EE521800

Application Acceleration with
High-Level Synthesis

FINAL PROJECT

Wireless Communication System
With Encryption

110061560 吳秉豐
110064521 江威霖
110064513 陳思熙

1. Introduction

Due to the popularity of mobile devices, information or files are often transmitted through wireless communication systems, so it is an inevitable trend to encrypt user privacy.

We try to use HLS to build a complete communication system, with AES-128 encryption algorithm, using U50 verifies whether there is a difference in the received signal of the encryption algorithm in MIMO system, and optimizes its performance.

2. System Architecture

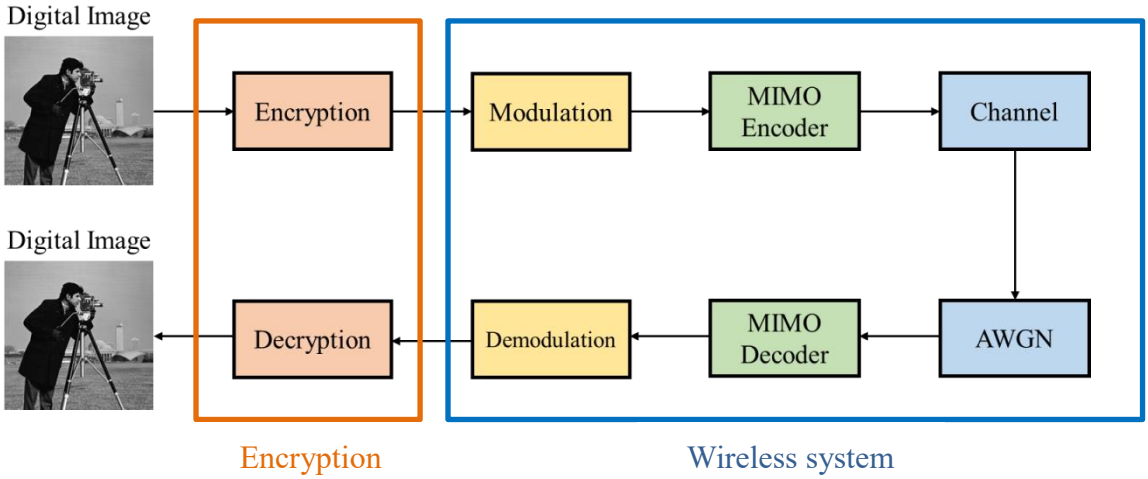


Fig1. System Diagram

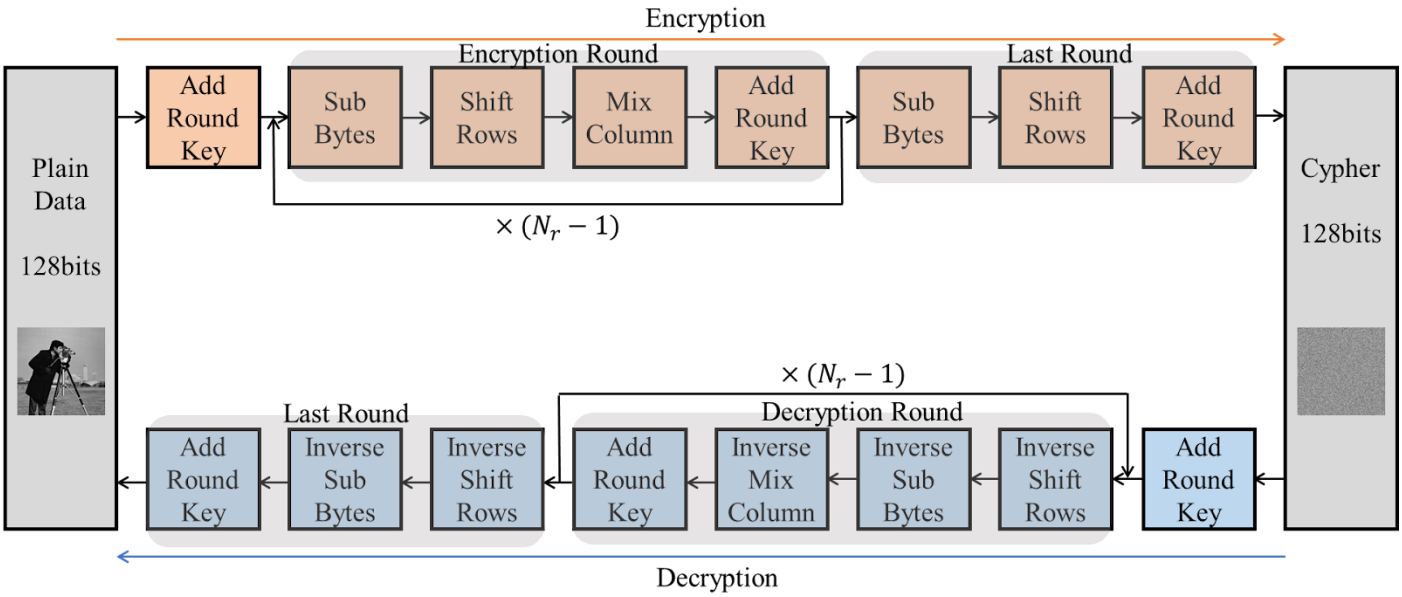
- Encryption
  a. In the transmitter we use AES128 Algorithm to encrypt the signals, after it goes through the channel, we use the same key to decrypt the signals, then compare the results in different SNR.

- Wireless system
  a. Modulation: QPSK
  b. MIMO system: 4 antennas in both transmitter and receiver
  c. Channel type: Rayleigh fading
  d. Noise: Adjustable Additive Gaussian White Noise

3. Algorithm and HLS Implementation

- AES-128
  a. AES (Advanced Encryption Standard) is based on a design principle known as a substitution–permutation network and is efficient in both software and hardware.
  b. The design and strength of all key lengths of the AES algorithm are sufficient to protect classified information up to the top level.

c. AES-128 could be divided into four steps and executed ten times:



I. SubBytes

SubBytes is an invertible non-linear transformation. It uses substitution tables (S-box) for mapping each byte of the state into another byte.

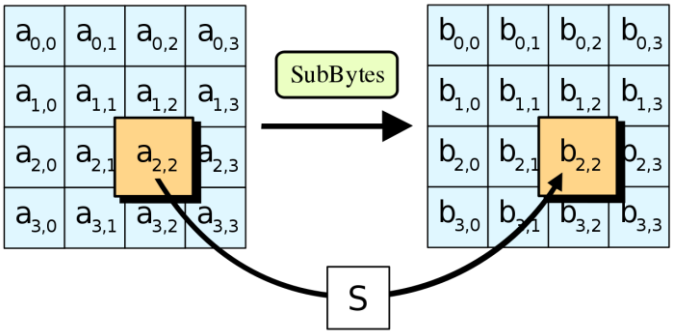
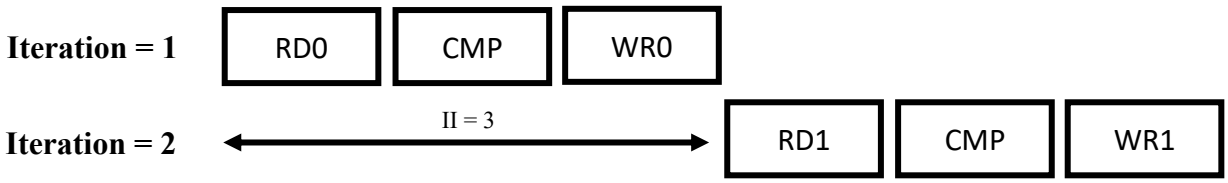


Fig3. SubBytes Schematic

```
static void subBytes(int array[4][4]){
        int i,j;
        subBytes_label0:for(i = 0; i < 4; i++)
                subBytes_label7:for(j = 0; j < 4; j++)
                        array[i][j] = getNumFromSBox(array[i][j]);
}
```

Fig4. SubBytes Implementation with HLS

We put the variable “array” into the subBytes function, then it returns the corresponding S-Box Results.

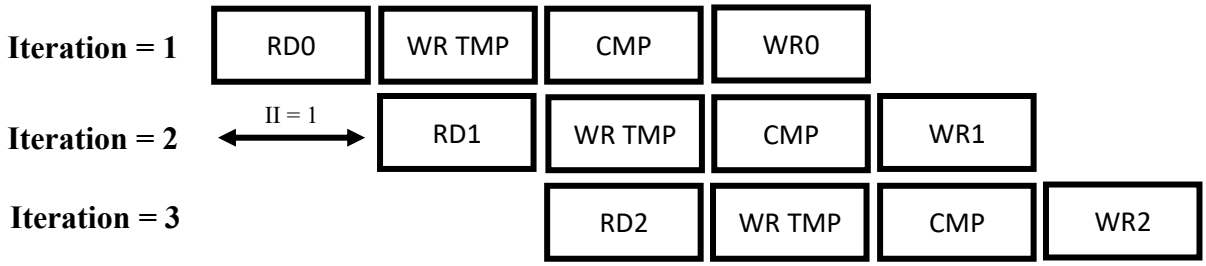


The variable “array” occur Read-After-Write Problem cause II=3 we can deal it with Renaming

```
static void subBytes(int array[4][4]){
        int i,j, tmp_array;
        subBytes_label0:for(i = 0; i < 4; i++)
                subBytes_label7:for(j = 0; j < 4; j++){
                        tmp_array    = array[i][j];
                        array[i][j] = getNumFromSBox( tmp_array );
                }
}
```

Fig5. SubBytes Optimization

After using the variable “temp_array” to renaming the data we can improve the II to 1



✧ Compare the performance

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)
TOP				-	14037	5.610E5
entry_proc				-	0	0.0
AES_En_De28	II Violation			-	7384	2.950E5
Rayleigh_1	II Violation			-	694	2.776E4

Fig6. Result before renaming

Modules & Loops	Issue Type	Violation Ty	Dista	Slack	Latency(cyc	Latency(ns)
TOP				-	14037	5.610E5
entry_proc				-	0	0.0
AES_En_De28	II Violation			-	6859	2.740E5
Rayleigh_1	II Violation			-	694	2.776E4

Fig7. Result after renaming

The latency reduces from 7384 cycles to 6859 cycles.

II. ShiftRows

ShiftRows operation consists basically of a left shift of the second, third and fourth rows of the state matrix by one, two, and three bytes respectively.

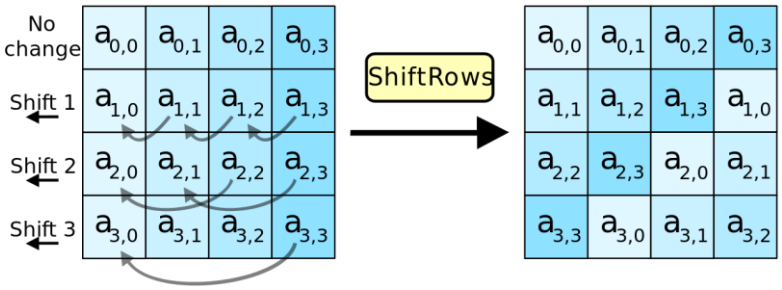


Fig8. ShiftRows Schematic

We only do the shift operation on the second, third, and fourth rows.

```
static void shiftRows(int array[4][4]) {
    int rowTwo[4], rowThree[4], rowFour[4];
    int i;
    shiftRows_label1:for(i = 0; i < 4; i++) {
        rowTwo[i] = array[1][i];
        rowThree[i] = array[2][i];
        rowFour[i] = array[3][i];
    }

    leftLoop4int(rowTwo, 1);
    leftLoop4int(rowThree, 2);
    leftLoop4int(rowFour, 3);

    shiftRows_label8:for(i = 0; i < 4; i++) {
        array[1][i] = rowTwo[i];
        array[2][i] = rowThree[i];
        array[3][i] = rowFour[i];
    }
}
```

Fig9. SubBytes Implementation with HLS

Using “ARRAY_PARTITION” to deal with the data dependence problem.

```
static void shiftRows(int array[4][4]) {
#pragma HLS ARRAY_PARTITION variable=array factor=3 dim=1
    int rowTwo[4], rowThree[4], rowFour[4];
    int i;
    shiftRows_label1:for(i = 0; i < 4; i++) {
        rowTwo[i] = array[1][i];
        rowThree[i] = array[2][i];
        rowFour[i] = array[3][i];
    }

    leftLoop4int(rowTwo, 1);
    leftLoop4int(rowThree, 2);
    leftLoop4int(rowFour, 3);

    shiftRows_label8:for(i = 0; i < 4; i++) {
        array[1][i] = rowTwo[i];
        array[2][i] = rowThree[i];
        array[3][i] = rowFour[i];
    }
}
```

Fig10. SubBytes Optimization

✧ Compare the performance

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)
TOP				-	14037	5.610E5
entry_proc				-	0	0.0
AES_En_De38	II Violation			-	6859	2.740E5
Rayleigh_1	II Violation			-	694	2.776E4
Modulation	II Violation			-	67	2.680E3

Fig11. Result before renaming

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)
TOP				-	14037	5.610E5
entry_proc				-	0	0.0
AES_En_De36	II Violation			-	2989	1.200E5
Rayleigh_1	II Violation			-	694	2.776E4
Modulation	II Violation			-	67	2.680E3

Fig12. Result after renaming

The latency reduces from 6859 cycles to 2989 cycles.

III. MixColumns

Performs a polynomial multiplication in GF(2^8) on each column.

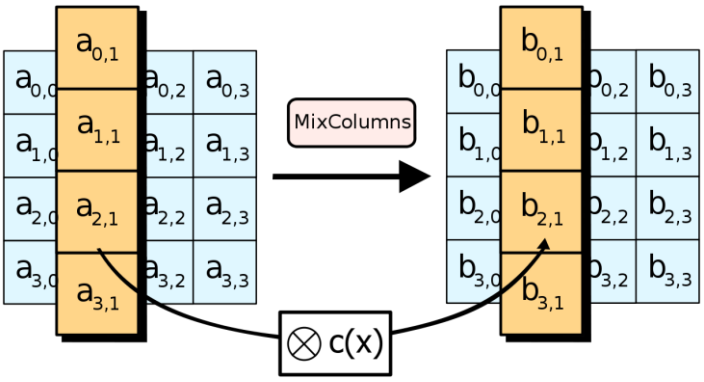


Fig13. MixColumns Schematic

```
static void mixColumns(int array[4][4]) {

    int tempArray[4][4];
    int i,j;
    mixColumns_label4:for(i = 0; i < 4; i++)
        mixColumns_label9:for(j = 0; j < 4; j++)
            tempArray[i][j] = array[i][j];

    mixColumns_label3:for(i = 0; i < 4; i++)
        mixColumns_label10:for(j = 0; j < 4; j++){

            array[i][j] = GFMul(colM[i][0],tempArray[0][j]) ^ GFMul(colM[i][1],tempArray[1][j])
                ^ GFMul(colM[i][2],tempArray[2][j]) ^ GFMul(colM[i][3], tempArray[3][j]);

        }
}
```

Fig14. MixColumns Implementation with HLS

IV. AddRoundKey

AddRoundKey consists in mixing the state array with a round key derived from the cipher key by XOR the bytes in respective positions of the array.

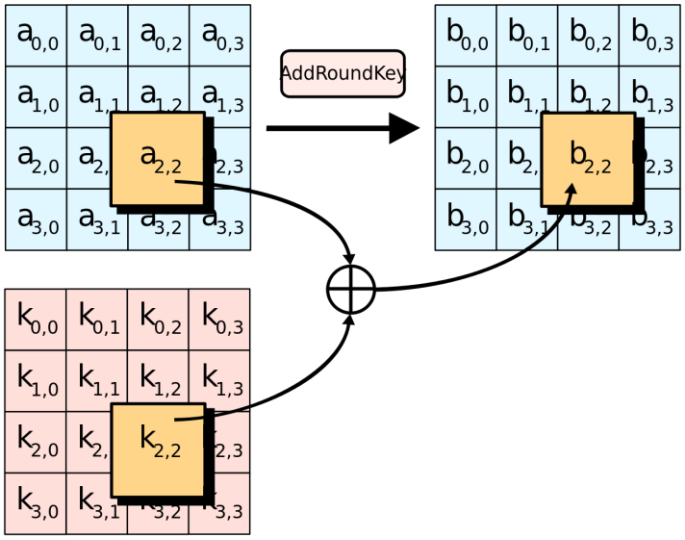


Fig15. AddRoundKey Schematic

```
static void addRoundKey(int array[4][4], int round, int *w) {
    int warray[4];
    int i,j;
    int tmp_array;
    addRoundKey_label0:for(i = 0; i < 4; i++) {
        splitIntToArray(w[round * 4 + i], warray);

        addRoundKey_label6:for(j = 0; j < 4; j++) {
            tmp_array = array[j][i];
            array[j][i] = tmp_array ^ warray[j];
        }
    }
}
```

Fig16. AddRoundKey Implementation with HLS

- Modulation
  a. QPSK(Quadrature phase-shift keying) is known as quadriphase PSK, 4-PSK, or 4-QAM.
  b. QPSK uses four points on the constellation diagram, equispaced around a circle. With four phases, QPSK can encode two bits per symbol, shown in the diagram with Gray coding to minimize the BER (bit error rate)

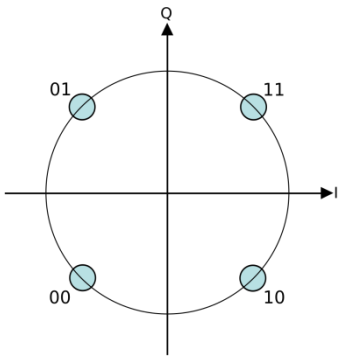


Fig17. QPSK symbols

```
using namespace std;
void Modulation(hls::stream<ap_uint<8>> &in_stream, hls::stream<FIXED_LEN> &xr, hls::stream<FIXED_LEN> &xi){
    int i, data_idx;
    ap_uint<1> temp[8];
    ap_uint<8> data_temp;

#pragma HLS dependence variable = xr inter false
#pragma HLS dependence variable = xi inter false
#pragma HLS dependence variable = xr intra false
#pragma HLS dependence variable = xi intra false

    for(data_idx=0; data_idx<16; data_idx++){
        data_temp = in_stream.read();


        for(i=0; i<8; i++){
            temp[i] = data_temp % 2;
            data_temp = data_temp / 2;
        }


        for(i=0; i<4; i++){
            if(temp[i*2] == 1) xr << 0.7071;
            else xr << -0.7071;

            if(temp[i*2+1] == 1) xi << 0.7071;
            else xi << -0.7071;
        }

    }
}
```

Fig18. Modulation Implementation with HLS

- ✧ Use the binary form to judge "in_stream", if the bit value is 1 → mapping the data to 0.7071, otherwise → mapping the data to -0.7071.
- ✧ The function will output stream xr & xi correspond to the real and image symbol.
- ✧ In different iteration variable xr didn't have dependency, use ***#pragma HLS dependence variable = xr inter false*** to reduce the latency
- ✧ In the same iteration, variable xr didn't have dependency, use ***#pragma HLS dependence variable = xr intra false*** to reduce the latency.
- ✧ Variable xi has the same situation as variable xr, use the same method for optimization.

➢ MIMO System

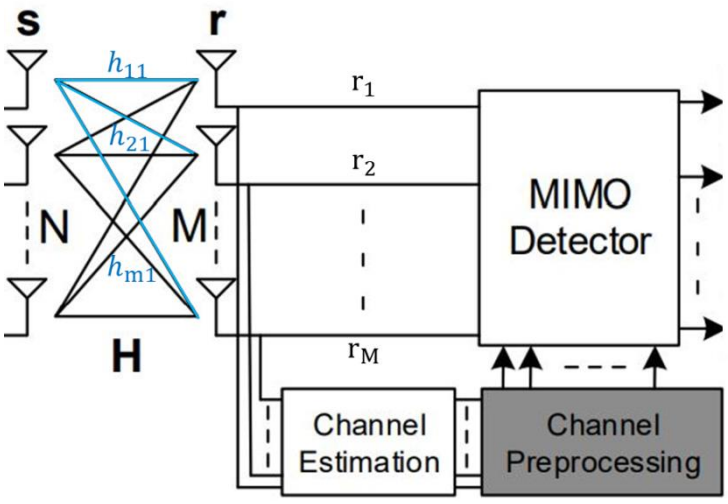


Fig19. MIMO Architecture

a. Matrix Representation

$$R = Hs + \mathcal{N}$$

$$\begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_M \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & \cdots & h_{1N} \\ h_{21} & h_{22} & \cdots & h_{2N} \\ \vdots & & \ddots & \vdots \\ h_{M1} & \cdots & \cdots & h_{MN} \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_N \end{bmatrix}$$

$$\text{R} \rightarrow \text{receive signal} \in \mathbb{C}^{M\times 1}$$
$$\text{H} \rightarrow \text{channel matrix} \in \mathbb{C}^{M\times N}$$
$$\text{S} \rightarrow \text{transmit signal} \in \mathbb{C}^{N\times 1}$$
$$\mathcal{N} \rightarrow \text{noise} \in \mathbb{C}^{M\times 1}$$
$$\text{N} \rightarrow \text{number of transmitter}$$
$$\text{M} \rightarrow \text{number of receiver}$$

b. Use MIMO Equivalent Real Signal Model 2 to represent the complex number

$$\begin{bmatrix} \Re(r_1) \\ \Im(r_1) \\ \Re(r_2) \\ \Im(r_2) \end{bmatrix} = \begin{bmatrix} \Re(h_{1,1}) & -\Im(h_{1,1}) & \Re(h_{1,2}) & -\Im(h_{1,2}) \\ \Im(h_{1,1}) & \Re(h_{1,1}) & \Im(h_{1,2}) & \Re(h_{1,2}) \\ \Re(h_{2,1}) & -\Im(h_{2,1}) & \Re(h_{2,2}) & -\Im(h_{2,2}) \\ \Im(h_{2,1}) & \Re(h_{2,1}) & \Im(h_{2,2}) & \Re(h_{2,2}) \end{bmatrix} \begin{bmatrix} \Re(x_1) \\ \Im(x_1) \\ \Re(x_2) \\ \Im(x_2) \end{bmatrix} + \begin{bmatrix} \Re(n_1) \\ \Im(n_1) \\ \Re(n_2) \\ \Im(n_2) \end{bmatrix}$$

$$\text{R} \rightarrow \text{receive signal} \in \mathbb{R}^{2M\times 1}$$
$$\text{H} \rightarrow \text{channel matrix} \in \mathbb{R}^{2M\times 2N}$$
$$\text{S} \rightarrow \text{transmit signal} \in \mathbb{R}^{2N\times 1}$$
$$\mathcal{N} \rightarrow \text{noise} \in \mathbb{R}^{2M\times 1}$$
$$\text{N} \rightarrow \text{number of transmitter}$$
$$\text{M} \rightarrow \text{number of receiver}$$

c. To get the received MIMO Signal we have to complete the following two steps

✧ Matrix Multiplication with wireless fading Channel

```
void channel_mult(hls::stream<FIXED_LEN> &H_real, hls::stream<FIXED_LEN> &H_imag,
                  hls::stream<FIXED_LEN> &x_real, hls::stream<FIXED_LEN> &x_imag,
                  hls::stream<FIXED_LEN> &out){
    FIXED_LEN DATA_TEMP[8];
    FIXED_LEN CHANNEL[8][8];
    FIXED_LEN TEMP0, TEMP1;
    FIXED_LEN MULT_TEMP;

    int i, j, data_idx;

#pragma HLS ARRAY_PARTITION variable=CHANNEL type=complete dim=0
    CHANNEL2REAL:for(i=0; i<16; i++){
        TEMP0 = H_real.read();
        TEMP1 = H_imag.read();
        CHANNEL[2*(i/4)][2*(i%4)]       =  TEMP0;
        CHANNEL[2*(i/4)+1][(2*(i%4))+1] =  TEMP0;
        CHANNEL[2*(i/4)][2*(i%4)+1]     = -TEMP1;
        CHANNEL[2*(i/4)+1][2*(i%4)]     =  TEMP1;

    }

    for(data_idx=0; data_idx<16; data_idx++){
        DATA2REAL:
        for(i=0; i<4; i++){
            DATA_TEMP[2*i]   = x_real.read();
            DATA_TEMP[2*i+1] = x_imag.read();
        }

        CHANNEL_MULT_LOOP:
        for(i=0; i<8; i++){
            for(j=0; j<8; j++){
                if(j==0) MULT_TEMP = 0;
                MULT_TEMP = MULT_TEMP + CHANNEL[i][j] * DATA_TEMP[j];
                if(j==7) out << MULT_TEMP;
            }
        }
    }
}
```

Fig20. Channel matrix multiplication Implementation with HLS

✧ Add AWGN

After the results come from Matrix Multiplication, we send the results into the AWGN module, then add the Additive Gaussian White Noise, the AWGN module using the Library from the "Vitis Quantitative Finance Library", import the ***xf::fintech::MT19937IcnRng*** to generate the normal distribution random data.

```
extern "C" void AWGN(hls::stream<FIXED_LEN> &din, hls::stream<FIXED_LEN> &dout, FIXED_LEN SNR){

    xf::fintech::MT19937IcnRng<double> rngMT19937ICN;
    rngMT19937ICN.seedInitialization(20); //SEED =20
    FIXED_LEN TEMP;

    for(int j=0; j< 128; j++){
        TEMP = rngMT19937ICN.next();
        dout << din.read() + TEMP * SNR;
    }
}
```

Fig21. AWGN Implementation with HLS

➢ MIMO Decode

a. QR Decomposition of channel matrix H

To Decode the MIMO signals we have to execute the QR decomposition of the channel matrix first, here is an example of QR decomposition to a 2×2 complex channel matrix using the Given Rotation Algorithm:

✧ Step 1

$$\mathbf{H}' = \begin{bmatrix} G_{1a} & G_{1b} & 0 & 0 \\ G_{1c} & G_{1d} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} H_{1,1} & H_{1,2} & H_{1,3} & H_{1,4} \\ H_{2,1} & H_{2,2} & H_{2,3} & H_{2,4} \\ H_{3,1} & H_{3,2} & H_{3,3} & H_{3,4} \\ H_{4,1} & H_{4,2} & H_{4,3} & H_{4,4} \end{bmatrix}$$

$$G_{1a} = G_{1d} = \frac{H_{1,1}}{\sqrt{{H_{1,1}}^2 + {H_{2,1}}^2}} \quad G_{1b} = -G_{1c} = \frac{H_{2,1}}{\sqrt{{H_{1,1}}^2 + {H_{2,1}}^2}}$$

✧ Step 2

$$\mathbf{H}'' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & G_{2a} & G_{2b} \\ 0 & 0 & G_{2c} & G_{2d} \end{bmatrix} \times \mathbf{H}'$$

$$G_{2a} = G_{2d} = \frac{{H'}_{3,1}}{\sqrt{{{H'}_{3,1}}^2 + {{H'}_{4,1}}^2}} \quad G_{2b} = -G_{2c} = \frac{{H'}_{4,1}}{\sqrt{{{H'}_{3,1}}^2 + {{H'}_{4,1}}^2}}$$

✧ Step 3

$$\mathbf{H}''' = \begin{bmatrix} G_{3a} & 0 & G_{3b} & 0 \\ 0 & 1 & 0 & 0 \\ G_{3c} & 0 & G_{3c} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \mathbf{H}''$$

$$G_{3a} = G_{3d} = \frac{{H''}_{1,1}}{\sqrt{{{H''}_{1,1}}^2 + {{H''}_{3,1}}^2}} \quad G_{2b} = -G_{2c} = \frac{{H''}_{3,1}}{\sqrt{{{H''}_{1,1}}^2 + {{H''}_{3,1}}^2}}$$

✧ Step 4

$$\mathbf{H}'''' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & G_{4a} & 0 & G_{4b} \\ 0 & 0 & 1 & 0 \\ 0 & G_{4c} & 0 & G_{4d} \end{bmatrix} \times \mathbf{H}'''$$

$$G_{4a} = G_{4d} = \frac{H'''_{2,2}}{\sqrt{H'''_{2,2}{}^2 + H''_{4,2}{}^2}} \quad G_{4b} = -G_{4c} = \frac{H'''_{4,2}}{\sqrt{H'''_{2,2}{}^2 + H'''_{4,2}{}^2}}$$

✧ Step 5

$$\mathbf{R} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & G_{5a} & G_{5b} \\ 0 & 0 & G_{5c} & G_{5d} \end{bmatrix} \times \mathbf{H}''''$$

$$G_{5a} = G_{5d} = \frac{H''''_{3,3}}{\sqrt{H''''_{3,3}{}^2 + H''''_{4,3}{}^2}} \quad G_{5b} = -G_{5c} = \frac{H''''_{4,3}}{\sqrt{H''''_{3,3}{}^2 + H''''_{4,3}{}^2}}$$

✧ Step 6

$$\mathbf{Q} = \begin{bmatrix} G_{1a} & G_{1b} & 0 & 0 \\ G_{1c} & G_{1d} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^T \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & G_{2a} & G_{2b} \\ 0 & 0 & G_{2c} & G_{2d} \end{bmatrix}^T \times \begin{bmatrix} G_{3a} & 0 & G_{3b} & 0 \\ 0 & 1 & 0 & 0 \\ G_{3c} & 0 & G_{3c} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^T \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & G_{4a} & 0 & G_{4b} \\ 0 & 0 & 1 & 0 \\ 0 & G_{4c} & 0 & G_{4d} \end{bmatrix}^T \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & G_{5a} & G_{5b} \\ 0 & 0 & G_{5c} & G_{5d} \end{bmatrix}^T$$

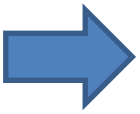
b. QR Decomposition with CORDIC

✧ CORDIC uses only shifts and adds to performing vector rotations

✧ Rotation mode CORDIC

$$\begin{aligned} x_{i+1} &= x_i - y_i \cdot d_i \cdot 2^{-i} \\ y_{i+1} &= y_i + x_i \cdot d_i \cdot 2^{-i} \\ z_{i+1} &= z_i - d_i \cdot \tan^{-1}(2^{-i}) \end{aligned}$$

$$\begin{aligned} d_i &= +1 \text{ when } y_i < 0 \\ d_i &= -1 \text{ when } y_i > 0 \end{aligned}$$

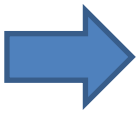


$$\begin{aligned} x_n &= A_n(x_0 \cos(z_0) - y_0 \sin(z_0)) \\ y_n &= A_n(x_0 \sin(z_0) + y_0 \cos(z_0)) \\ z_n &= 0 \\ A_n &= \prod_n \sqrt{1 + 2^{-2i}} \end{aligned}$$

✧ Vector mode CORDIC

$$\begin{aligned} x_{i+1} &= x_i - y_i \cdot d_i \cdot 2^{-i} \\ y_{i+1} &= y_i + x_i \cdot d_i \cdot 2^{-i} \\ z_{i+1} &= z_i - d_i \cdot \tan^{-1}(2^{-i}) \end{aligned}$$

$$\begin{aligned} d_i &= +1 \text{ when } y_i < 0 \\ d_i &= -1 \text{ when } y_i > 0 \end{aligned}$$



$$\begin{aligned} x_n &= A_n\sqrt{x_0{}^2 + y_0{}^2} \\ y_n &= 0 \\ z_n &= z_0 + \tan^{-1}(y_0/x_0) \\ A_n &= \prod_n \sqrt{1 + 2^{-2i}} \end{aligned}$$

✧ Given Rotation with CORDIC

For example $\mathbf{G_1} \times \begin{bmatrix} H_{1,1} & H_{1,3} \\ H_{2,1} & H_{2,3} \\ H_{3,1} & H_{3,3} \\ H_{4,1} & H_{4,3} \end{bmatrix} = \begin{bmatrix} \sqrt{H_{1,1}{}^2 + H_{2,1}{}^2} & H'_{1,3} \\ 0 & H'_{2,3} \\ H_{3,1} & H_{3,3} \\ H_{4,1} & H_{4,3} \end{bmatrix}$ $\mathbf{G_1} = \begin{bmatrix} G_{1a} & G_{1b} & 0 & 0 \\ G_{1c} & G_{1d} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

■ Step 1

Use vector mode CORDIC calculate the angle between $\begin{bmatrix} H_{1,1} \\ H_{2,1} \end{bmatrix}$ and $\sqrt{{H_{1,1}}^2 + {H_{2,1}}^2}$

Input：${x^v}_i = H_{1,1}$；${y^v}_i = H_{2,1}$；${z^v}_i = 0$

Output：${x^v}_o = \sqrt{{H_{1,1}}^2 + {H_{2,1}}^2}$；${z^v}_o = \tan^{-1}(H_{2,1}/H_{1,1}) = \theta$

■ Step2

Use rotation mode CORDIC to rotate $H_{1,3}$ and $H_{2,3}$ by the same angle then use another rotation mode CORDIC to calculate G_{1a} and G_{1b}

Input (cordic 1)：${x^{r1}}_i = H_{1,3}$；${y^{r1}}_i = H_{2,3}$；${z^{r1}}_i = \theta$

Output (cordic 1)：${x^{r1}}_o = H'_{1,3}$；${y^{r1}}_o = H'_{2,3}$

Input (cordic 2)：${x^{r2}}_i = 1$；${y^{r2}}_i = 0$；${z^{r2}}_i = -\theta$

Output (cordic 2)：${x^{r2}}_o = \cos(H_{2,1}/H_{1,1}) = G_{1a}$；${y^{r2}}_o = -\sin(H_{2,1}/H_{1,1}) = G_{1b}$

✧ CORDIC implantation with HLS

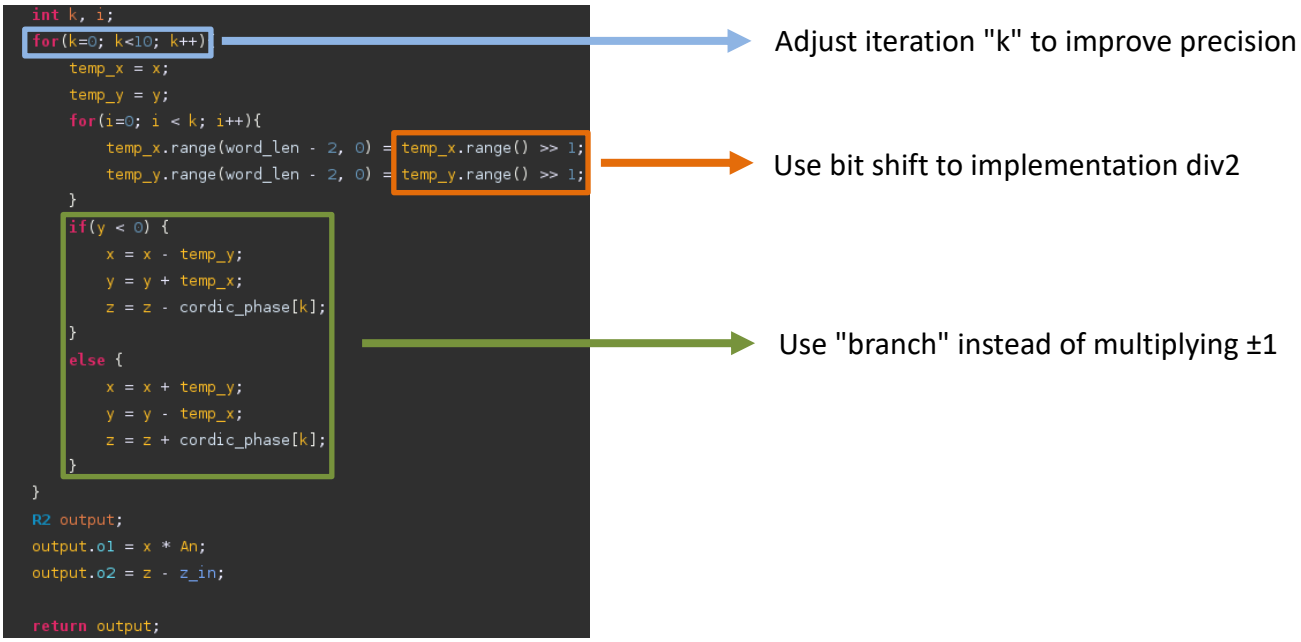


Fig21. CORDIC Implementation with HLS

✧ QR Decomposition optimization with HLS

```
#pragma HLS ARRAY_PARTITION variable=R_I type=complete dim=0
    FIXED_LEN temp;
    for(i=0; i<8; i++){
        for(j=i; j>=0; j--){
            if(j==i){

                R_I[j][0] = R_I[j][0] / H[j][i];
                R_I[j][1] = R_I[j][1] / H[j][i];
                R_I[j][2] = R_I[j][2] / H[j][i];
                R_I[j][3] = R_I[j][3] / H[j][i];
                R_I[j][4] = R_I[j][4] / H[j][i];
                R_I[j][5] = R_I[j][5] / H[j][i];
                R_I[j][6] = R_I[j][6] / H[j][i];
                R_I[j][7] = R_I[j][7] / H[j][i];

                temp = H[j][i];
                H[j][0] = H[j][0] / temp;
                H[j][1] = H[j][1] / temp;
                H[j][2] = H[j][2] / temp;
                H[j][3] = H[j][3] / temp;
                H[j][4] = H[j][4] / temp;
                H[j][5] = H[j][5] / temp;
                H[j][6] = H[j][6] / temp;
                H[j][7] = H[j][7] / temp;
            }
            else{
                R_I[j][0] = R_I[j][0] - R_I[i][0] * H[j][i];
                R_I[j][1] = R_I[j][1] - R_I[i][1] * H[j][i];
                R_I[j][2] = R_I[j][2] - R_I[i][2] * H[j][i];
                R_I[j][3] = R_I[j][3] - R_I[i][3] * H[j][i];
                R_I[j][4] = R_I[j][4] - R_I[i][4] * H[j][i];
                R_I[j][5] = R_I[j][5] - R_I[i][5] * H[j][i];
                R_I[j][6] = R_I[j][6] - R_I[i][6] * H[j][i];
                R_I[j][7] = R_I[j][7] - R_I[i][7] * H[j][i];

                temp = H[j][i];
                H[j][0] = H[j][0] - H[i][0] * temp;
                H[j][1] = H[j][1] - H[i][1] * temp;
                H[j][2] = H[j][2] - H[i][2] * temp;
                H[j][3] = H[j][3] - H[i][3] * temp;
                H[j][4] = H[j][4] - H[i][4] * temp;
                H[j][5] = H[j][5] - H[i][5] * temp;
                H[j][6] = H[j][6] - H[i][6] * temp;
                H[j][7] = H[j][7] - H[i][7] * temp;
            }
        }
    }
```

Use "array partition" to deal the dependency problem

Use "renaming" to deal the dependency problem

Fig22. QR Decomposition Implementation with HLS

```
#pragma HLS ARRAY_PARTITION variable=R_I type=complete dim=0
#pragma HLS ARRAY_PARTITION variable=H type=complete dim=0
    FIXED_LEN R_temp_0, R_temp_1, R_temp_2, R_temp_3, R_temp_4, R_temp_5, R_temp_6, R_temp_7;
    FIXED_LEN H_temp_0, H_temp_1, H_temp_2, H_temp_3, H_temp_4, H_temp_5, H_temp_6, H_temp_7;

    FIXED_LEN R1_temp_0, R1_temp_1, R1_temp_2, R1_temp_3, R1_temp_4, R1_temp_5, R1_temp_6, R1_temp_7;
    FIXED_LEN H1_temp_0, H1_temp_1, H1_temp_2, H1_temp_3, H1_temp_4, H1_temp_5, H1_temp_6, H1_temp_7;
    FIXED_LEN temp;
    for(i=0; i<8; i++){
        for(j=i; j>=0; j--){
            R_temp_0 = R_I[j][0];
            R_temp_1 = R_I[j][1];
            R_temp_2 = R_I[j][2];
            R_temp_3 = R_I[j][3];
            R_temp_4 = R_I[j][4];
            R_temp_5 = R_I[j][5];
            R_temp_6 = R_I[j][6];
            R_temp_7 = R_I[j][7];

            R1_temp_0 = R_I[i][0];
            R1_temp_1 = R_I[i][1];
            R1_temp_2 = R_I[i][2];
            R1_temp_3 = R_I[i][3];
            R1_temp_4 = R_I[i][4];
            R1_temp_5 = R_I[i][5];
            R1_temp_6 = R_I[i][6];
            R1_temp_7 = R_I[i][7];

            H_temp_0 = H[j][0];
            H_temp_1 = H[j][1];
            H_temp_2 = H[j][2];
            H_temp_3 = H[j][3];
            H_temp_4 = H[j][4];
            H_temp_5 = H[j][5];
            H_temp_6 = H[j][6];
            H_temp_7 = H[j][7];

            H1_temp_0 = H[i][0];
            H1_temp_1 = H[i][1];
            H1_temp_2 = H[i][2];
            H1_temp_3 = H[i][3];
            H1_temp_4 = H[i][4];
            H1_temp_5 = H[i][5];
            H1_temp_6 = H[i][6];
            H1_temp_7 = H[i][7];

            temp = H[j][i];
            if(j==i){
                R_I[j][0] = R_temp_0 / temp;
                R_I[j][1] = R_temp_1 / temp;
                R_I[j][2] = R_temp_2 / temp;
                R_I[j][3] = R_temp_3 / temp;
                R_I[j][4] = R_temp_4 / temp;
                R_I[j][5] = R_temp_5 / temp;
                R_I[j][6] = R_temp_6 / temp;
                R_I[j][7] = R_temp_7 / temp;

                H[j][0] = H_temp_0 / temp;
                H[j][1] = H_temp_1 / temp;
                H[j][2] = H_temp_2 / temp;
                H[j][3] = H_temp_3 / temp;
                H[j][4] = H_temp_4 / temp;
                H[j][5] = H_temp_5 / temp;
                H[j][6] = H_temp_6 / temp;
                H[j][7] = H_temp_7 / temp;
            }
            else{
                R_I[j][0] = R_temp_0 - R1_temp_0 * temp;
                R_I[j][1] = R_temp_1 - R1_temp_1 * temp;
                R_I[j][2] = R_temp_2 - R1_temp_2 * temp;
                R_I[j][3] = R_temp_3 - R1_temp_3 * temp;
                R_I[j][4] = R_temp_4 - R1_temp_4 * temp;
                R_I[j][5] = R_temp_5 - R1_temp_5 * temp;
                R_I[j][6] = R_temp_6 - R1_temp_6 * temp;
                R_I[j][7] = R_temp_7 - R1_temp_7 * temp;

                H[j][0] = H_temp_0 - H1_temp_0 * temp;
                H[j][1] = H_temp_1 - H1_temp_1 * temp;
                H[j][2] = H_temp_2 - H1_temp_2 * temp;
                H[j][3] = H_temp_3 - H1_temp_3 * temp;
                H[j][4] = H_temp_4 - H1_temp_4 * temp;
                H[j][5] = H_temp_5 - H1_temp_5 * temp;
                H[j][6] = H_temp_6 - H1_temp_6 * temp;
                H[j][7] = H_temp_7 - H1_temp_7 * temp;
            }
        }
    }
```

Fig23. QR Decomposition after optimization

Fig24. Result before optimization

Fig25. Result after optimization

c. MIMO Detector

According to the result of QR Decomposition, we can replace the system model with the following form to reduce the calculated complexity

$$\mathbf{y} = \mathbf{QRx} + \mathbf{n}$$
$$\mathbf{Q^H y} = \mathbf{Rx} + \mathbf{Q^H n}$$
$$\hat{\mathbf{y}} = \mathbf{Rx} + \mathbf{w}$$
$$\begin{bmatrix}\hat{y}_1\\ \hat{y}_2\\ \hat{y}_3\\ \hat{y}_4\end{bmatrix} = \begin{bmatrix}R_{1,1} & R_{1,2} & R_{1,3} & R_{1,4}\\ 0 & R_{2,2} & R_{2,3} & R_{2,4}\\ 0 & 0 & R_{3,3} & R_{3,4}\\ 0 & 0 & 0 & R_{4,4}\end{bmatrix}\begin{bmatrix}s_1\\ s_2\\ s_3\\ s_4\end{bmatrix} + \mathbf{w}$$
$$s_1 = \Re(x_1) \quad s_2 = \Im(x_1) \quad s_3 = \Re(x_2) \quad s_4 = \Im(x_2)$$

Therefore, the MIMO Detection problem is simplified to:

$$(\hat{s}_1, \hat{s}_2, \hat{s}_3, \hat{s}_4) = \arg \min_{s_1,s_2,s_3,s_4 \in A} \left|\hat{y}_4 - R_{4,4}s_4\right|^2 + \left|\hat{y}_3 - R_{3,3}s_3 - R_{3,4}s_4\right|^2 + \left|\hat{y}_2 - R_{2,2}s_2 - R_{2,3}s_3 - R_{2,4}s_4\right|^2 + \left|\hat{y}_1 - R_{1,1}s_1 - R_{1,2}s_2 - R_{1,3}s_3 - R_{1,4}s_4\right|^2$$

d. K-Best

We can reduce the complexity of the above equation with the K-Best algorithm

Initialization

1) Set one path at level $N_T + 1$ with $T_{N_T+1}(\underline{x}^{(0)}_{N_T+1}) = 0$

Expansion

For $i = N_T : -1 : 1$

2) Expand the K surviving paths to M new possible children in M - QAM domain and calculate the updated PED for each path.

$$T_i(x_k^{(i)}) = T_{i+1}(x_k^{(i+1)}) + \left|\hat{y}_i - \sum_{j=i+1}^{N} \hat{R}_{i,j} \cdot \hat{x}_j^{(k)} - \hat{R}_{i,i} \cdot \hat{x}_i^{(k)}\right|^2 \quad , \hat{x}_i^{(k)} \in C, k = 1, \ldots, KM$$

Sorting

3) sort the K smalles PED as **K**-best surviving paths

End

Fig22. K-Best Algorithm

The goal of FSD is to search the tree structure in a Breadth-First way. The surviving path of each layer of K-Best is fixed, It will retain K surviving path in each layer to reduce the complexity, then choose the smallest surviving path as a predicted result.

✧ K-BEST optimization with HLS

```
#pragma HLS ARRAY_PARTITION variable=x_guess type=complete dim=0

    for(data_idx=0; data_idx<16; data_idx++){



        for(i=0; i<8; i++){
            yy[i] = in_data.read() * sqr_2;
        }


        for(layer=7; layer>=0; layer--){
            PED[0] = 0;
            PED[1] = 0;
            PED[2] = 0;
            PED[3] = 0;

            x_guess[0][layer] =  1;
            x_guess[1][layer] = -1;
            x_guess[2][layer] =  1;
            x_guess[3][layer] = -1;


            for(i = 7; i > layer; i--){
                x_guess[0][i] = survival_path[0][i];
                x_guess[1][i] = survival_path[0][i];
                x_guess[2][i] = survival_path[1][i];
                x_guess[3][i] = survival_path[1][i];
            }
```

Fig26. KBEST Implementation with HLS

We found that variable x_guess causes the high latency so we use the Array Partition to optimize the LOOP.

Fig27. Result before partition x_guess

Fig28. Result after partition x_guess

Also with partition the other variables which have data dependency

```
#pragma HLS ARRAY_PARTITION variable=x_guess type=complete dim=0
#pragma HLS ARRAY_PARTITION variable=PED type=complete dim=0
#pragma HLS ARRAY_PARTITION variable=survival_path type=complete dim=0
#pragma HLS ARRAY_PARTITION variable=dist type=complete dim=0
```

Fig29. Partition other variables

Fig30. Result after partition other variables

We can see that the latency didn't decrease that much after partitioning other variables, we think it's because the variables didn't cause the critical path.

3. Host program and Dataflow Architecture

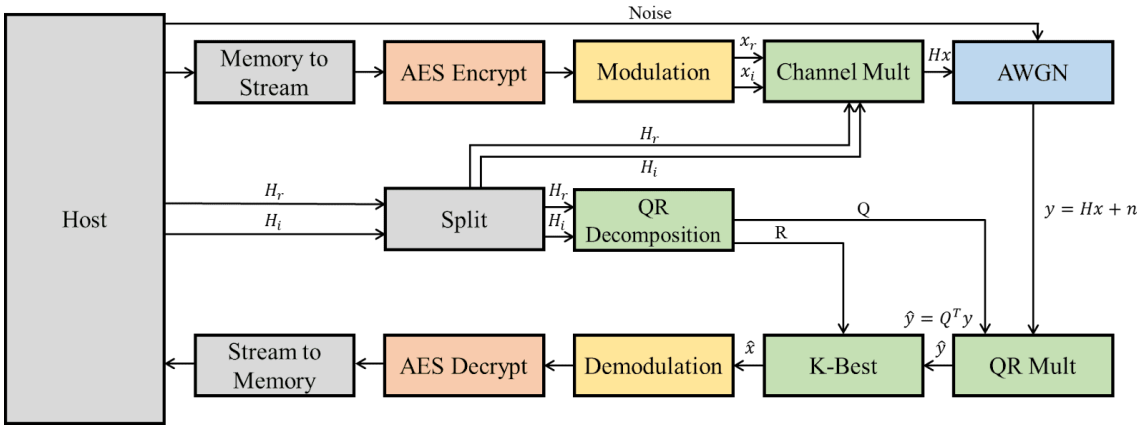


Fig31. Dataflow Architecture

- Host program

```
//traversing all Platforms To find Xilinx Platform and targeted
//Device in Xilinx Platform
cl::Platform::get(&platforms);
for(size_t i = 0; (i < platforms.size() ) & (found_device == false) ;i++){
    cl::Platform platform = platforms[i];
    std::string platformName = platform.getInfo<CL_PLATFORM_NAME>();
    if ( platformName == "Xilinx"){
        devices.clear();
        platform.getDevices(CL_DEVICE_TYPE_ACCELERATOR, &devices);
    if (devices.size()){
        device = devices[0];
        found_device = true;
        break;
    }
    }
}
if (found_device == false){
   std::cout << "Error: Unable to find Target Device "
       << device.getInfo<CL_DEVICE_NAME>() << std::endl;
   return EXIT_FAILURE;
}
```

Fig32. Get the platform & device information

```
OCL_CHECK(err, context = cl::Context(device, NULL, NULL, NULL, &err));
```

Fig33. Generate Context

```
cl::Kernel krnl_vector_add;
```

Fig34. Declaration the Kernel

```
// This call will get the kernel object from program. A kernel is an
// OpenCL function that is executed on the FPGA.
OCL_CHECK(err, krnl_vector_add = cl::Kernel(program,"TOP", &err));

// These commands will allocate memory on the Device. The cl::Buffer objects can
// be used to reference the memory locations on the device.
OCL_CHECK(err, cl::Buffer buffer_a(context, CL_MEM_READ_ONLY, size_in_bytes, NULL, &err));
OCL_CHECK(err, cl::Buffer buffer_result(context, CL_MEM_WRITE_ONLY, size_in_bytes, NULL, &err));
OCL_CHECK(err, cl::Buffer buffer_CHR(context, CL_MEM_READ_ONLY, CHANNEL_size_in_bytes, NULL, &err));
OCL_CHECK(err, cl::Buffer buffer_CHI(context, CL_MEM_READ_ONLY, CHANNEL_size_in_bytes, NULL, &err));
OCL_CHECK(err, cl::Buffer buffer_NOISE(context, CL_MEM_READ_ONLY, NOISE_size_in_bytes, NULL, &err));
//set the kernel Arguments
int narg=0;
OCL_CHECK(err, err = krnl_vector_add.setArg(narg++,buffer_a));
OCL_CHECK(err, err = krnl_vector_add.setArg(narg++,buffer_result));
OCL_CHECK(err, err = krnl_vector_add.setArg(narg++,buffer_CHR));
OCL_CHECK(err, err = krnl_vector_add.setArg(narg++,buffer_CHI));
OCL_CHECK(err, err = krnl_vector_add.setArg(narg++,buffer_NOISE));
```

Fig35. Set Buffer and Connect

```
OCL_CHECK(err, ptr_a      = (ap_uint<8> *)q.enqueueMapBuffer (buffer_a , CL_TRUE , CL_MAP_WRITE , 0, size_in_bytes, NULL, NULL, &err));
OCL_CHECK(err, ptr_result = (ap_uint<8> *)q.enqueueMapBuffer (buffer_result , CL_TRUE , CL_MAP_READ , 0, size_in_bytes, NULL, NULL, &err));
OCL_CHECK(err, ptr_CHR    = (FIXED_LEN *)q.enqueueMapBuffer (buffer_CHR , CL_TRUE , CL_MAP_READ , 0, CHANNEL_size_in_bytes, NULL, NULL, &err));
OCL_CHECK(err, ptr_CHI    = (FIXED_LEN *)q.enqueueMapBuffer (buffer_CHI , CL_TRUE , CL_MAP_READ , 0, CHANNEL_size_in_bytes, NULL, NULL, &err));
OCL_CHECK(err, ptr_NOISE  = (FIXED_LEN *)q.enqueueMapBuffer (buffer_NOISE , CL_TRUE , CL_MAP_READ , 0, NOISE_size_in_bytes, NULL, NULL, &err));
```

Fig36. Write the data

```
// Data will be migrated to kernel space
OCL_CHECK(err, err = q.enqueueMigrateMemObjects({buffer_a, buffer_CHR,  buffer_CHI, buffer_NOISE},0/* 0 means from host*/));
//Launch the Kernel
OCL_CHECK(err, err = q.enqueueTask(krnl_vector_add));

// The result of the previous kernel execution will need to be retrieved in
// order to view the results. This call will transfer the data from FPGA to
// source_results vector
OCL_CHECK(err, q.enqueueMigrateMemObjects({buffer_result},CL_MIGRATE_MEM_OBJECT_HOST));
OCL_CHECK(err, q.finish());
```

Fig37. Execute the kernel

```
OCL_CHECK(err, err = q.enqueueUnmapMemObject(buffer_a , ptr_a));
OCL_CHECK(err, err = q.enqueueUnmapMemObject(buffer_result , ptr_result));
OCL_CHECK(err, err = q.enqueueUnmapMemObject(buffer_CHR , ptr_CHR));
OCL_CHECK(err, err = q.enqueueUnmapMemObject(buffer_CHI , ptr_CHI));
OCL_CHECK(err, err = q.enqueueUnmapMemObject(buffer_NOISE , ptr_NOISE));
OCL_CHECK(err, err = q.finish());
```

Fig38. Store the result

- Dataflow

```
void TOP(ap_uint<8> *in1,  ap_uint<8> *out, FIXED_LEN *CHANNEL_R, FIXED_LEN *CHANNEL_I, FIXED_LEN *NOISE) {
#pragma HLS INTERFACE m_axi port = in1 bundle = gmem0
#pragma HLS INTERFACE m_axi port = out bundle = gmem0
#pragma HLS INTERFACE m_axi port = CHANNEL_R bundle = gmem1
#pragma HLS INTERFACE m_axi port = CHANNEL_I bundle = gmem2
#pragma HLS INTERFACE m_axi port = NOISE bundle = gmem3

    static hls::stream<ap_uint<8>> in1_stream("input_stream_1");

    static hls::stream<FIXED_LEN> NOISE_stream("NOISE_stream");
    static hls::stream<FIXED_LEN> CHANNEL_R_stream("CHANNEL_R_stream");
    static hls::stream<FIXED_LEN> CHANNEL_I_stream("CHANNEL_I_stream");


    static hls::stream<ap_uint<8>> AES_EN_out("AES_EN_out");


    static hls::stream<FIXED_LEN> xr("xr");
    static hls::stream<FIXED_LEN> xi("xi");
    static hls::stream<FIXED_LEN> H_real("H_real");
    static hls::stream<FIXED_LEN> H_imag("H_imag");
    static hls::stream<FIXED_LEN> Q("Q");
    static hls::stream<FIXED_LEN> R("R");
    static hls::stream<FIXED_LEN> H_real_spl0("H_real_spl0");
    static hls::stream<FIXED_LEN> H_real_spl1("H_real_spl1");
    static hls::stream<FIXED_LEN> H_imag_spl0("H_imag_spl0");
    static hls::stream<FIXED_LEN> H_imag_spl1("H_imag_spl1");
    static hls::stream<FIXED_LEN> channel_out("channel_out");
    static hls::stream<FIXED_LEN> noise_out("noise_out");
    static hls::stream<FIXED_LEN> MULQ_out("MULQ_out");
    static hls::stream<FIXED_LEN> KB_out("KB_out");
    static hls::stream<ap_uint<8>> demod_out("demod_out");


    static hls::stream<ap_uint<8>> AES_DE_out("AES_DE_out");
    static ap_uint<8> key[16] = { 0x87, 0x13, 0x11, 0x30,
                                  0x51, 0x87, 0x09, 0xad,
                                  0xcf, 0x1b, 0x66, 0xca,
                                  0xaa, 0xc5, 0x15, 0xb0 };



    static hls::stream<ap_uint<8>> out_stream("output_stream");
    static ap_uint<1> op_en = 0;
    static ap_uint<1> op_de = 1;
#pragma HLS dataflow
    // dataflow pragma instruct compiler to run following three APIs in parallel
    load_input(in1, in1_stream);
    load_CHANNEL(CHANNEL_R, CHANNEL_R_stream);
    load_CHANNEL(CHANNEL_I, CHANNEL_I_stream);
    load_NOISE(NOISE, NOISE_stream);
    AES_En_De(in1_stream, AES_EN_out, op_en, key);
    Modulation(AES_EN_out, xr, xi);
    Rayleigh(H_real, H_imag, CHANNEL_R_stream, CHANNEL_I_stream);
    split(H_real, H_real_spl0, H_real_spl1);
    split(H_imag, H_imag_spl0, H_imag_spl1);
    QRD(H_real_spl1, H_imag_spl1, Q, R);
    channel_mult(H_real_spl0, H_imag_spl0, xr, xi, channel_out);
    AWGN(channel_out, noise_out, NOISE_stream);
    matrix_mult(Q, noise_out, MULQ_out);
    KBEST(R, MULQ_out, KB_out);
    DeModulation(KB_out, demod_out);
    AES_En_De(demod_out, AES_DE_out, op_de, key);
    store_result(out, AES_DE_out);

}
```

Fig38. Kernel with dataflow

4. Result and discussion

➢ System Diagram

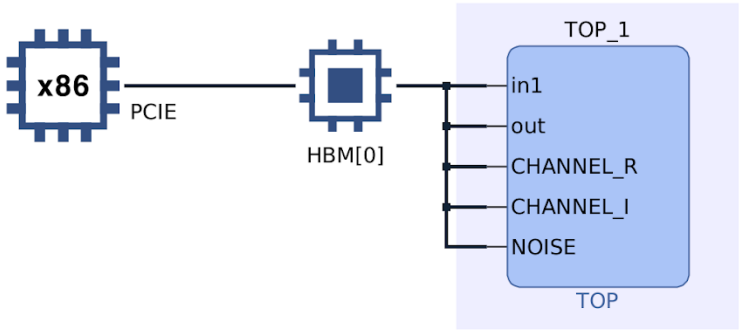


Fig39. System Diagram

➢ System Utilization

Name	Kernel	LUT (% Used)	Register (% Used)	BRAM (% Used)	URAM (% Used)	DSP (% Used)	Calls	CU Utilization (%)	Total Time (ms)	Avg Time (ms)
TOP_1	TOP	59,375 (6.81 %)	52,533 (N/A)	25 (1.86 %)	0 (N/A)	167 (2.81 %)	N/A	N/A	N/A	N/A

Fig40. System Utilization (without AES)

Name	Kernel	LUT (% Used)	Register (% Used)	BRAM (% Used)	URAM (% Used)	DSP (% Used)	Calls	CU Utilization (%)	Total Time (ms)	Avg Time (ms)
TOP_1	TOP	59,978 (6.88 %)	53,780 (N/A)	25 (1.86 %)	0 (N/A)	167 (2.81 %)	N/A	N/A	N/A	N/A

Fig41. System Utilization (with AES)

	LUT	Register	BRAM	DSP
Without ASE	59375	52533	25	167
With ASE	59978	53780	25	167

We can find that the utilization of the system which contains AES is larger than another one.

➢ Hardware Timeline (with AES)

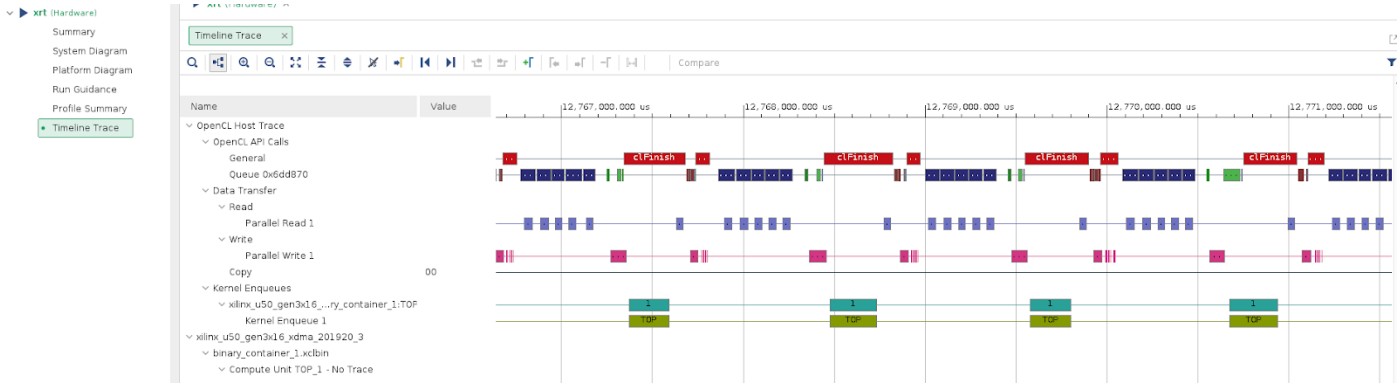


Fig42. Timeline (with AES)

The pattern size is 600×600 pixels, that is $600 \times 600 \times 3 = 1080000$ color data, each data has 4 bits. Using AES-128 can encrypt 16 bytes in one iteration so we need $108000 \div 32 = 33750$ iterations to complete the calculation.

➢ Runtime

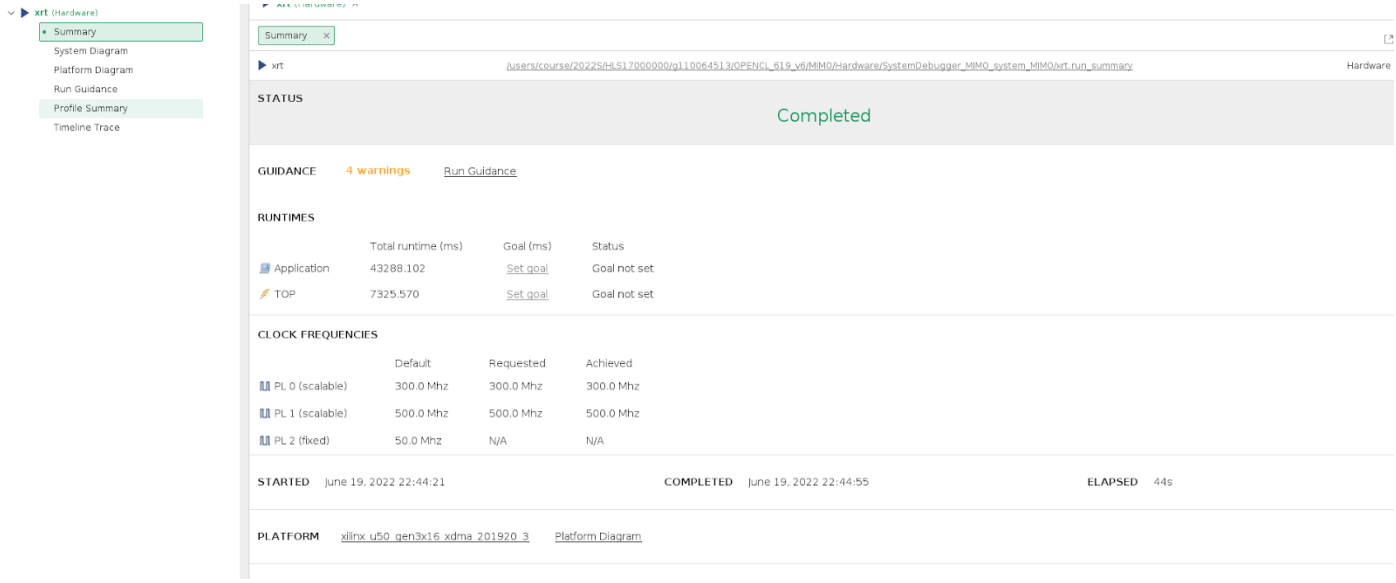


Fig43. Kernel Runtime (without AES)

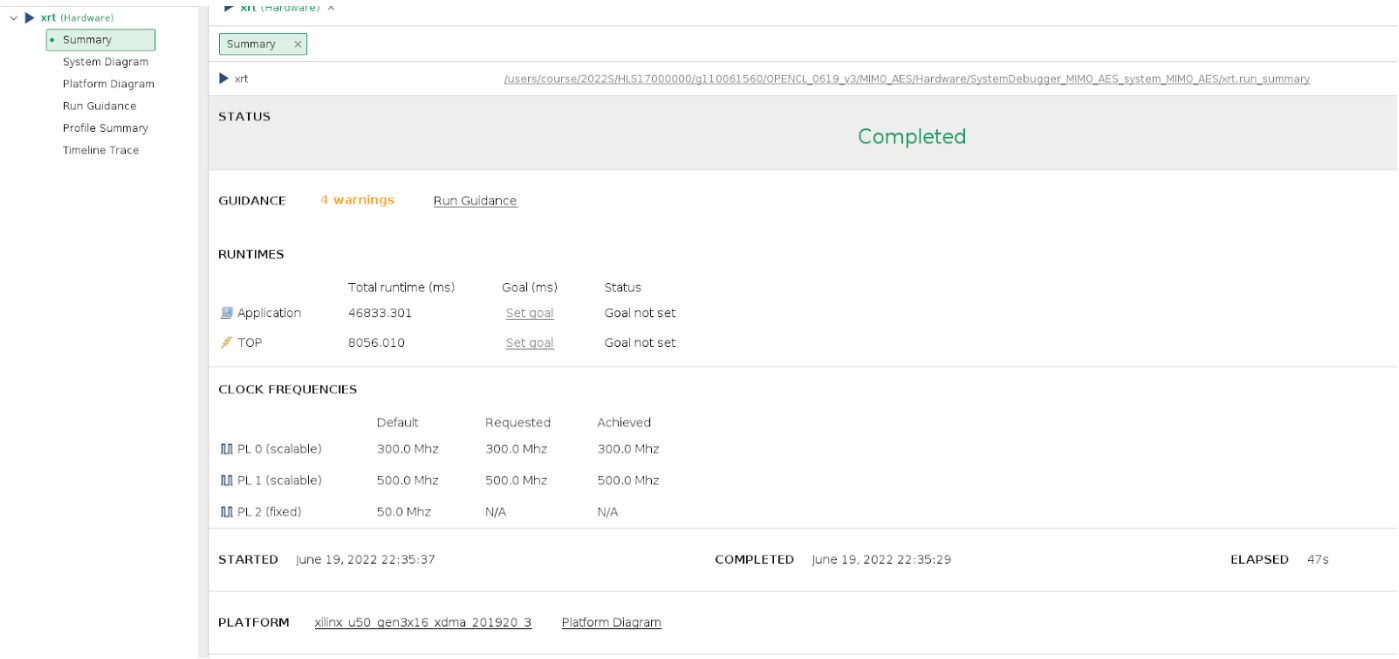
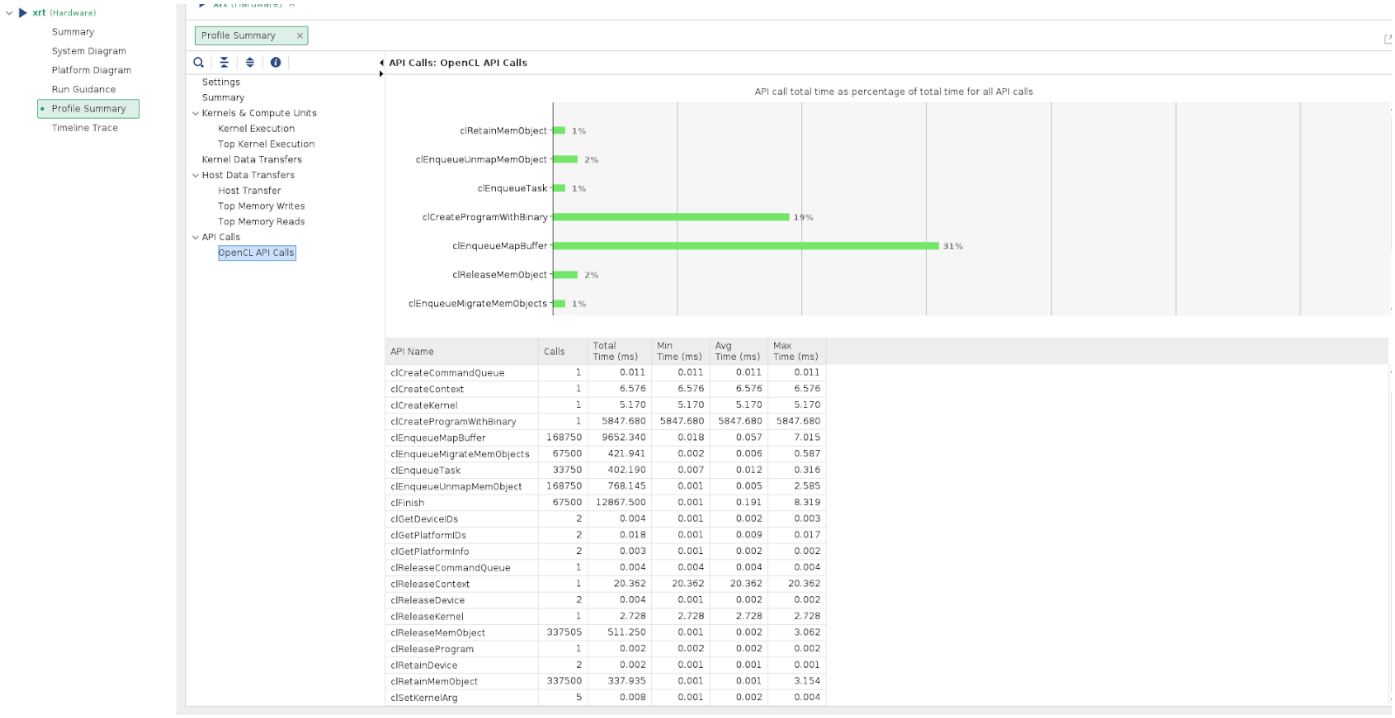


Fig44. Kernel Runtime (with AES)

	Without ASE	With ASE
Kernel runtime	7325.57	8056.01

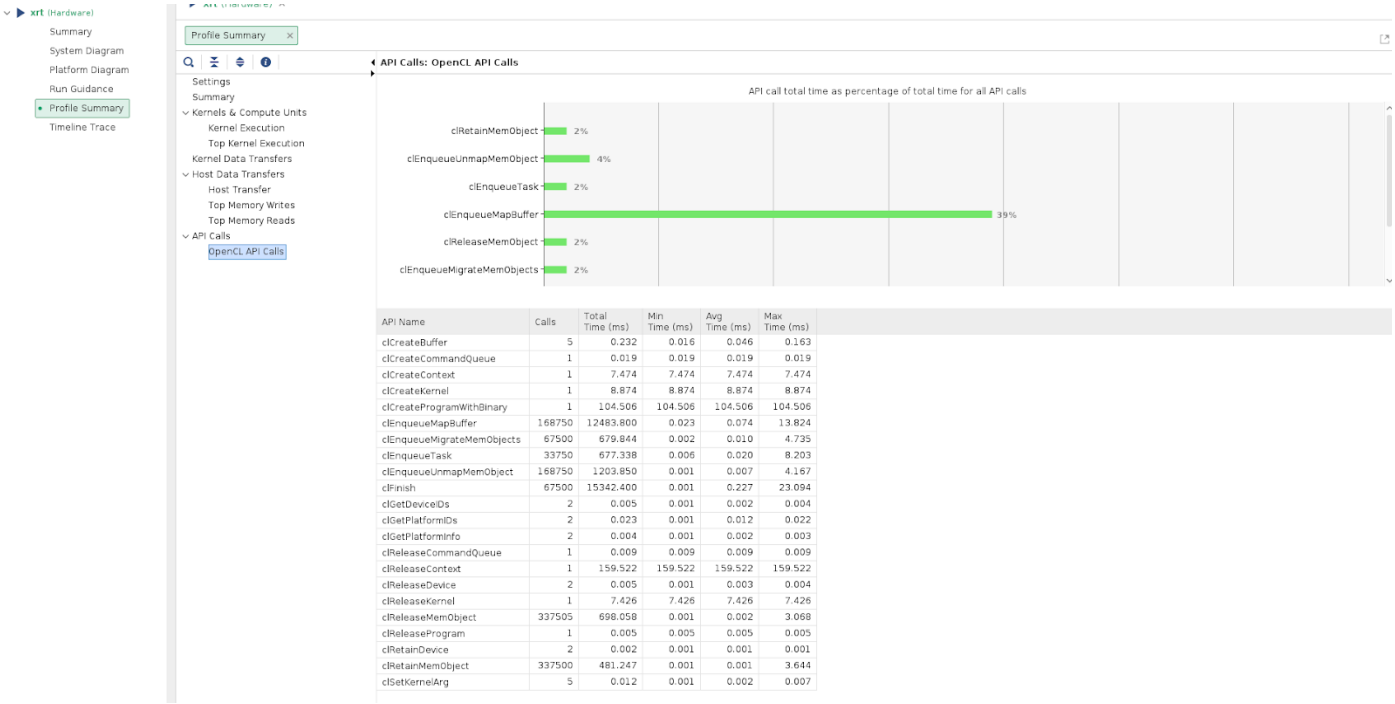
We can find that the runtime of the system which contains AES is larger than the other one

➢ Total API calls



API Name	Calls	Total Time (ms)	Min Time (ms)	Avg Time (ms)	Max Time (ms)
clCreateCommandQueue	1	0.011	0.011	0.011	0.011
clCreateContext	1	6.576	6.576	6.576	6.576
clCreateKernel	1	5.170	5.170	5.170	5.170
clCreateProgramWithBinary	1	5847.680	5847.680	5847.680	5847.680
clEnqueueMapBuffer	168750	9652.340	0.018	0.057	7.015
clEnqueueMigrateMemObjects	67500	421.941	0.002	0.006	0.587
clEnqueueTask	33750	402.190	0.007	0.012	0.316
clEnqueueUnmapMemObject	168750	768.145	0.001	0.005	2.585
clFinish	67500	12867.500	0.001	0.191	8.319
clGetDeviceIDs	2	0.004	0.001	0.002	0.003
clGetPlatformIDs	2	0.018	0.001	0.009	0.017
clGetPlatformInfo	2	0.003	0.001	0.002	0.002
clReleaseCommandQueue	1	0.004	0.004	0.004	0.004
clReleaseContext	1	20.362	20.362	20.362	20.362
clReleaseDevice	2	0.004	0.001	0.002	0.002
clReleaseKernel	1	2.728	2.728	2.728	2.728
clReleaseMemObject	337505	511.250	0.001	0.002	3.062
clReleaseProgram	1	0.002	0.002	0.002	0.002
clRetainDevice	2	0.002	0.001	0.001	0.001
clRetainMemObject	337500	337.935	0.001	0.001	3.154
clSetKernelArg	5	0.008	0.001	0.002	0.004

Fig45. Total API calls (without AES)



API Name	Calls	Total Time (ms)	Min Time (ms)	Avg Time (ms)	Max Time (ms)
clCreateBuffer	5	0.232	0.016	0.046	0.163
clCreateCommandQueue	1	0.019	0.019	0.019	0.019
clCreateContext	1	7.474	7.474	7.474	7.474
clCreateKernel	1	8.874	8.874	8.874	8.874
clCreateProgramWithBinary	1	104.506	104.506	104.506	104.506
clEnqueueMapBuffer	168750	12483.800	0.023	0.074	13.824
clEnqueueMigrateMemObjects	67500	679.844	0.002	0.010	4.735
clEnqueueTask	33750	677.338	0.006	0.020	8.203
clEnqueueUnmapMemObject	168750	1203.850	0.001	0.007	4.167
clFinish	67500	15342.400	0.001	0.227	23.094
clGetDeviceIDs	2	0.005	0.001	0.002	0.004
clGetPlatformIDs	2	0.023	0.001	0.012	0.022
clGetPlatformInfo	2	0.004	0.001	0.002	0.003
clReleaseCommandQueue	1	0.009	0.009	0.009	0.009
clReleaseContext	1	159.522	159.522	159.522	159.522
clReleaseDevice	2	0.005	0.001	0.003	0.004
clReleaseKernel	1	7.426	7.426	7.426	7.426
clReleaseMemObject	337505	698.058	0.001	0.002	3.068
clReleaseProgram	1	0.005	0.005	0.005	0.005
clRetainDevice	2	0.002	0.001	0.001	0.001
clRetainMemObject	337500	481.247	0.001	0.001	3.644
clSetKernelArg	5	0.012	0.001	0.002	0.007

Fig46. Total API calls (with AES)

- Execution Results (without AES)

a. Pattern 1

Fig47. Pattern1 result original image

Fig48. Pattern1 result SNR 12

Fig49. Pattern1 result SNR 9

Fig50. Pattern1 result SNR 6

Fig51. Pattern1 result SNR 3

Fig52. Pattern1 result SNR 0

b. Pattern 2

Fig53. Pattern2 result original image

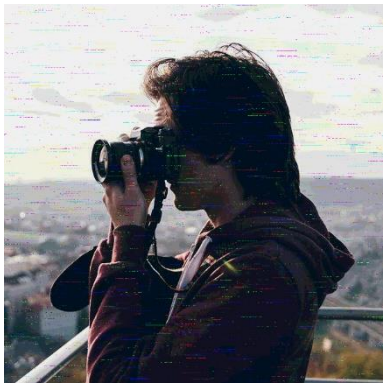


Fig54. Pattern2 result SNR 12

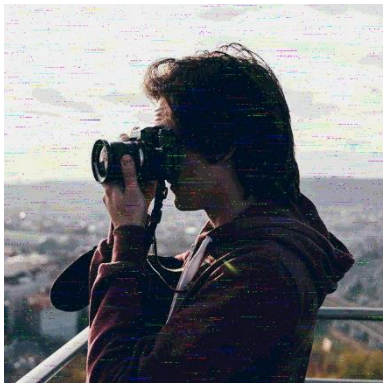


Fig55. Pattern2 result SNR 9

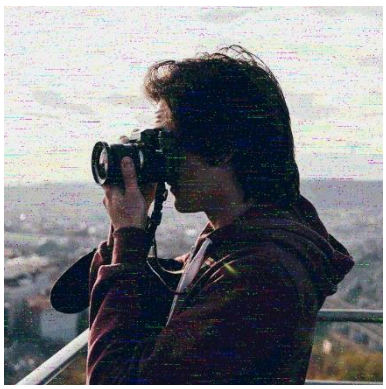


Fig56. Pattern2 result SNR 6

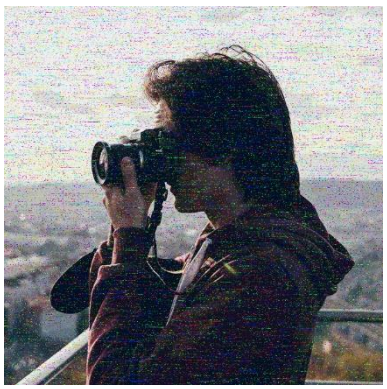


Fig57. Pattern2 result SNR 3

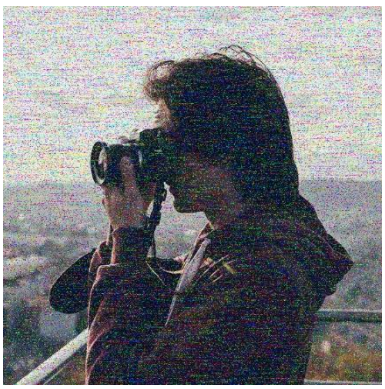


Fig58. Pattern2 result SNR 0

c. Pattern 3

Fig59. Pattern3 result original image

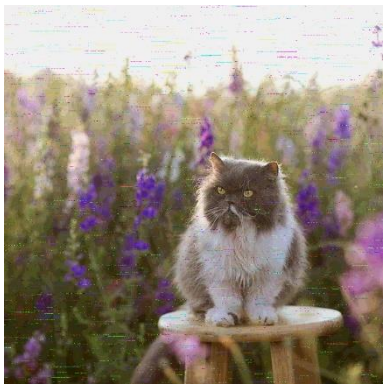


Fig60. Pattern3 result SNR 12

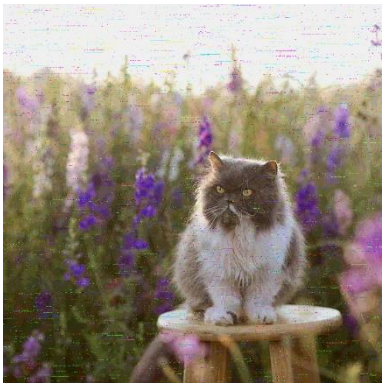


Fig61. Pattern3 result SNR 9

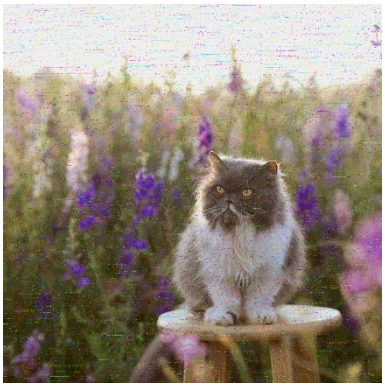


Fig62. Pattern3 result SNR 6

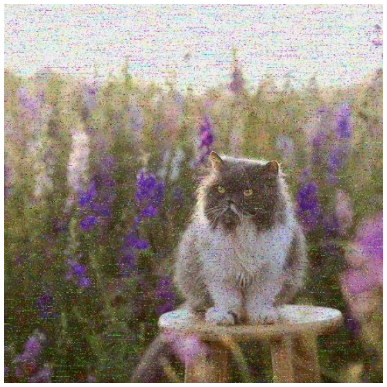
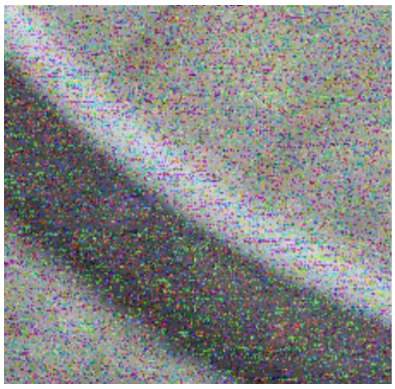


Fig63. Pattern3 result SNR 3

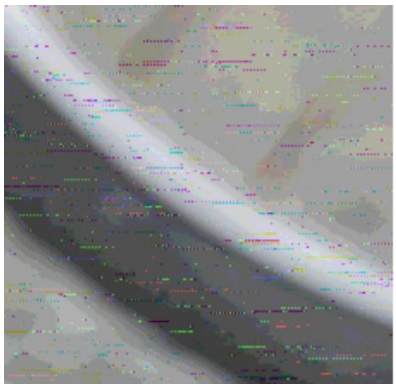
Fig64. Pattern3 result SNR 0

We can find that as the SNR value increases, the picture becomes clearer, while the smaller the SNR value, the more noise there will be.

d. The problem of continuous noise



SNR = 0dB



SNR = 12dB

At $\mathrm{SNR} = 12$, there are obvious continuous errors, because there is less noise now, and the signals are dominated by channel fading gain. When the channel fading gain is too large, it will cause all signals to transmit through the channel prediction error.

- Execution Results (with AES)

Fig65. Pattern1 result SNR 12

Fig66. Pattern1 result SNR 9

Fig67. Pattern1 result SNR 6

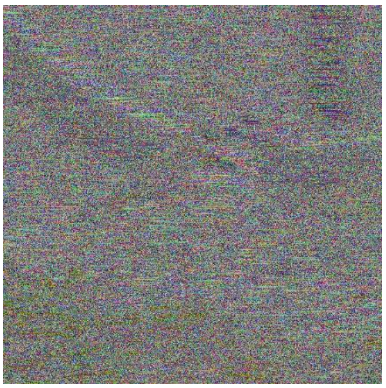


Fig68. Pattern1 result SNR 3

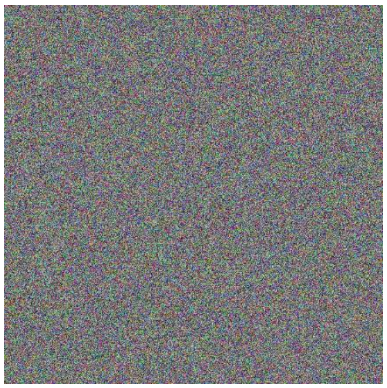


Fig69. Pattern1 result SNR 0

We can find that after AES, when the SNR is low, the data error will be more serious than that without AES. This is because as long as there is a little error in the calculation of AES, the decrypt data will be completely different.

- Analysis (without AES V.S. with AES)

SNR = 6 without AES

SNR = 6 with AES

The above two pictures are the result of comparing the SNR at 6dB. The right picture (with AES) has more obvious errors than the left picture (without AES) because the demodulation error occurs in the same block, which will cause all data in the block to be wrong.

5. Conclusion

- AES algorithm provides a method with high security and simple encrypt step.
- There are a lot of repetitive operations caused by the AES algorithm, so the performance can be improved with the paralleled computing hardware.
- HLS can be useful with optimizing loops, pragma PIPELINE help us to speed up development process.

6. GitHub link

https://github.com/caota985107/HLS_AES_MIMO.git

7. Reference

- https://github.com/WilliamsCeng/02Hero
- https://blog.csdn.net/qq_28205153/article/details/55798628
- https://hackmd.io/@yrHb-fKBRoyrKDEKdPSDWg/HJVX-WH2v
- https://github.com/Xilinx/Vitis_Libraries/tree/master/quantitative_finance/L1/tests/normalRNG