

LabB-FFT

Github link: https://github.com/aahwu/AAHLS_labB

Overall system:

本次 lab 實作了 FFT 演算法，相較 DFT 演算法利用了對稱性來加速運算。一開始提供了整個演算法跑在 software(CPU)上的版本，但若需要充分的使用硬體加速，就必須改寫其架構。

Project 設定:

Device: xc7z020-clg400-1

Period: 10ns

1. Bit reverse

由於 FFT 使用了 butterfly 的結構，可以對一開始輸入的資料重新排列，使得之後的 FFT 中，資料能夠 in-place 做計算。

下圖是 8bit FFT 的 bit reverse 範例:

Input Decimal Address	Input Binary Address	Reversed Binary Address	Reversed Decimal Address
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

於是此 function 可以分為兩部分:1. Address 的計算，2. 資料的移動

1. Address 的計算:

Source code 如下圖所示:

```

unsigned int reverse(unsigned int input, unsigned int size){
    int result = 0;
    for (int i = 0; i<size; i++) {
        result = (result << 1) | (input & 1);
        input = input >> 1;
    }
    return result;
}

```

當中的計算皆使用 logical operator 做運算來優化硬體資源。

2. 資料的移動

Source code 如下圖所示:

```

void bit_reverse(DTYPE X_R[SIZE], DTYPE X_I[SIZE]){
    //Write your code here.
    unsigned int dataWidth = LOG2_CEIL<SIZE>::val;
    ap_uint<32> m;
    ap_uint<SIZE> record = 0;
    Reverse: for(int i = 0; i<SIZE; i++) {
        if (record[i]==0) {
            ap_uint<32> temp = i;
            m = reverse(temp, dataWidth);
            record[m] = 1;

            DTYPE X_R_temp = X_R[i];
            DTYPE X_I_temp = X_I[i];
            X_R[i] = X_R[m];
            X_I[i] = X_I[m];
            X_R[m] = X_R_temp;
            X_I[m] = X_I_temp;
        }
    }
}

```

其中有兩個 data dependency 的部分。一是同個 array 在相同的 cycle 中需要寫兩次(WAW)，二是同個 array 在不同的 cycle 可能有 RAW。但根據實際上演算法的規律，這些 dependency 都可以忽略，於是我們可以下一些

pragma:

```

#pragma HLS DEPENDENCE variable=X_R intra WAW false
#pragma HLS DEPENDENCE variable=X_R inter RAW false
#pragma HLS DEPENDENCE variable=X_I intra WAW false
#pragma HLS DEPENDENCE variable=X_I inter RAW false

```

但最終的 II 仍然無法降到 1，原因是 load operation 需要 2 個 cycle，而又因為需同時讀取要互換的資料，用完了 2 個 port，故無法有效地做 pipeline。而因為 reverse 後的 address 是 random access，故也無法使用 array partition

來增加頻寬。

2. Software implementation

FFT 的核心演算法的 source code 如下圖所示:

```
stage_loop:
    for (stage = 1; stage <= M; stage++) { // Do M stages of butterflies
        DFTpts = 1 << stage;
        numBF = DFTpts / 2;
        k = 0;
        e = -6.283185307178 / DFTpts;
        a = 0.0;
        // Perform butterflies for j-th stage
        butterfly_loop:
            for (j = 0; j < numBF; j++) {
                c = cos(a);
                s = sin(a);
                a = a + e;
                // Compute butterflies that use same W**k
                dft_loop:
                    for (i = j; i < SIZE; i += DFTpts) {
                        i_lower = i + numBF; // index of lower point in butterfly
                        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;
    }
}
```

在這個架構下，因為最裡層的兩個 loop 皆有 variable bond，故無法有效地做優化。cos 和 sin 的計算可以使用 LUT 來優化。

3. Hardware acceleration

接下來，stage 再分為三個部分: first stage, last stage, other stages，另外 stage 和 stage 之間都使用了 buffer。其中 first stage 和 last stage 因為參數都固定了，可以很有效地做優化，且可以幫助我們了解在 general 的 case 下 stage 怎麼運作的。

下圖是 first stage 和 last stage 的 source code:

```

//stage 1
void fft_stage_first(DTYPE X_R[SIZE], DTYPE X_I[SIZE], DTYPE OUT_R[SIZE], DTYPE OUT_I[SIZE])
{
    DTYPE temp_R;      /*temporary storage complex variable*/
    DTYPE temp_I;      /*temporary storage complex variable*/

    int i;              /* loop indexes */
    int i_lower;        /* Index of lower point in butterfly */

    /*=====BEGIN: FFT=====*/
    DTYPE c2, s2;
    c2 = W_real[0];
    s2 = W_imag[0];
#pragma HLS DEPENDENCE variable=OUT_I intra WAW false
#pragma HLS DEPENDENCE variable=OUT_R intra WAW false
    DFTpts:for(i=0; i<SIZE; i += 2)
    {
        i_lower = i + 1;          //index of lower point in butterfly
        temp_R = X_R[i_lower]*c2- X_I[i_lower]*s2;
        temp_I = X_I[i_lower]*c2+ X_R[i_lower]*s2;

        OUT_R[i_lower] = X_R[i] - temp_R;
        OUT_I[i_lower] = X_I[i] - temp_I;
        OUT_R[i] = X_R[i] + temp_R;
        OUT_I[i] = X_I[i] + temp_I;
    }
}

void fft_stage_last(DTYPE X_R[SIZE], DTYPE X_I[SIZE], DTYPE OUT_R[SIZE], DTYPE OUT_I[SIZE])
{
    DTYPE temp_R;      /*temporary storage complex variable*/
    DTYPE temp_I;      /*temporary storage complex variable*/

    int i,j;           /* loop indexes */

    // Do M stages of butterflies
    DTYPE c2, s2;

    j = 512;
#pragma HLS DEPENDENCE variable=OUT_I intra WAW false
#pragma HLS DEPENDENCE variable=OUT_R intra WAW false
    butterfly:for(i=0; i<512; i++)
    {
        c2 = W_real[i];
        s2 = W_imag[i];

        temp_R = X_R[j]*c2- X_I[j]*s2;
        temp_I = X_I[j]*c2+ X_R[j]*s2;

        OUT_R[j] = X_R[i] - temp_R;
        OUT_I[j] = X_I[i] - temp_I;
        OUT_R[i] = X_R[i] + temp_R;
        OUT_I[i] = X_I[i] + temp_I;

        j += 1;
    }
}

```

因為有了 buffer，所以沒有了 RAW 的問題，且因為這兩個部分 memory access 是規律了，故可以使用 unroll 和 array partition 來進一步增加 throughput。

再來是其餘的 stage。為了讓 tool 做優化，可以將內層的兩個 loop 合併。因為實際上的 operation 數量是固定的，便能夠做 pipeline。

下圖為 source code:

```
void fft_stages(DTYPE X_R[SIZE], DTYPE X_I[SIZE], int stage, DTYPE OUT_R[SIZE], DTYPE OUT_I[SIZE])
{
    DTYPE temp_R;      /*temporary storage complex variable*/
    DTYPE temp_I;      /*temporary storage complex variable*/

    int i,j;           /* loop indexes */

    int DFTpts;
    int numBF;         /*Butterfly Width*/

    DTYPE c2, s2;
    ap_uint<32> e2, ec, one;
    DFTpts = 1 << stage;    // DFT = 2^stage = points in sub DFT
    numBF = DFTpts/2;       // Butterfly WIDTHS in sub-DFT
    one = numBF - 1;
    ec = SIZE >> stage;

#pragma HLS DEPENDENCE variable=OUT_I intra WAW false
#pragma HLS DEPENDENCE variable=OUT_R intra WAW false
#pragma HLS DEPENDENCE variable=OUT_I inter WAW false
#pragma HLS DEPENDENCE variable=OUT_R inter WAW false
    DFTpts:for(i = 0; i<SIZE; i++) {
        ap_uint<32> index = i;
        ap_uint<32> index_low = index;
        index_low[stage-1] = 1;
        j = 0 | index.range(stage-2, 0);
        c2 = W_real[j*ec];
        s2 = W_imag[j*ec];
        temp_R = X_R[index_low]*c2- X_I[index_low]*s2;
        temp_I = X_I[index_low]*c2+ X_R[index_low]*s2;

        OUT_R[index_low] = X_R[index] - temp_R;
        OUT_I[index_low] = X_I[index] - temp_I;
        OUT_R[index] = X_R[index] + temp_R;
        OUT_I[index] = X_I[index] + temp_I;
        if (j==one) {
            i += numBF;
        }
    }
}
```

若一開始 bit reverse 的部分可以不需要 in-place 而可以也另外用 buffer，則可以進一步優化 code 如下：

```

void bit_reverse(DTYPE X_R[SIZE], DTYPE X_I[SIZE], DTYPE Y_R[SIZE], DTYPE Y_I[SIZE]){
    //Insert your code here

    unsigned int dataWidth = LOG2_CEIL<SIZE>::val;
    ap_uint<32> m;
    ap_uint<SIZE> record = 0;
    Reverse: for(int i = 0; i<SIZE; i++) {
        ap_uint<32> temp = i;
        m = reverse(temp, dataWidth);

        Y_R[i] = X_R[m];
        Y_I[i] = X_I[m];
    }
}

```

這樣便將 II 降為 1 且能夠另外做 unroll。

4. Comparison

1. Software resource & timing

=====						
== Utilization Estimates						
=====						
* Summary:						
+	+	+	+	+	+	+
	Name	BRAM_18K	DSP	FF	LUT	URAM
+	+	+	+	+	+	+
	DSP	-	-	-	-	-
	Expression	-	-	0	129	-
	FIFO	-	-	-	-	-
	Instance	-	16	982	2064	-
	Memory	-	-	-	-	-
	Multiplexer	-	-	-	289	-
	Register	-	-	565	-	-
+	+	+	+	+	+	+
	Total	0	16	1547	2482	0
+	+	+	+	+	+	+
	Available	280	220	106400	53200	0
+	+	+	+	+	+	+
	Utilization (%)	0	7	1	4	0
+	+	+	+	+	+	+

Latency(Clock Cycles)		
+	+	+
	min	avg
+	+	+
	NA	NA
	68611	68611
+	+	+

2. Hardware resource & timing

```
=====
== Utilization Estimates
=====
* Summary:
```

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	-	-	-
FIFO	-	-	-	-	-
Instance	2	72	10032	12422	-
Memory	154	-	0	0	0
Multiplexer	-	-	-	7065	-
Register	-	-	26	-	-
Total	156	72	10058	19487	0
Available	280	220	106400	53200	0
Utilization (%)	55	32	9	36	0

Latency(Clock Cycles)		
min	avg	max
NA	NA	NA
5324	5324	5324

可以看到除了因為使用了 buffer 和 array partition 所以使用了許多 BRAM，其餘的資源約增加了 4~9 倍，而 Latency 卻縮短了近 13 倍。由此可知此為合理的加速。