Digital Communication IC Final Project
Final Project Report

2024/12/25

Student ID : R12K41036, R12K41030
Student Name: Tran Minh Khoa, Nutchanon Jariyanurut

Overview

This report presents the design, implementation, and evaluation of a VLSI system for 4x4 MIMO detection using sphere decoding algorithm. These operations are essential for secure communication in modern systems, including IoT and edge devices. Area and time were evaluated, with a focus on achieving low A*T value.

Basic Idea

Sphere Decoding Algorithm as the core algorithm used in the project. The algorithm aims to solve the Maximum Likelihood (ML) detection problem efficiently by restricting the search space to points within a hypersphere. Several modifications and optimizations, such as depth-first search traversal (DFS) and the use of simplified norms, are explored to enhance hardware implementation.

Design Consideration (Q1 + Q2)

Firstly, we decided to use the depth-first search as the search algorithm because it guarantees that it can find the ML solution. The next decision is how to calculate the cost function. Since from the paper [1], there are three considerations: L2 norm, L1 norm and L-infinity norm. From the Figure 4 of the paper [1], we can see that the L2 norm can perform best because it leads directly to the calculation of ML detection cost function. However, L1 norm is also attractive because it can also approximate the cost function result at the advantage of decreasing the hardware complexity, since it only includes adding and absolute function. L-infinity norm is also attractive because of more hardware simplicity but the degradation of the signal-to-noise ratio (SNR) required to bring the same level of bit error rate (BER) is very severed. Therefore, the L2 norm, L1 norm will be considered in our design consideration only.

First of all, from the received signal $y = \mathbf{H}x$, and from the QR decomposition of $\mathbf{H} = \mathbf{QR}$, then the R is the upper triangular matrix. From

$$y' = Q^H y$$

Then, assuming H is the 4x4 matrix, the cost function can be calculated from the partial Euclidean distance of

$$T_l(x) = \left| y'(l) - \sum_{j=l}^{4} r_{l,j}\, x(j) \right|$$

where $y'(l)$ is the l'th element of y, $x'(l)$ is the l'th element of x, $r_{i,j}$ is the i'th row and j'th column element of R. For the L2 norm, this is exactly how to perform a cost calculation. For the L1 norm, the partial Euclidean distance is approximated according to [1] by

$$T_{l(L1)}(x) = \left|Re\left(T_l(x)\right)\right| + \left|Im\left(T_l(x)\right)\right|$$

That is, to find the absolute value of both real and imaginary part then combine them together.

The L1 norm simulation is done in the MATLAB. Comparison of the SNR vs. BER performance is done with the ML detection using floating-point numbers. The L1 norm DFS detection in both floating-point number and fixed-point numbers using 8 binary fractional bits is used. The comparison of the implementation in VIVADO for fixed-point numbers using 8 binary fractional bits is also done. We use 500 **H** random matrices; each matrix is used to send 11 received signal y. So, the total bits are 500x11x4x3 = 66000 bits. The result of SNR vs. BER curves are shown in Figure 1.

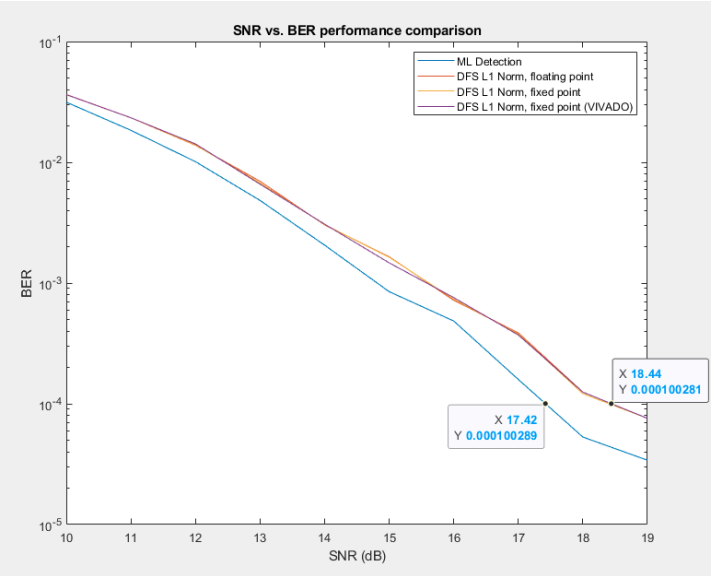


Figure 1: SNR vs. BER performance of using L1 norm in sphere encoding, compared to ML detection.

We can see that all the floating-point number and fixed-point numbers result from the L1 norm cost function calculation are almost the same. However, at the BER = 10^{-4}, the SNR degradation for those is more than 1 dB from the ML detection. Because in the score criteria, more than 0.5 dB of SNR degradation would result in score reduction. Therefore, we will use the L2 norm to calculate the cost function.

For the L2 norm simulation done in the MATLAB, comparison of the SNR vs. BER performance is done with the ML detection using floating-point numbers like before. The L2 norm DFS detection in both floating-point number and fixed-point numbers using 10 binary fractional bits is used. The comparison of the implementation in VIVADO for fixed-point numbers using 10 binary fractional bits is also done. We also use 500 **H** random matrices; each

matrix is used to send 11 received signal *y*. So, the total bits are 500x11x4x3 = 66000 bits. The result of SNR vs. BER curves are shown in Figure 2.

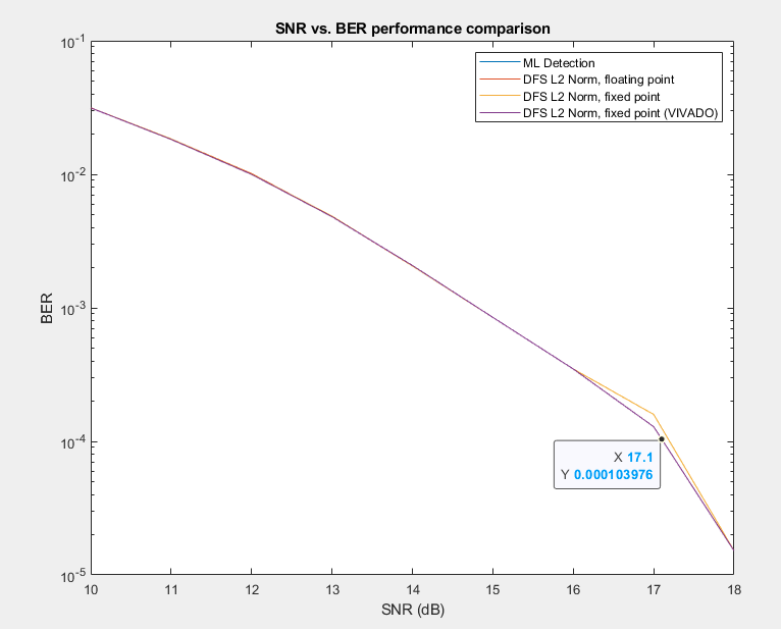


Figure 2: SNR vs. BER performance of using L2 norm in sphere encoding, compared to ML detection.

We can see that the value from the VIVADO's implementation of L2 norm DFS sphere decoding is aligned with the ML detection with no difference, therefore there is no SNR degradation from the ML detection from this implementation. The next question is how many bits in of the fractional bits should be used. To answer this question, we use the SNR=15 dB and measure how much the BER is degraded when we decrease the number of bits in the fractional part. (The reason that we don't use the SNR corresponding to BER equal to 10^{-4} because the error bit number is very low, even we use more than 2000 H matrix, so the result is somehow random at that point.) The result is in the below table:

ML detection BER at SNR = 15 dB	0.000848
L2-norm detection, 10 frac. bits BER at SNR = 15 dB	0.000848
L2-norm detection, 9 frac. bits BER at SNR = 15 dB	0.000833
L2-norm detection, 8 frac. bits BER at SNR = 15 dB	0.000833
L2-norm detection, 7 frac. bits BER at SNR = 15 dB	0.000924
L2-norm detection, 6 frac. bits BER at SNR = 15 dB	0.000924

From this data, it suggests that we should do the 8 bits of fractional part to keep the performance not degraded from ML detection. For leaving some margin, we use the 10 number of bits as the fractional part. The performance of our hardware implementation in VIVADO is already shown in Figure 2.

Hardware design and implementation (Q2 + Q4)

For this section, we will describe about our hardware design. Our hardware is separate into two parts: the first part is the finite-stage machine used to traverse on the tree for depth-first search algorithm, as shown in Figure 3, and the second part is the calculation of cost function which is shown in figure 4.

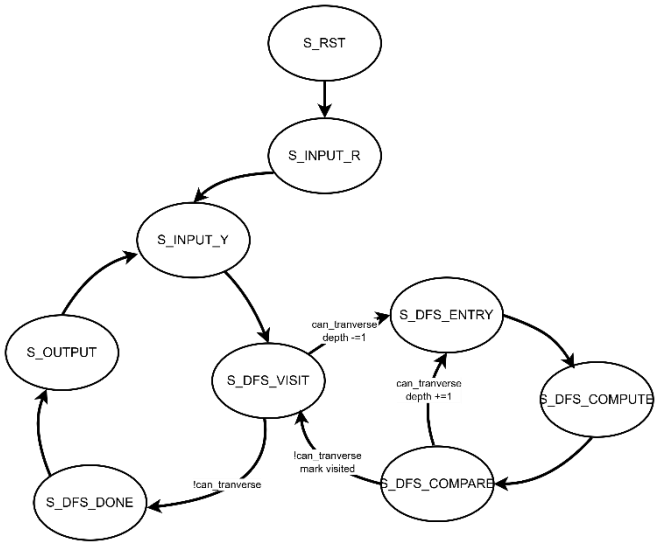


Figure 3: Stage diagram of our finite-stage machine for DFS operation.

The system can be divided into 3 main stages: Input – DFS transversal – Output. The reset signal is used to reset and initialize the system. In input state, we use first 4 cycles to load the R matrix, and the next cycle is for Y matrix. The Y matrix can be reloaded after the system finished the DFS transversal.

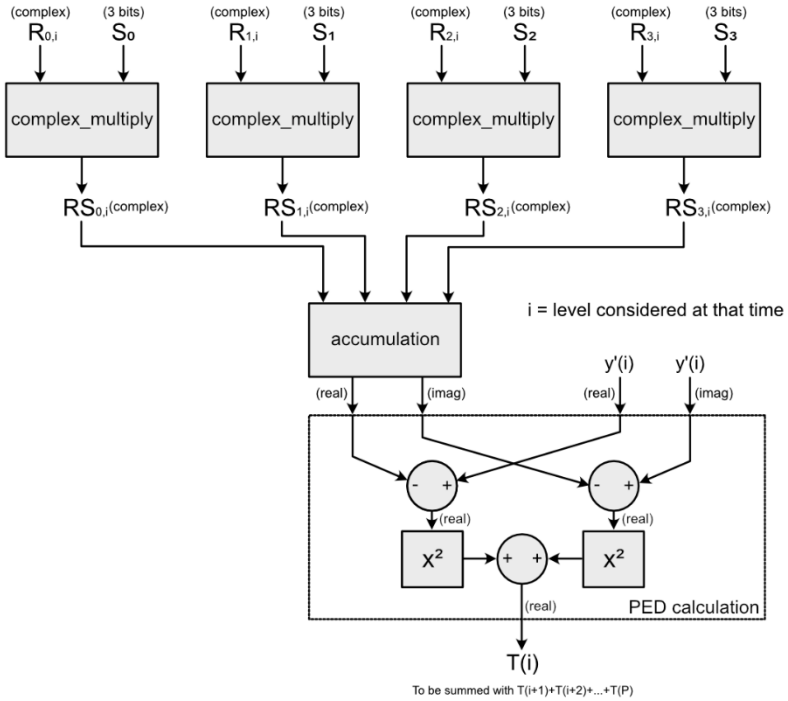


Figure 4: Cost function calculation

Figure 4 shows the part of our hardware which does the cost function calculation. We can see that there are only 4 complex multiplication modules. This is because we do the cost function calculation level by level, therefore the maximum multiplication that we must done in one clock cycle is only 4. For complex multiplication, because the choice of $S_{0,1,2,3}$ is only +1,-1, or +1/sqrt(2), -1/sqrt(2), therefore, for +1,-1 is just the flipping bits or not flipping bits, and for +1/sqrt(2), -1/sqrt(2) we approximate them by 724/1024. After all calculations, we will get the partial Euclidean distance $T(i)$ for the particular level.

The overall hardware system is given in Figure 5.

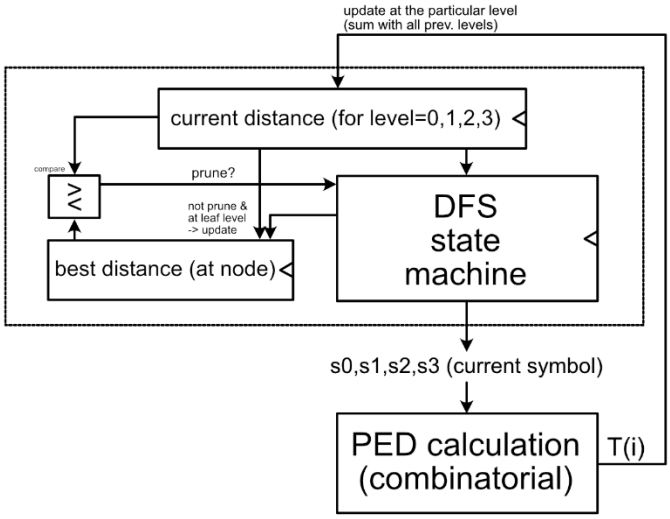


Figure 5: The overall hardware system

The depth-first search (DFT) state machine will traverse to each node. It will send that node to the PED calculation part to calculate the partial Euclidean distance $T(i)$. Then, the current distance registers will be updated according to which level the DFS currently situated. Updating will include to sum up all the previous partial Euclidean distance $T(i+1)+...+T(4)$ so that the current distance of that node is corresponding to that node in its branch. The current distance and the best distance will be compared. If the current distance is more than the best distance, then inform the DFS to not go deeper on that node (or to prune.) If the current distance is less than the best distance, and also the node being considered is the leaf node, then also update the best distance to the current distance.

Hardware I/O (Q3 + Q4)

For our implementation, we use input word length (W_I) of 16 bits with the fractional part of 10 bits. The output is directly the PSK-8 symbol (000, 001, ..., 111), therefore the output bits are 12.

The top level of our design includes these following ports:

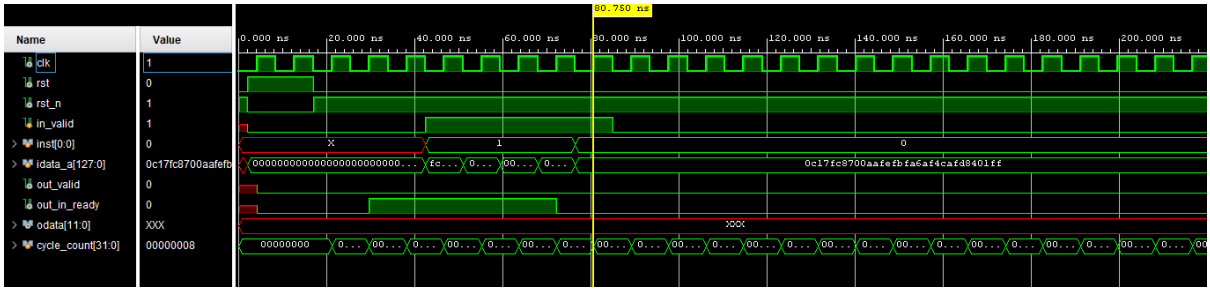
1. Clk: the clock signal. (input, 1 bit)
2. Reset: the reset signal. (input, 1 bit)
3. i_in_valid: to indicate that the incoming signals to Indata is valid, regardless of **R** or y'. (input, 1 bit)
4. flagChannelorData: to indicate that the incoming signals to Indata is **R** or y'. (input, 1 bit)
5. InData: input data (input, 4x2x W_I = 4x2x16 = 128 bits)
6. OutData: output data (output, 4x3 = 12 bits)
7. o_in_ready: to indicate that the input y' is ready to be received again (output, 1 bit)
8. OutputReady: to indicate that the OutData is ready. (output, 1 bit)

Verification (Q5)

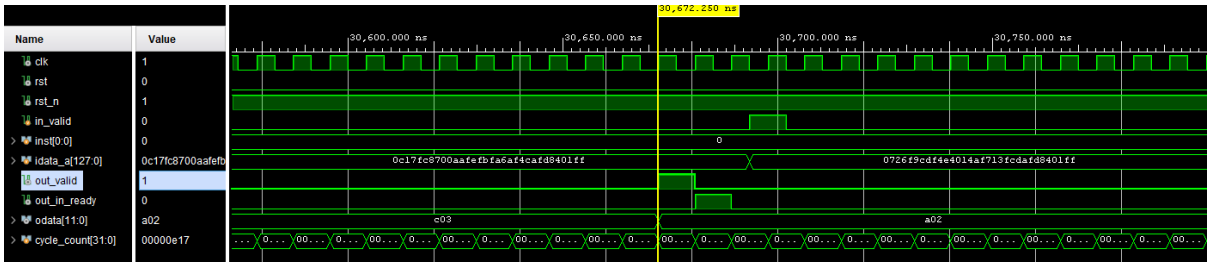
For the verification by the timing diagram, we use SNR = 12 dB to correspond with the BER = 10^{-2} according to Figure 2.

Behavioral simulation:

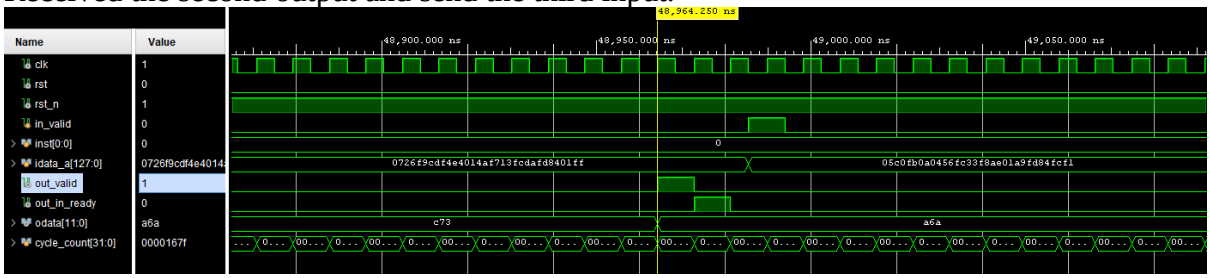
Sending the R and first input.



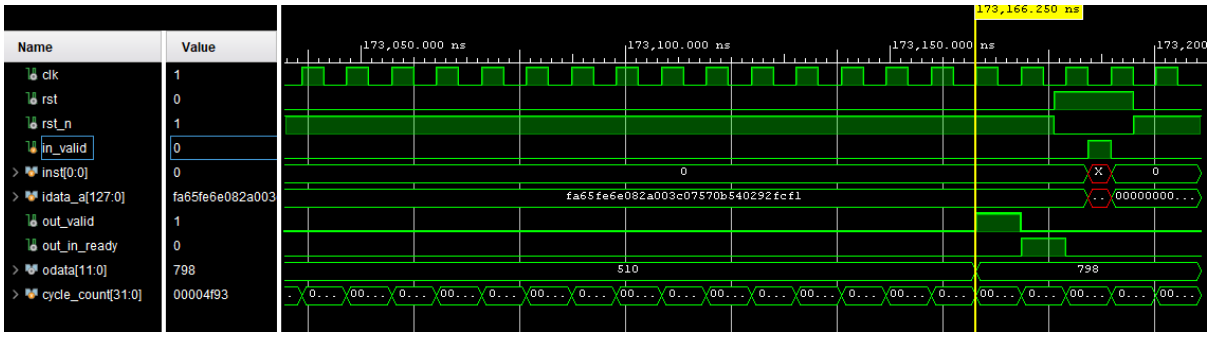
Received the first output and send the second input.



Received the second output and send the third input.

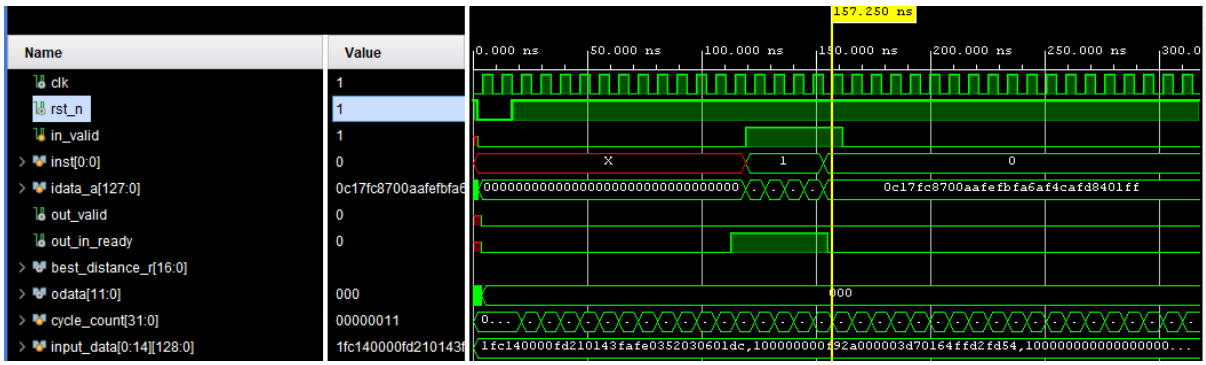


Received the last 11th output.

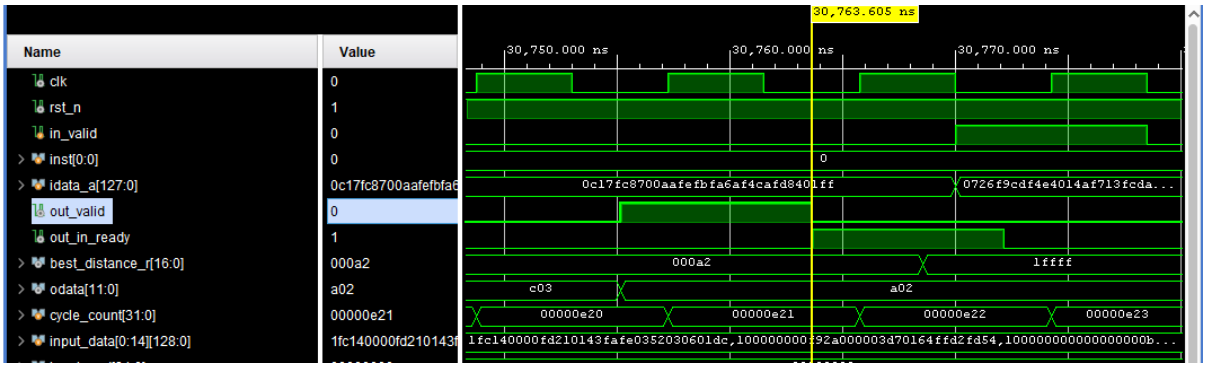


Post-synthesis timing simulation:

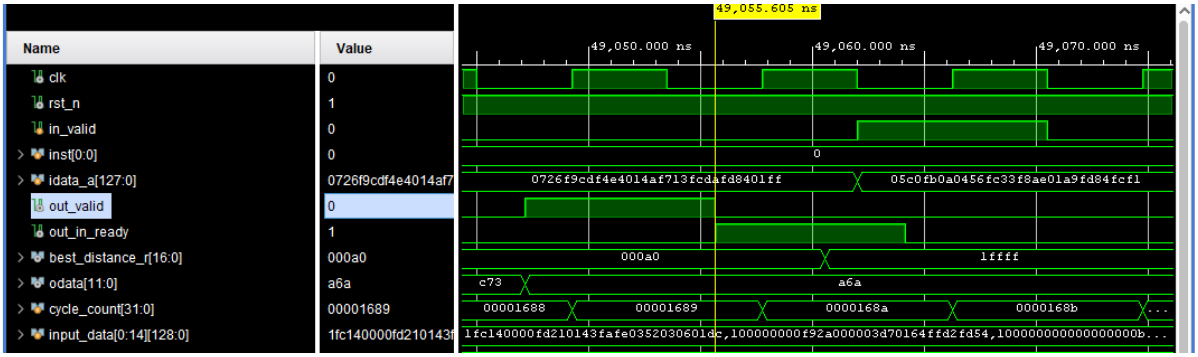
Sending the R and first input.



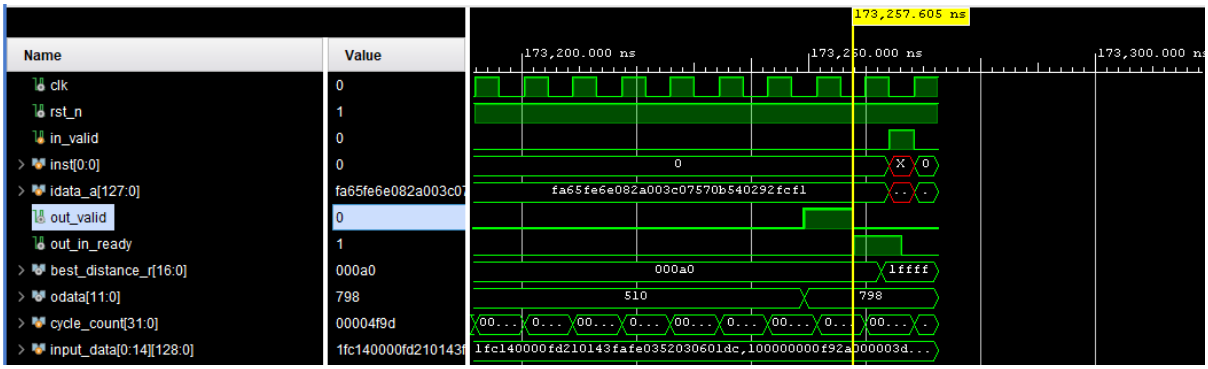
Received the first output and send the second input.



Received the second output and send the third input.



Received the last 11th output.



Clock period setting = 8.5 ns.

M = 173257.605 - 30763.605 = 142494 ns = 16764 clock cycles.

M' = 173257.605 - 157.250 = 173100.355 ns = 20364.75 clock cycles.

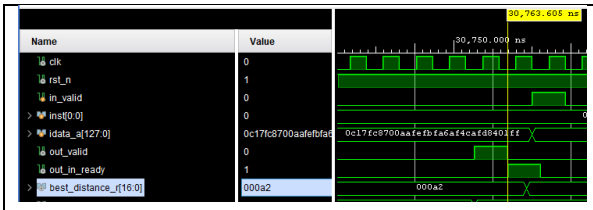
L = 30763.605 - 157.250 = 30606.355 ns = 3600.75 clock cycles.

Processing time = 142494*0.1 = 14249.4 ns.

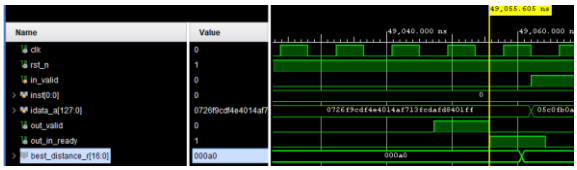
Since Sphere Decoding applies depth first search algorithm, the latency is non-deterministic, depending on the channel conditions. L should have the value at around the average value of 0.1*M.

Benchmark (Q6)

Verifying by the cost function (best_distance_r):

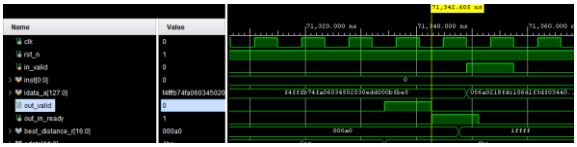


1st output: cost = a2

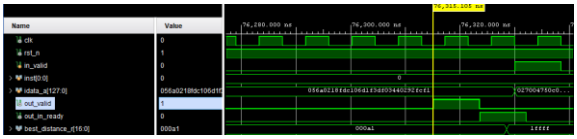


2nd output: cost = a0

3nd output: cost = a1

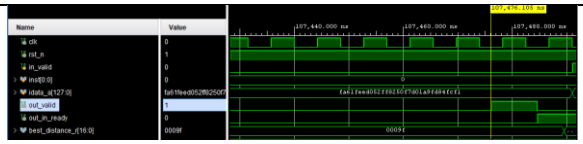


4th output: cost = a0

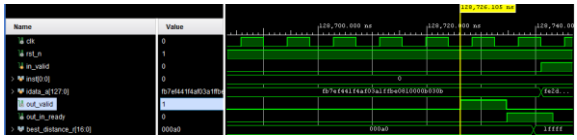


5th output: cost = a1

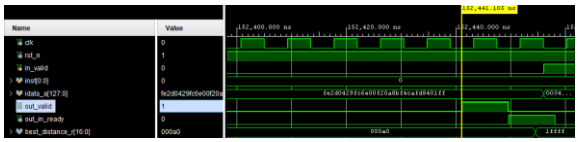
6th output: cost = 9f



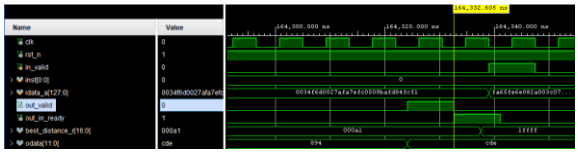
7th output: cost = 9f



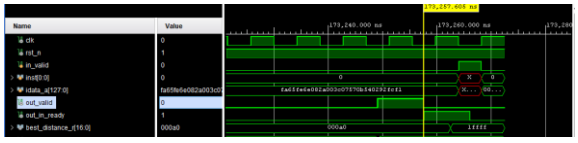
8th output: cost = a0



9th output: cost = a0



10th output: cost = a1



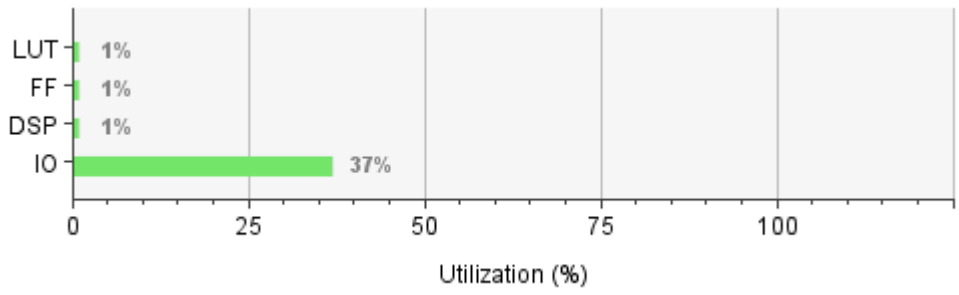
11th output: cost = a0

Cost function from	MATLAB	VIVADO
1st output	0.1572265625	0.158203125
2nd output	0.1572265625	0.15625
3rd output	0.158203125	0.1572265625
4th output	0.1572265625	0.15625
5th output	0.1572265625	0.1572265625
6th output	0.1572265625	0.1552734375
7th output	0.1572265625	0.1552734375
8th output	0.1572265625	0.15625
9th output	0.15625	0.15625
10th output	0.15625	0.1572265625
11th output	0.1552734375	0.15625

Note that the shown values are represented in 16bit fixed point format, which 10 fractional bits. Some discrepancy would come from rounding along the way in the hardware calculation.

AT Evaluation (Q7 + Q8)

Resource	Utilization	Available	Utilization %
LUT	1363	134600	1.01
FF	1278	269200	0.47
DSP	2	740	0.27
IO	146	400	36.50



NA = LUTs + DSPs × 280 + FFs = 3201.

Our design is running at Tcycle = 8.5ns. We run the post-synthesis simulation with test pattern that has BER = 10^{-2} as to evaluate the performance.

AT product = 3201 *14249.4 = 45,612,329.4.

Innovation (Q9)

To reduce the number of multipliers in hardware design, arithmetic shifts and additions are used. We implicitly define multiplication of $\frac{1}{\sqrt{2}}$ as the multiplication of $\frac{181}{256}$, in which multiplication of A*181 can be described as $A + (A \ll 2) + (A \ll 4) + (A \ll 5) + (A \ll 7)$, and division of A/255 can be described as $A \gg 8$. Hence, in our design, only 2 DSP units are used for calculating the square of imaginary part and real part of the distance increments.

The pruning technique is also implemented to reduce the number of paths for transversal.

The reduce the complexity of hardware implementation, we use floor approximation for the cost function, which still has the same result with the pruning decision.

Reference

[1] A. Burg, M. Borgmann, M. Wenk, M. Zellweger, W. Fichtner and H. Bolcskei, "VLSI implementation of MIMO detection using the sphere decoding algorithm," in *IEEE Journal of Solid-State Circuits*, vol. 40, no. 7, pp. 1566-1577, July 2005.