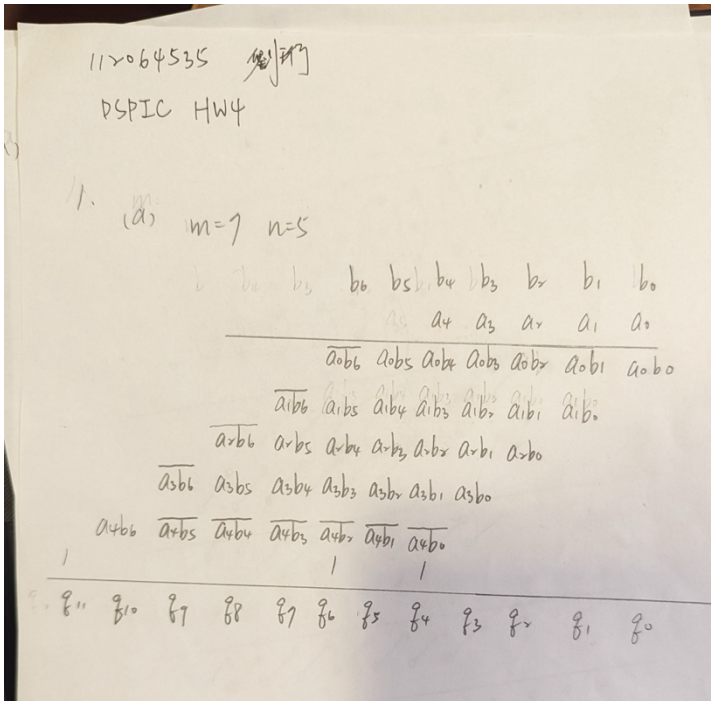
DSPIC HW4

學號：112064535

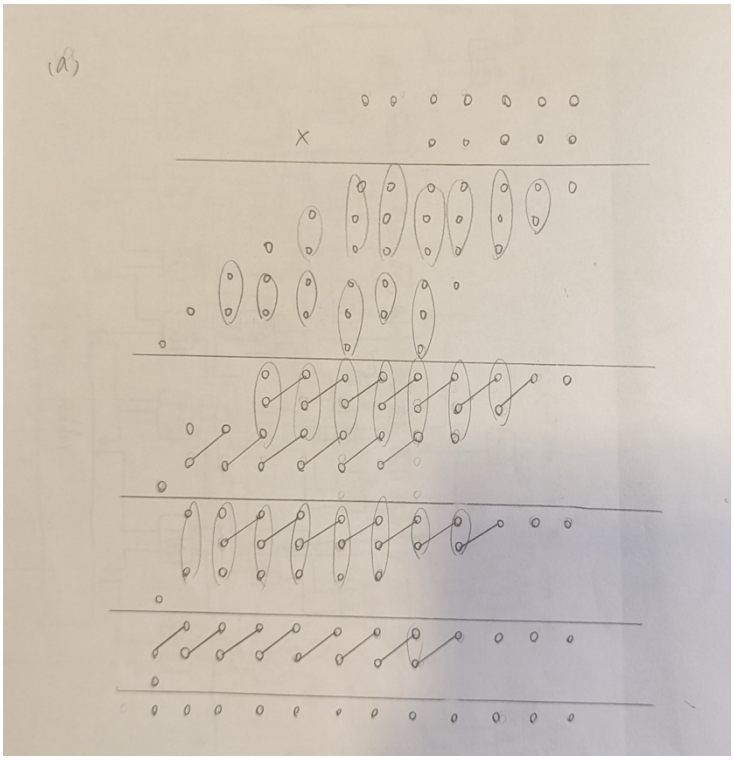
姓名：劉珩

1.

(a)

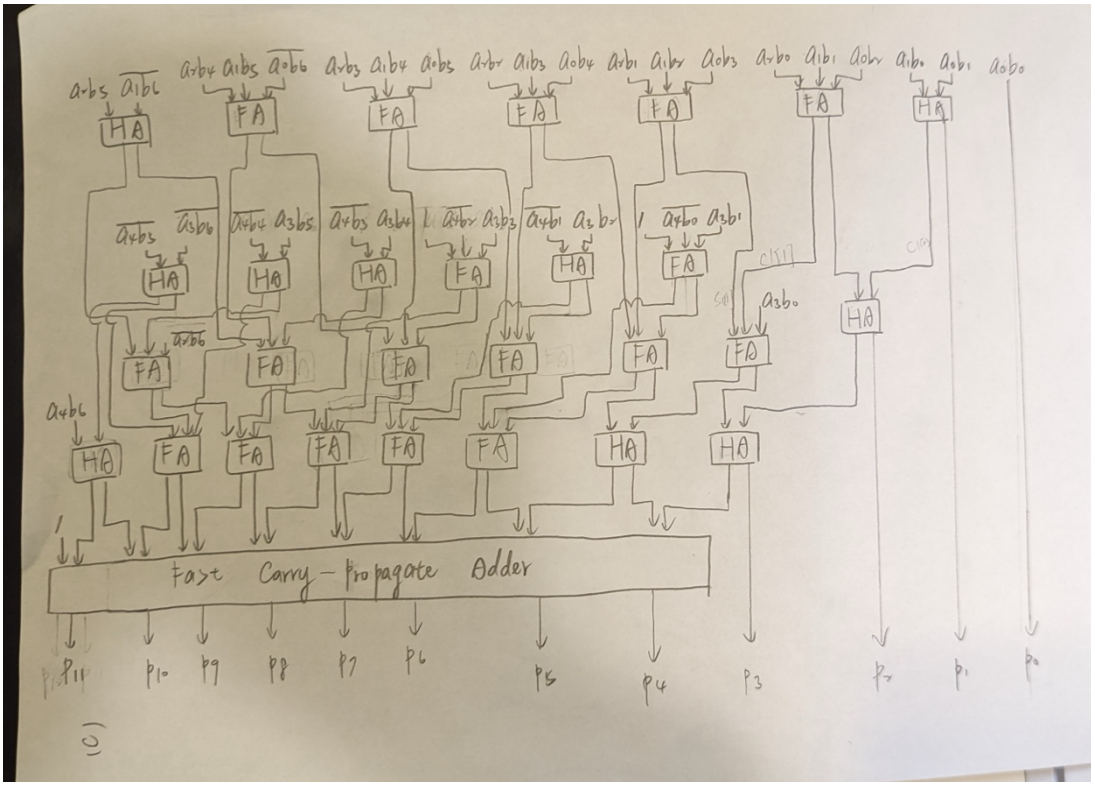


(圖一) 7X5 乘法器運算流程



(圖二) 運作流程(包含 18 個 FA 及 11 個 HA)

(b)

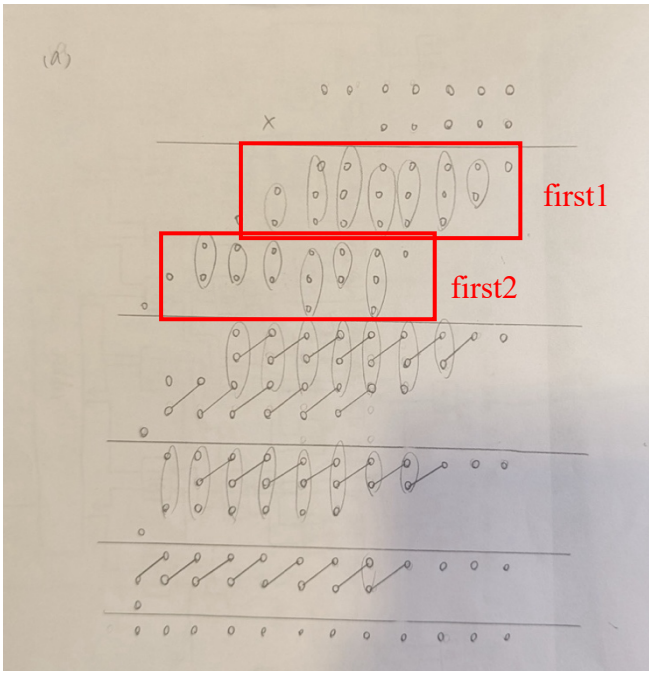


(圖三) Wallance-Tree 結構

(c)

Verilog code:

(1) 把加法器分成四層，第一層有兩個部分:



first1.v:

first1.v

```
1   `include "fulladd.v"
2   `include "halfadder.v"
3
4   module first1(
5       input  [5:0] a, //a有6個 a0b1~a0b6
6       input  [6:0] b, //a1b0~a1b6
7       input  [5:0] cin, // a2b0~a2b5
8       output [6:0] cout, //7個adder
9       output [6:0] sum //同上
10  );
11
12
13    half_add add1(
14      .A(a[0]),
15      .B(b[0]),
16      .sum(sum[0]), //p1
17      .cout(cout[0])
18  );
19
20    Full_Adder add2(
21      .A(a[1]),
22      .B(b[1]),
23      .Cin(cin[0]),
24      .Cout(cout[1]),
25      .Sum(sum[1])
26    );
27
```

```
28    Full_Adder add3(
29      .A(a[2]),
30      .B(b[2]),
31      .Cin(cin[1]),
32      .Cout(cout[2]),
33      .Sum(sum[2])
34    );
35
36      Full_Adder add4(
37      .A(a[3]),
38      .B(b[3]),
39      .Cin(cin[2]),
40      .Cout(cout[3]),
41      .Sum(sum[3])
42    );
43
44      Full_Adder add5(
45      .A(a[4]),
46      .B(b[4]),
47      .Cin(cin[3]),
48      .Cout(cout[4]),
49      .Sum(sum[4])
50    );
51
52      Full_Adder add6(
53      .A(a[5]),
54      .B(b[5]),
55      .Cin(cin[4]),
56      .Cout(cout[5]),
57      .Sum(sum[5])
58    );
59
60     half_add add7(
61      .A(b[6]),
62      .B(cin[5]),
63      .sum(sum[6]),
64      .cout(cout[6])
65     );
66
67
68  endmodule
```

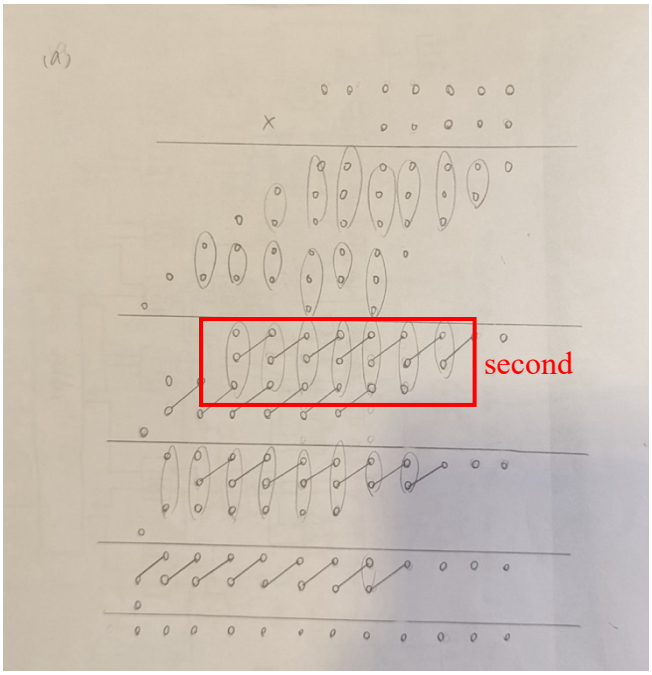
first2.v:

first2.v

```
1  // `include "fulladd.v"
2  // `include "halfadder.v"
3
4  module first2(
5      input  [5:0] a, //a有6個 a3b1~a3b6
6      input  [5:0] b, //a4b0~a4b5
7      //cin都是1 不需要設輸入
8      output [5:0] cout, //7個adder
9      output [5:0] sum //同上
10 );
11
12
13   Full_Adder add1(
14     .A(a[0]),
15     .B(b[0]),
16     .Cin(1'b1),
17     .Sum(sum[0]),
18     .Cout(cout[0])
19 );
20
21   half_add add2(
22     .A(a[1]),
23     .B(b[1]),
24     .cout(cout[1]),
25     .sum(sum[1])
26   );
27
28   Full_Adder add3(
29     .A(a[2]),
30     .B(b[2]),
31     .Cin(1'b1),
32     .Cout(cout[2]),
33     .Sum(sum[2])
34   );
35
36     half_add add4(
37     .A(a[3]),
38     .B(b[3]),
39     .cout(cout[3]),
40     .sum(sum[3])
41   );
42
43     half_add add5(
44     .A(a[4]),
45     .B(b[4]),
46     .cout(cout[4]),
47     .sum(sum[4])
48   );
```

```
50     half_add add6(
51     .A(a[5]),
52     .B(b[5]),
53     .cout(cout[5]),
54     .sum(sum[5])
55   );
56
57
58 endmodule
```

(2) 第二層:



second.v:

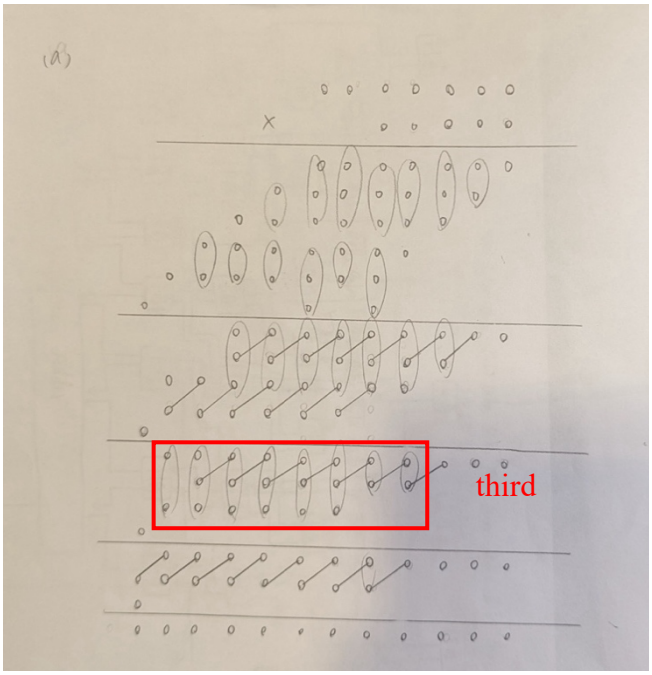
second.v

```
1   // `include "fulladd.v"
2   // `include "halfadder.v"
3
4   module second(
5       input  [6:0] a, //a有6個 從first1的第一個ha到a2b6
6       input  [6:0] b,
7       input  [5:0] cin,
8       output [6:0] cout, //7個adder
9       output [6:0] sum //同上
10  );
11
12
13    half_add add1(
14      .A(a[0]),
15      .B(b[0]),
16      .sum(sum[0]), //p2
17      .cout(cout[0])
18  );
19
20    Full_Adder add2(
21      .A(a[1]),
22      .B(b[1]),
23      .Cin(cin[0]), //a3b0
24      .Cout(cout[1]),
25      .Sum(sum[1])
26    );
27
28    Full_Adder add3(
29      .A(a[2]),
30      .B(b[2]),
31      .Cin(cin[1]),
32      .Cout(cout[2]),
33      .Sum(sum[2])
34    );
```

```
36      Full_Adder add4(
37      .A(a[3]),
38      .B(b[3]),
39      .Cin(cin[2]),
40      .Cout(cout[3]),
41      .Sum(sum[3])
42    );
```

```
44      Full_Adder add5(
45      .A(a[4]),
46      .B(b[4]),
47      .Cin(cin[3]),
48      .Cout(cout[4]),
49      .Sum(sum[4])
50    );
51
52      Full_Adder add6(
53      .A(a[5]),
54      .B(b[5]),
55      .Cin(cin[4]),
56      .Cout(cout[5]),
57      .Sum(sum[5])
58    );
59
60    Full_Adder add7(
61      .A(a[6]),
62      .B(b[6]),
63      .Cin(cin[5]),
64      .Cout(cout[6]),
65      .Sum(sum[6])
66    );
67
68
69  endmodule
```

(3) 第三層:

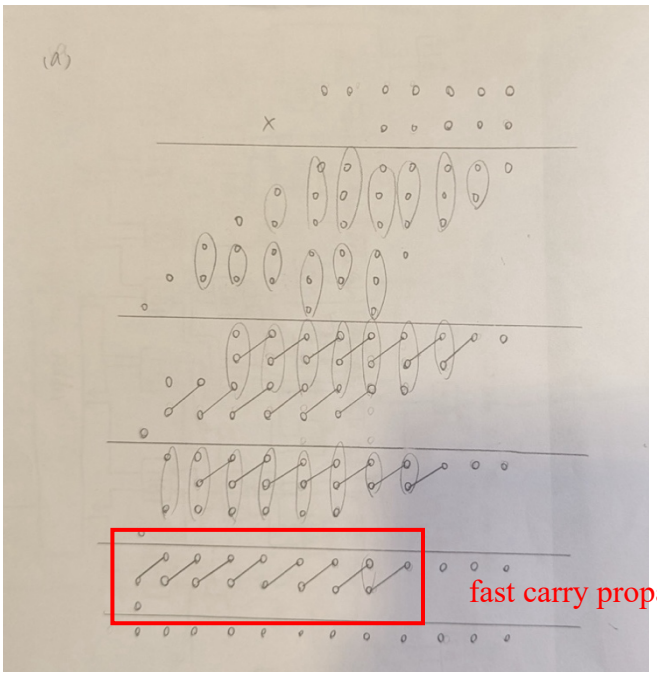


third.v:

```
module third(
    input  [7:0] a, //a有8個 每一個adder的第一的輸入
    input  [7:0] b, //同上
    input  [4:0] cin, // 每個adder的最後一個輸入
    output [7:0] cout, //8個adder
    output [7:0] sum //同上
);


  half_add add1(
    .A(a[0]),
    .B(b[0]),
    .sum(sum[0]), //p3
    .cout(cout[0])
);

  half_add add2(
    .A(a[1]),
    .B(b[1]),
    .cout(cout[1]),
    .sum(sum[1])
  );

  Full_Adder add3(
    .A(a[2]),
    .B(b[2]),
    .Cin(cin[0]),
    .Cout(cout[2]),
    .Sum(sum[2])
  );

    Full_Adder add4(
    .A(a[3]),
    .B(b[3]),
    .Cin(cin[1]),
    .Cout(cout[3]),
    .Sum(sum[3])
  );
```

```
    Full_Adder add5(
    .A(a[4]),
    .B(b[4]),
    .Cin(cin[2]),
    .Cout(cout[4]),
    .Sum(sum[4])
  );

    Full_Adder add6(
    .A(a[5]),
    .B(b[5]),
    .Cin(cin[3]),
    .Cout(cout[5]),
    .Sum(sum[5])
  );

    Full_Adder add7(
    .A(a[6]),
    .B(b[6]),
    .Cin(cin[4]),
    .Cout(cout[6]),
    .Sum(sum[6])
  );

   half_add add8(
    .A(a[7]),
    .B(b[7]),
    .sum(sum[7]),
    .cout(cout[7])
   );


endmodule
```

(4) 第四層:



fast_carry_prop.v:

```
module fast( //變成一般的carry propagate 一個ha 七個fa
    input  [7:0] a,
    input  [7:0] b, //a1b0~a1b6
    // input  [5:0] cin, // 每個人的carryin來自前一級就不用加了
    output cout, //最後不輸出p12
    output [7:0] sum //p4~p11
);

wire [6:0] w; //每一級的carryout 除了cout

  half_add add1(
    .A(a[0]),
    .B(b[0]),
    .sum(sum[0]), //p3
    .cout(w[0]) //下一級的carryin
);

  Full_Adder add2(
    .A(a[1]),
    .B(b[1]),
    .Cin(w[0]),
    .Cout(w[1]),
    .Sum(sum[1])
  );

  Full_Adder add3(
    .A(a[2]),
    .B(b[2]),
    .Cin(w[1]),
    .Cout(w[2]),
    .Sum(sum[2])
  );

    Full_Adder add4(
    .A(a[3]),
    .B(b[3]),
    .Cin(w[2]),
    .Cout(w[3]),
    .Sum(sum[3])
  );
```

```
45      Full_Adder add5(
46      .A(a[4]),
47      .B(b[4]),
48      .Cin(w[3]),
49      .Cout(w[4]),
50      .Sum(sum[4])
51    );
52
53      Full_Adder add6(
54      .A(a[5]),
55      .B(b[5]),
56      .Cin(w[4]),
57      .Cout(w[5]),
58      .Sum(sum[5])
59    );
60
61     Full_Adder add7(
62      .A(a[6]),
63      .B(b[6]),
64      .Cin(w[5]),
65      .Sum(sum[6]),
66      .Cout(w[6])
67     );
68
69      Full_Adder add8(
70      .A(a[7]),
71      .B(b[7]),
72      .Cin(w[6]),
73      .Sum(sum[7]),
74      .Cout(cout)
75     );
76
77
78
79    endmodule
```

(5) 在 top module 裡將所有 input a 跟 b 乘好再運算:

baugh_wooley.v:

```
 9   module baughwooley(a,b,p);
10
11   input [4:0] a;
12   input [6:0] b;
13
14   output [11:0] p;
15
16   wire [4:0] a;
17   wire [6:0] b;
18
19   wand [4:0][6:0]ab;
20
21   wire [6:0] c1;
22   wire [6:0] s1;// 給first1輸出
23   wire [5:0] c2;
24   wire [5:0] s2;// 給first2輸出
25
26   wire [6:0] a2; //第二層的輸入a
27   wire [5:0] b2; //第二層的輸入carryin
28
29   wire [6:0] c3;
30   wire [6:0] s3; //second的輸出
31
32   wire [7:0] a3; //third的輸入
33   wire [7:0] b3; //同上
34
35   wire [7:0] c4;
36   wire [7:0] s4; //third輸出
```

(圖四) 定義輸入及輸出

```
38  wire [7:0] a4; //最後一級carry propagate輸入
39
40  wire cout;
41
42  genvar i,j;
43  generate
44          for(i=0;i<4;i=i+1) begin
45              for(j=0;j<6;j=j+1) begin
46                  assign ab[i][j]=a[i]&b[j];
47              end
48          end
49  endgenerate
50
51  generate
52      // genvar i;
53      for(i=0;i<4;i=i+1) begin
54          assign ab[i][6]=~(a[i]&b[6]);
55      end
56  endgenerate
57
58  generate
59      // genvar j;
60      for(j=0;j<6;j=j+1)begin
61          assign ab[4][j]=~(a[4]&b[j]);
62      end
63  endgenerate
64
65      assign ab[4][6]=a[4]&b[6];
66
67      first1 adder1(
68          .a(ab[0][6:1]),
69          .b(ab[1][6:0]),
70          .cin(ab[2][5:0]),
71          .cout(c1),
72          .sum(s1)
73      );
```

(圖五) 配置所有的 ab 對

```
75          assign a2[5:0]=s1[6:1];
76          assign a2[6]=ab[2][6]; //定義second的輸入
77
78      first2 adder2(
79          .a(ab[3][6:1]),
80          .b(ab[4][5:0]),
81          .cout(c2),
82          .sum(s2)
83      );
84          assign b2[5:1]=s2[4:0];
85          assign b2[0]=ab[3][0];//定義second的輸入 等等不行就把assign拿掉
86
87      second adder3(
88          .a(a2), //a2b6+s1[6:1]
89          .b(c1),
90          .cin(b2), //s2[4:0]+a3b0
91          .cout(c3),
92          .sum(s3)
93      );
94
95          assign a3[7]=ab[4][6];
96          assign a3[6]=s2[5];
97          assign a3[5:0]=s3[6:1];
98
99          assign b3[7]=c2[5];
100         assign b3[6:0]=c3[6:0];
101
102     third adder4(
103         .a(a3), // a4b6+s2[5]+s3[6:1]
104         .b(b3), // c2[5]+c3[6:0]
105         .cin(c2[4:0]), //c2[4:0]
106         .cout(c4),
107         .sum(s4)
108     );
109
110         assign a4[7]=1'b1;
111         assign a4[6:0]=s4[7:1];
112     fast adder5(
113         .a(a4), // 1'b1+s4[7:1]
114         .b(c4), //c4[7:0]
115         .cout(cout),
116         .sum(p[11:4])
117     );
```

```
119     assign p[0]=ab[0][0];
120     assign p[1]=s1[0];
121     assign p[2]=s3[0];
122     assign p[3]=s4[0];
123
124 endmodule
```

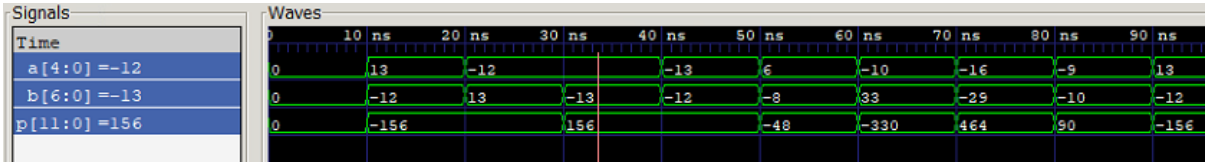
(圖六)(圖七) 運算後決定輸出

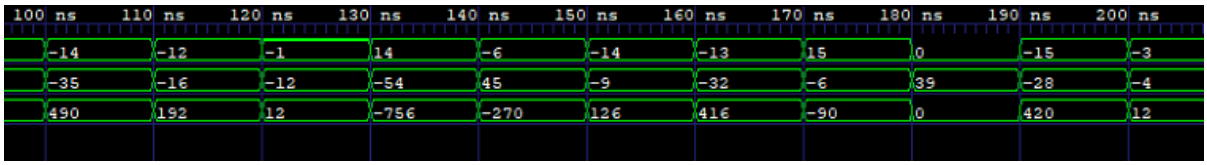
(6) test bench 輸入及輸出結果:

```
24  initial
25  begin
26      $dumpfile("baugh_tb.vcd");
27      $dumpvars;
28      #10
29      a=5'b01101; //13
30      b=7'b1110100; //-12
31      #10
32      a=5'b10100; //-13
33      b=7'b0001101; //12
34      #10
35      a=5'b10100; //-12
36      b=7'b1110011; //-13
37      #10
38      a=5'b10011; //-13
39      b=7'b1110100; //-12
40      #10
41      a=5'b00110; //6
42      b=7'b1111000; //-8
43      #10
44      a=5'b10110;
45      b=7'b0100001;
46      #10
47      a=5'b10000;
48      b=7'b1100011;
49      #10
50      a=5'b10111;
51      b=7'b1110110;
52      #10
53      a=5'b01101;
54      b=7'b1110100; //
55      #10
56      a=5'b10010;
57      b=7'b1011101;
58      #10
59      a=5'b10100;
60      b=7'b1110000;
61      #10
62      a=5'b11111;
63      b=7'b1110100;
64      #10
65      a=5'b01110;
66      b=7'b1001010;
67      #10
68      a=5'b11010;
69      b=7'b0101101;
```

```
70      #10
71      a=5'b10010;
72      b=7'b1110111;
73      #10
74      a=5'b10011;
75      b=7'b1100000;
76      #10
77      a=5'b01111;
78      b=7'b1111010;
```

```
79      #10
80      a=5'b00000;
81      b=7'b0100111;
82      #10
83      a=5'b10001;
84      b=7'b1100100;
85      #10
86      a=5'b11101;
87      b=7'b1111100;
88      #10
89      $finish;
90  end
91
92  endmodule
```





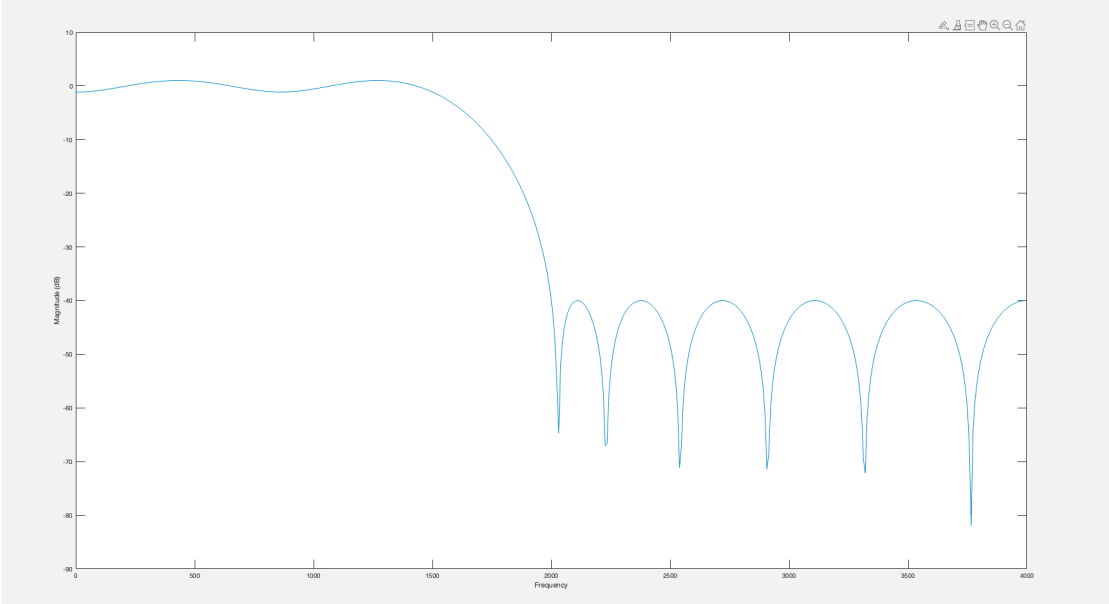
2.

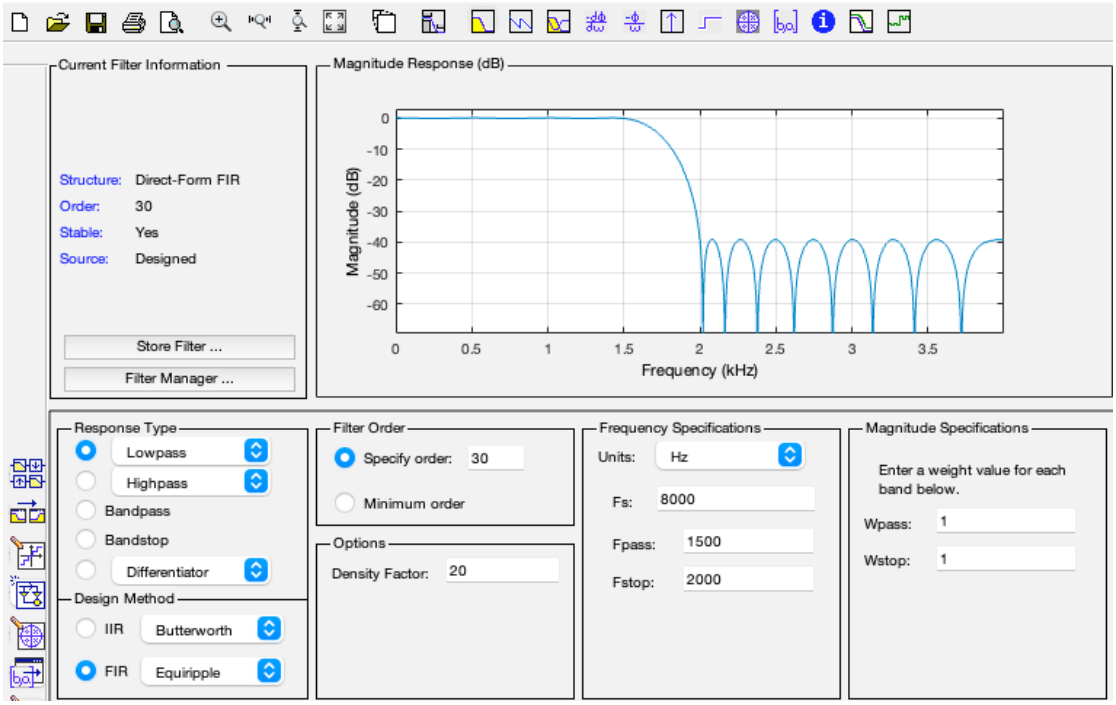
(a)

(i)利用 matlab 產生 21 個 tap 的 filter coefficient，以符合規定的 filter constrains，coefficient 為:

-0.0105963997986059
-0.0380214361435444
-0.0365217681824527
0.00336388262116807
0.0454851577619937
0.0194242970434525
-0.0610882554173864
-0.0752620124582160
0.0734976427892512
0.307268070442418
0.421992848704956
0.307268070442418
0.0734976427892512
-0.0752620124582160
-0.0610882554173864
0.0194242970434525
0.0454851577619937
0.00336388262116807
-0.0365217681824527
-0.0380214361435444
-0.0105963997986059

(ii)21 tap 的 FIR low pass filter 頻率響應為:





可以看到 tap 數更高的 filter 在 transition band 上更加 sharp

(b)

(i) c program:

```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <math.h>

#define NUM_SAMPLES 512

#define M 21   // 濾波器阶数


float generateRandomFloat() {
    // 生成随机整数
    int randomInt = rand();

    // 映射到浮点数范围 [0, 10] 要取範圍多少隨意
    float randomFloat = ((float)randomInt /RAND_MAX) ; //RAND_MAX

    // 生成带符号的随机浮点数
    return (rand() % 2 == 0) ? randomFloat : -randomFloat;
}
```

(圖八)生成亂數之副函式

```c
int main() {
    float inputSignal[NUM_SAMPLES]; // 输入信号
    float outputSignal[NUM_SAMPLES]; // 输出信号
    float filterCoeff[M] = {-0.0105963997986059, -0.0380214361435444, -0.0365217681824527, 0.00336388262116807, 0.0454851577619937, 0.0
    srand(time(NULL));

    // 设置随机种子
    srand((unsigned int)time(NULL));

    // 生成随机浮点数
    float randomSignal[NUM_SAMPLES];
    float sumOfSquares = 0;


    for (int i = 0; i < NUM_SAMPLES; i++) {
        randomSignal[i] = generateRandomFloat(); //-1~1之間的floating point
        sumOfSquares += randomSignal[i] * randomSignal[i];
    }

    // 调整幅度，使得平均功率为1
    float scaleFactor = 1 / sqrt(sumOfSquares);

    for (int i = 0; i < NUM_SAMPLES; i++) {
        randomSignal[i] *= scaleFactor;
        inputSignal[i]=randomSignal[i]*sqrt(NUM_SAMPLES);
        // printf("inputsignal = %f\n", randomSignal[i]);
    }

        // 执行 FIR 低通滤波器
    firLowPassFilter(inputSignal, outputSignal, filterCoeff, NUM_SAMPLES, M);

    // 计算平均功率
    float averagePower = 0;
    for (int i = 0; i < NUM_SAMPLES; i++) {
        averagePower += inputSignal[i] * inputSignal[i];
    }
```

```
61      averagePower /= NUM_SAMPLES;
62      printf("Generated signal with average power = %f\n", averagePower);
63
64      // 打印输入和输出信号
65      printf("Input Signal: ");
66      for (int i = 0; i < NUM_SAMPLES; i++) {
67          printf("%f ", inputSignal[i]);
68      }
69
70      printf("\nOutput Signal (FIR Low Pass Filtered): ");
71      for (int i = 0; i < NUM_SAMPLES; i++) {
72          printf("%f ", outputSignal[i]);
73      }
74      // printf("%f ", outputSignal[0]);
75
76      return 0;
77  }
```

(圖九)(圖十)將隨機生成的 512 筆資料 scaling 成 power 為 1 的 input

```
79  // FIR 低通滤波器的实现
80  void firLowPassFilter(float input[], float output[], float filterCoeff[], int inputLength, int filterLength) {
81      for (int n = 0; n < inputLength; n++) {
82          output[n] = 0.0;
83          for (int k = 0; k < filterLength; k++) {
84              if (n - k >= 0) {
85                  output[n] += filterCoeff[k] * input[n - k]; //convolution
86              }
87          }
88      }
89  }
```

(圖十一)convolution

```
Generated signal with average power = 1.000000
Input Signal: -1.340897 1.251119 -0.844923 0.247925 -0.887348 1.179609 1.643725 1.056794 0.463018 1.432070 -0.197500 0.112583 -1.533250 -0.043409 0.106639 0.108163 -1.586210
1.347913 1.321556 0.431794 1.172345 -0.515141 1.021076 0.263878 -0.314157 -0.530758 0.904643 -0.265815 -0.357032 -0.651819 0.902878 0.689262 -1.020968 -1.624012 0.167142 -0.
602604 -0.129730 0.124132 1.009884 0.843797 -1.647025 -0.745211 0.511055 0.086590 -1.342705 1.683588 1.378432 -0.100818 0.521633 -0.966550 -1.356561 -1.304256 -0.848987 -1.
210508 -1.295377 -0.784651 1.523307 1.197323 0.069293 0.277043 1.321774 0.179314 -0.639313 1.301270 1.006320 0.131201 -1.072039 -1.458300 -1.393583 -1.287966 1.654029 1.
031796 0.459251 0.709825 -0.495572 -1.013503 1.284369 0.487221 -0.135029 1.630933 -1.263616 -0.456713 -1.318913 -0.642574 1.247319 -0.045994 -1.654899 1.018750 -0.576685 0.
800087 1.023020 1.622964 0.538910 0.396396 1.253890 -1.186585 1.701861 0.248319 1.002769 0.292317 1.683487 -0.898294 -0.315223 0.217470 0.589907 1.513118 0.888889 -0.347308
1.042679 -1.244327 1.250611 -0.070538 -0.767815 -0.548631 -1.260231 1.574342 0.515514 0.838330 -1.586359 0.978552 -1.505648 0.043469 -0.219778 0.327659 -0.669412 -1.154548 0.
729453 -1.118269 0.797789 -0.243855 -1.406766 1.379433 0.095821 -0.438654 0.349278 0.540543 -0.063590 -1.025562 1.166448 -1.250657 -0.175825 0.461355 0.707910 0.710856 1.
122321 0.331916 -0.555407 1.506046 1.310367 0.214565 -0.100055 0.270378 -0.871462 -0.323179 -0.656657 0.321151 1.272910 -0.429397 -0.769408 0.374498 -0.692409 0.535040 -0.
347352 1.218808 0.270214 -1.446353 0.813023 -1.036573 1.346200 0.805085 1.572404 1.506537 -1.363393 1.070001 -1.304227 1.003989 1.118510 -0.874578 -0.074350 -1.508233 -0.
786311 -0.117200 0.265272 1.688133 1.231945 0.794815 -0.185625 0.662320 0.030310 -0.159164 0.464430 0.436112 0.680750 -1.683377 0.433673 -0.455476 -1.096529 0.421859 -1.
300582 1.051459 0.489514 1.110717 0.820032 0.400650 -0.848933 -1.073314 0.035361 1.691488 0.076417 1.486871 -0.886303 -0.286920 0.880352 1.389575 -0.439747 -0.246087 0.
878957 0.148962 -0.701663 0.317623 1.226954 1.691433 -1.508875 -1.595099 -0.717852 -1.207786 -0.746344 -0.303710 -1.177820 0.601206 -0.718670 -0.444709 0.420421 0.885413 1.
579721 1.243577 1.147667 0.385508 -0.067648 -1.372740 -1.309402 0.611204 -1.485840 0.691603 -1.107148 0.240531 -1.692831 -1.290636 -1.174779 0.570078 1.341841 -1.000793 0.
971718 1.142534 1.237153 -1.381915 0.101342 -0.222549 -0.539530 1.020218 -0.331026 0.822631 -0.630101 -1.697802 -0.021755 1.058503 -0.493556 1.234631 1.463369 0.853483 -0.
170545 0.792765 -0.031971 0.060141 -1.235940 -1.111859 1.215024 -1.097656 1.612068 1.405416 -0.829953 1.534622 -1.388294 -0.907094 1.659963 -0.539460 -1.505773 -0.546457 0.
777231 -1.444828 1.128361 -1.023729 -0.392280 1.189454 -0.652974 -1.570051 0.254738 1.411017 0.139449 1.094071 0.381414 -0.204770 0.377606 -1.188925 -0.035571 -0.678128 -0.
751288 0.052117 0.064686 0.129652 -0.457165 1.165515 0.597363 -1.309691 0.923338 -1.208943 1.369935 -0.743501 1.461069 0.203840 -0.256175 1.584688 0.117873 -0.983342 0.
984006 1.650323 -1.399669 1.036657 1.096071 0.157500 0.647044 1.466729 0.441388 -0.142248 -1.087245 -1.211816 -1.025378 -0.964874 -0.300952 1.283474 0.333115 -0.405726 1.
013431 -1.362199 -0.700468 1.701466 0.401974 1.484921 1.289550 1.447402 1.612040 -0.207670 1.335994 1.069628 0.754159 -0.089249 0.567537 1.207764 -0.964086 0.139459 1.103148
1.393326 1.203057 0.496276 -0.067949 -1.352859 -0.019473 0.657574 -0.808686 -0.216536 0.417847 1.250416 0.934381 -1.119588 0.747850 0.757554 0.877786 1.009820 0.136273 -0.
534875 0.411081 0.218929 -1.683439 1.180031 -1.146329 1.203454 1.259427 -0.799625 -0.240721 -0.453589 -0.585208 -1.191003 -1.052118 1.288926 -1.021865 0.185635 -1.384366 0.
949067 0.901888 0.725849 1.070175 -1.129866 1.702798 -1.465483 -0.589806 1.419686 0.027690 1.143105 -1.214917 1.400473 1.273540 0.370664 0.218918 -0.303264 -0.662637 0.
563798 0.012575 1.484026 -1.222558 -0.875885 1.395928 -1.314829 1.216661 0.846579 1.204776 1.362060 -0.466707 -1.430351 -1.463795 -1.314664 1.179940 0.933675 -1.538295 0.
060382 -0.466715 -1.672655 1.655649 1.370606 0.276275 -0.644441 -1.683887 -1.677718 -0.872339 -1.175624 -0.740736 0.356885 1.389802 -0.340963 1.165228 -0.717545 0.821167 -0.
814681 0.532812 -1.279313 0.886124 -1.686049 -1.367880 -1.549698 -1.301163 -0.935181 -1.632693 1.042613 -1.410396 -1.215071 -1.077581 -1.559938 -1.668904 1.052057 -1.699111
0.654957 0.190650 1.655616 -0.443161 0.863132 1.585442 0.417826 0.444929 -0.610340 -1.082473 -0.471199 -0.954626 0.670554 -0.303484 -0.066397 -0.442285 -1.420302 0.690664 1.
399194 -0.669061 0.445418 0.948447 -0.556906 -1.138837 -1.666861 1.295997 0.510018 -1.195501 -0.488323 1.653244 1.178752 -1.055091 1.340647 0.130769 0.365569 1.093264 1.
697911 -0.767615 -0.581339 -0.773018 -1.499973 0.239612 -0.745718
```

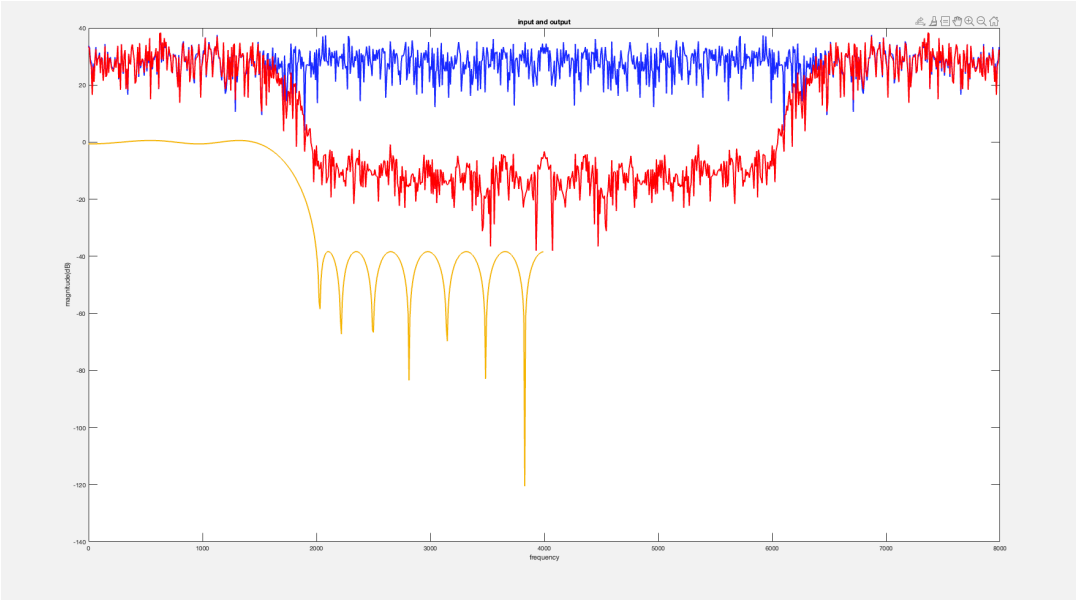


```
Output Signal (FIR Low Pass Filtered): 0.014209 0.037725 0.010356 -0.020706 -0.025948 0.040203 0.038757 -0.100407 -0.277796 -0.301043 -0.176006 -0.095156 -0.183321 -0.208233
0.078551 0.657388 1.191281 1.391535 1.133664 0.588225 -0.032312 -0.423800 -0.494860 -0.398954 -0.370026 -0.370451 -0.141273 0.377191 0.883410 1.015674 0.754887 0.401585 0.
184651 0.156647 0.189641 0.140220 -0.058941 -0.280826 -0.290199 -0.033306 0.245452 0.150249 -0.356195 -0.895385 -0.984360 -0.512349 0.151752 0.547937 0.467161 0.040052 -0.
359783 -0.494419 -0.354972 -0.063170 0.343116 0.777056 0.991753 0.653626 -0.148074 -0.880992 -1.170599 -1.169616 -1.233250 -1.328704 -1.089642 -0.331242 0.549171 1.056050 1.
050225 0.715608 0.361486 0.200486 0.378476 0.746996 0.855341 0.273796 -0.864936 -1.803462 -1.705111 -0.564592 0.754605 1.338829 0.924171 0.104588 -0.419163 -0.296140 0.
318256 0.901126 0.930272 0.294585 -0.628571 -1.174482 -0.963300 -0.255457 0.209885 0.043916 -0.393353 -0.532649 -0.068651 0.695343 1.213670 1.243007 0.940244 0.552653 0.
304368 0.246734 0.369199 0.669711 0.919506 0.862875 0.407420 -0.213896 -0.413264 0.064521 0.866594 1.217446 0.812029 0.096364 -0.182445 0.135000 0.412366 0.093712 -0.619608
-0.912888 -0.393285 0.510302 0.925788 0.481618 -0.291058 -0.744547 -0.541520 -0.085605 0.058731 -0.194077 -0.545236 -0.582351 -0.334756 -0.068470 -0.044295 -0.151648 -0.
146233 0.036943 0.216804 0.226091 0.125742 0.042811 0.031033 -0.067348 -0.296975 -0.469011 -0.304788 0.266135 0.834398 0.969066 0.642835 0.302565 0.359017 0.727726 1.004922
0.806559 0.226307 -0.393318 -0.741581 -0.627322 -0.177470 0.298745 0.428149 0.117139 -0.338835 -0.565495 -0.341626 0.122775 0.478984 0.495018 0.110394 -0.361923 -0.548638 -0.
143005 0.721438 1.472669 1.523863 0.818472 -0.086293 -0.466919 -0.126743 0.446619 0.602095 0.048665 -0.808084 -1.311199 -1.073168 -0.208627 0.728581 1.282115 1.247036 0.
799911 0.330393 0.077375 0.132998 0.341286 0.420322 0.243859 -0.139690 -0.453291 -0.543987 -0.464687 -0.424304 -0.465737 -0.379003 0.087910 0.787929 1.230130 0.935983 -0.
007489 -0.824801 -0.749956 0.155783 1.029678 1.027344 0.328858 -0.241403 -0.135701 0.377955 0.746438 0.680071 0.321827 -0.069498 -0.227500 0.036991 0.654748 1.008997 0.
605798 -0.437071 -1.358787 -1.568958 -1.175666 -0.711129 -0.505388 -0.476753 -0.449025 -0.343034 -0.098809 0.338179 0.962157 1.481947 1.614052 1.206001 0.452966 -0.351184 -0.
842848 -0.945772 -0.717065 -0.341336 -0.153294 -0.369208 -0.924093 -1.332667 -1.197826 -0.544526 0.119580 0.420943 0.528728 0.660517 0.841365 0.683474 0.099501 -0.500060 -0.
550507 -0.064491 0.430113 0.420053 -0.113193 -0.687521 -0.844821 -0.458484 0.235490 0.834872 1.042959 0.915918 0.663007 0.482841 0.373643 0.121556 -0.289478 -0.695262 -0.
777451 -0.341377 0.440183 1.015896 0.957350 0.373852 -0.214910 -0.308551 -0.037542 0.075919 -0.215642 -0.583189 -0.676159 -0.463521 -0.241178 -0.172349 -0.065901 0.019128 -0.
101047 -0.376135 -0.485992 -0.071143 0.626396 1.030819 0.841437 0.377587 -0.010798 -0.250465 -0.412168 -0.538431 -0.555454 -0.423926 -0.296600 -0.165037 0.021507 0.286934 0.
413382 0.243252 -0.095138 -0.253222 -0.115399 0.170693 0.413406 0.488744 0.482099 0.415497 0.294775 0.184704 0.194100 0.369678 0.512062 0.528224 0.435065 0.451219 0.643938 0.
866629 0.910316 0.562544 -0.141106 -0.881439 -1.349398 -1.352663 -0.837947 -0.110123 0.449500 0.528115 0.129034 -0.375176 -0.532200 -0.161338 0.508819 1.125503 1.428272 1.
400808 1.216112 1.009136 0.863400 0.780430 0.707860 0.659843 0.613597 0.431675 0.094464 -0.148246 0.079487 0.781868 1.357282 1.184950 0.338820 -0.420108 -0.582175 -0.347768
-0.218141 -0.245090 -0.042522 0.426261 0.687345 0.471954 0.130124 0.185475 0.673453 0.982503 0.763760 0.270119 -0.037591 -0.116648 -0.225482 -0.352920 -0.261334 0.141991 0.
556275 0.584278 0.214529 -0.368675 -0.844277 -1.002682 -0.753743 -0.304204 -0.052774 -0.201647 -0.485027 -0.404677 0.162921 0.833840 1.101793 0.740049 0.081312 -0.397574 -0.
351750 0.081644 0.494906 0.483165 0.206143 0.143306 0.568379 1.028940 0.853041 0.127451 -0.520148 -0.445546 0.138108 0.498336 0.332856 -0.121654 -0.379700 -0.298703 0.009931
0.530920 1.166801 1.460526 0.953878 -0.313576 -1.495346 -1.639706 -0.677579 0.474895 0.721022 -0.040133 -0.984035 -1.100891 -0.188927 0.935368 1.262572 0.522872 -0.706246 -1.
538800 -1.677854 -1.336545 -0.839218 -0.289312 0.218063 0.627438 0.714954 0.523496 0.247499 0.066411 0.013285 -0.018524 -0.113510 -0.389685 -0.846827 -1.324968 -1.611519 -1.
510007 -1.058377 -0.559270 -0.388291 -0.685363 -1.188161 -1.472197 -1.354713 -0.947553 -0.492738 -0.082012 0.286122 0.533557 0.655005 0.703019 0.879190 1.077217 0.975803 0.
366068 -0.501289 -1.096008 -0.959809 -0.352351 0.124828 0.041804 -0.389319 -0.639808 -0.467271 0.039126 0.550069 0.774728 0.549039 -0.060467 -0.649927 -0.825356 -0.490612 -0.
094849 -0.046865 -0.175699 0.006762 0.508280 0.857992 0.670294 0.238668
[Done] exited with code=0 in 2.464 seconds
```


(圖十二)輸出結果

```
>> mean(abs(c_input).^2)

ans =

    1.0000
```

在 matlab 中驗證 c 產生的 input average power 為 1



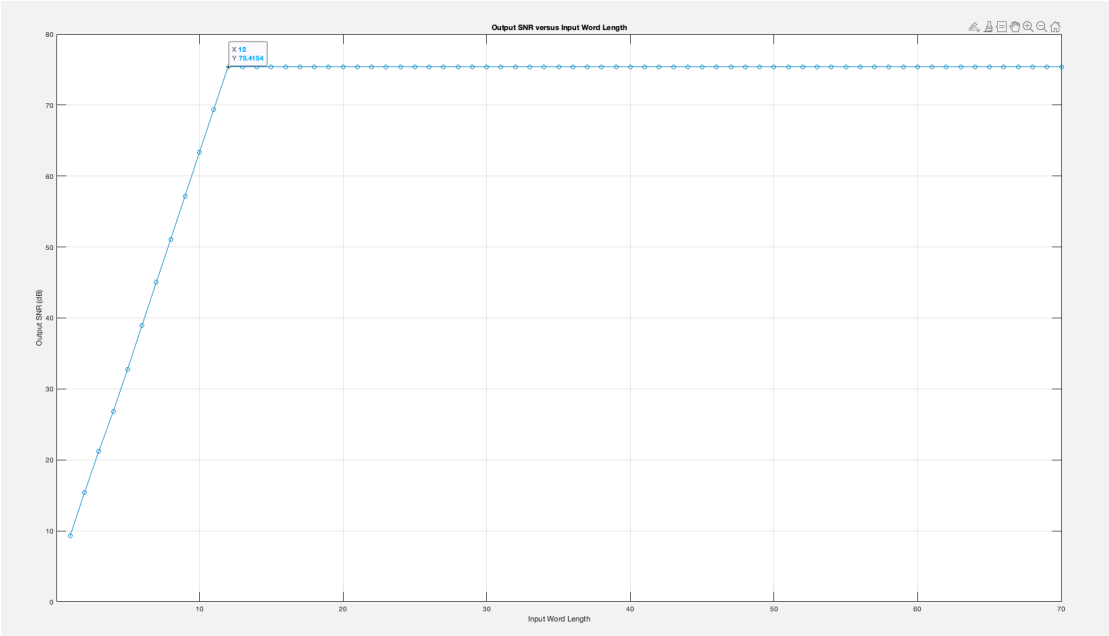
(圖十三)輸入與輸出頻譜對比

(ii)

(1) 先將 input word-length quantize 為 fix point，整數為 4bits，小數為 12bits，共 16bits ，再把 fix point input 跟 floating point input 比較:

```
>> inputWordLengths = 1:70;
% 初始化存储 SNR 的数组
outputSNR = zeros(size(inputWordLengths));
% 模拟不同输入字长下的 FIR 滤波器输出，并计算输出 SNR
for i = 1:length(inputWordLengths)
inputWordLength = inputWordLengths(i);
% 在这里模拟 FIR 滤波器的输出，得到outputSignal
% 计算输出 SNR
% 在这里计算 SNR，假设有变量 powerOfFloatingPointResult 表示浮点结果的功率
% 以及变量 powerOfDifference 表示浮点和定点结果之间的差异的功率
INSig=fi(quatizeinput,1,inputWordLength+4 ,inputWordLength,'RoundMode', 'floor');
IN=double(INSig);
OUTSig=filter(b,a,IN);
powerOfDifference= sum(abs(output-OUTSig).^2);
outputSNR(i) = 10 * log10(Pfloat / powerOfDifference);
end
% 绘制 SNR 与输入字长的关系图
figure;
plot(inputWordLengths, outputSNR, '-o');
xlabel('Input Word Length');
ylabel('Output SNR (dB)');
title('Output SNR versus Input Word Length');
grid on;
```

小數點位數

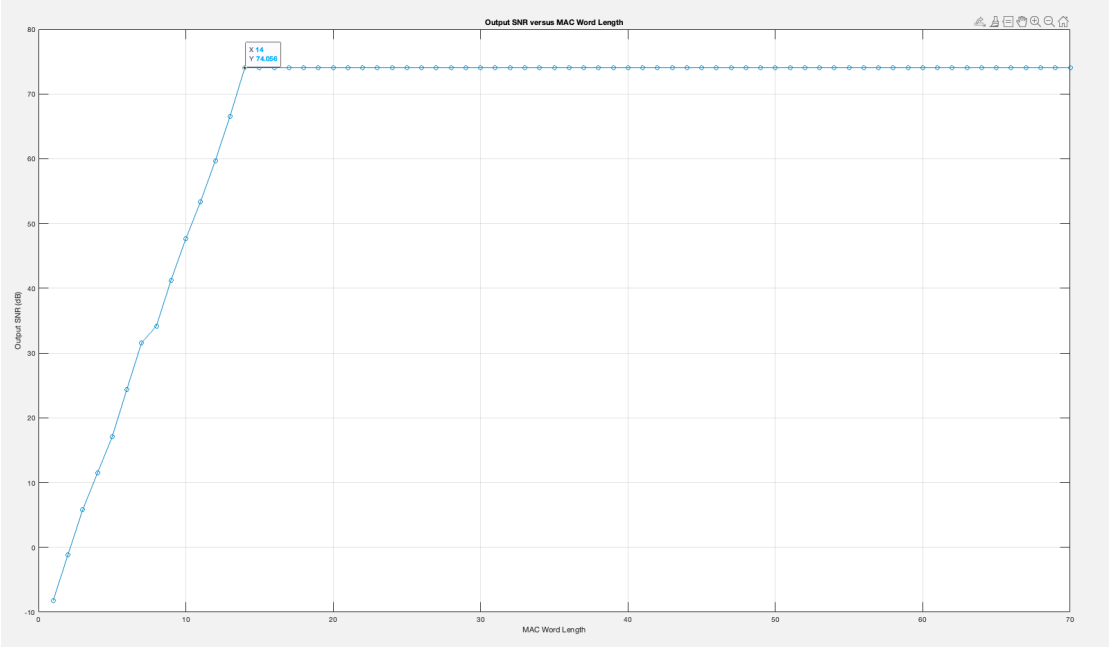


發現在小數點為 12bits 時開始 saturate，因此決定 fix point word-length 為 16bits，小數為 12bits

(2) 將 filter coefficient quantize 為 16bits word-length 的 fix point，小數位為 14bits，跟 floating point 的 filter coefficient 比較:

```
>> macWordLengths = 1:70;     % 例如：MAC字長的範圍
snrResultsMAC = zeros(size(macWordLengths));
for i = 1:length(macWordLengths)
macWordLength = macWordLengths(i);
% 模擬具有當前MAC字長的FIR濾波器
Qcoeff=fi(quantizecoeff,1,macWordLength+3 ,macWordLength 'RoundMode', 'floor');  ...
Qfilt=double(Qcoeff);  ...
outputSignal = filter(Qfilt, 1, input);  % X是您的輸入數據
powerOfDifference2=sum(abs(output-outputSignal).^2);  ...
snrResultsMAC(i) = 10*log10(Pfloat/powerOfDifference2);
end
% 绘制 SNR 与输入字长的关系图
figure;
plot(macWordLengths, snrResultsMAC, '-o');
xlabel('MAC Word Length');
ylabel('Output SNR (dB)');
title('Output SNR versus MAC Word Length');
grid on;
```

小數點位數



發現在小數點為 14bits 時開始 saturate，因此決定 MAC word-length 為 16bits，小數為 14bits

(c)

(i)Verilog code:

(1) fir.sv:

先將每個 coefficient 存在 reg tap 裡，coefficient 的形式根據上一題的結果，先將所有小數換成整數，例如：0.307250976562500 換成十進制(包含 sign bit)是 00.01001110101010，將小數點去掉後十六進制為 13AA。每個 tap 相乘完的結果兩兩相加，最後到 output 總共有 37bits，其中小數位為 26 bits(input 12bits+coefficient 14bits)，再將輸出結果設為 26bits 小數。

```
`timescale 1ns / 10ps

module FIR( //使用AXI stream interface
    input fir_clk,
    input signed [15:0] input_signal, //切成4bits 12bits
    output reg signed [36:0] filtered_signal //后輸出

    );
    // 21-tap FIR
    reg signed [15:0] buff0, buff1, buff2, buff3, buff4, buff5, buff6, buff7, buff8, buff9, buff10, buff11, buff12, buff13, buff14, buff15, buff16, buff17, buff18, buff19, buff20;
    wire signed [15:0] tap0, tap1, tap2, tap3, tap4, tap5, tap6, tap7, tap8, tap9, tap10, tap11, tap12, tap13, tap14, tap15, tap16, tap17, tap18, tap19, tap20;
    reg signed [31:0] acc0, acc1, acc2, acc3, acc4, acc5, acc6, acc7, acc8, acc9, acc10, acc11, acc12, acc13, acc14, acc15, acc16, acc17, acc18, acc19, acc20 ;
    reg signed [32:0] sum_0 [0:10];
    reg signed [33:0] sum_1 [0:5];
    reg signed [34:0] sum_2 [0:2];
    reg signed [35:0] sum_3 [0:1];
```

```
    //乘2^14 整數兩位 共16位
    /* Taps for LPF running @ 1MSps with a cutoff freq of 400kHz*/
    assign tap0 = 16'hFF52;  // -0.0106201171875000
    assign tap1 = 16'hFD91;  // -0.0380249023437500
    assign tap2 = 16'hFDA9;  // -0.0365600585937500
    assign tap3 = 16'h0037;  // 0.00335693359375000
    assign tap4 = 16'h02E9;  // 0.0454711914062500
    assign tap5 = 16'h013E;  // 0.0194091796875000
    assign tap6 = 16'hFC17;  // -0.0610961914062500
    assign tap7 = 16'hFB2E;  // -0.0753173828125000
    assign tap8 = 16'h04B4;  // 0.0734863281250000
    assign tap9 = 16'h13AA;  // 0.307250976562500
    assign tap10 = 16'h1B01; // 0.421936035156250
    assign tap11 = 16'h13AA;  // 0.307250976562500
    assign tap12 = 16'h04B4;  // 0.0734863281250000
    assign tap13 = 16'hFB2E;  // -0.0753173828125000
    assign tap14 = 16'hFC17;  // -0.0610961914062500
    assign tap15 = 16'h013E;  // 0.0194091796875000
    assign tap16 = 16'h02E9;  // 0.0454711914062500
    assign tap17 = 16'h0037;  // 0.00335693359375000
    assign tap18 = 16'hFDA9;  // -0.0365600585937500
    assign tap19 = 16'hFD91;  // -0.0380249023437500
    assign tap20 = 16'hFF52;  // -0.0106201171875000
```

```
53      always @ (posedge fir_clk)
54          begin
55                      buff0 <=  input_signal;
56                      buff1 <= buff0;
57                      buff2 <= buff1;
58                      buff3 <= buff2;
59                      buff4 <= buff3;
60                      buff5 <= buff4;
61                      buff6 <= buff5;
62                      buff7 <= buff6;
63                      buff8 <= buff7;
64                      buff9 <= buff8;
65                      buff10 <= buff9;
66                      buff11 <= buff10;
67                      buff12 <= buff11;
68                      buff13 <= buff12;
69                      buff14 <= buff13;
70                      buff15 <= buff14;
71                      buff16 <= buff15;
72                      buff17 <= buff16;
73                      buff18 <= buff17;
74                      buff19 <= buff18;
75                      buff20 <= buff19;
76          end
```

每一級 tap 的 delay

```
81      always @ (posedge fir_clk)
82          begin
83
84                      acc0 <= tap0 * buff0;
85                      acc1 <= tap1 * buff1;
86                      acc2 <= tap2 * buff2;
87                      acc3 <= tap3 * buff3;
88                      acc4 <= tap4 * buff4;
89                      acc5 <= tap5 * buff5;
90                      acc6 <= tap6 * buff6;
91                      acc7 <= tap7 * buff7;
92                      acc8 <= tap8 * buff8;
93                      acc9 <= tap9 * buff9;
94                      acc10 <= tap10 * buff10;
95                      acc11 <= tap11 * buff11;
96                      acc12 <= tap12 * buff12;
97                      acc13 <= tap13 * buff13;
98                      acc14 <= tap14 * buff14;
99                      acc15 <= tap15 * buff15;
100                     acc16 <= tap16 * buff16;
101                     acc17 <= tap17 * buff17;
102                     acc18 <= tap18 * buff18;
103                     acc19 <= tap19 * buff19;
104                     acc20 <= tap20 * buff20;
105         end
```

```
107     always @ (posedge fir_clk)
108         begin
109                     sum_0[0] <= acc0 + acc1;
110                     sum_0[1] <= acc2 + acc3;
111                     sum_0[2] <= acc4 + acc5;
112                     sum_0[3] <= acc6 + acc7;
113                     sum_0[4] <= acc8 + acc9;
114                     sum_0[5] <= acc10 + acc11;
115                     sum_0[6] <= acc12 + acc13;
116                     sum_0[7] <= acc14 + acc15;
117                     sum_0[8] <= acc16 + acc17;
118                     sum_0[9] <= acc18 + acc19;
119                     sum_0[10] <= acc20;
120         end
121 
122     always @ (posedge fir_clk)
123         begin
124                     sum_1[0] <= sum_0[0] + sum_0[1];
125                     sum_1[1] <= sum_0[2] + sum_0[3];
126                     sum_1[2] <= sum_0[4] + sum_0[5];
127                     sum_1[3] <= sum_0[6] + sum_0[7];
128                     sum_1[4] <= sum_0[8] + sum_0[9];
129                     sum_1[5] <= sum_0[10];
130         end
131 
132     always @ (posedge fir_clk)
133         begin
134                     sum_2[0] <= sum_1[0] + sum_1[1];
135                     sum_2[1] <= sum_1[2] + sum_1[3];
136                     sum_2[2] <= sum_1[4] + sum_1[5];
137         end
```

tap 的輸出兩兩相加

```
139     always @ (posedge fir_clk)
140         begin
141                     sum_3[0] <= sum_2[0] + sum_2[1];
142                     sum_3[1] <= sum_2[2];
143         end
144 
145 
146       /* Accumulate stage of FIR */
147     always @ (posedge fir_clk)
148         begin
149 //              filtered_signal <= acc0 + acc1 + acc2 + acc3 + acc4 + acc5 + acc6 + acc7 +
150 //          filtered_signal <= sum_4;
151             filtered_signal <= sum_3[0] + sum_3[1];
152 
153         end
154 
155 endmodule
```

(2) fir_tb.v:

在這裡使用 cordic IP 產生兩個分別為 1kHz 及 2kHz 的 sine wave，cordic clock 為 50MHz，取樣頻率 8kHz，input 用兩個 sine wave 相加表示，sine wave 為 15 bits，因此 input 最多可以用 16 bits 表示。

```
1  `timescale 1ns / 10ps
2
3  module tb_FIR;
4
5  localparam CORDIC_CLK_PERIOD = 20; // To create 50 MHz CORDIC sampling clock
6  //localparam FIR_CLK_PERIOD = 1250 ; // To create 100 MHz FIR lowpass filter sampling clock
7  localparam FIR_CLK_PERIOD = 125000 ; // To create 8KHz FIR lowpass filter sampling clock
8  localparam signed [15:0] PI_POS = 16'h6488; // +pi in fixed-point 1.2.13
9  localparam signed [15:0] PI_NEG = 16'h9B78; // -pi in fixed-point 1.2.13
10 localparam PHASE_INC_1KHZ = 1; // Phase jump for 1kHz sine wave synthesis
11 //localparam PHASE_INC_2KHZ = 3; // Phase jump for 2kHz sine wave synthesis 下降10倍
12 localparam PHASE_INC_2KHZ = 2; // Phase jump for 2kHz sine wave synthesis 下降10倍
13 reg cordic_clk = 1'b0;
14 reg fir_clk = 1'b0;
15 reg phase_tvalid = 1'b0;
16 reg signed [15:0] phase_1KHz = 0; // 1kHz phase sweep. 1.2.13
17 reg signed [15:0] phase_2KHz = 0; // 2kHz phase sweep. 1.2.13
18 wire sincos_1KHz_tvalid;
19 wire signed [15:0] sin_1KHz, cos_1KHz; // 1.1.14 1kHz sine/cosine
20 wire sincos_2KHz_tvalid;
21 wire signed [15:0] sin_2KHz, cos_2KHz; // 2.1.24 2kHz sine/cosine
22
23 reg signed [15:0] input_signal = 0; // Resampled 1kHz sine + 2kHz sine. 1.1.14
24 wire signed [36:0] filtered_signal; // Filtered signal output from FIR lowpass filter
```

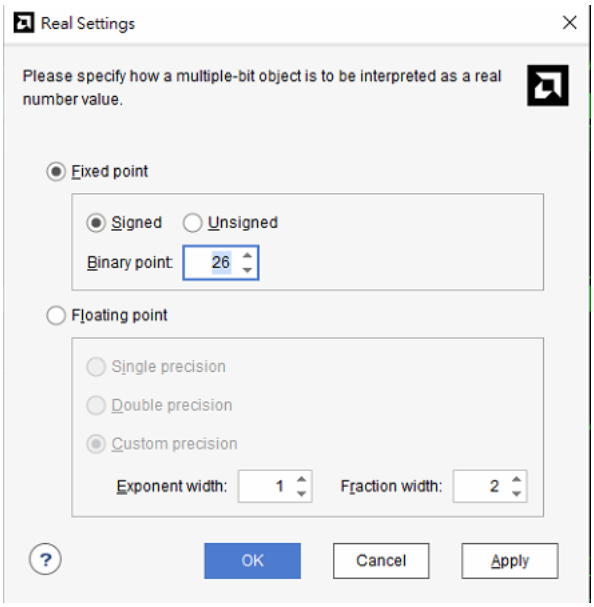
```
26 // Synthesize 1KHz sine
27 cordic_0 cordic_inst_0(
28     .aclk                (cordic_clk),
29     .s_axis_phase_tvalid (phase_tvalid),
30     .s_axis_phase_tdata  (phase_1KHz),
31     .m_axis_dout_tvalid  (sincos_1KHz_tvalid),
32     .m_axis_dout_tdata   ({sin_1KHz, cos_1KHz})
33 );
34 // Synthesize 2KHz sine
35 cordic_0 cordic_inst_1(
36   .aclk                (cordic_clk),
37   .s_axis_phase_tvalid (phase_tvalid),
38   .s_axis_phase_tdata  (phase_2KHz),
39   .m_axis_dout_tvalid  (sincos_2KHz_tvalid),
40   .m_axis_dout_tdata   ({sin_2KHz, cos_2KHz})
41 );
```

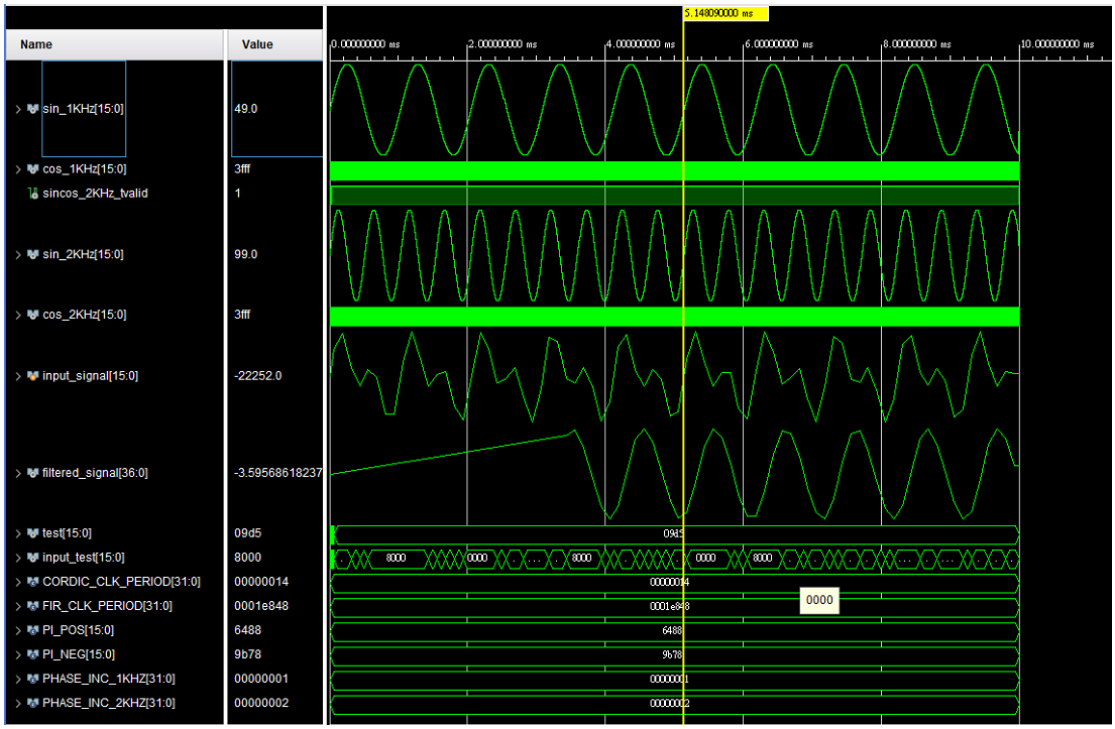
```
44 // Phase sweep
45 always @(posedge cordic_clk)
46 begin
47   phase_tvalid <= 1'b1;
48   // Sweep phase to synthesize 1kHz sine
49   if (phase_1KHz + PHASE_INC_1KHZ < PI_POS) begin
50     phase_1KHz <= phase_1KHz + PHASE_INC_1KHZ;
51   end
52   else
53   begin
54     phase_1KHz <= PI_NEG + (phase_1KHz+PHASE_INC_1KHZ - PI_POS);
55   end
```

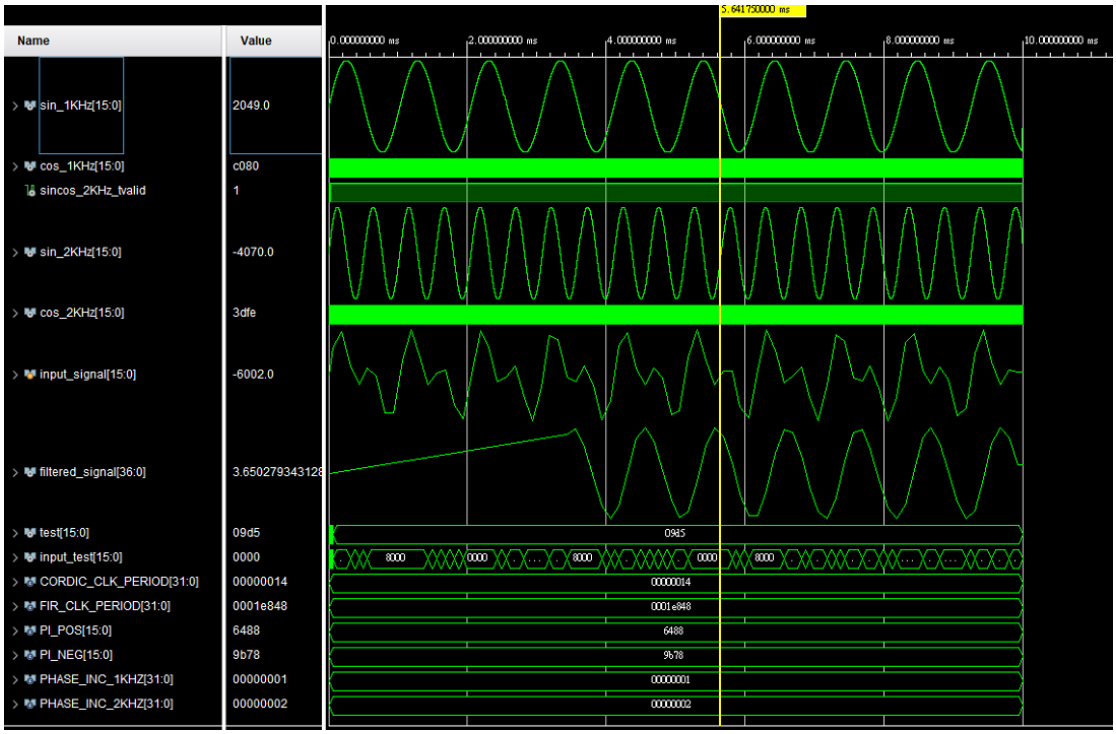
```
57      // Sweep phase to synthesize 2kHz sine
58      if (phase_2KHz + PHASE_INC_2KHZ <= PI_POS) begin
59        phase_2KHz <= phase_2KHz + PHASE_INC_2KHZ;
60      end
61      else
62      begin
63        phase_2KHz <= PI_NEG + (phase_2KHz+PHASE_INC_2KHZ - PI_POS);
64      end
65    end
66
67    // Create 50MHz Cordic clock 20ns
68    always begin
69    cordic_clk = #(CORDIC_CLK_PERIOD/2) ~cordic_clk;
70    end
```

```
75    always
76    begin
77    fir_clk = #(FIR_CLK_PERIOD/2) ~fir_clk;
78    end
79
80    always @(posedge fir_clk)
81    begin
82    // input_signal <= (sin_1KHz + sin_2KHz) / 2;
83    input_signal <= (sin_1KHz + sin_2KHz);
84    end
85
86        /* Instantiate FIR module to test. */
87        FIR FIR_i(
88            .clk(fir_clk),
89            .input_signal(input_signal),  //input
90            .filtered_signal(filtered_signal));
91
92
93    endmodule
```

(3) output waveform:




Name
Value
sin_1KHz[15:0]
49.0
cos_1KHz[15:0]
3fff
sincos_2KHz_tvalid
1
sin_2KHz[15:0]
99.0
cos_2KHz[15:0]
3fff
input_signal[15:0]
-22252.0
filtered_signal[36:0]
-3.59568618237
test[15:0]
09d5
input_test[15:0]
8000
CORDIC_CLK_PERIOD[31:0]
00000014
FIR_CLK_PERIOD[31:0]
0001e848
PI_POS[15:0]
6488
PI_NEG[15:0]
9b78
PHASE_INC_1KHZ[31:0]
00000001
PHASE_INC_2KHZ[31:0]
00000002
5.148090000 ms
0.000000000 ms
2.000000000 ms
4.000000000 ms
6.000000000 ms
8.000000000 ms
10.000000000 ms
0000


Name
Value
sin_1KHz[15:0]
2049.0
cos_1KHz[15:0]
c080
sincos_2KHz_tvalid
1
sin_2KHz[15:0]
-4070.0
cos_2KHz[15:0]
3dfe
input_signal[15:0]
-6002.0
filtered_signal[36:0]
3.650279343128
test[15:0]
09d5
input_test[15:0]
0000
CORDIC_CLK_PERIOD[31:0]
00000014
FIR_CLK_PERIOD[31:0]
0001e848
PI_POS[15:0]
6488
PI_NEG[15:0]
9b78
PHASE_INC_1KHZ[31:0]
00000001
PHASE_INC_2KHZ[31:0]
00000002
5.641750000 ms
0.000000000 ms
2.000000000 ms
4.000000000 ms
6.000000000 ms
8.000000000 ms
10.000000000 ms

(ii)circuit speed and resource number:

```
1 create_clock -period 8.000 -name fir_clk -waveform {0.000 4.000} [get_ports fir_clk]
2
```

Utilization — Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	213	53200	0.40
LUTRAM	40	17400	0.23
FF	382	106400	0.36
DSP	26	220	11.82
IO	54	125	43.20
BUFG	1	32	3.13

Power — **Summary** | On-Chip

Total On-Chip Power:	**0.188 W**
Junction Temperature:	**27.2 °C**
Thermal Margin:	57.8 °C (4.8 W)
Effective ϑJA:	11.5 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Implemented Power Report

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	4.011 ns	Worst Hold Slack (WHS):	0.047 ns	Worst Pulse Width Slack (WPWS):	3.020 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	1826	Total Number of Endpoints:	1826	Total Number of Endpoints:	489

All user specified timing constraints are met.