

BAN CƠ YẾU CHÍNH PHỦ
HỌC VIỆN KỸ THUẬT MẠT MÃ

-----○○BOOK○○-----

ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC

ĐỀ TÀI

THIẾT KẾ CÁC KHỐI IP GIAO TIẾP NGOẠI VI VÀ CHƯƠNG TRÌNH ỨNG DỤNG CHO VI ĐIỀU KHIỂN MICROBLAZE

Sinh viên thực hiện: Hoàng Minh Hướng

Khoa: DT4

Chuyên ngành: Kỹ thuật Điện tử - Viễn Thông

Mã ngành: 7520207

Người hướng dẫn: Ths. Nguyễn Thanh Ngọc

HÀ NỘI – 2024

i
LỜI CẢM ƠN

Trước tiên em xin gửi lời cảm ơn chân thành tới quý thầy cô trong khoa Điện tử viễn thông và toàn thể các thầy cô của Học viện Kỹ thuật mật mã đã truyền đạt cho em những kiến thức, kinh nghiệm quý báu và tạo điều kiện học tập trong suốt những năm tháng em học tại trường.

Đặc biệt, em xin gửi lời cảm ơn sâu sắc tới Ths. Nguyễn Thanh Ngọc đã nhiệt tình hướng dẫn và cho em mượn các dụng cụ thực hành trong suốt quá trình làm đề tài. Cô cũng đã dành rất nhiều thời gian của mình để định hướng, tư vấn và góp ý trong quá trình em hoàn thiện đồ án này.

Em cũng xin chân thành cảm ơn gia đình, bạn bè, đồng nghiệp đã định hướng, động viên, hỗ trợ và góp ý trong quá trình làm đồ án.

Tuy nhiên với điều kiện thời gian và kiến thức còn hạn chế, báo cáo này của em không thể tránh được những thiếu sót. Em rất mong nhận được sự chỉ bảo, đóng góp ý kiến của các thầy cô và các bạn để bổ sung và nâng cao chất lượng của đồ án.

Em xin chân thành cảm ơn!

Sinh viên thực hiện đồ án

Hoàng Minh Hướng

LỜI CAM ĐOAN

Tôi xin cam đoan bản đồ án này do tôi tự nghiên cứu dưới sự hướng dẫn của Ths. Nguyễn Thanh Ngọc.

Để hoàn thành đồ án này, tôi chỉ sử dụng những tài liệu đã ghi trong mục tài liệu tham khảo, ngoài ra không sử dụng bất cứ tài liệu nào khác mà không được ghi.

Nếu sai, tôi xin chịu mọi hình thức kỷ luật theo quy định của Học viện.

Hà Nội, ngày tháng năm 2024

Sinh viên thực hiện

(Ký tên và ghi rõ họ tên)

Hoàng Minh Hướng

MỤC LỤC

LỜI CẢM ƠN	i
LỜI CAM ĐOAN	ii
MỤC LỤC.....	iii
CÁC KÝ HIỆU, CHỮ VIẾT TẮT	vi
DANH MỤC BẢNG BIỂU.....	vii
DANH MỤC HÌNH VẼ.....	viii
MỞ ĐẦU	1
CHƯƠNG 1 : TỔNG QUAN VỀ ĐỒ ÁN.....	3
1.1 Lý do chọn đề tài	3
1.2 Phạm vi nghiên cứu	3
1.3 Tổng quan về hệ thống nhúng trên FPGA	4
<i>1.3.1 Tổng quan về hệ thống nhúng.....</i>	<i>4</i>
<i>1.3.2 Tổng quan về FPGA</i>	<i>4</i>
<i>1.3.3 Khối IP</i>	<i>5</i>
1.4 Giới thiệu tổng quan về nền tảng FPro	8
CHƯƠNG 2 : GIAO THÚC FPRO BUS	11
2.1 Ánh xạ bộ nhớ vào ra.....	11
2.2 Giới thiệu FPro bus	12
2.3 Thiết kế FPro bus	13
<i>2.3.1 Phân bổ địa chỉ</i>	<i>13</i>
<i>2.3.2 Đặc điểm kỹ thuật</i>	<i>15</i>
2.4 MCS2FPro bridge	16
<i>2.4.1 Xilinx MicroBlaze MCS</i>	<i>16</i>
<i>2.4.2 MicroBlaze MCS I/O bus</i>	<i>17</i>
<i>2.4.3 MCS2FPro bridge.....</i>	<i>19</i>
2.5 Hệ thống thiết bị ngoại vi	20

2.6 Hệ thống xử lý đồ họa.....	22
--------------------------------	----

CHƯƠNG 3 : HỆ THỐNG THIẾT BỊ NGOẠI VI 24

3.1 Thiết kế các khối ngoại vi cho nền tảng FPro	24
3.2 Khối GPO và GPI	25
3.2.1 Thiết kế phần cứng.....	25
3.2.2 Thiết kế trình điều khiển	26
3.3 Khối định thời	27
3.3.1 Thiết kế phần cứng.....	27
3.3.2 Thiết kế trình điều khiển	28
3.4 Khối UART.....	28
3.4.1 Tổng quan về giao thức UART	28
3.4.2 Thiết kế phần cứng.....	30
3.4.3 Thiết kế trình điều khiển	37
3.5 Khối I2C.....	37
3.5.1 Tổng quan về giao thức I2C	37
3.5.2 Thiết kế phần cứng.....	41
3.5.3 Thiết kế trình điều khiển	46

CHƯƠNG 4 : HỆ THỐNG XỬ LÝ ĐỒ HOẠ 47

4.1 Tổng quan về màn hình máy tính	47
4.1.1 Màn RGB	47
4.1.2 Tốc độ làm tươi	48
4.1.3 VGA	48
4.1.4 Giao diện luồng	49
4.2 Khối đồng bộ VGA.....	50
4.2.1 Sơ lược về màn hình CRT	50
4.2.2 Tín hiệu điều khiển và đồng bộ.....	51
4.2.3 Tốc độ truyền điểm ảnh và dữ liệu	53
4.2.4 Thiết kế phần cứng.....	54
4.3 Khối video IP	58

4.3.1 Khối kiểm tra màn hình	59
4.3.2 Spritecore.....	61
4.3.3 Textcore.....	65
4.3.4 Khối bộ đệm khung	68
CHƯƠNG 5 : THIẾT KẾ ÚNG DỤNG CHO HỆ THỐNG	73
5.1 Tổng quan	73
5.2 Lựa chọn các khối IP	73
5.3 Cấu trúc dự án.....	76
5.4 Sơ đồ mạch nối dây.....	76
5.5 Lưu đồ thuật toán.....	77
5.6 Đánh giá hệ thống	79
KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN.....	82
1. Những nội dung đã hoàn thành.....	82
2. Hạn chế của đề tài	82
3. Hướng phát triển	82
TÀI LIỆU THAM KHẢO	83
PHỤ LỤC	84

CÁC KÝ HIỆU, CHỮ VIẾT TẮT

Ký hiệu	Tên tiếng Anh	Tên tiếng Việt
ACK	Acknowledge	Xác nhận
BRAM	Block Random Access Memory	Khối bộ nhớ truy cập ngẫu nhiên
CD	Color Depth	Độ sâu màu
DAC	Digital to Analog Converter	Bộ chuyển đổi tín hiệu số sang tương tự
FIFO	First In First Out	Vào trước ra trước
FPGA	Field-Programmable Gate Array	Vi mạch cấu trúc mảng phần tử có thể lập trình
FPro	Fun and Professional	Vui và chuyên nghiệp
GPI	General Purpose Input	Đầu vào mục đích chung
GPO	General Purpose Output	Đầu ra mục đích chung
HDL	Hardware Description Language	Ngôn ngữ mô tả phần cứng
I2C	Inter-Integrated Circuit	Mạch tích hợp giữa các mô đun
IC	Integrated Circuit	Mạch tích hợp
IP	Intellectual Property	Sở hữu trí tuệ
MMIO	Memory-Mapped Input/Output	Ánh xạ bộ nhớ vào ra
NACK	Not-Acknowledge	Không xác nhận
SCL	Serial Clock	Tín hiệu đồng bộ nối tiếp
SDA	Serial Data	Dữ liệu nối tiếp
UART	Universal Asynchronous Receiver/Transmitter	Bộ nhận truyền dữ liệu nối tiếp không đồng bộ
VGA	Video Graphics Array	Mảng Đồ Họa Video

DANH MỤC BẢNG BIỂU

Bảng 4.1 Thông số tài nguyên Artix-7 FPGA	69
Bảng 5.1 Tài nguyên phần cứng sử dụng trên FPGA.....	80

DANH MỤC HÌNH VẼ

Hình 1.1 Phân loại khói IP	7
Hình 1.2 Kết hợp giữa IP mềm và IP cứng để tạo ra chip cụ thể	7
Hình 1.3 Nền tảng FPro	9
Hình 2.1 Ánh xạ địa chỉ của một hệ thống đơn giản	11
Hình 2.2 Các thanh ghi của khói timer	12
Hình 2.3 Sơ đồ thời gian của FPro bus	15
Hình 2.4 Khởi tạo MicroBlaze MCS trong phần mềm Vivado 2019.2....	17
Hình 2.5 Giản đồ thời gian quá trình đọc/ghi dữ liệu của MCS I/O bus..	18
Hình 2.6 MCS2FPro bridge	19
Hình 2.7 Sơ đồ kết nối của khói hệ thống thiết bị ngoại vi	20
Hình 2.8 Mô-đun <i>mmio_sys</i>	21
Hình 2.9 Sơ đồ khói của hệ thống thiết bị ngoại vi	21
Hình 2.10 Khối điều khiển MMIO	22
Hình 2.11 Hệ thống xử lý đồ họa.....	22
Hình 2.12 Mô-đun hệ thống xử lý đồ họa.....	23
Hình 3.1 Khối GPO.....	25
Hình 3.2 Khối GPI	25
Hình 3.3 Lớp GpoCore và GpiCore.....	26
Hình 3.4 Khối định thời	27
Hình 3.5 Lớp TimerCore	28
Hình 3.6 Quá trình truyền một byte dữ liệu.....	29
Hình 3.7 Sơ đồ khói UART	31
Hình 3.8 Khối UART.....	31
Hình 3.9 Bộ đệm FIFO	32
Hình 3.10 Bộ tạo tốc độ baud	33
Hình 3.11 Mô-đun bộ thu	33
Hình 3.12 Máy trạng thái của bộ thu	34
Hình 3.13 Mô-đun bộ phát.....	34
Hình 3.14 Máy trạng thái của bộ phát	35

Hình 3.15 Giao diện kết nối UART	36
Hình 3.16 Lớp UartCore	37
Hình 3.17 Hai vi điều khiển kết nối với các thiết bị ngoại vi trên cùng một đường truyền I2C	38
Hình 3.18 Cấu trúc của đường truyền I2C.....	38
Hình 3.19 Khuôn dạng dữ liệu của giao thức I2C	39
Hình 3.20 Tín hiệu start, stop, restart	40
Hình 3.21 Giản đồ thời gian khi truyền dữ liệu của I2C master	41
Hình 3.22 Các giai đoạn của quá trình truyền dữ liệu	42
Hình 3.23 Máy trạng thái của khối I2C	43
Hình 3.24 Sơ đồ khói của mạch điều khiển các thanh ghi dịch dữ liệu ...	44
Hình 3.25 Khối I2C.....	44
Hình 3.26 Giao diện kết nối của khối I2C	45
Hình 3.27 Lớp I2cCore	46
Hình 4.1 Hệ trục tọa độ của màn hình máy tính.....	47
Hình 4.2 Màu RGB	48
Hình 4.3 Cổng VGA	48
Hình 4.4 Pmod VGA.....	49
Hình 4.5 Các phương thức kiểm soát luồng trong giao diện luồng.....	49
Hình 4.6 Sơ đồ nguyên lý của màn hình CRT	51
Hình 4.7 Giản đồ thời gian của các tín hiệu điều khiển chùm tia điện tử quét theo chiều ngang	52
Hình 4.8 Giản đồ thời gian của các tín hiệu điều khiển chùm tia điện tử quét theo chiều dọc	53
Hình 4.9 Sơ đồ khói của khối đồng bộ VGA.....	54
Hình 4.10 Mô-đun bộ đệm dòng.....	55
Hình 4.11 Mô-đun VGA sync.....	56
Hình 4.12 Mô-đun bộ đếm khung.....	57
Hình 4.13 Máy trạng thái của mô-đun VGA sync.....	57
Hình 4.14 Mô-đun VGA sync wrapper	58

Hình 4.15 Cấu tạo của khối video IP	59
Hình 4.16 Khung hình được tạo ra từ khói kiểm tra màn hình	60
Hình 4.17 Mô-đun khói kiểm tra màn hình	60
Hình 4.18 Cường độ sáng của các màu trong phô cầu vòng	61
Hình 4.19 Cấu trúc lớp BarTest.....	61
Hình 4.20 Mạch tạo điểm ảnh của sprite core	62
Hình 4.21 Vị trí tương quan giữa tạo độ của điểm ảnh hiện tại và sprite.	62
Hình 4.22 Mô-đun sprite_ram	63
Hình 4.23 Mô-đun sprite core.....	63
Hình 4.24 Mô-đun sprite wrapper.....	64
Hình 4.25 Cấu trúc lớp SpriteCore	64
Hình 4.26 Bitmap của ký tự A	65
Hình 4.27 Sơ đồ khói của textcore.....	66
Hình 4.28 Mô-đun textcore.....	67
Hình 4.29 Cấu trúc lớp TextCore	68
Hình 4.30 Sơ đồ khói mô-đun RAM	69
Hình 4.31 Mô-đun frame_buffer_ram	70
Hình 4.32 Mô-đun frame_buffer_core	70
Hình 4.33 Cấu trúc lớp FrameCore.....	71
Hình 5.1 Trò chơi phá gạch	73
Hình 5.2 Sơ đồ kết nối màn hình LCD, PCF8574 và Arty A7 52T	74
Hình 5.3 Màn hình LCD hiển thị các chế độ	75
Hình 5.4 Dữ liệu được gửi/nhận bởi khói UART	75
Hình 5.5 Kết nối của hệ thống	77
Hình 5.6 Lưu đồ thuật toán của chương trình chính.....	78
Hình 5.7 Lưu đồ thuật toán của trò chơi phá gạch	79
Hình 5.8 Báo cáo về timing	79
Hình 5.9 Đồ thị thể hiện phần trăm sử dụng tài nguyên phần cứng trên FPGA	80
Hình 5.10 Năng lượng sử dụng trên FPGA	81

1 MỞ ĐẦU

Trong bối cảnh phát triển của công nghệ điện tử và viễn thông, các hệ thống nhúng đóng vai trò ngày càng quan trọng trong việc tạo ra những sản phẩm thông minh và tự động hóa. Đặc biệt, sự phát triển của FPGA (Field Programmable Gate Array) đã mở ra những cơ hội mới trong việc xây dựng các hệ thống tùy chỉnh hiệu suất cao. FPGA không chỉ giúp tăng cường hiệu suất xử lý mà còn mang lại tính linh hoạt trong thiết kế, cho phép các nhà phát triển dễ dàng cấu hình lại để đáp ứng các yêu cầu cụ thể của từng ứng dụng.

Vi điều khiển MicroBlaze, với kiến trúc mềm được nhúng trong FPGA của Xilinx, là một giải pháp hiệu quả để hiện thực hóa các hệ thống nhúng phức tạp. MicroBlaze cho phép tích hợp dễ dàng các khối IP giao tiếp ngoại vi để mở rộng chức năng của hệ thống. Việc thiết kế các khối IP ngoại vi không chỉ giúp tối ưu hóa quá trình giao tiếp với các thiết bị ngoại vi mà còn đảm bảo tính ổn định và hiệu quả cho hệ thống tổng thể. Thông qua đó, người thiết kế có thể tùy chỉnh hệ thống phù hợp với nhiều ứng dụng thực tế, từ các hệ thống tự động hóa công nghiệp đến các ứng dụng trong lĩnh vực IoT.

Do nhận thấy tiềm năng to lớn của công nghệ FPGA, Đề tài “*Thiết kế các khối IP giao tiếp ngoại vi và chương trình ứng dụng cho vi điều khiển MicroBlaze*” sẽ tập trung vào việc xây dựng và triển khai các khối IP giúp kết nối vi điều khiển MicroBlaze MCS với các thiết bị ngoại vi khác. Đề tài bao gồm 5 phần chính:

CHƯƠNG 1: TỔNG QUAN VỀ ĐỒ ÁN

CHƯƠNG 2: GIAO THỨC FPRO BUS

CHƯƠNG 3: HỆ THỐNG THIẾT BỊ NGOẠI VI

CHƯƠNG 4: HỆ THỐNG XỬ LÝ ĐỒ HOẠ

CHƯƠNG 5: THIẾT KẾ ỨNG DỤNG CHO HỆ THỐNG

Tuy nhiên, trong quá trình làm đồ án do kiến thức và khả năng trình bày của em còn hạn chế nên không thể tránh khỏi một vài thiếu sót. Do đó em rất mong nhận được những sự góp ý, đánh giá của các thầy cô và các bạn để đồ án này trở nên hoàn thiện hơn.

Em xin chân thành cảm ơn!

CHƯƠNG 1: TỔNG QUAN VỀ ĐỀ TÀI

1.1 Lý do chọn đề tài

Hiện nay, FPGA (Field Programmable Gate Array) ngày càng trở nên quan trọng trong lĩnh vực điện tử và công nghiệp nhúng nhờ vào tính linh hoạt và khả năng tùy biến cao. FPGA cho phép các nhà thiết kế triển khai các giải pháp phần cứng theo yêu cầu, điều chỉnh và nâng cấp hệ thống mà không cần thay đổi phần cứng vật lý. Điều này không chỉ giúp tiết kiệm chi phí mà còn tăng tốc độ phát triển sản phẩm, đồng thời cho phép dễ dàng tích hợp và tối ưu hóa các chức năng phức tạp. Nhận thấy tiềm năng to lớn của FPGA trong việc xây dựng các hệ thống nhúng phức tạp và hiệu năng cao, em quyết định chọn đề tài “Thiết kế các khối IP giao tiếp ngoại vi và chương trình ứng dụng cho vi điều khiển MicroBlaze”. Đề tài này không chỉ giúp em nắm vững kiến thức về thiết kế hệ thống nhúng trên FPGA mà còn cung cấp cơ hội thực hành với các khối IP (Intellectual Property) ngoại vi như UART (Universal Asynchronous Receiver-Transmitter), I2C (Inter-Integrated Circuit), VGA (Video Graphics Array), ... Hơn nữa, việc phát triển các trình điều khiển và chương trình ứng dụng trên vi điều khiển MicroBlaze cũng giúp em tích lũy kinh nghiệm quý báu trong việc lập trình và tối ưu hóa hệ thống nhúng.

1.2 Phạm vi nghiên cứu

Phạm vi nghiên cứu của đề tài “Thiết kế các khối IP giao tiếp ngoại vi và chương trình ứng dụng cho vi điều khiển MicroBlaze” bao gồm việc tìm hiểu kiến trúc và nguyên lý hoạt động của vi điều khiển MicroBlaze và FPGA, đặc biệt là các dòng FPGA của Xilinx. Đề tài tập trung vào các phần sau:

- Thiết kế nền tảng FPro [9] (Fun and Professional) cung cấp các ngoại vi I/O phổ biến được điều khiển bằng một giao thức bus đồng bộ đơn giản. Nền tảng FPro không những có thể được sử dụng cho vi điều khiển MiroBalze mà còn có thể được sử dụng cho nhiều loại vi điều khiển và vi xử lý khác, miễn là bus nội của chúng có thể được ánh xạ sang FPro bus.
- Nghiên cứu và tìm hiểu các chuẩn giao thức phần cứng thông dụng như: UART, I2C, VGA, ... trên bo mạch Arty A7-35T.

- Nghiên cứu, thiết kế các khối ngoại vi cho nền tảng FPro sử dụng ngôn ngữ mô tả phần cứng SystemVerilog.
- Phát triển trình điều khiển bằng ngôn ngữ C/C++ để điều khiển các khối ngoại vi đã thiết kế.
- Phát triển chương trình ứng dụng cho vi điều khiển MicroBlaze sử dụng các khối ngoại vi và trình điều khiển đã tạo.

1.3 Tổng quan về hệ thống nhúng trên FPGA

1.3.1 Tổng quan về hệ thống nhúng

Hệ thống nhúng là một loại hệ thống máy tính chuyên dụng được thiết kế để thực hiện một số chức năng cụ thể. Không giống như các hệ thống máy tính đa dụng như máy tính cá nhân, hệ thống nhúng thường được tối ưu hóa về kích thước, hiệu suất và chi phí để phục vụ một mục đích nhất định trong các thiết bị điện tử, công nghiệp và gia dụng. Các hệ thống nhúng có thể là một phần của hệ thống phức tạp hoặc một thiết bị độc lập, ví dụ như trong các thiết bị như điện thoại thông minh, ô tô, thiết bị y tế, và các thiết bị gia dụng.

Về mặt cấu trúc, hệ thống nhúng thường bao gồm ba thành phần chính: phần cứng (như vi điều khiển, vi xử lý), phần mềm (các chương trình nhúng) và các thiết bị ngoại vi. Việc tích hợp chặt chẽ giữa phần cứng và phần mềm cho phép hệ thống nhúng thực hiện nhiệm vụ với hiệu quả cao. Đặc biệt, hệ thống nhúng trong các ứng dụng thời gian thực đòi hỏi khả năng xử lý chính xác và nhanh chóng để đảm bảo các yêu cầu về độ trễ và hiệu suất.

1.3.2 Tổng quan về FPGA

Field-programmable gate array - FPGA là một loại mạch tích hợp cỡ lớn dùng cấu trúc mảng phần tử logic mà người dùng có thể lập trình được. Chữ field ở đây muốn chỉ đến khả năng tái lập trình từ "bên ngoài" của người sử dụng, không phụ thuộc vào dây chuyền sản xuất phức tạp của nhà máy bán dẫn. Vi mạch FPGA được cấu thành từ các bộ phận:

- Các khối logic cơ bản lập trình được (logic block)

- Hệ thống mạch liên kết lập trình được
- Khối vào/ra (IO Pads)
- Phần tử thiết kế sẵn khác như DSP (Digital Signal Processing) slice, RAM, ROM, nhân vi xử lý...

Thiết kế hay lập trình cho FPGA được thực hiện chủ yếu bằng các ngôn ngữ mô tả phần cứng – HDL (Hardware Description Language) như VHDL, Verilog, SystemVerilog, ... Các hãng sản xuất FPGA lớn như Xilinx, Altera thường cung cấp các gói phần mềm và thiết bị phụ trợ cho quá trình thiết kế. Các gói phần mềm này có khả năng thực hiện tất cả các bước của toàn bộ quy trình thiết kế IC (Integrated Circuit) chuẩn với đầu vào là mã thiết kế trên HDL (còn gọi là mã RTL - Register Transfer Level).

1.3.3 Khối IP

1.3.3.1 Khái niệm

Khối IP (Intellectual Property) hay còn được gọi là khối sở hữu trí tuệ, là một khối logic hoặc dữ liệu được sử dụng để tạo ra chip bán dẫn, FPGA hoặc mạch tích hợp dành riêng cho ứng dụng (ASIC - Application-Specific Integrated Circuits) cho một sản phẩm. Khối IP thường là tài sản của một cá nhân hay một công ty cụ thể nào đó như Intel, Xilinx, AMD,... Khối IP được tạo ra trong suốt quá trình thiết kế và có thể biến thành các thành phần để tái sử dụng. IP của bên thứ ba cũng có thể được mua và tích hợp vào các thiết kế bán dẫn. Như vậy, mỗi cá nhân cũng có thể tự thiết kế, tạo ra, kiểm định và sản xuất những khối IP của riêng mình tùy theo mục đích sử dụng hoặc thương mại bằng cách bán khói IP.

Khối IP với nhiều tính chất cơ bản như: tính bảo hộ về mặt pháp luật trong đó có quyền sở hữu và quyền tác giả; tính nhượng quyền sử dụng và kinh doanh dưới hình thức cấp bản quyền sử dụng bằng sáng chế hoặc bản quyền mã nguồn tồn tại trong thiết kế.

Lý tưởng nhất, khói IP phải hoàn toàn di động - nghĩa là có thể dễ dàng chèn vào bất kỳ công nghệ nhà cung cấp hoặc phương pháp thiết kế nào. Bộ thu /

phát không đồng bộ (UART), bộ xử lý trung tâm (CPU), bộ điều khiển Ethernet là một số ví dụ về khối IP.

Khối IP nào cũng có một datasheet hay một mô tả (description) của nó, thông qua tài liệu này của khói IP nắm được hai điểm quan trọng sau:

- Các tín hiệu giao tiếp
- Các thanh ghi cấu hình và điều khiển khói IP

Từ đó, có thể hiểu được chính xác những khói IP hoạt động như thế nào và làm chức năng gì để có thể sử dụng đúng và tối ưu nhất những khói IP này theo cách mà người thiết kế muốn.

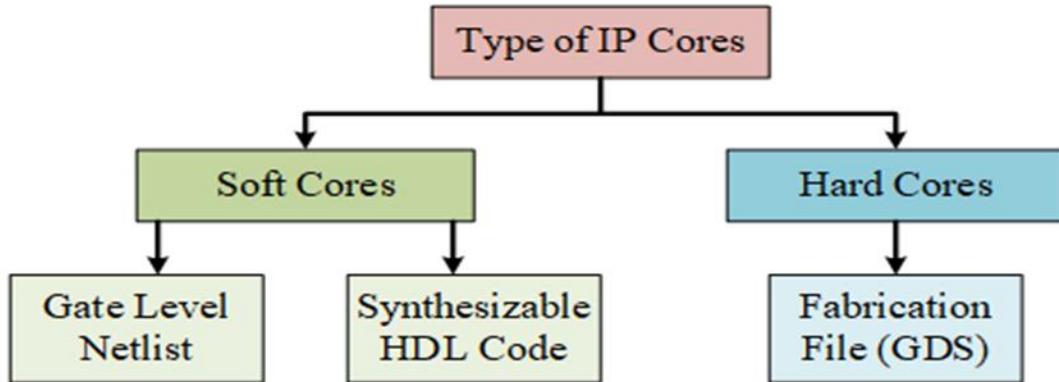
1.3.3.2 Phân loại

Khối IP thường được chia làm 2 loại: IP mềm (Soft IP core) và IP cứng (Hard IP core).

Khối IP mềm thường được cung cấp dưới dạng RTL tổng hợp bằng ngôn ngữ mô tả phần cứng như SystemVerilog hoặc VHDL. Chúng tương tự như các ngôn ngữ cấp thấp như C trong lĩnh vực lập trình máy tính, RTL cho phép các nhà thiết kế chip sửa đổi thiết kế ở cấp độ chức năng. Hoặc các khói IP mềm đôi khi cũng được cung cấp tổng hợp dưới dạng danh sách mạng cấp cổng, hay có thể nói là được biểu diễn tổng hợp như biểu diễn đại số boolean của một hàm logic được triển khai bằng các cổng logic. Ưu điểm của khói IP mềm là chúng có thể được tùy chỉnh trong giai đoạn thiết kế vật lý và được ánh xạ tới bất kỳ công nghệ xử lý nào.

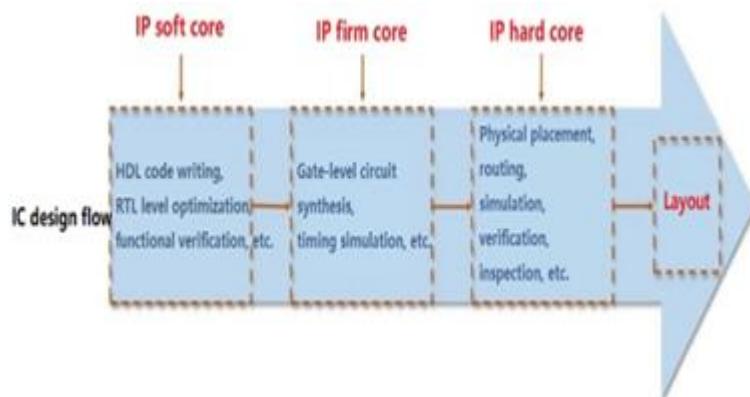
Khối IP cứng chính là thiết kế vật lý của IP, chúng thường được định nghĩa là mô tả vật lý ở cấp độ thấp hơn dành riêng cho một công nghệ xử lý cụ thể. Các khói cứng thường mang lại khả năng dự đoán tốt hơn về hiệu suất thời gian của chip và diện tích cho công nghệ cụ thể của chúng. Khối cứng không có tính di động hoặc linh hoạt, khi thiết kế cốt khói phải định cấu hình kết nối vị trí và giao diện với các mô-đun, đồng hồ và thiết lập lại khác. IP cứng có một vị trí cố định trong FPGA và không thể được chuyển sang các FPGA khác hoặc được tùy chỉnh cho các công nghệ xử lý khác nhau.

Tóm lại, khối IP mềm đơn giản chính là RTL code thực hiện một chức năng nào đó. Khối IP cứng được sinh ra từ khối IP mềm, khối IP sau khi được tổng hợp tạo ra khối IP cứng, khối IP cứng chính là bản layout theo một công nghệ sản xuất chip nào đó và có thể đem đi chế tạo thành chip.



Hình 1.1 Phân loại khối IP

Ưu điểm của IP mềm là tính linh hoạt và di động của nó khi mà có thể tái sử dụng cho nhiều ứng dụng, tuy nhiên, một trong những nhược điểm chính của khối IP mềm là giá thành của nó. Vì mã nguồn có thể sửa đổi nên khối IP mềm có xu hướng đắt hơn khối IP cứng hoặc khối cứng, mặt khác hiệu suất sử dụng cũng khác nhau tùy thuộc vào từng thiết kế khác nhau khi sử dụng nó.



Hình 1.2 Kết hợp giữa IP mềm và IP cứng để tạo ra chip cụ thể

Ưu điểm của khối IP cứng là làm giảm nhu cầu bảo trì mã. IP cứng cũng giảm thiểu vi phạm về thời gian, thúc đẩy hiệu suất và chức năng cao, đồng thời cung cấp tùy chọn khối IP chi phí thấp vì được bao gồm trong các mạch tích hợp hay FPGA. Hạn chế của khối IP cứng là thiếu tính di động và có các ràng buộc

được xác định trước, phù hợp với các sản phẩm logic tín hiệu tương tự và hỗn hợp như ADC, DAC...

1.3.3.3 Ứng dụng và đánh giá

Với việc sử dụng khối IP thì giúp các kỹ sư và nhà thiết kế điện tử sử dụng chúng để triển khai các thành phần logic và IC độc đáo nhanh hơn những gì họ có thể làm. Vì chúng có thể tái sử dụng nhiều lần các thành phần được thiết kế trước đó nên chúng đóng góp cho ngành tự động hóa thiết kế điện tử.

1.3.3.4 Quy trình phát triển khối IP cho hệ thống nhúng trên FPGA

Một tác vụ trong hệ thống nhúng có thể được thực hiện bởi phần mềm, phần cứng hoặc cả hai. Tuỳ theo các yêu cầu về hiệu năng, tài nguyên phần cứng, ... cần phải quyết định xem xử lý tác vụ đó theo cách nào. Khi xử lý tác vụ bằng phần cứng cần phải thực hiện các bước sau:

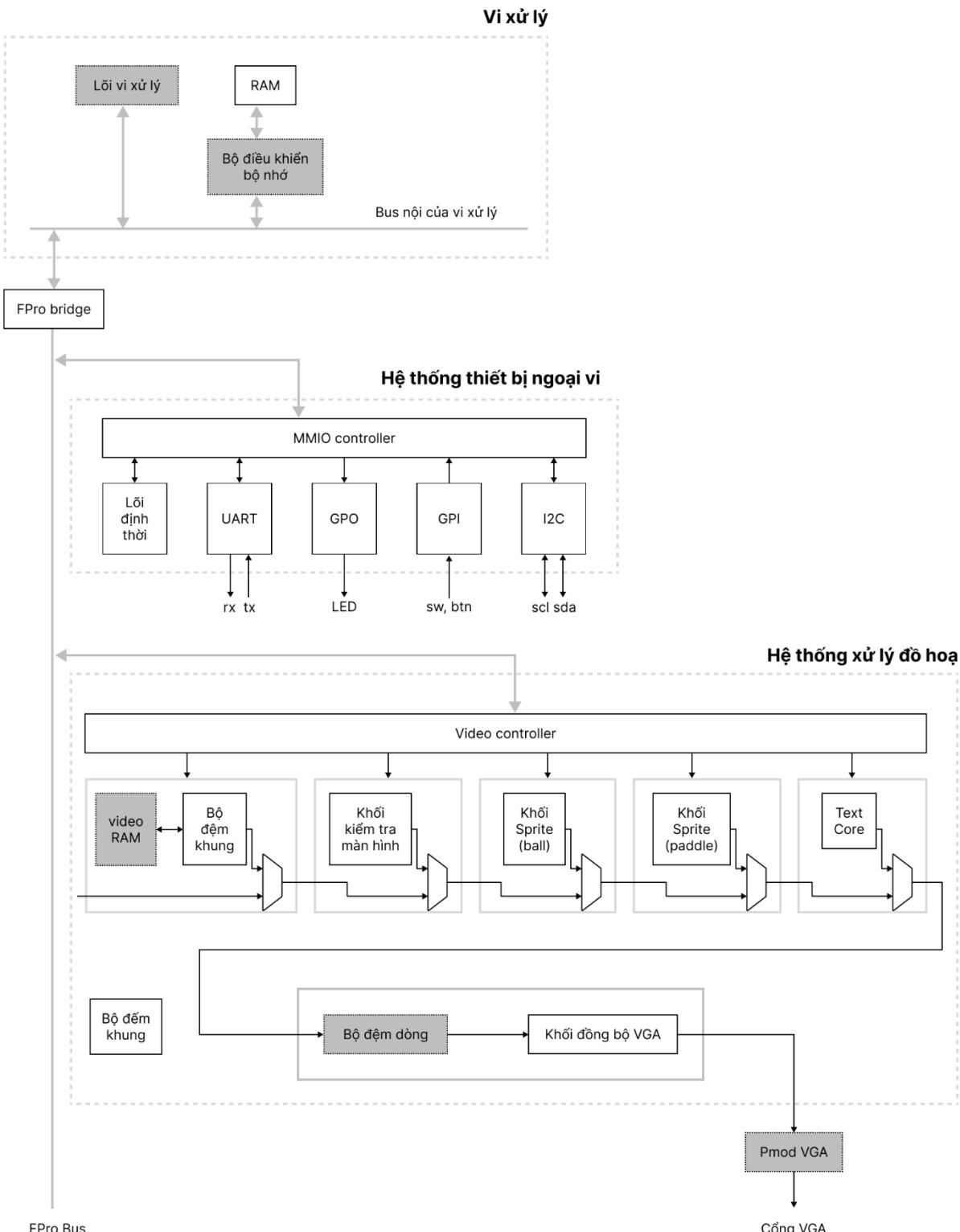
- Thiết kế một mạch logic (khối IP) để thực hiện việc tính toán hoặc thực thi tác vụ đề ra.
- Thiết kế một giao diện để kết khói IP với một bus hoặc các thành phần khác trong hệ thống.
- Viết trình điều khiển bằng ngôn ngữ lập trình C/C++ để điều khiển khói IP bằng phần mềm.

1.4 Giới thiệu tổng quan về nền tảng FPro

Nền tảng FPro bao gồm ba thành phần chính (Hình 1.3):

- FPro bridge và FPro bus. FPro bridge có tác dụng ánh xạ bus của vi xử lý sang FPro bus để kết nối các khói IP trong nền tảng FPro với vi xử lý.
- Hệ thống thiết bị ngoại vi bao gồm:
 - Các khói IP như: khói định thời (timer), UART, GPI, GPO, ...
 - MMIO (Memory-Mapped I/O) controller có tác dụng kết nối tối đa 64 khói IP với vi xử lý thông qua ánh xạ địa chỉ bộ nhớ.

- Hệ thống xử lý đồ họa có tác dụng điều phối hoạt động của các khối video IP để tạo hoặc xử lý luồng dữ liệu gửi đến màn hình thông qua cổng VGA.



Hình 1.3 Nền tảng FPro

Vì một số các khối IP như clock management IP, bộ đếm dòng, ... (được tô màu xám trong Hình 1.3) phụ thuộc vào phần cứng của thiết bị được sử dụng

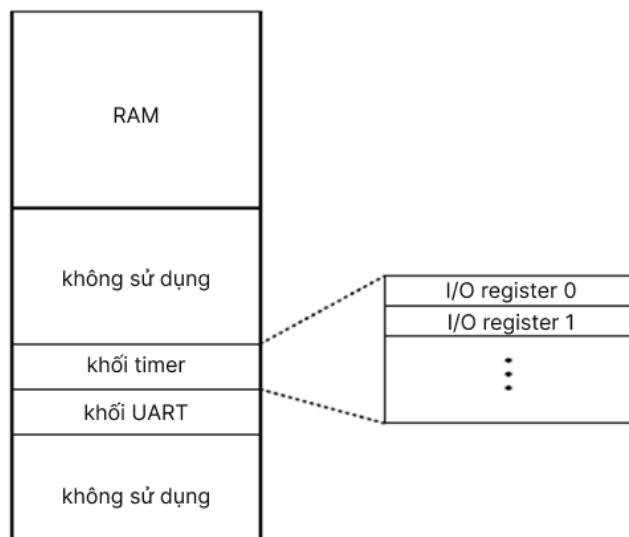
hoặc quá phức tạp để triển khai như khói vi xử lý nên đề tài này sử dụng các khói IP của nhà sản xuất cung cấp. Các IP này rất phổ biến và được hỗ trợ trong hầu hết các loại FPGA do đó đây không phải là một vấn đề quá lớn khi triển khai code ở trên các bo mạch khác nhau. Các khói IP còn lại sẽ được tạo từ đầu bằng ngôn ngữ SystemVerilog và có thể được sử dụng bởi các loại FPGA bất kỳ có mặt ở trên thị trường.

CHƯƠNG 2: GIAO THỨC FPRO BUS

2.1 Ánh xạ bộ nhớ vào ra

Một thiết bị ngoại vi vào ra thường dùng nhiều thanh ghi dùng để chứa các trạng thái và điều khiển. Để điều khiển các thiết bị ngoại vi, vi xử lý có thể ánh xạ các thanh ghi này vào không gian đại chỉ bộ nhớ trong (RAM). Một khi các thanh ghi I/O này được ánh xạ vào bộ nhớ, vi xử lý có thể sử dụng các lệnh đọc và ghi dữ liệu dùng cho bộ nhớ trong để điều khiển hoạt động của các thiết bị ngoại vi.

Hệ thống thường dành một không gian bộ nhớ liên tiếp cho các thanh ghi I/O của thiết bị ngoại vi được ánh xạ. Điều này có nghĩa là địa chỉ của một thanh ghi I/O có gồm có 2 thành phần: địa chỉ cơ sở và độ lệch địa chỉ (offset). Hình 2.1 minh họa cách phân chia không gian địa chỉ của một hệ thống đơn giản gồm các module RAM, timer và UART.



Hình 2.1 Ánh xạ địa chỉ của một hệ thống đơn giản

Địa chỉ bắt đầu của mỗi thiết bị ngoại vi được gọi là địa chỉ cơ sở (base address). Từ địa chỉ cơ sở của một thiết bị ngoại vi, việc truy xuất tới một word (gồm 32 bits dữ liệu) trong RAM hoặc một thanh ghi I/O có thể được thực hiện bằng cách cộng thêm một giá trị offset có giá trị bắt đầu từ 0. Tuy nhiên, không giống như RAM, các thanh ghi I/O của từng thiết bị ngoại vi lại có cấu trúc dữ

liệu và chức năng khác nhau. Ví dụ như trong Hình 2.2, có thể thấy khối timer có 3 thanh ghi với các giá trị offset được thể hiện bởi các số (0, 1, 2) bên trái, và các ký tự bên phải cho biết các thanh ghi này có thể sử dụng để đọc – r hay ghi – w.

	31	...	4	3	2	1	0	
0								counter lower word
1								counter upper word
2							clr	go

Hình 2.2 Các thanh ghi của khói timer

2.2 Giới thiệu FPro bus

Nền tảng FPro sử dụng một bus nội bộ đồng bộ đơn giản gồm các thao tác đọc và ghi. Vì xử lý (master) có thể điều khiển quá trình đọc/ghi với các khối ngoại vi (slaves) thông qua FPro bridge. Ngoài các tín hiệu clock và reset, FPro bus chứa các tín hiệu sau:

- *fp_addr* (master tới slave): bus định địa chỉ theo word (32 bits) có độ rộng 22 bits được sử dụng để chỉ định khối cần đọc/ghi trong nền tảng FPro.
- *fp_rd_data* (slave tới master): chứa dữ liệu đọc từ thanh ghi của slave có độ rộng 32 bits.
- *fp_wr_data* (master tới slave): chứa dữ liệu được gửi từ master để ghi xuống thanh ghi của slave.
- *fp_rd* (master tới slave): tín hiệu có độ rộng 1 bit dùng để điều khiển thao tác đọc.
- *fp_wr* (master tới slave): tín hiệu có độ rộng 1 bit dùng để điều khiển thao tác ghi.
- *fp_mmio_cs* (master tới slave): có độ rộng 1 bit được dùng để chọn chip trong hệ thống thiết bị ngoại vi.

- *fp_video_cs* (master tới slave): có độ rộng 1 bit được dùng để chọn chip trong hệ thống xử lý đồ họa.

2.3 Thiết kế FPro bus

Do Nền tảng FPro có chứa rất nhiều các khối IP, do đó vi xử lý và FPro bridge cần phải có cơ chế giải mã và ghép kênh để truy có thể truy cập và giao tiếp với một khối IP và các thanh ghi của nó.

2.3.1 Phân bổ địa chỉ

Nền tảng FPro sử dụng 22-bit để định địa chỉ theo word. Số lượng địa chỉ lớn như vậy là vì khối đệm khung của hệ thống xử lý đồ họa cần rất nhiều bộ nhớ để lưu trữ thông tin của các điểm ảnh trên màn hình. Đối với vi điều khiển MicroBlaze MCS, không gian địa chỉ của nền tảng FPro chiếm khoảng $\frac{2^{22}}{2^{30}} = \frac{1}{2^8} \approx 0.8\%$. Để đơn giản hóa mạch điều khiển và tạo điều kiện cho việc mở rộng của các khối I/O sau này, nền tảng FPro gán địa chỉ cho các khối IP như sau:

- Dài địa chỉ $0x xxxx xxxx xmmm mmmr rrrr$ được sử dụng cho $2^6 = 64$ khối MMIO I/O, mỗi khối có tối đa $2^5 = 32$ thanh ghi.
- Dài địa chỉ $10 xxxv vvrrr rrrrr rrrrr$ được sử dụng cho $2^3 = 8$ khối video IP, mỗi khối có tối đa 2^{14} thanh ghi.
- Dài địa chỉ $11 rrrr rrrr rrrr rrrr$ được sử dụng cho khối đệm khung với tối đa 2^{20} thanh ghi.

Trong đó:

- *x* đại diện cho “don’t care” bit.
- *r* đại diện cho bit địa chỉ thanh ghi của một khối.
- *m* đại diện cho bit được dùng để xác định khối MMIO I/O.
- *v* đại diện cho bit được dùng để xác định khối video IP.

Với cách chia này, MSB được sử dụng để lựa chọn giữa hệ thống thiết bị ngoại vi và hệ thống xử lý đồ họa. Với hệ thống xử lý đồ họa, bit có trọng số lớn thứ hai được sử dụng để phân biệt các khối video IP và khối bộ đệm khung.

Một bộ vi xử lý 32-bit thường có 32 bits địa chỉ và có thể có 2^{32} địa chỉ nhớ. Mặc dù vi xử lý 32-bit thực hiện tính toán trên từng word dữ liệu, dung lượng trong mỗi không gian địa chỉ nhớ của nó lại được tính theo byte. Nói cách khác, nếu độ rộng của bus địa chỉ là 32 bits thì nó có thể định địa chỉ cho 2^{32} bytes. Để biểu diễn 1 word, FPro bus sẽ kết hợp 4 bytes liên tiếp lại với nhau. Để đảm bảo dữ liệu được truy xuất một cách hiệu quả nhất, FPro bus căn chỉnh địa chỉ đầu tiên của một từ tại địa chỉ là bội của 4. Điều này có nghĩa là 2 chữ số cuối cùng (dạng hex) trong địa chỉ của byte đầu tiên của một word luôn luôn là 0x00. Nếu 2 chữ số cuối cùng của một word khác giá trị này thì khi truy xuất dữ liệu, vi xử lý có thể cần phải thực hiện nhiều lệnh truy xuất bộ nhớ hơn, điều này dẫn đến việc giảm hiệu suất của hệ thống.

Đối với mỗi khối IP được gắn vào hệ thống thiết bị ngoại vi tại vị trí thứ n , base address (địa chỉ byte) của nó có thể được tính từ vị trí của nó theo công thức như sau:

$$\text{bridge_base_address} + n * 32 * 4 \quad (2.1)$$

Địa chỉ cơ sở của các khối video IP có được gắn vào hệ thống xử lý đồ họa ở vị trí thứ n có thể được tính theo công thức sau:

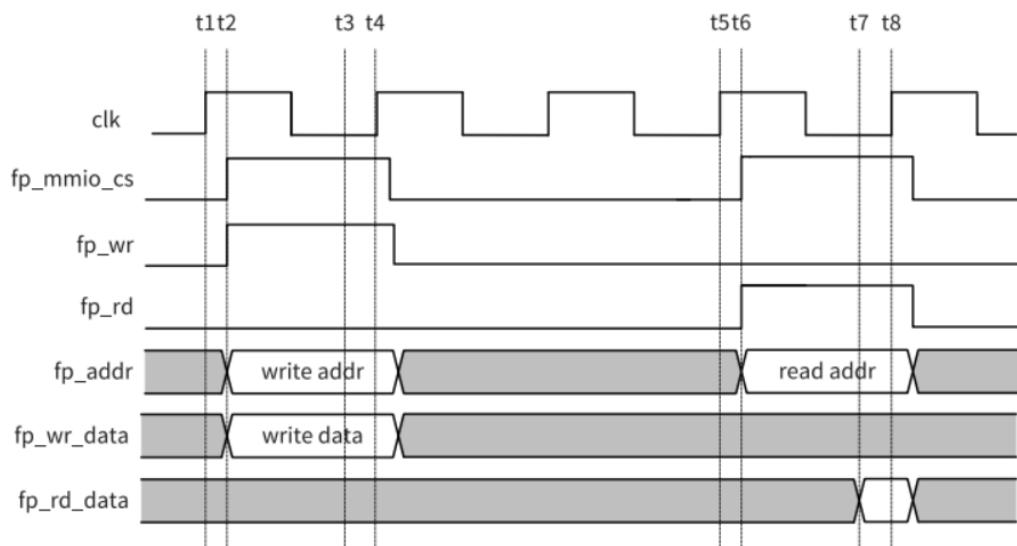
$$\text{bridge_base_address} + 0x800000 + n * 16384 * 4 \quad (2.2)$$

Trong đó giá trị 0x800000 đại diện cho bit thứ 21 trong địa chỉ word (tức là bit thứ 23 trong địa chỉ byte), còn giá trị 16384 tương ứng với 2^{14} thanh ghi 32 bits của mỗi khối video IP.

Cần phải lưu ý rằng độ rộng của đường địa chỉ và dữ liệu trong FPro bus là cố định và một khối IP có thể không dùng hết tất cả các địa chỉ mà nó được cung cấp để định địa chỉ cho các thanh ghi. Để tối ưu không gian địa chỉ nền tảng FPro có thể chia không gian địa chỉ tùy theo nhu cầu sử dụng của mỗi khối. Tuy nhiên, sự khác biệt về các thanh ghi giữa các khối IP làm cho mạch giải mã và ghép kênh cũng như cơ chế định địa chỉ trở nên phức tạp và khó triển khai hơn.

2.3.2 Đặc điểm kỹ thuật

Tất cả các thao tác đọc/ghi của FPro bus đều phải hoàn tất trong 1 chu kỳ xung clock. Ví dụ, thao tác ghi dữ liệu được thể hiện ở phía bên trái của Hình 2.3. Tại sườn dương của tín hiệu clk (t1), vi xử lý đưa ra lệnh ghi dữ liệu tới một địa chỉ của thanh ghi của khối I/O. Các tín hiệu này ổn định tại thời điểm t2 sau khi bị trễ một khoảng thời gian t_{CQ} (clock-to-q) và mạch giải mã bắt đầu hoạt động. Tại thời điểm t3, tín hiệu kích hoạt của mạch giải mã được truyền tới thanh ghi của khối I/O sau khoảng thời gian t_{DEC} . Cuối cùng, tại sườn lên của xung clk tiếp theo (t4) các thanh ghi được chỉ định lấy mẫu và lưu dữ liệu có trên đường bus fp_wr_data .



Hình 2.3 Sơ đồ thời gian của FPro bus

Giả sử chu kỳ hoạt động của hệ thống là t_{CLK} và thời gian thiết lập của thanh ghi là t_{SETUP} , khi đó mạch phải thoả mãn điều ràng buộc sau:

$$t_{CQ} + t_{DEC} + t_{SETUP} < t_{CLK}$$

$$\Leftrightarrow t_{DEC} < t_{CLK} - (t_{CQ} + t_{SETUP}) \quad (2.3)$$

Tương tự, quá trình đọc dữ liệu cũng cần phải xảy ra trong một chu kỳ clk và được thể hiện ở phía bên phải của Hình 2.3. Tại sườn lên của xung clk (t5), vi xử lý ra lệnh đọc dữ liệu từ thanh ghi của Khối I/O. Sau khoảng thời gian t_{CQ} ,

tại thời điểm t6, các tín hiệu này đều ổn định trên bus và quá trình ghép kênh bắt đầu diễn ra. Tại thời điểm t7, dữ liệu từ nguồn chỉ định được định tuyến đến fp_rd_data sau khoảng thời gian t_{MUX} . Cuối cùng, tại sườn lên của xung clk tiếp theo (t8), vi xử lý lấy mẫu dữ liệu từ fp_rd_data .

Ràng buộc thời gian đối với quá trình đọc dữ liệu được thể hiện theo công thức như sau:

$$\begin{aligned} t_{CQ} + t_{MUX} + t_{SETUP} &< t_{CLK} \\ \Leftrightarrow t_{MUX} &< t_{CLK} - (t_{CQ} + t_{SETUP}) \end{aligned} \quad (2.4)$$

Do mạch giải mã và ghép kênh của FPro bus khá đơn giản, nên cả 2 ràng buộc trong quá trình đọc và ghi dữ liệu đều có thể thoả mãn khi hệ thống hoạt động ở tần số 100MHz. Điều này có thể được kiểm chứng bằng cách xem report timing summary như được trình bày trong mục 5.6 - Đánh giá hệ thống.

2.4 MCS2FPro bridge

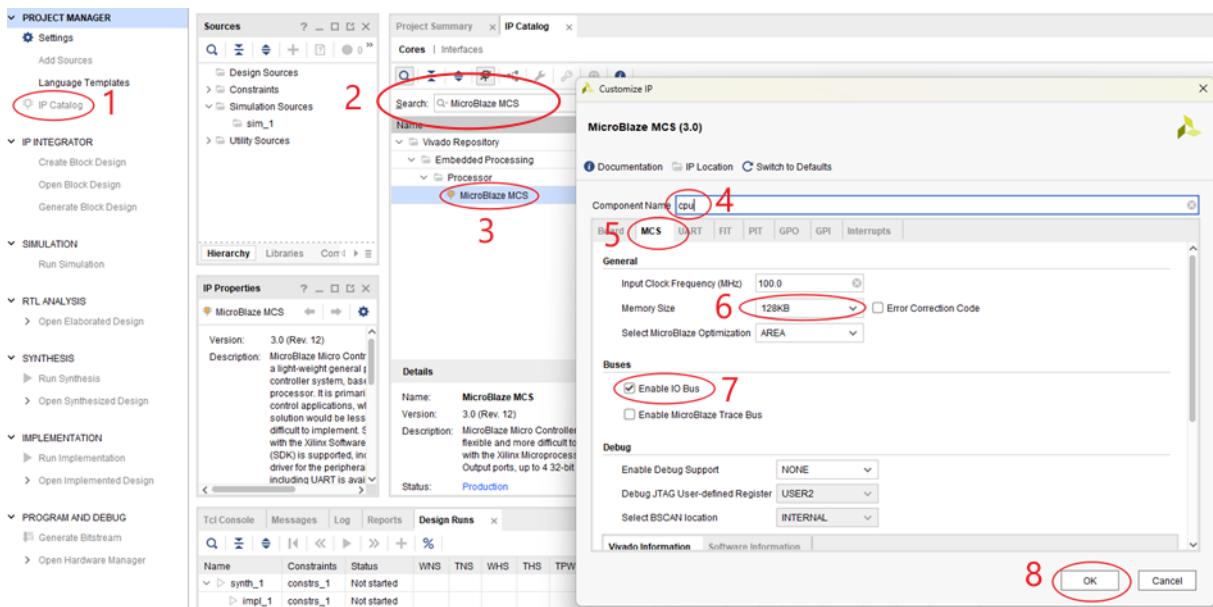
MCS-to-FPro bridge giúp Nền tảng FPro không bị ràng buộc bởi bất kỳ vi điều khiển cụ thể nào, miễn là bus của vi điều khiển đó có thể được ánh xạ sang FPro bus. Điều này giúp tăng tính di động của cả phần cứng và phần mềm trong hệ thống, cho phép các khói I/O được sử dụng trên nhiều nền tảng khác nhau mà không cần thay đổi quá nhiều về cấu trúc. Tuy nhiên do những hạn chế về mặt tài chính, thời gian cũng như nhân lực, đề tài sẽ chỉ triển khai hệ thống cho vi điều khiển MicroBlaze MCS.

2.4.1 Xilinx MicroBlaze MCS

MicroBlaze MCS (Micro Controller System) là một vi điều khiển 32 bits lõi mềm được phát triển bởi Xilinx. Từ mèn ở đây ám chỉ rằng vi điều khiển được tạo thành từ các ô logic khả trình trên FPGA. Vi điều khiển MicroBlaze MCS [2] cho khả năng tuy biến cao với nhiều tính năng như bộ đệm lệnh, bộ đệm dữ liệu, bộ dấu phẩy động, bộ quản lý bộ nhớ, ... Xilinx cung cấp rất nhiều các khói IP được tạo nghĩa sẵn có thể tích hợp vào MicroBlaze như bộ tăng tốc phần cứng

chuyên dụng, bộ quản điều khiển bộ nhớ, ... và các khối ngoại vi cơ bản như timer, UART, GPIO, ...

MicroBlaze MCS [2, 6] có thể được khởi tạo bằng công cụ IP Catalog của Vivado. Chi tiết cách cấu hình sẽ được trình bày trong phần phụ lục, với phạm vi đề tài này, MicroBlaze MCS cấu hình với các thông số bộ nhớ trong 128KB và kích hoạt I/O bus port để kết nối với FPro bus.



Hình 2.4 Khởi tạo MicroBlaze MCS trong phần mềm Vivado 2019.2

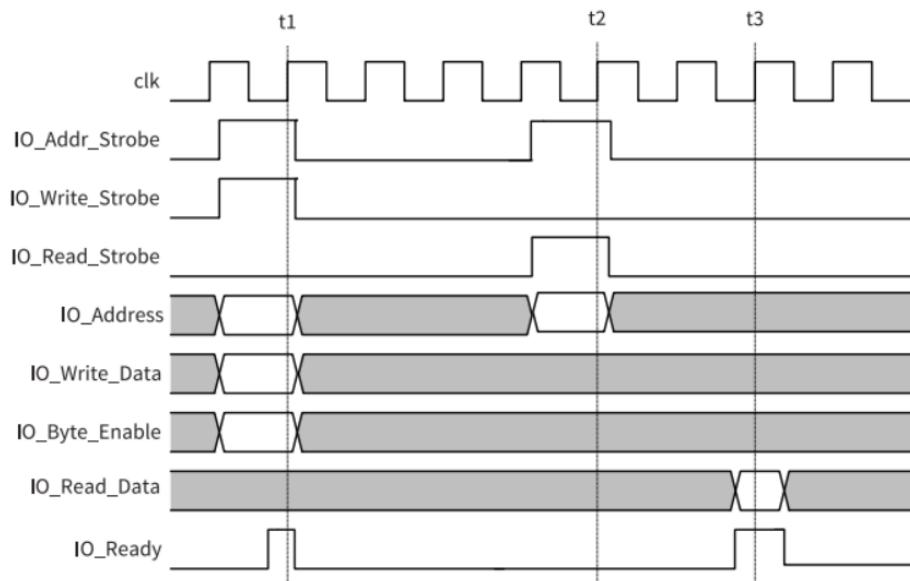
2.4.2 MicroBlaze MCS I/O bus

MicroBlaze MCS sử dụng một cổng bus I/O đơn giản làm để kết nối với các thiết bị ngoại vi. Một dải không gian địa chỉ theo byte có độ rộng 32 bits từ 0xc0000000 tới 0xffffffff [2, 9] được sử dụng để định địa chỉ cho các thanh ghi được kết của các thiết bị ngoại vi. MCS I/O bus là một bus đồng bộ. Ngoài các tín hiệu về *clk* hệ thống và *reset* thì nó còn bao gồm các tín hiệu sau:

- *IO_Address* (master tới slave): bus địa chỉ có độ rộng 32 bit.
- *IO_Read_Data* (slave tới master): bus dữ liệu có độ rộng 32 bit.
- *IO_Write_Data* (master tới slave): bus dữ liệu có độ rộng 32 bit.
- *IO_Addr_Strobe* (master tới slave): 1 bit tín hiệu báo hiệu rằng một địa chỉ hợp lệ đã được đặt lên bus địa chỉ.

- *IO_Read_Strobe* (master tới slave): 1 bit tín hiệu báo hiệu rằng MicroBlaze muốn đọc dữ liệu từ một thiết bị ngoại vi.
- *IO_Write_Strobe* (master tới slave): 1 bit tín hiệu báo hiệu rằng MicroBlaze muốn ghi dữ liệu vào một thiết bị ngoại vi.
- *IO_byte_enable* (master tới slave): tín hiệu điều khiển có độ rộng 4 bits chỉ định những bytes nào của *IO_Write_Data* được sử dụng trong thao tác ghi dữ liệu.
- *IO_ready* (slave tới master): tín hiệu có độ rộng 1 bit cho biết thiết bị ngoại vi đã sẵn sàng nhận lệnh tiếp theo hay chưa.

Phía bên trái Hình 2.5 mô tả thao tác ghi dữ liệu của MCS I/O bus [2, 9]. Đầu tiên, vi xử lý đặt địa chỉ và dữ liệu cần ghi lên bus *IO_Address* và *IO_Write_Data*. Đồng thời nó cũng cần kích hoạt các tín hiệu *IO_Addr_Strobe* và *IO_Write_Strobe* để bắt đầu thao tác ghi dữ liệu. Thiết bị ngoại vi I/O được chỉ định xử lý dữ liệu và kích hoạt tín hiệu *IO_ready* sau khi hoàn thành. Tín hiệu *IO_byte_enable* có thể được sử dụng nếu như chỉ có một phần của word được sử dụng trong quá trình ghi dữ liệu.



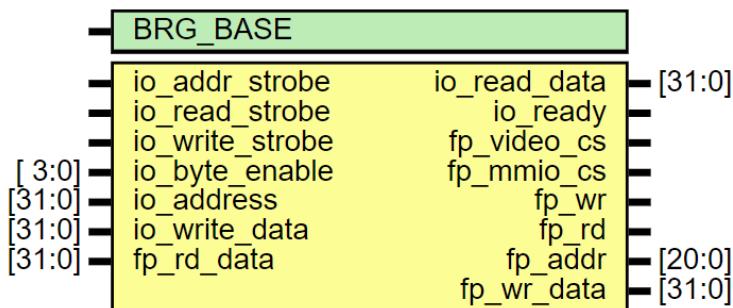
Hình 2.5 Giản đồ thời gian quá trình đọc/ghi dữ liệu của MCS I/O bus

Tương tự như vậy quá trình đọc dữ liệu của MCS I/O bus được thể hiện ở phía bên phải của Hình 2.5 [2, 9]. Để bắt đầu quá trình đọc dữ liệu từ thiết bị ngoại

vi, vi xử lý cần phải đặt địa chỉ của thiết bị ngoại vi đó lên *IO_Address*, đồng thời kích hoạt tín hiệu *IO_Addr_Strobe* và *IO_Read_Strobe*. Thiết bị ngoại vi truy xuất và đặt dữ liệu mà vi xử lý yêu cầu lên trên *IO_Read_Data* và kích hoạt tín hiệu *IO_ready* sau khi hoàn thành.

MicroBlaze MCS I/O bus sử dụng tín hiệu *IO_ready* để thực hiện giao thức bắt tay. Thiết bị ngoại vi có thể giữ *IO_ready* ở mức thấp khi dữ liệu chưa sẵn sàng trong nhiều xung *clk*. Điều này giúp cho hệ thống có thể kết nối nhiều thiết bị ngoại vi có tốc độ xử lý chậm hơn, ví dụ như quá trình đọc trong Hình 2.5 diễn ra trong 2 xung *clk*. Tuy nhiên nếu một thiết bị ngoại vi vì một lý do nào đó không kích hoạt tín hiệu *IO_ready*, toàn bộ hệ thống bị treo. Do đó cần phải hết sức cẩn trọng trong quá trình thiết kế các module I/O.

2.4.3 MCS2FPro bridge



Hình 2.6 MCS2FPro bridge

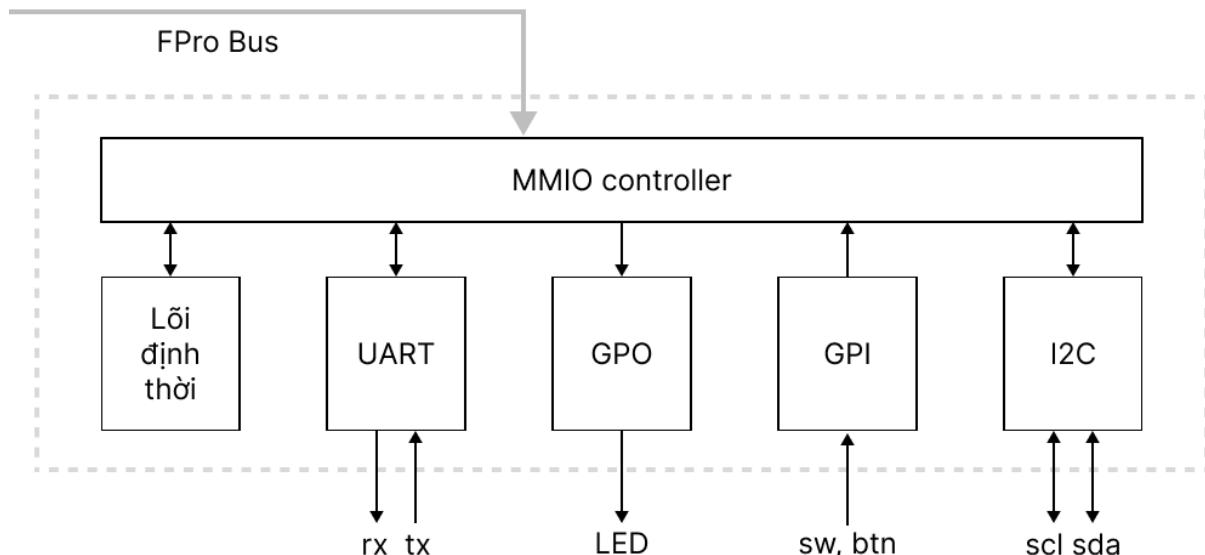
MCS2FPro bridge có nhiệm vụ kết nối giữa MCS I/O bus với FPro bus. MCS2FPro bridge chuyển đổi địa chỉ byte có độ rộng 32 bits của MCS I/O bus sang địa chỉ word có độ rộng 21 bits của FPro bus. Nói cách khác hệ thống FPro I/O được coi là một module I/O duy nhất với 24 bits địa chỉ byte (22 bits địa chỉ word) được kết nối với MicroBlaze MCS tại địa chỉ cơ sở 0xc0000000. Quá trình chuyển đổi 32 bits địa chỉ diễn ra như sau:

- Bit thứ 31 tới 24 được sử dụng để xác định địa chỉ cơ sở của nền tảng FPro.
- Bit thứ 23 được sử dụng làm để chọn giữa 2 hệ thống của nền tảng FPro.

- Bit thứ 22 tới 2 được sử dụng để xác định cách thanh ghi của I/O hoặc vùng nhớ trong hệ thống thiết bị ngoại vi hoặc hệ thống xử lý đồ họa.
- Bit 1 tới 0 không được sử dụng vì nền tảng FPro bus sử dụng địa chỉ theo word.

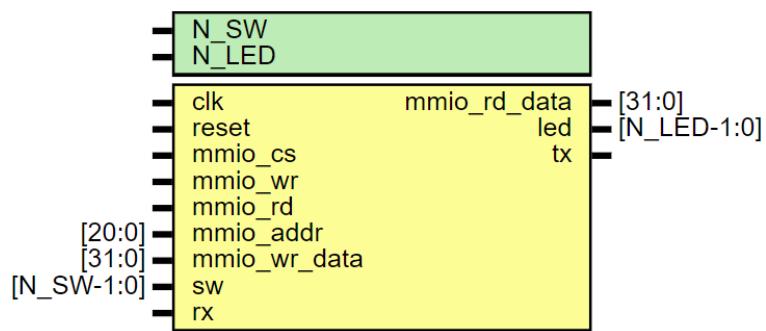
Tín hiệu *IO_Ready* luôn ở mức 1 vì quá trình đọc ghi của FPro bus chỉ diễn ra trong một chu kỳ và không cần phải có cơ chế bắt tay. Tín hiệu *IO_Addr_Strobe* cũng không được kiểm tra vì nó luôn được kích hoạt mỗi khi MicroBlaze MCS thực hiện thao tác đọc và ghi. Tín hiệu *IO_byte_enable* cũng không được kiểm tra vì các thao tác đọc ghi với hệ thống FPro luôn luôn thực hiện theo word. Các tín hiệu còn lại được kết nối trực tiếp từ MCS I/O bus tới FPro bus mà không cần có bất kỳ sự chuyển đổi nào.

2.5 Hệ thống thiết bị ngoại vi



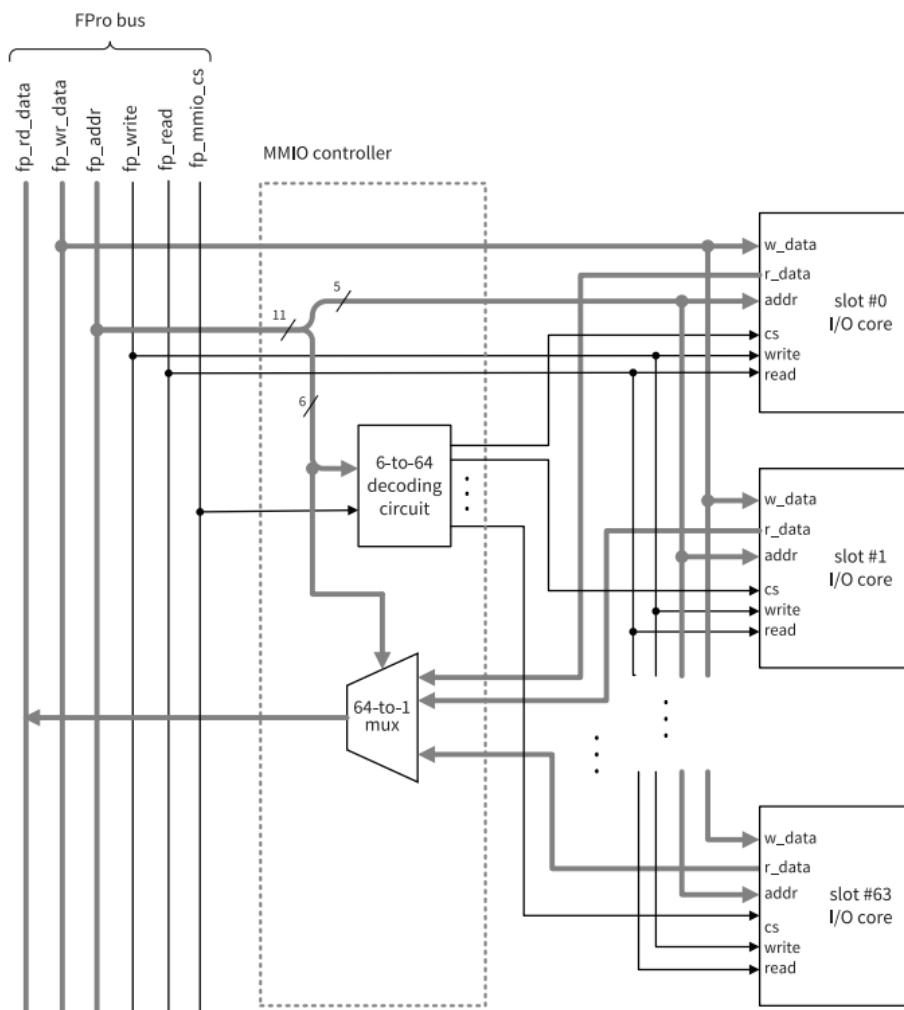
Hình 2.7 Sơ đồ kết nối của khối hệ thống thiết bị ngoại vi

Hệ thống thiết bị ngoại vi bao gồm khối điều khiển MMIO và các khối I/O (được trình bày trong CHƯƠNG 3). Mô-đun *mmio_sys* điều khiển việc ghi dữ liệu vào thanh ghi của khối I/O mà vi xử lý chỉ định cũng như định tuyến dữ liệu từ một thanh ghi của một khối I/O tới vi xử lý bằng cách sử dụng các mạch giải mã và mạch ghép kênh được cung cấp bởi mô-đun *mmio_controller*.



Hình 2.8 Mô-đun mmio_sys

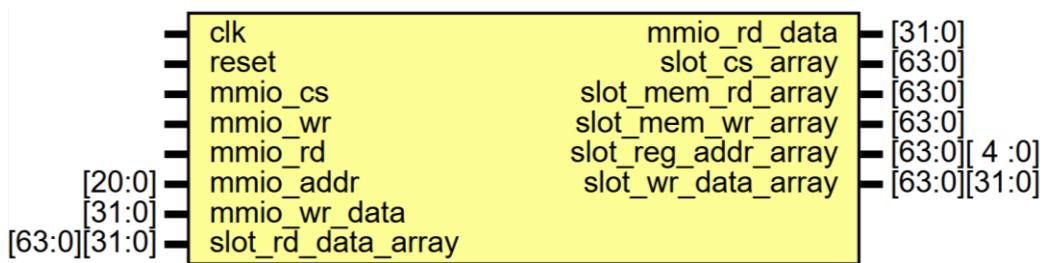
Khối điều khiển MMIO cung cấp một giao diện để kết nối khối MMIO với FPro bus để bộ vi xử lý có thể truy cập và điều khiển. Khối điều khiển MMIO có thể kết nối tối đa được với $2^6 = 64$ khối MMIO. Sơ đồ khái của hệ thống MMIO được thể hiện như trong Hình 2.9.



Hình 2.9 Sơ đồ khái của hệ thống thiết bị ngoại vi

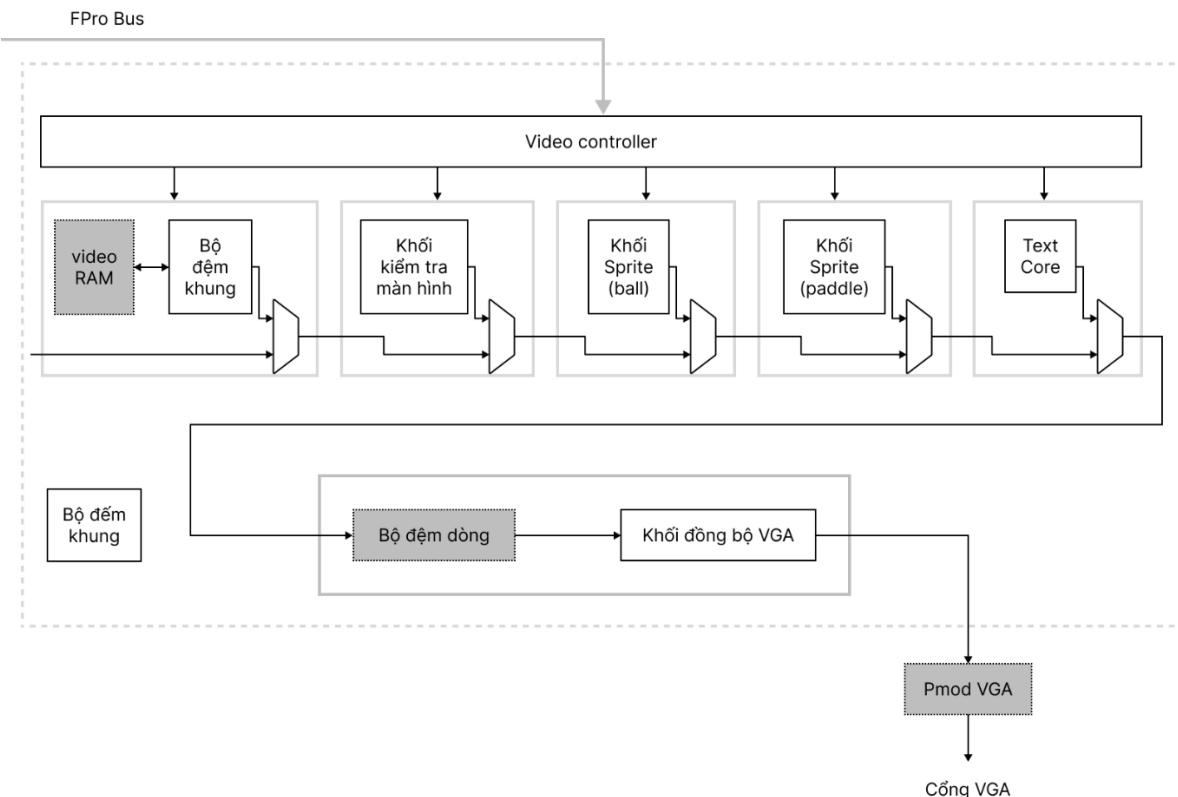
Khối điều khiển MMIO sử dụng các tín hiệu từ điều khiển từ FPro bus để thực hiện việc giải mã chọn các khối I/O cũng như định tuyến dữ liệu từ các khối

I/O này tới vi xử lý. MMIO sử dụng 11 bits địa chỉ có trọng số thấp nhất của PPro bus cho quá trình này. Trong đó 5 bits có trọng số thấp nhất được sử dụng để đánh địa chỉ của các thanh ghi trong Khối I/O, 6 bits còn lại được sử dụng để chọn module bằng cách sử dụng các tín hiệu này để truy cập vào các phần tử của mảng *slot_rd_data_array*, *slot_cs_array*. Tất cả các tín hiệu khác của FPro bus như *mmio_rd*, *mmio_wr*, *mmio_wr_data* cũng như 5 bits có trọng số thấp nhất của *mmio_addr* được truyền tới tất cả các khối I/O.



Hình 2.10 Khối điều khiển MMIO

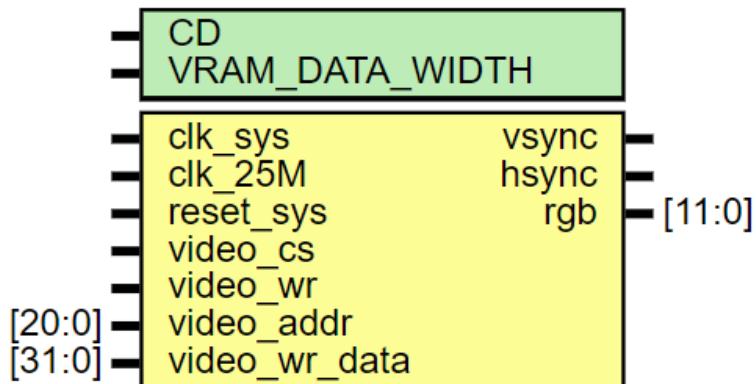
2.6 Hệ thống xử lý đồ họa



Hình 2.11 Hệ thống xử lý đồ họa

Hệ thống xử lý đồ họa có chức năng tạo, xử lý và hiển thị các dữ liệu hình ảnh lên màn hình thông qua cổng VGA. Hệ thống xử lý đồ họa bao gồm tập hợp

các khối video IP và một video controller như được thể hiện ở Hình 2.11. Khác với các khối I/O, khói video IP được kết nối theo kiểu nối tiếp, tức là tín hiệu đầu ra của một khói là tín hiệu đầu vào của một khói khác. Một điểm ảnh được tạo ra đầu tiên ở khói đệm khung, sau đó nó được truyền lần lượt qua từng khói ở phía sau. Mỗi khói chuyển tiếp, sửa đổi hoặc thay thế điểm ảnh mà nó nhận được để thay đổi hình ảnh hiển thị trên màn hình. Các điểm ảnh này cuối cùng được truyền đến khói đồng bộ VGA để đồng dữ liệu hình ảnh được tạo ra từ các khói video IP với tốc độ hiển thị của màn hình. Thông thường vi điều khiển không trực tiếp thao tác với dòng dữ liệu được gửi thông qua cổng VGA để hiển thị lên trên màn hình, thay vào đó nó ghi vào các thanh ghi của các khói video IP để điều khiển cách các khói này thao tác với dòng dữ liệu được gửi đi.



Hình 2.12 Mô-đun hệ thống xử lý đồ họa

Về cơ bản mô-đun video controller giống với mô-đun điều khiển MMIO. Tuy nhiên, vì các khói video IP hầu như không cần gửi dữ liệu về vi xử lý, nên video controller chỉ cần giải mã địa chỉ và ghi dữ liệu vào khói được chỉ định mà không cần phải thực hiện việc định tuyến dữ liệu từ các thanh ghi của các khói IP như khói điều khiển MMIO. Do các thanh ghi của các khói video IP cần phải lưu trữ dữ liệu hình ảnh hiển thị lên màn hình nên nó cũng yêu cầu không gian nhớ lớn hơn so với các khói I/O. Hệ quả là số lượng các khói video IP ít hơn đáng kể so với khói I/O. Ngoài ra do khói đệm khung được sử dụng để lưu trữ thông tin về các điểm ảnh được hiển thị trên màn hình nên nó cần một không gian địa chỉ lớn hơn rất nhiều so với các khói video IP thông thường.

CHƯƠNG 3: HỆ THỐNG THIẾT BỊ NGOẠI VI

3.1 Thiết kế các khối ngoại vi cho nền tảng FPro

Để có thể thiết kế một khối ngoại vi cho nền tảng FPro, cần thực hiện các bước như sau:

1. Thiết kế phần cứng (mạch số) cho khối ngoại vi bằng ngôn ngữ mô tả phần cứng SystemVerilog.
2. Xác định các thanh ghi cho khối ngoại vi.
3. Tạo giao diện kết nối với khối điều khiển MMIO.
4. Thiết kế trình điều khiển cho khối ngoại vi.

Giao diện giữa mỗi khối ngoại vi với FPro bus được định nghĩa như sau:

- *addr* (nối từ bus tới khói IP): sử dụng 5 bits bus địa chỉ để chỉ định thanh ghi 32 bits trong khói IP.
- *rd_data* (nối từ khói IP ra bus): Đường dữ liệu 32 bits chứa dữ liệu của cần đọc từ thanh ghi nội của khói IP.
- *wr_data* (nối từ bus tới khói IP): Đường dữ liệu 32 bits chứa dữ liệu để ghi vào thanh ghi nội của khói IP.
- *read* (bus tới khói IP): Có độ rộng 1 bit, được sử dụng để kích hoạt chế độ đọc dữ liệu từ thanh ghi của khói IP.
- *write* (bus tới khói IP): Có độ rộng 1 bit, được sử dụng để kích hoạt chế độ ghi dữ liệu vào thanh ghi của khói IP.
- *cs* (bus tới khói IP): Có độ rộng 1 bit được sử dụng để chọn khói IP nào hoạt động.

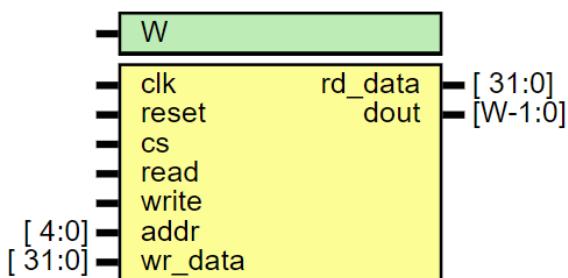
Để tránh nhầm lẫn trong khi phát triển phần cứng và phần mềm cũng như dễ dàng chuyển đổi và nâng cấp sau này, các hằng số về tần số hoạt động của hệ thống, địa chỉ cơ sở của MCS2FPro bridge và các khói của nền tảng FPro được định nghĩa trong tệp tin *io_map.h* (sử dụng khi phát triển phần mềm) và *io_map.sh* (sử dụng khi phát triển phần cứng). Đối với mỗi khói IP được gắn vào hệ thống thiết bị ngoại vi tại vị trí thứ *n*, base address của nó có thể được tính từ vị trí của nó theo công thức (2.1).

3.2 Khối GPO và GPI

GPI (General Purpose Input) và GPO (General Purpose Output) là các thành phần mà bất cứ hệ thống nhúng hoặc vi điều khiển nào cũng cần phải có để người dùng có thể để tương tác với các thiết bị ngoại vi. Để người dùng có thể tương tác được với hệ thống, FPro sử kết nối khối GPO với các led đơn trên mạch và các nút nhấn cũng như công tắc với GPI.

3.2.1 Thiết kế phần cứng

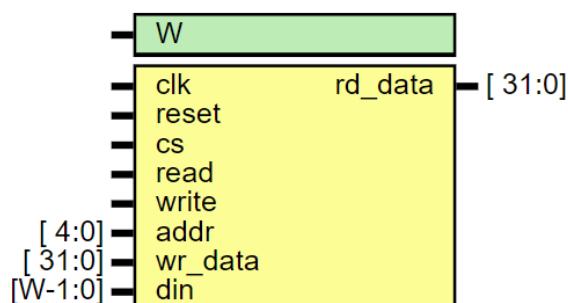
3.2.1.1 Khối GPO



Hình 3.1 Khối GPO

Khối GPO chỉ đơn giản là một thanh ghi được sử dụng để lưu trữ giá trị đầu ra. Dữ liệu được lưu trữ tại thanh ghi này khi *cs* và *write* được kích hoạt ở mức cao. Vì chỉ có một thanh ghi được sử dụng trong khối GPO nên việc giải mã tín hiệu *addr* để chọn thanh ghi này bị bỏ qua để tránh lãng phí tài nguyên phần cứng. Ngoài ra, tín hiệu *rd_data* cũng không được sử dụng nên nó được gán giá trị 0 và được Vivado tối ưu hóa trong quá trình tổng hợp.

3.2.1.2 Khối GPI

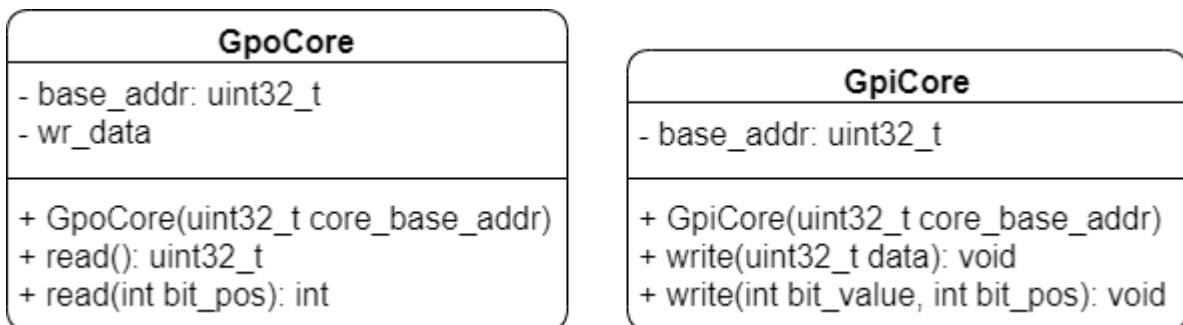


Hình 3.2 Khối GPI

Khối GPO gồm một thanh ghi chứa dữ liệu mà nó lấy mẫu được từ các chân đầu vào của nút nhấn hoặc công tắc trên bo mạch Arty A7-35T ở mỗi sườn dương của xung *clk*. Tương tự như GPO, do chỉ có duy nhất một thanh ghi để đọc giá trị nên khối GPI cũng không cần phải sử dụng mạch ghép kênh và các tín hiệu như *addr*, *wr_data*.

3.2.2 Thiết kế trình điều khiển

Hai lớp GpoCore và GpiCore cung cấp một số phương thức để đọc và ghi dữ liệu vào các thanh ghi của khối GPO và GPI, nội dung của mỗi lớp được thể hiện như trong Hình 3.3.



Hình 3.3 Lớp GpoCore và GpiCore

Vì GPO chỉ có 4 bits đầu ra nên nó chỉ có duy nhất một thanh ghi *DATA_REG* với địa chỉ offset là 0. GPO có 2 phương thức *write* trong đó phương thức thứ nhất viết cả một word dữ liệu xuống thanh ghi *DATA_REG*, phương thức thứ hai được sử dụng viết một bit xuống vị trí được chỉ định trong thanh ghi *DATA_REG*. GPO có 2 thuộc tính là *base_addr* và *wr_data*, trong đó:

- *base_addr* lưu trữ địa chỉ cơ sở của đối tượng
- *wr_data* chứa bản sao của dữ liệu được ghi xuống thanh ghi của GPO.

Dữ liệu này được dùng trong trường hợp cần ghi một bit dữ liệu xuống thanh ghi của GPO.

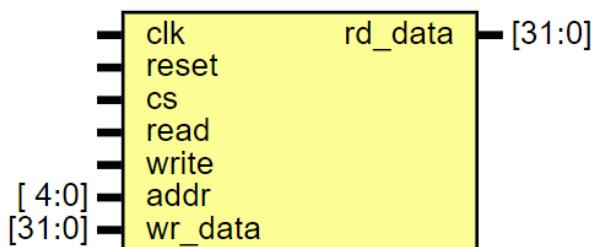
Tương tự GPO, GPI cũng chỉ có duy nhất một thanh ghi *DATA_REG* với địa chỉ offset là 0. Tuy nhiên việc sao chép thông tin của thanh ghi này là không cần thiết vì dù là đọc 1 byte hay 1 bit thì xử lý đều cần phải đọc trực tiếp giá trị từ khối GPI.

Lưu ý dữ liệu của các thanh ghi của cả hai IP GPO và GPO cũng như dữ liệu được truyền xuống các thanh ghi này đều là 32 bits. Do đó quá trình ghi dữ liệu chỉ sử dụng các bits có trọng số thấp nhất, các bits khác đều bị bỏ qua. Tương tự như vậy chỉ có các bits có trọng số thấp nhất chứa giá trị đọc được từ các nút nhấn và công tắc, các bit còn lại có thể chứa một giá trị bất kỳ.

3.3 Khối định thời

Khối định thời là một bộ đếm thời gian sau khi vi điều khiển được cấp nguồn hoặc nút reset được nhấn. Khối định thời được tích hợp trong các vi điều khiển hoặc hệ thống nhúng để thực hiện các tác vụ liên quan tới thời gian thời gian như tạm dừng hoạt động của hệ thống (sleep).

3.3.1 Thiết kế phần cứng



Hình 3.4 Khối định thời

Khối định thời sử dụng 3 thanh ghi như được thể hiện trong Hình 2.2. Về bản chất khói định thời là một bộ đếm có thể chạy/dừng hoặc xoá bởi các tín hiệu điều khiển. Hai thanh ghi 0 và 1 tương ứng với 32 bits trọng số thấp nhất và cao nhất của bộ đếm. Nói cách khác khói timer là một bộ đếm 64 bits và có thể đếm được 2^{64} xung *clk*. Với tần số hoạt động của hệ thống là 100MHz, khói định thời có thể hoạt động chính xác trong khoảng $\frac{2^{64}}{10^7 \times 60 \times 60 \times 24 \times 365} \approx 5.8$ nghìn năm trước khi xảy ra tràn số.

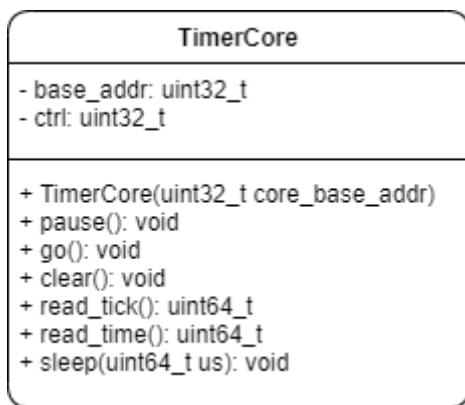
Bộ vi xử lý có thể thực hiện các thao tác sau với khói định thời:

- Đọc giá trị của bộ đếm được lưu trữ trong hai thanh ghi 0 và 1.
- Tạm dừng hoặc tiếp tục chạy khói định thời bằng cách ghi giá trị 0 hoặc 1 vào bit có trọng số thấp nhất của thanh ghi 2.

- Xoá giá trị của các thanh ghi 0 và 1 bằng cách ghi giá trị 1 vào bit 1 của thanh ghi 2.

Để thực hiện được các thao tác trên khối định thời sử dụng một thanh ghi điều khiển, một mạch giải mã cũng như ghép kênh. Mạch giải mã có tác dụng lựa chọn thanh ghi mà vi xử lý thao tác. Thanh ghi điều khiển có tác dụng lưu trữ giá trị của tín hiệu trạng thái tạm dừng hay tiếp tục hoạt động của khối định thời. Mạch ghép kênh định tuyến dữ liệu phần trên 32 bit có trọng số lớn nhất hoặc nhỏ nhất của bộ đếm tới *rd_data* dựa trên bit 0 của *addr* (offset 0 hoặc 1).

3.3.2 Thiết kế trình điều khiển



Hình 3.5 Lớp TimerCore

Phương thức pause của lớp TimerCore có tác dụng dừng bộ đếm. Ngược lại phương thức go có tác dụng kích hoạt bộ đếm. Phương thức clear xoá giá trị của bộ đếm về 0. Phương thức read_tick và read_time trả về số lượng sườn dương của tín hiệu xung nhịp mà bộ đếm được và số micro giây đã trôi qua kể từ lần cuối cùng phương thức clear được gọi. Phương thức sleep có tác dụng tạm dừng hoạt động của vi xử lý trong một khoảng thời gian được chỉ định.

3.4 Khối UART

3.4.1 Tổng quan về giao thức UART

Giao thức UART (Universal Asynchronous Receiver-Transmitter) là một giao thức truyền thông nối tiếp không đồng bộ được sử dụng rộng rãi trong các hệ thống nhúng và thiết bị viễn thông để truyền dữ liệu giữa các thiết bị.

UART thường bao gồm một thiết bị phát và một thiết bị nhận. Thiết bị phát làm một thanh ghi dịch có chức năng tải dữ liệu ở dạng song song và sau đó dịch từng bit ra ở một tốc độ cụ thể. Trái ngược với thiết bị phát, thiết bị thu nhận và dịch từng bit một để chuyển dữ liệu về dạng song song. Khi không có dữ liệu trên đường truyền thì điện áp trên đường truyền ở mức cao. Quá trình truyền dữ liệu bắt đầu với một bit khởi đầu có giá trị 0, theo sau bởi các bit dữ liệu, và một bit chẵn lẻ (có thể có hoặc không), và kết thúc bởi bit dừng có giá trị 1. Số lượng bit dữ liệu được truyền có thể là 6, 7 hoặc 8. Bit chẵn lẻ được sử dụng để phát hiện lỗi trong quá trình truyền, bằng cách đảm bảo số lượng bit 1 được truyền đi là một số chẵn hoặc lẻ. Nếu bit chẵn lẻ là 0 (tính chẵn), thì tổng các bit 1 trong khung dữ liệu phải là một số chẵn. Nếu bit chẵn lẻ là 1 (tính lẻ), các bit 1 trong khung dữ liệu tổng thành một số lẻ. Dữ liệu được coi là truyền đúng khi bit chẵn lẻ khớp với dữ liệu. Số lượng dừng bit có thể là 1, 1.5 hoặc 2 tùy thuộc vào từng loại cấu hình cụ thể.

UART là một giao thức không đồng bộ, tức là không cần xung nhịp chung giữa thiết bị phát và thiết bị thu. Trước khi quá trình truyền dữ liệu bắt đầu, thiết bị phát và thu cần phải thống nhất trước một số tham số bao gồm tốc độ baud (số bit truyền trên một giây), số bit dữ liệu được truyền trong một gói tin, số lượng bit dừng, và có sử dụng bit chẵn lẻ hay không.



Hình 3.6 Quá trình truyền một byte dữ liệu

Thông thường tần số lấy mẫu dữ liệu của UART bằng 16 lần tốc độ baud [9]. Khối UART chỉ lấy mẫu tín hiệu của một bit ở giữa quá trình truyền vì lúc này tín hiệu ổn định nhất. Nhìn chung với một gói tin gồm N bits dữ liệu và M bit dừng, quá trình lấy mẫu diễn ra như sau:

1. Khởi động bộ đếm mod 16 khi phát hiện bit khởi đầu trên đường truyền (đường truyền từ mức cao về mức thấp).

2. Xoá giá trị của bộ đếm về 0 và khởi động lại nó khi nó đếm đến 7 vì lúc này tín hiệu mà khói UART nhận được đang ở giữa bit start.
3. Khi bộ đếm đạt tới giá trị 15, tín hiệu mà trên đường truyền lúc này đang ở giữa của bit dữ liệu có trọng số thấp nhất, khói UART lấy mẫu giá trị này để lưu vào thanh ghi dịch và khởi động lại bộ đếm.
4. Lặp lại bước 3 N-1 lần để nhận hết các bit dữ liệu còn lại.
5. Lặp lại bước 3 một lần nữa nếu như có sử dụng bit kiểm tra chẵn lẻ.
6. Lặp lại bước 3 M lần để nhận bit stop.

Mặc dù thiết bị nhận không có thông tin chính xác về thời gian quá trình truyền dữ liệu bắt đầu thì thời gian lấy mẫu sai lệch nhiều nhất cũng chỉ là $\frac{1}{16}$ lần tốc độ baud. Cũng bởi vì cơ chế lấy mẫu này nên tốc độ baud luôn luôn nhỏ hơn so với tần số hoạt động của hệ thống. Do đó tốc độ truyền dữ liệu của giao thức UART thường không cao. Tuy nhiên do tính chất đơn giản nên giao thức UART vẫn được sử dụng rộng rãi trong các hệ thống cần truyền dữ liệu ở tốc độ thấp.

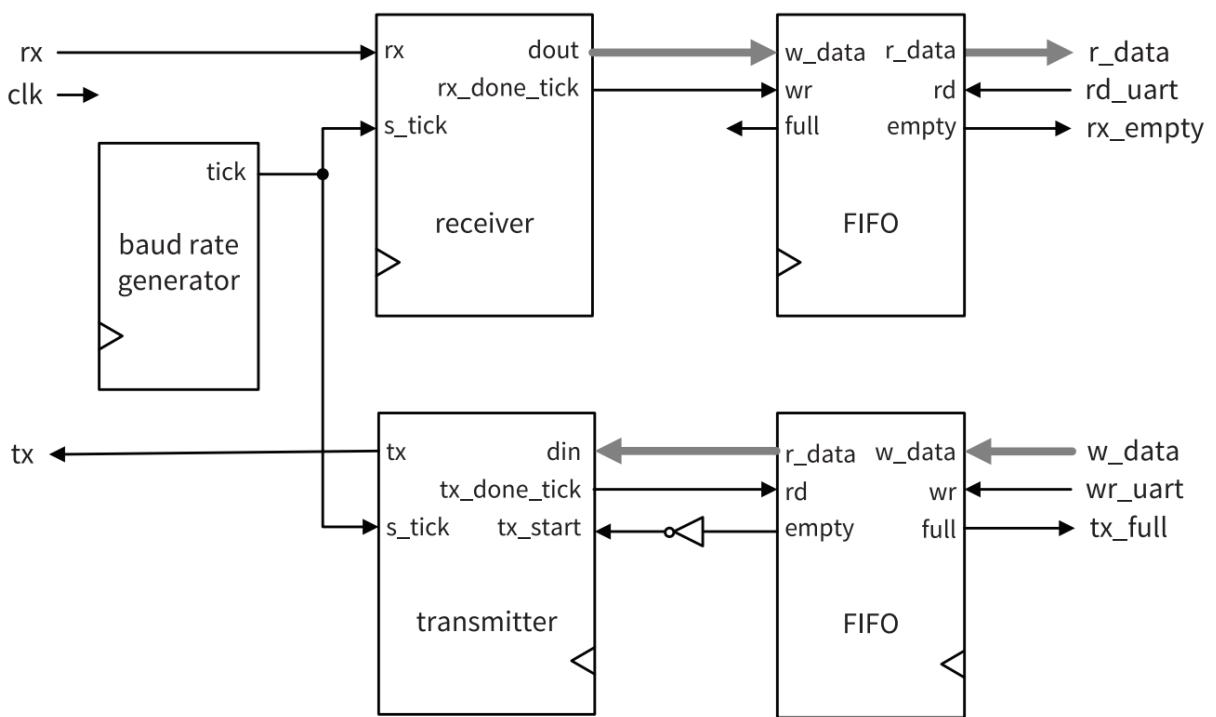
3.4.2 Thiết kế phần cứng

3.4.2.1 Khối UART

Một khói UART thường bao gồm 5 thành phần chính:

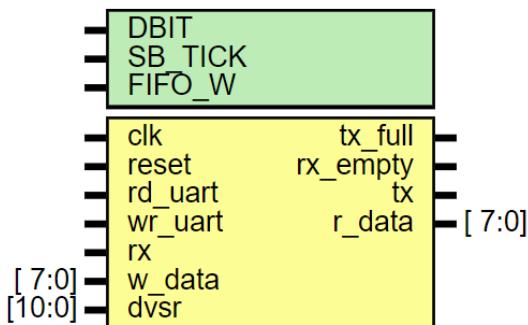
- Bộ tạo tốc độ baud được sử dụng để tạo ra tín hiệu lấy mẫu.
- Thiết bị thu thực hiện việc chuyển đổi dữ liệu từ dạng nối tiếp sang song song.
- Thiết bị phát thực hiện việc chuyển đổi dữ liệu từ dạng song song sang nối tiếp.
- Hai bộ đếm FIFO được sử dụng để kết nối thiết bị thu và phát với vi xử lý.

Khối UART cần phải có các bộ đếm FIFO vì tốc độ truyền và xử lý của khói IP thường chậm hơn rất nhiều so với tốc độ của vi xử lý lõi mềm MicroBlaze. Các bộ đếm này cho phép vi xử lý gửi nhiều bytes liên tục cho UART và thực hiện các công việc khác thay vì phải đợi UART truyền từng byte một riêng lẻ.



Hình 3.7 Sơ đồ khái niệmUART

Khối UART có nhiệm vụ kết các mô-đun UART receiver, transmitter, bộ tạo tốc độ baud và các bộ đệm FIFO như được minh họa trong Hình 3.7.

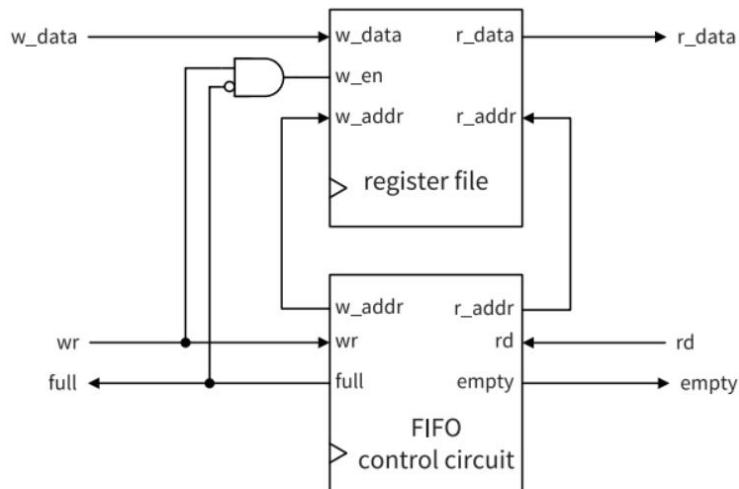


Hình 3.8 KhốiUART

3.4.2.2 Bộ đệm FIFO

Bộ đệm FIFO (First In First Out) sử dụng hai tín hiệu điều khiển là wr và rd cho các thao tác đọc và ghi dữ liệu. Khi tín hiệu wr được kích hoạt, dữ liệu được ghi vào cuối của bộ đệm. Khi tín hiệu rd được kích hoạt, dữ liệu được lấy ra và loại bỏ khỏi đầu của bộ đệm. Dữ liệu lấy ra từ bộ đệm có thứ tự giống với dữ liệu được ghi vào bộ đệm.

Bộ đệm FIFO có thể được triển khai bằng cách sử dụng một bộ nhớ RAM cùng kết hợp với một mạch điều khiển như ở Hình 3.9. Mạch điều khiển tạo ra các tín hiệu như w_addr, r_addr, full, empty dựa trên các tín hiệu rd và wr.



Hình 3.9 Bộ đệm FIFO

Để xác định được địa chỉ ghi và đọc mạch điều khiển sử dụng hai con trỏ wr_ptr và rd_ptr. Sau mỗi lần ghi dữ liệu giá trị của wr_ptr được tăng lên một đơn vị. Với mỗi lần đọc dữ liệu giá trị của rd_ptr cũng được tăng lên một đơn vị. Khi giá trị của con trỏ rd_ptr và wr_ptr bằng nhau thì hoặc là bộ đệm đang không có dữ liệu, hoặc là bộ đệm đang đầy dữ liệu. Để có thể phân biệt được hai trạng thái này sử dụng hai thanh ghi empty và full để lưu trạng thái bộ đệm đang rỗng hoặc đầy. Sau khi reset thanh ghi empty chưa giá trị 1 còn full chứa giá trị 0. Với mỗi lần ghi dữ liệu (chỉ diễn ra giá trị của thanh ghi full khác 0) giá trị con trỏ wr_ptr tăng lên một đơn vị và giá trị của thanh ghi empty chuyển về mức logic 0. Nếu như sau khi ghi dữ liệu được vào bộ đệm mà giá trị của hai con trỏ bằng nhau thì bộ đệm đã đầy dữ liệu, lúc này giá trị của thanh ghi full được chuyển sang mức logic 1.

Tương tự như vậy, với mỗi lần đọc dữ liệu (chỉ diễn ra khi thanh ghi empty có giá trị khác 0) giá trị của con trỏ rd_ptr được tăng lên một đơn vị và giá trị của thanh ghi full chuyển về mức logic 0. Nếu như sau khi đọc dữ liệu từ bộ đệm mà giá trị của hai con trỏ bằng nhau thì bộ đệm đang không chứa dữ liệu, lúc này giá trị của thanh ghi empty được chuyển sang mức logic 1.

3.4.2.3 Bộ tạo tốc độ baud



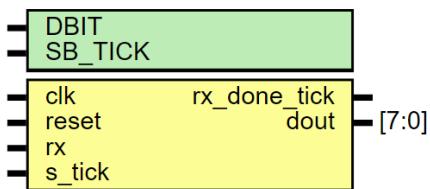
Hình 3.10 Bộ tạo tốc độ baud

Bộ tạo tốc độ baud là một bộ đếm và nó tạo ra tín hiệu lấy mẫu có tần số bằng 16 lần tốc độ truyền được chỉ định của khói UART. Vì giá trị của bộ đếm có thể được thiết lập thông qua phần mềm, nên để giảm độ phức tạp của phần cứng và tiết kiệm tài nguyên khói UART sử dụng một bộ đếm thay vì phải tạo thêm một bộ nhân và trừ để thực hiện đếm đúng 16 lần tốc độ baud được thiết lập trước quá trình truyền. Giá trị được tải xuống bộ đếm này có thể được xác định bằng công thức sau:

$$dvsr = \frac{f_{sys}}{16 * b} - 1 \quad (3.1)$$

Để tránh tạo ra một miền tần số hoạt động khác trong nền tảng FPro, khói UART sử dụng tín hiệu lấy mẫu làm tín hiệu enable thay vì tín hiệu xung nhịp của bộ thu và phát.

3.4.2.4 Mô-đun bộ thu

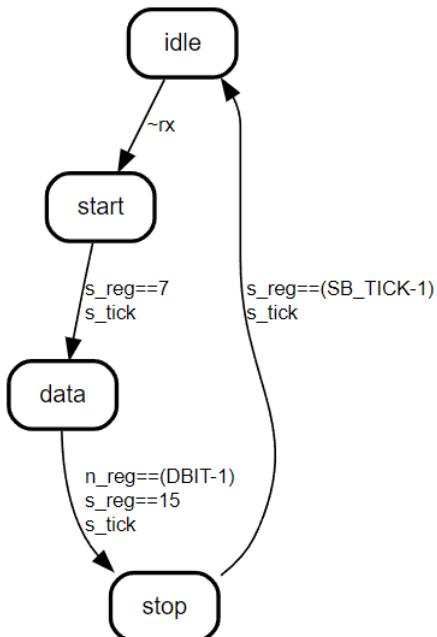


Hình 3.11 Mô-đun bộ thu

Bộ thu nhận dữ liệu trên đường truyền bằng cách lấy mẫu mỗi bit ở giữa quá trình truyền dựa trên tín hiệu *s_tick* từ bộ tạo tốc độ baud. Mô-đun bộ thu sử dụng tham số DBIT để biểu thị số lượng bit dữ liệu và SB_TICK để biểu thị số lượng xung của *s_tick* cần thiết cho 1 bit stop (16, 24 hoặc 32 với bit stop có độ dài là 1, 1.5 và 2).

Quá trình lấy mẫu được thể hiện bằng máy trạng thái như trong Hình 3.12, bao gồm 4 trạng thái idle, start, data, stop lần lượt đại diện cho các trạng thái trên đường truyền. Máy trạng thái sử dụng 2 bộ đếm s và n, trong đó:

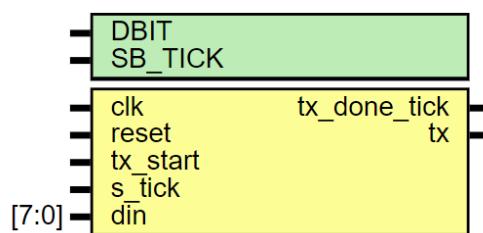
- s đếm số lượng xung của tín hiệu *s_tick*. bộ đếm này đếm đến 7 ở trạng thái start, 15 ở trạng thái data, và SBTICK ở trạng thái stop.
- n đếm số lượng bits dữ liệu đã nhận được ở trong trạng thái data. Mỗi khi đọc giá trị một bit mới (khi bộ đếm s có giá trị 15) thì giá trị này được dịch chuyển vào thanh ghi b.



Hình 3.12 Máy trạng thái của bộ thu

Tín hiệu *rx_done_tick* được kích hoạt trong một chu kỳ sau khi quá trình nhận dữ liệu hoàn tất.

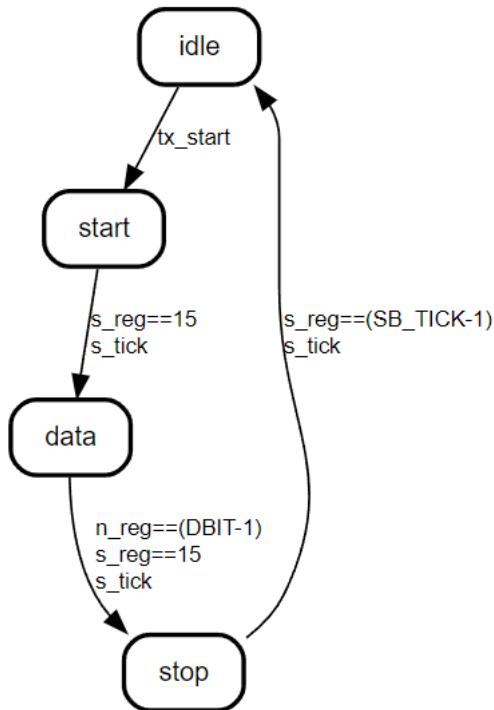
3.4.2.5 Mô-đun bộ phát



Hình 3.13 Mô-đun bộ phát

Bộ phát về bản chất là một thanh ghi dịch. Tốc độ dịch được kiểm soát bởi bộ tạo tốc độ baud, một bits được truyền đi trong khoảng 16 xung của tín hiệu

s_tick. Máy trạng thái của bộ phát được thể hiện như trong Hình 3.14. Sau khi tín hiệu *tx_start* được kích hoạt, bộ phát lưu dữ liệu ở *din* và một thanh ghi nội và dịch dữ liệu trong thanh ghi này ra *tx* trong trạng thái data. Cuối cùng sau khi hoàn thành việc truyền dữ liệu bộ phát kích hoạt tín hiệu *tx_done_tick* trong một chu kỳ.

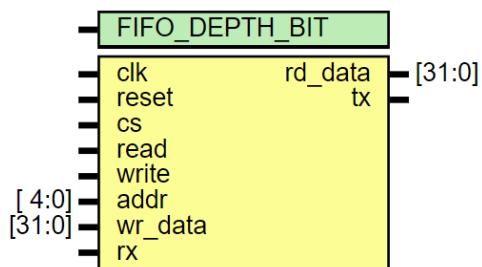


Hình 3.14 Máy trạng thái của bộ phát

3.4.2.6 Giao diện kết nối UART

Các thao tác mà vi xử lý có thể thực hiện được với khối UART:

- Thiết lập giá trị của thanh ghi *dvsr* của bộ tạo tốc độ baud theo công thức (3.1).
- Kiểm tra bộ đếm FIFO của bộ thu có rỗng hay không.
- Nhận một byte dữ liệu từ bộ đếm FIFO của bộ thu.
- Xoá một byte dữ liệu từ bộ đếm FIFO của bộ thu.
- Kiểm tra xem bộ đếm FIFO của bộ phát có đã đầy hay chưa.
- Gửi một byte dữ liệu tới bộ đếm FIFO của bộ đếm phát.



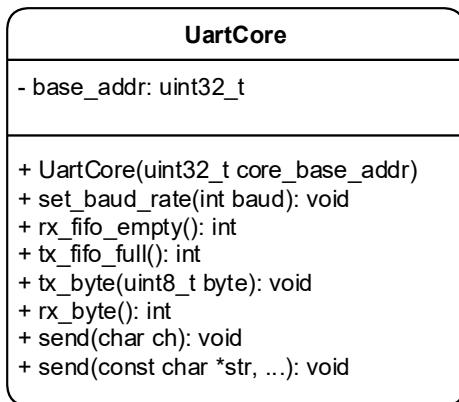
Hình 3.15 Giao diện kết nối UART

Giao diện kết nối UART có tác dụng chuyển đổi các tín hiệu mà vi xử lý ghi hoặc đọc từ cách thanh ghi thành các tín hiệu để điều khiển quá trình truyền hoặc nhận dữ liệu của khói UART thông qua bộ giải mã và ghép kênh. Các thanh ghi sau được sử dụng để điều khiển quá trình hoạt động của khói UART:

- Offset 0 (thanh ghi chỉ đọc):
 - Bit 7-0: chứa 8 bits dữ liệu nhận được của bộ thu.
 - Bit 8: chứa tín hiệu empty của bộ đệm FIFO thu.
 - Bit 9: chứa tín hiệu full của bộ đệm FIFO phát.
- Offset 1 (thanh ghi chỉ ghi):
 - Bit 10-0: giá trị của *dvsr* của bộ tạo tốc độ baud.
- Offset 2 (thanh ghi chỉ ghi):
 - Bit 7-0: chứa 8 bits dữ liệu cần truyền đi.
- Offset 3 (thanh ghi chỉ ghi): ghi dữ liệu bất kỳ xuống thanh ghi này trong một chu kỳ sẽ xoá một byte nhận được trong bộ đệm FIFO của bộ thu.

Vì UART thường được sử dụng với 8 bits dữ liệu và 1 bit dừng nên khói UART thiết lập các tham số DBIT và SB_TICK của khói UART với giá trị lần lượt là 8 và 16 trong đồ án này. Do chỉ sử dụng 4 thanh ghi nên chỉ có 2 bits trọng số thấp nhất của tín hiệu *addr* và *cs* để giải mã và lựa chọn thanh ghi được ghi dữ liệu. Vì chỉ có một thanh ghi để đọc dữ liệu nên giao diện kết nối UART không sử dụng bộ ghép kênh để tiết kiệm tài nguyên phần cứng.

3.4.3 Thiết kế trình điều khiển



Hình 3.16 Lớp UartCore

Phương thức *set_baud_rate* được sử dụng để tính toán giá trị của thanh ghi *dvsr* dựa trên tốc độ baud bằng công thức (3.1). Khi tạo mới một đối tượng thì tốc độ baud mặc định là 115200.

Phương thức *rx_fifo_empty* trả về 1 nếu bộ đệm FIFO thu rỗng, *tx_fifo_full* trả về 1 nếu bộ đệm phát đầy. Trong các trường hợp khác các phương thức này trả về giá trị 0.

Phương thức *tx_byte* đợi cho đến khi bộ đệm phát rỗng và viết một byte xuống bộ đệm này. Phương thức *rx_byte* kiểm tra xem bộ đệm thu có rỗng hay không. Nếu rỗng thì nó trả về giá trị -1, ngược lại nó lấy một byte dữ liệu ra khỏi bộ đệm thu.

Phương thức *send* được sử dụng để truyền một ký tự hoặc một chuỗi các ký tự.

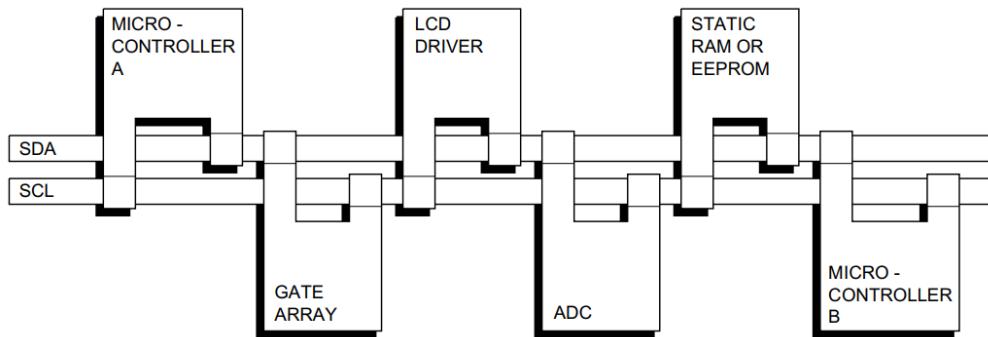
3.5 Khối I2C

3.5.1 Tổng quan về giao thức I2C

Giao thức I2C (Inter-Integrated Circuit) là một giao thức giao tiếp nối tiếp đồng bộ được phát triển bởi Philips Semiconductors (nay là NXP Semiconductors) vào năm 1982. Bus của I2C bao gồm hai dây:

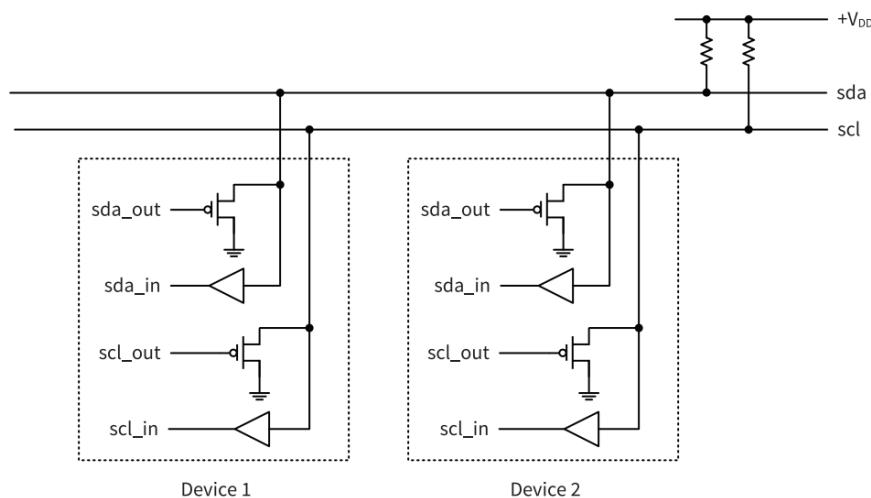
1. SDA (serial data): đường truyền dữ liệu nối tiếp hai chiều.

2. SCL (serial clock): đường truyền tín hiệu đồng hồ. Các bit dữ liệu trên SDA được truyền đi trong các khoảng thời gian đều đặn được thiết lập bởi tín hiệu đồng hồ này.



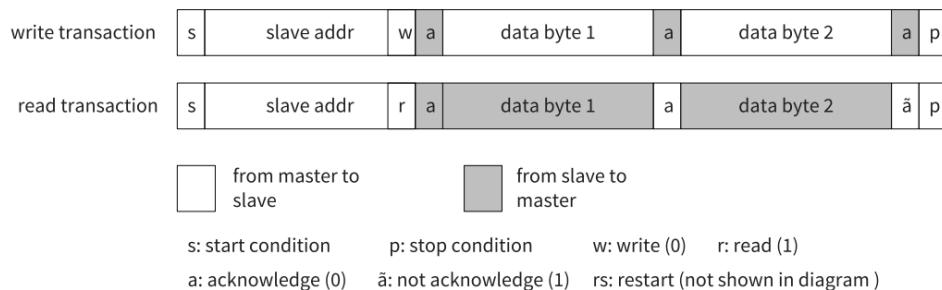
Hình 3.17 Hai vi điều khiển kết nối với các thiết bị ngoại vi trên cùng một đường truyền I2C

Giao thức I2C có thể được sử dụng để truyền dữ liệu từ một hay nhiều thiết bị master tới các slave khác nhau. Trong quá trình truyền dữ liệu, chỉ có một thiết bị trên đường truyền đóng vai trò là master, các thiết bị khác đóng vai trò là slave. Thiết bị master có vai trò tạo ra tín hiệu xung đồng hồ trên đường SCL cũng như khởi tạo và kết thúc quá trình truyền dữ liệu. Mỗi thiết bị slave trên bus I2C đều có những địa chỉ khác nhau. Master sử dụng địa chỉ này để chỉ định thiết bị slave mà nó muốn trao đổi dữ liệu. Khi bắt đầu quá trình truyền dữ liệu các thiết bị slave lắng nghe và phản hồi lại master nếu địa chỉ của nó được gửi đi trên đường truyền SDA. Sau khi thiết lập kết nối thiết bị master và slave được chỉ định trao đổi dữ liệu trên đường truyền SDA.



Hình 3.18 Cấu trúc của đường truyền I2C

I2C sử dụng công nghệ open-drain, điều này có nghĩa là các chân SCL và SDA của thiết bị được điều khiển bởi một bóng bán dẫn. Việc sử dụng công nghệ open-drain không những giúp kết nối được nhiều thiết bị với nhau thông qua I2C bus mà còn đảm bảo rằng các thiết bị không bị hỏng do ngắn mạch khi có một thiết bị gửi ra SDA hoặc SCL mức điện áp cao trong khi thiết bị khác lại gửi ra mức điện áp thấp.



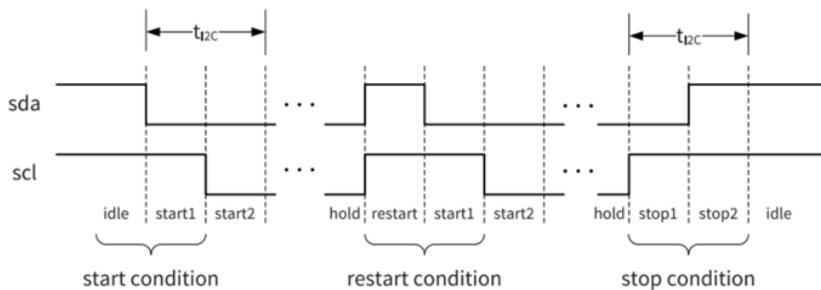
Hình 3.19 Khuôn dạng dữ liệu của giao thức I2C

Quá trình gửi hoặc nhận dữ liệu trên bus I2C cơ bản diễn ra như sau:

1. Khi một thiết bị (master) muốn sử dụng I2C bus thì nó phải tạo ra một tín hiệu start.
2. Thiết bị master gửi một byte chứa bảy bit địa chỉ của thiết bị slave mà nó muốn kết nối và một bit direction để xác định xem nó muốn gửi hay nhận dữ liệu từ slave. Nếu master muốn đọc dữ liệu từ slave thì bit này phải có giá trị là 1, ngược lại nếu master muốn gửi dữ liệu tới slave thì bit này có giá trị là 0.
3. Thiết bị slave có địa chỉ giống với địa chỉ được gửi từ master gửi một bit ACK (acknowledge) vào xung clock thứ chín.
4. Thiết bị master (hoặc slave nếu bit direction là 1) truyền tám bit data (byte data đầu tiên).
5. Thiết bị slave (hoặc master nếu bit direction là 1) gửi bit ACK vào xung clock tiếp theo báo hiệu đã nhận được dữ liệu từ slave.
6. Thiết bị master (hoặc slave nếu bit direction là 1) truyền tám bit data (byte data thứ hai).

7. Thiết bị slave (hoặc master nếu bit direction là 1) gửi bit ACK vào xung clock tiếp theo.
8. Thiết bị master tạo tín hiệu stop báo hiệu kết thúc quá trình truyền và nhả đường truyền cho các thiết bị master khác có thể sử dụng. Cần phải lưu ý rằng, trong quá trình master đọc dữ liệu, tín hiệu nack sẽ được gửi đi thay vì ack trước khi master tạo ra tín hiệu stop và kết thúc nhận dữ liệu.

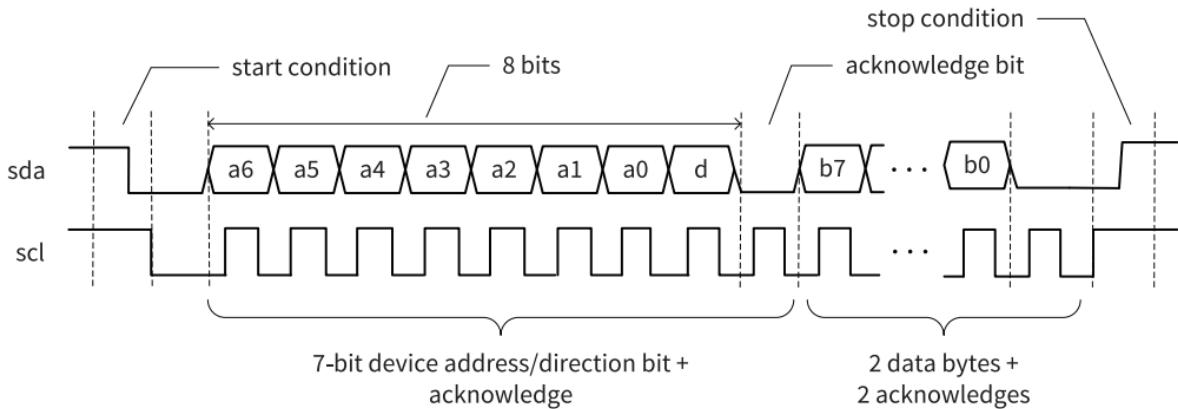
Trong trường hợp thiết bị master muốn tiếp tục một phiên truyền dữ liệu khác, nó có thể tạo một tín hiệu restart (hay còn được gọi là tín hiệu repeated start). Sau đó master tạo tín hiệu stop sau phiên truyền dữ liệu cuối cùng để nhả đường truyền cho các thiết bị khác sử dụng. Về cơ bản tín hiệu restart được sử dụng để kết thúc một phiên truyền dữ liệu giống như tín hiệu stop, nhưng nó không nhả đường truyền I2C ra để các thiết bị khác chiếm dụng. Master thường tạo ra tín hiệu restart ở giữa quá trình truyền địa chỉ và đọc giá trị của một thanh ghi bên trong thiết bị slave.



Hình 3.20 Tín hiệu start, stop, restart

Giản đồ thời gian của các tín hiệu start, stop và restart được thể hiện như trong Hình 3.20. Chi tiết quá trình truyền hai bytes dữ liệu được mô tả trong Hình 3.21. Đầu tiên đường truyền I2C ở trạng thái rảnh (idle), tức là cả hai đường truyền SCL và SDA đều ở mức cao. Thiết bị master tạo tín hiệu start, nói một cách khác master kéo SDA xuống mức thấp trong khi SCL đang ở mức cao để chiếm dụng I2C bus. Sau đó master tạo ra tín hiệu xung clock, và đồng thời gửi một byte đầu tiên chứa bảy bit địa chỉ của thiết bị slave và một bit direction. Thiết bị master phải đảm bảo dữ liệu trên SDA phải ổn định khi SCL ở mức cao. Điều này cũng

có nghĩa là master chỉ có thể đặt dữ liệu lên SDA khi SCL đang ở mức thấp. Ngoài ra việc thay đổi SDA khi SCL đang ở mức thấp cũng đảm bảo rằng các tín hiệu start, stop, restart không bị nhầm lẫn là dữ liệu được truyền trên đường truyền.



Hình 3.21 Giản đồ thời gian khi truyền dữ liệu của I2C master

Sau khi truyền xong tám bit đầu tiên, thiết bị master nhả đường truyền SDA để nhận bit ACK từ slave. Thiết bị slave được chỉ định nên kèo SCL xuống mức thấp để tạo ra tín hiệu ACK trong xung CLK thứ chín vì master đọc bit ACK này để xác định sự tồn tại của thiết bị slave có địa chỉ được chỉ định. Thiết bị master tiếp tục truyền hai byte dữ liệu và nhận tín hiệu ACK từ slave sau khi truyền xong mỗi byte. Cuối cùng master tạo ra tín hiệu stop (SDA thay đổi từ mức logic thấp lên mức logic cao khi SCL đang ở mức 1) để kết thúc quá trình truyền cũng như chuyển I2C bus về trạng thái nghỉ.

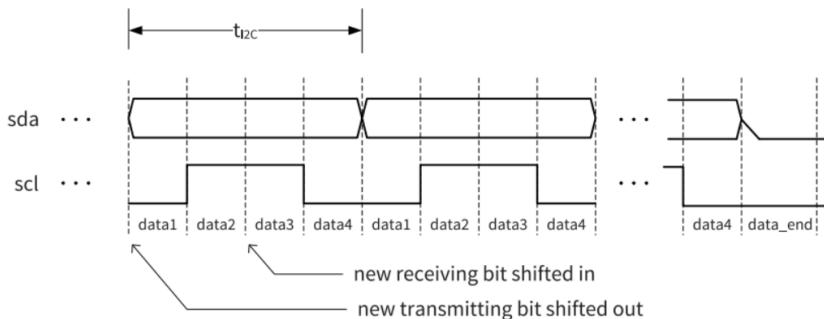
3.5.2 Thiết kế phần cứng

Vì chuẩn I2C không chỉ định số lượng byte được truyền đi trong một lần trao đổi dữ liệu, nên khôi I2C chỉ hỗ trợ một số thao tác trong quá trình trao đổi dữ liệu. Lập trình viên phầm mềm phải viết driver để phối hợp các thao tác này lại với nhau để có thể truyền và nhận dữ liệu tới thiết bị slave. Các thao tác này bao gồm:

1. Viết tám bit và kiểm tra ack bit.
2. Đọc tám bit và gửi đi bit ack hoặc nack.
3. Tạo ra tín hiệu khởi đầu quá trình truyền/nhận dữ liệu.
4. Tạo ra tín hiệu kết thúc quá trình truyền/nhận dữ liệu.

5. Tạo ra tín hiệu bắt đầu lại quá trình truyền/nhận dữ liệu.

Mỗi hành động này có thể được chia thành nhiều giai đoạn khác nhau. Các giai đoạn của các tín hiệu khởi đầu, kết thúc và bắt đầu lại được thể hiện như ở trong Hình 3.20. Giả sử tần số của I2C là f_{i2c} và có chu kỳ $t_{i2c} = \frac{1}{f_{i2c}}$, mỗi giai đoạn của các tín hiệu này chiếm $\frac{t_{i2c}}{2}$, và đều chiếm một chu kỳ của I2C.



Hình 3.22 Các giai đoạn của quá trình truyền dữ liệu

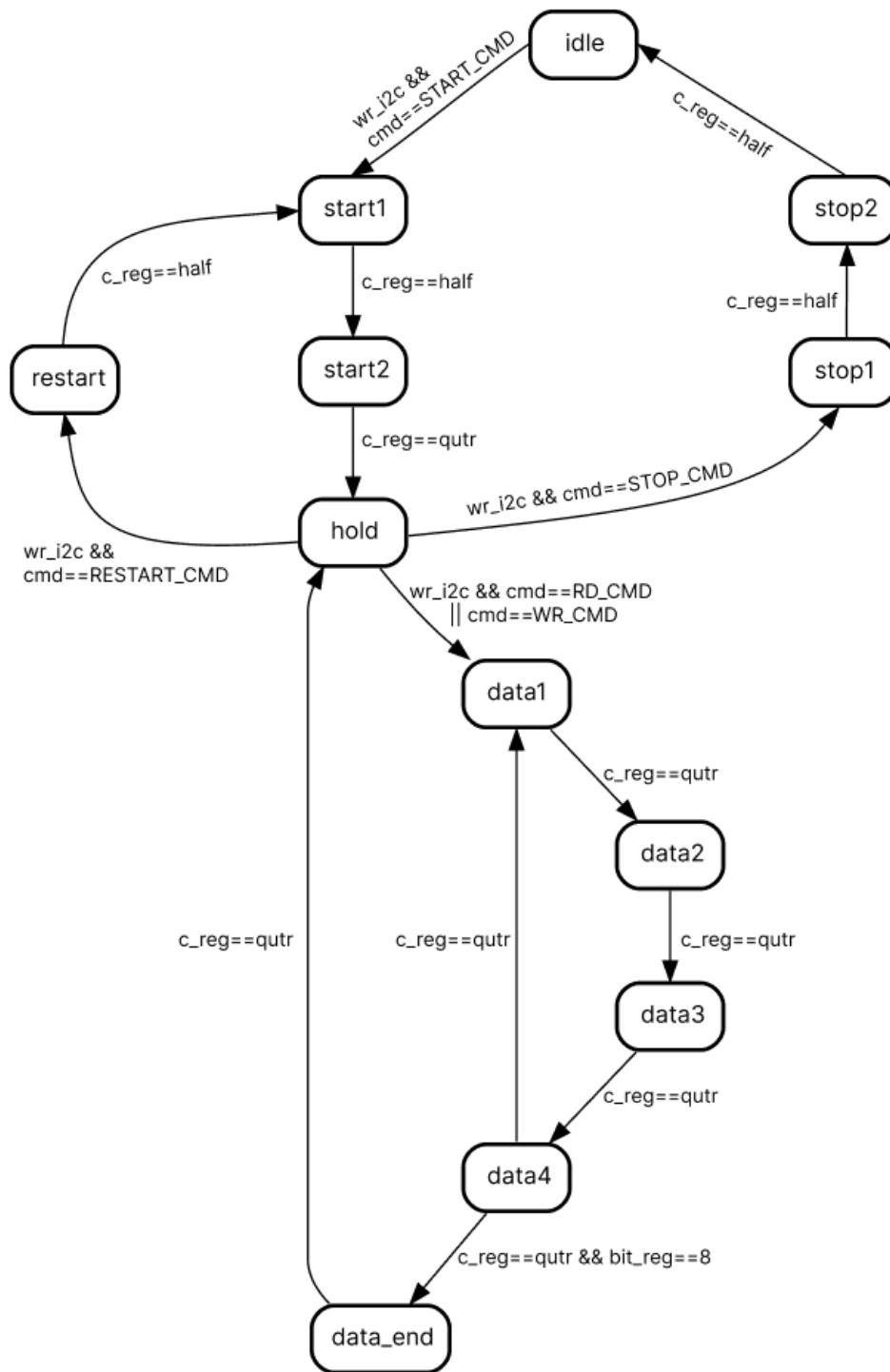
Việc đọc/ghi một bit dữ liệu trên đường truyền I2C diễn ra trong một chu kỳ t_{i2c} và được chia làm bốn giai đoạn có thời gian bằng nhau: data1, data2, data3, data4. Dữ liệu được truyền đi tại thời điểm bắt đầu của data1 và được đọc ở thời điểm chuyển tiếp giữa data2 và data3. Sau khi truyền/nhận xong chín bit các đường scl và sda được kéo xuống mức thấp (data_end) trước khi chuyển sang hành động tiếp theo.

Máy trạng thái của khối I2C được thể hiện ở trong Hình 3.23. Tín hiệu cmd chỉ định hành động một trong năm hành động mà khối I2C cần thực hiện. Máy trạng thái sử dụng nhiều trạng thái khác nhau để mô tả lại các giai đoạn của từng hành động. Các tín hiệu scl và sda được tạo ra dựa trên các trạng thái này. Quá trình trao đổi dữ liệu được bắt đầu bởi START_CMD. Sau đó khối I2C lần lượt chuyển qua các trạng thái start1 và start2 để tạo ra tín hiệu khởi đầu. Các lệnh WR_CMD và RD_CMD được thực hiện ngay sau đó. Cả hai hành động đọc và ghi đều sử dụng các trạng thái data1, data2, data3, data4 chín lần để xử lý các bit dữ liệu hoặc bit ack. Sau đó khối I2C chuyển về trạng thái data_end.

Mỗi khi hoàn thành xong một lệnh (trừ lệnh STOP_CMD) khối I2C chuyển về trạng thái hold và đợi lệnh tiếp theo. Ở trạng thái này cả hai tín hiệu scl và sda

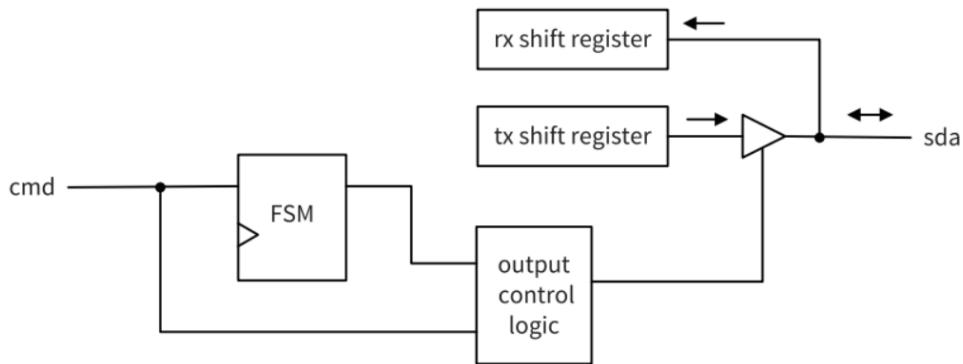
đều ở mức thấp. Khối I2C cũng kích hoạt tín hiệu ready khi ở trong trạng thái idle và hold để báo hiệu rằng nó sẵn sàng nhận lệnh mới.

Các lệnh STOP_CMD và RESTART_CMD được sử dụng kết thúc qua trình truyền dữ liệu. Điểm khác biệt duy nhất giữa hai lệnh này là quá trình truyền mới được bắt đầu lại ngay sau khi khối I2C nhận được lệnh RESTART_CMD.



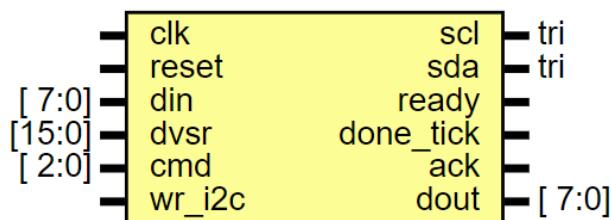
Hình 3.23 Máy trạng thái của khối I2C

Dữ liệu truyền hoặc nhận trong khi trao đổi dữ liệu được lưu trữ trong hai thanh ghi dịch (một dùng để nhận dữ liệu - rx, một dùng để gửi dữ liệu - tx) có độ rộng chín bits (tám bits dữ liệu + một bit ack). Thanh ghi dịch rx nhận dữ liệu ở thời điểm chuyển tiếp giữa trạng thái data2 và data3. Ngược lại mỗi bit của thanh ghi tx được dịch từng bit một ở thời điểm đầu tiên của trạng thái data1. Trong quá trình ghi dữ liệu mạch điều khiển đầu ra kích hoạt bộ đếm ba trạng thái để truyền tám bit đầu của thanh ghi tx. Ở bit thứ chín bộ đếm ba trạng thái không được kích hoạt để thiết bị slave gửi tín hiệu ack. Tín hiệu ack này được lưu ở bit cuối cùng của thanh ghi rx.



Hình 3.24 Sơ đồ khái niệm của mạch điều khiển các thanh ghi dịch dữ liệu

Tương tự như vậy trong quá trình đọc tám bit đầu tiên mạch điều khiển đầu ra không kích hoạt bộ đếm ba trạng thái để tạo điều kiện cho thiết bị slave gửi dữ liệu tới thanh ghi rx của khối I2C core. Ở bit thứ chín bộ đếm ba trạng thái được kích hoạt và tín hiệu ack được gửi đi từ thanh ghi tx tới thiết bị slave thông qua đường sda.



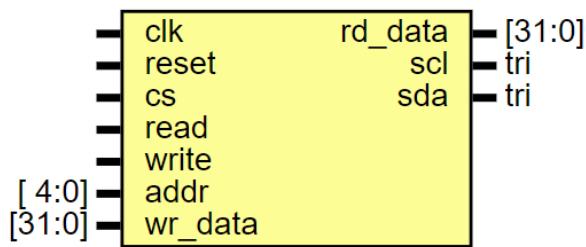
Hình 3.25 Khối I2C

Để có thể tạo ra được tần số thích hợp khói I2C sử dụng một cỗng dvsr để nhận vào số chia để chia được một phần tư chu kỳ t_{i2c} từ tần chu kỳ hoạt động của hệ thống:

$$dvsr = \frac{f_{sys}}{4 \times f_{i2c}} - 1 \quad (3.2)$$

Dữ liệu từ cổng din được truyền đi khi thực hiện thao tác ghi dữ liệu. Trong thao tác đọc dữ liệu LSB (din[0]) được sử dụng để truyền tín hiệu ack tới thiết bị tớ. Tín hiệu wr_i2c được sử dụng để chốt dữ liệu tại các cổng cmd và din.

Cổng dout chứa tám bit dữ liệu được từ thiết bị tớ. Tín hiệu ack ở mức thấp khi quá trình viết dữ liệu diễn ra thành công. Tín hiệu done_tick được kích hoạt trong một chu kỳ sau khi đã nhận hoặc gửi chín bit trên sda.



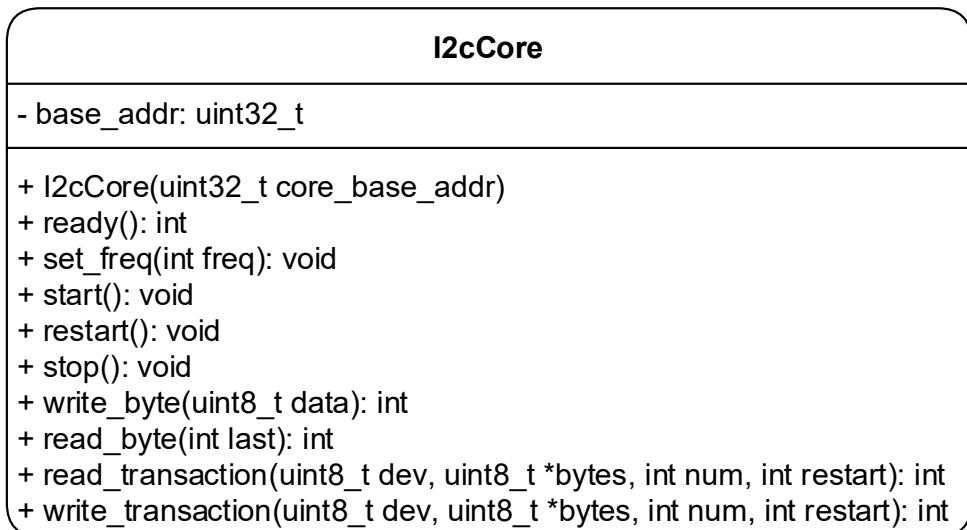
Hình 3.26 Giao diện kết nối của khói I2C

Giao diện kết nối I2C có tác dụng chuyển đổi các tín hiệu mà vi xử lý ghi hoặc đọc từ cách thanh ghi thành các tín hiệu để điều khiển quá trình truyền hoặc nhận dữ liệu của I2C core thông qua bộ giải mã và ghép kênh. Các thanh ghi sau được sử dụng để điều khiển quá trình hoạt động của khói I2C:

- Offset 0 (thanh ghi chỉ đọc):
 - Bit 7-0: chứa 8 bits dữ liệu nhận được của từ thiết bị slave.
 - Bit 8: trạng thái của tín hiệu ready.
 - Bit 9: trạng thái của tín hiệu ack.
- Offset 1 (thanh ghi chỉ ghi):
 - Bit 15-0: chứa hai bytes giá trị của thanh ghi *dvsr*
- Offset 2 (thanh ghi chỉ ghi):
 - Bit 7-0: chứa 8 bit dữ liệu của din
 - Bit 10-8: chứa 3 bit cmd

3.5.3 Thiết kế trình điều khiển

Hàm khởi tạo I2cCore lưu trữ địa chỉ cơ sở và đặt tần số mặc định của tín hiệu scl là 100KHz. Phương thức set_freq dùng để cài đặt tần số xung nhịp của bus I2C. Nó tính toán giá trị chia theo công thức (3.2).



Hình 3.27 Lớp I2cCore

Phương thức ready đọc từ thanh ghi và lấy ra bit ready để kiểm tra xem hệ thống có đang ở trạng thái rảnh hay không. Các phương thức start, restart, và stop chờ cho đến khi khối I2C sẵn sàng rồi gửi các lệnh tương ứng.

Phương thức write_byte ghi một byte dữ liệu và trả về trạng thái. Nó gửi các lệnh cần thiết để gửi một byte dữ liệu ra sda. Phương thức chờ cho đến khi hoàn tất quá trình ghi và sau đó lấy bit xác nhận (acknowledge bit). Nếu thiết bị slave không tạo ra bit xác nhận đúng, phương thức trả về -1 để báo lỗi ghi.

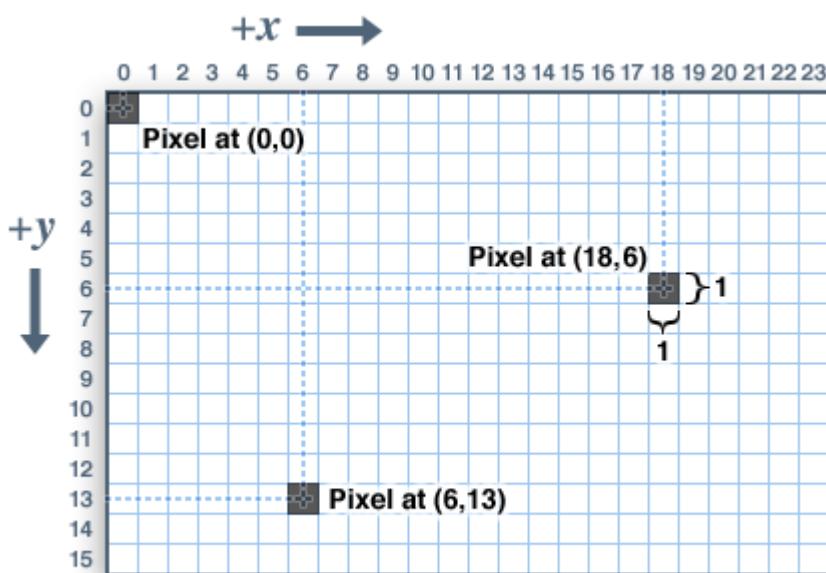
Phương thức read_byte lấy một byte dữ liệu từ thiết bị slave. Tham số last, có giá trị là 0 hoặc 1, dùng để đánh dấu xem thao tác đọc có phải là thao tác cuối trong quá trình đọc dữ liệu hay không.

Phương thức read_transaction và write_transaction thực hiện việc đọc/ghi num byte dữ liệu từ thiết bị slave có địa chỉ dev. Các byte dữ liệu nhận hoặc gửi đi sẽ được lưu trong mảng mà con trỏ bytes trỏ đến. Tham số restart chỉ định tín hiệu restart hoặc stop sẽ được tạo sau khi đọc/ghi dữ liệu trên đường truyền.

CHƯƠNG 4: HỆ THỐNG XỬ LÝ ĐỒ HOẠ

4.1 Tổng quan về màn hình máy tính

Màn hình máy tính là thiết bị hiển thị hình ảnh dựa trên các điểm ảnh (pixel). Mỗi màn hình được cấu thành từ nhiều điểm ảnh nhỏ được sắp xếp theo dạng lưới hai chiều, tạo thành một khung hình đầy đủ. Độ phân giải của màn hình thể hiện số lượng điểm ảnh theo mỗi chiều và thường được biểu diễn dưới dạng chiều rộng x chiều cao. Một hàng ngang của điểm ảnh được gọi là một dòng, và toàn bộ màn hình là một khung hình. Đặc biệt, trực tọa độ theo chiều dọc tăng từ trên xuống dưới như được thể hiện trong Hình 4.1.

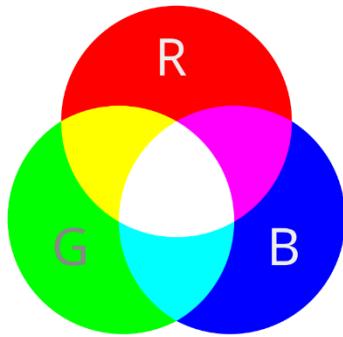


Hình 4.1 Hệ trục tọa độ của màn hình máy tính

Do các hạn chế về mặt phần cứng của bo mạch Arty A7-35T đề tài chỉ sử dụng độ phân giải 640x480. Với độ phân giải này, mỗi khung hình bao gồm 480 dòng theo chiều dọc, mỗi dòng có 640 điểm ảnh theo chiều ngang.

4.1.1 Màu RGB

Màn hình máy tính sử dụng mô hình màu RGB để hiển thị các màu sắc. Theo mô hình này, màu sắc được tạo thành từ ba màu cơ bản là đỏ (Red), xanh lá (Green), và xanh dương (Blue). Bằng cách điều chỉnh cường độ của từng màu cơ bản trong một điểm ảnh, màn hình có thể tái hiện nhiều màu sắc và hình dạng khác nhau.



Hình 4.2 Màu RGB

Số lượng bit được sử dụng để biểu diễn màu sắc của mỗi điểm ảnh được gọi là độ sâu màu (color depth - CD). Nếu mỗi màu cơ bản chỉ sử dụng một bit (tức là màu đó chỉ có thể bật hoặc tắt), thì độ sâu màu là 3 bit và có thể hiển thị tám màu khác nhau ($2^3 = 8$). Tuy nhiên, trong chế độ màu thực (true color), mỗi màu cơ bản được biểu diễn bằng 8 bit, tạo ra độ sâu màu 24 bit.

Nhờ vậy, màn hình có thể hiển thị hơn 16 triệu màu sắc khác nhau (2^{24}). Ngược lại, trong chế độ đơn sắc (monochrome), một bit được sử dụng chung cho cả ba màu cơ bản, cho phép hiển thị hai màu là đen (tất cả các màu đều tắt) và trắng (tất cả các màu đều bật). Để tiết kiệm không gian bộ nhớ sử dụng cung như do giới hạn phần cứng (PmodVGA [8] chỉ hỗ trợ CD=12), trong đề tài này, hệ thống xử lý đồ họa sử dụng độ sâu màu 12 bit.

4.1.2 Tốc độ làm tươi

Một điểm ảnh chỉ có thể giữ được thông tin trong một thời gian rất ngắn và cần được làm tươi liên tục. Dữ liệu hiển thị được truyền đến màn hình theo từng điểm ảnh và từng dòng, cho đến khi hoàn thành một khung hình. Để tránh hiện tượng nhấp nháy (flicker), màn hình cần được làm tươi khoảng 60 lần mỗi giây, tương đương với tần số làm tươi là 60 Hz [8]. Điều này có nghĩa là dữ liệu cho một khung hình phải được truyền đến màn hình trong mỗi 1/60 giây.

4.1.3 VGA



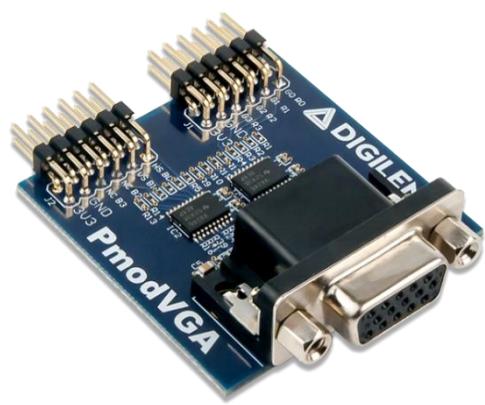
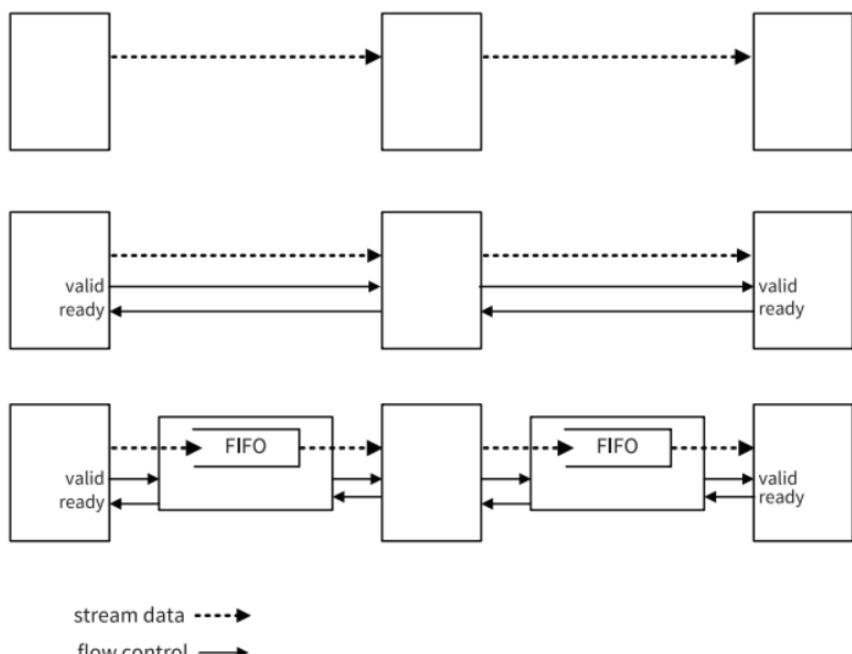
Hình 4.3 Cổng VGA

VGA (Video Graphics Array) là một chuẩn phần cứng hiển thị được giới thiệu lần đầu tiên cùng với máy tính IBM PS/2 vào những năm 1980. Mặc dù phần cứng đồ họa đã phát triển rất nhiều theo thời gian, VGA vẫn được coi là chuẩn tối thiểu và được hầu hết các card đồ họa và màn

hình hỗ trợ. Vì VGA vẫn là lựa chọn phổ biến cho các hệ thống cơ bản và các thiết bị cũ, nên trong đề tài này, hệ thống xử lý đồ họa sử dụng cổng VGA để kết nối và điều khiển việc hiển thị của các điểm ảnh trên màn hình máy tính. Do trên bo mạch Arty A7-35T không có cổng VGA được tính hợp sẵn, nên đề tài này kết nối mô-đun Pmod VGA [8] thông qua hai Pmod header JC và JD.

4.1.4 Giao diện luồng

Trong các hệ thống xử lý tín hiệu và truyền thông, giao diện luồng (stream interface) là một dạng giao diện truyền dữ liệu liên tục tốc độ cao theo một hướng với một định dạng đã được xác định trước đó giữa hai thành phần. Giao diện luồng không cần sử dụng đường địa chỉ, mà thay vào đó chỉ cần một đường dữ liệu và các tín hiệu điều khiển để điều chỉnh tốc độ truyền. Giao diện luồng này đặc biệt phù hợp cho các kết nối liên tục giữa hai điểm cố định, chẳng hạn như kết nối từ hệ thống xử lý đồ họa của FPro đến màn hình máy tính. Có ba phương thức [9] cơ bản để kiểm soát luồng trong giao diện luồng:



Hình 4.4 Pmod VGA

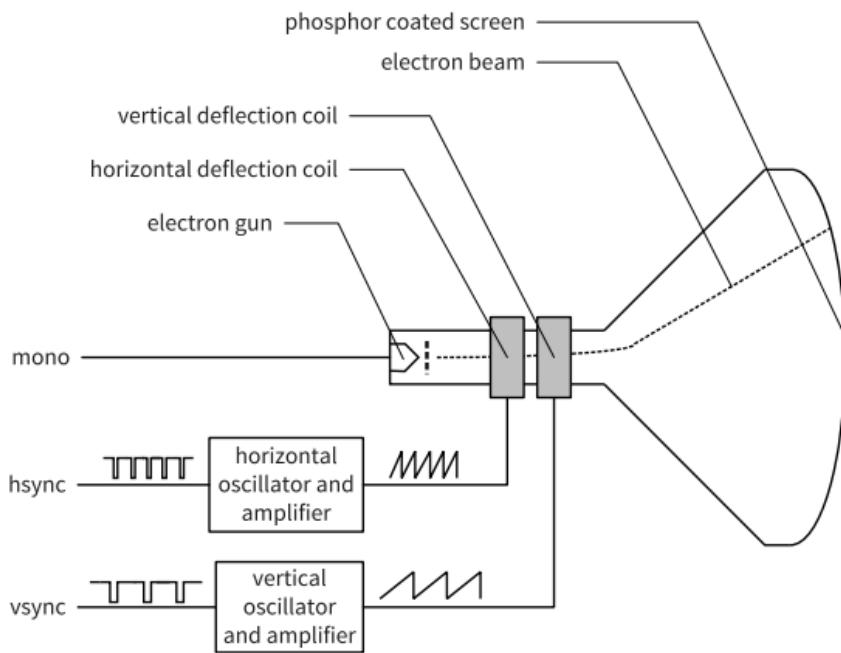
Hình 4.5 Các phương thức kiểm soát luồng trong giao diện luồng

1. Kiểm soát luồng tập trung: Sử dụng một mạch điều khiển trung tâm để quản lý toàn bộ quá trình truyền dữ liệu. Trong cấu hình này, chỉ có một tín hiệu dữ liệu duy nhất giữa thiết bị nguồn và thiết bị đích, và mạch điều khiển trung tâm kiểm tra trạng thái của các thành phần để điều chỉnh tốc độ truyền dữ liệu.
2. Kiểm soát luồng phân tán với phản hồi (back pressure): Sử dụng một giao thức bắt tay đơn giản giữa thiết bị nguồn và thiết bị đích. Trong cấu hình này, cặp tín hiệu “valid” và “ready” được sử dụng. Thiết bị đích kích hoạt tín hiệu “ready” khi nó sẵn sàng nhận dữ liệu mới, trong khi thiết bị nguồn kích hoạt tín hiệu “valid” khi có dữ liệu mới sẵn sàng. Khi đó, thiết bị đích có thể kiểm soát tốc độ tạo dữ liệu của nguồn bằng cách sử dụng phản hồi từ giao thức bắt tay này.
3. Kiểm soát luồng phân tán với bộ đệm FIFO: Trong trường hợp mà thiết bị nguồn cần gửi một lượng lớn dữ liệu theo các đợt ngắn (do việc đọc liên tục dữ liệu tốn ít chi phí và thời gian hơn so với việc đọc từng dữ liệu một) với tốc độ cao hơn so với thiết bị đích, một bộ đệm FIFO có thể được thêm vào giữa thiết bị nguồn và thiết bị đích. Bộ đệm FIFO giúp làm giảm sự mất cân bằng về tốc độ giữa nguồn và thiết bị đích bằng cách tạm lưu trữ các gói dữ liệu. Các tín hiệu trạng thái của FIFO, chẳng hạn như “empty” và “full,” có thể được sử dụng để điều khiển giao thức bắt tay trong giao diện luồng.

4.2 Khối đồng bộ VGA

4.2.1 Sơ lược về màn hình CRT

Chuẩn VGA ban đầu được thiết kế dành cho màn hình CRT (cathode ray tube), dù công nghệ CRT hiện nay không còn được sử dụng rộng rãi. Tuy nhiên, việc nghiên cứu cách thức hoạt động cơ bản của CRT sẽ giúp hiểu rõ hơn về nguyên lý hoạt động của khối đồng bộ VGA.



Hình 4.6 Sơ đồ nguyên lý của màn hình CRT

Trong màn hình CRT, súng điện tử (cathode) tạo ra một chùm tia điện tử hội tụ, truyền qua một ống chân không và cuối cùng đánh vào màn huỳnh quang (phosphorescent screen). Ánh sáng được phát ra tại điểm mà các điện tử chạm vào màn hình, và cường độ sáng phụ thuộc vào mức điện áp của tín hiệu video đầu vào (mono). Tín hiệu này là tín hiệu tương tự (analog), với mức điện áp nằm trong khoảng từ 0 đến 0,7 V.

Hai cuộn dây lệch (deflection coils) dọc và ngang bên ngoài ống CRT tạo ra các trường từ để điều khiển chùm điện tử di chuyển, xác định vị trí mà chùm điện tử tác động lên màn hình. Chùm điện tử quét màn hình từ trái sang phải, từ trên xuống dưới theo một quy luật nhất định.

4.2.2 Tín hiệu điều khiển và đồng bộ

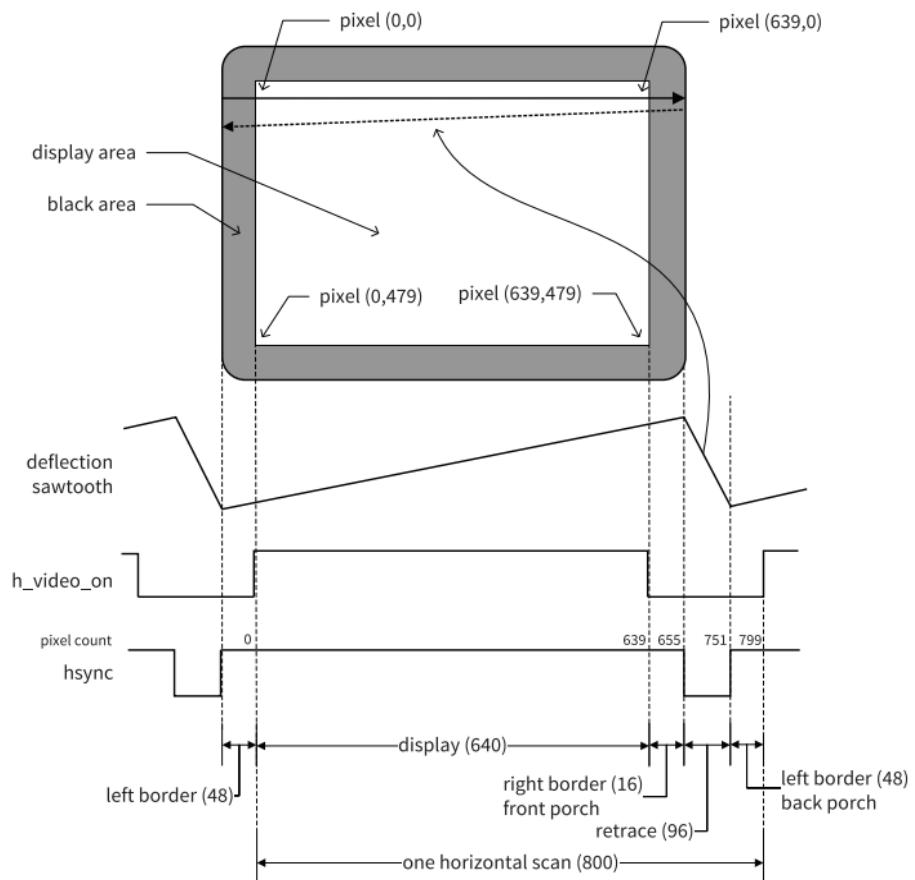
Khối VGA sync core có chức năng tạo ra các tín hiệu đồng bộ ngang (hsync) và đồng bộ dọc (vsync). Hai tín hiệu này kiểm soát thời gian cần thiết để chùm tia điện tử quét hết một dòng và toàn bộ màn hình.

Để có thể đồng bộ hóa dữ liệu điểm ảnh, giao diện VGA sử dụng hai loại tín hiệu số: tín hiệu đồng bộ ngang (hsync) và tín hiệu đồng bộ dọc (vsync). Hai tín hiệu này điều khiển các bộ dao động và khuếch đại bên trong của màn hình để

tạo ra các sóng răng cưa để điều khiển hai cuộn dây lệch giúp định hướng chùm tia điện tử chiếu lên trên màn hình.

Một chu kỳ của tín hiệu hsync tương đương với thời gian chùm tia điện tử quét qua 800 điểm ảnh [9] và có thể được chia thành bốn giai đoạn:

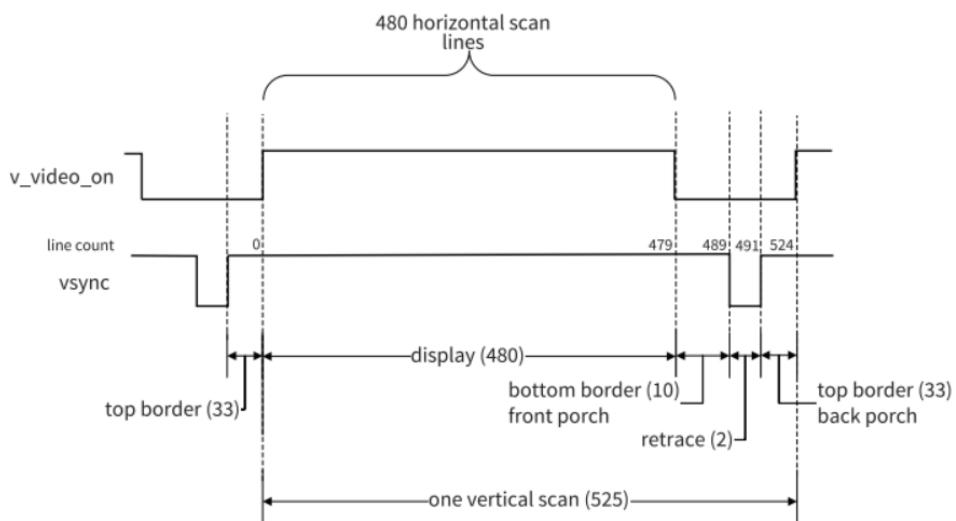
1. Hiển thị: giai đoạn điểm ảnh được hiển thị trên màn hình, kéo dài 640 điểm ảnh.
2. Quét ngược (Retrace): giai đoạn để chùm tia điện tử trở lại mép trái màn hình. Tín hiệu video bị vô hiệu hóa (tức là màn hình hiển thị đen), dài 96 điểm ảnh.
3. Biên phải (left border/back porch): giai đoạn chùm tia quét qua biên bên phải của vùng hiển thị, dài 16 điểm ảnh.
4. Biên trái (right border/front porch): giai đoạn chùm tia quét qua biên bên trái của vùng hiển thị, dài 48 điểm ảnh.



Hình 4.7 Giản đồ thời gian của các tín hiệu điều khiển chùm tia điện tử quét theo chiều ngang

Giản đồ thời gian của tín hiệu hsync được thể hiện như trong Hình 4.7. Trong đó việc hiển thị điểm ảnh trên màn hình được điều khiển bởi tín hiệu *h_video_on*. Tín hiệu này chỉ được kích hoạt ở trong giai đoạn hiển thị.

Tín hiệu vsync điều khiển quá trình quét dọc, đảm bảo tia điện tử di chuyển từ đỉnh xuống đáy và trở lại đỉnh để bắt đầu quét khung hình mới. Một chu kỳ của tín hiệu vsync tương đương với thời gian chùm tia điện tử quét qua 525 dòng [9] và được chia thành các giai đoạn tương tự như tín hiệu hsync:



Hình 4.8 Giản đồ thời gian của các tín hiệu điều khiển chùm tia điện tử quét theo chiều dọc

1. Hiển thị: Khu vực hiển thị trên màn hình, kéo dài 480 dòng. Tín hiệu *v_video_on* chỉ được kích hoạt trong giai đoạn này để chỉ định việc hiển thị hình ảnh của màn hình.
2. Quét ngược (Retrace): Giai đoạn chùm tia điện tử trở lại đỉnh của màn hình, kéo dài 2 dòng.
3. Biên dưới (Bottom Border/Front Porch): Giai đoạn chùm tia điện tử di chuyển ở biên dưới của vùng hiển thị, dài 10 dòng.
4. Biên trên (Top Border/Back Porch): Giai đoạn chùm tia điện tử di chuyển ở biên trên của vùng hiển thị, dài 33 dòng.

4.2.3 Tốc độ truyền điểm ảnh và dữ liệu

Dữ liệu điểm ảnh cần được truyền với tốc độ chính xác để đạt được độ phân giải và tần số làm tươi mong muốn.

Các tham số ảnh hưởng tới tốc độ truyền điểm ảnh (pixel clock rate) là:

- p : Số lượng điểm ảnh trong một dòng quét ngang. Với độ phân giải 640x480, $p = 800$ điểm ảnh/dòng.
- l : Số dòng trong một khung hình. Với độ phân giải 640x480, $l = 525$ dòng/màn hình.
- s : Số lần làm tươi màn hình mỗi giây, thường đặt ở $s = 60$ màn hình/giây để giảm hiện tượng nháy nháy và tạo chuyển động mượt mà.

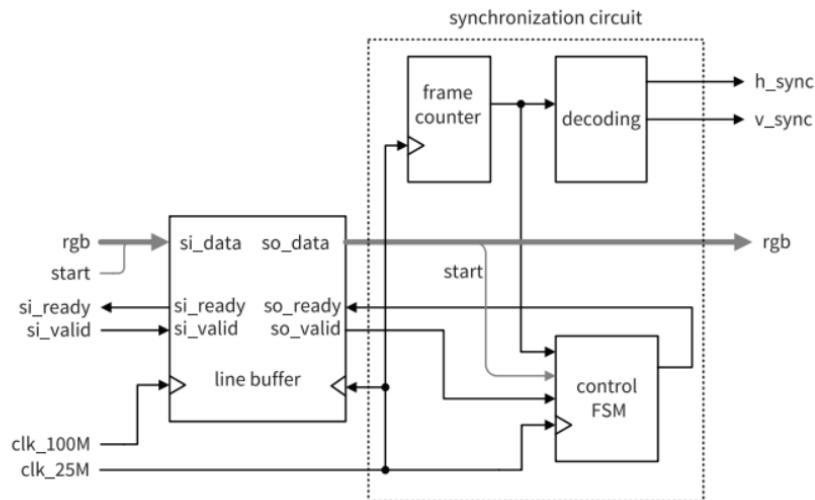
Dựa vào các tham số này, tốc độ truyền điểm ảnh có thể được tính như sau:

$$\text{pixel clock rate} = p \times l \times s \approx 25\text{MHz} \quad (4.1)$$

Tốc độ truyền dữ liệu thấp hơn vì hệ thống chỉ cần truyền dữ liệu khi ở trong giai đoạn hiển thị:

$$\text{pixel data rate} = 640 \times 480 \times 60 \approx 18.4\text{MHz} \quad (4.2)$$

4.2.4 Thiết kế phần cứng



Hình 4.9 Sơ đồ khối của khối đồng bộ VGA

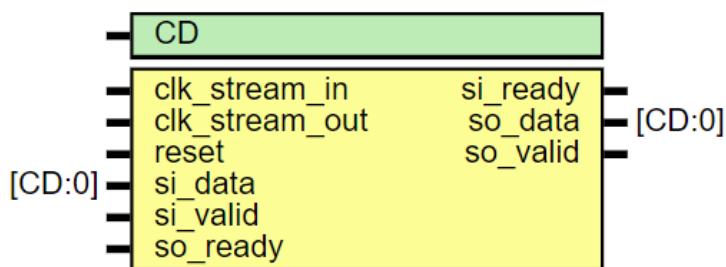
Hệ thống xử lý đồ họa sử dụng năm tín hiệu trong cổng VGA để kết nối với màn hình máy tính (qua Pmod VGA) gồm:

- Hai tín hiệu đồng bộ hsync và vsync.
- Ba tín hiệu tương tự về cường độ màu sắc của các màu cơ bản như đỏ, xanh lá cây và xanh l López biển. Các tín hiệu tương tự này được

chuyển đổi từ tín hiệu số thông qua bộ chuyển đổi DAC có sẵn trên PmodVga.

4.2.4.1 Bộ đếm dòng

Khối đồng bộ VGA có nhiệm vụ cung cấp các điểm ảnh tới màn hình máy tính với tốc độ 25MHz. Do tốc độ này có thể thay đổi tuỳ thuộc vào độ phân giải cũng như tần số quét của màn hình nên nó hoạt động ở một tần số khác so với tần số của nền tảng FPro. Để giải quyết vấn đề này, hệ thống xử lý đồ họa sử dụng khối clock management IP (xem chi tiết ở phần phụ lục) do Vivado cung cấp để lấy tín hiệu xung nhịp cho bộ khói đồng bộ VGA. Để có thể truyền được luồng dữ liệu điểm ảnh giữa hai miền tần số hoạt động, hệ thống xử lý đồ họa sử dụng phương pháp kiểm soát luồng phân tán với bộ đếm FIFO. Ngoài dữ liệu về điểm ảnh, bộ đếm FIFO còn cung cấp thêm một bit dữ liệu start để chỉ định rằng điểm ảnh hiện tại có phải là điểm ảnh đầu tiên trong khung hình (khi bộ đếm ngang và đọc đều có giá trị là 0) hay không. Dung lượng của bộ đếm thường có sức chứa lớn hơn lượng dữ liệu điểm ảnh trên một dòng của màn hình. Vì dữ liệu hình ảnh chỉ được truyền đi ở giai đoạn hiển thị nên các khói IP có thể tận dụng được khoảng thời gian ở các giai đoạn khác để tạo và xử lý các điểm ảnh một cách liên tục. Do đó, bộ đếm FIFO có độ rộng 13 bits (12 bit độ sâu màu + 1 start bit) và có ít nhất $\lceil \log_2 640 \rceil = 10$ bus địa chỉ.



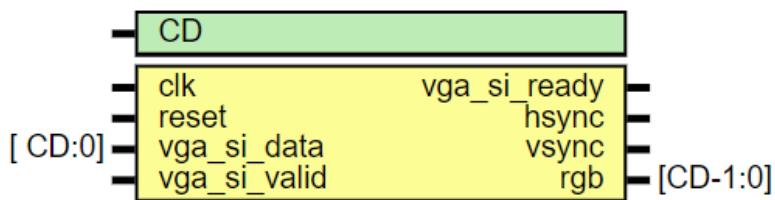
Hình 4.10 Mô-đun bộ đếm dòng

FIFO buffer sử dụng hai tín hiệu xung nhịp khác nhau. Trong đó việc ghi dữ liệu sử dụng tín hiệu xung nhịp có tần số 100MHz. Việc đọc dữ liệu sử dụng xung nhịp 25MHz. Để ổn định việc đọc ghi dữ liệu, khói đồng bộ VGA cấu hình một mô-đun 18Kb BRAM (Block Random Access Memory) thành một bộ đếm FIFO có độ rộng dữ liệu là 13 bit và 10 bit địa chỉ. Chi tiết về quá trình cấu hình

IP được trình bày trong phần phụ lục. Hình 4.10 mô tả giao diện kết nối của mô-đun 18Kb BRAM, mô-đun này có tác dụng cấu hình BRAM và ánh xạ các tín hiệu cần thiết của khối IP về với stream interface.

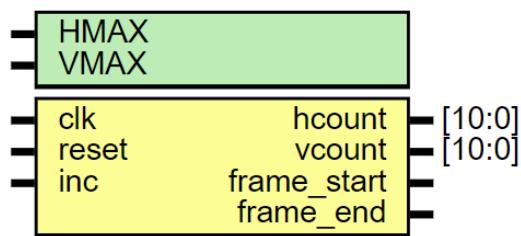
Tín hiệu *si_ready* được kích hoạt khi bộ đệm FIFO có đủ không gian để lưu trữ dữ liệu. Tín hiệu có tác dụng yêu cầu các khối video IP tạo ra dữ liệu điểm ảnh mới. Một khi đã có dữ liệu mới, khối video IP kết nối với bộ đệm dòng cần phải kích hoạt tín hiệu *si_valid* để ghi vào bộ đệm FIFO. Tương tự như vậy, tín hiệu *so_valid* được kích hoạt mỗi khi bộ đệm FIFO có dữ liệu. Mạch đồng bộ kích hoạt tín hiệu *so_ready* mỗi khi nó đọc xong dữ liệu.

4.2.4.2 VGA sync



Hình 4.11 Mô-đun VGA sync

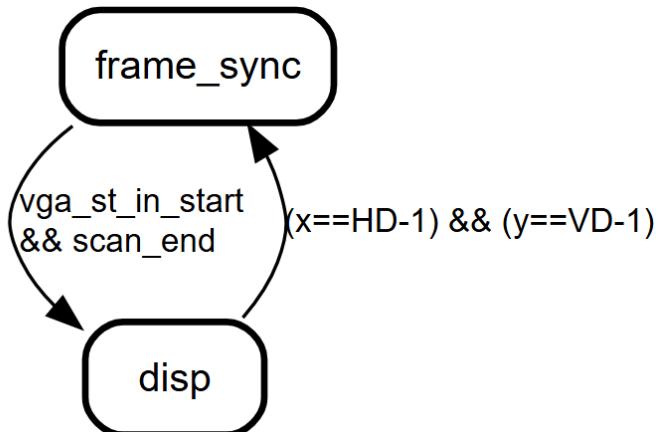
Mô-đun VGA sync có tác dụng lấy dữ liệu từ bộ đệm dòng cũng như tạo ra các tín hiệu đồng bộ hsync và vsync để điều khiển việc hiển thị của màn hình máy tính. Mô-đun này sử dụng cờ start (được truyền cùng với các dữ liệu về màu sắc của điểm ảnh) để đồng bộ dữ liệu điểm ảnh với các tín hiệu hsync và vsync. Để tạo ra các tín hiệu hsync và vsync như Hình 4.7 và Hình 4.8, mô-đun VGA sync sử dụng khung đếm (gồm hai bộ đếm ngang và dọc). Để thuận tiện cho việc xác định vị trí của điểm ảnh, các bộ đếm này sẽ được căn chỉnh để bắt đầu từ 0 ở thời điểm bắt đầu của giai đoạn hiển thị. Với độ phân giải 640x480, bộ đếm ngang và dọc lần lượt là các bộ đếm mod 800 (VMAX) và mod 525 (VMAX). Các bộ đếm chỉ hoạt động khi tín hiệu *inc* được thiết lập ở mức cao. Ngoài ra có các tín hiệu *frame_start*, và *frame_end* lần lượt được sử dụng để báo hiệu sự bắt đầu hoặc kết thúc của một khung hình. Dữ liệu về điểm ảnh chỉ được truyền đi khi giá trị của các bộ đếm ngang nhỏ hơn 640 và bộ đếm dọc nhỏ hơn 480. Với các giá trị khác mô-đun VGA sync truyền đi giá trị 0.



Hình 4.12 Mô-đun bộ đếm khung

Mạch giải mã tạo ra tín hiệu *hsync*, và *vsync* dựa trên giá trị của các bộ đếm. Tín hiệu *hsync* ở mức thấp khi giá trị của bộ đếm ngang nằm trong khoảng từ 656 tới 751 như được thể hiện trong Hình 4.7. Tín hiệu *vsync* ở mức thấp khi giá trị của bộ đếm dọc có giá trị 490 hoặc 491 Hình 4.8.

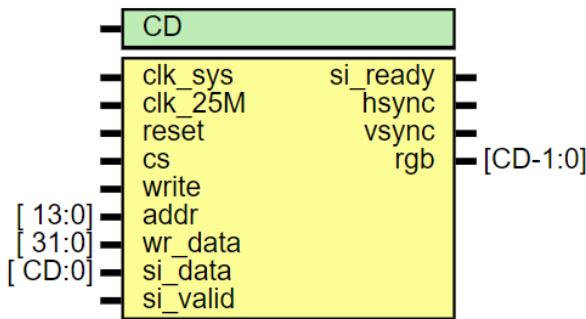
Mô-đun VGA sync sử dụng một máy trạng thái (Hình 4.13) để đồng bộ điểm ảnh đầu tiên của một khung hình (được lấy từ bộ đếm dòng) với giá trị của bộ đếm khung. Khi ở trạng thái *frame_sync*, mô-đun đợi cho đến khi một frame mới bắt đầu (giá trị của bộ đếm ngang và dọc là 0) và giá trị đọc được từ bộ đếm dòng là giá trị của điểm ảnh đầu tiên để chuyển sang trạng thái *disp*. Trong các trường hợp còn lại, mô-đun VGA sync liên tục đọc các giá trị từ bộ đếm dòng cho đến khi tìm được giá trị của điểm ảnh đầu tiên trong một khung hình.



Hình 4.13 Máy trạng thái của mô-đun VGA sync

Khi ở trạng thái *disp*, mô-đun VGA sync lấy các dữ liệu về điểm ảnh từ bộ đếm dòng khi và chỉ khi các giá trị của các bộ đếm ngang và dọc nằm trong vùng hiển thị. Sau khi nhận và truyền điểm ảnh cuối cùng, mô-đun quay trở về trạng thái *frame_sync*.

4.2.4.3 Giao diện kết nối VGA sync



Hình 4.14 Mô-đun VGA sync wrapper

Mô-đun VGA sync wrapper được sử dụng để kết nối hai mô-đun bộ đệm dòng và mô-đun VGA sync với FPro bus, các tín hiệu của mô-đun này được thể hiện như trong Hình 4.14.

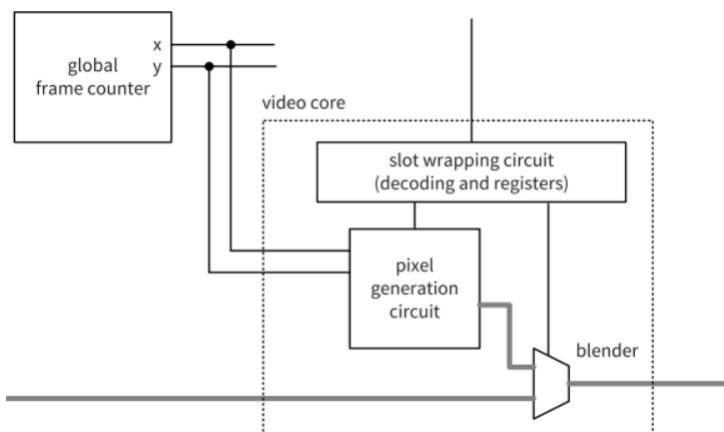
4.3 Khối video IP

Về cơ bản các khối video IP có cùng một kiến trúc vào giao diện kết nối. Đầu vào của các khối IP này là một luồng điểm ảnh. Dựa trên đầu vào này, các khối video IP sẽ thực hiện tính toán hoặc thêm nội dung mới vào luồng này và xuất ra luồng điểm ảnh đã được xử lý. Để kết hợp dữ liệu được sinh ra từ các khối lại với nhau thành một khung hình duy nhất hệ thống xử lý đồ họa sẽ sử dụng phép trộn (blending operation). Phép trộn thường được thực hiện trên hai khung hình xếp chồng lên nhau, trong đó khung hình phía dưới được gọi là background, khung hình ở trên background được gọi là foreground. Trong phạm vi của đồ án, hai phép trộn sau sẽ được sử dụng:

- Multiplexing (Binary Alpha Blending): chọn một điểm ảnh từ một trong hai nguồn video và định tuyến nó đến đầu ra giống như cách mạch chọn kênh hoạt động.
- Chroma-Key Blending: chỉ chọn các điểm ảnh của foreground có giá trị khác với một màu được thiết lập từ trước hay còn được gọi là chroma key. Khi kết hợp các điểm ảnh của hai khung hình lại với nhau, các điểm ảnh có giá trị chroma key của foreground được thay thế bằng các điểm ảnh tương ứng của background.

Cấu tạo của một khối video IP (Hình 4.15) bao gồm các thành phần sau:

- Mạch tạo điểm ảnh: tạo ra các điểm ảnh dựa trên tọa độ x, y (được tạo ra từ bộ đếm khung) và giá trị của điểm ảnh đầu vào.
- Blender: thực hiện phép trộn luồng điểm ảnh gốc với điểm ảnh đã được xử lý. Tất cả các khối video IP đều có một thanh ghi (bypass) ở địa chỉ 0x2000 được sử dụng để truyền trực tiếp luồng dữ liệu điểm ảnh ở đầu vào tới đầu ra mà không có bất kỳ sự thay đổi nào bằng cách sử dụng phép trộn multiplexing.
- Video Slot Wrapping Circuit: cung cấp giao diện kết nối giữa video core và video controller giúp vi xử lý có thể ghi dữ liệu vào các khối IP để điều khiển quá trình xử lý điểm ảnh.

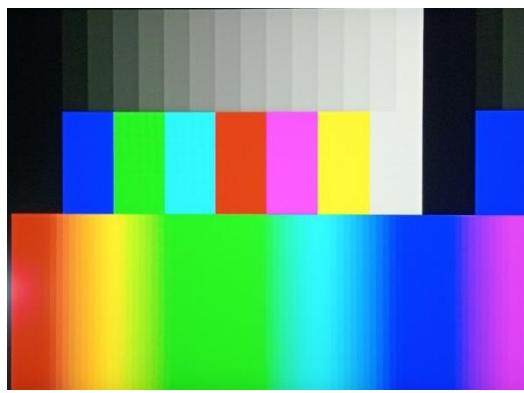


Hình 4.15 Cấu tạo của khối video IP

Để đảm bảo các khối IP có thể kết nối và hoạt động được với nhau, luồng dữ được ra từ các pixel generation circuit phải có độ trễ giống nhau và bằng với khối có độ trễ lớn nhất. Trong hệ thống xử lý đồ họa, text core có độ trễ lớn nhất là hai xung nhịp đồng hồ. Do đó, các khối có độ trễ nhỏ hơn cần phải đệm dữ liệu đầu ra của chúng bằng các bảng một số thanh ghi thích hợp.

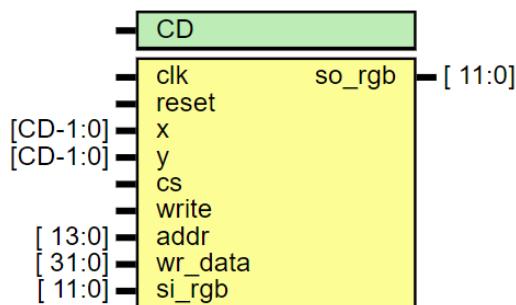
4.3.1 Khối kiểm tra màn hình

Khối kiểm tra màn hình có tác dụng tạo ra một khung (Hình 4.16) chứa một dải các màu minh họa khả năng hiển thị các màu sắc của màn hình. Khối sẽ chia màn hình thành ba hàng, trong đó:



Hình 4.16 Khung hình được tạo ra từ khói kiểm tra màn hình

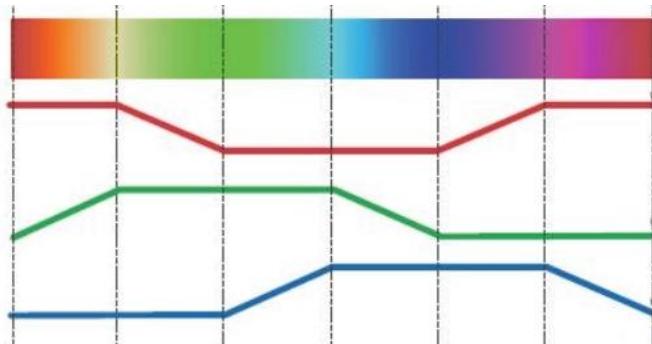
- Hàng thứ nhất chứa các sắc thái của màu xám (có cường độ các màu đỏ, xanh lá, xanh lóp biển giống nhau). Với CD = 12, mỗi màu sử dụng bốn bit để thể hiện cường độ sáng, do đó có tất cả $2^4 = 16$ sắc thái khác nhau của màu xám.
- Hàng thứ hai chứa các màu cơ bản như đỏ, xanh lá, xanh lóp biển và các màu thứ cấp (được tạo ra bằng cách kết hợp các màu cơ bản lại với nhau).
- Hàng thứ ba chứa phổ cầu vòng. Do CD=12 nên hệ thống xử lý đồ họa có khả năng hiển thị tất cả $2^{12} = 4096$ màu sắc khác nhau.



Hình 4.17 Mô-đun khói kiểm tra màn hình

Khối kiểm tra màn hình tạo ra màu sắc cho các điểm ảnh dựa trên tọa độ của chúng. Cụ thể, tọa độ y được sử dụng để chia khung hình thành ba hàng. Ở hàng thứ nhất, các bit của x[8:5] được gán cho các bit màu đỏ, xanh lá và xanh dương. Đối với hàng thứ hai, bit x[8] được gán cho 4 bit của màu đỏ, bit x[7] cho 4 bit của màu xanh lá, và bit x[6] cho 4 bit của màu xanh dương. Ở hàng cuối cùng, các bit của x[9:7] được sử dụng để chia màn hình thành các vùng với giá trị

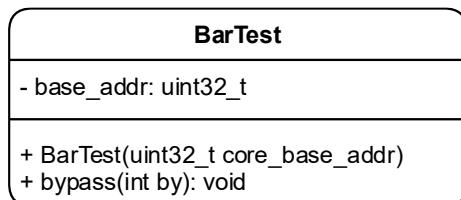
cực đại (0xF), cực tiêu (0x0), hoặc tăng/giảm (bằng cách gán giá trị là $x[6:3]$ hoặc $\sim x[6:3]$) cho các bit màu đỏ, xanh lá và xanh dương, như thể hiện trong Hình 4.18.



Hình 4.18 Cường độ sáng của các màu trong phổ cầu vồng

Vì sử dụng mạch tổ hợp để tạo luồng dữ liệu đầu ra dựa trên vị trí của các điểm ảnh, nên cần phải thêm hai thanh ghi để tạo độ trễ hai xung clock. Ngoài ra, do khung hình được tạo ra bởi khói kiểm tra màn hình chủ yếu chỉ dùng để kiểm tra khả năng hiển thị của màn hình và dữ liệu đầu ra của nó thường không cần phải trộn với bất kỳ khung hình nào khác, một mạch ghép kênh được sử dụng để chọn giữa điểm ảnh của khói IP video phía trước hoặc điều ảnh từ khói kiểm tra màn hình.

Khối kiểm tra màn hình có thể được điều khiển bằng cách sử dụng lớp BarTest có cấu trúc như được thể hiện ở trong Hình 4.19. Trong đó phương thức bypass được sử dụng chọn hoặc bỏ qua các điểm ảnh được tạo ra bởi khói này.

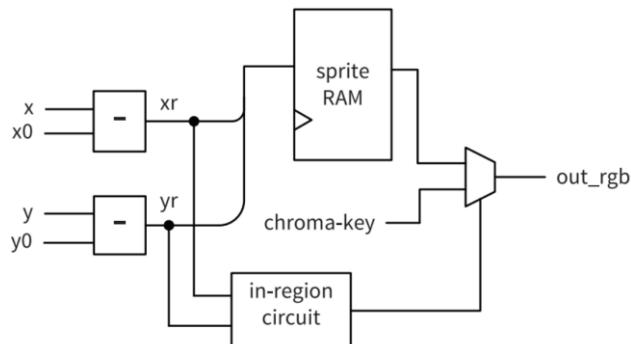


Hình 4.19 Cấu trúc lớp BarTest

4.3.2 Spritecore

Trong đồ họa máy tính, một sprite là một hình ảnh bitmap nhỏ có vùng hiển thị chỉ chiếm một phần nhỏ của khung hình. Sprite có thể được thực hiện thông qua phần cứng hoặc phần mềm. Hệ thống xử lý đồ họa sử dụng phương pháp phần cứng để tạo ra một khói video sprite tích hợp chúng lên một khung hình bằng phương pháp trộn chroma-key blending. Các ứng dụng chính của sprite bao gồm

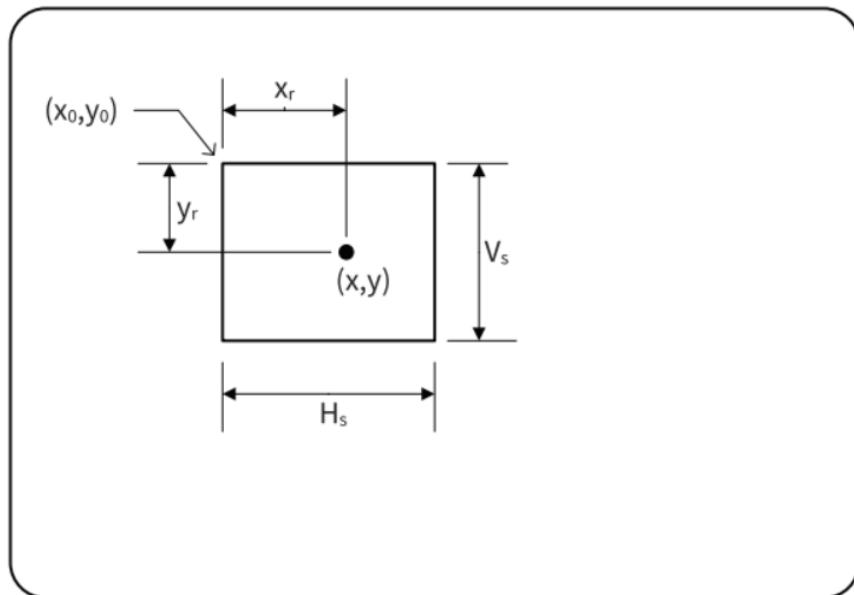
việc tích hợp các đối tượng đồ họa nhỏ vào cảnh lớn hơn, như tạo một quả bóng và di chuyển nó trên màn hình.



Hình 4.20 Mạch tạo điểm ảnh của sprite core

Mạch tạo điểm ảnh của sprite core (Hình 4.20) bao gồm:

- Hai mạch trừ được sử dụng để tính toán vị trí tương đối của điểm ảnh cần được xử lý (x, y) với tọa độ của sprite (x_0, y_0) trên màn hình máy tính như được minh họa trong Hình 4.21.



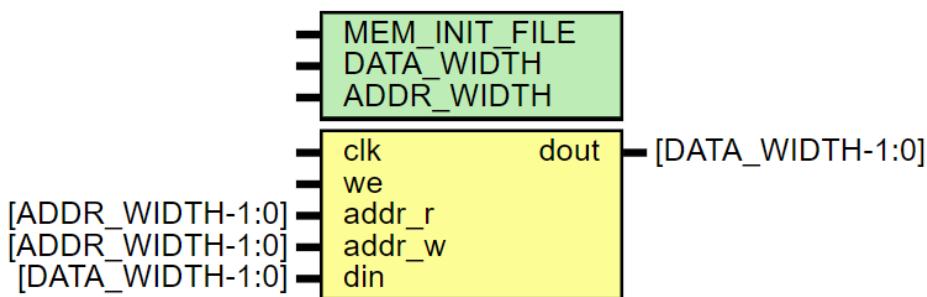
Hình 4.21 Vị trí tương quan giữa tọa độ của điểm ảnh hiện tại và sprite

- Mạch in-region được sử dụng để kiểm tra xem vị trí tương đối được tính toán bởi hai mạch trừ có nằm trong bitmap của sprite hay không.
- Sprite RAM được sử dụng để lưu trữ thông tin về các điểm ảnh của sprite. Độ rộng dữ liệu của sprite RAM bằng với CD của điểm ảnh. Toạ độ của các điểm ảnh trong bitmap của sprite (có kích thước

$H_s \times V_s$) có thể được dùng để xác định địa chỉ của chúng trong RAM theo công thức sau: $addr = y_r * H_s + x_r$. Vì phép nhân trong phần cứng tồn rất nhiều tài nguyên, do đó H_s thường được chọn với giá trị là luỹ thừa của 2 để biến phép nhân thành phép dịch bit:

$$addr = y_r * H_s + x_r = y_r * 2^h + x_r = y_r \ll h + x_r \quad (4.3)$$

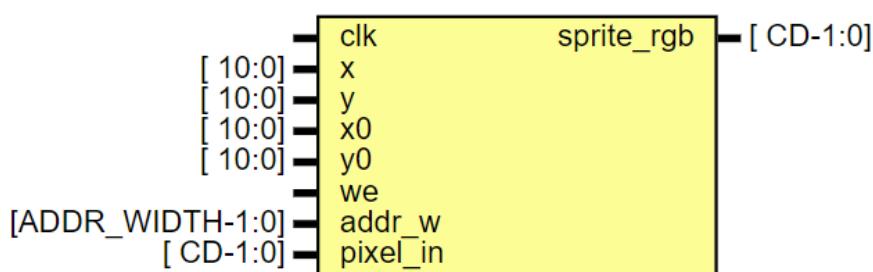
- Mạch ghép kêtch được sử dụng để chọn giữa điểm ảnh của sprite trong RAM khi đầu ra của in-region là 1 và chroma-key trong trường hợp còn lại.



Hình 4.22 Mô-đun sprite_ram

Mô-đun sprite_ram sử dụng ba tham số sau để tạo nên sprite RAM dùng để lưu trữ các thông tin điểm ảnh của một sprite:

- MEM_INIT_FILE chỉ định tệp tin được sử dụng để khởi tạo giá trị ban đầu cho sprite RAM.
- DATA_WIDTH chỉ định độ rộng dữ liệu của dữ liệu được lưu trữ trong RAM
- ADDR_WIDTH chỉ định độ rộng dữ liệu của bit địa chỉ.

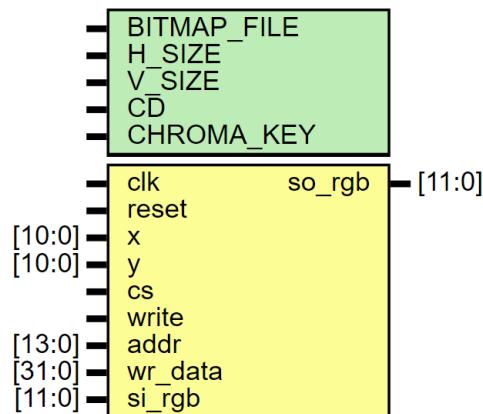


Hình 4.23 Mô-đun sprite core

Mô-đun spritecore sử dụng các tín hiệu x, y, x0, y0 để tính toán việc hiển thị sprite bitmap trên màn hình. Ngoài ra mô-đun này cũng cho phép người dùng

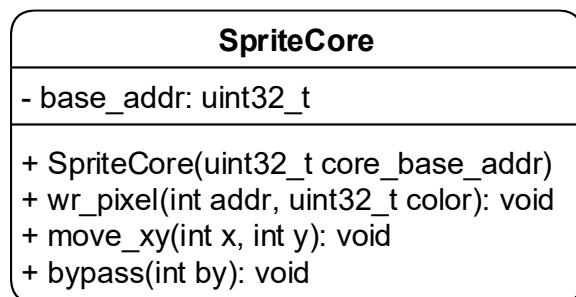
thay đổi giá trị của các điểm ảnh trong RAM bằng cách sử dụng các tín hiệu we, addr_w, và pixel_in.

Cuối cùng mô-đun sprite wrapper được sử dụng để tạo ra một giao diện kết nối giữa sprite core với hệ thống xử lý đồ họa. Vì xử lý có thể điều khiển sprite core bằng cách sử dụng các thanh ghi sau:



Hình 4.24 Mô-đun sprite wrapper

- Ghi vào các bit 11-0 (12 bit RGB) của thanh ghi có địa chỉ 00 00aa aaaa aaaa để thay đổi giá trị của các điểm ảnh được lưu trữ trong RAM. Địa chỉ của một điểm ảnh có thể được tính toán bằng cách sử dụng công thức (4.3).
- Ghi giá trị 1 vào LSB vào 10 0000 0000 0000 để loại bỏ các điểm ảnh được tạo ra bởi sprite core khỏi khung hình.
- Ghi giá trị x0 vào 10 LSB của địa chỉ 10 0000 0000 0001 để thay đổi giá trị x0 của sprite core trên màn hình máy tính.
- Ghi giá trị y0 vào 10 LSB của địa chỉ 10 0000 0000 0010 để thay đổi giá trị y0 của sprite core trên màn hình máy tính.



Hình 4.25 Cấu trúc lớp SpriteCore

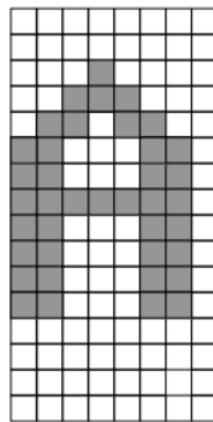
Để điều khiển spritecore sử dụng lớp SpriteCore có cấu trúc như được thể hiện ở trong Hình 4.25. Trong đó phương thức:

- *wr_pixel* thay đổi màu sắc của điểm ảnh được lưu trữ trong RAM.
- *move_xy* được sử dụng để thay đổi giá trị tọa độ của sprite trên màn hình.
- *bypass* được sử dụng để loại bỏ các điểm ảnh được tạo ra bởi sprite core nếu tham số *by* có giá trị là 1.

4.3.3 Textcore

Spritecore có thể được sử dụng để hiển thị các ký tự văn bản lên màn hình máy tính. Tuy nhiên khi sử dụng phương pháp này, số lượng tài nguyên phần cứng cần sử dụng là vô cùng lớn vì mỗi một ký tự trên màn hình đều cần phải có một spritecore. Thay vào đó, hệ thống xử lý đồ họa sẽ chia màn hình thành dạng lưới với các ô (tile) có kích thước 8x16 điểm ảnh. Với kích thước màn hình 640x480, lưới có tổng cộng 80 ô theo chiều ngang và 30 ô theo chiều dọc.

Để hiển thị được văn bản, mỗi ô vuông trên lưới lưu trữ một mã ASCII của ký tự trong ô RAM. Hệ thống xử lý đồ họa tạo ra các điểm ảnh của một ô bằng cách tham chiếu đọc tham chiếu giá trị của mã ASCII trong font ROM (chứa bitmap của các ký tự).

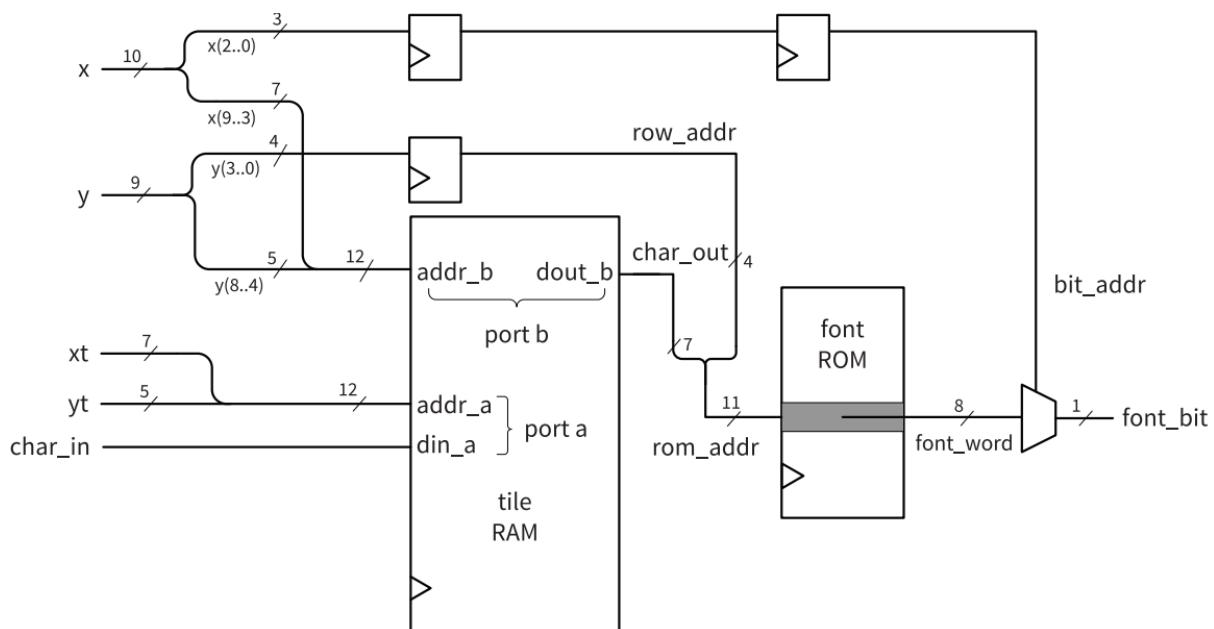


Hình 4.26 Bitmap của ký tự A

Để tiết kiệm tài nguyên sử dụng, font ROM chỉ sử dụng 1 bit CD để lưu trữ thông tin điểm ảnh của một ký tự. Độ rộng dữ liệu của font ROM có kích thước bằng với độ rộng của một ô trong lưới (8 bit). Do cần phải lưu trữ 128 ký tự, mỗi

ký tự có 16 dòng, độ rộng của bus địa chỉ là $\lceil \log_2 128 * 16 \rceil = 11$ bit. Trong đó 7 MSB đại diện cho mã ASCII của ký tự cần hiển thị, 4 LSB được sử dụng để chọn hàng trong bitmap của ký tự được chỉ định.

Tile RAM được sử dụng để lưu trữ giá trị mã ASCII từng ô trong lưới, do đó cần $\lceil \log_2 80 * 30 \rceil = 12$ bit địa chỉ. Vì quá trình đọc và ghi dữ liệu diễn ra độc lập với nhau nên textcore sẽ sử dụng dual-port RAM với hai cổng đọc ghi riêng biệt. Vì xử lý sử dụng cổng ghi để ghi chỉ định ký tự cần hiển thị ở mỗi ô. Để địa chỉ hóa toạ độ các ô trong lưới, 5 MSB của tile RAM có giá trị là toạ độ dòng còn 7 LSB có giá trị là toạ độ cột của ô.



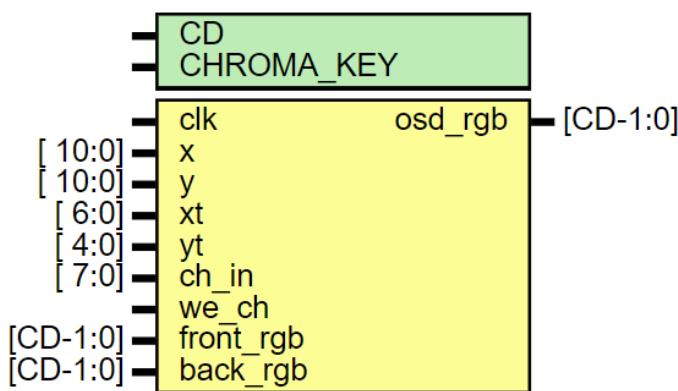
Hình 4.27 Sơ đồ khái niệm của textcore

Textcore sử dụng cổng đọc của tile RAM để xác định giá trị mã ASCII của ký tự mà một ô cần hiển thị. Giá trị các bit y[8:4], x[9:3] của toạ độ điểm ảnh hiện tại được sử dụng lần lượt là MSB và LSB của địa chỉ cần đọc. Mã ASCII này được dùng kết hợp với 3 LSB của tín hiệu y để đọc ra hàng dữ liệu của bitmap ứng với ký tự được chỉ định.

Cuối cùng để xác định được chính xác điểm ảnh cần hiển thị trong bitmap, textcore sử dụng 3 LSB của tín hiệu x để làm đầu vào cho mạch chọn kênh 8 sang 1. Do tile RAM và font ROM là các mạch đồng bộ nên cần tạo độ trễ cho các tín

hiệu x và y bằng các thanh ghi để giúp mạch có thể hoạt động được một cách chính xác.

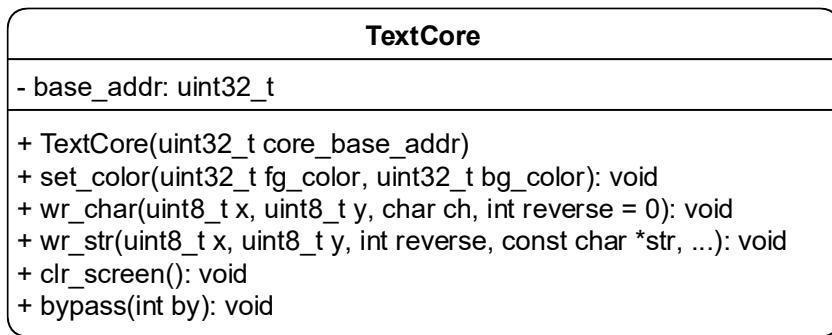
Ngoài các tín hiệu về tọa độ và đọc/ghi dữ liệu, mô-đun textcore có thêm hai tín hiệu *font_rgb*, và *back_rgb* cho phép chỉ định màu sắc của các ký tự. MSB của tín hiệu *ch_in* chỉ định xem liệu ký tự trong ô có bị đảo ngược màu *font_rgb*, và *back_rgb* khi được hiển thị trên màn hình hay không (khi đảo ngược thì text được highlight). Để làm một ô trở nên trong suốt, cần ghi mã ASCII 0 (null char) vào ô RAM có địa chỉ tương ứng với ô đó.



Hình 4.28 Mô-đun textcore

Mô-đun text wrapper được sử dụng để tạo ra một giao diện kết nối giữa textcore với hệ thống xử lý đồ họa. Vì xử lý có thể điều khiển textcore bằng cách sử dụng các thanh ghi sau:

- Thanh ghi có địa chỉ 00 aaaa aaaa aaaa (địa chỉ của tile RAM, được tạo bởi việc ghép tọa độ của ô lại với nhau):
 - Ghi vào bit 7 để hoán đổi màu *font_rgb* và *back_rgb*.
 - Ghi vào các bit 6-0 để chỉnh định mã ASCII cho ô.
- Ghi giá trị 1 vào địa chỉ 10 0000 0000 0000 để loại bỏ các điểm ảnh được tạo ra bởi khói văn bản khỏi khung hình.
- Ghi vào các bit 11-0 ở địa chỉ 10 0000 0000 0001 để chỉ định 12 bit màu *font_rgb*.
- Ghi vào các bit 11-0 ở địa chỉ 10 0000 0000 0010 để chỉ định 12 bit màu *back_rgb*.



Hình 4.29 Cấu trúc lớp TextCore

Để điều khiển textcore sử dụng lớp TextCore có cấu trúc như được thể hiện ở trong Hình 4.29. Trong đó phương thức:

- *set_color* thay đổi màu sắc hiển thị của các ký tự trên màn hình.
- *wr_char* được sử dụng để hiển thị ký tự ở ô có toạ độ (x, y).
- *wr_str* được sử dụng để viết một dòng ký tự bắt đầu từ ô có toạ độ (x, y).
- *clr_screen* được sử dụng để xoá tất cả các ký tự có trên màn hình.
- *bypass* được sử dụng để loại bỏ tất cả các điểm ảnh được tạo ra bởi textcore khỏi khung hình nếu tham số by có giá trị là 1.

4.3.4 Khối bộ đệm khung

Khối bộ đệm khung là một khối IP đặc biệt có chức năng lưu trữ và cho phép vi xử lý thao tác với tất cả các điểm ảnh có trong một khung hình. Vì dữ liệu của các điểm ảnh được lưu trữ ở trong bộ đệm khung nên hình ảnh có thể liên tục được hiển thị trên màn hình máy tính ngay cả khi vi xử lý không gửi bất kỳ dữ liệu nào cho hệ thống xử lý đồ họa. Thông thường, các khối video IP sửa đổi các điểm ảnh được sinh ra bởi bộ đệm khung để tạo thêm các hình ảnh mới và truyền dữ liệu này tới màn hình thông qua cổng VGA.

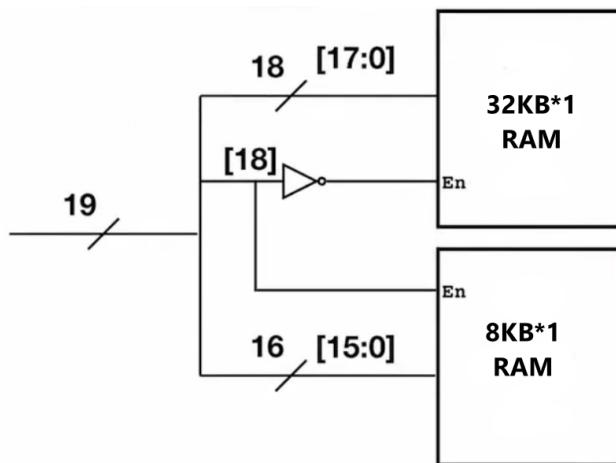
Ở độ phân giải 640x480, một khung hình có $\frac{640 \times 480}{1024 \times 8} = 38,4K$ điểm ảnh.

Nếu lưu trữ giá trị màu sắc của các điểm ảnh này ở dạng 12 bit thì cần khoảng $38,4K \times 12B = 460.8KB$. Trong khi đó, theo như Bảng 4.1 [10] bo mạch Arty A7-35T chỉ hỗ trợ bộ nhớ tối đa là $1,800Kb = 200KB$. Do đó cần phải giảm dung lượng nhớ của mỗi điểm ảnh trong bộ đệm khung xuống còn một bit để phù hợp

với dung lượng nhớ mà bo mạch này hỗ trợ. Điều này có nghĩa là một điểm ảnh chỉ có một trong hai màu được chỉ định từ trước. Khi sử dụng một bit để lưu trữ dữ liệu của các điểm ảnh, bộ đệm khung chiếm khoảng 38.4KB bộ nhớ mà FPGA hỗ trợ. Các dung lượng còn lại được phân bổ cho vi xử lý và các khối IP khác.

Bảng 4.1 Thông số tài nguyên Artix-7 FPGA

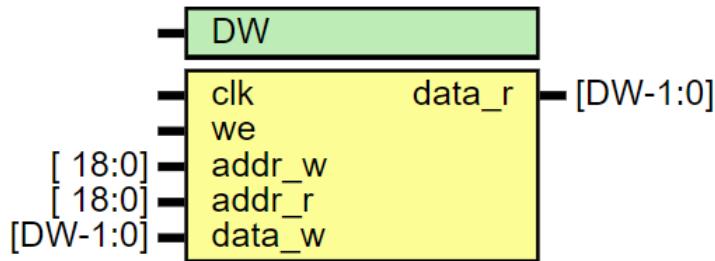
Thiết bị	Số lượng Logic Cells	Số lượng 36Kb-BRAM	Kích thước BRAM (Kb)	Số lượng MMCM
XC7A12T	12,800	20	720	3
XC7A15T	16,640	25	900	5
XC7A25T	23,360	45	1,620	3
XC7A35T	33,280	50	1,800	5
XC7A50T	52,160	75	2,700	5
XC7A75T	75,520	105	3,780	6
XC7A100T	101,440	135	4,860	6
XC7A200T	215,360	365	13,140	10



Hình 4.30 Sơ đồ khối mô-đun RAM

Giống như khái niệm văn bản, khái niệm bộ đệm khung cũng cần sử dụng một mô-đun RAM hai cổng. Trong đó vi xử lý sử dụng một cổng để ghi dữ liệu về các điểm ảnh trên màn hình, cổng còn lại được sử dụng để lấy các dữ liệu về màu sắc của điểm ảnh dựa trên tọa độ của nó. Với độ phân giải 640×480 cần tổng cộng $\lceil \log_2 640 \times 480 \rceil = 18.22 \approx 19$ bit để định địa chỉ cho các điểm ảnh. Nếu chỉ sử

dụng một mô-đun RAM $2^{19}bit * 1$ thì có khoảng $\frac{2^{19}-640 \times 480}{8} \approx 27KB$ bộ nhớ không được sử dụng đến. Thay vào đó mô-đun RAM được tách ra thành hai mô-đun RAM $2^{18}bit * 1$ và $2^{16}bit * 1$ và sử dụng MSB để lựa chọn việc đọc/ghi dữ liệu giữa hai mô-đun này (Hình 4.30). Khi đó chỉ lãng phí $\frac{2^{18}+2^{16}-640 \times 480}{8} = 2.5KB$ bộ nhớ của FPGA.

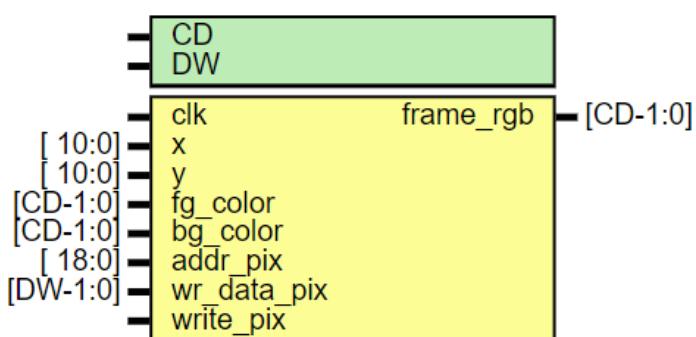


Hình 4.31 Mô-đun frame_buffer_ram

Mô-đun frame_buffer_core (Hình 4.32) có nhiệm vụ chuyển đổi các tín hiệu của tọa độ hình điểm ảnh x và y thành địa chỉ RAM. Mô-đun này tạo tín hiệu đầu ra dựa trên dữ liệu đọc được dữ liệu đọc được từ RAM và các tín hiệu fg_color, bg_color. Công thức sau được sử dụng để chuyển đổi tọa độ của các điểm ảnh thành địa chỉ trong RAM:

$$addr = 640 \times y + x = 512 \times y + 128 \times y + x$$

$$\Leftrightarrow addr = y \ll 9 + y \ll 7 + x \quad (4.4)$$

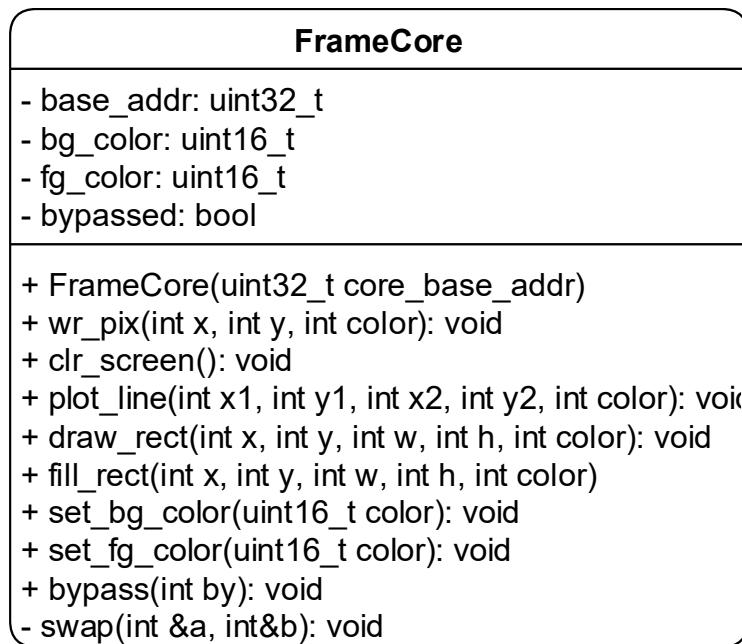


Hình 4.32 Mô-đun frame_buffer_core

Ngoài ra mô-đun cũng sử dụng hai tín hiệu addr_pix, wr_data_pix và write_pix để kiểm soát việc ghi dữ liệu vào RAM.

Mô-đun frame_buffer_wrapper được sử dụng để tạo ra một giao diện kết nối giữa khói bộ đệm khung với hệ thống xử lý đồ họa. Vì xử lý có thể điều khiển khói bộ đệm khung bằng cách sử dụng các thanh ghi sau:

- Ghi 1 vào bit 0 của các thanh ghi có địa chỉ 0aaa aaaa aaaa aaaa để chỉ định màu của điểm ảnh được xác định theo công thức (4.4) là fg_color hoặc 0 để chỉ định màu bg_color.
- Thanh ghi có địa chỉ 1111 1111 1111 1111 1111:
 - Ghi vào bit 0 để loại bỏ các điểm ảnh được tạo ra bởi textcore khỏi khung hình.
 - Ghi vào các bit 12-1 để chỉ định màu foreground.
 - Ghi vào các bit 24-13 để chỉ định màu background.



Hình 4.33 Cấu trúc lớp FrameCore

Để điều khiển khói bộ đệm khung sử dụng lớp FrameCore có cấu trúc như được thể hiện ở trong Hình 4.33. Trong đó phương thức:

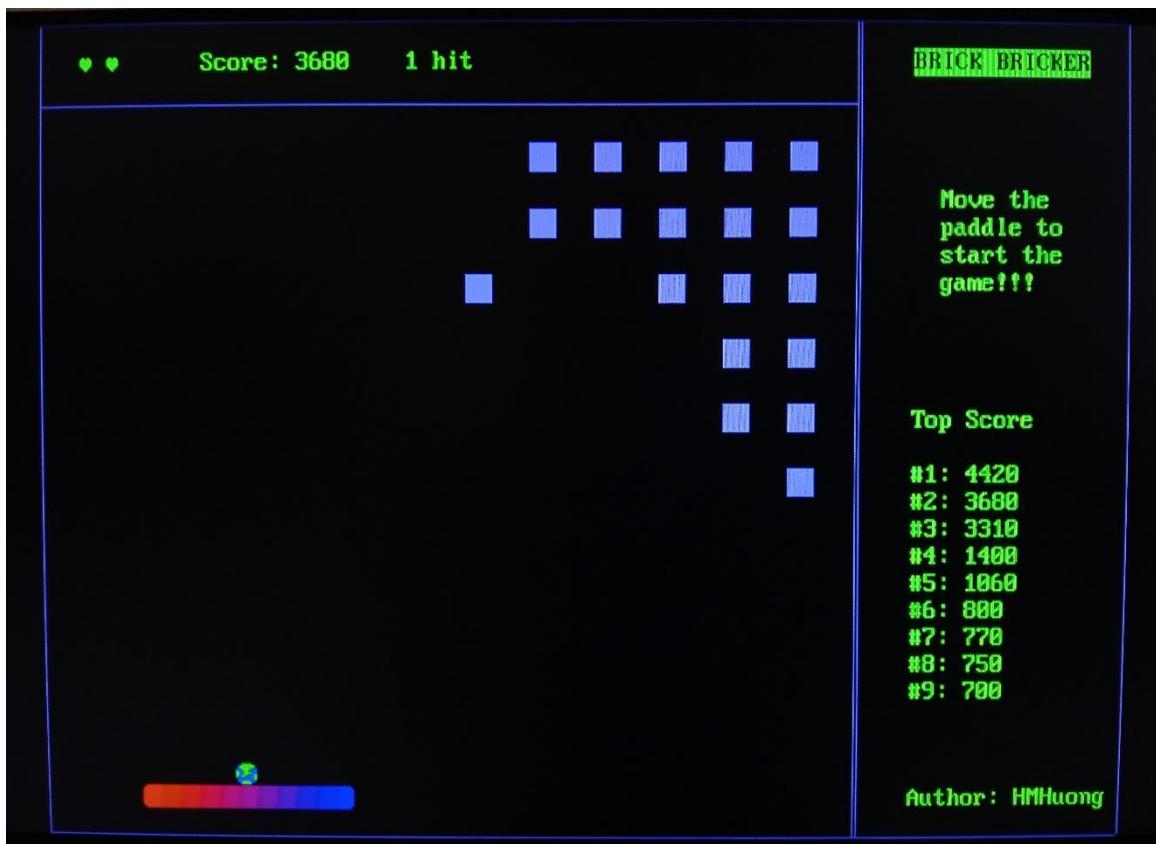
- *wr_pix* được sử dụng để thay đổi dữ liệu của điểm ảnh ở trong RAM của khói frame buffer.
- *clr_screen* dùng để tô tắt cả các điểm ảnh có trong bộ đệm khung thành màu nền (*bg_color*).

- *plot_line* được sử dụng để vẽ một đường thẳng với hai đầu mút có toạ độ lần lượt là (x1, y1) và (x2, y2).
- *draw_rect* và *fill_rect* được sử dụng để vẽ đường viền hoặc hình chữ nhật được tại toạ độ (x, y) với độ rộng w và chiều cao h.
- *set_bg_color* và *set_fg_color* được sử dụng để thiết lập màu background và màu foreground.
- *bypass* được sử dụng để loại bỏ tất cả các điểm ảnh được tạo ra bởi textcore khỏi khung hình nếu tham số by có giá trị là 1.
- *swap* được sử dụng để hoán đổi hai giá trị của biến a và b cho nhau.

CHƯƠNG 5: THIẾT KẾ ỨNG DỤNG CHO HỆ THỐNG

5.1 Tổng quan

Để kiểm chứng hoạt động của các khối IP, đề tài thiết kế ứng dụng trò chơi phá gạch như được thể hiện trong Hình 5.1. Trong trò chơi này, người dùng sử dụng các nút bấm để di chuyển thanh gang bên dưới để ngăn không cho bóng di chuyển xuống cạnh dưới màn hình. Mỗi khi quả bóng va vào các cạnh trên, trái, phải, viên gạch hoặc thanh ngang ở dưới màn hình thì quả bóng nảy với một góc ngược lại. Mỗi khi quả bóng va vào một viên gạch thì viên gạch đó biến mất và điểm số tăng lên. Trò chơi kết thúc khi người chơi để quả bóng đi xuống cạnh dưới của màn hình quá ba lần hoặc tất cả số gạch trên màn hình đều bị phá.



Hình 5.1 Trò chơi phá gạch

5.2 Lựa chọn các khối IP

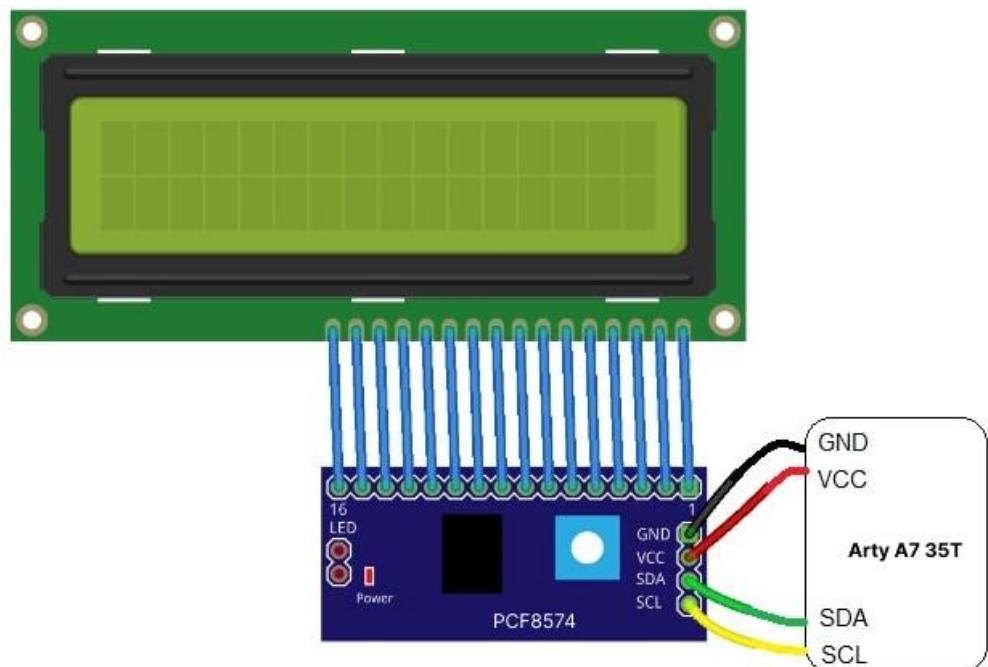
Để kiểm tra hoạt động của hệ thống, đồ án sử dụng tất cả các khối IP đã thiết kế bao gồm: khối IP timer, UART, I2C, GPI, GPO, khối đồng bộ VGA, khối kiểm tra màn hình, spritecore, textcore, và bộ đếm khung. Các khối này sẽ được kết nối với nhau theo Hình 1.3. Trong đó tám LSB của GPI lần lượt được kết nối

với công tắc và nút nháy có trên bo mạch Arty A7-35T. Các nút nháy được sử dụng để điều khiển hoạt động của trò chơi. Các công tắc được sử dụng để chuyển giữa chế độ trò chơi và chế độ kiểm tra màu sắc của màn hình. Giá trị đọc được từ các công tắc được gửi tới GPO (kết nối với các led đơn có trên bo mạch) để hiển thị trạng thái của các trạng thái hoạt động của hệ thống.

Các hình ảnh về quả bóng và thanh ngang ở bên dưới được tạo ra bằng cách sử dụng spritecore. Vì các quả bóng và thanh ngang liên tục di chuyển trên màn hình nên để giảm số lượng phép tính mà vi xử lý phải thực hiện, các spritecore được sử dụng. Khi đó vi xử lý chỉ cần ghi giá trị tọa độ của quả bóng hoặc thanh ngang, spritecore sẽ tự động xử lý và hiển thị hình ảnh lên trên màn hình máy tính.

Khối bộ đếm khung được sử dụng để phân vùng và hiển thị các viên gạch trên màn hình. Textcore cũng được sử dụng để hiển thị điểm số, tên trò chơi và các thông tin khác liên quan đến trò chơi.

Khối kiểm tra màn hình được sử dụng để hiển thị các màu sắc khác nhau lên màn hình. Ngoài ra dữ liệu hình ảnh được tạo ra bởi các video IP sẽ được gửi đồng bộ tới màn hình máy tính thông qua khối đồng bộ VGA.



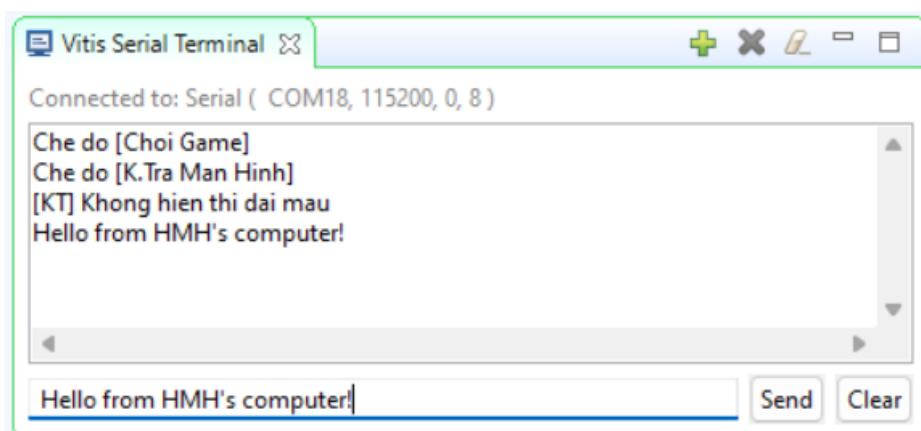
Hình 5.2 Sơ đồ kết nối màn hình LCD, PCF8574 và Arty A7 52T

Khối I2C sẽ kết nối với mô-đun PCF8574 (xem chi tiết tại phần phụ lục) thông qua hai chân SCL và SDA trên bo mạch để điều khiển màn hình LCD. Màn hình LCD có nhiệm vụ hiển thị thông tin về chế độ chơi game hay kiểm tra màn hình máy tính. Từ đó giúp kiểm chứng sự hoạt động của khối I2C cũng như giúp người dùng dễ dàng phân biệt hai chế độ trong quá trình thao tác.



Hình 5.3 Màn hình LCD hiển thị các chế độ

Vì chương trình điều khiển trực tiếp phần cứng của hệ thống và chạy trong môi trường độc lập không có hệ điều hành cũng như không thể truy cập vào các thư viện vào ra chuẩn của ngôn ngữ lập trình C/C++, do đó để việc phát triển dễ dàng hơn, một số hàm tiện ích đơn giản đã được tạo ra để lấy thông tin về thời gian hoạt động của hệ thống (through qua khói định thời), tạm dừng hoạt động của hệ thống (sleep), gửi dữ liệu tới máy tính để hiển thị một số thông tin gỡ lỗi (through qua giao thức UART). Các hàm tiện ích này đều được định nghĩa sẵn ở trong thư viện init.h. Các tệp mã nguồn khác chỉ cần sử dụng thư viện này để gọi các hàm tiện ích.



Hình 5.4 Dữ liệu được gửi/nhận bởi khói UART

5.3 Cấu trúc dự án

Việc phát triển dự án [6, 7] bao gồm hai công đoạn chính phát triển phần mềm và phần cứng. Phát triển phần cứng trên FPGA bằng công cụ Vivado. Một số thư mục và tệp tin cần chú ý bao gồm:

- \hardware\hardware.srcs\sources_1\new: chứa toàn bộ mã System Verilog của hệ thống.
- \hardware\hardware.srcs\sources_1\ip: Chứa các IP của bên thứ ba được sử dụng của hệ thống.
- \hardware\hardware.srcs\constrs_1\new\Arty_A7_35T.xdc: chứa cấu hình các chân đầu ra của nền tảng FPro.

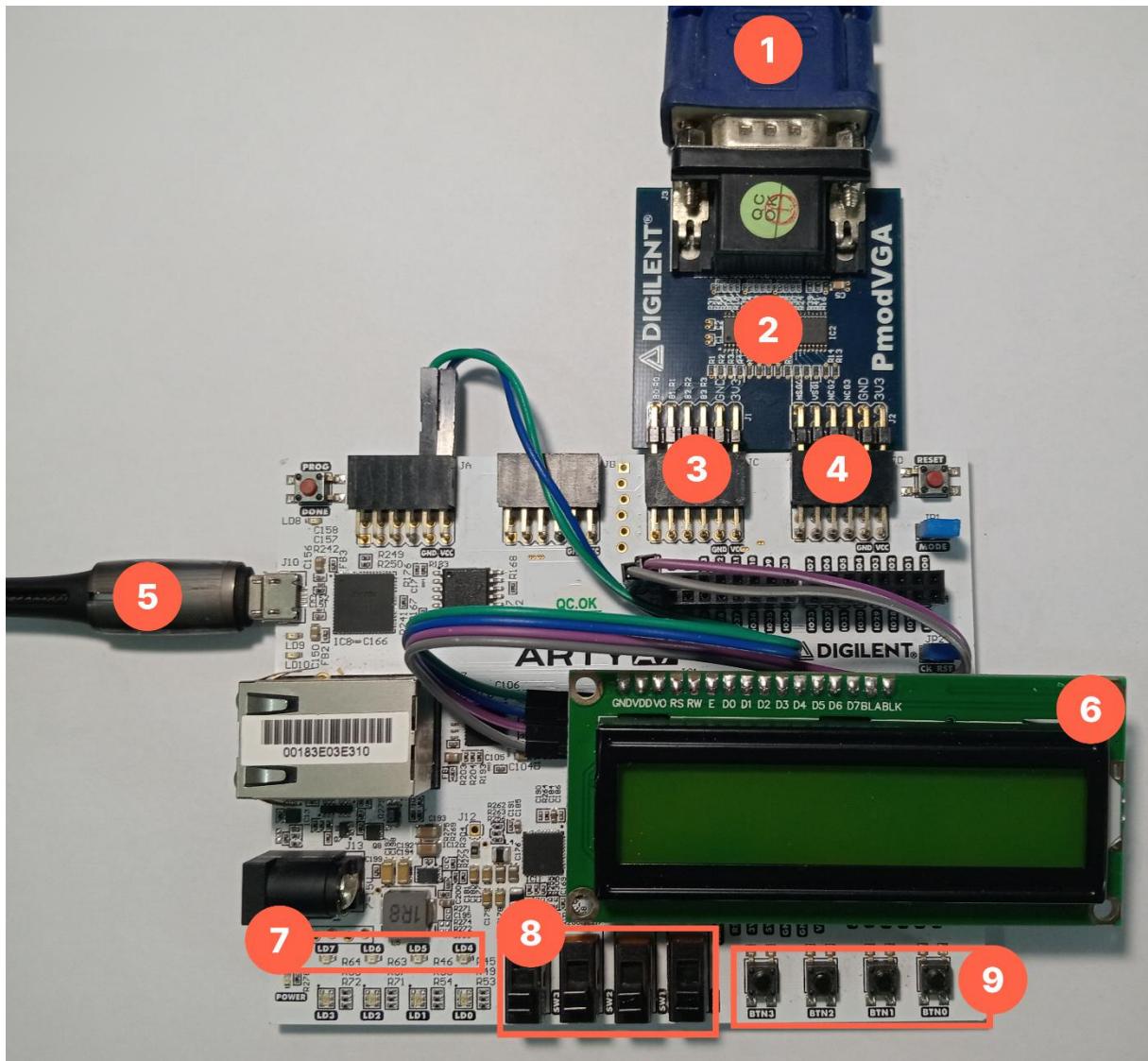
Việc phát triển các trình điều khiển và chương trình ứng dụng có thể được thực hiện bằng công cụ Vitis [6], với các tệp mã nguồn được lưu trữ trong thư mục software của dự án. Các thư mục và tệp tin quan trọng bao gồm:

- \software\app\src\main.cpp: tệp tin mã nguồn của chương trình chính được sử dụng trong dự án.
- \software\app\src\driver\platform\: thư mục chứa mã nguồn các trình điều khiển của các khối trong nền tảng FPro.
- \software\app\src\driver\device\: thư mục chứa mã nguồn của các trình điều khiển thiết bị được kết nối với nền tảng FPro. Các tệp mã nguồn trong thư mục này được xây dựng dựa trên mã nguồn được cung cấp trong thư mục \software\app\src\driver\platform.
- \software\app\src\game\: thư mục chứa mã nguồn của trò chơi phá gạch.

5.4 Sơ đồ mạch nối dây

Hệ thống FPro được triển khai trên bo mạch Arty A7-35T [1, 4, 5, 10] và được kết nối với các thiết bị ngoại vi như được thể hiện ở trong Hình 5.5. Trong đó cổng VGA được kết nối từ màn hình máy tính với bo mạch thông qua Pmod VGA được kết nối ở Pmod Header JC và JD ở bo mạch. Khối UART cũng được kết nối với máy tính thông qua cổng USB UART có sẵn ở trên bo mạch. Cổng

USB UART cũng được sử dụng để nạp chương trình và cấp nguồn cho toàn bộ hệ thống. Màn hình LCD được cấp nguồn và kết nối với hệ thống thông qua mô-đun PCF8574 bằng giao thức I2C. Ngoài ra hệ thống sử dụng các nút nhấn, công tắc, và led có sẵn trên bo mạch để kết nối với các khối GPI và GPO.

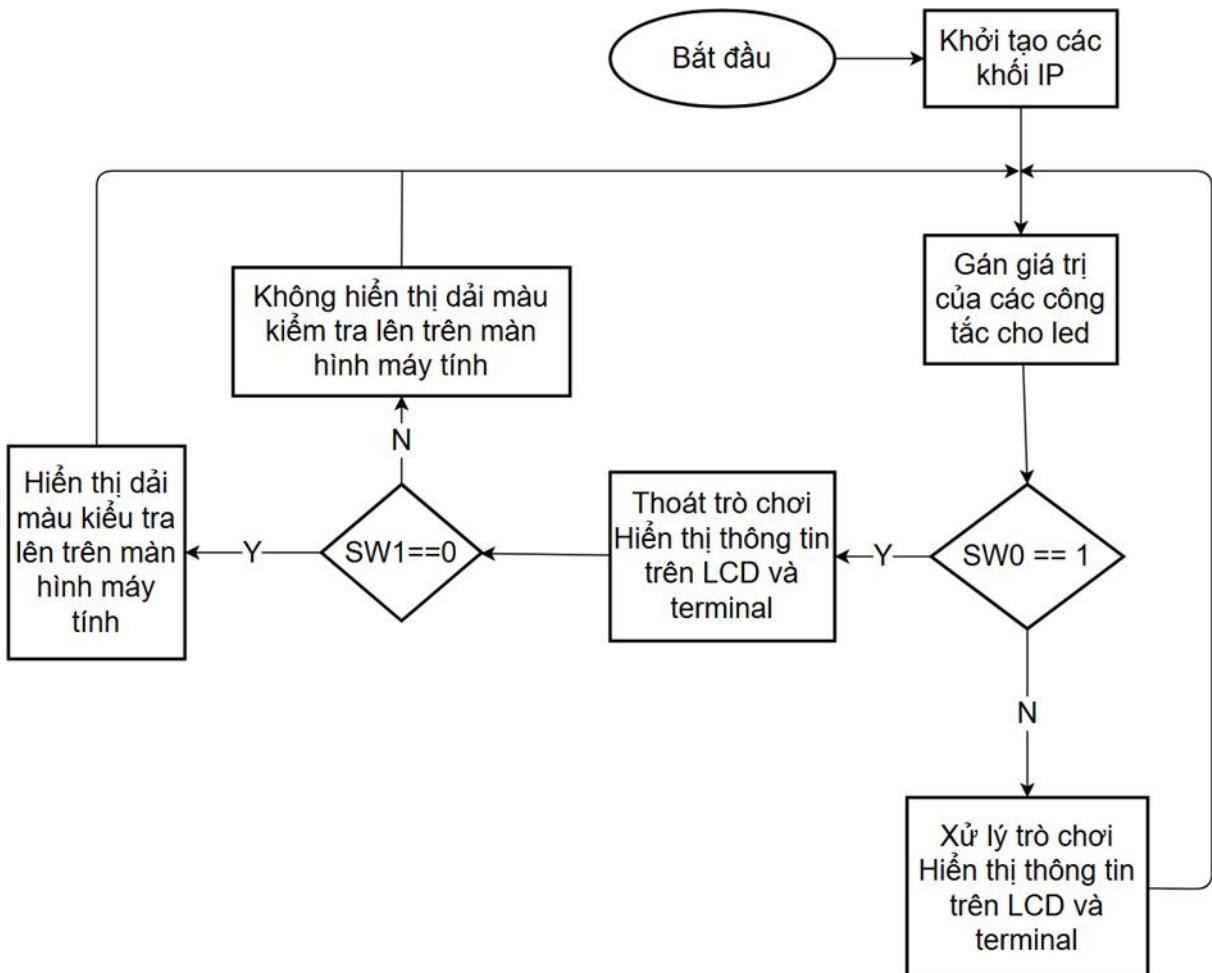


Hình 5.5 Kết nối của hệ thống. 1. Cổng VGA 2. Pmod VGA 3. Pmod header JC 4. Pmod header JD 5. USB UART 6. Màn hình LCD 7. Bốn LED xanh lá cây 8. Bốn công tắc 9. Bốn nút nhấn

5.5 Lưu đồ thuật toán

Hình 5.6 mô tả lưu đồ thuật toán của chương trình ứng dụng. Công tắc 0 được sử dụng để chọn chế độ chơi hay kiểm tra màn hình máy tính. Trong khi đó công tắc 1 được sử dụng để lựa chọn giữa việc hiển thị hay không hiển thị các điểm ảnh được sinh ra bởi khôi kiểm tra màn hình. Việc xử lý trò chơi diễn ra

trong một khoảng thời gian rất ngắn và được lặp lại liên tục. Nếu công tắc 0 được bật trong khi trò chơi đang diễn ra thì mọi dữ liệu của trò chơi sẽ bị mất và chương trình chính sẽ ngay lập tức chuyển sang chế độ kiểm tra màn hình máy tính.

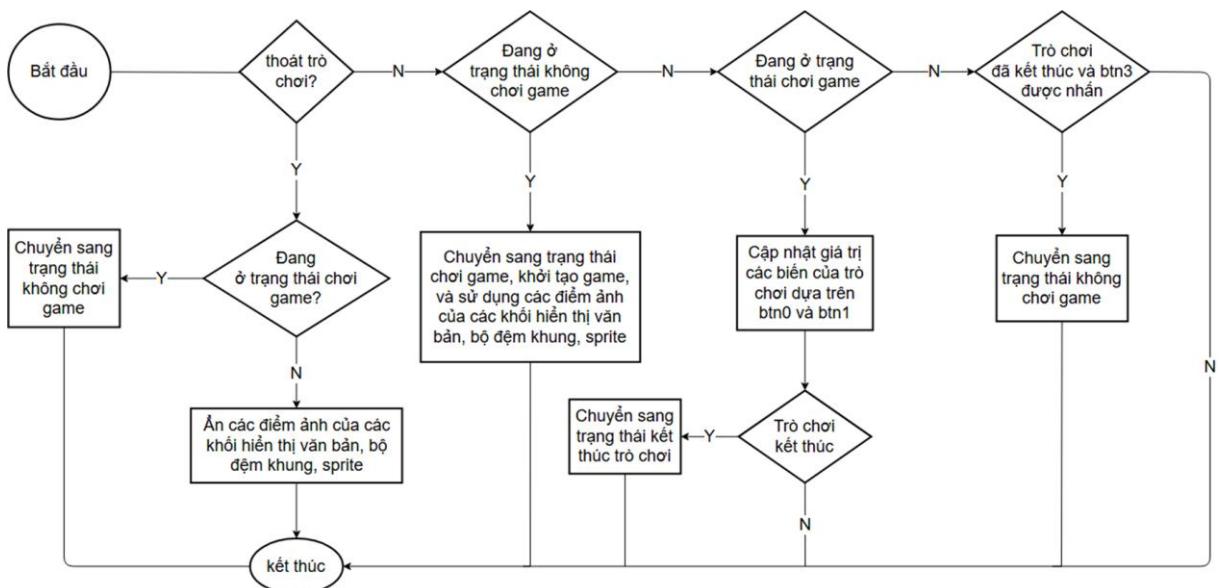


Hình 5.6 Lưu đồ thuật toán của chương trình chính

Lưu đồ thuật toán xử lý trò chơi được thể hiện như ở trong Hình 5.7. Việc kiểm soát trò chơi được thực hiện thông qua các trạng thái như:

- Đang chơi game.
- Đang không chơi game.
- Trò chơi đã kết thúc.

Phần mềm chuyển đổi trò chơi sang các trạng thái khác nhau dựa trên trạng thái hiện tại và yêu cầu thoát trò chơi từ chương trình chính. Ngoài ra các btn0, btn1 ở trên bo mạch cũng được sử dụng để điều khiển trạng thái của thanh ngang dưới màn hình. Đặc biệt khi trò chơi đã kết thúc người dùng có thể nhấn btn3 trên bo mạch để chơi lại.



Hình 5.7 Lưu đồ thuật toán của trò chơi phá gạch

Cũng cần phải lưu ý rằng mỗi lần thực hiện quá trình xử lý hay thoát trò chơi ở chương trình chính (Hình 5.6) thì các bước được liệt kê ở lưu đồ thuật toán của trò chơi phá gạch (Hình 5.7) đều được thực hiện.

5.6 Đánh giá hệ thống

Sau khi chạy synthesis và implementation trên phần mềm Vivado, các báo cáo về timing cũng như tài nguyên sử dụng của hệ thống như được thể hiện trong Hình 5.8, Bảng 5.1, Hình 5.9 và Hình 5.10.

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 1.362 ns	Worst Hold Slack (WHS): 0.044 ns	Worst Pulse Width Slack (WPWS): 3.000 ns
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWNS): 0.000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 4952	Total Number of Endpoints: 4952	Total Number of Endpoints: 1178

All user specified timing constraints are met.

Hình 5.8 Báo cáo về timing

Dựa trên báo cáo timing của phần mềm Vivado, chu kỳ clock tối thiểu để mạch có thể hoạt động bình thường là:

$$T_{min} = Constraints\ Period - WNS$$

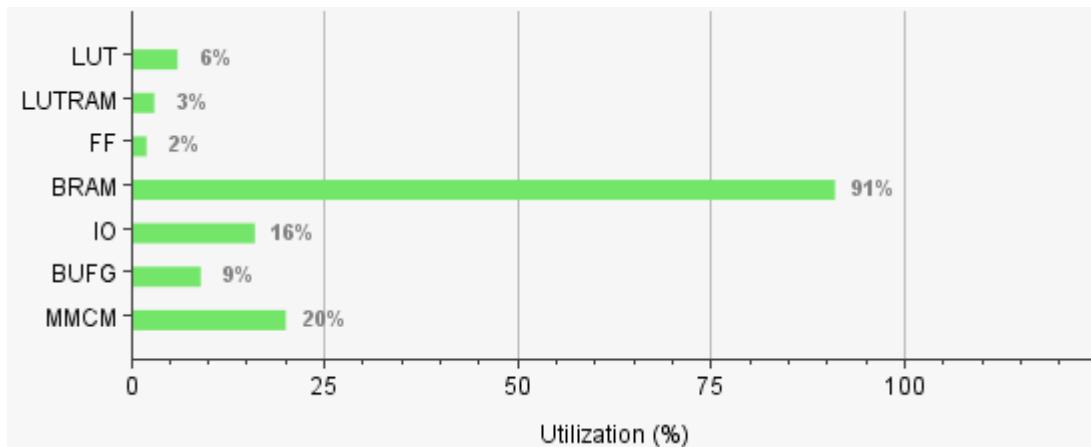
$$= 10ns - 1.362ns = 8.998ns$$

$$\Rightarrow \text{Tần số hoạt động tối đa: } f_{max} = \frac{1}{T_{min}} \approx 115,77 \text{ MHz}$$

Bảng 5.1 Tài nguyên phần cứng sử dụng trên FPGA

Tài nguyên	Sử dụng	Sẵn có
LUT	1228	20800
LUTRAM	267	9600
FF	803	41600
BRAM	45.5	50
IO	34	210
BUFG	3	32
MMCM	1	5

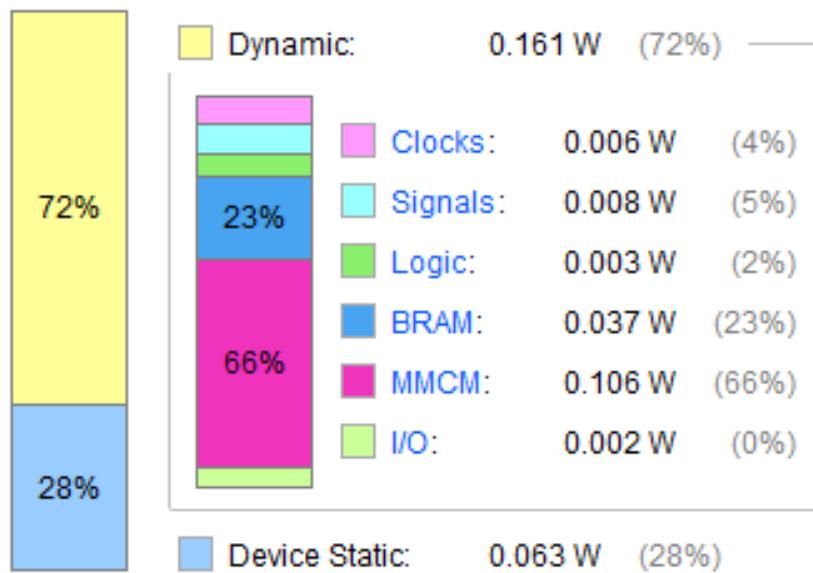
Dựa trên Bảng 5.1 và Hình 5.9, có thể thấy tài nguyên được sử dụng nhiều nhất là BRAM. Điều này khá dễ hiểu vì hệ thống xử lý đồ họa cần rất nhiều bộ nhớ để có thể lưu trữ các thông tin về điểm ảnh có trên màn hình. Do đe tài triển khai nền tảng FPro trên bo mạch Arty A7-35T của Xilinx, nên các đoạn mã SystemVerilog tạo ra bộ nhớ RAM, ROM bộ đệm FIFO, ... sẽ được phần mềm Vivado triển khai dưới dạng các mô-đun BRAM có sẵn trên FPGA. Các mô-đun này khác với các ô logic thông thường của FPGA và có thể được coi là các mô-đun SRAM đồng bộ với kích thước lớn.



Hình 5.9 Đồ thị thể hiện phần trăm sử dụng tài nguyên phần cứng trên FPGA

Hình 5.10 cho thấy tổng năng lượng tiêu thụ của toàn bộ hệ thống bao gồm cả MicroBlaze MCS và nền tảng FPro là 0.224W. Trong đó năng lượng tiêu thụ khi FPGA đang hoạt động là 0.161W (chiếm 72%) và năng lượng cơ bản mà

FPGA cần để duy trì và đảm bảo các mạch bên trong sẵn sàng hoạt động là 0.063W (chiếm 28%).



Hình 5.10 Năng lượng sử dụng trên FPGA

MMCM (Multiply-Accumulate Clock Manager) là thành phần chiếm nhiều năng lượng nhất trong hệ thống (66%). MMCM là một khối chức năng quan trọng trong FPGA, chịu trách nhiệm tạo ra các tín hiệu đồng hồ chính xác và ổn định cho các thành phần khác hoạt động. Việc MMCM tiêu thụ năng lượng cao là do MMCM có tần số hoạt động cao cũng như cần phải chia hai xung đồng hồ 100MHz và 25MHz cho cả hệ thống thiết bị ngoại vi và hệ thống xử lý đồ họa.

BRAM cũng là một thành phần tiêu thụ nhiều năng lượng của hệ thống do chúng chiếm một số lượng lớn trong thiết kế và việc truy cập và đọc/ghi dữ liệu từ các BRAM này cũng tiêu tốn một lượng năng lượng đáng kể.

Các thành phần còn lại như clocks, signals, logic, và I/O tiêu thụ một lượng năng lượng nhỏ hơn rất nhiều so với hai thành phần trên. Từ đó có thể thấy rằng, các thành phần tiêu tốn nhiều năng lượng nhất đều là các IP có sẵn trên bo mạch được nhà sản xuất cung cấp.

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

1. Những nội dung đã hoàn thành

- Thiết kế thành công nền tảng FPro cung cấp các ngoại vi I/O phô biến.
- Nghiên cứu và thiết kế các khối IP như GPI, GPO, Timer, UART, I2C, VGA, ... bằng ngôn ngữ SystemVerilog.
- Tìm hiểu sơ lược về MicroBlaze MCS.
- Phát triển các trình điều khiển cho các khối IP đã thiết kế bằng ngôn ngữ C/C++.
- Phát triển một trò chơi đơn giản sử dụng các khối IP bằng cách sử dụng các công cụ Vivado, Vitis.

2. Hạn chế của đề tài

- Chia địa chỉ cho các khối IP trong nền tảng FPro chưa hiệu quả vì còn tồn tại những khói IP trong nền tảng chưa được tối ưu để sử dụng không gian của dải địa chỉ được cung cấp một cách hiệu quả.
- Nền tảng FPro chưa hỗ trợ các cơ chế ngắt và DMA. Điều này có nghĩa là vì xử lý sẽ xử lý các lỗi này một cách tuần tự.
- Khối I2C chưa hỗ trợ cơ chế đa master để đơn giản thiết kế. Đây không phải là một vấn đề quá lớn vì hầu hết các ứng dụng trong hệ thống nhúng ít khi yêu cầu giao thức I2C phải hoạt động ở chế độ này.

3. Hướng phát triển

- Tích hợp thêm nhiều khói IP khác vào hệ thống ví dụ như khói xử lý âm thanh, bàn phím và chuột, ...
- Phát triển thêm các cơ chế ngắt và DMA cho các khói IP.
- Tự tạo ra một vi xử lý theo kiến trúc ARM hoặc RISC-V bằng ngôn ngữ SystemVerilog thay vì sử dụng vi xử lý của các bên thứ ba cung cấp (MicroBlaze MCS).

TÀI LIỆU THAM KHẢO

- [1]. AMD, 7 Series FPGAs Configuration User Guide, 2023.
- [2]. AMD, MicroBlaze Micro Controller System v3.0, 2021.
- [3]. David Harris, Sarah Harris, “Digital Design and Computer Architecture”, Morgan Kaufmann, 2012.
- [4]. Digilent, Arty A7 Revision E.2 Schematic, 2022.
- [5]. Digilent, Arty™ FPGA Board Reference Manual, 2017.
- [6]. Digilent, Getting Started with Vivado and Vitis for Baremetal Software Projects, 2024.
- [7]. Digilent, Getting Started with Vivado for Hardware-Only Designs, 2024.
- [8]. Digilent, Pmod VGA Reference Manual, 2024.
- [9]. Pong P. Chu, “FPGA Prototyping by SystemVerilog Examples: Xilinx MicroBlaze MCS SoC Edition”, Wiley, 2018.
- [10]. Xilinx, 7 Series FPGAs Data Sheet, 2020.

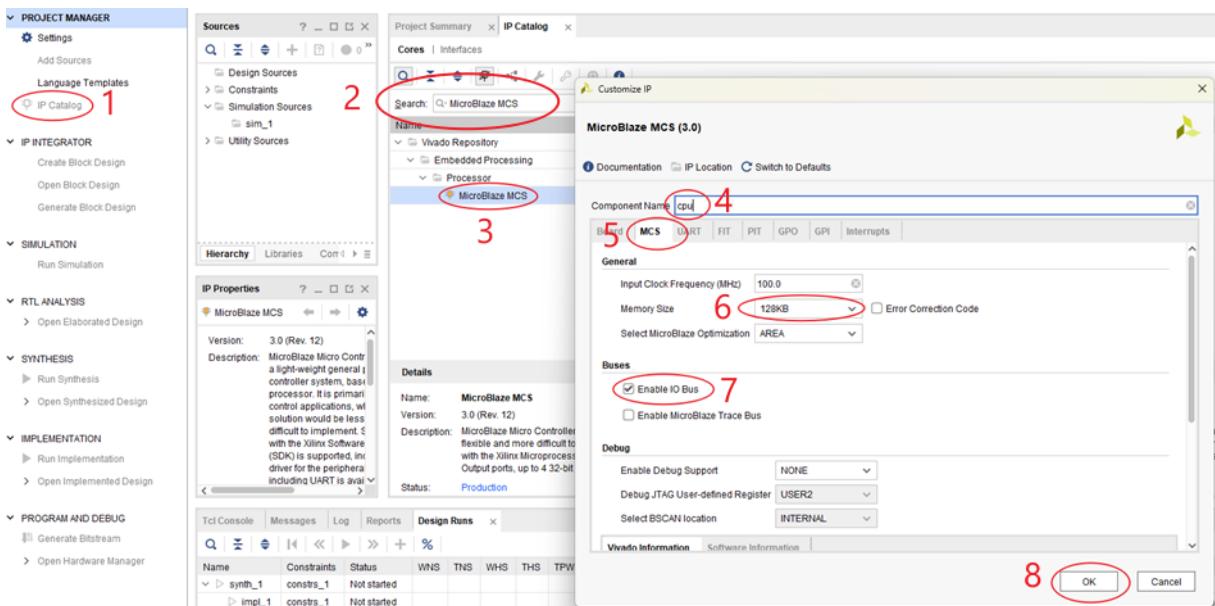
PHỤ LỤC

Mã nguồn

Toàn bộ mã nguồn của đồ án có thể được tải xuống từ đường dẫn sau:

<https://github.com/HM-Huong/DATN>

MicroBlaze MCS



Hình 0.1 Tạo MicroBlaze MCS IP

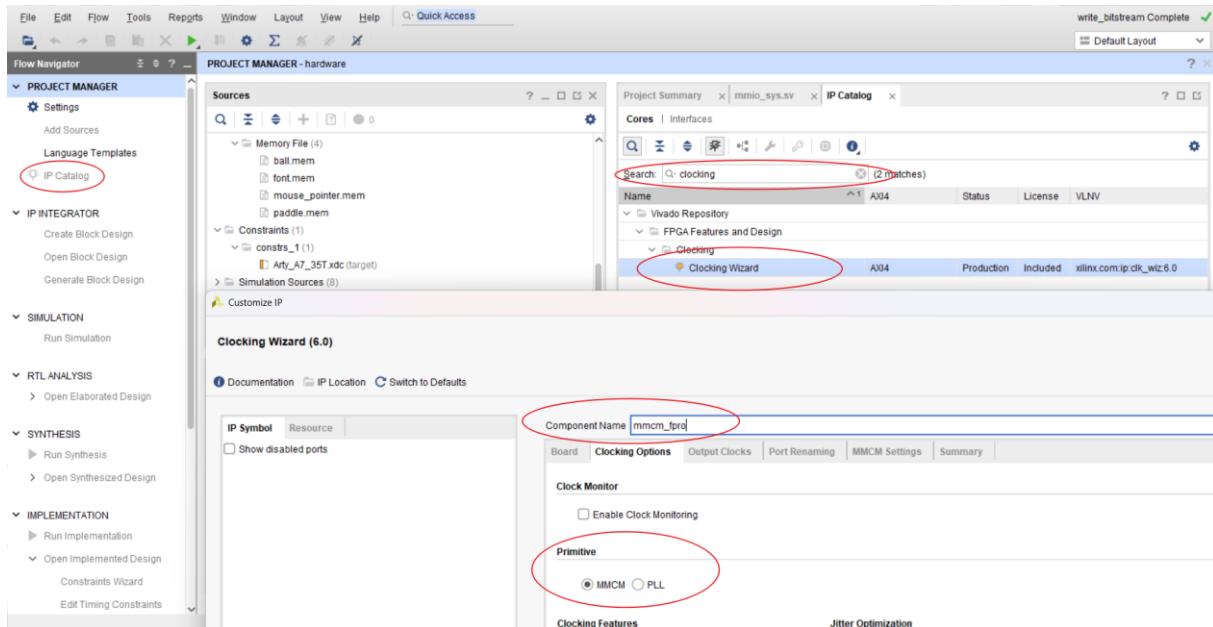
Thực hiện các bước sau đây để tạo và cấu hình MicroBlaze MCS:

1. Trong thanh Flow Navigator chọn mục IP Catalog ở trong phần Project Manager.
2. Sau khi cửa sổ IP Catalog xuất hiện, hãy tìm kiếm MicroBlaze MCS và chọn nó.
3. Đổi tên component name thành cpu.
4. Trong mục MCS chọn Memory size là 128KB và tích chọn mục enable IO Bus.

Clock management IP

FPro platform sử dụng các tín hiệu xung nhịp từ clock management IP để làm tín hiệu đồng bộ cho cả hai hệ thống và các khối IP. Do khối đồng bộ VGA sử cần có tín hiệu xung nhịp ở tần số 25MHz (khác với tần số hoạt động của hệ

thống ở 100MHz) clock management IP cần phải được cấu hình theo các bước sau:



Hình 0.2 Tạo Clock management IP

- Trong thanh Flow Navigator chọn mục IP Catalog ở trong phần Project Manager.
- Sau khi cửa sổ IP Catalog xuất hiện, hãy tìm kiếm Clocking Wizard và chọn nó.
- Chuyển sang tab mục clocking output và chọn MMCM (Mixed Mode Clock Manager) ở mục Primitive do MMCM có nhiều sự cải tiến hơn so với PLL (Phase Locked Loop)
- Đổi tên component name thành mmcm_fpro.
- Trong mục Output clocks sử tích chọn clk_out2 và sửa requested output freq thành 25. Sửa port name của clk_out1 thành clk_100M và clk_out2 thành clk25M.
- Nhấn button OK rồi nhấn Generate ở cửa sổ xuất hiện sau đó.

Component Name: mmcmm_fpro

Board | Clocking Options | **Output Clocks** | Port Renaming | MMCM Settings | Summary

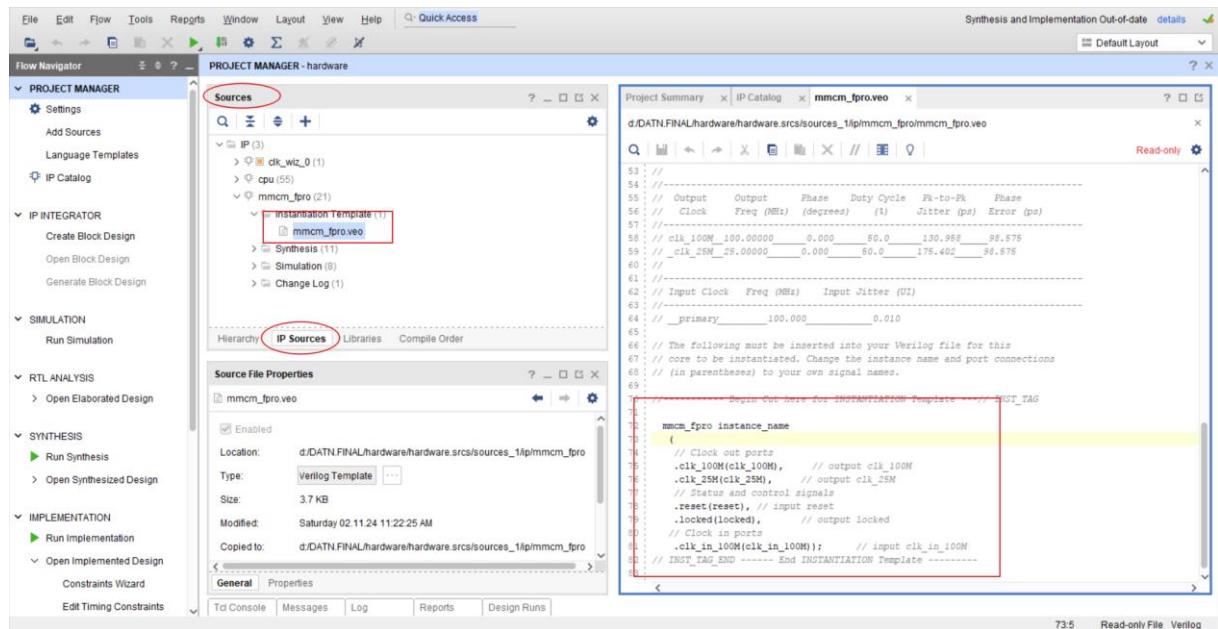
The phase is calculated relative to the active input clock.

Output Clock	Port Name	Output Freq (MHz)		Phase (degrees)	
		Requested	Actual	Requested	Actual
<input checked="" type="checkbox"/> clk_out1	clk_100M	100.000	100.00000	0.000	0.000
<input checked="" type="checkbox"/> clk_out2	clk_25M	25	25.00000	0.000	0.000
<input type="checkbox"/> clk_out3	clk_out3	100.000	N/A	0.000	N/A
<input type="checkbox"/> clk_out4	clk_out4	100.000	N/A	0.000	N/A
<input type="checkbox"/> clk_out5	clk_out5	100.000	N/A	0.000	N/A
<input type="checkbox"/> clk_out6	clk_out6	100.000	N/A	0.000	N/A
<input type="checkbox"/> clk_out7	clk_out7	100.000	N/A	0.000	N/A

USE CLOCK SEQUENCING **Clocking Feedback**

Hình 0.3 Thiết lập thông số cho Clock management IP

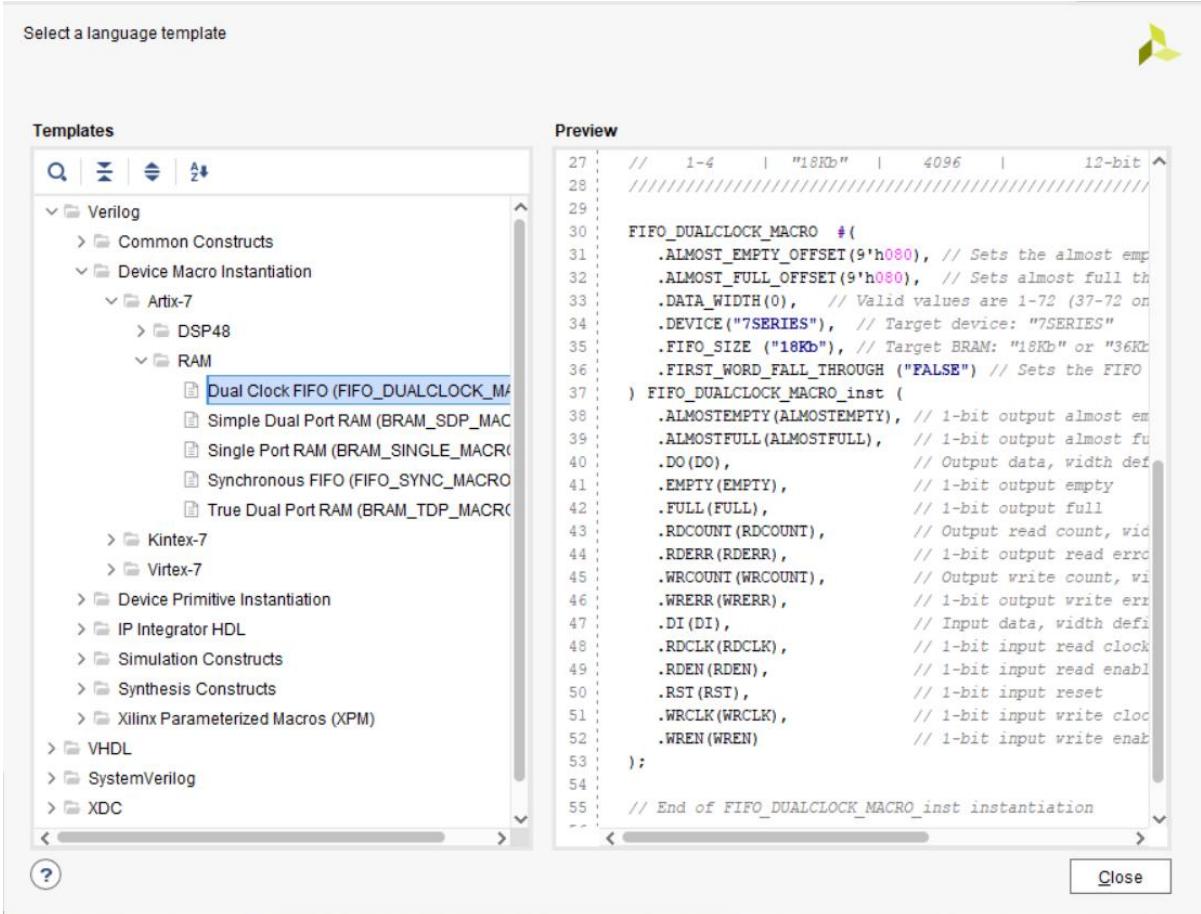
Sau khi tạo xong IP, ở cửa sổ source, chọn tab IP Sources chọn mmcmm_fpro
 > Instantiation Template > mmcmm_fpro.veo để lấy đoạn mã khởi tạo IP này.



Hình 0.4 Đoạn mã mẫu được dùng để khởi tạo Clock management IP

Dual-clock FIFO IP

Dual-clock FIFO IP có thể được tạo ra bằng cách sử dụng BRAM bằng cách cấu hình theo các đoạn mã mẫu được cung cấp bởi Vivado trong mục Tools > Language Templates. Sau khi chọn xong một cửa sổ mới hiện lên, chọn Verilog > Device Macro instantiation > Artix-7 > RAM > Dual Clock FIFO.

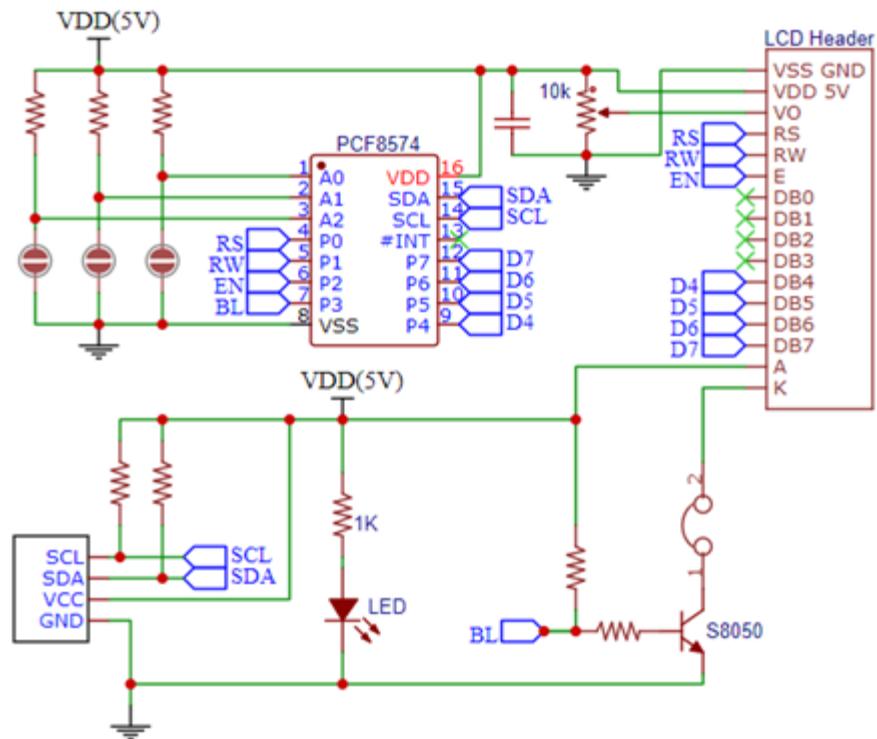


Hình 0.5 Đoạn mã mẫu được dùng để khởi tạo Dual-clock FIFO IP

Mô-đun PCF8574

Mô-đun PCF8574 là một bộ mở rộng cổng vào/ra (I/O) giao tiếp qua chuẩn I2C, được sử dụng phổ biến trong các ứng dụng nhúng để mở rộng số lượng chân I/O của vi điều khiển hoặc vi xử lý. PCF8574 có thể được cấu hình như một bộ mở rộng 8 chân I/O, trong đó các chân này có thể hoạt động độc lập ở chế độ đầu vào hoặc đầu ra. Ngoài ra, nhiều mô-đun PCF8574 có thể được nối với nhau trên cùng một bus I2C nhờ khả năng địa chỉ hóa linh hoạt của giao thức truyền/nhận dữ liệu.

Mô-đun PCF8574 sẽ được sử dụng để kết nối với màn hình LCD 16x2. Việc điều khiển màn hình LCD 16x2 theo cách truyền thông đòi hỏi ít nhất 6 chân GPIO từ vi điều khiển để kết nối với các chân RS, EN và các đường dữ liệu (D4–D7). Tuy nhiên, khi tài nguyên GPIO hạn chế, mô-đun PCF8574 là giải pháp hiệu quả để giảm số lượng chân cần thiết. PCF8574, với giao tiếp I2C, cho phép điều khiển toàn bộ màn hình chỉ qua hai chân (SDA và SCL) từ vi điều khiển.



Hình 0.6 Sơ đồ kết nối màn hình LCD với mô-đun PCF8574

Khi kết hợp PCF8574 với màn hình LCD 16x2, các tín hiệu điều khiển và dữ liệu được ánh xạ thông qua các chân của PCF8574 như được thể hiện ở trong Hình 0.6. Khi đó màn hình LCD sẽ hoạt động ở chế độ 4-bit. Dữ liệu được gửi từ Arty A7-35T tới mô-đun PCF8574 bao gồm 8-bit trong đó 4-bit trọng số thấp (LSB) tương ứng với các chân BL, EN, RW, RS của LCD, 4-bit có trọng số cao (MSB) là dữ liệu được truyền tới LCD. Do đó để gửi 1-byte $b_7b_6b_5b_4b_3b_2b_1b_0$ tới LCD thì ta phải truyền 4 lần với khuôn dạng dữ liệu như sau:

Bảng 0.1 Giá trị các bit dữ liệu gửi tới mô-đun PCF8574 để điều khiển màn hình LCD

Lần	P₇	P₆	P₅	P₄	P₃	P₂	P₁	P₀
	D₇	D₆	D₅	D₄	BL	EN	RW	RS
1	b_7	b_6	b_5	b_4	BL	1	RW	RS
2	b_7	b_6	b_5	b_4	BL	0	RW	RS
3	b_3	b_2	b_1	b_0	BL	1	RW	RS
4	b_3	b_2	b_1	b_0	BL	0	RW	RS

Trong đó:

- BL: backlight (0: off, 1: on)
- EN: enable
- RW: read/write (0: write, 1: read)
- RS: register select (0: command, 1: data)
- D_7, D_6, D_5, D_4 : 4 chân data có trọng số cao của LCD
- $P_7, P_6, P_5, P_4, P_3, P_2, P_1, P_0$: Là các chân của mô-đun PCF8574