T5 LabC Report

DFT

R11943018 林子軒

Github link: https://github.com/ezpzsyuan00/AAHLS_LabC_DFT.git

1. Briefly introduction to the algorithm or overall system:

   The following picture is the definition of Discrete Fourier Transform(DFT)

$$X[k] = \sum_{n=0}^{N-1} x[n]\, e^{-j2\pi \frac{k}{N} n} = \sum_{n=0}^{N-1} x[n] \left(e^{j2\pi(\frac{-1}{N})}\right)^{kn} = \sum_{n=0}^{N-1} x[n]\, W_N^{kn}$$

   The W term is called twiddle factor. The frequency domain X[k] is the multiplication and addition of the time domain x[k] and W.

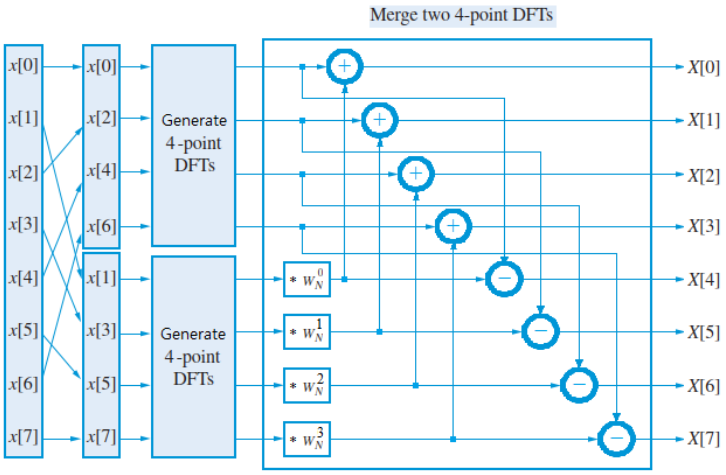
   In fact, twiddle factor is to cut n equal parts on the complex plane unit circle.

   Take advantage of this "rotation" feature. We can use Radix-2 method to separate X[k] into odd and even terms. And we combine twiddle factors to linearly combine them. The time complexity can be reduced from O(n^2) to O(nlogn).

   Of course, cutting the unit circle into more pieces can get the answer faster. However, Radix-2 method has a simpler structure than radix 4 or higher and its performance is good enough.

$$\begin{aligned} X[k] &= X_e[k] + W_N^k \cdot X_o[k] \\ X[k+\frac{N}{2}] &= X_e[k+\frac{N}{2}] + W_N^{k+\frac{N}{2}} \cdot X_o[k+\frac{N}{2}] = X_e[k] - W_N^k \cdot X_o[k] \end{aligned}$$

   Here are the impressions with N=8



2. How you optimize the design and the trade-off you make

   Non radix version:

   Use #pragma HLS PIPELINE II=1 rewind to expand inner loop.

```
void dft(DTYPE real_sample[1024], DTYPE imag_sample[1024],DTYPE real_op[1024],DTYPE imag_op[1024])
{
    //Write your code here
    #pragma HLS INTERFACE mode=s_axilite port=return
    #pragma HLS INTERFACE mode=m_axi bundle=A depth=1024 port=real_sample
    #pragma HLS INTERFACE mode=m_axi bundle=B depth=1024 port=imag_sample
    #pragma HLS INTERFACE mode=m_axi bundle=C depth=1024 port=real_op
    #pragma HLS INTERFACE mode=m_axi bundle=D depth=1024 port=imag_op

    dft_label0:for(int i=0; i<1024; i++){
        int index;
        DTYPE c;
        DTYPE s;
        DTYPE rc;
        DTYPE is;
        DTYPE rs;
        DTYPE ic;

        dft_label1:for(int j=0; j<1024; j++){
            #pragma HLS PIPELINE II=1 rewind
            index = (i*j)&1023;
            c = cos_coefficients_table[index];
            s = sin_coefficients_table[index];
            rc = real_sample[i]*c;
            is = imag_sample[i]*s;
            rs = real_sample[i]*s;
            ic = imag_sample[i]*c;

            real_op[j] += (rc+is);
            imag_op[j] += (rs-ic);
        }
    }
}
```

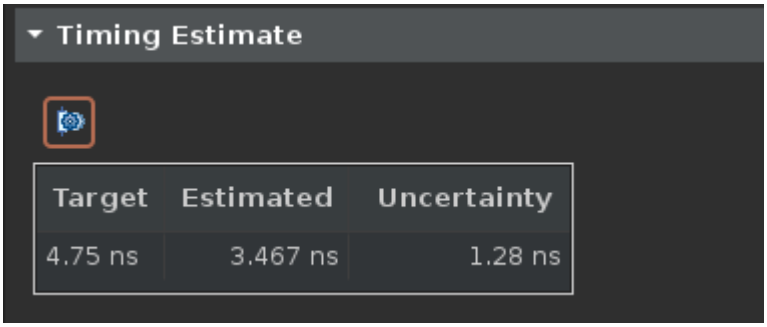
   Try to use radix 2 to accelerate DFT

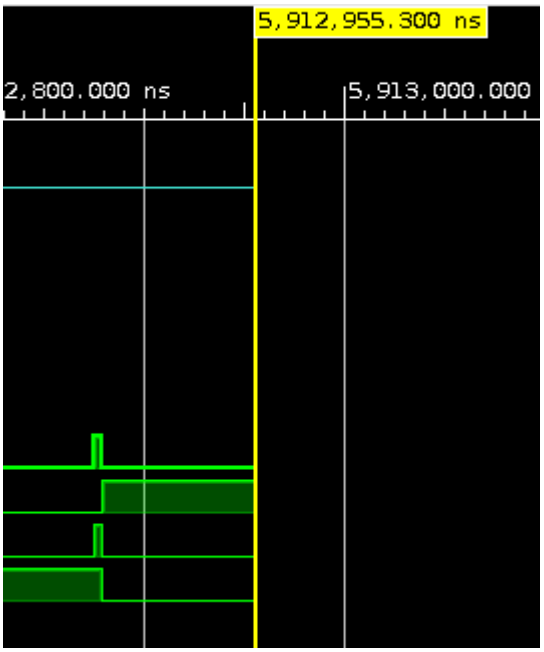
```
dft_label1:for(int k=0; k<512; k++){
    //#pragma HLS PIPELINE rewind
    index1 = (2*m*k)&1023;
    index2 = k;
    c = cos_coefficients_table[index1];
    s = sin_coefficients_table[index1];
    wc = cos_coefficients_table[index2];
    ws = sin_coefficients_table[index2];
    rce = real_sample[2*m]*c;
    ise = imag_sample[2*m]*s;
    rse = real_sample[2*m]*s;
    ice = imag_sample[2*m]*c;

    rco = real_sample[2*m+1]*c;
    iso = imag_sample[2*m+1]*s;
    rso = real_sample[2*m+1]*s;
    ico = imag_sample[2*m+1]*c;


    Ro = (rco+iso);
    Io = (rso-ico);
    Re = (rce+ise);
    Ie = (rse-ice);

    WRo = (Ro*wc-Io*ws);
    WIo = (Ro*ws+Io*wc);

    real_op[k] += (Re + WRo);
    imag_op[k] += (Ie + WIo);
    real_op[k+512] += (Re - WRo);
    imag_op[k+512] += (Ie - WIo);
}
```

   Calculate odd and even parts simultaneously.

3. Explain what problem you encountered and how you solved it
   Using the original method can achieve ii = 1, but after changing to radix 2, the ii-violation encountered.
   I tried to fix that problem and came up with two methods, changing the interface or adding a cycle to collect the required inputs.
   However, the former method is against the rules of the competition, so I tried the latter one.
   But adding cycles for buffer and calculating even and odd at the same time will increase the total operation time. This doesn't seem like a wise choice.
   Additionally, by doing so, the Target T_clock can't be lowered because of the slack problem. So even if we make a version of radix 2, it won't get better performance from the scoring formula.
   Therefore, the two-layer n=1024 for loop with the inner layer pipelined that looks violent but has the best performance.

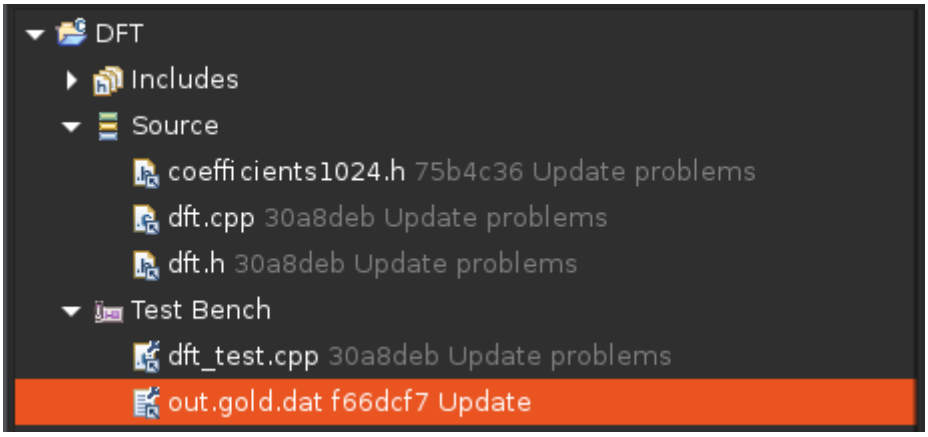
4. Score of each sub-topic (for ranking)





F_max = 288350634 Hz
T_clock = 4.75 ns
tou_simulation = 5912955.3ns
performance = 231.6380632

5. How to run
    1. Download all file from Github link
    2. Open Vitis_hls
    3. Select device xc7z020-clg-400-1
    4. Arrange the files as shown below



    5. Set top function as “dft”
    6. Run C simulation, C synthesis and Co-simulation

T5 LabC Report

CV Canny – Canny Edge Detector

R11922029 吳泓毅

Github link: https://github.com/Anderson-Wu/AAHLS_LABC_Canny

What is Canny algorithm?

Canny algorithm is used to detect edges of an object, we do convert the image to gray scale first, then use gaussian blur to remove noise, and then we apply two convolution filters, Sobel X and Sobel Y to detect vertical and horizontal edges, after getting two results, we merged then by adding square of values on every pixel. At this time, we can get the edge of object, but you can see that the edge is so thick, which may not satisfy the situation we want, so we do non-maximum suppression, which can thin edges. We can determine the direction of edge by getting the arctangent value of Sobel X's value divided by Sobel Y's value, then we compare two neighbor pixels' magnitude according to the direction, if the pixel's magnitude is not the biggest, then this pixel is not an edge, else, we retain it magnitude and send pixel to the final stage. In final stage, we compare the magnitude with two threshold value, upperthreshold and lowerthreshold. If magnitude of pixel is greater or equal to upperthreshold, it means that it is a strong edge, else if it is less than upperthreshold but greater than lowerthreshold, than it is a weak edge. If magnitude is lower than lowerthreshold, then we consider that it is not an edge. After doing operations mentioned above, we can get a thinner and more accuracy edge of object.

$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

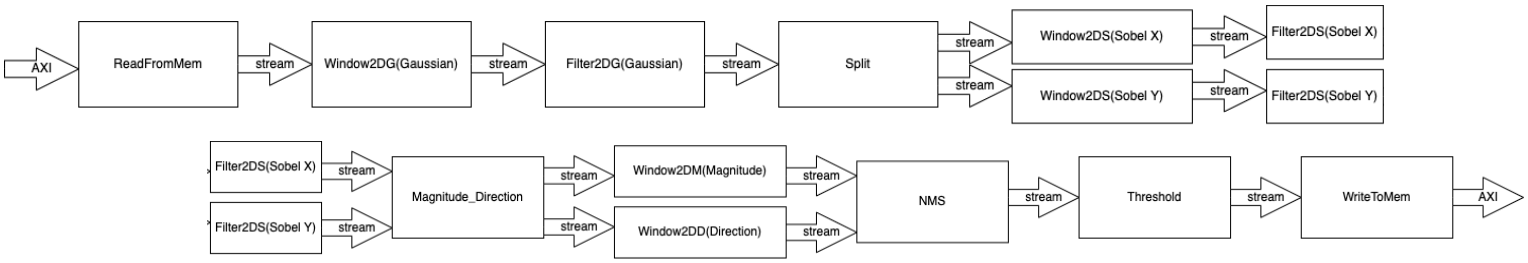
Sobel X

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

Sobel Y

image

Implement Canny with HLS – System Dataflow with code

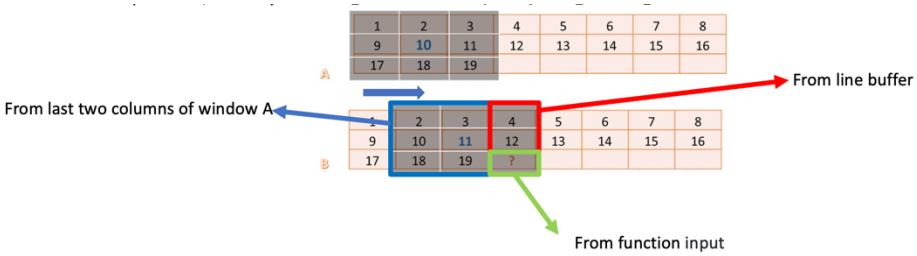



```
void canny(DTYPE* src, DTYPE* dst, int upperThresh, int lowerThresh)
{

#pragma HLS DATAFLOW
#pragma HLS INTERFACE m_axi depth=16384 port=src
#pragma HLS INTERFACE m_axi depth=16384 port=dst
    // Stream of pixels from kernel input to filter, and from filter to output
    hls::stream<Float,2>    coefs_stream;
    hls::stream<DTYPE,64>      pixel_stream;
    hls::stream<window,5>  window_stream; // Set FIFO depth to 0 to minimize resources
    hls::stream<DTYPE,64>     output_stream;
    hls::stream<Float,64>     gaussian_output_stream;
    hls::stream<Float,64>     sobelx_stream;
    hls::stream<Float,64>     sobely_stream;
    hls::stream<windowf,3>  sobelx_window_stream; // Set FIFO depth to 0 to minimize resources
    hls::stream<windowf,3>  sobely_window_stream; // Set FIFO depth to 0 to minimize resources
    hls::stream<DTYPE,64>     sobelx_output_stream;
    hls::stream<DTYPE,64>     sobely_output_stream;
    hls::stream<DTYPE,64>     Magnitude_Direction_output_stream;
    hls::stream<DTYPE,64>     gradient_output_stream;
    hls::stream<window,5>     Magnitude_Direction_window_stream;
    hls::stream<window,5>     gradient_window_stream;
    hls::stream<DTYPE,64>     nms_output_stream;


    int stride=128;
    int width = 128;
    int height = 128;

    Float gaussianCoeffs[3][3]={
            {24,30,24},
            {30,37,30},
            {24,30,24}
    };



    Float sobelxCoeffs[3][3]={
            {-1,0,1},
            {-2,0,2},
            {-1,0,1}
    };

    Float sobelyCoeffs[3][3]={
            {-1,-2,-1},
            {0,0,0},
            {1,2,1}
    };


    ReadFromMem(width, height, stride, src, pixel_stream);

    Window2DG(width, height, pixel_stream, window_stream);


    Filter2DG(width, height, gaussianCoeffs, window_stream, gaussian_output_stream);


    Split(width, height, gaussian_output_stream, sobelx_stream,sobely_stream);

    Window2DS(width, height, sobelx_stream, sobelx_window_stream);

    Filter2DS(width, height, sobelxCoeffs, sobelx_window_stream, sobelx_output_stream);


    Window2DS(width, height, sobely_stream, sobely_window_stream);
    Filter2DS(width, height, sobelyCoeffs, sobely_window_stream, sobely_output_stream);

    Magnitude_Direction(sobelx_output_stream,sobely_output_stream,Magnitude_Direction_output_stream,gradient_output_stream);

    Window2DM(width, height, Magnitude_Direction_output_stream, Magnitude_Direction_window_stream);

    Window2DD(width, height, gradient_output_stream, gradient_window_stream);

    NMS(width, height,Magnitude_Direction_window_stream,gradient_window_stream,nms_output_stream);

    Threshold(upperThresh, lowerThresh,nms_output_stream,output_stream);

    WriteToMem(width, height, stride, output_stream, dst);
}
```

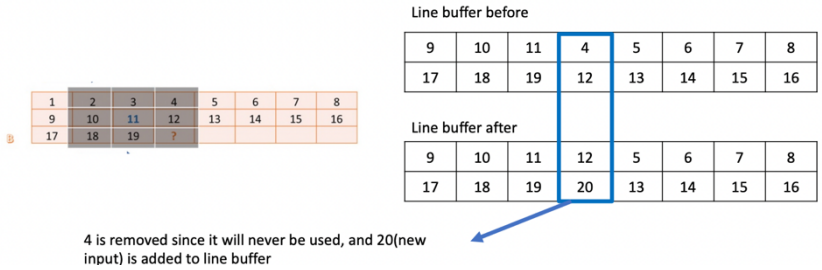
Explain: We use dataflow to connect every function, and using stream as interface to connect each other, inside each function, we use pragma HLS PIPELINE to create task level parallelism too.

Concepts required

Window: Holds 3 x 3 values, when we shift window 1 pixel, two columns will retain for the next operation, and the last column of pixels will come from line buffer and new input from previous function. For a new operation, only need to get one more input by using caching method.



Line buffer: Used to buffer 2 image lines. The total number of pixels held by the line buffer is 2 x WIDTH.



BORDER_DEFAULT: padding method in opencv, if image contains letters "**abcdefgh**" then output will be "**gfedcb|abcdefgh|gfedcba**".

Functions in Dataflow

ReadFromMem: Reads pixel data

Window2DG: Create 3x3 pixel window for gaussian operation

Filter2DG: Do gaussian convolution

Split: Generate two same streams

Window2DS: Create 3x3 pixel window for Sobel operation

Filter2DS: Do Sobel convolution

Magnitude_Direction: Calculate magnitude and direction

Window2DM: Create 3x3 magnitude window for NMS operation

Window2DD: Create 3x3 Direction window for NMS operation

NMS: Do NMS operation

Threshold: Determine strong edge, weak edge and not edge

WriteToMem: Writes output data

Functions' Details

ReadFromMem: Reads pixel data, use for loop and pipeline to do burst transmission. We need to padding like BORDER_DEFAULT in opencv, but the problem may happen which cause II can't be 1. That is, if we get a pixel, it may go to three positions at a same time, which means that we can't do this operation in one cycle due to port limitation of memory. To solve this problem, I use four buffers to store values on left, top, right, and down sides, which can prevent the situation mentioned above, and after finish input, copy the buffer to memory, and also produce four corners' values.

```
void ReadFromMem(DTYPE width,DTYPE height,DTYPE stride,DTYPE *src,hls::stream<DTYPE> &pixel_stream )
{
#pragma HLS INTERFACE m_axi depth=16384 port=src
    DTYPE pad[130*130];
#pragma HLS DEPENDENCE variable=pad inter false
#pragma HLS DEPENDENCE variable=pad intra false
    int n = 0;
    int index = 0;
    int temp;
    int tempup[130];
    int tempdown[130];
    int templeft[130];
    int tempright[130];
    for(int i = 1;i <= height;i++){
        for(int j = 1;j <= width;j++){
#pragma HLS PIPELINE
            pad[i*(130)+j] =src[n];
            if(i == 2){
                tempup[j] =src[n];
            }
            if(j == 2){
                templeft[i] =src[n];
                //pad[i*(128+2)] =src[n];
            }
            if(i == 127){
                tempdown[j] = src[n];
            }
            if(j == 127){
                tempright[i] =src[n];
                //pad[i*(128+2)+129] = src[n];
            }
            n++;
        }
    }
        for(int i = 1;i <=width;i++){
            pad[i] = tempup[i];
            pad[129*130+i] = tempdown[i];
        }

        for(int i = 1;i <=height;i++){
            pad[i*(128+2)] = templeft[i];
            pad[i*(128+2)+129] = tempright[i];
        }
        pad[0] = pad[2*(128+2)+2];

        pad[129] = pad[2*(128+2)+127];

        pad[129*(128+2)] = pad[127*(128+2)+2];


        pad[129*(128+2)+129] =  pad[127*(128+2)+127];


    while(index <16900){
        pixel_stream.write( pad[index]);
        index++;
    }
}
```

Window2DG: Use Window and Line buffer to generate 1 window per cycle, but we don't send window to the next stage because when we encounter borders, we need two cycles to refresh first two columns of window by Line buffer. By setting ARRAY_PARTITION and tell HLS compiler that LineBuffer has no inter and intra loop dependencies, II can reach to 1.

```
void Window2DG(
        DTYPE          width,
        DTYPE        height,
        hls::stream<DTYPE>      &pixel_stream,
        hls::stream<window>  &window_stream)
{
    // Line buffers - used to store [FILTER_V_SIZE-1] entire lines of pixels
    DTYPE LineBuffer[FILTER_V_SIZE-1][MAX_IMAGE_WIDTH+2];
#pragma HLS ARRAY_PARTITION variable=LineBuffer dim=1 complete
#pragma HLS DEPENDENCE variable=LineBuffer inter false
#pragma HLS DEPENDENCE variable=LineBuffer intra false

    // Sliding window of [FILTER_V_SIZE][FILTER_H_SIZE] pixels
    window Window;

    height+=2;
    width+=2;

    unsigned col_ptr = 0;
    unsigned ramp_up = 2*width*((FILTER_V_SIZE-1)/2)+(FILTER_H_SIZE-1)/2+1;
    unsigned num_pixels = width*height;
    unsigned num_iterations = num_pixels + ramp_up;
    num_iterations = 16900;
    int mod1 = 390;
    int mod2 = 391;

    // Iterate until all pixels have been processed
    update_window: for (int n=0; n<num_iterations; n++)
    {
#pragma HLS PIPELINE II=1

        // Read a new pixel from the input stream
        DTYPE new_pixel =pixel_stream.read();
        // Shift the window and add a column of new pixels from the line buffer
        for(int i = 0; i < FILTER_V_SIZE; i++) {
            for(int j = 0; j < FILTER_H_SIZE-1; j++) {
                Window.pix[i][j] = Window.pix[i][j+1];
            }
            Window.pix[i][FILTER_H_SIZE-1] = (i<FILTER_V_SIZE-1) ? LineBuffer[i][col_ptr] : new_pixel;
        }

        // Shift pixels in the column of pixels in the line buffer, add the newest pixel
        for(int i = 0; i < FILTER_V_SIZE-2; i++) {
            LineBuffer[i][col_ptr] = LineBuffer[i+1][col_ptr];
        }
        LineBuffer[FILTER_V_SIZE-2][col_ptr] = new_pixel;

        // Update the line buffer column pointer
        if (col_ptr==(width-1)) {
            col_ptr = 0;
        } else {
            col_ptr++;
        }

        // Write output only when enough pixels have been read the buffers and ramped-up
        if (n>=ramp_up && n != mod1 && n != mod2) {
            window_stream.write(Window);
        }

        if(n == mod1){
            mod1+=130;
        }
        if(n == mod2){
            mod2+=130;
        }

    }
}
```

Filter2DG: This function does convolution by using gaussian filter, I use user-defined type Float which is ap_fixed<43,20> to do floating point operation. If using c++ inbuild float directly, cosim will fail because C++'s float needs too much hardware resources. For doing convolution, we can let II be 1 by using pipeline pragma and doing array partition on gaussian coefficient matrix. But this caused another problem that if we decrease the clock period to 8, time violation would occur since we unroll all the for loop below pipeline. And this function has problem like ReadFromMem mentioned before. Because of BORDER_DEFAULT padding rule, we need to create four buffers for storing left, up, right, down sides of values. We need to partition coeffs matrix so that we can do parallel multiplication.

```
void Filter2DG(
        DTYPE       width,
        DTYPE       height,
        Float   coeffs[][3],
        hls::stream<window> &window_stream,
        hls::stream<Float>      &pixel_stream )
{
#pragma HLS ARRAY_PARTITION variable=coeffs complete dim=0
    height +=2;
    width+=2;
    Float temp[130*130];
    int index=0;
    int writeindex=0;
    Float tempup[130];
    Float tempdown[130];
    Float templeft[130];
    Float tempright[130];
    apply_filter: for (int y = 1; y < height-1; y++)
    {
        for (int x = 1; x < width-1; x++)
        {
#pragma HLS PIPELINE II=1
            // Read a 2D window of pixels
            window w = window_stream.read();
            // Apply filter to the 2D window
            Float sum = 0;
            for(int row=0; row<FILTER_V_SIZE; row++)
            {
                for(int col=0; col<FILTER_H_SIZE; col++)
                {
                    Float pixel;
                    int xoffset = (x+col-(FILTER_H_SIZE/2));
                    int yoffset = (y+row-(FILTER_V_SIZE/2));
                    // Deal with boundary conditions : clamp pixels to 0 when outside of image

                    pixel = w.pix[row][col];

                    sum += pixel*coeffs[row][col];
                }
            }
            sum = sum/253;
            //pixel_stream.write(sum);
            temp[y*(128+2)+x] =sum;
$|
            if(y == 2){
                tempup[x] =sum;
            }

            if(x == 2){
                templeft[y] =sum;
            }

            if(y == 127){
                tempdown[x] =sum;
            }

            if(x == 127){
                tempright[y] = sum;
            }

        }
    }

        for(int i = 1;i <=width;i++){
            temp[i] = tempup[i];
            temp[129*130+i] = tempdown[i];
        }

        for(int i = 1;i <=height;i++){
            temp[i*(128+2)] = templeft[i];
            temp[i*(128+2)+129] = tempright[i];
        }

        temp[0] = temp[2*(128+2)+2];

        temp[129] =  temp[2*(128+2)+127];

        temp[129*(128+2)] =  temp[127*(128+2)+2];

        temp[129*(128+2)+129] =   temp[127*(128+2)+127];

    while(index < 16900){
        pixel_stream.write(temp[index]);
        index++;
    }
}
```

Split: After gaussian operation, the result will be used to do Sobel X and Sobel Y edge detection. But stream has one producer one consumer constraint, so we need to produce two streams with same value.

```
void Split(
        DTYPE       width,
        DTYPE       height,
        hls::stream<Float> &input_stream,
        hls::stream<Float>     &sobelx_stream,
        hls::stream<Float>     &sobely_stream){

    int numiterations = 130*130;
    for(int n = 0;n < numiterations;n++){
#pragma HLS PIPELINE II=1
        Float new_pixel = input_stream.read();
        sobelx_stream.write(new_pixel);
        sobely_stream.write(new_pixel);
    }

}
```

Window2DS: Do almost same thing with Window2DG mentioned before, generate window by using Window and Line buffer. This hardware has two Window2DS, one for Sobel X, one for Sobel Y.

```
void Window2DS(
        DTYPE          width,
        DTYPE        height,
        hls::stream<Float>      &pixel_stream,
        hls::stream<windowf>  &window_stream)
{
    // Line buffers - used to store [FILTER_V_SIZE-1] entire lines of pixels
    Float LineBuffer[FILTER_V_SIZE-1][MAX_IMAGE_WIDTH+2];
#pragma HLS ARRAY_PARTITION variable=LineBuffer dim=1 complete
#pragma HLS DEPENDENCE variable=LineBuffer inter false
#pragma HLS DEPENDENCE variable=LineBuffer intra false

    // Sliding window of [FILTER_V_SIZE][FILTER_H_SIZE] pixels
    windowf Window;

    height+=2;
    width+=2;

    unsigned col_ptr = 0;
    unsigned ramp_up = 2*width*((FILTER_V_SIZE-1)/2)+(FILTER_H_SIZE-1)/2+1;
    unsigned num_pixels = width*height;
    unsigned num_iterations = num_pixels + ramp_up;
    num_iterations = 16900;
    int mod1 = 390;
    int mod2 = 391;
    // Iterate until all pixels have been processed
    update_window: for (int n=0; n<num_iterations; n++)
    {

#pragma HLS PIPELINE II=1

        // Read a new pixel from the input stream
        Float new_pixel =  pixel_stream.read();

        // Shift the window and add a column of new pixels from the line buffer
        for(int i = 0; i < FILTER_V_SIZE; i++) {
            for(int j = 0; j < FILTER_H_SIZE-1; j++) {
                Window.pix[i][j] = Window.pix[i][j+1];
            }
            Window.pix[i][FILTER_H_SIZE-1] = (i<FILTER_V_SIZE-1) ? LineBuffer[i][col_ptr] : new_pixel;
        }


        // Shift pixels in the column of pixels in the line buffer, add the newest pixel
        for(int i = 0; i < FILTER_V_SIZE-2; i++) {
            LineBuffer[i][col_ptr] = LineBuffer[i+1][col_ptr];
        }
        LineBuffer[FILTER_V_SIZE-2][col_ptr] = new_pixel;

        // Update the line buffer column pointer
        if (col_ptr==(width-1)) {
            col_ptr = 0;
        } else {
            col_ptr++;
        }

        // Write output only when enough pixels have been read the buffers and ramped-up
        if (n>=ramp_up && n!= mod1 && n!=mod2) {
            window_stream.write(Window);


        }

        if(n == mod1){
            mod1+=130;
        }
        if(n == mod2){
            mod2+=130;
        }

    }
}
```

Filter2DS: Do almost same things with Filter2DG. This hardware has two Filter2DS, one for Sobel X, one for Sobel Y. Sobel X is used to detect vertical edges and Sobel Y is used to detect horizontal edges.

```
void Filter2DS(
        DTYPE       width,
        DTYPE       height,
        Float   coeffs[][3],
        hls::stream<windowf> &window_stream,
        hls::stream<DTYPE>     &pixel_stream )
{

#pragma HLS ARRAY_PARTITION variable=coeffs complete dim=0


    // Process the incoming stream of pixel windows

    height +=2;
    width+=2;
    Float temp[130*130];
    Float shift=0.5;
    int index=0;
    int writeindex=0;
    int start = 0;
    apply_filter: for (int y = 1; y < height-1; y++)
    {
        start = y*130;
        for (int x = 1; x < width-1; x++)
        {
#pragma HLS PIPELINE II=1
            // Read a 2D window of pixels
            windowf w = window_stream.read();

            // Apply filter to the 2D window
            Float sum = 0;
            for(int row=0; row<FILTER_V_SIZE; row++)
            {
                for(int col=0; col<FILTER_H_SIZE; col++)
                {
                    Float pixel;
                    int xoffset = (x+col-(FILTER_H_SIZE/2));
                    int yoffset = (y+row-(FILTER_V_SIZE/2));

                    pixel = w.pix[row][col];

                    sum += pixel*coeffs[row][col];
                }
            }

            temp[start+x] =sum;

        }

    }
    while(index < 16900){
        if(temp[index] >=0){
            temp[index]+=shift;
        }
        else{
            temp[index]-=shift;
        }
        pixel_stream.write(DTYPE(temp[index]));
        index++;
    }
}
```

Magnitude_Direction: This function gets two input streams from Sobel X and Sobel Y. In this function, get the magnitude by adding square of two inputs and get the direction according to arctangent formula. Due to speed and hardware resources, if using atan2() function provided by C++, cosim will fail. So I use user-defined float type and using multiply operation rather than divide operation on gradient of Y and X to determine the direction.

```
void Magnitude_Direction(
        hls::stream<DTYPE> &sobelx_output_stream,
        hls::stream<DTYPE>      &sobely_output_stream,
        hls::stream<DTYPE>      &Magnitude_Direction_output_stream,
        hls::stream<DTYPE>      &gradient_output_stream){

    int numiterations = 130*130;
    for(int n = 0;n < numiterations;n++){
#pragma HLS PIPELINE II=1
        DTYPE x_grad = sobelx_output_stream.read();
        DTYPE y_grad = sobely_output_stream.read();
        DTYPE res = x_grad*x_grad + y_grad*y_grad;
        Magnitude_Direction_output_stream.write(res);
        Float coeff225 = 6.625;
        Float coeff675 = 38.625;
        DTYPE grad;
        if (x_grad >= 0 && y_grad >=0){
            Float gx225 = coeff225*x_grad;
            Float gx675 = coeff675*x_grad;
            y_grad = y_grad*16;

            if (y_grad >gx225 && y_grad <= gx675){
                grad = 0;
            }
            else if (y_grad <= gx225){
                grad = 3;
            }
            else if(y_grad > gx675){
                grad = 1;
            }
        }

        if (x_grad <= 0 && y_grad <=0){
            x_grad = -x_grad;
            y_grad = -y_grad;
            Float gx225 = coeff225*x_grad;
            Float gx675 = coeff675*x_grad;
            y_grad = y_grad*16;

            if (y_grad >gx225 && y_grad <= gx675){
                grad = 0;
            }
            else if (y_grad <= gx225){
                grad = 3;
            }
            else if(y_grad > gx675){
                grad = 1;
            }
        }


        if (x_grad <= 0 && y_grad >=0){
            x_grad = -x_grad;
            Float gx225 = coeff225*x_grad;
            Float gx675 = coeff675*x_grad;
            y_grad = y_grad*16;

            if (y_grad >gx225 && y_grad <= gx675){
                grad = 2;
            }
            else if (y_grad <= gx225){
                grad = 3;
            }
            else if(y_grad > gx675){
                grad = 1;
            }
        }

        if (x_grad >= 0 && y_grad <=0){
            y_grad = -y_grad;
            Float gx225 = 6.625*x_grad;
            Float gx675 = 38.625*x_grad;
            y_grad = y_grad*16;

            if (y_grad >gx225 && y_grad <= gx675){
                grad = 2;
            }
            else if (y_grad <= gx225){
                grad = 3;
            }
            else if(y_grad > gx675){
                grad = 1;
            }
        }
        gradient_output_stream.write(grad);
    }

}
```

Window2DM, Window2DD: Both of functions create window for function NMS

```
void Window2DM(
        DTYPE          width,
        DTYPE         height,
        hls::stream<DTYPE>      &pixel_stream,
        hls::stream<window>  &window_stream)
{
    // Line buffers - used to store [FILTER_V_SIZE-1] entire lines of pixels
    DTYPE LineBuffer[FILTER_V_SIZE-1][MAX_IMAGE_WIDTH+2];
#pragma HLS ARRAY_PARTITION variable=LineBuffer dim=1 complete
#pragma HLS DEPENDENCE variable=LineBuffer inter false
#pragma HLS DEPENDENCE variable=LineBuffer intra false

    // Sliding window of [FILTER_V_SIZE][FILTER_H_SIZE] pixels
    window Window;

    height+=2;
    width+=2;

    unsigned col_ptr = 0;
    unsigned ramp_up = 2*width*((FILTER_V_SIZE-1)/2)+(FILTER_H_SIZE-1)/2+1;
    unsigned num_pixels = width*height;
    unsigned num_iterations = num_pixels + ramp_up;
    num_iterations = 16900;
    int mod1 = 390;
    int mod2 = 391;
    // Iterate until all pixels have been processed
    update_window: for (int n=0; n<num_iterations; n++)
    {

#pragma HLS PIPELINE II=1

        // Read a new pixel from the input stream
        DTYPE new_pixel =pixel_stream.read();
        // Shift the window and add a column of new pixels from the line buffer
        for(int i = 0; i < FILTER_V_SIZE; i++) {
            for(int j = 0; j < FILTER_H_SIZE-1; j++) {
                Window.pix[i][j] = Window.pix[i][j+1];
            }
            Window.pix[i][FILTER_H_SIZE-1] = (i<FILTER_V_SIZE-1) ? LineBuffer[i][col_ptr] : new_pixel;
        }


        // Shift pixels in the column of pixels in the line buffer, add the newest pixel
        for(int i = 0; i < FILTER_V_SIZE-2; i++) {
            LineBuffer[i][col_ptr] = LineBuffer[i+1][col_ptr];
        }
        LineBuffer[FILTER_V_SIZE-2][col_ptr] = new_pixel;

        // Update the line buffer column pointer
        if (col_ptr==(width-1)) {
            col_ptr = 0;
        } else {
            col_ptr++;
        }

        // Write output only when enough pixels have been read the buffers and ramped-up
        if (n>=ramp_up && n!= mod1 && n!=mod2) {
            window_stream.write(Window);
        }

        if(n == mod1){
            mod1+=130;
        }
        if(n == mod2){
            mod2+=130;
        }



    }
}
```

```
void Window2DD(
        DTYPE          width,
        DTYPE         height,
        hls::stream<DTYPE>      &pixel_stream,
        hls::stream<window>  &window_stream)
{
    // Line buffers - used to store [FILTER_V_SIZE-1] entire lines of pixels
   DTYPE LineBuffer[FILTER_V_SIZE-1][MAX_IMAGE_WIDTH+2];
#pragma HLS ARRAY_PARTITION variable=LineBuffer dim=1 complete
#pragma HLS DEPENDENCE variable=LineBuffer inter false
#pragma HLS DEPENDENCE variable=LineBuffer intra false

    // Sliding window of [FILTER_V_SIZE][FILTER_H_SIZE] pixels
    window Window;

    height+=2;
    width+=2;

    unsigned col_ptr = 0;
    unsigned ramp_up = 2*width*((FILTER_V_SIZE-1)/2)+(FILTER_H_SIZE-1)/2+1;
    unsigned num_pixels = width*height;
    unsigned num_iterations = num_pixels + ramp_up;
    num_iterations = 16900;
    int mod1 = 390;
    int mod2 = 391;
    // Iterate until all pixels have been processed
    update_window: for (int n=0; n<num_iterations; n++)
    {

#pragma HLS PIPELINE II=1

        // Read a new pixel from the input stream
        DTYPE new_pixel =pixel_stream.read();

        // Shift the window and add a column of new pixels from the line buffer
        for(int i = 0; i < FILTER_V_SIZE; i++) {
            for(int j = 0; j < FILTER_H_SIZE-1; j++) {
                Window.pix[i][j] = Window.pix[i][j+1];
            }
            Window.pix[i][FILTER_H_SIZE-1] = (i<FILTER_V_SIZE-1) ? LineBuffer[i][col_ptr] : new_pixel;
        }


        // Shift pixels in the column of pixels in the line buffer, add the newest pixel
        for(int i = 0; i < FILTER_V_SIZE-2; i++) {
            LineBuffer[i][col_ptr] = LineBuffer[i+1][col_ptr];
        }
        LineBuffer[FILTER_V_SIZE-2][col_ptr] = new_pixel;

        // Update the line buffer column pointer
        if (col_ptr==(width-1)) {
            col_ptr = 0;
        } else {
            col_ptr++;
        }

        // Write output only when enough pixels have been read the buffers and ramped-up
        if (n>=ramp_up && n!= mod1 && n!= mod2) {
            window_stream.write(Window);
        }

        if(n == mod1){
            mod1+=130;
        }
        if(n == mod2){
            mod2+=130;
        }



    }
}
```

NMS: Get two windows from Window2DM and Window2DG, compare the pixel's magnitude with its neighbors determined by its direction. It can be noticed that we only need one new pixel to generate new result, which is same with filter operation.

```
void NMS(
        DTYPE       width,
        DTYPE       height,
        hls::stream<window> &Magnitude_Direction_window_stream,
        hls::stream<window> &gradient_window_stream,
        hls::stream<DTYPE> &nms_output_stream
        )
{



    height +=2;
    width+=2;
    Float temp[130*130];
    int index=0;
    int g1,g2;
    int res;
    apply_filter: for (int y = 1; y < height-1; y++)
    {
        for (int x = 1; x < width-1; x++)
        {
            #pragma HLS PIPELINE II=1
            // Read a 2D window of pixels
            window w = Magnitude_Direction_window_stream.read();
            window g = gradient_window_stream.read();

            if (g.pix[1][1]==2){
            g1 = w.pix[0][2];
            g2 = w.pix[2][0];
            }

            else if (g.pix[1][1]==1){
                g1 = w.pix[0][1];
                g2 = w.pix[2][1];
            }

            else if (g.pix[1][1]==0){
                g1 = w.pix[0][0];
                g2 = w.pix[2][2];
            }

            else{
                g1 = w.pix[1][0];
                g2 = w.pix[1][2];
            }
            if(w.pix[1][1]  < g1 || w.pix[1][1]  < g2){
                res = 0;
            }
            else{
                res = w.pix[1][1];
            }
            nms_output_stream.write(res);
        }
    }
}
```

Threshold: Determined whether the pixel is strong edge, weak edge or not edge.

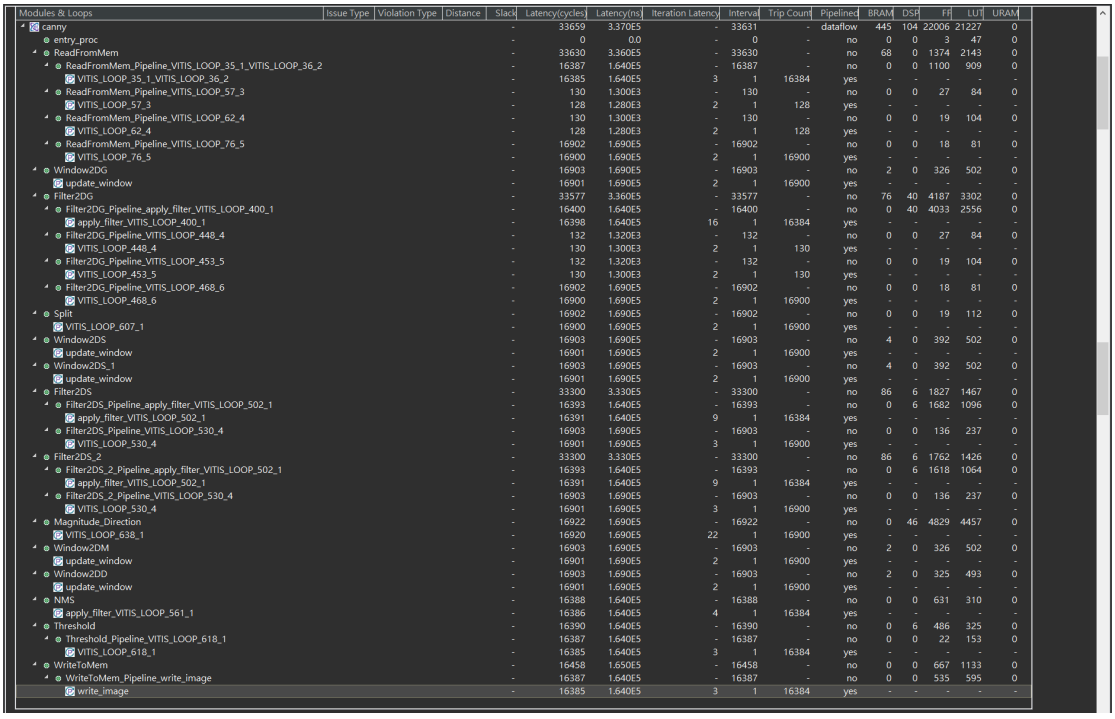
```
void  Threshold(int upperThresh, int lowerThresh ,hls::stream<DTYPE> &nms_output_stream,hls::stream<DTYPE> &threshold_output_stream){
    for(int i = 0;i < 16384;i++){
#pragma HLS PIPELINE II=1
        DTYPE val = nms_output_stream.read();
        if (val >= upperThresh*upperThresh)
            threshold_output_stream.write(255);
        else if (val > lowerThresh*lowerThresh )
            threshold_output_stream.write(127);
        else{
            threshold_output_stream.write(0);
        }
    }
}
```

WriteToMem: Send result to host

```
void WriteToMem(
        DTYPE       width,
        DTYPE       height,
        DTYPE       stride,
        hls::stream<DTYPE>     &pixel_stream,
        DTYPE       *dst)
{
#pragma HLS INTERFACE m_axi depth=16384 port=dst

    write_image: for (int n = 0; n < 16384; n++) {
#pragma HLS PIPELINE II=1
        DTYPE pix = pixel_stream.read();
        dst[n] = pix;
    }
}
```

Synthesis Summary

According to the report, we can have II be 1 on every function, so all functions are always ready to get data



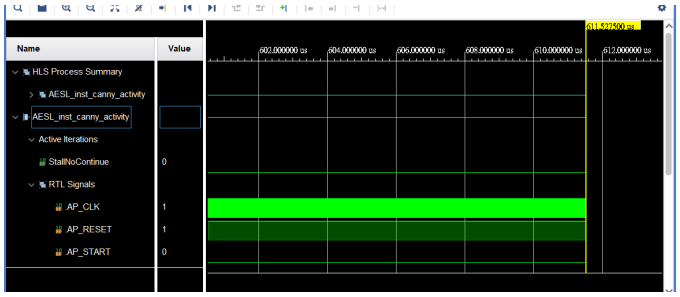
Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
canny				-	33659	3.370E5	-	33631	-	dataflow	445	104	22006	21227	0
entry_proc				-	0	0.0	-	0	-	no	0	0	3	47	0
ReadFromMem				-	33630	3.360E5	-	33630	-	no	68	0	1374	2143	0
ReadFromMem_Pipeline_VITIS_LOOP_35_1_VITIS_LOOP_36_2				-	16387	1.640E5	-	16387	-	no	0	0	1100	909	0
VITIS_LOOP_35_1_VITIS_LOOP_36_2				-	16385	1.640E5	3	1	16384	yes	-	-	-	-	-
ReadFromMem_Pipeline_VITIS_LOOP_57_3				-	130	1.300E3	-	130	-	no	0	0	27	84	0
VITIS_LOOP_57_3				-	128	1.280E3	2	1	128	yes	-	-	-	-	-
ReadFromMem_Pipeline_VITIS_LOOP_62_4				-	130	1.300E3	-	130	-	no	0	0	19	104	0
VITIS_LOOP_62_4				-	128	1.280E3	2	1	128	yes	-	-	-	-	-
ReadFromMem_Pipeline_VITIS_LOOP_76_5				-	16902	1.690E5	-	16902	-	no	0	0	18	81	0
VITIS_LOOP_76_5				-	16900	1.690E5	2	1	16900	yes	-	-	-	-	-
Window2DG				-	16903	1.690E5	-	16903	-	no	2	0	326	502	0
update_window				-	16901	1.690E5	2	1	16900	yes	-	-	-	-	-
Filter2DG				-	33577	3.360E5	-	33577	-	no	76	40	4187	3302	0
Filter2DG_Pipeline_apply_filter_VITIS_LOOP_400_1				-	16400	1.640E5	-	16400	-	no	0	40	4033	2556	0
apply_filter_VITIS_LOOP_400_1				-	16398	1.640E5	16	1	16384	yes	-	-	-	-	-
Filter2DG_Pipeline_VITIS_LOOP_448_4				-	132	1.320E3	-	132	-	no	0	0	27	84	0
VITIS_LOOP_448_4				-	130	1.300E3	2	1	130	yes	-	-	-	-	-
Filter2DG_Pipeline_VITIS_LOOP_453_5				-	132	1.320E3	-	132	-	no	0	0	19	104	0
VITIS_LOOP_453_5				-	130	1.300E3	2	1	130	yes	-	-	-	-	-
Filter2DG_Pipeline_VITIS_LOOP_468_6				-	16902	1.690E5	-	16902	-	no	0	0	18	81	0
VITIS_LOOP_468_6				-	16900	1.690E5	2	1	16900	yes	-	-	-	-	-
Split				-	16902	1.690E5	-	16902	-	no	0	0	19	112	0
VITIS_LOOP_607_1				-	16900	1.690E5	2	1	16900	yes	-	-	-	-	-
Window2DS				-	16903	1.690E5	-	16903	-	no	4	0	392	502	0
update_window				-	16901	1.690E5	2	1	16900	yes	-	-	-	-	-
Window2DS_1				-	16903	1.690E5	-	16903	-	no	4	0	392	502	0
update_window				-	16901	1.690E5	2	1	16900	yes	-	-	-	-	-
Filter2DS				-	33300	3.330E5	-	33300	-	no	86	6	1827	1467	0
Filter2DS_Pipeline_apply_filter_VITIS_LOOP_502_1				-	16393	1.640E5	-	16393	-	no	0	6	1682	1096	0
apply_filter_VITIS_LOOP_502_1				-	16391	1.640E5	9	1	16384	yes	-	-	-	-	-
Filter2DS_Pipeline_VITIS_LOOP_530_4				-	16903	1.690E5	-	16903	-	no	0	0	136	237	0
VITIS_LOOP_530_4				-	16901	1.690E5	3	1	16900	yes	-	-	-	-	-
Filter2DS_2				-	33300	3.330E5	-	33300	-	no	86	6	1762	1426	0
Filter2DS_2_Pipeline_apply_filter_VITIS_LOOP_502_1				-	16393	1.640E5	-	16393	-	no	0	6	1618	1064	0
apply_filter_VITIS_LOOP_502_1				-	16391	1.640E5	9	1	16384	yes	-	-	-	-	-
Filter2DS_2_Pipeline_VITIS_LOOP_530_4				-	16903	1.690E5	-	16903	-	no	0	0	136	237	0
VITIS_LOOP_530_4				-	16901	1.690E5	3	1	16900	yes	-	-	-	-	-
Magnitude_Direction				-	16922	1.690E5	-	16922	-	no	0	46	4829	4457	0
VITIS_LOOP_638_1				-	16920	1.690E5	22	1	16900	yes	-	-	-	-	-
Window2DM				-	16903	1.690E5	-	16903	-	no	2	0	326	502	0
update_window				-	16901	1.690E5	2	1	16900	yes	-	-	-	-	-
Window2DD				-	16903	1.690E5	-	16903	-	no	2	0	325	493	0
update_window				-	16901	1.690E5	2	1	16900	yes	-	-	-	-	-
NMS				-	16388	1.640E5	-	16388	-	no	0	0	631	310	0
apply_filter_VITIS_LOOP_561_1				-	16386	1.640E5	4	1	16384	yes	-	-	-	-	-
Threshold				-	16390	1.640E5	-	16390	-	no	0	6	486	325	0
Threshold_Pipeline_VITIS_LOOP_618_1				-	16387	1.640E5	-	16387	-	no	0	0	22	153	0
VITIS_LOOP_618_1				-	16385	1.640E5	3	1	16384	yes	-	-	-	-	-
WriteToMem				-	16458	1.650E5	-	16458	-	no	0	0	667	1133	0
WriteToMem_Pipeline_write_image				-	16387	1.640E5	-	16387	-	no	0	0	535	595	0
write_image				-	16385	1.640E5	3	1	16384	yes	-	-	-	-	-

Score

Clock period: 9ns

Score = $\frac{9.00ns}{6.570ns} \times \frac{1}{611.527500us}$ = 2240.06772

Target	Estimated	Uncertainty
9.00 ns	6.570 ns	2.43 ns



How to reproduce the result?

1. Clone repository from github link https://github.com/Anderson-Wu/AAHLS_LABC_Canny
2. Open Vitis HLS 2022.1 Command Prompt
3. Go to file directory
4. Type command vitis_hls -f tcl_script.tcl and then C-sim, synthesis, Co-sim will run
5. Type command vitis_hls -p canny and then GUI will show on screen
6. Open the synthesis report file and waveform to check the result

T5 LabC Report

Graph Betweenness

R11943022 范詠為

Github link: https://github.com/BrianEE07/course-lab_C_betweenness.git

1. Brief introduction to the overall system

   The system performs the algorithm finding Betweenness Centrality of a graph stored in Compressed Sparse Row (CSR) format. The betweenness centrality of a node v is given by the expression below, which sigma_st is the total number of shortest paths from node s to t and sigma_st(v) is the number of those paths that pass through.

$$g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

   The host side first transfers data to HBM on PL side, and the kernel uses the AXI-Master with burst transfer as input interface. After kernel finish execution, the result is transferred back to host.

2. Algorithm & Implementation

Algorithm 1: Betweenness centrality in unweighted graphs

```
C_B[v] ← 0, v ∈ V;
for s ∈ V do
    S ← empty stack;
    P[w] ← empty list, w ∈ V;
    σ[t] ← 0, t ∈ V;   σ[s] ← 1;
    d[t] ← −1, t ∈ V;   d[s] ← 0;
    Q ← empty queue;
    enqueue s → Q;
    while Q not empty do
        dequeue v ← Q;
        push v → S;
        foreach neighbor w of v do
            // w found for the first time?
            if d[w] < 0 then
                enqueue w → Q;
                d[w] ← d[v] + 1;
            end
            // shortest path to w via v?
            if d[w] = d[v] + 1 then
                σ[w] ← σ[w] + σ[v];
                append v → P[w];
            end
        end
    end
    δ[v] ← 0, v ∈ V;
    // S returns vertices in order of non-increasing distance from s
    while S not empty do
        pop w ← S;
        for v ∈ P[w] do δ[v] ← δ[v] + (σ[v]/σ[w]) · (1 + δ[w]);
        if w ≠ s then C_B[w] ← C_B[w] + δ[w];
    end
end
```

The above picture is the algorithm for finding betweenness centrality. First, we perform breadth-first search (BFS) on the unweighted graph and update the sigma value for the next step. Then we calculate the delta from sigma value and update the final betweenness value.

Although the baseline C++ code is provided, its use of the standard template library (STL) is prohibited in HLS due to the dynamic memory allocation, therefore, the provided code can only pass c-sim (sw_emu). To run the co-sim (hw_emu, hw), we have to modify the code and remove origin STL container like stack, queue, list and vector. Furthermore, to get the information of min/max latency and II, we set the MaxNumVert(3600) and MaxNumEgde(42000) for the iteration upperbound of for-loop and conditional exit the loop.

Original baseline code:

```
for (int i = 0; i < numVert; i++) {
  btwn[i] = 0;
}
for (int i = 0; i < numVert; i++) {
  std::stack<unsigned> s;
  std::vector<std::list<unsigned>> p(numVert);
  std::vector<float> sigma(numVert);
  std::vector<int> dist(numVert);
  std::queue<unsigned> q;
  unsigned source = i;

  for (int j = 0; j < numVert; j++) {
    sigma[j] = 0;
    dist[j] = -1;
  }
  sigma[source] = 1;
  dist[source] = 0;

  q.push(source);
  while (!q.empty()) {
    unsigned v = q.front();
    s.push(v);
    for (int j = offset[v]; j < offset[v + 1]; j++) {
      unsigned w = column[j];
      if (dist[w] < 0) {
        q.push(w);
        dist[w] = dist[v] + 1;
      }
      if (dist[w] == dist[v] + 1) {
        sigma[w] = sigma[w] + sigma[v];
        p[w].push_back(v);
      }
    }
    q.pop();
  }

  std::vector<float> delta(numVert);
  for (int j = 0; j < numVert; j++) {
    delta[j] = 0;
  }
  while (!s.empty()) {
    unsigned w = s.top();
    if (source != w) {
      btwn[w] = btwn[w] + delta[w];
    }
    for (std::list<unsigned>::iterator it = p[w].begin(); it != p[w].end(); it++) {
      unsigned v = *it;
      delta[v] = delta[v] + (sigma[v] / sigma[w]) * (1 + delta[w]);
    }
    s.pop();
  }
}
```

Revised baseline code (only main part):

```
// graph traversal (BFS) for shortest path
LOOP_MAIN_BFS:
for (int j = 0;j < MaxNumVert;++j){
    if (q_tail != q_head) {
        unsigned v = q[q_head++];
        s[s_top++] = v;
        for (int k = offset[v];k < offset[v + 1];++k) {
            unsigned w = column[k];
            if (dist[w] < 0) {
                dist[w] = dist[v] + 1;
                q[q_tail++] = w;
            }
            if (dist[w] == dist[v] + 1) {
                sigma[w] += sigma[v];
                p[w][p_tail[w]++] = v;
            }
        }
    }
    else break;
}
// dependency accumulation
LOOP_MAIN_ACC:
for (int j = 0;j < MaxNumVert;++j){
    if (s_top != 0) {
        unsigned w = s[--s_top];
        if (source != w) {
            btwn[w] += delta[w];
        }
        for (int k = 0;k < p_tail[w];++k) {
            unsigned v = p[w][k];
            delta[v] += (sigma[v] / sigma[w]) * (1 + delta[w]);
        }
    }
    else break;
}
```

3. Result & Optimization

a. Baseline

i. Execution time

(us)	Large	Small
SW	1053036	183514
HW	19463302	6213506

ii. Synthesis report

```
+ Detail:
    * Instance:
    +----------------------------------------+-----------------------------+---------+---------+----------+----------+-----+------+---------+
    |                                        |                             |  Latency (cycles) |  Latency (absolute)  |  Interval  | Pipeline|
    |                Instance                |            Module           |   min   |   max   |    min   |    max   | min |  max |   Type  |
    +----------------------------------------+-----------------------------+---------+---------+----------+----------+-----+------+---------+
    |grp_dut_Pipeline_LOOP_INIT_fu_430       |dut_Pipeline_LOOP_INIT       |       40|     3639|  0.133 us|  12.129 us|   40|  3639|       no|
    |grp_dut_Pipeline_LOOP_MAIN_INIT_fu_440  |dut_Pipeline_LOOP_MAIN_INIT  |        3|     3602|  9.999 ns|  12.005 us|    3|  3602|       no|
    |grp_dut_Pipeline_VITIS_LOOP_186_1_fu_449|dut_Pipeline_VITIS_LOOP_186_1|        ?|        ?|         ?|         ?|    ?|     ?|       no|
    |grp_dut_Pipeline_VITIS_LOOP_212_2_fu_467|dut_Pipeline_VITIS_LOOP_212_2|        ?|        ?|         ?|         ?|    ?|     ?|       no|
    +----------------------------------------+-----------------------------+---------+---------+----------+----------+-----+------+---------+

    * Loop:
    +------------------+---------+---------+----------+-----------+-----------+----------+----------+
    |                  |  Latency (cycles) | Iteration|  Initiation Interval  |   Trip   |          |
    |     Loop Name    |   min   |   max   |  Latency |  achieved |   target  |   Count  | Pipelined|
    +------------------+---------+---------+----------+-----------+-----------+----------+----------+
    |- LOOP_MAIN       |        ?|        ?|         ?|          -|          -|  1 ~ 3600|        no|
    | + LOOP_MAIN_BFS  |        ?|        ?|         ?|          -|          -|  1 ~ 3600|        no|
    | + LOOP_MAIN_ACC  |        ?|        ?|         ?|          -|          -|  1 ~ 3600|        no|
    +------------------+---------+---------+----------+-----------+-----------+----------+----------+
```

```
================================================================
== Utilization Estimates
================================================================
* Summary:
+---------------------+---------+------+---------+--------+-----+
|         Name        | BRAM_18K|  DSP |    FF   |   LUT  | URAM|
+---------------------+---------+------+---------+--------+-----+
|DSP                  |        -|     -|        -|       -|    -|
|Expression           |        -|     -|        0|     564|    -|
|FIFO                 |        -|     -|        -|       -|    -|
|Instance             |        0|     5|     4502|    6102|    0|
|Memory               |        8|     -|        0|       0|    6|
|Multiplexer          |        -|     -|        -|    1728|    -|
|Register             |        -|     -|     1236|       -|    -|
+---------------------+---------+------+---------+--------+-----+
|Total                |        8|     5|     5738|    8394|    6|
+---------------------+---------+------+---------+--------+-----+
|Available SLR        |     1344|  2976|   871680|  435840|  320|
+---------------------+---------+------+---------+--------+-----+
|Utilization SLR (%)  |       ~0|    ~0|       ~0|       1|    1|
+---------------------+---------+------+---------+--------+-----+
|Available            |     2688|  5952|  1743360|  871680|  640|
+---------------------+---------+------+---------+--------+-----+
|Utilization (%)      |       ~0|    ~0|       ~0|      ~0|   ~0|
+---------------------+---------+------+---------+--------+-----+
```

b. Optimization

i. Methods

1. Since the baseline code frequently read and write data (offset, column, btwn) from HBM, which has long cycle delay, we add three buffers (off, col, btwn_buff) in the function to first collect data (off, col) from HBM in the beginning and dump to buffer (btwn_buff) when updating betweenness and finally write to HBM at the end. These buffers are mapped to BRAMs in FPGA so can reduce the latency to access data. This significantly improve the performance to 50% execution time reduction.
2. With the added directives like
   a. #pragma HLS PIPELINE II=1 rewind
   b. #pragma HLS UNROLL factor=16 and #pragma HLS ARRAY_PARTITION factor=8 (to unroll the BRAMs initialization)

These can slightly improve the performance.

ii. Execution time

(us)	Large	Small
SW	1204974	104463
HW	7705132	786250

We can observe that HW has 60% improvement to baseline but is still slower than SW execution.

iii. Synthesis report

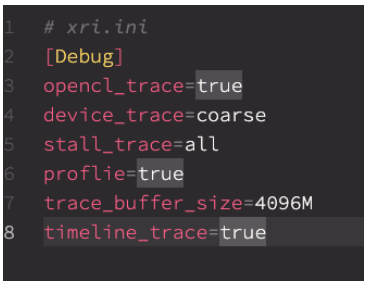
```
+ Detail:
    * Instance:
    N/A

    * Loop:
    +----------------------+---------+---------+----------+-----------+-----------+-----------+----------+
    |                      |  Latency (cycles) | Iteration|  Initiation Interval  |    Trip   |          |
    |       Loop Name      |   min   |   max   |  Latency |  achieved |   target  |   Count   | Pipelined|
    +----------------------+---------+---------+----------+-----------+-----------+-----------+----------+
    |- LOOP_BUFF_OFFSET    |       40|     3639|        41|          1|          1|  1 ~ 3600|       yes|
    |- LOOP_BUFF_COLUMN    |       40|    42039|        41|          1|          1|  1 ~ 42000|       yes|
    |- LOOP_INIT           |        1|     3600|         2|          1|          1|  1 ~ 3600|       yes|
    |- LOOP_MAIN           |        ?|        ?|         ?|          -|          -|  1 ~ 3600|        no|
    | + LOOP_MAIN_INIT     |        1|      225|         1|          1|          1|   1 ~ 225|       yes|
    | + LOOP_MAIN_BFS      |        ?|        ?|         ?|          -|          -|  1 ~ 3600|        no|
    |  ++ VITIS_LOOP_177_1 |        ?|        ?|        11|          6|          1|         ?|       yes|
    | + LOOP_MAIN_ACC      |        7|        ?|  7 ~ ?   |          -|          -|  1 ~ 3600|        no|
    |  ++ VITIS_LOOP_201_2 |       20|        ?|        21|         13|          1|     1 ~ ?|       yes|
    |- LOOP_WRITE          |       38|     3637|        39|          1|          1|  1 ~ 3600|       yes|
    +----------------------+---------+---------+----------+-----------+-----------+-----------+----------+
```

```
================================================================
== Utilization Estimates
================================================================
* Summary:
+---------------------+---------+------+---------+--------+-----+
|         Name        | BRAM_18K|  DSP |    FF   |   LUT  | URAM|
+---------------------+---------+------+---------+--------+-----+
|DSP                  |        -|     -|        -|       -|    -|
|Expression           |        -|     -|        0|    1996|    -|
|FIFO                 |        -|     -|        -|       -|    -|
|Instance             |        0|     5|     3555|    5694|    0|
|Memory               |       56|     -|        0|       0|    4|
|Multiplexer          |        -|     -|        -|    2776|    -|
|Register             |        -|     -|     2862|     256|    -|
+---------------------+---------+------+---------+--------+-----+
|Total                |       56|     5|     6417|   10722|    4|
+---------------------+---------+------+---------+--------+-----+
|Available SLR        |     1344|  2976|   871680|  435840|  320|
+---------------------+---------+------+---------+--------+-----+
|Utilization SLR (%)  |        4|    ~0|       ~0|       2|    1|
+---------------------+---------+------+---------+--------+-----+
|Available            |     2688|  5952|  1743360|  871680|  640|
+---------------------+---------+------+---------+--------+-----+
|Utilization (%)      |        2|    ~0|       ~0|       1|   ~0|
+---------------------+---------+------+---------+--------+-----+
```

We can observe that since we use BRAM to buffer data, the area is larger than baseline to reach shorter latency.

4. The Encountered Problems (thanks to T4 group for help)
   a. Since the graph is large, the execution of hw_emu cannot finish in a reasonable time, so we generate a tiny graph data to perform hw_emu to get the synthesis report and execution result (require much less time than perform hw).
   b. Originally, there is no xrt.run_summary dumped out after execution, so we modify the Makefile as below picture and add the xri.ini to root directory.

```
99
100  # get global setting
101  ifeq ($(HOST_ARCH), x86)
102  CXXFLAGS += -fmessage-length=0 -I$(CUR_DIR)/src/ -I$(XILINX_XRT)/include -I$(XILINX_HLS)/include -std=c++14 -O3 -Wall -Wno-unknown-pragmas -Wno-unused-label
103  LDFLAGS += -pthread -L$(XILINX_XRT)/lib -L$(XILINX_HLS)/lnx64/tools/fpo_v7_0  -Wl,--as-needed -lOpenCL -lxrt_coreutil -lgmp -lmpfr -lIp_floating_point_v7_0_bitacc_cmodel
104  VPP_FLAGS += -t $(TARGET) --platform $(XPLATFORM) --profile.data all:all:all --profile.stall all:all:all  --profile.exec all:all --profile.trace_memory HBM[31] --save-temps
105  VPP_LDFLAGS += --optimize 2 -R 2
106  else ifeq ($(HOST_ARCH), aarch64)
107  ifeq ($(PS_ON_X86), true)
```



```
# xri.ini
[Debug]
opencl_trace=true
device_trace=coarse
stall_trace=all
proflie=true
trace_buffer_size=4096M
timeline_trace=true
```

5. Final Score

   cntbig = 3534

   cntsmall = 1015

   time (large) = 7705132 us

   time (small) = 786250 us

   score = 0.6*(cntbig^4/7705132)+0.4*(cntsmall^4/786250) = 12686076.24

6. How to run? (p.s. maybe need chmod 777 -R hw_exec/ first if permission denied)

   a. Github link: https://github.com/BrianEE07/course-lab_C_betweenness.git

   b. Large

   cd betweenness/

   ./hw_exec/host.exe -xclbin ./hw_exec/dut.xclbin -o ./data/large-csr-offset.mtx -i ./data/large-csr-indicesweights.mtx | tee large_case.log

   c. Small

   cd betweenness/

   ./hw_exec/host.exe -xclbin ./hw_exec/dut.xclbin -o ./data/small-csr-offset.mtx -i ./data/small-csr-indicesweights.mtx | tee small_case.log