CVSD final report
Team 46

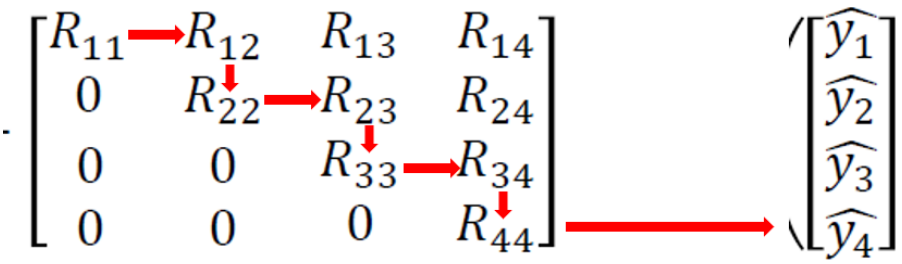
組員: 顏柏聖 R12943009 張柏彥 R12K41003

Algorithm design

QR decomposition design

我們演算法的設計主要是參照介紹投影片的流程來設計的，在每個iteration中把H vector先做標準化為Q vector並得出Rii，再用此更新其他H vector 並得出Rij，把所有Q vector都計算出來後再計算出y hat vector，以此便結束了一筆資料的計算。

對於演算法來說，假設標準化的所需時間大於內積時間的2.5倍時，critical path應為圖一的紅箭頭所示，箭頭未經過的Rij計算可與 Rii同時做計算以節省時間，例如: R12計算完後可馬上開始進行R22的計算，R13與R14可在此時同時進行運算，因為R22與(R13、R14)分別為標準化與內積的運算，兩者使用的硬體並不會發生衝突。



cordic algorithm

CORDIC 是 Coordinate Rotation Digital Computer 的簡稱，主要用於解決三角 函數、反三角函數、平方根與乘除法等運算問題。CORDIC 迭代運算可使用在圓周系統、線性系統以及雙曲系統，並分成旋轉模式與向量模式，提供了一種數學 計算的逼近方法，前述所提之複雜運算分解為一系列的加減和移位運算。

開平方根運算：假設輸入為 a，在雙曲系統的向量模式下，令 x=a+1, y=a-1, 經過13次 CORDIC 迭代運算後，再將 x*0.60374807 即可為逼近後的根號 a，其演算法如下：

Alogrithm 5 CORDIC based square root operation

Input: Input value a ; Iteration counter N
Output: Output value z

$x_0 = a + 1,\ y_0 = a - 1,\ k = 4$
for i = 1,. . . ,N do
  if y<0 then
    $x_i = x_{i-1} + y_{i-1} \ll i$
    $y_i = y_{i-1} + x_{i-1} \ll i$
  else y≥0 do
    $x_i = x_{i-1} - y_{i-1} \ll i$
    $y_i = y_{i-1} - x_{i-1} \ll i$
  end if
  if i==k then
    $x_i = x_{i-1} + y_{i-1} \ll i$
    $y_i = y_{i-1} + x_{i-1} \ll i$
  else y≥0 do
    $x_i = x_{i-1} - y_{i-1} \ll i$
    $y_i = y_{i-1} - x_{i-1} \ll i$
  end if
  k = 3*k + 1
end for
$z = x_N * 0.60374807$

除法運算:假設輸入為x, y, 在線性系統的向量模式下, 經過13次CORDIC迭代運算後, 即可獲得逼近後之y/x, 其演算法如下

Alogrithm 6 CORDIC based division

Input: Denominator x ; Numerator y ; Iteration counter N
Output: Output value z

1: $x_0 = x,\ y_0 = y,\ z_0 = 1$
2: **for** i = 1,...,N **do**
3: **if** y<0 **then**
4: $x_i = x_{i-1}$
5: $y_i = y_{i-1} + x_{i-1} \ll i$
6: $z_i = z_{i-1} - 1 \ll i$
7: **else** y≥0 **do**
8: $x_i = x_{i-1}$
9: $y_i = y_{i-1} - x_{i-1} \ll i$
10: $z_i = z_{i-1} + 1 \ll i$
11: **end if**
12: **end for**

FXP setting
輸入:我們把S1.22的input先裁剪為S1.10, 其實部與虛部的寬度各為12bit, 接著以三個SRAM的8 bit I/O 同時進行存入, 而需要使用時, 把從SRAM讀出的S1.10 data先進行擴充為S3.10再用對應的寬度硬體進行後續的所有運算。
輸出:將計算結果一律處理成S3.16, 整數部分在計算中的寬度都會是3以上, 因此超過3的部分就從MSB直接捨去, 小數部分則有可能比16多或少, 若比16少就往LSB補0, 若比16多就裁切LSB, 將結果裁切為S3.16後才會存入output buffer以便在稍後輸出。

Hardware implementation

Hardware function

```
QR_Engine.v
├── normalizer.v
│   └── cordic.v
├── dot_and_sub.v
│   └── complex_mul.v
└── sram_256x8.v
```

- QR_Engine.v: 將所有操作用FSM包起來
- normalizer.v: 利用cordic來對向量長度平方做根號處理來得出向量長度, 利用cordic來對向量來進行除法以得出標準化向量。
- cordic.v: 進行cordic演算法, 可選擇除法或根號模式
- dot_and_sub.v: 利用複數乘法來計算內積、向量伸縮、向量長度平方, 與利用減法器來計算向量相減。
- complex_mul.v: 計算複數乘法, 可選擇是否讓input做conjugate
- sram_256x8.v: 8 bit I/O SRAM

Hardware scheduling

Input:

在INPUT進來時，若為第一筆矩陣，則直接將其輸入 Reg buffer來進行運算，但由於10筆矩陣是連續輸入的，故在等待第一筆矩陣算完前，會將接著輸入的2~10筆矩陣存在SRAM當中，等到計算完畢後會從SRAM中提取出一筆新的矩陣資料來計算，如此重複到第10筆矩陣也計算完畢後便開始接受新的10筆矩陣。

Normalization (Rii 計算):

依序進行了三個步驟: 計算長度、長度開根號、將vector除以長度

1. 計算長度的部份使用vector與其conjugate進行內積來計算 (dot_and_sub.v)
2. 長度開根號使用cordic來計算 (normalizer.v)
3. 除法也使用cordic來計算 (normalizer.v)

由於以上三個步驟有data dependency，必須依序計算，所以cordic會共用同個硬體

Update H vector (Rij 計算):

依序進行了三個步驟: Q-H內積、Q 長度調整、將H vector與調整後的Q vector相減

1. Q-H內積直接計算內積 (dot_and_sub.v)
2. Q長度調整需要使用複數乘法，故也在同個sub module內進行 (dot_and_sub.v)
3. 相減所需的時間與資源較小，故花1個cycle直接在同個sub module內運算完畢 (dot_and_sub.v)

由於dot_and_sub需兼具內積與update H vector的功能，可拉一條控制訊號來控制其運作模式，但其中乘法器硬體為共用的

Obtain y_hat:

其矩陣乘法實質上為 4個H vector的conjugate與 Y vector各做一次內積，故只使用 dot_and_sub.v

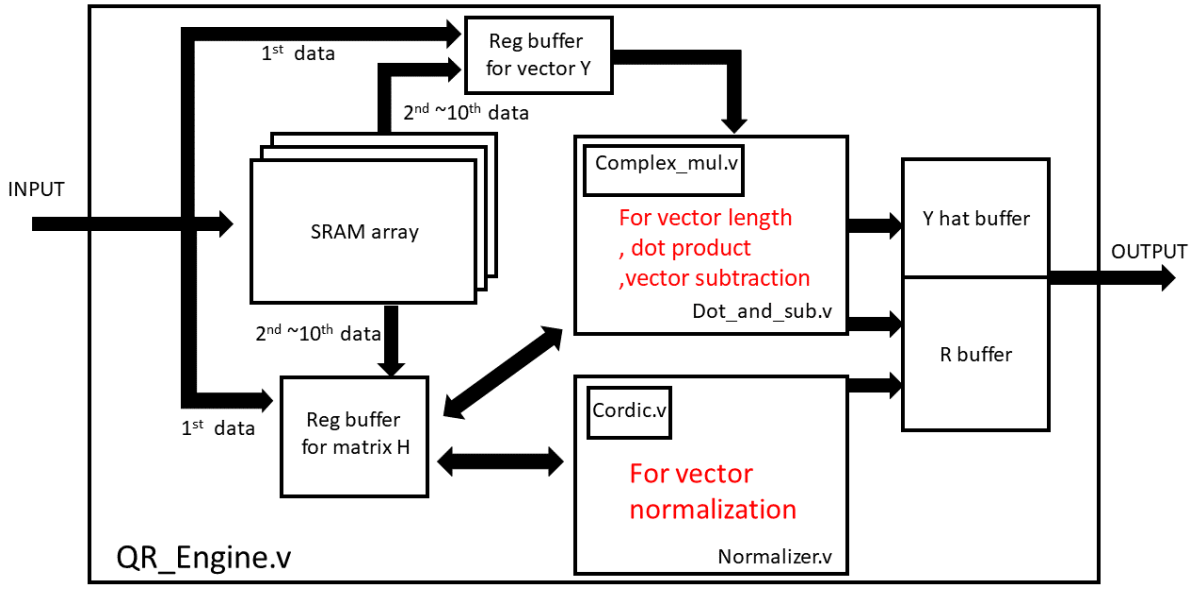
Output:

將R buffer與Y hat buffer裡的資料輸出

各種operation與對應的使用硬體如下表所示，可看見橘色部分為dot_and_sub.v為閒置的時間段，因此我們可以照第一頁algorithm design中所述的方式將一部份的Update H vector(Rij 計算)移至橘色的時間段來進行計算以節省時間。

	Normalization		Update H vector	Obtain Y hat
Operation	Vector length	Sqrt and div	Update H	Obtain Y hat
Normalizer.v		busy		
Dot_and_sub.v	busy		busy	busy

Block diagram



Technique sharing

Input prefetch:

在輸入第一筆資料時，因為此時的運算還沒開始，與其將第1筆資料存進SRAM，不如直接開始對其進行運算，而2~10筆資料則必須要等待第1筆資料計算完畢，所以只能讓SRAM來進行儲存。

hardware sharing:

為了節省資源，我們的dot_and_sub.v與normalizer.v等submodule各使用一個而已，當多筆資料要使用同一種計算的話就要排隊等待前面的資料計算完，雖然所花費的時間較長，但area與leakage power都會下降，故我們採用這種策略。

buffer sharing:

而在H matrix計算的過程中，Q vector與更新後的H vector皆會寫回去H matrix buffer以重複使用buffer space，當所有Q vector都計算完後，此時的H matrix buffer便為Q matrix，可以用此與y buffer直接來進行 y hat 的運算。

Clock gating:

我們使用CVSD講義所提到的Manual CG方法，對所有bit width大於4bit的regsiter做CG，最後達到97.65%的覆蓋率。

Clock Gating Report by Origin

	Actual (%) Count
Number of tool-inserted clock gating elements	473 (100.00%)
Number of pre-existing clock gating elements	0 (0.00%)
Number of gated registers	1867 (97.65%)
Number of tool-inserted gated registers	1867 (97.65%)
Number of pre-existing gated registers	0 (0.00%)
Number of ungated registers	45 (2.35%)
Number of registers	1912

此外，我們也嘗試對SRAM做CG，然而講義上提供的Auto CG以及Manual CG無法成功合成ICG cell，而是會直接把i_clk跟en經過AND gate後直接接到SRAM的clk port，因此我們直接instance standard cell library中的ICG cell，最後成功合成出ICG cell，Power的部分也從14mW降低至6mW，減少57.14%的功耗。

```
icg_inst clk_gate_sram_addr_buffer ( .CLK(i_clk), .EN(sram_addr_flag),
    .ENCLK(gclk_sram_addr), .TE(1'b0) );
```

```
sram_256x8 sram_0__s0 ( .Q(sram_data_real[9:2]), .A(sram_addr), .D({n2937,
    n2934, n2931, n2928, n2925, n2922, n2919, n2916}), .CLK(gclk_sram_addr), .CEN(1'b0), .WEN(sram_read_2_) );
sram_256x8 sram_1__s0 ( .Q({sram_data_imag[5:2], sram_data_real_15,
    sram_data_real[12:10]}), .A(sram_addr), .D({n2913, n2910, n2907, n2904,
    i_data[23:22], n2900, n2897}), .CLK(gclk_sram_addr), .CEN(1'b0), .WEN(
    sram_read_2_) );
sram_256x8 sram_2__s0 ( .Q({sram_data_imag_15, sram_data_imag[12:6]}), .A(
    sram_addr), .D({i_data[47], n2894, n2892, n2890, n2888, n2886, n2883,
    n2880}), .CLK(gclk_sram_addr), .CEN(1'b0), .WEN(sram_read_2_) );
```

Before applying clock gate to SRAM

```
  Net Switching Power  = 1.386e-03   ( 9.52%)
  Cell Internal Power  =    0.0130   (89.32%)
  Cell Leakage Power   = 1.675e-04   ( 1.15%)
                         ---------
Total Power            =    0.0146  (100.00%)
```

After applying clock gate to SRAM

```
  Net Switching Power  = 9.989e-04   (16.33%)
  Cell Internal Power  = 4.940e-03   (80.76%)
  Cell Leakage Power   = 1.776e-04   ( 2.90%)
                         ---------
Total Power            = 6.117e-03  (100.00%)
```

Hard-macro placement:

因為input port在CORE的上緣，為了讓SRAM能快速得到input data，我們將3塊SRAM放置於上方，並將port朝下以利下方的cell可以較快用到SRAM的資料。