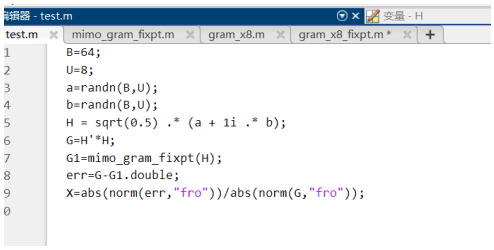
面向大规模 *MIMO* 检测的矩阵乘法设计

组长 吴非 519021910924
组员 贾鑫鹏 5190121911317

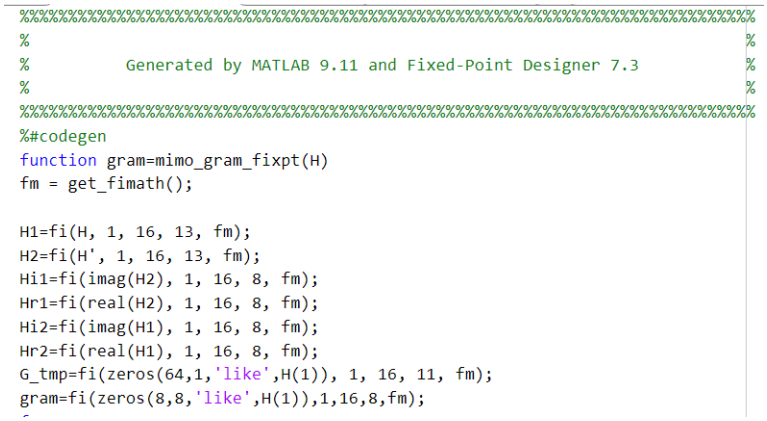
一、软件 Gram 矩阵乘法设计

1.代码



```
B=64;
U=8;
a=randn(B,U);
b=randn(B,U);
H = sqrt(0.5) .* (a + 1i .* b);
G=H'*H;
G1=mimo_gram_fixpt(H);
err=G-G1.double;
X=abs(norm(err,"fro"))/abs(norm(G,"fro"));
```

这部分代码由实验指导中给出是浮点型的 Gram 矩阵算法，并且列出了误差的计算。



```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                                 %
%          Generated by MATLAB 9.11 and Fixed-Point Designer 7.3                  %
%                                                                                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%#codegen
function gram=mimo_gram_fixpt(H)
fm = get_fimath();

H1=fi(H, 1, 16, 13, fm);
H2=fi(H', 1, 16, 13, fm);
Hi1=fi(imag(H2), 1, 16, 8, fm);
Hr1=fi(real(H2), 1, 16, 8, fm);
Hi2=fi(imag(H1), 1, 16, 8, fm);
Hr2=fi(real(H1), 1, 16, 8, fm);
G_tmp=fi(zeros(64,1,'like',H(1)), 1, 16, 11, fm);
gram=fi(zeros(8,8,'like',H(1)),1,16,8,fm);
```

该部分是对计算需要的值进行定义和提取这是定点化后的代码。

```
gram=fi(zeros(8,8,'like',H(1)),1,16,8,fm);
for x=1:8
    for y=1:8
        for n=1:64
            a=fi(Hr1(x,n), 1, 16, 13, fm);b=fi(Hi1(x,n), 1, 16, 13, fm);
            c=fi(Hr2(n,y), 1, 16, 13, fm);d=fi(Hi2(n,y), 1, 16, 13, fm);
            t1=fi(a*c, 1, 16, 12, fm);
            t2=fi(b*d, 1, 16, 12, fm);
            t3=fi(a*d, 1, 16, 13, fm);
            t4=fi(c*b, 1, 16, 13, fm);
            G_tmp(n)=t1-t2+(t3+t4)*fi(1i, 0, 1, 0, fm);

        end
        gram(x,y)=sum(G_tmp(:));
    end
end
```

矩阵的计算过程，在硬件结构中我们采用的是(a+bi)*(c+di)=(ac+bd)-(ad+bc)i 的计算公式，所以在计算过程中产生的中间变量不多，分别列出并提取，用工具箱对其进行量化,主要以 fi 函数对数据进行量化，例 fi(x,1,16,8),即对 x 变量进行定点量化，有符号数，16 位字长，8 位是小数。量化后分别得到了 G1，H1，在做差值，得到顶点量化后的误差 X1，通过多次的运行得到结果.

X1　　0.0117　　0.0010006　　0.0084　　0.0079　　0.0595　　0.0073

可以看出误差基本稳定在 6%以下符合要求。

```
end
    I1=reshape(Hi1,512,1);
I2=I1.bin;
R1=reshape(Hr1,512,1);
R2=R1.bin;
writematrix(R2,'R.txt')
writematrix(I2,'I.txt')
writematrix(gram.hex,'Gout.txt')
end


function fm = get_fimath()
    fm = fimath('RoundingMethod', 'Floor',...
         'OverflowAction', 'Wrap',...
         'ProductMode','FullPrecision',...
         'MaxProductWordLength', 128,...
         'SumMode','FullPrecision',...
         'MaxSumWordLength', 128);
end
```

在将需要的 H1 矩阵的虚步实部输出作为硬件的输入，Gout 作为相应参考的结果数据。

2.定点量化的考虑

对于输出矩阵 G 而言，在定点量化的过程中，首先在浮点型运行了几次代码，得到 G 矩阵，其中的结果不会超过 200，而将整数位设置为 8 位完全满足了整数部分的需求，而小数部分 8 位的精度也使得误差不会超过 6%，并且整数小数各 8 位的模式也方便了硬件部分的输入输出及结果整理。

对于输入数据 H 的量化，也采用各自 8 位的方式，整数部分会超出预期的性能，但是在硬件中会方便实现，并在且在后续的调试过程中更容易发现问题。

软件的计算过程为了保证精度均采用了 13 位小数的方式，但后面为了验证与硬件电路的适配性而改为了 8 位。

G | G1 | err

8x8 complex double

	1	2	3	4	5	6	7	8
1	68.1732 ...	-6.5251 - ...	-1.1782 +...	4.6243 + ...	-4.9825 - ...	-7.1936 +...	-4.5819 - ...	3.1888 + ...
2	-6.5251 +...	81.8880 ...	6.4711 - 0...	-5.9491 +...	9.5592 - 1...	-4.4757 - ...	-2.3720 - ...	-3.6325 - ...
3	-1.1782 - ...	6.4711 + ...	65.3973 ...	-5.7107 - ...	-8.7478 +...	-1.4288 - ...	0.6414 + ...	-7.9692 - ...
4	4.6243 - 1...	-5.9491 - ...	-5.7107 +...	64.6518 ...	-4.3853 - ...	11.1995 - ...	-1.5189 - ...	12.2776 - ...
5	-4.9825 +...	9.5592 + ...	-8.7478 - ...	-4.3853 +...	72.9526 ...	4.1925 - 9...	10.8959 ...	-5.0407 - ...
6	-7.1936 - ...	-4.4757 +...	-1.4288 +...	11.1995 ...	4.1925 + ...	62.7913 ...	2.1793 - 0...	1.3694 - 4...
7	-4.5819 +...	-2.3720 +...	0.6414 - 0...	-1.5189 +...	10.8959 - ...	2.1793 + ...	59.1355 ...	-2.8416 +...
8	3.1888 - 3...	-3.6325 +...	-7.9692 +...	12.2776 ...	-5.0407 +...	1.3694 + ...	-2.8416 - ...	61.7813 ...
9								

编辑器 - mimo_gram_fixpt.m　　变量 - G1

G | G1 | err

8x8 complex fi

	1	2	3	4	5	6	7	8
1	68.0703 - ...	-6.6914 - ...	-1.3164 +...	4.4766 + ...	-5.1250 - ...	-7.3203 +...	-4.7148 - ...	3.0430 + ...
2	-6.6484 +...	81.7148 - ...	6.3633 - 0...	-6.0938 +...	9.3867 - 1...	-4.6250 - ...	-2.5391 - ...	-3.7852 - ...
3	-1.2617 - ...	6.3984 + ...	65.3242 - ...	-5.8125 - ...	-8.8633 +...	-1.5430 - ...	0.5391 + ...	-8.0625 - ...
4	4.5195 - 1...	-6.0742 - ...	-5.8242 +...	64.5352 - ...	-4.5195 - ...	11.1016 - ...	-1.6758 - ...	12.1641 - ...
5	-5.0781 +...	9.4297 + ...	-8.8711 - ...	-4.4883 +...	72.8125 - ...	4.0742 - 9...	10.7617 ...	-5.1875 - ...
6	-7.2891 - ...	-4.5938 +...	-1.5430 +...	11.0938 ...	4.0781 + ...	62.7070 - ...	2.0664 - 0...	1.2305 - 4...
7	-4.6641 +...	-2.5039 +...	0.5430 - 0...	-1.6602 +...	10.7539 - ...	2.0703 - 0...	59.0078 - ...	-3.0117 +...
8	3.0859 - 3...	-3.7773 +...	-8.0703 +...	12.1406 ...	-5.2188 +...	1.2227 + ...	-3.0313 - ...	61.6953 - ...
9								

G | G1 | err

8x8 complex double

	1	2	3	4	5	6	7	8
1	0.1029 + ...	0.1663 + ...	0.1382 + ...	0.1478 + ...	0.1425 + ...	0.1267 + ...	0.1329 + ...	0.1458 + ...
2	0.1234 + ...	0.1731 + ...	0.1078 + ...	0.1446 + ...	0.1725 + ...	0.1493 + ...	0.1670 + ...	0.1527 + ...
3	0.0835 + ...	0.0726 + ...	0.0730 + ...	0.1018 + ...	0.1155 + ...	0.1142 + ...	0.1024 + ...	0.0933 + ...
4	0.1048 + ...	0.1251 + ...	0.1135 + ...	0.1167 + ...	0.1342 + ...	0.0979 + ...	0.1569 + ...	0.1135 + ...
5	0.0956 + ...	0.1295 + ...	0.1233 + ...	0.1030 + ...	0.1401 + ...	0.1183 + ...	0.1342 + ...	0.1468 + ...
6	0.0955 + ...	0.1180 + ...	0.1142 + ...	0.1057 + ...	0.1143 + ...	0.0843 + ...	0.1129 + ...	0.1390 + ...
7	0.0821 + ...	0.1319 + ...	0.0985 + ...	0.1412 + ...	0.1420 + ...	0.1090 + ...	0.1277 + ...	0.1702 + ...
8	0.1029 + ...	0.1449 + ...	0.1011 + ...	0.1369 + ...	0.1781 + ...	0.1468 + ...	0.1897 + ...	0.0860 + ...
9								

U 8

X 0.0074

单次软件运行结果及误差截图，实部虚部是随机生成具，数值上平等，通过对实部的观察可以看出，软件的仿真符合逻辑且运行正确。

二、硬件 Gram 矩阵乘法设计

1. 基本架构设计及数据的流动

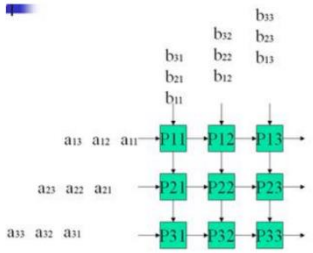


图 1 pe_array 基本结构

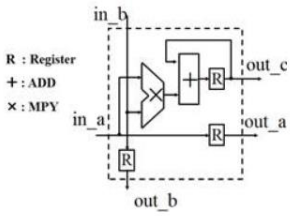


图 2 PE 单元结构

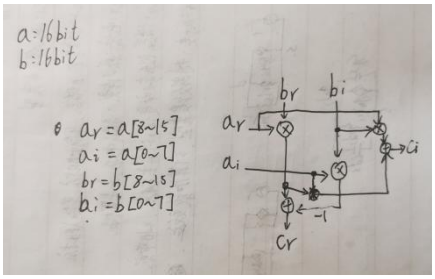
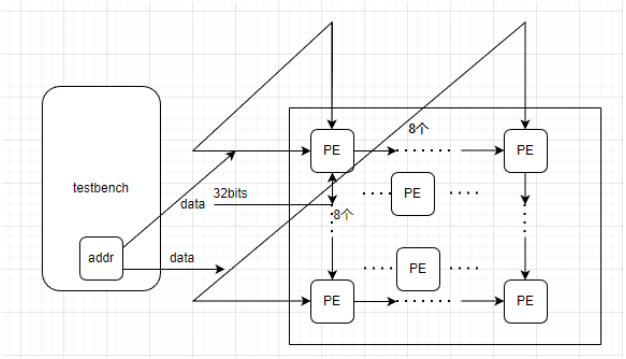


图 3 PE 中乘法器结构

整体图：



31:16bits：实部 15：0 虚部

以(3,3)*(3,3)的矩阵计算为例，相应的基本计算结构如上图，右侧为左图中对应的 PE 单元的具体架构设计，以 PE11 为例，前矩阵的第一行与后矩阵中的第一列依次流入并相乘，在通过其中的寄存器进行累加，则的到的结果为 PE11=a11*b11+a12*b21+a13*b31，此时 PE11 的结果为最终结果 G11 的值，对应的本次实验的矩阵算法为（8*64）*（64*8）的矩阵乘法，则需要 8*8 的 PE 基本单元，每个 PE 需要流入 64*2 个基本数据。

在 PE 的乘法器部分，是一个单独的 32 位的乘法计算器，将输入的 32bit 的 a，b 两个数据分为 16bit 的 4 部分，由公式(a+b*i)*(c+d*i)=ac-bd+(ad+bc)i 得到最终计算结果，再通过公式 ac-bd=a(c-d)+d(a-b)与 ad+bc=b(c+d)+d(a-b)优化得到对应得硬件结果。

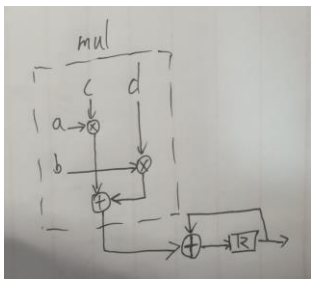
最终 64 个 PE 单元的输出是最终的结果。

2. 模块接口和功能的主要描述

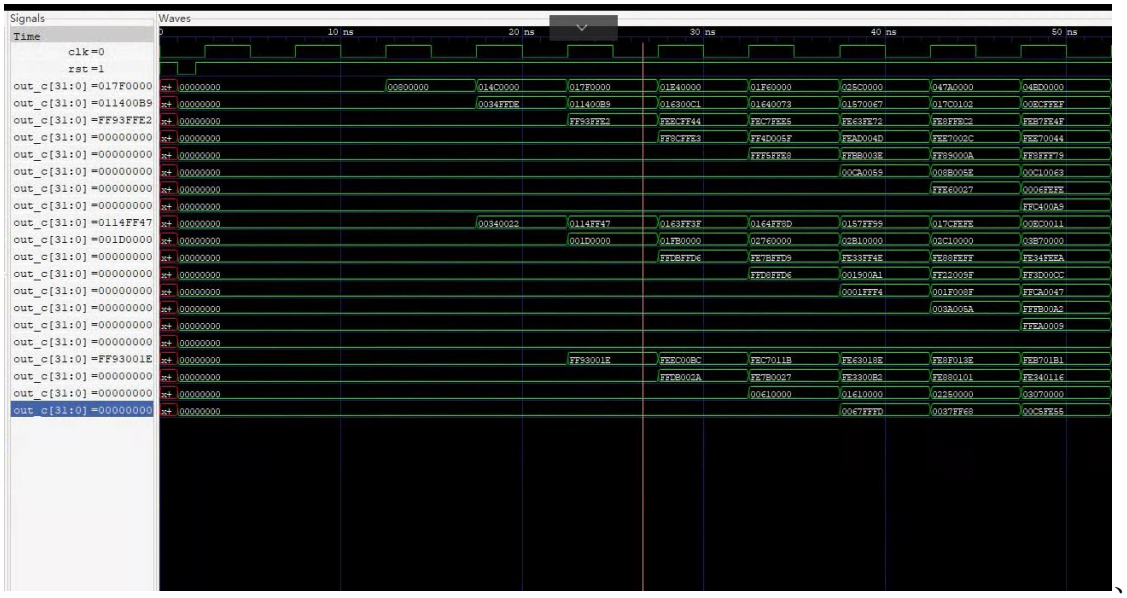
对于 PE 部分，输入的数据为每个周期进入的两个 32bit 数据，输出由三部分组成，两个个是对当前输入数据进行一个时钟的延迟后传递给后面的 PE，另一个输出是在达到某一周期后所得到的最终结果。
对于 pe_array，输入数据为按照顺序输入的 H 矩阵中的元素，同时拥有 16 个输入端口，在输出方面，共有 64 个输出端口，即 PE 单元的数量。
delay 文件用来产生部分使能信号，来负责控制电路需要的时序。
而 testbench 文件中，将已有的输入数据读取到 memory 中，同时产生地址（地址的产生较为简单，所以在其内部生成），输出相应的数据输入进模块内。
乘法器部分由 4 个基本的乘法器组成，如上图所示，以两个矩阵中的两个元素为输入，将其分实部，虚部共 4 个部分，其中高 16 位是实部，低十六位是虚部，在经过四个乘法器与两个加法器的计算得到两者之积。

3. 关键路径

在此次的架构中主要的关键路径来自于乘法器部分，即单个的 PE 单元中，在 PE 之间都存在寄存器相互连接，所以只能在其内部，我们的算法中在乘法器部分会存在这四个乘法器并行，两个加法器并行，从而产生了一个加法器一个乘法器的路径，而在乘法器外部，数据仍要与上一个周期的数据进行累加操作，所以外部存在一个加法器，所以关键路径为两个加法器与一个乘法器。



4. 结果与时序图



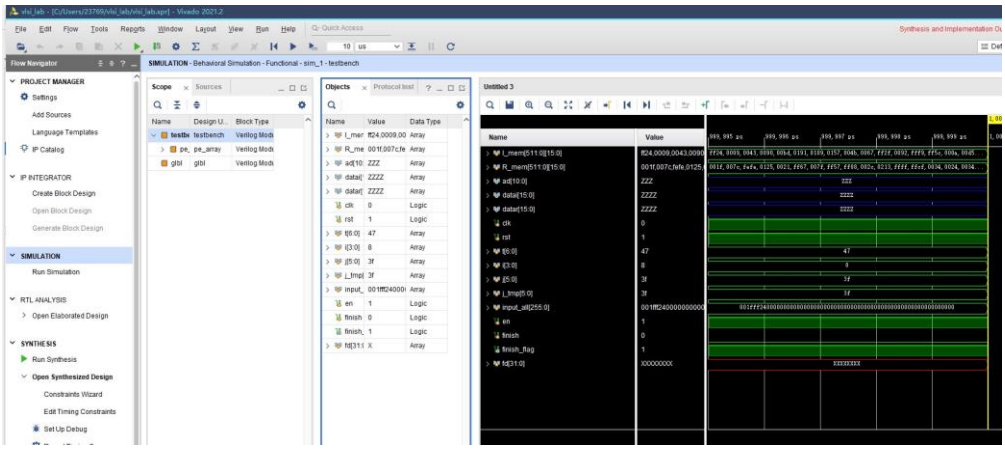
输出的开始截图满足架构的时序。

3226 + ffd6 i 0416 + fe33 i fa24 + f931 i 07ed + f576 i 019d + faf4 i f92f + f758 i 040c + ff3c i 00dd + 05b1 i
041b + 0184 i 3c12 + ffe6 i fb37 + 06bb i 0207 + 07b1 i fe94 + 06f0 i 0b12 + f9a9 i fc0f + 0cd4 i fc46 + f925 i
fa20 + 0683 i fb2f + f906 i 3dab + ffe0 i 074c + 006d i 0252 + f9d4 i f9f2 + 011b i 0868 + ff58 i f401 + 01d8 i
07e6 + 0a42 i 0201 + f806 i 0752 + ff47 i 3882 + ffdf i 024d + f63c i fc33 + fa05 i 0404 + fc2c i fee3 + fa7b i
0190 + 04c9 i fe8d + f8d6 i 024e + 05ef i 0240 + 0986 i 390a + ffe7 i fbf0 + fa20 i 0f70 + 05bf i fcf0 + 05e1 i
f939 + 085e i 0b19 + 0614 i fa06 + fea4 i fc44 + 05b1 i fc05 + 05a3 i 40a9 + ffdd i f5ce + 05fa i f646 + 022c i
03f4 + 0084 i fc00 + f2f1 i 085d + 006b i 03f5 + 038f i 0f6f + fa09 i f5b6 + f9c9 i 3d46 + ffe7 i f909 + 05c4 i
00d7 + fa11 i fc37 + 06a0 i f3fb + fdeb i fede + 054c i fcf4 + f9ed i f637 + fd96 i f916 + fa08 i 4088 + ffe5 i

输出的波形以及部分最终得到的数值，按顺序与软件所得的数据进行对比，结果存在差距，这在后面的部分会进行说明，整体的数值与理论结果相近，可以说明架构的正确性，通过观察与数据的追踪来看，两者的误差存在于实部或虚部的低五位里面，即误差范围在 0.125 以内。

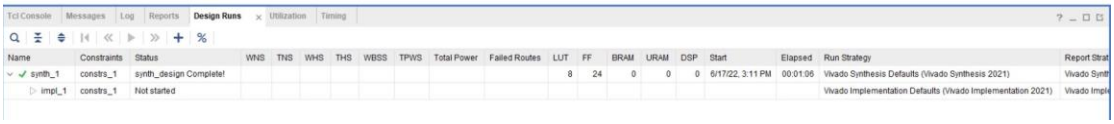
三、综合结果

波形：

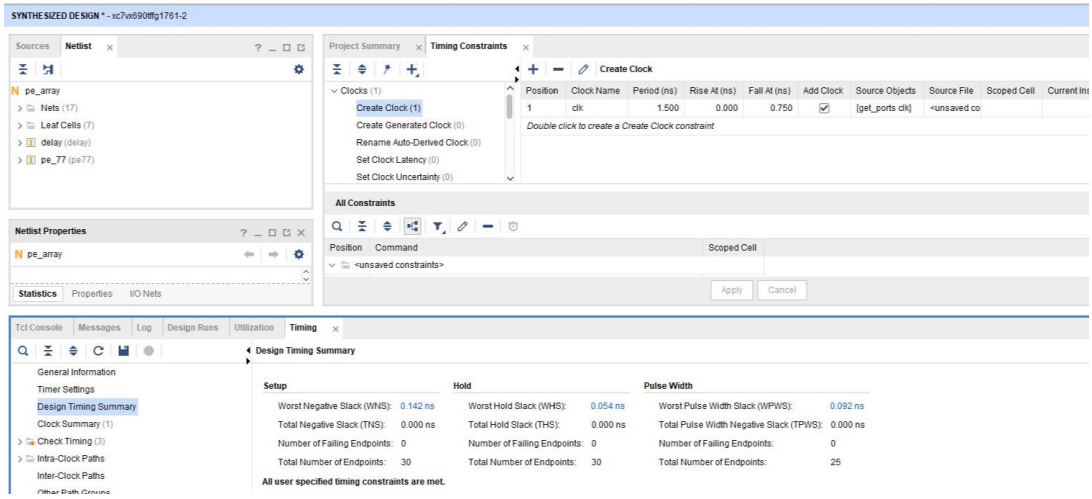


Synthesis:

资源：

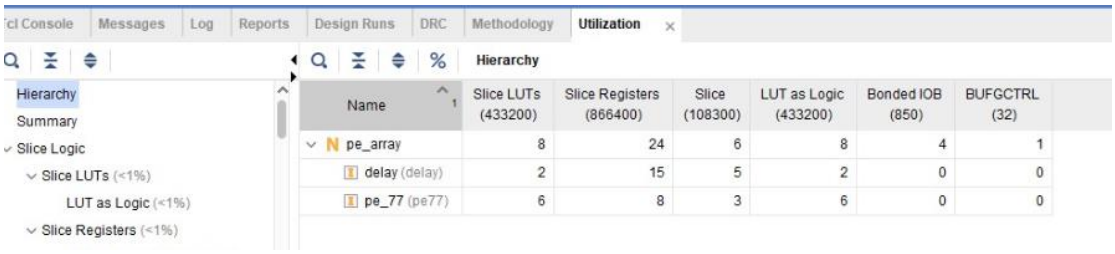


时序：

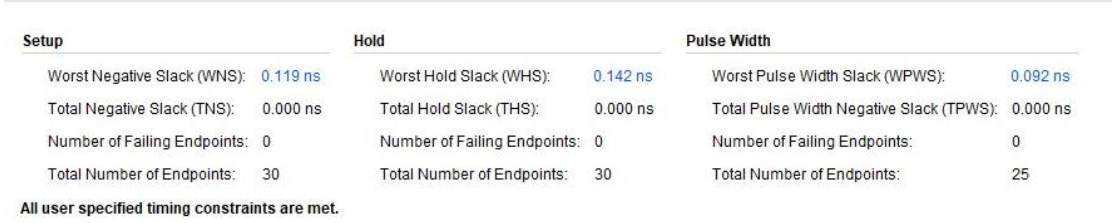


Implement：

资源：



时序：



1. 对截图中得到的数据进行提取得到表格

面积 1：

name	Slice LUTs(433200)	Slice Registers(866400)	Slice(108300)	LUTs as Logic(433200)	Bonded IOB(850)	BUFGCTRL(32)
pe_array	8	24	6	8	4	1
delay	2	15	5	2	0	0
pe_77	6	8	3	6	0	0

时钟频率：

Position	Clockname	Period(ns)	RiseAt(ns)	FallAt(ns)
1	clk	1.5	0	0.75

吞吐率：

Name	Contraints	status	LUT	FF	URAM	BRAM	DSP	Elapsed
Synth_1	Contrs_1	synth_design Complete	8	24	0	0	0	0:01:06

2. 性能的计算

计数单位的计数周期为 71(64+7)，即需要 71 个时钟周期来计算一个 Gram 矩阵，而从综合结果来看，结构可以满足 clk=1.5ns，则 fc=660MHz 所以计算得吞吐率为

Throughput=(1/71)*fclk=9.3e+6

进一步可以得到硬件效率

Hardware Efficiency=Throughput/(LUTs+FFs+DSP*280)=2.9e+5

四、遇到的问题

1. 在一开始的运行中，最终结果与实际得到的 Gram 矩阵始终存在着巨大差异，在经过排查后发现，复数矩阵的转置与常规矩阵的转置并不相同，在经过修改后得到了和实际矩阵结果接近的正确结果。

2. 在实际的运行过程中，硬件结果与软件结果始终无法完全匹配，在小数部分会存在低六位 bit 存在误差，通过逐步对比与分析，得出结论在 matlab 中的计算存在一定问题。在经过了大量的运行和多种修改后仍未得到想要结果，下面是在软硬件相匹配的过程中进行的尝试。

```
Hr2=fi(real(H1),1,16,8,fm);
G1=quantizenumeric(zeros(8,8),1,16,8,'fix');
% G_tmp=fi(zeros,1,16,8);
for x=1:8
    for y=1:8
        for n=1:64
            t1=quantizenumeric(Hr1(x,n)*Hr2(n,y),1,16,8,'fix');t1x=fi(t1,1,16,8).hex;
            t2=quantizenumeric(Hi1(x,n)*Hi2(n,y),1,16,8,'fix');t2x=fi(t2,1,16,8).hex;
            t3=quantizenumeric(Hr1(x,n)*Hi2(n,y),1,16,8,'fix');t3x=fi(t3,1,16,8).hex;
            t4=quantizenumeric(Hi1(x,n)*Hr2(n,y),1,16,8,'fix');t4x=fi(t4,1,16,8).hex;
            G_tmp(n)=quantizenumeric(t1-t2+(t3+t4)*1i,1,16,8,'fix');
%           G_tmp(n)=H2(x,n)*H1(n,y);
            G_t=fi(G_tmp,1,16,8);
            Gtxt=G_t.hex;
        end
        writematrix(Gtxt,'Gtxt')
         G1(x,y)=sum(G_tmp(:));
    end
```

在最开始的软件尝试中，我们采用了 quantizer 与 quantize 的组合来进行进一步的量化，但没有得到想要的结果，之后又尝试了 quantizenumeric 函数，但得到的结果与 fi 函数所给出的结果并无太大差距。

第二次我们尝试令过程更加细化，将每一步的计算都采用定点的形式，将数据产生多份，在一定程度上增加了接近率，但没有得到想要的结果。

第三次尝试改变精度，首先在软件上通过对精度的改变来获得不同组的数据，

```
Hi1=fi(imag(H2), 1, 16, 8, fm);
Hr1=fi(real(H2), 1, 16, 8, fm);
Hi2=fi(imag(H1), 1, 16, 8, fm);
Hr2=fi(real(H1), 1, 16, 8, fm);
```

即改变了图中“8”的大小。

但当数据的小数位大于 11 或者小于 6 时都会产生数据的溢出，所以该方法并不可通。

在 verilog 中，我们尝试将硬件中间过程的算法从原来的 32 位扩展到 64 位，来观察最终与软件的差异，最终结果虽然有所接近但是仍存在误差，

```
for x=1:8
    for y=1:8
        for n=1:64
            a=fi(Hr1(x,n), 1, 16, 13, fm);b=fi(Hi1(x,n), 1, 16, 8, fm);
            c=fi(Hr2(n,y), 1, 16, 13, fm);d=fi(Hi2(n,y), 1, 16, 8, fm);
            t1=fi(a*c, 1, 16, 12, fm);%t1x=t1.hex;disp(t1x)
            t2=fi(b*d, 1, 16, 12, fm);%t2x=t2.hex;disp(t2x)
            t3=fi(a*d, 1, 16, 13, fm);%t3x=t3.hex;disp(t3x)
            t4=fi(c*b, 1, 16, 13, fm);%t4x=t4.hex;disp(t4x)
            t5=fi(t1-t2,1,16,8,fm);%t5x=t5.hex;disp(t5x)
            t6=fi(t3+t4,1,16,8,fm);%t6x=t6.hex;;disp(t6x)
            G_tmp(n)=fi(t5+t6*fi(1i, 0, 1, 0, fm),1,16,8,fm);
        end
        gram(x,y)=sum(G_tmp(:));
```

在最后我们通过追踪过程中的值进行逐步对比校验以及人工计算，发现了产生误差的部分，但是却没有办法去更改这一状况，在排查中发现

```
t5=fi(t1-t2,1,16,8,fm);
t6=fi(t3+t4,1,16,8,fm);
```

这两部中的结果与硬件中的结果存在一定差异，在计算过程中，每次的结果与硬件相比都会在最低位的部分相差 1，-1 或者相同，但在 64 次的累加后可能会在低 6 位的部分产生误差。

五、分工

吴非：

架构的设计和架构图的绘制

verilog 硬件部分代码的编写

和贾同学讨论(debug)并修改代码。

用 python 生成了测试输入向量(1，1+i 等)。

vivado 的综合

贾鑫鹏：

软件代码的编写与修改

参与 Verilog 的调试

编写实验报告。