HLS Lab B: FFT

D10943004 林亮昕

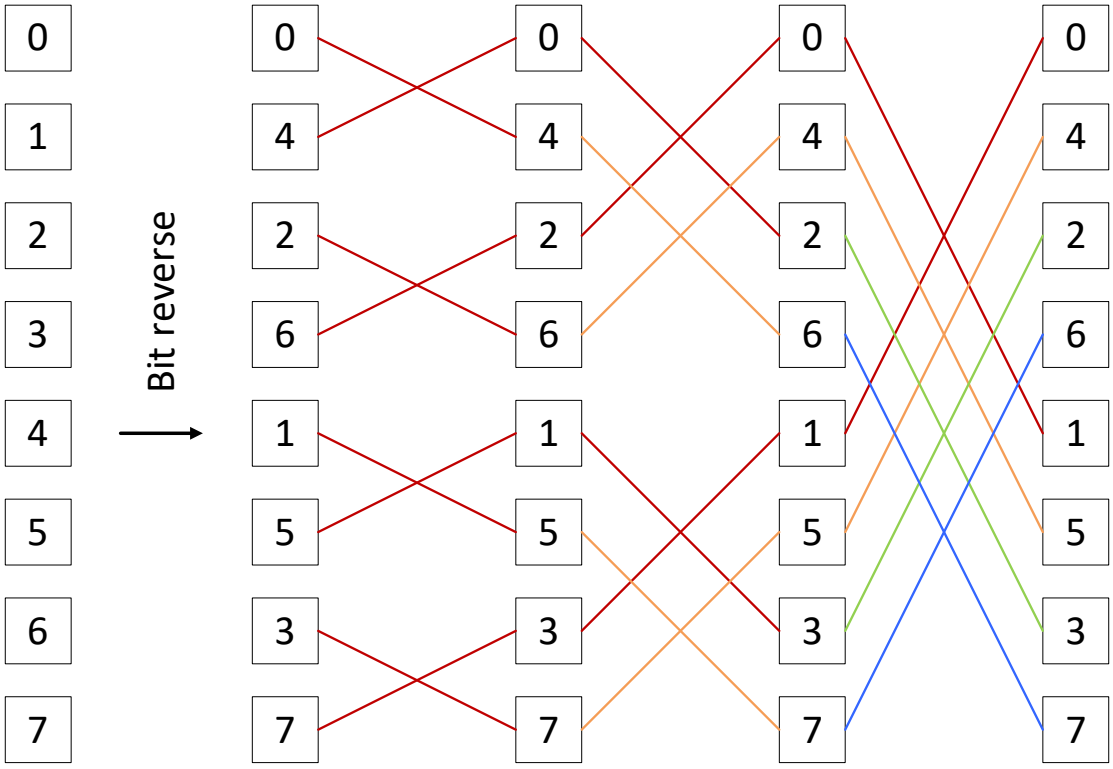
Github: konosuba-lin/HLS

- Baseline FFT Code

  FFT 主要可以分成三個 loop 完成，由外而內分別是: stages loop、butterfly loop、跟 point loop。Baseline 的 c-code 如下:

```
stages:for(stage=1; stage<= M; stage++)
{
    DFTpts = 1 << stage;        // DFT = 2^stage = po
    numBF = DFTpts/2;           // Butterfly WIDTHS i
    k=0;

    e = -6.283185307178/DFTpts;

    a = 0.0;
    // Perform butterflies for j-th stage
    butterfly:for(j=0; j<numBF; j++)
    {

        c = cos(a);
        s = sin(a);
        a = a + e;

        // Compute butterflies that use same W**k
        DFTpts:for(i=j; i<SIZE; i += DFTpts)
        {

            i_lower = i + numBF;          //index o
            temp_R = X_R[i_lower]*c- X_I[i_lower]*s;
            temp_I = X_I[i_lower]*c+ X_R[i_lower]*s;

            X_R[i_lower] = X_R[i] - temp_R;
            X_I[i_lower] = X_I[i] - temp_I;
            X_R[i] = X_R[i] + temp_R;
            X_I[i] = X_I[i] + temp_I;
        }
        k+=step;
    }
    step=step/2;
}
```

  其流程圖(以 8-points FFT 為例)為下



其中橫軸的方向對應到的 stages loop，而不同顏色的 butterfly 結構代表使用不一樣的 twiddle factor。

- FFT Optimization

  1. Create Perfect Loop

  首先為了讓 butterfly loop、跟 point loop 能更有效率地進行，我們將 stages loop 展開，並將 butterfly loop 與 point loop 寫成 perfect loop 的形式

```
void fft(DTYPE X_R[SIZE], DTYPE X_I[SIZE], DTYPE OUT_R[SIZE], DTYPE OUT_I[SIZE])
{
#pragma HLS INTERFACE mode=ap_ctrl_none port=return
#pragma HLS INTERFACE mode=s_axilite port=OUT_R
#pragma HLS INTERFACE mode=s_axilite port=OUT_I
#pragma HLS INTERFACE mode=s_axilite port=X_I
#pragma HLS INTERFACE mode=s_axilite port=X_R

#pragma HLS DATAFLOW

    DTYPE Stage_R0[SIZE],  Stage_I0[SIZE];
    DTYPE Stage_R1[SIZE],  Stage_I1[SIZE];
    DTYPE Stage_R2[SIZE],  Stage_I2[SIZE];
    DTYPE Stage_R3[SIZE],  Stage_I3[SIZE];
    DTYPE Stage_R4[SIZE],  Stage_I4[SIZE];
    DTYPE Stage_R5[SIZE],  Stage_I5[SIZE];
    DTYPE Stage_R6[SIZE],  Stage_I6[SIZE];
    DTYPE Stage_R7[SIZE],  Stage_I7[SIZE];
    DTYPE Stage_R8[SIZE],  Stage_I8[SIZE];
    DTYPE Stage_R9[SIZE],  Stage_I9[SIZE];
    bit_reverse(X_R, X_I, Stage_R0, Stage_I0);
    fft_first(Stage_R0, Stage_I0,Stage_R1, Stage_I1);
    fft_stages(2,Stage_R1, Stage_I1, Stage_R2, Stage_I2);
    fft_stages(3,Stage_R2, Stage_I2, Stage_R3, Stage_I3);
    fft_stages(4,Stage_R3, Stage_I3, Stage_R4, Stage_I4);
    fft_stages(5,Stage_R4, Stage_I4, Stage_R5, Stage_I5);
    fft_stages(6,Stage_R5, Stage_I5, Stage_R6, Stage_I6);
    fft_stages(7,Stage_R6, Stage_I6, Stage_R7, Stage_I7);
    fft_stages(8,Stage_R7, Stage_I7, Stage_R8, Stage_I8);
    fft_stages(9,Stage_R8, Stage_I8, Stage_R9, Stage_I9);
    fft_last(Stage_R9, Stage_I9,OUT_R,OUT_I);

}
```

```
void fft_stages(int stage, DTYPE X_R[SIZE], DTYPE X_I[SIZE], DTYPE OUT_R[SIZE], DTYPE OUT_I[SIZE])
{
    //Write a code that computes any arbitary stages of the FFT
    //int DFTpts;
    //int numBF;
    int i,j;
    int i_lower;
    DTYPE temp_R;       //temporary storage complex variable
    DTYPE temp_I;       //temporary storage complex variable

    int t,step_t;
    int idx_higher, idx_lower;
    DTYPE sin_t,cos_t;
    step_t = (SIZE >> stage);
    t = 0;
    int n1,n2;
    n1 = (1 << (stage-1)); // 2^(stage-1) = (2^stage)/2
    n2 = (SIZE >> stage); // SIZE/2^stage
    butterfly:for(j=0; j<n1; j++)
    {
        DFTpts:for(i=0; i<n2; i ++)
        {
#pragma HLS PIPELINE
            sin_t = SIN[j*step_t];
            cos_t = COS[j*step_t];
            idx_higher = j + 2*i*n1;
            idx_lower = idx_higher + n1;
            temp_R = X_R[idx_lower]*cos_t- X_I[idx_lower]*sin_t;
            temp_I = X_I[idx_lower]*cos_t+ X_R[idx_lower]*sin_t;
            OUT_R[idx_lower] = X_R[idx_higher] - temp_R;
            OUT_I[idx_lower] = X_I[idx_higher] - temp_I;
            OUT_R[idx_higher] = X_R[idx_higher] + temp_R;
            OUT_I[idx_higher] = X_I[idx_higher] + temp_I;
        }
    }
}
```

2. Lookup Table

   為了減少 cos 與 sin 的計算量，我們利用 python 生成 lookup table 對 sin 和 cos 進行查表

```
const DTYPE SIN[] = {
        -0.000000,-0.006136,-0.012272,-0.018407,-0.024541,-0.030675,-0.036807,-0.042938,-0.049068,-0.055195,
        -0.061321,-0.067444,-0.073565,-0.079682,-0.085797,-0.091909,-0.098017,-0.104122,-0.110222,-0.116319,
        -0.122411,-0.128498,-0.134581,-0.140658,-0.146730,-0.152797,-0.158858,-0.164913,-0.170962,-0.177004,
        -0.183040,-0.189069,-0.195090,-0.201105,-0.207111,-0.213110,-0.219101,-0.225084,-0.231058,-0.237024,
        -0.242980,-0.248928,-0.254866,-0.260794,-0.266713,-0.272621,-0.278520,-0.284408,-0.290285,-0.296151,
```

```
const DTYPE COS[] = {
         1.000000, 0.999981, 0.999925, 0.999831, 0.999699, 0.999529, 0.999322, 0.999078, 0.998795, 0.998476,
         0.998118, 0.997723, 0.997290, 0.996820, 0.996313, 0.995767, 0.995185, 0.994565, 0.993907, 0.993212,
         0.992480, 0.991710, 0.990903, 0.990058, 0.989177, 0.988258, 0.987301, 0.986308, 0.985278, 0.984210,
         0.983105, 0.981964, 0.980785, 0.979570, 0.978317, 0.977028, 0.975702, 0.974339, 0.972940, 0.971504,
```

3. Dataflow Pragma 與 Pipeline Pragma

   為了增加 FFT 的運算效率，我們首先利用 Pipeline Pragma 使得每個 stages 的運算時間差不多為 530 個 cycle，再利用 Dataflow Pragma 進一步增加運算效率。

   此外我們發現如果 Pipeline Pragma 是放在 butterfly loops 而非 points loop 的話每個 stages 的運算時間會隨著 stages 增加而變長，這是因為放在 butterfly loops 的話會使得 points loop 被 unroll，這時反而會因為資源不夠而使得時間變長。

- HLS Results

  我們可以發現 HLS 後的結果與預期相同，總共產生十個 stages 的 function

```
+ Detail:
    * Instance:
    +----------------+-------------+---------+---------+-----------+-----------+------+------+---------+
    |                |             | Latency (cycles) |   Latency (absolute)  |   Interval  | Pipeline|
    |    Instance    |    Module   |   min   |   max   |    min    |    max    |  min |  max |   Type  |
    +----------------+-------------+---------+---------+-----------+-----------+------+------+---------+
    |bit_reverse_U0  |bit_reverse  |     1026|     1026| 10.260 us | 10.260 us |  1026|  1026|       no|
    |fft_first_U0    |fft_first    |      520|      520|  5.200 us |  5.200 us|   520|   520|       no|
    |fft_stages_1_U0 |fft_stages_1 |      530|      530|  5.300 us |  5.300 us|   530|   530|       no|
    |fft_stages_2_U0 |fft_stages_2 |      530|      530|  5.300 us |  5.300 us|   530|   530|       no|
    |fft_stages_3_U0 |fft_stages_3 |      530|      530|  5.300 us |  5.300 us|   530|   530|       no|
    |fft_stages_4_U0 |fft_stages_4 |      530|      530|  5.300 us |  5.300 us|   530|   530|       no|
    |fft_stages_5_U0 |fft_stages_5 |      530|      530|  5.300 us |  5.300 us|   530|   530|       no|
    |fft_stages_6_U0 |fft_stages_6 |      530|      530|  5.300 us |  5.300 us|   530|   530|       no|
    |fft_stages_7_U0 |fft_stages_7 |      530|      530|  5.300 us |  5.300 us|   530|   530|       no|
    |fft_stages_U0   |fft_stages   |      530|      530|  5.300 us |  5.300 us|   530|   530|       no|
    |fft_last_U0     |fft_last     |     1042|     1042| 10.420 us | 10.420 us|  1042|  1042|       no|
    +----------------+-------------+---------+---------+-----------+-----------+------+------+---------+
```

而 Pragma Dataflow 對時間的幫助卻有限，

```
+---------+---------+-----------+-----------+------+------+----------+
|  Latency (cycles) |   Latency (absolute)  |   Interval  | Pipeline |
|   min   |   max   |    min    |    max    |  min |  max |   Type   |
+---------+---------+-----------+-----------+------+------+----------+
|     8902|     8902| 89.020 us | 89.020 us |  2075|  2075| dataflow |
+---------+---------+-----------+-----------+------+------+----------+
```

因為每個 stages 之間的讀取順序並非規則，如果需要增加 Dataflow 的表現得化需要利用 complete array partition，但合成時間極長，目前仍無成功。

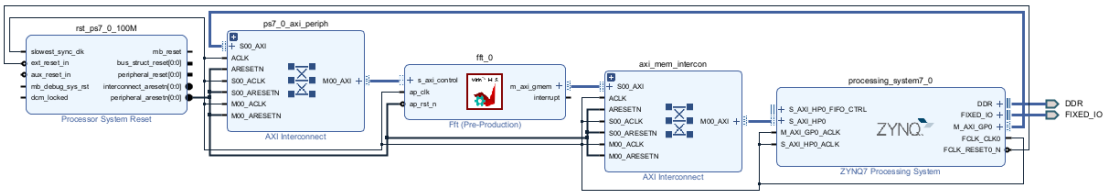
同時我們發現在 input 與 output 相接的兩個階段時間最長，這是因為 AXI interface 的緣故導致。

+ Detail:
    * Instance:

Instance	Module	Latency (cycles) min	Latency (cycles) max	Latency (absolute) min	Latency (absolute) max	Interval min	Interval max	Pipeline Type
entry_proc_U0	entry_proc	0	0	0 ns	0 ns	0	0	no
bit_reverse_U0	bit_reverse	2058	2058	20.580 us	20.580 us	2058	2058	no
fft_first_U0	fft_first	520	520	5.200 us	5.200 us	520	520	no
fft_stages_1_U0	fft_stages_1	530	530	5.300 us	5.300 us	530	530	no
fft_stages_2_U0	fft_stages_2	530	530	5.300 us	5.300 us	530	530	no
fft_stages_3_U0	fft_stages_3	530	530	5.300 us	5.300 us	530	530	no
fft_stages_4_U0	fft_stages_4	530	530	5.300 us	5.300 us	530	530	no
fft_stages_5_U0	fft_stages_5	530	530	5.300 us	5.300 us	530	530	no
fft_stages_6_U0	fft_stages_6	530	530	5.300 us	5.300 us	530	530	no
fft_stages_7_U0	fft_stages_7	530	530	5.300 us	5.300 us	530	530	no
fft_stages_U0	fft_stages	530	530	5.300 us	5.300 us	530	530	no
fft_last_U0	fft_last	2074	2074	20.740 us	20.740 us	2074	2074	no

- System Architecture

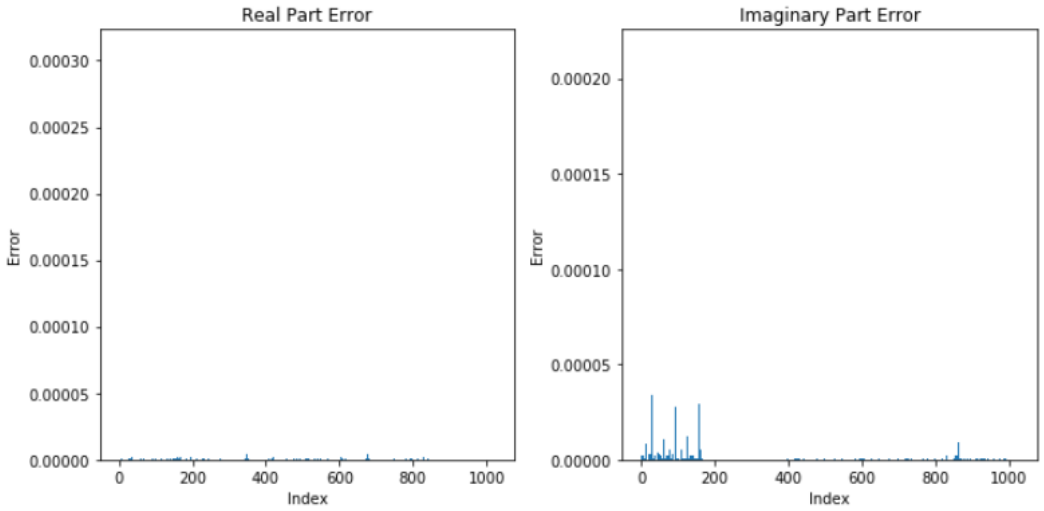
  我們仿照 Lab2 的方式利用 AXI 介面將 HLS 的 IP 接上 ZYNQ7

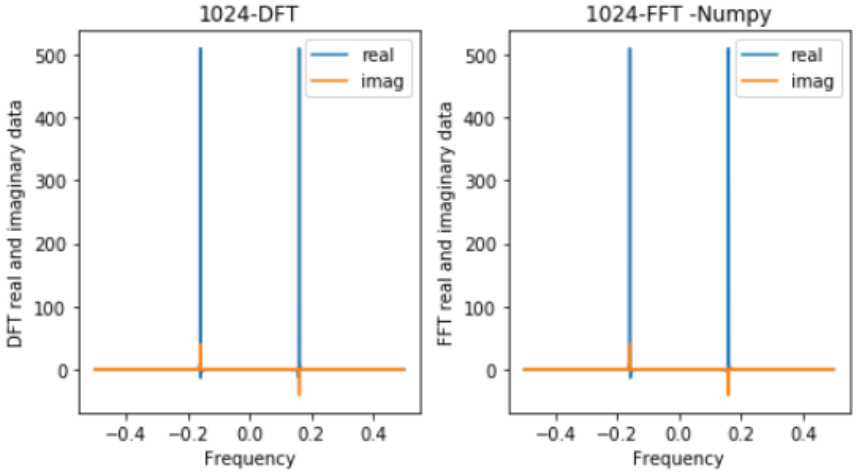


- PYNQ Verification:

  我們將結果放到 PYNQ 上做測試，並得到正確結果

```
Real Part RMSE:  1.7840247029189356e-05 Imaginary Part RMSE: 1.3904950782005702e-05
PASS
```




1024-DFT
real
imag
DFT real and imaginary data
500
400
300
200
100
0
−0.4
−0.2
0.0
0.2
0.4
Frequency
1024-FFT -Numpy
real
imag
FFT real and imaginary data
500
400
300
200
100
0
−0.4
−0.2
0.0
0.2
0.4
Frequency