**哈尔滨工业大学（威海）**

**《EDA技术高级应用》**

**课 程 报 告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **调 研** | 优 □ | 良 □ | 中 □ | 差 □ | **成 绩** |
| **设 计** | 优 □ | 良 □ | 中 □ | 差 □ |
| **结 果** | 优 □ | 良 □ | 中 □ | 差 □ |  |
| **报 告** | 优 □ | 良 □ | 中 □ | 差 □ |
| **评 语** |  | | | | |

|  |  |
| --- | --- |
| **题 目：** | 非相参积累算法仿真实现 |
| **学 号：** |  |
| **姓 名：** |  |

**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 | 1. 仿真平台：Quartus II 13.0以上； 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； 3. 关键程序行和参数设置处均需标明注释； 4. 功能仿真； | | |
| 设计要求 | 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。 | | |

## 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。主要是对非相参算法仿真，完成双精度浮点数加法模块、ROM模块等开发，最后输出Modelsim仿真结果。

输出VHDL仿真结果后，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。

1. 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。

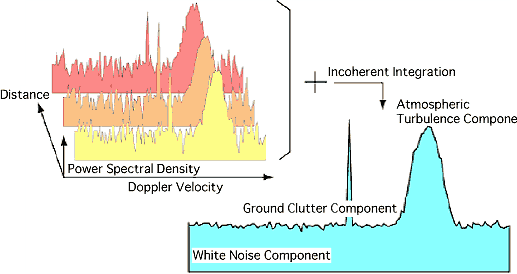
### 1、非相参积累算法

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波形信噪比；
6. 结合5和3，分析非相参累积增益。

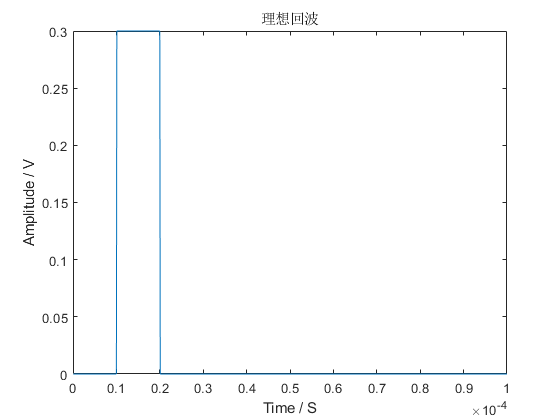
#### 2.1 模拟回波信号的生成与信噪比分析

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脉宽的脉冲，周期为的时域回波采样信号。信号幅度为。采样率为。白噪声信噪比为。经过采样后，单周期信号长度为1000点。

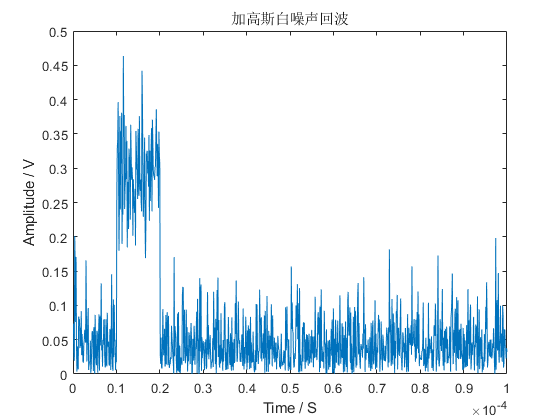
1. real\_period\_time = 100e-6 ;   %50us
2. real\_single\_time = 10e-6 ;    %10us
3. fs = 10e+6               ;   %10Mhz采样率
4. %%采样时间处理
5. t\_all = 1/fs:1/fs:real\_period\_time;
6. real\_signal\_length = length(t\_all);
7. t\_single = 1/fs:1/fs:real\_single\_time;
8. %信号波形生成
9. pluse\_amplitude = 0.3;   %单位V，范围0~5V
10. real\_pluse\_signal = pluse\_amplitude \* rectpuls(t\_single);
11. real\_pluse\_signal\_length = length(real\_pluse\_signal);
12. real\_idel\_signal = [zeros(1,100),real\_pluse\_signal(1:real\_pluse\_signal\_length),zeros(1,real\_signal\_length-real\_pluse\_signal\_length-100)];
13. gaussian\_noise\_snr = 25;
14. 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. plot(t\_all,real\_noise\_signal);
6. title("加高斯白噪声回波");
7. xlabel("Time / S");
8. 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)

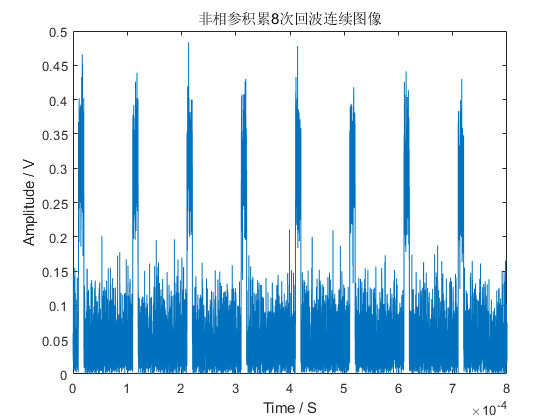
得到结果。

#### 2.2 使用非相参积累算法，提高信噪比

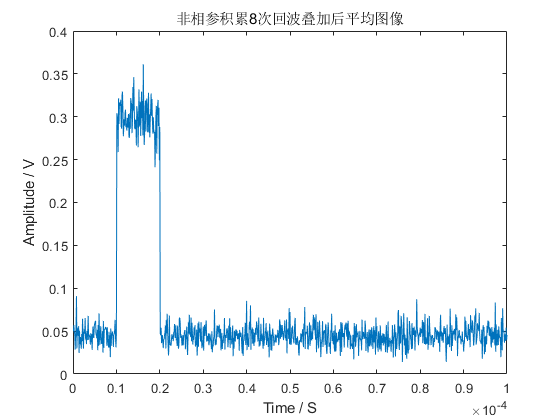
生成8个周期的带噪回波数据，将这8个周期数据进行叠加，然后求均值。具体代码如下：

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. data\_acc\_result = 1/waveforms\_number.\*data\_acc\_result;
11. t\_acc\_all = 1/fs:1/fs:waveforms\_number\*real\_period\_time;
12. length\_t\_acc\_all = length(t\_acc\_all);
13. plot(t\_acc\_all,mif\_file\_data);
14. title("非相参积累8次回波连续图像");
15. xlabel("Time / S");
16. ylabel("Amplitude / V");
17. plot(t\_all,data\_acc\_result);
18. title("非相参积累8次回波叠加后平均图像");
19. xlabel("Time / S");
20. 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

分析非相参累积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：。

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：。

### 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仿真，主要需要以下步骤：

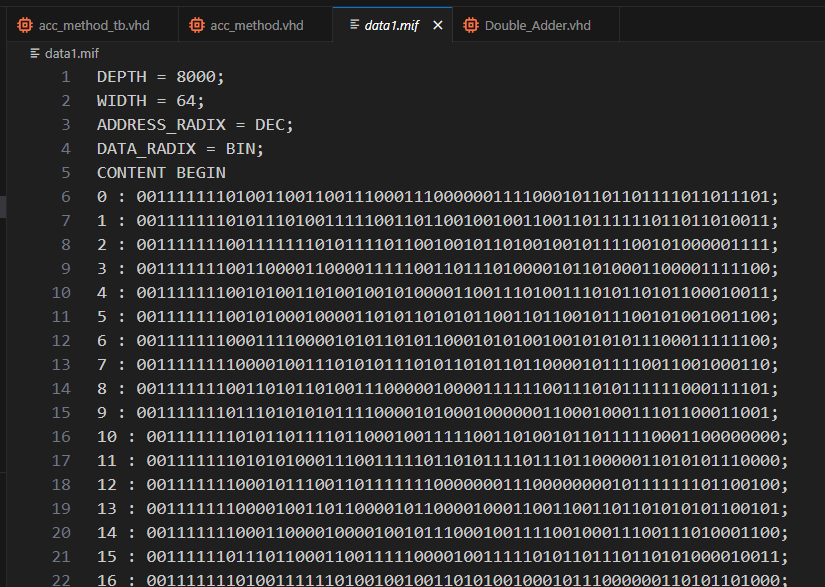
1. 将MATLAB中的8个周期的带噪回波数据导出为mif文件；
2. 在Quartus中，生成ROM IP，并将1中文件与该IP绑定；
3. 实现非相参累积的算法编写，这里使用VHDL语言；
4. 对3中算法仿真，在Modelsim中分析波形，并导出结果波形数据文件；
5. 对4中的波形数据在Excel中进行第一步处理，并保存为表格文件；
6. 对5中表格，利用MATLAB进行数据处理，将64位二进制表示的双精度数转换为小数形式；
7. 对6中数据绘制波形，分析信噪比。

#### 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的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. fid = fopen('data1.mif','w');
6. fprintf(fid,'%s\n','DEPTH = 8000;');
7. fprintf(fid,'%s\n','WIDTH = 64;');
8. fprintf(fid,'%s\n','ADDRESS\_RADIX = DEC;');
9. fprintf(fid,'%s\n','DATA\_RADIX = BIN;');
10. fprintf(fid,'%s\t','CONTENT');
11. fprintf(fid,'%s\n','BEGIN');
12. for i=1:size(r,1)
13. fprintf(fid, '%d',i-1);            %address
14. fprintf(fid, '%s',' : ');
15. fprintf(fid, '%s',r(i,1));
16. fprintf(fid, '%s',r(i,2));
17. fprintf(fid, '%s',r(i,3));
18. fprintf(fid, '%s',r(i,4));
19. fprintf(fid, '%s',r(i,5));
20. fprintf(fid, '%s',r(i,6));
21. fprintf(fid, '%s',r(i,7));
22. fprintf(fid, '%s',r(i,8));
23. fprintf(fid, '%s',r(i,9));
24. fprintf(fid, '%s',r(i,10));
25. fprintf(fid, '%s',r(i,11));
26. fprintf(fid, '%s',r(i,12));
27. fprintf(fid, '%s',r(i,13));
28. fprintf(fid, '%s',r(i,14));
29. fprintf(fid, '%s',r(i,15));
30. fprintf(fid, '%s',r(i,16));
31. fprintf(fid, '%s',r(i,17));
32. fprintf(fid, '%s',r(i,18));
33. fprintf(fid, '%s',r(i,19));
34. fprintf(fid, '%s',r(i,20));
35. fprintf(fid, '%s',r(i,21));
36. fprintf(fid, '%s',r(i,22));
37. fprintf(fid, '%s',r(i,23));
38. fprintf(fid, '%s',r(i,24));
39. fprintf(fid, '%s',r(i,25));
40. fprintf(fid, '%s',r(i,26));
41. fprintf(fid, '%s',r(i,27));
42. fprintf(fid, '%s',r(i,28));
43. fprintf(fid, '%s',r(i,29));
44. fprintf(fid, '%s',r(i,30));
45. fprintf(fid, '%s',r(i,31));
46. fprintf(fid, '%s',r(i,32));
47. fprintf(fid, '%s',r(i,33));
48. fprintf(fid, '%s',r(i,34));
49. fprintf(fid, '%s',r(i,35));
50. fprintf(fid, '%s',r(i,36));
51. fprintf(fid, '%s',r(i,37));
52. fprintf(fid, '%s',r(i,38));
53. fprintf(fid, '%s',r(i,39));
54. fprintf(fid, '%s',r(i,40));
55. fprintf(fid, '%s',r(i,41));
56. fprintf(fid, '%s',r(i,42));
57. fprintf(fid, '%s',r(i,43));
58. fprintf(fid, '%s',r(i,44));
59. fprintf(fid, '%s',r(i,45));
60. fprintf(fid, '%s',r(i,46));
61. fprintf(fid, '%s',r(i,47));
62. fprintf(fid, '%s',r(i,48));
63. fprintf(fid, '%s',r(i,49));
64. fprintf(fid, '%s',r(i,50));
65. fprintf(fid, '%s',r(i,51));
66. fprintf(fid, '%s',r(i,52));
67. fprintf(fid, '%s',r(i,53));
68. fprintf(fid, '%s',r(i,54));
69. fprintf(fid, '%s',r(i,55));
70. fprintf(fid, '%s',r(i,56));
71. fprintf(fid, '%s',r(i,57));
72. fprintf(fid, '%s',r(i,58));
73. fprintf(fid, '%s',r(i,59));
74. fprintf(fid, '%s',r(i,60));
75. fprintf(fid, '%s',r(i,61));
76. fprintf(fid, '%s',r(i,62));
77. fprintf(fid, '%s',r(i,63));
78. fprintf(fid, '%s',r(i,64));
79. fprintf(fid,'%s\n',';');
80. end
81. fprintf(fid,'%s\n','END;');
82. fclose(fid);

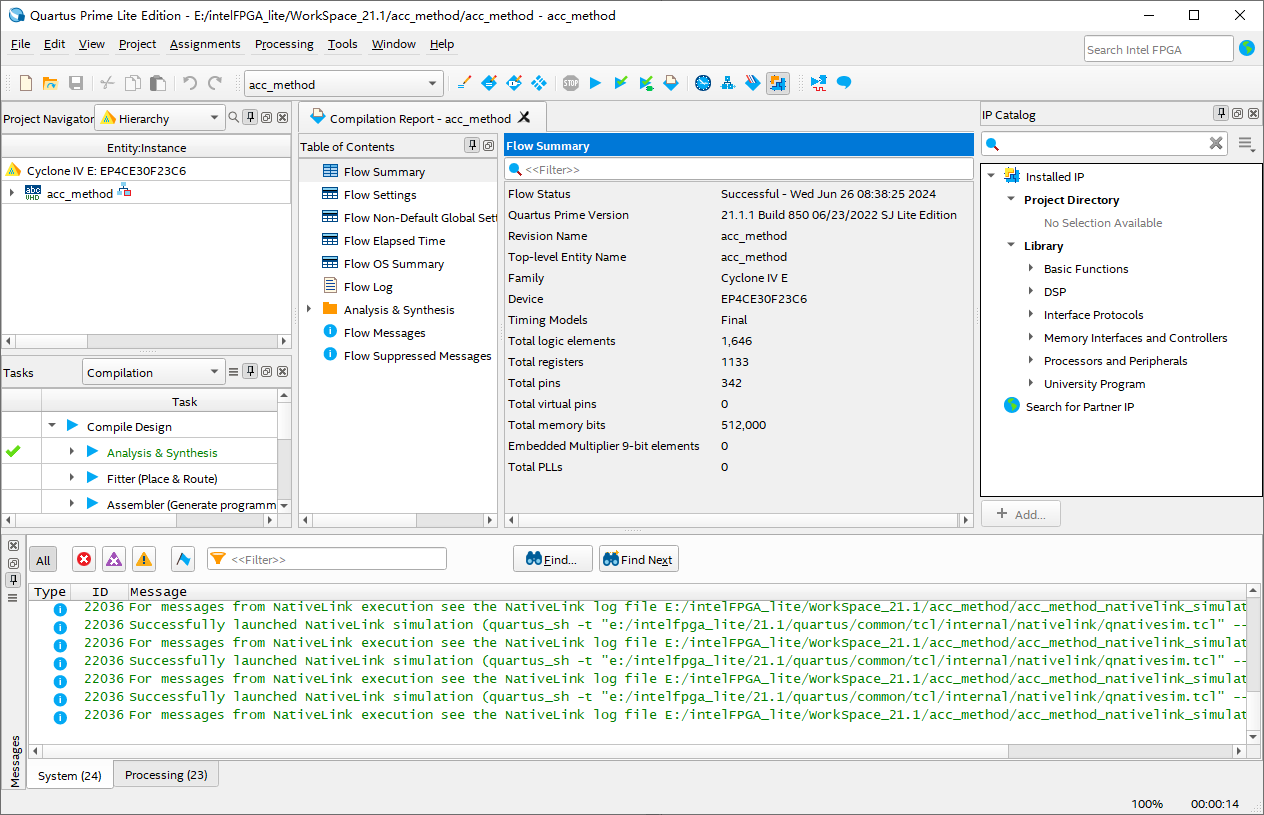
生成文件data1.mif，其部分内容如图所示：



MATLAB生成mif文件

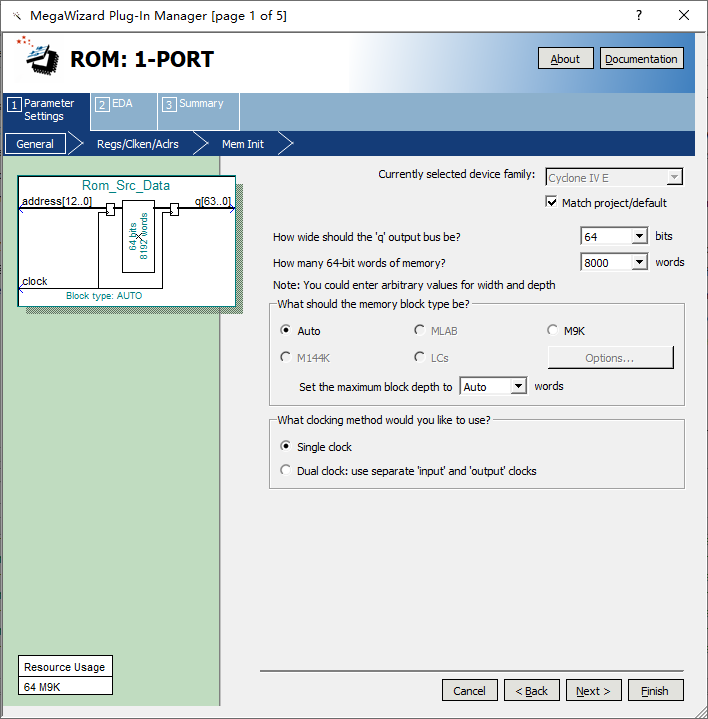
#### 3.2 Quartus生成ROM IP

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。



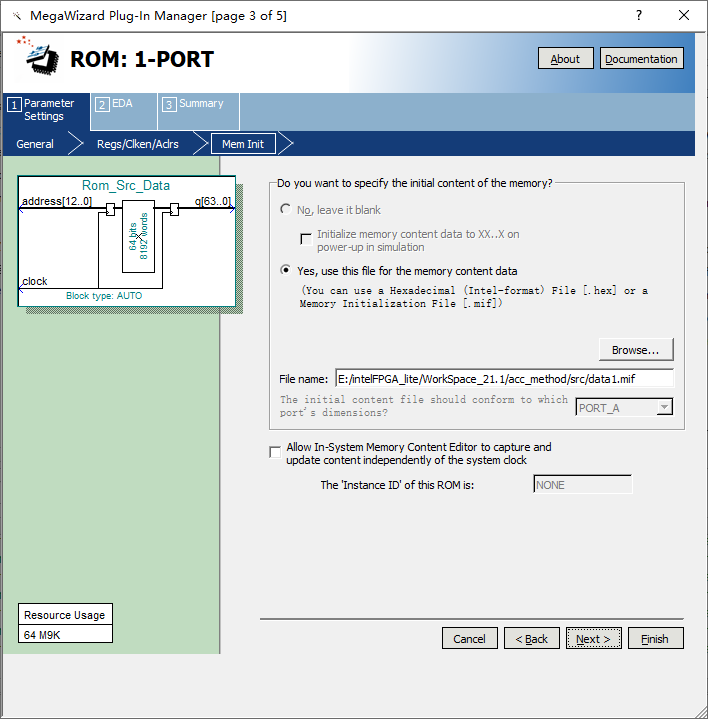
Quartus的工程界面

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）。

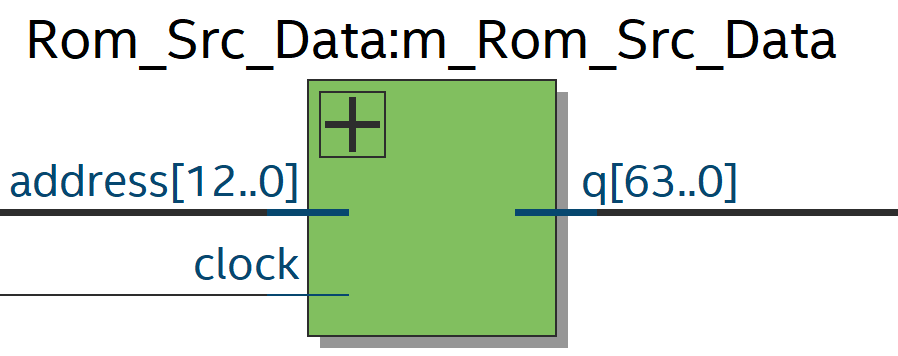


为ROM IP指定位宽和字长

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



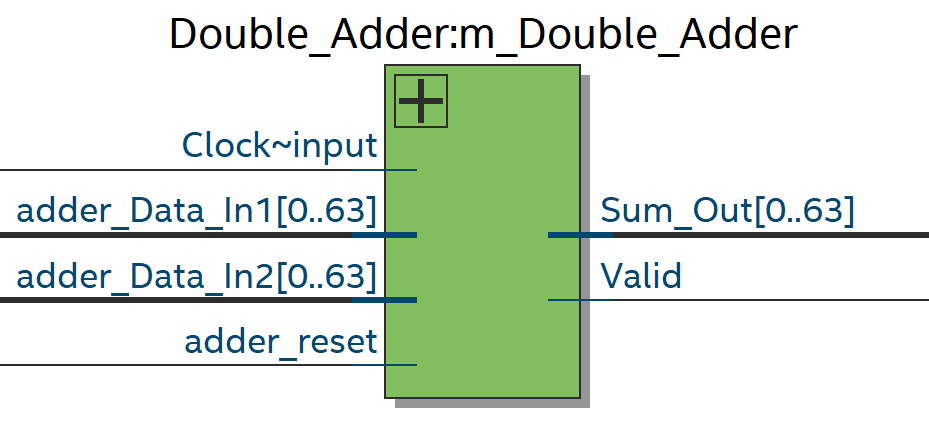
RTL查看器中实例化的ROM模块

#### 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，仍然是一个范围的小数（0~40V），也不涉及到溢出问题。

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，有关指数部分的知识如下：

* 
* 指数为11位，范围为
* 
* 
* 

双精度double的十进制计算，如下表的双精度数：

|  |
| --- |
| 00111111101011101001111100110110 |
| 01001001100110111111011011010011 |

要计算的64位双精度数

* ，表示负数；，表示正数
* ，
* （1.是隐藏的）
* 

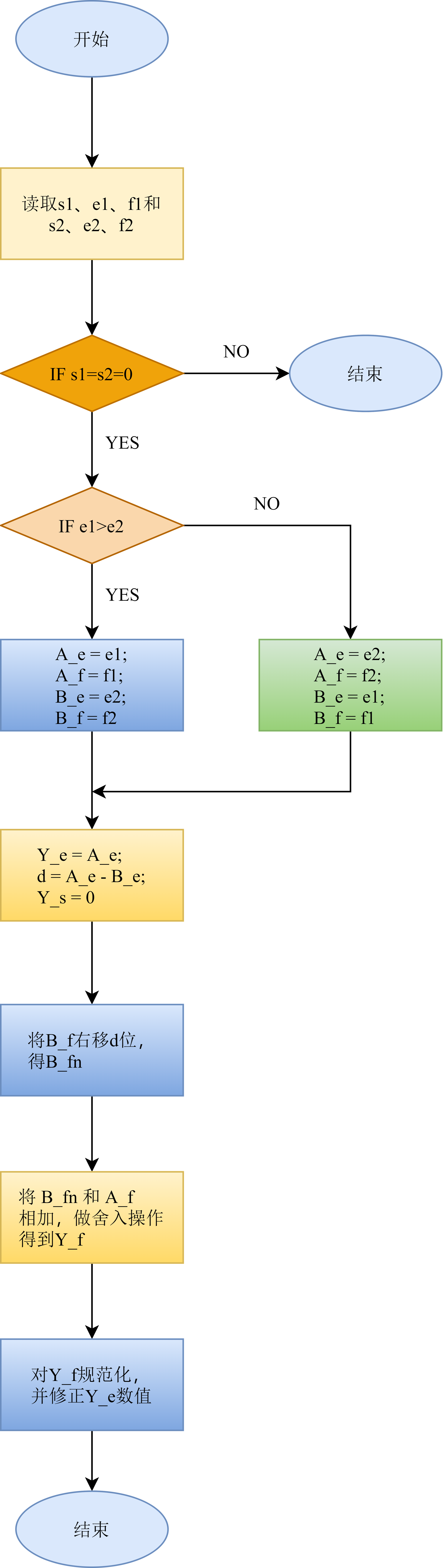
接下来，分析本实验所需的双精度加法器实现步骤。假设有两个双精度数（double）和，计算两者之和的步骤如下：

* 计算指数差：，如果，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，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
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. -- Clock 2: Makt that Exp\_diff is recognized
21. --   Get Y\_s
22. --   Calculate a\_f' and b\_f'
23. when 2 =>
24. if A\_s = B\_s then
25. Y\_s <= A\_s;
26. Y\_e <= A\_e;
27. A\_f2 <= "1" & A\_f;
28. if  unsigned( A\_e ) = unsigned( B\_e ) then
29. N := 0;
30. else
31. N := conv\_integer( A\_e - B\_e );
32. end if;
33. else
34. clock\_count := 0;
35. end if;
37. -- without regard to overfolws
39. -- Clock 3: Add b\_f' and a\_f'
40. when 3 =>
41. B\_f2 <= to\_stdlogicvector( to\_bitvector( "1" & B\_f ) SRL N );
43. when 4 =>
44. Y\_f <= "0" & A\_f2 + B\_f2;
46. -- Clock 5: Get Y\_fn
47. when 5 =>
48. if (Y\_f( 53 downto 0 )="000000000000000000000000000000000000000000000000000000") then
49. Y <= ( others=>'0' );
50. else
51. if Y\_f(53) = '1' then
52. Y\_e <= Y\_e + 1;
53. Y\_fn <= Y\_f( 52 downto 1 );
54. symbol := 1;
55. else
56. Y\_fn <= Y\_f( 51 downto 0 );
57. symbol := 0;
58. end if;
59. end if;
60. -- Clock 6: round Y\_fn and output Y
61. when 6 =>
62. if symbol > 0  then
63. if Y\_f(0) = '1' then
64. Y\_fn <= Y\_fn + 1;
65. end if ;
66. end if ;
67. -- Y( 51 downto 0 ) <= Y\_fn;
68. -- Y( 62 downto 52 ) <= Y\_e;
69. -- Y( 63 ) <= Y\_s;
70. Y <= Y\_s & Y\_e( 10 downto 0 ) & Y\_fn( 51 downto 0 );
71. when 7 =>
72. Sum\_Out <= Y;
74. when others =>
75. clock\_count := 0;
76. Valid <= '1';
77. 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，除数为8，除法算法的过程为：

* 读取被除数指数
* 
* 得结果

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：

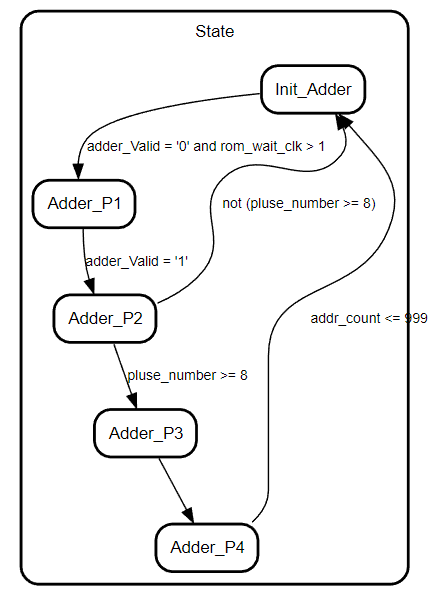
|  |
| --- |
| 00111111011111101001111100110110 |
| 01001001100110111111011011010011 |

除法结果的64位双精度数

* 
* 
* 

#### 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。



非相参累积算法简化状态图

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，Init\_Adder状态。否则，停止程序。

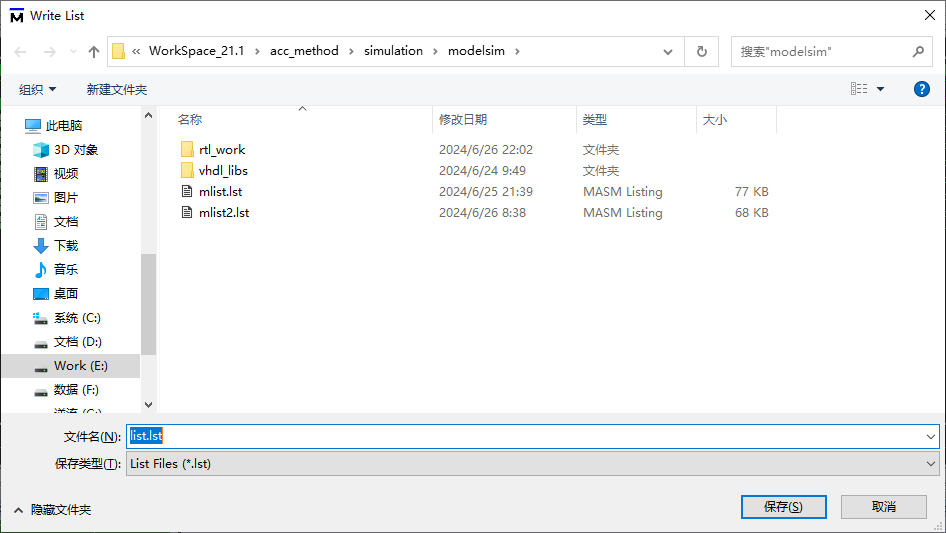
上述过程的主要实现代码如下：

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 =>
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. when Adder\_P2 =>
31. if pluse\_number >= 8 then    -- next point
32. addr\_count := addr\_count + 1;
33. pluse\_number := 0;
34. acc\_result\_e <= adder\_Sum\_Out\_RE( 62 downto 52 );
35. acc\_result\_f <= adder\_Sum\_Out\_RE( 51 downto 0 );
36. acc\_result\_s <= adder\_Sum\_Out\_RE( 63 );
37. State   <= Adder\_P3;
38. -- rom\_out\_data\_RE <= rom\_out\_data;
39. else
40. State   <= Init\_Adder;
41. -- rom\_out\_data\_RE <= rom\_out\_data;
42. end if ;
43. adder\_reset     <= '1';
44. --
45. when Adder\_P3 =>
46. if addr\_count < 1000 then
47. rom\_address     <= std\_logic\_vector(to\_unsigned(addr\_count, rom\_address'length));
48. end if;
49. adder\_Sum\_Out\_RE <= ( others=>'0' );
50. acc\_result\_e <= acc\_result\_e - 3;
51. clk\_count := 0;
52. State   <= Adder\_P4;
53. when Adder\_P4 =>
54. acc\_result <= acc\_result\_s & acc\_result\_e( 10 downto 0 ) & acc\_result\_f( 51 downto 0 );
55. Result\_Valid <= '1';
56. if addr\_count <= 999 then
57. State   <= Init\_Adder;
58. elsif addr\_count >= 1000 then
59. clk\_count := 0;
60. end if;
61. end case;

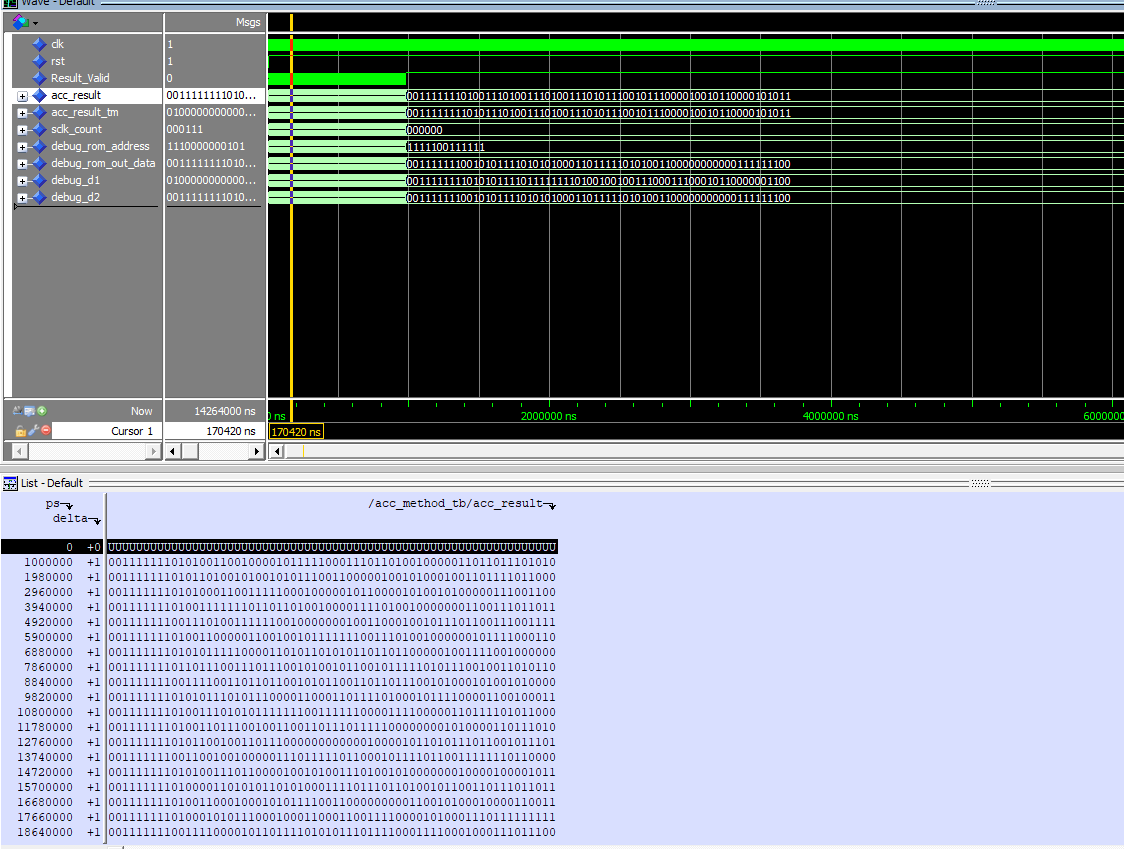
#### 3.6 ModelSim仿真结果导出

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。



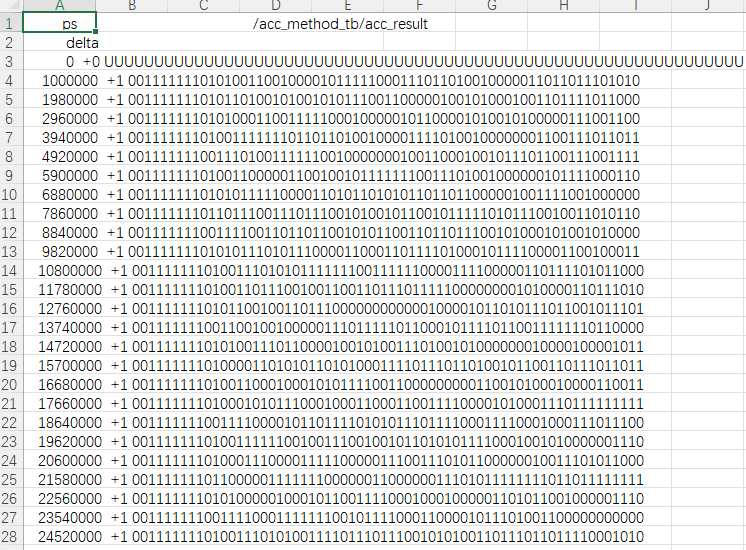
保存lst文件窗口



拖拽目标信号到List窗口

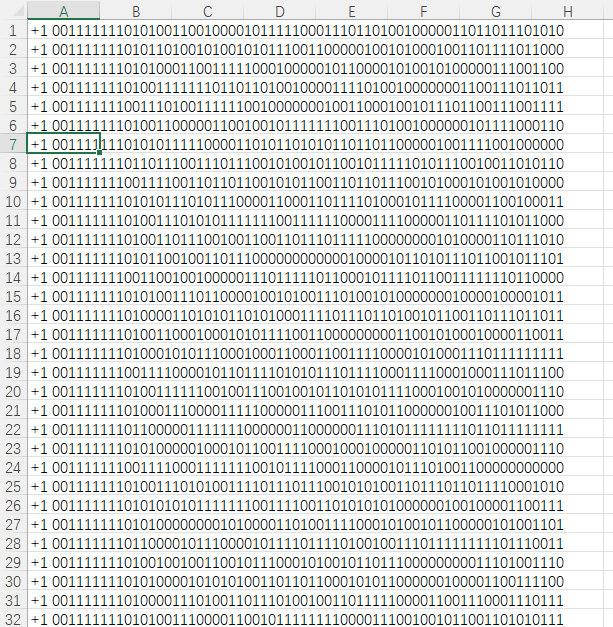
#### 3.7 Excel预处理数据

接下来，对3.6节输出的数据进行预处理。使用Excel打开lst文件。打开后文件如下图所示。



lst文件源数据

为了能够对数据统一处理，这里仅保留起始符号和64位数据。并将文件另存为表格类型。处理后数据如下：



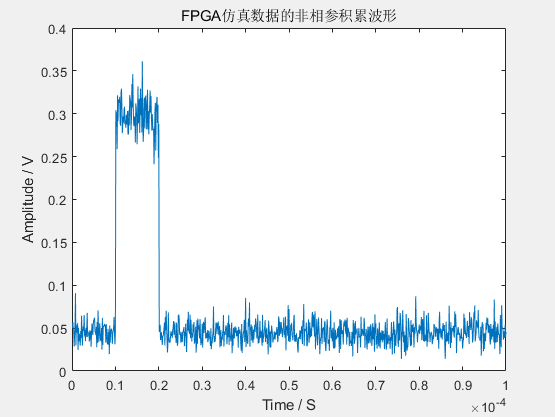
lst文件处理后的表格数据

#### 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,raw] = xlsread(filename,sheet,xlRange);
9. length = length(txt);
10. earse\_str = '+1 ';
11. for i=1:length
12. str = txt(i,1);
13. str = erase(str,earse\_str)
14. cell\_str = cell2mat(str)
16. r\_wave(i) = hex2num(m64bits\_to\_hex(cell\_str));
17. end
18. real\_period\_time = 100e-6 ;   %50us
19. real\_single\_time = 10e-6 ;   %10us
20. fs = 10e+6               ;   %10Mhz采样率
21. %%采样时间处理
22. t\_all = 1/fs:1/fs:real\_period\_time;
23. plot(t\_all,r\_wave);
24. title("FPGA仿真数据的非相参积累波形");
25. xlabel("Time / S");
26. ylabel("Amplitude / V");
27. function r\_hex = m64bits\_to\_hex(r)
28. t\_hex = [];
29. for divd\_time = 1:4
30. %将二进制字符划分为4段16位，转换为十六进制
31. start\_index = (divd\_time-1) \* 16 + 1;
32. stop\_index = divd\_time \* 16;
33. dec\_r1 = r(start\_index : stop\_index);
34. temp\_hex = dec2hex(bin2dec(dec\_r1));
35. if length(temp\_hex) < 4
36. temp\_hex = ['0',temp\_hex];
37. end
38. t\_hex = [t\_hex,temp\_hex];
39. end
40. r\_hex = t\_hex;
41. 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. %信号波形生成
4. pluse\_amplitude = 0.3;   %单位V，范围0~5V
5. real\_pluse\_signal = pluse\_amplitude \* rectpuls(t\_single);
6. real\_pluse\_signal\_length = length(real\_pluse\_signal);
7. real\_idel\_signal = [zeros(1,100),real\_pluse\_signal(1:real\_pluse\_signal\_length),zeros(1,real\_signal\_length-real\_pluse\_signal\_length-100)];
8. snr\_fpga\_single\_period = snr(real\_idel\_signal,r\_wave-real\_idel\_signal)

其结果为：。

## 设计流程图

FPGA生成ROM IP流程图：



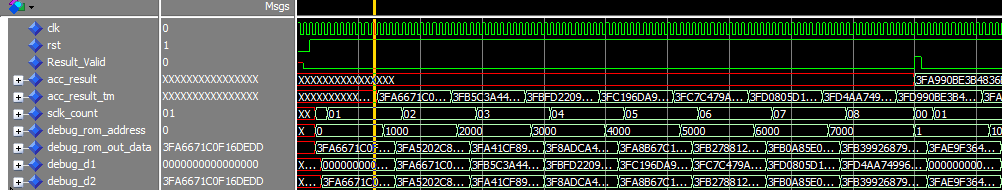
FPGA生成ROM IP

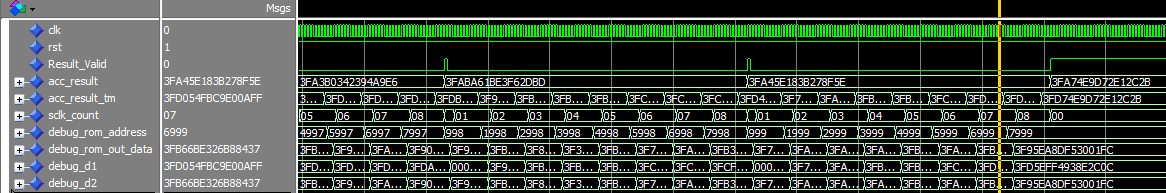
FPGA非相参积累算法实现主要过程流程图：



FPGA非相参积累算法实现主要过程

## 设计结果和分析（重点）





## 结论

实验目的

实验目的

## 思考

实验目的

实验目的