

哈尔滨工业大学（威海）

《EDA 技术高级应用》

课 程 报 告

调 研	优 <input type="checkbox"/>	良 <input type="checkbox"/>	中 <input type="checkbox"/>	差 <input type="checkbox"/>	成 绩
设 计	优 <input type="checkbox"/>	良 <input type="checkbox"/>	中 <input type="checkbox"/>	差 <input type="checkbox"/>	
结 果	优 <input type="checkbox"/>	良 <input type="checkbox"/>	中 <input type="checkbox"/>	差 <input type="checkbox"/>	
报 告	优 <input type="checkbox"/>	良 <input type="checkbox"/>	中 <input type="checkbox"/>	差 <input type="checkbox"/>	
评 语					

题 目：非相参积累算法仿真实现

学 号：23S030123

姓 名：任笑雨

信息科学与工程学院

2024 年 6 月

课程报告任务书

单位	信息科学与工程学院	专业	信息与通信工程
课程	EDA 技术高级应用	年级	2023 级研究生
设计题目	非相参积累算法仿真实现		
设计环境	<div>1. 仿真平台：Quartus II 13.0 以上；</div> <div>2. 程序描述：顶层设计使用图形模块连线搭建，底层功能模块均使用 VHDL 语言编写；</div> <div>3. 关键程序行和参数设置处均需标明注释；</div> <div>4. 功能仿真；</div>		

设计要求	<p>设计一个浮点非相参积累算法运算器，具体要求如下：</p> <ol style="list-style-type: none">1. 输入和输出均为 IEEE 的 64 位浮点有符号数格式；2. 脉冲积累个数为 8 个；3. 使用 VHDL 编程；4. 进行功能仿真；5. 输入的时域采样数据（带高斯白噪声，信噪比要求合理，脉宽 10us，采样率 10MHz，脉冲信号幅度范围 0~5V）自行使用 MATLAB 仿真生成；6. 对比电脑 MATLAB 仿真的运算结果和 FPGA 运算器的运算结果，对结果精度进行对比分析。
------	--

设计目的

设计一个浮点非相参积累算法运算器，具体要求如下：

1. 输入和输出均为 IEEE 的 64 位浮点有符号数格式；
2. 脉冲积累个数为 8 个；
3. 使用 VHDL 编程；
4. 进行功能仿真；
5. 输入的时域采样数据（带高斯白噪声，信噪比要求合理，脉宽 10us，采样率 10MHz，脉冲信号幅度范围 0~5V）自行使用 MATLAB 仿真生成；
6. 对比电脑 MATLAB 仿真的运算结果和 FPGA 运算器的运算结果，对结果精度进行对比分析。

参考标准

- "IEEE Standard for Floating-Point Arithmetic," in IEEE Std 754-2019 (Revision of IEEE 754-2008) , vol., no., pp.1-84, 22 July 2019, doi: 10.1109/IEEESTD.2019.8766229. keywords: {IEEE Standards;Floating-point arithmetic;arithmetic;binary;computer;decimal;exponent;floating-point;format;IEEE 754;interchange;NaN;number;rounding;significand;subnormal.},

设计总体方案

本实验需要先使用 MATLAB 仿真生成雷达回波的数据，然后使用非相参算法处理数据，计算使用非相参算法后结果的信噪比增益。

完成 MATLAB 处理后，使用 Quartus 进行 FPGA 平台的代码编写（使用 VHDL 语言）和仿真。该过程主要包括对非相参算法，双精度浮点数加法器、ROM 模块等设计，并输出 Modelsim 仿真结果。然后，回到 MATLAB 中，完成 Modelsim 结果波形的重建以及计算其信噪比增益。

最后，比较 MATLAB 和 FPGA 编写的非相参算法仿真结果的信噪比增益，并得出结论。

设计参数指标

1. FPGA 加法器的精度

使用 VHDL 语言编写的 64 位双精度 double 加法器算法是完成非相参积累算法的重要算法，所以，加法器算法的精度是本实验设计的重要参数指标。

2. 非相参累积信噪比增益

实验结果需要比较 MATLAB 和 FPGA 仿真代码的波形结果，计算出各自的信噪比增益，从而评价本文的 VHDL 编写的非相参积累算法的仿真结果的优劣。

使用的设计工具

- Quartus (Quartus Prime 21.1)
- MATLAB (2020a)
- Modelsim (SE-64 10.1c)
- VSCODE

设计方法及步骤

本实验主要是对比 MATLAB 和 FPGA 两种代码计算的结果的精度，通过对比实验结果得出对 FPGA 代码的一个评价。

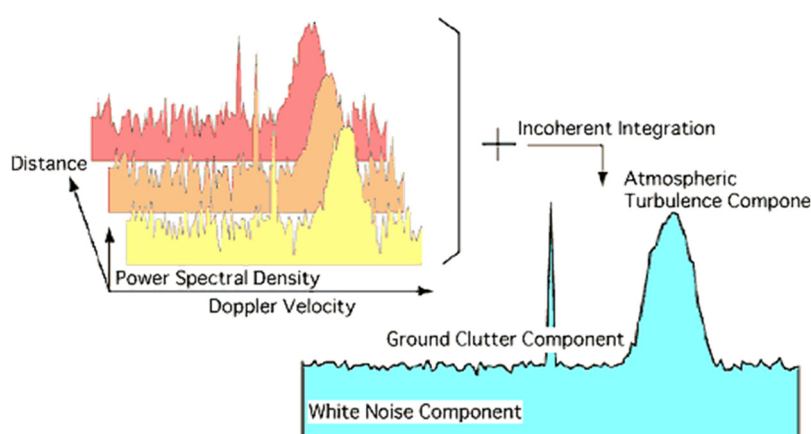
1、非相参积累算法

非相参积累算法往往在雷达信号中的弱信号处理。其方法是仅考虑信号的幅度，将多周期回波进行累加。其作用是提高信噪比。

下面以雷达接收的回波信号举例，来简单介绍该算法的使用。

雷达接收的信号如下图所示，中间的尖峰为地杂波，即周围区域的静态散射体（山，大型建筑等）反射的信号。右边的宽峰为被大气湍流散射的信号。

回波中一般包含大量噪声，大气散射分量往往被噪声所掩盖。通过使用非相参积累，可以提高信噪比，从而区分出目标信号。其算法是将回波信号的多周期信号叠加求和取均值，最终得到信噪比提高后的波形。



非相参积累算法示例雷达波形

2、MATLAB 的非相参积累算法实现方法

利用 MATLAB 进行非相参积累算法仿真，主要步骤如下：

1. 生成一个周期的理想回波信号；
2. 对 1 中的理想回波添加指定参数的高斯白噪声；
3. 对 2 中的带噪回波分析信噪比；
4. 生成 8 个周期的带噪回波，并使用非相参算法计算结果；
5. 分析非相参算法结果的信噪比；
6. 结合 5 和 3，分析非相参累积增益。

2.1 模拟回波信号的生成与信噪比分析

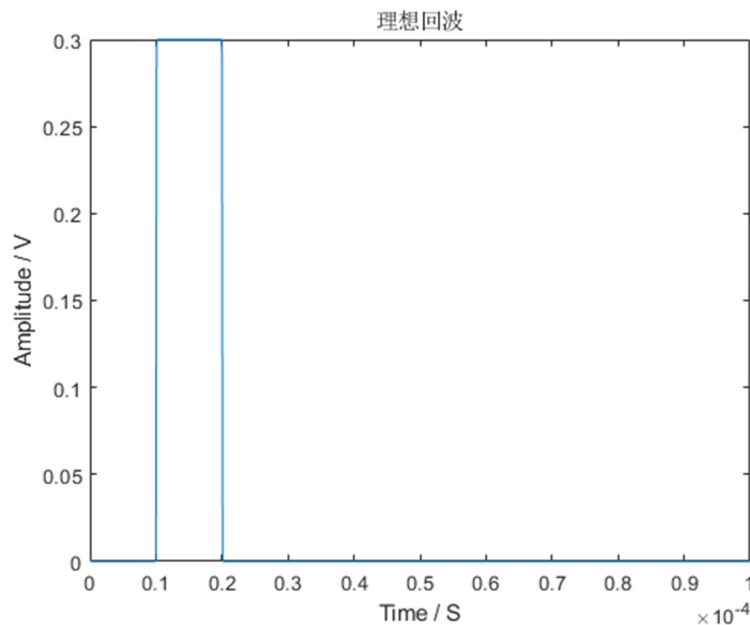
对雷达回波使用非相参积累算法处理前，需要先进行理想回波的模拟生成。按照实验要求，需要生成时域采样波形，下面代码生成了一个包含 10us 脉宽的脉冲，周期为 100us 的时域回波采样信号。脉冲信号幅度为 0.3V。采样率为 10MHz。白噪声信噪比为 50dB。经过采样后，单周期信号长度为 1000 点。

```
1. real_period_time = 100e-6 ; %100us
2. real_single_time = 10e-6 ; %10us
3. fs = 10e+6 ; %10Mhz 采样率
4.
5. %%采样时间处理
```

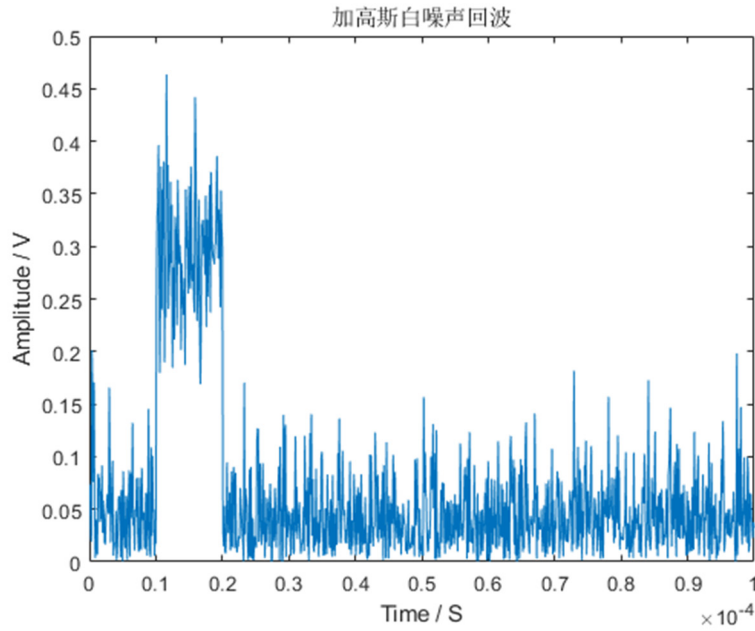
```
6. t_all = 1/fs:1/fs:real_period_time;
7. real_signal_length = length(t_all);
8.
9. t_single = 1/fs:1/fs:real_single_time;
10.
11. %信号波形生成
12. pluse_amplitude = 0.3; %单位 V, 范围 0~5V
13. real_pluse_signal = pluse_amplitude * rectpuls(t_single);
14. real_pluse_signal_length = length(real_pluse_signal);
15. real_idel_signal = [zeros(1,100),real_pluse_signal(1:real_pluse_signal_length),zeros(1,real_signal_length-real_pluse_signal_length-100)];
16.
17. gaussian_noise_snr = 25;
18. real_noise_signal = abs(awgn(real_idel_signal,gaussian_noise_snr));
```

绘制理想的回波采样波形和加入高斯白噪声后的回波采样波形。

```
1. plot(t_all,real_idel_signal);
2. title("理想回波");
3. xlabel("Time / S");
4. ylabel("Amplitude / V");
5.
6. plot(t_all,real_noise_signal);
7. title("加高斯白噪声回波");
8. xlabel("Time / S");
9. ylabel("Amplitude / V");
```



理想回波时域采样信号



加高斯白噪声回波时域采样信号

接下来，分析带噪回波信号的信噪比。

```
1. snr_single_period = snr(real_idel_signal,real_noise_signal-real_idel_signal)
```

得到结果 $\text{snr_single_period} = 4.5850$ 。

2.2 使用非相参积累算法，提高信噪比

生成 8 个周期的带噪回波数据，将这 8 个周期数据进行叠加，然后求均值。具体代码如下：

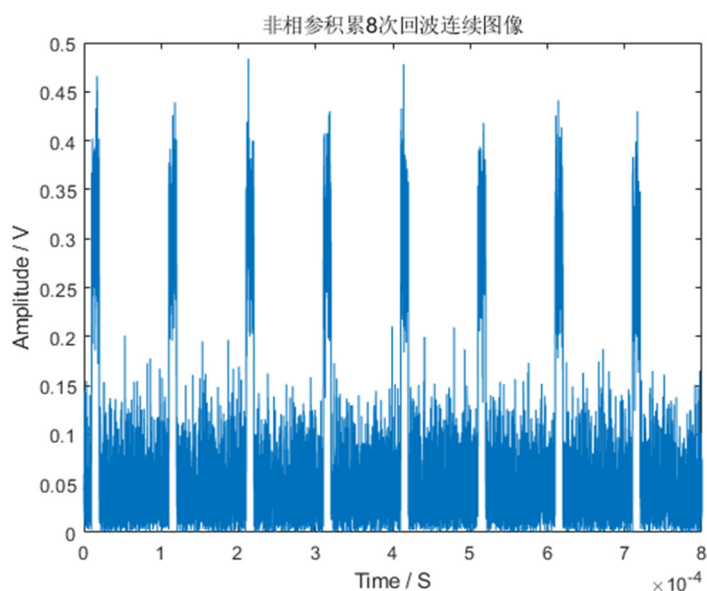
```
1. waveforms_number = 8
2. mif_file_data = []; %用于生成 mif 的内存
3. data_acc_result = abs(awgn(real_idel_signal,gaussian_noise_snr));
4. mif_file_data = [mif_file_data(1:length(mif_file_data)),data_acc_result(1:real
   signal_length)];
5. for number = 1:waveforms_number-1
6.     real_noise_signal = abs(awgn(real_idel_signal,gaussian_noise_snr));
7.     data_acc_result = data_acc_result + real_noise_signal;
8.     mif_file_data = [mif_file_data(1:length(mif_file_data)),real_noise_signal(1:
   real_signal_length)];
9. end
10.
11. data_acc_result = 1/waveforms_number.*data_acc_result;
12.
13. t_acc_all = 1/fs:1/fs:waveforms_number*real_period_time;
14. length_t_acc_all = length(t_acc_all);
15.
16. plot(t_acc_all,mif_file_data);
17. title("非相参积累 8 次回波连续图像");
18. xlabel("Time / S");
19. ylabel("Amplitude / V");
20.
```

```

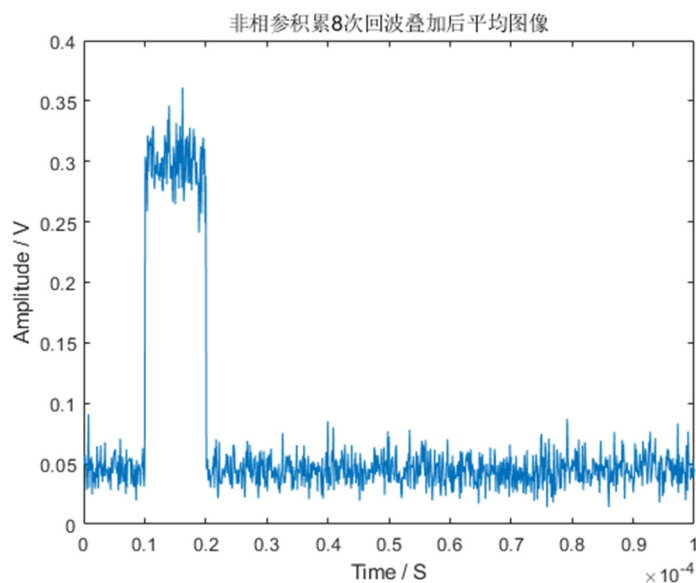
21.
22. plot(t_all,data_acc_result);
23. title("非相参积累 8 次回波叠加后平均图像");
24. xlabel("Time / S");
25. ylabel("Amplitude / V");

```

其输出图形如下：



非相参积累的 8 次回波连续图像波形



非相参积累的 8 次回波叠加后平均图像波形

分析非相参积累算法的回波结果信噪比：

```

1. snr_non_coherent_acc = snr(real_idel_signal,data_acc_result-real_idel_signal)

```

结果为：snr_non_coherent_acc = 6.6407。

进一步的，得到非相参积累增益：

```

1. Gnc = snr_non_coherent_acc/snr_single_period

```


结果为：Gnc = 1.4484。

3、FPGA 的非相参积累算法仿真实现方法

要在 FPGA 中实现非相参积累算法设计，主要需要以下步骤：

1. 将 MATLAB 中的 8 个周期的带噪回波数据导出为 mif 文件；
2. 在 Quartus 中，生成 ROM IP，并将 1 中文件与该 IP 绑定；
3. 实现非相参累积的算法编写，这里使用 VHDL 语言；
4. 对 3 中算法仿真，在 Modelsim 中分析波形，并导出结果波形数据文件；
5. 对 4 中的波形数据在 Excel 中进行预处理，并保存为表格文件；
6. 对 5 中表格，利用 MATLAB 进行数据处理，将 64 位二进制表示的双精度数转换为小数形式；
7. 对 6 中数据绘制波形，分析信噪比。

3.1 MATLAB 导出 8 个周期的带噪回波数据

在 MATLAB 中，本章的 2.2 一节中已生成了 8 个周期的带噪回波数据，现将数据导出为 mif 文件，以便在 ROM IP 核中调用这些数据，使用如下代码生成 mif 文件。

```

1.  r = [];
2.  for i = 1:length_t_acc_all
3.      r = [r;ieee_754_double_to_64bits(mif_file_data(i))];
4.  end
5.
6.
7.  fid = fopen('data1.mif','w');
8.  fprintf(fid,'%s\n','DEPTH = 8000;');
9.  fprintf(fid,'%s\n','WIDTH = 64;');
10. fprintf(fid,'%s\n','ADDRESS_RADIX = DEC;');
11. fprintf(fid,'%s\n','DATA_RADIX = BIN;');
12. fprintf(fid,'%s\t','CONTENT');
13. fprintf(fid,'%s\n','BEGIN');
14. for i=1:size(r,1)
15.     fprintf(fid,'%d',i-1);      %address
16.     fprintf(fid,'%s',' : ');
17.     fprintf(fid,'%s',r(i,1));
18.     fprintf(fid,'%s',r(i,2));
19.     fprintf(fid,'%s',r(i,3));
20.     fprintf(fid,'%s',r(i,4));
21.     fprintf(fid,'%s',r(i,5));
22.     fprintf(fid,'%s',r(i,6));
23.     fprintf(fid,'%s',r(i,7));
24.     fprintf(fid,'%s',r(i,8));
25.     fprintf(fid,'%s',r(i,9));
26.     fprintf(fid,'%s',r(i,10));
27.     fprintf(fid,'%s',r(i,11));
28.     fprintf(fid,'%s',r(i,12));
29.     fprintf(fid,'%s',r(i,13));
30.     fprintf(fid,'%s',r(i,14));
31.     fprintf(fid,'%s',r(i,15));

```

```
32.    fprintf(fid, '%s',r(i,16));
33.    fprintf(fid, '%s',r(i,17));
34.    fprintf(fid, '%s',r(i,18));
35.    fprintf(fid, '%s',r(i,19));
36.    fprintf(fid, '%s',r(i,20));
37.    fprintf(fid, '%s',r(i,21));
38.    fprintf(fid, '%s',r(i,22));
39.    fprintf(fid, '%s',r(i,23));
40.    fprintf(fid, '%s',r(i,24));
41.    fprintf(fid, '%s',r(i,25));
42.    fprintf(fid, '%s',r(i,26));
43.    fprintf(fid, '%s',r(i,27));
44.    fprintf(fid, '%s',r(i,28));
45.    fprintf(fid, '%s',r(i,29));
46.    fprintf(fid, '%s',r(i,30));
47.    fprintf(fid, '%s',r(i,31));
48.    fprintf(fid, '%s',r(i,32));
49.    fprintf(fid, '%s',r(i,33));
50.    fprintf(fid, '%s',r(i,34));
51.    fprintf(fid, '%s',r(i,35));
52.    fprintf(fid, '%s',r(i,36));
53.    fprintf(fid, '%s',r(i,37));
54.    fprintf(fid, '%s',r(i,38));
55.    fprintf(fid, '%s',r(i,39));
56.    fprintf(fid, '%s',r(i,40));
57.    fprintf(fid, '%s',r(i,41));
58.    fprintf(fid, '%s',r(i,42));
59.    fprintf(fid, '%s',r(i,43));
60.    fprintf(fid, '%s',r(i,44));
61.    fprintf(fid, '%s',r(i,45));
62.    fprintf(fid, '%s',r(i,46));
63.    fprintf(fid, '%s',r(i,47));
64.    fprintf(fid, '%s',r(i,48));
65.    fprintf(fid, '%s',r(i,49));
66.    fprintf(fid, '%s',r(i,50));
67.    fprintf(fid, '%s',r(i,51));
68.    fprintf(fid, '%s',r(i,52));
69.    fprintf(fid, '%s',r(i,53));
70.    fprintf(fid, '%s',r(i,54));
71.    fprintf(fid, '%s',r(i,55));
72.    fprintf(fid, '%s',r(i,56));
73.    fprintf(fid, '%s',r(i,57));
74.    fprintf(fid, '%s',r(i,58));
75.    fprintf(fid, '%s',r(i,59));
76.    fprintf(fid, '%s',r(i,60));
77.    fprintf(fid, '%s',r(i,61));
78.    fprintf(fid, '%s',r(i,62));
79.    fprintf(fid, '%s',r(i,63));
80.    fprintf(fid, '%s',r(i,64));
81.    fprintf(fid, '%s\n',';');
```

```

82. end
83. fprintf(fid,'%s\n','END;');
84.
85.
86. fclose(fid);

```

生成文件 data1.mif，其部分内容如图所示：

```

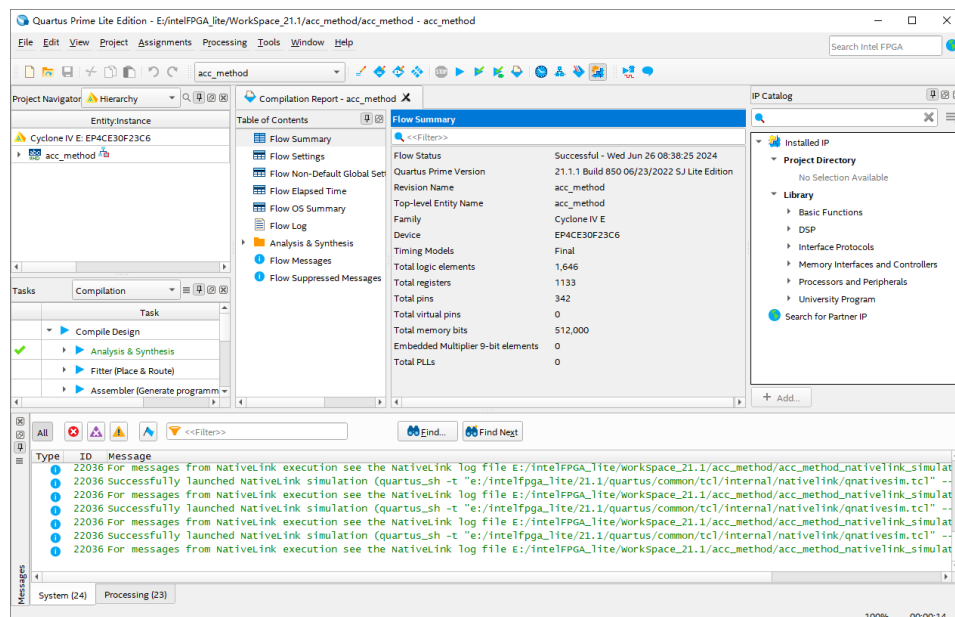
1 DEPTH = 8000;
2 WIDTH = 64;
3 ADDRESS_RADIX = DEC;
4 DATA_RADIX = BIN;
5 CONTENT BEGIN
6 0 : 0011111101001100110011000111000000111000101101101111011011101;
7 1 : 001111110101110100111100110110010010011011111011011010011;
8 2 : 0011111100111111011101100100110100100101111001000001111;
9 3 : 0011111100110000110001111100110111010000101101000110000111100;
10 4 : 001111110010100110100100100001100111010011101101101100010011;
11 5 : 00111111001010001000110101101011011011001011100101001001100;
12 6 : 00111111000111100001010110101100010101001010101110001111100;
13 7 : 00111111100001001110101110101101101100001011110011001000110;
14 8 : 0011111100110101101001110000010000111110011101011111000111101;
15 9 : 0011111101110101011110000101000100000011000100011101100011001;
16 10 : 00111111010110111101100010011110011010101101111000110000000;
17 11 : 001111110101010001110011110110101111011011000001010101110000;
18 12 : 00111111000101110011011111100000001100000001011111101100100;
19 13 : 00111111000010011011000010110000100011001101101010101100101;
20 14 : 001111110001100001000010010111000100111100100011100111010001100;
21 15 : 0011111101110110001100111110000100111101011011010101000010011;
22 16 : 00111111010011111010010011010100100010111000000110101101000;

```

MATLAB 生成 mif 文件

3.2 Quartus 生成 ROM IP

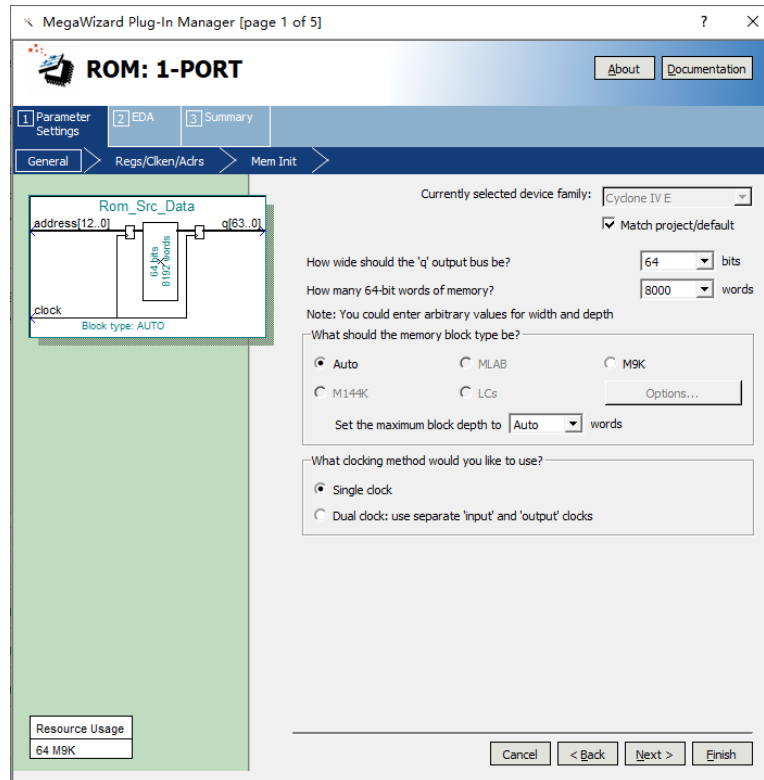
接下来使用 Quartus 生成所要使用的 ROM IP。Quartus 的工程界面如下图所示。



Quartus 的工程界面

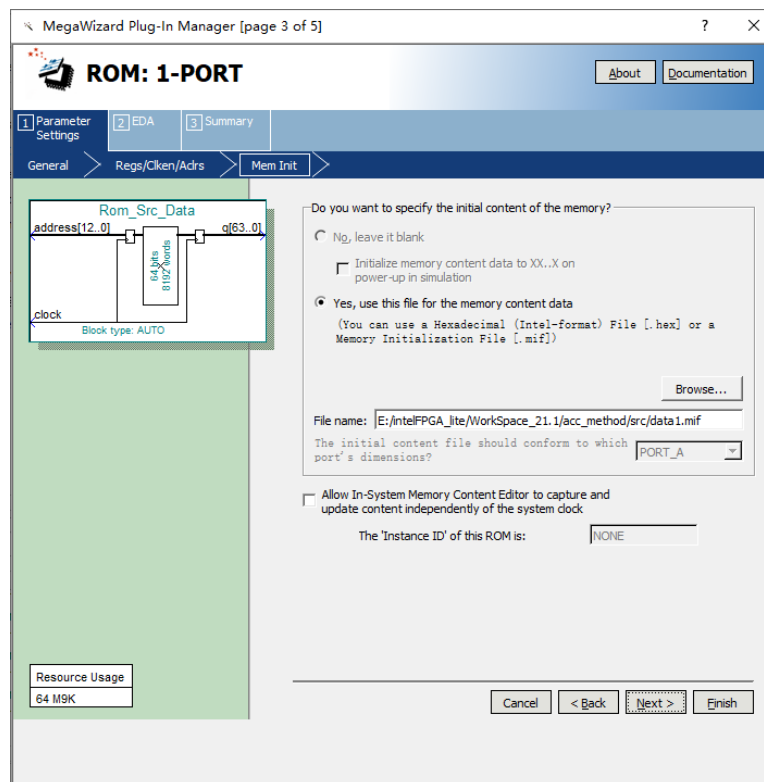
下面，调用 ROM IP 创建窗口，这里选择的是单口 ROM，指定位宽为 64，

64 位宽的字为 8000 个（单周期为 1000 点，8 个周期即 8000 点）。

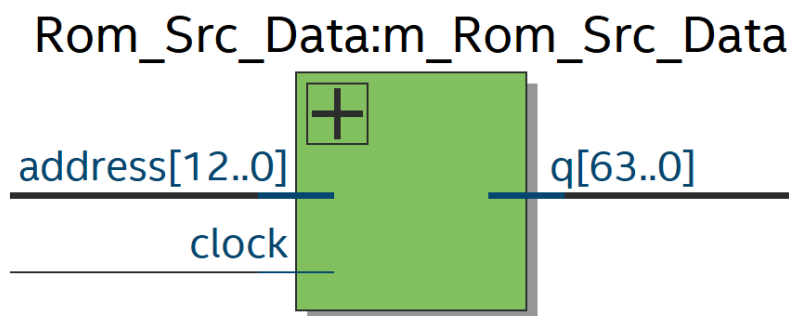


为 ROM IP 指定位宽和字长

在创建时，需要指定数据源，这里填写 mif 文件的绝对地址，如下图所示。



指定 mif 文件的绝对地址



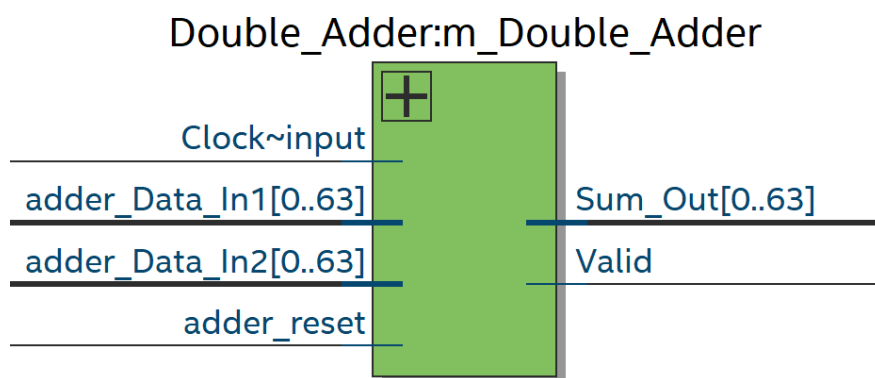
RTL 查看器中实例化的 ROM 模块

3.3 FPGA 的双精度浮点数（double）加法器设计

根据本实验要求，生成数据比较特殊，所以这里设计的 64 位双精度浮点数（double）加法器模块也是针对实验数据做了简化。

实验要求，生成数据的电平为 0~5V，即都为非负数。8 个周期的数据加和后，仍然是一个特定范围的小数（0~40V），故而不涉及到溢出问题。

接下来，说明加法器模块设计思路。



RTL 查看器中实例化的双精度加法器模块

双精度数（64 位）的组成由 1 位符号位，11 位指数位和 52 位小数位构成。对于双精度 double 指数 E 占 11 位，有关指数部分本文给出如下理解过程：

- $Val(E) = E - Bias$
- 指数为 11 位，范围为 0 ~ 2047
- $Bias = 1023$ (half of 2046)
- $Val(E) = E - 1023$
- $Val(E=1) = -1022, Val(E=1023) = 0, Val(E=2046) = 1023$

下面本文给出双精度 double 的十进制计算例子，假设有下列的双精度数需转换为十进制 double 小数格式：

00111111101011101001111100110110
01001001100110111111011011010011

要计算的 64 位双精度数

- $Sign = 1$ ，表示负数； $Sign = 0$ ，表示正数
- $E = (01111111010)_2 = -5$ ， $Val(E=1018) = E - bias = 1018 - 1023 = -5$

- $Significand = (1.1110...1)_2 = 1.913870132003855$ （1.是隐藏的）
- $value(Dec) = 1.913870132003855 \times 2^{-5} = 0.059808441625120$

接下来，分析本实验所需的双精度加法器实现步骤。假设有两个双精度数（double） $X1=(s1,e1,f1)$ 和 $X2=(s2,e2,f2)$ ，计算两者之和的步骤如下：

- 计算指数差： $d = e1 - e2$ ，如果 $e1 < e2$ ，交换两个小数的位置。将较大的指数暂定为结果的指数；
- 读取符号位，应该有 $s1 = s2 = 0$ ，将读取的符号位赋值给结果的符号位；
- 先对齐小数，将较小的小数右移 d 位；
- 将两个小数部分相加，得到暂定结果的小数位；
- 对小数结果做舍入；
- 规范化结果。

主要代码如下：

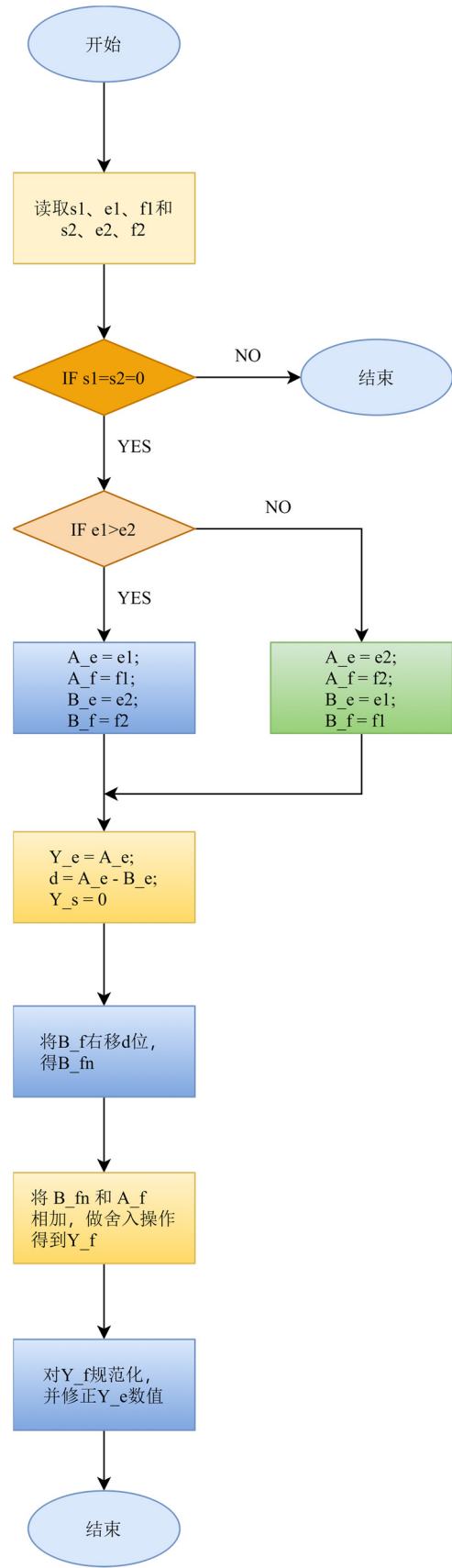
```

1.  case clock_count is
2.
3.      -- CLock 1 : Make that A is always bigger than B
4.      when 1 =>
5.          if Data_In1( 62 downto 52 ) > Data_In2( 62 downto 52 ) then
6.              A_e <= Data_In1( 62 downto 52 );
7.              B_e <= Data_In2( 62 downto 52 );
8.              A_f <= Data_In1( 51 downto 0 );
9.              B_f <= Data_In2( 51 downto 0 );
10.             A_s <= Data_In1( 63 );
11.             B_s <= Data_In2( 63 );
12.         else
13.             A_e <= Data_In2( 62 downto 52 );
14.             B_e <= Data_In1( 62 downto 52 );
15.             A_f <= Data_In2( 51 downto 0 );
16.             B_f <= Data_In1( 51 downto 0 );
17.             A_s <= Data_In2( 63 );
18.             B_s <= Data_In1( 63 );
19.         end if;
20.
21.      -- Clock 2: Makt that Exp_diff is recognized
22.      -- Get Y_s
23.      -- Calculate a_f and b_f
24.      when 2 =>
25.          if A_s = B_s then
26.              Y_s <= A_s;
27.              Y_e <= A_e;
28.              A_f2 <= "1" & A_f;
29.              if unsigned( A_e ) = unsigned( B_e ) then
30.                  N := 0;
31.              else
32.                  N := conv_integer( A_e - B_e );
33.              end if;
34.          else

```


84. end case;

下面是加法器实现的流程图：



64 位双精度数加法器算法实现流程图

3.4 FPGA 的双精度浮点数（double）整数除法设计

在使用双精度加法器后，可以得到叠加后的值，接下来需要做整数除法，以便得到平均值。由于这里的除数比较特殊，为 8（8 个周期），是 2 的整数次幂。所以可以采用下面几步实现双精度浮点数（double）的特殊整数除法。假设，被除数 $X=(s,e,f)$ ，除数为 8，除法算法的过程为：

- 读取被除数指数 e
- $e' = e - 3$
- 得结果 $X'=(s,e',f)$

以 3.3 节中的双精度数为例，按照上述步骤可得到除以 8 后的结果如下表的双精度数：

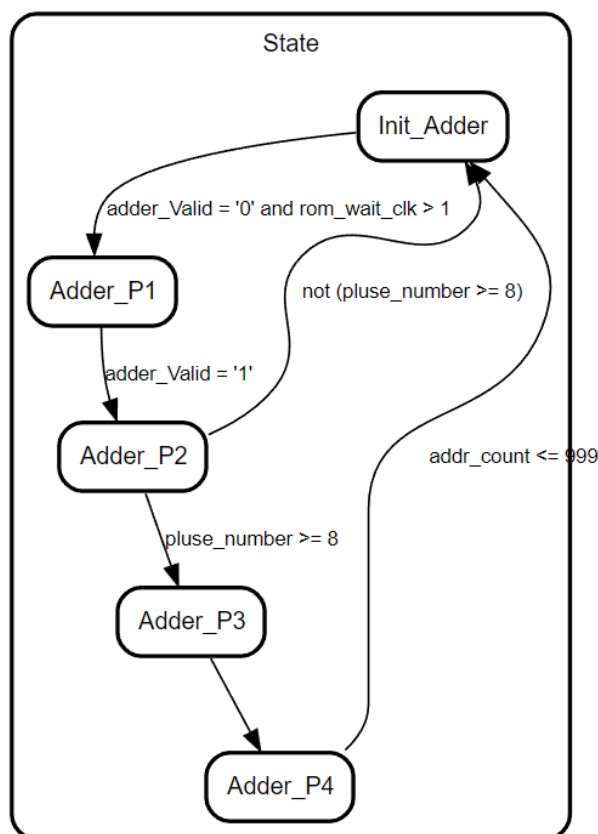
00111111011111010011111001101110
01001001100110111111011011010011

除法结果的 64 位双精度数

- $value(Dec) = 1.913870132003855 \times 2^{-5} = 0.059808441625120$
- $value(Dec) / 8 = 0.059808441625120 / 8 = 0.007476055203140$
- $value(X') = 1.913870132003855 \times 2^{-8} = 0.007476055203140$

3.5 非相参积累算法实现

本节介绍非相参积累算法的主要实现步骤。非相参积累算法采用有限状态机，设置了 5 个状态，根据激励和状态转换条件在不同状态间转换。所设计的状态机简化状态图如下图所示。



非相参积累算法简化状态图

状态 Init_Adder: 将要加和的两个数读入, 并使能加法器, 接着跳转到 Adder_P1 状态。

状态 Adder_P1: 得出加法和, 增加 rom 地址, 然后跳转到 Adder_P2 状态。

状态 Adder_P2: 判断 8 个周期是否累加完毕, 若 8 个周期累加完毕, 准备整数除法, 然后跳转到 Adder_P3; 否则, 跳转到 Init_Adder 状态。

状态 Adder_P3: 处理下一个 rom 地址, 并做整数除法, 然后跳转到 Adder_P4 状态。

状态 Adder_P4: 输出非相参累积结果, 并判断数据是否输出完毕, 若还有数据未输出, 跳转到 Init_Adder 状态。否则, 停止程序。

上述过程的主要实现代码如下:

```

1.  case State is
2.      --
3.      when Init_Adder =>
4.          rom_wait_clk := rom_wait_clk + 1;
5.          Result_Valid <= '0';
6.          if adder_Valid = '0' and rom_wait_clk > 1 then
7.              adder_Data_In1 <= adder_Sum_Out_RE;
8.              adder_Data_In2 <= rom_out_data;
9.              debug_d1 <= adder_Sum_Out_RE;
10.             debug_d2 <= rom_out_data;
11.             adder_reset <= '0';
12.             State <= Adder_P1;
13.             clk_count := clk_count + 1;
14.             rom_wait_clk := 0;
15.         end if;
16.     --
17.     when Adder_P1 =>
18.
19.         if adder_Valid = '1' then
20.             if clk_count < 8 then
21.                 rom_address <= rom_address + addr_incr_constant_1000;
22.             end if;
23.             adder_Sum_Out_RE <= adder_Sum_Out;
24.             acc_result_tm <= adder_Sum_Out;
25.             pluse_number := pluse_number + 1;
26.             State <= Adder_P2;
27.         end if;
28.
29.     --
30.     when Adder_P2 =>
31.
32.         if pluse_number >= 8 then -- next point
33.             addr_count := addr_count + 1;
34.             pluse_number := 0;
35.
36.             acc_result_e <= adder_Sum_Out_RE( 62 downto 52 );
37.             acc_result_f <= adder_Sum_Out_RE( 51 downto 0 );
38.             acc_result_s <= adder_Sum_Out_RE( 63 );

```

```

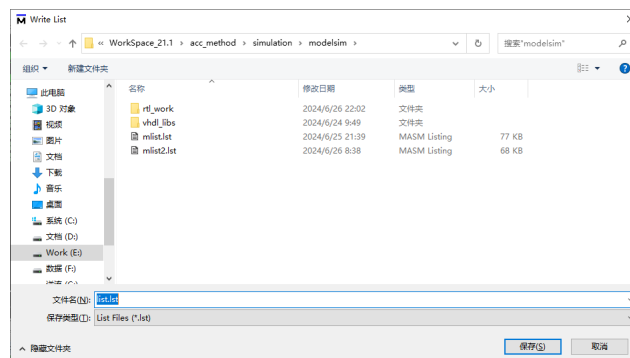
39.
40.     State <= Adder_P3;
41.     -- rom_out_data_RE <= rom_out_data;
42. else
43.     State <= Init_Adder;
44.     -- rom_out_data_RE <= rom_out_data;
45. end if;
46. adder_reset <= '1';
47. --
48. when Adder_P3 =>
49.     if addr_count < 1000 then
50.         rom_address <= std_logic_vector(to_unsigned(addr_count, rom_a
address'length));
51.     end if;
52.
53.     adder_Sum_Out_RE <= ( others=>'0' );
54.     acc_result_e <= acc_result_e - 3;
55.     clk_count := 0;
56.     State <= Adder_P4;
57.
58. when Adder_P4 =>
59.     acc_result <= acc_result_s & acc_result_e( 10 downto 0 ) & acc_result
_f( 51 downto 0 );
60.     Result_Valid <= '1';
61.     if addr_count <= 999 then
62.         State <= Init_Adder;
63.     elsif addr_count >= 1000 then
64.         clk_count := 0;
65.     end if;
66.
67. end case;

```

3.6 ModelSim 仿真结果导出

将 ModelSim 仿真输出的非相参积累结果输出需要以下步骤：

- 在 ModelSim 中，选择“View” -> “List”，List 窗口会弹出
- 回到 Wave 窗口，拖动目标信号到 List 窗口，释放
- 在 List 窗口，选择“File” -> “Write List” -> “Tabular List”，保存文件。
- 保存后的 lst 文件可以使用 Excel 打开并处理。



保存 lst 文件窗口

	A	B	C	D	E	F	G	H
1	+1	0011111110101001100100001011111000111011010010000011011011101010						
2	+1	001111111010110100101001010111001100000100101000100110111011000						
3	+1	0011111110101000110011111000100000101100001010010100000111001100						
4	+1	001111111010011111110101101001000011110100100000001100111011011						
5	+1	00111111100111010011111100100000010010001001011101100111001111						
6	+1	0011111110100110000011001001011111110011101001000000101111000110						
7	+1	001111111010101111100001101011010101101101000001001111001000000						
8	+1	00111111101100111001110111001010010110010111110101110010011010110						
9	+1	0011111110011110011011011001010110011011100101000101001010000						
10	+1	0011111110101011101011100001100011011110100010111100001100100011						
11	+1	0011111110100111010101111100111110000111100000110111101011000						
12	+1	0011111110100110111001001100110111011111000000001010000110111010						
13	+1	001111111010110010011011100000000000100001011010111011001011101						
14	+1	001111111001100100100000111011111011000101111011001111110110000						
15	+1	0011111110101001110110000100101001110100101000000010000100001011						
16	+1	0011111110100001101010110101000111101110110100101100110111011011						
17	+1	001111111010011000100010101111001100000000110010100010000110011						
18	+1	001111111010001010111000100011000110011110000101000111011111111						
19	+1	0011111110011110000101101111010101110111100011110001000111011100						
20	+1	0011111110100111111001001110010010110101011110001001010000001110						
21	+1	0011111110100011100001111100000111001110101100000010011101011000						
22	+1	00111111101100000111111000000110000001110101111111011011111111						
23	+1	0011111110101000001000101100111100010001000001101011001000001110						
24	+1	00111111100111100011111100101111000110000101110100110000000000						
25	+1	00111111101001110101001111011101110010100110111011011110001010						
26	+1	001111111010101010111111001111001101010101000000100100001100111						
27	+1	001111111010100000001010000110100111100010100101100000101001101						
28	+1	00111111101100001011100001011110111010010011101111111111101110011						
29	+1	0011111110100100100110010111000101001011011100000000011101001110						
30	+1	0011111110101000010101010011011011000101011000000100001100111100						
31	+1	0011111110100001110100110111010010011011111000011001110001110111						
32	+1	001111111010100111000011001011111110000111001001011001101010111						

lst 文件处理后的表格数据

3.8 MATLAB 处理数据

接着对 3.7 节中的表格数据进行处理，以便描绘出结果波形和分析信噪比。使用如下代码，对数据进行读取和处理：

```

1.  clc;
2.  clear all;
3.  format long
4.  %读取表格数值和文本数据
5.  filename = 'm_wave_result.xlsx'; %文件名称
6.  sheet = 1; %文件的第几个表格
7.  xlRange = 'A1:A1000'; %表格范围
8.  [num,txt,row] = xlsread(filename,sheet,xlRange);
9.
10. length = length(txt);
11.
12. earse_str = '+1 ';
13.
14. for i=1:length
15.     str = txt(i,1);
16.     str = erase(str,earse_str)
17.     cell_str = cell2mat(str)
18.
19.     r_wave(i) = hex2num(m64bits_to_hex(cell_str));
20. end
21.
22. real_period_time = 100e-6 ; %100us
23. real_single_time = 10e-6 ; %10us
24. fs = 10e+6 ; %10Mhz 采样率

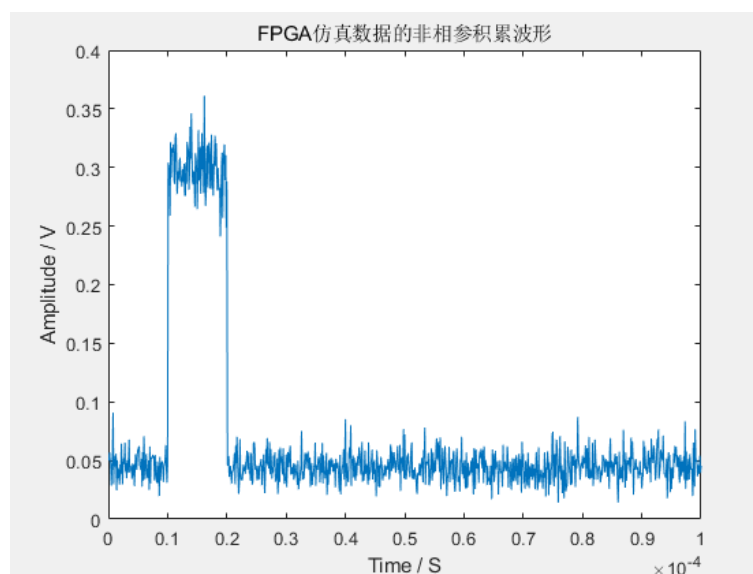
```

```

25.
26. %%采样时间处理
27. t_all = 1/fs:1/fs:real_period_time;
28.
29. plot(t_all,r_wave);
30. title("FPGA 仿真数据的非相参积累波形");
31. xlabel("Time / S");
32. ylabel("Amplitude / V");
33.
34.
35.
36.
37. function r_hex = m64bits_to_hex(r)
38.     t_hex = [];
39.     for divd_time = 1:4
40.         %将二进制字符划分为 4 段 16 位，转换为十六进制
41.         start_index = (divd_time-1) * 16 + 1;
42.         stop_index = divd_time * 16;
43.         dec_r1 = r(start_index : stop_index);
44.         temp_hex = dec2hex(bin2dec(dec_r1));
45.         if length(temp_hex) < 4
46.             temp_hex = ['0',temp_hex];
47.         end
48.         t_hex = [t_hex,temp_hex];
49.     end
50.     r_hex = t_hex;
51. end

```

绘制结果波形如下图：



FPGA 仿真数据的非相参积累波形

使用以下代码分析信噪比：

```

1. real_signal_length = length(t_all);
2. t_single = 1/fs:1/fs:real_single_time;

```

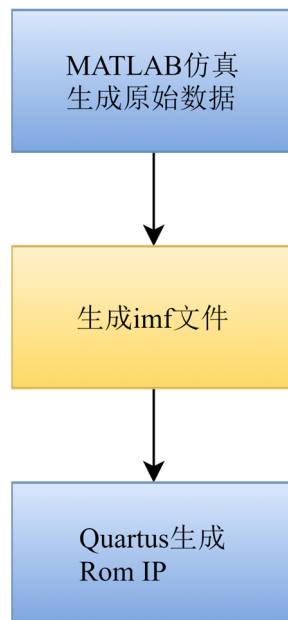
```
3.  
4.  %信号波形生成  
5.  pluse_amplitude = 0.3; %单位 V， 范围 0~5V  
6.  real_pluse_signal = pluse_amplitude * rectpuls(t_single);  
7.  real_pluse_signal_length = length(real_pluse_signal);  
8.  real_idel_signal = [zeros(1,100),real_pluse_signal(1:real_pluse_signal_length)  
                      ,zeros(1,real_signal_length-real_pluse_signal_length-100)];  
9.  
10. snr_fpga_single_period = snr(real_idel_signal,r_wave-real_idel_signal)
```

其结果为： $\text{snr_fpga_single_period} = 6.640637012304276$ 。

计算 FPGA 算法结果的信噪比增益为： $\text{Gnc_fpga} = 1.448339588288828$ 。

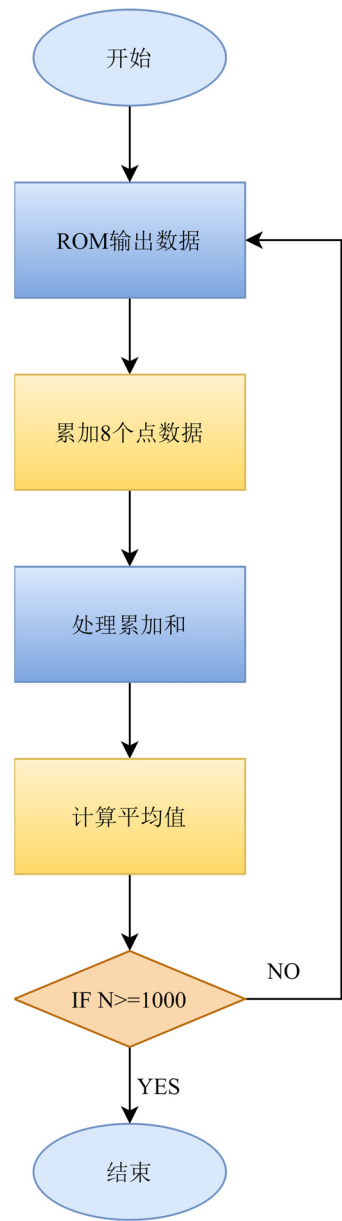
设计流程图

- FPGA 生成 ROM IP 流程图：



FPGA 生成 ROM IP

- FPGA 非相参积累算法实现主要过程流程图：

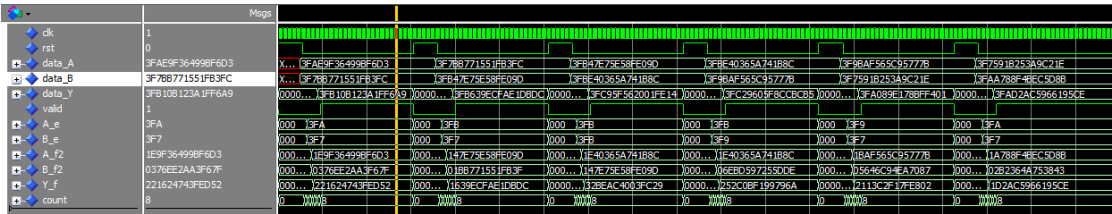


FPGA 非相参积累算法实现主要过程

设计结果和分析

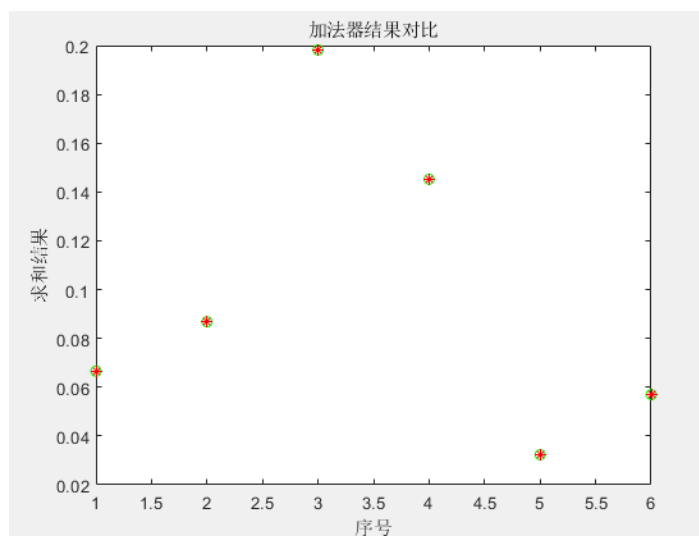
● FPGA 加法器仿真结果和分析

FPGA 加法器测试采用 6 组数据进行加法运算，运算结果和 MATLAB 的运算结果进行比较，得出相对误差。



FPGA 加法器仿真结果

使用 MATLAB 对仿真结果分析，绘制如下结果对比图。



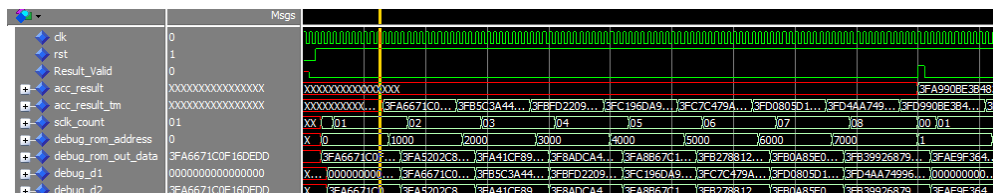
FPGA 加法器仿真结果对比图

绿色圆点数据为 FPGA 仿真结果，红色星型数据为 MATLAB 输出结果。接着计算相对误差，取平均值得到如下结果：

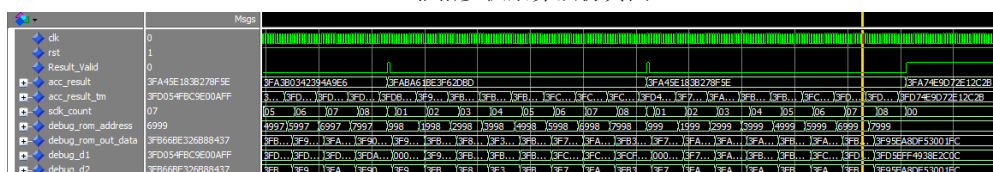
$$relative_error = 1.218056106415363e-16。$$

● 非相参积累算法的结果与分析

FPGA 的非相参积累算法仿真结果如下图所示。



FPGA 非相参积累算法仿真图 1



FPGA 非相参积累算法仿真图 2

对比上文中的非相参积累算法的信噪比增益：

两个信噪比增益对比表

	信噪比增益 (Gnc)
MATLAB 结果	1.4484
FPGA 结果	1.44834

可以得到信噪比增益的相对误差：

$$r_error = \frac{|G_{nc_matlab} - G_{nc_fpga}|}{G_{nc_matlab}} = \frac{|1.4484 - 1.44834|}{1.4484} = 4.142502071247514e-05$$

结论

从实验的结果可以得到以下结论：

- FPGA 实现的非相参积累算法与 MATLAB 计算出的结果有一定的误差；
- 上述误差精度小于万分之一；
- 产生误差的主要原因是 FPGA 加法器中的舍入过程；
- 通过改进舍入算法，可以进一步逼近 MATLAB 的计算结果。

思考

从设计思路来看，FPGA 的设计思路是自顶向下设计，各个模块之间互相配合完成最终的功能。完成本实验时，我大量使用了 MATLAB 作为辅助工具，处理 FPGA 生成的数据，并构建简单算法拓扑结构，验证算法的功能。这个经验可以在今后的工程中继续使用。