**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA KỸ THUẬT MÁY TÍNH**

Logo, company name

Description automatically generated

**BÁO CÁO ĐỒ ÁN 2**

**TÌM HIỂU VÀ THIẾT KẾ KIỂM TRA ĐỌC GHI ẢNH BẰNG SYSTEM VERILOG**

|  |  |
| --- | --- |
| **GIẢNG VIÊN HƯỚNG DẪN:** | **TRƯƠNG VĂN CƯƠNG** |
| **SINH VIÊN THỰC HIỆN:** | **CAO CHÍ NHÂN - 19520794** |

**Thành phố Hồ Chí Minh, 1/2023**

MỤC LỤC

[CHƯƠNG 1. GIỚI THIỆU TỔNG QUAN 5](#_Toc123981073)

[CHƯƠNG 2. TÌM HIỂU LÝ THUYẾT 6](#_Toc123981074)

[2.1. System Verilog 6](#_Toc123981075)

[2.2. Thành phần trong SystemVerilog testbench 6](#_Toc123981076)

[2.2.1. DUT 7](#_Toc123981077)

[2.2.2. Interface 7](#_Toc123981078)

[2.2.3. Driver 7](#_Toc123981079)

[2.2.4. Generator 8](#_Toc123981080)

[2.2.5. Monitor 8](#_Toc123981081)

[2.2.6. Scoreboard 8](#_Toc123981082)

[2.2.7. Test 9](#_Toc123981083)

[CHƯƠNG 3. THIẾT KẾ VÀ HIỆN THỰC 10](#_Toc123981084)

[3.1. Thiết kế SystemVerilog testbench cho bộ Adder 10](#_Toc123981085)

[3.1.1. Kiến trúc của một SystemVerilog testbench đơn giản 10](#_Toc123981086)

[3.2. Tìm hiểu SystemVerilog testbench trong xử lý ảnh 20](#_Toc123981087)

[3.2.1. APIs cho Thư viện xử lý Bitmap 21](#_Toc123981088)

[3.2.2. Các tham số, Interface và Task trong axis\_video\_image\_in\_vip 22](#_Toc123981089)

[3.2.3. Các tham số, Interface và Task trong axis\_video\_image\_out\_vip 24](#_Toc123981090)

[CHƯƠNG 4. ĐÁNH GIÁ KẾT QUẢ 26](#_Toc123981091)

[4.1. Thiết kế kiểm tra bộ cộng bằng SystemVerilog Testbench 26](#_Toc123981092)

[4.2. Kết quả thực tế khi thực thi Verification IP tham khảo 27](#_Toc123981093)

[CHƯƠNG 5. HẠN CHẾ VÀ HƯỚNG PHÁT TRIỂN 28](#_Toc123981094)

[5.1. Hạn chế 28](#_Toc123981095)

[5.2. Hướng phát triển 28](#_Toc123981096)

[TÀI LIỆU THAM KHẢO 29](#_Toc123981097)

**DANH MỤC HÌNH ẢNH**

[Hình 3.1 Kiến trúc của SystemVerilog testbench 10](#_Toc123981098)

[Hình 4.1 Kết quả Adder 26](#_Toc123981099)

[Hình 4.2 Kết quả mô phỏng thực tế của IP tham khảo 27](#_Toc123981100)

**DANH MỤC BẢNG**

[Bảng 2.1 Các thành phần trong testbench 6](#_Toc123981101)

[Bảng 3.1 Class Transaction 10](#_Toc123981102)

[Bảng 3.2 Class Generator 11](#_Toc123981103)

[Bảng 3.3 Interface 12](#_Toc123981104)

[Bảng 3.4 Class Driver 13](#_Toc123981105)

[Bảng 3.5 Class Monitor 14](#_Toc123981106)

[Bảng 3.6 Class Scoreboard 16](#_Toc123981107)

[Bảng 3.7 Class Environment 17](#_Toc123981108)

[Bảng 3.8 Program Test 19](#_Toc123981109)

[Bảng 3.9 Thành phần trong gói thư viện 21](#_Toc123981110)

[Bảng 3.10 Các biến trong Enum BitmapReadError 21](#_Toc123981111)

[Bảng 3.11 Các biến trong Enum BitmapWriteError 22](#_Toc123981112)

[Bảng 3.12 Tham số của module “axis\_video\_image\_in\_vip” 22](#_Toc123981113)

[Bảng 3.13 Interface trong module “axis\_video\_image\_in\_vip” 22](#_Toc123981114)

[Bảng 3.14 Tham số của module “axis\_video\_image\_out\_vip” 24](#_Toc123981115)

[Bảng 3.15 Interface trong “axis\_video\_image\_out\_vip” 24](#_Toc123981116)

# GIỚI THIỆU TỔNG QUAN

Các Ngôn ngữ Mô tả Phần cứng (HDL) như Verilog và VHDL được sử dụng để mô tả hành vi của phần cứng để có thể chuyển đổi nó sang các khối digital được tạo thành từ các cổng tổ hợp (combinational gates) và các phần tử tuần tự (sequential elements). Để kiểm tra rằng mô tả phần cứng trong HDL là chính xác, cần có một ngôn ngữ có nhiều tính năng hơn trong hướng đối tượng (OOP) sẽ hỗ trợ các quy trình kiểm tra phức tạp và thường được gọi là Ngôn ngữ xác minh phần cứng – Hardware Verification Language.

Trở lại những năm 1990, Verilog là ngôn ngữ chính để xác minh chức năng của các thiết kế nhỏ, không quá phức tạp và có ít tính năng hơn. Khi độ phức tạp của thiết kế tăng lên, thì yêu cầu về các công cụ tốt hơn để thiết kế và xác minh nó cũng tăng theo. SystemVerilog vượt trội hơn nhiều so với Verilog vì khả năng thực hiện các kích thích ngẫu nhiên bị hạn chế, sử dụng các tính năng OOP trong xây dựng testbench, phạm vi chức năng, xác nhận trong số nhiều tính năng khác.

SystemVerilog là một phần mở rộng của Verilog với nhiều tính năng xác minh như vậy cho phép các kỹ sư xác minh thiết kế bằng cách sử dụng các cấu trúc testbench phức tạp và kích thích ngẫu nhiên trong mô phỏng.

Verification là quá trình đảm bảo rằng một thiết kế phần cứng nhất định hoạt động như mong đợi. Thiết kế chip là một quá trình rất rộng rãi, tốn thời gian và tiêu tốn hàng triệu đô la để chế tạo. Các lỗi chức năng trong thiết kế nếu được phát hiện ở giai đoạn sớm hơn trong quá trình thiết kế sẽ giúp tiết kiệm chi phí. Nếu một lỗi được tìm thấy sau đó trong quy trình thiết kế, thì tất cả các bước thiết kế phải được lặp lại một lần nữa, điều này sẽ sử dụng nhiều tài nguyên, tiền bạc và thời gian hơn. Nếu toàn bộ quy trình thiết kế phải được lặp lại, thì nó được gọi là một lần *respin* của chip.

# TÌM HIỂU LÝ THUYẾT

## System Verilog

SystemVerilog (SV) là chuẩn ngôn ngữ hợp nhất dành cho việc kiểm tra (verification), mô tả kỹ thuật (specification) và thiết kế (design) phần cứng. SV là một chuẩn được mở rộng từ chuẩn Verilog HDL giúp nâng cao khả năng mô tả thiết kế phần cứng và xây dựng thành phần kiểm tra.

SystemVerilog được phát triển với mục tiêu tạo ra một ngôn ngữ hợp nhất hỗ trợ cho cả thiết kế phần cứng, mô phỏng và mô tả hệ thống.SV được phát triển lần đầu bởi tổ chức Accellera trước khi được chuẩn hóa bởi hiệp hội IEEE.

## Thành phần trong SystemVerilog testbench

Testbench cho phép xác minh chức năng của một thiết kế thông qua mô phỏng. Là một containter nơi thiết kế được đặt và điều khiển với các kích thích đầu vào khác nhau.

Bảng . Các thành phần trong testbench

|  |  |
| --- | --- |
| **Thành phần** | **Mô tả** |
| Transaction | Xác định hoạt động cấp độ pin được tạo bởi agent (để lái tới DUT thông qua trình điều khiển) hoặc hoạt động phải được quan sát bởi agent (Phần placeholder hoạt động được giám sát theo dõi trên tín hiệu DUT) |
| Generator | Tạo các stimulus đầu vào khác nhau để được chuyển đến DUT |
| Interface | Chứa các tín hiệu thiết kế có thể được điều khiển hoặc giám sát |
| Driver | Điều khiển các stimulus được tạo ra cho thiết kế |
| Monitor | Giám sát các cổng đầu vào-đầu ra của thiết kế để nắm bắt hoạt động thiết kế |
| Scoreboard | Kiểm tra đầu ra từ thiết kế với hành vi dự kiến |
| Environment | Chứa tất cả các thành phần verification được đề cập ở trên |
| Test | Chứa môi trường có thể được điều chỉnh với các cài đặt cấu hình khác nhau |

### DUT

DUT là viết tắt của Design Under Test và là thiết kế phần cứng được viết bằng Verilog hoặc VHDL. DUT là một thuật ngữ thường được sử dụng để xác nhận sau khi silicon sau khi chip được chế tạo. Trong quá trình xác thực trước, nó còn được gọi là Thiết kế đang được xác minh (Design Under Verification), viết tắt là DUV.

### Interface

Nếu thiết kế chứa hàng trăm tín hiệu cổng thì việc kết nối, duy trì và sử dụng lại các tín hiệu đó sẽ trở nên cồng kềnh. Thay vào đó, có thể đặt tất cả các cổng đầu vào-đầu ra được thiết kế vào một containter sẽ trở thành interface cho DUT. Sau đó, thiết kế có thể được định hướng với các giá trị thông qua giao diện này.

### Driver

Driver là thành phần verification thực hiện thao tác vặn chốt (pin-wiggling) của DUT, thông qua một tác vụ được xác định trong interface.

Khi trình điều khiển phải điều khiển một số giá trị đầu vào cho thiết kế, nó chỉ cần gọi tác vụ được xác định trước này trong interface mà không thực sự biết mối quan hệ thời gian giữa các tín hiệu này.

Thông tin thời gian được xác định trong task được cung cấp bởi interface. Đây là mức độ trừu tượng cần thiết để làm cho testbenches linh hoạt hơn và có thể mở rộng. Trong tương lai, nếu interface thay đổi, driver mới có thể gọi cùng một tác vụ và điều khiển các tín hiệu theo một cách khác.

### Generator

Generator là một verification component có thể tạo các transaction dữ liệu hợp lệ và gửi chúng đến driver. Sau đó, driver có thể chỉ cần điều khiển dữ liệu do trình tạo cung cấp cho nó thông qua giao diện. Công việc của driver là lấy đối tượng dữ liệu và dịch nó thành thứ gì đó mà DUT có thể hiểu được.

### Monitor

Nhận các tín hiệu interface và chuyển đổi hoạt động ở mức tín hiệu sang mức transaction. DUT xử lý dữ liệu đầu vào và gửi kết quả đến các chân đầu ra. Monitor chọn dữ liệu đã xử lý, chuyển đổi nó thành một đối tượng dữ liệu và gửi nó đến scoreboard thông quan mailbox.

### Scoreboard

Scoreboard có thể có một mô hình tham chiếu hoạt động giống như DUT. Mô hình này phản ánh hành vi dự kiến ​​của DUT. Đầu vào được gửi đến DUT cũng được gửi đến mô hình tham chiếu này. Vì vậy, nếu DUT có vấn đề về chức năng, thì đầu ra từ DUT sẽ không khớp với đầu ra từ mô hình tham chiếu.

Vì vậy, việc so sánh kết quả đầu ra từ thiết kế và mô hình tham chiếu sẽ cho biết liệu có khiếm khuyết về chức năng trong thiết kế hay không. Điều này thường được thực hiện trong scoreboard

### Test

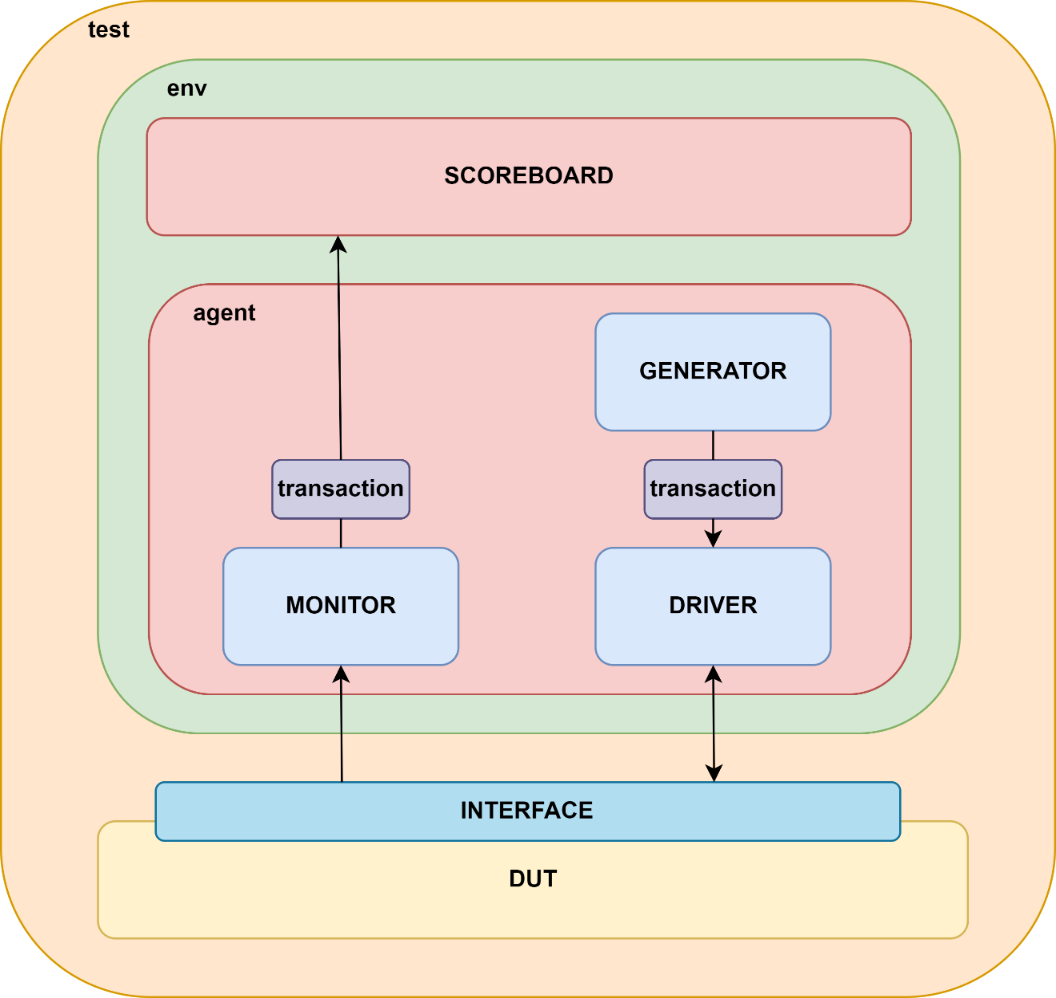
Test sẽ khởi tạo một đối tượng của environment và định cấu hình nó theo cách mà module test muốn. Có thể sẽ có hàng nghìn test và việc thực hiện các thay đổi trực tiếp đối với environment cho mỗi test là không khả thi.

Thay vào đó, ta muốn các parameter nhất định trong môi trường có thể được điều chỉnh cho mỗi lần test. Bằng cách đó, thử nghiệm sẽ có khả năng kiểm soát cao hơn đối với việc tạo ra stimulus và sẽ hiệu quả hơn.

# THIẾT KẾ VÀ HIỆN THỰC

## Thiết kế SystemVerilog testbench cho bộ Adder

### Kiến trúc của một SystemVerilog testbench đơn giản



Hình . Kiến trúc của SystemVerilog testbench

Lấy ví dụ của một testbench cho bộ Adder có đủ Monitor và Scoreboard;

**Transaction class** có chức năng tạo ra cái stimulus ngẫu nhiên:

Bảng . Class Transaction

|  |
| --- |
| *//Khai báo class transaction*  class transaction;  *//khai báo các thành phần có trong transaction*  rand bit [3:0] a; //tạo các stimulus ngẫu nhiên  rand bit [3:0] b;  bit [6:0] c;  function void display(string name);  *//thêm display() để hiển thị các transaction property*  $display("-------------------------");  $display("- %s ",name);  $display("-------------------------");  $display("- a = %0d, b = %0d",a,b);  $display("- c = %0d",c);  $display("-------------------------");  endfunction  endclass |

**Generator** **class** đảm nhận việc tạo ra các stimulus bằng cách tạo ngẫu nhiên các transaction class và gửi các transaction đã được tạo ngẫu nhiên cho Driver. Thêm vào mailbox và event, trong đó, mailbox được sử dụng để gửi các transaction ngẫu nhiên đến Driver, event để chỉ ra việc tạo các transaction của generator kết thúc.

Khai báo mailbox và event, lấy mailbox handle từ class environment, thêm biến để kiểm soát số lượng transaction packet được tạo ra. Thêm một event để chỉ ra quá trình tạo ra đã hoàn thành, event sẽ được kích hoạt khi quá trình tạo ra transaction hoàn thành.

Bảng . Class Generator

|  |
| --- |
| class generator;  //Khai báo transaction class  rand transaction trans;  //Khai báo mailbox  mailbox gen2driv;  *//repeat count, chỉ ra số lượng transaction cần tạo*  int repeat\_count;  *//event, chỉ ra việc tạo ra transaction kết thúc*  event ended;  *//khởi tạo mailbox*  function new(mailbox gen2driv);  *//lấy mailbox handle từ environment*  this.gen2driv = gen2driv;  endfunction  *//task main, tạo ngẫu nhiên số lượng repeat\_count các packet transaction và truyền vào mailbox*  task main();  repeat(repeat\_count) begin  trans = new();  if( !trans.randomize() ) $fatal("Gen:: trans randomization failed");  gen2driv.put(trans);  -> ended; *//generator kết thúc*  endtask  endclass |

**Interface** có chức năng nhóm các tín hiệu, một interface đơn giản không có modport và clocking block.

Bảng . Interface

|  |
| --- |
| interface intf(input logic clk,reset);  //khai báo các tín hiệu  logic valid;  logic [3:0] a;  logic [3:0] b;  logic [6:0] c;  endinterface |

**Driver class** đảm nhận việc nhận stimulus được tạo ra từ Generator và lái đến DUT bằng cách chuyển các giá trị của transaction class đến các tín hiệu từ interface. Các công việc bao gồm khai báo interface và mailbox, lấy interface và mailbox handle thông qua một hàm khởi tạo.

Thêm vào task reset để khởi tạo tín hiệu interface ở giá trị mặc định, task driver để lái các transaction packet đến các tín hiệu của interface. Thêm một biến cục bộ để theo dấu các packet đã được điều khiển.

Bảng . Class Driver

|  |
| --- |
| class driver;  *//đếm số lượng transaction*  int no\_transactions;  *//tạo virtual interface handle*  virtual intf vif;  *//tạo mailbox handle*  mailbox gen2driv;  *//hàm khởi tạo*  function new(virtual intf vif,mailbox gen2driv);  *//nhận interface*  this.vif = vif;  *//nhận mailbox handle từ environment*  this.gen2driv = gen2driv;  endfunction  *//task reset, reset lại các tín hiệu interface thành giá trị mặc định*  task reset;  wait(vif.reset);  $display("[ DRIVER ] ----- Reset Started -----");  vif.a <= 0;  vif.b <= 0;  vif.valid <= 0;  wait(!vif.reset);  $display("[ DRIVER ] ----- Reset Ended -----");  endtask  task main;  forever begin  transaction trans;  gen2driv.get(trans);  @(posedge vif.clk);  vif.valid <= 1;  vif.a <= trans.a;  vif.b <= trans.b;  @(posedge vif.clk);  vif.valid <= 0;  trans.c <= vif.c;  @(posedge vif.clk);  trans.display("[ Driver ]");  no\_transactions++;  end  endtask  endclass |

**Monitor Class** có chức năng lấy mẫu các tín hiệu interface và chuyển đổi hoạt động ở mức tín hiệu thành mức transaction, gửi các transaction mẫu đến scoreboard thông qua mailbox.

Bảng . Class Monitor

|  |
| --- |
| class monitor;  *//tạo handle virtual interface*  virtual intf vif;  *//tạo mailbox handle*  mailbox mon2scb;  *//hàm khởi tạo*  function new(virtual intf vif,mailbox mon2scb);  this.vif = vif;  this.mon2scb = mon2scb;  endfunction    *//lấy mẫu tín hiệu interface và gửi đến scoreboard*  task main;  forever begin  transaction trans;  trans = new();  @(posedge vif.clk);  wait(vif.valid);  trans.a = vif.a;  trans.b = vif.b;  @(posedge vif.clk);  trans.c = vif.c;  @(posedge vif.clk);  mon2scb.put(trans);  trans.display("[ Monitor ]");  end  endtask  endclass |

**Scoreboard Class** nhận các packet mẫu từ monitor và so sánh với kết quả mong muốn, lỗi sẽ được thông báo nếu so sánh giữa các kết quả không trùng khớp. Công việc bao gồm khai báo mailbox và biến để đếm transactions, kết nối handle thông qua hàm khởi tạo.

Bảng . Class Scoreboard

|  |
| --- |
| class scoreboard;    *//tạo mailbox handle*  mailbox mon2scb;    *//được sử dụng để đếm số lượng transactions*  int no\_transactions;    *//hàm khởi tạo*  function new(mailbox mon2scb);  this.mon2scb = mon2scb;  endfunction    *//So sánh kết quả thực tế với kết quả mong muốn*  task main;  transaction trans;  forever begin  mon2scb.get(trans);  if((trans.a+trans.b) == trans.c)  $display("Result is as Expected");  else  $error("WrongResult.\n\tExpeced: %0d Actual: %0d",(trans.a+trans.b),trans.c);  no\_transactions++;  trans.display("[ Scoreboard ]");  end  endtask  endclass |

**Environment Class** là một class chứa Mailbox, Generator và Driver, Monitor. Các công việc bao gồm tạo ra mailbox, generator và driver cùng chia sẻ mailbox handle giữa generator và driver.

Để truy cập tốt hơn, hoạt động của Generator và Driver có thể được chia và kiểm soát bởi ba task: pre\_test() – khởi tạo, reset; test() – gọi hàm tạo và điều khiển Stimulus; post\_test() – đợi việc tạo và điều khiển hoàn thành.

Bảng . Class Environment

|  |
| --- |
| `include "transaction.sv"  `include "generator.sv"  `include "driver.sv"  `include "monitor.sv"  `include "scoreboard.sv"  class environment;  generator gen;  driver driv;  monitor mon;  scoreboard scb;    mailbox gen2driv;  mailbox mon2scb;    virtual intf vif;    *//hàm khởi tạo*  function new(virtual intf vif);  //get the interface from test  this.vif = vif;    *//tạo mailbox(handle giống nhau sẽ được chia sẻ giữa generator và driver)*  gen2driv = new();  mon2scb = new();  *//tạo generator, driver, monitor và scoreboard*  gen = new(gen2driv);  driv = new(vif,gen2driv);  mon = new(vif,mon2scb);  scb = new(mon2scb);  endfunction  //  task pre\_test();  driv.reset();  endtask    task test();  fork  gen.main();  driv.main();  mon.main();  scb.main();  join\_any  endtask    task post\_test();  wait(gen.ended.triggered);  wait(gen.repeat\_count == driv.no\_transactions); //Optional  wait(gen.repeat\_count == scb.no\_transactions);  endtask    task run;  pre\_test();  test();  post\_test();  $finish;  endtask  endclass |

**Program Test** đảm nhận việc tạo ra environment, điều chỉnh testbench và số lượng transaction cần tạo ra, khởi tạo điều khiển stimulus.

Bảng . Program Test

|  |
| --- |
| `include "environment.sv"  program test(intf intf);  environment env;  initial begin  *//tạo ra environment*  env = new(intf);  *//cho số lượng transaction mà generator tạo ra là 10*  env.gen.repeat\_count = 10;  env.run();  end  endprogram |

**TestBench Top** có chức năng kết nối giữa DUT và Testbench, bao gồm DUT, Test và các biến Interface

|  |
| --- |
| `include "interface.sv"  `include "random\_Test.sv"  module tb\_top;  bit clk;  bit reset;    always #5 clk = ~clk;    initial begin  reset = 1;  #5 reset =0;  end    intf i\_intf(clk,reset);    test t1(i\_intf);  adder DUT (  .clk(i\_intf.clk),  .reset(i\_intf.reset),  .a(i\_intf.a),  .b(i\_intf.b),  .valid(i\_intf.valid),  .c(i\_intf.c)  );  initial begin  $dumpfile("dump.vcd"); $dumpvars;  end  endmodule |

## Tìm hiểu SystemVerilog testbench trong xử lý ảnh

Để có thể kiểm tra IP video hoặc hình ảnh, cần phải đọc ảnh thực tế vào thiết kế, gửi dữ liệu thông qua một interface. Sau đó nhận được output từ interface và chuyển đổi sang ảnh mới.

Để giải quyết vấn đề phức tạp này, nhóm tác giả [1] đã xây đựng một thư viện có tên là “Bitmap Processing Library” và IP kiểm tra (VIP) “AXI-Stream Video Image VIP” có thể đơn giản hóa các bước thiết kế kiểm tra. Một thư viện xử lý bitmap có thể đọc và ghi các file bitmap thành một mảng bit bằng SystemVerilog. Với ưu điểm là giao tiếp API dễ dàng, mã nguồn mở, tương thích với các công cụ EDA phổ biến như Modelsim.

Bảng . Thành phần trong gói thư viện

|  |  |
| --- | --- |
| Tên file | Mô tả |
| bitmap\_processing.sv | Thư viện xử lý bitmap |
| axi\_stream\_video\_image.sv | Thư viện luồng AXI video và hình ảnh (Được sử dụng cho Callback class) |
| axi\_stream\_video\_image\_in\_vip.sv | IP kiểm tra đọc bitmap và gửi đi cho interface |
| axi\_stream\_video\_image\_out\_vip.sv | IP kiểm tra hiển thị interface và lưu ảnh vào bitmap |
| axi\_stream\_video\_image\_example.sv | Ví dụ cho sử dụng IP kiểm tra và thư viện |

### APIs cho Thư viện xử lý Bitmap

\_ Class Bitmap: Là lớp cơ sở của bitmap. Cần khởi tạo một instance của lớp này để sử dụng được các hàm API. Một instance của lớp Bitmap thường lưu trữ và tương ứng một bức ảnh.

\_ Enum Bitmap::BitmapReadError: Liệt kê các lỗi đọc file bitmap

Bảng . Các biến trong Enum BitmapReadError

|  |  |
| --- | --- |
| **Tên biến** | **Mô tả** |
| BMPR\_ERR\_OK | Không có lỗi |
| BMPR\_ERR\_FILE | Lỗi mở file, file không tồn tại |
| BMPR\_ERR\_FILE\_FORMAT | Dạng file không phải là bitmap (.bmp) |
| BMPR\_ERR\_BPP\_FORMAT | Không phải là file bitmap 24-bpp hoặc 32-bpp |
| BMPR\_ERR\_BMP\_COMPRESSED | File bị nén, không thể đọc |

\_ Enum Bitmap::BitmapWriteError: Liệt kê các lỗi ghi file bitmap

Bảng . Các biến trong Enum BitmapWriteError

|  |  |
| --- | --- |
| **Tên biến** | **Mô tả** |
| BMPR\_ERR\_OK | Không có lỗi |
| BMPR\_ERR\_FILE\_FILE | Dạng file không phải là bitmap (.bmp) |

### Các tham số, Interface và Task trong axis\_video\_image\_in\_vip

Bảng . Tham số của module “axis\_video\_image\_in\_vip”

|  |  |  |
| --- | --- | --- |
| **Tham số** | **Kiểu dữ liệu** | **Mô tả** |
| IMAGE\_WIDTH | int | Chiều rộng của bitmap được đọc vào và ghi vào interface |
| IMAGE\_HEIGHT | int | Chiều cao của bitmap được đọc vào và ghi vào interface |
| PIXEL\_PER\_CLK | int | pixel trên mỗi clock trong interface, có thể là 1, 2, 4, 8 |
| BITS\_PER\_PIXEL | int | Số bit trên mỗi pixel, được fix ở 32bpp |

Bảng . Interface trong module “axis\_video\_image\_in\_vip”

|  |  |  |
| --- | --- | --- |
| **Port** | **Kiểu dữ liệu** | **Mô tả** |
| frameBitmap | Bitmap | Tham chiếu của một bitmap instance |
| callbacks | AxisVideoImageCallback | Lớp ảo của VIP. VIP sẽ gọi hàm callback khi hoàn thành việc gửi một ảnh trong interface. |
| clk | logic | Clock của interface AXI-Stream |
| m\_axis\_video\_out\_tdata | logic | Dòng dữ liệu của interface master AXI-Stream. Độ rộng của port này được quyết định bởi tham số PIXEL\_PER\_CLK |
| m\_axis\_video\_out\_tvalid | logic | Tín hiệu handshake có hiệu lực của interface master AXI-Stream |
| m\_axis\_video\_out\_tlast | logic | Tín hiệu dòng cuối cùng, khi ảnh ở pixel cuối cùng của mộ dòng, tín hiệu này sẽ là “1” |
| m\_axis\_video\_out\_tuser | logic | Tín hiệu bắt đầu khung hình, khi ảnh ở pixel đầu tiên của khung hình, tín hiệu này sẽ là “1” |
| m\_axis\_video\_out\_tready | logic | Giám sát tín hiệu handshake của interface slave AXI-Stream |

- Các Task có trong ip bao gồm ReadAndSendBitmap(fileName) và SendBitmap.

Task ReadAndSendBitmap tự động mở và đọc bitmap từ thiết bị đến instance *framBitmap*. Sau đó gửi đến interface AXI-Stream ngay lập tức.

Task SendBitmap() gửi hình ảnh trong frameBitmap đến interface AXI-Stream. Phải đảm bảo đã có một tấm ảnh trong instance frameBitmap, trong trường hợp chưa có phải tạo hoặc đọc từ một file.

### Các tham số, Interface và Task trong axis\_video\_image\_out\_vip

Bảng . Tham số của module “axis\_video\_image\_out\_vip”

|  |  |  |
| --- | --- | --- |
| **Tham số** | **Kiểu dữ liệu** | **Mô tả** |
| IMAGE\_WIDTH | int | Chiều rộng của bitmap được đọc vào và ghi vào interface |
| IMAGE\_HEIGHT | int | Chiều cao của bitmap được đọc vào và ghi vào interface |
| PIXEL\_PER\_CLK | int | pixel trên mỗi clock trong interface, có thể là 1, 2, 4, 8 |
| BITS\_PER\_PIXEL | int | Số bit trên mỗi pixel, được fix ở 32bpp |

Bảng . Interface trong “axis\_video\_image\_out\_vip”

|  |  |  |
| --- | --- | --- |
| **Port** | **Kiểu dữ liệu** | **Mô tả** |
| frameBitmap | Bitmap | Tham chiếu của một bitmap instance |
| callbacks | AxisVideoImageCallback | Lớp ảo của VIP. VIP sẽ gọi hàm callback khi hoàn thành việc gửi một ảnh trong interface. |
| clk | logic | Clock của interface AXI-Stream |
| m\_axis\_video\_in\_tdata | logic | Dòng dữ liệu của interface slave AXI-Stream. Độ rộng của port này được quyết định bởi tham số PIXEL\_PER\_CLK |
| m\_axis\_video\_in\_tvalid | logic | Tín hiệu handshake có hiệu lực của interface slave AXI-Stream |
| m\_axis\_video\_in\_tlast | logic | Tín hiệu dòng cuối cùng, khi ảnh ở pixel cuối cùng của mộ dòng, tín hiệu này sẽ là “1” |
| m\_axis\_video\_in\_tuser | logic | Tín hiệu bắt đầu khung hình, khi ảnh ở pixel đầu tiên của khung hình, tín hiệu này sẽ là “1” |
| m\_axis\_video\_in\_tready | logic | Giám sát tín hiệu handshake của interface slave AXI-Stream |

Task trong IP này là ReceiveAndSaveBitmap(filename). Task sẽ bắt đầu nghe một khung hình mới trên interface AXI-Stream monitor. Khi một khung hình mới bắt đầu truyền trên interface, VIP sẽ tự động ghi lại hình ảnh và lưu vào instance *frameBitmap*.

# ĐÁNH GIÁ KẾT QUẢ

## Thiết kế kiểm tra bộ cộng bằng SystemVerilog Testbench

Text, letter

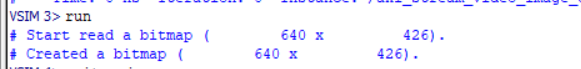
Description automatically generated

Hình . Kết quả Adder

Generator tạo ra 5 transaction ngẫu nhiên, sau đó Driver dẫn các tín hiệu mang giá trị của a, b, c vào Monitor lấy kết quả mẫu, sau đó Scoreboard thực hiện việc so sánh giữa kết quả thực tế và kết quả mong muốn, nếu kết quả trùng khớp sẽ in ra màn hình “Result as Expected”.

## Kết quả thực tế khi thực thi Verification IP tham khảo

Thử nghiệm với ảnh một ảnh dạng bitmap (.bmp) 640x426, IP có thể đọc được ảnh, tuy nhiên không thể hoàn thành đầy đủ các chức năng trong transaction như hiển thị được thông báo đã nhận được frame, ghi bitmap vào file mới, chưa hiển thị được xung clock trong dạng sóng,tín hiệu hoàn thành và chưa xuất ra được một tấm ảnh ở đầu ra (output).



Hình . Kết quả mô phỏng thực tế của IP tham khảo

# HẠN CHẾ VÀ HƯỚNG PHÁT TRIỂN

## Hạn chế

Chưa tìm ra hướng giải quyết của IP có sẵn để hoàn thiện việc đọc ảnh ở đầu vào và quá trình kiểm tra để có được ảnh ở đầu ra.

Chưa thể tự thiết kế được bộ kiểm tra hình ảnh bằng SystemVerilog mà chỉ đừng lại ở việc tìm hiểu lý thuyết và cài đặt các IP có sẵn.

## Hướng phát triển

Hoàn thiện đọc ảnh và quá trình kiểm tra (verification) để xuất ra được tấm ảnh mong muốn.

Nghiên cứu tìm hiểu phương pháp UVM (Universal Verification Methodology) trong xử lý ảnh

# TÀI LIỆU THAM KHẢO

[1] SystemVerilog Ports & Data Types For Simple, Efficient and Enhanced HDL Modeling

[2] SystemVerilog's Virtual World - An Introduction to Virtual Classes, Virtual Methods and Virtual Interface Instances

[3]<https://verificationguide.com/systemverilog/systemverilog-tutorial> [Online]

[4] <https://www.chipverify.com/systemverilog/systemverilog-tutorial> [Online]

[5]<https://github.com/Aperture-Electronic/SystemVerilog-Bitmap-Library-AXI-Image-VIP> [Online]