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

Bùi Thành Đạt - 21207001

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

BÁO CÁO THỰC TẬP THỰC TẾ
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 CÁC HÌNH **iii**

DANH SÁCH CÁC BẢNG **vii**

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 9

2.5.1 Mục tiêu và Bối cảnh: Truyền dữ liệu giữa Bộ nhớ On-Chip . . 9

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`) 12

2.5.6 Tích hợp và Tín hiệu Giao tiếp 13

2.5.7 Tín hiệu giao tiếp của DMA Controller 13

2.5.8 Hiện thực hóa Giao diện Avalon-MM trong DMA Controller . . 16

3 TRIỂN KHAI VÀ KIỂM THỬ HỆ THỐNG NIOS V VỚI DMA **19**

3.1 Xây dựng Hệ thống Phần cứng trên Quartus Prime 19

3.2 Phát triển Ứng dụng Phần mềm Nios V 33

4 Mô phỏng Hệ thống Nios V và DMA **47**

4.1 Công cụ Mô phỏng và Yêu cầu Môi trường 47

4.2 Quy trình Thiết lập và Chạy Mô phỏng 48

4.3 Gỡ lỗi Phần cứng với Signal Tap 51

TÀI LIỆU THAM KHẢO **53**

A MÃ NGUỒN CỦA THIẾT KẾ **55**

A.1 Mã Verilog Top-Level cho DMANiosV 55
A.2 Mã Verilog DMA Controller 55
A.3 Mã Verilog Control Slave 57
A.4 Mã Verilog Read Master 60
A.5 Mã Verilog Write Master 63
A.6 Mã nguồn C kiểm thử DMA 68

B MỘT SỐ LỖI THƯỜNG GẶP **77**

B.1 Lỗi Tạo BSP hoặc Biên dịch Ứng dụng 77
B.2 Sự cố Kết nối JTAG và Debug 77
B.3 Không có file '.sof' để nạp xuống board 78
B.4 Nên sử dụng bản Quartus nào khi xây dựng hệ thống Nios V 79
B.5 Lấy Giấy phép (License) Nios V/m 79

DANH SÁCH CÁC HÌNH

2.1 Kiến trúc Nios V/m [**niosv_architecture**]. 4
2.2 Các giao dịch đọc và ghi Avalon-MM điển hình với tín hiệu waitrequest [1]. 6
2.3 Giản đồ sóng đọc và ghi của giao tiếp Avalon-MM Slave. 6
2.4 Giản đồ sóng đọc và ghi của giao tiếp Avalon-MM Slave. 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 [3]. 8
2.6 Giản đồ sóng đọc và ghi bộ nhớ Memory qua giao tiếp Avalon-MM. . . 8
2.7 Sơ đồ khối tổng thể của bộ điều khiển DMA. 10
2.8 Sơ đồ tuần tự hoạt động của DMA. 14
2.9 Sơ đồ trạng thái của module `CONTROL_SLAVE`. 17
2.10 Sơ đồ trạng thái các module. 18

3.1 Quartus: New Project Wizard: Chỉ định đường dẫn thư mục và tên Project. 22
3.2 Quartus: New Project Wizard: Device Selector 22
3.3 Thư mục dự án hiển thị các tệp Verilog DMA tùy chỉnh đã sao chép. . 23
3.4 Cấu hình IP Nios V/m: Bật "Enable Reset from Debug Module". . . . 23
3.5 Cấu hình IP Bộ nhớ Trên Chip: Đặt kích thước thành 128 KB. 24
3.6 Cấu hình IP JTAG UART. 25
3.7 Component Editor: Các tệp sau khi Phân tích và Sao chép từ Tổng hợp. 26
3.8 Component Editor: Các loại tín hiệu clock và reset đã được sửa. 27
3.9 Component Editor: Chế độ xem sơ đồ khối giao diện Avalon Slave. . . 28
3.10 Component Editor: Chế độ xem giao diện Reset và Clock sink. 28
3.11 Platform Designer: Hệ thống có kết nối giữa `instruction_master` với s1 (của `onchip_memory2_1`). 29
3.12 Platform Designer: Gán Địa chỉ Cơ sở (Assign Base Addresses). 29
3.13 Platform Designer: Mô hình hệ thống hoàn chỉnh (sau khi ngắt kết nối `instruction_master` với `s1` trên `onchip_memory2_1`). 30
3.14 Cấu hình IP Nios V/m: Đặt Bộ nhớ Vector Reset (Reset Vector Memory). 30
3.15 Platform Designer: Hộp thoại Tạo HDL (Generate HDL). 31
3.16 Quartus: Menu Project -> Add/Remove Files in Project... 31
3.17 Quartus: Cài đặt Dự án - Cửa sổ Tệp hiển thị tệp .qip đã thêm. 32

3.18 Quartus: Menu Assignments -> Import Assignments... 32
3.19 Quartus: Hộp thoại Nhập Gán chân (Import Assignments) để chọn tệp .qsf. 32
3.20 Quartus: Compile Design thành công. 33
3.21 Quartus: Trình chỉnh sửa Gán chân (Assignment Editor) hiển thị các gán chân đã nhập. 33
3.22 Cấu trúc thư mục Software gồm các thư mục con `app` và `bsp`. 36
3.23 Thư mục `software/app` chứa tệp mã nguồn `source.c`. 36
3.24 Vị trí của `niosv-shell.exe`. 36
3.25 Cửa sổ Nios V Shell. 36
3.26 Đầu ra từ việc thực thi lệnh `niosv-bsp`. 37
3.27 Đầu ra từ việc thực thi lệnh `niosv-app`. 37
3.28 Khởi chạy Ashling RiscFree™ IDE từ Nios V Shell, chọn đường dẫn Workspace. 37
3.29 Màn hình chào mừng Ashling RiscFree™ IDE: Chọn "Create a project...". 38
3.30 Ashling IDE: Chọn C/C++ ⟶ C Project. 38
3.31 Ashling IDE: Chọn CMake driven ⟶ Empty Project. 39
3.32 Ashling IDE: Project Explorer hiển thị dự án 'app'. 39
3.33 Ashling IDE: nhấp chuột phải 'app' ⟶ Build Project. 40
3.34 Ashling IDE: Console thông báo Build thành công. 40
3.35 Quartus Programmer: Nạp tệp .sof xuống board thành công. 41
3.36 Ashling IDE: Nạp Firmware xuống lõi Nios V (Run As ⟶ Ashling RISC-V Hardware Debugging. 41
3.37 Ashling IDE: Cấu hình Gỡ lỗi - Chỉ định ứng dụng C/C++ (.elf). . . . 42
3.38 Ashling IDE: Cấu hình Gỡ lỗi - Cài đặt tab Debugger. 43
3.39 Ashling IDE: Debug Console cho biết đang chờ kết nối (Nhấn nút đỏ để ngừng chạy). 43
3.40 Đầu ra JTAG UART Terminal: Thông báo ban đầu và reset DMA. . . 44
3.41 Đầu ra JTAG UART Terminal: Trạng thái bộ đệm và cấu hình DMA. 45
3.42 Đầu ra JTAG UART Terminal: Xác minh truyền DMA thành công. . . 46

4.1 Generate Testbench System trong Platform Designer trên Ubuntu 22.04.5 thành công. 48
4.2 Dạng sóng mô phỏng cho các tín hiệu DMA và giao dịch Avalon-MM. . 51

B.1 Lỗi niosv-app báo đường dẫn hoặc tệp source code không tồn tại. . . . 77

B.2 Khởi động lại Altera JTAG Server trong Services của Windows. 78
B.3 Trang đăng nhập Cổng Intel Licensing Portal. 81
B.4 Email thông báo tài khoản Intel FPGA Self-Service Licensing đã được tạo thành công. 82
B.5 Chọn "Sign up for Evaluation or No-Cost Licenses". 83
B.6 Chọn bộ xử lý Nios V/m để cấp phép License. 83
B.7 Tìm ID Card Giao diện Mạng (Network Interface Card - NIC ID) (Địa chỉ Vật lý). 84
B.8 Nhập thông tin máy tính (Tên, Loại, NIC ID) cho giấy phép. 84
B.9 Tạo giấy phép cố định (fixed license) dựa trên chi tiết máy tính. 84
B.10 Email xác nhận chứa tệp giấy phép đính kèm. 85
B.11 Truy cập Cài đặt Giấy phép (License Setup) từ menu Công cụ (Tools) của Quartus. 86
B.12 Cửa sổ Cài đặt Giấy phép Quartus (Quartus License Setup) hiển thị tệp giấy phép Nios V/m đã thêm. 86

DANH SÁCH CÁC BẢNG

2.1 Bảng mô tả tín hiệu chính của DMA Controller. 15

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 Lĩnh 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:

1. **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.

2. **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.

3. **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. (`nnhoang@hcmus.edu.vn`)
- Huỳnh Thị Minh Tuyến, 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 [8]. 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.

- **Nios V/m (Microcontroller):** 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, nhấn mạnh vào diện tích logic nhỏ và hiệu quả năng lượng. Nó phù hợp cho các nhiệm vụ điều khiển, quản lý ngoại vi và xử lý cơ bản trong hệ thống nhúng, đáp ứng nhu cầu của một bộ vi xử lý cơ bản.

- **Kiến trúc RISC-V:** Dựa trên ISA RISC-V mã nguồn mở, cung cấp một kiến trúc hiện đại, module hóa và không ràng buộc về bản quyền. Nios V/m triển khai tập lệnh cơ sở RV32I cùng với các mở rộng tiêu chuẩn như M (Integer Multiplication and Division), A (Atomic Instructions), Zicsr (Control and Status Register Access).

Hình 2.1: Kiến trúc Nios V/m [**niosv_architecture**].

- **Pipeline (Tùy chọn):** Nios V/m có thể được cấu hình với pipeline 5 tầng kinh điển (Fetch, Decode, Execute, Memory, Write back) để tăng hiệu suất hoặc không có pipeline (non-pipelined) để tối ưu diện tích. Khi sử dụng pipeline, có thể xảy ra hiện tượng stall (trì hoãn) do phụ thuộc dữ liệu (data dependency) hoặc tranh chấp tài nguyên (resource stall).

2.3 Truy cập Bộ nhớ Trực tiếp (DMA)

Direct Memory Access (DMA) là một tính năng 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 thông lượng 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.

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

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.

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

Hình 2.2: Các giao dịch đọc và ghi Avalon-MM điển hình với tín hiệu waitrequest [1].

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 Slave.

(b) Giản đồ sóng ghi Slave.

Hình 2.3: Giản đồ sóng đọc và ghi của giao tiếp Avalon-MM Slave.

- **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

(a) Giản đồ sóng đọc của giao tiếp Avalon-MM Master.

(b) Giản đồ sóng ghi của giao tiếp Avalon-MM Master.

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

nguyên `address` và `read` cho đến khi `waitrequest` bị hủy kích hoạt (`=0`). Slave trả về `readdata` trong chu kỳ clock mà `read=1` và `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`.
- Giản đồ sóng đọc điển hình được minh họa trong Hình 2.2 (phần Read), Hình 2.3(a), 2.4a, và 2.6(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 2.3(b), 2.4b, và 2.6(b).

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 [3].

(a) Giản đồ sóng đọc Memory.

(b) Giản đồ sóng ghi Memory.

Hình 2.6: Giản đồ sóng đọc và ghi bộ nhớ Memory qua giao tiếp Avalon-MM.

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.

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, 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, như được minh họa trong các giản đồ sóng đọc/ghi bộ nhớ (Hình 2.6). Mặc dù vậy, thiết kế vẫn tuân thủ đầy đủ đặc tả Avalon-MM, 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 khối 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.

Hình 2.7: Sơ đồ khối tổng thể của bộ điều khiển DMA.

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.

2.5.3 Thiết kế Module Điều khiển ('CONTROL_SLAVE')

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

- **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:** Khi `iChipselect=1`, module giải mã `iAddress` từ CPU để xác định thanh ghi nào đang được truy cập cho thao tác đọc (`iRead=1`) hoặc ghi (`iWrite=1`).

  - **Logic đọc/ghi:** Nếu đọc, dữ liệu từ thanh ghi được chọn sẽ được đưa ra `oReaddata`. Nếu ghi, dữ liệu từ `iWritedata` sẽ được ghi vào thanh ghi được chọn (ngoại trừ thanh ghi trạng thái chỉ có thể xóa bit 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).

- **Mã nguồn tham khảo:** Phụ lục A.3.

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 tạo tín hiệu `oRM_read`, `oRM_readaddress`. Nó phản hồi với `iRM_waitrequest` bằng cách giữ nguyên trạng thái và tín hiệu yêu cầu đọc. Nó chỉ chấp nhận dữ liệu `iRM_readdata` khi `iRM_readdatavalid=1`.

  - **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):** Quản lý chu trình đọc: Chờ `Start` (IDLE), kiểm tra FIFO không gần đầy (CHECK_FIFO), gửi yêu cầu đọc Avalon (REQUEST), chờ `waitrequest=0` và sau đó là `readdatavalid=1` (WAIT_DATA), ghi dữ liệu vào FIFO khi hợp lệ (WRITE_FIFO), cập nhật địa chỉ/bộ đếm và quay lại kiểm tra FIFO/gửi yêu cầu mới (WAIT_FIFO), lặp lại cho đến khi đọc đủ `Length` byte.

- **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 tạo tín hiệu `oWM_write=1`, đưa ra địa chỉ ghi `oWM_writeaddress`, dữ liệu ghi `oWM_writedata` (lấy từ

thanh ghi nội bộ chứa dữ liệu đọc từ FIFO), và `oWM_byteenable=4'b1111`. Nó phản hồi với `iWM_waitrequest=1` bằng cách giữ nguyên trạng thái và tất cả tín hiệu đầu ra Avalon.

- **Máy trạng thái hữu hạn (FSM):** Quản lý chu trình ghi: Chờ `Start` (IDLE), kiểm tra FIFO không rỗng (CHECK_FIFO), yêu cầu đọc FIFO (READ_FIFO), chờ dữ liệu FIFO được chốt (WAIT_FIFO_DATA), gửi yêu cầu ghi Avalon (START_WRITE), chờ `waitrequest=0` (WAIT_WRITE_ACK cập nhật địa chỉ/bộ đếm (UPDATE_CNT). Lặp lại từ CHECK_FIFO. Khi hoàn thành (`bytes_remaining <= 4` trong trạng thái UPDATE_CNT), FSM chuyển về IDLE và kích hoạt `WM_done` trong một chu kỳ.

- **Mã nguồn tham khảo:** Phụ lục A.5.

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 diện với CPU (qua Avalon Slave) và với bộ nhớ (qua Avalon Master).

2.5.7 Tín hiệu giao tiếp của DMA Controller

Bảng 2.1 liệt kê các tín hiệu chính của module DMA Controller tùy chỉnh ở mức cao nhất (‘DMAController.v‘), thể hiện giao diện kết nối với CPU và hệ thống bus Avalon.

Hình 2.8: 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_byteenable`	Out	4	Cho phép ghi từng byte (luôn là 4'b1111)

2.5.8 Hiện thực hóa Giao diện Avalon-MM trong DMA Controller

Dựa trên đặc tả Avalon-MM, các giao diện của bộ điều khiển DMA tùy chỉnh được thiết kế như sau:

1. **Giao diện Slave của `CONTROL_SLAVE`:** 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.

   - **Khi đọc (`iRead=1`):** Dữ liệu từ thanh ghi tương ứng (địa chỉ nguồn/đích, độ dài, control, status) được đưa ra `oReaddata`. Hình 2.3a.
   - **Khi ghi (`iWrite=1`):** Dữ liệu `iWritedata` được ghi vào thanh ghi cấu hình (địa chỉ nguồn/đích, độ dài) hoặc bit GO của thanh ghi điều khiển. Ghi 1 vào bit 0 của thanh ghi trạng thái sẽ xóa cờ Done. Hình 2.3b.

2. **Giao diện Master của `READ_MASTER`:** Module chủ động tạo giao dịch đọc, xử lý `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.

3. **Giao diện Master của `WRITE_MASTER`:** Module chủ động tạo giao dịch ghi, xử lý `waitrequest`.

   - Kích hoạt `oWM_write=1`, đưa ra `oWM_writeaddress`, `oWM_writedata`, và `oWM_byteenable = 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. Hình: 2.4b.

Việc thiết kế các giao diện DMA dựa trên và tuân thủ chặt chẽ đặc tả Avalon-MM 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 của module `CONTROL_SLAVE`.

(a) READ_MASTER. (b) WRITE_MASTER.

Hình 2.10: Sơ đồ trạng thái các module.

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 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™ 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". Đảm 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):** Đi 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 [4], [5]. 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

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**	*no attributes*
DMAController.v	DMAC/DMAController.v	**Verilog HDL**	**Top-level File**
FIFO_IP.qip	DMAC/FIFO_IP.qip	**Other**	*no attributes*
FIFO_IP.v	DMAC/FIFO_IP.v	**Verilog HDL**	*no attributes*
FIFO_IP_bb.v	DMAC/FIFO_IP_bb.v	**Verilog HDL**	*no attributes*

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**	*no attributes*
DMAController.v	DMAC/DMAController.v	**Verilog HDL**	*no attributes*
FIFO_IP.qip	DMAC/FIFO_IP.qip	**Other**	*no attributes*
FIFO_IP.v	DMAC/FIFO_IP.v	**Verilog HDL**	*no attributes*
FIFO_IP_bb.v	DMAC/FIFO_IP_bb.v	**Verilog HDL**	*no attributes*

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: Chế độ xem sơ đồ khối giao diện Avalon Slave.

Hình 3.10: Component Editor: Chế 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.

Hình 3.20: Quartus: Compile Design thành công.

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):

```
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):

```
# -c: create BSP, -t=hal: type HAL, --sopcinfo: path to hardware info
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`):

```
# -a: app dir, -b: bsp dir, -s: source dir (relative to app dir)
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:**

- **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:

```
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`.

Hình 3.25: Cửa sổ Nios V Shell.

```
D:\FETEL_WorkSoft\intelFPGA
[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-defaul
ts\bsp-set-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_niosv_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 .rwdata 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++ $\longrightarrow$ 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.

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: Cấu hình Gỡ lỗi - Chỉ định ứng dụng C/C++ (.elf).

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

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

```
D:\FETEL_WorkSoft\intelFPGA

[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:\FETEL_WorkSoft\intelFPGA

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

  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 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. Chương này trình bày quy trình và các công cụ được sử dụng để mô phỏng hệ thống SoC Nios V/m tích hợp bộ điều khiển DMA tùy chỉnh.

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 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 [6]). Có 2 cách để tạo testbench cho hệ thống Nios V:

1. **Sử dụng Quartus Prime Pro:** Phiên bản Pro này có nhiều tính năng hơn, tuy nhiên phiên bản này tốn dung lượng ổ cứng từ 40-145 GB.

2. **Generate Testbench System trên Linux:** Thực hiện cài đặt Quartus Prime Lite/Standard trên hệ điều hành Linux. Sau đó sử dụng Platform Designer để Generate Testbench System cho hệ thống `system.qsys`. Sau khi testbench được tạo trên Linux thành công (hình 4.1), các tệp này có thể được sao chép sang môi trường Windows để thực hiện mô phỏng bằng QuestaSim/ModelSim trên Windows nếu muốn.

Ngoài ra, để mô phỏng hoạt động của bộ xử lý Nios V chạy mã phần mềm, tệp chương trình thực thi (`.elf`) cần được biên dịch và chuyển đổi thành định dạng bộ nhớ (`.hex`) để nạp vào mô hình bộ nhớ trong quá trình mô phỏng.

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

4.2 Quy trình Thiết lập và Chạy Mô phỏng

1. **Tạo Testbench System:** Như đã đề cập, bước này cần thực hiện trên Linux dùng Platform Designer. Chọn File -> Generate Testbench System...

2. **Chuẩn bị Thư mục Mô phỏng trên Windows:**

   - Tạo một thư mục cho mô phỏng trên Windows (ví dụ: `D:\...\DMANiosVIntern_Sim`).
   - Sao chép toàn bộ thư mục `testbench` được tạo bởi Platform Designer trên Linux vào thư mục mô phỏng trên Windows này. Cấu trúc thư mục sẽ tương tự như: `.../testbench/system_tb/simulation/`.

3. **Biên dịch Phần mềm và Tạo file .hex:**

   - Sử dụng Ashling RiscFree™ IDE (chạy từ Nios V Shell) để biên dịch dự án phần mềm C (thư mục `software/app`) như mô tả trong Chương 3. Thao tác này tạo ra tệp `app.elf`.
   - Chạy lệnh sau trong Nios V Shell để chuyển đổi tệp `.elf` thành các tệp `.hex` cho bộ nhớ on-chip được sử dụng trong mô phỏng:

```
elf2hex --input=software/app/app.elf \
```

```
2 --output=system/testbench/system_tb/simulation/submodules/
    system_onchip_memory2_0.hex \
3 --base=0x20000 --end=0x3FFFF --width=32 --little-endian-mem --
    create-lanes=0
4
5 elf2hex --input=software/app/app.elf \
6 --output=system/testbench/system_tb/simulation/submodules/
    system_onchip_memory2_1.hex \
7 --base=0x40000 --end=0x5FFFF --width=32 --little-endian-mem --
    create-lanes=0
8
```

Listing 4.1: Command to generate .hex file from .elf

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.

4. **(Nếu Di chuyển Testbench) Chỉnh sửa tệp TCL và C:**

   - Mở tệp `<project_dir>/system/testbench/system_tb/simulation/msim_setup.`
   - Tìm và thay đổi đường dẫn thư viện QuestaSim từ định dạng Linux (`/`) sang định dạng Windows (\\ hoặc `/`). Ví dụ: Thay đổi `D:/intelFPGA/23.1std/...` thành `D:/intelFPGA/23.1std/...` (thường thì dấu '/' vẫn hoạt động trên Windows trong TCL). Đảm bảo đường dẫn trỏ đến thư mục cài đặt QuestaSim chính xác trên máy Windows.
   - Trong mã nguồn C (`hello_world.c`), nếu có các định nghĩa `#ifdef` dựa trên hệ điều hành (ví dụ: `#ifdef linux`), chúng cần được điều chỉnh hoặc loại bỏ để đảm bảo mã hoạt động đúng trên Windows khi biên dịch cho mô phỏng.

5. **Chạy Mô phỏng bằng QuestaSim:**

   - Mở QuestaSim.
   - Trong cửa sổ Transcript, thay đổi thư mục làm việc đến thư mục `simulation` của testbench:

```
1 cd D:/FETEL_WorkDir/Y4S2_DMANiosVIntern_Sim/testbench/
    system_tb/simulation/
```

```

```

Listing 4.2: Setup QuestaSim environment

- Thực thi tệp script cài đặt mô phỏng:

```
do msim_setup.tcl

```

Listing 4.3: Run QuestaSim setup script

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.

- Nạp thiết kế cấp cao nhất và kích hoạt gỡ lỗi (nếu script chưa làm):

```
ld_debug

```

Listing 4.4: Load design and enable debug

- Thêm các tín hiệu cần quan sát vào cửa sổ Wave:

```
# Example: Add all signals from the DMA controller
add wave -r sim:/system_tb/system_inst/dma_controller_0/*

# Or add specific signals and group them
add wave -divider "Control Slave"
add wave sim:/system_tb/system_inst/dma_controller_0/
    u_CONTROL_SLAVE/*
add wave -divider "Read Master"
add wave sim:/system_tb/system_inst/dma_controller_0/
    u_READ_MASTER/*
# ... (similarly for WRITE_MASTER, FIFO, and Avalon interfaces
    ) ...

```

Listing 4.5: Add signals to Wave window

- Chạy mô phỏng:

```
run -all

```

Listing 4.6: Run simulation

6. **Phân tích Kết quả:** Quan sát các dạng sóng (waveforms) trong cửa sổ Wave để kiểm tra hoạt động của các tín hiệu điều khiển DMA, các giao dịch Avalon-MM, trạng thái FSM, và dữ liệu truyền qua FIFO. Kiểm tra output trong

Hình 4.2: Dạng sóng mô phỏng cho các tín hiệu DMA và giao dịch Avalon-MM.

cửa sổ Transcript (từ các lệnh `printf` trong mã C thông qua mô hình JTAG UART) để xác minh kết quả truyền dữ liệu.

Dạng sóng từ 3228702804 ps tới 3237620332 ps với argsimulationzero

4.3 Gỡ lỗi Phần cứng với Signal Tap

Bên cạnh mô phỏng, công cụ **Signal Tap Logic Analyzer** tích hợp trong Quartus Prime là một phương pháp hiệu quả để gỡ lỗi thiết kế trực tiếp trên phần cứng FPGA. Signal Tap cho phép chọn và quan sát các tín hiệu nội bộ trong thiết kế theo thời gian thực khi hệ thống đang chạy trên bo mạch.

Trong dự án này, Signal Tap có thể được sử dụng để:

- Quan sát trạng thái của các FSM bên trong `READ_MASTER` và `WRITE_MASTER`.
- Theo dõi các giao dịch trên bus Avalon-MM (tín hiệu `read`, `write`, `address`, `readdata`, `writedata`, `waitrequest`, `readdatavalid`).
- Kiểm tra các tín hiệu điều khiển FIFO (`FF_writerequest`, `FF_readrequest`, `FF_full`, `FF_empty`).
- Xác minh tín hiệu `Start` và `WM_done`.

Việc thiết lập Signal Tap bao gồm chọn các tín hiệu cần theo dõi (Nodes to Tap), thiết lập điều kiện kích hoạt (Trigger Conditions), và cấu hình bộ nhớ đệm (Buffer

Depth). Sau khi biên dịch lại thiết kế với cấu hình Signal Tap, có thể sử dụng công cụ Signal Tap trong Quartus để nạp cấu hình và quan sát dạng sóng khi chạy ứng dụng phần mềm trên Nios V. Đây là một công cụ bổ trợ hữu ích cho mô phỏng, đặc biệt khi cần gỡ lỗi các vấn đề liên quan đến thời gian thực hoặc tương tác phần cứng phức tạp không dễ tái tạo trong mô phỏng.

TÀI LIỆU THAM KHẢO

Tiếng Anh

[1] *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.

[2] *Document Revision History for the Nios® V Processor Reference Manual - Nios V Embedded Processor Design Handbook*. 24.3.1. Altera Corporation, An Intel Company. 2025. URL: https://www.intel.com/content/www/us/en/docs/programmable/683632/24-3-1/document-revision-history-for-the-processor.html.

[3] *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.

[4] FPGAcademy. *DE10-Standard Quartus Settings File*. URL: https://fpgacademy.org/Downloads/DE10_Standard.qsf.

[5] FPGAcademy. *FPGAcademy Boards Information*. URL: https://fpgacademy.org/boards.html.

[6] 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.

[7] 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.

[8] Intel Corporation. *Nios V Embedded Processor Design Handbook*. Version 24.3.1. 2024. URL: https://www.intel.com/content/www/us/en/docs/programmable/726952/24-3-1/about-the-embedded-processor.html.

CHƯƠNG A: MÃ NGUỒN CỦA THIẾT KẾ

A.1 Mã Verilog Top-Level cho DMANiosV

```
module DMANiosV(input CLOCK_50, input [0:0] KEY);
  system Nios_system (
    .clk_clk        (CLOCK_50),
    .reset_reset_n(KEY[0])
  );
endmodule
```

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

A.2 Mã Verilog DMA Controller

```
module DMAController (
    input           iClk,
    input           iReset_n,
    // Avalon Slave Interface
    input           iChipselect,
    input           iRead,
    input           iWrite,
    input   [2:0]   iAddress,
    input   [31:0]  iWritedata,
    output  [31:0]  oReaddata,
    // Avalon Read Master Interface
    output          oRM_read,
    output  [31:0]  oRM_readaddress,
    input           iRM_readdatavalid,
    input           iRM_waitrequest,
    input   [31:0]  iRM_readdata,
    // Avalon Write Master Interface
    output          oWM_write,
    output  [31:0]  oWM_writeaddress,
    output  [31:0]  oWM_writedata,
    output  [3:0]   oWM_byteenable, // <<< ADDED PORT
    input           iWM_waitrequest
);
    // Internal Signals
    wire            Start;
    wire  [31:0]    Length;
```

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

```
        .iRM_readdata(iRM_readdata)
    );

    // Instantiate WRITE_MASTER (Use revised version with byteenable)
    WRITE_MASTER u_WRITE_MASTER (
        .iClk(iClk),
        .iReset_n(iReset_n),
        .Start(Start),
        .Length(Length),
        .WM_startaddress(WM_startaddress),
        .FF_empty(FF_empty),
        .FF_readrequest(FF_readrequest),
        .FF_q(FF_q),
        .oWM_write(oWM_write),
        .oWM_writeaddress(oWM_writeaddress),
        .oWM_writedata(oWM_writedata),
        .oWM_byteenable(oWM_byteenable), // <<< CONNECTED PORT
        .iWM_waitrequest(iWM_waitrequest),
        .WM_done(WM_done)
    );

    // Instantiate FIFO
    wire FF_full;
    FIFO_IP FIFO_IP_inst (
        .aclr (~iReset_n),
        .clock ( iClk ),
        .data ( FF_data ),
        .rdreq ( FF_readrequest ),
        .wrreq ( FF_writerequest ),
        .almost_full ( FF_almostfull ),
        .empty ( FF_empty ),
        .full ( FF_full ),
        .q ( FF_q )
  );

endmodule
```

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

A.3 Mã Verilog Control Slave

```
module CONTROL_SLAVE (
```

```
    input           iClk,
    input           iReset_n,
    // Avalon Slave Interface
    input           iChipselect,
    input           iRead,
    input           iWrite,
    input   [2:0]   iAddress,
    input   [31:0]  iWritedata,
    output  [31:0]  oReaddata,
    // DMA Configuration/Control Outputs
    output  [31:0]  RM_startaddress,
    output  [31:0]  WM_startaddress,
    output  [31:0]  Length,
    output          Start,
    // DMA Status Input
    input           WM_done
);
    // Parameter Definitions for Addresses
    localparam ADDR_READADDRESS  = 3'd0;
    localparam ADDR_WRITEADDRESS = 3'd1;
    localparam ADDR_LENGTH       = 3'd2;
    // Address 3'd3 is unused
    localparam ADDR_CONTROL      = 3'd4;
    localparam ADDR_STATUS       = 3'd5;
    // Addresses 3'd6, 3'd7 are unused

    // Register Definitions
    reg [31:0] readaddress_reg;
    reg [31:0] writeaddress_reg;
    reg [31:0] length_reg;
    reg        control_go;      // Internal GO signal, latched
    reg        status_busy;
    reg        status_done;

    // Output Assignments
    assign RM_startaddress = readaddress_reg;
    assign WM_startaddress = writeaddress_reg;
    assign Length          = length_reg;
    assign Start           = control_go; // Start signal follows the latched GO bit

    // 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};
       // Return current GO bit status
56                 ADDR_STATUS:       readdata_reg = {30'd0, status_busy,
   status_done}; // Return Busy and Done status
57                 default:           readdata_reg = 32'd0; // Reads from
   unused addresses return 0
58             endcase
59         end
60     end
61
62     // Register Write and State Logic (Sequential)
63     always @(posedge iClk or negedge iReset_n) begin
64         if (!iReset_n) begin
65             readaddress_reg  <= 32'd0;
66             writeaddress_reg <= 32'd0;
67             length_reg       <= 32'd0;
68             control_go       <= 1'b0;
69             status_busy      <= 1'b0;
70             status_done      <= 1'b0;
71         end else begin
72             // Handle register writes from CPU
73             if (iChipselect && iWrite) begin
74                 case (iAddress)
75                     ADDR_READADDRESS:  readaddress_reg  <= iWritedata;
76                     ADDR_WRITEADDRESS: writeaddress_reg <= iWritedata;
77                     ADDR_LENGTH:       length_reg       <= iWritedata;
78                     ADDR_CONTROL:      control_go       <= iWritedata
   [0]; // Latch the GO bit from CPU
79                     ADDR_STATUS: begin
80                         // Bit 0: Write 1 to clear Done status
```

```
                        if (iWritedata[0]) begin
                            status_done <= 1'b0;
                        end
                        // Other bits are read-only or unused
                    end
                    default: ; // No action for unused addresses
                endcase
            end

            if (control_go && !status_busy && !status_done) begin
                status_busy <= 1'b1;
                status_done <= 1'b0;
            end

            // If the Write Master signals completion THIS cycle
            if (WM_done) begin
                status_busy <= 1'b0;
                status_done <= 1'b1;
                control_go  <= 1'b0; // <<< Clear GO bit automatically on completion
            end

            // If CPU clears GO bit while busy, we might need an abort mechanism (currently ignored)
        end
    end

endmodule
```

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

A.4 Mã Verilog Read Master

```
module READ_MASTER (
    input           iClk,
    input           iReset_n,
    input           Start,
    input   [31:0]  Length,
    input   [31:0]  RM_startaddress,
    input           FF_almostfull,
    output reg      FF_writerequest,
    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,
    WAIT_DATA = 3'b011, WRITE_FIFO = 3'b100, WAIT_FIFO = 3'b101;
18      reg [2:0] current_state, next_state;
19
20      // Internal Registers
21      reg [31:0] bytes_remaining;
22      reg [31:0] total_bytes;
23
24      // Initialization
25      always @(posedge iClk or negedge iReset_n) begin
26          if (!iReset_n) begin
27              current_state     <= IDLE;
28              oRM_read          <= 1'b0;
29              oRM_readaddress   <= 32'd0;
30              FF_writerequest   <= 1'b0;
31              FF_data           <= 32'd0;
32              bytes_remaining   <= 32'd0;
33              total_bytes       <= 32'd0;
34          end else begin
35              current_state <= next_state;
36              FF_writerequest <= 1'b0;
37
38              case (current_state)
39                  IDLE: begin
40                      oRM_read         <= 1'b0;
41                      FF_writerequest <= 1'b0;
42                      if (Start && oRM_readaddress < RM_startaddress +
    Length) begin
43                          bytes_remaining <= Length;
44                          oRM_readaddress <= RM_startaddress;
45                          total_bytes     <= Length;
46                      end
47                  end
48                  CHECK_FIFO: begin
49                      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      end
75
76      // Next State Logic
77      always @(*) begin
78          case (current_state)
79              IDLE: begin
80                  if (Start && oRM_readaddress < RM_startaddress + Length
    && !FF_almostfull)
81                      next_state = CHECK_FIFO;
82                  else
83                      next_state = IDLE;
84              end
85              CHECK_FIFO: begin
86                  if (!FF_almostfull)
87                      next_state = REQUEST;
88                  else
89                      next_state = CHECK_FIFO;
90              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 ==
    RM_startaddress + Length || FF_almostfull)
110                     next_state = IDLE;
111                 else
112                     next_state = REQUEST; */
113                 //next_state = WAIT_FIFO;
114             end
115             WAIT_FIFO: begin
116                 if (bytes_remaining == 32'd0 || oRM_readaddress ==
    RM_startaddress + Length || FF_almostfull)
117                     next_state = IDLE;
118                 else
119                     next_state = REQUEST;
120             end
121             default: next_state = IDLE;
122         endcase
123     end
124 endmodule
```

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

A.5 Mã Verilog Write Master

```
1 module WRITE_MASTER (
```

```
    input           iClk,
    input           iReset_n,
    // Control Inputs
    input           Start,            // From CONTROL_SLAVE
    input   [31:0]  Length,           // Total bytes to write
    input   [31:0]  WM_startaddress,  // Start address for writing
    // FIFO Interface
    input           FF_empty,         // FIFO empty signal
    output reg      FF_readrequest,   // Request to read from FIFO
    input   [31:0]  FF_q,             // Data read from FIFO
    // Avalon Write Master Interface
    output reg      oWM_write,        // Avalon write signal
    output reg [31:0] oWM_writeaddress, // Avalon write address
    output reg [31:0] oWM_writedata,    // Avalon write data
    output reg [3:0]  oWM_byteenable, // Avalon byte enable
    input           iWM_waitrequest, // Avalon wait request
    // Status Output
    output reg      WM_done           // DMA Write Master Done
);
    // State Machine States
    parameter IDLE            = 3'b000;
    parameter CHECK_FIFO      = 3'b001; // Check if data is available in FIFO
    parameter READ_FIFO       = 3'b010; // Request data from FIFO
    parameter WAIT_FIFO_DATA = 3'b011; // Wait for FIFO data (usually available next cycle)
    parameter START_WRITE     = 3'b100; // Assert write, address, data, byteenable
    parameter WAIT_WRITE_ACK = 3'b101; // Wait for waitrequest=0 first time
    parameter UPDATE_CNT      = 3'b111; // Update counters, check completion

    reg [2:0] current_state, next_state;

    // Internal Registers
    reg [31:0] bytes_remaining;
    reg [31:0] current_address;
    reg [31:0] data_to_write; // Register to hold data from FIFO

    // Initialization and State Register
    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
    new transfer
59              end
60              // Set WM_done when transitioning to IDLE from UPDATE_CNT (
    last word processed)
61              if (current_state == UPDATE_CNT && bytes_remaining <= 4)
    begin
62                  WM_done <= 1'b1;
63              end
64              // Latch data from FIFO in WAIT_FIFO_DATA state (uses state
    *before* edge)
65              // Moved this inside the 'else' block for clarity, happens
    concurrently
66              if (current_state == READ_FIFO) begin
67                  data_to_write <= FF_q;
68              end
69          end
70      end
71
72      // Output Logic (Registered Outputs)
73      always @(posedge iClk or negedge iReset_n) begin
74          if (!iReset_n) begin
75              FF_readrequest    <= 1'b0;
```

```
            oWM_write         <= 1'b0;
            oWM_writeaddress <= 32'd0;
            oWM_writedata     <= 32'd0;
            oWM_byteenable    <= 4'b0000;
        end else begin
            // Default assignments for next cycle (registered outputs)
            FF_readrequest <= 1'b0;
            oWM_write       <= 1'b0;
            oWM_byteenable <= 4'b0000;
            // oWM_writeaddress and oWM_writedata hold previous value unless changed

            // Drive outputs based on state (using state *before* edge)
            case (current_state)
                READ_FIFO: begin // State 2
                    FF_readrequest <= 1'b1; // Request data for next cycle
                end
                START_WRITE: begin // State 4
                    oWM_write         <= 1'b1;
                    oWM_writeaddress <= current_address;
                    oWM_writedata     <= FF_q; // Use value latched on *previous* edge
                    oWM_byteenable    <= 4'b1111;
                end
                WAIT_WRITE_ACK: begin // State 5
                    oWM_write         <= 1'b1;            // Keep write asserted
                    oWM_writeaddress <= current_address; // Keep address stable
                    oWM_writedata     <= FF_q;    // Keep data stable
                    oWM_byteenable    <= 4'b1111;       // Keep byteenable asserted
                end
                // In other states, outputs driven low/inactive by defaults above
                default: begin
                end
            endcase
        end
    end

```

```
    // Next State Logic (Combinatorial)
    always @(*) begin
        next_state = current_state; // Default to stay in current state
        if (!Start) begin
            next_state = IDLE; // Reset to IDLE if Start is deasserted
        end else
        case (current_state)
            IDLE: begin // State 0
                if (Start && Length > 0) begin
                    next_state = CHECK_FIFO; // -> 1
                end else begin
                    next_state = IDLE; // -> 0
                end
            end
            CHECK_FIFO: begin // State 1
                if (!FF_empty) begin
                    next_state = READ_FIFO; // -> 2
                end else begin
                    if (bytes_remaining > 0 && Start) begin
                       next_state = CHECK_FIFO; // -> 1
                    end else begin
                       next_state = IDLE; // -> 0
                    end
                end
            end
            READ_FIFO: begin // State 2
                next_state = WAIT_FIFO_DATA; // -> 3
            end
            WAIT_FIFO_DATA: begin // State 3
                 next_state = START_WRITE; // -> 4
            end
            START_WRITE: begin // State 4
                // Immediately move to wait if write is asserted
                next_state = WAIT_WRITE_ACK; // -> 5
            end
            WAIT_WRITE_ACK: begin // State 5
                if (!iWM_waitrequest) begin
                    next_state = UPDATE_CNT; // -> 6 (Write accepted)
                end else begin
                    next_state = WAIT_WRITE_ACK; // -> 5 (Keep waiting)
                end
            end
```

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

```
// Reset Timeout (number of status polls)
#define DMA_RESET_TIMEOUT 10000

// Buffer size
#define BUFFER_SIZE 32

#define arg_simulation 1 // Set to 0 for simulation, 1 for hardware

// Source data buffer (likely placed in onchip_memory2_0 by linker)
// Initialized with values 2 to 33
alt_u32 pdata0[BUFFER_SIZE] = {
    2, 3, 4, 5, 6, 7, 8, 9,
    10, 11, 12, 13, 14, 15, 16, 17,
    18, 19, 20, 21, 22, 23, 24, 25,
    26, 27, 28, 29, 30, 31, 32, 33};

// Destination data buffer pointer - explicitly placed in onchip_memory2_1
// Adding an offset within the memory for clarity/safety
volatile alt_u32 *pdata1 = (alt_u32*) (ONCHIP_MEMORY2_1_BASE + 0x200); // Offset 0x100 within mem 1

// Write to DMA register (using word offset for IOWR)
static inline void dma_write_reg(alt_u32 reg_offset, alt_u32 value)
{
    IOWR(DMA_CONTROLLER_0_BASE, reg_offset, value);
}

// Read from DMA register (using word offset for IORD)
static inline alt_u32 dma_read_reg(alt_u32 reg_offset)
{
    return IORD(DMA_CONTROLLER_0_BASE, reg_offset);
}

// --- NEW FUNCTION: Reset DMA Controller ---
// Attempts to reset the DMA state via software writes
bool reset_dma()
{
    printf("Resetting DMA Controller...\n");

    // 1. Clear the GO bit in the Control Register to stop new transfers
    printf("  Clearing GO bit (Reg %d)...\n", DMA_REG_CONTROL);
```

```
    dma_write_reg(DMA_REG_CONTROL, 0);

    // 2. Clear configuration registers
    printf("  Clearing Address/Length Regs (%d, %d, %d)...\n",
           DMA_REG_READADDRESS, DMA_REG_WRITEADDRESS, DMA_REG_LENGTH);
    dma_write_reg(DMA_REG_READADDRESS, 0);
    dma_write_reg(DMA_REG_WRITEADDRESS, 0);
    dma_write_reg(DMA_REG_LENGTH, 0);

    // 3. Clear the DONE status bit (Write-1-to-clear)
    // Check if DONE is already set before clearing
    if (dma_read_reg(DMA_REG_STATUS) & DMA_STATUS_DONE)
    {
        printf("  Clearing DONE status bit (Reg %d)...\n", DMA_REG_STATUS);
        dma_write_reg(DMA_REG_STATUS, DMA_STATUS_DONE); // Write 1 to DONE bit
    }

    // 4. Wait for the BUSY flag to clear (optional but recommended)
    printf("  Waiting for BUSY bit to clear (Reg %d)...\n", DMA_REG_STATUS);
    int timeout = DMA_RESET_TIMEOUT;
    while ((dma_read_reg(DMA_REG_STATUS) & DMA_STATUS_BUSY) && (timeout > 0))
    {
        timeout--;
        // Optional delay
        // for(volatile int d=0; d<10; d++);
    }

    if (timeout == 0)
    {
        printf("  Error: DMA reset timeout - BUSY bit did not clear!\n");
        return false;
    }

    printf("DMA Reset complete. Final Status: 0x%lx\n", dma_read_reg(DMA_REG_STATUS));
    return true;
}
```

```
// --- End of reset_dma function ---

// Start DMA transfer
void start_dma_transfer(alt_u32 src_addr, alt_u32 dst_addr, alt_u32 length_bytes)
{
    printf("Configuring DMA Regs:\n");
    printf("  Reg %d (Read Addr) : 0x%08lx\n", DMA_REG_READADDRESS, src_addr);
    printf("  Reg %d (Write Addr): 0x%08lx\n", DMA_REG_WRITEADDRESS, dst_addr);
    printf("  Reg %d (Length)    : %lu bytes\n", DMA_REG_LENGTH, length_bytes);
    printf("  Reg %d (Control)   : 0x%08X\n", DMA_REG_CONTROL, (alt_u32)DMA_CONTROL_GO);

    // Write the source and destination addresses
    dma_write_reg(DMA_REG_READADDRESS, src_addr);
    dma_write_reg(DMA_REG_WRITEADDRESS, dst_addr);

    // Write the length (number of bytes to transfer)
    dma_write_reg(DMA_REG_LENGTH, length_bytes);
    dma_write_reg(DMA_REG_CONTROL, DMA_CONTROL_GO);
    printf("DMA GO bit set.\n");
}

// Wait for DMA to complete
bool wait_for_dma_done()
{
    alt_u32 status;
    printf("Polling DMA Status (Reg %d)...\n", DMA_REG_STATUS);
    // Add a timeout to prevent infinite loops
    int timeout = 1000000; // Adjust as needed
    while (timeout > 0)
    {
        status = dma_read_reg(DMA_REG_STATUS);
        // printf("Status: 0x%08lx\n", status); // Debug print
        if (status & DMA_STATUS_DONE)
        {
            printf("DMA DONE bit detected.\n");
            // Optional: Clear the DONE bit after detecting it by writing 1 to it
```

```
            // dma_write_reg(DMA_REG_STATUS, DMA_STATUS_DONE);
            return true;
        }
        timeout--;
    }
    printf("Error: Timeout waiting for DMA DONE bit. Final Status: 0x%lx
   \n", status);
    return false;
}

// Verify DMA transfer
bool verify_dma_transfer(alt_u32 *src, alt_u32 *dst, alt_u32 num_words)
{
    bool success = true;
    printf("Verifying %lu words...\n", num_words);
    for (int i = 0; i < num_words; i++)
    {
        // Use volatile pointers for reading destination to bypass cache
    optimization if needed,
        // although cache flush is the proper method.
        volatile alt_u32 read_data = dst[i];
        if (read_data != src[i])
        {
            printf("Mismatch at index %d: expected 0x%08lX, got 0x%08lX\
   n",
                   i, (alt_u32)src[i], (alt_u32)read_data);
            success = false;
            // return false; // Optionally exit on first mismatch
        }
        else
          printf("index %d = %d\n", i, *(dst + i));
    }
    return success;
}

int main()
{
    alt_u32 src_address = (alt_u32)pdata0;
    alt_u32 dst_address = (alt_u32)pdata1;

    // Transfer size in bytes
    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         // --- Reset DMA at the start ---
182         if (!reset_dma())
183         {
184             printf("DMA failed to reset. Halting.\n");
185             return -1;
186         }
187         // --- DMA should be idle now ---
188
189         // Ensure source data is flushed from cache *before* DMA reads
    it
190         alt_dcache_flush((void *)pdata0, sizeof(pdata0));
191
192         printf("\nSystem Base Addresses:\n");
193         printf("  ONCHIP_MEMORY2_0_BASE: 0x%08lx\n", (alt_u32)
    ONCHIP_MEMORY2_0_BASE);
194         printf("  ONCHIP_MEMORY2_1_BASE: 0x%08lx\n", (alt_u32)
    ONCHIP_MEMORY2_1_BASE);
195         printf("  DMA_CONTROLLER_0_BASE:   0x%08lx\n", (alt_u32)
    DMA_CONTROLLER_0_BASE);
196         printf("Actual pdata0 address: %p\n", pdata0); // Verify source
    location
197         printf("Actual pdata1 address: %p\n", pdata1); // Verify
    destination location
198
199         // Check if DMA is busy before starting a new transfer (shouldn'
    t be after reset)
200         if (dma_read_reg(DMA_REG_STATUS) & DMA_STATUS_BUSY)
201         {
202             printf("Error: DMA is busy even after reset! Status: 0x%lx\n
    ", dma_read_reg(DMA_REG_STATUS));
203             return -1;
204         }
```

```
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,
    BUFFER_SIZE);
211
212         // Print destination buffer content *before* clearing and
    transfer
213         // Note: This read might be from cache if previously accessed
214         printf("\nDestination Buffer Before Clearing (potentially cached
    ):\n");
215         for (int i = 0; i < BUFFER_SIZE; i++)
216         {
217             printf("  index %2d: 0x%08lx\n", i, (alt_u32)pdata1[i]);
218             if ((i + 1) % 8 == 0)
219                 printf("\n"); // Formatting
220         }
221
222         // Reset the destination buffer using CPU writes
223         printf("\nResetting destination buffer via CPU...\n");
224         for (int i = 0; i < BUFFER_SIZE; i++)
225         {
226             pdata1[i] = 0;
227         }
228         // Flush the cache *after* CPU writes zeros to ensure zeros are
    in memory
229         alt_dcache_flush((void *)dst_address, length_bytes);
230         printf("Destination buffer cleared and cache flushed.\n");
231
232         // Optional: Read back immediately after flush to confirm zeros
233         printf("\nDestination Buffer After Reset & Flush (read back):\n"
    );
234         for (int i = 0; i < BUFFER_SIZE; i++)
235         {
236             printf("  index %2d: 0x%08lx\n", i, (alt_u32)pdata1[i]);
237             if ((i + 1) % 8 == 0)
238                 printf("\n"); // Formatting
239         }
240     }
241
```

```c
    // Start the actual DMA
    start_dma_transfer(src_address, dst_address, length_bytes);

    // Wait for DMA to complete
    if (!wait_for_dma_done())
    {
        printf("DMA transfer failed or timed out!\n");
        // Optional: Attempt another reset here?
        // reset_dma();
        return -1;
    }

    // !!! Crucial: Flush cache for destination region BEFORE verification !!!
    printf("\nFlushing D-Cache for destination buffer...\n");
    alt_dcache_flush((void *)dst_address, length_bytes);
    printf("D-Cache flushed.\n");

    // Verify DMA transfer by reading memory (now hopefully coherent)
    printf("\nVerifying DMA transfer...\n");
    if (verify_dma_transfer(pdata0, pdata1, BUFFER_SIZE))
    {
        printf("DMA Transfer Verification Successful!\n");
    }
    else
    {
        printf("DMA Transfer Verification Failed!\n");
    }

    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

hưởng đến debug. Cộng đồng Intel cũng lưu ý việc phê duyệt tài khoản license có thể mất vài ngày [7].

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. [2]

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.

1. **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:** Tệp giấy phép (ví dụ: `LR-XXXXXXXX_License.dat`) sẽ được gửi qua email (Hình B.10). Lưu tệp '.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).

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

Intel Corporation-NoReply
wsm-postmaster@intel.com

Reply | Reply All | Forward | Archive | Junk | Delete | More

To btdat2506@gmail.com <btdat2506@gmail.com> 1/18/2024, 11:36 AM

Welcome to PSG Licensing

Welcome to Intel® FPGA Self-Service Licensing Center

Intel

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".

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

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

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

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

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

Your Intel FPGA Software and IP License Inbox ×

authorization@intel.com <authorization@intel.com> 19:15 (0 minutes ago)
to me

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
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.