玄鐵開放原始碼專案的模擬工具鏈的改進與實驗分析– Verilator Simulation

Taurus
復旦大學微電子學院研究生

近期，平頭哥的 4 款玄鐵處理器在 GitHub 上開源，不僅引起了較大的關注，也為 RISC-V 開源社區的發展提供了一個不錯的平台。玄鐵開放原始碼專案中最先支援了三款模擬器—Iverilog、VCS 和 NC。而本文描述了如何在玄鐵開放原始碼專案中支援另一款高性能開源模擬器 Verilator，並測試比較了其與 Iverilog 和某商用 simulator 的模擬速度。

注 ： 本文介紹的 Verilator 模擬指令碼已經合併到 openc910 的 main 分支中 https://link.zhihu.com/?target=https%3A//github.com/T-head-Semi/openc910, **讀者可以直接 clone 下來嘗試**
（Verilator 版本最好為 4.215 以上，最新版的 Verilator 可以在 Github 上獲取 https://link.zhihu.com/?target=https%3A//github.com/verilator/verilator)

一、模擬器簡介

1. VCS

VCS 是 Synopsys 公司提供的一款商用等級的 HDL 模擬器，支援 Verilog、VHDL、System Verilog HDL 語言的模擬模擬，在行業內有較高的模擬性能。VCS 結合了節拍式演算法和事件驅動的演算法，適用從功能模擬到後端時序模擬的所有流程模擬，同時也提供了覆蓋率監測、斷言技術等驗證技術。

VCS 基本流程分為 analyze、elaborate 和 simulation。

1. analyze：此階段會分析檔案的語法並生成 elaborate 需要的中間檔案。
2. elaborate：使用上述產生的中間檔案，並將這些檔案層次關係 build 起來，最後生成二進制可執行檔案(默認命名為 simv)。
3. simulation：運行產生的可執行檔案，進行模擬。

2. Iverilog

Iverilog（Icarus Verilog）是一款輕量級的開源 Verilog 模擬器，同時據其官方介紹稱，Iverilog 還可以用作生成電路網表的綜合工具。Iverilog 的工作流程大致可以分為 compile，elaborate 和 simulation 三個流程。

- **compile**：Iverilog 主要完成每個 verilog 檔案的語法分析、參數分析、預處理一些如 define、include 命令以及解析檔案的 parser 結構，verilog 檔案之間還沒有建立聯絡。這個階段會輸出一種 pform 格式的資料。
- **elaborate**：將模組的結構給連接起來，包括例化模組、任務、函數、generate 的語句，還包括參數的傳播以及呼叫 initial 等操作。此階段會輸出 vvp 格式的類彙編程式碼。
- **simulation**：呼叫一種類似虛擬機器的 vvp 程序去模擬 elaborate 階段生成的檔案。

3. Verilator

Verilator 是一款高性能的 Verilog/System Verilog 開源模擬工具。運用 Verilator package，我們可以將 Verilog 和 System Verilog HDL 語言設計編譯轉換成 C++或者 SystemC 模型，所以從這個意義上來說，Verilator 更應該被成為是一個編譯器(Compiler)而不是一個傳統意義上的模擬器。

通常情況下，Verilator 的工作流程如下所示：

1. 首先 Verilator 將讀取特定的 HDL 檔案並檢查其程式碼，同時還可以選擇支援檢查覆蓋率和 debug 波形的生成。然後將原始檔編譯成原始碼級的多執行緒(source level multithreaded) C++或 SystemC 模型。其輸出的模型會以.cpp 和.h 檔案存在。這個階段的過程叫做"to Varilate"，輸出的檔案叫做"Verilated Model"。
2. 為了能夠完成模擬，Verilator 需要一個使用者自行編寫的 C++ wrapper，這個 wrapper 與傳統的 Verilog Testbench 功能類似，主要是為了連接頂層模組，並給予相應的激勵。
3. 在 C++ 編譯器的工作下，所有的之前生成的檔案（C++ wrapper 以及 Verilated Model）以及庫檔案（Verilator 提供的 runtime library 或者 SystemC 庫檔案）會被一同合併成一個可執行檔案。
4. 執行生成的可執行檔案，就可以開始實際的模擬，此時成為"simulation runtime"

5. 最後如果在編譯階段將特定的編譯選項打開，可執行檔案也會生成波形和覆蓋率等資訊。

二、玄鐵開源平台的模擬測試

平頭哥開發的玄鐵開源平台中提供了三種模擬指令碼，分別是商用級模擬器 VCS 和 NC，以及**開源模擬器** Icarus Verilog。經過初步的嘗試，我們發現由於 Icarus Verilog 是面向輕量級項目的輕量模擬器，其在模擬玄鐵 C910 這樣較為複雜龐大的處理器核心時有模擬速度慢的劣勢。而商用級模擬器 VCS 和 NC 雖然具有較快的模擬速度，但是由於需要 license 支援，不利於開放原始碼專案的發展。因此，我們選擇將同樣是**開源**的，但是具有**較高模擬性能**的 Verilator 的模擬指令碼支援到玄鐵項目中去。

1. Verilator 的模擬

根據上面簡述的步驟，我們首先要將相應的 HDL 檔案讀取到 Verilator 中進行編譯。而像玄鐵 C910 這種較為複雜的項目通常具有複雜的檔案結構，一般會將所需的 HDL 檔案寫成一個 filelist 檔案，這個 filelist 檔案存放在 `/openc910/smart_run/logical/filelists` 中，在裡面存放著四個 filelist：

- **ip.fl** ： 包含了玄鐵 c910 核心的檔案列表。
- **smart.fl** ： 包含了名叫 smart 的 SoC 平台的檔案列表。

- **tb.fl** ： 包含了 Verilog Testbench 的 tb 檔案。
- **sim.fl** ： 將上述三個檔案進行了合併，Verilator 可以單獨建立一個 sim_verilator.fl 的檔案。

有了上面的 filelist 檔案，我們就可以比較方便的讀取 HDL 原始檔了。但是由於 Verilator 還需要一個 C++ wrapper 來對 HDL 頂層檔案施加激勵，所以我們還需要編寫一個 C++的 testbench。由於 openc910 的 verilog testbench 還是具有一定的複雜度，方便起見我們可以在模擬階段將這個 testbench 當作設計檔案的頂層，這樣我們的 C++ testbench 就只需要施加時鐘的激勵就可以了（可以根據 Verilator 倉庫的 example/make_tracing_c/sim_main.cpp 進行改寫）。

當我們完成了 sim_main.cpp 的改寫後，還需要修改一下 verilog testbench 中的一些語法，這是因為我們將 tb 看成了設計檔案的頂層，那麼所有的延時語法(#10)都無法被支援，所以我們需要將除了我們 C++ testbench 會施加的激勵時鐘外的訊號都變成與 clk 有關的形式。修改方法可以參考下面的做法：

```
integer jclkCnt;
 initial
 begin
   jclk = 0;
   jclkCnt = 0;
   // the logic before
   //forever begin
   //  #(`TCLK_PERIOD/2) jclk = ~jclk;
```

```
    //end
  end
// the logic after modification
  always@(posedge clk) begin
    if(jclkCnt < `TCLK_PERIOD / `CLK_PERIOD / 2 - 1) begin
      jclkCnt = jclkCnt + 1;
    end
    else begin
      jclkCnt = 0;
      jclk = !jclk;
    end
  end
```

完成了上述的步驟後，我們就可以開始利用 Verilator 進行檔案的 compile 和 build 了。

1. **compile** ： `-cc` 表示我們的模型為 C++模型，`--exe` 表示要生成可執行檔案，`-Wno-fatal` 表示忽略除了 fatal 之外的 Warning，以讓編譯順利跑完，`--top-module` 用於指定頂層模組，這裡為我們的 tb，`-f` 表示將檔案中的內容包含進來。其他可選的選項有多執行緒模擬`--threads x`（x 為執行緒數）和 debug 波形選項`--trace`。最終運行的命令可以是

   ```
   $ verilator -Os -x-assign 0 --threads 16 -Wno-fatal -cc --exe --top-module top --trace -f sim_verilator.fl
   ```

2. **build** ： 可以通過 Verilator 官方給出的 `makefile_obj` 指令碼進行 aotu_build，具體命令為

   ```
   $ make -j -C obj_dir -f ../Makefile_obj
   ```

成功完成 build 後，我們可以在 obj_dir 檔案中找到一個 Vtop 的可執行檔案，我們再將要執行的程序 hex 檔案通過 openc910 官方指令碼中的 make buildcase CASE=XXX 生成到 work 目錄就可以開始模擬了！

2.支援 Verilator 的模擬指令碼

當我們成功跑通了這個流程後，我們可以將這些流程寫進 Makefile 指令碼中，這樣以後可以通過 make 命令進行快速的模擬。**支援上述流程的 Makefile 檔案已經合併到平頭哥玄鐵開放原始碼專案 openc910 中（Verilator 版本最好為 4.215 以上）**。

通過下述的流程就可以完成分步的編譯模擬，

```
$ meke cleanVerilator
$ make compile SIM=verilator
$ make buildVerilator
$ make buildcase CASE=xxx
$ make runVerilator
```

或者通過

```
$ make runcase CASE=xxx SIM=verilator
```

完成整個流程。

但我們**推薦**分步進行，因為通常 compile 和 build 需要花費較多時間，所以在沒有改變 RTL 程式碼的情況下可以直接通過 make buildcase CASE=xxx + make run Verilator 來進行快速的模擬。

3．不同模擬器的測試結果

我們對三種模擬器模擬 openc910 的效率進行了對比。

註：Verilator 的可執行檔案使用 Clang 在某些情況下可最佳化性能

測試的 CPU 為 Intel Xeon Gold 5218 2.3Ghz，測試程序為 hello_world，測試對象為 Iverilog、單執行緒運行的 Verilator 以及單執行緒運行的某商用模擬器，編譯選項沒有新增 debug 選項（不會 dump 波形）。

對象	Iverilog	Verilator	某商用模擬器
模擬時間	20 分鐘	10 秒	2.8 秒

測試的 CPU 為 Intel Xeon Gold 5218 2.3Ghz，測試程序為 1 次迭代的 coremark，測試對象為 Iverilog、單執行緒運行的 Verilator、多執行緒運行的 Verilator 以及單執行緒運行的某商用模擬器，編譯選項沒有新增 debug 選項（不會 dump 波形）。

對象	Iverilog	Verilator 單執行緒	Verilator 8 執行緒	Verilator 16 執行緒	某商用模擬器
模擬時間	6.5 小時	4 分鐘	2 分鐘	2 分鐘	2 分鐘

測試的 CPU 為 Intel Xeon Gold 5218 2.3Ghz，測試程序為 100 次迭代的 coremark，測試對象為 Iverilog、單執行緒運行的 Verilator、多執行緒運行的 Verilator 以及單執行緒運行的某商用模擬器，編譯選項沒有新增 debug 選項（不會 dump 波形）。

對象	Iverilog	Verilator 單執行緒	Verilator 8 執行緒	Verilator 16 執行緒	某商用模擬器
模擬時間	300+小時	4.5 小時	2 小時	50 分鐘	2.5 小時

測試的 CPU 為 Intel Xeon Gold 5218 2.3Ghz，測試程序為 100 次迭代的 coremark，測試對象為多執行緒運行的 Verilator 以及單執行緒運行的某商用模擬器，編譯選項為新增 debug 選項。

對象	Verilator 8 執行緒	Verilator 16 執行緒	某商用模擬器
模擬時間	10.5 小時	8.5 小時	6 小時

三、總結

通過測試我們可以發現，作為同樣是開源模擬器的 Verilator，其性能比輕量級的 Iverilog 高很多。對於較為複雜且龐大的 c910 核心或者 smart 平台來說，Verilator 會更適合來對設計進行模擬驗證。

雖然某商用模擬器在單執行緒的性能上會比 Verilator 優秀，但是某商用模擬器的多執行緒功能需要高級 license 才能使用，對開源社區並不友好，而

Verilator 線上程數較多時也能與某商用模擬器單執行緒性能相當甚至更高。同時 Verilator 更開放的使用權限也能讓開源社區更加活躍。

還有一個 dump 波形的問題，由於 Verilator 的 debug 波形選擇的是 vcd 格式，其體積較為龐大，而某商用模擬器對波形檔案做了特定的最佳化，所以在 dump 波形時 Verilator 會有較大的劣勢，而 Verilator 也支援 fst 的精簡波形檔案（具體打開方式可以參考 Verilator 手冊第 45 頁），採用這種方式會提升模擬速度，缺點就是只能用 gtkwave 打開波形查看了。當然，在驗證完核心硬體後，可以選擇將 `--trace` 功能關閉以加快模擬速度。

總的來說，Verilator 十分適合像玄鐵這種具有一定規模的開放原始碼專案使用，其開放的特性與高性能的特點十分契合開放原始碼的思想。

四、參考資料

[1]. verilator 手冊

[2]. 玄鐵 C910 倉庫