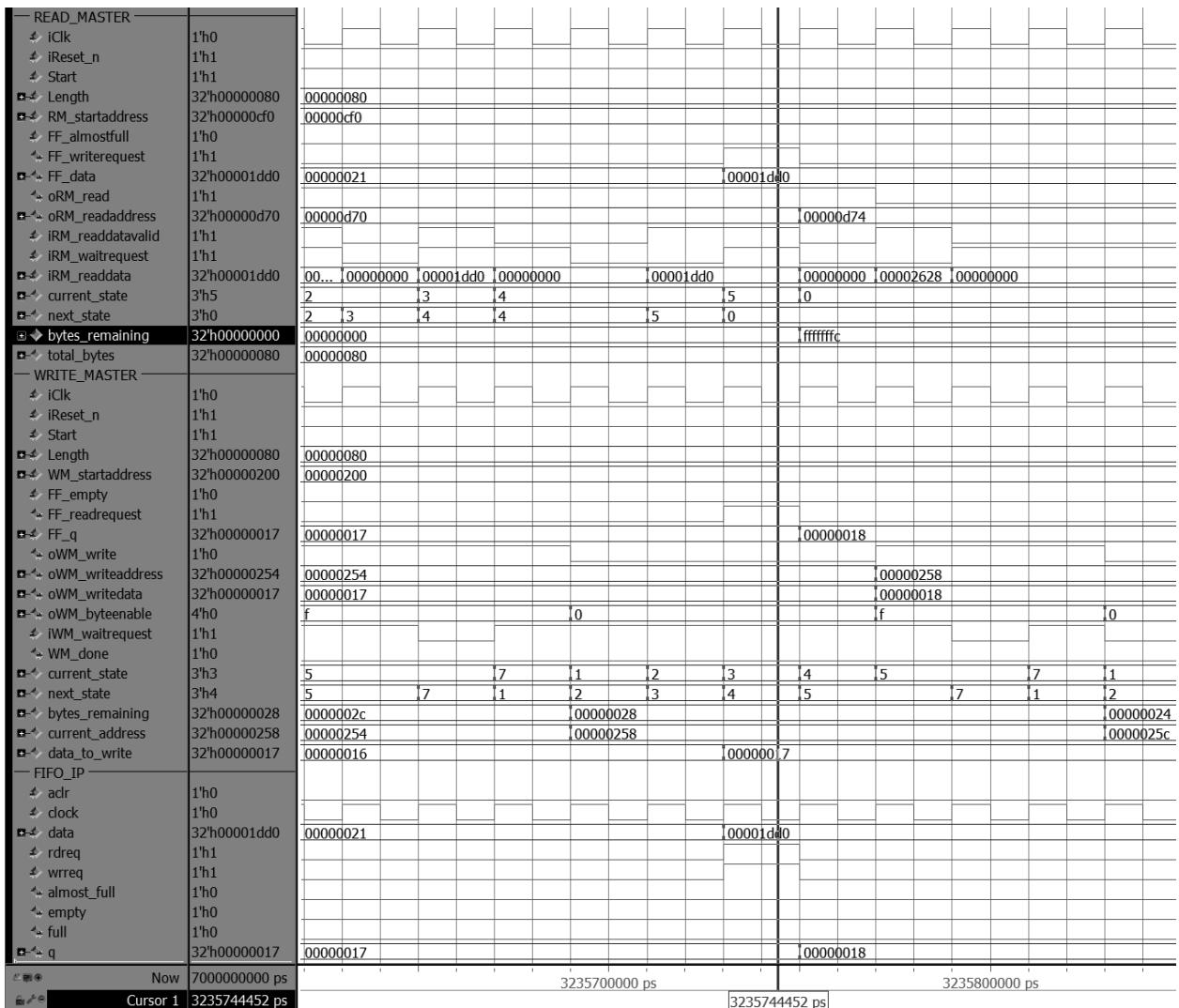
Hình 4.3: Mô phỏng: Ghi địa chỉ Write Master Start Address vào giao diện DMA Control Slave.



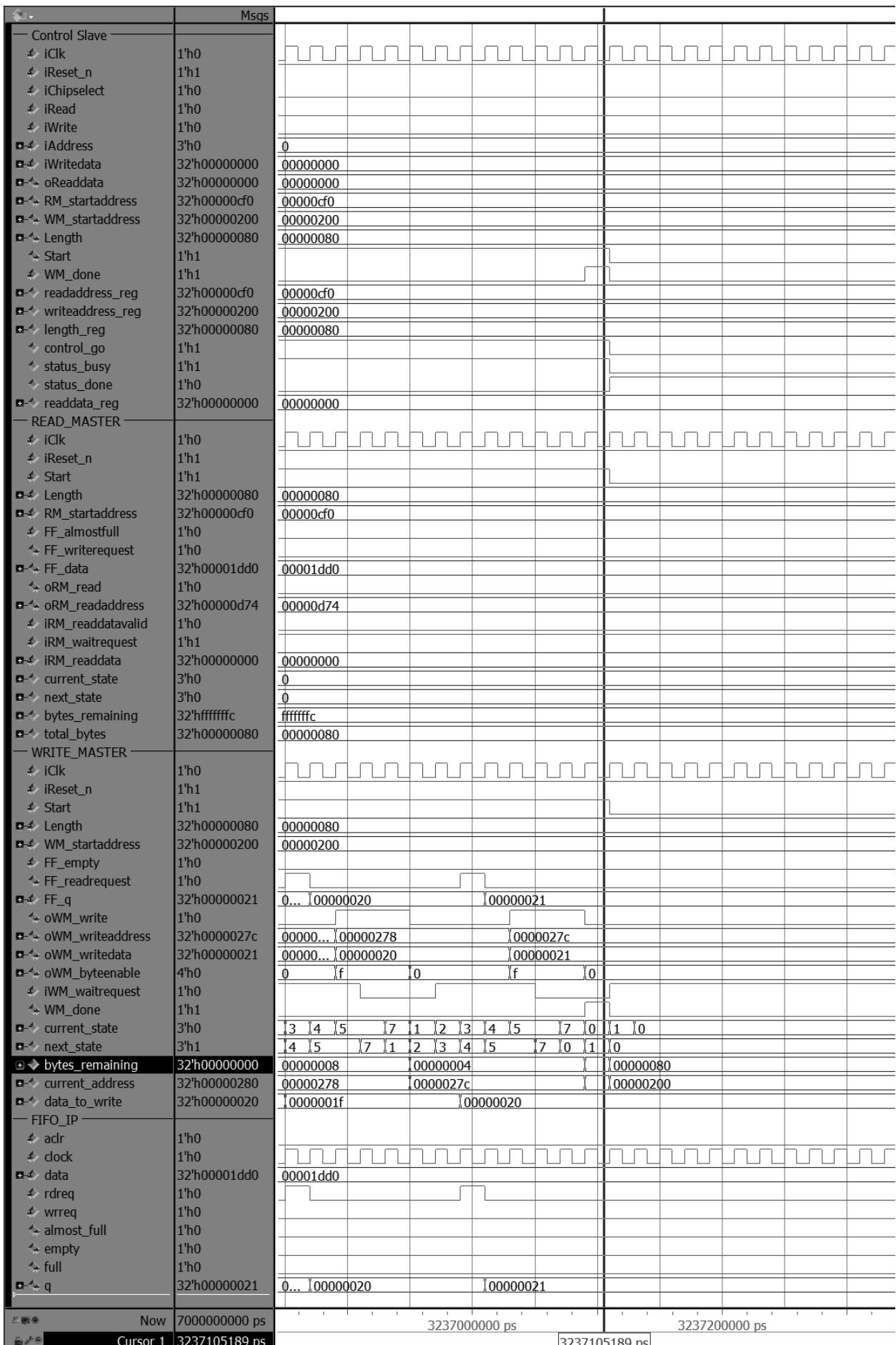
Hình 4.4: Mô phỏng: Thiết lập độ dài truyền (Transfer Length) qua giao diện DMA Control Slave.



Hình 4.5: Mô phỏng: Khởi tạo truyền DMA bằng cách đặt bit Control GO.



Hình 4.6: Mô phỏng: Hoàn thành các hoạt động của Read Master trong quá trình truyền DMA.



Hình 4.7: Mô phỏng: Tín hiệu hoàn thành (WM_done và status_done) cho biến truyen DMA thành công.

Library sim Memory List

Transcript

```

# av_lock          0 --> 0
# av_chipselect   0 --> 0
# av_debugaccess  0 --> 0
# DMA GO bit set.
# Polling DMA Status (Reg 5)...
# DMA DONE bit
#
# Flushing D-Cache for destination buffer...
# D-Cache flushed.
#
# Verifying DMA transfer...
# index 0 = 2
# index 1 = 3
# index 2 = 4
# index 3 = 5
# index 4 = 6
# index 5 = 7
# index 6 = 8
# index 7 = 9
# index 8 = 10
# index 9 = 11
# index 10 = 12
# index 11 = 13
# index 12 = 14
# index 13 = 15
# index 14 = 16
# index 15 = 17
# index 16 = 18
VSIM 19> run 1 ms
# index 17 = 19
# index 18 = 20
# index 19 = 21
# index 20 = 22
# index 21 = 23
VSIM 20> run 2 ms
# index 22 = 24
# index 23 = 25
# index 24 = 26
# index 25 = 27
# index 26 = 28
# index 27 = 29
# index 28 = 30
# index 29 = 31
# index 30 = 32
# index 31 = 33
VSIM 21> run 3 ms
# DMA Transfer Verification Successful!

VSIM 22>

```

Now: 13 ms Delta: 3 sim:/system_tb/system_inst/dma_controller_0

Hình 4.8: Mô phỏng: Transcript hiển thị quá trình và kết quả truyền DMA.

4.3 Kết luận Chương 4

Quá trình mô phỏng hệ thống SoC Nios V tích hợp bộ điều khiển DMA tùy chỉnh đã được thực hiện thành công bằng công cụ Questa Advanced Simulator. Các kết quả mô phỏng, được thể hiện qua các dạng sóng tín hiệu Avalon và cửa sổ transcript (Hình 4.2 đến 4.8), đã xác minh các chức năng chính của bộ điều khiển DMA:

- Khả năng nhận cấu hình (địa chỉ nguồn, đích, độ dài) từ bộ xử lý Nios V thông qua giao diện Avalon-MM Slave.
- Khả năng khởi động quá trình truyền dữ liệu khi nhận được lệnh GO.
- Hoạt động độc lập của các module Read Master và Write Master trong việc đọc dữ liệu từ nguồn, đệm qua FIFO, và ghi dữ liệu vào đích, tuân thủ đúng giao thức Avalon-MM [3].
- Khả năng báo hiệu hoàn thành (DONE) một cách chính xác sau khi hoàn tất việc truyền dữ liệu.

Kết quả mô phỏng khẳng định thiết kế bộ điều khiển DMA hoạt động đúng như mong đợi ở cấp độ Register Transfer Level (RTL), tạo tiền đề vững chắc cho việc triển khai và kiểm thử trên phần cứng FPGA thực tế được trình bày ở Chương 3.

CHƯƠNG 5: KẾT QUẢ ĐẠT ĐƯỢC VÀ THÁI ĐỘ

5.1 Về kiến thức

Qua quá trình thực tập tại CESLAB, tôi đã đạt được những kiến thức chuyên môn sau:

- **Kiến thức chuyên sâu về bộ xử lý Nios II¹:** Hiểu rõ về kiến trúc, cách thức hoạt động và các thành phần chính của bộ xử lý Nios II. Nắm vững cách cấu hình và tùy chỉnh bộ xử lý cho các ứng dụng cụ thể.
- **Hiểu biết về DMA và cơ chế truyền dữ liệu:** Hiểu rõ về cơ chế hoạt động của DMA, các loại truyền dữ liệu và các phương pháp tối ưu hóa hiệu suất truyền dữ liệu.
- **Kiến thức về thiết kế hệ thống nhúng:** Nắm vững các nguyên tắc thiết kế hệ thống nhúng, cách tổ chức và quản lý bộ nhớ, cũng như các kỹ thuật tối ưu hóa hiệu suất.
- **Hiểu biết về lập trình nhúng:** Phát triển kỹ năng lập trình cho hệ thống nhúng, bao gồm lập trình driver, lập trình giao tiếp với phần cứng và xử lý các vấn đề về đồng bộ hóa.
- **Kiến thức về các công cụ phát triển:** Làm quen với các công cụ phát triển phần cứng và phần mềm cho hệ thống nhúng, bao gồm Quartus Prime, ModelSim/Questa Sim, và các công cụ phát triển phần mềm cho Nios V (như Ashling RiscFree™ IDE).

5.2 Về kỹ năng

Ngoài những kiến thức chuyên môn, tôi cũng đã phát triển và cải thiện nhiều kỹ năng quan trọng:

- **Kỹ năng phân tích và giải quyết vấn đề:** Học được cách phân tích các vấn đề phức tạp, xác định nguyên nhân gốc rễ và đề xuất các giải pháp hiệu quả.

¹Mặc dù dự án chính tập trung vào Nios V, quá trình tìm hiểu và so sánh có thể bao gồm cả Nios II.

- **Kỹ năng lập trình và debug:** Cải thiện khả năng viết mã sạch, dễ bảo trì và có khả năng mở rộng.
- **Kỹ năng thiết kế và tối ưu hóa hệ thống:** Học được cách thiết kế hệ thống đáp ứng các yêu cầu kỹ thuật và tối ưu hóa hiệu suất bằng cách điều chỉnh các tham số hệ thống (ví dụ: cấu hình IP trong Platform Designer).
- **Kỹ năng trình bày và báo cáo:** Cải thiện khả năng trình bày ý tưởng và kết quả một cách rõ ràng, ngắn gọn và thuyết phục thông qua việc chuẩn bị báo cáo này.

5.3 Vẽ thái độ

Trong quá trình thực tập, tôi đã phát triển và thể hiện những thái độ tích cực sau:

- **Tinh thần học hỏi và cầu tiến:** Luôn giữ tâm thế sẵn sàng học hỏi kiến thức mới, chủ động tìm hiểu thêm về các công nghệ và kỹ thuật mới liên quan đến FPGA, SoC, và RISC-V.
- **Tinh thần trách nhiệm:** Luôn có trách nhiệm với công việc được giao, hoàn thành đúng thời hạn và đạt chất lượng yêu cầu.
- **Sự kiên trì và kiên nhẫn:** Không nản lòng trước khó khăn, luôn kiên trì tìm hiểu và giải quyết các vấn đề phức tạp, đặc biệt là trong quá trình debug phần cứng và phần mềm.
- **Thái độ cầu thị và tiếp nhận phản hồi:** Luôn sẵn sàng tiếp nhận phản hồi từ người hướng dẫn, coi đó là cơ hội để học hỏi và cải thiện bản thân.

Những kết quả đạt được trong quá trình thực tập không chỉ giúp tôi hoàn thành tốt nhiệm vụ được giao mà còn là nền tảng vững chắc cho sự phát triển nghề nghiệp của tôi trong tương lai trong lĩnh vực thiết kế hệ thống nhúng và FPGA.

TÀI LIỆU THAM KHẢO

Tiếng Anh

- [1] *AN 985: Nios® V Processor Tutorial.* 784468. Altera Corporation, An Intel Company. July 2024. URL: <https://www.intel.com/content/www/us/en/docs/programmable/784468/current/an-985-processor-tutorial.html>.
- [2] *Ashling® RiscFree® Integrated Development Environment (IDE) for Altera® FPGAs User Guide.* 25.1. 730783. Intel Corporation. 2025. URL: <https://www.intel.com/content/www/us/en/docs/programmable/730783/25-1/about-this-document.html>.
- [3] *Avalon® Interface Specifications.* 22.3. Intel Corporation. Sept. 2022. URL: <https://www.intel.com/content/www/us/en/docs/programmable/683091/22-3/typical-read-and-write-transfers.html>.
- [4] *Embedded Memory (RAM: 1-PORT, RAM: 2-PORT, ROM: 1-PORT, and ROM: 2-PORT) User Guide.* 17.0. Intel Corporation. Sept. 2021. URL: <https://www.intel.com/content/www/us/en/docs/programmable/683240/17-0/byte-enable.html>.
- [5] FPGAcademy. *DE10-Standard Quartus Settings File.* URL: https://fpgacademy.org/Downloads/DE10_Standard.qsf.
- [6] FPGAcademy. *FPGAcademy Boards Information.* URL: <https://fpgacademy.org/boards.html>.
- [7] Intel Community. *Error while generating Nios V/g simulation model in Platform Designer.* 2024. URL: <https://community.intel.com/t5/Intel-Quartus-Prime-Software/Error-while-generating-Nios-V-g-simulation-model-in-Platform/m-p/1613181/highlight/true?profile.language=pt>.
- [8] Intel Community. *Questa license cannot create an account in FPGA Self Service.* 2022. URL: <https://community.intel.com/t5/Intel-FPGA-Software-Installation/Questa-license-cannot-create-an-account-in-FPGA-Self-Service/td-p/1456416>.
- [9] Intel Corporation. *Nios V Embedded Processor Design Handbook.* 24.3.1. 2025. URL: <https://www.intel.com/content/www/us/en/docs/programmable/726952/24-3-1/about-the-embedded-processor.html>.

- [10] *Nios® V Processor Reference Manual*. 24.3.1. Altera Corporation, An Intel Company. 2025. URL: <https://www.intel.com/content/www/us/en/docs/programmable/683632/24-3-1/overview.html>.
- [11] *Nios® V Processor Software Developer Handbook*. 25.1. 743810. Intel Corporation. 2025. URL: <https://www.intel.com/content/www/us/en/docs/programmable/743810/25-1/overview-of-embedded-processor-development.html>.
- [12] Terasic Inc. *DE10-Standard Development Kit*. URL: <https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=1081>.

CHƯƠNG A: MÃ NGUỒN CỦA THIẾT KẾ

A.1 Mã Verilog Top-Level cho DMANiosV

```
1 module DMANiosV( input CLOCK_50,  input [0:0] KEY);
2   system Nios_system (
3     .clk_clk      (CLOCK_50),
4     .reset_reset_n(KEY[0])
5   );
6 endmodule
```

Listing A.1: DMANiosV.v - Top Level Module

A.2 Mã Verilog DMA Controller

```
1 module DMAController (
2   input          iClk ,
3   input          iReset_n ,
4   // Avalon Slave Interface
5   input          iChipselect ,
6   input          iRead ,
7   input          iWrite ,
8   input [2:0]    iAddress ,
9   input [31:0]   iWritedata ,
10  output [31:0]  oReaddata ,
11  // Avalon Read Master Interface
12  output         oRM_read ,
13  output [31:0]  oRM_readaddress ,
14  input          iRM_readdatavalid ,
15  input          iRM_waitrequest ,
16  input [31:0]   iRM_readdata ,
17  // Avalon Write Master Interface
18  output         oWM_write ,
19  output [31:0]  oWM_writememory ,
20  output [31:0]  oWM_writedata ,
21  output [3:0]   oWM_bytelenable , // <<< ADDED PORT
22  input          iWM_waitrequest
23 );
24  // Internal Signals
25  wire           Start ;
26  wire [31:0]    Length ;
```

```

27     wire [31:0]      RM_startaddress;
28     wire [31:0]      WM_startaddress;
29     wire              WM_done;
30
31 // FIFO Signals
32     wire              FF_almostfull;
33     wire              FF_writerequest;
34     wire [31:0]      FF_data;
35     wire              FF_empty;
36     wire [31:0]      FF_q;
37     wire              FF_readrequest;
38
39 // Instantiate CONTROL_SLAVE
40 CONTROL_SLAVE u_CONTROL_SLAVE (
41     .iClk(iClk),
42     .iReset_n(iReset_n),
43     .iChipselect(iChipselect),
44     .iRead(iRead),
45     .iWrite(iWrite),
46     .iAddress(iAddress),
47     .iWritedata(iWritedata),
48     .oReaddata(oReaddata),
49     .RM_startaddress(RM_startaddress),
50     .WM_startaddress(WM_startaddress),
51     .Length(Length),
52     .Start(Start),
53     .WM_done(WM_done)
54 );
55
56 // Instantiate READ_MASTER (Use revised version)
57 READ_MASTER u_READ_MASTER (
58     .iClk(iClk),
59     .iReset_n(iReset_n),
60     .Start(Start),
61     .Length(Length),
62     .RM_startaddress(RM_startaddress),
63     .FF_almostfull(FF_almostfull),
64     .FF_writerequest(FF_writerequest),
65     .FF_data(FF_data),
66     .oRM_read(oRM_read),
67     .oRM_readaddress(oRM_readaddress),
68     .iRM_readdatavalid(iRM_readdatavalid),
69     .iRM_waitrequest(iRM_waitrequest),

```

```

69      .iRM_readdata(iRM_readdata)
70  );
71
72 // Instantiate WRITE_MASTER (Use revised version with byteenable)
73 WRITE_MASTER u_WRITE_MASTER (
74     .iClk(iClk),
75     .iReset_n(iReset_n),
76     .Start(Start),
77     .Length(Length),
78     .WM_startaddress(WM_startaddress),
79     .FF_empty(FF_empty),
80     .FF_readrequest(FF_readrequest),
81     .FF_q(FF_q),
82     .oWM_write(oWM_write),
83     .oWM_writeaddress(oWM_writeaddress),
84     .oWM_writedata(oWM_writedata),
85     .oWM_byteenable(oWM_byteenable), // <<< CONNECTED PORT
86     .iWM_waitrequest(iWM_waitrequest),
87     .WM_done(WM_done)
88 );
89
90 // Instantiate FIFO
91 wire FF_full;
92 FIFO_IP FIFO_IP_inst (
93     .aclr(~iReset_n),
94     .clock(iClk),
95     .data(FF_data),
96     .rdreq(FF_readrequest),
97     .wrreq(FF_writerequest),
98     .almost_full(FF_almostfull),
99     .empty(FF_empty),
100    .full(FF_full),
101    .q(FF_q)
102 );
103
104 endmodule

```

Listing A.2: DMAController.v - DMA Controller Top Module

A.3 Mă Verilog Control Slave

```

1 module CONTROL_SLAVE (

```

```

2      input          iClk ,
3      input          iReset_n ,
4  // Avalon Slave Interface
5      input          iChipselect ,
6      input          iRead ,
7      input          iWrite ,
8      input [2:0]    iAddress ,
9      input [31:0]   iWritedata ,
10     output [31:0]  oReaddata ,
11  // DMA Configuration/Control Outputs
12     output [31:0]  RM_startaddress ,
13     output [31:0]  WM_startaddress ,
14     output [31:0]  Length ,
15     output          Start ,
16  // DMA Status Input
17     input          WM_done
18 );
19 // Parameter Definitions for Addresses
20 localparam ADDR_READADDRESS = 3'd0;
21 localparam ADDR_WRITEADDRESS = 3'd1;
22 localparam ADDR_LENGTH      = 3'd2;
23 // Address 3'd3 is unused
24 localparam ADDR_CONTROL     = 3'd4;
25 localparam ADDR_STATUS      = 3'd5;
26 // Addresses 3'd6, 3'd7 are unused
27
28 // Register Definitions
29 reg [31:0] readaddress_reg;
30 reg [31:0] writeaddress_reg;
31 reg [31:0] length_reg;
32 reg          control_go;      // Internal GO signal, latched
33 reg          status_busy;
34 reg          status_done;
35
36 // Output Assignments
37 assign RM_startaddress = readaddress_reg;
38 assign WM_startaddress = writeaddress_reg;
39 assign Length         = length_reg;
40 assign Start          = control_go; // Start signal follows the
latched GO bit
41
42 // Read Data Output Logic (Combinatorial)

```

```

43  reg [31:0] readdata_reg;
44  assign oReaddata = readdata_reg;
45
46 // Combinatorial block for read data multiplexing
47 always @(*) begin
48     // Default read data to 0
49     readdata_reg = 32'd0;
50     if (iChipselect && iRead) begin
51         case (iAddress)
52             ADDR_READADDRESS: readdata_reg = readaddress_reg;
53             ADDR_WRITEADDRESS: readdata_reg = writeaddress_reg;
54             ADDR_LENGTH: readdata_reg = length_reg;
55             ADDR_CONTROL: readdata_reg = {31'd0, control_go};
56             // Return current GO bit status
57             ADDR_STATUS: readdata_reg = {30'd0, status_busy,
58             status_done}; // Return Busy and Done status
59             default: readdata_reg = 32'd0; // Reads from
60             unused addresses return 0
61         endcase
62     end
63 end
64
65 // Register Write and State Logic (Sequential)
66 always @(posedge iClk or negedge iReset_n) begin
67     if (!iReset_n) begin
68         readaddress_reg <= 32'd0;
69         writeaddress_reg <= 32'd0;
70         length_reg <= 32'd0;
71         control_go <= 1'b0;
72         status_busy <= 1'b0;
73         status_done <= 1'b0;
74     end else begin
75         // Handle register writes from CPU
76         if (iChipselect && iWrite) begin
77             case (iAddress)
78                 ADDR_READADDRESS: readaddress_reg <= iWritedata;
79                 ADDR_WRITEADDRESS: writeaddress_reg <= iWritedata;
80                 ADDR_LENGTH: length_reg <= iWritedata;
81                 ADDR_CONTROL: control_go <= iWritedata
82             [0]; // Latch the GO bit from CPU
83             ADDR_STATUS: begin
84                 // Bit 0: Write 1 to clear Done status

```

```

81                     if (iWritedata[0]) begin
82                         status_done <= 1'b0;
83                     end
84                     // Other bits are read-only or unused
85                 end
86             default: ; // No action for unused addresses
87         endcase
88     end
89
90     if (control_go && !status_busy && !status_done) begin
91         status_busy <= 1'b1;
92         status_done <= 1'b0;
93     end
94
95     // If the Write Master signals completion THIS cycle
96     if (WM_done) begin
97         status_busy <= 1'b0;
98         status_done <= 1'b1;
99         control_go <= 1'b0; // <<< Clear GO bit automatically
100        on completion
101    end
102
103    // If CPU clears GO bit while busy, we might need an abort
104    mechanism (currently ignored)
105
106 endmodule

```

Listing A.3: CONTROL_SLAVE.v - Avalon Slave for Control/Status

A.4 Mā Verilog Read Master

```

1 module READ_MASTER (
2     input          iClk ,
3     input          iReset_n ,
4     input          Start ,
5     input [31:0]   Length ,
6     input [31:0]   RM_startaddress ,
7     input          FF_almostfull ,
8     output reg    FF_writerequest ,
9     output reg [31:0] FF_data ,

```

```

10    output reg      oRM_read,
11    output reg [31:0] oRM_readaddress ,
12    input       iRM_readdatavalid ,
13    input       iRM_waitrequest ,
14    input [31:0]  iRM_readdata
15 );
16 // State Machine States
17 parameter IDLE = 3'b000 , CHECK_FIFO = 3'b01 , REQUEST = 3'b010 ,
18 WAIT_DATA = 3'b011 , WRITE_FIFO = 3'b100 , WAIT_FIFO = 3'b101 ;
19 reg [2:0] current_state , next_state ;
20
21 // Internal Registers
22 reg [31:0] bytes_remaining ;
23 reg [31:0] total_bytes ;
24
25 // Initialization
26 always @(posedge iClk or negedge iReset_n) begin
27   if (!iReset_n) begin
28     current_state <= IDLE;
29     oRM_read        <= 1'b0;
30     oRM_readaddress <= 32'd0;
31     FF_writerequest <= 1'b0;
32     FF_data         <= 32'd0;
33     bytes_remaining <= 32'd0;
34     total_bytes      <= 32'd0;
35   end else begin
36     current_state <= next_state;
37     FF_writerequest <= 1'b0;
38
39     case (current_state)
40       IDLE: begin
41         oRM_read        <= 1'b0;
42         FF_writerequest <= 1'b0;
43         if (Start && oRM_readaddress < RM_startaddress +
44             Length) begin
45           bytes_remaining <= Length;
46           oRM_readaddress <= RM_startaddress;
47           total_bytes      <= Length;
48         end
49       end
50       CHECK_FIFO: begin
51         if (!FF_almostfull) begin

```

```

50          oRM_read <= 1'b1;
51      end else begin
52          oRM_read <= 1'b0;
53      end
54  end
55 REQUEST, WAIT_DATA: begin
56
57      //oRM_read <= 1'b0;
58  end
59 WRITE_FIFO: begin
60     if (iRM_readdatavalid) begin
61         //oRM_read           <= 1'b0;
62         FF_writerequest <= 1'b1;
63         FF_data          <= iRM_readdata;
64     end else begin
65         FF_writerequest <= 1'b0;
66     end
67  end
68 WAIT_FIFO: begin
69     oRM_readaddress <= oRM_readaddress + 4;
70     bytes_remaining <= bytes_remaining - 4;
71  end
72 endcase
73 end
74
75
76 // Next State Logic
77 always @(*) begin
78     case (current_state)
79         IDLE: begin
80             if (Start && oRM_readaddress < RM_startaddress + Length
81             && !FF_almostfull)
82                 next_state = CHECK_FIFO;
83             else
84                 next_state = IDLE;
85         end
86         CHECK_FIFO: begin
87             if (!FF_almostfull)
88                 next_state = REQUEST;
89             else
90                 next_state = CHECK_FIFO;
91         end

```

```

91      REQUEST: begin
92          if (iRM_waitrequest)
93              next_state = REQUEST;
94          else
95              next_state = WAIT_DATA;
96      end
97      WAIT_DATA: begin
98          if (iRM_readdatavalid)
99              next_state = WRITE_FIFO;
100         else
101             next_state = WAIT_DATA;
102     end
103     WRITE_FIFO: begin
104         if (iRM_readdatavalid) begin
105             next_state = WAIT_FIFO;
106         end else begin
107             next_state = WRITE_FIFO;
108         end
109         /* if (bytes_remaining == 32'd0 || oRM_readaddress ==
110            RM_startaddress + Length || FF_almostfull)
111             next_state = IDLE;
112         else
113             next_state = REQUEST; */
114         //next_state = WAIT_FIFO;
115     end
116     WAIT_FIFO: begin
117         if (bytes_remaining == 32'd0 || oRM_readaddress ==
118            RM_startaddress + Length || FF_almostfull)
119             next_state = IDLE;
120         else
121             next_state = REQUEST;
122     end
123     default: next_state = IDLE;
124 endcase
125 end
126 endmodule

```

Listing A.4: READ_MASTER.v - Avalon Read Master Module

A.5 Mā Verilog Write Master

```

1 module WRITE_MASTER (

```

```

2      input          iClk ,
3      input          iReset_n ,
4      // Control Inputs
5      input          Start ,           // From CONTROL_SLAVE
6      input [31:0]    Length ,         // Total bytes to write
7      input [31:0]    WM_startaddress , // Start address for writing
8      // FIFO Interface
9      input          FF_empty ,       // FIFO empty signal
10     output reg     FF_readrequest , // Request to read from FIFO
11     input [31:0]    FF_q ,          // Data read from FIFO
12     // Avalon Write Master Interface
13     output reg     oWM_write ,      // Avalon write signal
14     output reg [31:0] oWM_writeaddress , // Avalon write address
15     output reg [31:0] oWM_writedata ,   // Avalon write data
16     output reg [3:0]  oWM_bytenable ,   // Avalon byte enable
17     input          iWM_waitrequest , // Avalon wait request
18     // Status Output
19     output reg     WM_done          // DMA Write Master Done
20 );
21     // State Machine States
22     parameter IDLE          = 3'b000 ;
23     parameter CHECK_FIFO    = 3'b001; // Check if data is available in
FIFO
24     parameter READ_FIFO     = 3'b010; // Request data from FIFO
25     parameter WAIT_FIFO_DATA = 3'b011; // Wait for FIFO data (usually
available next cycle)
26     parameter START_WRITE    = 3'b100; // Assert write , address , data ,
bytenable
27     parameter WAIT_WRITE_ACK = 3'b101; // Wait for waitrequest=0 first
time
28     parameter UPDATE_CNT     = 3'b111; // Update counters , check
completion
29
30     reg [2:0] current_state , next_state ;
31
32     // Internal Registers
33     reg [31:0] bytes_remaining ;
34     reg [31:0] current_address ;
35     reg [31:0] data_to_write ; // Register to hold data from FIFO
36
37     // Initialization and State Register
38     always @ (posedge iClk or negedge iReset_n) begin

```

```

39      if (!iReset_n) begin
40          current_state <= IDLE;
41          bytes_remaining <= 32'd0;
42          current_address <= 32'd0;
43          WM_done <= 1'b0;
44          // *** ADD RESET FOR data_to_write ***
45          data_to_write <= 32'd0;
46          // *** Outputs are reset below ***
47      end else begin
48          current_state <= next_state;
49          // Update counters in UPDATE_CNT state
50          if (current_state == UPDATE_CNT) begin
51              current_address <= current_address + 4;
52              bytes_remaining <= bytes_remaining - 4;
53          end
54          // Latch configuration in IDLE state when starting
55          if (next_state == CHECK_FIFO && current_state == IDLE) begin
56              bytes_remaining <= Length;
57              current_address <= WM_startaddress;
58              WM_done <= 1'b0; // Clear done when starting a
59              new transfer
60          end
61          // Set WM_done when transitioning to IDLE from UPDATE_CNT (
62          last word processed)
63          if (current_state == UPDATE_CNT && bytes_remaining <= 4)
64      begin
65          WM_done <= 1'b1;
66      end
67      // Latch data from FIFO in WAIT_FIFO_DATA state (uses state
68      *before* edge)
69      // Moved this inside the 'else' block for clarity , happens
70      concurrently
71      if (current_state == READ_FIFO) begin
72          data_to_write <= FF_q;
73      end
74      end
75  end
76
77  // Output Logic (Registered Outputs)
78  always @(posedge iClk or negedge iReset_n) begin
79      if (!iReset_n) begin
80          FF_readrequest <= 1'b0;

```

```

76          oWM_write      <= 1'b0;
77          oWM_writeaddress <= 32'd0;
78          oWM_writedata    <= 32'd0;
79          oWM_byteenable   <= 4'b0000;
80      end else begin
81          // Default assignments for next cycle (registered outputs)
82          FF_readrequest <= 1'b0;
83          oWM_write      <= 1'b0;
84          oWM_byteenable <= 4'b0000;
85          // oWM_writeaddress and oWM_writedata hold previous value
unless changed
86
87          // Drive outputs based on state (using state *before* edge)
88      case (current_state)
89          READ_FIFO: begin // State 2
90              FF_readrequest <= 1'b1; // Request data for next
cycle
91          end
92          START_WRITE: begin // State 4
93              oWM_write      <= 1'b1;
94              oWM_writeaddress <= current_address;
95              oWM_writedata    <= FF_q; // Use value latched on *
previous* edge
96              oWM_byteenable   <= 4'b1111;
97          end
98          WAIT_WRITE_ACK: begin // State 5
99              oWM_write      <= 1'b1;           // Keep write
asserted
100             oWM_writeaddress <= current_address; // Keep address
stable
101             oWM_writedata    <= FF_q; // Keep data stable
102             oWM_byteenable   <= 4'b1111; // Keep byteenable
asserted
103         end
104         // In other states , outputs driven low/inactive by
defaults above
105         default: begin
106             end
107         endcase
108     end
109 end
110

```

```

111 // Next State Logic (Combinatorial)
112 always @(*) begin
113     next_state = current_state; // Default to stay in current state
114     if (!Start) begin
115         next_state = IDLE; // Reset to IDLE if Start is deasserted
116     end else
117     case (current_state)
118         IDLE: begin // State 0
119             if (Start && Length > 0) begin
120                 next_state = CHECK_FIFO; // -> 1
121             end else begin
122                 next_state = IDLE; // -> 0
123             end
124         end
125         CHECK_FIFO: begin // State 1
126             if (!FF_empty) begin
127                 next_state = READ_FIFO; // -> 2
128             end else begin
129                 if (bytes_remaining > 0 && Start) begin
130                     next_state = CHECK_FIFO; // -> 1
131                 end else begin
132                     next_state = IDLE; // -> 0
133                 end
134             end
135         end
136         READ_FIFO: begin // State 2
137             next_state = WAIT_FIFO_DATA; // -> 3
138         end
139         WAIT_FIFO_DATA: begin // State 3
140             next_state = START_WRITE; // -> 4
141         end
142         START_WRITE: begin // State 4
143             // Immediately move to wait if write is asserted
144             next_state = WAIT_WRITE_ACK; // -> 5
145         end
146         WAIT_WRITE_ACK: begin // State 5
147             if (!iWM_waitrequest) begin
148                 next_state = UPDATE_CNT; // -> 6 (Write accepted)
149             end else begin
150                 next_state = WAIT_WRITE_ACK; // -> 5 (Keep waiting)
151             end
152         end

```

```

153     UPDATE_CNT: begin // State 6
154         if (bytes_remaining <= 4) begin // Check before
155             decrement
156             next_state = IDLE; // -> 0 (Transfer complete)
157         end else begin
158             next_state = CHECK_FIFO; // -> 1 (More data to write
159         )
160         end
161     default: next_state = IDLE;
162     endcase
163 end
164
165 endmodule

```

Listing A.5: WRITE_MASTER.v - Avalon Write Master Module

A.6 Mã nguồn C kiểm thử DMA

```

1 #include <stdio.h>
2 #include <stdint.h>
3 #include <stdbool.h>
4 #include "system.h"           // System-specific definitions
5 #include "io.h"              // I/O functions (if needed)
6 #include "sys/alt_cache.h"   // <<< Cache management functions
7 #include "alt_types.h"       // <<< For alt_u32
8
9 #define DMA_REG_READADDRESS 0 // Offset 0 (Byte Address 0x00)
10 #define DMA_REG_WRITEADDRESS 1 // Offset 1 (Byte Address 0x04)
11 #define DMA_REG_LENGTH 2    // Offset 2 (Byte Address 0x08)
12 // Offset 3 is unused
13 #define DMA_REG_CONTROL 4 // Offset 4 (Byte Address 0x10)
14 #define DMA_REG_STATUS 5 // Offset 5 (Byte Address 0x14)
15
16 // Control Register Bits
17 #define DMA_CONTROL_GO (1 << 0)
18
19 // Status Register Bits
20 #define DMA_STATUS_DONE (1 << 0)
21 #define DMA_STATUS_BUSY (1 << 1)
22

```

```

23 // Reset Timeout (number of status polls)
24 #define DMA_RESET_TIMEOUT 10000
25
26 // Buffer size
27 #define BUFFER_SIZE 32
28
29 #define arg_simulation 1 // Set to 0 for simulation, 1 for hardware
30
31 // Source data buffer (likely placed in onchip_memory2_0 by linker)
32 // Initialized with values 2 to 33
33 alt_u32 pdata0[BUFFER_SIZE] = {
34     2, 3, 4, 5, 6, 7, 8, 9,
35     10, 11, 12, 13, 14, 15, 16, 17,
36     18, 19, 20, 21, 22, 23, 24, 25,
37     26, 27, 28, 29, 30, 31, 32, 33};
38
39 // Destination data buffer pointer - explicitly placed in
40 // onchip_memory2_1
41 // Adding an offset within the memory for clarity/safety
41 volatile alt_u32 *pdata1 = (alt_u32*) (ONCHIP_MEMORY2_1_BASE + 0x200);
42 // Offset 0x100 within mem 1
43
44 // Write to DMA register (using word offset for IOWR)
45 static inline void dma_write_reg(alt_u32 reg_offset, alt_u32 value)
46 {
47     IOWR(DMA_CONTROLLER_0_BASE, reg_offset, value);
48 }
49
50 // Read from DMA register (using word offset for IORD)
51 static inline alt_u32 dma_read_reg(alt_u32 reg_offset)
52 {
53     return IORD(DMA_CONTROLLER_0_BASE, reg_offset);
54 }
55
55 // — NEW FUNCTION: Reset DMA Controller —
56 // Attempts to reset the DMA state via software writes
57 bool reset_dma()
58 {
59     printf("Resetting DMA Controller...\\n");
60
61     // 1. Clear the GO bit in the Control Register to stop new transfers
62     printf("    Clearing GO bit (Reg %d)...\\n", DMA_REG_CONTROL);

```

```

63     dma_write_reg(DMA_REG_CONTROL, 0);
64
65 // 2. Clear configuration registers
66 printf("  Clearing Address/Length Regs (%d, %d, %d) ... \n",
67        DMA_REG_READADDRESS, DMA_REG_WRITEADDRESS, DMA_REG_LENGTH);
68     dma_write_reg(DMA_REG_READADDRESS, 0);
69     dma_write_reg(DMA_REG_WRITEADDRESS, 0);
70     dma_write_reg(DMA_REG_LENGTH, 0);
71
72 // 3. Clear the DONE status bit (Write-1-to-clear)
73 // Check if DONE is already set before clearing
74 if (dma_read_reg(DMA_REG_STATUS) & DMA_STATUS_DONE)
75 {
76     printf("  Clearing DONE status bit (Reg %d) ... \n",
77 DMA_REG_STATUS);
78     dma_write_reg(DMA_REG_STATUS, DMA_STATUS_DONE); // Write 1 to
79 DONE bit
80 }
81
82 // 4. Wait for the BUSY flag to clear (optional but recommended)
83 printf("  Waiting for BUSY bit to clear (Reg %d) ... \n",
84 DMA_REG_STATUS);
85 int timeout = DMA_RESET_TIMEOUT;
86 while ((dma_read_reg(DMA_REG_STATUS) & DMA_STATUS_BUSY) && (timeout
87 > 0))
88 {
89     timeout--;
90     // Optional delay
91     // for(volatile int d=0; d<10; d++);
92 }
93
94 if (timeout == 0)
95 {
96     printf("  Error: DMA reset timeout - BUSY bit did not clear!\n");
97     return false;
98 }
99
100 printf("DMA Reset complete. Final Status: 0x%lx\n", dma_read_reg(
101 DMA_REG_STATUS));
102 return true;
103 }
```

```

99 // — End of reset_dma function —
100
101 // Start DMA transfer
102 void start_dma_transfer(alt_u32 src_addr, alt_u32 dst_addr, alt_u32
103 length_bytes)
104 {
105     printf("Configuring DMA Regs:\n");
106     printf("  Reg %d (Read Addr) : 0x%08lx\n", DMA_REG_READADDRESS,
107 src_addr);
108     printf("  Reg %d (Write Addr): 0x%08lx\n", DMA_REG_WRITEADDRESS,
109 dst_addr);
110     printf("  Reg %d (Length)      : %lu bytes\n", DMA_REG_LENGTH,
111 length_bytes);
112     printf("  Reg %d (Control)     : 0x%08X\n", DMA_REG_CONTROL, (alt_u32)
113 DMA_CONTROL_GO);
114
115     // Write the source and destination addresses
116     dma_write_reg(DMA_REG_READADDRESS, src_addr);
117     dma_write_reg(DMA_REG_WRITEADDRESS, dst_addr);
118 }
119
120 // Wait for DMA to complete
121 bool wait_for_dma_done()
122 {
123     alt_u32 status;
124     printf("Polling DMA Status (Reg %d)...\n", DMA_REG_STATUS);
125     // Add a timeout to prevent infinite loops
126     int timeout = 1000000; // Adjust as needed
127     while (timeout > 0)
128     {
129         status = dma_read_reg(DMA_REG_STATUS);
130         // printf("Status: 0x%08lx\n", status); // Debug print
131         if (status & DMA_STATUS_DONE)
132         {
133             printf("DMA DONE bit detected.\n");
134             // Optional: Clear the DONE bit after detecting it by
writing 1 to it

```

```

135         // dma_write_reg(DMA_REG_STATUS, DMA_STATUS_DONE) ;
136         return true ;
137     }
138     timeout-- ;
139 }
140 printf("Error: Timeout waiting for DMA DONE bit. Final Status: 0x%lx
141 \n", status) ;
142 return false ;
143 }
144 // Verify DMA transfer
145 bool verify_dma_transfer(alt_u32 *src , alt_u32 *dst , alt_u32 num_words)
146 {
147     bool success = true ;
148     printf("Verifying %lu words... \n", num_words) ;
149     for (int i = 0; i < num_words; i++)
150     {
151         // Use volatile pointers for reading destination to bypass cache
152         optimization if needed ,
153         // although cache flush is the proper method .
154         volatile alt_u32 read_data = dst[i] ;
155         if (read_data != src[i])
156         {
157             printf("Mismatch at index %d: expected 0x%08lx , got 0x%08lx\"
158             n" ,
159             i , (alt_u32)src[i] , (alt_u32)read_data) ;
160             success = false ;
161             // return false ; // Optionally exit on first mismatch
162         }
163         else
164             printf("index %d = %d\n" , i , *(dst + i)) ;
165     }
166     return success ;
167 }
168 int main()
169 {
170     alt_u32 src_address = (alt_u32)pdata0 ;
171     alt_u32 dst_address = (alt_u32)pdata1 ;
172
173     // Transfer size in bytes
174     alt_u32 length_bytes = BUFFER_SIZE * sizeof(alt_u32) ;

```

```

174
175     if (arg_simulation)
176     {
177         printf("=====\\n");
178         printf("= 21207001 - Bui Thanh Dat - HCMUS - FETEL - CESLAB =\\n");
179         printf("= Nios V DMA Example =\\n");
180         printf("=====\\n");
181     };
182     // --- Reset DMA at the start ---
183     if (!reset_dma())
184     {
185         printf("DMA failed to reset. Halting.\\n");
186         return -1;
187     }
188     // --- DMA should be idle now ---
189
190     // Ensure source data is flushed from cache *before* DMA reads
191     it
192         alt_dcache_flush((void *)pdata0, sizeof(pdata0));
193
194         printf("\\nSystem Base Addresses:\\n");
195         printf("  ONCHIP_MEMORY2_0_BASE: 0x%08lx\\n", (alt_u32)
196             ONCHIP_MEMORY2_0_BASE);
197         printf("  ONCHIP_MEMORY2_1_BASE: 0x%08lx\\n", (alt_u32)
198             ONCHIP_MEMORY2_1_BASE);
199         printf("  DMA_CONTROLLER_0_BASE:    0x%08lx\\n", (alt_u32)
200             DMA_CONTROLLER_0_BASE);
201
202         printf("Actual pdata0 address: %p\\n", pdata0); // Verify source
203         location
204         printf("Actual pdata1 address: %p\\n", pdata1); // Verify
205         destination location
206
207         // Check if DMA is busy before starting a new transfer (shouldn't
208         // be after reset)
209         if (dma_read_reg(DMA_REG_STATUS) & DMA_STATUS_BUSY)
210         {
211             printf("Error: DMA is busy even after reset! Status: 0x%lx\\n",
212                 dma_read_reg(DMA_REG_STATUS));
213             return -1;
214         }

```

```

205
206    // Start DMA transfer
207    printf("\nStarting DMA transfer...\n");
208    printf("Source Address: 0x%08lx (pdata0)\n", src_address);
209    printf("Destination Address: 0x%08lx (pdata1)\n", dst_address);
210    printf("Length: %lu bytes (%d words)\n", length_bytes,
211           BUFFER_SIZE);

212    // Print destination buffer content *before* clearing and
213    // transfer
214    // Note: This read might be from cache if previously accessed
215    printf("\nDestination Buffer Before Clearing (potentially cached
216    ):");
217    for (int i = 0; i < BUFFER_SIZE; i++)
218    {
219        printf(" index %2d: 0x%08lx\n", i, (alt_u32)pdata1[i]);
220        if ((i + 1) % 8 == 0)
221            printf("\n"); // Formatting
222    }

223    // Reset the destination buffer using CPU writes
224    printf("\nResetting destination buffer via CPU...\n");
225    for (int i = 0; i < BUFFER_SIZE; i++)
226    {
227        pdata1[i] = 0;
228    }
229    // Flush the cache *after* CPU writes zeros to ensure zeros are
230    // in memory
231    alt_dcache_flush((void *)dst_address, length_bytes);
232    printf("Destination buffer cleared and cache flushed.\n");

233    // Optional: Read back immediately after flush to confirm zeros
234    printf("\nDestination Buffer After Reset & Flush (read back):\n");
235    for (int i = 0; i < BUFFER_SIZE; i++)
236    {
237        printf(" index %2d: 0x%08lx\n", i, (alt_u32)pdata1[i]);
238        if ((i + 1) % 8 == 0)
239            printf("\n"); // Formatting
240    }
241

```

```

242 // Start the actual DMA
243 start_dma_transfer(src_address, dst_address, length_bytes);
244
245 // Wait for DMA to complete
246 if (!wait_for_dma_done())
247 {
248     printf("DMA transfer failed or timed out!\n");
249     // Optional: Attempt another reset here?
250     // reset_dma();
251     return -1;
252 }
253
254 // !!! Crucial: Flush cache for destination region BEFORE
255 // verification !!!
256 printf("\nFlushing D-Cache for destination buffer...\n");
257 alt_dcache_flush((void *)dst_address, length_bytes);
258 printf("D-Cache flushed.\n");
259
260 // Verify DMA transfer by reading memory (now hopefully coherent)
261 printf("\nVerifying DMA transfer...\n");
262 if (verify_dma_transfer(pdata0, pdata1, BUFFER_SIZE))
263 {
264     printf("DMA Transfer Verification Successful!\n");
265 }
266 else
267 {
268     printf("DMA Transfer Verification Failed!\n");
269 }
270
271 return 0;

```

Listing A.6: hello_world.c - Nios V DMA Test Application

CHƯƠNG B: MỘT SỐ LỖI THƯỜNG GẶP

Phần này bao gồm một số lỗi thường gặp và cách khắc phục trong quá trình triển khai và gỡ lỗi hệ thống Nios V với DMA.

B.1 Lỗi Tạo BSP hoặc Biên dịch Ứng dụng

Một lỗi phổ biến khi chạy lệnh `niosv-app` là báo không tìm thấy tệp hoặc thư mục nguồn (Hình B.1).

- **Nguyên nhân:** Đường dẫn đến thư mục `app`, `bsp`, hoặc tệp mã nguồn C (`-s=...`) không chính xác so với vị trí hiện tại (hoặc không tồn tại) của Nios V Shell.
- **Khắc phục:** Đảm bảo đã `cd` vào thư mục gốc của dự án Quartus (DMANiosVIntern) trước khi chạy lệnh `niosv-app`. Kiểm tra lại các đường dẫn tương đối (`software/app`, `software/bsp`) trong lệnh. Tên tệp mã nguồn C trong thư mục `app` phải khớp với tên được chỉ định bởi tham số `-s`.

```
[niosv-shell] D:\FETEL_WorkDir\Y4S2_DMANiosVIntern> niosv-app -a=software/app -b=software/bsp -s=software/app  
2025.04.10.15:37:43 Error: Source file or directory "software\app" does not exist.
```

Hình B.1: Lỗi `niosv-app` báo đường dẫn hoặc tệp source code không tồn tại.

B.2 Sự cố Kết nối JTAG và Debug

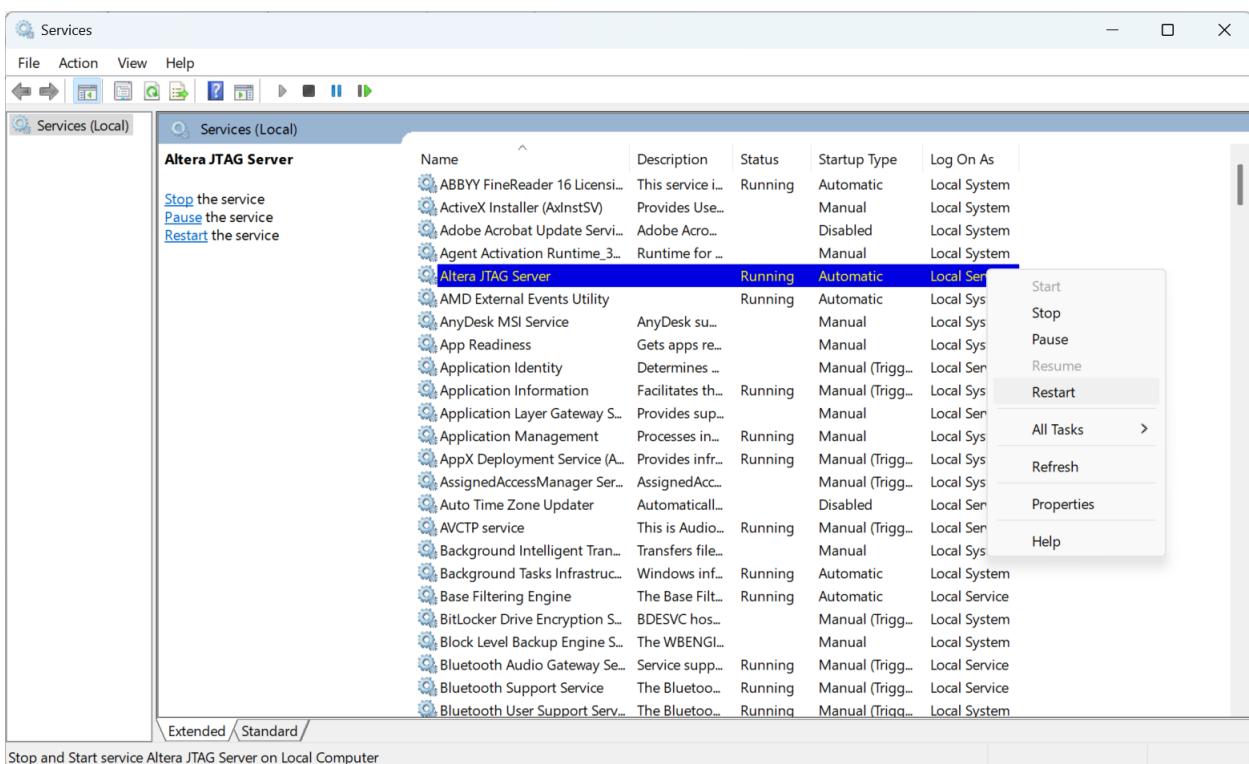
Giao tiếp JTAG có thể bị treo hoặc không ổn định, dẫn đến các vấn đề như:

- Quartus Programmer không nhận diện được bo mạch DE10-Standard.
- Ashling IDE không thể kết nối với Nios V core khi Debug.
- Cửa sổ `juart-terminal` không hiển thị output hoặc không phản hồi.
- Không thể tải tệp ELF lên bộ xử lý.

Khắc phục:

- **Kiểm tra kết nối vật lý:** Đảm bảo cáp USB-Blaster được kết nối chắc chắn giữa PC và bo mạch.

- **Khởi động lại JTAG Server:** Dịch vụ Altera JTAG Server chạy nền trên Windows có thể bị lỗi. Khởi động lại nó thông qua Task Manager -> Services (tìm "Altera JTAG Server" hoặc "JTAGServer") hoặc dùng lệnh `jtagconfig -stopserver` và `jtagconfig -startserver` trong Command Prompt. Có thể cần quyền Administrator. (Xem Hình B.2 để khởi động lại từ Services).
- **Kiểm tra Driver USB-Blaster:** Đảm bảo driver cho USB-Blaster đã được cài đặt đúng cách. Kiểm tra trong Device Manager của Windows.
- **Đóng các ứng dụng khác:** Đảm bảo không có instance nào khác của Quartus Programmer, Ashling IDE, hoặc juart-terminal đang cố gắng truy cập vào kết nối JTAG cùng lúc.



Hình B.2: Khởi động lại Altera JTAG Server trong Services của Windows.

B.3 Không có file ‘.sof’ để nạp xuống board

Nếu gặp lỗi khi tạo tệp ‘.sof’ (liên quan đến biên dịch thất bại), hãy kiểm tra lại trạng thái giấy phép Nios V/m trong Quartus License Setup (Tham khảo Mục B.5). Đôi khi giấy phép hết hạn hoặc cấu hình sai có thể gây lỗi biên dịch, gián tiếp ảnh

hướng đến debug. Cộng đồng Intel cũng lưu ý việc phê duyệt tài khoản license có thể mất vài ngày [8].

B.4 Nên sử dụng bản Quartus nào khi xây dựng hệ thống Nios V

Khi làm việc trên Nios II, người dùng có xu hướng dùng bản Quartus 18.x vì tính ổn định, cũng như không cần phải cài thêm WSL1 và tự cài thủ công Eclipse cho Nios II.

Tuy nhiên, trái ngược với Nios II đã ra mắt từ năm 2000, Nios V chỉ mới ra mắt từ năm 2021, và tài liệu đầu tiên của Intel cung cấp cho Nios V là cho nền tảng Quartus 21.3. Lúc này Intel chỉ cung cấp một bản Nios V/m duy nhất. Mãi đến gần giữa 2023 với sự ra mắt của Quartus 23.1 thì đây cũng là lần đầu tiên xuất hiện Nios V/g. Không lâu sau đó thì Nios V/c cũng được ra mắt trên bản Quartus 23.3. Tuy nhiên thì từ đó cho đến hiện tại Intel vẫn liên tục cập nhật các thông tin kỹ thuật, thay đổi các tập lệnh, cũng như đánh giá về mặt phần cứng của Nios V. [10]

Chính vì vậy, cách khuyến khích tốt nhất để dùng Nios V đó là nên sử dụng bản Quartus mới nhất để được cập nhật đầy đủ các tính năng, cũng như sửa lỗi cho trình IDE Ashling, và ưu tiên dùng phiên bản theo thứ tự từ cao đến thấp là Pro, Standard rồi đến Lite Edition. Một số lỗi như lỗi tạo Testbench trên Platform Designer khi làm việc trên hệ điều hành Windows chỉ gặp khi sử dụng bản Lite. Có 2 giải pháp để không bị lỗi này đó là chuyển sang bản Standard hoặc Pro (vẫn dùng Windows), hoặc chuyển sang Linux (vẫn dùng bản Lite).

Ngoài ra cũng cần phải chú ý là từ các bản về sau, bộ Quartus chuyển sang dùng Questa Sim (trước kia là ModelSim). Questa Sim cần phải có License để có thể sử dụng. Các bước lấy License cho Questa Sim cũng tương tự như cho Nios V B.5.

B.5 Lấy Giấy phép (License) Nios V/m

Bộ xử lý Nios V, cụ thể là phiên bản vi điều khiển (microcontroller) Nios V/m được sử dụng trong dự án này, yêu cầu một tệp giấy phép miễn phí (license file) từ Intel để có thể tạo ra các tệp lập trình nạp xuống board ('.sof') trong phần mềm Quartus Prime.

- Đăng ký Tài khoản:** Truy cập Intel FPGA Self-Service Licensing Center (SSLC) <https://licensing.intel.com/psg/s/> (Hình B.3).

2. **Đăng nhập và Chọn Giấy phép:** Sau khi tài khoản được kích hoạt, đăng nhập (Hình B.4) và chọn tùy chọn "Sign up for Evaluation or No-Cost Licenses" (Hình B.5).
3. **Chọn Nios V/m:** Chọn bộ xử lý "Nios V/m" từ danh sách IP/phần mềm có sẵn (Hình B.6).
4. **Thông tin Máy chủ (Host):** Cung cấp chi tiết cho máy tính sẽ sử dụng giấy phép. Chọn "FIXED" cho Loại Giấy phép (License Type) và "NIC ID" cho Loại Máy tính (Computer Type). Nhập Tên Máy tính (Computer Name) mong muốn. Quan trọng là tìm Địa chỉ Vật lý (Physical Address - MAC Address) của thẻ giao diện mạng (network interface card - NIC) của bạn (ví dụ: sử dụng lệnh `ipconfig /all` trong Windows Command Prompt, Hình B.7) và nhập nó vào trường "Primary Computer ID", loại bỏ mọi dấu gạch nối hoặc dấu hai chấm (Hình B.8).
5. **Tạo Giấy phép:** Đồng ý với các điều khoản và tạo giấy phép (Hình B.9).
6. **Nhận và Lưu Giấy phép:** Tập giấy phép (ví dụ: LR-XXXXXXX_License.dat) sẽ được gửi qua email (Hình B.10). Lưu tập 'dat' này vào một vị trí thích hợp trên máy tính của bạn.
7. **Cấu hình Quartus:** Trong Quartus Prime, điều hướng đến Tools -> License Setup... (Hình B.11). Trong cửa sổ License Setup, duyệt đến và chọn tệp giấy phép '.dat' đã tải xuống (Hình B.12). Với giấy phép được thêm thành công, Quartus sẽ có thể tạo các tệp lập trình '.sof' cần thiết trong quá trình biên dịch (compile).

The screenshot shows the Intel® FPGA Self-Service Licensing Center website. At the top, there is a navigation bar with links for PRODUCTS, SUPPORT, SOLUTIONS, DEVELOPERS, PARTNERS, and FOUNDRY. On the right side of the navigation bar are icons for user profile, language selection (ENGLISH), and a search bar labeled "Search Intel.com". Below the navigation bar, the title "Intel® FPGA Self-Service Licensing Center" is displayed. A dark blue header bar contains the text "Home". The main content area is divided into two sections: "Sign In" on the left and "Enroll" on the right. The "Sign In" section includes links for "Already enrolled? - Sign In here", "Contact customer support", and "Frequently Asked Questions". The "Enroll" section includes a link for "Enroll for Intel® FPGA Self Service Licensing Center (SSLC)". At the bottom of the page, there is a footer with links for Company Overview, Contact Intel, Newsroom, Investors, Careers, Corporate Responsibility, Inclusion, Public Policy, and a Site Map. The Intel logo is also present in the footer.

Hình B.3: Trang đăng nhập Cổng Intel Licensing Portal.

Intel Corporation-NoReply  wsm-postmaster@intel.com

To btdat2506@gmail.com <btdat2506@gmail.com>      More 

1/18/2024, 11:36 AM

Welcome to PSG Licensing

Welcome to Intel® FPGA Self-Service Licensing Center 

Dat,

Welcome to the Intel® FPGA Self Service Licensing Center (SSLC). You can now view, manage and, where necessary, activate your license entitlements for Intel PSG Software and IP.

The SSLC uses a guided approach to help you perform the basic license entitlement functions. There are also documentation provided for reference.

Please save your log-in information for future reference:

Email Address: btdat2506@gmail.com

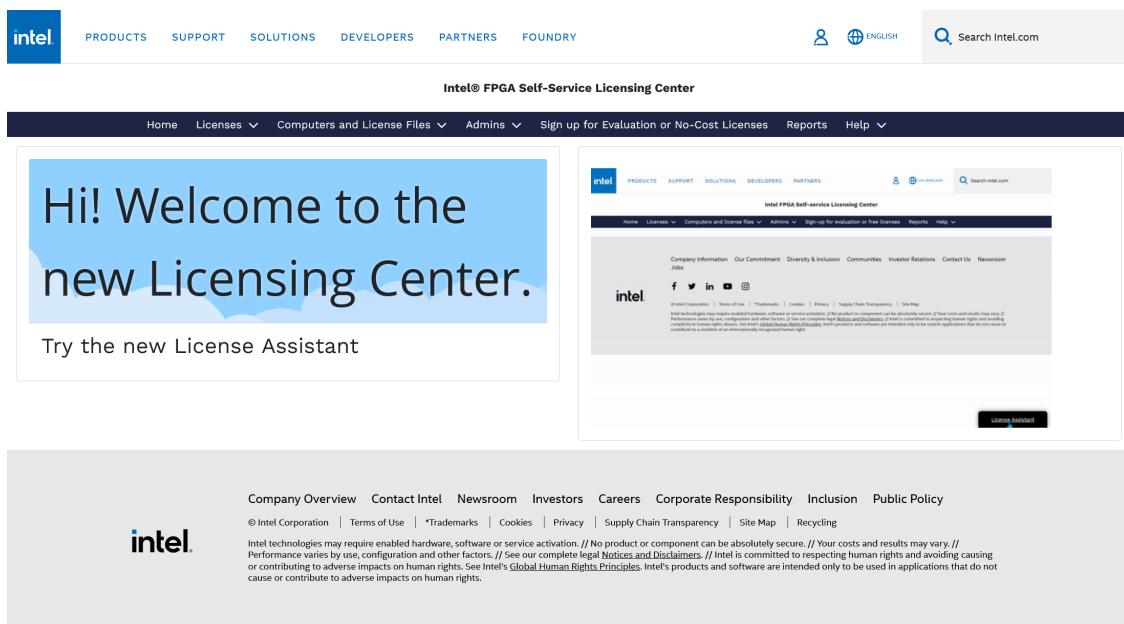
Your password should be protected as confidential. The use of this password on Intel's websites is governed by Intel's Terms and Conditions of Use linked from the bottom of each respective site's web pages.

Log in to licensing.intel.com to view and manage your Intel® PSG Software and IP License Entitlements.

Thank you,
Your Intel® PSG SSLC Team

This message was sent to you as an important business communication related to your FPGA Software and/or IP licensing entitlements with Intel® Programmable Solutions Group.
© 2023 Intel Corporation
Intel Corporation, 2200 Mission College Blvd., M/S RNB4-145, Santa Clara, CA 95054 USA www.intel.com
[Privacy](#) | [Cookies](#)

Hình B.4: Email thông báo tài khoản Intel FPGA Self-Service Licensing đã được tạo thành công.



Hình B.5: Chọn "Sign up for Evaluation or No-Cost Licenses".

<input type="checkbox"/> Intel® Quartus® Prime Software 90-Day Evaluation (Standard and Pro Editions) (License: EVALUATION-LIC)	2025-07-10	2025-07-09
<input type="checkbox"/> Agilex™ 5 E-Series FPGA Software Enablement (License: SW-AGILEX-5E)	2026-04-10	2026-04-10
<input type="checkbox"/> Questa*-Intel® FPGA Starter Edition (License: SW-QUESTA)	2026-04-10	
<input checked="" type="checkbox"/> Nios® V/m Microcontroller Intel® FPGA IP (License: IP-NIOSVM)	2026-04-10	
<input type="checkbox"/> Nios® V/g General Purpose Processor Intel® FPGA IP (License: IP-NIOSVG)	2026-04-10	
<input type="checkbox"/> Nios® V/c Compact Microcontroller Intel® FPGA IP (License: IP-NIOSVC)	2026-04-10	
<input type="checkbox"/> MIPI CSI 2 Intel® FPGA IP (License: IP-MIPI-CSI-2)	2026-04-10	
<input type="checkbox"/> AXI Multichannel DMA for PCI Express (License: IP-PCIECMCDMA-AXI)	2026-04-10	
<input type="checkbox"/> GTS Auto-negotiation/Link Training Feature for Ethernet (License: IP-ETH-ANLT)	2026-04-10	
<input type="checkbox"/> Auto-Negotiation/Link Training Feature F-Tile Hard IP for Ethernet (License: IP-ETH-F-ANLT)	2026-04-10	
<input type="checkbox"/> F-Tile Hard IP for Ethernet, supporting from 10G to 400G Ethernet with optional 1588 PTP feature (License: IP-ETH-FTILEHIP)	2026-04-10	
<input type="checkbox"/> KR/CR (AN/LT) for H-tile Ethernet HIP (100GE) (License: IP-ETH-HTILEKRCR)	2026-04-10	
<input type="checkbox"/> H-tile Ethernet Hard-IP (100GE) (License: IP-ETH-HTILEHIP)	2026-04-10	
<input type="checkbox"/> E-tile Ethernet Hard-IP (10GE/25GE/100GE) (License: IP-ETH-ETILEHIP)	2026-04-10	
<input type="checkbox"/> AXI Multichannel DMA for PCI Express, supporting Agilex 5 (License: IP-PCIECMCDMA-AG5)	2026-04-10	
<input type="checkbox"/> Discontinued - Nios® II/f Processor Intel® FPGA IP (License: IP-NIOS)	2025-07-10	
<input type="checkbox"/> Discontinued - MAX+PLUS® II Software License for Student and University Members (License: PLS-WEB)	2026-04-10	
<input type="checkbox"/> Discontinued - Intel® Quartus® II Software (License: SW-QUARTUS-WE-FIX)	2026-04-10	
<input type="checkbox"/> Discontinued - MAX+PLUS® II Software (License: MAXPLUS2WEB)	2026-04-10	

* # of Seats Next

Hình B.6: Chọn bộ xử lý Nios V/m để cấp phép License.

```

Windows PowerShell

Ethernet adapter vEthernet (Default Switch):

Connection-specific DNS Suffix . . . . . : Hyper-V Virtual Ethernet Adapter
Description . . . . . : Hyper-V Virtual Ethernet Adapter
Physical Address . . . . . : 00-15-5 [REDACTED]
DHCP Enabled . . . . . : No
Autoconfiguration Enabled . . . . . : Yes
Link-local IPv6 Address . . . . . : fe80::deee:5534:d67a:669b%43(PREFERRED)
IPv4 Address . . . . . : 172.22.176.1(PREFERRED)
Subnet Mask . . . . . : 255.255.240.0
Default Gateway . . . . . :
DHCPv6 IAID . . . . . : 721425757
DHCPv6 Client DUID. . . . . : 00-01-00-01-2F-7D-5B-B0-0E-9A-43-B8-9A-B1
NetBIOS over Tcpip. . . . . : Disabled

```

Hình B.7: Tìm ID Card Giao diện Mạng (Network Interface Card - NIC ID) (Địa chỉ Vật lý).

Create Computer

An error occurred while trying to update the record. Please try again.

* Computer Name	BTDAT-TEST	* Computer Type	NIC ID
* License Type	FIXED	How to find your hardware information (NIC/Host/Guard ID)?	
Companion Computer ID 1		* Primary Computer ID	00155D
Primary Admin	Dat Thanh	Companion Computer ID 2	

Hình B.8: Nhập thông tin máy tính (Tên, Loại, NIC ID) cho giấy phép.

Intel® FPGA Self-Service Licensing Center

Home Licenses Computers and License Files Admins Sign up for Evaluation or No-Cost Licenses Reports Help

Add Host & Generate License

* Generate License (Create a New Computer Or Choose an Existing Computer)

Choose an Existing Computer 00155D BTDAT-TEST
 [REDACTED]
 [REDACTED]

Create a New Computer

I have read and agree to the terms of use of this license as listed below

Maintenance for this license is valid for 12 months from the date you sign up for this license.

Check this box if you don't want Intel to contact you for feedback. Your feedback helps us improve the product.

Hình B.9: Tạo giấy phép cố định (fixed license) dựa trên chi tiết máy tính.

Your Intel FPGA Software and IP License  Inbox x

 **authorization@intel.com** <authorization@intel.com>
to me ▾

🕒 19:15 (0 minutes ago)    

Dear btdat2506@gmail.com

Thank you for choosing Intel. Your Perpetual license request has been processed. The attached license file will enable your Intel® FPGA software and/or intellectual property(IP) entitlement(s). If you need more information about the type of license you have received, please visit www.intel.com.

Product Details:

Products	:	Intel® FPGA IP IP-NIOSVM
Primary Machine	:	BTDAT-TEST
Primary Machine ID	:	00155D[REDACTED]
Host Type	:	NIC ID
Companion Host ID	:	N/A
Companion Host ID 2	:	N/A
Redundant Server#2	:	N/A
Redundant Server#3	:	N/A

1.0 License Installation Instructions

a. Save the attached license file to your computer's hard drive. Intel recommends saving the file with a ".dat" file extension.

Note: If no license file is attached to this email, refer to "2.0 Getting your License File from Intel® FPGA Self-Service Licensing Centre" for more information.

b. Use the appropriate method below (1.1 or 1.2) to set up your License.

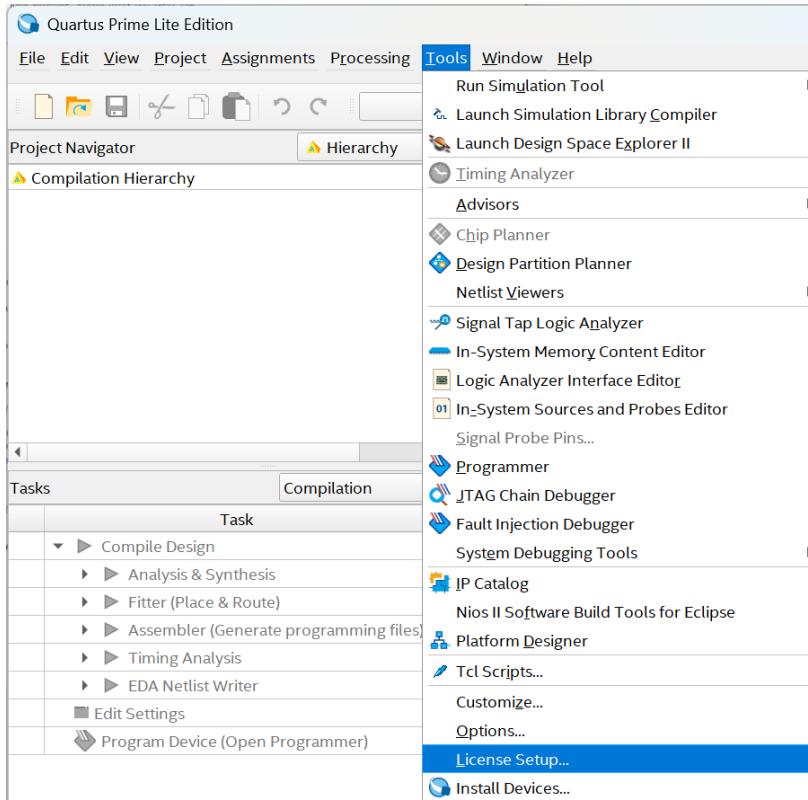
1.1 Setting Up Fixed Node License, Specifying your License File within Intel® Quartus® Prime Software
You can find instructions for setting up Fixed License here.
➤ <https://www.intel.com/content/www/us/en/docs/programmable/683472/current/set-up-a-fixed-license.html>

Note: Please be aware that when using third-party EDA tools like Questa*-Intel® FPGA Edition and ModelSim*-Intel® FPGA Edition software, the license configuration process differs. You will need to establish an environment variable (`LM_LICENSE_FILE`) that directs/point to the license's location.
➤ [Specifying the License for the Questa*-Intel® FPGA Edition Software](#)

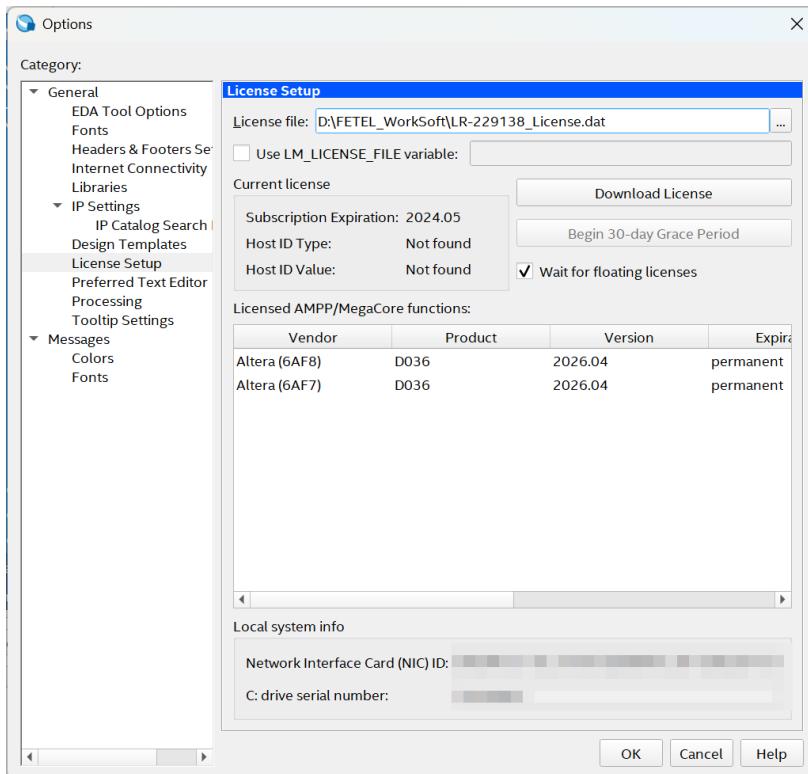
1.2 Setting Up Floating Licenses
You can find instructions for setting up Floating License servers here.
➤ <https://www.intel.com/content/www/us/en/docs/programmable/683472/current/set-up-a-license-file-in-the-license-server.html>

2.0 Getting Your License File from Intel® FPGA Self Service Licensing Center
1. Get started by visiting <https://licensing.intel.com>.

Hình B.10: Email xác nhận chứa tệp giấy phép đính kèm.



Hình B.11: Truy cập Cài đặt Giấy phép (License Setup) từ menu Công cụ (Tools) của Quartus.



Hình B.12: Cửa sổ Cài đặt Giấy phép Quartus (Quartus License Setup) hiển thị tệp giấy phép Nios V/m đã thêm.

B.6 Quy trình Generate Testbench System trên Linux và Chạy Mô phỏng Nios V dùng Questa Sim trên Windows

1. **Tạo Testbench System:** Như đã đề cập, bước này cần thực hiện trên Linux dùng Platform Designer. Chọn Generate -> Generate Testbench System...
2. **Chuẩn bị Thư mục Mô phỏng trên Windows:** Chuyển thư mục project từ Linux về Windows (giả sử đường dẫn là D:/FETEL_WorkDir/Y4S2_DMANiosVIntern_Ubuntu).
3. Trong đường dẫn ./system/testbench/mentor có file msim_setup.tcl. Ta thay đoạn code B.1 thành B.5 để thay đổi đường dẫn đến thư viện Quartus Prime.

```
1 if ! [info exists QUARTUS_INSTALL_DIR] {  
2     set QUARTUS_INSTALL_DIR "/home/btdat/intelFPGA_lite/24.1std/  
3     quartus/"  
}
```

Listing B.1: msim_setup.tcl gốc

```
1 if ! [info exists QUARTUS_INSTALL_DIR] {  
2     set QUARTUS_INSTALL_DIR "D:/FETEL_WorkSoft/intelFPGA_lite/24  
3     .1std/quartus/"  
}
```

Listing B.2: msim_setup.tcl đã chỉnh sửa

4. **Biên dịch Phần mềm và Tạo file .hex:** Tiếp theo, ta chạy lệnh sau B.3 để build HEX file cho 2 Memory. Việc cần build này là vì trong file CMakeLists.txt có đoạn code B.4. Đoạn code này sẽ generate ra 2 file HEX tương ứng cho 2 On-chip Memory. Lưu ý: Trước khi chạy lệnh, nếu ta muốn mô phỏng nhanh hơn, ta nên thay #define arg_simulation 1 thành 0 trong đoạn code C. Việc thay này sẽ bỏ qua các bước kiểm tra hệ thống, từ đó mô phỏng nhanh hơn.

```
1 cd D:\FETEL_WorkDir\Y4S2_DMANiosVIntern_Ubuntu\software\app\build\  
2 Debug  
2 cmake --build .
```

Listing B.3: Chạy CMake để build HEX file cho 2 Memory

```
1 # Generate HEX file(s) from app.elf using elf2hex tool.  
2 # Note : If ECC Full is enabled, width of 39 is set for NiosV TCM.  
3 Otherwise, 32.
```

```

3 add_custom_command(
4     OUTPUT "onchip_memory2_1.hex"
5     DEPENDS app.elf
6     COMMAND elf2hex app.elf -o onchip_memory2_1.hex -b 0x00000000
7     -w 32 -e 0x0001FFFF -r 4
8     COMMENT "Creating onchip_memory2_1.hex."
9     VERBATIM
10 )
11 add_custom_command(
12     OUTPUT "onchip_memory2_0.hex"
13     DEPENDS app.elf
14     COMMAND elf2hex app.elf -o onchip_memory2_0.hex -b 0x00020000
15     -w 32 -e 0x0003FFFF -r 4
16     COMMENT "Creating onchip_memory2_0.hex."
17     VERBATIM
18 )
19 add_custom_target(create-hex ALL DEPENDS "onchip_memory2_0.hex" "
20                   onchip_memory2_1.hex")

```

Listing B.4: CMakeLists.txt

Lưu ý: Các địa chỉ base (-base) và end (-end) phải khớp với địa chỉ của các khối onchip_memory2_0 và onchip_memory2_1 trong hệ thống Platform Designer. Các tệp .hex này sẽ được tự động đọc bởi testbench khi mô phỏng.

5. **Sao chép Tệp HEX:** Sau khi build, ta được hai tệp onchip_memory2_0.hex và onchip_memory2_1.hex trong thư mục build. Có hai cách để xử lý các tệp này:

Cách 1:

- Xóa dòng file_copy ở cuối tệp msim_setup.tcl
- Sao chép hai tệp HEX về thư mục D:/FETEL_WorkDir/Y4S2_DMANiosVIntern_Ubuntu
- Đổi tên thành system_onchip_memory2_0.hex và system_onchip_memory2_1.hex

Giải thích: Điều này hoạt động vì msim_setup.tcl có đoạn lệnh sao chép tệp sau:

```

1 alias file_copy {
2     echo "[exec] file_copy"
3     file copy -force $QSYS_SIMDIR/system_tb/simulation/submodules/
4         csr_mlab.mif ./

```

```

4   file copy -force $QSYS_SIMDIR/system_tb/simulation/submodules/
debug_rom.mif ./
5   file copy -force $QSYS_SIMDIR/system_tb/simulation/submodules/
system_onchip_memory2_1.hex ./
6   file copy -force $QSYS_SIMDIR/system_tb/simulation/submodules/
system_onchip_memory2_0.hex ./
7 }
8

```

Cách 2:

- Vào thư mục D:/FETEL_WorkDir/Y4S2_DMANiosVIntern_Ubuntu /system/testbench/system_tb/simulation/submodules
- Xóa hai tệp system_onchip_memory2_0.hex và system_onchip_memory2_1.hex hiện có
- Sao chép các tệp HEX mới vào và đổi tên lại thành tên tệp gốc (có chữ system_ ở đầu).

6. Chạy Mô phỏng bằng QuestaSim:

- Mở QuestaSim.
- Trong cửa sổ Transcript, thay đổi thư mục làm việc đến thư mục simulation của testbench, và thực thi tệp script cài đặt mô phỏng. Script này sẽ biên dịch các tệp Verilog/SystemVerilog cần thiết (bao gồm mã nguồn DMA, các IP hệ thống, và testbench) và nạp thiết kế.

```

1 cd D:/FETEL_WorkDir/Y4S2_DMANiosVIntern_Ubuntu/system/
      testbench/mentor
2 do msim_setup.tcl

```

Listing B.5: Thiết lập môi trường mô phỏng bằng QuestaSim

- Nạp thiết kế và kích hoạt gõ lỗi, và thêm các tín hiệu cần quan sát vào cửa sổ Wave, và chạy mô phỏng:

```

1 ld_debug
2
3 add wave -divider "Control Slave"
4 add wave -position end sim:/system_tb/system_inst/
      dma_controller/u_CONTROL_SLAVE/*
5 add wave -divider "READ_MASTER"

```

```

6 add wave -position end sim:/system_tb/system_inst/
    dma_controller/u_READ_MASTER/*
7 add wave -divider "WRITE_MASTER"
8 add wave -position end sim:/system_tb/system_inst/
    dma_controller/u_WRITE_MASTER/*
9 add wave -divider "FIFO"
10 add wave -position end sim:/system_tb/system_inst/
    dma_controller/FIFO_IP_inst/*
11 add wave -divider "onchip_memory2_1"
12 add wave -position end sim:/system_tb/system_inst/
    onchip_memory2_1/*
13
14 run 7 ms

```

Listing B.6: Biên dịch và nạp các thiết kế, thêm hiển thị dạng sóng các tín hiệu, và chạy mô phỏng

Ghi chú: Có thể dùng lệnh sau để đặt điểm dừng (breakpoint):

```

1 when {sim:/system_tb/system_inst/onchip_memory2_1/address =
2     32'h800} {stop}

```

B.7 Signal Tap chỉ thu thập được một phần nhỏ tín hiệu

Do tín hiệu DMA khá dài, nên khi sử dụng Signal Tap, ta cần tăng số lượng mẫu (sample) để thu thập được tín hiệu đầy đủ. Ngoài ra, cũng cần phải đặt Trigger Conditions và quan sát lần lượt từng tín hiệu. Nếu đặt quá nhiều Trigger Conditions, Signal Tap sẽ không thể thu thập được tín hiệu đầy đủ.