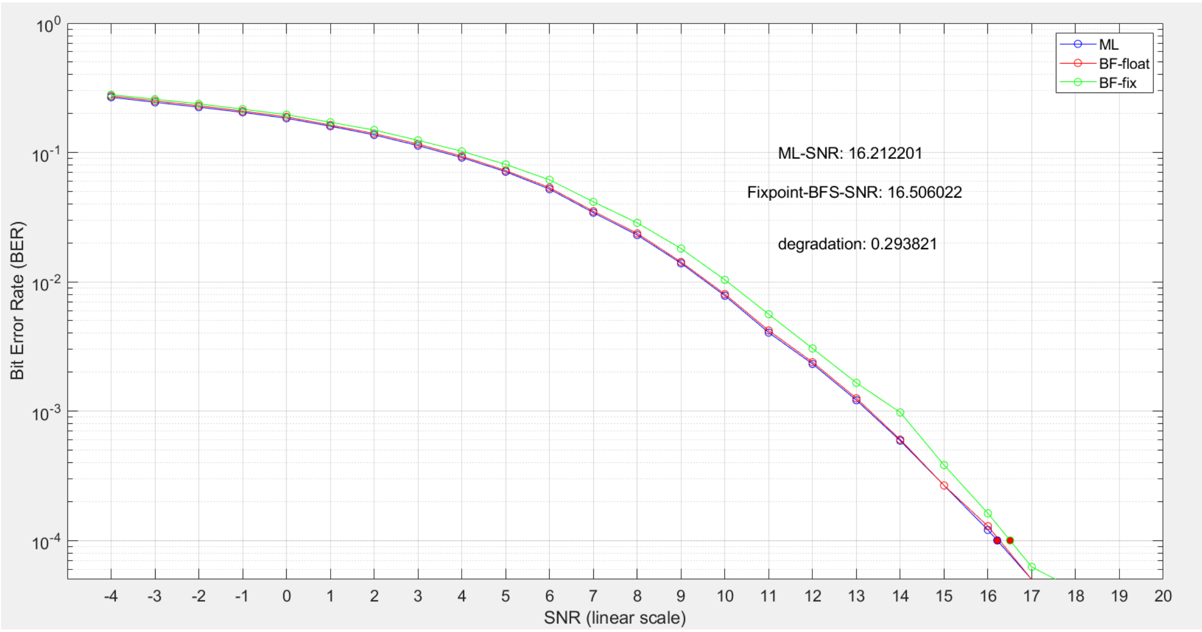
Digital Communication IC Design

Final Project Report

R13943124 施伯儒　B10502076 金家逸

1. The algorithm that we use and the performance of our design

- 我們 Sphere decoding 使用的演算法是 8-best Breadth-First Search.
- ML-detection、Floating-point BFS、Fixed-point BFS 的 simulation 作圖結果如下:



2. Design concept and flow chart of our algorithm

- 程式碼一開始先固定random seed，然後產生 500 組 channel matrix，並且每一組 channel 再分別產生 40 組 size 為 4x1 的 transmitted signal. 我們simulation 的 SNR 模擬範圍是從 -4 ~ 25 間隔為 1。
- ML detection 、Floating-point BFS 、Fixed-point BFS simulation 會對每組 channel 算出一條 BER vs SNR 的curve，然後最後把再將 500組 channel 算出來的 curve 取平均得到最後的 BER vs SNR 的 curve。
- Floating-point BFS 以及 Fixed-point BFS simulation 我們都是使用 K=8 來進行模擬。
- Fixed-point 的 Quantization 我們使用 4 bits integer，6 bits fraction 來進行數值運算，我們在 fixed-point BFS simulation 的時候把 channel matrix、received signal、以及 constellation point 的 0.707 都先經過 quantization，軟體的 quantization 都是直接使用 floor 來模擬硬體計算的 truncation，也就是如果中間運算超過我們 fraction bits 所能表示的範圍就直接丟掉。

- 另外為了能夠模擬硬體在計算負數乘法的情況，我們在軟體使用 2's complement 的表示法來確保跟硬體的計算完全一樣，也就是我們把 constellation point 的 0.707 quantization 成 “0000101101”，把 -0.707 quantization 成 “1111010011”。
- 計算 R*constellation point 時，為了跟確保跟硬體實作完全一樣，我們先把乘法都變成絕對值相乘，然後經過 quantization 模擬 truncation 的情況，最後再依據一開始的被乘數的正負決定最後結果的正負號。
- 在 fixed-point BFS simulation 裡面我們使用下面的公式來近似 PED 計算，並且模擬在硬體實作時的 degradation 的大小。

$$X_i \approx |X_{i+1}| + |e_i|$$

$$|e_i| \approx |\Re\{e_i\}| + |\Im\{e_i\}|$$

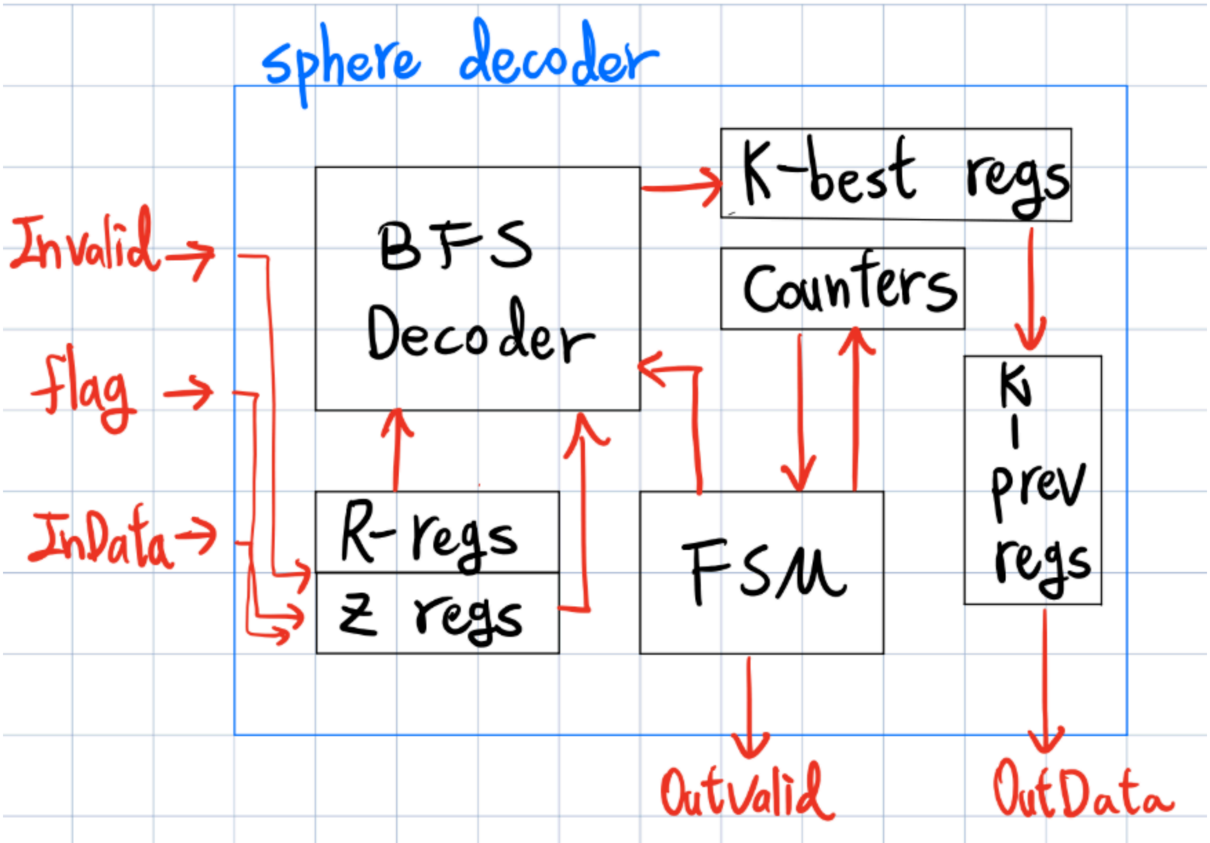
- 最後在 plot BER vs SNR 的 curve 的時候，我們使用線性內插來計算在 $BER = 10^{-4}$ 分別對應到 ML-detection 、fixed-point BFS 的 SNR 各是多少，再把兩個 SNR 值相減得到最後的 degradation。
- 為了生成 test pattern 測試我們的硬體實作是否正確，我們把軟經過 quantization 後的 fixed-point input z 、channel matrix 還有運算後 fixed-point 的 x-hat 以及對應的 distance 都當作pattern 輸入到我們的 testbench。
- 最後產生的 pattern 有 channel.dat、z.dat、golden.dat、distance.dat。

3. Design Interface

我們自定義了兩個額外的IO，分別是Input valid 和 output valid。另外在OutData的bits數，我把demapper放在testbench中，只輸出detected symbol，故只會輸出12 bits.

```
module sphere_decoder #(
    parameter WI = 10
)(
    input Clk,
    input Reset,
    input in_valid,
    input flagChannelorData,
    input [ (4*2*WI)-1 : 0] InData,
    output [11 : 0] OutData,
    output out_valid
);
```

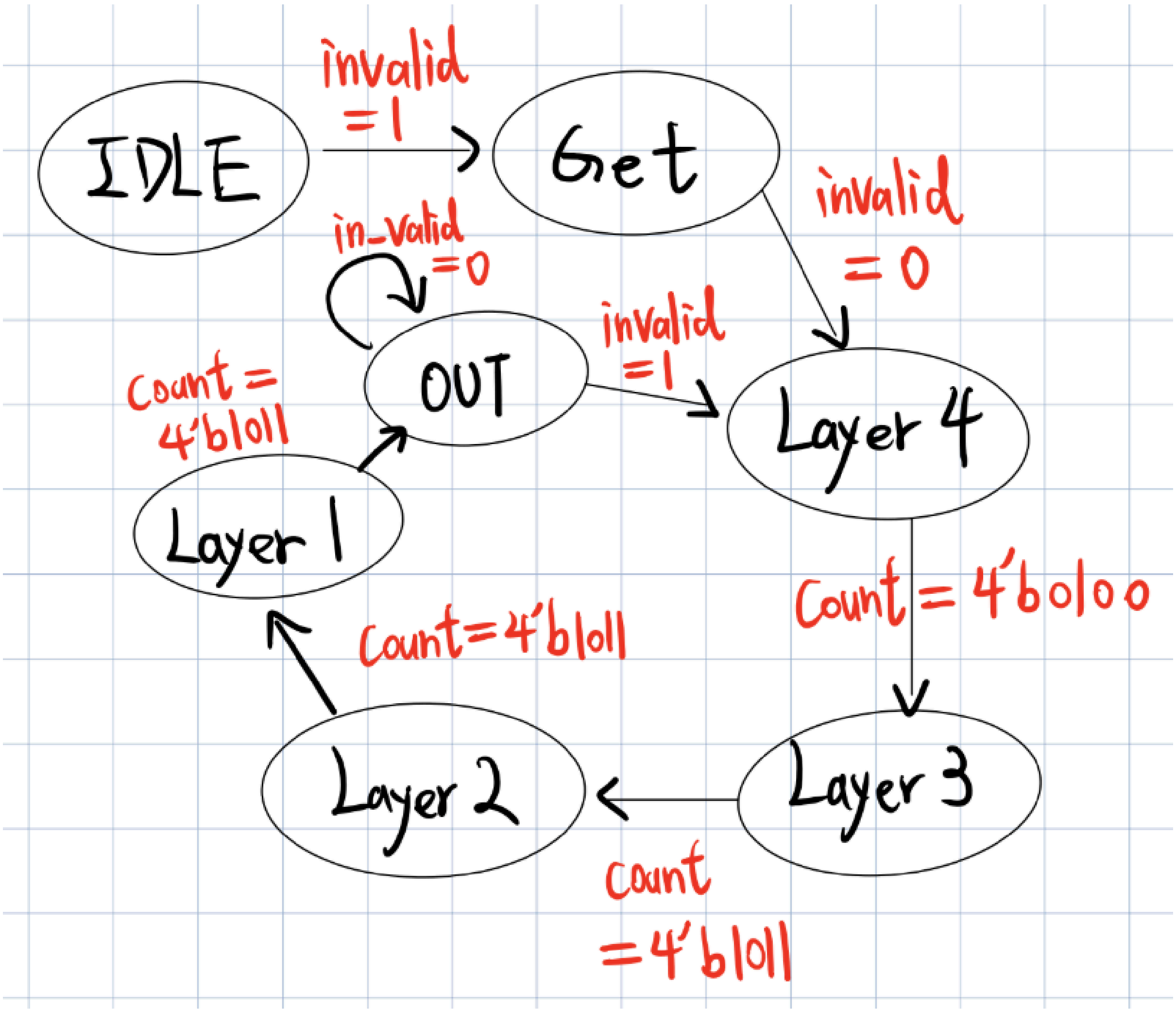
4. Design concept and block diagram



如上圖，我們採用的是BFS演算法在做解碼，每個區塊的功能主要分為::

BFS decoder 負責解碼，R-regs 儲存 channel 相關的資料，z-regs 儲存 received signal，K-best regs 儲存 layer 裡面最佳的8個點，K-prev regs 儲存上個 layer 裡面最佳的 8 個點，FSM 負責控制電路，counter 負責計算目前在第幾個運算點。

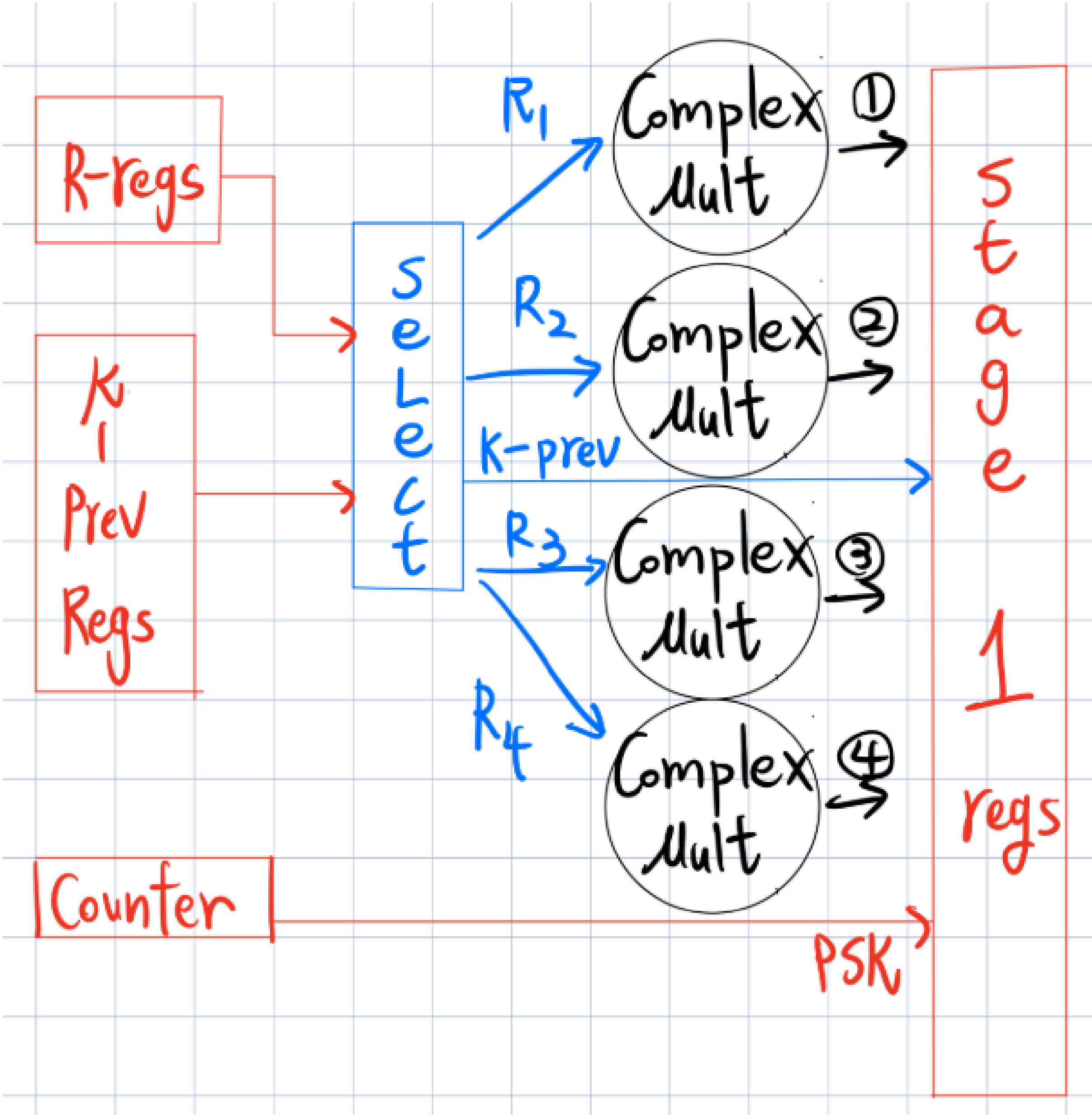
整體電路的FSM運作如下:



為了盡可能減少硬體資源，考慮到 BFS 得針對 64 個點去排序，我們決定採用 pipelined 的方式。每個 cycle 只計算一個點，並跟當下的 k-best regs 內容去比對，選擇捨棄或是插入到理想的位置。如此一來，便不需要使用排序，也可以順利找出8-best。

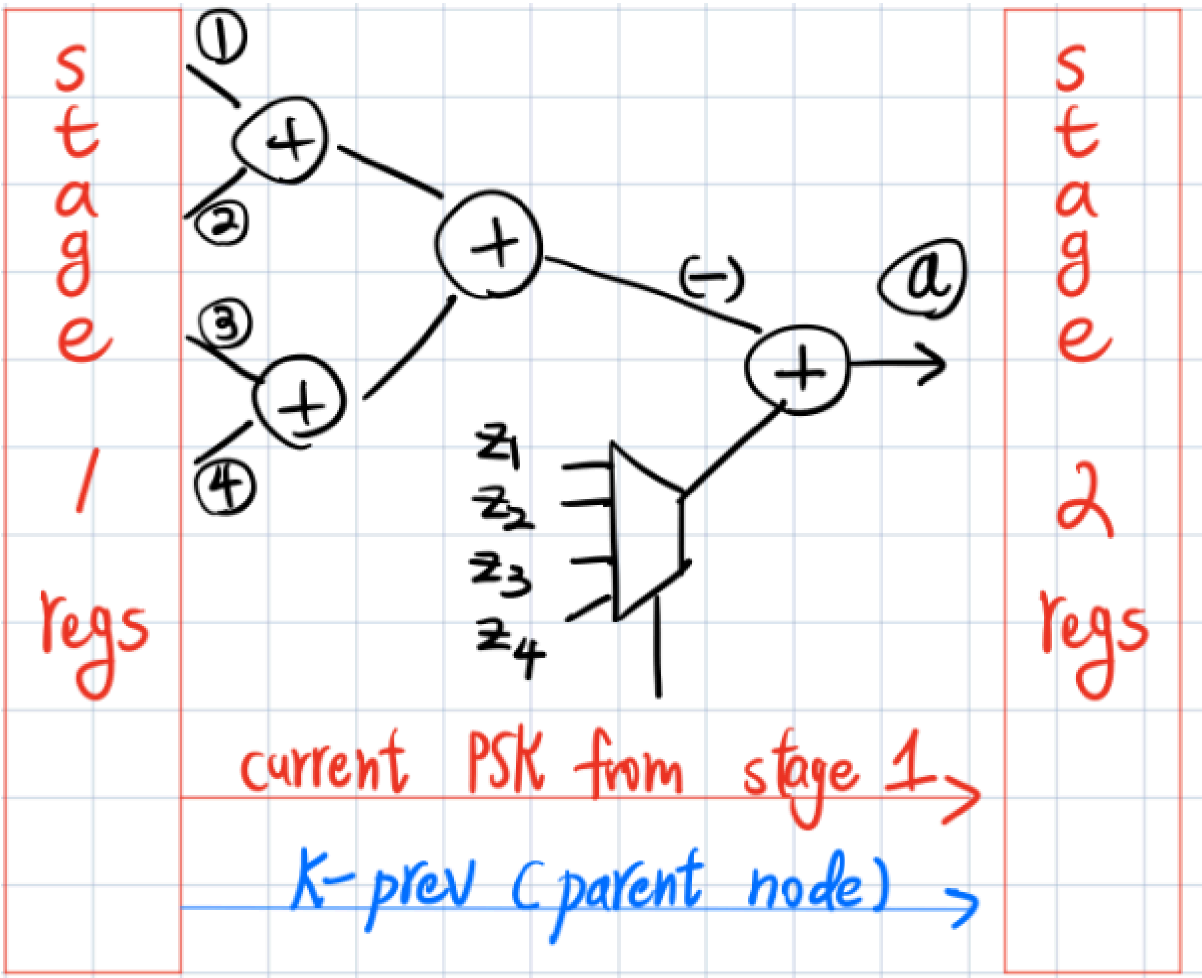
底下我將一一貼上 BFS decoder 內部不同 stage 的運算細節:

首先是 stage1，會交由 select 去選擇出目前所需要的 channel data 以及相關資料並交給 complex mult unit 去運算。此處可以注意的是，需要把當下的 counter (也就是第幾個PSK點) 和從 k-prev regs 取出的上層 layer value 都送入 pipeline，因為在後續的運算以及更新8-best 點，都需要他們的資料。



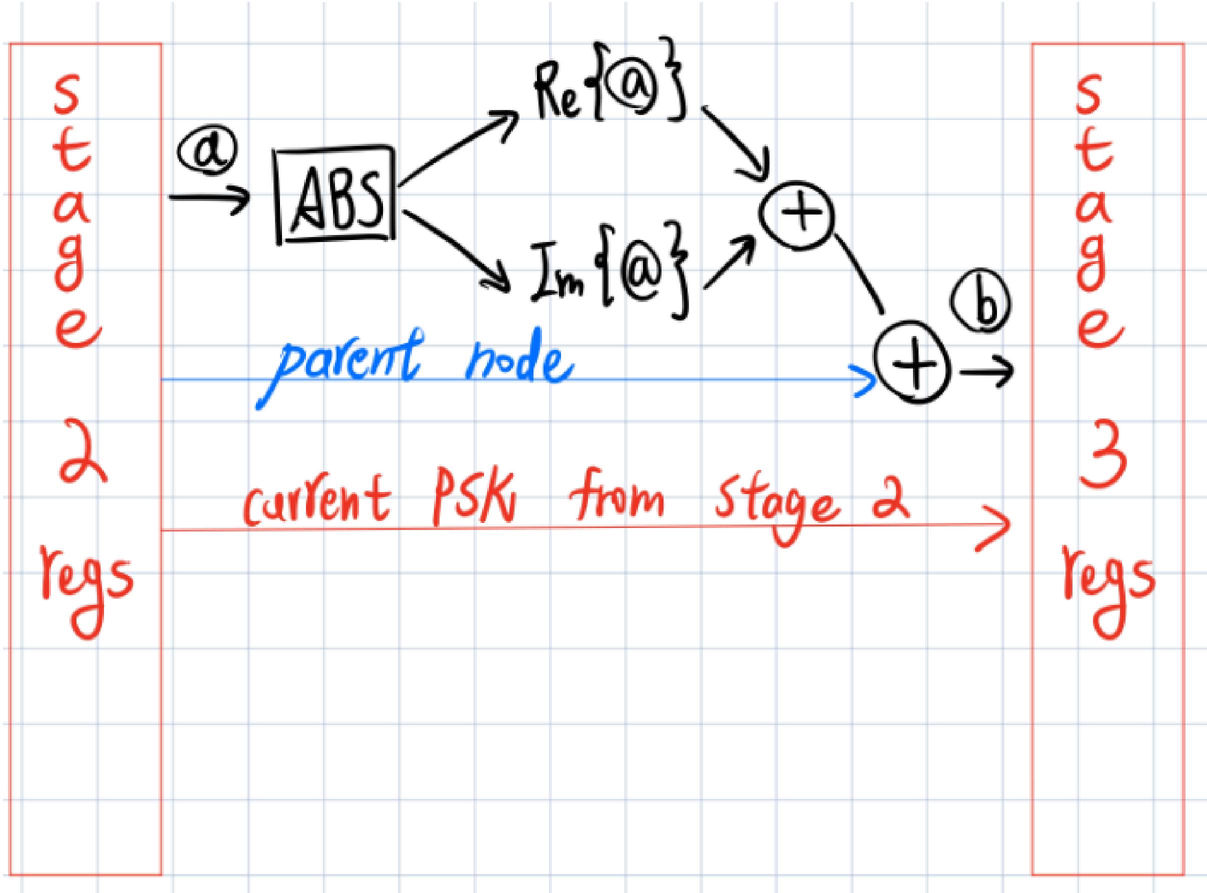
接著是 stage1 ~ stage2 :

將stage1算出的四個複數資料相加之後，再根據目前在哪層 layer 選出相對應的 component in received vector，和其相減，同時繼續把 PSK和parent node 資料傳入下個 stage。



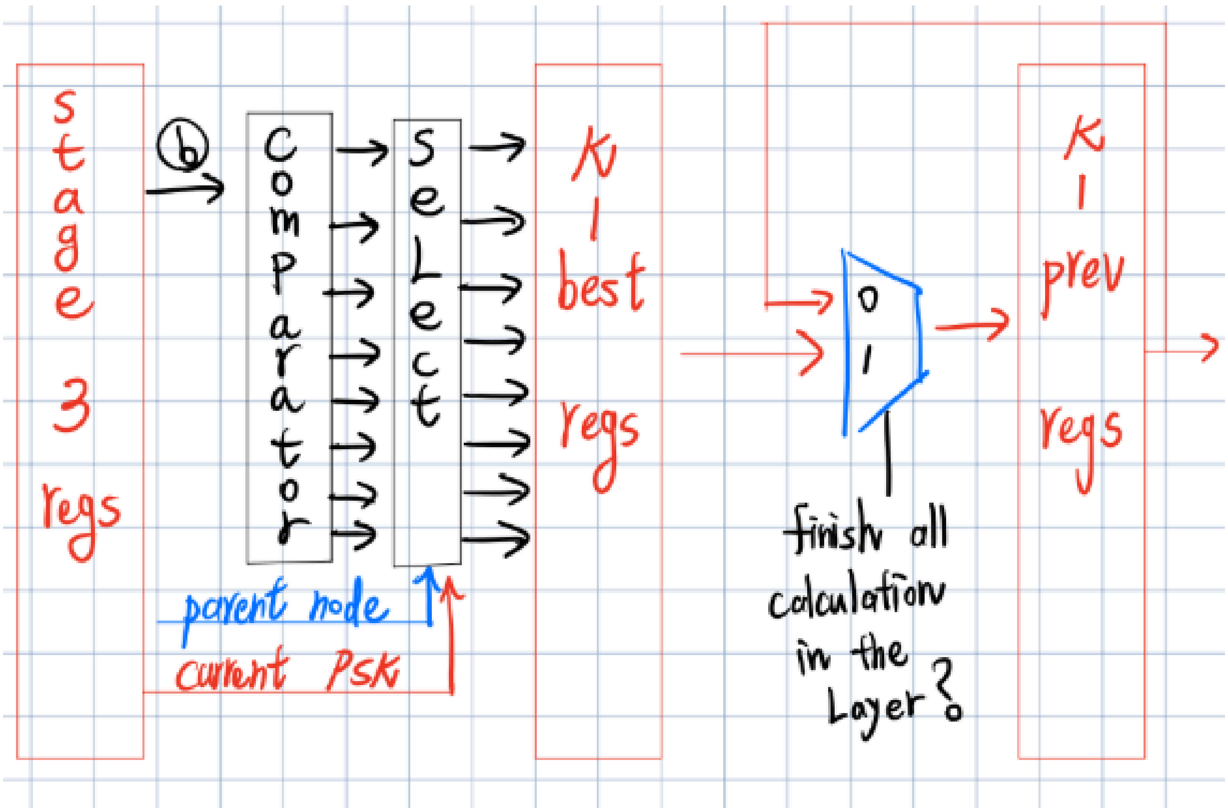
再來是stage2 ~ stage3 :

我們使用 “ A. Burg, M. Borgmann, M. Wenk, M. Zellweger, W. Fichtner and H. Bolcskei, "VLSI implementation of MIMO detection using the sphere decoding algorithm," 此篇論文內提及的 L2-norm 近似方式，將 stage2 計算出的值取出絕對值之後相加，使用 L1-norm 去逼近 L2-norm 的結果，同時繼續把 PSK 送進去下個 stage。

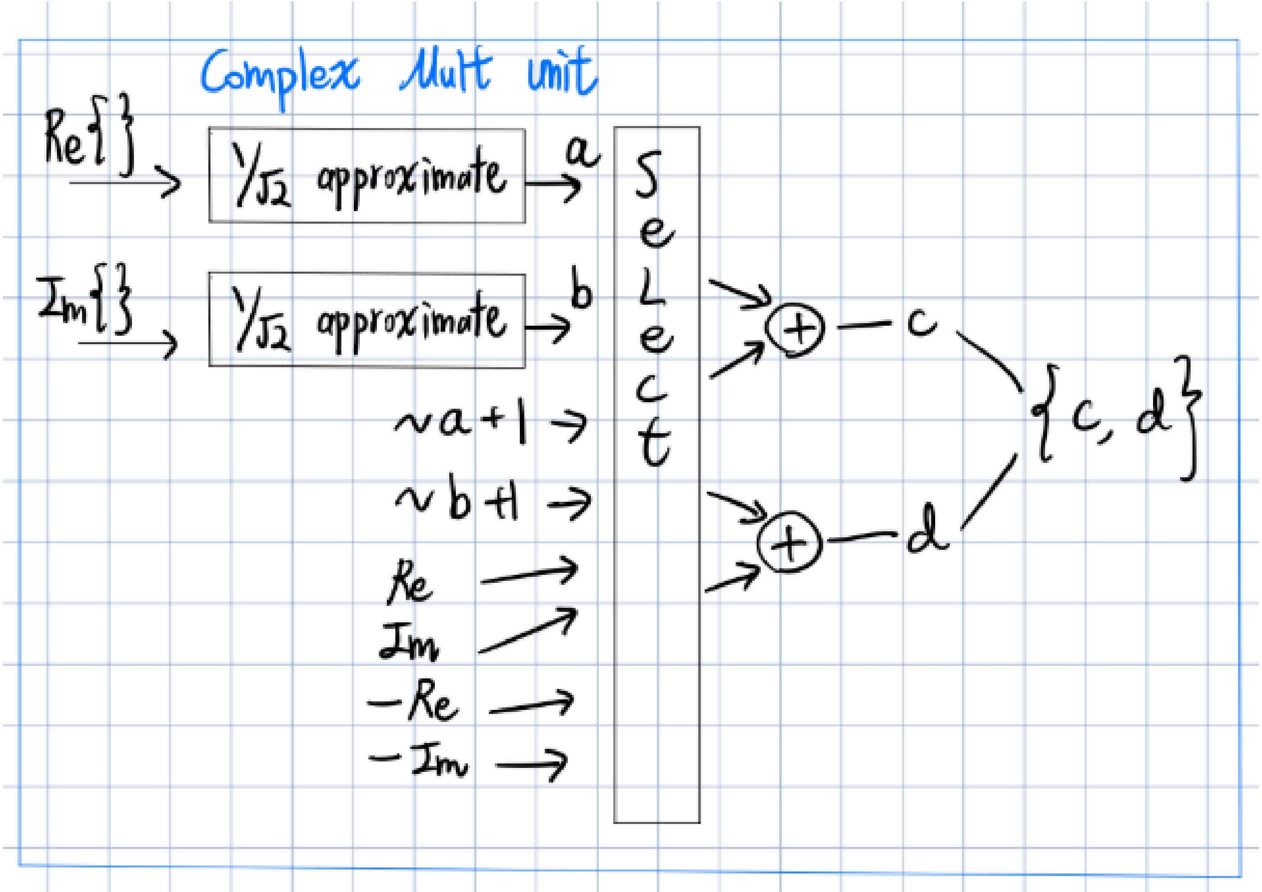


最後的 stage3 ~ k-best regs:

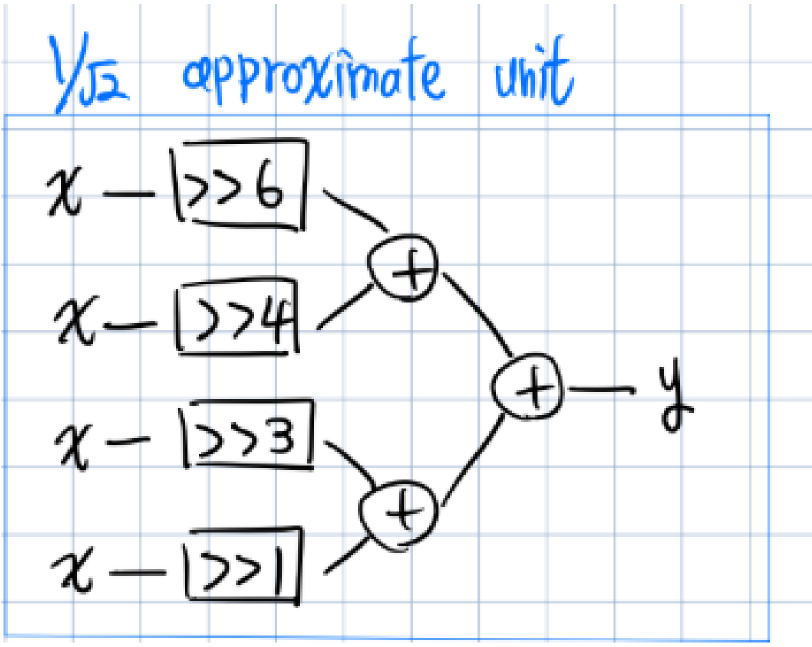
把計算出的 PED 送入比較器，同時和當下每個 k-best regs 內的 PED 去比較，得出該插入的 position 之後，交由 select 模組挪動 k-best regs 的位置。加入 parent node 的 index 資訊和當下 PSK 資訊，合併成一組資料，就可以存入k-best regs了。最後當 layer 運算完成時，要進到下個 layer 之前，得把完整的 k-best regs 備份到 k-prev regs 中，因為下層 layer 的運算會用到上一層的資料。



下圖為我們所使用的 complex multiplier unit，因為 8 PSK基本上只會有 -1,0, 1, 0.707, -0.707 因此只需要事先生成這些值，再根據PSK選出對應的即可。



下圖為我們近似0.707的所使用的方式，因為軟體模擬的結果為fractional 6 bits
所以我們採用 0.707 的二進位制取到小數點之後第六位，去做 shift and add逼近。



根據我們的 pipelined 架構，延伸出兩個問題:

1. pipelined 該切在哪邊？
2. layer 跟 layer 之間切換時，仍在 pipelined stage 裡面的資料該如何處理?

問題一:

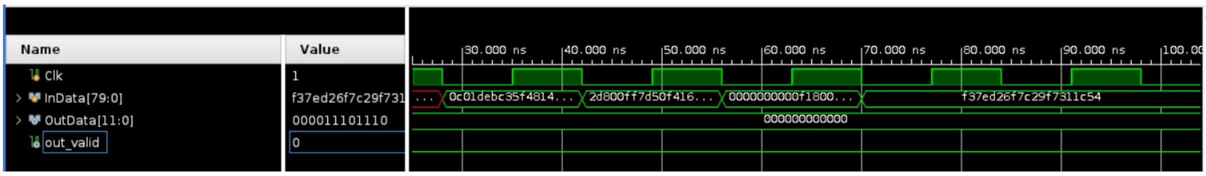
考慮到 timing 問題，我將 pipelined 切法平均的切在經過 2~3 個加法器的時間長度

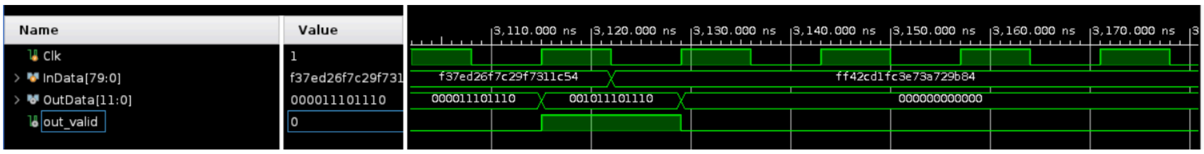
問題二:

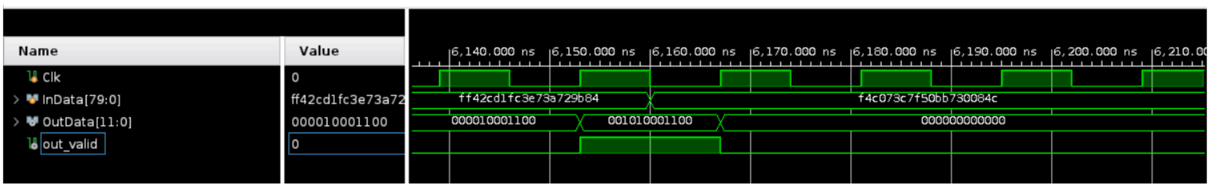
在 pipelined stage 之中，我使用 counter 計算需要花多少額外的 cycle 才能把剩餘資料都處理完，並在處理完之前都先 stall pipeline。雖然 stall pipeline 會降低 throughput，但卻是必要的。因為下個 layer 需要有上個 layer 的完整資料，如果並未等上個 layer算完，就直接送下個 layer 資料的話，在 stage1 會因為得存取 k-prev regs 的資料，而產生差錯，可能會錯誤的拿到不該在8 best內的點。

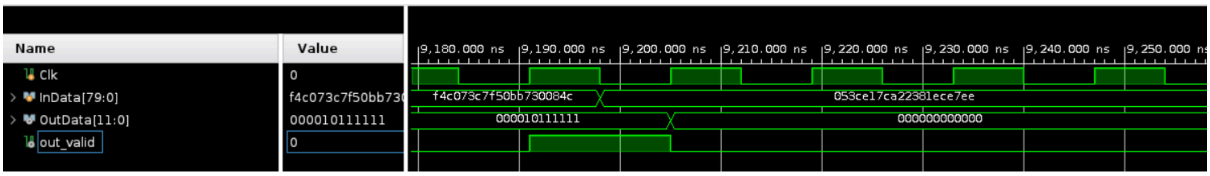
5. The timing of the test pattern:

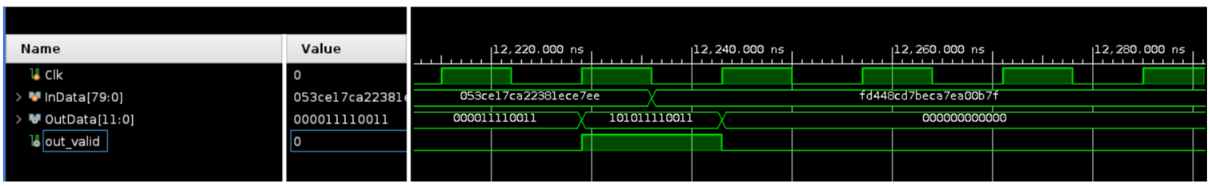
(a) Behavior simulation (SNR setting = 10 db)

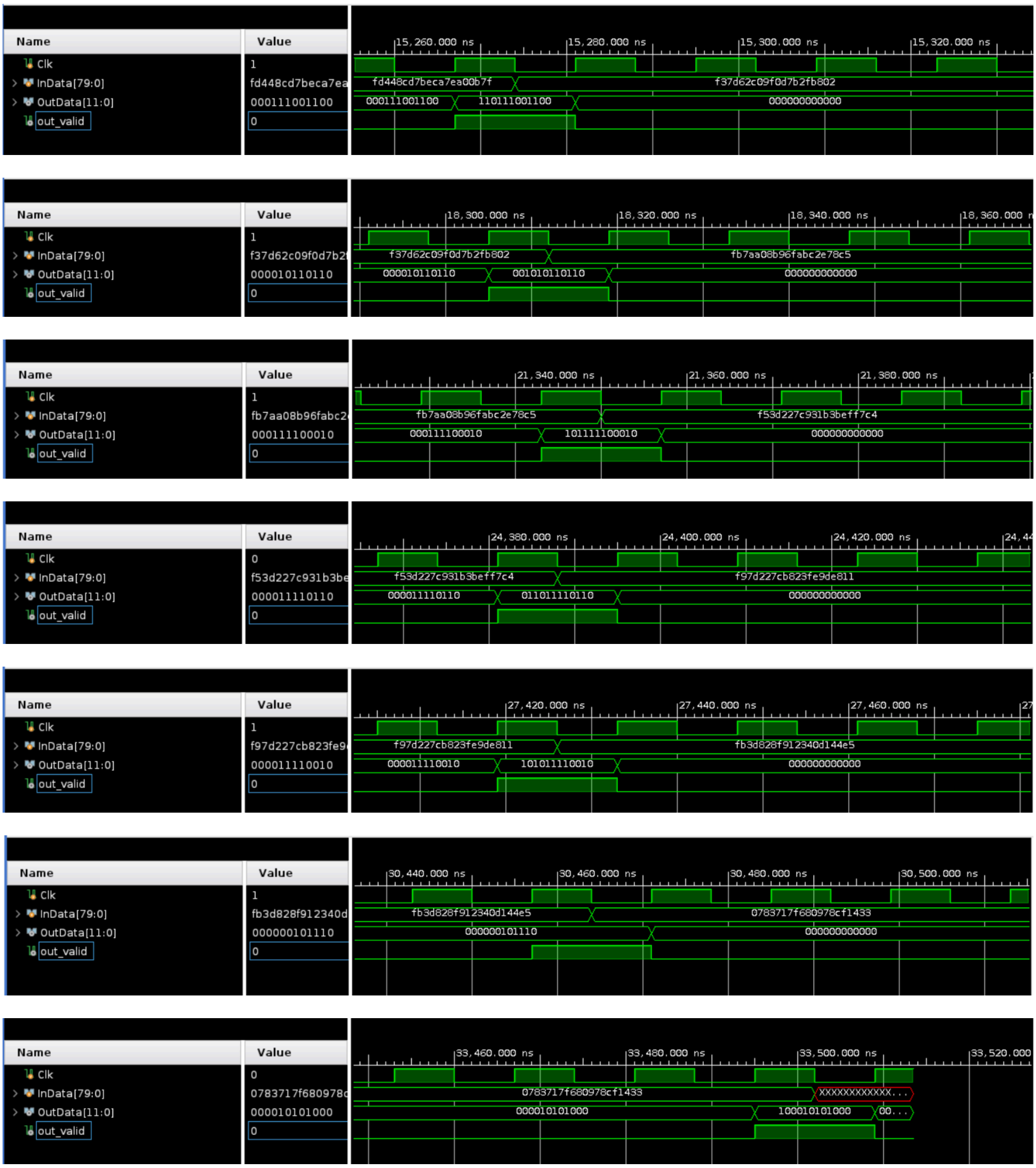




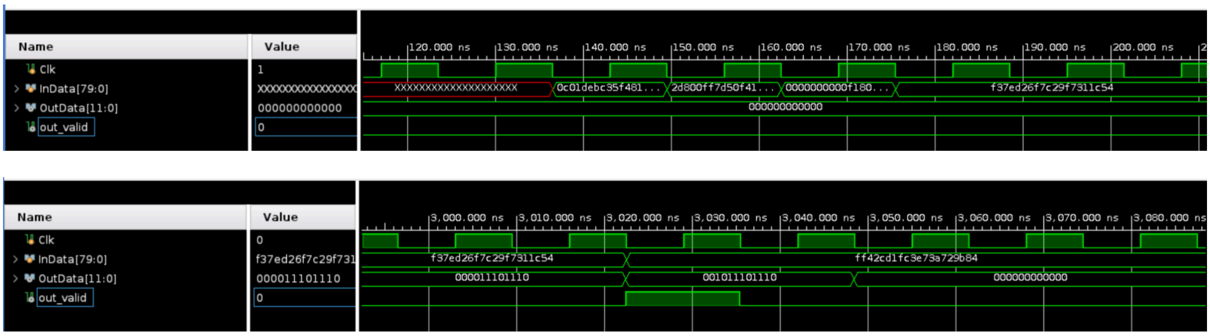


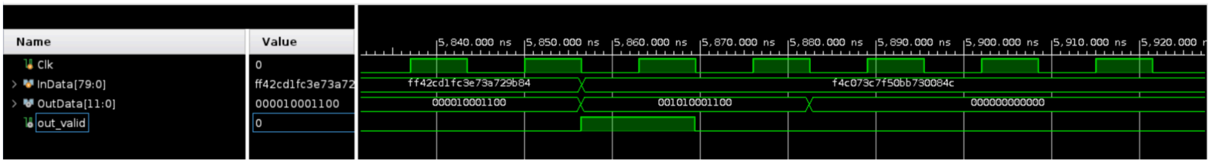




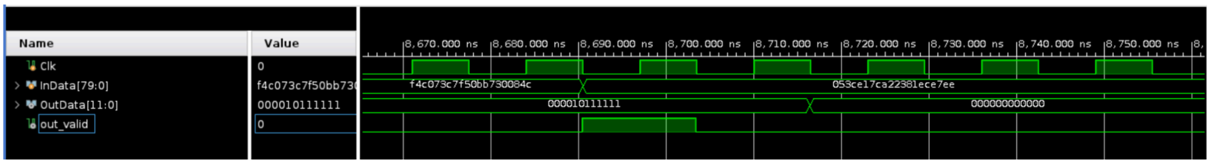


(b) post-synthesis simulation (clock period T_s = 13 ns)

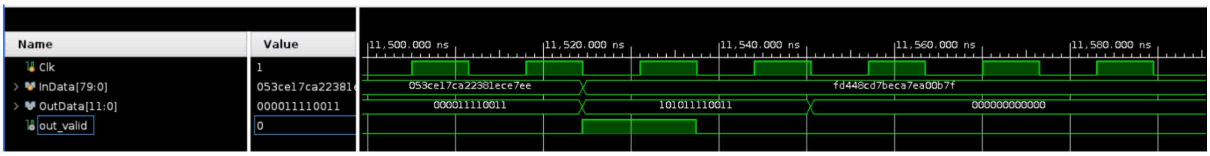




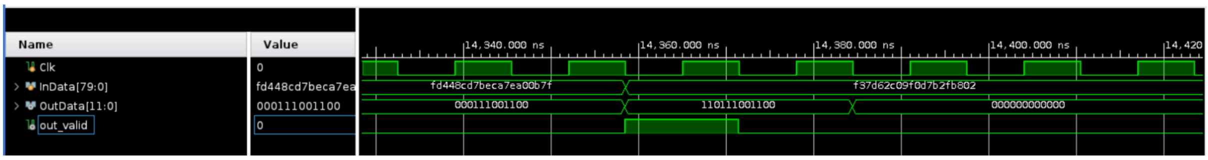
Name
Value
Clk
InData[79:0]
OutData[11:0]
out_valid
0
ff42cd1fc3e73a72
000010001100
0
5,840.000 ns
5,850.000 ns
5,860.000 ns
5,870.000 ns
5,880.000 ns
5,890.000 ns
5,900.000 ns
5,910.000 ns
5,920.000
ff42cd1fc3e73a729b84
f4c073c7f50bb730084c
000010001100
001010001100
000000000000



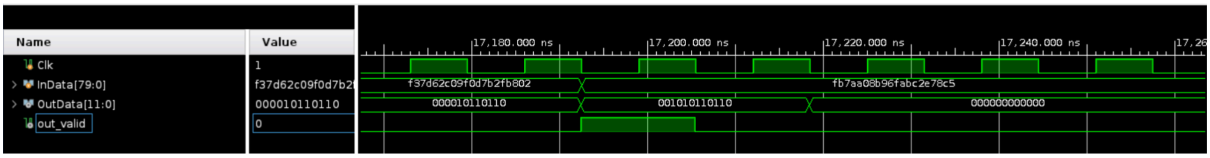
Name
Value
Clk
InData[79:0]
OutData[11:0]
out_valid
0
f4c073c7f50bb73
000010111111
0
8,670.000 ns
8,680.000 ns
8,690.000 ns
8,700.000 ns
8,710.000 ns
8,720.000 ns
8,730.000 ns
8,740.000 ns
8,750.000 ns
f4c073c7f50bb730084c
053ce17ca22381ece7ee
000010111111
000000000000



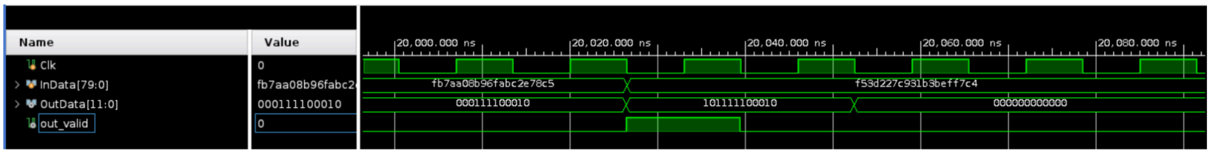
Name
Value
Clk
InData[79:0]
OutData[11:0]
out_valid
1
053ce17ca22381
000011110011
0
11,500.000 ns
11,520.000 ns
11,540.000 ns
11,560.000 ns
11,580.000 ns
053ce17ca22381ece7ee
fd448cd7beca7ea00b7f
000011110011
101011110011
000000000000



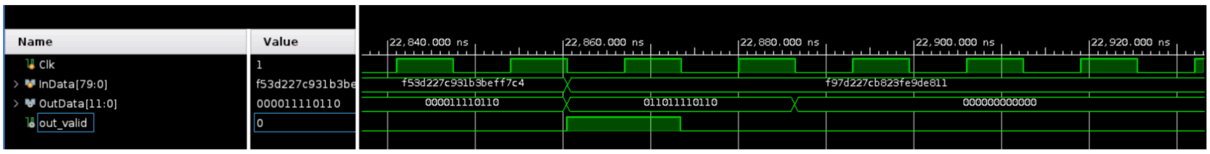
Name
Value
Clk
InData[79:0]
OutData[11:0]
out_valid
0
fd448cd7beca7ea
000111001100
0
14,340.000 ns
14,360.000 ns
14,380.000 ns
14,400.000 ns
14,420
fd448cd7beca7ea00b7f
f37d62c09f0d7b2fb802
000111001100
110111001100
000000000000



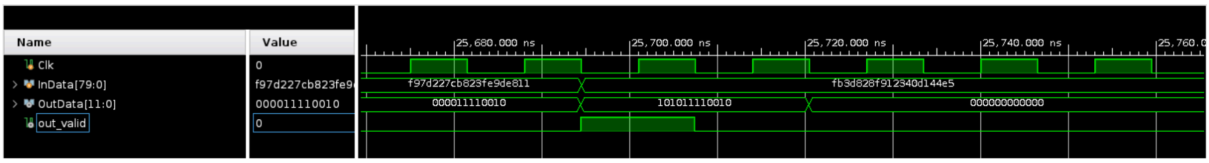
Name
Value
Clk
InData[79:0]
OutData[11:0]
out_valid
1
f37d62c09f0d7b2
000010110110
0
17,180.000 ns
17,200.000 ns
17,220.000 ns
17,240.000 ns
f37d62c09f0d7b2fb802
fb7aa08b96fabc2e78c5
000010110110
001010110110
000000000000



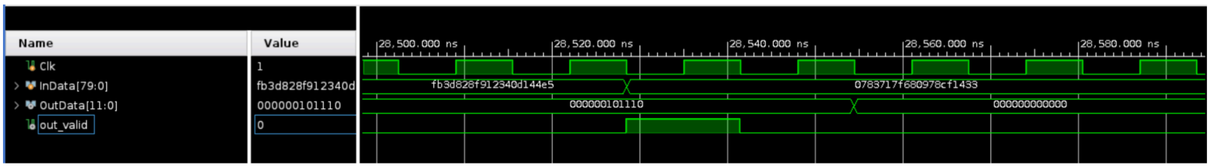
Name
Value
Clk
InData[79:0]
OutData[11:0]
out_valid
0
fb7aa08b96fabc2
000111100010
0
20,000.000 ns
20,020.000 ns
20,040.000 ns
20,060.000 ns
20,080.000 ns
fb7aa08b96fabc2e78c5
f53d227c931b3beff7c4
000111100010
101111100010
000000000000



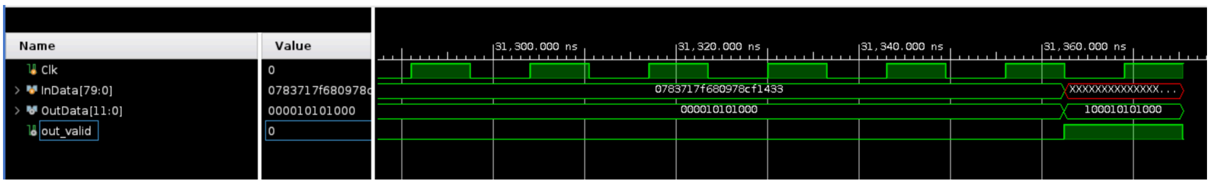
Name
Value
Clk
InData[79:0]
OutData[11:0]
out_valid
1
f53d227c931b3be
000011110110
0
22,840.000 ns
22,860.000 ns
22,880.000 ns
22,900.000 ns
22,920.000 ns
f53d227c931b3beff7c4
f97d227cb823fe9de811
000011110110
011011110110
000000000000



Name
Value
Clk
InData[79:0]
OutData[11:0]
out_valid
0
f97d227cb823fe9
000011110010
0
25,680.000 ns
25,700.000 ns
25,720.000 ns
25,740.000 ns
f97d227cb823fe9de811
fb3d828f912340d144e5
000011110010
101011110010
000000000000



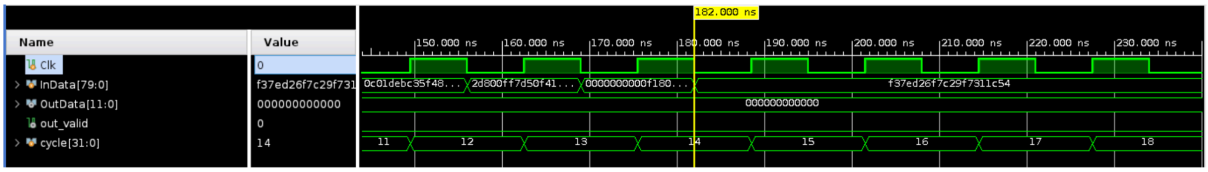
Name
Value
Clk
InData[79:0]
OutData[11:0]
out_valid
1
fb3d828f912340d
000000101110
0
28,500.000 ns
28,520.000 ns
28,540.000 ns
28,560.000 ns
28,580.000 ns
fb3d828f912340d144e5
0783717f680978cf1433
000000101110
000000000000



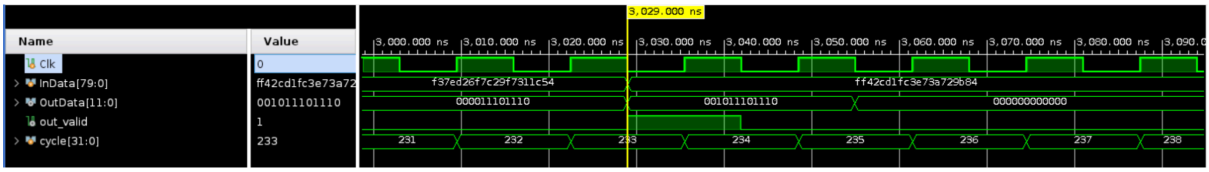
Name
Value
Clk
InData[79:0]
OutData[11:0]
out_valid
0
0783717f680978c
000010101000
0
31,300.000 ns
31,320.000 ns
31,340.000 ns
31,360.000 ns
0783717f680978cf1433
XXXXXXXXXXXXXX...
000010101000
100010101000

- M, L, M' calculation:

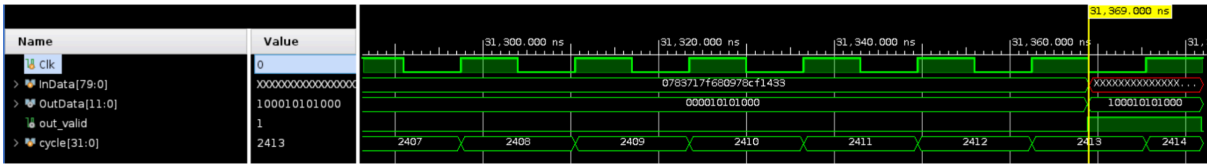
(1) The first sending vector y_1 is at cycle 14:



(2) The first detected output from y_1 is at cycle 233:



(3) The last detected output from y_{11} is at cycle 2413:



From above observation,

M' = 2413 - 14 = 2399 cycles. M = 2413 - 233 = 2180 cycles. L = 233 - 14 = 219 cycles.

(c) The processing time = 0.1 * 2180 * 13 (ns) = 2834 (ns).

6. Show that implementation is consistent with fixed-point simulation.

- Behavior simulation result:
- 我們和golden對照Euclidean distance的部分，只能在behavioral simulation執行，原因是合成完會沒有那個register，無法從合成完的list抓出來，但從我們behavioral simulation算出一模一樣的distance可知，synthesis後的結果應該也是完全一樣，畢竟在cost都很小而且很接近的情況之下，不算的一模一樣根本無法解碼出正確index
- 如果需要確認behavioral下的distance，請取消下面兩行的註解

```
$display("Your answer is correct: %b", OutData);
$display("Golden is: %b", golden[count]);
$display("Your answer is: %b", OutData);

// 拿掉下面兩行註解測試distance的運算結果，但只能在behavior simulation執行。
// $display("Here is the Euclidean distance from the golden detected symbol: %b", golden_distance[count]);
// $display("Here is your Euclidean distance: %b", dut.k_prev_r[0][0 +: WI]);
```

```
Your answer is correct: 001011101110
Golden is: 001011101110
Your answer is: 001011101110
Here is the Euclidean distance from the golden detected symbol: 0001101101
Here is your Euclidean distance: 0001101101
Here is your detected symbol:
The detected symbol of x1 is: -1 + j0
The detected symbol of x2 is: -0.707 + j0.707
The detected symbol of x3 is: 0.707 - j0.707
The detected symbol of x4 is: 0.707 + j0.707


Your answer is correct: 001010001100
Golden is: 001010001100
Your answer is: 001010001100
Here is the Euclidean distance from the golden detected symbol: 0001111000
Here is your Euclidean distance: 0001111000
Here is your detected symbol:
The detected symbol of x1 is: -1 + j0
The detected symbol of x2 is: 0 + j1
The detected symbol of x3 is: -1 + j0
The detected symbol of x4 is: 0 - j1


Your answer is correct: 000010111111
Golden is: 000010111111
Your answer is: 000010111111
Here is the Euclidean distance from the golden detected symbol: 0001000110
Here is your Euclidean distance: 0001000110
Here is your detected symbol:
The detected symbol of x1 is: -0.707 -j0.707
The detected symbol of x2 is: 0 + j1
The detected symbol of x3 is: 1 + j0
The detected symbol of x4 is: 1 + j0
```

```
Your answer is correct: 101011110011
Golden is: 101011110011
Your answer is: 101011110011
Here is the Euclidean distance from the golden detected symbol: 0001110100
Here is your Euclidean distance: 0001110100
Here is your detected symbol:
The detected symbol of x1 is: 0.707 - j0.707
The detected symbol of x2 is: -0.707 + j0.707
The detected symbol of x3 is: 0.707 + j0.707
The detected symbol of x4 is: -0.707 + j0.707


Your answer is correct: 110111001100
Golden is: 110111001100
Your answer is: 110111001100
Here is the Euclidean distance from the golden detected symbol: 0000111111
Here is your Euclidean distance: 0000111111
Here is your detected symbol:
The detected symbol of x1 is: 0.707 + j0.707
The detected symbol of x2 is: 1 + j0
The detected symbol of x3 is: -1 + j0
The detected symbol of x4 is: 0 - j1


Your answer is correct: 101111100010
Golden is: 101111100010
Your answer is: 101111100010
Here is the Euclidean distance from the golden detected symbol: 0010001111
Here is your Euclidean distance: 0010001111
Here is your detected symbol:
The detected symbol of x1 is: 0.707 - j0.707
The detected symbol of x2 is: 1 + j0
The detected symbol of x3 is: 0 - j1
The detected symbol of x4 is: 0 + j1
```

```
Your answer is correct: 011011110110
Golden is: 011011110110
Your answer is: 011011110110
Here is the Euclidean distance from the golden detected symbol: 0001010001
Here is your Euclidean distance: 0001010001
Here is your detected symbol:
The detected symbol of x1 is: -0.707 + j0.707
The detected symbol of x2 is: -0.707 + j0.707
The detected symbol of x3 is: 0.707 + j0.707
The detected symbol of x4 is: 0.707 + j0.707


Your answer is correct: 101011110010
Golden is: 101011110010
Your answer is: 101011110010
Here is the Euclidean distance from the golden detected symbol: 0001010101
Here is your Euclidean distance: 0001010101
Here is your detected symbol:
The detected symbol of x1 is: 0.707 - j0.707
The detected symbol of x2 is: -0.707 + j0.707
The detected symbol of x3 is: 0.707 + j0.707
The detected symbol of x4 is: 0 + j1


Your answer is correct: 000000101110
Golden is: 000000101110
Your answer is: 000000101110
Here is the Euclidean distance from the golden detected symbol: 0001011100
Here is your Euclidean distance: 0001011100
Here is your detected symbol:
The detected symbol of x1 is: -0.707 -j0.707
The detected symbol of x2 is: -0.707 -j0.707
The detected symbol of x3 is: 0.707 - j0.707
The detected symbol of x4 is: 0.707 + j0.707
```

```
Your answer is correct: 100010101000
Golden is: 100010101000
Your answer is: 100010101000
Here is the Euclidean distance from the golden detected symbol: 0001010000
Here is your Euclidean distance: 0001010000
Here is your detected symbol:
The detected symbol of x1 is: 0 - j1
The detected symbol of x2 is: 0 + j1
The detected symbol of x3 is: 0.707 - j0.707
The detected symbol of x4 is: -0.707 -j0.707


Your answer is correct: 001010110110
Golden is: 001010110110
Your answer is: 001010110110
Here is the Euclidean distance from the golden detected symbol: 0001000100
Here is your Euclidean distance: 0001000100
Here is your detected symbol:
The detected symbol of x1 is: -1 + j0
The detected symbol of x2 is: 0 + j1
The detected symbol of x3 is: 0.707 + j0.707
The detected symbol of x4 is: 0.707 + j0.707


All test patterns are correct, congratulations
```

7. Synthesis report

```
Copyright 1986-2020 Xilinx, Inc. All Rights Reserved.
-------------------------------------------------------------------------------------------------------------------
| Tool Version : Vivado v.2020.2 (lin64) Build 3064766 Wed Nov 18 09:12:47 MST 2020
| Date         : Tue Dec 24 22:37:30 2024
| Host         : cad33 running 64-bit CentOS Linux release 7.9.2009 (Core)
| Command      : report utilization -file sphere decoder utilization synth.rpt -pb sphere decoder utilization synth.pb
| Design       : sphere decoder
| Device       : 7a200tfbg676-1
| Design State : Synthesized
-------------------------------------------------------------------------------------------------------------------

Utilization Design Information

Table of Contents
-----------------
1. Slice Logic
1.1 Summary of Registers by Type
2. Memory
3. DSP
4. IO and GT Specific
5. Clocking
6. Specific Feature
7. Primitives
8. Black Boxes
9. Instantiated Netlists

1. Slice Logic
--------------

+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             | 1528 |     0 |    134600 |  1.14 |
|   LUT as Logic          | 1528 |     0 |    134600 |  1.14 |
|   LUT as Memory         |    0 |     0 |     46200 |  0.00 |
| Slice Registers         |  867 |     0 |    269200 |  0.32 |
|   Register as Flip Flop |  867 |     0 |    269200 |  0.32 |
|   Register as Latch     |    0 |     0 |    269200 |  0.00 |
| F7 Muxes                |   19 |     0 |     67300 |  0.03 |
| F8 Muxes                |    0 |     0 |     33650 |  0.00 |
+-------------------------+------+-------+-----------+-------+
* Warning! The Final LUT count, after physical optimizations and full implementation, is typically lower. Run opt design after synthesis, if not already completed, for a more realistic count.


1.1 Summary of Registers by Type
--------------------------------

+-------+--------------+-------------+--------------+
| Total | Clock Enable | Synchronous | Asynchronous |
+-------+--------------+-------------+--------------+
| 0     |            _ |           - |            - |
| 0     |            _ |           - |          Set |
| 0     |            _ |           - |        Reset |
| 0     |            _ |         Set |            - |
| 0     |            _ |       Reset |            - |
| 0     |          Yes |           - |            - |
| 1     |          Yes |           - |          Set |
| 866   |          Yes |           - |        Reset |
| 0     |          Yes |         Set |            - |
| 0     |          Yes |       Reset |            - |
+-------+--------------+-------------+--------------+
```

```
2. Memory
---------

+----------------+------+-------+-----------+-------+
|    Site Type   | Used | Fixed | Available | Util% |
+----------------+------+-------+-----------+-------+
| Block RAM Tile |    0 |     0 |       365 |  0.00 |
|   RAMB36/FIFO* |    0 |     0 |       365 |  0.00 |
|   RAMB18       |    0 |     0 |       730 |  0.00 |
+----------------+------+-------+-----------+-------+
* Note: Each Block RAM Tile only has one FIFO logic available and therefore can accommodate only one FIFO36E1 or one FIFO18E1. However, if a FIFO18E1 occupies a Block RAM Tile, that tile can still accommodate a RAMB18E1


3. DSP
------

+-----------+------+-------+-----------+-------+
| Site Type | Used | Fixed | Available | Util% |
+-----------+------+-------+-----------+-------+
| DSPs      |    0 |     0 |       740 |  0.00 |
+-----------+------+-------+-----------+-------+


4. IO and GT Specific
---------------------

+-----------------------------+------+-------+-----------+-------+
|          Site Type          | Used | Fixed | Available | Util% |
+-----------------------------+------+-------+-----------+-------+
| Bonded IOB                  |   97 |     0 |       400 | 24.25 |
| Bonded IPADs                |    0 |     0 |        26 |  0.00 |
| Bonded OPADs                |    0 |     0 |        16 |  0.00 |
| PHY CONTROL                 |    0 |     0 |        10 |  0.00 |
| PHASER REF                  |    0 |     0 |        10 |  0.00 |
| OUT FIFO                    |    0 |     0 |        40 |  0.00 |
| IN FIFO                     |    0 |     0 |        40 |  0.00 |
| IDELAYCTRL                  |    0 |     0 |        10 |  0.00 |
| IBUFDS                      |    0 |     0 |       384 |  0.00 |
| GTPE2 CHANNEL               |    0 |     0 |         8 |  0.00 |
| PHASER OUT/PHASER OUT PHY   |    0 |     0 |        40 |  0.00 |
| PHASER IN/PHASER IN PHY     |    0 |     0 |        40 |  0.00 |
| IDELAYE2/IDELAYE2 FINEDELAY |    0 |     0 |       500 |  0.00 |
| IBUFDS GTE2                 |    0 |     0 |         4 |  0.00 |
| ILOGIC                      |    0 |     0 |       400 |  0.00 |
| OLOGIC                      |    0 |     0 |       400 |  0.00 |
+-----------------------------+------+-------+-----------+-------+


5. Clocking
-----------

+------------+------+-------+-----------+-------+
|  Site Type | Used | Fixed | Available | Util% |
+------------+------+-------+-----------+-------+
| BUFGCTRL   |    1 |     0 |        32 |  3.13 |
| BUFIO      |    0 |     0 |        40 |  0.00 |
| MMCME2 ADV |    0 |     0 |        10 |  0.00 |
| PLLE2 ADV  |    0 |     0 |        10 |  0.00 |
| BUFMRCE    |    0 |     0 |        20 |  0.00 |
| BUFHCE     |    0 |     0 |       120 |  0.00 |
| BUFR       |    0 |     0 |        40 |  0.00 |
+------------+------+-------+-----------+-------+
```

LUTs = 1528, DSPs = 0, FFs = 867

NA = 1528 + 867 = 2395

8. Calculate AT product:

2395 * 2834 = 6787430

9. List the working items and weightings:

B10502076 金家逸	Matlab software algorithm simulation Vivado hardware simulation debugging
R13943124 施伯儒	Verilog hardware system design Vivado hardware simulation debugging