CVSD Final report

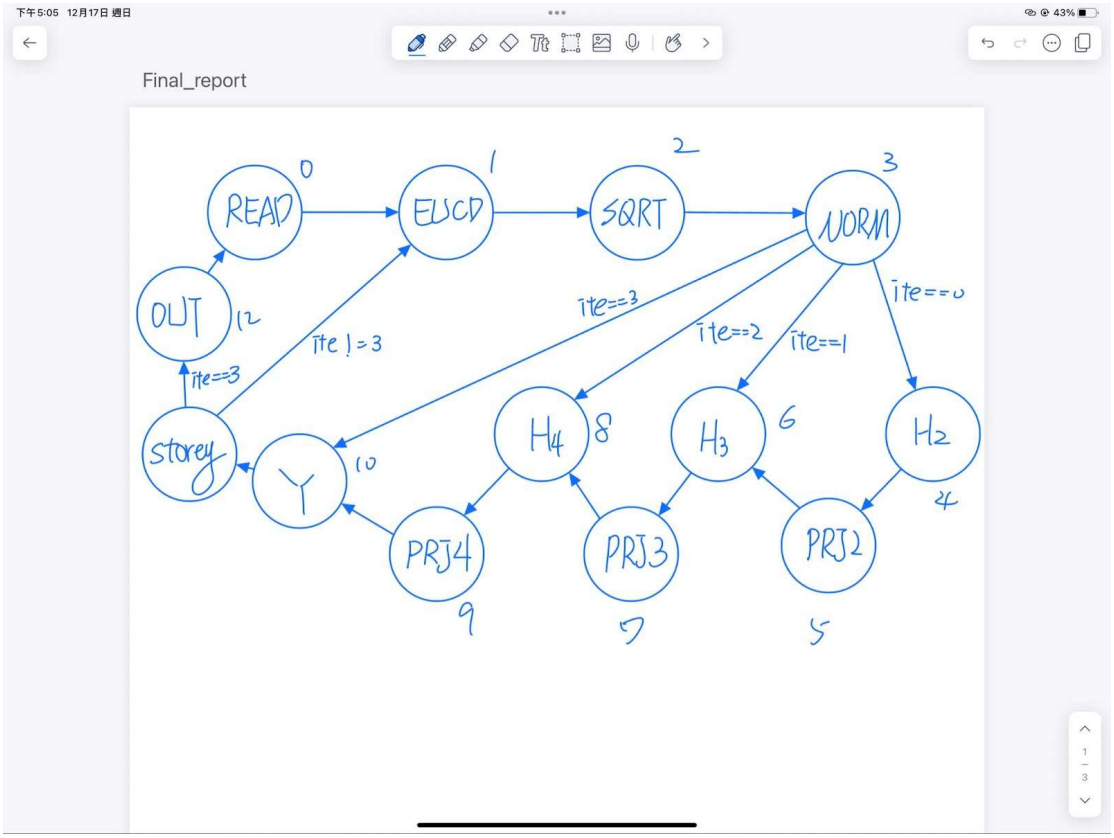
Team 11　　b09901194 陳品睿　b09901081 施伯儒

一、 Algorithm

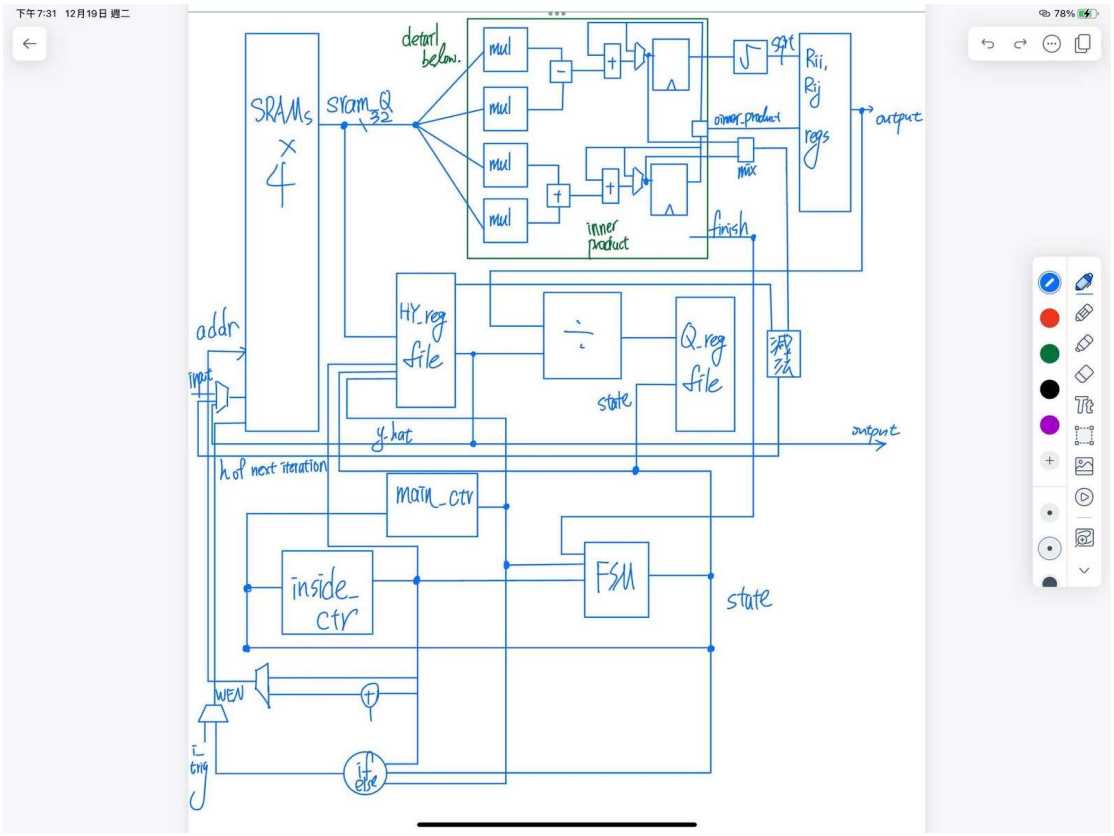
傳統的Gram-Schmidt 演算法在數值計算引入捨入誤差時，CGS 最終產生的 Q 矩陣的行向量正交性效果可能會很差。因此本次專題中我們使用助教提供的 modified Gram-Schmidt 演算法，來降低數值計算造成的誤差。

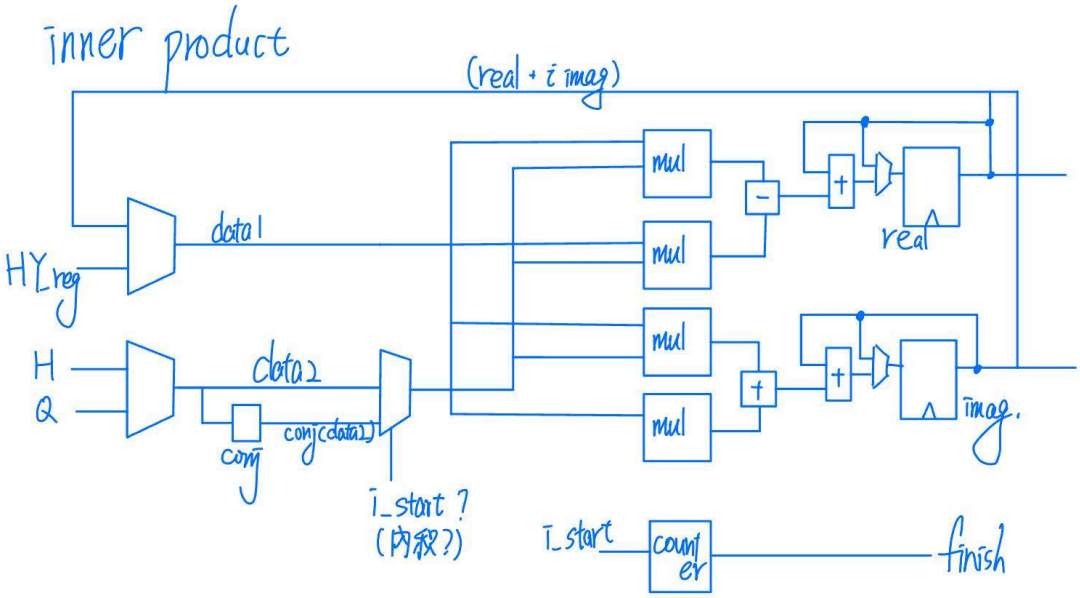
二、 Hardware Implementation

1. Finite State Machine



2. Block Diagram (ps. signal like “r11_nxt = r11” in registers are not drawn)


下午7:31 12月19日 週二
78%
SRAMs
×
4
sram_Q
32
detail below.
mul
mul
mul
mul
inner product
inner_product
sqrt
Rii,
Rj
regs
output
max
finish
addr
input
HY_reg file
÷
Q_reg file
state
y_hat
output
h of next iteration
main_ctr
inside_ctr
FSM
state
WEN
i_trig
idle


inner product
(real + i imag)
data1
HY_reg
H
Q
data2
conj
conj(data2)
i_start ?
(內積?)
mul
mul
mul
mul
real
imag.
i_start
counter
finish

3. Algorithm introduction

首先，會先讀取 200 筆資料並存進 SRAM(S_READ)。接著，會從第一組 PE 開始處理。對於每組資料中，會先計算 iteration 0 的 Euclidean Distance，即利用內積模組計算四個向量的平方總和(S_EUCD)，其中，我們會將讀取出的向量先暫存在 HY 這個 register file，因為接下來計算向量正規標準化時會用到。接著，將這個總和送進 designware 的 sqrt_pipe 這個模組將其開根號並將結果存至 Rii 的暫存器。接著，將暫存在 HY 的向量與 Rii 以 pipeline 的形式讀入 designware 的 div_pipe 做除法得到 Q 並存在 Q_regfile 這個 register file 裡面供之後使用。接下來要做內積與計算正交向量的步驟。但因為不同 iteration 做的次數與內容不一樣，所以會根據當下的 iteration 決定下一步是做 H 的哪一條向量的內積。決定完之後開始從 SRAM 讀取該向量的資料並與 Q 做內積，在讀取的同時，也會將其存在 HY register file 內供稍等使用。做完內積之後，與課本不一樣的地方是我們會先對這個向量做講義中的步驟三，因為此時要使用的 H 剛好被讀出來，所以可以邊計算新的 H 邊把新的 H 存回 SRAM 中，而不用再花費時間從 SRAM 讀取資料，即我們的設計會在步驟二跟三間輪流進行。都做完以後，會計算這個 iteration 的 Y_hat 並存回 SRAM，因為在我的設計中

H 跟 Y_hat 共用 register file。將 Y_hat 存回 SRAM 後，進行下一個 iteration。直到所有 iteration 都做完以後，會再把 Y_hat 從 SRAM 讀出來並與 R 一起輸出。接著進行下一組 PE 的計算。

4. SRAM

我們採用的架構中有 4 個 SRAM，其中兩個儲存實部，另外兩個儲存虛部。對於從 testbench 讀入的數值中，我們會將其整數部分延伸、小數部分縮減使其滿足 S3.12 的格式並存入 SRAM。而對於整個 SRAM，位址 0~199 儲存 testbench 的輸入，而位址 $11010000_2 \sim 11010101_2$ 則會儲存每組 PE 的 y_hat 結果，以便在最後在輸出前讀出。其中， y_hat 的儲存格式為 S3.16，代表一個 y_hat 會占用兩個位址，分別儲存實部和虛部。

5. Designware

dw_sqrt_pipe

我們的設計中針對計算 Euclidean Distance 需要開根號的部分使用這個 designware，增加設計的便利以及彈性和表現。(設計過程中曾經嘗試過 dw_sqrt_seq 這個方案但總體效果不如 dw_sqrt_pipe)

dw_div_pipe

我們的設計中針對做 Normalization 時需要做除法的部分使用這個 designware，這個模組可以連續的輸入被除數與除數，讓需

要除 8 次除法的 Normalization 可以 pipeline 的形式完成，減少了很多時間。

dw02_mult_2_stage

我們的設計中例如內積需要複數乘法，然而因為乘法面積不小，所以盡量地讓不同階段可以共用同一個乘法。然而，這樣乘法前面與後面都會需要一些運算以及判斷讓乘法有正確的讀入資料，使 latency 增加至必須 pipeline。如果不使用 designware，我們只有辦法切在乘法前面或後面，如論何種都會讓其中一邊的 latency 較大甚至可能 violation，因此我們使用這個 designware，讓 EDA tool 判斷切在哪邊或是乘法中間能使效果最好，不僅 violation 問題解決，同時也讓面積降低了。

6. FXP setting

對於所有從 testbench 讀入的資料，我們都會將整數部分延伸、小數部分縮減使其滿足 S3.12 的格式並存入 SRAM 中。

對於 dw_sqrt_pipe，我設定的輸入是 S7.18，並得到 S3.10 的輸出，並在存至 Rii 暫存器時延伸至 S3.16 的形式(雖然後六位貌似在合成時被移除了)。

開完根號後，緊接著要做除法。因為除法沒有小數這個概念，所以要獲得小數的方法就是讓除數與被除數產生位數差如

在被除數後方補 0，如此即可獲得我們需要的小數部分。而我們的形式選擇是將被除數後面補 3 個 0，而除數則閹割至 11 bits，經過反覆實驗後，這個比例可以在 error rate 不超過最高標準的條件下有最好的面積。

而對於乘法，我們採用的是 14 bits * 14 bits 的方式，會如此設計是因為計算多採 S3.16 的方式，也就是如果算太多位後面的位數一樣會被捨棄造成資源浪費，因此剔除太後面比較不會造成影響的位數來增加整體表現。

除了 bit width 的選擇，我們也嘗試了各種除法與開根號的 pipeline 數。因為數量太小無法滿足 setuphold，太大又會讓時間增加。在各種嘗試後，我們最後開根號選擇 4 級 pipeline，而除法選擇了 5 級的 pipeline。

二 討論

7. HW scheduling

作業中遇到小數位過多時會採用 rounding 的方式，但我們的設計中幾乎都採取無條件捨棄的方式以簡化整個設計。

我們在設計整個流程時，使用 excel 把每個階段需要做的事情先列出來，初步了解需要做的事情。之後再隨著如 pipeline 數

或是 SRAM 輸入需要擋一個 stall 才能滿足 timing 等條件進一步修改設計。

	Cycle	SRAM R/W	SRAM addr need	SRAM output	What	What to do	Need	Get	note
ITER	0~3	R	$h_1(0)$	$h_1(0)$	ED	Accumulate $h_1(0)$	IP	$(R_{11})^2$	Store $h_1(0)$ to H-reg
	4				ISR	Inverse square root // let ISR		$1/R_{11}$	
	5				ED	SM(ISR, result of 0~3)	SM	R_{11}	
	6~9	R	y	y	NORM	SM(ISR, $h_1(0)$), Accumulate y	SM	Q11 Q21 Q31 Q41	Get e1
	10	stall				Accumulate y			
	11~14	R	$h_2(0)$	$h_2(0)$	R12	Accumulate IP($h_2(0)$, Q)	IP	R12	Store $h_2(0)$ to reg
	15~18	W	$h_2(0)$	$h_2(1)$ (IN)	$h_2(1)$	$h_2(0)$ – IP(R12, Q)	IP	$h_2(1)$	Store $h_2(1)$ to SRAM
	19	Stall	$h_3(0)$						
	20~23	R	$h_3(0)$	$h_3(0)$	R13	Accumulate IP($h_3(0)$, Q)	IP	R13	Store $h_3(0)$ to reg
	24~27	W	$h_3(0)$	$h_3(1)$ (IN)	$h_3(1)$	$h_3(0)$ – IP(R13, Q)	IP	$h_3(1)$	Store $h_3(1)$ to SRAM
	28	stall	$h_4(0)$						
	29~32	R	$h_4(0)$	$h_4(0)$	R14	Accumulate IP($h_4(0)$, Q)	IP	R14	Store $h_4(0)$ to reg
	33~36	W	$h_4(0)$	$h_4(1)$(IN)	$h_4(1)$	$h_4(0)$ – IP(R14, Q)	IP	$h_4(1)$	Store $h_4(1)$ to SRAM

8. Technique sharing for HW improvement

作業四中我有使用 ICG 讓 EDA tool 盡可能合成最好的 clock gating，然而此次設計中若使用 ICG 會讓乘法的暫存器在 gate_level 時有 setuphold 的問題，所以我後來都改成 manual 的形式避開乘法這個地方。

在作業四中可以觀察到 timing, area 和 power 之間有 tradeoff 的問題，而這次更加入了 error rate 進來比較。而我們採取的作法是讓 error rate 比例能在維持最高標準的情況下盡可能壓低運算的 bitwidth，以此減少面積和 power。而 timing 則盡可能讓資料流水線的處理，如從 SRAM 讀取資料的同時做前一筆的內積運算，或是算完一筆就先存進 SRAM 讓整個架構除了 sqrt 和 div 以外盡量不要出現閒置的情況。

我們最一開始本想使用 fast inverse square root 計算 normalization 的部分，但實作後發現精度不如預期，於是趕快將這個部分改成較方便使用的 designware 除法模組。雖然花費了一周設計的 inverse square root 最後沒有成功應用在期末專題上，但這個過程仍讓我們學到了很多有關這方面的知識以及前人的智慧。

在運算過程中，我們設計讓H跟Y共用同一個 module，因為 Y 只會每個 iteration 只會使用一次，如果一直存在 register 會占用到空間。而因為 Y 是輸出，這個 register 必須存在，所以我們在運算過程時將這個空間留給 H 使用，如此一來在計算投影時就可以直接使用上一個 state 讀出來 H，並將新的 H 直接存回 SRAM，減少了面積的浪費。

9. Progress of improving

設計過程中，我們從最初可以運作的版本，一步步優化，如把存取H和 Y 的暫存器合併減少面積；再到把除法的運算以 pipeline 形式運作，再把做投影和內積的複數乘法合併成一個；再把複數乘法替代成 2_stage 的 designware；最後調整 bitwidth 參數與加上 clock gating 讓整體表現更好，終於完成了這份期末專題。

10. Area / Power / Latency report
    Area: 336312.65
    Power: 24.3
    Latency: 1215205