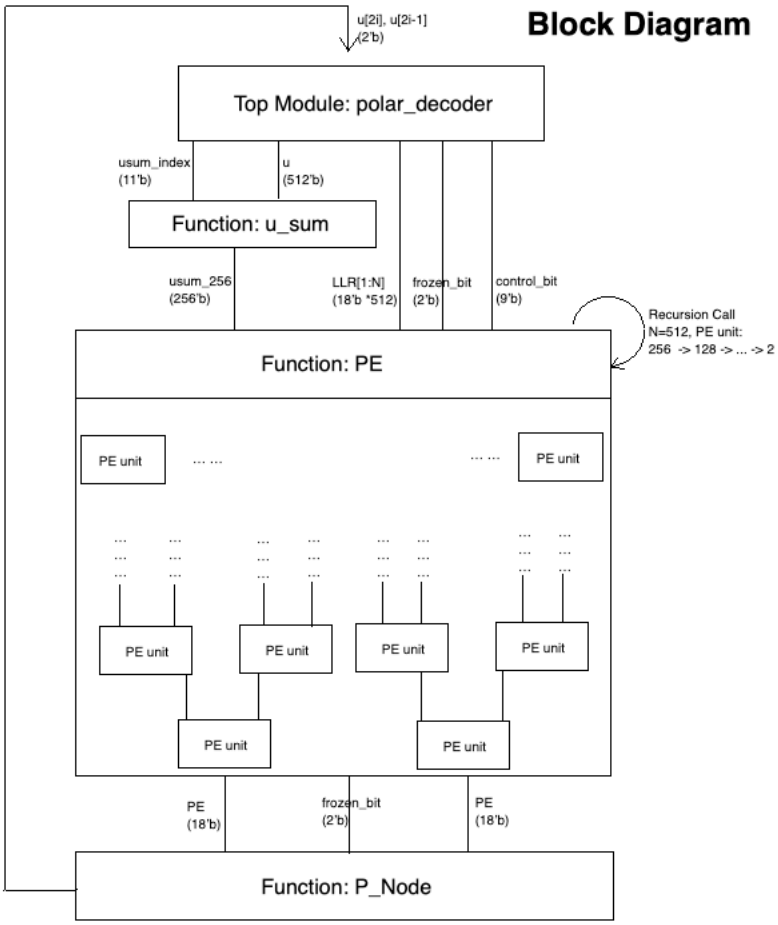
Computer-Aided VLSI System Design
Final Project Report

Team IDs: 20

Student Names: 李杰銘 梁峻瑋

Questions and Discussion

1. 硬體架構圖 Block Diagram – 功能性基礎版



答：在”polar_decoder.v”部分，我參考了[1]這篇論文中提出的設計。具體來說，我使用了三個函數來完成 SC decoder 功能——"u_sum()”, “pnode()”, “PE()”。

首先，我們把兩兩成對的 f(),g()組合成一個 PE()，以及把最後一層和 h()組合成 pnode()。其次，我們額外設計一個 function u_sum()，從 counter 和 u 來模擬 u[i] bit 之間做 xor 的規律，得到 usum 值。最後，我們把 usum 值和 LLR , control bit, frozen bit 一起丟給 pnode 函數，並且先交由多層 PE()迭代運算，再由最後一層 pnode 輸出。這樣就得到一組{u[2i], u[2i-1]}。

經過 N/2 次依序的運算，以及在運算同時取出 non-frozen bit 的值，就能夠在運算完同時輸出到 DEC memory。並且完成 SC decoder 功能。可以參考下方左圖的 N=8 SC dcoder 和下方右圖的[1]論文提出設計。

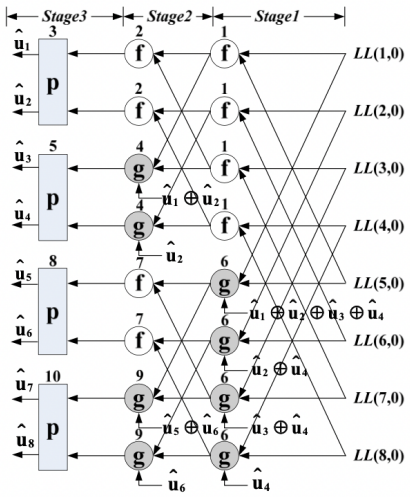


Fig. 2.5. The decoding procedure of 2b-SC algorithm with n=8.

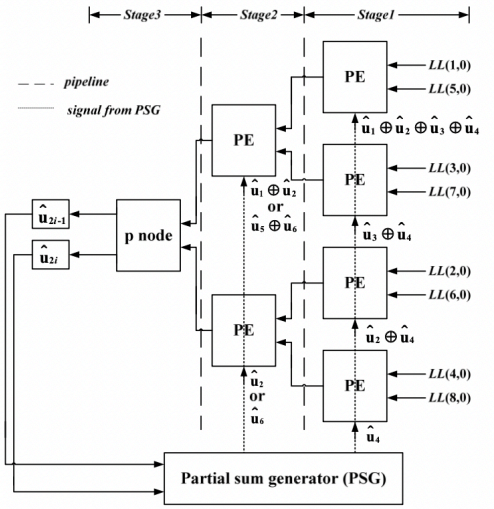


Fig. 2.8. The tree-based 2b-SC architecture with n=8.

2. 優化技巧 Optimization Technique

答：我們先講” 功能性基礎版”的實作方式，再談後面兩個優化的動機和手法。

#2-1 實作方式一一"功能性基礎版”

” 功能性基礎版”的主要實作方法，就是用 look-up table 實作 function()。首先 u_sum()分成 usum_2, usum_4, usum_8, …, usum_256。例如，上方右圖的中間那排兩個 PE 的 input 就是 usum_2 和前一層 PE 結果，而前一層四個 PE 的 input 就是 usum_4 和 LLR。規律就是用(u[2*i-2], …,u[2*i-2-N+1])和 kronecker matrix 做矩陣乘法。此外，N=128 的{u[2i], u[2i-1]}，也可以用 6 層 PE()展開作為 pnode 的 input，來得到結果。

這樣做的動機，第一是考量到遞迴成本。以 gcc/g++的編譯器而言，如果我們實作 QuickSort，由於遞迴的 compiler framework 要傳遞變數, 地址等等開銷，因此實務上需要 O(logn)的 stack space [2]。依此類推，我們認為在 verilog 的編譯器中，使用 module 做遞迴，也會造成一定複雜度的額外開銷。第二，我們認為使用 bit operation & logic operation level 做運算，可以讓 IC Compiler 做到更優質的簡化。這個道理，就好比 C/C++的實作效能通常是 python 的數十倍，而組合語言的實作效能又比 C/C++更好。畢竟 compiler 先天上就是考慮到各種可能來優化，不同的表達方式和表達語言，會造成迥然不同的優化結果。

而能辦到這樣實作的原因，一方面是參考[1]的 PE()結構，讓我們只需要設計一個結構，就能夠重複使用；二方面是因為用 function 實作，不需要受限於 module 的硬性規定，能夠手動展開來呼叫函數。

```
function [63:0] usum_64;
input [10:0] i;
input [512:0] u;
usum_64 =
{u[2*i-2],u[2*i-2]^u[2*i-34],u[2*i-2]^u[2*i-18],u[2*i-2]^u[2*i-18]^u[2*i-34]^u[2*i-50],u[2*i-2]^u[2*i-10],u[2*i-2]^u[2*i-10]^u[2*i-34]^u[2*i-42],u[2*i-2]^u[2*i-10]^u[2*i-18]^u[2*i-26],u[2*i-2]^u[2*i-10]^u[2*i-18]^u[2*i-26]^u[2*i-34]^u[2*i-42]^u[2*i-50]^u[2*i-58],u[2*i-2]^u[2*i-6],u[2*i-2]^u[2*i-6]^u[2*i-34]^u[2*i-38],u[2*i-2]^u[2*i-6]^u[2*i-18]^u[2*i-22],u[2*i-2]^u[2*i-6]^u[2*i-18]^u[2*i-22]^u[2*i-34]^u[2*i-38]^u[2*i-50]^u[2*i-54],u[2*i-2]^u[2*i-6]^u[2*i-10]^u[2*i-14],u[2*i-2]^u[2*i-6]^u[2*i-10]^u[2*i-14]^u[2*i-34]^u[2*i-38]^u[2*i-42]^u[2*i-46],u[2*i-2]^u[2*i-6]^u[2*i-10]^u[2*i-14]^u[2*i-18]^u[2*i-22]^u[2*i-26]^u[2*i-30],u[2*i-2]^u[2*i-6]^u[2*i-10]^u[2*i-14]^u[2*i-18]^u[2*i-22]^u[2*i-26]^u[2*i-30]^u[2*i-34]^u[2*i-38]^u[2*i-42]^u[2*i-46]^u[2*i-50]^u[2*i-54]^u[2*i-58]^u[2*i-62],u[2*i-2]^u[2*i-4],u[2*i-2]^u[2*i-4]^u[2*i-34]^u[2*i-36],u[2*i-2]^u[2*i-4]^u[2*i-18]^u[2*i-20],u[2*i-2]^u[2*i-4]^u[2*i-18]^u[2*i-20]^u[2*i-34]^u[2*i-36]^u[2*i-50]^u[2*i-52],u[2*i-2]^u[2*i-4]^u[2*i-10]^u[2*i-12],u[2*i-2]^u[2*i-4]^u[2*i-10]^u[2*i-12]^u[2*i-34]^u[2*i-36]^u[2*i-42]^u[2*i-44],u[2*i-2]^u[2*i-4]^u[2*i-10]^u[2*i-12]^u[2*i-18]^u[2*i-20]^u[2*i-26]^u[2*i-28],u[2*i-2]^u[2*i-4]^u[2*i-10]^u[2*i-12]^u[2*i-18]^u[2*i-20]^u[2*i-26]^u[2*i-28]^u[2*i-34]^u[2*i-36]^u[2*i-42]^u[2*i-44]^u[2*i-50]^u[2*i-52]^u[2*i-58]^u[2*i-60],u[2*i-2]^u[2*i-4]^u[2*i-6]^u[2*i-8],u[2*i-2]^u[2*i-4]^u[2*i-6]^u[2*i-8]^u[2*i-34]^u[2*i-36]^u[2*i-38]^u[2*i-40],u[2*i-2]^u[2*i-4]^u[2*i-6]^u[2*i-8]^u[2*i-18]^u[2*i-20]^u[2*i-22]^u[2*i-24],u[2*i-2]^u[2*i-4]^u[2*i-6]^u[2*i-8]^u[2*i-18]^u[2*i-20]^u[2*i-22]^u[2*i-24]^u[2*i-34]^u[2*i-36]^u[2*i-38]^u[2*i-40]^u[2*i-42]^u[2*i-44]^u[2*i-46],u[2*i-2]^u[2*i-4]^u[2*i-6]^u[2*i-8]^u[2*i-10]^u[2*i-12]^u[2*i-14]^u[2*i-16],u[2*i-2]^u[2*i-4]^u[2*i-6]^u[2*i-8]^u[2*i-10]^u[2*i-12]^u[2*i-14]^u[2*i-16]^u[2*i-18]^u[2*i-20]^u[2*i-22]^u[2*i-24]^u[2*i-26]^u[2*i-28]^u[2*i-30]^u[2*i-32],u[2*i-2]^u[2*i-4]^u[2*i-6]^u[2*i-8]^u[2*i-10]^u[2*i-12]^u[2*i-14]^u[2*i-16]^u[2*i-18]^u[2*i-20]^u[2*i-22]^u[2*i-24]^u[2*i-26]^u[2*i-28]^u[2*i-30]^u[2*i-32]^u[2*i-34]^u[2*i-36]^u[2*i-38]^u[2*i-40]^u[2*i-42]^u[2*i-44]^u[2*i-46]^u[2*i-48]^u[2*i-50]^u[2*i-52]^u[2*i-54]^u[2*i-56]^u[2*i-58]^u[2*i-60]^u[2*i-62]^u[2*i-64],u[2*i-2]^u[2*i-3],u[2*i-2]^u[2*i-3]^u[2*i-34]^u[2*i-35],u[2*i-2]^u[2*i-3]^u[2*i-18]^u[2*i-19],u[2*i-2]^u[2*i-3]^u[2*i-18]^u[2*i-19]^u[2*i-34]^u[2*i-35]^u[2*i-50]^u[2*i-51],u[2*i-2]^u[2*i-3]^u[2*i-10]^u[2*i-11],u[2*i-2]^u[2*i-3]^u[2*i-10]^u[2*i-11]^u[2*i-34]^u[2*i-35]^u[2*i-42]^u[2*i-43],u[2*i-2]^u[2*i-3]^u[2*i-10]^u[2*i-11]^u[2*i-18]^u[2*i-19]^u[2*i-26]^u[2*i-27],u[2*i-2]^u[2*i-3]^u[2*i-10]^u[2*i-11]^u[2*i-18]^u[2*i-19]^u[2*i-26]^u[2*i-27]^u[2*i-34]^u[2*i-35]^u[2*i-42]^u[2*i-43]^u[2*i-50]^u[2*i-51]^u[2*i-58]^u[2*i-59],u[2*i-2]^u[2*i-3]^u[2*i-6]^u[2*i-7],u[2*i-2]^u[2*i-3]^u[2*i-6]^u[2*i-7]^u[2*i-34]^u[2*i-35]^u[2*i-38]^u[2*i-39],u[2*i-2]^u[2*i-3]^u[2*i-6]^u[2*i-7]^u[2*i-18]^u[2*i-19]^u[2*i-22]^u[2*i-23],u[2*i-2]^u[2*i-3]^u[2*i-6]^u[2*i-7]^u[2*i-18]^u[2*i-19]^u[2*i-22]^u[2*i-23]^u[2*i-34]^u[2*i-35]^u[2*i-38]^u[2*i-39]^u[2*i-42],u[2*i-2]^u[2*i-3]^u[2*i-6]^u[2*i-7]^u[2*i-10]^u[2*i-11]^u[2*i-14]^u[2*i-15],u[2*i-2]^u[2*i-3]^u[2*i-6]^u[2*i-7]^u[2*i-10]^u[2*i-11]^u[2*i-14]^u[2*i-15]^u[2*i-18]^u[2*i-19]^u[2*i-22]^u[2*i-23]^u[2*i-26]^u[2*i-27],u[2*i-2]^u[2*i-3]^u[2*i-6]^u[2*i-7]^u[2*i-10]^u[2*i-11]^u[2*i-14]^u[2*i-15]^u[2*i-18]^u[2*i-19]^u[2*i-22]^u[2*i-23]^u[2*i-26]^u[2*i-27]^u[2*i-30],u[2*i-2]^u[2*i-3]^u[2*i-4]^u[2*i-5],u[2*i-2]^u[2*i-3]^u[2*i-4]^u[2*i-5]^u[2*i-34]^u[2*i-35],u[2*i-2]^u[2*i-3]^u[2*i-4]^u[2*i-5]^u[2*i-18]^u[2*i-19],u[2*i-2]^u[2*i-3]^u[2*i-4]^u[2*i-5]^u[2*i-18]^u[2*i-19]^u[2*i-34]^u[2*i-35],u[2*i-2]^u[2*i-3]^u[2*i-4]^u[2*i-5]^u[2*i-10]^u[2*i-11],u[2*i-2]^u[2*i-3]^u[2*i-4]^u[2*i-5]^u[2*i-10]^u[2*i-11]^u[2*i-34]^u[2*i-35],u[2*i-2]^u[2*i-3]^u[2*i-4]^u[2*i-5]^u[2*i-10]^u[2*i-11]^u[2*i-18]^u[2*i-19],u[2*i-2]^u[2*i-3]^u[2*i-4]^u[2*i-5]^u[2*i-10]^u[2*i-11]^u[2*i-18]^u[2*i-19]^u[2*i-34]^u[2*i-35],u[2*i-2]^u[2*i-3]^u[2*i-4]^u[2*i-5]^u[2*i-10]^u[2*i-11]^u[2*i-12]^u[2*i-13],u[2*i-2]^u[2*i-3]^u[2*i-4]^u[2*i-5]^u[2*i-10]^u[2*i-11]^u[2*i-12]^u[2*i-13]^u[2*i-34]^u[2*i-35],u[2*i-2]^u[2*i-3]^u[2*i-4]^u[2*i-5]^u[2*i-10]^u[2*i-11]^u[2*i-12]^u[2*i-13]^u[2*i-18]^u[2*i-19]^u[2*i-20]^u[2*i-21],u[2*i-2]^u[2*i-3]^u[2*i-4]^u[2*i-5]^u[2*i-10]^u[2*i-11]^u[2*i-12]^u[2*i-13]^u[2*i-18]^u[2*i-19]^u[2*i-20]^u[2*i-21]^u[2*i-34]^u[2*i-35],u[2*i-2]^u[2*i-3]^u[2*i-4]^u[2*i-5]^u[2*i-6]^u[2*i-7]^u[2*i-8]^u[2*i-9],u[2*i-2]^u[2*i-3]^u[2*i-4]^u[2*i-5]^u[2*i-6]^u[2*i-7]^u[2*i-8]^u[2*i-9]^u[2*i-10]^u[2*i-11]^u[2*i-12]^u[2*i-13]^u[2*i-14]^u[2*i-15]^u[2*i-16]^u[2*i-17]^u[2*i-34]^u[2*i-35]^u[2*i-36],u[2*i-2]^u[2*i-3]^u[2*i-4]^u[2*i-5]^u[2*i-6]^u[2*i-7]^u[2*i-8]^u[2*i-9]^u[2*i-10]^u[2*i-11]^u[2*i-12]^u[2*i-13]^u[2*i-14]^u[2*i-15]^u[2*i-16]^u[2*i-17]^u[2*i-18]^u[2*i-19]^u[2*i-20]^u[2*i-21]^u[2*i-22]^u[2*i-23]^u[2*i-24]^u[2*i-25],u[2*i-2]^u[2*i-3]^u[2*i-4]^u[2*i-5]^u[2*i-6]^u[2*i-7]^u[2*i-8]^u[2*i-9]^u[2*i-10]^u[2*i-11]^u[2*i-12]^u[2*i-13]^u[2*i-14]^u[2*i-15]^u[2*i-16]^u[2*i-17]^u[2*i-18]^u[2*i-19]^u[2*i-20]^u[2*i-21]^u[2*i-22]^u[2*i-23]^u[2*i-24]^u[2*i-25]^u[2*i-26],u[2*i-2]^u[2*i-3]^u[2*i-4]^u[2*i-5]^u[2*i-6]^u[2*i-7]^u[2*i-8]^u[2*i-9]^u[2*i-10]^u[2*i-11]^u[2*i-12]^u[2*i-13]^u[2*i-14]^u[2*i-15]^u[2*i-16]^u[2*i-17]^u[2*i-18]^u[2*i-19]^u[2*i-20]^u[2*i-21]^u[2*i-22]^u[2*i-23]^u[2*i-24]^u[2*i-25]^u[2*i-26]^u[2*i-27],u[2*i-2]^u[2*i-3]^u[2*i-4]^u[2*i-5]^u[2*i-6]^u[2*i-7]^u[2*i-8]^u[2*i-9]^u[2*i-10]^u[2*i-11]^u[2*i-12]^u[2*i-13]^u[2*i-14]^u[2*i-15]^u[2*i-16]^u[2*i-17]^u[2*i-18]^u[2*i-19]^u[2*i-20]^u[2*i-21]^u[2*i-22]^u[2*i-23]^u[2*i-24]^u[2*i-25]^u[2*i-26]^u[2*i-27]^u[2*i-28],u[2*i-2]^u[2*i-3]^u[2*i-4]^u[2*i-5]^u[2*i-6]^u[2*i-7]^u[2*i-8]^u[2*i-9]^u[2*i-10]^u[2*i-11]^u[2*i-12]^u[2*i-13]^u[2*i-14]^u[2*i-15]^u[2*i-16]^u[2*i-17]^u[2*i-18]^u[2*i-19]^u[2*i-20]^u[2*i-21]^u[2*i-22]^u[2*i-23]^u[2*i-24]^u[2*i-25]^u[2*i-26]^u[2*i-27]^u[2*i-28]^u[2*i-29],u[2*i-2]^u[2*i-3]^u[2*i-4]^u[2*i-5]^u[2*i-6]^u[2*i-7]^u[2*i-8]^u[2*i-9]^u[2*i-10]^u[2*i-11]^u[2*i-12]^u[2*i-13]^u[2*i-14]^u[2*i-15]^u[2*i-16]^u[2*i-17]^u[2*i-18]^u[2*i-19]^u[2*i-20]^u[2*i-21]^u[2*i-22]^u[2*i-23]^u[2*i-24]^u[2*i-25]^u[2*i-26]^u[2*i-27]^u[2*i-28]^u[2*i-29]^u[2*i-30],u[2*i-2]^u[2*i-3]^u[2*i-4]^u[2*i-5]^u[2*i-6]^u[2*i-7]^u[2*i-8]^u[2*i-9]^u[2*i-10]^u[2*i-11]^u[2*i-12]^u[2*i-13]^u[2*i-14]^u[2*i-15]^u[2*i-16]^u[2*i-17]^u[2*i-18]^u[2*i-19]^u[2*i-20]^u[2*i-21]^u[2*i-22]^u[2*i-23]^u[2*i-24]^u[2*i-25]^u[2*i-26]^u[2*i-27]^u[2*i-28]^u[2*i-29]^u[2*i-30]^u[2*i-31]^u[2*i-32]^u[2*i-33]^u[2*i-34]^u[2*i-35]^u[2*i-36]^u[2*i-37]^u[2*i-38]^u[2*i-39]^u[2*i-40]^u[2*i-41]^u[2*i-42]^u[2*i-43]^u[2*i-44]^u[2*i-45]^u[2*i-46]^u[2*i-47]^u[2*i-48]^u[2*i-49]^u[2*i-50]^u[2*i-51]^u[2*i-52]^u[2*i-53]^u[2*i-54]^u[2*i-55]^u[2*i-56]^u[2*i-57]^u[2*i-58]^u[2*i-59]^u[2*i-60]^u[2*i-61]^u[2*i-62]^u[2*i-63]^u[2*i-64]^u[2*i-65]};
endfunction
```

```
wire [(LLR_bit-1):0] input1_128 =
PE(PE(PE(PE(PE(PE(LLR[1],LLR[65],u_counter[5],usum64[0]),PE(LLR[33],LLR[97],u_counter[5],usum64[1]),u_counter[4],usum32[0]),PE(PE(LLR[17],LLR[81],u_counter[5],u
sum64[2]),PE(LLR[49],LLR[113],u_counter[5],usum64[3]),u_counter[4],usum32[1]),u_counter[3],usum16[0]),PE(PE(PE(LLR[9],LLR[73],u_counter[5],usum64[4]),PE(LLR[41
],LLR[105],u_counter[5],usum64[5]),u_counter[4],usum32[2]),PE(PE(LLR[25],LLR[89],u_counter[5],usum64[6]),PE(LLR[57],LLR[121],u_counter[5],usum64[7]),u_counter[4]
,usum32[3]),u_counter[3],usum16[1]),u_counter[2],usum8[0]),PE(PE(PE(PE(LLR[5],LLR[69],u_counter[5],usum64[8]),PE(LLR[37],LLR[101],u_counter[5],usum64[9]),u_coun
ter[4],usum32[4]),PE(PE(LLR[21],LLR[85],u_counter[5],usum64[10]),PE(LLR[53],LLR[117],u_counter[5],usum64[11]),u_counter[4],usum32[5]),u_counter[3],usum16[2]),PE
(PE(PE(LLR[13],LLR[77],u_counter[5],usum64[12]),PE(LLR[45],LLR[109],u_counter[5],usum64[13]),u_counter[4],usum32[6]),PE(PE(LLR[29],LLR[93],u_counter[5],usum64[1
4]),PE(LLR[61],LLR[125],u_counter[5],usum64[15]),u_counter[4],usum32[7]),u_counter[3],usum16[3]),u_counter[2],usum8[1]),u_counter[1],usum4[0]),PE(PE(PE(PE(LL
R[3],LLR[67],u_counter[5],usum64[16]),PE(LLR[35],LLR[99],u_counter[5],usum64[17]),u_counter[4],usum32[8]),PE(PE(LLR[19],LLR[83],u_counter[5],usum64[18]),PE(LLR[
51],LLR[115],u_counter[5],usum64[19]),u_counter[4],usum32[9]),u_counter[3],usum16[4]),PE(PE(PE(LLR[11],LLR[75],u_counter[5],usum64[20]),PE(LLR[43],LLR[107],u_co
unter[5],usum64[21]),u_counter[4],usum32[10]),PE(PE(LLR[27],LLR[91],u_counter[5],usum64[22]),PE(LLR[59],LLR[123],u_counter[5],usum64[23]),u_counter[4],usum32[11
]),u_counter[3],usum16[5]),u_counter[2],usum8[2]),PE(PE(PE(PE(LLR[7],LLR[71],u_counter[5],usum64[24]),PE(LLR[39],LLR[103],u_counter[5],usum64[25]),u_counter[4],
usum32[12]),PE(PE(LLR[23],LLR[87],u_counter[5],usum64[26]),PE(LLR[55],LLR[119],u_counter[5],usum64[27]),u_counter[4],usum32[13]),u_counter[3],usum16[6]),PE(PE(P
E(LLR[15],LLR[79],u_counter[5],usum64[28]),PE(LLR[47],LLR[111],u_counter[5],usum64[29]),u_counter[4],usum32[14]),PE(PE(LLR[31],LLR[95],u_counter[5],usum64[30]),
PE(LLR[63],LLR[127],u_counter[5],usum64[31]),u_counter[4],usum32[15]),u_counter[3],usum16[7]),u_counter[2],usum8[3]),u_counter[1],usum4[1]),u_counter[0],usum2[0
]);

wire [(LLR_bit-1):0] input2_128 =
PE(PE(PE(PE(PE(PE(LLR[2],LLR[66],u_counter[5],usum64[32]),PE(LLR[34],LLR[98],u_counter[5],usum64[33]),u_counter[4],usum32[16]),PE(PE(LLR[18],LLR[82],u_counter[5
],usum64[34]),PE(LLR[50],LLR[114],u_counter[5],usum64[35]),u_counter[4],usum32[17]),u_counter[3],usum16[8]),PE(PE(PE(LLR[10],LLR[74],u_counter[5],usum64[36]),PE
(LLR[42],LLR[106],u_counter[5],usum64[37]),u_counter[4],usum32[18]),PE(PE(LLR[26],LLR[90],u_counter[5],usum64[38]),PE(LLR[58],LLR[122],u_counter[5],usum64[39]),
u_counter[4],usum32[19]),u_counter[3],usum16[9]),u_counter[2],usum8[4]),PE(PE(PE(PE(LLR[6],LLR[70],u_counter[5],usum64[40]),PE(LLR[38],LLR[102],u_counter[5],usu
m64[41]),u_counter[4],usum32[20]),PE(PE(LLR[22],LLR[86],u_counter[5],usum64[42]),PE(LLR[54],LLR[118],u_counter[5],usum64[43]),u_counter[4],usum32[21]),u_counter
[3],usum16[10]),PE(PE(PE(LLR[14],LLR[78],u_counter[5],usum64[44]),PE(LLR[46],LLR[110],u_counter[5],usum64[45]),u_counter[4],usum32[22]),PE(PE(LLR[30],LLR[94],u_
counter[5],usum64[46]),PE(LLR[62],LLR[126],u_counter[5],usum64[47]),u_counter[4],usum32[23]),u_counter[3],usum16[11]),u_counter[2],usum8[5]),u_counter[1],usum4[
2]),PE(PE(PE(PE(PE(LLR[4],LLR[68],u_counter[5],usum64[48]),PE(LLR[36],LLR[100],u_counter[5],usum64[49]),u_counter[4],usum32[24]),PE(PE(LLR[20],LLR[84],u_counter
[5],usum64[50]),PE(LLR[52],LLR[116],u_counter[5],usum64[51]),u_counter[4],usum32[25]),u_counter[3],usum16[12]),PE(PE(PE(LLR[12],LLR[76],u_counter[5],usum64[52])
,PE(LLR[44],LLR[108],u_counter[5],usum64[53]),u_counter[4],usum32[26]),PE(PE(LLR[28],LLR[92],u_counter[5],usum64[54]),PE(LLR[60],LLR[124],u_counter[5],usum64[55
]),u_counter[4],usum32[27]),u_counter[3],usum16[13]),u_counter[2],usum8[6]),PE(PE(PE(PE(LLR[8],LLR[72],u_counter[5],usum64[56]),PE(LLR[40],LLR[104],u_counter[5]
,usum64[57]),u_counter[4],usum32[28]),PE(PE(LLR[24],LLR[88],u_counter[5],usum64[58]),PE(LLR[56],LLR[120],u_counter[5],usum64[59]),u_counter[4],usum32[29]),u_cou
nter[3],usum16[14]),PE(PE(PE(LLR[16],LLR[80],u_counter[5],usum64[60]),PE(LLR[48],LLR[112],u_counter[5],usum64[61]),u_counter[4],usum32[30]),PE(PE(LLR[32],LLR[96
],u_counter[5],usum64[62]),PE(LLR[64],LLR[128],u_counter[5],usum64[63]),u_counter[4],usum32[31]),u_counter[3],usum16[15]),u_counter[2],usum8[7]),u_counter[1],us
um4[3]),u_counter[0],usum2[1]);
```

#2-2 記憶體使用量優化

優化記憶體使用量的動機，是因為我們使用了 look-up table 直接把所有功能做成 bit-level operation。所以，在 synthesis 讀檔時，我們會因為工作站的記憶體空間不足，而被砍掉任務。雖然工作站空閒的狀態下，有機會能合成，但考量到最後幾天的狀況，我們決定轉而優化記憶體使用量。

首先，在 usum_2, usum_4, usum_8, …, usum_256 函數部分，我們做了兩件事。第一，我們發現 usum_256 的偶數位元組成 usum_128, usum_128 的偶數位元組成 usum_64, …, 依此類推。因此，我們只需要保留一個 usum_256 的 look-up table，並且適當的 output corresponding bits 就好。其次，雖然給定回合數 i，我們可以輸出對應的 usum_2^{power}(i)值，但把變數作為 index 會造成過多的 mux。因此，我們決定把 u 在第 i 回合定義成{u[2i], u[2i-1], u[2i-2], …, u[2i+1]}，每個回合做出下圖這樣的迭代方式。這樣一來，就可以用常數組成的 look-up table 來計算 usum_2^{power}。

```
assign pnode_128 = pnode(input1_128, input2_128, frozen[counter], frozen[counter+1]);
assign pnode_256 = pnode(input1_256, input2_256, frozen[counter], frozen[counter+1]);
assign pnode_512 = pnode(input1_512, input2_512, frozen[counter], frozen[counter+1]);
assign pnode_N   = (N==128) ? pnode_128 : ((N==256) ? pnode_256 : pnode_512);

always @(posedge clk or negedge rst_n) begin
        if(!rst_n)
                u <= 0;
        else if(state==DONE)
                u <= 0;
        else if(state==DECODE && counter[0]==0 && counter<N)
                u <= {pnode_N,u[255:2]};
        else
                u <= u;
end
```

```
assign usum_32  =
{usum_256[255],usum_256[247],usum_256[239],usum_256[231],usum_256[223],usum_256[215],usum_256[207],usum_256[199],usum_256[191],usum_2
56[183],usum_256[175],usum_256[167],usum_256[159],usum_256[151],usum_256[143],usum_256[135],usum_256[127],usum_256[119],usum_256[111]
,usum_256[103],usum_256[95],usum_256[87],usum_256[79],usum_256[71],usum_256[63],usum_256[55],usum_256[47],usum_256[39],usum_256[31],u
sum_256[23],usum_256[15],usum_256[7]};
assign usum_16  =
{usum_256[255],usum_256[239],usum_256[223],usum_256[207],usum_256[191],usum_256[175],usum_256[159],usum_256[143],usum_256[127],usum_2
56[111],usum_256[95],usum_256[79],usum_256[63],usum_256[47],usum_256[31],usum_256[15]};
assign usum_8   = {usum_256[255],usum_256[223],usum_256[191],usum_256[159],usum_256[127],usum_256[95],usum_256[63],usum_256[31]};
assign usum_4   = {usum_256[255],usum_256[191],usum_256[127],usum_256[63]};
assign usum_2   = {usum_256[255],usum_256[127]};
```

其次，在 LLR 讀入部分，我們也可以用類似手段，避免掉變數出現在 index 的窘境，讓編譯器的優化更上一層樓。換言之，為了避免編譯器處理棘手的 for loop 展開+register as index of register，我們再度選擇手動展開，來達到最佳的記憶體空間使用量。

```
case (counter) // synopsys full_case
        2:  begin for(i=1; i<=16; i=i+1) LLR[i]     <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        3:  begin for(i=1; i<=16; i=i+1) LLR[i+16]  <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        4:  begin for(i=1; i<=16; i=i+1) LLR[i+32]  <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        5:  begin for(i=1; i<=16; i=i+1) LLR[i+48]  <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        6:  begin for(i=1; i<=16; i=i+1) LLR[i+64]  <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        7:  begin for(i=1; i<=16; i=i+1) LLR[i+80]  <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        8:  begin for(i=1; i<=16; i=i+1) LLR[i+96]  <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        9:  begin for(i=1; i<=16; i=i+1) LLR[i+112] <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        10: begin for(i=1; i<=16; i=i+1) LLR[i+128] <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        11: begin for(i=1; i<=16; i=i+1) LLR[i+144] <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        12: begin for(i=1; i<=16; i=i+1) LLR[i+160] <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        13: begin for(i=1; i<=16; i=i+1) LLR[i+176] <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        14: begin for(i=1; i<=16; i=i+1) LLR[i+192] <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        15: begin for(i=1; i<=16; i=i+1) LLR[i+208] <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        16: begin for(i=1; i<=16; i=i+1) LLR[i+224] <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        17: begin for(i=1; i<=16; i=i+1) LLR[i+240] <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        18: begin for(i=1; i<=16; i=i+1) LLR[i+256] <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        19: begin for(i=1; i<=16; i=i+1) LLR[i+272] <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        20: begin for(i=1; i<=16; i=i+1) LLR[i+288] <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        21: begin for(i=1; i<=16; i=i+1) LLR[i+304] <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        22: begin for(i=1; i<=16; i=i+1) LLR[i+320] <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        23: begin for(i=1; i<=16; i=i+1) LLR[i+336] <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        24: begin for(i=1; i<=16; i=i+1) LLR[i+352] <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        25: begin for(i=1; i<=16; i=i+1) LLR[i+368] <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        26: begin for(i=1; i<=16; i=i+1) LLR[i+384] <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        27: begin for(i=1; i<=16; i=i+1) LLR[i+400] <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        28: begin for(i=1; i<=16; i=i+1) LLR[i+416] <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        29: begin for(i=1; i<=16; i=i+1) LLR[i+432] <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        30: begin for(i=1; i<=16; i=i+1) LLR[i+448] <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        31: begin for(i=1; i<=16; i=i+1) LLR[i+464] <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        32: begin for(i=1; i<=16; i=i+1) LLR[i+480] <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
        33: begin for(i=1; i<=16; i=i+1) LLR[i+496] <={{(LLR_bit-12){in_data[(i*12-1)]}}, in_data[(i*12-1) -: 12]}; end
endcase
```

#2-3 邏輯優化

雖然在 systemverilog 理，可以使用 constant array (localparam)來內建按照 reliability index 的頻道排序；然而，在 verilog 中，據我所知，我們只能使用 register array，並且在改變 N 值時決定對應的頻道排序。這將會合成為一個 10*512 bits 的 flip flop！

不過，考慮到 K<=140 的題目規範，我們第一步就可以把這個 register array 砍到 10*140 bits，減少 73%的 bits 數，自然同時優化 area, power, cycle time threshold。進一步沿著這個思路，由於我們是使用 register array 來決定 frozen bit 是 0 或是 1，我們甚至能直接用 boolean operation 寫出 512 個 frozen bits 的邏輯判斷。換句話說，我們用 boolean operation 就取代掉一個 5120 bits 的 flip flop！如同下圖所示意。

```
if(!rst_n) begin
        for(i=0; i<512; i=i+1)
                frozen[i] <= 1;
end else begin
        frozen[0] <= ( (N==128 && K>=128) ) ? 0 : 1;
        frozen[1] <= ( (N==128 && K>=127) ) ? 0 : 1;
        frozen[2] <= ( (N==128 && K>=126) ) ? 0 : 1;
        frozen[3] <= ( (N==128 && K>=124) ) ? 0 : 1;
        frozen[4] <= ( (N==128 && K>=120) ) ? 0 : 1;
        frozen[5] <= ( (N==128 && K>=112) ) ? 0 : 1;
        frozen[6] <= ( (N==128 && K>=96) ) ? 0 : 1;
        frozen[7] <= ( (N==128 && K>=125) ) ? 0 : 1;
        frozen[8] <= ( (N==128 && K>=123) ) ? 0 : 1;
        frozen[9] <= ( (N==128 && K>=64) ) ? 0 : 1;
        frozen[10] <= ( (N==128 && K>=119) ) ? 0 : 1;
        frozen[11] <= ( (N==128 && K>=122) ) ? 0 : 1;
        frozen[12] <= ( (N==128 && K>=111) ) ? 0 : 1;
        frozen[13] <= ( (N==128 && K>=118) ) ? 0 : 1;
        frozen[14] <= ( (N==128 && K>=110) ) ? 0 : 1;
        frozen[15] <= ( (N==128 && K>=116) || (N==256 && K>=128) ) ? 0 : 1;
        frozen[16] <= ( (N==128 && K>=95) ) ? 0 : 1;
        frozen[17] <= ( (N==128 && K>=63) ) ? 0 : 1;
        frozen[18] <= ( (N==128 && K>=108) ) ? 0 : 1;
        frozen[19] <= ( (N==128 && K>=94) ) ? 0 : 1;
        frozen[20] <= ( (N==128 && K>=104) ) ? 0 : 1;
        frozen[21] <= ( (N==128 && K>=92) ) ? 0 : 1;
        frozen[22] <= ( (N==128 && K>=121) ) ? 0 : 1;
        frozen[23] <= ( (N==128 && K>=62) ) ? 0 : 1;
        frozen[24] <= ( (N==128 && K>=117) || (N==256 && K>=127) ) ? 0 : 1;
        frozen[25] <= ( (N==128 && K>=88) ) ? 0 : 1;
        frozen[26] <= ( (N==128 && K>=60) ) ? 0 : 1;
        frozen[27] <= ( (N==128 && K>=109) ) ? 0 : 1;
        frozen[28] <= ( (N==128 && K>=115) ) ? 0 : 1;
        frozen[29] <= ( (N==128 && K>=80) || (N==256 && K>=126) ) ? 0 : 1;
        frozen[30] <= ( (N==128 && K>=114) ) ? 0 : 1;
        frozen[31] <= ( (N==128 && K>=56) ) ? 0 : 1;
```

3. 技巧使用後的 PPA 比較/理論比較

首先，在” 功能性基礎版”中，我們可以達到演算法上的最少回合數。因為，姑且不考慮 input 和遞迴實作，我們至少要花費一個回合來計算 u[2i],u[2i-1]的結果。然而，因為我們使用 u_sum()和 PE()的 look-up table。所以我們也確實只花費一個回合就達成。然而，考量到 critical path，我們決定把 usum()和 pnode()拆分到奇數回合和偶數回合做計算，讓 crticial path 砍到接近對半。從下圖的 state machine，就可以算出我們的回合數。

```
always @(*) begin
    case(state)
        IDLE:    state_nxt = (en) ? SUB : IDLE;
        SUB:      state_nxt = SUB2;
        SUB2:     state_nxt = READ;
        READ:    state_nxt = (counter == 33) ? DECODE: READ;
        DECODE:  state_nxt = (counter == N-1) ? DONE : DECODE;
        DONE:    state_nxt = SUB;
                 default: state_nxt = IDLE;
    endcase
end
```

其次，在第一階段的優化中，我們從原先無法在 synthesis 開頭階段成功把 DESIGN 讀入，優化到能夠在 5 分鐘內成功讀入，並且在大約 4.5 小時的時間內把 DESIGN 成功合成。在這一版本的合成結果，以 Baseline 為準，cycle count 為 10079 次、cycle time 為 100ns (slack 約 30ns)、power 大約是 1e-2 mW、die area 大約是 5.4 億。

最後，在第二階段的優化中，我們不但減少了一個 5120 bits 的 flip flop，更避免了 register as register index 的窘境(Ex: frozen[sortedCH[i]] for i in range(0,K))。這也將大幅加速了讀取檔案的速度和編譯的難度。在這個階段，我們只要幾秒鐘就能成功讀入檔案，並且在大約 12.5 小時的時間內把 DESIGN 成功合成，編譯時間拉長是因為 cycle time 抓比較嚴格。在這一版本的合成結果，以 Baseline 為準，cycle count 為 10079 次、cycle time 為 50ns (slack 約 0.13ns)、power 大約是 13 mW、die area 大約是 3782 萬、core area 大約是 361 萬。

Reference:

[1] Yuan, Bo. *Algorithm and VLSI Architecture for polar codes decoder*. Diss. University of Minnesota, 2015.

[2] Sorting algorithm - Wikipedia