

國立臺灣大學電機資訊學院電子工程學研究所

碩士論文

Graduate Institute of Electronics Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis



處理器的故障注入攻擊檢測與回復

Fault Injection Attack Detection and Recovery for
Processor

石文緯

Wen-Wei Shih

指導教授：黃俊郎 博士

Advisor: Jiun-Lang Huang, Ph.D.

中華民國 111 年 2 月

February, 2022



致謝

在讀研究所這段時間，經歷了大大小小的事情，有挫折也有成就感，從一開始的懷疑自己，到一步一步完成研究與計畫。進台大之前，可能有很多人並不認可，但謝謝當初那個堅持往這條路的自己，並沒有因為別人的看法而放棄；也謝謝教授在這段時間給予的幫忙，在我們陷入研究撞牆期的泥淖時，給予及時的關懷，一步一步的建立起信心，並且完成研究。

也要謝謝實驗室的同學，尤其是 Security 組的同學正儒、顯峯、天慈以及子元，在我研究或其他事情忙不過來時給予幫助，雖然在做計畫的時候，常常因為失敗而沮喪，但我知道至少，我不是一個人在努力著，而成功實作出來時，也能一起分享成功的果實。

再來，要感謝我的家人，不僅沒有給我任何壓力，甚至在於我快支撐不住時，提供給我最實質的幫助，一句話就能讓我安心許多，讓我也能夠沒有後顧之憂的去做我想做的事情。

最後，要感謝口試委員願意在一過完年，就撥空前來參與我的口試。



摘要

隨著嵌入式系統與物聯網的快速發展，資安的議題也隨之被更加重視，然而，嵌入式系統因為資源與成本的限制，往往僅配備軟體資安的防護機制，例如加解密或完整性驗證等。但此機制是建立在假設硬體是安全的情況下，而嵌入式系統總是被以相同規格與設計大量生產，只要惡意的攻擊者可以破解一個裝置，便能以相同手法成功攻擊其他的設備。

處理器作為整個系統的核心，安全的議題更應該被重視，而攻擊者卻可以透過簡單實現且低成本的故障注入攻擊影響處理器的運作，運行在系統上的軟體安全機制便不再安全。

因此，本論文提出一個輕量化且能快速偵測故障注入攻擊的偵測器以及提供一個使處理器能夠在被故障注入攻擊時，回復至已知且未受攻擊的最後狀態，並且繼續執行，如此一來，攻擊者便無法輕易改變處理器的運作，進而達到防護的效果。

關鍵字：故障注入攻擊，時鐘短時脈衝波形干擾，偵測器，處理器，回復



Abstract

With the rapid development of embedded systems and the Internet of Things (IoT), the security issue becomes more important. However, due to limited resources, embedded systems are often only equipped with software security mechanisms, such as encryption and decryption or integrity check, etc. But, this mechanism is based on the assumption that the hardware is secure, and embedded systems are always mass-produced with the same specifications and designs. As long as a malicious attacker can attack one device, they can successfully attack other devices in the same way.

The processor is the core of entire system, security issue should be paid more attention. But, the attacker can use a simple and low cost fault injection attack to affect the processor. Thus, the software security mechanisms running on the system are no longer secure.

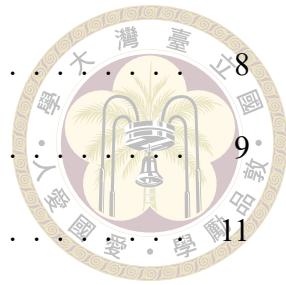
Therefore, we proposed a lightweight detector that can quickly detect fault injection attacks and provides a method that enables the processor to recover to the last known and unaffected state when it is attacked by fault injection attack and continue to execute. So that, the attacker can not easily change the behavior of the processor and then achieve the effect of protection.

Keywords: Fault Injection Attack, Clock Glitch, Detector, Processor, Recovery

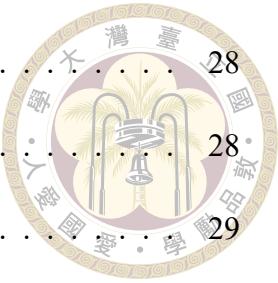


目錄

	Page
致謝	i
摘要	ii
Abstract	iii
第一章 Introduction	1
1.1 Importance of Embedded Systems Security	1
1.2 Security Vulnerability of Processor	1
1.3 Related Work	2
1.3.1 Detection	2
1.3.2 Recovery	3
1.4 Motivation	3
1.5 Contribution	3
1.6 Organizations of The Thesis	4
第二章 Preliminaries	5
2.1 Timing Constraints of Digital Circuits	5
2.2 Fault Injection Attack	6
2.2.1 Implementation of Fault Injection Attack	7
2.3 Load-Store Architecture	8



2.4	5-Stage Pipelined MIPS32 Architecture	8
2.5	Static Random Access Memory	9
2.6	Threat Model	11
2.6.1	Fault Injection Attack Methods Description	11
2.6.1.1	Clock Fault Injection Attack	12
2.6.1.2	Increase The Propagation Delay	12
2.6.2	The Impact of Fault Injection Attack for Processor	12
第三章	Proposed Method	14
3.1	Overview of The Fault Injection Attack Detector and Recovery Mechanism	14
3.2	Hardware Architecture of Proposed Method	15
3.3	Detector Mechanism and Design	16
3.3.1	Core Idea of The Detector	16
3.3.2	Design of Delay Chain	17
3.3.3	Discussion	17
3.4	Recovery Mechanism	19
3.4.1	Core Idea of Recovery Mechanism	19
3.4.2	Recovery Mechanism	20
3.4.2.1	Register File	20
3.4.2.2	Data Memory	22
3.4.2.3	Flush The Pipelined Register	24
3.4.2.4	Program Counter	25
3.4.3	Discussion	26
第四章	Experimental Result	27
4.1	Experiment Setup	27

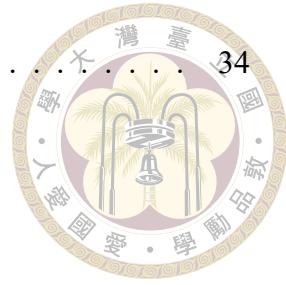


4.2	Experiment Architecture	28
4.2.1	I/O of Each Component	28
4.3	Utilization of FPGA Resource of Each Component	29
4.4	Comparison with Previous Work	29
4.5	Fault Injection Detector	30
4.6	Instruction Skipping	30
4.7	Recovery Mechanism (Simulation)	31
4.8	Summary	35
第五章	Conclusion and Future Work	36
5.1	Future Work	36
5.2	Conclusion	38
參考文獻		39



圖目錄

2.1	Timing Constraints of Digital Circuits [1]	5
2.2	Fault Injection Attack [2]	6
2.3	Different Methods of Generating Clock Glitch [3]	7
2.4	Clock Glitch Generation	8
2.5	5-Stage Pipelined MIPS32[4]	8
2.6	SRAM Cell [5]	10
2.7	Waveform of SRAM Write Operation	10
2.8	Waveform of SRAM	11
3.1	Hardware Architecture of Proposed Method	15
3.2	Implementation of Fault Injection Attack Detector	16
3.3	Timing Diagram of Delay Chain	17
3.4	Timing Diagram of Detector (Normal Situation)	18
3.5	Timing Diagram of Detector (Timing Violation)	18
3.6	Overall Recovery Mechanism	19
3.7	Register File with Shadow Register	21
3.8	Data Memory with Data Buffer	22
4.1	Experiment Architecture	28
4.2	Waveform of Detector	30
4.3	Waveform of Instruction Skipping	31
4.4	Waveform of Recovery Mechanism without Recovery Requirements	32
4.5	Waveform of Recovery Mechanism with 2 Recovery Requirement	32
4.6	Flush The Pipeline	33
4.7	Recover The Register File Using Shadow Register	33



表目錄

4.1 Utilization of FPGA Resource of Each Component	29
4.2 Comparison with Previous Work for Detection	29



第一章 Introduction

1.1 Importance of Embedded Systems Security

在現今社會中，隨著科技的進步、嵌入式系統的普及，特別是物聯網 (Internet of Things) 的快速發展，嵌入式系統在我們生活周遭已經變得隨手可得，因此，嵌入式系統的安全變得更加重要。況且，由於物聯網需要與其他裝置溝通的特性，若是一台裝置被駭客攻擊並且控制了，有時候不僅是影響到該裝置而已，甚至可能危害到整個生態系。然而，因為受限於嵌入式系統資源有限的情況下，往往實現於嵌入式系統的安全等級並不會如個人電腦般的好，而嵌入式系統總是被大量以同樣規格、同樣的設計製造，因此只要駭客成功攻擊一個裝置，便能使用相同的手法攻擊其他一樣的裝置，進而對嵌入式系統造成極大的威脅。

1.2 Security Vulnerability of Processor

首先，因為嵌入式系統的普及造成安全議題也顯得被更加重視，然而嵌入式系統之中，處理器便是不可或缺的角色，因此，如何保護處理器的安全也是一門重要的課題。在資安的領域中，軟體的安全皆是建立於假設所有執行的硬體皆是安全且可信任的情況下，但硬體的安全卻更難做到全面的防護，如果無法從硬體的層面去發展，軟體的安全便無法完全的被信任。一個撰寫完成的軟體勢必需要硬體的配合，才能運算出預期中的結果，而最常使用的便是處理器，此時，如果

攻擊者可以透過特定攻擊手法而改變處理器的行為，整個系統便處在風險之中。

舉例來說，安全開機是指在開機過程中，能夠確保整個系統所執行的程式或作業系統 (Operating System) 是被驗證過的，也就是說，所執行的程式不能被竄改過。安全開機的過程很常使用的方式有使用完整性檢查 [6] 確保程式未被竄改、身分驗證 (Authentication) 來識別該程式是否為製造商所認可，甚至使用加密的方式來保護 [7],[8]。然而以上這些方法都仰賴著硬體的正常執行，如果攻擊者可以直接或間接影響了處理器的行為，使得上述的方法失效，安全開機的機制便存在著風險。

1.3 Related Work

1.3.1 Detection

[9] 提出了一個基於偵測違反時序 (Timing Violation) 的偵測器，利用加入延遲鏈 (Delay Chain) 的方式，使兩個時脈訊號產生相位差，若發生了基於違反時序的故障注入攻擊，便會改變兩個時脈的相位，進而偵測到攻擊的發生，此方法與本論文所提出的方法皆是使用加入延遲鏈來偵測，他所提出的方法可以處理老化效應 (Aging Effect)，但比本論文提出的方法需耗費較多的硬體資源。

[10] 演示了將一樣的資料以管線 (Pipelined) 的方式送進去進階加密標準 (Advanced Encryption Standard) 的硬體做加密，並且在運算時不斷比較所先後計算出的結果，用來偵測是否發生了攻擊，但也因為如此需要耗費較多的時間與資源。

[11] 描述了如何使用複製一些特定電路，使在加密時保持資料及鑰匙的一致性，若複製的電路與原有的電路產生不一樣的結果，便能偵測出故障注入攻擊的發生，因為需要複製較多的電路，所需增加的硬體資源也會隨之增加。



前述所提及的皆是針對此種攻擊而特別設計的研究，而偵測違反時序的研究也非常多，如 [12] 透過了 timing-borrowing 的技術，使用暫存器以及正反器儲存時機的不同來偵測違反時序的發生，然而其他偵測時序違反的研究是否適合用於此種攻擊，還值得研究。

1.3.2 Recovery

[13] 在偵測到此種攻擊時，使用額外設計的例外模式 (exception mode) 去做回復，此種做法可以有效處理回復的問題，但因為還透過軟體的輔助，會造成運算效能的降低，以及透過軟體的方式進行回復，若是軟體本身就被竄改而移除此項功能時，此種攻擊還是能夠被實現。

1.4 Motivation

故障注入攻擊是屬於實體攻擊 (Physical Attack) 的一種，因為攻擊可以直接對於裝置做最大限度地操控，因此實體攻擊普遍被認為是較難以防範的攻擊，也因為如此，研究此種攻擊防禦手段的研究相對較少。然而，故障注入攻擊是一種非常低成本就能實現的實體攻擊，且能製造出顯著的效果，造成巨大的威脅。

不同於其他論文，大多數只僅針對如何偵測做研究，較少透過實際的回復機制，使處理器得以回復至未被攻擊的最後狀態及正常的執行，進而降低此種攻擊所帶來的影響。

1.5 Contribution

本論文提出一個輕量的偵測器用以偵測基於違反時序的故障注入攻擊，此偵測器可以在發生攻擊時迅速偵測到，輔助回復的機制可以正常執行。



在本論文裡，也提出了一個偵測到故障注入攻擊時，迫使系統回復並且繼續正常執行的方法。最後我們也實現了一個能夠自我回復的機制在管線(5 Stage Pipelined)的MIPS32上，使此處理器可以回復至未被攻擊且已知的最後狀態並繼續正常的執行，降低故障注入攻擊在此種處理器的影響，並利用概念驗證的實現方式證明可行性。

1.6 Organizations of The Thesis

本章說明了研究動機、相關研究與研究貢獻。第二章將介紹一些預備知識包含威脅模型以及一些常見的故障注入攻擊手法。第三章將介紹本論文提出的偵測攻擊的方法以及該如何回復。第四章為本論文提出的實驗結果與分析。最後，第五章提除未來的研究方向並總結本論文。



第二章 Preliminaries

2.1 Timing Constraints of Digital Circuits

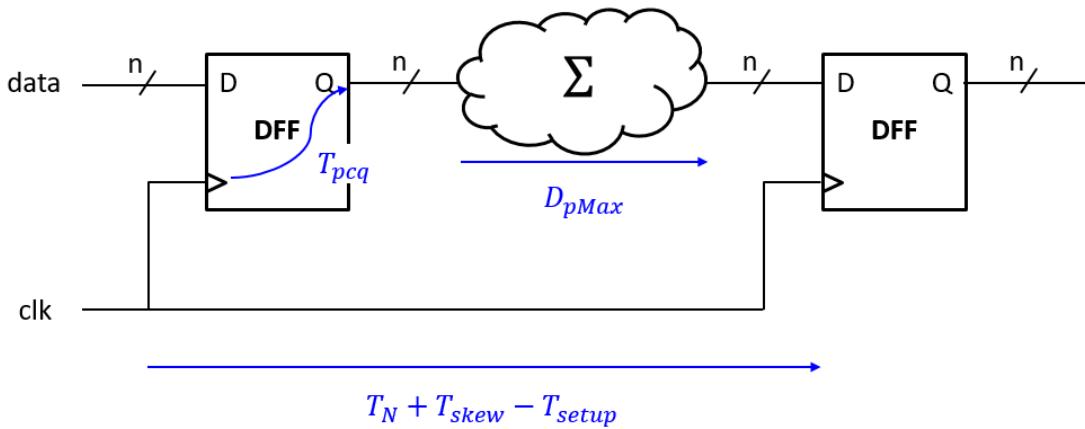


Figure 2.1: Timing Constraints of Digital Circuits [1]

在設計數位電路時，皆會遵從著時序的約束 (Timing Constraints) 以確保設計出來的電路可以正確的運作 [1]。如 Figure 2.1 所示，會使用一個共同的時鐘 (Clock) 訊號來同步正反器 (Flip-Flop) 的運作，中間則用 Σ 符號來表示組合邏輯電路 (Combinational Logic Circuits)，資料會在時鐘訊號正緣 (Positive Edge) 觸發時，經過組合邏輯電路運算後並在下一個正緣觸發時門 (Latch) 入下一級的正反器。其中 T_{pcq} 代表時鐘訊號觸發時，從時鐘到正反器的輸出完全穩定時所需的傳播延遲 (Propagation Delay)， D_{pMax} 代表組合邏輯電路的最大容許傳播延遲， T_N 代表時鐘週期 (Clock Period)， T_{skew} 代表兩個不同正反器之間在接收到時鐘訊號上存在的些許時間差， T_{setup} 則代表資料必須在時鐘訊號觸發之前保持穩定的時間量，才能使



電路正常的運作。因此我們可以得知時間方程式 2.1 的關係，再加以移向整理成式 2.2。

$$T_N > T_{pcq} + D_{pMax} + T_{setup} - T_{skew}$$

$$D_{pMax} < T_N - (T_{pcq} + T_{setup} - T_{skew}) \quad (2.2)$$

2.2 Fault Injection Attack

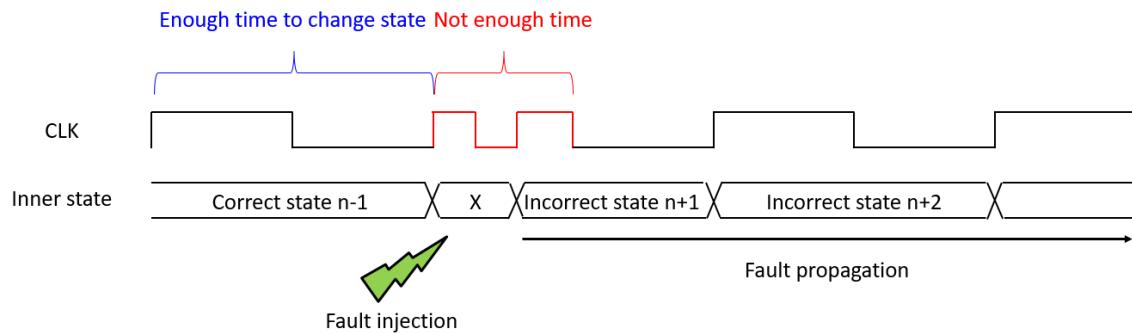


Figure 2.2: Fault Injection Attack [2]

故障注入攻擊是一個簡單實現又低成本的攻擊，如果攻擊者可以操控裝置的電壓、時鐘訊號或是工作溫度等因素便能使裝置的行為改變。首先，由 Figure 2.2 可得知，電路的內部狀態因為尚未受到故障注入的攻擊，在正常情況下，內部的狀態皆能保持正確；一旦故障注入之後，便會因為運算時間不足而改變電路的內部狀態，並且這個錯誤的狀態也會隨著時間往下傳遞，久而久之便無法得知正確的結果。

然而，因為現今的電路也越來越複雜，在攻擊造成的損害隨著往下傳遞時，影響的層面也會變得更加難以分析。因此如果可以越早偵測到此種攻擊的發生，要將電路回復以及對於電路的損害降低也會變得相對困難，浪費的時間與資源也會相對較多。



2.2.1 Implementation of Fault Injection Attack

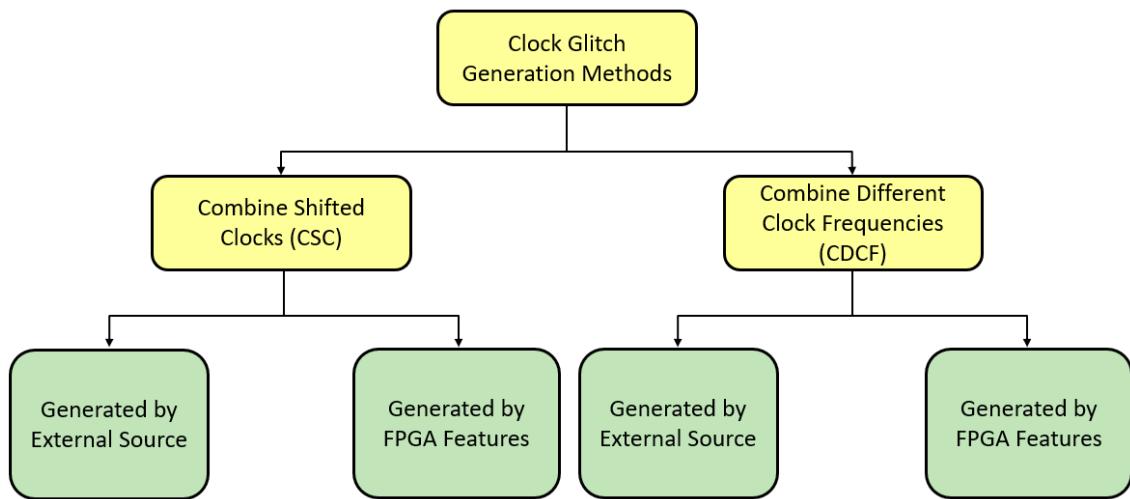


Figure 2.3: Different Methods of Generating Clock Glitch [3]

因為本論文所要發展的技術是可以偵測到故障注入攻擊並且回復，因此我們攻擊的實現參考了 [3] 所提及且適合我們開發環境的方式，使用場域可程式化邏輯開陣列去設計一個可以產生時鐘短時脈衝波形干擾 (Clock Glitch)，後面用短時脈衝波形干擾產生器 (Glitcher) 稱之。

我們由 Figure 2.3 可知，時鐘短時脈衝波形干擾可以透過結合不同相位或者不同頻率的時鐘訊號，而我們使用的是結合兩個不同頻率的時鐘訊號，較快的時鐘訊號，我們使用了開發板上的石英震盪器所產生的訊號，提供 100MHz 的時鐘訊號；較慢的時鐘訊號則是使用了除頻器 (Frequency Divider) 產生 10MHz 的時鐘訊號，此訊號為下一級電路的時鐘訊號來源，其中包括了管線化 MIPS32 以及後面所介紹的偵測器。結合兩個時鐘訊號時，將兩個時鐘訊號用一個多工器去做選擇，一旦觸發訊號被觸發，便會將時鐘的來源切換到較快的時鐘訊號，產生如 Figure 2.4 的波形。因此，在正常情形下，處理器以及偵測器會操作在 10MHz 的頻率下，然而當觸發訊號一發生，時鐘訊號馬上會切換成 100MHz，產生時鐘短時脈衝波形干擾，進而實現故障注入攻擊。

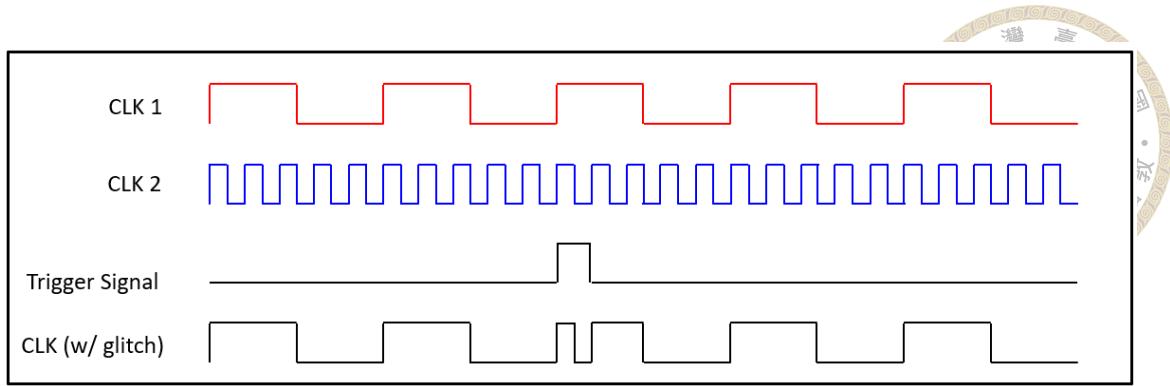


Figure 2.4: Clock Glitch Generation

2.3 Load-Store Architecture

載入-儲存是一種指令架構，將指令區分為從記憶體載入資料到暫存器堆(Register File)，以及算術邏輯運算，運算所需的資料來源只會是暫存器堆，最後結果也寫回暫存器堆。現今的精簡指令集計算機(RISC)指令集架構也都是採用載入-儲存架構，例如:PowerPC，ARM架構，RISC-V，MIPS。

2.4 5-Stage Pipelined MIPS32 Architecture

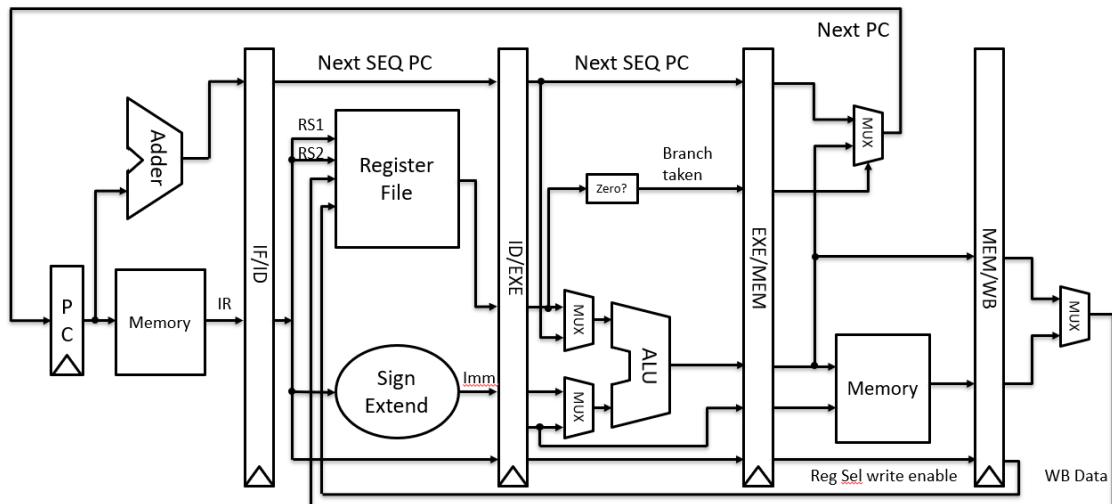


Figure 2.5: 5-Stage Pipelined MIPS32[4]

本論文所使用的處理器架構為管線式 MIPS32，MIPS (Microprocessor without Interlocked Pipeline Stages)，是一種採取精簡指令集 (RISC) 的指令集架構 (ISA)，由美國 MIPS 電腦系統公司開發，現為美普思科技。



管線式 MIPS32 將指令分為五個階段，分別是

- (1) 指令取得 (Instruction Fetch)
- (2) 指令解碼 (Instruction Decode)
- (3) 執行 (Execute)
- (4) 記憶體存取 (Memory Access)
- (5) 寫回 (Write Back)

2.5 Static Random Access Memory

靜態隨機存取記憶體 (SRAM) 為一種揮發性記憶體，靜態代表只有在通電時，能夠儲存著資料；一旦將裝置斷電，儲存在記憶體的資料便無法保留，而它也與動態隨機存取記憶體 (DRAM) 不同，不需要週期性去更新每個記憶體細胞 (Cell) 的資料。

如 Figure 2.6 為一個靜態隨機記憶體的一個記憶體細胞，通常此記憶體會用將每個細胞以陣列的方式組合而成，而細胞的選擇則依靠圖中的 WL (Word Line) 以及 BL (Bit Line)，當需要寫入資料至記憶體時，首先得像 Figure 2.7 所示，將 WL 提升至額定電壓 (VDD) 以及將想寫入的資料透過 BL、BLB 寫入，若想寫入 0，則將 BL 設為 0，BLB 設為 1；想寫入 1，則將 BL 設為 1，BLB 設為 0。

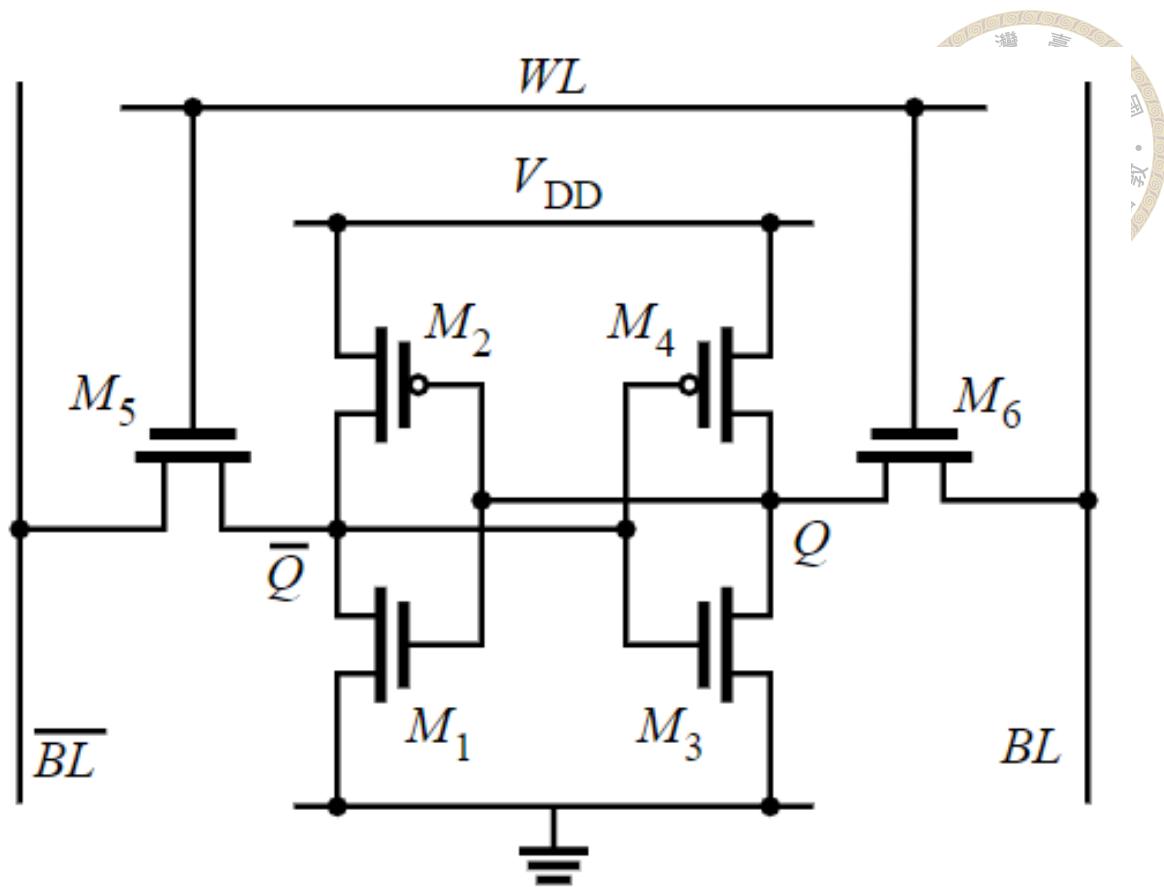


Figure 2.6: SRAM Cell [5]

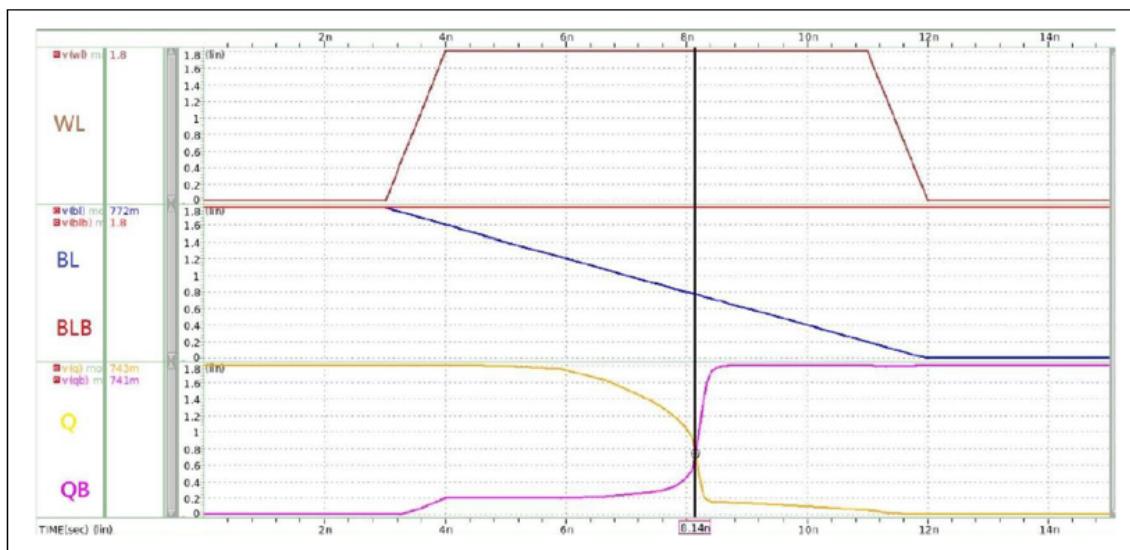


Figure 2.7: Waveform of SRAM Write Operation



根據 [14] 所提及，現今的靜態隨機存取記憶體為了提高在讀寫的速度，會將存在於 BL 與 BLB 的電容預先充電 (Precharged) 至額定電壓。然而，記憶體在做寫入操作時，則必須將 BL 或 BLB 放電至低電位，以達成前述的寫入操作，也因為如此，在整個寫入資料的動作，此步驟為最後階段且最耗費時間，如同 Figure 2.7 所展示的結果一樣。

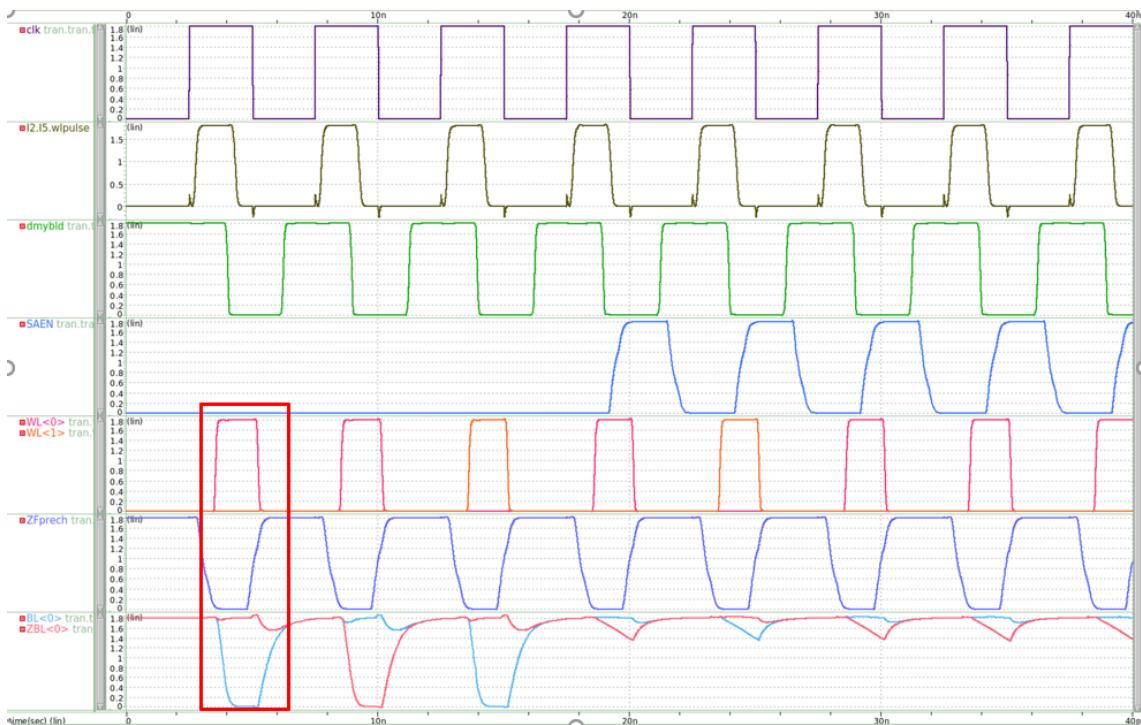


Figure 2.8: Waveform of SRAM

2.6 Threat Model

2.6.1 Fault Injection Attack Methods Description

從式 2.1 可以發現，攻擊者想要產生有效的攻擊，可以操控兩個變數，控制時鐘訊號的週期長短以及提高組合邏輯電路的傳播延遲，最終達成目的的方法皆是造成電路違反時序，以下將直接控制時鐘訊號與增加傳播延遲分開來做介紹。



2.6.1.1 Clock Fault Injection Attack

在此種攻擊中，通常會挑選該裝置使用外部時鐘訊號資源，又或者使用現今嵌入式系統經常具備的動態電壓頻率調整 (Dynamic Voltage Frequency Scaling) 的功能，以達到操控時鐘訊號的目的 [15]，然而此種攻擊通常會針對關鍵路徑 (Critical Path) 去攻擊，只影響部分電路，造成期待的效果，例如：跳過某個指令或運算結果錯誤等。[16],[17] 演示了如何操控時鐘訊號，產生時鐘短時脈衝波形干擾 (Clock Glitch)，因此組合邏輯電路能運算的時間縮短，如同式 2.3，發生違反時序的情形，最終成功做到故障注入攻擊。

$$T_g < T_{\text{pcq}} + D_{\text{pMax}} + T_{\text{setup}} - T_{\text{skew}} \quad (2.3)$$

2.6.1.2 Increase The Propagation Delay

透過式 2.1 可知，攻擊者也能透過增加組合邏輯電路的傳播延遲，造成故障注入攻擊的效果，如式 2.4 關係式。首先，[18] 使用了功率不足 (Underpowering) 的技術，增加組合邏輯電路的傳播延遲，成功演示了故障注入攻擊；而 [19] 使用短暫的電磁脈波 (Electromagnetic Pulse) 攻擊，會引發短暫的功率不足的效果，進而導致傳播延遲的增加，最後一樣能造成電路違反時序，讓進階加密標準加密運算不論是硬體或軟體實現都不再安全；最後，過熱 (overheating) 則是利用溫度上升，增加傳播延遲而觸發了設置時間違規 (Setup Time Violation)[20]。

$$T_N < T_{\text{pcq}}' + D_{\text{pMax}}' + T_{\text{setup}}' - T_{\text{skew}} \quad (2.4)$$

2.6.2 The Impact of Fault Injection Attack for Processor

由 Figure 2.5 可知，管線與管線之間皆是組合邏輯電路，如果發生時鐘短時脈衝波形干擾時，會造成電路來不及運算完成，產生錯誤的結果，然而這些錯誤的



結果便會隨著管線往下傳遞，造成不可回復的錯誤。

舉例來說，如果該系統要執行的程式放置於外部記憶體，代表執行的指令必須從外部記憶體讀取，此種記憶體通常是非揮發性記憶體 (Non-volatile Memory)，像是閃存記憶體 (Flash Memory)，所以在讀取上需要耗費較多的時間，此時如果發生了故障注入攻擊，處理器有很大的機率因為時間不夠而讀取不到指令的情形，最終此條指令就被跳過，而透過 [16],[19] 的演示與分析，可以見到兩者皆做到了將指令跳過的攻擊，而 [21] 也提到了透過故障注入攻擊使得安全開機的機制失效。



第三章 Proposed Method

3.1 Overview of The Fault Injection Attack Detector and Recovery Mechanism

本論文提出一個基於違反時序的故障注入攻擊偵測器，並且在偵測到此種攻擊時，將處理器回復到先前未被攻擊時的最後狀態，此狀態的資料都是已知且可控制的，因此處理器得以繼續執行。攻擊者在攻擊時通常會選擇關鍵路徑攻擊，並且保持其餘功能正常運作，否則此攻擊便無法達到預期的效果，但在設計回復的機制時，電路也不能設計得太複雜，導致所設計的方法也被故障注入攻擊所影響。



3.2 Hardware Architecture of Proposed Method

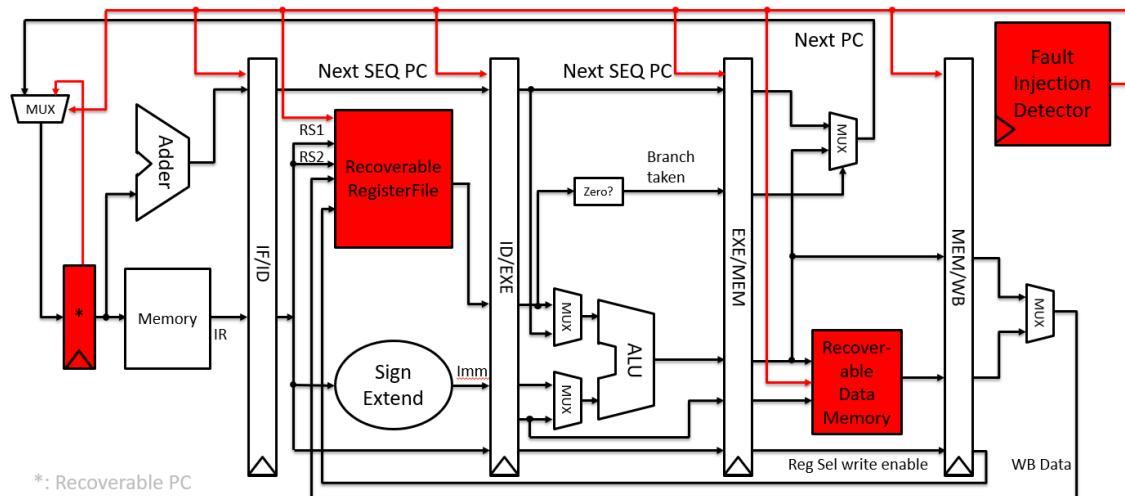


Figure 3.1: Hardware Architecture of Proposed Method

如 Figure 3.1 所示，白色區塊為管線 MIPS32 的原始設計，紅色即為本論文所改變之硬體，整體架構會由偵測器作為主導，當偵測器偵測到故障注入攻擊時，會傳送至必要的硬體並做出反應。



3.3 Detector Mechanism and Design

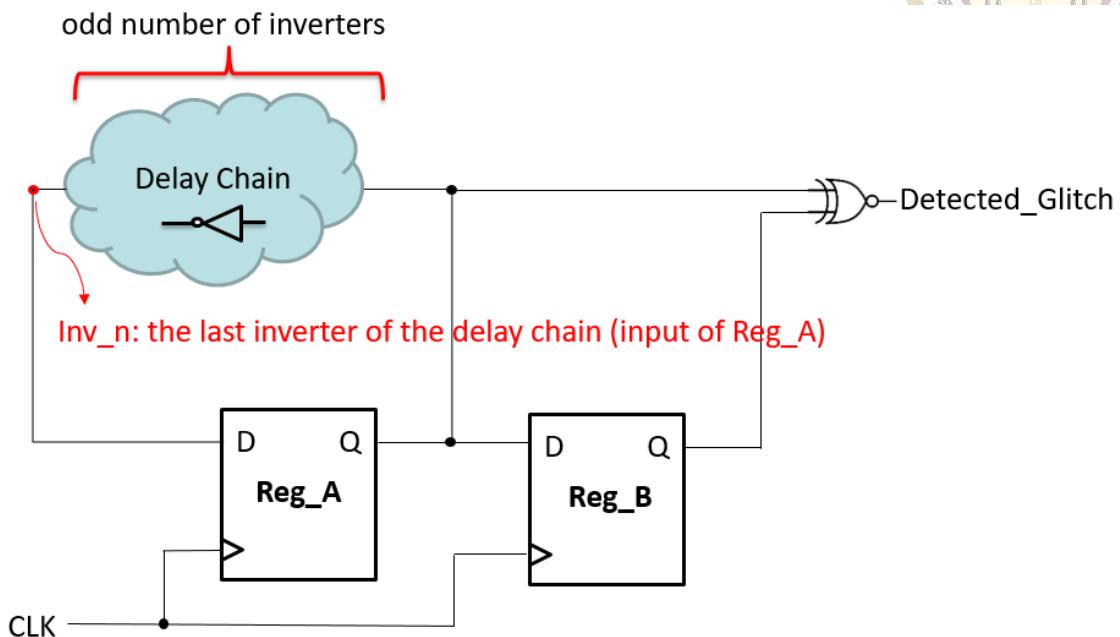
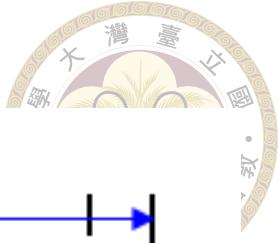


Figure 3.2: Implementation of Fault Injection Attack Detector

3.3.1 Core Idea of The Detector

因為本論文所考慮的威脅模型為基於違反時序的故障注入攻擊，所以提出一個簡單又能偵測違反時序的偵測器。由式 2.1 可知，若要形成有效的攻擊，必須讓電路發生違反時序的行為，因此我們可以利用這個特性，設計一個強迫組合邏輯電路的傳輸延遲提高到可以正常運行的最大值，換句話說，只要有違反時序的條件成立，此偵測器就會發生與正常狀態不一樣的行為，進而偵測到故障注入攻擊的發生。如 Figure 3.2 所示，我們在一個正反器 Reg_A 的輸入與輸出加入一個反向延遲鏈，並且利用另外一個正反器 Reg_B 去儲存上一個時鐘週期時，Reg_A 所儲存的狀態。因此在正常情況下，Reg_A 的值會像是環形震盪器產生 0-1 震盪的效果；如果發生違反時序的情況，代表 Reg_A 的值因為時鐘週期的不足導致來不及更新暫存器的值，因此 Reg_A 與 Reg_B 的值相同，便能得知前一個時鐘週期發生了違反時序，再來我們只要使用一個反互斥或閘 (XNOR) 去比對，即可偵測攻



擊的發生。

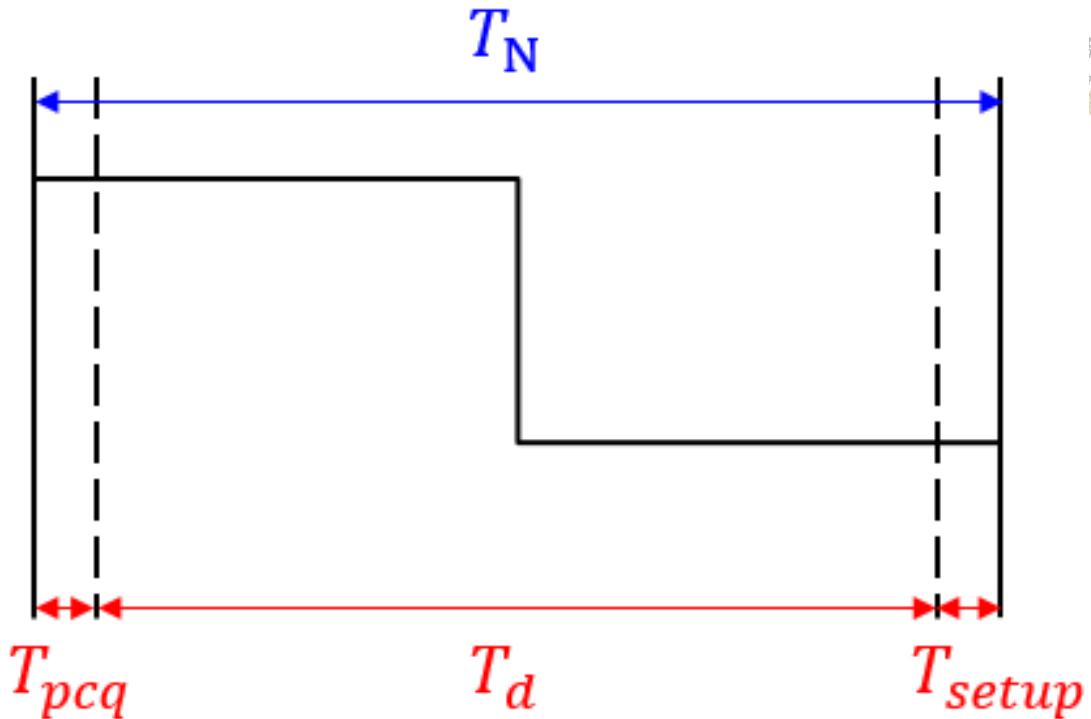


Figure 3.3: Timing Diagram of Delay Chain

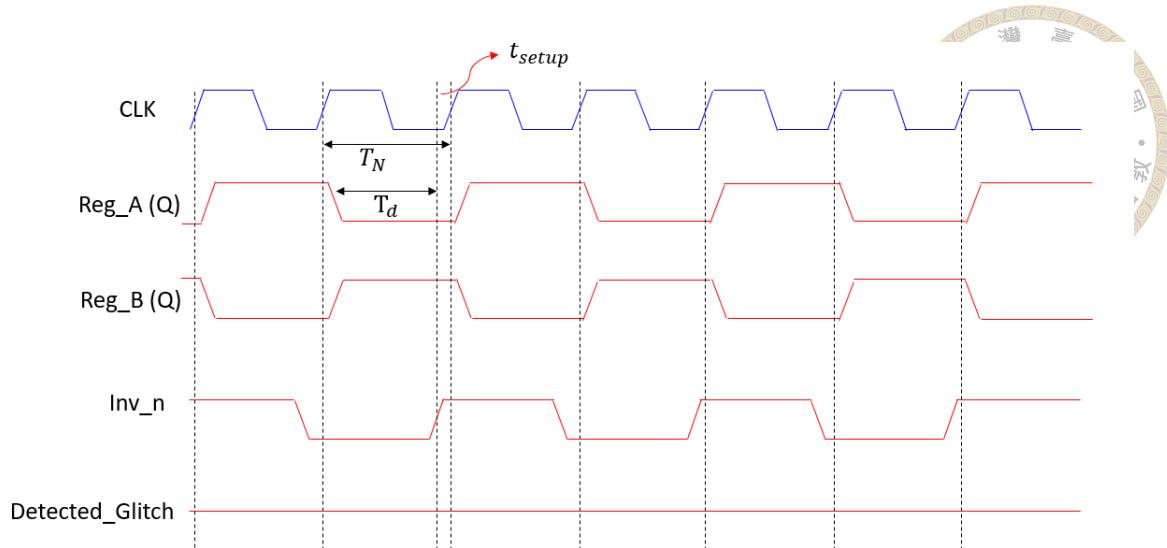
3.3.2 Design of Delay Chain

因為我們想要達到覆蓋率最大化，藉由式 2.2 的設定，可以將 D_{pMax} 置換成 t_d ，代表反向延遲鏈的傳播延遲時間，也就是將反向延遲鏈 (Delay Chain) 所需的時間拉長至整個電路的關鍵路徑所需的時間，如 Figure 3.3，時間關係可以整理成式 3.1。因此只要發生違反時序，便能偵測到。

$$T_d \approx T_N - T_{pcq} - T_{setup} \quad (3.1)$$

3.3.3 Discussion

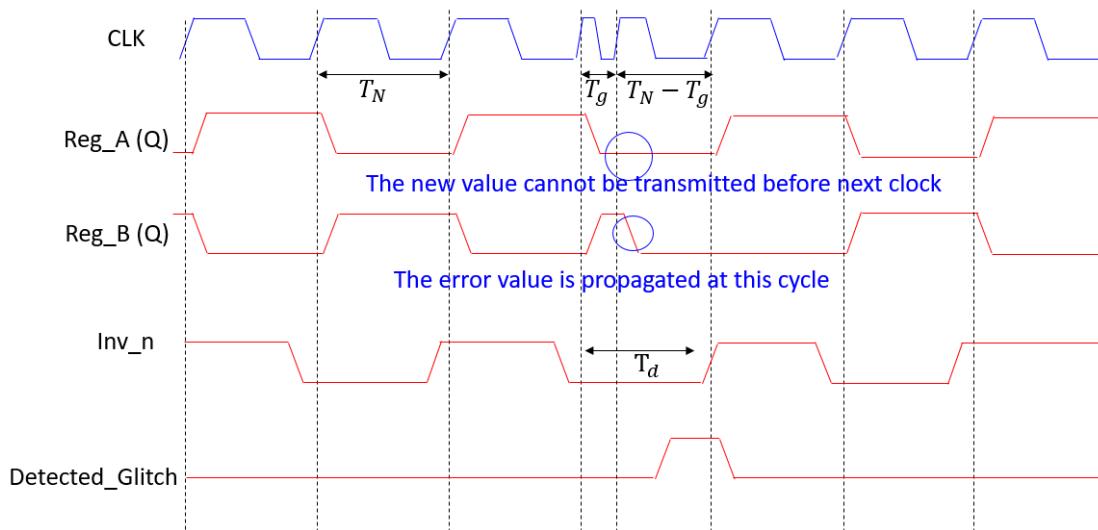
在正常情況下，Reg_A 的行為會如同 Figure 3.4 所示，因為根據式 3.1 的關係去設計反向延遲鏈，0-1 不停地震盪，Reg_B 則會與 Reg_A 相差一個時鐘週期。



T_d : propagation delay of the delay chain T_N : nominal clock period

Figure 3.4: Timing Diagram of Detector (Normal Situation)

若是發生了違反時序時，如 Figure 3.5 所示， T_g 的時鐘週期並不足以讓 Reg_A 完成反向的動作，因此在 T_N-T_g 的正緣觸發時，Reg_B 便門入錯誤的值，此時偵測訊號便能偵測到此種攻擊。



T_d : propagation delay of the delay chain T_N : nominal clock period T_g : period of the clock glitch

Figure 3.5: Timing Diagram of Detector (Timing Violation)



3.4 Recovery Mechanism

3.4.1 Core Idea of Recovery Mechanism

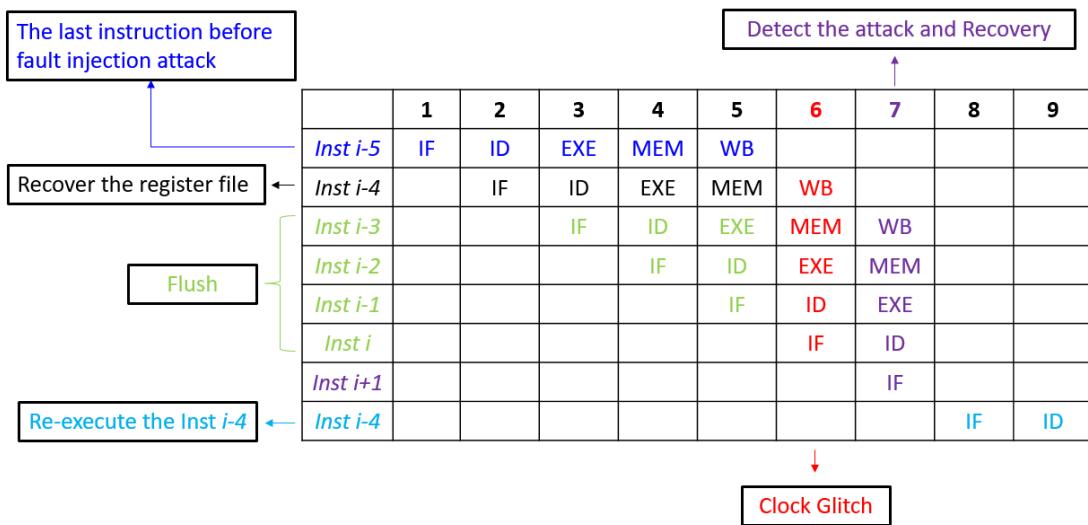


Figure 3.6: Overall Recovery Mechanism

為了要回復到一個全部都已知的狀態，最簡單的方法就是記錄下所有儲存元件，例如: MIPS32 裡面的暫存器堆、資料記憶體 (Data Memory)，以及所有的暫存器，但使用這種方式會造成大量的硬體浪費，相當於製造一個雙核心的處理器，卻永遠只使用其中一個核心。因此，我們透過只儲存必要之資料來達到此種效果。

現今的處理器為了將效率提高，多會使用管線的架構，而本論文所使用的便是管線化的處理器-MIPS32，而如同 MIPS32 這種精簡指令集計算機皆使用2.3所提及的載入-儲存架構，代表如果每個指令在執行時所需要的運算元必定來自暫存器堆，結果最後也必須寫回去暫存器堆或是資料記憶體，此種設計給了我們良好的切入點，若我們能掌握暫存器堆以及資料記憶體，我們便能決定處理器該從哪裡回復與繼續執行，並且獲得一樣的結果，而在3.3.1所提出的的偵測方法可以幫助我們在被攻擊的下一個週期偵測到並且開始回復。



在設計回復電路時，我們得假設程式計數器並不會受到故障注入攻擊的影響，由 Figure 3.1 所示，程式計數器的路徑相對其他電路是較短的，因此，若程式計數器也受到攻擊的影響，代表其他電路也沒辦法正常運作，無法達到攻擊者想要達到的攻擊。

3.4.2 Recovery Mechanism

回復的方法，將會分為下列四個部分來解釋。

- (1) Register File
- (2) Data Memory
- (3) Flush The Pipeline Register (Control Signal)
- (4) Program Counter

3.4.2.1 Register File

因為載入-儲存架構的設計，暫存器堆可以視為每個指令執行的起點；只要我們能夠回復到暫存記堆尚未受到故障注入攻擊的影響時，便能使後面的指令取得正確的運算元。若由管線的設計來看，如 Figure 3.6 所示，假設指令 i (Inst i) 在指令取得時受到故障注入攻擊，代表指令 $i-4$ (Inst $i-4$) 的資料寫回也受到影響，因此最佳的回復狀態便是指令 $i-3$ 完整執行完五個階段的時候。為了確保資料有辦法回復，我們將所有暫存器堆裡的暫存器，用相同大小與數量的影子暫存器 (Shadow Register) 備份，以時序圖來看，對於管線化設計的處理器來說，將暫存器堆回復上一個時鐘週期的資料，等同於要回復到指令 $i-5$ (Inst $i-5$) 的狀態，並且繼續往下執行。

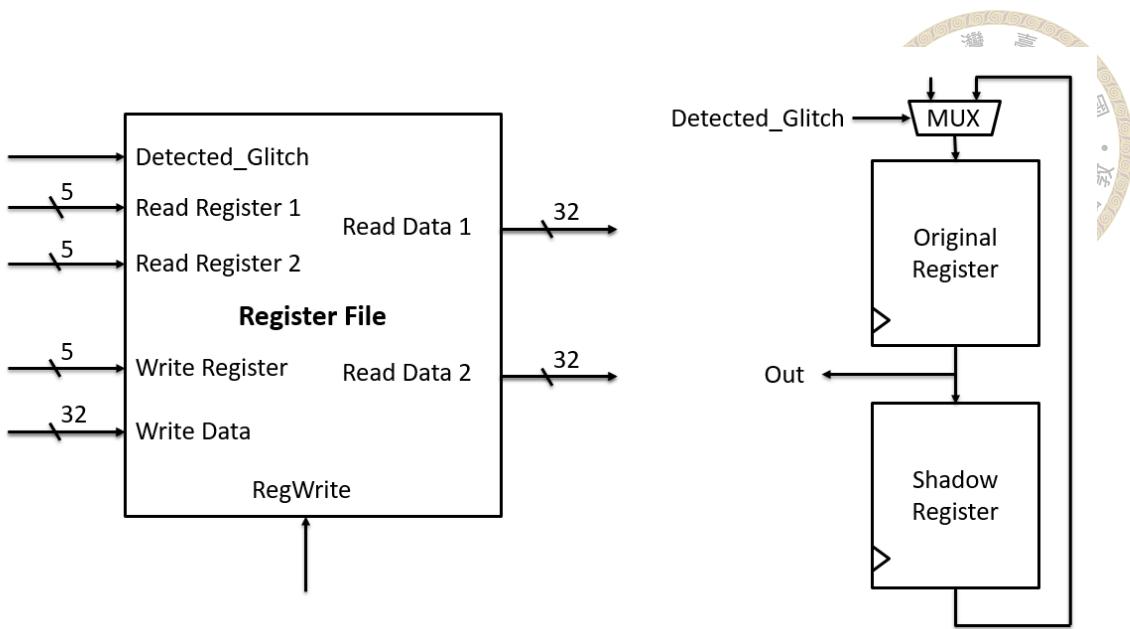


Figure 3.7: Register File with Shadow Register

接著我們在暫存器堆的輸入端接上由偵測器所送出的訊號，如 Figure 3.7 所示。偵測器並未偵測到攻擊時，影子暫存器會不斷地備份資料，一旦偵測器偵測到故障注入攻擊的發生，便會將影子暫存器的資料覆蓋原始暫存器的資料，透過此種方式便能確保下一個指令進來時，能取到未受到故障注入攻擊影響的資料。



3.4.2.2 Data Memory

資料記憶體經常使用靜態隨機存取記憶體實現，因此在設計資料記憶體的防禦機制時，我們考慮了此種記憶體在做寫入操作時，會有什麼樣的特性。在2.5有提及，當寫入靜態隨機記憶體，並且在最後真正要寫入資料時，將BL或BLB降低至低電位時最花時間，代表著如果電路發生故障注入攻擊，減少了可以運作的時間，只會造成資料未被真正寫入或者寫入錯誤的資料，並不會發生寫錯記憶體位置的事情發生，因此，本論文所考慮的情形，只有在發生故障注入攻擊時，未真正寫入記憶體或是寫錯資料。

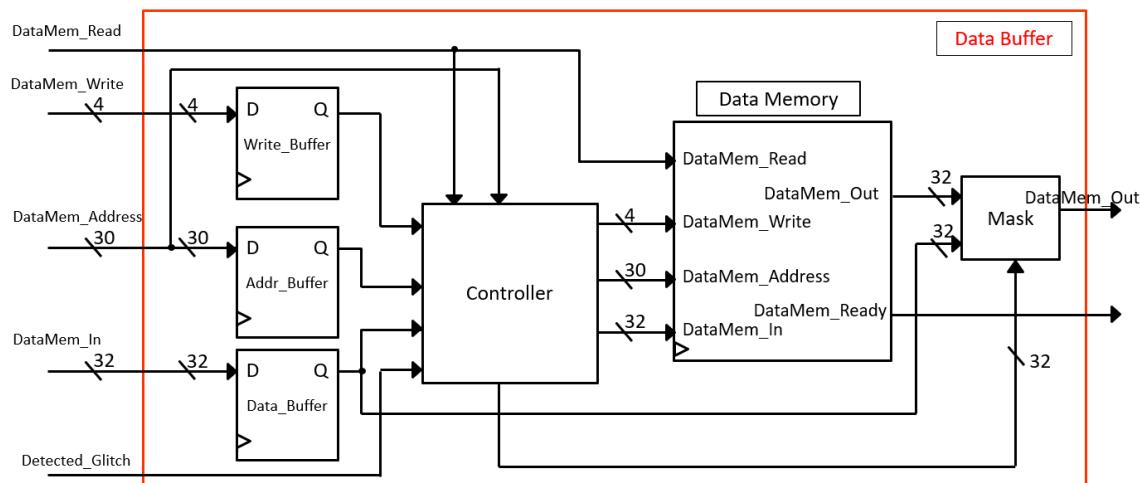


Figure 3.8: Data Memory with Data Buffer

因為備份整個資料記憶體是一個極為浪費且不切實際的方式，我們設計另外的架構用以確保在發生故障注入攻擊時，不會將受到影響的資料寫入記憶體。如Figure 3.8所示，Figure 3.8中的資料記憶體為一般的記憶體，其餘為額外設計的電路，可以看到每筆資料在真正寫入到資料記憶體之前，都會先被儲存在暫存器之中，等到下一個時鐘週期時，會先確保前一個週期未有故障注入攻擊的發生，資料才會被寫入記憶體當中；也就是說寫入記憶體的動作會被強制延後一個時鐘週期。但如果是要做資料讀取時，則不受到影響，會直接將資料轉送(Forward)到輸出端。然而因為這樣的設計改變了原來的結構，我們必須做額外的資料衝突

(Data Hazard) 分析。

以下分為寫後寫 (Write-After-Write)、讀後寫 (Write-After-Read)，以及寫後讀 (Read-After-Write) 做分析。



(1) 寫後寫 (WAW)

此種衝突是發生在後面的寫入指令優先於先前的寫入指令被執行，造成資料錯誤。在我們的架構下，因為每一個關於寫入記憶體的指令皆在管線化設計的同樣階段，代表後面才執行寫入的指令永遠不會優先於先前的寫入指令，所以此種衝突並不會發生。

(2) 讀後寫 (WAR)

此種衝突是發生在後面執行的寫入指令優先於先前的讀取指令被執行，造成資料被覆蓋，最終讀取到錯誤的資料。在我們的架構下，因為寫入的指令永遠晚於讀取指令被執行，所以此種衝突也不會發生。

(3) 寫後讀 (RAW)

此種衝突是發生在後面執行的讀取指令優先於先前的寫入指令被執行，造成讀取的指令讀取到錯誤的資料。此種衝突發生於讀寫位址一樣時，在本論文提出的架構下又可以分為下列四種。

(a) 儲存字 (Store Word) 接著執行載入字 (Load Word)

因為我們新增了資料暫存器的架構，如果這兩個指令剛好需要讀寫到一樣地址時，會造成資料衝突，因此我們在內部新增了控制器 (Controller) 去解決這個問題，如 Figure 3.8 所示，當發生此種情形時，代表資料還存在於暫存器中，控制器便會將資料轉送到輸出端，進而解決了這種案例。

(b) 儲存位元組 (Store Byte) 接著載入讀取字 (Load Word)



因為寫入時只寫入以位元組為單位的資料，但讀取時卻是以字為單位，代表有部分的資料還儲存在資料暫存器中未寫入；有些資料則儲存於資料記憶體內，此時可以利用了遮罩 (Mask) 的技術，將暫存器與資料記憶體的資料混和之後送到輸出端，確保處理器可以讀到正確的資料。

(c) 儲存字 (Store Word) 接著執行載入位元組 (Load Byte)

這種案例代表要讀取的資料為上一個寫入資料的部分，因此只需要透過遮罩以及控制器輸出對應的資料即可。

(d) 儲存位元組 (Store Byte) 接著執行載入位元組 (Load Byte)

這種案例較為單純，代表後面要讀取的資料在資料暫存器裡面，解決方式也是將暫存器的資料直接轉送到輸出端即可。

然而如果讀寫的地址不相同的話，因為本論文採用的記憶體為單埠隨機存取記憶體 (Single-Port SRAM)，所以會造成讀取時無法讀取到正確的資料。因此需要借助編譯器 (Compiler) 的輔助，若是發生寫後讀以及地址相同時，在兩個指令中間加入空操作指令 (NOP)，強迫避免發生此種案例，但也因為如此，加入過多的無操作指令會造成處理性能的降低 (Performance Degradation)。但因為本論文聚焦在開機過程的保護上，如果用一點性能的降低可以換來更好的安全防護，我們認為是值得的。

3.4.2.3 Flush The Pipelined Register

因為本論文提出的核心想法是將為執行完所有管線階段的指令視為被故障注入攻擊所影響，所以被影響到的指令所執行的結果都應該被捨棄。也就是由 Figure 3.6 可知，如果我們想要回復到指令 i-5(Inst i-5)，代表從指令 i-4(Inst i-4) 到指令 i(Inst i) 所執行的結果都應當被捨棄。使用的方法如同處理器在解決在做分支指令預測 (Branch Prediction) 時，預測錯誤要做的措施-清空管線 (Pipeline Flush)，用來確保錯誤執行的結果不會寫入到暫存器堆以及資料記憶體中，其中需要清空

的有暫存器堆及資料記憶體的寫入權限，還必須清空管線所暫存的指令。如此一來，之後回復到先前狀態並且繼續執行時，才不會讓錯誤的執行結果影響到。下列則是我們對於管線所做的處置。



- (1) MEM: 清空有關於記憶體讀寫的控制訊號以及將 MEM/WB 管線中的指令替代成 nop。
- (2) EX : 清空執行時所需的控制訊號以及將 EX/WB 管線中的指令替代成 nop。
- (3) ID : 清空剛解碼完所產生的控制訊號以及將 ID/EX 管線中的指令替代成 nop。
- (4) IF : 將 IF/ID 管線中的指令替代成 nop。

3.4.2.4 Program Counter

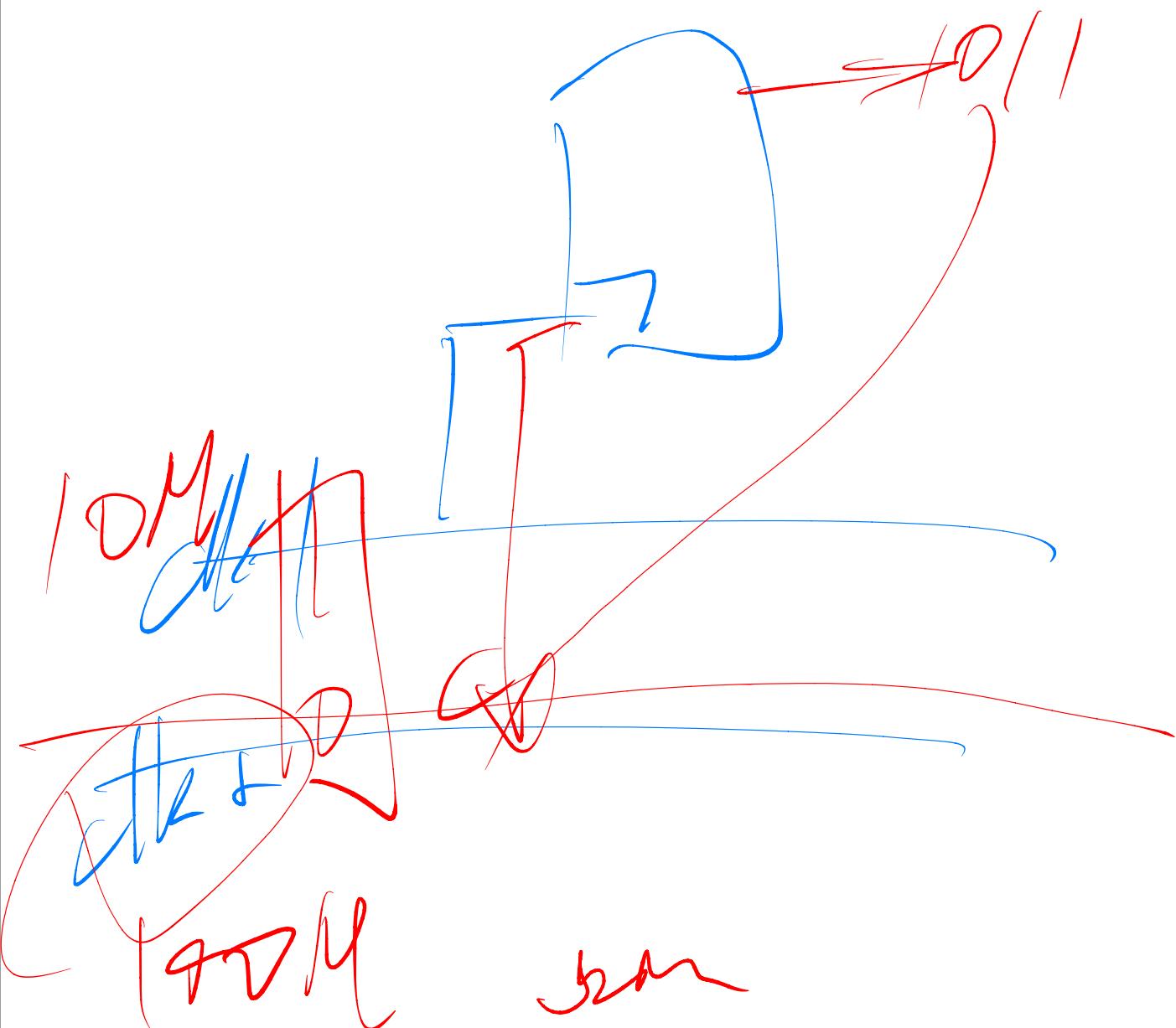
最後，等到我們確保資料已經都回復到已知的狀態，並且也將錯誤執行的結果都清空後，就能將程式計數器回復到預先儲存的位址，也就是在 Figure 3.1 的 Recoverable PC 所負責的，其中要注意的重點是，因為我們要回復的是以指令為單位，而不是單純地址的回溯，如果在回復時若是遇到已經進入處理器的指令之間存在著資料相依 (Data Dependency)，因此處理器被強制停頓 (Stall) 的情形時，要避免儲存到錯誤的位址，不能重複儲存一樣的指令位址。

而我們在設計儲存程式計數器時，必須把握一個原則，就是將回復的電路簡單化，太過於複雜的儲存方式也會造成回復電路也會違反時序，導致在回復時無法正確回復。



3.4.3 Discussion

本論文所提出的方法在實現時，盡可能地將備份的方式做到最簡單，以確保回復電路不會受到故障注入攻擊的影響。而此方法也適用於開機的過程，可以有效且主動去偵測攻擊的發生，並且回復到未被攻擊的狀態，也能使得1.2所提及安全開機的威脅降低。





第四章 Experimental Result

4.1 Experiment Setup

在本論文中，在3所提出的架構以及想法將每個元件實現於場域可程式化邏輯閘陣列 (Field Programmable Gate Array) 中做概念驗證。本論文所使用的開發板是 Xilinx Zedboard，配備 Xilinx Zynq®-7000 All Programmable SoC，其中包括 ARM Cortex-A9 以及 FPGA 的資源，因為本論文所使用的資源皆是場域可程式化邏輯閘陣列，所以附上此開發板所提供的規格說明。

- (1) Clock Source: 100MHz on board oscillator
- (2) Programmable logic cells: 85K
- (3) Look-up tables (LUTs): 53200
- (4) Flip-Flop: 106400
- (5) DSP slice: 220
- (6) Block RAM: 4.9Mb

4.2 Experiment Architecture

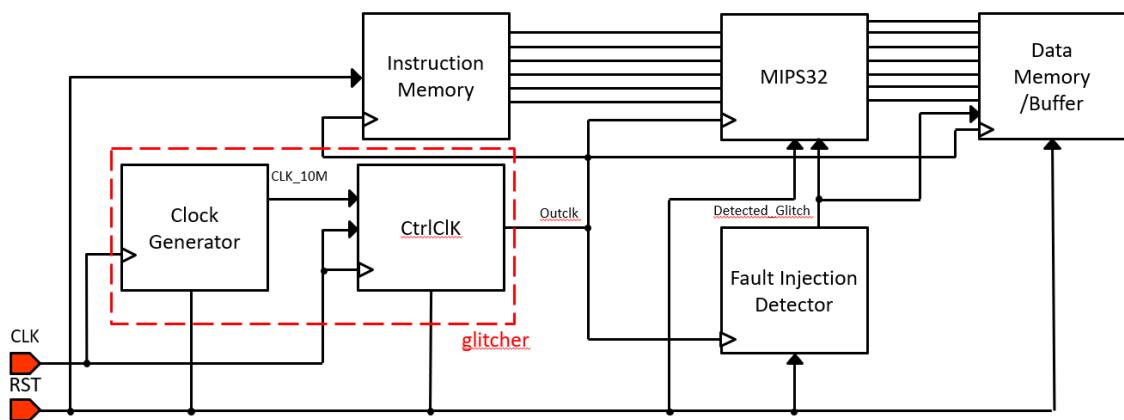
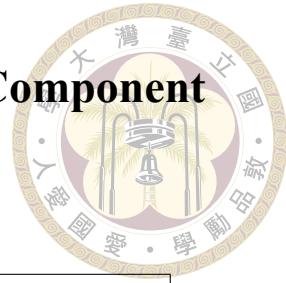


Figure 4.1: Experiment Architecture

4.2.1 I/O of Each Component

- (1) Clock Generator: CLK, RST, CLK_10M (Hz)
- (2) CtrlClk: CLK, RST, Outclk
- (3) Fault injection detector: CLK, RST, Detected_Glitch
- (4) MIPS32 : CLK, RST, Interrupt (4), NMI, InstMem_In (32), InstMem_Address (30),
InstMem_In (32), InstMem_Ready, DataMem_In (32), DataMem_Ready, DataMem_Read,
DataMem_Write (4), DataMem_Address (30), DataMem_Out (32), Detected_Glitch
- (5) Instruction memory: CLK, RST, InstMem_In (32), InstMem_Address (30), Inst-
Mem_In (32), InstMem_Ready
- (6) Data memory/buffer: CLK, RST, DataMem_In (32), DataMem_Ready, DataMem_Read,
DataMem_Write (4), DataMem_Address (30), DataMem_Out (32), Detected_Glitch



4.3 Utilization of FPGA Resource of Each Component

此結果皆不包含整合邏輯分析儀 (Integrated Logic Analyzer)

	LUT	Flip-Flop	BRAM	DSP Slices
MIPS32 w/o Recovery Circuit	3732	1858	0	8
MIPS32 w/ Recovery Circuit	4080	2051	0	8
Detector w/ delay chain	152	2	0	0
Data Memory w/ Buffer	3622	4258	0	0
Data Memory w/o Buffer	3404	4128	0	0

Table 4.1: Utilization of FPGA Resource of Each Component

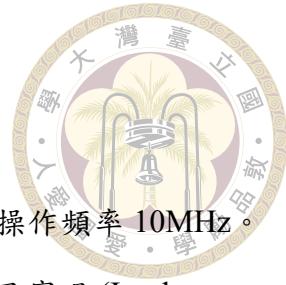
4.4 Comparison with Previous Work

先前有提到的研究，是對於特定要保護的電路做複製，用來確保可以偵測出是否被攻擊而算出錯誤的結果，例如 [11]，將整個加密演算法複製，因此需要花費較多額外的硬體，而 [9] 則是與本論文相似，提出一個直接可以偵測此攻擊的方法，但他提出的方法也需延遲鏈的輔助，以及額外的硬體。

	Platform	Hardware Overhead
Proposed	Xilinx Zedboard	2 個暫存器以及反向延遲鏈
[9]	Xilinx Virtex-7 FPGA	0.06%，此方法需要額外的硬體及延遲鏈輔助
[11]	ASIC	104%

Table 4.2: Comparison with Previous Work for Detection

根據 [13] 的做法，需要在受到攻擊時觸發例外模式，進入回復狀態，而需要加入額外的指令，需要多耗費 0-5.4% 的執行時間，雖說被攻擊並不是常態作為，但如果持續被攻擊，也會造成較多負擔，而此方法也被限制在軟體並沒有受到竊改的情況下，較易被攻破。



4.5 Fault Injection Detector

根據3.3.2所述，此偵測器必須的反向延遲鍾必須要趨近於操作頻率 10MHz。

這裡必須要注意的點在於因為開發軟體會在合成 (Synthesis) 以及實現 (Implementation) 階段時去優化整個電路，時常會將此反向延遲鍾判斷為可優化路徑，因此我們利用了 (`* Dont_touch = "true"`) 的特殊語法告知開發軟體不要優化此電路，才能成功實現了偵測器的反向延遲鍾，而從 Figure 4.2可見，在發生2.2.1所設計的短時脈衝波形干擾產生器成功產生了短時脈衝波形干擾，本論文所提出的偵測器架構能有效的偵測違反時序的發生。

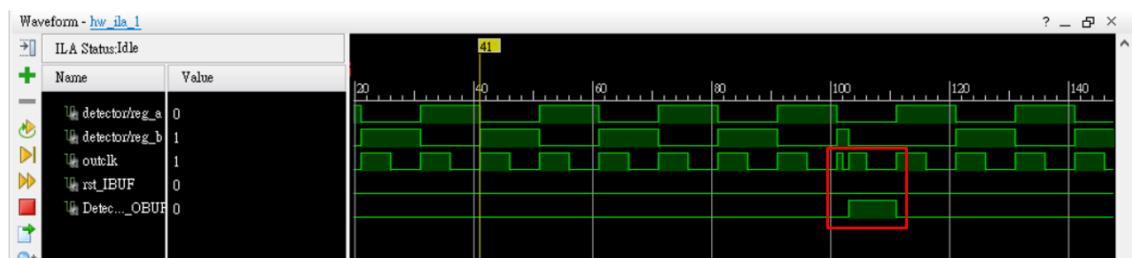


Figure 4.2: Waveform of Detector

4.6 Instruction Skipping

由2.6.2所提及，在發生故障注入攻擊時，因為違反時序的發生，會造成處理器來不及完成指令取得的動作，但由於在處理器中，程式計數器去計算該去哪個地址取得的動作花費較少的時間，因此造成指令被跳過的情形。如 Figure 4.3所示，短時脈衝波形干擾的產生使程式計數器計算完下一個指令的位址，但卻因為違反時序的問題，處理器並未真正去取得指令。因此，可以證明故障注入攻擊會如同2.6.2所述，成功跳過一個指令。

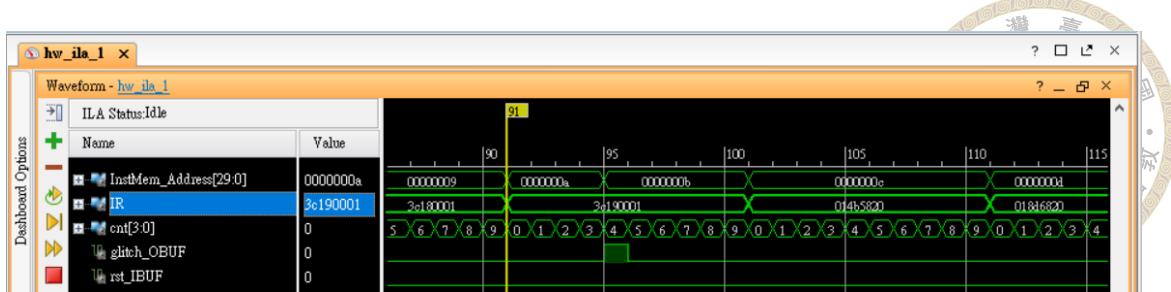


Figure 4.3: Waveform of Instruction Skipping

4.7 Recovery Mechanism (Simulation)

回復時我們使用了概念驗證及模擬的方式來證明管線化 MIPS32 如同3所述的方法的可行性，在此實驗中，我們對整個系統攻擊了 2 次，也就是說會觸發 2 次的回復。以下為此實驗的設定與分析參數。

- (1) 時鐘週期: 100ns
- (2) 需要回復的次數: 2
- (3) 每次回復需要花費的週期: ≥ 6 個週期 (包含停頓的次數)
- (4) 回復時所增加的停頓: 2

圖4.4所代表的是，未受到攻擊時，完整執行完所預先設定的程式所花費的時間為 2400.1ns，而如 Figure 4.5所示，所需要的時間為 3800.1ns，因此可以計算出回復時所需要的時間週期應是

$$\frac{\frac{\text{total time w/ recovery} - \text{total time w/o recovery}}{\text{clock period}} - \text{number of stall}}{\text{number of recovery requirement}} \quad (4.1)$$

由式 4.1可得知，最終可得回復時每次所需的時間週期與3所設計的一樣，並且可以由 Figure 4.5的結果來看，所運算的結果與未受到攻擊前相同。

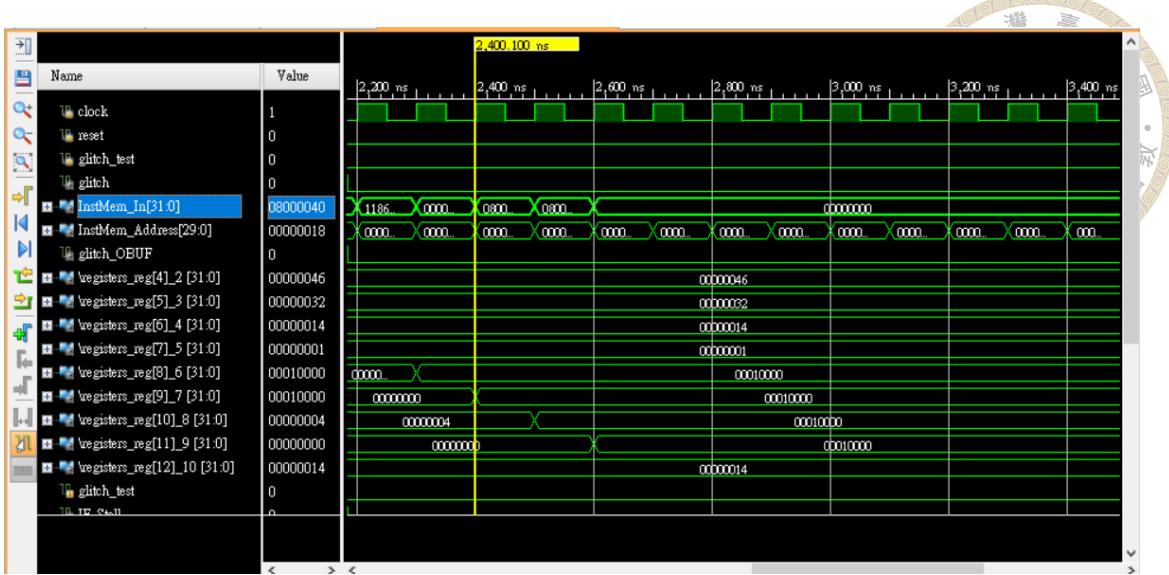


Figure 4.4: Waveform of Recovery Mechanism without Recovery Requirements

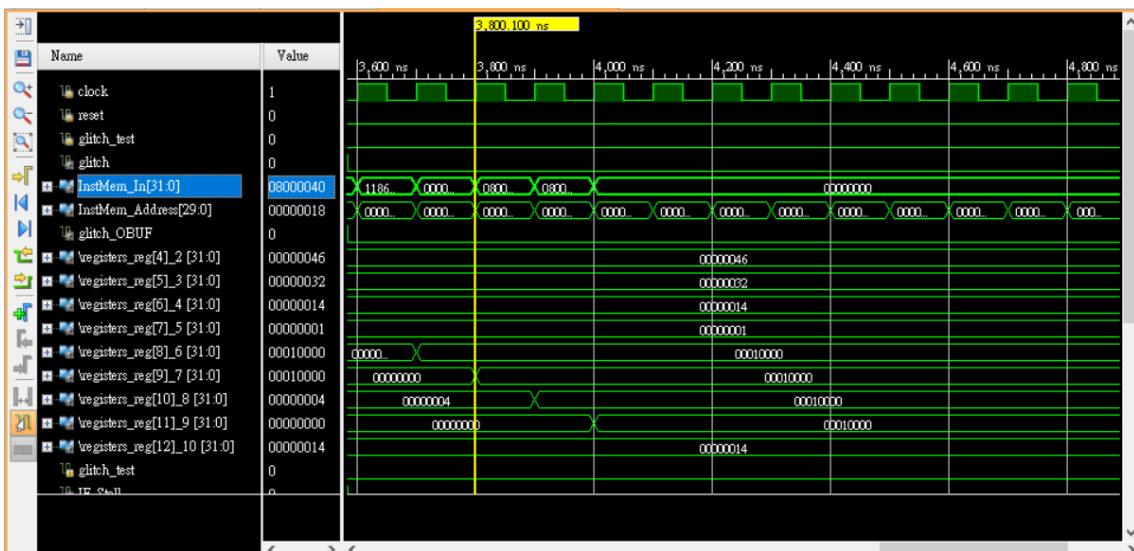


Figure 4.5: Waveform of Recovery Mechanism with 2 Recovery Requirement

由 Figure 4.6 所示，當回復訊號被觸發時，會將 Flush 的訊號設置為 1，藉此來清空管線裡的錯誤執行結果以及不應再繼續執行的指令。而從 Figure 4.7 所示，資料暫存器堆的暫存器會因為回復訊號被觸發，而將備份在影子暫存器的資料去回復處理器至過去已知的狀態，最後便能去將程式計數器回復到預先儲存的指令位址，如 Figurer 4.8 所示。



Figure 4.6: Flush The Pipeline

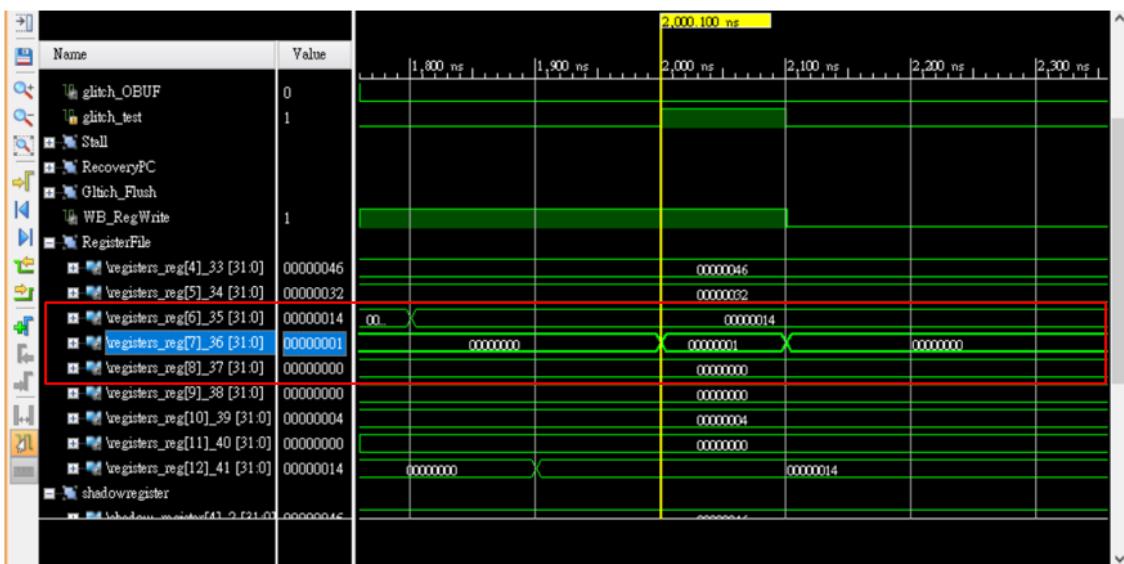


Figure 4.7: Recover The Register File Using Shadow Register

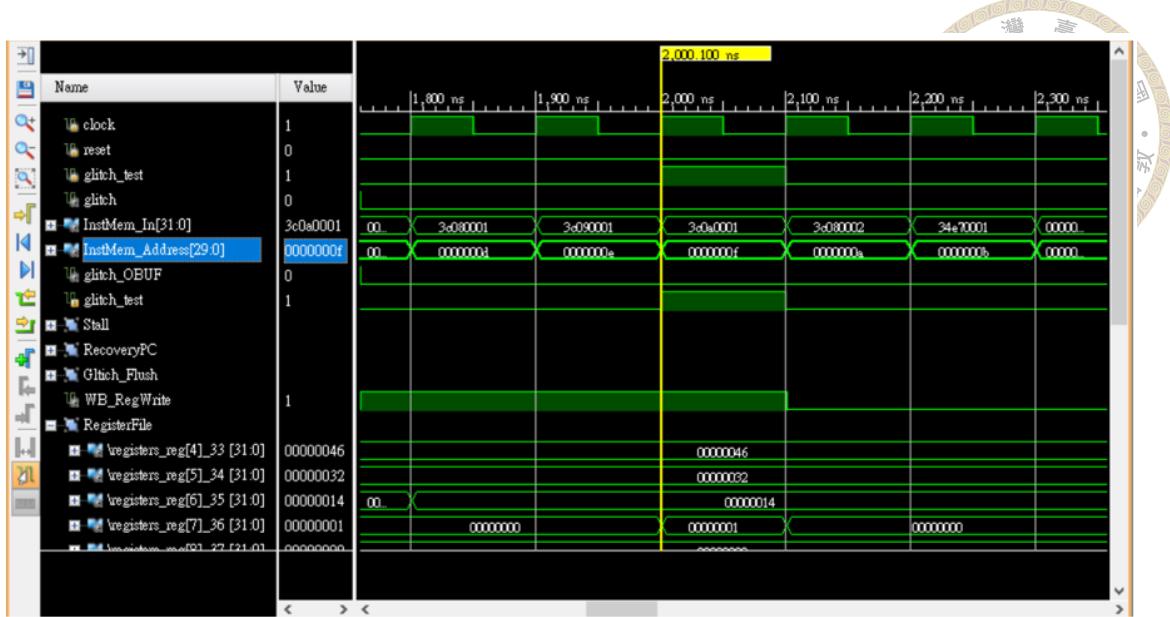


Figure 4.8: Control The Program Counter



4.8 Summary

在實驗結果中，我們成功實現了短時脈衝波形干擾器去達到故障注入攻擊的效果，有效的跳過了一條指令，影響了最終的運算結果。之後我們運用了本論文所提出的故障注入攻擊偵測器去主動偵測，一旦發生了違反時序的故障注入攻擊，此偵測器便能得知並且傳遞給處理器去做回復的動作。

在前述的模擬結果中，我們也分析了做回復時會產生的效果，以及每次回復會消耗的時間週期，由於我們回復時是以指令為單位，因此若往前回推的指令中有產生停頓的情況發生，回復時也會再一次的重現。雖然回復時會有額外的時間被浪費，但卻能因此確保處理器能正確且正常的執行，以達到當初設計者所設定的結果。



第五章 Conclusion and Future Work

5.1 Future Work

目前本論文所實現的架構非常適用於安全開機時，因為在開機的過程中不需要運行一些過於複雜的程式，因此在回復時，考慮的切入點相對於開機之後較為簡單。開機後的行為模式則需要再多做更多的分析，例如本論文所提出的方法，在回復時並不考慮中斷 (Interrupt) 的發生，原因是在開機的時候，中斷通常會暫時先被去能 (Disable)，以確保在開機的階段並不被其他事件所打擾，因此未來如果能加上對於產生中斷時所需做出的反應，應能漸漸推廣到開機之後的保護。

本論文所實現的方法也暫不適用於具備超純量 (Super Scalar) 功能的處理器，因為對於市面上來說，大多數的嵌入式系統所配備的處理器為了節省成本並不會使用太複雜的架構，以 ARM 專屬嵌入式系統或可攜式裝置所配備的 Cortex-M 處理器來說，大多數也都不具備此項功能，因此未來如果能加以分析，此技術的應用層面也會隨之更加寬廣。

在違反時序的研究上，有各式各樣的技術，像是 DVFS 的處理技術以及延遲鏈的設計或如同在1.3所提及的技術，可以用來偵測，但是否能夠適用於防範此種攻擊，需要做更多的分析，由此一來也能找到更適合偵測基於違反時序的故障注入攻擊。

本論文在實現回復機制時，此方法限制在假設程式計數器為最短路徑，因此

在被故障注入攻擊時，不會影響到其功能，然而，若處理器的程式計數器非為最短路徑，此方法也會隨之失效，在未來，可以朝此方向做研究。

本論文所實現之回復機制，目前只有以模擬的方式驗證可行性，未來需要再以實際的硬體做完整性的分析。





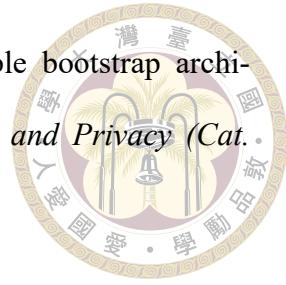
5.2 Conclusion

處理器幾乎是日常所有裝置一定會配備的功能，但卻因為故障注入攻擊簡單實現且低成本的特性下，承受著巨大的威脅。因此，本論文實現了產生短時脈衝波形干擾器的故障注入攻擊去輔助開發防守此種攻擊的方法，並且發展了一個輕量化且有效偵測此攻擊的偵測器，最後提出了一個回復機制可以使處理器在遇到此種攻擊時，能夠回復至先前已正確執行的最後狀態並重新繼續執行的機制。如此一來，攻擊者便無法輕易跳過所設計的安全機制。



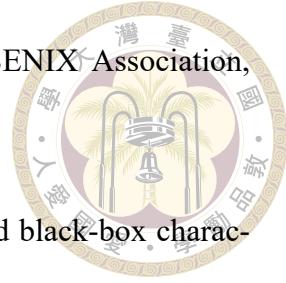
參考文獻

- [1] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits*. USA: Prentice Hall Press, 3rd ed., 2008.
- [2] F. E. Potestad-Ordóñez, C. J. Jiménez-Fernández, and M. Valencia-Barrero, “Vulnerability analysis of trivium fpga implementations,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 12, pp. 3380–3389, 2017.
- [3] Z. Kazemi, D. Hely, M. Fazeli, and V. Beroulli, “A review on evaluation and configuration of fault injection attack instruments to design attack resistant mcu-based iot applications,” *Electronics*, vol. 9, no. 7, 2020.
- [4] 维基百科, “Mips 架構 — 维基百科, 自由的百科全書,” 2021. [Online; fetch at 0117, 2022].
- [5] 维基百科, “靜態隨機存取記憶體 — 维基百科, 自由的百科全書,” 2021. [Online; fetch at 0117, 2022].
- [6] B. Yuce, P. Schaumont, and M. Witteman, “Fault attacks on secure embedded software: Threats, design, and evaluation,” *Journal of Hardware and Systems Security*, vol. 2, p. 111–130, May 2018.
- [7] M. Gasser, A. Goldstein, C. Kaufman, and B. Lampson, “The digital distributed system security architecture,” *National Computer Security Conf., NIST/NCSC, Baltimore*, vol. 12, pp. 305–319, Oct. 1989.



- [8] W. Arbaugh, D. Farber, and J. Smith, “A secure and reliable bootstrap architecture,” in *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No.97CB36097)*, pp. 65–71, 1997.
- [9] M. Zhang and Q. Liu, “A digital and lightweight delay-based detector against fault injection attacks,” in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2021.
- [10] P. Maistri and R. Leveugle, “Double-data-rate computation as a countermeasure against fault analysis,” *IEEE Transactions on Computers*, vol. 57, no. 11, pp. 1528–1539, 2008.
- [11] M. Doulcier-Verdier, J.-M. Dutertre, J. Fournier, J.-B. Rigaud, B. Robisson, and A. Tria, “A side-channel and fault-attack resistant aes circuit working on duplicated complemented values,” in *2011 IEEE International Solid-State Circuits Conference*, pp. 274–276, 2011.
- [12] P. Rathnala, T. Wilmhurst, and A. H. Kharaz, “Timing error detection and correction for power efficiency: an aggressive scaling approach,” *IET Circuits, Devices & Systems*, 2018.
- [13] B. Yuce, C. Deshpande, M. Ghodrati, A. Bendre, L. Nazhandali, and P. Schaumont, “A secure exception mode for fault-attack-resistant processing,” *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 3, pp. 388–401, 2019.
- [14] H. M. D. Kabir and M. Chan, “Sram precharge system for reducing write power,” *HKIE Transactions*, vol. 22, no. 1, pp. 1–8, 2015.
- [15] A. Tang, S. Sethumadhavan, and S. Stolfo, “CLKSCREW: Exposing the perils of Security-Oblivious energy management,” in *26th USENIX Security Symposium*

(*USENIX Security 17*), (Vancouver, BC), pp. 1057–1074, USENIX Association, Aug. 2017.



- [16] J. Balasch, B. Gierlich, and I. Verbauwhede, “An in-depth and black-box characterization of the effects of clock glitches on 8-bit mcus,” in *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 105–114, 2011.
- [17] T. Korak and M. Hoefler, “On the effects of clock and power supply tampering on two microcontroller platforms,” in *2014 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 8–17, 2014.
- [18] N. Timmers, A. Spruyt, and M. Witteman, “Controlling pc on arm using fault injection,” in *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pp. 25–35, 2016.
- [19] A. Dehbaoui, J.-M. Dutertre, B. Robisson, and A. Tria, “Electromagnetic transient faults injection on a hardware and a software implementations of aes,” in *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 7–15, 2012.
- [20] M. Hutter and J.-M. Schmidt, “The temperature side-channel and heating fault attacks,” in *Smart Card Research and Advanced Applications – CARDIS 2013, 12th International Conference, Berlin, Germany, November 27-29* (P. Rohatgi and A. Francillon, eds.), Springer, 2013.
- [21] A. Cui and R. Housley, “BADFET: Defeating modern secure boot using second-order pulsed electromagnetic fault injection,” in *Workshop on Offensive Technologies (WOOT)*, 2017.