Xây dựng hệ thống mô phỏng và kiểm chứng cho bộ mã hoá tín hiệu video H.264/AVC

Bùi Duy Hiếu, Đặng Nam Khánh, Nguyễn Ngọc Mai, Nguyễn Kiêm Hùng, Trần Xuân Tú
PTN mục tiêu Hệ thống tích hợp thông minh, Trường Đại học Công nghệ, Đại học Quốc gia Hà Nội
144 đường Xuân Thuỷ, Cầu Giấy, Hà Nội, Việt Nam
Email: {hieubd,tutx}@vnu.edu.vn

***Tóm tắt*—Sự phức tạp của hệ thống mã hoá tín hiệu video H.264/AVC đã dẫn tới nhiều khó khăn trong việc kiểm tra, kiểm chứng và kiểm thực các thiết kế ở mức hệ thống của bộ mã hoá này. Những khó khăn chúng tôi gặp phải khi thiết kế và thực thi hệ thống này là việc quản lý các kịch bản kiểm tra; thay đổi các tham số của hệ thống và của kịch bản kiểm tra; và thời gian chạy mô phỏng lâu đối với các tập dữ liệu dùng để kiểm tra và kiểm chứng hệ thống, đặc biệt là khi hệ thống được ghép nối với một hệ thống trên chip (System-on-chip). Trong báo cáo này, chúng tôi trình bày một phương pháp xây dựng hệ thống mô phỏng và kiểm chứng ở mức hệ thống bộ mã hoá tín hiệu video H.264/AVC dựa trên việc thiết kế kịch bản kiểm tra và kiểm chứng tối giản và linh động, kết hợp với các phần mềm mã nguồn mở sẵn có như GNU Make, ngôn ngữ Python để tạo thành một hệ thống kiểm tra và kiểm chứng tự động, cho phép tự động mô phỏng thiết kế từ mức hành xử cho đến mức cổng lô-gíc. Hệ thống này cho phép chạy nhiều mô phỏng tại cùng một thời điểm với các kịch bản kiểm tra khác nhau. Nhờ việc này, chúng tôi đã giảm được thời gian chạy mô phỏng từ 2 đến 10 lần phụ thuộc vào khả năng của máy chạy mô phỏng. Kịch bản kiểm tra được thực hiện trên VHDL để kiểm tra thiết kế của bộ mã hoá tín hiệu video H.264/AVC thực thi trên công nghệ $130nm$ của hãng Global Foundry.**

***Từ khóa*—Mô hình hoá, mô phỏng, kiểm chứng, H.264/AVC, hệ thống trên chip, kịch bản kiểm tra**

I. GIỚI THIỆU

Vấn đề kiểm tra và kiểm chứng hệ thống là một trong những vấn đề quan trọng và cần được xem xét ngay từ thời điểm bắt đầu thiết kế hệ thống. Thiết kế càng phức tạp thì vấn đề kiểm tra và kiểm chứng thiết kế lại càng quan trọng và khó thực hiện. Đối với các khối nhỏ, việc kiểm tra và kiểm chứng thiết kế không khó thực hiện, vì số trường hợp phải kiểm tra không nhiều. Tuy nhiên, đối với các hệ thống phức tạp, bao gồm nhiều khối chứng năng giao tiếp với nhau và giao tiếp với bộ xử lý và bộ nhớ ngoài như hệ thống mã hóa tín hiệu video H.264/AVC, việc kiểm tra và kiểm chứng hệ thống là một vấn đề phức tạp.

Các hệ thống lớn như bộ mã hoá H.264/AVC có nhiều khối chức năng và nhiều tham số. Do vậy, để kiểm tra các chức năng này đòi hỏi kịch bản kiểm tra phải linh động, cho phép thay đổi tham số mà không phải thay đổi các mã nguồn kiểm tra. Từ đó, hệ thống có thể được kiểm tra với các tham số thay đổi theo ý muốn của người thiết kế mà không phải sửa lại mã nguồn của chương trình và không cần phải biên dịch lại thiết kế cho mỗi lần chạy mô phỏng. Bằng việc quản lý các kịch bản kiểm tra, hệ thống mô phỏng có thể được thực thi tự động dựa vào tập kịch bản có sẵn. Việc này giúp hệ thống có thể được mô phỏng trên các hệ máy chủ có hiệu năng cao, thậm chí là phân chia các tập kịch bản kiểm tra khác nhau trên các hệ mô phỏng khác nhau.

Do có độ phức tạp cao và có nhiều các khối chức năng, nên thời gian để chạy mô phỏng của hệ thống mã hoá tín hiệu video kéo dài. Trong hệ thống của chúng tôi, thời gian mô phỏng cho một đoạn Video với 5 khung hình ảnh có thể kéo dài đến 2 giờ. Trong khi đó, các công cụ mô phỏng phần cứng hiện nay thường không hỗ trợ mô phỏng đa luồng (multi-thread). Mặc định, chương trình chạy mô phỏng sẽ chỉ sử dụng một lõi CPU trong khi các máy tính hiện đại đặc biệt là các hệ tính toán hiệu năng cao có thể có hàng trăm lõi xử lý. Do vậy, các công cụ này không tận dụng được sức mạnh của các hệ máy tính có nhiều CPU hoặc các CPU có nhiều lõi xử lý.

Các chương trình mô phỏng phần cứng thương mại đều hỗ trợ giao diện lập trình và điều khiển sử dụng ngôn ngữ thông dịch Tcl [1]. Các chương trình này được chạy bởi chương trình mô phỏng để giúp các nhà thiết kế biên dịch thiết kế, điều khiển chương trình mô phỏng, theo dõi quá trình mô phỏng và các tác vụ gỡ rối v.v... Tuy nhiên, các chương trình Tcl này mặc định cũng là các chương trình thực hiện tuần tự, không giải quyết được vấn đề mô phỏng đa luồng (multi-thread) và đa tiến trình (multi-process). Tóm lại, các chương trình Tcl này không tận dụng được khả năng tính toán của các hệ thống có nhiều lõi xử lý và không rút ngắn được thời gian biên dịch cũng như mô phỏng thiết kế.

Trong báo cáo này, chúng tôi đề xuất một phương án xây dựng hệ thống kiểm tra thiết kế linh động sử dụng các tính năng hỗ trợ của công cụ mô phỏng, của ngôn ngôn ngữ VHDL/Verilog và công cụ Make, áp dụng vào hệ mã hoá tín hiệu video H.264/AVC VENGME [2] do chúng tôi thiết kế để giảm thiểu thời gian biên dịch thiết kế, quản lý các cấu hình chạy mô phỏng và quản lý các tiến trình chạy mô phỏng. Hệ thống có thể được cấu hình với các tham số khác nhau để chạy mô phỏng bằng cách thay đổi các tham số (biến môi trường) trong môi trường chạy thiết kế. Hệ thống mô phỏng hỗ trợ hai công cụ mô phỏng thông dụng hiện nay là QuestaSim của hãng Mentor Graphics và VCS của hãng Synopsys. Số lượng các tiến trình mô phỏng tại một thời điểm và thời gian thực hiện mô phỏng với các kịch bản kiểm tra phụ thuộc vào số lượng lõi vi xử lý có trong một máy chạy mô phỏng và số lượng máy được sử dụng để chạy mô phỏng.

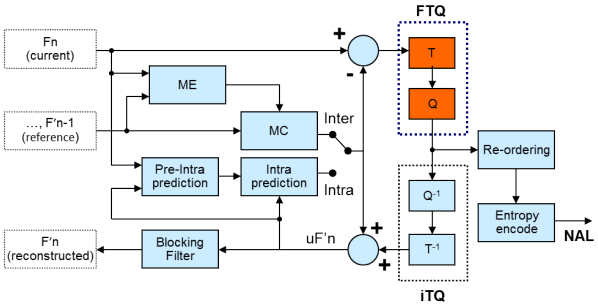
Các phần còn lại của báo cáo được sắp xếp như sau. Phần II giới thiệu về chuẩn mã hoá tín hiệu video H.264/AVC và hệ thống mã hoá tín hiệu video H.264/AVC VENGME do chúng tôi thiết kế. Phần III trình bày thiết kế kịch bản kiểm tra và kiểm chứng của hệ thống mã hoá VENGME. Phần IV là các công cụ hỗ trợ biên dịch, quản lý kịch bản chạy mô phỏng, và kết quả đạt được sau khi sử dụng các công cụ hỗ trợ này cùng với kịch bản kiểm tra do chúng tôi đề xuất. Cuối cùng là một số kết luận được trình bày trong phần V.

II. Chuẩn mã hoá tín hiệu video H.264/AVC và bộ mã hoá tín hiệu video VENGME

A. Giới thiệu chung về chuẩn H.264/AVC

Chuẩn mã hoá video H.264/AVC [3] được công bố lần đầu tiên vào năm 2003. Đây là một trong những chuẩn nén tín hiệu video hiệu quả nhất hiện nay (ngoại trừ chuẩn H.265/HEVC vừa được công bố vào tháng 3/2013), cung cấp video với chất lượng cao hơn với tốc độ bit (bit-rate) thấp hơn các chuẩn nén video trước đây. Chuẩn mã hoá video H.264/AVC được phát triển bởi nhóm chuyên gia hỗn hợp JVT thông qua hợp tác giữa ITU-T VCEG[1] và ISO/IEC MPEG[2]. Chuẩn này được trang bị một tập các công cụ mã hoá có khả năng hỗ trợ cho nhiều ứng dụng khác nhau, từ các dịch vụ di động và hội nghị truyền hình, truyền hình số... cho tới các ứng dụng truyền hình độ phân giải cao, truyền hình IP và các thiết bị lưu trữ số. Nếu so sánh với các chuẩn mã hoá video trước thì chuẩn H.264/AVC có thể giảm được lượng tốc độ bit đáng kể [4], 39% so với MPEG-4 [5], 49% so với H.263 [6], và 64% so với MPEG-2 [7].

Để có được tỷ lệ nén cao như vậy, chuẩn H.264/AVC được trang bị một loạt các kỹ thuật mã hoá tiên tiến nhằm loại bỏ các dư thừa về không gian và thời gian. Tuy nhiên, việc trang bị nhiều công cụ mã hoá tiên tiến này cũng làm cho hệ thống mã hoá theo chuẩn này trở nên rất phức tạp, đặc biệt là khi chúng ta thực hiện cứng hoá hoàn toàn bộ mã hoá/giải mã. Hình 1 minh hoạ bộ mã hoá H.264/AVC với các khối chức năng chính.



Hình 1. Sơ đồ khối chức năng của bộ mã hoá video theo chuẩn H264/AVC.

Theo Khuyến nghị H.264/AVC [3], các định dạng (format) hay cú pháp (syntax) cho tín hiệu video bị nén và phương pháp để giải mã cú pháp này để tạo ra một chuỗi tín hiệu video có thể hiển thị được đều được định nghĩa một cách rõ ràng. Tuy nhiên, trong các tài liệu của chuẩn không hề đề cập đến cách mã hoá (nén) tín hiệu video số (digital video). Phần nén tín hiệu video được dành cho các nhà sản xuất bộ mã hoá tín hiệu video.

B. Bộ mã hoá tín hiệu video VENGME

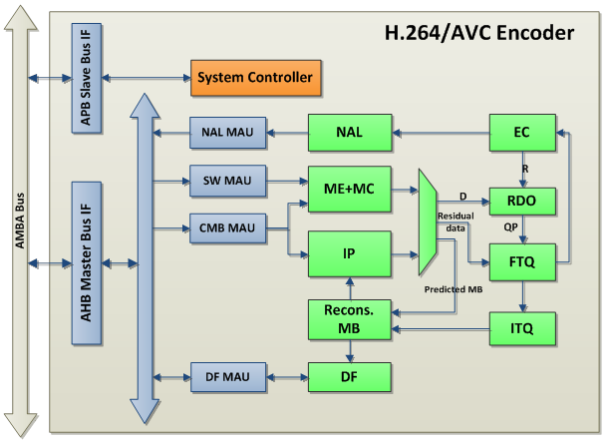
Mục tiêu của bộ mã hoá tín hiệu video VENGME là hướng tới các thiết bị di động hỗ trợ Main Profile của chuẩn mã hoá tín hiệu video H.264/AVC. Với Main profile - Level 2, chip mã hoá sẽ thực hiện có thể đáp ứng độ phân giải yêu cầu của hội nghị truyền hình với loại hình ảnh CIF và tốc độ khung là $30fps$. Sau đây là một số tính năng của chip mã hoá được xác định với lựa chọn trên:

- Tính năng phỏng đoán liên ảnh (Inter Prediction)
- Tính năng phỏng đoán nội ảnh (Intra Prediction)
- Hỗ trợ kỹ thuật B slice
- Kích thước các khối ảnh có thể thay đổi (variable block size)
- Tín năng bù chuyển động Quarter–Pixel
- Hỗ trợ kỹ thuật mã hoá entropy CAVLC
- Tính năng dự đoán có trọng số (weighted prediction)
- Bộ lọc giải khối lặp (In-Loop deblocking filter)

[1]The ITU-T Video Coding Experts Group (VCEG)
[2]ISO/IEC Moving Picture Experts Group (MPEG)

- Định dạng màu (4:2:0)
- Tính năng định dạng dòng bit NAL (NAL bitstream formating)

Kiến trúc thiết kế cụ thể của bộ mã hoá tín hiệu video H.264/AVC VENGME được trình bày trong Hình 2.



Hình 2. Kiến trúc đề xuất cụ thể của bộ mã hoá video H.264/AVC.

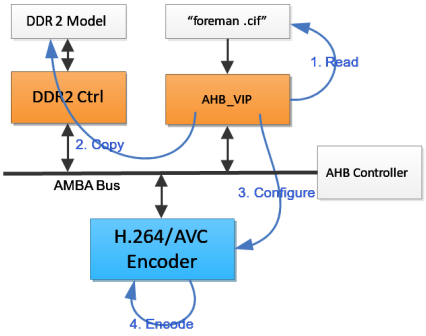
Với kiến trúc phần cứng này, cả bộ mã hoá tín hiệu video H.264/AVC sẽ được kết nối với hệ thống thông qua AMBA bus. Để có thể giao tiếp với AMBA bus, bộ mã hoá tín hiệu video H.264/AVC được trang bị hai giao diện: Giao diện bus chủ AHB (AHB Master Bus IF) và Giao diện bus tớ APB (APB Slave Bus IF). Việc cấu hình các thông số cho bộ mã hoá tín hiệu video H.264/AVC được thực hiện thông qua giao diện bus tớ APB trong khi đó các dữ liệu hình ảnh được trao đổi thông qua giao diện bus chủ AHB. Sau khi được cấu hình, bộ điều khiển hệ thống sẽ thực hiện điều khiển các quá trình xử lý mã hoá của hệ bộ mã hoá video H.264/AVC.

Vì dữ liệu nguồn, dữ liệu tham chiếu và dữ liệu mã hoá đều được lưu trữ trên bộ nhớ ngoài nên các khối chức năng của bộ mã hoá video H.264/AVC cần phải giao tiếp với hệ thống bên ngoài thông qua giao diện chủ bus AHB. Tuy nhiên, các khối chức năng này không thể đồng loạt trao đổi với AMBA bus mà phải trao đổi với giao tiếp chủ bus AHB thông qua một bus nội để có thể quản lý một cách hiệu quả các quá trình trao đổi dữ liệu. Việc các khối chức năng này thực hiện trao đổi dữ liệu thông qua bus nội được triển khai thông qua các khối giao tiếp đặc biệt gọi là đơn vị truy cập bộ nhớ (MAU: Memory Access Unit). Với thiết kế này, bộ mã hoá VENGME có các đơn vị truy cập bộ nhớ: CMB MAU (đơn vị truy cập bộ nhớ đọc dữ liệu MB hiện tại), SW MAU (đơn vị truy cập bộ nhớ cấp dữ liệu cho cửa sổ tìm kiếm), DF MAU (đơn vị truy cập bộ nhớ của bộ lọc giải khối), và NAL MAU (đơn vị truy cập bộ nhớ để ghi dữ liệu mã hoá ra bộ nhớ ngoài).

III. Xây dựng kịch bản kiểm tra và kiểm chứng thiết kế

Với thiết kế của bộ mã hoá VENGME đã được trình bày trong phần II-B, môi trường kiểm tra và kiểm chứng chức năng của bộ mã hoá H.264/AVC cần các khối chức năng sau:

- Lõi mã hoá tín hiệu video H.264/AVC được ghép nối với hệ thống bus AMBA AHB/APB 2.0.
- Lõi CPU hoặc bất kỳ một lõi IP nào có khả năng lập trình truy cập tới các thanh ghi bên trong bộ mã hoá.
- Lõi điều khiển ngắt để xử lý các ngắt của hệ thống trong đó có ngắt từ bộ mã hoá.
- Mô-đun bộ nhớ ngoài. Bộ mã hoá sẽ sử dụng bộ nhớ này để lưu các ảnh dự đoán cũng như video gốc trong quá trình mã hoá.



Hình 3. Kiến trúc của hệ thống kiểm tra được đề xuất.

Hình 3 miêu tả hệ thống đề xuất để kiểm tra và kiểm chứng lõi mã hoá H.264/AVC VENGME. Mô hình được xây dựng dựa trên các khối có sẵn trong thư viện GRLIB [8] nhưng được tích hợp thêm lõi mã hoá H.264/AVC và lõi làm nhiệu vụ kiểm chứng hay còn gọi là Verification IP. Hệ thống này bao gồm:

- Bộ mã hoá H.264/AVC đã được thiết kế: Thiết kế cần kiểm chứng (Design Under Test – DUT)
- DDR2 Memory model: mô hình DDR2 SDRAM
- AHB_VIP: có chức năng đọc nội dung video cần mã hoá từ một file video (ví dụ foreman_qcif.yuv), sau đó copy nội dung này vào DDR2 SDRAM theo từng khung hình để chuẩn bị cho quá trình mã hoá; cài đặt các tham số và

kích hoạt quá trình hoạt động của bộ mã hoá H.264/AVC.

Mô hình AHB_VIP được mô hình hoá sử dụng ngôn ngữ VHDL, được tham số hoá sử dụng các tham số dưới dạng *generic* để có thể thay đổi kịch bản kiểm tra mà không cần phải biên dịch lại thiết kế. Các tham số này sẽ được thay đổi tại thời điểm bắt đầu chạy mô phỏng. Định dạng của kịch bản kiểm tra sẽ được trình bày trong phần III-B. Ngoài ra, bộ mã hoá cũng được xây dựng với một số tham số khác tương tự như cách mô hình hoá AHB_VIP để hỗ trợ việc kiểm tra từng chức năng, từ chế độ mã hoá, và trích xuất dữ liệu phục vụ mục đích gỡ rối (debug) quá trình mã hoá.

A. Mô hình AHB_VIP

Bộ mã hoá tín hiệu video không đọc trực tiếp dữ liệu video gốc. Việc này được thực hiện bởi khối kiểm chứng thiết kế AHB_VIP. Khối này có thể đóng vai trò như một bộ xử lý để thực hiện việc sao chép dữ liệu video gốc lên trên bộ nhớ, thiết lập các chế độ hoạt động cho bộ mã hoá, trước khi kích hoạt bộ mã hoá. Khối này cũng đảm nhiệm việc xử lý ngắt khi bộ mã hoá đã hoàn thành mã hoá một chuỗi video.

Các bước điều khiển bộ mã hoá H.264/AVC VENGME để thực hiện mã hoá một tập tin video được lập trình trong AHB_VIP như sau:

1) AHB_VIP đọc các khung hình ảnh từ video gốc vào bộ nhớ DDR RAM. Số lượng khung hình ảnh được đọc vào bộ nhớ có thể cấu hình qua tham số của kịch bản kiểm tra để tiết kiệm bộ nhớ.
2) AHB_VIP ghi các tham số cho quá trình mã hoá (kích thước khung hình, địa chỉ lưu giữ các khung hình đã được tái tạo, kết quả sau khi mã hoá, hệ số lượng tử hoá v.v...) vào các thanh ghi của bộ mã hoá H.264/AVC.
3) AHB_VIP kích hoạt quá trình mã hoá bằng cách đặt bit 'enable' trong thanh ghi điều khiển lên '1'.
4) AHB_VIP chờ ngắt từ bộ mã hoá H.264/AVC để thông báo kết thúc mã hoá một khung hình hoặc một chuỗi khung hình.
5) Sau khi quá trình mã hoá kết thúc, AHB_VIP đọc dữ liệu hình ảnh đã được mã hoá, và viết ra tập tin văn bản.

Sau khi quá trình mã hoá hoàn tất, các tập tin được trích xuất ra ngoài trong quá trình mô phỏng sẽ được phân tích để đánh giá hiệu quả mã hoá và kiểm tra lỗi. Tập tin dữ liệu đã được mã hoá ở đầu ra sẽ được giải mã bằng phần mềm tham chiếu JM [9] của chuẩn mã hoá tín hiệu video H.264/AVC để chứng minh dữ liệu đã được mã hoá tuân theo chuẩn nén tín hiệu Video H.264/AVC. AHB_VIP cũng có thể dừng chương trình mô phỏng trong trường hợp thời gian mô phỏng vượt quá một ngưỡng cho phép được đặt bởi người thiết kế. Việc này sẽ giúp giảm thời gian mô phỏng và nhanh chóng xác định nguyên nhân gây lỗi và tiết kiệm thời gian mô phỏng.

B. Kịch bản kiểm tra

Kịch bản kiểm tra là một tập tin văn bản với các tham số của các tín hiệu video đầu vào và các tham số của quá trình mã hoá. Dựa vào các kịch bản kiểm tra này, chương trình Make [10] sẽ sinh ra các cấu hình tương ứng để thay đổi các tham số của AHB_VIP tại thời điểm gọi chương trình mô phỏng. Ứng với mỗi kịch bản kiểm tra sẽ có một tập tin đầu ra. Tập tin lưu kịch bản kiểm tra có dạng *<tên-video-nguồn>_<video-format>*.cfg ví dụ: *forman_qcif.cfg*. Một số cấu hình quan trọng của một kịch bản kiểm tra điển hình có dạng như sau:

```
CFG_VIP_FILENAME=foreman_qcif.yuv
CFG_VIP_FM_W=176
CFG_VIP_FM_H=144
CFG_VIP_INIT_QP=6
CFG_VIP_GOP_M=4
CFG_VIP_GOP_N=2
CFG_VIP_FMENC=5
CFG_VIP_IRQ_EN=1
CFG_VIP_CMB_NFM=2
```

- `CFG_VIP_FILENAME` là đường dẫn đến tập tin video gốc.
- `CFG_VIP_FM_W` × `CFG_VIP_FM_H` là độ phân giải của video gốc.
- `CFG_VIP_INIT_QP` là tham số lượng tử hoá trong chuẩn H.264/AVC (giá trị từ 0 đến 54).
- Hai tham số `CFG_VIP_GOP_M` và `CFG_VIP_GOP_N` là hai tham số quy định định dạng của một nhóm hình ảnh (Group of Pictures).
- `CFG_VIP_FMENC` là số khung hình ảnh sẽ được mã hoá. Tham số này phải nhỏ hơn hoặc bằng số khung hình ảnh thực có trong tập tin video gốc.
- `CFG_VIP_IRQ_EN` cho phép kích hoạt ngắt trong bộ mã hoá.
- `CFG_VIP_CMB_NFM` là số khung hình ảnh từ video gốc sẽ được nạp lên bộ nhớ. Sau khi mã hoá xong `CFG_VIP_CMB_NFM` khung hình ảnh, AHB_VIP sẽ nạp thêm số `CFG_VIP_CMB_NFM` khung hình mới vào bộ nhớ và kích hoạt bộ mã hoá tiếp tục mã hoá các khung mới cho đến khi số khung hình được mã hoá đạt đến số khung hình trong `CFG_VIP_FMENC`.

Bằng cách sử dụng kịch bản kiểm tra, các tập tin kết quả sẽ được sinh ra dựa trên các kịch bản kiểm tra. Kịch bản kiểm tra cũng có thể được sinh ra một cách tự động từ đó quá trình mã hoá có thể được tự động hoá đến mức tối đa. Một ưu điểm khác của kịch bản kiểm tra là có thể chạy song song nhiều kịch bản kiểm tra mà không sợ bị xung đột dữ liệu. Các tham số trong quá trình mô phỏng sẽ được thiết lập dựa vào các kịch bản kiểm tra.

C. Các cấu hình của bộ mã hoá

Bên cạnh kịch bản kiểm tra, các cấu hình khác của bộ mã hoá cũng có thể được cấu hình thông qua các biến môi trường. Mặc định, công cụ Make hỗ trợ nhận thông tin từ các biến môi trường. Dựa vào các biến môi trường này và công cụ Make, các kịch bản kiểm tra và các tham số khác có thể được tiếp nhận từ biến môi trường, và hệ thống có thể được kiểm tra với các chế độ khác nhau. Việc chạy mô phỏng ở mức hành xử hay mức cổng logic cũng được cấu hình qua các biến môi trường. Các biến môi trường này có thể được lưu vào tập tin để sử dụng lại trong các lần mô phỏng khác nhau. Sau đây là một ví dụ của việc đặt giá trị cho các biến môi trường trong môi trường bash shell:

```
export PROJECT_ROOT=../FRONT-END
export SIM_TRACE_LEVEL=10
export SIM_PARA_FILE=foreman_qcif.cfg
export COMPILER_OPTS=-quiet
```

Biến môi trường quan trọng nhất là `PROJECT_ROOT`. Biến này trỏ đến thư mục chứa mã nguồn của cả hệ thống. Sau khi biến này được đặt giá trị, hệ thống Makefile sẽ có thể được hoạt động. Hệ thống mô phỏng hoàn toàn tự động từ biên dịch thiết kế, đến chạy mô phỏng, và gỡ rối. Một số biến khác như `SIM_PARA_FILE` thông báo kịch bản sẽ được sử dụng để chạy mô phỏng. Biến này có thể được đặt nhiều kịch bản mô phỏng, phân cách nhau bằng khoảng trắng.

IV. Các công cụ mô phỏng, hỗ trợ mô phỏng và quản lý kịch bản mô phỏng và cấu hình của bộ mã hoá

Trong hệ thống kiểm tra và kiểm chứng thiết kế VENGME, chúng tôi sử dụng công cụ Make để quản lý các tập tin sinh ra trong quá trình biên dịch, chạy mô phỏng, và kiểm tra kết quả mã hoá. Bên cạnh công cụ Make, chúng tôi còn sử dụng một số ngôn ngữ thông dịch như bash/sh script và Python. Hai công cụ mô phỏng ngôn ngữ miêu tả phần cứng đang được sử dụng để chạy mô phỏng hệ thống VENGME là công cụ ModelSim [11] của hãng Mentor Graphics và VCS [12] của hãng Synopsys.

A. Hỗ trợ kiểm tra thiết kế bằng GNU Make

GNU Make[10] là một công cụ đã được sử dụng rộng rãi để hỗ trợ biên dịch các ngôn ngữ lập trình khác nhau. Ưu điểm của Make là chúng ta có thể viết các luật đơn giản để biên dịch các tập tin mã nguồn theo một thứ tự nhất định. Khi một tập tin thay đổi, thì các tập tin phụ thuộc vào nó cần được biên dịch lại. Việc này có thể rút ngắn thời gian biên dịch thiết kế do không phải biên dịch lại các tập tin không phụ thuộc vào các tập tin bị thay đổi. Tuy nhiên, Make không hỗ trợ tốt các ngôn ngữ miêu tả phần cứng. Hãng Mentor Graphics và Synopsys đều cung cấp các công cụ để sinh ra Makefile, tuy nhiên, các tập tin này hoàn toàn phụ thuộc vào các công cụ và khó thay đổi theo yêu cầu của người lập trình.

Để hỗ trợ nhiều công cụ mô phỏng, chúng tôi sử dụng Emacs và VHDL-mode [13] để sinh ra các tập tin Makefile cho ngôn ngữ VHDL. Tuy nhiên, VHDL-mode không hỗ trợ sinh Makefile cho nhiều thư viện. Do vậy, chúng tôi đã thay đổi VHDL-mode để sinh ra tập tin Makefile cho từng thư viện riêng lẻ có trong thiết kế. Sau đó, chúng tôi viết tập tin Makefile tổng hợp cho phép biên dịch tất cả các thư viện và các tập tin phụ thuộc. Hiện tại, Makefile do chúng tôi thiết kế có thể biên dịch hệ thống với cả hai công cụ, hỗ trợ chạy mô phỏng đến mức cổng logic (gate-level simulation) một cách tự động hoàn toàn dựa vào các biến môi trường và kịch bản kiểm tra. Makefile của chúng tôi cũng hỗ trợ biên dịch song song và chạy mô phỏng song song.

Để chạy mô phỏng đa tiến trình (multi-process) với kịch bản mô phỏng bất kỳ (thậm chí với kịch bản do người dùng tự viết không có trong hệ thống), luật Makefile đã được viết sử dụng các luật tự động (pattern rule). Ví dụ, để sinh ra tập tin <tên-kịch-bản>.h264.txt từ một kịch bản kiểm tra bất kỳ (tập tin <tên-kịch-bản>.cfg), chúng tôi dùng luật của GNU make như sau:

```
%.h264.txt : %.cfg
    $(eval SIM_RUN_FILE=$<)
    $(SIM_CMD) $(SIM_OPTS);
```

Ứng với mỗi kịch bản kiểm tra (tập tin <tên-kịch-bản>.cfg), sẽ được sinh ra một tập tin đầu ra tương ứng bằng các chạy công cụ mô phỏng với các tham số tương ứng với kịch bản kiểm tra. Để tạo ra tập tin nhị phân <tên-kịch-bản>.h264 và tập tin sau khi giải mã bằng JM, chúng tôi cũng sử dụng các luật tự động tương tự như trên:

```
%.h264 : %.h264.txt
    @python $(script_src)/h264.py -i $< -o $@
%.dec.yuv : %.h264
    @$(REF_DECODER) -i $< -o $@
```

Do các tập tin ở đầu ra của hệ thống là các tập tin văn bản, do vậy, chúng tôi đã viết một chương trình

Python [14] để chuyển tập tin văn bản này thành tập tin nhị phân trước khi đưa kết quả vào bộ giải mã chuẩn để kiểm tra.

Bằng cách sử dụng các luật tự động, chúng tôi có thể chạy mô phỏng với bất kỳ kịch bản kiểm tra nào đúng định dạng như đã trình bày ở phần III-B. Tùy vào yêu cầu của người thiết kế, người thiết kế có thể chọn chạy hệ thống với tất cả các kịch bản kiểm tra, hoặc chạy với một số kịch bản kiểm tra, chạy mô phỏng với nhiều tiến trình hoặc đơn tiến trình. Cú pháp để chạy mô phỏng với toàn bộ các kịch bản kiểm tra đã được định nghĩa sẵn sử dụng 4 tiến trình song song như sau:

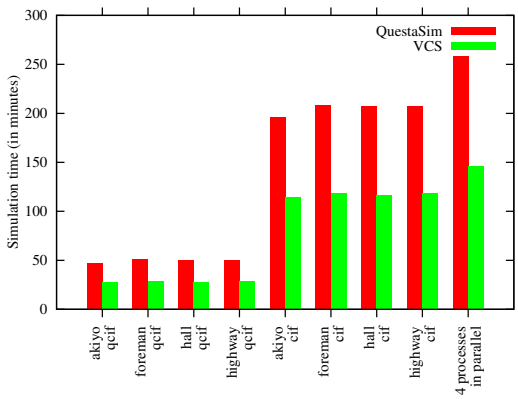
```
$ make regression -j4
```

hoặc chạy với một số kịch bản kiểm tra do người dùng định nghĩa với 2 tiến trình chạy song song:

```
$ export SIM_PARA_FILE="f_qcif.cfg a_qcif.cfg"
$ make run -j2
```

B. Kết quả đạt được

Hình 4 là thời gian chạy mô phỏng của từng kịch bản kiểm tra riêng lẻ, và tổng thời gian chạy mô phỏng trong trường hợp chạy song song 4 tiến trình sử dụng hệ thống mô phỏng và kiểm chứng được đề xuất trên hai chương trình mô phỏng phổ biến là Questasim và VCS. Tổng thời gian mô phỏng cho cả 8 video trong trường hợp sử dụng bốn tiến trình mô phỏng chạy song song tại một thời điểm sử dụng chương trình QuestaSim là 258 phút. Trong khi đó, nếu chúng ta chạy lần lượt từng kịch bản, tổng thời gian chạy mô phỏng của các kịch bản này là 1017 phút. Như vậy, nếu chạy song song bốn kịch bản tại một thời điểm, hệ thống này có thể giảm thời gian mô phỏng xuống xấp xỉ 4 lần. Với tám kịch bản kiểm tra



Hình 4. Thời gian chạy mô phỏng của hệ thống với một số kịch bản kiểm tra bằng chương trình QuestaSim và VCS.

này, hệ thống cũng đạt được tổng độ bao phủ (coverage) của 8 kịch bản kiểm tra khoảng 95%. Dựa vào kết quả giải mã của chương trình JM và khả năng tự kiểm tra lỗi của AHB_VIP, các lỗi có thể được phát hiện sớm, đảm bảo các thay đổi trong quá trình phát triển không ảnh hưởng đến chức năng và chất lượng của bộ mã hoá.

V. KẾT LUẬN

Mô phỏng, kiểm tra và kiểm chứng thiết kế là những vấn đề quan trọng có ảnh hưởng đến chất lượng của các thiết kế phần cứng. Bằng việc áp dụng các công cụ phần mềm mã nguồn mở, kết hợp với tính năng của các công cụ mô phỏng và kiểm chứng, hệ thống kiểm tra và kiểm chứng cho bộ mã hoá tín hiệu video H.264/AVC VENGME đã được xây dựng một cách linh động. Hệ thống có thể rút ngắn thời gian biên dịch thiết kế, rút ngắn thời gian mô phỏng bằng cách sử dụng các tiến trình chạy song song. Hệ thống này cũng giúp cho người lập trình không cần phải quan tâm đến các chi tiết về quá trình biên dịch, mô phỏng và kiểm chứng. Nó cho phép người thiết kế có thể chạy mô phỏng trên các hệ tính toán hiệu năng cao, và tận dụng được tối đa tài nguyên của hệ thống. Với tám kịch bản kiểm tra và chạy song song 4 kịch bản tại một thời điểm, hệ thống mô phỏng và kiểm chứng đề xuất đã rút ngắn thời gian chạy mô phỏng xuống 4 lần so với chạy lần lượt từng kịch bản với độ bao phủ đạt 95%.

TÀI LIỆU THAM KHẢO

[1] "Tool command language (tcl)," http://www.iis.ee.ethz.ch/~zimmi/emacs/vhdl-mode.html.

[2] Ngoc-Mai Nguyen, Edith Beigne, Suzanne Lesecq, Duy-Hieu Bui, Nam-Khanh Dang, and Xuan-Tu Tran, "H.264/AVC hardware encoders and low-power features," in *Proceedings of the 2014 IEEE Asia Pacific Conference on Circuits and Systems (IEEE APCCAS 2014)*, Okinawa, Japan, 2014.

[3] "ITU-T Recommendation and International Standard of Joint Video Specification," March 2005.

[4] A. Joch, F. Kossentini, H. Schwarz, T. Wiegand, and G. J. Sullivan, "Performance comparison of video coding standards using lagragian coder control," in *IEEE International Conference on Image Processing (ICIP 02)*, 2002, pp. 501–504.

[5] "Information Technology—Coding of Audio-Visual Objects—Part 2: Visual," 1999.

[6] "Video Coding for Low Bit Rate Communication," 1998.

[7] "Information Technology—Generic Coding of Moving Pictures and Associated Audio Information: Video," 1996.

[8] Aeroflex Gaisler, "Grlib ip library," http://www.gaisler.com/index.php/products/ipcores/soclibrary.

[9] Fraunhofer Institute, "H.264/AVC reference software," http://iphome.hhi.de/suehring/tml/.

[10] GNU Software Foundation, "GNU Make," http://www.gnu.org/software/make/.

[11] Mentor Graphics, "Modelsim," http://www.mentor.com/products/fpga/simulation/modelsim.

[12] Synopsys, "VCS," http://www.gnu.org/software/make/.

[13] Reto Zimmermann and Rod Whitby, "Emacs vhdl mode," http://www.iis.ee.ethz.ch/~zimmi/emacs/vhdl-mode.html.

[14] Python Software Foundation, "Python programming language," http://www.python.org/.