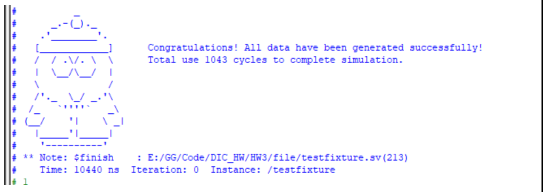
2025 Digital IC Design Homework 3

NAME	黃啟桓
Student ID	P76134927

Simulation Result

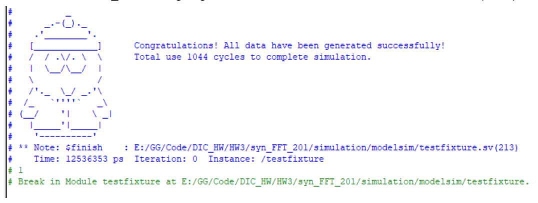
Functional simulation	Pass	Pre-Layout simulation	Pass



```
#
#          _
#      _.-(_)._
#    .'________'.
#   [____________]      Congratulations! All data have been generated successfully!
#   /  / .\/. \  \      Total use 1043 cycles to complete simulation.
#   |  \__/\__/  |
#   \            /
#   /'._  \_/ _.'\
#  /_   `''''`   _\
# (__/    '|    \ _|
#   |_____'|_____|
#    '----------'
# ** Note: $finish    : E:/GG/Code/DIC_HW/HW3/file/testfixture.sv(213)
#    Time: 10440 ns  Iteration: 0  Instance: /testfixture
# 1
```

Cycle : 1043

Please specify your clock width: 14 (ns)

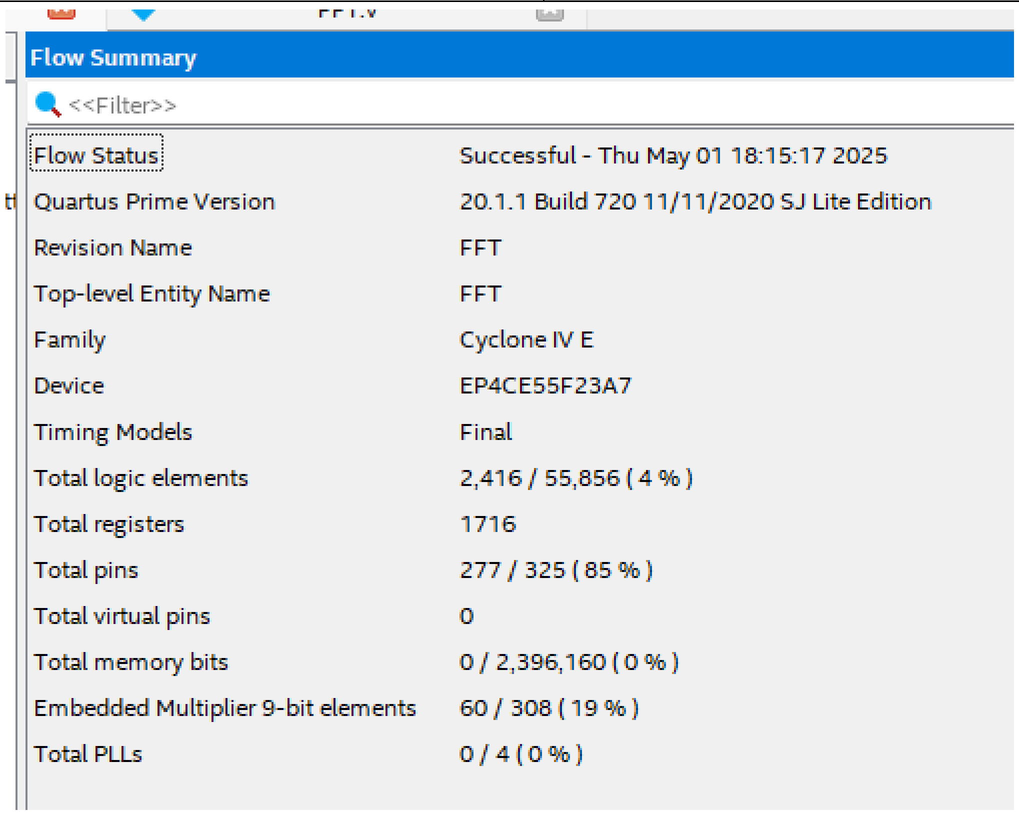


```
#
#          _
#      _.-(_)._
#    .'________'.
#   [____________]      Congratulations! All data have been generated successfully!
#   /  / .\/. \  \      Total use 1044 cycles to complete simulation.
#   |  \__/\__/  |
#   \            /
#   /'._  \_/ _.'\
#  /_   `''''`   _\
# (__/    '|    \ _|
#   |_____'|_____|
#    '----------'
# ** Note: $finish    : E:/GG/Code/DIC_HW/HW3/syn_FFT_201/simulation/modelsim/testfixture.sv(213)
#    Time: 12536353 ps  Iteration: 0  Instance: /testfixture
# 1
# Break in Module testfixture at E:/GG/Code/DIC_HW/HW3/syn_FFT_201/simulation/modelsim/testfixture.
```

Cycle : 1044

Synthesis Result

Total logic elements	2416
Total memory bits	0
Total registers	1716
Embedded multiplier 9-bit elements	60



Flow Summary

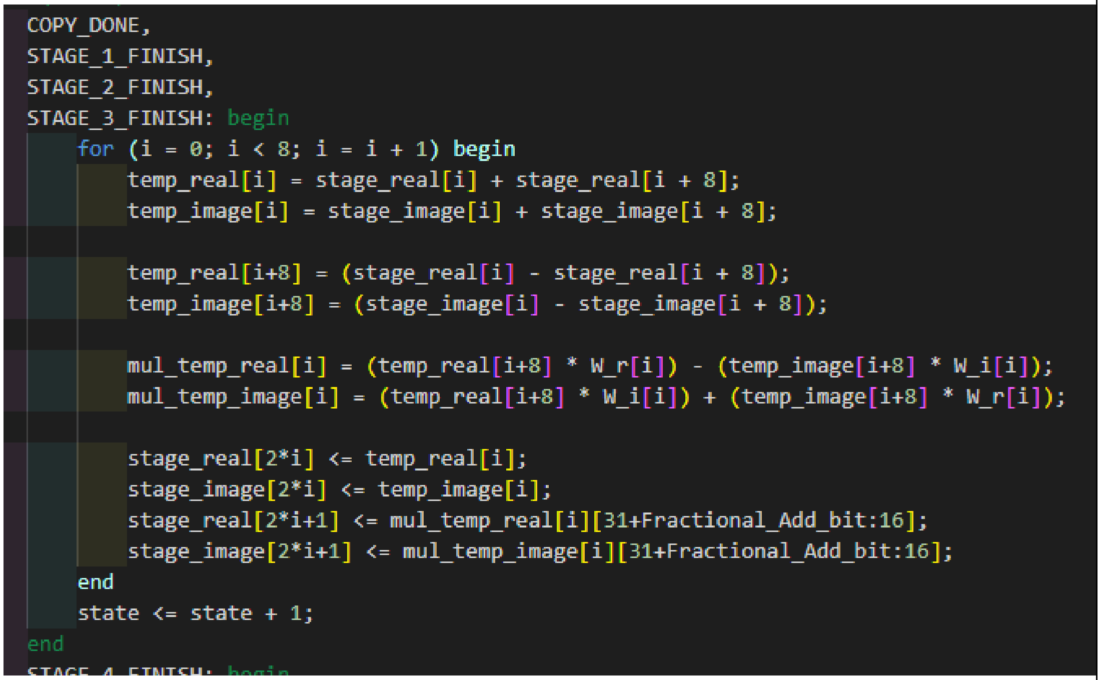
<<Filter>>

Flow Status	Successful - Thu May 01 18:15:17 2025
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	FFT
Top-level Entity Name	FFT
Family	Cyclone IV E
Device	EP4CE55F23A7
Timing Models	Final
Total logic elements	2,416 / 55,856 (4 %)
Total registers	1716
Total pins	277 / 325 (85 %)
Total virtual pins	0
Total memory bits	0 / 2,396,160 (0 %)
Embedded Multiplier 9-bit elements	60 / 308 (19 %)
Total PLLs	0 / 4 (0 %)

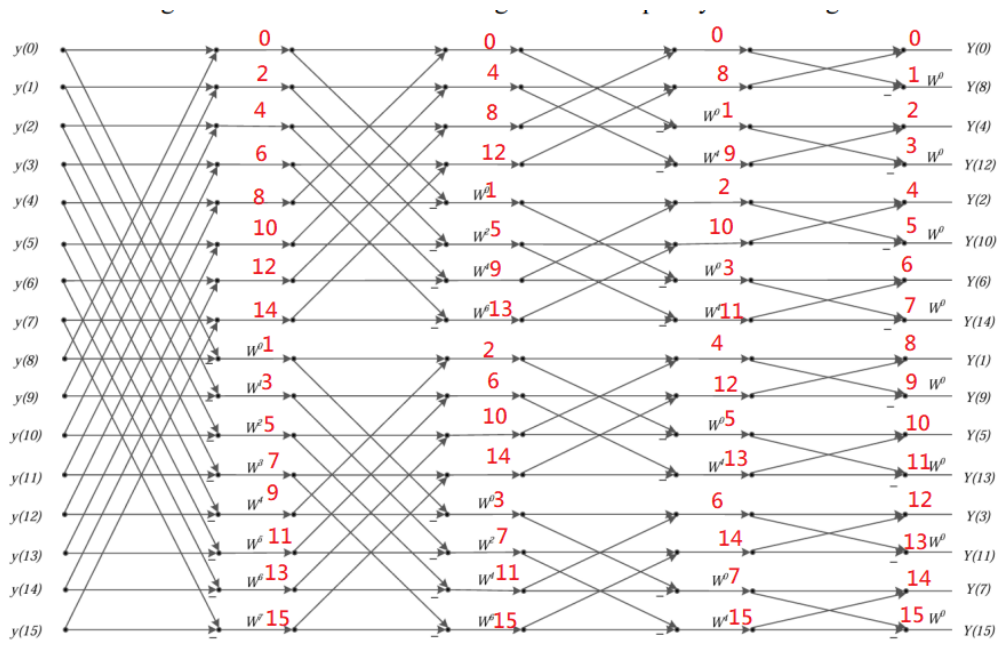
Description of your design

最大的改良是我讓每個 stage 都使用同樣的算法，每個 stage 都有 8 個需要乘 Weight，8 個不需要乘 Weight。只要設計的電路不要按照圖示，即 index 不要按照順序 0 ~ 15，而是規律的變換。變動的是 Weight，這樣就能使用更少的乘法器。

下圖是簡潔的示意圖，但為了縮短 time cost，所以實際分成 3 步，使用 non-blocking assignment。

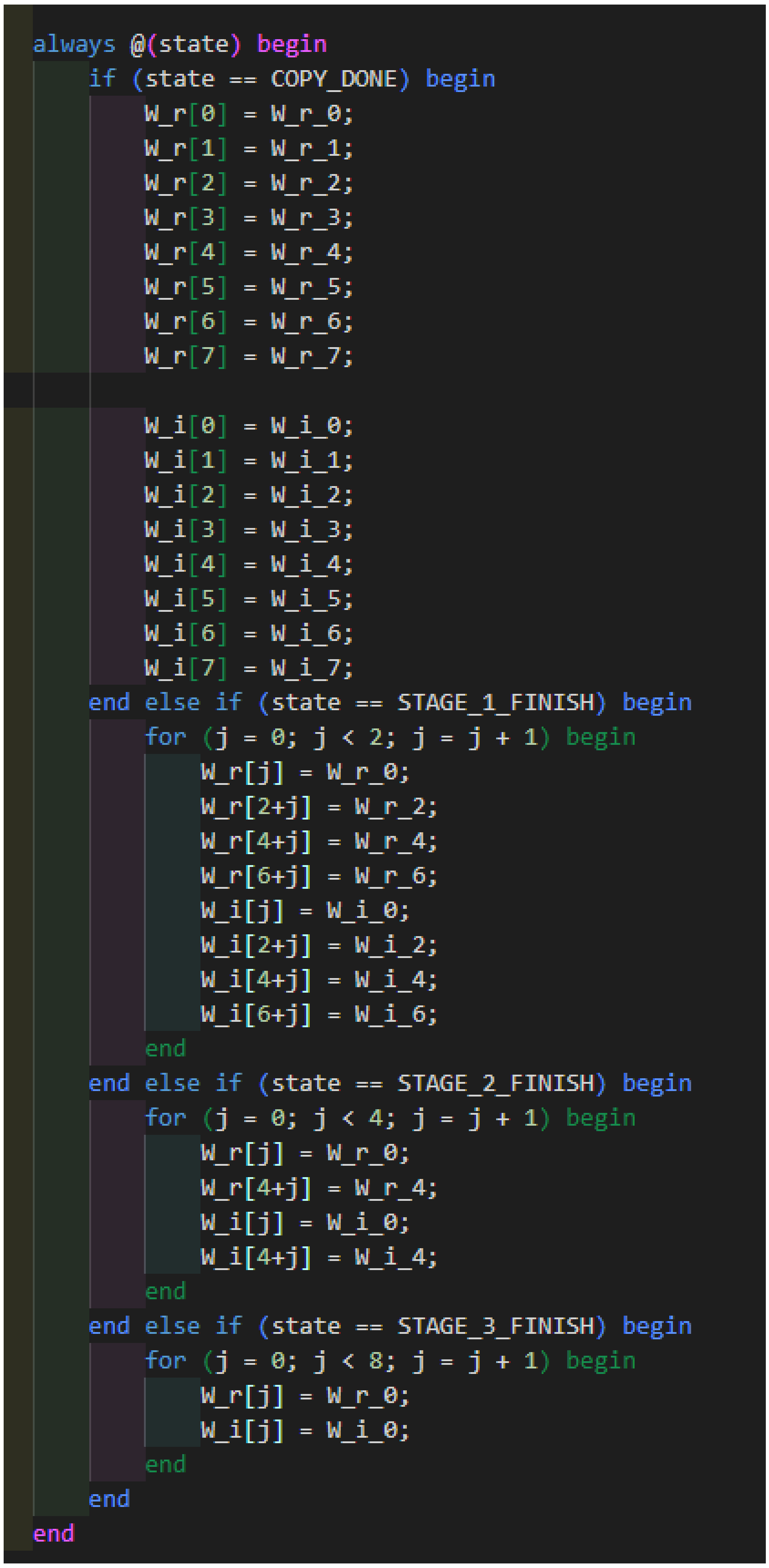


```
COPY_DONE,
STAGE_1_FINISH,
STAGE_2_FINISH,
STAGE_3_FINISH: begin
    for (i = 0; i < 8; i = i + 1) begin
        temp_real[i] = stage_real[i] + stage_real[i + 8];
        temp_image[i] = stage_image[i] + stage_image[i + 8];

        temp_real[i+8] = (stage_real[i] - stage_real[i + 8]);
        temp_image[i+8] = (stage_image[i] - stage_image[i + 8]);

        mul_temp_real[i] = (temp_real[i+8] * W_r[i]) - (temp_image[i+8] * W_i[i]);
        mul_temp_image[i] = (temp_real[i+8] * W_i[i]) + (temp_image[i+8] * W_r[i]);

        stage_real[2*i] <= temp_real[i];
        stage_image[2*i] <= temp_image[i];
        stage_real[2*i+1] <= mul_temp_real[i][31+Fractional_Add_bit:16];
        stage_image[2*i+1] <= mul_temp_image[i][31+Fractional_Add_bit:16];
    end
    state <= state + 1;
end
STAGE_4_FINISH: begin
```



可以理解成，中間的 stage 他的 output 的 index 是變化的，不是單純的 0 ~ 15，index 保持一樣的規律，舉例來說: index:0 永遠和 8 相乘，輸出是 index

0。Index: 1 永遠和 9 相乘，輸出永遠是 2…。基本上定義完 stage 1 的狀態就能確定後面的輸出的狀態。所以 stage 1 的 output index 是[0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15]。你會發現變化的 index 後，stage 2 和 stage 1 的算法一樣，都是 index 8~15 需要乘上一個 Weight。
而 weight 就是根據上面的規則變化。比如 Index 8 他就要和 W[0] 相乘。至於 W[0]是甚麼，看 stage 是甚麼。



```
always @(state) begin
    if (state == COPY_DONE) begin
        W_r[0] = W_r_0;
        W_r[1] = W_r_1;
        W_r[2] = W_r_2;
        W_r[3] = W_r_3;
        W_r[4] = W_r_4;
        W_r[5] = W_r_5;
        W_r[6] = W_r_6;
        W_r[7] = W_r_7;

        W_i[0] = W_i_0;
        W_i[1] = W_i_1;
        W_i[2] = W_i_2;
        W_i[3] = W_i_3;
        W_i[4] = W_i_4;
        W_i[5] = W_i_5;
        W_i[6] = W_i_6;
        W_i[7] = W_i_7;
    end else if (state == STAGE_1_FINISH) begin
        for (j = 0; j < 2; j = j + 1) begin
            W_r[j] = W_r_0;
            W_r[2+j] = W_r_2;
            W_r[4+j] = W_r_4;
            W_r[6+j] = W_r_6;
            W_i[j] = W_i_0;
            W_i[2+j] = W_i_2;
            W_i[4+j] = W_i_4;
            W_i[6+j] = W_i_6;
        end
    end else if (state == STAGE_2_FINISH) begin
        for (j = 0; j < 4; j = j + 1) begin
            W_r[j] = W_r_0;
            W_r[4+j] = W_r_4;
            W_i[j] = W_i_0;
            W_i[4+j] = W_i_4;
        end
    end else if (state == STAGE_3_FINISH) begin
        for (j = 0; j < 8; j = j + 1) begin
            W_r[j] = W_r_0;
            W_i[j] = W_i_0;
        end
    end
end
```