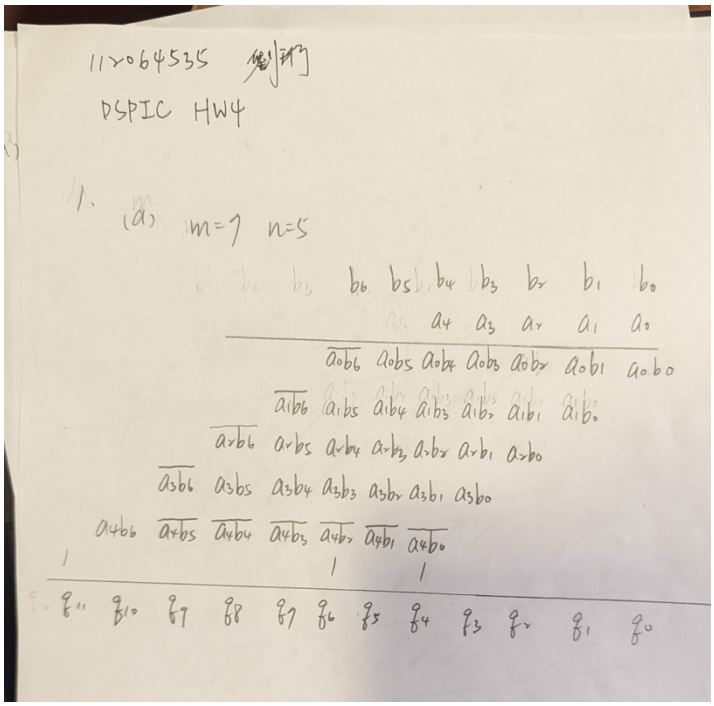
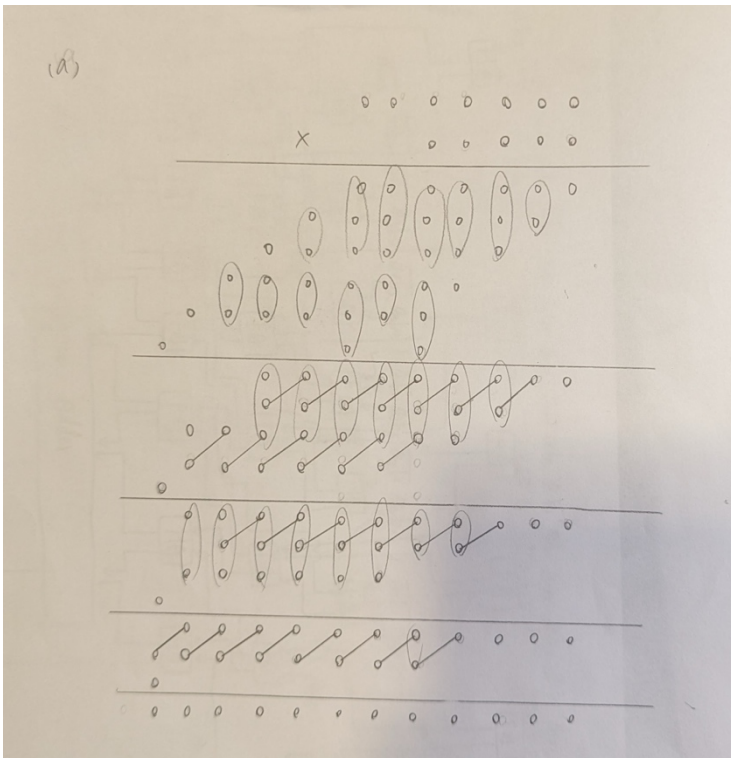
DSPIC HW4

學號：112064535
姓名：劉珩

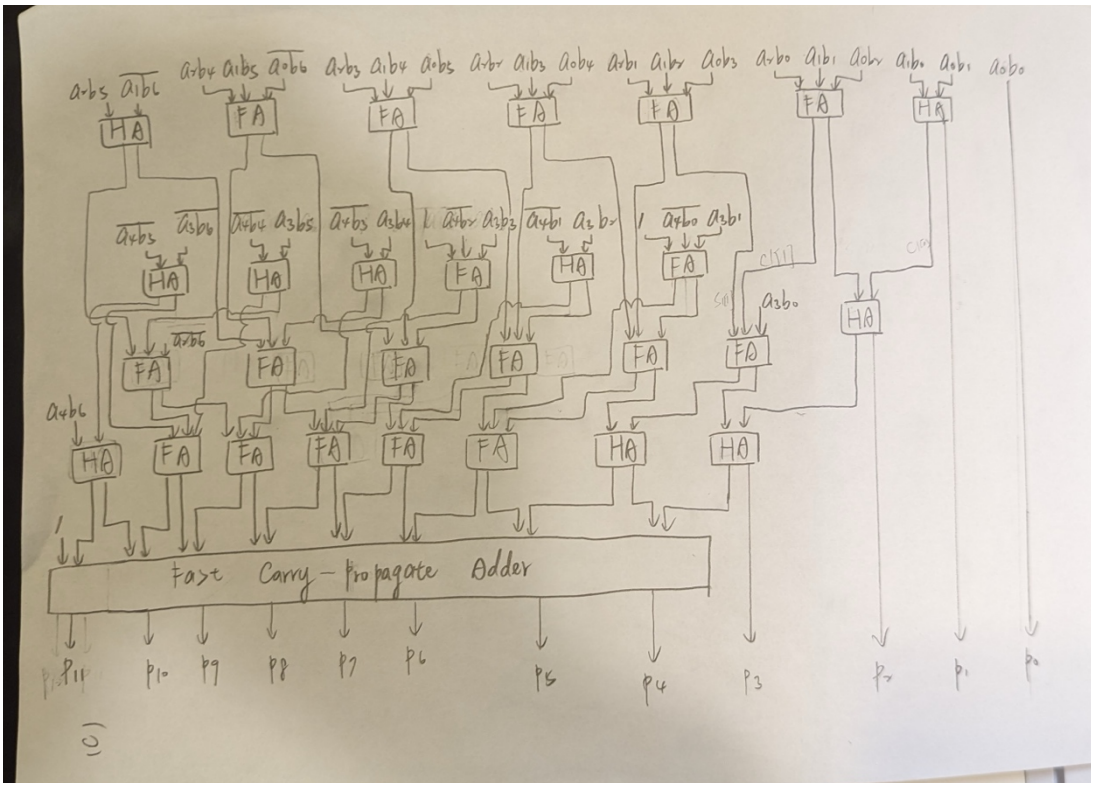
1)




(圖一) 7*5 乘法器結構




(圖二) 運作流程（包含 18 個 FA, 11 個 HA）




(圖三) Wallance-Tree 結構

Verilog code:

1.把加法器分成四層，第一層有兩個部分：

first1.v

```v
`include "fulladd.v"
`include "halfadder.v"

module first1(
    input  [5:0] a, //a有6個 a0b1~a0b6
    input  [6:0] b, //a1b0~a1b6
    input  [5:0] cin, // a2b0~a2b5
    output [6:0] cout, //7個adder
    output [6:0] sum //同上
);


 half_add add1(
   .A(a[0]),
   .B(b[0]),
   .sum(sum[0]), //p1
   .cout(cout[0])
);

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

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

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

    Full_Adder add5(
    .A(a[4]),
    .B(b[4]),
    .Cin(cin[3]),
    .Cout(cout[4]),
    .Sum(sum[4])
  );

    Full_Adder add6(
    .A(a[5]),
    .B(b[5]),
    .Cin(cin[4]),
    .Cout(cout[5]),
    .Sum(sum[5])
  );

   half_add add7(
    .A(b[6]),
    .B(cin[5]),
    .sum(sum[6]),
    .cout(cout[6])
   );


endmodule
```

first2.v

```v
// `include "fulladd.v"
// `include "halfadder.v"

module first2(
    input  [5:0] a, //a有6個 a3b1~a3b6
    input  [5:0] b, //a4b0~a4b5
    //cin都是1 不需要設輸入
    output [5:0] cout, //7個adder
    output [5:0] sum //同上
);


  Full_Adder add1(
    .A(a[0]),
    .B(b[0]),
    .Cin(1'b1),
    .Sum(sum[0]),
    .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(1'b1),
    .Cout(cout[2]),
    .Sum(sum[2])
  );

    half_add add4(
    .A(a[3]),
    .B(b[3]),
    .cout(cout[3]),
    .sum(sum[3])
  );

    half_add add5(
    .A(a[4]),
    .B(b[4]),
    .cout(cout[4]),
    .sum(sum[4])
  );
```

```v
    half_add add6(
    .A(a[5]),
    .B(b[5]),
    .cout(cout[5]),
    .sum(sum[5])
  );


endmodule
```

2.第二層：

second.v

```v
// `include "fulladd.v"
// `include "halfadder.v"

module second(
    input  [6:0] a, //a有6個 從first1的第一個ha到a2b6
    input  [6:0] b,
    input  [5:0] cin,
    output [6:0] cout, //7個adder
    output [6:0] sum //同上
);


  half_add add1(
    .A(a[0]),
    .B(b[0]),
    .sum(sum[0]), //p2
    .cout(cout[0])
);

  Full_Adder add2(
    .A(a[1]),
    .B(b[1]),
    .Cin(cin[0]), //a3b0
    .Cout(cout[1]),
    .Sum(sum[1])
  );

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

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

```v
    Full_Adder add5(
    .A(a[4]),
    .B(b[4]),
    .Cin(cin[3]),
    .Cout(cout[4]),
    .Sum(sum[4])
  );

    Full_Adder add6(
    .A(a[5]),
    .B(b[5]),
    .Cin(cin[4]),
    .Cout(cout[5]),
    .Sum(sum[5])
  );

   Full_Adder add7(
    .A(a[6]),
    .B(b[6]),
    .Cin(cin[5]),
    .Cout(cout[6]),
    .Sum(sum[6])
   );


endmodule
```

3.第三層：

```
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 propagation)：

```
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])
  );
```

```
    Full_Adder add5(
    .A(a[4]),
    .B(b[4]),
    .Cin(w[3]),
    .Cout(w[4]),
    .Sum(sum[4])
   );

    Full_Adder add6(
    .A(a[5]),
    .B(b[5]),
    .Cin(w[4]),
    .Cout(w[5]),
    .Sum(sum[5])
   );

    Full_Adder add7(
    .A(a[6]),
    .B(b[6]),
    .Cin(w[5]),
    .Sum(sum[6]),
    .Cout(w[6])
   );

    Full_Adder add8(
    .A(a[7]),
    .B(b[7]),
    .Cin(w[6]),
    .Sum(sum[7]),
    .Cout(cout)
   );


endmodule
```

5.在 top module 裡把所有 a 跟 b 乘好在做加法：

```
genvar i,j;
generate
        for(i=0;i<4;i=i+1) begin
            for(j=0;j<6;j=j+1) begin
                assign ab[i][j]=a[i]&b[j];
            end
        end
endgenerate

generate
    // genvar i;
    for(i=0;i<4;i=i+1) begin
        assign ab[i][6]=~(a[i]&b[6]);
    end
endgenerate

generate
    // genvar j;
    for(j=0;j<6;j=j+1)begin
        assign ab[4][j]=~(a[4]&b[j]);
    end
endgenerate

    assign ab[4][6]=a[4]&b[6];
```

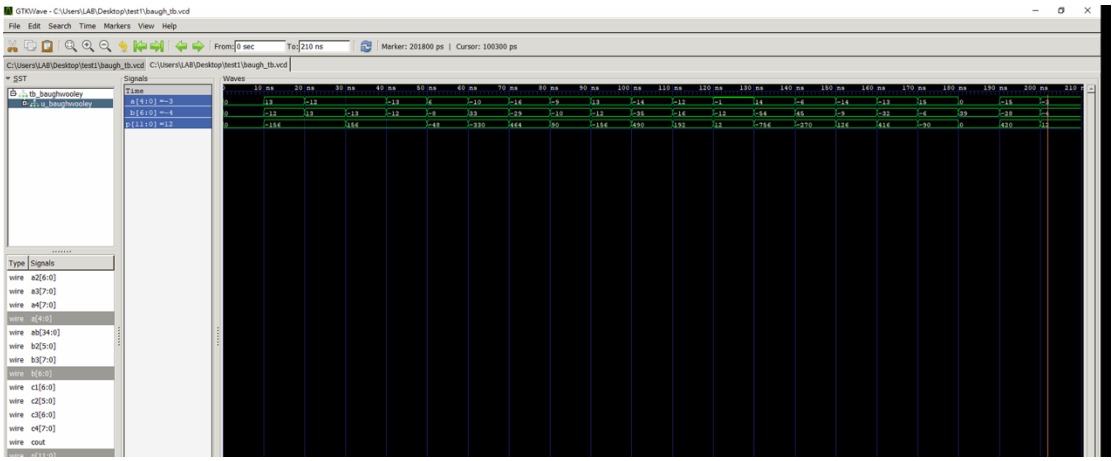
```
  first1 adder1(
      .a(ab[0][6:1]),
      .b(ab[1][6:0]),
      .cin(ab[2][5:0]),
      .cout(c1),
      .sum(s1)
  );

      assign a2[5:0]=s1[6:1];
      assign a2[6]=ab[2][6]; //定義second的輸入

  first2 adder2(
      .a(ab[3][6:1]),
      .b(ab[4][5:0]),
      .cout(c2),
      .sum(s2)
  );
      assign b2[5:1]=s2[4:0];
      assign b2[0]=ab[3][0];//定義second的輸入 等等不行就把assign拿掉

  second adder3(
      .a(a2), //a2b6+s1[6:1]
      .b(c1),
      .cin(b2), //s2[4:0]+a3b0
      .cout(c3),
      .sum(s3)
  );

      assign a3[7]=ab[4][6];
      assign a3[6]=s2[5];
      assign a3[5:0]=s3[6:1];

      assign b3[7]=c2[5];
      assign b3[6:0]=c3[6:0];
  third adder4(
      .a(a3), // a4b6+s2[5]+s3[6:1]
      .b(b3), // c2[5]+c3[6:0]
      .cin(c2[4:0]), //c2[4:0]
      .cout(c4),
      .sum(s4)
  );
```

```
111         assign a4[7]=1'b1;
112         assign a4[6:0]=s4[7:1];
113     fast adder5(
114         .a(a4), // 1'b1+s4[7:1]
115         .b(c4), //c4[7:0]
116         .cout(cout),
117         .sum(p[11:4])
118     );
119
120     assign p[0]=ab[0][0];
121     assign p[1]=s1[0];
122     assign p[2]=s3[0];
123     assign p[3]=s4[0];
124
125 endmodule
```

5.testbench 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
```

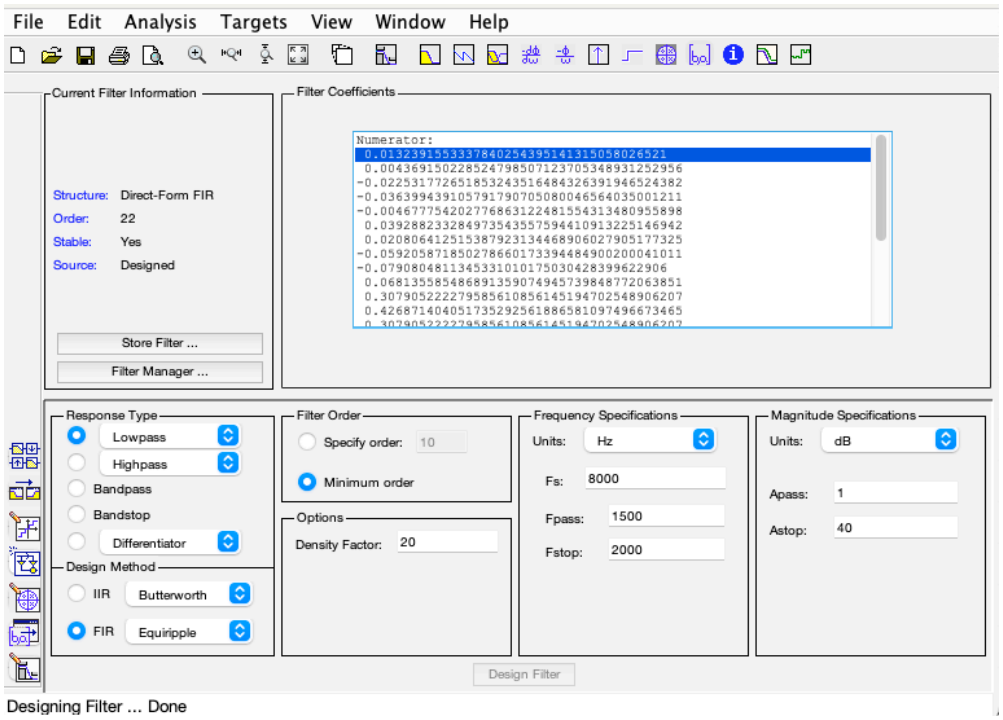


2)

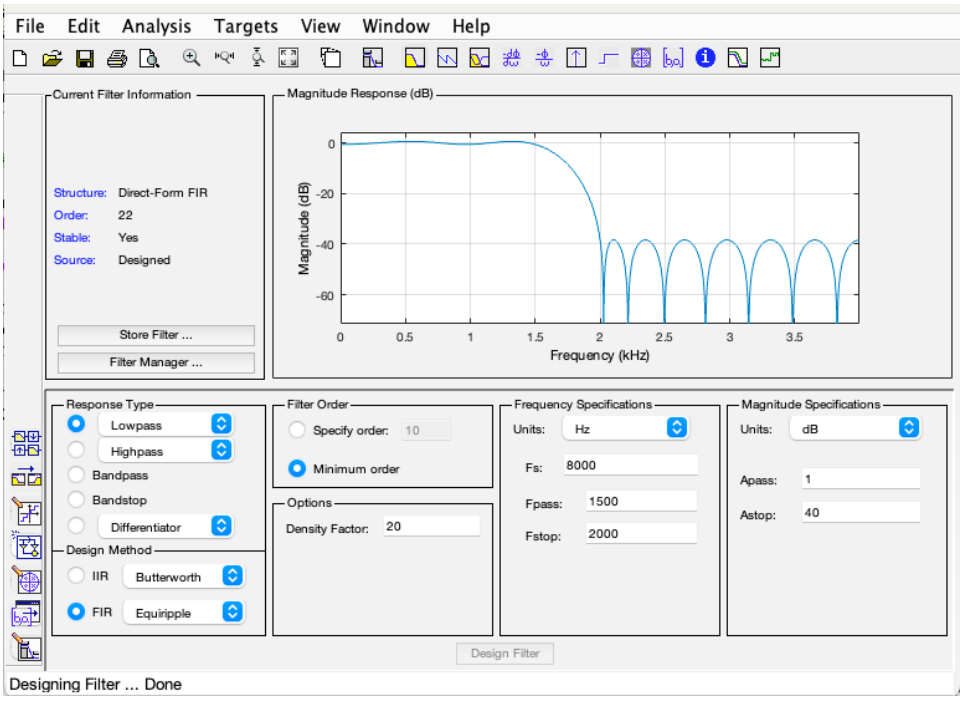
(a)

(i)在matlab的filter designer輸入欲產生的filter constraints，選擇minimum order，將產生order為22的FIR low pass filter corfficients:

0.0132391553337840
0.00436915022852480
-0.0225317726518532
-0.0363994391057918
-0.00467775420277686
0.0392882332849735
0.0208064125153879
-0.0592058718502787
-0.0790804811345331
0.0681355854868914
0.307905222279586
0.426871404051735
0.307905222279586
0.0681355854868914
-0.0790804811345331
-0.0592058718502787
0.0208064125153879
0.0392882332849735
-0.00467775420277686
-0.0363994391057918
-0.0225317726518532
0.00436915022852480
0.0132391553337840


File Edit Analysis Targets View Window Help
Current Filter Information
Structure: Direct-Form FIR
Order: 22
Stable: Yes
Source: Designed
Store Filter ...
Filter Manager ...
Filter Coefficients
Numerator:
0.013239155333784025439514131505802652 1
0.004369150228524798507123705348931252956
-0.022531772651853243516484326391946524382
-0.036399439105791790705080046564035001211
-0.004677754202776863122481554313480955898
0.039288233284973543557594410913225146942
0.020806412515387923134468906027905177325
-0.059205871850278660173394484900200041011
-0.079080481134533101017503042839622906
0.068135585486891359074945739848772063851
0.307905222279585610856145194702548906207
0.426871404051735292561886581097496673465
0.307905222279585610856145194702548906207
Response Type
Lowpass
Highpass
Bandpass
Bandstop
Differentiator
Design Method
IIR Butterworth
FIR Equiripple
Filter Order
Specify order: 10
Minimum order
Options
Density Factor: 20
Frequency Specifications
Units: Hz
Fs: 8000
Fpass: 1500
Fstop: 2000
Magnitude Specifications
Units: dB
Apass: 1
Astop: 40
Design Filter
Designing Filter ... Done

(ii)


File Edit Analysis Targets View Window Help
Current Filter Information
Structure: Direct-Form FIR
Order: 22
Stable: Yes
Source: Designed
Store Filter ...
Filter Manager ...
Magnitude Response (dB)
Magnitude (dB)
0
-20
-40
-60
0 0.5 1 1.5 2 2.5 3 3.5
Frequency (kHz)
Response Type
Lowpass
Highpass
Bandpass
Bandstop
Differentiator
Design Method
IIR Butterworth
FIR Equiripple
Filter Order
Specify order: 10
Minimum order
Options
Density Factor: 20
Frequency Specifications
Units: Hz
Fs: 8000
Fpass: 1500
Fstop: 2000
Magnitude Specifications
Units: dB
Apass: 1
Astop: 40
Design Filter
Designing Filter ... Done

(b)

(i)c 的內容及輸出的結果:

C 2b_floating.c > main()

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

#define NUM_SAMPLES 1000
// #define N 100  // 輸入信号长度
#define M 23   // 滤波器阶数


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;
}


// 函数原型
void firLowPassFilter(float input[], float output[], float filterCoeff[], int inputLength, int filterLength);
// int OUT_NUM=NUM_SAMPLES+M-1; //output數量

int main() {
    float inputSignal[NUM_SAMPLES]; // 輸入信号
    float outputSignal[NUM_SAMPLES]; // 输出信号
    float filterCoeff[M] = {0.0132391553337840, 0.00436915022852480, -0.0225317726518532, -0.0363994391057918, -0.00467775420277686, 0.0392882332849735, 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;
```

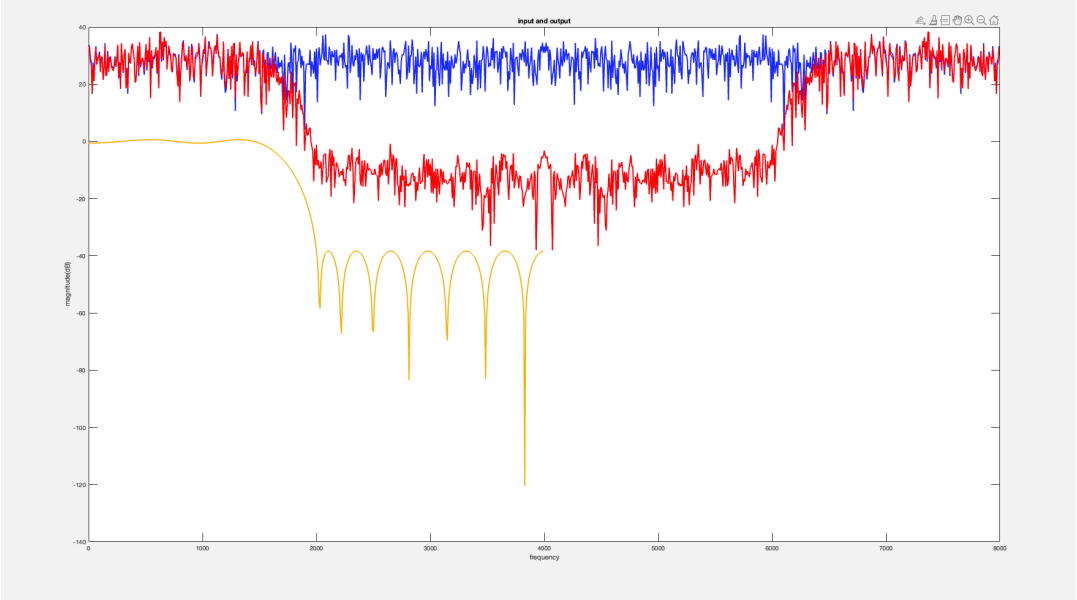
問題 輸出 偵錯主控台 終端機 連接埠 Code

```
Generated signal with average power = 1.000001
Input Signal: 0.474154 -1.540156 0.378412 -0.433787 0.553478 0.767403 0.858489 -0.371457 -0.836116 0.752850 -0.997900 1.693951 1.227352 1.292462 -0.156509 -0.363907 1.559188
-0.412348 0.312278 -1.431685 0.050416 -0.564296 -1.180776 -0.447702 -0.629838 0.718327 -1.619364 0.963984 -0.554386 1.508406 -1.206132 -0.927347 -0.694049 0.251645 0.842433
-0.113828 -0.919785 -1.259677 0.095079 1.424272 -1.186913 -1.285354 -1.490730 0.156284 0.744936 0.613609 -1.602506 -1.027262 -1.673569 0.290727 -1.211956 -0.710707 0.556921
1.550450 1.629125 1.704022 1.653345 -1.155932 -0.792301 0.354834 0.992447 0.718368 1.305866 -0.653361 1.474443 0.698846 -0.686322 -0.938817 -1.075585 1.425498 -1.713562 1.
441740 0.509642 -1.469325 0.126490 -0.353920 -0.774848 -0.046323 0.194021 -0.754432 1.490287 -0.642461 -1.641899 1.524752 0.789521 1.275176 -1.283649 -1.078628 1.264809 -0.
433760 0.765855 -1.475573 1.133619 1.167680 0.117261 -0.343499 -0.703566 1.051232 0.630784 -0.785755 -0.005368 -1.634483 -1.203846 0.543919 1.711367 1.228579 -0.249372 0.
161303 0.529934 -1.536879 0.982486 -1.599395 -1.687016 -1.035830 -0.672223 -0.083773 1.161582 1.457454 1.346774 -0.734289 1.165246 -1.361478 1.493380 0.099376 1.004472 1.
326184 -0.197581 0.247951 -0.061232 0.063836 0.879273 0.842304 -0.074425 1.159916 -0.505781 -0.658666 -0.687993 0.435077 1.184578 -1.146407 -0.232883 0.684031 -1.559837 1.
099584 0.262944 1.003932 0.933526 1.064200 0.150037 -1.096060 -1.333183 1.699305 -1.551993 -0.429485 -1.092478 1.351415 0.431969 -1.269718 0.276853 -1.087796 0.497285 -1.
422735 -1.392829 1.664875 -0.020699 -0.675512 -0.697524 1.544879 -0.952397 0.284349 -0.460934 0.566796 0.312930 -1.318236 1.711653 -1.668982 1.218271 -0.829473 0.241211 -1.
515955 0.913485 -0.742509 1.707635 0.247928 -0.517808 -0.753533 -1.301709 1.582466 0.150955 -0.057828 1.584326 0.108619 0.542330 -1.580657 -1.688588 -0.867958 -1.315733 -0.
841491 0.449964 -1.697416 1.456443 0.611056 -1.471597 1.655064 -0.847361 0.131852 -1.478826 -0.129182 1.496039 0.286026 1.456854 0.380794 1.312705 0.892105 0.309439 1.462657
-0.954316 -0.474296 0.386474 -1.464608 0.501731 -1.022459 0.915491 0.463273 -1.525417 -1.575167 0.978514 -1.576899 1.307522 1.183662 -0.969344 -0.431284 -1.663212 -1.310709
0.330068 -0.528696 1.479791 -1.483453 0.588342 1.114136 0.713913 -0.752194 -0.155851 -0.389251 -0.847819 -0.149790 0.580775 1.502592 -1.059864 1.167482 1.419595 -0.031000 1.
007842 -0.820578 1.154098 1.655561 -0.581612 -1.018968 1.203781 -0.568750 1.198009 0.498872 1.445931 0.381412 -0.499941 -1.025007 0.807564 0.098370 -1.105694 0.772353 1.
567614 0.321615 -1.348296 0.200300 1.401994 -1.679269 1.014903 0.186843 1.009075 1.627140 0.100541 -1.051636 -1.186351 -0.385680 1.376759 1.328464 -0.141140 0.558677 1.
135735 1.234553 1.202885 -1.273145 -0.036614 -0.117653 -1.682585 1.494619 1.534664 0.006425 -0.205129 -0.134027 1.470887 1.660421 0.015370 -0.798584 1.672248 -1.092981 -1.
230345 -1.169859 -0.850654 0.206434 -1.347583 -0.093101 1.382707 1.229619 -0.525462 -0.485955 -0.310219 1.164630 1.168933 -0.848552 -1.226603 -0.807680 -1.275147 0.243836 -0.
314031 -0.138903 1.090940 -0.240475 -1.653846 1.251441 1.301622 1.444911 1.077617 1.362220 -0.018314 0.222568 -0.183262 0.326412 -1.292383 0.162133 1.561617 0.400152 0.
949012 -1.156775 0.869195 0.384829 -1.050354 0.161401 -0.845791 0.704198 -0.593082 0.564103 -0.137085 1.028263 -0.618349 0.780823 -1.172286 -0.347598 1.617362 0.036239 -1.
501143 1.584418 1.585608 0.347519 -0.321037 1.410003 1.366704 0.040880 -1.647696 1.267606 0.366614 -0.830275 -1.025673 1.019827 0.058367 -0.529075 -1.704115 -1.626713 -0.
699443 -0.335177 -0.397306 -0.851982 0.214579 -1.335568 -1.099001 -0.963021 -1.207092 -1.063422 0.968416 -0.594500 -1.095509 0.671975 -0.625009 0.684935 0.989396 -1.622900 0.
983198 -0.578595 0.503821 1.043144 1.576568 0.702302 -1.262504 0.226573 -0.468105 -1.115570 -0.247903 1.303839 0.296298 -0.966810 -1.015925 0.105098 0.657291 0.920736 0.
026876 -1.667627 0.719246 -1.397265 -0.449157 0.901098 -0.867218 -1.200792 -0.988583 0.515676 1.536937 1.085807 1.225513 -1.294061 0.731809 -0.745797 1.127860 -0.238623 0.
194872 -0.787147 -1.701699 0.736241 -0.665582 -1.109379 0.470323 0.933499 -0.163755 -1.509071 0.735997 -1.012868 -1.524399 -1.515084 -1.658116 0.775638 -1.121624 -0.474949 0.
752569 1.163003 -1.131872 -0.168650 -1.399664 1.644513 -1.254667 0.004977 -0.178929 1.471909 0.836705 1.543209 -1.304815 -0.502791 -0.086326 0.156449 0.117628 1.664389 0.
263398 -1.053183 -1.710993 -0.474310 0.500700 1.145110 1.120951 -1.403281 -1.046865 -0.446690 -0.077615 0.880566 1.313719 -1.376981 1.483012 0.027814 0.837679 -0.645243 0.
479078 -0.834116 -1.310228 1.299192 -1.618563 1.616707 -1.527939 0.145912 -1.587256 -1.140617 0.596582 -0.261127 0.700277 -0.491693 -1.677355 -0.869574 0.084619 0.921516 0.
080021 0.804275 -1.318556 -1.342386 -0.231716 -0.965730 -1.567142 -1.588510 -0.632484 -1.686406 1.269157 1.673983 1.626292 -1.451068 0.693232 1.244697 0.138770 0.373051 -0.
359719 0.414320 0.542161 1.161631 -0.161816 1.084029 1.627664 -1.507292 1.601336 1.152918 -1.552563 1.315654 1.560505 1.589129 -0.550968 -1.165028 1.037701 0.012595 -0.
097606 0.558217 -0.946847 -0.956398 -0.792632 -0.845677 -0.538043 1.408207 -0.201408 -0.165988 -0.365939 -0.980010 0.368536 -0.960918 0.082368 0.243407 1.588199 0.381509 0.
868239 -1.237015 0.871609 1.352462 -1.357323 -0.319259 1.676919 0.926456 -0.389224 -0.538934 0.550769 -0.393948 -0.579431 0.025559 -1.431488 -1.593407 -1.177164 1.667534 -1.
```

在 matlab 上驗證平均功率為 1:

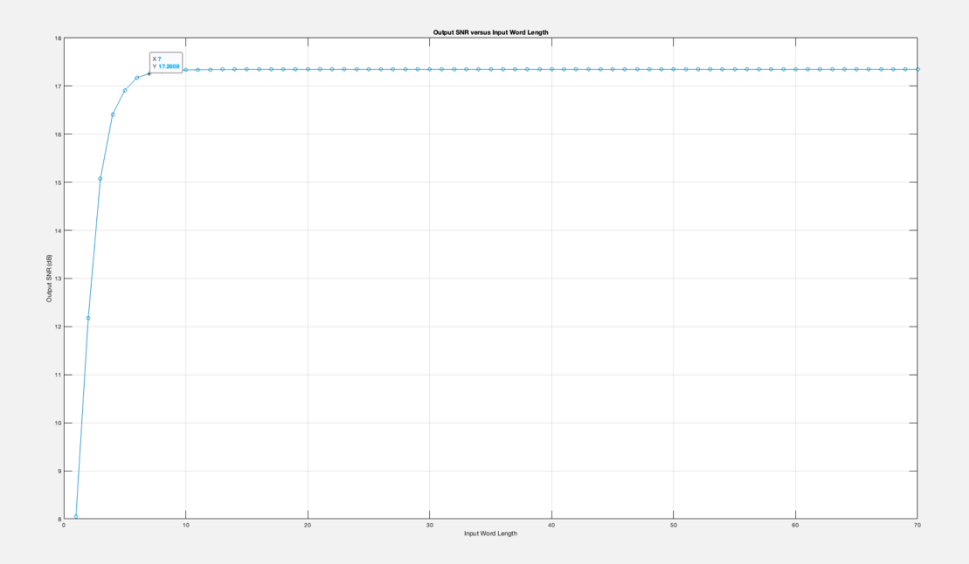
```
>> mean(abs(input3).^2)

ans =

    1.0000
```

Filter 輸出結果：

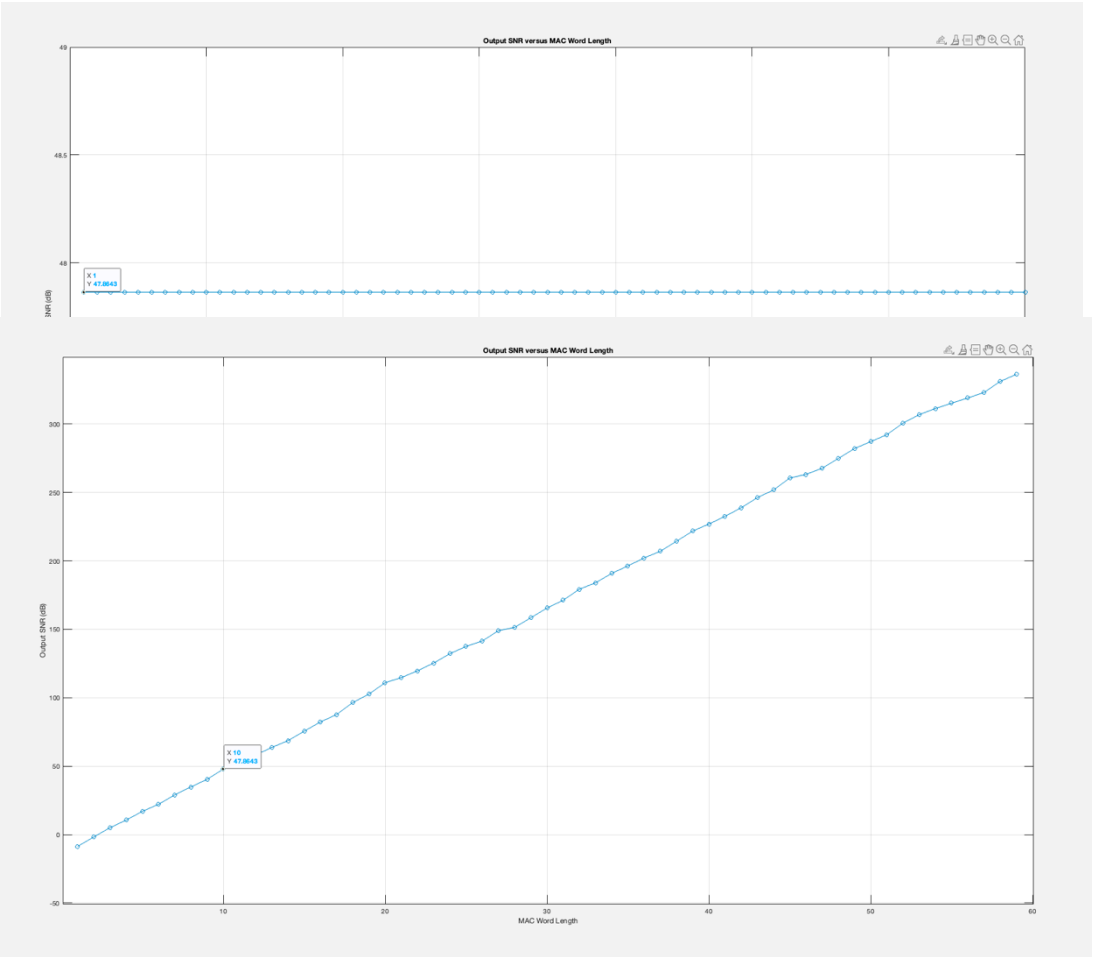



(ii)將 input word-length 整數字長可接受範圍定為 2，小數部分大約在 word length 為 7 的時候 saturate，定為可接受的 word-length

```
>> 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(input,1,inputWordLength+2 , inputWordLength,'RoundMode', 'floor');
IN=double(INSig);
OUTSig=filter(filtcoeff,1,IN);
powerOfDifference=mean(abs(output-OUTSig).^2)*length(output);
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;
```




可以發現，MAC 的 word-length 整數部分無論為何，都在 saturation，但小數部分隨著 word-length 增加 SNR 會一直持續增至長，因此整數定為 1bit，小數定為 16bits：



3. 第三題尚未完成...