

國立中央大學

資訊工程學系
碩士論文

ReActNet-XGBoost 硬體加速器設計與實現
——資源受限場景的應用探索

ReActNet-XGBoost Hardware Accelerator:
Design, Implementation, and Application
Exploration in Resource-Constrained
Scenarios

研究生：蕭如珊

指導教授：陳慶瀚博士

中華民國一一四年一月

國立中央大學圖書館學位論文授權書

填單日期：__114__/_01__/_21__

2019.9 版

授權人姓名	蕭如珊	學 號	111522162
系所名稱	資訊工程學系	學位類別	<input checked="" type="checkbox"/> 碩士 <input type="checkbox"/> 博士
論文名稱	ReActNet-XGBoost 硬體加速器設計與實現——資源受限場景的應用探索	指導教授	陳慶瀚教授

學位論文網路公開授權

授權本人撰寫之學位論文全文電子檔：

- 在「國立中央大學圖書館博碩士論文系統」.
☐ 同意立即網路公開
☒ 同意 於西元 2030 年 01 月 21 日網路公開
☐ 不同意網路公開，原因是：_____
- 在國家圖書館「臺灣博碩士論文知識加值系統」
☐ 同意立即網路公開
☒ 同意 於西元 2030 年 01 月 21 日網路公開
☐ 不同意網路公開，原因是：_____

依著作權法規定，非專屬、無償授權國立中央大學、台灣聯合大學系統與國家圖書館，不限地域、時間與次數，以文件、錄影帶、錄音帶、光碟、微縮、數位化或其他方式將上列授權標的基於非營利目的進行重製。

學位論文紙本延後公開申請 (紙本學位論文立即公開者此欄免填)

本人撰寫之學位論文紙本因以下原因將延後公開

- 延後原因
☐ 已申請專利並檢附證明，專利申請案號：
☒ 準備以上列論文投稿期刊
☐ 涉國家機密
☐ 依法不得提供，請說明：_____

• 公開日期：西元 2030 年 01 月 21 日

※繳交教務處註冊組之紙本論文(送繳國家圖書館)若不立即公開，請加填「國家圖書館學位論文延後公開申請書」

研究生簽名： 蕭如珊

指導教授簽名： 陳慶瀚

*本授權書請完整填寫並親筆簽名後，裝訂於論文封面之次頁。

國家圖書館學位論文延後公開申請書

Application for Embargo of Thesis/Dissertation

申請日期：民國 114 年 01 月 21 日

Application Date: 2025 / 01 / 21 (YYYY/MM/DD)

申請人姓名 Applicant Name	蕭如珊	學位類別 Graduate Degree	<input checked="" type="checkbox"/> 碩士 Master <input type="checkbox"/> 博士 Doctor	畢業年月 Graduation Date (YYYY/MM)	民國 114 年 01 月 2025 / 01
學校名稱 University	國立中央大學		系所名稱 School/Department	資訊工程學系	
論文名稱 Thesis / Dissertation Title	ReActNet-XGBoost 硬體加速器設計與實現——資源受限場景的應用探索				
延後公開原因 Reason for embargo	<input type="checkbox"/> 涉及機密 Contains information pertaining to the secret. <input type="checkbox"/> 專利事項，申請案號： Filing for patent registration. Registration number: <input checked="" type="checkbox"/> 依法不得提供，請說明：論文投稿 Withheld according to the law. Please specify.				
申請項目 Options	<input checked="" type="checkbox"/> 紙本論文延後公開 Delay public access to the printed copies of my thesis, but leave the online bibliographic record open to the public.			<input type="checkbox"/> 書目資料延後公開 Delay public access to online bibliographic record of my thesis.	
公開日期 Delayed Until	民國 119 年 01 月 21 日 2030 / 01 / 21 (YYYY/MM/DD)			<input type="checkbox"/> 不公開 Prohibited from public access.	

申請人簽名：

Applicant Signature:

蕭如珊

指導教授簽名：

Advisor Signature:

陳慶瀚

學校認定/審議單位章戳：

Seal of the Authorization Institute:



【說明】

- 依教育部107年12月5日臺教高(二)字第1070210758號函及109年3月13日臺教高通字第1090027810號函，請據實填寫本申請書並檢附由學校認定或審議單位認定之證明文件，經由學校向本館提出申請，無認定或審議單位章戳者退回學校處理。
- 論文尚未送交國家圖書館，請於提送論文時，夾附親筆簽名申請書1份。
- 論文已送達國家圖書館，請將親筆簽名申請書一式2份掛號郵寄10001臺北市中山南路20號國家圖書館館藏發展及書目管理組，並於信封註明「學位論文延後公開申請書」。
- 本館保存之學位論文依學位授予法應提供公眾於館內閱覽紙本，或透過獨立設備讀取電子資料檔，二者依表單填寫日期公開。

【Notes】

- Please fill in all blanks and attach the certification documents approved by the university and apply through the university. The application form will not be accepted for processing until all information, signatures, and stamps are included.
- If the thesis or dissertation is not yet submitted to the NCL, please attach the signed application form to the thesis or dissertation.
- If the thesis or dissertation has been submitted to the NCL, please send a registered letter with 2 copies of the signed application form attached. The letter should be addressed to "Collection Development Division", National Central Library with a note in the envelope indicating "Application for delay of public release" to the following address. No.20, Zhongshan S. Rd., Zhongzheng District, Taipei City 10001, Taiwan (R.O.C.)
- The delayed date of printed copies and the independent viewing equipment will synchronize.

(申請者免填，以下由國家圖書館填寫 For Internal Use)

承辦單位_館藏組： 日期/處理狀況：

典藏地： 登錄號： 索書號：

會辦單位_知服組： 日期： ☐ 移送並註記，原上架日期：

論文系統： 日期：

國立中央大學碩士班研究生
論文指導教授推薦書

資訊工程學系碩士班 學系/研究所 蕭如珊 研究生

所提之論文 ReActNet-XGBoost 硬體加速器設計與實現——資源
受限場景的應用探索

係由本人指導撰述，同意提付審查。

指導教授

陳慶瑜

(簽章)

113 年 12 月 23 日

國立中央大學碩士班研究生
論文口試委員審定書

資訊工程學系碩士班 學系/研究所 蕭如珊 研究生

所提之論文 ReActNet-XGBoost 硬體加速器設計與實現——資源
受限場景的應用探索

經由委員會審議，認定符合碩士資格標準。

學位考試委員會召集人

委

員

謝昇寧

陳慶瀚

李明義

中 華 民 國 114 年 1 月 14 日

摘要

隨著工業 4.0 和邊緣運算技術的快速發展，智慧醫療與工業監控等場景對即時資料處理和智慧化分析提出了高效能、低功耗與資源優化的嚴苛需求。然而，傳統卷積神經網路 (CNN) 因其高運算量與記憶體需求，難以滿足資源受限邊緣裝置的應用場景。為解決此一挑戰，本研究提出了一種結合 ReActNet 架構與 XGBoost 分類器的硬體加速器，專注於一維時間序列資料的處理。

本設計以 1D CNN 的輕量化特性為基礎，通過二值卷積技術顯著降低運算負擔，嘗試取得低功耗與高準確度之間的平衡，並且為智慧醫療中的生理訊號監測以及工業監控中的設備異常檢測提供了創新的解決方案，展現出應用於資源受限邊緣裝置的潛力。

Abstract

With the rapid advancement of Industry 4.0 and edge computing technologies, applications in smart healthcare and industrial monitoring demand highly efficient, low-power, and resource-optimized solutions for real-time data processing and intelligent analysis. However, traditional convolutional neural networks (CNNs), with their significant computational and memory requirements, struggle to meet the constraints of resource-limited edge devices. Addressing these challenges, this study introduces a hardware accelerator combining the ReActNet architecture with an XGBoost classifier, specifically designed for processing one-dimensional time-series data.

Built upon the lightweight characteristics of 1D CNNs, the proposed design leverages binary convolution techniques to significantly reduce computational overhead, aiming to achieve an optimal balance between low power consumption and high accuracy. This innovative approach offers promising solutions for real-time physiological signal monitoring in smart healthcare and anomaly detection in industrial monitoring, demonstrating substantial potential for deployment in resource-constrained edge devices.

致謝

第一次踏進中央校園拜訪教授的那天，恍若還在不久以前，才過去短短一個月。然而，時光匆匆流轉，轉眼間已經到了研究告一段落，準備離開校園的時刻。在這兩年的期間內，有幸能跟隨陳慶瀚教授的步伐，對這個領域進行摸索、探討與研究。教授的見解總是獨到而精闢，尤其在產業趨勢的掌握上非凡的敏銳度，總是能前瞻性地洞悉技術發展的潛力，引導我們將研究方向與產業需求緊密結合。在此衷心感謝教授的悉心指導與無私分享，這段學習歷程將成為學生未來持續精進的寶貴基石。

同時，誠摯感謝口試委員謝昇憲教授與林明義教授，自最初寄送邀請函開始，便從與兩位教授的信件往來中深刻感受到教授們的親切與專業。在口試過程中，教授傾囊相授，提供了許多寶貴的建議，使我在研究內容的描述上更為嚴謹與完善，並收穫豐富的知識與啟發，進一步拓展對研究領域的視野與思維。衷心感謝兩位教授的無私付出與指導。

在身為研究生的這段日子裡，承蒙許多人的幫助，才能夠順利進行各項研究工作。首先，誠摯感謝博士班的冠維學長、銘芳學長與志豪學長，每當我在研究過程遇到問題時，學長們總是願意暫時放下手邊的工作，耐心與我共同討論研究方向，並協助思考解決方法。尤其感謝座位在同一間實驗室的志豪學長，從踏入實驗室的第一天起，就受到學長許多照顧，無論在學術研究或生活處事上，都毫無保留地分享寶貴經驗並提供指導。從學長身上，我不僅學習到積極進取的生活態度，更見識到深厚的專業實力，成為我努力精進、砥礪自我的榜樣。

接著，我想感謝從最初加入實驗室就給予我諸多照顧與指導的學姐、學長們。衷心感謝鈺盈學姐、珮慈學姐、之宇學長、勁為學長、承學學長、星宇學長、治嘉學長和佾均學長，無論在課業上、研究上，抑或生活上，都不吝分享自身的寶

貴經驗，並在我遇到困難時提供即時的協助與鼓勵。在這段日子中，能有各位學長姐的陪伴與支持，真的很幸運！

此外，我也要感謝一路以來並肩作戰的夥伴們——忻柔、瑞庭、適杰、岳峻、祺文、至偉和柏任。在這段求學過程中，有緣與你們共同面對課業與研究上的挑戰，並在閒暇時間一同歡笑放鬆，無疑是我研究生涯中最珍貴的回憶。謝謝你們在繁忙的研生活中，帶來歡笑與溫暖，使這段旅程更加充實而有意義，也謝謝你們在我研究最忙碌的收尾階段，總會時不時跑來關心我，並協助分擔助教課程的許多事項，讓我能更專注於研究上。雖然我們一起度過的這些日子，在人生的這個階段中留下了酸、甜、苦、辣各種風味的紀錄，並不是每一天都充滿順遂與喜悅，但我相信日後回想起來，絕對會是很充實、富有幸福感的篇章！

除了學術上的夥伴，我想特別感謝一路陪伴並支持我的好友們。儘管我們在不同的專業領域努力，沒辦法深入討論技術，仍然在精神上給予我莫大的支持，並無私地分享研究的方法、工具，以及在撰寫論文時需注意的細節。能夠在彼此的領域各自耕耘，但攜手成長、互相鼓勵並一同向前，實屬人生難得的幸運！

最後，最深的感謝要獻給我的家族。感謝所有給予關心、一路陪伴並支持我的長輩、親戚，以及我最親愛的父母，因為有你們無微不至的支持與鼓勵，我才能夠如此無所畏懼地追尋學問，勇敢面對每一個挑戰。儘管在研究過程中，總會遇到許多挫折，也有無數感到疲累不堪的時刻，但每當想到我擁有這麼堅強的後盾，以及你們滿溢的關愛，總是能再次打起精神，繼續努力向前。這份深厚的情感與溫暖，我將永遠銘記於心。

在此，誠摯感謝所有關心並幫助過我的貴人，衷心祝願大家都能平安順遂、心想事成，並在未來的日子裡持續實現夢想，愉快地度過每一天。

目錄

摘要.....	i
Abstract.....	ii
致謝.....	iii
目錄.....	v
圖目錄.....	viii
表目錄.....	x
第一章、緒論.....	1
1.1 研究背景.....	1
1.2 研究目標.....	3
1.3 論文架構.....	4
第二章、技術回顧.....	6
2.1 二值卷積神經網路.....	6
2.1.1 Binarized Neural Network	6
2.1.2 XNOR-Net	8
2.1.3 ReActNet.....	11
2.2 XGBoost	15
2.2.1 多決策樹硬體加速器.....	18
2.3 MIAT 系統設計方法論.....	21
2.3.1 IDEF0 階層式模組化設計.....	22

2.3.2	GRAFCET 離散事件建模	24
2.3.3	硬體高階合成.....	27
第三章、ReActNet-XGBoost 硬體加速器系統設計		30
3.1	系統架構.....	30
3.1.1	IDEF0 階層式模組化設計	31
3.1.2	GRAFCET 離散事件建模	32
3.2	ReActNet 特徵提取硬體模組.....	33
3.2.1	IDEF0 階層式模組化設計	34
3.2.2	GRAFCET 離散事件建模	34
3.3	XGBoost 分類器硬體模組	41
3.3.1	XGBoost 硬體化工具	41
3.3.2	IDEF0 階層式模組化設計	47
3.3.3	GRAFCET 離散事件建模	49
第四章、實驗結果.....		54
4.1	實驗軟硬體開發環境.....	54
4.2	實驗說明.....	55
4.2.1	實驗資料集.....	56
4.2.2	實驗細節補充說明.....	57

4.3	ModelSim 波形模擬驗證.....	57
第五章、結論與未來展望.....		60
5.1	結論.....	60
5.2	未來展望.....	60
參考文獻.....		62

圖目錄

圖 2.1.1 XNOR 與 Bitcount 的卷積運算過程	10
圖 2.1.2 RSign 函數與 RReLU 函數圖形示意圖	11
圖 2.1.3 ReActNet 二值卷積模組.....	13
圖 2.2.1 多決策樹演算法硬體架構.....	19
圖 2.2.2 多決策樹硬體加速器中樹與節點的儲存方式.....	20
圖 2.3.1 MIAT 系統設計方法論架構.....	22
圖 2.3.2 IDEF0 基本模組功能方塊.....	23
圖 2.3.3 IDEF0 階層式模組化架構.....	24
圖 2.3.4 Sub-GRFCET 子模型建模範例.....	27
圖 3.1.1 本研究提出的 ReActNet-XGBoost 硬體加速器 IDEF0	32
圖 3.1.2 本研究提出的 ReActNet-XGBoost 硬體加速器 GRFCET.....	33
圖 3.2.1 A2 ReActNet 特徵提取硬體模組 IDEF0.....	34
圖 3.2.2 A1 ReActNet 特徵提取硬體硬體模組 GRFCET.....	35
圖 3.2.3 1x3 卷積層模組 GRFCET	36
圖 3.2.4 二值卷積層模組 GRFCET	37
圖 3.2.5 1x3 二值卷積層模組 GRFCET	38
圖 3.2.6 1x1 二值卷積層模組 GRFCET	40
圖 3.3.1 XGBoost 模型硬體化流程圖	42
圖 3.3.2 硬體化工具所使用的類別之間的 UML 關係圖.....	44
圖 3.3.3 Node 類別中的成員與 gen() 功能的定義	45
圖 3.3.4 生成 leafnode 的 behavior 例子.....	45
圖 3.3.5 Tree.init() 的核心功能	46
圖 3.3.6 Tree.gen() 功能.....	46
圖 3.3.7 更改 Q-format 表示法的程式碼片段.....	47

圖 3.3.8 模型硬體化流程 IDEF0 示意圖.....	48
圖 3.3.9 XGBoost 硬體化工具 IDEF0 示意圖.....	49
圖 3.3.10 模型硬體化流程 GRAFCET 示意圖	50
圖 3.3.11 XGB 硬體規格再訓練模組 GRAFCET 示意圖	51
圖 3.3.12 XGBoost 硬體化工具 GRAFCET 示意圖	52
圖 3.3.13 決策樹結構產生器 GRAFCET 示意圖	53
圖 4.2.1 實驗流程示意圖.....	55
圖 4.3.1 系統波形圖.....	58
圖 4.3.2 最終實驗結果放大波形圖.....	59

表目錄

表 2.1.1 ReActNet 網路參數表.....	14
表 2.3.1 GRAFCET 基本元件介紹	25
表 2.3.2 Verilog 高階合成範例.....	28
表 3.3.1 XGBoost 在使用 <code>xgb.train()</code> 時的部分可控變數	43
表 4.1.1 實驗開發環境.....	54
表 4.2.1 ECG MIT-BIH Arrhythmia Dataset 總樣本分布狀況.....	56
表 4.2.2 重新定義之樣本狀態與類別區分方式.....	57

第一章、緒論

1.1 研究背景

工業 4.0 作為第四次工業革命的核心概念，強調透過物聯網 (Internet of Things, IoT)、人工智慧 (Artificial Intelligence, AI)、大數據分析、邊緣計算及自動化技術的整合，實現智慧製造與資源最佳化配置，從而提高工業流程的效率，並將有限的資源得以有效發揮，達到生產效益的最大化。

物聯網系統通常由與各式感測器連接的裝置組成，在感測器收集環境與設備的資訊後，透過裝置之間互連進行資料交換與協同工作，進而執行特定功能的高效處理。在現代工業中，通過物聯網技術即時監控與分析機器狀況，已成為許多工廠提升競爭力與效率的常見方法[1]。

人工智慧物聯網 (The Artificial Intelligence of Things, AIoT) 在智慧工廠佔有重要的角色，例如在工業監控中，透過振動、聲學和氣體濃度資料的變化進行設備異常檢測與預測性維護 (Predictive Maintenance, PdM)，保障工業設施的高效運行；在環境監測中，以氣象數值、溫度和濕度的變化，優化工業場景中的環境調控策略，進一步提高系統穩定性[2]。除此之外，AIoT 在許多不同應用中也有亮眼的表現。例如，在智慧城市的應用中，試圖使用人工智慧與物聯網相結合的模型，減少環境中的交通堵塞狀況[3]，以及使用特定感測器協助城市監測空氣、控管用水品質[4]；在智慧家居的應用中，使民眾得以透過智慧型手機或智慧互動音響控制家中的開關與電器[5]；在智慧農業領域，則使用感測器進行針對性的測量，透過各項數值監測大範圍作物的狀況[6]，進一步智慧施肥或控制收成，實現精準農業[7]。此外，在醫療保健與居家照護領域更有多項貢獻，如可應用於防治 Covid19 的行動感測流行病監測系統[8]、心電圖分類系統、帕金森氏症 (Parkinson's disease) 預測系統、藥物追蹤系統等[9]。這些應用展現了物聯網技術為現今不論是在科技發展、環境永續、甚至是生活與社會福祉等議題改良上，都提供了無限的

可能性。

邊緣運算同樣作為工業 4.0 的關鍵技術，其通過將計算能力部署於資料生成源附近，顯著降低資訊傳輸的延遲，以提高處理效率和系統可靠性[10]。然而，邊緣裝置通常面臨資源受限的挑戰，特別是在處理大量複雜資料或執行高效能深度學習任務時，像是在工業監控中，需要靈活應對振動或聲學信號進行設備異常檢測的多樣化場景需求，以及在智慧醫療場景中實現即時分析與提供個性化診斷，需要延長裝置運行時間的低功耗需求。同時，邊緣運算設備減少了對網絡穩定性的依賴，對於偏遠地區，顯著提升了健康監測的可靠性。

機器學習 (Machine Learning, ML) 和深度學習 (Deep Learning, DL) 為 AI 的子領域，兩者分別提供了強大的決策能力及特徵提取能力，在實際應用中已廣泛被應用[11-14]。其中，相較於傳統仰賴手動標記特徵的機器學習方法，深度學習技術如卷積神經網路 (Convolutional Neural Network, CNN) 能從原始數據中直接學習特徵，提升效率與準確度。然而，儘管 AI 模型功能強大，其複雜性、高計算需求和記憶體佔用都限制了其在邊緣裝置上的應用，成為進一步提升應用性能的瓶頸。針對這些挑戰，研究者對於傳統模型的壓縮技術如剪枝 (Pruning)、量化 (Quantization)、知識蒸餾 (Knowledge Distillation, KD) 等進行了廣泛探討[15]。

不過，隨著 AI 技術的快速發展，模型的結構日益複雜，許多演算法因缺乏可解釋性被形容為黑箱 (black box)，減少了其於應用上的可靠性，特別是在安全導向的問題，及例如醫療診斷、金融風險控管等希望決策能越透明越好的應用中，更是進一步增加了潛在風險與爭議[16]。針對此一問題，有研究者開始探索具備可解釋能力的模型，例如 C. Yue 與 N. K. Jha 提出的可微分邏輯網路 (Differentiable Logic Networks, DLNs)[17]。DLNs 結合了邏輯運算與可微分的結構，能在保持高準確度的同時，生成易解釋的邏輯規則，甚至因其架構相對簡單，導致推理過程使用的硬體資源比傳統神經網路小了一萬倍，在邊緣裝置的部署上具有優勢。

在具體實現設計的方面，雖然 AI 硬體加速器的選擇多樣，涵蓋圖形處理器 (Graphics Processing Unit, GPU)、張量處理器 (Tensor Processing Unit, TPU)、(Neural network Processing Unit, NPU)、特定應用積體電路 (Application-Specific Integrated Circuit, ASIC)、數位訊號處理器 (Digital Signal Processor, DSP) 和現場可程式化邏輯閘陣列 (Field Programmable Gate Array, FPGA) 等架構。這些加速器各具優勢，但在應用場景中也存在明顯的局限與差異。

GPU 因其在大规模並行計算上的強大能力，廣泛應用於深度學習的訓練和推論階段。然而，GPU 的功耗較高且靈活性不足，限制了其在資源受限的邊緣裝置中的應用；TPU 和 NPU 專為神經網路計算設計，能提供更高效能的推論能力，但主要針對雲端和資料中心環境，難以兼顧多樣化的應用場景；ASIC 則以其針對特定任務優化的設計實現了極高的計算效能與最低的功耗，但其設計缺乏靈活性，無法適應智慧醫療或工業監控等場景中多變的需求。而 DSP 在處理訊號濾波或快速傅立葉轉換 (Fast Fourier Transform, FFT) 運算等特定應用時表現出色，但面對深度學習等高計算量任務時性能有限。

相比之下，透過波形模擬驗證，先準確驗證設計每個細節之功能性後，再進一步燒錄至 FPGA 實作，模組化與可重構的特性為硬體設計提供了顯著的優勢，對於邊緣場景中靈活且多樣化的應用需求尤為合適。

1.2 研究目標

AI 技術與邊緣運算的快速發展為智慧醫療和工業監控等應用場景帶來了深遠的影響。然而，這些場景對設備的高效能、低功耗和資源優化提出了更高要求，傳統加速器（如 ASIC）雖然可以在特定場景中表現出色，但缺乏靈活性，難以滿足多樣化的應用需求。特別是在智慧醫療中，心跳、腦波等生理信號的即時監測與分析需要設備在有限資源條件下實現高效運算與快速響應；而在工業監控中，針對振動資料或聲學信號的異常檢測則要求硬體能靈活適應多樣化的應用場景。

感測器作為資料收集的關鍵裝置，可以即時捕捉環境資訊或設備狀態並生成多種類型的資料。這些資料包括高維度的影像資料與低維度的一維時間序列資料。與高維度資料相比，低維度資料（如溫度、壓力、振動等一維時間序列資料）具備其資料量小、處理速度快的優勢，能以低成本和高效率支持多場景的即時應用[18]。因此，一維時間序列資料在智慧醫療和工業監控中展現了重要價值，成為邊緣運算中不可或缺的資料來源。

此外，針對一維訊號，一維卷積神經網絡 (1D CNN) 利用了其特性進行設計，逐漸成為解決資源使用限制問題的重要工具[19]。與傳統的二維卷積神經網絡 (2D CNN) 相比，1D CNN 通過避免維度展開的雙層運算，顯著降低了計算複雜度。同時，其輕量化結構可有效減少所需之參數量，使其特別適合於資源受限的邊緣裝置。

本研究以此為基礎，旨在針對一維時間序列資料的低處理資源優勢，設計並實現一種結合 ReActNet[20] 架構與 XGBoost[21] 架構的硬體加速器，為邊緣裝置的即時處理提供創新解決方案。

本研究初步聚焦於設計針對一維資料處理的高效能硬體架構，並探索其在智慧醫療和工業監控中的實際應用，例如，在智慧醫療中，可支持心跳和腦波等生理信號的即時監測，滿足高準確性和低延遲的需求；在工業監控中，則可用於振動與聲學資料的異常檢測，展現靈活適配多場景應用的能力。預期成果將為智慧醫療、工業監控及其他邊緣運算場景提供低功耗、高效能的硬體解決方案，並為未來多樣化應用的硬體架構設計奠定基礎。

1.3 論文架構

本論文共分為五個章節，第一章說明研究的背景與動機，並引出本研究的目標與設計方向。第二章為技術回顧，介紹與本研究相關的技術，包括二值卷積神經網路 (BinaryCNN)，以及 XGBoost 的理論基礎，並探討相關文獻的成果與局限性。第三章詳細闡述硬體加速器的設計方法與系統架構，描述 ReActNet 二值卷積網路與 XGBoost

分類器在硬體架構上的設計與實現方法。第四章描述實驗方法與分析波形模擬驗證的結果。第五章為本研究的結論與未來展望，總結本研究的成果，並提出在多維資料處理和智慧邊緣裝置應用中的改進方向。

第二章、技術回顧

本研究提出了一個結合二值卷積網路與機器學習模型的系統架構。本章節旨在回顧本研究使用到的技術，2.1 節會對由 Liu 等研究者提出的二值卷積神經網路模型——ReActNet 進行回顧、2.2 節會介紹 XGBoost 的理論基礎。

2.1 二值卷積神經網路

二值卷積神經網路 (Binary Convolutional Neural Network, BNN) 是近年來在資源受限場域中提升深度學習效能的重要研究方向。BNN 通過將卷積層中的權重與活化值二值化，大幅減少了計算的複雜性並同時提升推論速度，特別適合於邊緣裝置中的應用。然而，二值化過程常導致模型資訊損失與準確率下降，因此，如何在高效能與準確率之間取得平衡成為了相關研究的重要挑戰。在接下來的內容中，我們將對二值卷積神經網路的發展進行回顧，包括 BNN、XNOR-Net 及其衍生模型 ReActNet 技術，並探討其在硬體實現上的應用與挑戰。

2.1.1 Binarized Neural Network

二值卷積神經網路的概念最早可以由 Courbariaux 等人在其研究 BinaryConnect[22] 中提出，該方法將神經網路的權重二值化，在神經網路訓練的前向傳播 (forward propagation) 與反向傳播 (backward propagation) 階段，採用隨機性二值化將浮點數權重量化為 +1 或 -1，顯著壓縮了模型大小並提高訓練效率，但仍保留浮點數參數進行更新。隨機性二值化的計算方式如公式 2.1 所示：

$$w_b = \begin{cases} +1, & \text{with probability } p = \sigma(w) \\ -1, & \text{with probability } 1 - p \end{cases} \quad (2.1)$$

其中， w 為浮點數權重， w_b 為經過二值化的權重；而 σ 為 Hard Sigmoid 函數，如公式 2.2 所示。

$$\sigma(x) = \text{clip}\left(\frac{x+1}{2}, 0, 1\right) = \max\left(0, \min\left(1, \frac{x+1}{2}\right)\right) \quad (2.2)$$

在神經網路的參數更新階段，為了防止浮點數權重數值過大而導致二值化效果失效，採用了 clip 函數，將權重限制在 $[-1, 1]$ 的範圍內。BinaryConnect 通過對神經網路的權重進行二值化處理的策略，實現對模型參數的顯著壓縮，同時也提升了神經網路訓練速度並降低計算複雜度。該研究實驗結果顯示，儘管權重經過二值化處理，其網路模型仍能達到不錯的表現。

隨後，Courbariaux 等人進一步提出了二值化神經網路 (Binarized Neural Network, BNN)[23]，作為 BinaryConnect 的改良與擴展。與 BinaryConnect 不同，BNN 在訓練階段的前向傳播 (forward propagation) 和反向傳播 (backward propagation) 中，將權重與活化值均二值化，使得推論階段幾乎完全由加法與位元運算 (bitwise operation) 替代原本的浮點數乘法運算，大幅提升運算效率。在權重量化的設計中，雖然隨機二值化 (stochastic binarization) 在理論上具有更佳表現，但其硬體實作需依賴亂數產生器，實現難度較高。因此，BNN 選擇了更適合硬體設計的確定性二值化 (deterministic binarization)，即採用常見的符號函數 (Sign function)，確定性二值化如公式 2.3 所示。

$$x_b = \text{sign } x = \begin{cases} +1, & \text{if } x \geq 0 \\ -1, & \text{otherwise} \end{cases} \quad (2.3)$$

其中， x 表示浮點數權重或活化值， x_b 是二值化權重或活化值。在神經網路訓練過程中，更新權重所需的梯度是根據二值化後的活化值計算得來。然而，由於 Sign 函數的導數幾乎都為零，無法與反向傳播兼容，因此在倒傳遞 (Backpropagation) 階段，該研究將 Sign 函數近似為 Htanh 函數，以便獲取梯度進行參數更新。Htanh 函數如公式 2.4 所示：

$$Htanh(x) = Clip(x, -1, 1) = \max(-1, \min(1, x)) \quad (2.4)$$

受益於對權重與活化值進行二值化，Binarized Neural Network (BNN) 利用位元運算取代了深度神經網路中大量使用的乘加運算，顯著提升了推論速度，同時使其更易於硬體實現。在該論文實驗中，可以觀察到 BNN 在解決小規模且簡單的影像分類資料集(如 MNIST[24]、CIFAR-10[25]) 問題時，其性能與傳統浮點數神經網路相比差異不大。然而，在硬體資源使用、耗電量和晶片面積方面，BNN 顯示出顯著優勢，大幅優於浮點數神經網路。

2.1.2 XNOR-Net

在 Binarized Neural Network 被提出不久後，Rastegari 等人針對大規模的影像分類資料集，提出兩種 Binary-Weight-Networks(BWN) 與 XNOR-Networks (XNOR-Net)[26]。BWN 將卷積神經網路中的實數權重近似為縮放係數與二值權重的乘積，使用近似值公式 $W \approx \alpha B$ 將實值權重 W 表示為以二值權重 B 與縮放因子 α 的結合，該論文最終推導出 B 與 α 的方法如公式 2.5 所示。

$$I * W \approx (I \oplus B)\alpha \quad (2.5)$$

其中， I 為輸入張量， W 為實數權重， B 為二值權重， α 為縮放因子， $*$ 表示標準的卷積運算，而 \oplus 則表示不包含乘法的卷積運算。 B 和 α 的計算方法分別如公式 2.6 和公式 2.7 所示：

$$B = \text{sign}(W) \quad (2.6)$$

$$\alpha = \frac{\sum |W_i|}{n} \quad (2.7)$$

$\sum |W_i|$ 表示該實值權重卷積核中所有權重之絕對值總和， n 代表卷積核大小。透過將權重二值化，Binary-Weight-Networks (BWN) 將標準實數權重卷積運算轉化為二值權重卷積運算，實現無需乘法的計算方式。這種方法顯著提升了推論效率，在 CPU 上的推論速度提高約 2 倍，同時參數儲存空間需求減少至原來的 1/32。此外，相較於標準卷積神經網路，BWN 的準確度下降幅度較小，性能依然可接受。

該論文提出的另一種網路 XNOR-Networks，則是使用類似於 Binary-Weight-Networks (BWN) 的近似方法，即 $X \approx \beta H$ ，將輸入張量 X 分割為與卷積核大小相等的子集 (sub-tensor)，其中 β 是輸入子集的縮放因子， H 是二值化後的輸入子集， β 與 H 的求得方法分別如公式 2.8 與公式 2.9 所示：

$$\beta = \frac{\sum |X_i|}{n} \quad (2.8)$$

$$H = \text{sign}(X) \quad (2.9)$$

在計算輸入子集的縮放因子 β 時，需對輸入張量 I 進行特別處理。首先計算一個二維矩陣 A ，其計算方式如公式 2.10 所示。

$$A_{ij} = \frac{\sum |I_{ij}|}{c} \quad (2.10)$$

其中， $\sum |I_{ij}|$ 代表在輸入張量 I 的 (i, j) 位置上所有元素之絕對值總和， c 代表輸入張量的通道數，因此， A 代表通道中所有輸入向量元素之絕對值平均。此步驟的目的是為了減少每個子集向量縮放因子 β 的冗餘計算。

接著，將計算得到的二維矩陣 A 與二維卷積核 k 進行卷積運算，最終得到矩陣 K ，即 $K = A * k$ 。其中二維卷積核 k 中的元素皆為 $1/n$ ， n 為二維卷積核 k 的大小。矩陣 K 包含所有輸入張量子集的縮放因子，其中元素 K_{ij} 對應於以 (i, j) 位置為中心的

輸入子集張量之縮放因子 β 。一旦有權重之縮放因子 α 和矩陣 K ，即代表可以得到近似卷積運算的二值卷積運算，其計算方式如公式 2.11 所示。

$$I * W \approx (\text{sign}(I) \circledast \text{sign}(W)) \odot K\alpha \quad (2.11)$$

其中， I 為輸入張量， W 為實值權重， $*$ 為標準卷積運算； $\text{sign}(I)$ 為二值輸入張量， $\text{Sign}(W)$ 為二值權重， \circledast 代表使用 XNOR 與 Bitcount 的二值卷積運算，其運算如下圖 2.1.1 所示，而最後的 \odot 代表對應元素的乘積。

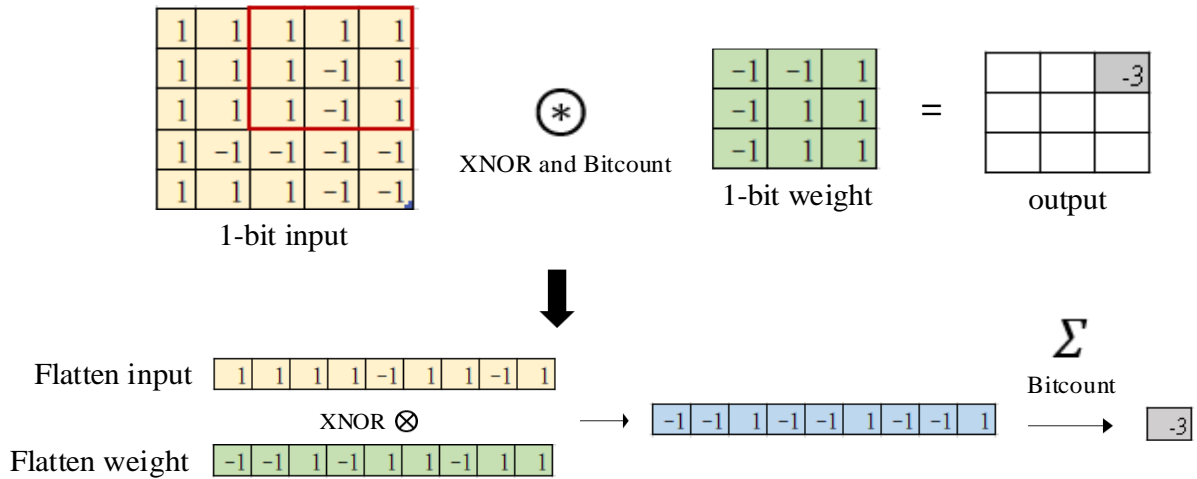


圖 2.1.1 XNOR 與 Bitcount 的卷積運算過程

採用上述近似卷積運算方法，XNOR-Net 在 CPU 上的推論速度實現了高達 58 倍的提升，並大幅減少 32 倍的儲存空間需求，為在資源受限的設備上實現即時運算提供了可行性。然而，與標準卷積神經網路相比，XNOR-Net 的準確度仍有所下降。因此，自 XNOR-Net 的提出以來，許多研究開始致力於改善二值卷積神經網路的準確度，目標是在保有低儲存空間需求與高計算效率的基礎上，將其準確度提升至接近標準卷積神經網路的水準，從而使深度神經網路的應用能夠進一步拓展至硬體資源受限的邊緣裝置。

2.1.3 ReActNet

在 2020 年，Liu 等人提出了一種新的二值卷積神經網路架構—ReActNet[20]。該架構以 MobileNetV1[27] 為基礎進行改良，不僅保留了二值神經網路的低運算量特性，還在影像識別問題上展現了近年來二值化神經網路研究中最為突出的準確度表現。

ReActNet 的網路設計將原架構中的深度卷積運算 (Depthwise Convolution) 替換為標準卷積運算，並通過合併輸出結果的方式增加通道數，同時引入捷徑 (shortcut) 連結以強化不同區塊間的資訊傳遞。在二值卷積運算方面，ReActNet 參考了 XNOR-Net 的運算方法，採用二值化的權重和活化值進行計算。此外，Liu 等人發現活化值的數值分布對二值卷積神經網路的語意特徵表示 (semantic feature representation) 有顯著影響。為改善活化值的數值分布，ReActNet 引入了兩個新函數：ReAct-Sign(RSign) 和 ReAct-PReLU (RReLU)，分別為公式 2.12 和公式 2.13。

$$x^b = RSign(x) = \begin{cases} +1, & \text{if } x > \alpha \\ -1, & \text{if } x \leq \alpha \end{cases} \quad (2.12)$$

$$y = RReLU(x) = \begin{cases} x - \gamma + \delta, & \text{if } x > \gamma \\ \beta(x - \gamma) + \delta, & \text{if } x \leq \gamma \end{cases} \quad (2.13)$$

其函數圖形如下圖 2.1.2 所示。

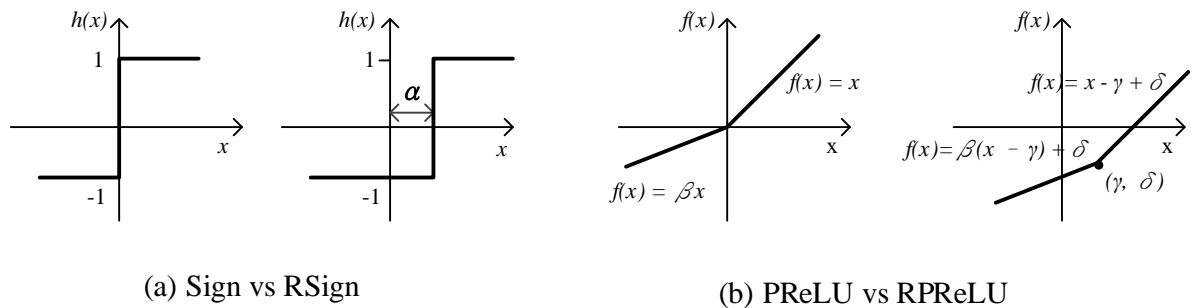


圖 2.1.2 RSign 函數與 RReLU 函數圖形示意圖

其中， x^b 為二值化後之權重， α 為 RSign 函數經過學習後獲得的閾值， y 為經過 RPreLU 函數獲得之權重， γ 及 δ 為 RPreLU 函數經過學習後獲得的偏移值， β 為 RPreLU 函數經過學習後獲得的斜率值。透過引入 RSign 和 RPreLU 函數，ReActNet 能在進行二值卷積運算時省略輸入張量縮放因子 K 的計算（詳見公式 2.11），進一步簡化運算流程。

在降採樣 (downsampling) 的捷徑設計中，Liu 等人採用平均池化 (average pooling) 將捷徑上的特徵圖尺寸調整至與降採樣的輸出一致。此外，為了解決輸入與輸出之間存在的通道數差異，Liu 等人使用了活化值複製 (duplication) 與串接 (concatenation) 的策略。ReActNet 的完整二值卷積模組架構如圖 2.1.3 所示，其中，圖 2.1.3 (a) 為 ReActNet 二值卷積模組之一般區塊 (Normal Block)，而圖 2.1.3 (b) 為 ReActNet 二值卷積模組之殘差區塊 (Reduction Block)。ReActNet 網路參數表如表 2.1.1，表中以淺綠色標示的部分對應圖 2.1.3 中的 Normal Block，以淺藍色標示的部分對應圖 2.1.3 中的 Reduction Block。

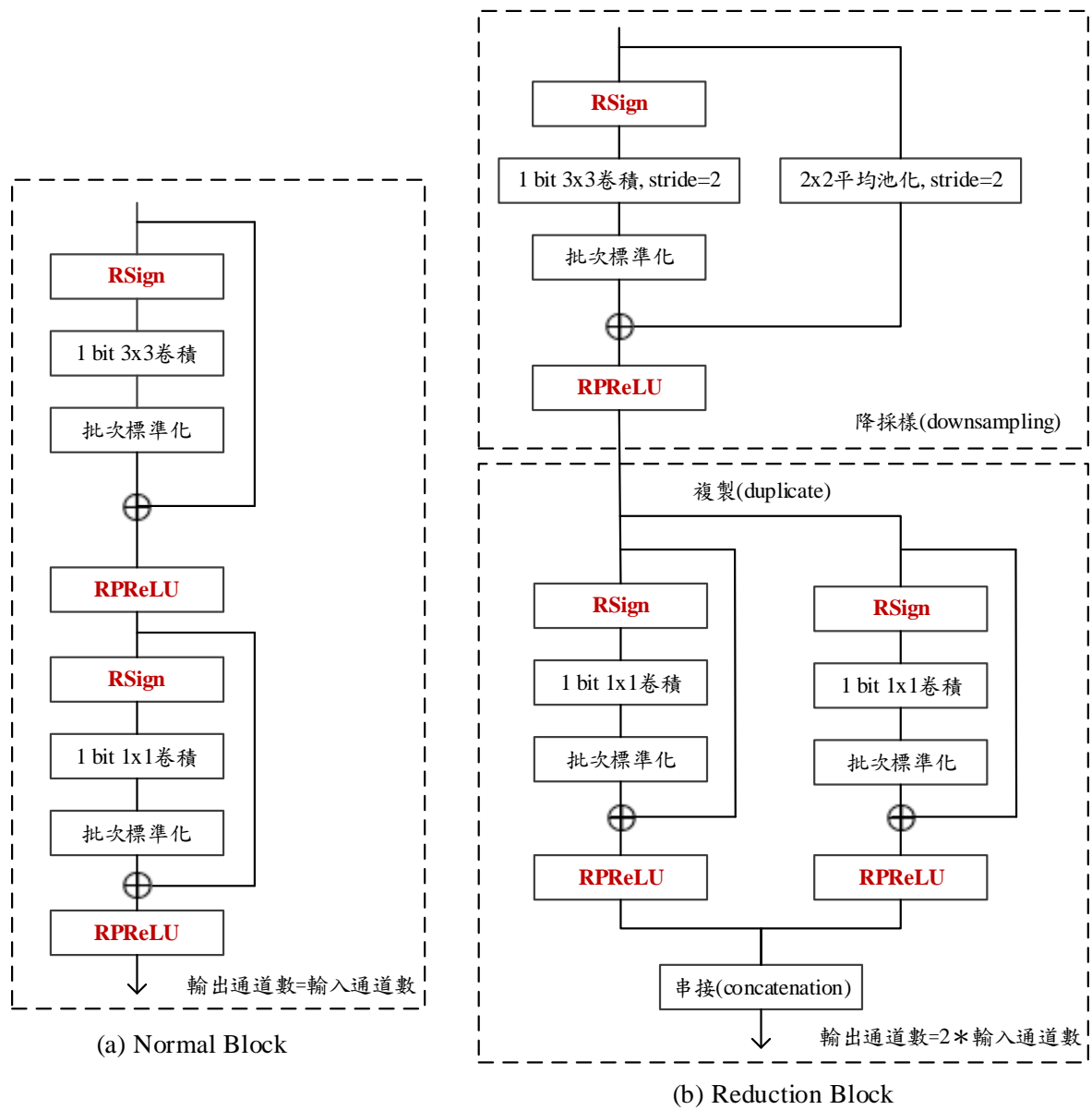


圖 2.1.3 ReActNet 二值卷積模組

表 2.1.1 ReActNet 網路參數表

運算方式		步長	卷積核大小 (長 x 寬)	卷積核 輸入通道數	卷積核 輸出通道數
Conv		2	3x3	3	32
1-bit 3x3 Conv		1	3x3	32	32
1-bit 1x1 Conv		1	1x1	32	64
1-bit 3x3 Conv		2	3x3	64	64
1-bit 1x1 Conv		1	1x1	64	128
1-bit 3x3 Conv		1	3x3	128	128
1-bit 1x1 Conv		1	1x1	128	128
1-bit 3x3 Conv		2	3x3	128	128
1-bit 1x1 Conv		1	1x1	128	256
1-bit 3x3 Conv		1	3x3	256	256
1-bit 1x1 Conv		1	1x1	256	256
1-bit 3x3 Conv		2	3x3	256	256
1-bit 1x1 Conv		1	1x1	256	512
5x	1-bit 3x3 Conv	1	3x3	512	512
	1-bit 1x1 Conv	1	1x1	512	512
1-bit 3x3 Conv		2	3x3	512	512
1-bit 1x1 Conv		1	1x1	512	1024
1-bit 3x3 Conv		1	3x3	1024	1024
1-bit 1x1 Conv		1	1x1	1024	1024
平均池化層		1	1x1	1024	1024
全連接層					

在訓練階段，ReActNet 採用分布損失 (distributional loss) 作為損失函數，並以一般實數卷積神經網路作為目標學習網路。透過計算 ReActNet 與目標網路輸出之間的損失，旨在縮小二值卷積神經網路與實數卷積神經網路在性能表現上的差距，從而提升二值化網路的準確度與可靠性。分布損失函數公式如公式 2.14 所示。

$$Loss = -\frac{1}{n} \sum_c \sum_{i=1}^n S_c^R(x_i) \log \left(\frac{S_c^B(x_i)}{S_c^R(x_i)} \right) \quad (2.14)$$

其中， c 為辨識類別， n 為批次大小， x_i 代表當前批次中第 i 筆資料的各類別輸出值， S 為 softmax 函數， R 與 B 則分別對應實數卷積神經網路與二值卷積神經網路，則分別為實數卷積神經網路與二值卷積神經網路，該篇論文之開源程式以 ResNet-34 作為預設學習對象。

在實驗結果分析中發現，ReActNet 在相對較低的硬體資源需求條件下，能夠達到接近 ResNet-18 的辨識性能，展現其在邊緣計算應用中的顯著潛力。因此，本研究選擇將 ReActNet 網路架構作為核心設計參考。

2.2 XGBoost

XGBoost (eXtreme Gradient Boosting) 是一種基於 Gradient Boosting 架構改良而成的高效機器學習演算法，本節將先介紹 Gradient Boosting 的基本概念，隨後說明 XGBoost 所作的改進之處。Gradient Boosting 作為集成學習 (Ensemble Learning) 中的 Boosting 技術，最早由 Friedman 等人於 2001 年提出[28]，其核心概念是在每次迭代中逐步訓練新的分類器，修正前一分類器的錯誤，最終構建出強大的集成模型。

具體來說，第 m 個迭代後的大分類器 $F_m(x)$ 是由前一個迭代的分類器 $F_{m-1}(x)$ 與新加入的子分類器 $\rho_m h_m(x)$ 組合而成，其中 $h(x)$ 表示新加入的函數， ρ 則為該函數的權重，如公式 2.15 所示。

$$F_m(x) = F_{m-1}(x) + \rho_m h_m(x) \quad (2.15)$$

Gradient boosting 與傳統 Boosting 的不同之處在於其引入了負梯度與損失函數的概念，用以指導新子分類器的生成。每次新增的子分類器 $\rho_m h_m(x)$ 的目的都是為了最大限度地降低損失函數的值，如公式 2.16 所示。該方法的核心在於尋找最佳的 $\rho h(x)$ 組合，使得損失函數達到最小化。

$$\rho_m h_m(x) = \operatorname{argmin}_{\rho, h} \sum_{i=1}^N L(y_i, F_{m-1}(x_i) + \rho h(x_i)) \quad (2.16)$$

為了使模型有效收斂，每次加入新的 $\rho_m h_m$ 可以視為對理想模型 $F^*(x)$ 的一次近似。在實際操作中，於生成新的 h_m 之前，會先構建一組錯誤殘差訓練集 $D = \{x_i, r_{mi}\}_{i=1}^N$ ，其中， r_{mi} 稱為錯誤殘差 (false residual) 或稱作負梯度 (negative gradient)。 r_{mi} 用以表示分類器的輸出與真實標籤 (true label) 之間的差異，如公式 2.17 所示。

$$r_{mi} = \left[\frac{\delta L(y_i, F_{m-1}(x))}{\delta F_{m-1}(x)} \right] \quad (2.17)$$

透過錯誤殘差訓練集可以生成新的 $h(x)$ ，接著利用線搜尋 (line search) 方法找到最佳的權重 ρ ，即完成一次迭代。完整的 Gradient Boosting 演算法流程如下所示：

Algorithm gradient boosting

Input : 訓練資料 $S = (x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)$

M : 子決策樹的數量，即最後子決策樹的數量

$L(y, F(x))$: 微分損失函數

Procedure:

1) 使用常數值初始化模型數值

$$F_0 = \operatorname{argmin}_{\alpha} \sum_{i=1}^N L(y_i, \alpha)$$

2) **for** $m = 1, \dots, M$:

a) 計算殘差

$$r_{mi} = \left[\frac{\delta L(y_i, F(x))}{\delta F(x)} \right], \text{ for } i = 1, \dots, n$$

b) 使用錯誤殘差訓練集 $D = \{x_i, r_{mi}\}_{i=1}^N$ 訓練新的子分類器

c) 使用線搜尋的方式找到最佳的 ρ_m

d) 加入新的子分類器並更新模型，並使用參數 v 調節每次 step 的下降速度以防止過擬合，通常 $v=0.1$ 。[29]

$$F_m(x) = F_{m-1}(x) + v\rho_m h_m(x)$$

end for

Output : 經過 M 次迭代的分類器 $F_M(x)$

XGBoost 是由 Chen 等人提出的一種改良演算法[21]，其架構與 Gradient boosting 基本相同，但針對損失函數進行了優化，有效緩解了 Gradient boosting 中的過擬合問題。XGBoost 在損失函數中加入了正則化 (Regularization) 項 $\Omega(h_m)$ ，如公式 2.18 所示，此一正則化項在損失函數中發揮作用，進一步提升了模型的泛化能力。

$$L_M(F(x_i)) = \sum_{i=1}^M L(y_i, F(x_i)) + \sum_{m=1}^M \Omega(h_m) \quad (2.18)$$

正規化項 $\Omega(h)$ 如公式 2.19 所示，其中， T 代表樹的葉節點數量，而 ω 為葉節點的權重， λ 則表示葉節點權重的懲罰參數，用於防止模型過度依賴單一節點。此外， γ 為節點數量的懲罰參數，適當提高其值可以限制樹的結構複雜度，避免單棵樹過於複雜化。

$$\Omega(h) = \gamma T + \frac{1}{2} \lambda \|\omega\|^2 \quad (2.19)$$

透過加入正規化項，XGBoost 能有效限制單棵子樹的模型複雜度，從而減少使用 Boosting 技術時常見的過擬合問題，提升模型的泛化能力。

2.2.1 多決策樹硬體加速器

在 A. Alcolea 等人的研究中[30]，定義了一個多決策樹的硬體電路 IP，並在此基礎上加入了平行化與管線化的架構，使模型在運算時能夠充分享有硬體化帶來的加速特性。該研究也明確規範了決策樹的格式，透過限縮單一 node 的資料長度，讓決策樹的 node 能夠更好地被配置在硬體的儲存單元中，也減少了硬體在解碼 node 資訊時的負擔。

該研究提出的多決策樹硬體加速器架構如圖 2.2.1 所示，每個 class 模組都擁有自己的多決策樹模型，當資料 (feature) 傳進 class 模組後，就會輸出自身負責類別的分數 (result)，接著求出這些 class 模組之間 result 的最大值，作為最終的預測結果 (pred)。class 模組的多決策樹模型會含有多棵小決策樹，並以 pre-ordering 的方式排列自身的 node。

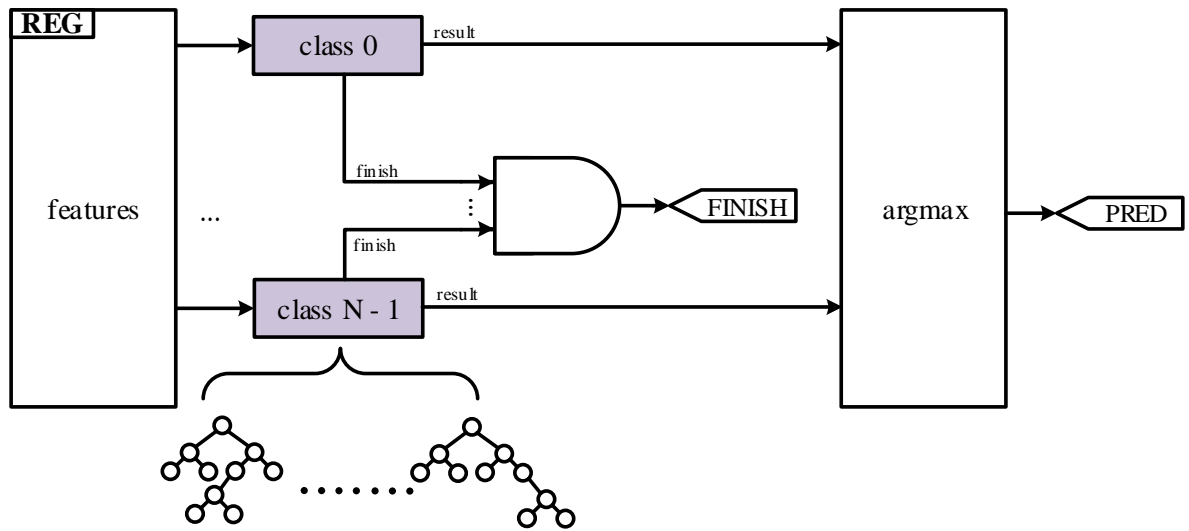


圖 2.2.1 多決策樹演算法硬體架構

此外，決策樹中的 node 分為兩種：tree node 與 leafnode，如圖 2.2.2 所示。tree node 與 leafnode 會使用第 0 位元的位置作為 flag，用來辨別是哪一種 node。在 tree node 中則儲存了比較值 (cmp_val)、輸入資料的索引 (@_feature)、右子樹的相對位址 (rel@_right_child)。在預測時 tree node 會比較自身的 cmp_val 與輸入資料的索引中之值，以決定接下來往左子樹還是右子樹走。而 leafnode 作為單棵樹的終點，會將 node 中所存放的權重 (leaf_val) 累加到 class 模組的 result 中，並跳轉到下棵決策樹進行預測，因此，leafnode 除了要儲存自身的 leaf_val 外，還需要知道下一棵決策樹的絕對位址 (@_next_tree)。此外，leafnode 還在第 1 位元的位置標示了當前決策樹是否為最後一棵的資訊，讓更上層的電路得以知道決策樹的預測是否已經結束。

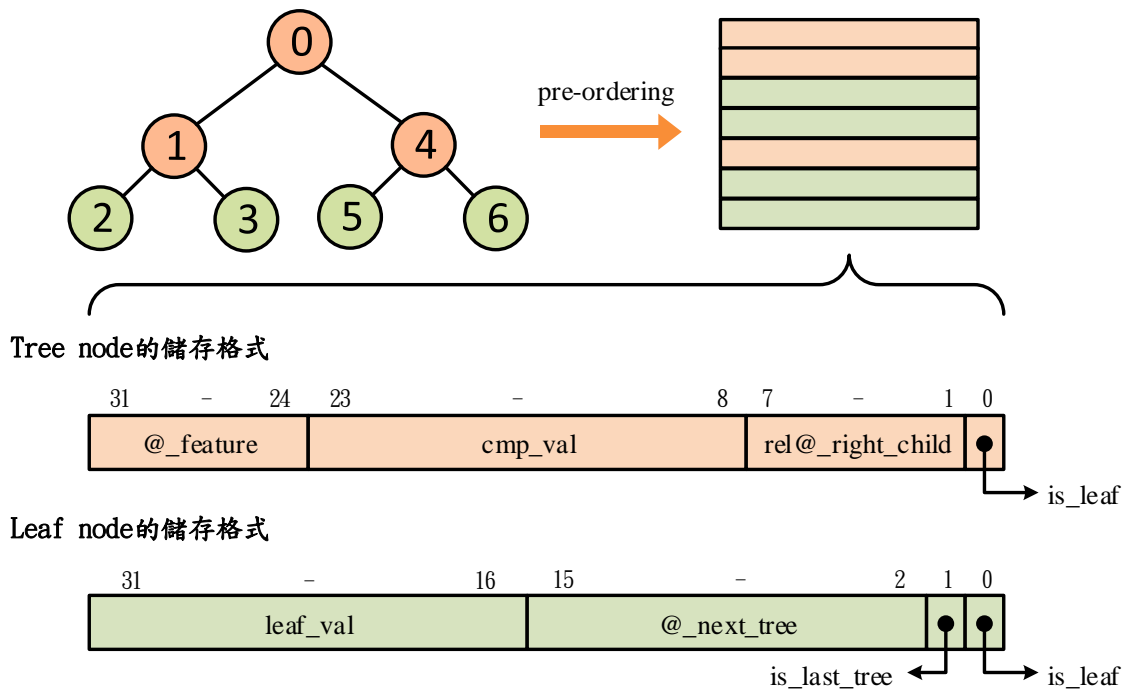


圖 2.2.2 多決策樹硬體加速器中樹與節點的儲存方式

如前所述，此一多決策樹硬體加速器針對節點的格式有明確的定義。其中，每個單一節點的資訊都會統一被壓縮在長度為 32-bit 的儲存單元中，具體體現在決策樹上的限制如下所示：

1. 輸入的 data 長度最長為 2^8 維（受限於 tree node 的 @_feature）。
2. 單棵 tree 的 node 總數不能超過 2^7 （受限於 tree node 的 rel@_right_child）。
3. 節點的 weight 最大只能使用 16 bits 來儲存（不論是 tree node 的 cmp_val 還是 leaf node 的 leaf_val）。
4. 單一個 class 所有 tree 的 node 總數加起來不能超過 2^{14} （受限於 leaf node 的 @_next_tree）。

2.3 MIAT 系統設計方法論

傳統的系統與工程開發過程通常遵循由需求分析、設計、實作、測試到部署的線性流程。在這種典型架構下，系統被分解為硬體與軟體兩大部分，各自再細分為子系統進行開發與測試。然而，隨著現代嵌入式系統逐漸應用於多模態資料處理、邊緣運算與人工智慧等複雜場景，這種線性開發模式已難以應對日益增長的設計挑戰，特別是在系統複雜度 (Complexity)、資源優化 (Optimization) 以及市場推廣時間 (Time-to-Market) 等方面。

嵌入式系統設計中的複雜度往往體現為多模組之間的交互作用和資源競爭，使得系統設計者需要在效能、功耗、成本和體積之間尋找最佳平衡。同時，縮短開發時間以快速投入市場成為企業競爭的關鍵。然而，這些目標彼此間的矛盾性，對系統設計方法論提出了更高的要求。

在這樣的背景下，MIAT 系統設計方法論應運而生，其以模組化和層次化的設計思想為核心，結合 IDEF0 階層式模組化設計和 GRAFCET 離散事件建模等技術，主要設計架構流程如圖 2.3.1 所示。不僅在應對系統設計的複雜度方面表現優越，也能顯著提升設計效率並縮短開發週期，同時增強了系統設計的可解釋性。

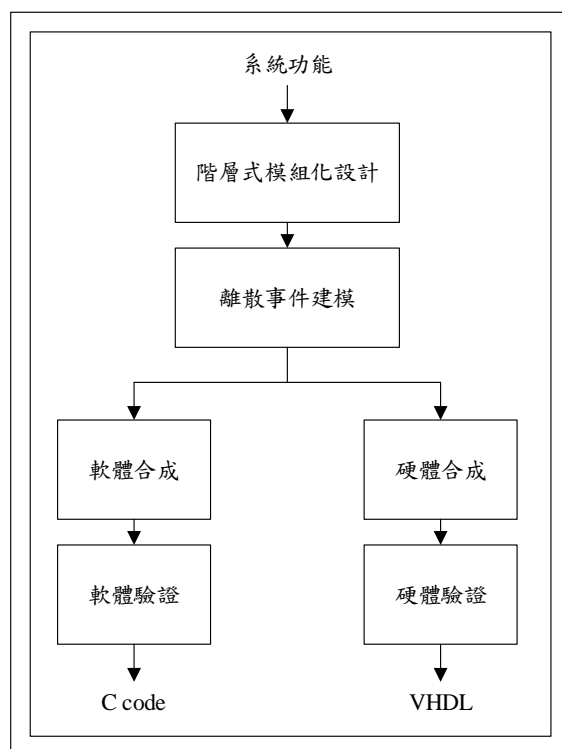


圖 2.3.1 MIAT 系統設計方法論架構

後續篇幅將詳細介紹 MIAT 系統設計方法論的理論基礎與實踐應用，包括其如何透過層次化分解和動態行為建模來應對設計挑戰。

2.3.1 IDEF0 階層式模組化設計

在嵌入式系統與硬體設計的過程中，系統的複雜度往往成為開發團隊面臨的主要挑戰之一。IDEF0 (Integration Computer Aided Manufacturing (ICAM) DEFinition languages) 是一種由美國空軍於 1960 年代發展的建模工具，旨在應對複雜系統設計所帶來的挑戰。

IDEF0 是 IDEF 方法論的分支之一，透過結構化和圖形化的方式，將系統分解成多層次的功能模組。其中，每個基本模組如圖 2.3.2 所示，使用功能方塊 (Function Block) 表示，並通過箭頭來描述模組間的資料流與控制關係。

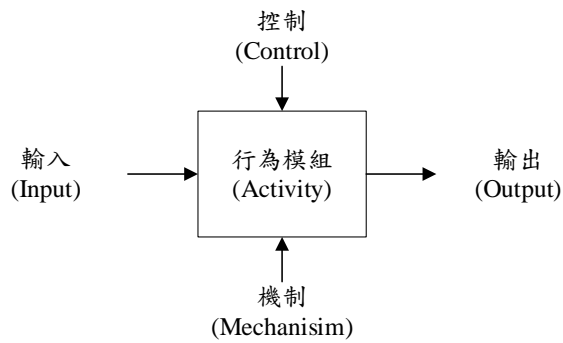


圖 2.3.2 IDEF0 基本模組功能方塊

其中，輸入 (Input) 表示模組運作所需的資料或控制訊號；控制 (Control) 表示用以規範模組執行的條件或約束，如政策或標準；輸出 (Output) 為模組執行後產生的資料或控制訊號結果；機制 (Mechanism) 則表示模組執行所需的資源，例如硬體、演算法或人力。

在設計上，IDEF0 採用自頂向下 (Top-Down) 的階層式模組化設計策略，如圖 2.3.3 所示，將主系統 (A0) 分解為多個子系統 (如 A1、A2、A3)，並根據子系統的複雜程度進一步細分 (如 A31、A32、A33)。這種分層設計方式不僅有助於開發團隊的分工與模組整合，還能大幅縮短開發時程，提高設計效率。

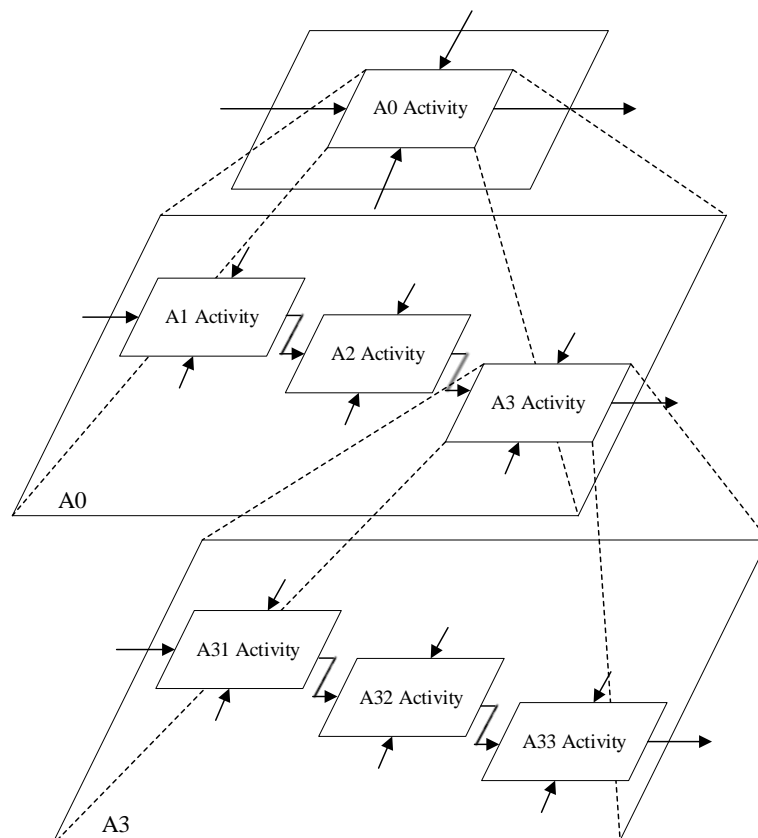


圖 2.3.3 IDEF0 階層式模組化架構

IDEF0 之一大特點在於其圖形化與結構化的設計方法能清晰地展示模組間的交互關係，同時能輕易地對複雜的系統進行分解，不僅提升了設計的直觀性，還使模組的功能定位與資料傳遞變得更為清晰。每層功能模組明確地定義輸入、輸出、控制和機制，使得各模組之間的交互作用更加規範，適用於跨領域、跨功能的系統設計。這種分層架構對於協同團隊工作和整體系統優化具有重要意義。

2.3.2 GRAFCET 離散事件建模

隨著嵌入式系統和自動化控制系統的日益複雜，描述系統內部動態行為和狀態轉換的需求越來越強烈。為了解決此問題，GRAFCET (Graphe Fonctionnel de Commande Etape-

Transition) 最初由法國工業界與學術界共同制定，作為一種圖形化的離散事件建模方法，其提供了一套完整的工具來描述系統的動態行為，廣泛應用於可程式化邏輯控制器 (Programmable Logic Controller, PLC) 以及其他嵌入式與自動化系統。

GRAFCET 使用簡潔的圖形化元件來描述系統的狀態變化與動作以及系統中的平行或分歧架構，這在多任務操作系統（如無人機飛行任務或 DDS 通訊網路流程）中尤為重要。透過平行結構的展示，開發者可以清晰了解各事件之間的依賴關係與執行邏輯，進一步提高系統設計的可靠性和維護性。其基本元件如表 2.3.1 所示。

表 2.3.1 GRAFCET 基本元件介紹

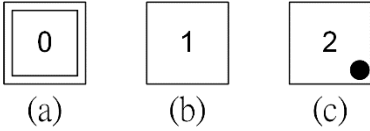
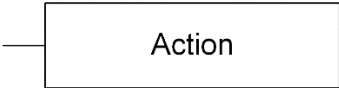
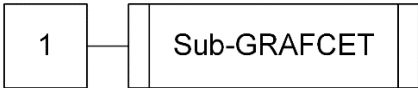
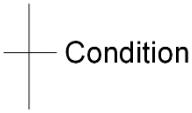
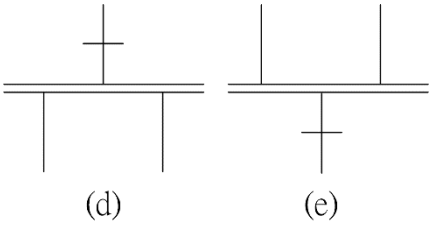
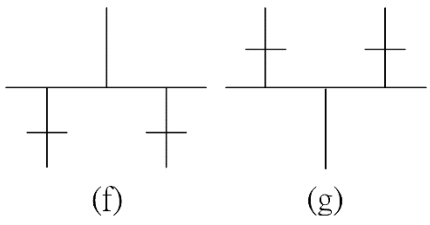
基本元件	說明
 (a) (b) (c)	(a) 基本狀態方塊，若為雙線方塊則表示為初始狀態，否則為一般狀態； (b) 狀態方塊中若無黑圓點表示狀態閒置中； (c) 狀態方塊中若有黑圓點表示狀態動作中。
	動作方塊，與狀態方塊連接，用於描述狀態成立後所需執行的動作。
	狀態方塊與 Sub-GRAFCET 方塊連接之圖示，Sub-GRAFCET 方塊表示此動作方塊是由另一個 GRAFCET 模型所組成。

表 2.3.1 GRAFCET 基本元件介紹（續前頁）

基本元件	說明
	<p>狀態轉移條件圖示，直線上下端連結兩個狀態，橫線右方以文字描述轉移條件，若當前狀態滿足條件則會轉移至下一個狀態。</p>
 <p>(d) (e)</p>	<p>(d) Divergence AND 與 (e) Convergence AND 被用來表示 GRAFCET 模型中的平行架構。</p>
 <p>(f) (g)</p>	<p>(d) Divergence OR 與 (e) Convergence OR 被用來表示 GRAFCET 模型中的分歧架構。</p>

GRAFCET 的模組化結構支持將大型系統分解為多個子模組，以簡化複雜度並促進團隊協作。此外，在階層化建模過程中，較為複雜的狀態與動作可以進一步細分為子模組 (Sub-GRAFCET)，形成分層的控制邏輯，如圖 2.3.4 所示。這樣的方法不僅提升了系統的可維護性，還同時縮短了開發與除錯的時間。

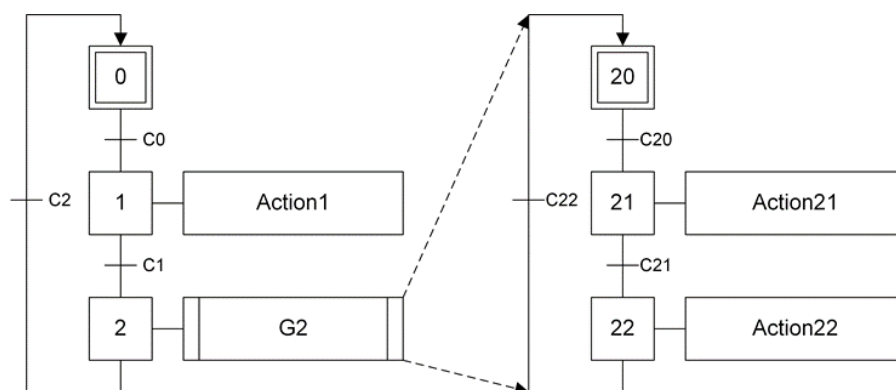


圖 2.3.4 Sub-GRAF CET 子模型建模範例

隨著嵌入式系統與自動化控制技術的發展，透過由 GRAFCET 與 IDEF0 結合的 MIAT 系統設計方法論來進行開發的優勢越來越明顯[31]。這種階層式、模組化的設計理念為開發系統的過程添加了更加靈活與可擴展性，為系統設計者甚至一整個設計團隊提供了能輕易進行調整且易於表達理念的設計流程。

2.3.3 硬體高階合成

依照第 2.3.2 節描述的繪製流程，GRAFCET 能夠簡便地應用於各系統模組的設計。由於 GRAFCET 具有清晰的軟硬體整合規則，表 2.3.2 中列舉了一些基本元件轉換為硬體實現的範例，這些範例可作為參考，將 GRAFCET 直接映射為硬體描述語言，從而實現高階硬體合成。

然而，硬體設計與軟體設計有顯著差異，特別是在硬體中，所有指令會在每個時脈週期內同步執行。例如，表 2.3.2 中的 Convergence OR 元件，當條件 X1 和 X2 同時處於活動狀態時，Action 1 和 Action 2 中描述的功能會同步執行。這種特性與傳統軟體的線性執行邏輯截然不同，也是初學者在硬體撰寫時經常遇到的挑戰。

透過 MIAT 方法論的高階硬體合成流程，開發者可以快速進行硬體設計，並專注於管線化與演算法的優化，顯著提升開發效率與系統性能。

表 2.3.2 Verilog 高階合成範例

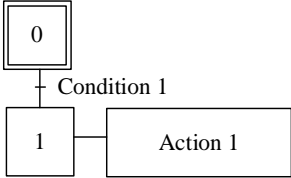
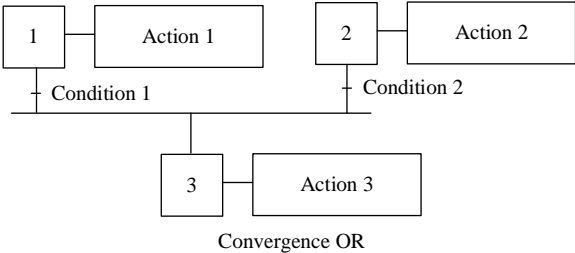
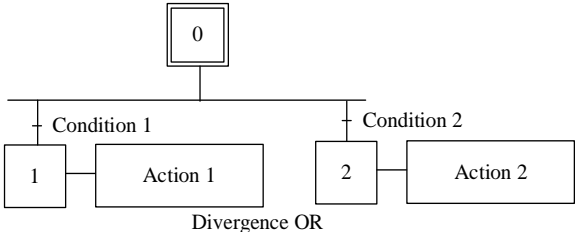
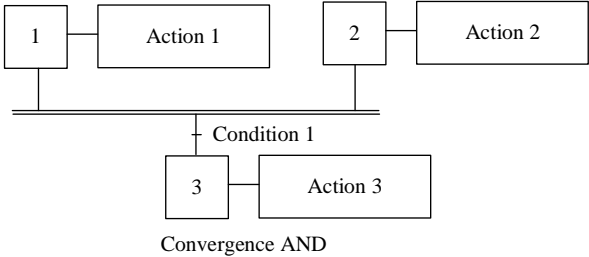
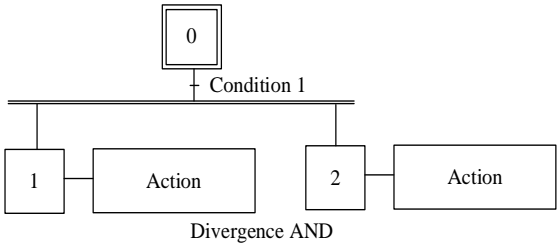
GRAFCET	Verilog Code
 <pre> graph TD S0[0] -- Condition 1 --> S1[1] S1 --> A1[Action 1] </pre>	<pre> if((X0 == 1) && (Condition 1 == 1)) begin X0 <= 0; X1 <= 1; end if(X1 == 1) begin Action 1 end </pre>
 <pre> graph TD S1[1] -- Condition 1 --> S3[3] S2[2] -- Condition 2 --> S3 S1 --> A1[Action 1] S2 --> A2[Action 2] S3 --> A3[Action 3] </pre> <p style="text-align: center;">Convergence OR</p>	<pre> if((X1 == 1) && (Condition 1 == 1)) begin X1 <= 0; X3 <= 1; end if((X2 == 1) && (Condition 2 == 1)) begin X2 <= 0; X3 <= 1; end if(X1 == 1) begin Action 1 end if(X2 == 1) begin Action 2 end if(X3 == 1) begin Action 3 end </pre>
 <pre> graph TD S0[0] -- Condition 1 --> S1[1] S0 -- Condition 2 --> S2[2] S1 --> A1[Action 1] S2 --> A2[Action 2] </pre> <p style="text-align: center;">Divergence OR</p>	<pre> if(X0 == 1) begin if(Condition 1 == 1) begin X0 <= 0; X1 <= 1; end else if(Condition 2 == 1) begin X0 <= 0; X2 <= 1; end end if(X1 == 1) begin Action 1 end if(X2 == 1) begin Action 2 end </pre>

表 2.3.2 Verilog 高階合成範例（續前頁）

GRAFCET	Verilog Code
 <p>Convergence AND</p>	<pre> if((X1 == 1) && (X2 == 1) && (Condition 1 == 1)) begin X1 <= 0; X2 <= 0; X3 <= 1; end if(X1 == 1) begin Action 1 end if(X2 == 1) begin Action 2 end if(X3 == 1) begin Action 3 end </pre>
 <p>Divergence AND</p>	<pre> if((X0 == 1) && (Condition 1 == 1)) begin X0 <= 0; X1 <= 1; X2 <= 1; end if(X1 == 1) begin Action 1 end if(X2 == 1) begin Action 2 end </pre>

接續本章節對於 MIAT 系統設計方法論的描述，第三章將深入探討本系統如何使用此套方法論進行系統架構的設計與實現。

第三章、ReActNet-XGBoost 硬體加速器系統設計

本系統的目標是實現一個針對資源受限的邊緣裝置所設計的高效、低功耗分類系統，設計上旨在通過模組化設計和硬體加速技術，平衡準確性、計算成本和硬體資源使用率，同時滿足即時處理的需求。

本章將詳細闡述系統的設計過程，涵蓋整體架構與設計目標，依據 MIAT 方法論的階層化設計原則，逐層設計與描述各模組的功能、資料流與控制邏輯，並實際透過 GRAFCET 進行高階合成，展示系統的實現步驟與流程。

3.1 系統架構

本系統選擇使用 ReActNet 作為特徵提取網路，主要原因在於其作為二值卷積神經網路，在權重和活化值的二值化上具有顯著優勢，能夠大幅減少計算成本與記憶體需求，特別適合資源受限的嵌入式系統場景。與傳統實數卷積神經網路 (Real-valued CNN) 相比，ReActNet 在保留較高準確率的同時，通過簡化運算過程降低了硬體資源的使用量。此外，其經優化的活化函數和結構設計，能高效提取資料特徵，成為應對邊緣運算需求的理想選擇。然而，在傳統的二值卷積神經網路 (BCNN) 中，全連接層由於其高參數量和計算密集度，成為整體系統中的資源消耗瓶頸。儘管硬體實現中通常避免浮點運算，但全連接層的結構特性仍導致大規模的資料存取和矩陣運算，對硬體資源需求極高。

另一方面，XGBoost 是一種基於決策樹的梯度提升演算法，其樹狀結構會透過梯度和葉節點的損失函數計算，動態決定最佳分裂點，使模型能有效地找到資料的最優分界點。作為分類器，XGBoost 不僅具有高效的並行處理能力，能在硬體實現中充分發揮性能優勢；同時，通過集成多棵決策樹，得以提升模型的泛化能力，進一步提高分類準確率。此外，XGBoost 的設計對硬體資源的需求相對較低，相比傳統全連接層可以用更低的計算成本完成相同的分類任務。因此，XGBoost 成為取代 ReActNet 全連接層的理想解決方案。

文獻研究[32] 更進一步表明，將 XGBoost 應用於取代 ReActNet 的全連接層，能顯著緩解全連接層的計算瓶頸，並大幅降低系統計算複雜度和硬體資源需求。具體而言，使用 XGBoost 替代全連接層可以減少 7.14% 的浮點運算 (FLOPs) 需求，並縮減 1.02% 的模型參數量，從而進一步壓縮模型大小，優化嵌入式應用的實現條件。同時，實驗結果顯示，結合 XGBoost 的 ReActNet 架構在 FashionMNIST 基準測試中，其分類準確率較原始設計提高了 1.47%，顯著縮小了二值網路與實數網路之間的性能差距，證明該方法在準確率和資源效率上的平衡效果。

基於上述考量，本系統針對感測器中常見的一維資料處理場景，實現了一個 ReActNet-XGBoost 硬體加速器。

3.1.1 IDEF0 階層式模組化設計

透過 MIAT 方法論，本研究提出之 ReActNet-XGBoost 硬體加速器架構可由 IDEF0 表示如圖 3.1.1，包含 A1 初始化、A2 ReActNet 特徵提取硬體模組，與 A3 XGBoost 分類器硬體。其中，A2 ReActNet 特徵提取模組利用二值卷積運算提取輸入資料之重要特徵，接著將特徵向量傳送予 A3 XGBoost 分類器硬體進一步進行分類，實現流暢完整的分類任務。

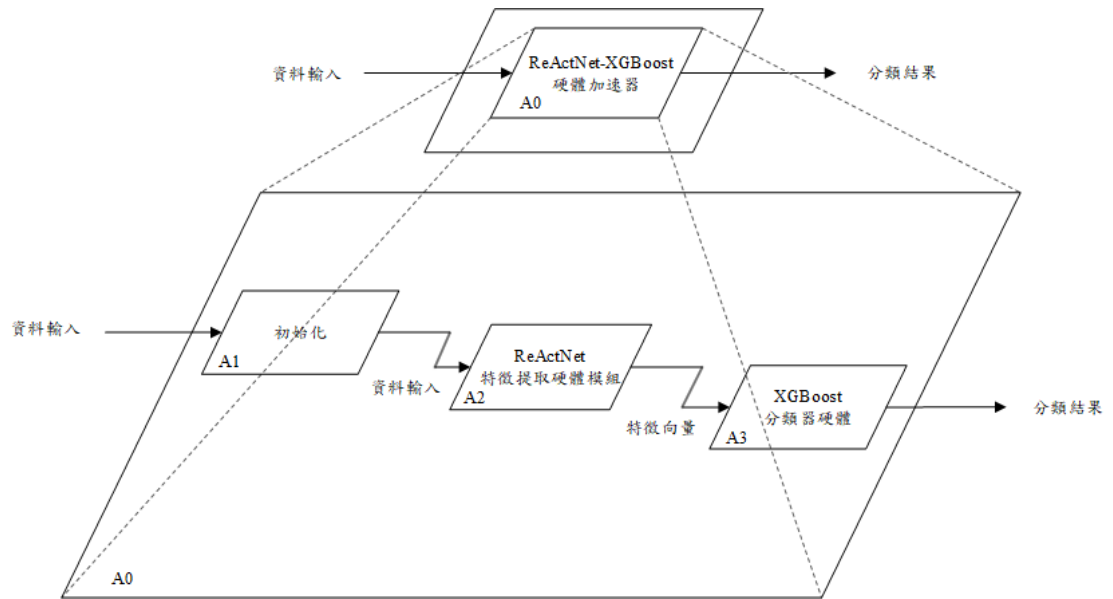


圖 3.1.1 本研究提出的 ReActNet-XGBoost 硬體加速器 IDEF0

3.1.2 GRAFCET 離散事件建模

透過 IDEF0 對系統進行階層式模組化的分解後，ReActNet-XGBoost 硬體加速器的 GRAFCET 離散事件建模可表示如圖 3.1.2。其中，X0 狀態表示系統處於閒置階段，待條件 start 成立後，狀態將進行轉移，從 X0 轉移至 X1 狀態，初始化系統的各項參數並等待 input 條件的成立。input 條件成立後，狀態會從 X1 轉移至 X2 狀態，開始執行 ReActNet 特徵提取硬體模組。當 ReActNet 特徵提取硬體模組完成後，X24 條件成立，使狀態再一次進行轉移，從 X2 轉移到 X3 狀態，執行 XGBoost 分類器硬體，最後轉移到 X4 狀態，表示系統的一次流程已運行完畢，接著回到 X0 狀態等待系統下一次的運行。

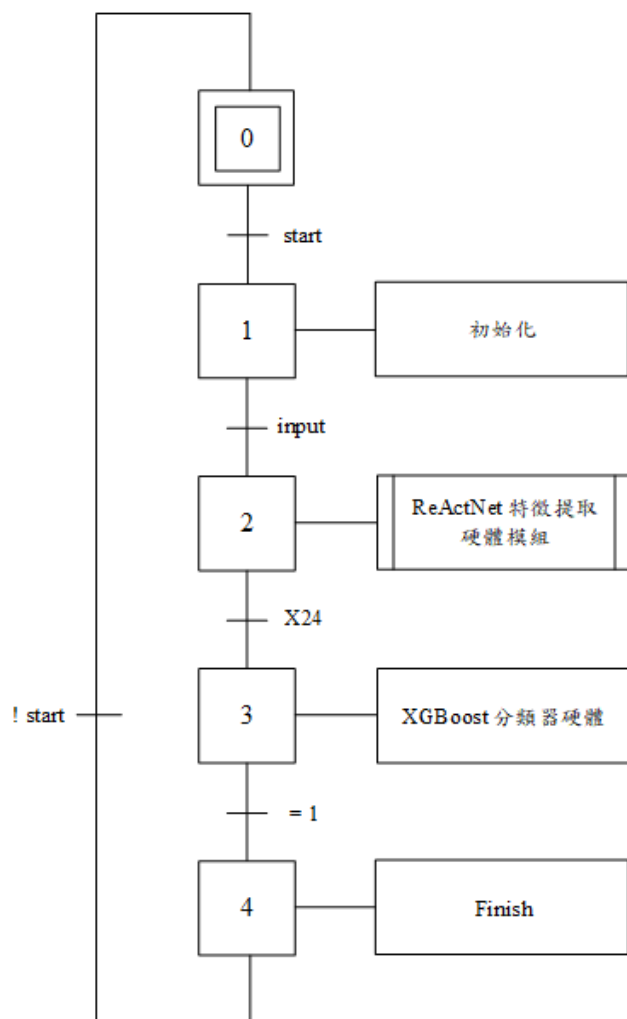


圖 3.1.2 本研究提出的 ReActNet-XGBoost 硬體加速器 GRAFCET

3.2 ReActNet 特徵提取硬體模組

本研究主要以 2.1.3 節所提到的 ReActNet 二值卷積神經網路為參考基礎，改良其架構為適合一維輸入資料的簡化版本，設計出一個二值卷積網路硬體加速器。

具體實現上，同樣使用 MIAT 方法論，首先，以 IDEF0 對架構進行階層化模組式的構想，接著以 GRAFCET 進一步實現各模組的離散事件模型。3.2.1 節將初步以模組的 IDEF0 說明架構，而 3.2.2 節將深入描述模組內部的運算過程。

3.2.1 IDEF0 階層式模組化設計

A2 ReActNet-XGBoost 特徵提取硬體模組的架構如圖 3.2.1 所示，包含 A21 卷積層模組、A22 二值卷積層模組、A23 平均池化層。其中，A21 ReActNet 特徵提取模組利用上述運行流程，此模組得以提取輸入資料之重要特徵，提供後續 A3 XGBoost 分類器硬體進一步使用以完整實現分類任務。

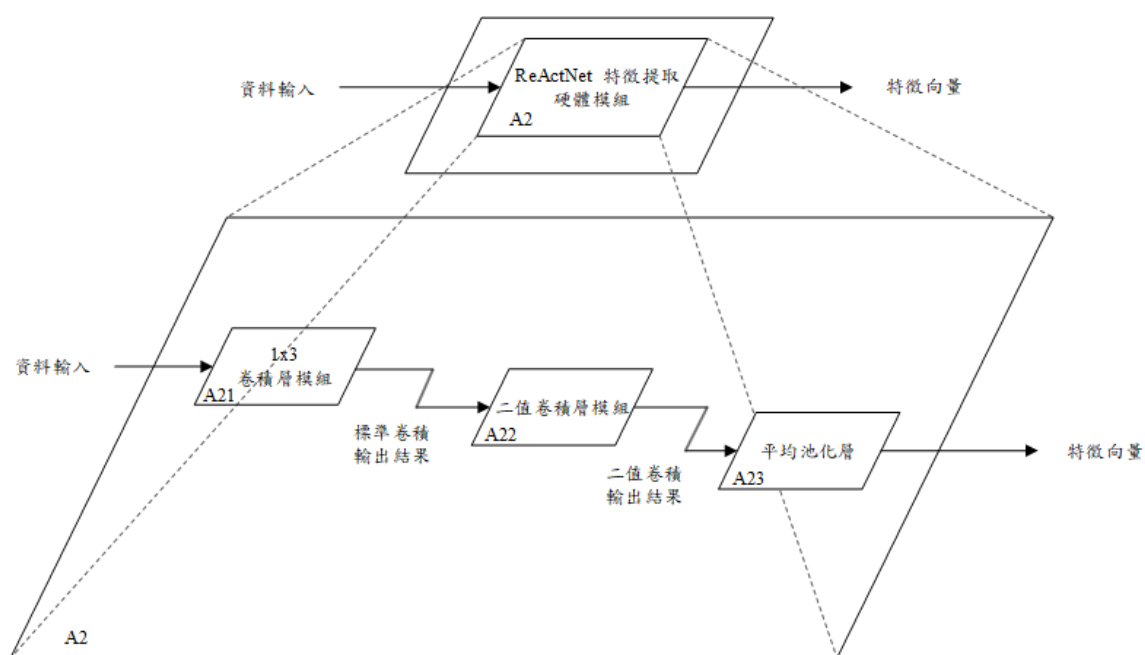


圖 3.2.1 A2 ReActNet 特徵提取硬體模組 IDEF0

3.2.2 GRAFCET 離散事件建模

透過 IDEF0 對 A2 ReActNet 特徵提取模組進行分解後，其 GRAFCET 離散事件建模可表示如圖 3.2.2。其中，X20 狀態表示模組處於閒置階段，待條件 X2 成立後，狀態將進行轉移，從 X20 轉移至 X21 狀態，開始執行 1x3 卷積層模組。當 1x3 卷積層模組運算完成後，X213 條件成立，接著狀態進行轉移，從 X21 轉移到 X22 狀態，開始執行二值卷積層模組。當二值卷積層模組運算完成後，X223 條件成立，使狀態再一次進行轉移，從 X22 轉移到 X23 狀態，執行平均池化層的運算。待平均池化層運算

完畢，X23 條件成立，狀態將轉移到 X24，表示模組的一次流程已運行完畢，最後，回到 X20 狀態等待模組下一次的運行。

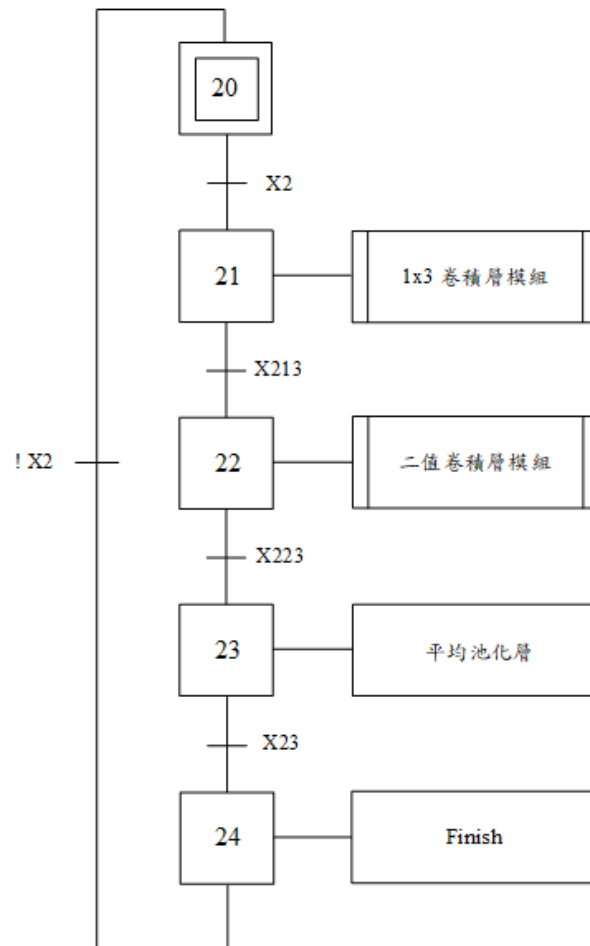


圖 3.2.2 A1 ReActNet 特徵提取硬體硬體模組 GRAFCET

(1) 1x3 卷積層模組

1x3 卷積層模組的 GRAFCET 如圖 3.2.3 所示。當條件 X21 成立時，模組自狀態 X211 開始進行卷積運算，接著轉移至狀態 X212 進行批次正規化運算，以上運算完成後轉移到 X213，表示模組的一次流程已運行完畢。

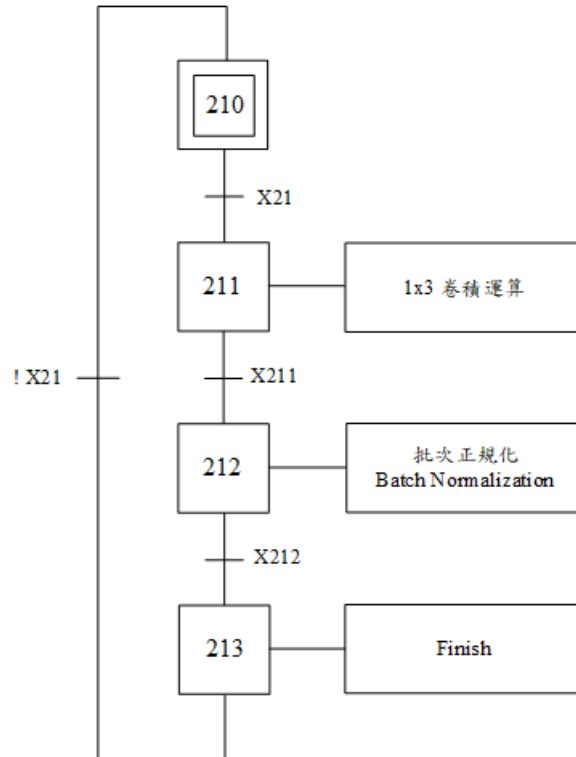


圖 3.2.3 1x3 卷積層模組 GRAFCET

(2) 二值卷積層模組

二值卷積層模組的 GRAFCET 如圖 3.2.4 所示。當前一 1x3 卷積層模組運行完畢並輸出結果後，此模組將依序執行狀態 X221 的 1x3 二值卷積層模組，與狀態 X222 的 1x1 二值卷積層模組，至此完成一次的二值卷積模組層動作。接下來會判斷是否已達到網路所設定之層數，決定要再次回到 X221 狀態或是轉移到 X223 狀態，則模組運行完畢。

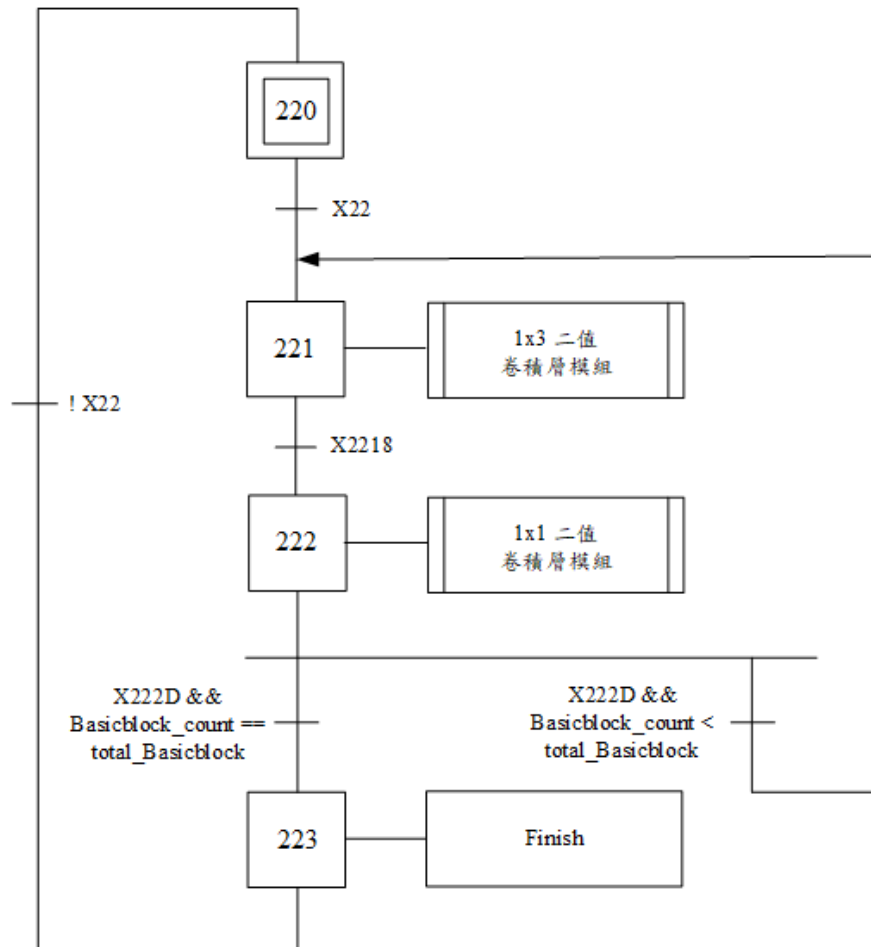


圖 3.2.4 二值卷積層模組 GRAFCET

其中，1x3 二值卷積層模組的 GRAFCET 如圖 3.2.5 所示。模組啟動後由狀態 X2211 首先開始動作，將輸入資料經由式 2.12 所述之 RSign 函數進行二值化，完成後狀態轉移到 X2212，運行一次為因應硬體管線化設計所做的空運算，等待下一批輸入資訊。隨後狀態轉移至 X2213，對二值化後之輸入資訊與 1x3 二值卷積核進行卷積運算，並接著轉移至 X2214 狀態對卷積結果進行正規化運算。接續會判斷 stride 是否為 2，亦即降採樣運算，決定狀態是否需轉移到 X2215 對原始特徵進行平均池化運算，隨後再轉移至 X2216 將卷積結果與池化後之特徵進行加總運算，得到捷徑結果；或是若 stride 不為 2，亦即該層的二值卷積運算非降採樣運算，則狀態從 X2214 直接轉移到 X2216，並計算捷徑結果。當 X2216 狀態結束時，系統狀態將轉移至 X2217，

將捷徑結果經由式 2.13 所述之 RPRReLU 函數得到活化值，最後轉移至狀態 X2218，表示模組運算完成。

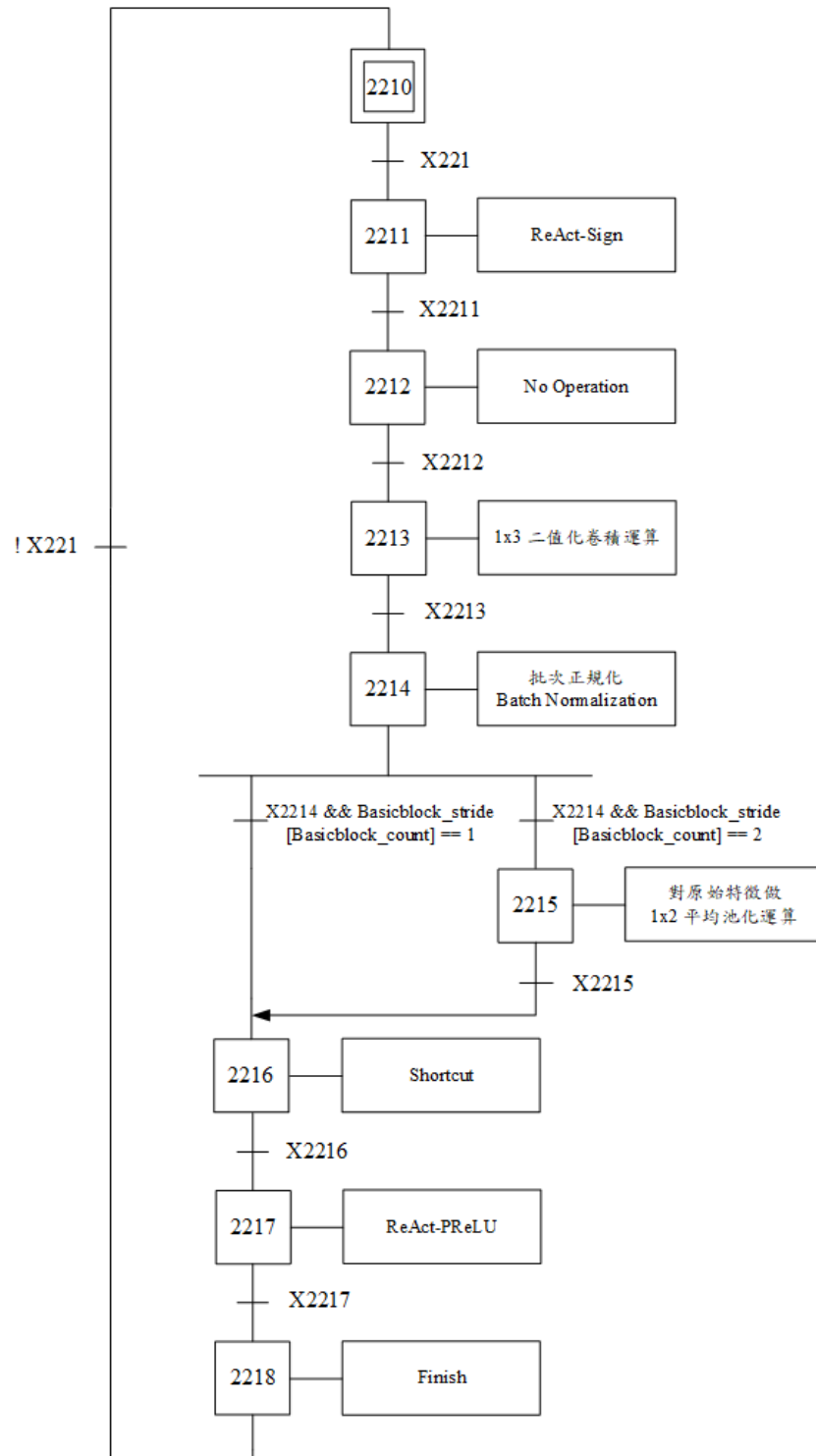


圖 3.2.5 1x3 二值卷積層模組 GRAFCET

而 1×1 二值化卷積層模組的 GRAFCET 則如圖 3.2.6 所示。模組啟動後由狀態 X2221 開始動作，將輸入資料經由式 2.12 所述之 RSign 函數進行二值化，完成後會根據此層之輸入通道數和輸出通道數之間的關係來決定狀態將如何轉移。若輸入通道數與輸出通道數相同，則狀態轉移至 X2222，進行二值化輸入與 1×1 二值卷積核的運算，並依序轉移至狀態 X2223 和 X2224，分別執行批次正規化與捷徑運算，接著狀態將跳至 X222C，經由式 2.13 所述之 RReLU 函數得到活化值，最後轉移至狀態 X222A；若輸入通道數為輸出通道數的兩倍，則狀態將同時轉移至 X2225 與 X2228，兩個平行運行的狀態。狀態 X2228、X2229 與 X222A 執行的運算分別與狀態 X2225、X2226、X2227 相同，當狀態 X2227 與狀態 X222A 的動作均已完成時，狀態將進一步從 X2227 與 X222A 轉移至 X222B，以合併這兩個狀態的運算結果，並接續將狀態轉移至 X222C，將合併後之輸出經由 RReLU 函數計算出活化值，最後轉移至 X222D 狀態。當狀態轉移至 X222D 時即代表此一模組運算完成。

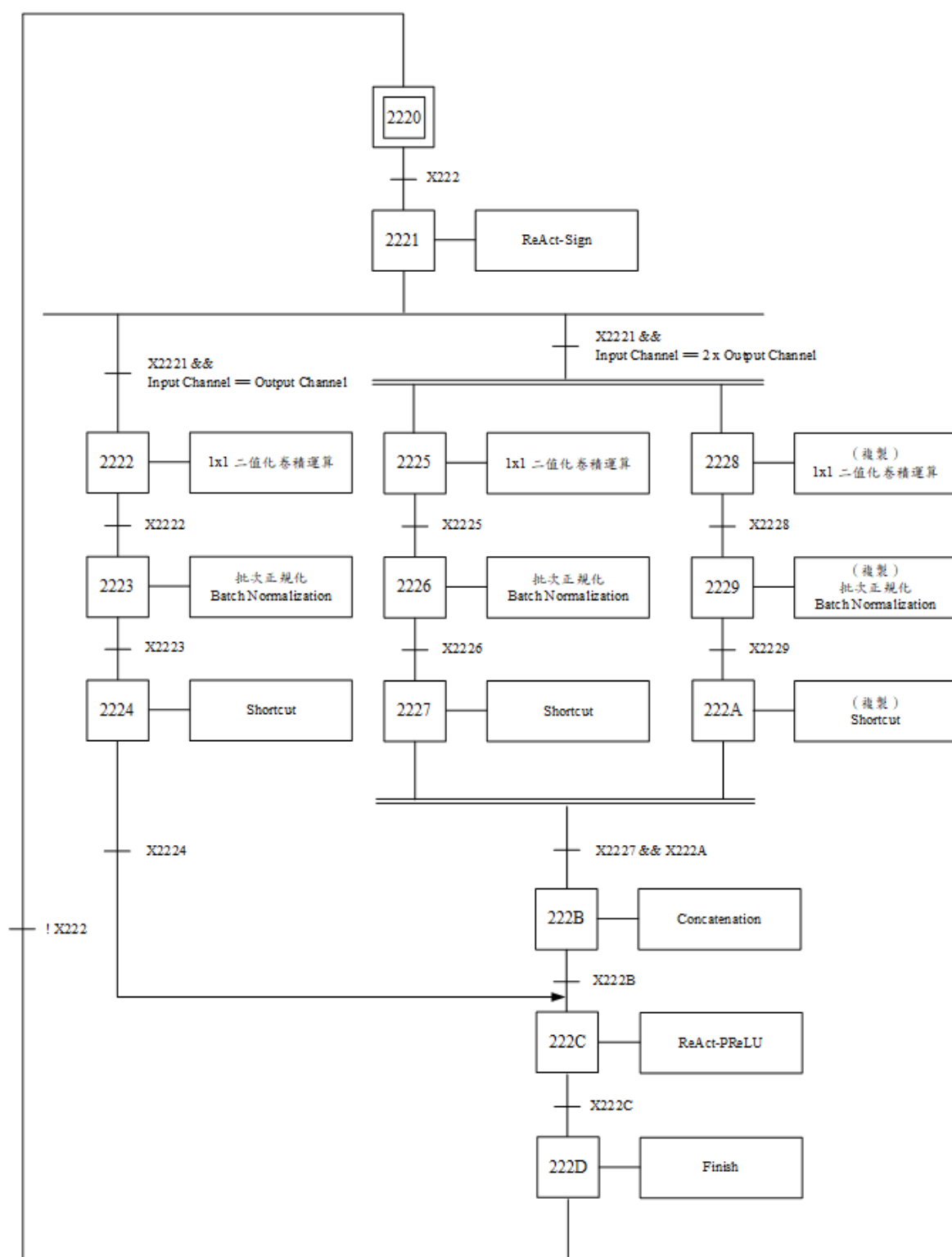


圖 3.2.6 1x1 二值卷積層模組 GRAFCET

3.3 XGBoost 分類器硬體模組

本研究之 XGBoost 分類器硬體主要使用本實驗室發展的一套 XGBoost 硬體化工具生成，接續將對此硬體化工具的具體設計以 MIAT 方法論進行說明。

此工具提供了一套從軟體端 XGBoost 模型生成的樹結構檔 (.json) 轉變成 XGBoost 硬體的流程，透過這個工具，使用者得以更快速、容易地在硬體端實現 XGBoost 模型。

在 3.3.1 節，我們將初步探討此 XGBoost 硬體化工具的設計概念與功能，而後於 3.3.2 節，透過 MIAT 系統設計方法論的 IDEF0 和 GRAFCET 流程，將進一步剖析此工具的階層與內部模組之實際樣貌與實現方式。

3.3.1 XGBoost 硬體化工具

在 XGBoost 硬體設計上，我們主要以 A.Alcolea 等人之研究[30] 為參考基礎。首要目的是希望能遵照該研究採用的格式，在硬體端產生對應的可用 XGBoost 決策樹結構，如此便可以直接使用該研究提供的現有 IP 來進行設計。因此，我們主要依循該研究提及的決策樹格式進行設計，於本文中提到此一規範將統一以「XGB 硬體規格」作為代稱，詳細的格式內容可參考 2.2.1 節。

(1) XGBoost 硬體化設計

我們的開發工作，主要聚焦在實現符合該 XGB 硬體規格的決策樹結構，讓在 Python 中訓練出的決策樹模型得以順利轉為硬體運行。為了達成這項目標，工作內容可分為以下兩部分：①重新訓練 XGBoost 決策樹模型，輕量化原先模型以符合 XGB 硬體規格；以及 ②將重新訓練好的 XGBoost 模型轉成硬體描述語言，以便在硬體上執行。整體流程如圖 3.3.1 所示。

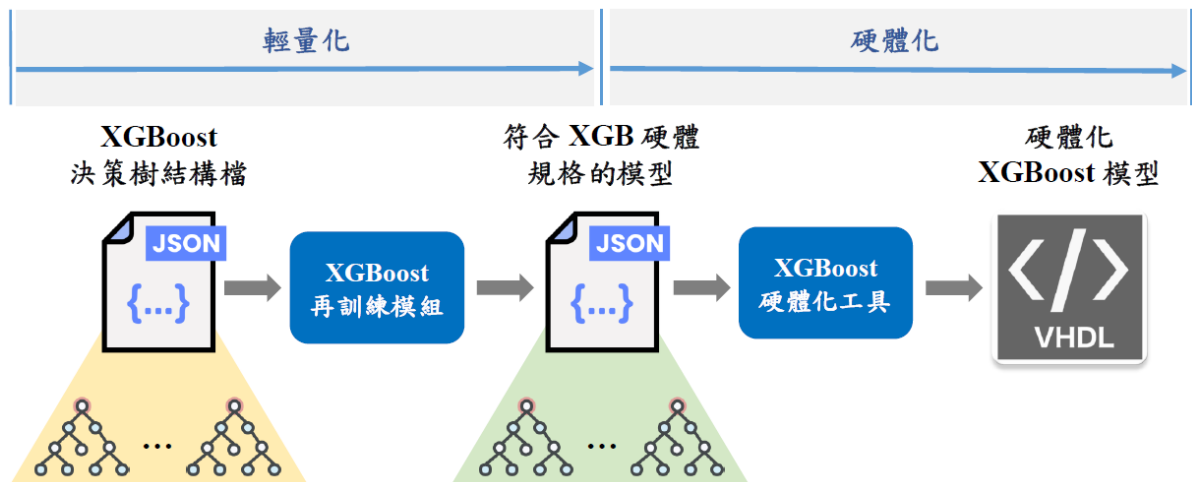


圖 3.3.1 XGBoost 模型硬體化流程圖

(2) XGBoost 再訓練模組

如本文第 2.2.1 節所述，由於 XGB 硬體規格中對節點資訊的存放長度有限，因此，需對原先的 XGBoost 模型進行再訓練以符合需求。其中，主要面臨兩項限制：①節點跳轉地址 (jump address) 的大小受限，需限制單棵樹的節點數量及整體樹的數量；② 每個節點的權重值僅能使用 16 位元表示。

針對第一個限制，我們透過 XGBoost 函式庫中的功能，固定樹的最大深度來限制單棵樹的節點總數，並透過固定最大迭代次數來限制整體樹的數量。使用到的 XGBoost 函式庫功能詳細如表 3.3.1 所示。針對第二個限制，我們則將涉及浮點數運算的部分統一轉換為 16 位元定點數值表示，以符合其規範。

表 3.3.1 XGBoost 在使用 `xgb.train()` 時的部分可控變數

變數名稱	描述
<code>params['max_depth']</code>	單顆子樹的最大深度
<code>num_boost_round</code>	訓練的最大迭代次數，即 class 最大子樹數量
<code>early_stopping_round</code>	早停次數設定，過程中會去抓 <code>custom_metric</code> 中的指標決定是否觸發早停

此外，對於 `tree node` 中 `feature index` 的範圍被限縮最大為 2^8 的限制，我們使用 XGBoost 函式庫內部提供的 `get_score` 功能作為選拔 `feature` 的參照。`get_score` 函式會根據 `feature` 在模型中被使用到的次數乘上所在 `tree` 在整體模型的權重值，得出單一 `feature` 的 `score` 值，該值越大代表其在模型中的重要性越大。透過此功能，我們製作了一個簡單的轉換器，將所有長度超過 256 的輸入特徵降維至所需的大小。

(3) XGBoost 硬體化工具

在接下來的篇幅我們將解析符合 XGB 硬體規格的模型，藉此生成具有決策樹結構資訊的硬體描述語言。我們使用 Python 設計了一套自動化的工具，該工具會透過管理一系列封裝好的類別 (`class`)，讓系統整理具有較高的擴展性與軟體可重用性。在此我們以統一模語言 (UML diagram) 來描述系統中類別與類別之間的關係，如圖 3.3.2 所示。

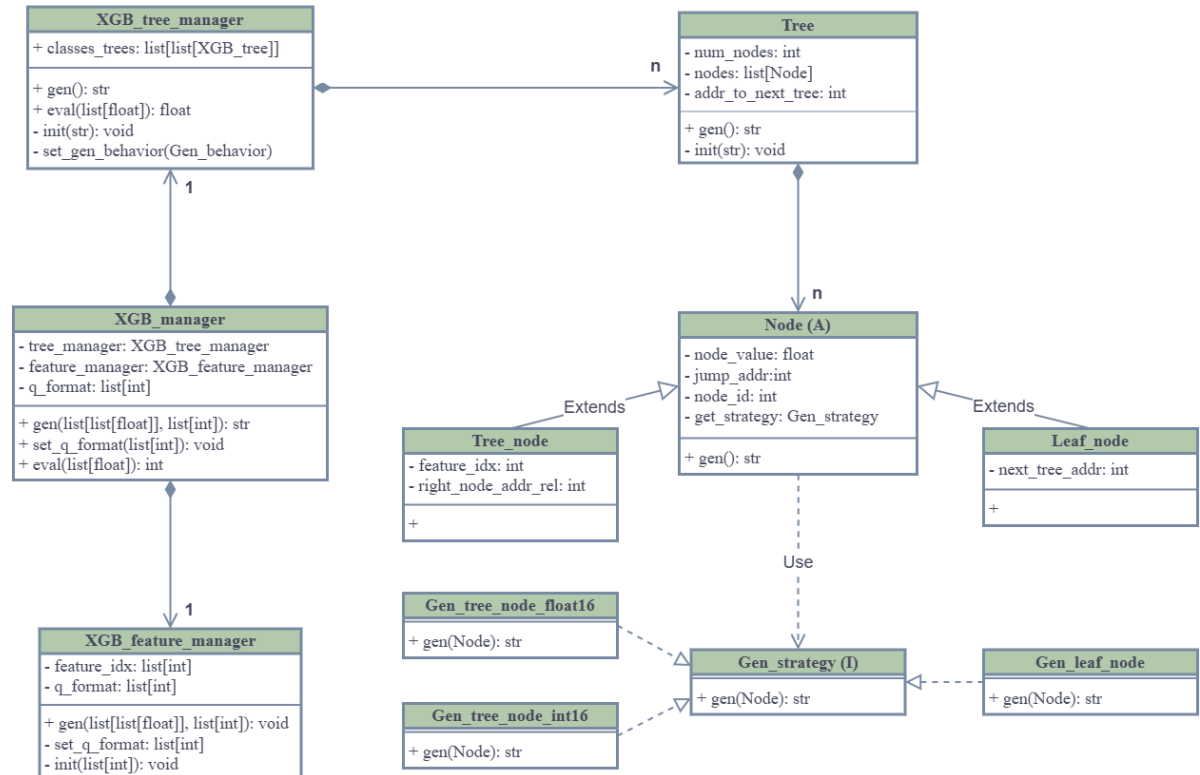


圖 3.3.2 硬體化工具所使用的類別之間的 UML 關係圖

首先是建構 Node 的部分，Node 中會保存 node value、jump address 等資訊，如圖 3.3.3 所示。此外，Node 還會延伸出 tree node 與 leaf node 等子類別，讓 Tree 得以使用多型的方式來呼叫生成程式碼的函式 gen()。gen() 函式的目的在各個類別中的概念都是相同的，其透過自身所載有的資訊來生成對應的硬體描述語言。我們預想 Node.gen() 的變動性在系統中應該較高，因此在實作上會使用 Gen_behavior 這個 interface 來提高系統的可擴充性，當使用者想要生成不同結構的 node 規格時，只需要新增新的 behavior 繼承自 Gen_behavior 就可以達成，具體的例子如圖 3.3.4 所示。

```

@dataclass
class Node():
    node_val:float
    jump_addr:int
    node_id:int
    feature_idx:int
    gen_behavior:Gen_behavior = Gen_behavior()

    def __init__(self, node_val:float, jump_addr:int, node_id:int) -> None:
        self.node_val = node_val
        self.jump_addr = jump_addr
        self.node_id = node_id

    # generate FPGA bit string
    def gen(self) -> str:
        return self.gen_behavior.gen(self, self)

```

圖 3.3.3 Node 類別中的成員與 gen() 功能的定義

```

#
# Leaf node representation:
# |31 - 16|15 - 2| 1 | 0 |
# |- pred_value -|- next_tree -|- is_last_tree -|- is_leaf -|
class Gen_fix16_leaf_node(Gen_behavior):
    # node val + accumulate of nodes num & cur node num + node flag + class func
    def gen(self, _node):
        from node import Node
        last_tree_flag = "1" if _node.is_last_tree else "0"
        return f_to_fix16_b(_node.node_val) + i_to_b(_node.jump_addr, 14) + last_tree_flag + "1"

```

圖 3.3.4 生成 leafnode 的 behavior 例子

xgb_tree 主要載有的資訊有二：所管理的 node 資訊，以及自身在多決策樹中的絕對位置，接續將分別講述其設計方式。首先是 node 的儲存方式，由於 XGB 硬體規格中使用的是 pre-ordering 排列，亦即當 tree node 想要跳到右子樹的位置時需要知道左子樹的大小，為解決此問題，我們使用簡單的 iterative 結合 stack 的方式，透過前序遍歷 (preorder traversal) 的方法建構決策樹。多決策樹中絕對位置的資訊則因為多決策樹

是被連續放在記憶體中，因此，在實作上每棵 `xgb_tree` 在建立前都會去要求上棵樹提供在自己之前的 `node` 數量，藉此使得位址能連續。在圖 3.3.5 中我們可以看到在 `iterative` 的過程中，遇到不同 `Node` 的子類別時會去呼叫對應的建構子並 `push` 到 `nodes` 這個私有成員中，而 `Tree.nodes` 則負責儲存這棵樹的所有節點。

```
node_stack = [0]
while node_stack:
    cur_node_id = node_stack.pop()
    # case : leaf node
    if list_of_lc[cur_node_id] == -1 and list_of_lc[cur_node_id] == -1:
        self.nodes.append(LeafNode(list_of_node_vals[cur_node_id],
                                   is_last_tree,
                                   self.addr_to_next_tree,
                                   cur_node_id))
        pass

    # case : tree node
    else :
        # add new tree node to self.nodes
        self.nodes.append(TreeNode(list_of_node_vals[cur_node_id],
                                   self._get_lc_tree_size(cur_node_id)+1,
                                   node_ref_feature[cur_node_id],
                                   cur_node_id))

        # push lchild and rchild to stack
        node_stack.append(list_of_rc[cur_node_id])
        node_stack.append(list_of_lc[cur_node_id])
        pass
```

圖 3.3.5 `Tree.init()` 的核心功能

由於先前已經將 `node` 多型，當 `xgb_tree` 在實現 `gen()` 功能時只需要依序呼叫每個 `node` 在父類別中定義好的 `gen()` 功能即可，如圖 3.3.6 所示。

```
def gen(self) -> str:
    vhd_code = ""
    for idx, node in enumerate(self.nodes):
        vhd_code += self._addr_deco(node.gen(), self.cur_tree_addr + idx)
    return vhd_code
```

圖 3.3.6 `Tree.gen()` 功能

我們先將多個 `xgb_tree` 結合起來構成 `xgboost` 中的 `class` 模組，再將多個 `class` 模組結合起來構成 `xgboost` 整體的模型，至此決策樹部分的功能便已經完成得差不多了。其中，所有的 `class` 模組會透過 `XGB_tree_manager` 進行管理。

除了決策樹之外，由於送入 `XGBoost` 硬體模型時一樣需要將特徵資料轉成 `bit string`，因此，我們也一併將該功能整合進系統中。這項功能主要由 `XGB_feature_manager` 負責實現，其功能模組會接收特徵資料與標記資料，並以 `bit string` 的資料形式結合進硬體描述語言中。

而我們會使用 `XGB_manager` 來管理 `XGB_tree_manager` 跟 `XGB_feature_manager` 物件，因此，使用者在操作時只需要使用在 `XGB_manager` 所提供的功能函式即可。

至於浮點數格式的問題，由於使用的硬體 IP 在處理比較、加減法等運算時都使用有號整數來做處理，因此，我們也提供自定義調整浮點數轉定點數格式 (Q-format) 的功能，如圖 3.3.7 所示。使用者可以輸入不同的整數與小數位元來改變整體浮點數的精度。此外，在更改 Tree 中 Node 的浮 Q-format 時，由於先前已將 Node 中的 `gen_behavior` 使用 `interface` 的方式來實現，若想要更改 `gen()` 生成的浮點數格式，我們只需要定義新的類別去替換即可。

```
def set_q_foramt(self, _q_format:list[int]):
    class New_leaf_gen_behavior(Gen_behavior):
        # override
        def gen(self, _node):
            return f_to_fix16_b(_node.node_val, _q_format[0], _q_format[1]) + i_to_b(_node.jump_addr, 14) + "0" + "1"
    self.set_leaf_node_gen_behavior(New_leaf_gen_behavior)
```

圖 3.3.7 更改 Q-format 表示法的程式碼片段

3.3.2 IDEF0 階層式模組化設計

模型的硬體化流程可由圖 3.3.8 表示。軟體之決策樹模型資訊檔 (.json) 將透過 `XGB` 再訓練模組，調整為符合 `XGB` 硬體規格限制的模型，接著傳遞給 `XGBoost` 硬體

化工具以生成可與硬體 IP 結合使用之硬體描述語言。

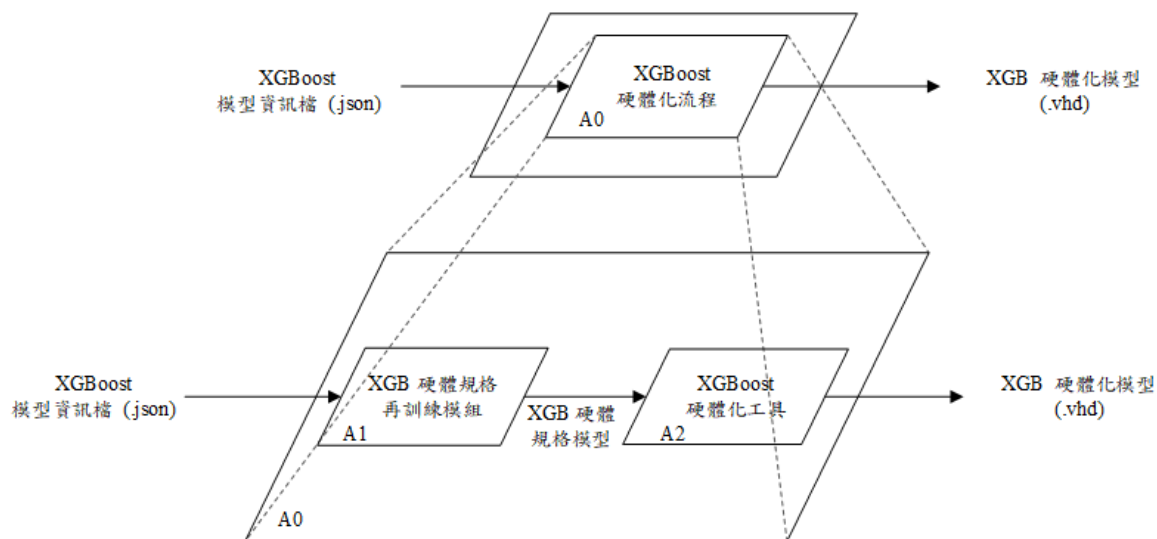


圖 3.3.8 模型硬體化流程 IDEF0 示意圖

XGBoost 硬體化工具的運作流程則可由圖 3.3.9 表示，其中包含 XGB 決策樹結構解析、多決策樹硬體演算法資源，以及硬體決策樹模型產生模組三部分。首先，決策樹結構解析模組會分析輸入的 XGB 硬體規格模型，將必要的資訊提供給後續流程使用；多決策樹硬體演算法資源接收決策樹結構的資訊去修改自身函式庫，使調整後的函式庫可以正常執行我們設計的多決策樹結構，而後作為硬體決策樹模型產生模組的輸入，進一步分析決策樹的結構資訊，最終順利生成硬體描述語言並輸出。

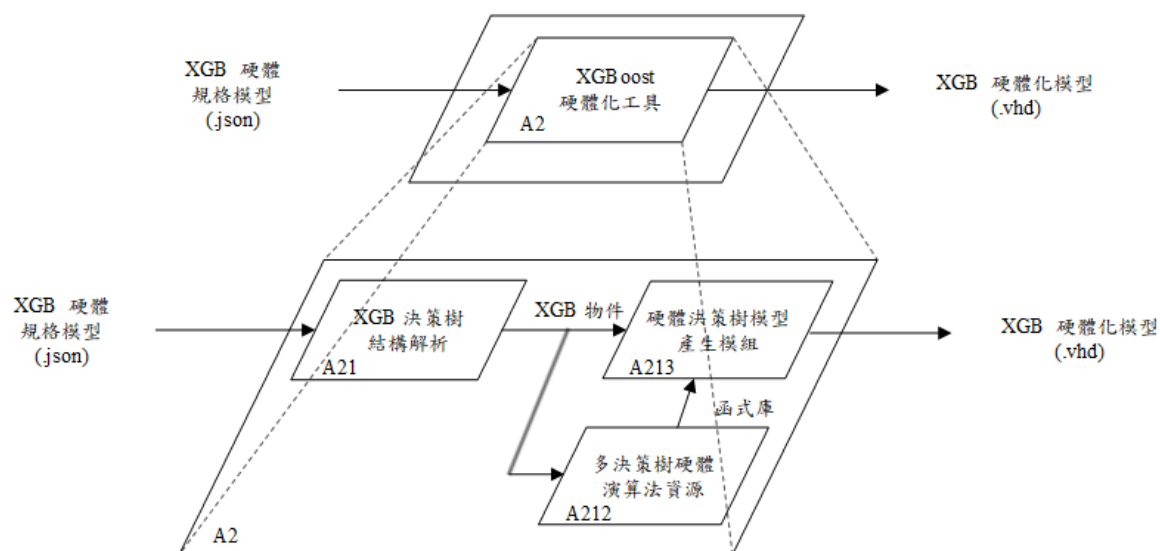


圖 3.3.9 XGBoost 硬體化工具 IDEF0 示意圖

3.3.3 GRAFCET 離散事件建模

模型的硬體化流程 GRAFCET 可由圖 3.3.10 表示。條件成立時，流程便開始運作，最初的 XGB 模型資訊檔 (.json) 首先將輸入至 XGB 硬體規格再訓練模組，重新訓練以符合 XGB 硬體規格。待再訓練模組完成，則轉移至下一個狀態進入硬體化工具，開始執行模型硬體化的任務。

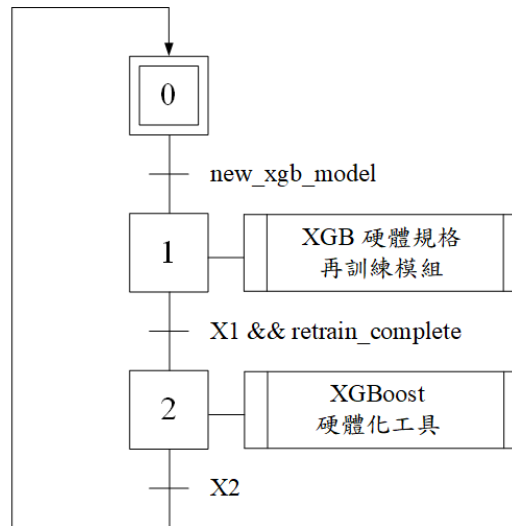


圖 3.3.10 模型硬體化流程 GRAFCET 示意圖

其中，XGB 硬體規格再訓練模組的運作流程如圖 3.3.11 所示。當模組接收到有新的 XGBoost 模型需要處理的訊號便會啟動，首先，狀態從 X10 轉移至 X11 讀取模型的資訊。接著轉移至 X12 狀態，使用 XGBoost 自帶的 `get_score()` 函式透過不同 feature 在 XGBoost 模型分類時被參照的次數乘上子決策樹的權重，計算每個 feature 的重要性，以決定 feature 在 XGBoost 分類時扮演的角色。完成 feature 分數的計算後，狀態會從 X12 轉移至 X13，根據前一階段的分數高低對原先的特徵向量進行降維。

下一步則轉移至 X14 狀態，對 XGBoost 訓練時的超參數進行調整。接著轉移到狀態 X15，使用降維的特徵資訊與前一階段的設定進行模型再訓練，訓練出符合 XGB 硬體規格的模型。最後，轉移到狀態 X216 對模型整體再一次進行性能評估並於 X217 狀態保存。

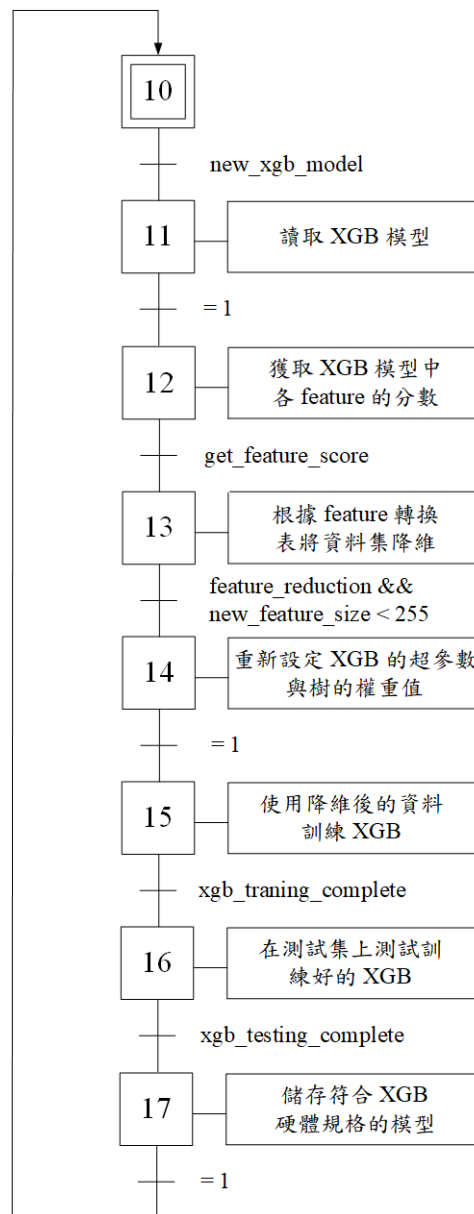


圖 3.3.11 XGB 硬體規格再訓練模組 GRAFCET 示意圖

XGB 硬體化工具的運作流程則如圖 3.3.12 所示。當前一模組輸出符合 XGB 硬體規格的再訓練模型後，此模組就會啟動並轉移到狀態 X21，取出與決策樹的相關的所需資料，過程中我們會建立一個類別：xgb_manager，以管理與儲存 XGBoost 相關資訊。建立完 xgb_manager 後會轉移到狀態 X22，設定我們主要參考的多決策樹硬體演算法資源，其中，我們會更改部分硬體元件結構，使其可以正常運行之後要產生的決策樹資

料結構，並於設定完後儲存成函式庫的形式，提供 `xgb_manager` 使用。當新的函式庫被建立完成後狀態便轉移到 X23，逐步對 `xgb_manager` 的各成員進行初始化，再使用 `xgb_manager` 定義好的功能，產生對應的硬體描述語言。

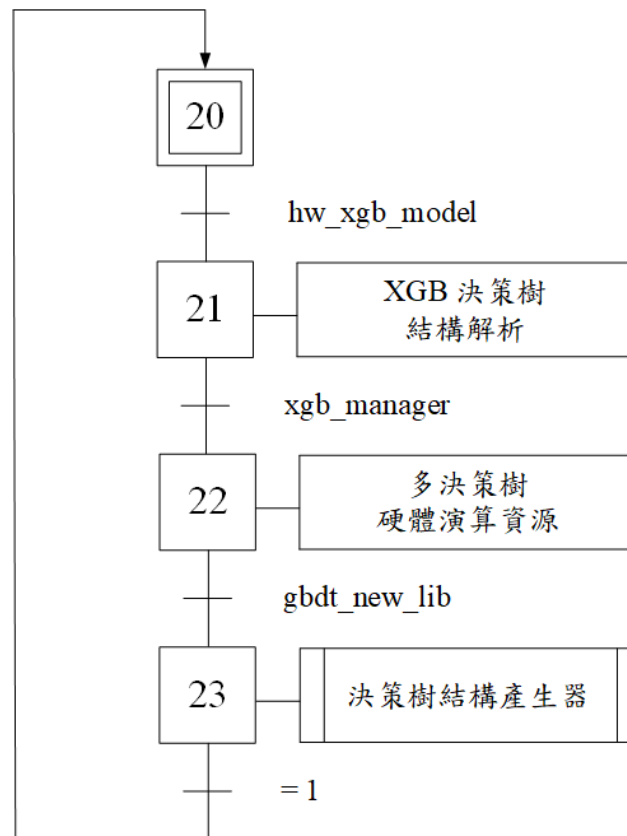


圖 3.3.12 XGBoost 硬體化工具 GRAFCET 示意圖

下圖 3.3.13 展示了 XGBoost 硬體化工具中的決策樹結構產生器 GRAFCET，描述了初始化 `xgb_manager` 到產出硬體描述語言的完整經過。首先，當模組從狀態 X230 轉移到狀態 X231 時，會將 `xgb_manager` 物件使用硬體資源函式庫對自身的成員進行初始化。初始化完成後，會將狀態轉移到 X232 與狀態 X233，為 `xgb_manager` 初始化 `xgb_tree_manager` 跟 `xgb_feature_manager` 這兩個物件，它們的功能是負責產生決策樹結構與特徵資料的硬體描述語言。當兩樣物件都建構完成後，接續會轉移到狀態 X234，設定硬體浮點數的格式，完成後再進一步轉移至狀態 X235。在狀態 X2235 中會呼叫

xgb_manager 的功能函數，如此便可以根據先前在 xgb_manager 中事先初始化好的各成員資料生成對應的硬體描述語言。

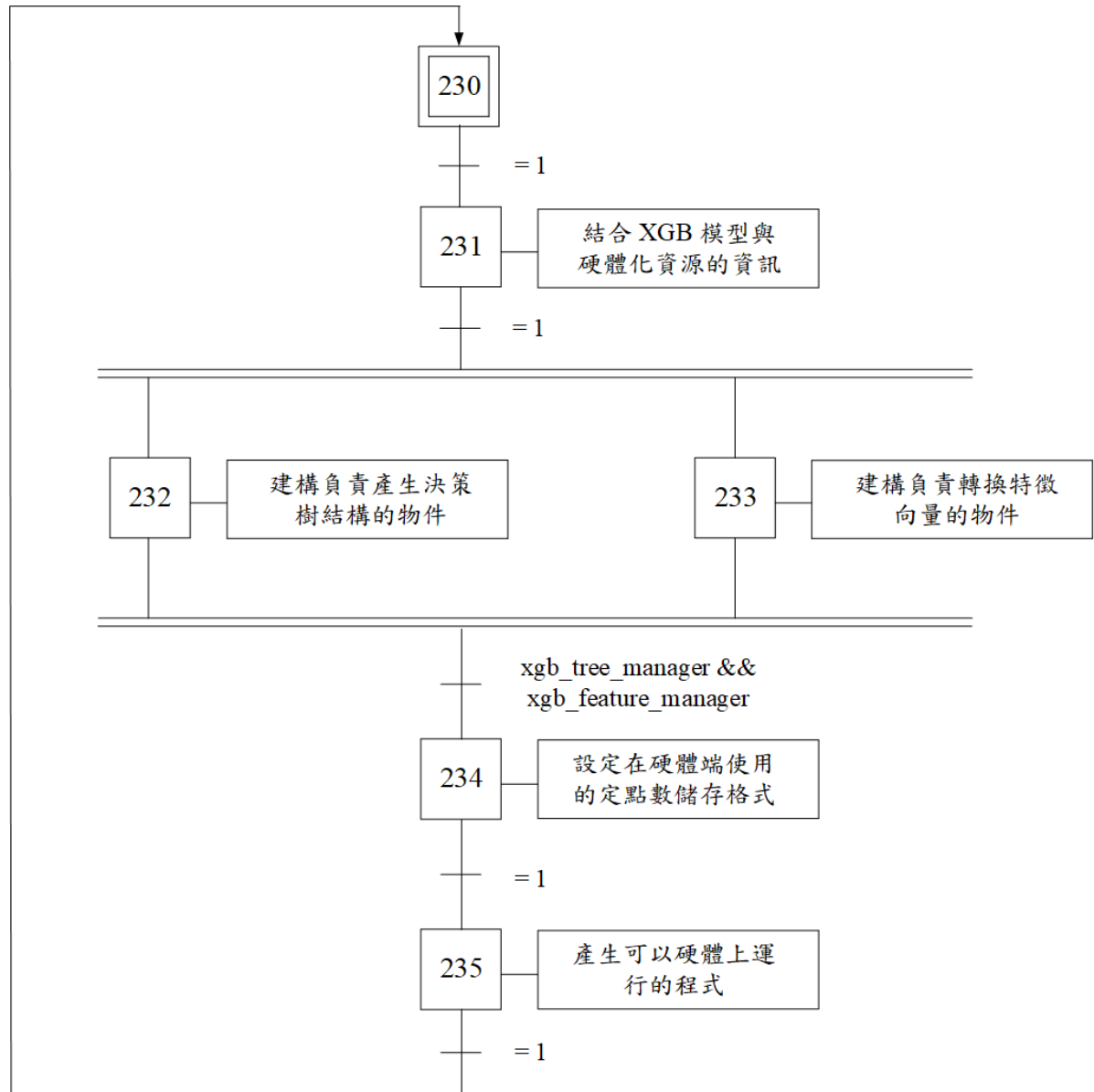


圖 3.3.13 決策樹結構產生器 GRAFCET 示意圖

第四章、實驗結果

本章節以提出之 ReActNet-XGBoost 硬體加速器架構進行驗證分析。首先，我們透過軟體模型進行訓練，獲取權重後則進一步於硬體實現推論，並透過波形模擬驗證每一階段的訊號正確性。

4.1 實驗軟硬體開發環境

本研究的實驗環境如表 4.1.1 所示，其中大多數實驗以伺服器為實驗平台，而波形模擬驗證實驗則是以電腦主機為實驗平台。

表 4.1.1 實驗開發環境

項目	規格
伺服器主機	作業系統：Ubuntu 20.04.6 LTS 64-bits 處理器：Intel® Core™ i9-10900K CPU 記憶體：DDR4 2666MHz 32G SSD：WD WDS100T2B0C-00PXH0 1TB HDD：ST8000DM004-2CX188 8TB 圖形處理器：NVIDIA® GeForce GTX™ 3080
電腦主機	作業系統：Windows 11 企業版 64-bits 處理器：Intel® Core™ i7-4790 CPU 記憶體：12.00 GB DDR3 1600MHz HDD：Seagate ST1000DM010-2EP102 1TB
程式開發環境	Python 3.10.15 PyTorch 2.5.1+cu124 torchvision 0.21.0 NVIDIA® CUDA® 12.4
硬體模擬軟體	ModelSim - Intel FPGA Starter Edition 10.5b

4.2 實驗說明

實驗階段主要可分為軟體層面與硬體層面兩部分，完整流程如圖 4.2.1 所示。在軟體層面，我們首先將資料集輸入 ReActNet 模型進行特徵提取，接著，將其輸出的特徵向量作為 XGBoost 模型的輸入以進行訓練與驗證，此步驟包含進入再訓練模組調整為符合 XGB 硬體規格的模型。最後，藉由硬體化工具將經過調整、符合 XGB 硬體規格的 XGBoost 模型硬體化。而硬體層面上，我們則直接將 ReActNet 進行硬體化，並導入軟體模型訓練出的參數以實現推論，與上述軟體層面的運作流程相似，在完成特徵向量的提取後，我們會將特徵向量作為 XGBoost 硬體的輸入，最終輸出分類結果。

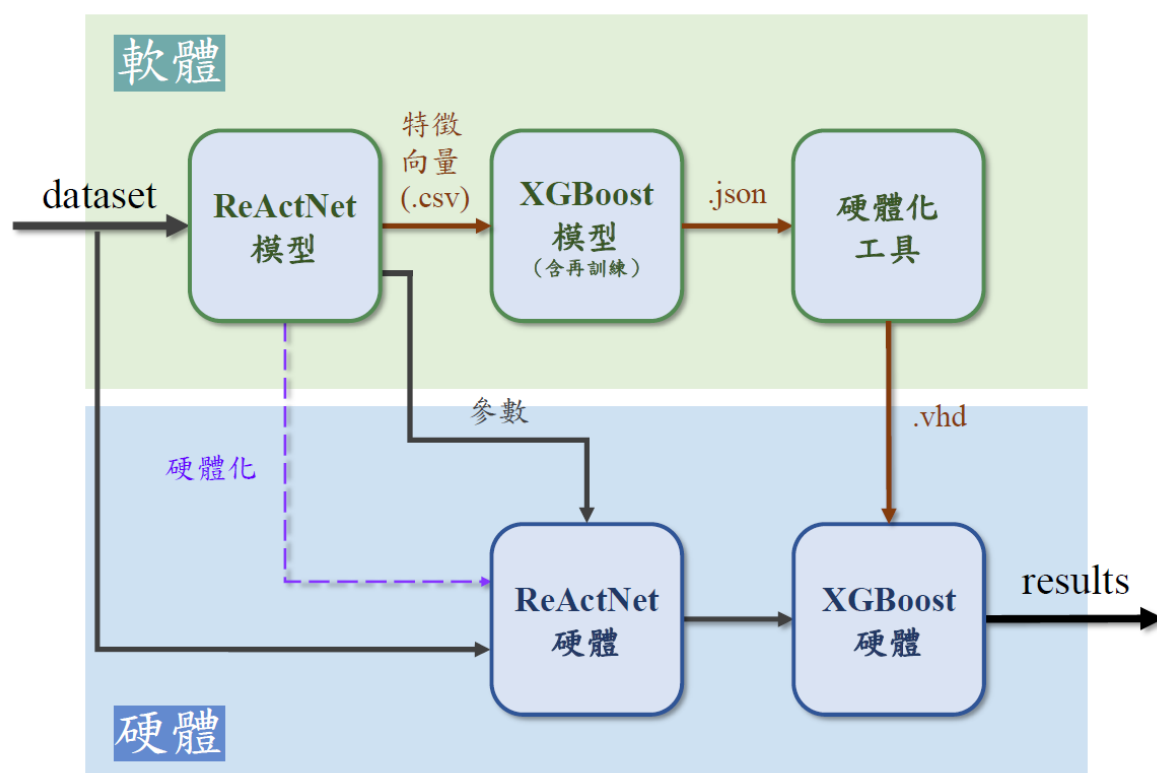


圖 4.2.1 實驗流程示意圖

4.2.1 實驗資料集

因本研究旨在探討針對一維資料設計的分類器架構在不同於以往使用場景中的潛力與實用性，而非精準測量此設計實際帶來的功耗與性能表現，因此，首先考慮到邊緣裝置中最具代表性的智慧醫療應用場景，使用了 Kaggle 開源資料集平台 (<https://www.kaggle.com/>) 上的預處理 ECG MIT-BIH Arrhythmia Dataset 心跳資料集[33] 進行實驗。該資料集共包含 0: N (Normal beat)、1: S (Supraventricular premature beat)、2: V (Premature ventricular contraction)、3: F (Fusion of ventricular and normal beat)，以及 4: Q (Unclassifiable beat) 五個分類項目。然而，由於正常的生理訊號本身較容易取得，此類資料集經常有樣本極為不平衡的狀況，而我們使用的資料集總樣本分布具體情況則如表 4.2.1 所示。

表 4.2.1 ECG MIT-BIH Arrhythmia Dataset 總樣本分布狀況

樣本類別	樣本數量 (%)
0: N	90587 (82.8%)
1: S	2779 (2.5%)
2: V	7236 (6.6%)
3: F	803 (0.7%)
4: Q	8039 (7.4%)

其中，可以觀察到 N 類樣本數明顯多於其他類別，Q 類樣本則為其次，這樣的樣本不平衡現象可能導致多分類模型在訓練過程中無法有效捕捉少類別的資料特徵。為緩解此一問題並聚焦於研究的核心目標，在實驗階段我們決定專注於區分樣本所表示的正常或異常狀態，將原始的多分類問題重新定義為異常偵測問題，並改以類別 0 表示正

常狀態、類別 1 表示異常狀態，如表 4.2.2 所示。這一簡化不僅有助於緩解樣本不平衡的影響，也使我们得以更加專注於探討此類針對一維資料的分類器架構是否具備潛力與實用性。

表 4.2.2 重新定義之樣本狀態與類別區分方式

表示狀態	樣本類別
正常 (0)	0: N
異常 (1)	1: S
	2: V
	3: F
	4: Q

4.2.2 實驗細節補充說明

本研究設計了一個 ReActNet-XGBoost 硬體加速器，其中，由於浮點數運算與定點數運算相比需耗費更多的硬體資源，在我們的應用場景中並不適當。因此，本研究在實驗階段採用 Q2.6 定點數格式，以 2 位元表示整數部份、6 位元表示小數部份的方式進行。

4.3 ModelSim 波形模擬驗證

訊號波形模擬是硬體驗證中重要的一環，透過直觀且高效的模擬環境，設計者能夠清楚地觀察從輸入到輸出間的訊號傳遞、資料傳輸過程，以及控制訊號的觸發狀況是否合理正確，從而快速檢測並深入模組內部確認各項細節。因此，本研究透過 ModelSim

波行模擬作為實驗的最後驗證步驟，全面檢查各階層之間的資料是否正確交換，並獲取最後的分類結果狀況。這項步驟不僅確保了硬體設計的功能正確性，也為後續的硬體實現提供了可靠的基礎。

下圖 4.3.1 展示了本系統運行的波形概觀。從圖中可以初步確認各個被拉出檢視的訊號或資料在不同時間點都有實際數值，指示在系統運行的過程中，沒有模組未順利運行、資料傳輸失敗，或是邏輯條件未思慮周全所出現的未知值 (unknown value) 錯誤狀況發生。這證明了系統訊號的傳遞和模組間的資料交換均符合設計預期。

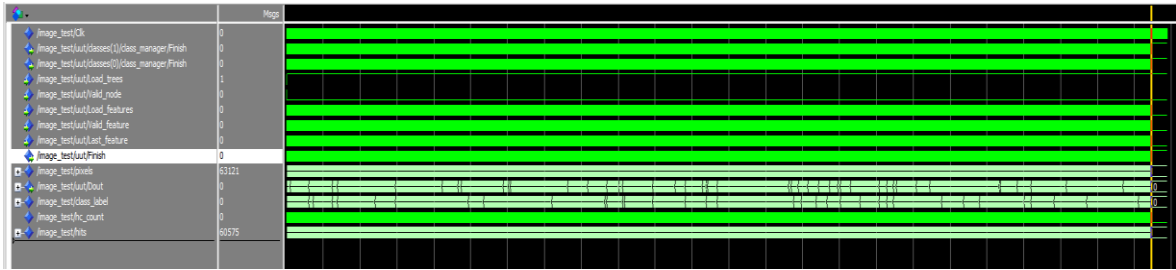


圖 4.3.1 系統波形圖

下圖 4.3.2 則展示了本系統最終實驗結果的波形放大圖。波形末端顯示的數值為系統完整運行後的最終值。圖中由方框進一步標示的兩個數值，自上而下分別為輸入的樣本數與命中 (hit) 數量。其中，由於在硬體化流程中建構 XGBoost 節點時，我們設計的實驗流程包含將資料集中該樣本的正確結果也作為其中一項參考數值輸入，故這裡的命中數量即代表本系統推論的分類結果與原資料集中正確結果相符的次數。因此，經由波形圖中的這兩個數值：63121 與 60576，不僅可推算這次實驗的準確度約為 95.97%，也同時驗證了系統的功能正確性。

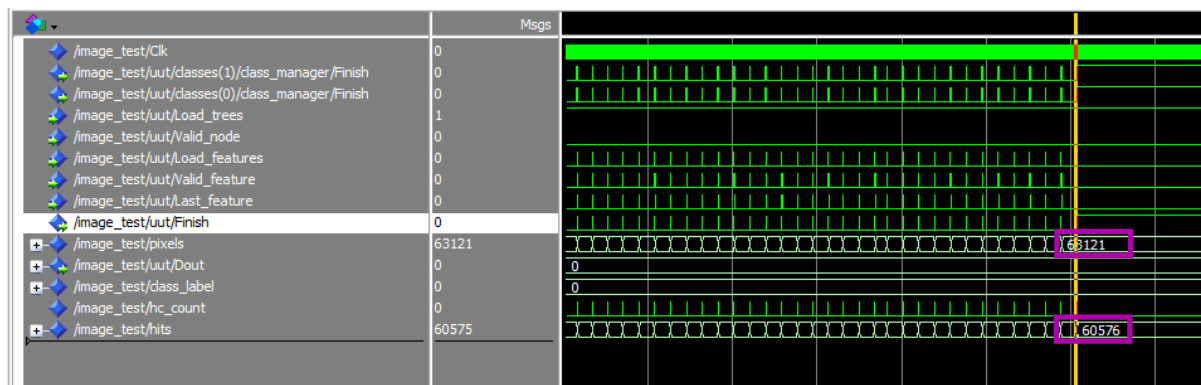


圖 4.3.2 最終實驗結果放大波形圖

第五章、結論與未來展望

5.1 結論

本研究提出了一種基於 ReActNet 結合 XGBoost 的硬體加速器架構，試圖探索出與多數現有系統較為不同，可應用於資源受限場景的系統架構可能性。儘管我們提出的架構目前的設計規模對於低成本且易取得的 FPGA 平台仍有負擔，於現階段無法提供完整的硬體實驗結果，但根據模型縮減帶來的資源需求減少，以及相關文獻的理論支持進行推估，此設計依然展現出適用於資源受限硬體環境的潛力。雖然在實驗與性能比較方面未能詳盡涵蓋與各類現有架構與系統的對比，仍期許本研究能作為參考基礎，為資源受限的應用場景之系統開發引領出一個具備潛力的方向。

5.2 未來展望

本系統目前僅對 ECG 心跳資料集進行設計與實驗，未來可進一步拓展應用範圍。例如，可將本系統應用於 PPG 脈搏訊號或其他一維序列資料的分析，從而涵蓋更多不同訊號的監測場景。此外，本研究於實驗階段暫時採用二分類問題來緩解樣本不平衡的狀況可能對模型帶來的影響，但我們深知在現實世界的應用上，這樣的方法並沒有從根本改善問題，未來將考慮引入高效的資料增強等技術，進一步提高模型的泛化能力。為了提升模型的多樣性與適應性，還可以考慮結合二維影像資料進行研究，透過專業的影像處理技術保留影像的空間特徵，將二維影像轉換為一維特徵序列後作為系統輸入，進一步驗證系統的處理能力與通用性。

另外，本研究所提出的硬體加速器架構已通過波形模擬驗證其功能正確性與設計可行性。儘管現階段無法提供完整的硬體實驗結果，此架構依然展現出適用於資源受限硬體環境的潛力。未來可考慮採用中階或更高階的 FPGA 平台實際燒錄實驗，以獲取更精確的實驗結果，並針對資源配置策略進一步優化，提升整體效率。亦可直接運用 EDA 工具完成完整的晶片設計流程，進一步將系統推向實際部署場域。如此一來，本研究的架構將進一步拓展其實用性與價值，為未來相關的應用需求提供更高效且可靠的解決方案。

參考文獻

- [1] T. Kalsoom, S. Ahmed, P. M. Rafi-ul-Shan, M. Azmat, P. Akhtar, Z. Pervez, M. A. Imran, and M. Ur-Rehman, "Impact of IoT on Manufacturing Industry 4.0: A New Triangular Systematic Review," *Sustainability*, vol. 13, no. 22, p. 12506, 2021.
- [2] T.-W. Sung, P.-w. Tsai, T. Gaber, and C.-Y. Lee, "Artificial Intelligence of Things (AIoT) Technologies and Applications," *Wirel. Commun. Mob. Comput.*, vol. 2021, pp. 9781271:1-9781271:2, 2021.
- [3] S. Soomro, M. H. Miraz, A. Prasanth, and M. Abdullah, "Artificial intelligence enabled IoT: Traffic congestion reduction in smart cities," in *Smart Cities Symposium 2018*, pp. 1-6, 2018.
- [4] I. Rojek and J. Studzinski, "Detection and Localization of Water Leaks in Water Nets Supported by an ICT System with Artificial Intelligence Methods as a Way Forward for Smart Cities," *Sustainability*, vol. 11, no. 2, p. 518, 2019.
- [5] C. Stolojescu-Crisan, C. Crisan, and B.-P. Butunoi, "An IoT-Based Smart Home Automation System," *Sensors*, vol. 21, no. 11, p. 3784, 2021.
- [6] A. Koubaa, A. Aldawood, B. Saeed, A. Hadid, M. Ahmed, A. Saad, H. Alkhoulja, A. Ammar, and M. Alkanhal, "Smart Palm: An IoT Framework for Red Palm Weevil Early Detection," *Agronomy*, vol. 10, no. 7, p. 987, 2020.
- [7] M. J. O'Grady, D. Langton, and G. M. P. O'Hare, "Edge computing: A tractable model for smart agriculture?," *Artificial Intelligence in Agriculture*, vol. 3, pp. 42-51, 2019/09/01/ 2019.
- [8] A. W. Sardar, F. Ullah, J. Bacha, J. Khan, F. Ali, and S. Lee, "Mobile sensors based platform of Human Physical Activities Recognition for COVID-19 spread minimization," *Computers in Biology and Medicine*, vol. 146, p. 105662, 2022/07/01/ 2022.
- [9] W. Li, Y. Chai, F. Khan, S. R. U. Jan, S. Verma, V. G. Menon, Kavita, and X. Li, "A Comprehensive Survey on Machine Learning-Based Big Data Analytics for IoT-Enabled Smart Healthcare System," *Mobile Networks and Applications*, vol. 26, no. 1, pp. 234-252, 2021/02/01 2021.
- [10] M.-Y. Lin, C.-H. Chen, Z.-B. Dong, and C.-C. Chen, "Gigabit Modbus user datagram protocol fieldbus network integrated with industrial vision communication,"

Microprocessors and Microsystems, vol. 94, p. 104682, 2022/10/01/ 2022.

- [11] J. Wang, D. Wang, S. Wang, W. Li, and K. Song, "Fault Diagnosis of Bearings Based on Multi-Sensor Information Fusion and 2D Convolutional Neural Network," *IEEE Access*, vol. 9, pp. 23717-23725, 2021.
- [12] R. Zanc, T. Cioara, and I. Anghel, "Forecasting Financial Markets using Deep Learning," *2019 IEEE 15th International Conference on Intelligent Computer Communication and Processing (ICCP)*, pp. 459-466, 2019.
- [13] A. K. Arslan, Ş. Yaşar, and C. Colak, "An Intelligent System for the Classification of Lung Cancer Based on Deep Learning Strategy," *2019 International Artificial Intelligence and Data Processing Symposium (IDAP)*, pp. 1-4, 2019.
- [14] D. Zhang and S. Liu, "Top-Down Saliency Object Localization Based on Deep-Learned Features," *2018 11th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pp. 1-9, 2018.
- [15] J. Kim, S. Chang, and N. Kwak, "PQK: model compression via pruning, quantization, and knowledge distillation," *arXiv preprint arXiv:2106.14681*, 2021.
- [16] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nature Machine Intelligence*, vol. 1, no. 5, pp. 206-215, 2019/05/01 2019.
- [17] C. Yue and N. K. Jha, "Learning Interpretable Differentiable Logic Networks," *IEEE Transactions on Circuits and Systems for Artificial Intelligence*, 2024.
- [18] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman, "1D convolutional neural networks and applications: A survey," *Mechanical systems and signal processing*, vol. 151, p. 107398, 2021.
- [19] J. Lu, D. Liu, Z. Liu, X. Cheng, L. Wei, C. Zhang, X. Zou, and B. Liu, "Efficient Hardware Architecture of Convolutional Neural Network for ECG Classification in Wearable Healthcare Device," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 7, pp. 2976-2985, 2021.
- [20] Z. Liu, Z. Shen, M. Savvides, and K.-T. Cheng, "Reactnet: Towards precise binary neural network with generalized activation functions," in *Computer Vision—ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIV 16*, pp. 143-159, 2020.

- [21] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785-794, 2016.
- [22] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *Advances in neural information processing systems*, vol. 28, 2015.
- [23] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," *Advances in neural information processing systems*, vol. 29, 2016.
- [24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, 1998.
- [25] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.
- [26] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European conference on computer vision*, pp. 525-542, 2016.
- [27] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.
- [28] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189-1232, 2001.
- [29] Y. Zhang and A. Haghani, "A gradient boosting method to improve travel time prediction," *Transportation Research Part C: Emerging Technologies*, vol. 58, pp. 308-324, 2015.
- [30] A. Alcolea and J. Resano, "FPGA accelerator for gradient boosting decision trees," in *Electronics* vol. 10, ed, p. 314, 2021.
- [31] C.-H. Chen, M.-Y. Lin, and X.-C. Guo, "High-level modeling and synthesis of smart sensor networks for Industrial Internet of Things," *Computers & Electrical Engineering*, vol. 61, pp. 48-66, 2017/07/01/ 2017.

- [32] P. H. Chu and C. H. Chen, "ReActXGB: A Hybrid Binary Convolutional Neural Network Architecture for Improved Performance and Computational Efficiency," in *2024 International Conference on Consumer Electronics - Taiwan (ICCE-Taiwan)*, pp. 327-328, 2024.
- [33] S. Fazeli, "ECG Heartbeat Categorization Dataset", Kaggle, <https://www.kaggle.com/datasets/shayanfazeli/heartbeat>, 2018, (accessed Jan. 2025)