华中科技大学光学与电子信息学院

课程设计报告

（2018——2019 年度第 1 学期）

名　　称：数字集成电路课程设计

题　　目：简易频谱分析仪

院　　系：光学与电子信息学院

班　　级：集卓 1601 班

学生姓名：胡晓峰

同组组员：潘钦竞、陈琳、梅玉帆

指导教师：郑朝霞

成　　绩：

日　　期：2019 年 1 月 14 日

简易频率分析仪

摘要：本设计实现的电压信号频率分析仪，基于Artix－7系列FPGA，通过ADC采样电压信号，实现对输入信号进行频率、幅度分析功能，并用LCD电容屏进行打印与显示峰值。系统由ADC采样与存储模块、FFT计算模块和串口与显示模块三部分组成。ADC采样与存储模块由AD8865芯片和片上RAM组成，可实现对数据的采集和存储。FFT计算模块可实现1024点64bit复数输入的离散傅里叶变换计算。串口与显示模块由 UART 串口模块、电容屏显示模块和峰值检测模块和构成，可以实现数据的传递与显示。

关键词：FFT，频率分析，电压检测，串口通信

目录

一、 系统方案

1.1 系统简介：

系统由 ADC 模块为输入，两块 FPGA 板分别负责 FFT 计算和数据存储与显示，板间用 UART 串口模块通信，TFT 电容屏打印输出计算结果。

数据通路简述：第一块 FPGA 通过按键控制 ADC 开始采样 1024 个点，采样率 200KHz，采样信号存入 FIFO。采集完成后发送使能信号给 FFT 模块开始一次 FFT 转换，FFT 模块计算完毕之后输出给串口模块，通过 UART 有线通信，数据传递到第二块板上存储至一个 FIFO，并经由 AXI 总线由 Microblaze 软核处理器进行读取，完成一次读取之后进行显示刷新，至此完成一次测量。

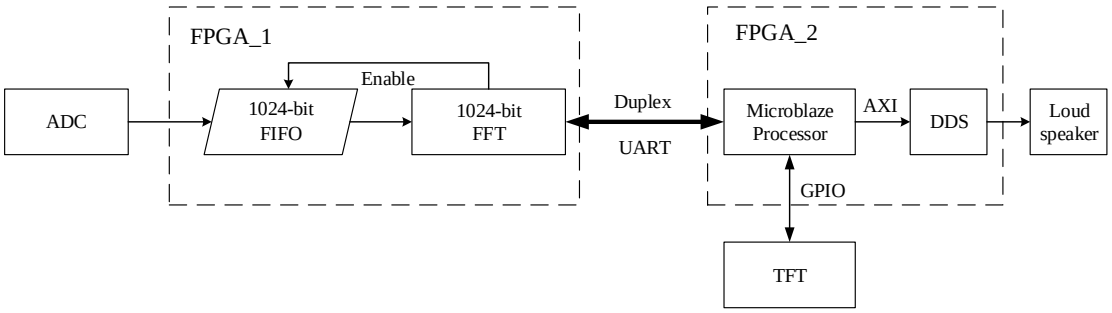


图 1 整体系统框图

1.2 方案描述：

FFT 模块：本设计采用了 64-bit 复数输入，Radix-4 的 FFT 算法，输入序列按照时间顺序抽取，串行输入，串行输出，采用了精度为 10-bit 的旋转因子进行计算。复位后，当模块接受到输入使能时开始把 FIFO 输入存至本地的 BRAM 中，输入 1024 点完毕后 FSM 进入计算状态，由于序列总共有 1024 个点，采取基-4 算法总共需要计算 5 次抽取，每次抽取由 256 个蝶形运算分三级流水进行，两个 RAM 轮流作为输入和输出进行乒乓操作。计算完毕后串行输出 1024 点的计算结果。

显示与峰值检测模块： 采用 LCD 电容屏输出，读写时序采用 I2C 协议读写寄存器实现电容屏交互。峰值检测与打印频谱函数共用一个循环，当串口接受了 1024 帧数据时开始对频谱进行长度归一并打印。峰值检测模块用长度为 10 的窗，当峰值大于窗内另外九个值平均的五倍时，在屏幕上打印对应峰值的幅值和频率。

完成 FFT 模块之后后续的工作在于用 AXI 总线将数据依次读取到 Microblaze 软核并编写显示函数打印频谱，该工作较繁琐，且并不在本设计的算法核心范畴内，所以在此不做过多介绍。

1.3 本设计的特点：

本设计主要看重了模块的通用性和并行性，在通用性上，采用了大部分总线通行的 32 位接口，实际数据位宽为 64 位（32bit 实部+32bit 虚部）。在并行性上，蝶形操作的输入输出采用四组三级缓冲并行操作，同时在存储上用了两组 RAM 组进行乒乓操作。在蝶形单元内运用了 modified booth 乘法器，采用三级流水进行部分积的加法，在最大化并行性的同时兼顾面积。最终模块可以在 1300 个时钟周期内完成一次 FFT 的全部计算工作， 资源的占用以及时序分析会在（四）内详细分析。

二、 理论分析与模块详解

2.1 FFT 模块功能描述

FFT 模块整体框图：

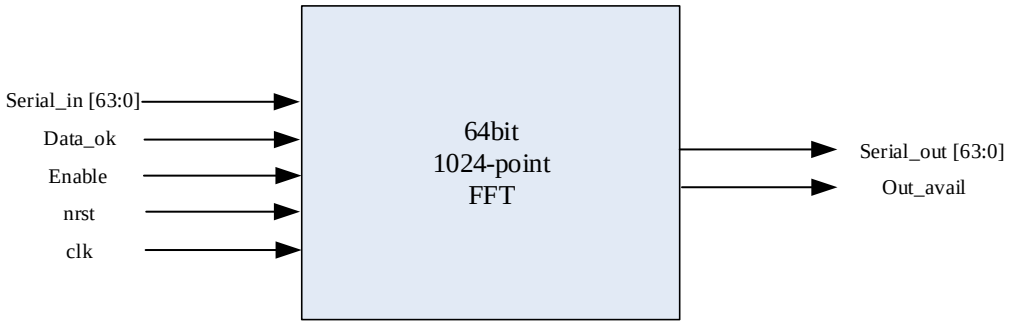


图 2 FFT 模块整体框图

模块内部互联框图：

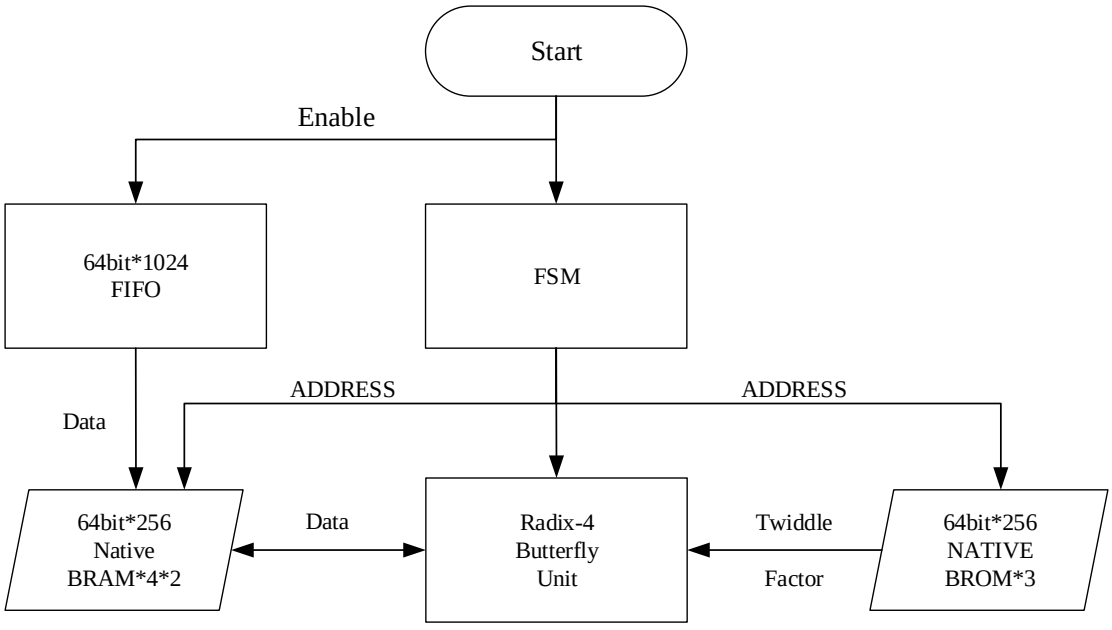


图 3 模块内部互联

接口及功能特点说明：

1. 接口声明

Interface	Declaration	Direction
Serial_in	串行输入 64bit 数据，高 32 位实部+低 32 位虚部	Input
Data_ok	输入数据有效标志，1bit	Input
Enable	FFT 模块使能信号，1bit，高电平有效	Input
nrst	异步复位信号，1bit，低电平有效	Input
clk	模块总时钟	Input
out_avail	输出结果有效标志，1bit，高电平指示有效	Output
Serial_out	串行输出 64bit 数据，高 32 位实部+低 32 位虚部	Output

表 1 FFT 模块接口定义

2. 模块功能说明：

该模块实现了 64 位 1024 点的快速傅里叶变换，以基-4 蝶形单元为基础，有限状态机作为主控，在 100M 时钟主频下完成全部 1024 点的计算需要 33.79us, 其中计算部分耗时 13.25us，串行输入、输出占 20.54us。板载 Xilinx Artix-7 系列的 FPGA，100M 晶振时钟，资源使用以及极限频率等报告在下一章节给出。

FFT 内部并行模块说明及状态转移图

该模块内部从顶层向下主要有三个子模块: 1. 基 4 蝶形运算单元，其中包括了 64*64 复数乘法器 2.状态机控制及缓冲寄存器组 3. RAM/ROM 地址产生模块。

1. 64bit Radix-4 蝶形运算单元。运算单元以 ROM/RAM 的数据为输入，采用了四周期流水，分别对输入做 modified booth 编码，16 个部分积相加（三周期完成），输出运算与寄存器赋值。由于采用了基-4 蝶形算法，所以同时需要并行的 4 输入与 4 输出端口，又考虑到系统的并行性，本设计采用了 8 个深度为 256 的 BRAM 分为 2 组进行乒乓操作，1024 点序列全部计算完毕共需要 5 次抽取。

2. 状态机控制及缓冲寄存器组。由于基 4 蝶形同时会有 4 个并行输入和四个并行输出，并且本设计采用的排序方法需要单次计算的四个输出写入同一个 RAM 中，所以设计采用了三级缓冲。进入计算后，四组缓冲器同时工作完成数据输入。FSM 与缓冲寄存器组协同工作完成输出数据的传递，FSM 负责整体状态的控制，RAM/ROM 读写使能的产生，缓冲寄存器的读写控制，缓冲寄存器负责将输出串行存至每次作为输出的那一组 BRAM 中。状态转移表如下：

Stages	Block RAM 1	Block RAM 2
Input	Write	-
Stage 0	Read	Write
Stage 1	Write	Read
Stage 2	Read	Write
Stage 3	Write	Read
Stage 4	Read	Write
Output	-	Read

表 2 FFT 模块状态转移图

3. RAM/ROM 地址产生模块。以 FSM 的主控计数器为标志，通过判断不同的计算状态，地址产生模块会产生相应状态需要的 RAM/ROM 地址，其中 RAM 存放计算结果序列，ROM 存放旋转因子。

时序关系图

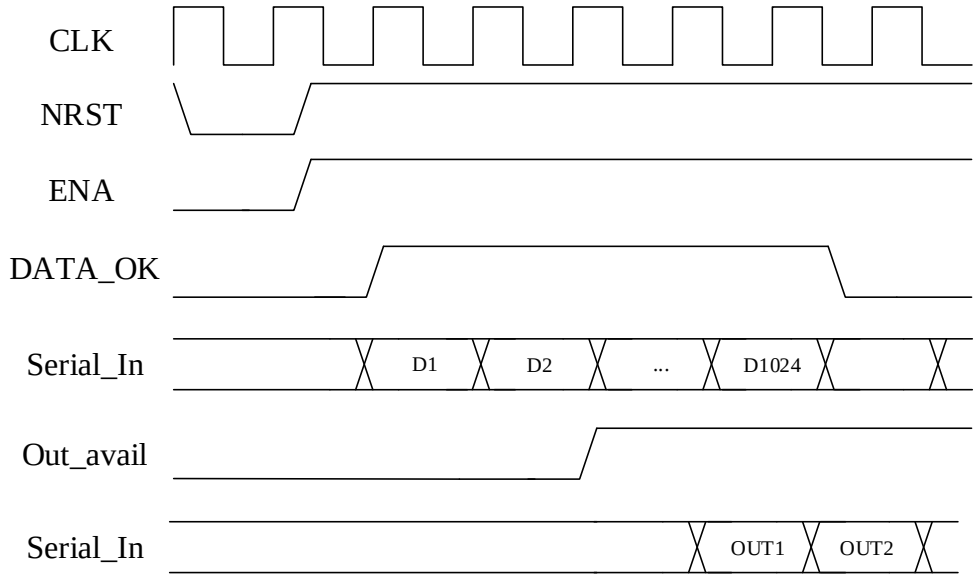


图 4 FFT 模块时序关系图（作图：visio 2016）

对时序关系图的说明：复位后，当 ENA 为高时系统进入正常工作状态，DATA_OK 标志为高时开始接受 1024 帧数据，1024 帧数据结束后，系统进入计算阶段，计算结束后 Out_avail 标志为 1 时输出信号有效，串行输出 1024 帧数据。输出、输入的高 32 位为虚部，低 32 位为实部。

基-4 FFT 原理简述

根据根据 DFT 的定义 $X(k)=\sum_{n=0}^{N-1}x(n)W_N^{kn}$，按时间基 4 抽取整个序列，可得

$$X(k)=\sum_{m=0}^{N/4-1}x(4m)W_N^{k\cdot 4m}+W_N^k\sum_{m=0}^{N/4-1}x(4m+1)W_{N/4}^{k\cdot m}$$
$$+W_N^{2k}\sum_{m=0}^{N/4-1}x(4m+2)W_{N/4}^{k\cdot m}+W_N^{3k}\sum_{m=0}^{N/4-1}x(4m+3)W_{N/4}^{k\cdot m}$$

令 $Y_0(k)=\sum_{m=0}^{N/4-1}x(4m)W_N^{k\cdot 4m}$ $Y_1(k)=\sum_{m=0}^{N/4-1}x(4m+1)W_N^{k\cdot 4m}$

$Y_2(k)=\sum_{m=0}^{N/4-1}x(4m+2)W_{N/4}^{k\cdot m}$，$Y_3(k)=\sum_{m=0}^{N/4-1}x(4m+3)W_N^{k\cdot 4m}$，m=0,1,2…,N/4-1

扩展到 N 个点，即 N 点的 DFT 序列被分解成 4 个 N/4 点子序列，

$$X(k)=Y_0(k)+W_N^kY_1(k)+W_N^{2k}Y_2(k)+W_N^{3k}Y_3(k)$$
$$X(k+\frac{N}{4})=Y_0(k)-jW_N^kY_1(k)-W_N^{2k}Y_2(k)+jW_N^{3k}Y_3(k)$$
$$X(k+\frac{2N}{4})=Y_0(k)-W_N^kY_1(k)+W_N^{2k}Y_2(k)-W_N^{3k}Y_3(k)$$
$$X(k+\frac{3N}{4})=Y_0(k)+jW_N^kY_1(k)-W_N^{2k}Y_2(k)-jW_N^{3k}Y_3(k)$$

令：$U_0=Y_0(k)+W_N^{2k}Y_2(k)$，$U_1=Y_0(k)-W_N^{2k}Y_2(k)$

$V_0=W_N^kY_1(k)+W_N^{3k}Y_3(k)$，$V_1=W_N^kY_1(k)-W_N^{3k}Y_3(k)$

继续化简可得：$Z_0=U_0+V_0$，$Z_1=U_1-jV_1$，$Z_2=U_0-V_0$，$Z_3=U_1+jV_1$

以上即为一次蝶型运算过程。蝶型运算单元按输入的 *k* 和 *FN* 信号选择适当旋转因子参数 W_N^{ik} 分别与输入的 *x1,x2,x3* 相乘。

蝶形运算的一次复数乘法包含 4 次实数乘法和 2 次实数加/减法。

按照上述不断抽取，1024 点序列的 DFT 可以被 5 次抽取简化成 256 个 4 点 DFT 的线性组合。

序列抽取方式

64 位串行数据顺序输入到存储器空间，存放顺序如下：

x																
RAM0	0	1	2	3	4	…	…	…	…	…	…	…	…	…	…	255
RAM1	256	257	258	259	260	…	…	…	…	…	…	…	…	…	…	511
RAM2	512	513	514	515	516	…	…	…	…	…	…	…	…	…	…	717
RAM3	768	769	770	771	712	…	…	…	…	…	…	…	…	…	…	1023

表 3 输入存放顺序

第一次抽取按照上表 RAM 的地址顺序进行，本设计采用的是按照 k 值（时间顺序，也即计算后的频率）存放数据，为了五次计算的输出数据存放方式一致，每次抽取的顺序需要进行一定的改变，第二次抽取的顺序为：首先抽取 k=0 的所有序列（见表 4），按照首项 0-64-128-192-1-65-129-193….的顺序进行 64 次抽取，64 次抽取完毕之后按照相同的顺序分别抽取 k=1,k=2,k=3 的序列，这样抽取的好处是能确保一次抽取的输出经过三级缓冲输入同一个 RAM，四组缓冲轮流写入，大大提高了读写的并行性。 事实上，本设计采用的五次抽取的存放顺序和位置均一样，但是抽取序列在 RAM 中的位置会因为计算整体序列的变长而改变，造成改变的原因是随着计算序列的不断变长,k 的取值（频率量）在不断扩充，以后的抽取与第二次抽取类似，遵循着：按照 k 值从小到大分组抽取，输出结果不变后续抽取过程不再赘述。

RAM 块说明：(4n)序列结果存入第一个 RAM0，(4N+1)序列存入 RAM1，(4N+2)序列存入 RAM2，(4N+3)序列存入 RAM3，存储序列由四组深度为 3 的缓冲寄存器组按照三级流水顺序完成。(第一次抽取中的前两组输入以及计算结果的存放位置已在表中分别标记黄色和绿色)

x	k=0				k=1				k=2				k=3			
RAM0	0	4	…	252	0	4	…	252	0	4	…	252	0	4	…	252
RAM1	1	5	…	253	1	5	…	253	1	5	…	253	1	5	…	253
RAM2	2	6	…	254	2	6	…	254	2	6	…	254	2	6	…	254
RAM3	3	7	…	255	3	7	…	255	3	7	…	255	3	7	…	255

表 4 第一次抽取结果存放顺序

表格说明：表 4 中的每一个数字都代表了以该项为首的一个四点序列 DFT（如 k=1 列的“0”即为序列“x(0),x(256),x(512),x(768)”经过蝶形后得出的第一个输出）。

2.2 Booth 乘法器模块功能描述

使用 Modifed Booth 算法对乘数进行重编码可减少一半加法次数。Booth 编码器对乘数一次检查 3 个位，以决定对被乘数进行如下操作：（1）相加（2）左移一位相加（3）相减（4）左移一位相减（5）不操作。Booth 编码器要求乘数为偶数，并在乘数最右侧添零，若乘数为奇数则符号扩展一位。

蝶形运算的一次复数乘法包含 4 次实数乘法和 2 次实数加/减法

$$(A+jB)(C+jD)=(AC-BD)+j(AD+BC)$$

若将乘数扩大 1 位计算(A-B), (C+D), (C-D)可将计算化简为 3 次实数乘和 5 次实数加/减法：

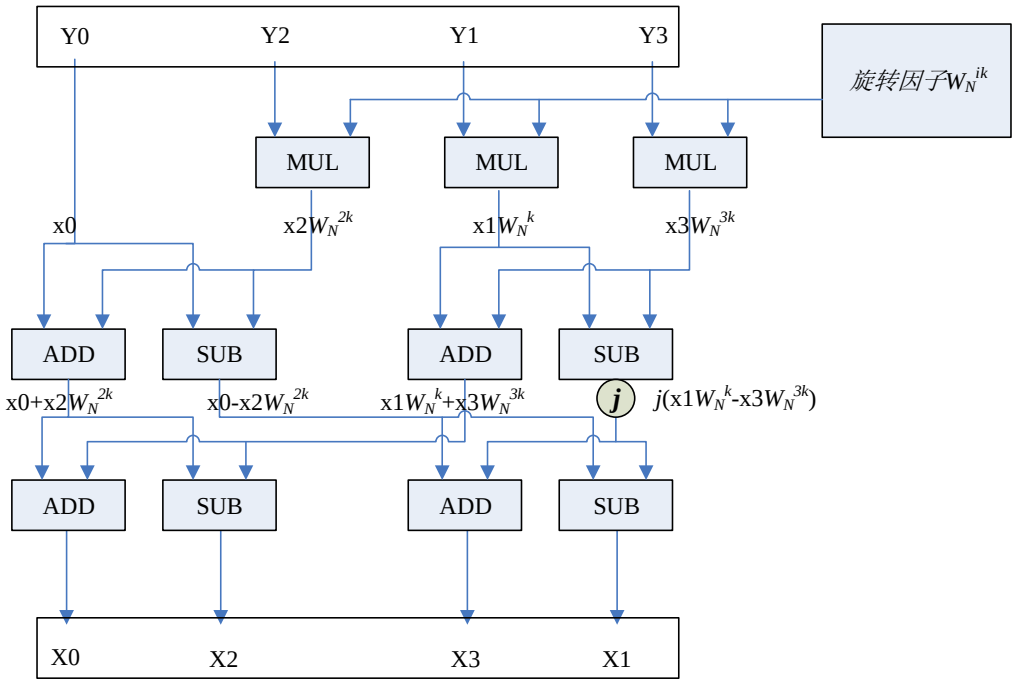
$$(A+jB)(C+jD)=[B(C-D)+C(A-B)]+j[A(C+D)-C(A-B)]$$

m_{i+1}	m_i	m_{i-1}	值	操作
0	0	0	0	不操作
0	0	1	+1	相加
0	1	0	+1	相加
0	1	1	+2	移一位相加
1	0	0	−2	移一位相减
1	0	1	−1	相减
1	1	0	−1	相减
1	1	1	0	不操作

表 5　Modified Booth 编码

基-4 蝶形单元

框图如下：



蝶形单元示意框图

三、 功能仿真结果

3.1 Booth 乘法器仿真结果

由于 ADC 输入为 12 位有符号数，则对 booth 乘法器的测试模块中，两个乘数分别在$[-2^{11}, 2^{11}-1]$范围内线性取值。测试同时直接用两数相乘给出正确结果与模块计算结果相比较，若 booth 得出的结果与直接相乘的结果不一致，则错误计数器 cnt 加一。可以看到，在仿真 10000 次不同输入组合后，cnt 仍然为 0，则认为该乘法器在功能上无误。

图中 wright_prod 为直接相乘所得结果，part_prod 为 booth 乘法器计算结果，mult1,mult2 为两个乘数输入，cnt 为指示错误计数器。

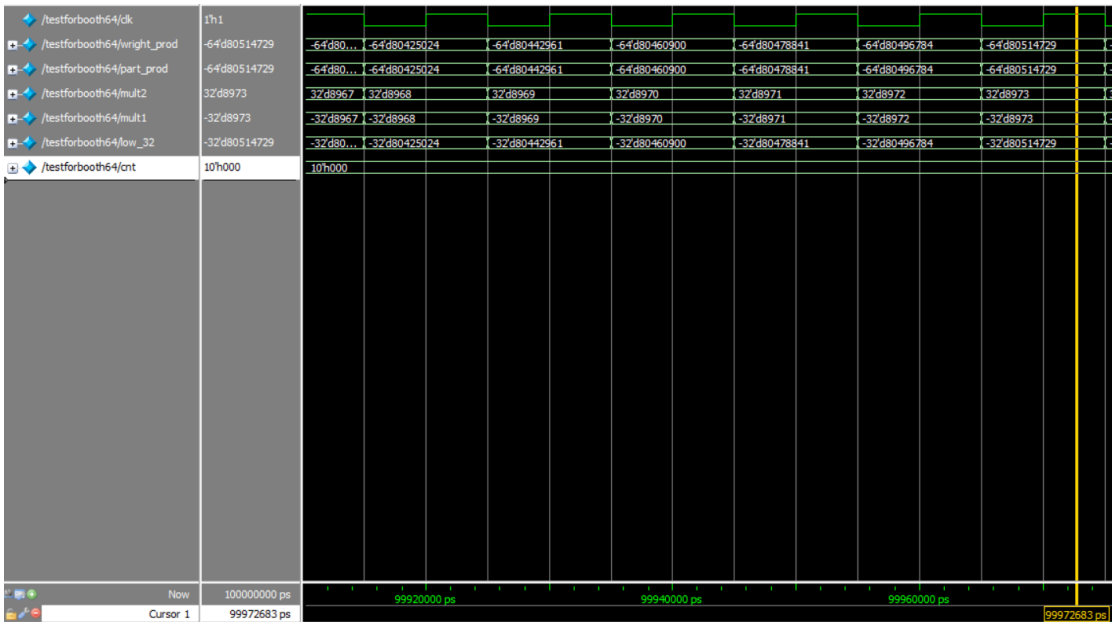


图 5 Booth 乘法器仿真图

3.2 蝶形单元、fft 模块功能仿真及 matlab 验证

由于使用了 Vivado BRAM IP, 以下采用了 Vivado 2017.4 Simulator 进行仿真。

该测试模块使用了 1024 点方波序列{256*sb0；256*sb1024；256*sb0；256*sb1024}进行功能测试，可以看到复位后收到 data_ok 持续高电平信号则模块进入输入模式（state=0），ram00,ram01,ran02,ram03 的写数据依次有效 256 个周期，写地址按照（二）中所说为顺序存放，由 FSM 产生，当所有数据写入完毕之后，FSM 控制状态转移至计算状态。

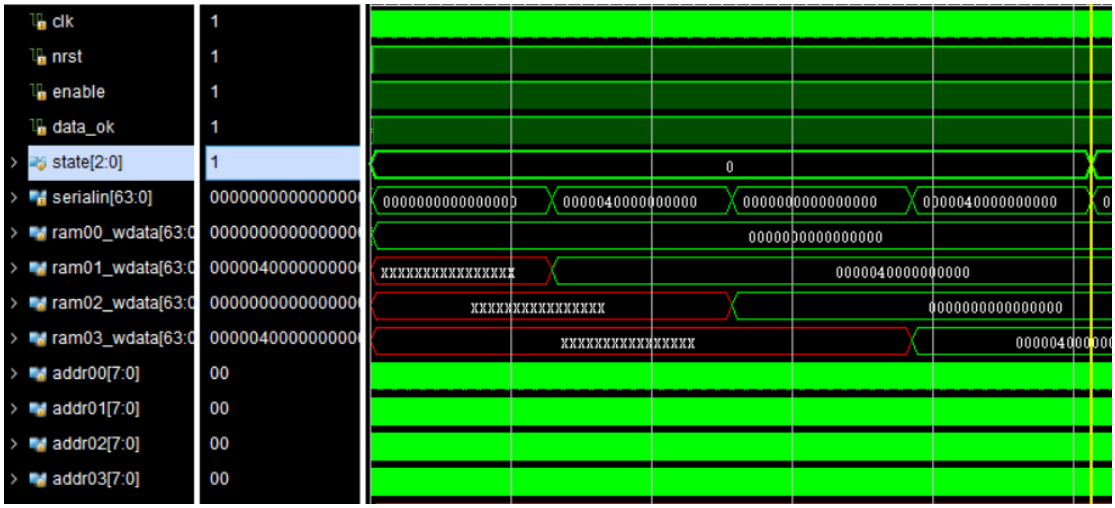


图 6 数据输入状态

图 7 为数据输入完毕后 state 进入第一个计算周期（“0”-“1”，第一条蓝线所示处），Y 为蝶形输入，X 为蝶形输出，fft_available 指示输出有效，可以看到当 Y 有效之后（第一条蓝线），经过三级流水 X0 有效（第二条蓝线），fft_available 随即置高，缓冲开始载入数据（第三条蓝线，对应变量为 buf_wdata0），可以看到 buf_waddr0 信号在缓冲载入之后开始跳变为相应的地址，由于是第一次抽取，所以第一次计算结果数据应该存在第一个 BRAM 的 00H，40H，80H，E0H 的位置。后三级缓冲没有列出，情况相似。

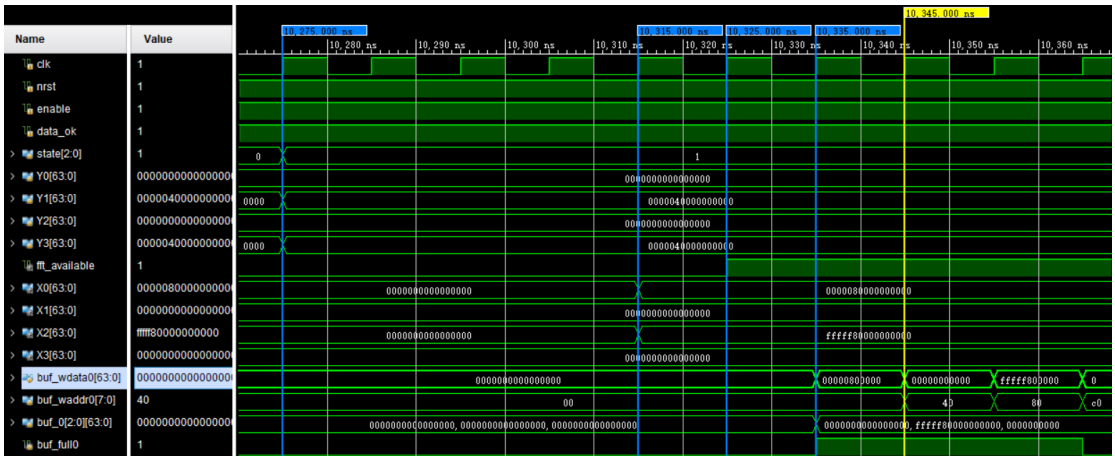


图 7 蝶形模块输入输出与数据缓冲

图 8 为第一次抽取计算完毕之后的状态转换，当所有缓冲数据都写入 RAM 之后，STATE 寄存器会进入状态 2，之后开始第二轮的抽取，可以看到 buffer_0 内的数据在第一个蓝色光标处写完（buf_full0 信号已不再有效），由于总共有四个缓冲，另外三个信号并未列出，等待另外三个缓冲写入都完成后，FSM 控制模块进入了第二次抽取，伴随着状态改变的是图 9、图 10 中 ROM 读地址的改变（旋转因子的改变）以及 buffer 写地址的改变。

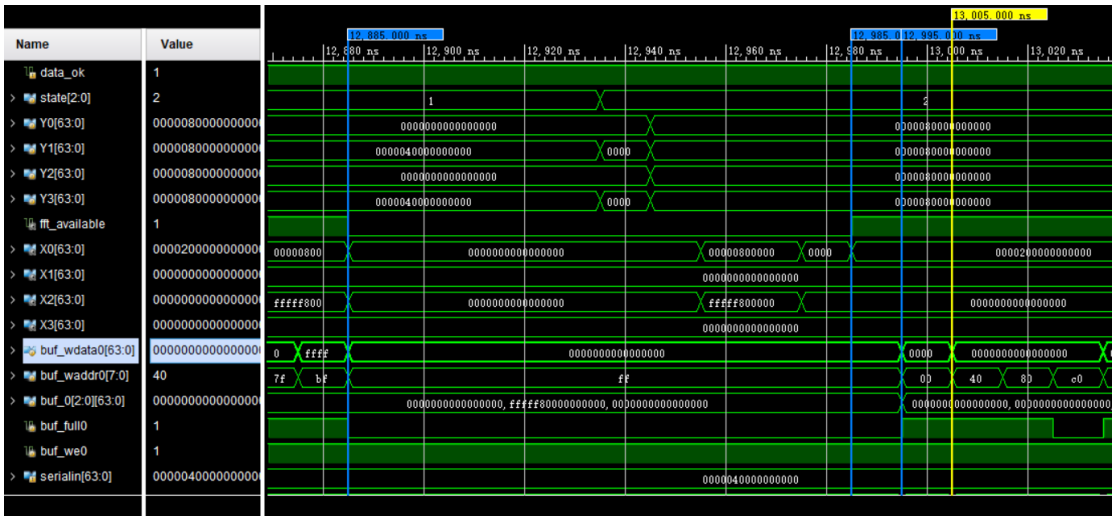


图 8 状态转移

图 9 中可以看到 rom 的读地址（rom_addr）改变，标志着蝶形变换相应的旋转因子在不同计算周期下会发生相应改变。（TF1,TF2,TF3）

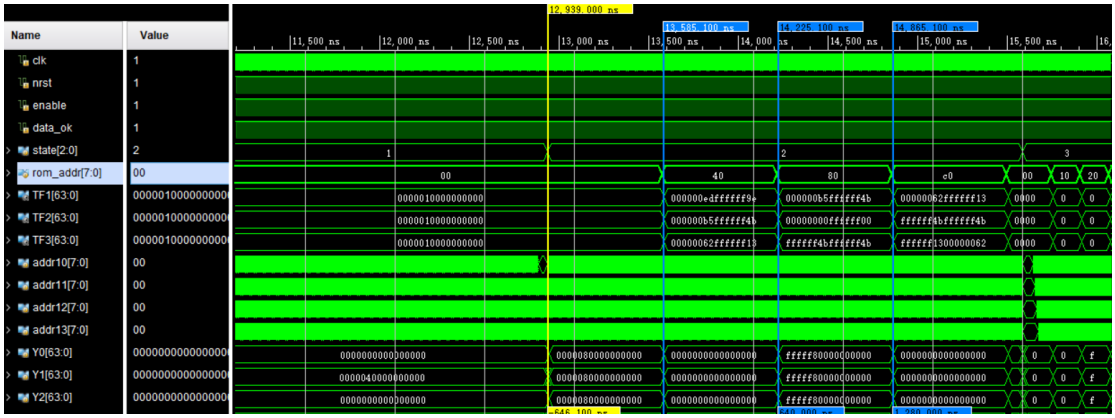


图 9 ROM 读地址相应改变

图 10 可以看到 ram 读地址（addr1_）序列在状态 2 发生了变化，由状态 1 中 00H-40H-80H-E0H 变成了 00H-10H-20H-30H（光标），这与（二）中的描述一致。

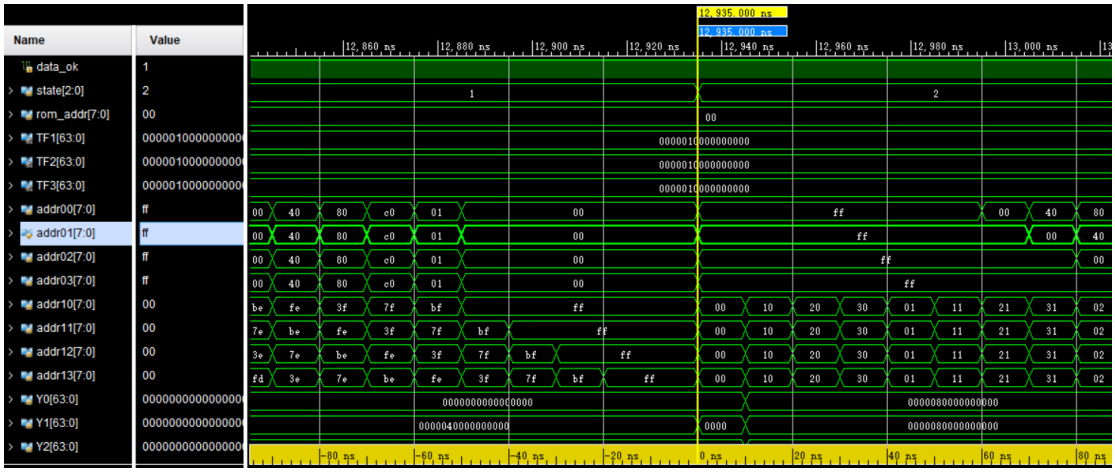


图 10 抽取序列相应的读地址改变

图 11 为五次计算结束后的串行输出，可以看到状态由“5”转变成了“6”，紧接着 out_avail 信号置高指示输出有效，开始串行输出。测试模块采用了两种不同的方波序列进行测试，用 matlab 计算该序列得出参考结果与模块计算出的结果做出比较，结果在下表中列出。

表 6 为序列 1:{256*32’b0,256*32’d1024,256*32’d0,256*d0}的 FFT 计算结果对比，可以看到在直流和主频以及谐波分量上计算结果与 Matlab 计算结果差别在 1%以内。

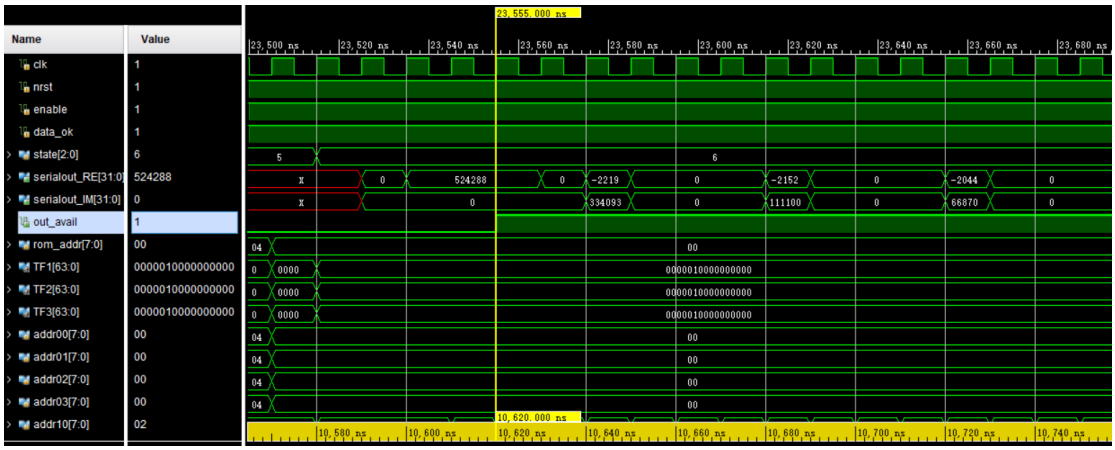


图 11 序列 1 输出结果

k	0	1	2	3	…	1022	1023
Matlab 实部	520190	38	2047	-113	…	-2047	38
FFT 实部	524288	0	2219	0	…	2177	0
Matlab 虚部	0	4096	336900	4096	…	-336900	-4096
FFT 虚部	0	0	334093	0	…	-334113	0

表 6 序列 1 功能验证结果

表 7 为序列 2:{0x0,0x400,0x0,0x400.....}的 FFT 计算结果对比，该序列与 matlab 计算结果完全一致，图 12 为相应仿真波形，cntd2 为输出计数。

k	0	1	…	512	…	1022	1023
Matlab 实部	524288	0	…	-524288	…	0	0
FFT 实部	524288	0	…	-524288	…	0	0
Matlab 虚部	0	0	…	0	…	0	0
FFT 虚部	0	0	…	0	…	0	0

表 7 序列 2 功能验证结果

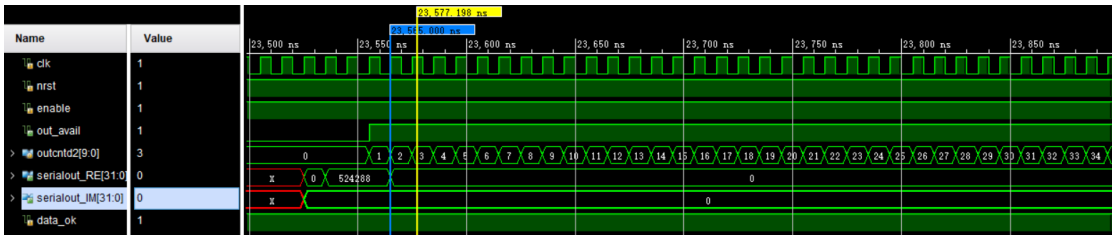


图 12 序列 2 输出结果

四、 综合实现结果及资源占用

4.1 时序约束与 Timing_report 分析

以下为在 100MHz 片上晶振时钟的频率下的时序报告，所有时序要求均满足，值得一提的是，由于一开始 booth 乘法器的部分和序列并没有采用流水，直接将 16 个部分积相加，出现了 22ns 以上的 holdup_slack，修改逻辑之后将部分积加法分三级进行，slack 大大改善，Timing_failed 状况也被解决（见图 13 与图 14）。

```
160     always@(posedge clk)
161     begin
162         part_prodA<=part_A1[31:0]+
163                     {part_A2[29:0],2'sb0};
164         part_prodE<={part_A3[27:0],4'sb0}+
165                     {part_A4[25:0],6'sb0};
166         part_prodB<={part_A5[23:0],8'sb0}+
167                     {part_A6[21:0],10'sb0};
168         part_prodF<={part_A7[19:0],12'sb0}+
169                     {part_A8[17:0],14'sb0};
170         part_prodC<={part_A9[15:0],16'sb0}+
171                     {part_AA[13:0],18'sb0};
172         part_prodG<={part_AB[11:0],20'sb0}+
173                     {part_AC[9:0],22'sb0};
174         part_prodD<={part_AD[7:0],24'sb0}+
175                     {part_AE[5:0],26'sb0};
176         part_prodH<={part_AF[3:0],28'sb0}+
177                     {part_A0[1:0],30'sb0};
178
179         part_prod1<=part_prodA+part_prodB+part_prodC+part_prodD;
180         part_prod2<=part_prodE+part_prodF+part_prodG+part_prodH;
181
182         part_prod<=part_prod1+part_prod2;
183     end
184
```

图 13 修改后的三级流水实现部分积加法

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.465 ns	Worst Hold Slack (WHS):	0.034 ns	Worst Pulse Width Slack (WPWS):	3.750 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:	19223	Total Number of Endpoints:	19223	Total Number of Endpoints:	10463

All user specified timing constraints are met.

图 14 修改后时序约束通过的时序报告

通过查看时序报告，可以发现关键路径产生在部分积加法阵列中。

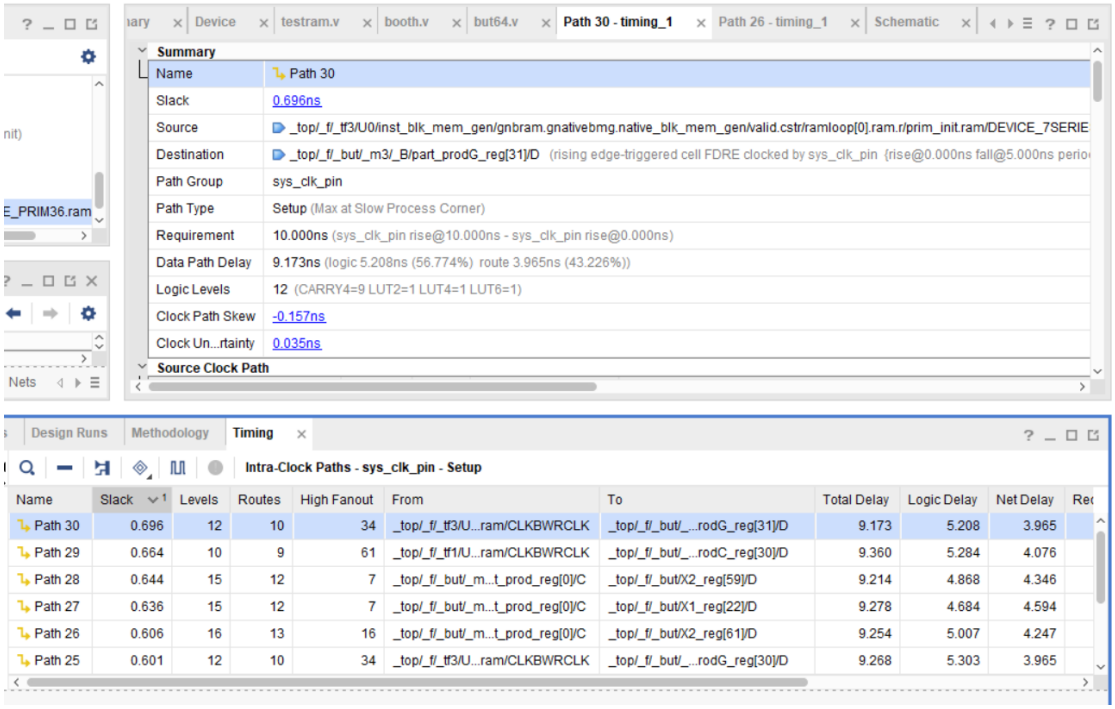


图 15 出现 slack 最大的路径为关键路径

观察图 16 可发现，时许报告指出，所有的 slack 均发生在部分积加法阵列。

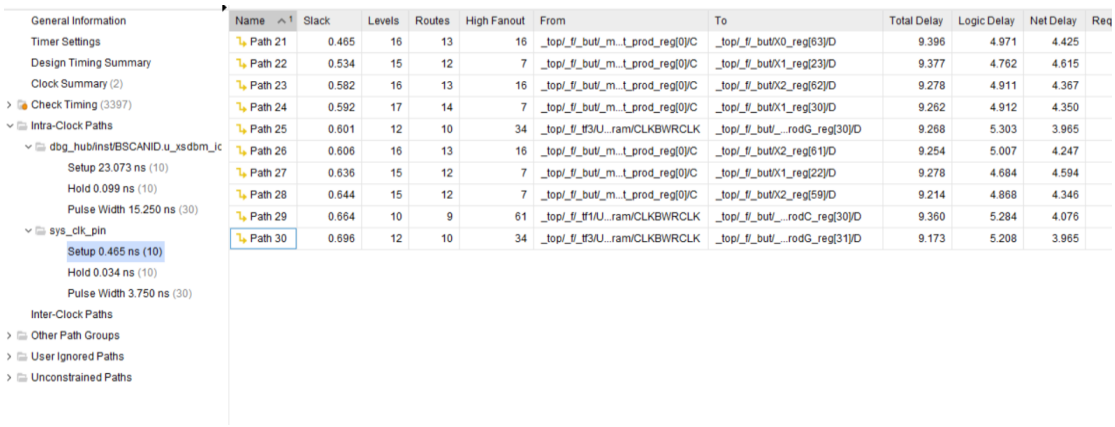


图 16 所有时序 slack 均发生在部分积加法阵列

时序小结：根据时序报告，制约系统工作频率的为部分积加法阵列，采用 vivado 直接综合得到的 64 位加法器会被综合成十六个逐位进位加法器(图 17)，进位信号的传递与储存大大增大了延迟与面积（Slice LUT）。由于时间原因没有更多时间优化加法阵列，初步的优化想法是用 Wallace 树圈划的方式进行压缩部分积阵列，可以大大简化加法的逻辑，进而减少面积并改善时序要求。

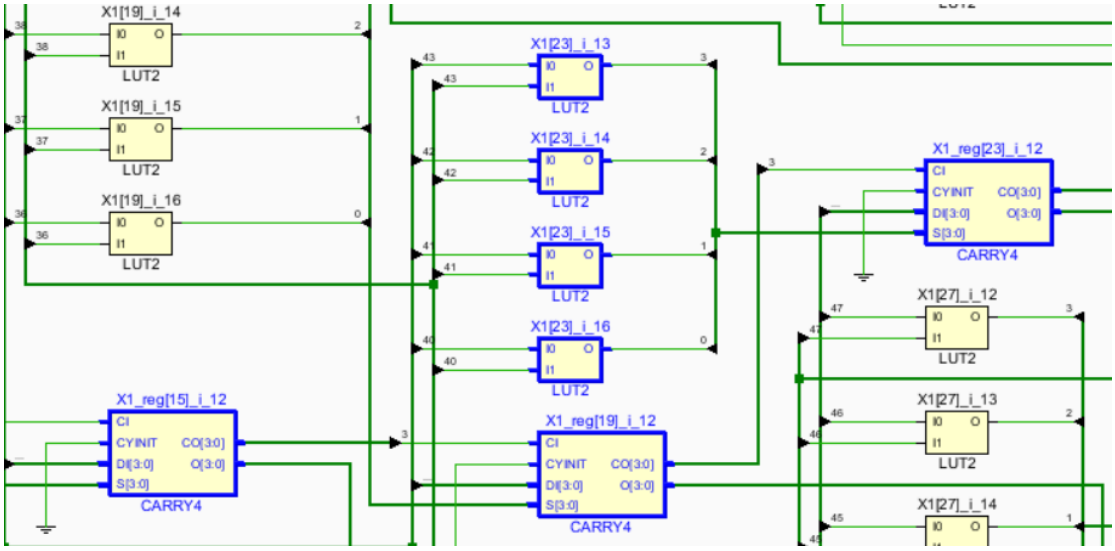


图 17 部分积加法阵列被综合成了 4 位加法器链与查找表

4.2 资源占用分析与对比

资源占用情况已在表 8 中列出，并与 Vivado 提供的 FFT Logic-core 进行了对比。由 implementation report 可知百分之八十以上的 LUT 都用来构建了部分积加法阵列（图 18），由于采用了三级流水，面积更是以三倍同等增加，由于时间原因无法继续优化加法器部分，这也是本设计的一个缺陷与遗憾。

Parameter	My design	Xilinx Logic Core
Channel	1	1
Point Size	1024	1024
Input Data Width	64	32
Phase Factor Width	32	16
Xilinx Part	XC7a35tftg256-1	XC7a35tftg256-1
Slice LUTs	8262	2750
Slice Registers	3918	1608
BRAMs	11	10
Latency(clock cycle)	3379	3453

表 8 资源占用对比

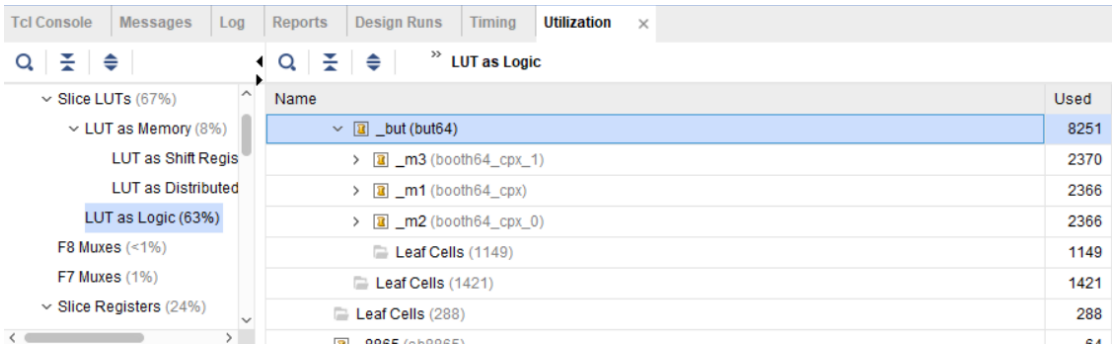


图 18 implementation 结果

五、 测试效果展示与误差分析

5.1 作品展示

作品的展示由录制的视频给出。

5.2 误差分析

在频率测量过程中，可以发现对于各分量的频率测量频谱分析仪可以得到十分准确的结果（误差均不超过 1%），但是对幅度的测量在多频信号的测试中可以明显发现偏离理论值，主要的原因为：1.10bit 旋转因子带来的量化误差，2.由于 ADC 采样造成频谱泄露带来的误差。下面逐点分析。

由旋转因子位数带来的量化误差与优化

由旋转因子带来的误差无法避免，会存在最大$[-1024*\frac{1}{2^{-10}}，1024*\frac{1}{2^{-10}}]$的误差，可通过扩大旋转因子的量化位数来改善。

由频谱泄露（spectral leakage）带来的误差与优化

一般来说，频谱泄露都是相对于窗函数而言，对于我们通常处理的离散信号，无论是周期或是非周期，都要进行截断后进行运算，ADC 进行的采样就是一种截断，而截断就是一种最简单的窗函数，也即在定义域内为 1，非定义域内为 0。时域进行的所有操作都相当将信号与特定的窗函数相乘，相当于频域卷积，窗函数会带来频谱中本来不应该存在的频域分量，这称为频谱泄露(spectral leakage)。由于傅里叶变换将信号进行了周期延拓，所以当截取周期与信号周期不一致或者不成整数倍时，通过周期延拓无法获得原来的信号，必然会引入新的频域分量。以上，对非周期信号的所有截取与周期性信号的非周期性截取都必然会带来频谱

泄露。由于频谱泄露，所有主瓣分量附近出现幅值较大的旁瓣带来误差，这也是在测量多频信号中幅值分量偏离傅里叶展开级数关系的最大原因。

改善频谱泄露的手段是针对不同的分析要求，在信号采样时对信号乘上不同的加权窗函数（如分析频率时用 Blackman 窗，分析幅值时用 Hamming 窗），由于课设时间有限，考试时间临近，本设计并没有采用加窗技术。

六、问题分析与总结致谢

6.1 问题汇总

本次设计从最底层的乘法器开始搭建，总体流程：booth 乘法器-蝶形单元-FFT 状态控制 FSM-RAM/ROM 组读写控制-FFT 整体测试模块-AXI_IP 核 package-Microblaze 顶层读取-显示界面编写与峰值打印。遇到了许多逻辑与时序错误，以下列出一些最主要，最共性的问题：

有符号数运算及赋值问题

Booth 乘法器设计时，由于采用补码运算，按照 Verilog 的基本语法，如果表达式中混合有符号数和无符号数，会一律当作无符号数运算，而有符号数运算和无符号数运算时位数扩展以及赋值的规则并不相同，所以在设计初期在这个坑上花了许多时间。

解决方法：有符号数的计算模块所有参与运算的变量一律选用有符号数，所有运算的结果结果在一开始就进行位数扩展，最后按照输入的范围进行相应的低位/高位截取。

软件定时器的数据读取问题

在一开始软件中的数据读取采用了定时器中断的读取方式，后来发现无法读取到完整的数据，存在数据丢失的问题，设计后期在这个问题上也花了不少时间。

解决方法：传统的解决方法有两种，第一种是先用 FIFO 缓冲数据后再把软件读取地址映射到底层 IP 读 FIFO 的操作上，第二种是直接在设定时间内不断地读取底层的值，同时读取该值的编号，当读取编号达到序列长度时，停止读取。本设计采用了第二种，但是第二种操作直接影响了软件的并行性，需要 CPU 停下所有动作来处理读取数据。由于本设计不需要添加别的功能，所以运行效率上的成本可以接受。

6.2 总结与致谢

在所有的工作收尾之际，不得不说我在本课设上花的时间大大超过了预期，甚至挤占了一部分期末复习的时间，FFT 模块在 12.17 日开始编写，12.23 日测试完毕，接下来进入了繁琐的系统互联与软件联调阶段，由于要和队友的电子琴模块相配合，在模块协同工作上花去了许多的时间，直到 12.31 日晚才算基本完成了所有的工作，拍摄完了视频。个人最大的想法就是：有时候我认为最核心的算法，可能并不是工程中最重要的，重要的是一个模块的通用性，如何和别的总线与系统匹配、应用。

同时，另一个血的教训就是，当一个小组协同工作时，组内的工作需要一开始就协调好模块接口的统一性，并确保自己的模块经过测试后在联调之前一定要处于工作无误的状态！否则会因为互联上的问题花掉很多时间进行调试，且这一问题花费的时间和精力会随着系统的增大几何级数的上升。以至于最后两天我在华中科技大学启明学院昏天地暗地进行了多次 ILA 调试，最后竟发现结果是队友的显示函数中出现了“Printf”延迟极大影响了数据的读取！

总结一句话：是我太菜。

最后，感谢此次课设中与我一起工作，共同熬夜的三位队友以及给我提供了许多帮助的电工基地王贞炎老师和实验室徐尚成学长以及郑朝霞老师。他们都在系统的互联以及 IP 核的运用上给了我许多帮助。

七、源码及参考文献

源码

Fft 模块源码已经上传个人 github 主页: https://github.com/hxfycy/1024-point-fft

参考文献

1. Sabu M. ,ThampiAlexander ,GelbukhJayanta Mukhopadhyay: Advances in Signal Processing and Intelligent Recognition Systems(2014)

2. Xilinx® Logic coreTM, Fast Fourier Transform V8.0, Xilinx® (2012)

3. Oh, J.-Y., Lim, M.-S.: Area and power efficient pipeline FFT algorithm.In: IEEE Workshop on Signal Processing Systems Design and Implementation, November 2-4, pp. 520–525 (2005)
4.王贞炎，《FPGA 应用开发与仿真》（2017）
5. Zhong, G., Zheng, H., Jin, Z., Chen, D., Pang, Z.: 1024-Point Pipeline FFT Processor with Pointer FIFOs based on FPGA. In: 2011 IEEE/IFIP 19th International Conference on VLSI and System-on-Chip (2011)