Verification Process for Digital IC Design

Hung-Chin Jang, Yao-Nan Lien, Shi-Kung Chou
Department of Computer Science
National Chengchi University
jang@cs.nccu.edu.tw

摘　要

本研究旨在為 IC 設計之驗證，設計一套有助於提高 IC 產品品質的軟體工程驗證模型 。過去在 IC 設計領域並沒有一套完整流程，明確規範每個驗證步驟該有的程序與方法。我們參考軟體開發模型中的現有架構並加以修改，設計出一套符合軟體工程需求的標準流程(SOP)，藉由該流程，IC 驗證能被標準化，並能明確掌握所需時程與人力，降低成本，提高產能。由於一個軟體設計流程需要歷經多年的使用改進方能達到成熟階段。本研究所發展的驗證流程與工具不僅能適用於 USB3.0 開發，並能適用於任何一種數位 IC 的驗證流程。

關鍵字：驗證模型、IC 設計

一、背景

對於一個已經依照需求完成設計的 IC，需要進入驗證階段，此時的元件稱為 Device Under Test (DUT)。驗證的最終目標是期望能得到功能正確，值得信賴的元件。通常為了找出邏輯上的錯誤，會使用許多不同的條件與輸入針對該元件進行測試，其中包含一些正常工作時不可能出現的錯誤狀況(error case)，以確定元件在錯誤的工作環境下不至於『鎖死』(lock up)。此外，也要確保元件能夠達到預期的工作效率，同時確認在罕見的輸入條件下，也要能正常工作(稱之為"corner cases")，如此測試才能確保元件的工作狀況完全依照規格書的要求。在進行數位 IC 設計時，最初提出的是此 IC 的構想與演算法，接著才逐步定義出此 IC 的規格要求(specification)。此時可由硬體描述語言以"行為描述"(behavioral description)的方式撰寫此 IC 功能。下一步則以暫存器轉移(Register Transfer Logic，RTL)的方式描述資料與訊號轉變的行為，接著再以邏輯閘(gate)描述邏輯電路。經驗證確定一切無誤後才能進行布局(layout)，實際下線(tape-out)做成硬體 IC。數位 IC 設計有四個主要設計階段，每個階段的需求與目標分別如下：

- Architectural/Algorithmic Level：以高階語言，演算法或功能描述的方式設計此 IC
- Register-Transfer Logic (RTL) Level：以暫存器中的資料流(data flow)的行徑描述此 IC 應該具備的功能與資料處理程序
- Gate Level：描述 IC 內邏輯閘(gate)及閘與閘之間的連線方式
- Transistor/Switch/Physical Level：設計最底層的電路，實際進行電晶體布局與連線

每個過程間，須經過合成(synthesis)，將硬體描述語言的敘述轉換為更底層，更細部的描述，其間需經過轉換(transformation)的過程。然而，一旦進行合成與轉換後，無法確保其電路是否仍然保持原本設計的特性。因此，即使在行為描述與 RTL 邏輯通過驗證的設計，再經過更進一步的合成為 Gate Level 與 Transistor Level 後，仍然需要逐一進行驗證。

針對一個設計完成的 DUT (Device Under Test)元件進行驗證時，首先須建立其測試環境(包括輸入參數設定介面，輸出結果介面與報告產生等)，接著可依照其涵蓋範圍分三階段進行測試：

- Preliminary verification：針對規格書中所描寫的特定工作流程，進行有規律的測試，以證明此 IC 在一切正常狀況下能正常工作。此部份為最基本也是最簡單的測試，但即使通過此部份測試，此 DUT 仍不能算是正常，因為使用者實際使用 IC 時未必依照規格書規定的資料或訊號輸入。此時該元件是否能保持正常工作，即使遇到錯誤也能正確做出必要回應，而不至於『鎖死』，在此階段均無從得知。
- Broad-spectrum verification：以各種可能的訊號組合方式，產生 test pattern 作為輸入資料，觀察此 DUT 的工作情形。此部份的測試對於驗證完成度而言，涵蓋範圍相當大，由於驗證者無法確切得知使用者會以何種資料輸入要求此元件工作，因此只能以排列組合方式找出各種可能的輸入組合。因為所有可能的組合數量可能是天文數字(指數函數)，因此難以測試到所有可能狀況。
- Corner-case verification：在驗證時，以非常極端的參數值或非常罕見的情境(scenario)作為輸入，驗證此 IC 遇到極為罕見的錯誤時，是否能正常反應而不至於鎖死。在實際使用時，此狀

況極少發生，因此該部份測試的完成度涵蓋率並不高。即使不進行驗證，大部分的設計成品仍然可正常工作，但穩定性卻大打折扣。即使進行此部份測試，仍然可能疏漏一些極為少見的狀況。

國內大部分 IC 設計廠商為搶奪 time-to-market 先機，大多僅進行 preliminary verification，以致產品在異常操作的情況下其可靠度及穩定度均有所不足。而限於硬體工程師對軟體工程的陌生，大部分國內廠商均未能對 broad-spectrum verification 及 corner-case verification 發展出成熟的 IC 驗證模型。本研究期望能在 broad-spectrum verification 與 corner-case verification 方面發展出一套合適的軟體工程程序，藉由開發一套完整流程，由 test plan 的分析開始，驗證此測試階段所需要的 direct test pattern 與 random test pattern 是否符合需求，並分析其對 IC 驗證的涵蓋率。我們係針對目前 IC 驗證缺乏標準化程序的缺陷加以補足，詳細規範從文件分析至驗證步驟實行的時間與流程，讓未來所有 IC 驗證工作都可套用這套流程，減少開發時間與人力。

二、驗證流程設計

軟體開發需要謹慎且完整的規劃，缺乏完善的規劃則開發出來的產品是不被信賴的。在本研究中，我們建議以適合中小規模的敏捷模型(Agile Model)為軟體開發模型，並配合我們所開發的『狀態機模擬器產生器』(Generator of FSM Simulator，FSMGEN)，使 IC 設計廠商可依據我們所開發的 test bench 產生器，產生 test case 來驗證廠商的產品。軟體開發絕非一蹴可幾，需要經過不斷的討論和風險分析，適時修正，才能開發出好的軟體。我們設計一套數位 IC 設計開發流程，並在驗證過程中加入 FSMGEN，再以 USB 3.0 產品做為目標來驗證我們設計的開發流程是否符合數位 IC 設計廠商的需求。

2.1 Agile Model (敏捷模型)簡介

我們使用 Agile Model 作為軟體工程開發模型。Agile 是敏捷的意思，強調有別於以往龐大耗時的傳統軟工模式。在一般台灣小規模公司裡開發人力原本就不多，常常需趕上市時間(time to market)，往往跟客戶談好需求後，幾乎沒寫太多文件即立刻開始撰寫程式。一面和客戶即時調整需求，一面釋出程式的新版本，同時又撰寫測試碼來保證程式的品質，如此不斷地重複以便讓客戶得到他們心中真正想要的軟體。遵循 Agile Model 可及時檢查出在規格或是設計上的錯誤，由於開發速度較快，團隊合作密切，做事效率可大量提升。Agile Model 的開發模型有下列幾項特點：

- 能達成目標(fulfill the purpose)
- 容易理解(understandable)
- 足夠精確(sufficiently accurate)
- 足夠一致(sufficiently consistent)
- 足夠詳細(sufficiently detailed)
- 盡可能簡單(as simple as possible)

在建立好需求模組及系統架構後，便能實現系統的內部架構以進行驗證測試。在第一次的修改與檢討後，再進行下次的程式撰寫與驗證測試，不斷重複這些步驟，直到系統開發完成。

2.2 數位 IC 設計開發流程

2.2.1 需求

數位 IC 產品的開發，需要先了解數位 IC 設計開發過程中的需求。與軟體開發流程相同，數位 IC 設計的開發步驟是需求分析(analysis)、需求擷取和分析(elicitation and analysis)、需求規格(requirement specification)、需求確認(validation)、需求管理(management)，這些步驟都需要反覆討論、分析才可寫出一個好的需求文件。

(a) 需求分析可以決定提出的系統是否有價值。這是一項短暫的流程，著重檢查下列各點：

- 此數位 IC 產品對公司的整體目標是否有貢獻？
- 此數位 IC 產品是否可以利用目前的技術，在有限的成本(budget)以及時程限制(deadline)下完成？
- 此數位 IC 產品是否可和其他現有數位 IC 產品整合？
- 此數位 IC 產品的資訊規格(specification)是否完整，可否開發出來？

實作可行性分析需求評估資訊、蒐集資訊以及撰寫報告。因此，我們需要詢問組織中的人員下列問題：

- 若缺少該數位 IC 產品，會有什麼問題？
- 目前有哪些程序(process)問題？
- 所提出的數位 IC 產品會有何幫助？
- 有哪些整合上的問題？
- 是否需要新的技術？若是，會是哪些技術？
- 所提出的數位 IC 產品需支援哪些功能(facilities)？

接著根據這些調查以決定此數位 IC 產品是否符合要求、所設計出的數位 IC 產品能否產生出最大的效益。

(b) 需求擷取和分析是讓專案關係人(stakeholders)間一起討論相關的應用領域(application domain)、應提供的服務及系統的操作限制(operational constraint)等。專案關係人包括終端使用者(end-users)、經理人員(managers)、負責維護的工程師(engineers)、領域專家(domain experts)或是工會(trade unions)等。經過專案關係人之間的討論後，需要分析和協商(analysis and negotiation)。分析和協商可用來分析及理解不同客戶間不同需求的關係：

- 專案關係人(stakeholders)並不真正知道他們所要的是什麼？而且專案關係人會以自己的辭彙來表示需求。不同的專案關係人可能會提出互相衝突的需求(conflicting requirements)。
- 公司組織和策略的因素可能也會影響系統的需求。
- 分析過程中若有需求遭變更可能會出現新的專案關係人以及造成商業環境的改變。

最後我們理出這些需求之間的關係以便獲取成功的結果。

(c) 需求規格是從需求中建立有實質的模型，是將各種需求建立一個模型並定義其規格及內容。此模型(system modeling)不但需要可被檢驗(assessed)其正確性(correctness)，也要使用完全性(completeness)和一致性(consistency)的需求表示方式。

(d) 需求確認是確定需求是否正確定義顧客真正想要的系統。若不實行此步驟，之後若因需求錯誤而造成的錯誤成本(error cost)將會非常可觀。在系統交付之後修正需求錯誤(requirements error)的成本可能會比修復實作的錯誤(implementation error)高出 100 倍之多。因此要有需求檢驗(requirements checking)以確認以下幾點要求：

- 有效性(validity)：數位 IC 產品提供的功能是否能支援客戶的最大需求(customer's needs)？
- 一致性(consistency)：是否有任何需求彼此衝突(conflicts)？
- 完整性(completeness)：是否包含客戶所需要的所有功能？
- 可實現性(realism)：需求是否可在可用的預算和技術下實作(implemented)完成？
- 可驗證性(verifiability)：需求是否可以被檢驗？

擬定需求規格時必須定期召開會議進行審查，其中客戶和承包商雙方人員都應參與審查。這些審查可以是正式的(有完整文件)或非正式的。開發者、客戶和使用者之間若有良好的溝通便可及早解決問題。會議中，需確認以下特性：

- 驗證性(verifiability)：需求是否真能進行測試(realistically testable)？
- 理解性(comprehensibility)：需求是否能夠正確的被理解(properly understood)？
- 追溯性(traceability)：需求來源(origin)是否清楚的記錄？
- 調適性(adaptability)：需求是否可因環境而適度調整，並不會嚴重影響其它系統需求？

(e) 需求管理(requirements management)是在需求工程程序及系統開發期間管理需求變更的程序。因為需求通常是不完整且不一致的，在程序期間當商業需求改變或是對正在開發的系統有更多的瞭解，就會出現新的需求。且不同觀點會有不同的需求，而這些需求常會相互矛盾。所以我們需要確認(identify)、控制(control)及追蹤(track)系統的需求及其變化。因此在需求工程期間我們必須要有下列規劃：

- 需求確認(requirements identification)：如何逐一確認每項個別需求(requirements are individually identified)
- 變動管理程序(change management process)：分析需求變更(requirements change)後產生的程序
- 追溯性策略(traceability policies)：指關於所維護之需求間關係(requirements relationships)的資訊，是指需求、來源及系統設計(system design)間的關係
- CASE 工具支援(CASE tool support)：有助於管理需求變更時所需的支援工具

在此我們以 USB 3.0 的設計開發為例。廠商首先針對需求做分析。分析 USB 3.0 對公司的整體目標是否有貢獻、是否可以利用目前的技術並在有限的成本以及時程限制下完成、是否可以和其他現有數位 IC 產品作整合、USB 3.0 的資訊規格(specification)是否完整，可否開發出來？為此須詢問公司人員若缺少 USB 3.0 設計開發案，會有什麼問題？目前會有哪些程序問題？會有何幫助？有哪些整合上的問題？是否需要新的技術？若是，會是哪些技術？需支援哪些功能？接著進行協商與 USB 3.0 有關的專案關係人相互討論，確保其需求

不會互相衝突，並分析公司對 USB 3.0 的政策及其過程若因需求更改所造成的風險。然後根據需求分析，定出 USB 3.0 的規格。可能會因不同廠商而有不同的工程規格(Engineering SPEC)。必須在之後驗證過程中分析廠商制定的 Engineering SPEC 是否與 USB 3.0 formal SPEC 相同。確認其所制定的 USB 3.0 文件與規格是否符合公司政策與市場需求。最後規劃 USB 3.0 的需求管理，以確保針對 USB 3.0 的數位 IC 設計開發需求是完善的。

2.2.2 規格

已確定的需求應有清晰且精確的描述。一般我們把描述需求的文件稱做規格文件或軟體需求規格書。為確切表達使用者對軟體的輸入與輸出要求，需編寫初步使用者手冊，反映使用者的具體要求。在整體開發過程中，文件扮演的角色非常重要。文件好比整體開發過程制度之建立，沒有文件，軟體系統開發常會事倍功半。當軟體需求階段完成時，開發者須將需求階段的成果寫成規格文件，讓使用者過目後雙方簽字憑證。如此，軟體需求分析的工作才算完成。規格文件，是完成軟體需求分析後的必要文件。對於這樣重要的文件需有以下幾方面的要求：

- 正確性與完整性(correctness and completeness)：按照不正確或不完整的需求開發出來的產品不是使用者所要的。因此我們對規格文件要求正確性與完整性。
- 一致性(consistency)：不一致的規格在其各部分規定的需求，是互相矛盾的。因而我們對各文件要求一致性。
- 不混淆的(unambiguous)：混淆的需求，會讓不同人做出不同的解釋。由於各種解釋都行得通，所以到時候會不清楚哪個才是使用者真正的需求。因而我們對規格文件要求也是不混淆的。
- 可追蹤(tracking)和易修改的：寫下來的需求應該可被檢索、分割和交互追蹤搜尋，以便於使用、修改和擴充。
- 簡單易懂：規格文件要讓使用者容易看懂。注意避免套用很多軟體技術的專業術語。

規格文件的重要無可比擬。無論需求分析做了多少事情，若規格文件沒有做好，則一切努力將化為虛空。在 USB 3.0 開發過程中，廠商將需求分析寫成規格文件後，再依據 USB 3.0 SPEC 中的 protocol layer 及 link layer 層將其細節功能與限制要求寫入 USB 3.0 的 Engineering SPEC 中，之後便可根據規格書設計 USB 3.0 產品。

2.2.3 設計

在進行數位 IC 設計時，最初提出的是此 IC 的構想與演算法，接著才是逐步定義出此 IC 的規格要求。此時可由硬體描述語言以"行為描述"(behavioral description)的方式撰寫此 IC 功能，下一步會以暫存器轉移(Register Transfer Logic，RTL)的方式描述資料與訊號轉變的行為，接著再以邏輯閘(gate)方式描述邏輯電路工作方式，經驗證確定一切無誤才能進行布局(layout)，實際下線(tape-out)做成硬體 IC。在 USB 3.0 的數位 IC 設計過程中，廠商將規格書所列的 USB 3.0 模組先以抽象化規格的方式簡單列出，並簡述其內容元件。然後再使用 System Verilog 和 System C 程式語言撰寫完整的元件模組。最後根據 USB 3.0 的 Engineering SPEC，將 link layer 及 protocol layer 中運作的 state 及 substate 所敘述的功能撰寫出來。如此完成 USB 3.0 與硬體 IC 合成前的設計與實作。

2.2.4 驗證和確認

在數位 IC 設計需與硬體整合前，須先進行驗證與確認。

- 驗證(verification)：驗證建立的產品與其規格是否相符。
- 確認(validation)：確認軟體執行使用者真正的需求。

驗證(verification)過程可分為靜態與動態驗證(static and dynamic verification)。靜態驗證是利用靜態系統表示的分析來發現有關的問題(即檢查)，可用文件和原始碼分析工具作輔助。動態驗證與實作和觀察產品行為有關(即測試)，就是系統執行測試資料和觀察測試資料的操作行為。驗證與確認的計畫應早在開發過程就開始並需在靜態驗證與動態驗證間取得平衡。在驗證過程中，先實行檢查(即靜態驗證)，檢查包括檢視系統的原始碼，由於不需要系統的執行，所以可在程式實作前進行。若能在檢查中發現缺失，就可減少測試時所花費的時間及人力。檢查和測試是互補的，並非對立的驗證技術，可一起於驗證與確認過程中使用。檢查是檢查系統是否與規格相符，而非檢查是否與客戶的需求相符。由於檢查是屬於靜態驗證，因此在檢查過程中無法檢查非功能性的特徵，如效能、可使用性等。檢查又可分為規格檢查與程式檢查階段。

規格檢查(specification review)：根據需求被制定出的規格在靜態驗證過程中需被檢測。規格檢查需定期舉行會議討論來找出其中的缺失。規格檢查

的規範如下[1]：

- 根據參與人員的經驗選出會議主持人(moderator)
- 所有核心人員均需出席
- 在 review 會議結束後，所有參與者需至少提出一個建議
- 每位參與會議人員，需事先以每小時閱讀 20 頁的速度來閱讀規格文件
- 如果 moderator 覺得參與者太少或 meeting 的效率可能會不好則可延期會議
- 每次會議討論應以每小時 15 頁的速度進行
- 每項 review 的建議需在達成共識後記錄起來
- 所有紀錄要交由另一位品質管理人員以審查此次 review 會議
- 根據這些紀錄，品質管理人員需查看該次會議是否有效率

程式檢查(program inspection)在靜態驗證過程中是一個重要程序，須以正式的方法審查。檢查出來的缺失可能是邏輯錯誤、程式碼中資料格式錯誤，如未被宣告的變數或資料超過設定值等。檢查前，需遵守下列規範：

- 程式檢查需有明確的規格(precise specification)來遵循
- 程式碼中需包含 20%以上的註解說明
- 程式碼必須有架構及可維護姓
- 所有函式、變數等的命名必須一致
- 應準備好一份錯誤檢核清單
- 經營者須接受在軟體開發過程中因檢查而增加的成本

在程式檢查的過程中向檢查團隊報告系統概要後，將程式碼和相關文件發送給相關的檢查人員，記下檢查會議的時間與發現的錯誤，根據錯誤來進行修正，最後再重新檢查直到所有程式檢查完成。由於程式檢查需耗費大量人力資源，因此檢查團隊至少由四位成員組成，包括撰寫程式的程式工程師、負責尋找錯誤的檢查工程師、解釋程式碼的工程師、主持會議及記錄檢查核定清單的會議主席，會議主席(moderator)須根據參與人員的經驗選出。接下來需要檢查核定清單記錄中所發現的錯誤。為確保檢查品質，須制定檢查速率，在會議中應以每小時檢視約 500 行程式碼的速度進行。在個人準備時，應以每小時檢視約 125 行程式碼的速度事先 review。動態驗證過程中(即測試過程)，需要三項輸入：測試案例(test case)、數位 IC 產品程式、需求規格(specification)。測試為將測試案例放入數位 IC 產品程式中執行，然後把執行結果和需求規格的預期結果作評估比較。評估比較後會有正確與錯誤兩種狀況，若發現錯誤，則說明產品有誤。相反，若比較結果是正確的，則表示產品沒有錯誤。

數位 IC 產品程式和需求規格都是測試輸入，測試過程最關鍵的就是在測試案例的設計。傳統數位 IC 驗證測試案例，採用 Formal Test 與 Random Test 並行：

(a) Formal Test

正規測試(Formal Test)採用排列組合的方式，將所有可能的輸入狀況輸入至 IC 的輸入端，記錄每一種輸入狀況得到的輸出結果，並與預期的輸出結果做比對，以證明 IC 是否動作正確。此方法若用於沒有記憶功能的組合邏輯(combinatorial logic)尚可接受，因其輸入的可能性受到輸入接腳數量的限制。若輸入數量為 N 條，則需測試的狀況只有 N^2 種(僅考慮 1/0 輸入)。以 2 輸入的 AND 閘而言，僅需要 4 次輸入即可測試所有狀況。但對於循序邏輯(sequential logic)而言，狀況並不單純。循序邏輯 IC 內部是由狀態機(state machine)所構成，每次輸入除了產生對應輸出外，其內部還會記錄其執行的『狀態』。輸出端每次輸出的結果除了與本次輸入相關，同時與先前輸入所造成的狀態有關。狀態數量越多，IC 內部能紀錄的次數就越多，其狀況如同下棋一樣，均為指數成長的複雜度。以簡單的狀態機為例：具有一條輸入(0/1)，三種狀態的 IC，每次輸出至少與前 2 次輸入有關。故需將所有狀態都測試一次，至少需測試 2^3=8 次輸入。當輸入腳位數量增加，且內部狀態也增加時，需測試的組合將以指數方式增加。一個有 5 個狀態(state)和內部狀態有 10 種的 IC，至少需$(2^5)^{10}$種輸入組合才能測試一次(數量高達百兆種組合)。雖目前電腦速度驚人，但若加上考慮狀態機重複循環的狀況，則測試組合如同無限迴圈一般無窮無盡，即使超級電腦也無能為力。因此一般進行 IC 測試時，通常僅針對關鍵的核心進行 formal test。至於『核心』的範圍界定與測試條件的嚴格程度，則需視人力成本與時間來決定。

(b) Random Test

隨機測試(Random Test)就是在這種狀況下應運而生的折衷方法。同樣針對具有狀態機的循序邏輯 IC，採隨機選取某些組合進行測試，此即隨機測試，選取的組合數量同樣取決於成本多寡。此方式雖節省時間與經費，但卻犧牲測試的可靠性：如何以有限的組合測試各種未知的問題，隨機組合的數量再多，都無法涵蓋完整。數位 IC 必然有極難測試的邊界狀況(corner case)存在，在此狀況下僅能靠運氣找出錯誤。

(c) Machine Assisted Selective Random Test

即機器協助選擇性隨機測試(Machine Assisted Selective Random Test)，數位 IC 的工作通常具有特定的標準流程。廠商提供一份 IC 規格書，使用者按照此規格設計輸入資料，因此測試人員可以預期一些特定的輸入狀態組合，此預期的輸入狀態組合可採用全盤測試的 formal test 進行。但最困難的就是錯誤的，不合規定的輸入組合，這些組合有可能將 IC 內部狀態機引導至無法預期的狀態，導致無法恢復正常工作的嚴重錯誤，此時整個系統會發生錯誤。但由於這種不合規定的輸入組合是無法預期的，造成的混亂情況也可能是驗證軟體無法辨認的，隨機測試未必能涵蓋所有的狀態組合，此時就需仰賴人工手動操作，由具有經驗的測試人員，選擇特定的輸入組合，針對比較有意義的錯誤輸入進行模擬。這步驟我們稱之為『Selective Random Test』，這個階段如有適當的工具協助，則稱為『Machine Assisted Selective Random Test』。

傳統上，要進行這種測試時，僅能依靠工程師以程式編輯工具撰寫驗證程式(test bench)，並且逐次輸入測試程式，對照輸出結果。驗證無誤時，再開啟程式檔案，重新組合程式碼，再次執行。所有流程都是在純文字檔案中進行，對測試者而言是枯燥且繁複的手續，不僅缺乏組織規劃可言，且缺乏有效率，有意義的記錄方式。我們對此需求，設計了一套『狀態機模擬器產生器』(Generator of FSM Simulator)，可自動產生『狀態機動態模擬器』，僅需簡單步驟即可產生狀態機模擬網頁，讓測試人員透過圖形畫操作介面，輕易產生有意義的狀態機輸入組合之程式碼[4]。

在 USB 3.0 數位設計開發或任何 IC 設計的測試上，使用者(測試人員)可依據輸入格式輸入有限狀態機的描述，並利用我們提供的工具，產生出模擬器，然後測試人員利用任何支援 JavaScript 的網頁瀏覽器透過圖形化操作介面，操作此模擬器，就像操作飛行模擬器般進行飛行之模擬。模擬過程可自動紀錄成 test bench 程式碼，然後將此程式碼放入 USB 3.0 數位設計程式中作測試。若運作錯誤，先釐清非由狀態模擬機所產生的錯誤後，再判斷此錯誤是廠商對 USB 3.0 formal SPEC 的誤解或是廠商在 USB 3.0 數位 IC 程式設計中發生錯誤抑或是 USB 3.0 formal SPEC 上的錯誤。然後根據此錯誤作修正，最後根據廠商的有限資源、人力及時間，得到高品質的驗證。

2.2.5 維護

產品演進與維護是數位 IC 設計開發過程中的最後一個步驟。當 IC 產品交由使用者正式上線運轉數天、數月甚至數年後，若需要發行下一版本，或發現有些部分出現缺陷，有些部分需要補強，或使用者覺得有些地方要更改需求與規格，甚至大翻修時，便需進行維護。數位 IC 設計開發產品維護已有現成的工具可以使用，在本研究中我們使用 Cadence 提供的 EDA 工具來維護並升級我們的數位 IC 產品。

三、結論

在本研究中，我們根據 IC 設計需求及參考軟體開發模型中的現有架構並加以修改，設計出一套符合軟體工程需求的標準流程(SOP)，藉由該流程，IC 驗證能被標準化，並能明確掌握所需時程與人力，降低成本，提高產能。此外由於所有循序邏輯 IC 內部都是採用有限狀態機方式完成狀態轉移，驗證時須設計各種不同狀態轉移進行詳細測試。有鑑於傳統正規測試花費時間成本過於龐大，而隨機測試無法找到有意義條件，本研究提出採用『機器協助選擇性隨機測試』(Machine Assisted Selective Random Test)的方式進行最有效率的驗證流程。我們特別針對 IC 設計的『狀態機』提出了一套驗證工具，稱為『狀態機模擬器產生器』(Generator of FSM Simulator)。藉由此工具可針對任何一種狀態機，輕易的產生多階層/圖形化的狀態機模擬器。一個軟體設計流程需要歷經多年的使用改進方能達到成熟階段。本研究所發展的驗證流程與工具不僅能適用於 USB3.0 開發，並能適用於任何一種數位 IC 的驗證流程。

參考文獻

[1] Ian Sommerville, “Software Engineering”, 7th ed., Addison-Wesley, 2004.

[2] Scott W. Ambler, “The Agile System Development Life Cycle (SDLC)”, http://www.ambysoft.com/essays/agileLifecycle.html. Retrieve October 2010.

[3] Mike M and Robert E., 2007, “Hardware Verification with SystemVerilog”, 1st Edition, New York, Springer, pp. 10-46.

[4] Yao-Nan Lien and Li-Cheng Chi, "A Generator for Finite State Machine Simulators", submitted to 22th Conference on Object-Oriented Technology and Applications (OOTA), Jul. 8-9, 2011.

[5] 邱郁惠, "UML-SystemC 晶片設計實務," 碁峰資訊, 2008.

[6] 趙善中、郭麗齡、尤柄文, "以架構為中心的軟體工程," 儒林圖書, 2006.