**FFT处理器的设计**

|  |  |
| --- | --- |
| 姓名： | 赵鑫蕾 |
| 学号： | 2024E8013282114 |
|  |  |

中国科学院大学

2025年2月15日

目录

[1 设计要求 1](#_Toc190782746)

[2 FFT算法原理 1](#_Toc190782747)

[2.1 基本原理 1](#_Toc190782748)

[2.2 FFT/IFFT 复用 2](#_Toc190782749)

[2.3 数据输入输出的顺序 2](#_Toc190782750)

[2.4 FFT实现结构 2](#_Toc190782751)

[2.5 流水线FFT结构设计 3](#_Toc190782752)

[2.6 数据的表示 4](#_Toc190782753)

[3 FFT Matlab算法 4](#_Toc190782754)

[3.1 算法实现 4](#_Toc190782755)

[3.2 算法仿真 5](#_Toc190782756)

[4 RTL实现 6](#_Toc190782757)

[4.1 整体结构 6](#_Toc190782758)

[4.2 接口及时序 7](#_Toc190782759)

[4.3 主要模块设计 8](#_Toc190782760)

[4.3.1 Radix-2蝶形运算模块 8](#_Toc190782761)

[4.3.2 Counter计数器模块设计 9](#_Toc190782762)

[4.3.3 FFT top设计 10](#_Toc190782763)

[5 整体仿真验证与综合 11](#_Toc190782764)

[5.1 联合仿真验证 11](#_Toc190782765)

[5.2 工程文件结构 12](#_Toc190782766)

[5.3 VCS编译仿真 12](#_Toc190782767)

[5.4 Verdi 波形查看 15](#_Toc190782768)

[5.5 ASIC综合 16](#_Toc190782769)

[6 总结 18](#_Toc190782770)

[6.1 工程总结 18](#_Toc190782771)

[6.2 个人总结 18](#_Toc190782772)

# 设计要求

采用硬件RTL代码实现FFT算法，可对连续输入串行数据以64点进行分组完成FFT运算。

# FFT算法原理

## 基本原理

设长度为N的输入序列，DFT 表达式如下：

 （2-1）

相应的IDFT为：

 （2-2）

直接实现上述运算需要O(*N*2)的计算复杂度，当*N*较大时，计算复杂度极高，以至于难以实时处理。FFT算法的引入可以有效降低DFT的计算复杂度，Cooley-Tukey算法在计算量以及相关控制上优势明显，被广泛运用于数字电路中。

按照数据组合方式划分，FFT 算法分为时间抽取 FFT(DIT)和频率抽取 FFT(DIF)两类，按照逐次分解的块(蝶形单元)的大小划分，又可以分为Radix-2、Radix-4、Radix-8和Radix-16等多种实现方式。

对于DIF Radix-2 FFT算法（Cooley-Tukey算法），长度*N*输入序列为，根据*n*的属性，可将*n*分为前后两组。

 （2-3）

DFT表达式：

  （2-4）

根据*k*的奇偶性，可将划分为奇偶两部分

 （2-5）

因此，Radix-2的蝶形单元可以表示为如图 2.1所示：



图 2.1 DIFRadix-2蝶形单元

## FFT/IFFT 复用

设序列为有限长序列，长度为M，则定义的点离散傅里叶变换如式2-1所示，对应的N离散傅里叶逆变换如式2-2所示。观察两式，离散傅里叶逆变化可有傅里叶变换公式的输入输出取共轭得到如式2-6所示，即可用DFT相同的算法得到IDFT的结果。

 （2-6）

可以在输入输出做一个2选1的MUX，若为IFFT则对输入输出取共轭，从而支持FFT与DFT的分时复用。

## 数据输入输出的顺序

输入自然序，输出倒位序，如图 2.3所示（同址运算结构）；



图 2.3自然序输入倒位序输出（右）

## FFT实现结构

FFT的结构可以分成两种主要的类型：存储器型结构和流水线型结构。

存储器型结构根据其所采用的存储器的方式，又可以表现为单存储型，双存储器型和Cache缓存型，但是其原理是统一的。

流水线型结构根据结构特点又可以分为单路延迟反馈(Single-path Delay Feedback, SDF)，多路延迟交换 (Multi-path Delay Commutator, MDC)和多路延迟反馈 (Multipath Delay Feedback, MDF)。

存储器型FFT结构通常都只包含一个蝶形处理单元 (Processing Element, PE)，多个存储器块和控制通路。存储器型FFT可以将多级蝶形单元通过迭代在一个PE当中进行处理，其核心的计算资源较小。对于长计算点数和多种计算长度集合的FFT计算，可以通过配置存储器型FFT处理器的蝶形单元的计算级数进行有效的处理，但控制相对繁琐。



图 2.5存储器型FFT结构

流水线型FFT结构将不同级的蝶形单元通过流水线的方式进行串联，其处理过程是前向的不存在关键路径或者反馈环路的问题。另外，可以通过增加每一级蝶形单元的PE数量进行数据吞吐率的扩展，适用于超高速数据的实时处理。



图 2.6 流水线型FFT结构

存储器型FFT在运算时需考虑原位运算，控制较为繁琐，输入输出延时较大但资源利用率高，流水线型FFT计算速度快，输入输出延时较小，可扩展性强，但资源消耗相对较大。

本设计中采用了流水线型FFT架构。

## 流水线FFT结构设计

存储型FFT需要配合控制器来进行运算，运算速度相比于流水线型FFT更慢。

1. FFT/IFFT运算需要在装入全部数据后进行，由于运算电路为串行输入，因此必须要通过FIFO进行装入，待数据全部装入完成后开始运算
2. FFT/IFFT运算结果的输出是并行同时输出的，由于运算电路为串行输出，因此必须要通过FIFO将数据由并行转成串行输出，即结果计算完成后装入FIFO然后依次排出

以上的两个FIFO需要配合counter.v进行控制，FIFO输入数据后开始启动计时器，历经256拍后装满/排空FIF。

通过与蝶形单元输入和输出共享相同的存储空间，Radix-2单路延迟反馈(Radix-2 Single-path Delay Feedback，R2SDF)可以有效地利用延迟单元，*N*点R2SDF FFT运算的结构如图 2.7所示。



图 2.7 N点R2SDF FFT运算的结构

输入的数据序列为串行输入，第1级通过转换器将前*N*/2个输入数据送入延时缓存单元中，待第*N*/2+1个输入数据到来时，将第1个输入数据从延时缓存单元中取出，两个数据同时送入第1级Radix-2蝶形单元进行数据间隔为*N*/2的蝶形运算。将蝶形单元相加端的数据直接送到下一级蝶形运算单元，相减端的数据送回第1级的延时缓存单元中空出来的地址上进行存储，存储完成之后再取出第一个数据并乘上蝶形因子并输出到下一级，第2级执行第1级相似的操作，蝶形运算间隔变为*N*/4，直到最后一级运算完成输出。

可将256点FFT分解为：2x2x2x2x2x2，即8级R2SDF。

## 数据的表示

根据运算过程中对数据位数取位和表示形式的不同，可以将FFT的运算分为浮点、定点和块浮点三种类型。它们在实现时对于系统资源的要求是不同的，而且有着不同的适用范围。

浮点FFT计算是较难溢出。但浮点运算在硬件实现上有相当大的难度，不仅占用的系统资源多，而且硬件运行的速度慢。

定点的表示方法和浮点数相对应，数据直接用二进制表示，且小数点的位置固定，运算简单，没有浮点数位数对齐和归一化问题。因此，在硬件实现上以定点数为基础的算法占用的面积少，易实现，但是由于定点数的动态范围小，很容易出现溢出，必须要有溢出控制。

块浮点是介于它们之间的一种运算方式，它基于自动增益控制的思想，在一个数据块上实现浮点，即一组数据共用一个指数因子。这样在硬件实现上相对于传统的浮点运算有着更小的代价，是浮点和定点的折衷。

设计中采用了定点FFT算法。

对于256点FFT，输出数据位宽比输入数据位宽至少多7位，若输入数据位宽为8位，则输出数据位宽至少15位。

# FFT Matlab算法

## 算法实现

在Matlab中编写与RTL对应的算法原型实现256点FFT以验证设计的正确性，算法流程如下所示：

* 首先生成所需的256点旋转因子，定点量化为16位有符号数；
* 根据是正FFT还是逆FFT对输入数据进行取共轭的操作；
* 然后对蝶形单元进行分级、分组、分单元进行操作，
* 数据全部运算完成后，根据是正FFT还是逆FFT对输入数据进行取共轭的操作；

## 算法仿真

* 实现一个 256 点的 FFT 算法。首先生成旋转因子和输入信号，通过反序操作对输入数据进行重排，然后使用三层循环和蝶形运算完成 FFT 计算。代码还调用 MATLAB 内置的 fft 函数进行对比，并绘制频谱图和误差图，最后计算 SQNR 评估算法精度。整体实现了 FFT 的核心功能，并通过对比验证了其正确性。。
* 通过 DIT-FFT 方法计算输入信号的频谱。
* 与 MATLAB 内置的 fft 函数结果进行比较，验证算法的正确性。
* 计算信号与误差的 SQNR（Signal-to-Quantization Noise Ratio，信噪比），评估算法性能。

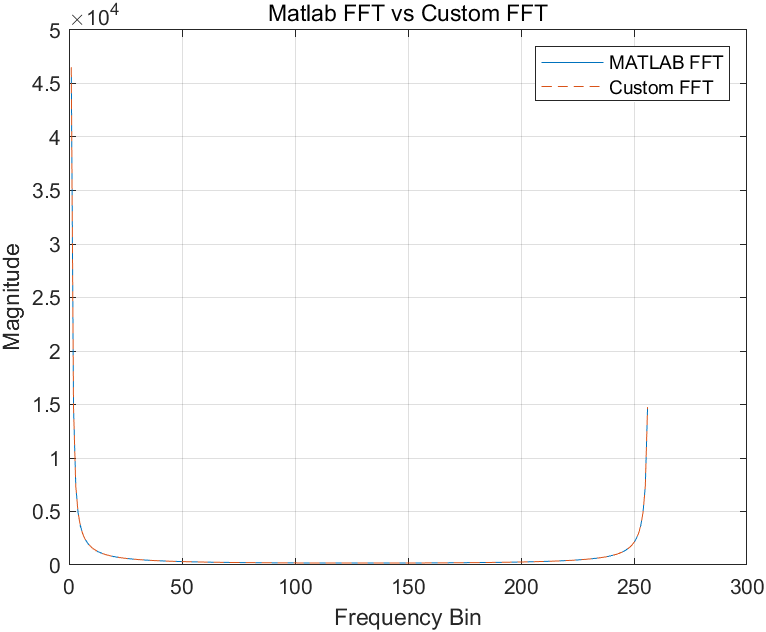


图 3.1 设计的FFT算法与Matlab自带的FFT函数的输出结果

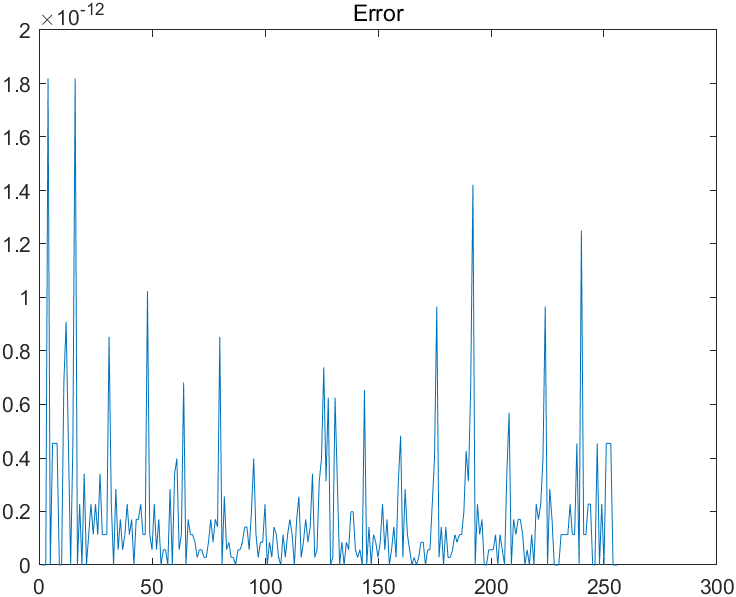


图3.2 输出设计算法和matlab数据进行误差比较

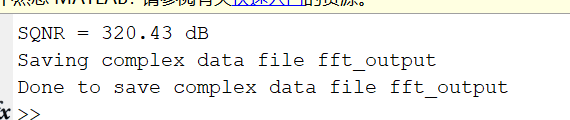


图3.3 SQNR输出，得到320dB，证明噪声干扰很小

由此可以验证得到设计的FFT算法的正确性，后续设计的RTL只需要与Matlab算法一一映射即可。

# RTL实现

## 整体结构

整体结构采用Radix-2单路延迟反馈流水线结构，分为了6个子模块，如下图所示，其中：

* 输入级模块（*counter1和input FIFO*）：对数据的数据进行寄存，并根据FFT和IFFT的运算方法对数据进行重映射。
* Radix-2流水级（*fft module*）：利用8级流水线模式进行蝶形运算；每一级蝶形运算利用一个寄存器进行存储。
* 输出级（*output FIFO*）：对运算完成的数据进行整理，包括：根据正FFT或逆FFT对输入数据取共轭操作，对数据进行截位，输出时序生成。



图 4.1 256点流水线FFT整体结构

## 接口及时序

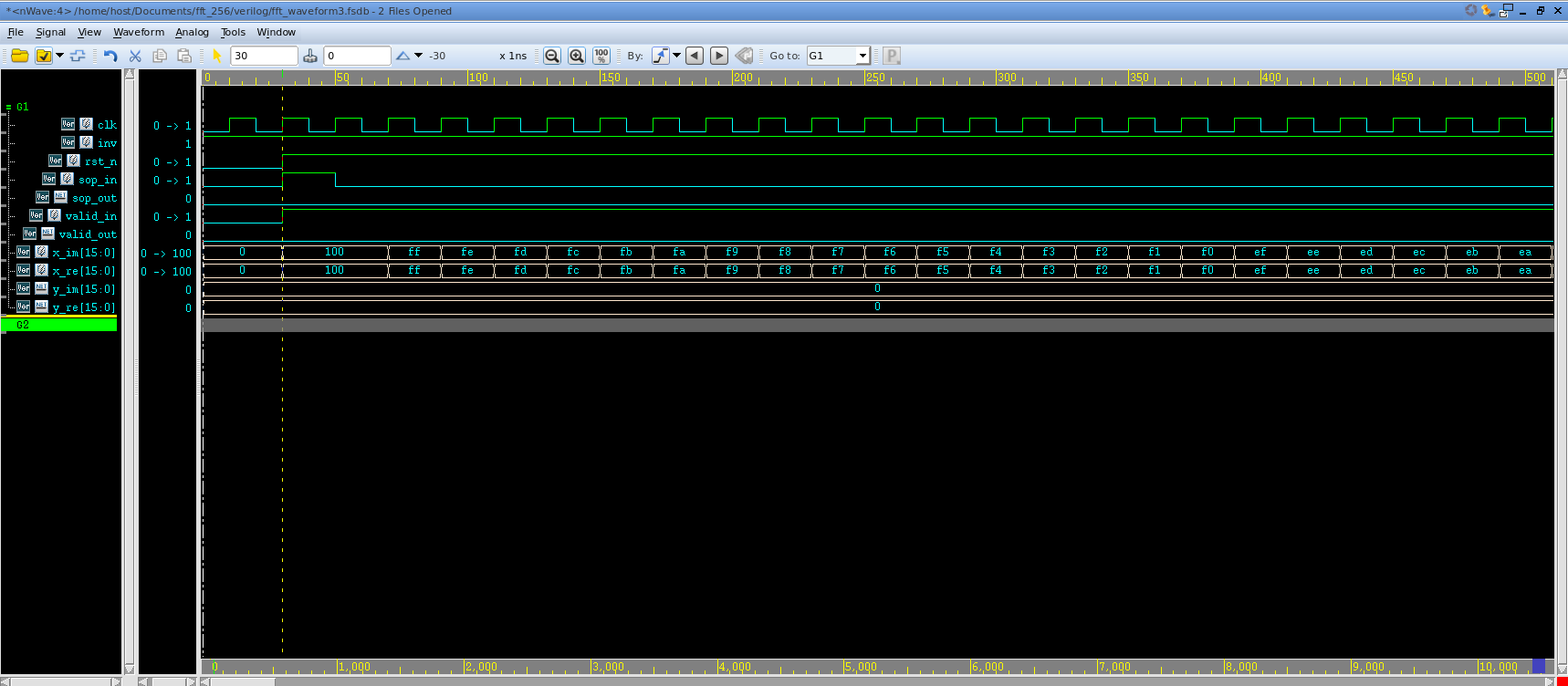


图4.1 在使能信号上升沿时， sop\_in，valid\_in输入为1时启动数据输入

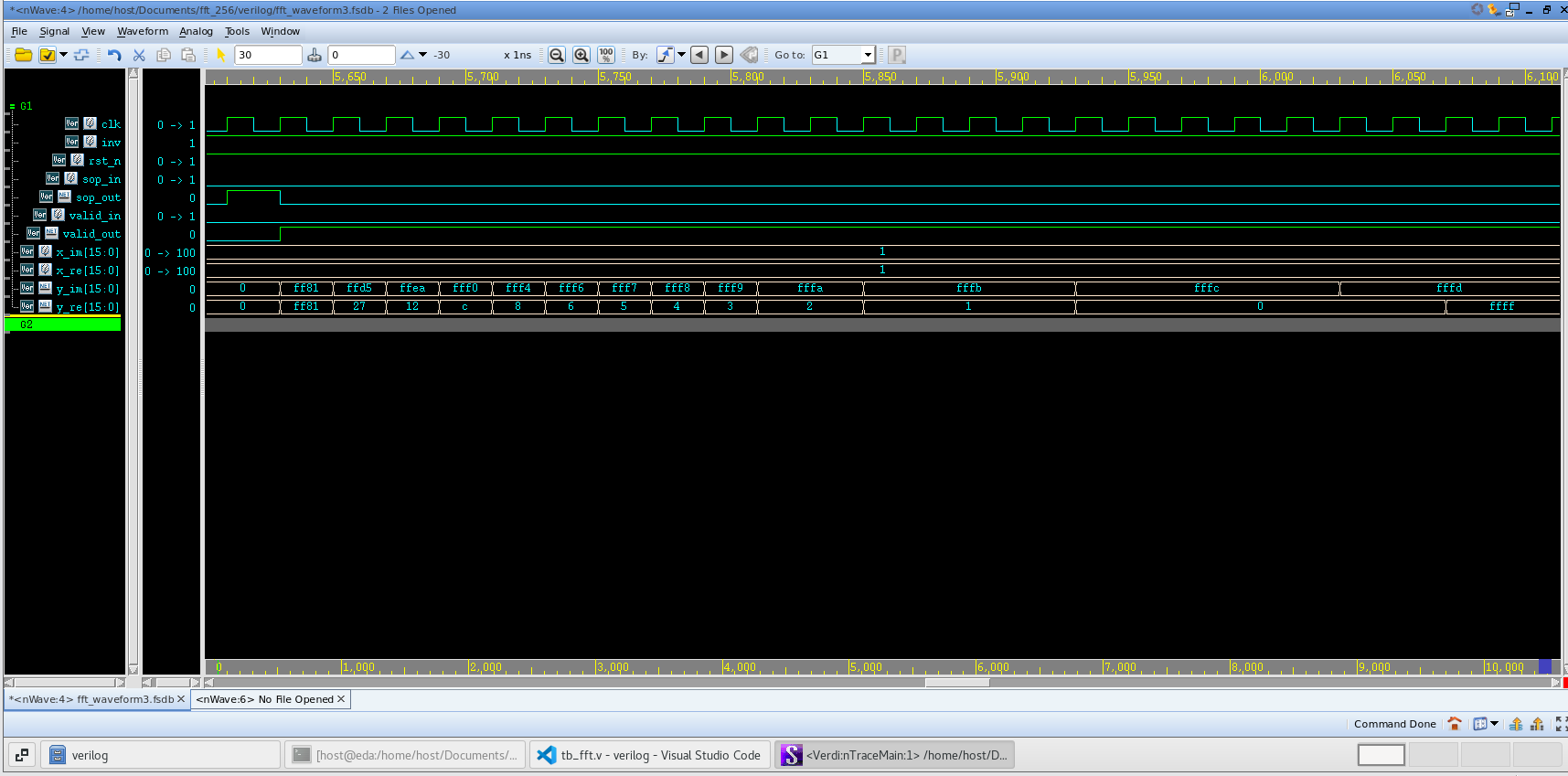


图4.2 在使能信号上升沿时， sop\_out，valid\_out输入为1时启动数据输出

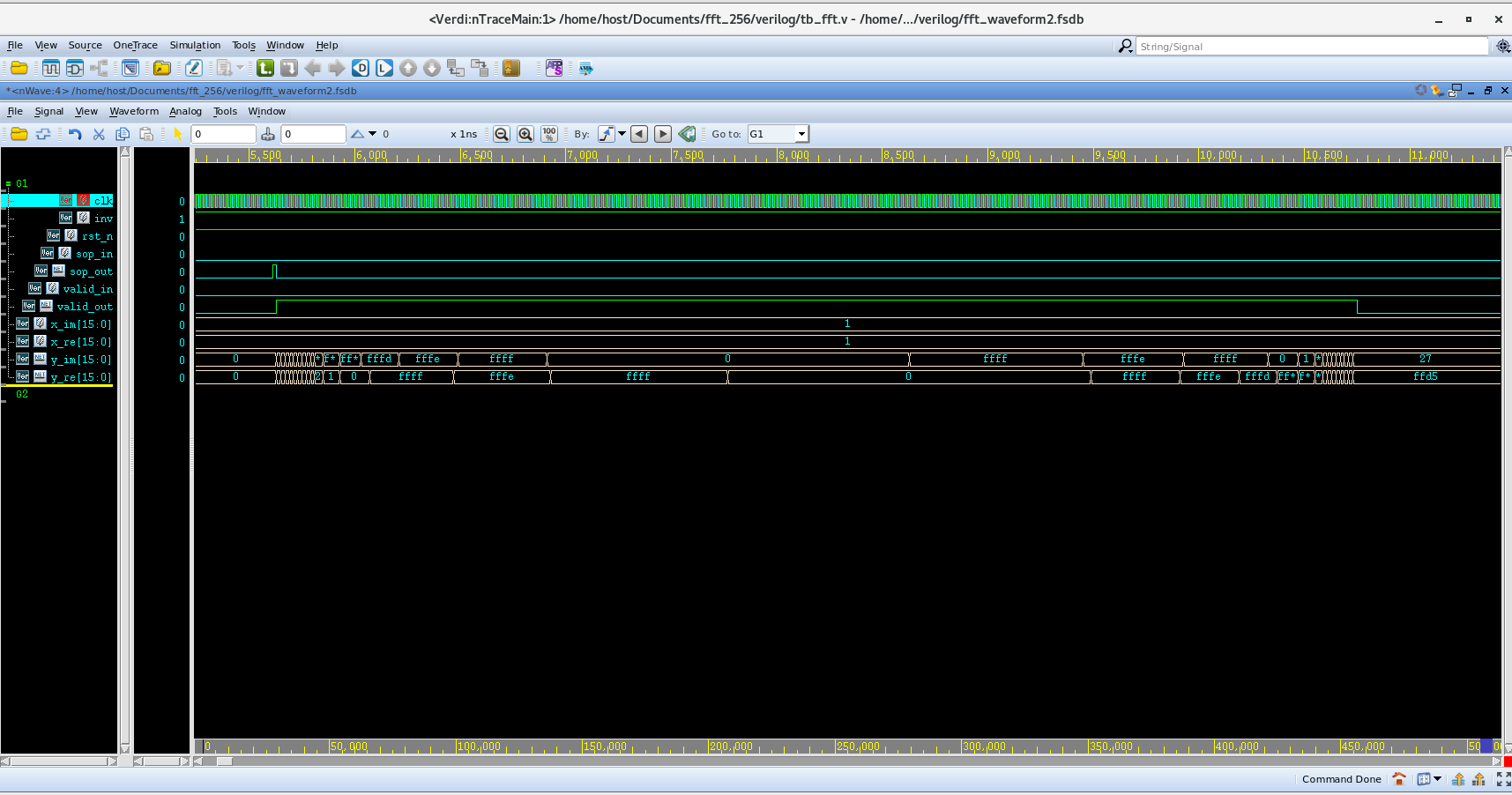


图 4.3 总接口接口时序

## 主要模块设计

### Radix-2蝶形运算模块



图4.4 Radix-2蝶形运算模块接口

该模块实现了一个典型的 Radix-2 蝶形运算，完成以下计算：

1. **输入**：两个复数 *Xp*​ 和 *Xq*​，以及旋转因子 *WN*​。
2. **输出**：两个复数 *Yp*​ 和 *Yq*​，计算公式如下：

Yp=Xp+Xq⋅WN

*Yp*​=*Xp*​+*Xq*​⋅*WN*​

Yq=Xp−Xq⋅WN

*Yq*​=*Xp*​−*Xq*​⋅*WN*​

其中，*WN*​ 是旋转因子，用于将 *Xq*​ 旋转到正确的位置。

1. **模块结构**
2. **输入信号**：
   * xp\_re, xp\_im：复数 *Xp*​ 的实部和虚部。
   * xq\_re, xq\_im：复数 *Xq*​ 的实部和虚部。
   * factor\_re, factor\_im：旋转因子 *WN*​ 的实部和虚部。
   * clk, rst\_n, en：时钟、复位和使能信号。
3. **输出信号**：
   * yp\_re, yp\_im：复数 *Yp*​ 的实部和虚部。
   * yq\_re, yq\_im：复数 *Yq*​ 的实部和虚部。
   * vld：输出有效信号，表示计算结果可用。
4. **流水线设计**：
   * 模块采用三级流水线结构，确保高吞吐量：
     1. **第 1 拍**：计算 *Xq*​ 与旋转因子 WN*WN*​ 的乘积，并对 *Xp*​ 进行移位对齐。
     2. **第 2 拍**：合并乘积结果，完成旋转因子的乘法运算。
     3. **第 3 拍**：完成蝶形加减运算，得到最终结果 *Yp*​ 和 *Yq*​。
5. **关键操作**：
   * **复数乘法**：*Xq*​ 与旋转因子 *WN*​ 的乘法通过拆分实部和虚部计算实现。
   * **移位对齐**：*Xp*​ 的实部和虚部左移 15 位，确保计算精度。
   * **蝶形加减**：最终结果通过加法和减法运算得到。

### Counter计数器模块设计



图4.5 counter模块接口图

**1.功能介绍：**

* FFT/IFFT运算需要在装入全部数据后进行，由于运算电路为串行输入，因此必须要通过FIFO进行装入，待数据全部装入完成后开始运算。
* FFT/IFFT运算结果的输出是并行同时输出的，由于运算电路为串行输出，因此必须要通过FIFO将数据由并行转成串行输出，即结果计算完成后装入FIFO然后依次排出。

**计数功能**：在使能信号有效时，计数器从 0 开始递增，直到达到设定的阈值 thresh。

**状态控制**：通过状态机控制计数器的启动和停止。

**2. 输入输出信号**

* + **输入信号**：
  + clk：时钟信号，驱动计数器和状态机。
  + rst\_n：异步复位信号，低电平有效。
  + thresh：9 位阈值信号，用于设置计数器的最大值。
  + start：启动信号，高电平有效，用于启动计数器。
  + valid：有效信号，高电平有效，用于控制计数器的递增。
* **输出信号**：
  + not\_zero：标志信号，当计数器值大于 0 时有效。
  + full：标志信号，当计数器值达到阈值 thresh 时有效。
* **3. 状态机设计**
* **状态定义**：
  + START：计数器运行状态。
  + STOP：计数器停止状态。
* **状态转移逻辑**：
  + 当 start 信号为高电平时，状态从 STOP 转移到 START，并启动计数器。
  + 当计数器达到阈值 thresh 时，状态从 START 转移到 STOP，并停止计数器。
* **4. 计数器逻辑**
* **复位时**：
  + 计数器 cnt 被清零。
  + 标志信号 not\_zero 和 full 被置为 0。
* **计数时**：
  + 当 valid 信号为高电平且计数器使能信号 cnt\_en 为高电平时，计数器 cnt 递增。
  + 当计数器值达到阈值 thresh 时：
    - 计数器 cnt 被清零。
    - 标志信号 full 被置为 1。
  + 当计数器值大于 0 时，标志信号 not\_zero 被置为 1。
* **5. 算法流程**

1. **复位阶段**：
   * 当 rst\_n 为低电平时，计数器 cnt、标志信号 not\_zero 和 full 被清零，状态机进入 STOP 状态。
2. **启动阶段**：
   * 当 start 信号为高电平时，状态机从 STOP 转移到 START，并启动计数器。
3. **计数阶段**：
   * 在 START 状态下，当 valid 信号为高电平时，计数器 cnt 递增。
   * 如果计数器值达到阈值 thresh，则计数器清零，并置位 full 标志。
4. **停止阶段**：
   * 当计数器达到阈值 thresh 时，状态机从 START 转移到 STOP，并停止计数器。

### FFT top设计

该模块实现了一个基于 Radix-2 算法的 256 点快速傅里叶变换（FFT），能够高效地将时域信号转换为频域信号。

* **1. 模块功能**
  + **FFT 计算**：对输入的 256 点复数序列进行快速傅里叶变换。
  + **支持正反变换**：通过 inv 信号控制，支持正向 FFT 和反向 IFFT。
  + **流水线设计**：采用多级流水线结构，实现高吞吐量和低延迟。
  + **位反转处理**：对输入数据进行位反转排序，以满足 FFT 算法的要求。
  + **旋转因子计算**：通过预定义的旋转因子表（Twiddle Factor Table）实现复数乘法运算。
* **2. 输入输出信号**
* **输入信号**：
  + clk：时钟信号，驱动模块运行。
  + rst\_n：异步复位信号，低电平有效。
  + inv：变换方向控制信号，0 表示 FFT，1 表示 IFFT。
  + valid\_in：输入数据有效信号。
  + sop\_in：输入数据帧起始信号。
  + x\_re, x\_im：输入复数信号的实部和虚部（16 位有符号数）。
* **输出信号**：
  + valid\_out：输出数据有效信号。
  + sop\_out：输出数据帧起始信号。
  + y\_re, y\_im：输出复数信号的实部和虚部（16 位有符号数）。
* **3. 模块结构**

1. **输入缓冲区**：
   * 对输入数据进行缓冲和位反转排序，以满足 FFT 算法的要求。
2. **旋转因子表**：
   * 预定义了 128 个旋转因子（Twiddle Factor），用于复数乘法运算。
3. **8 级蝶形运算**：
   * 采用 Radix-2 算法，将 FFT 分解为 8 级蝶形运算。
   * 每级蝶形运算由多个并行的蝶形单元（butterfly）组成。
4. **输出缓冲区**：
   * 对蝶形运算的结果进行缓冲和输出。
5. **控制逻辑**：
   * 通过计数器（counter）和状态机控制数据的输入、计算和输出。

# 整体仿真验证与综合

## 联合仿真验证

本设计中的RTL严格按照编写的Matlab算法进行实现的，因此相同激励下，RTL的输出应当与Matlab输出完全一致，在vcs中对RTL进行仿真并对RTL的输出进行比对得出验证结果。

## 工程文件结构

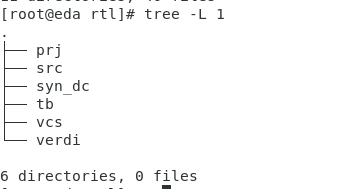


图5.1 rtl代码结构图

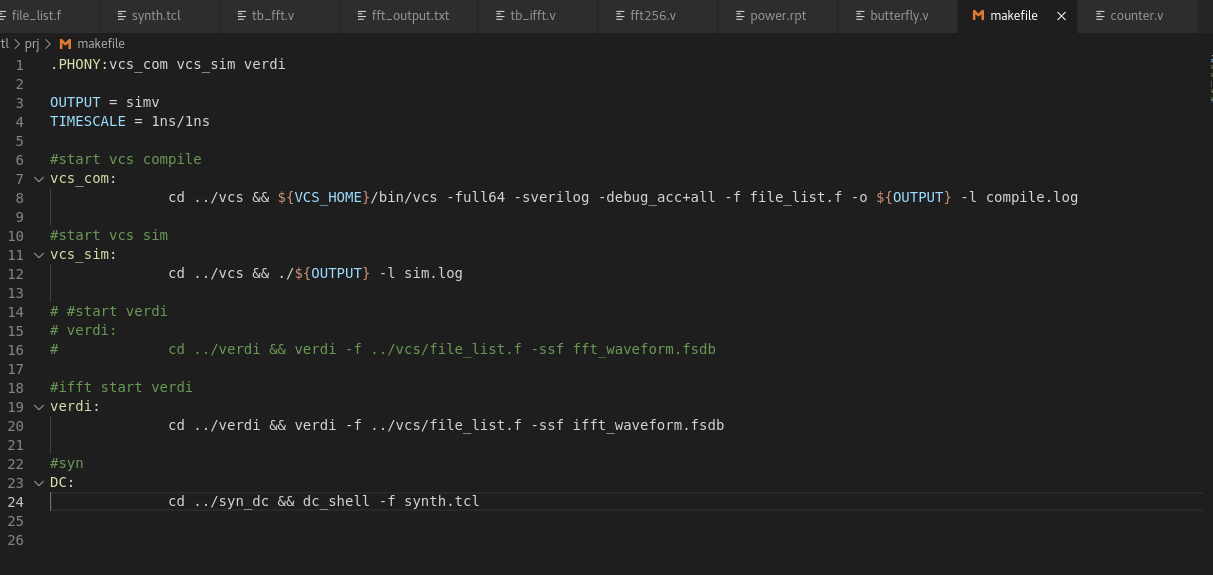


图5.2 Makefile定义编译仿真综合语句

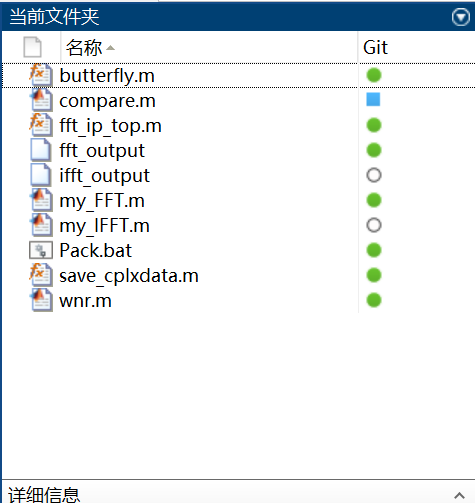


图5.3 matlab代码结构图

RTL代码仿真结构：其中makefile在prj文件中，src和tb为源码和测试代码。vcs，verdi存放仿真的数据，syn\_dc存放综合后的数据。利用执行语句vcs\_com和vcs\_sim即可以进行编译和仿真。

Matlab代码验证结构：利用my\_FFT和my\_IFFT进行仿真，得到数组结果利用compare进行输出。

## VCS编译仿真

利用vcs仿真得到如下图结果：

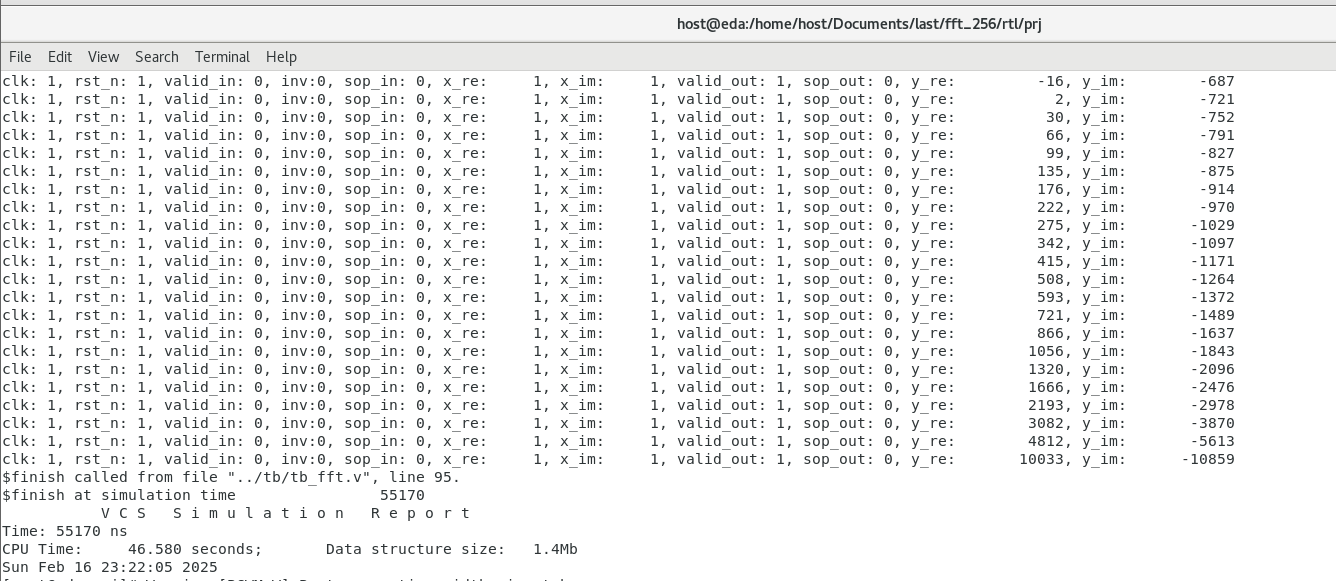


图5.4 FFT运行时输出数据

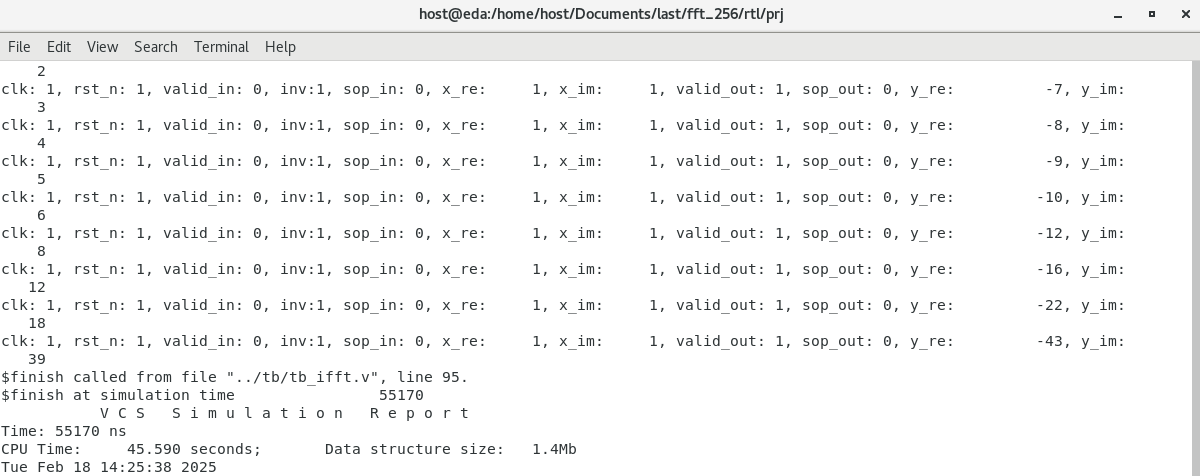
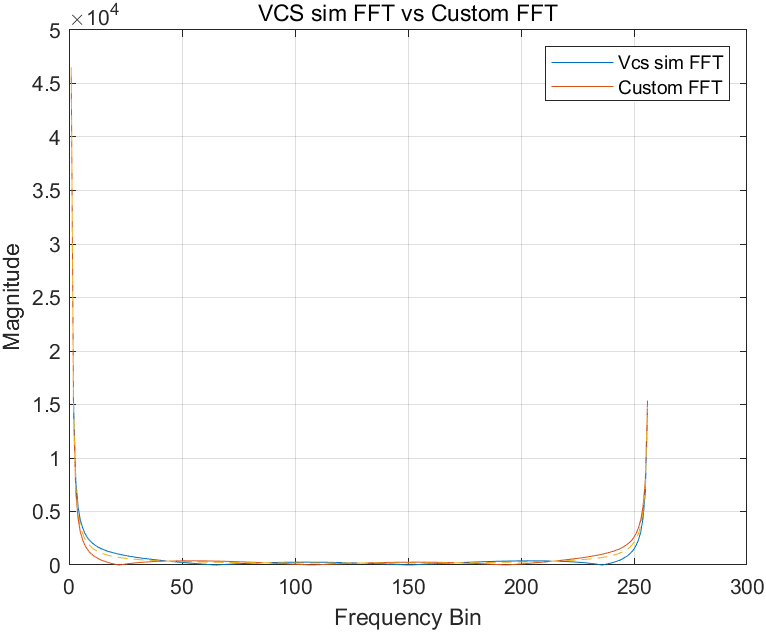


图5.5 IFFT运行输出数据

结果输出到fft\_out.txt/ifft\_out.txt中利用matlab进行Vcs仿真定点数和matlab代码浮点数进行比较：

  
图5.6 仿真定点数FFT和matlab 算法浮点数FFT对比

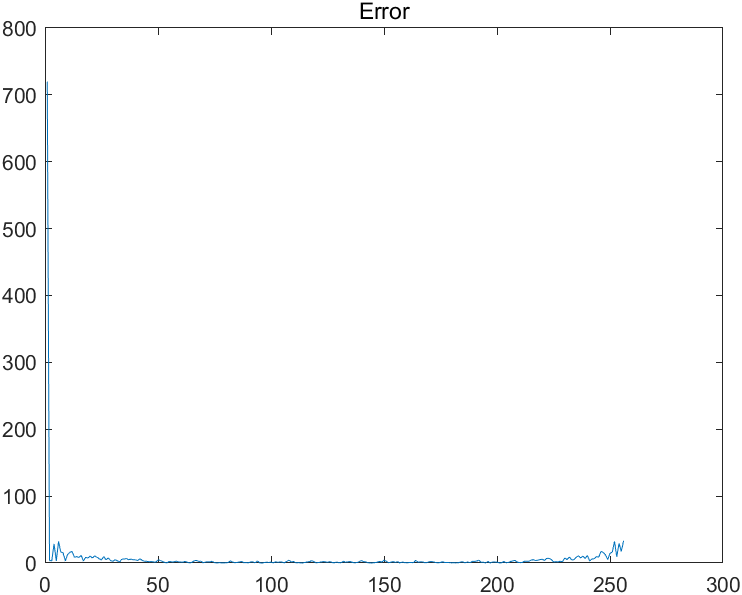


图5.7 仿真定点数FFT和matlab 算法浮点数FFT误差

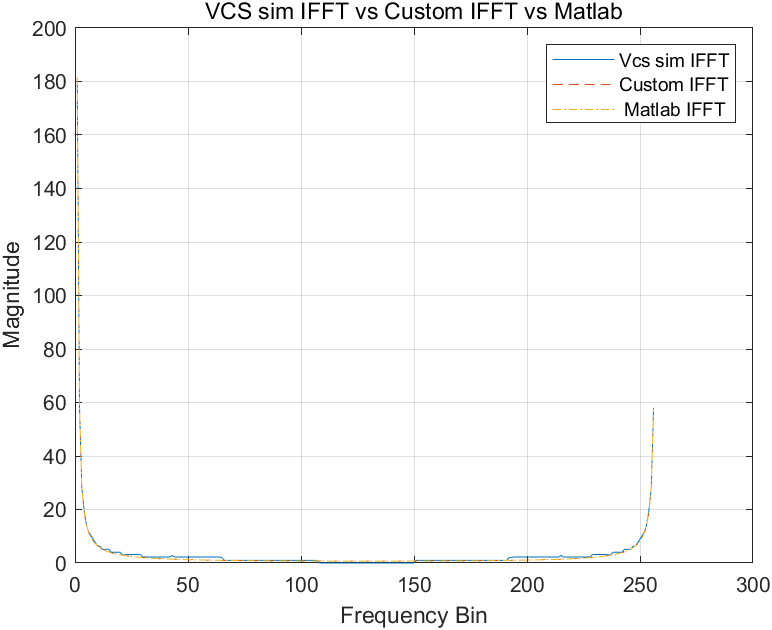


图5.8 仿真定点数IFFT和matlab 算法浮点数IFFT对比

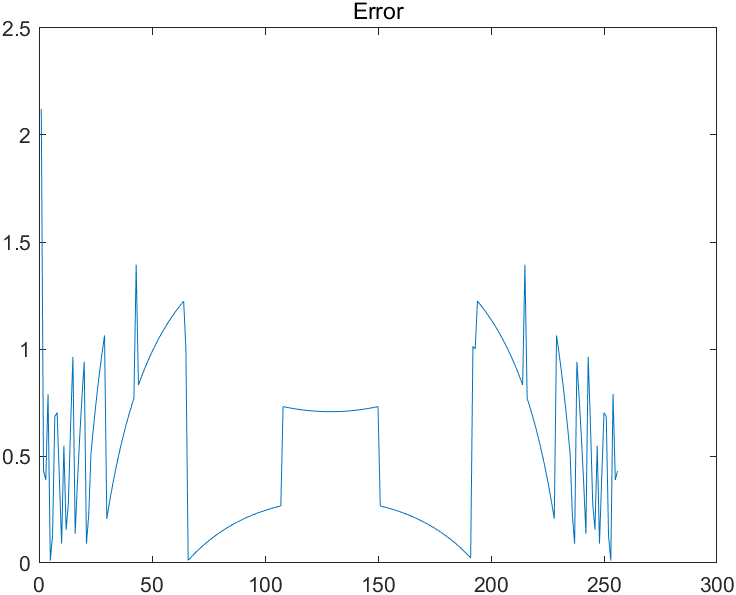


图5.9 仿真定点数IFFT和matlab 算法浮点数FFT误差

将verilog计算的结果与matlab中浮点数计算的结果进行比较，结果如上图所示，可见虽然FFT和IFFT定点化带来了一定的精度损失，但结果仍然是正确的，且误差很小。

## Verdi 波形查看

利用verdi进行波形查看：可以看到sop\_in上升沿后，数据输入，valid\_out为0，等待256拍，数据全部输入完成，进行8级蝶形运算，计算出结果后sop\_out置1，进行y\_re和y\_im的输出。

满足功能的时序要求。

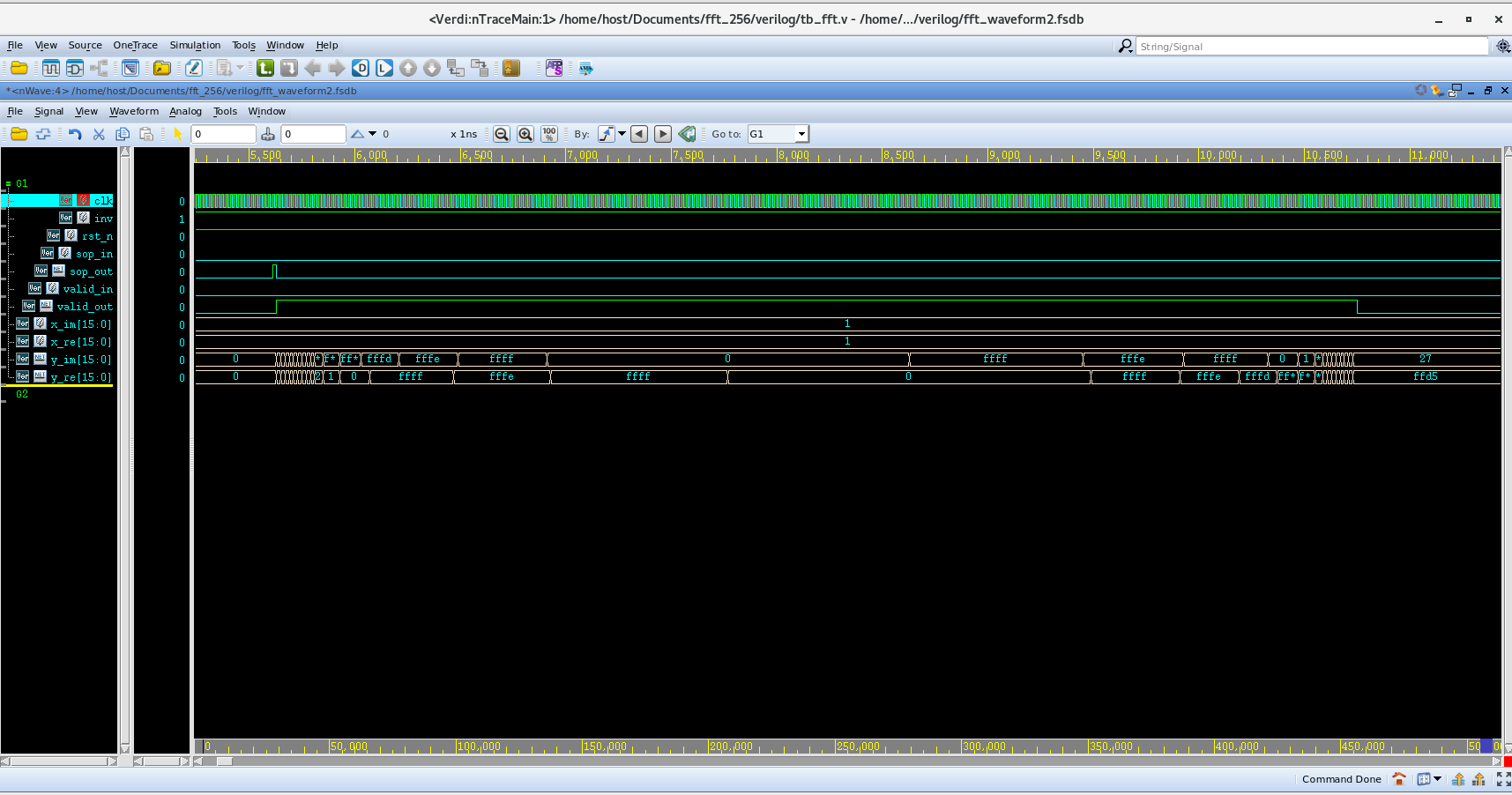


图5.10 仿真过程中的时序波形图

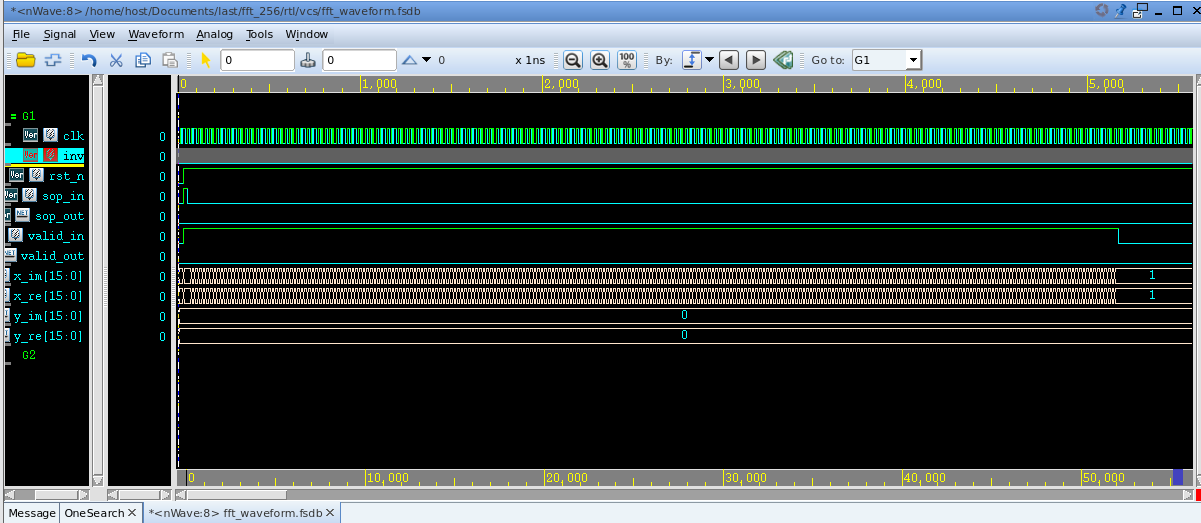
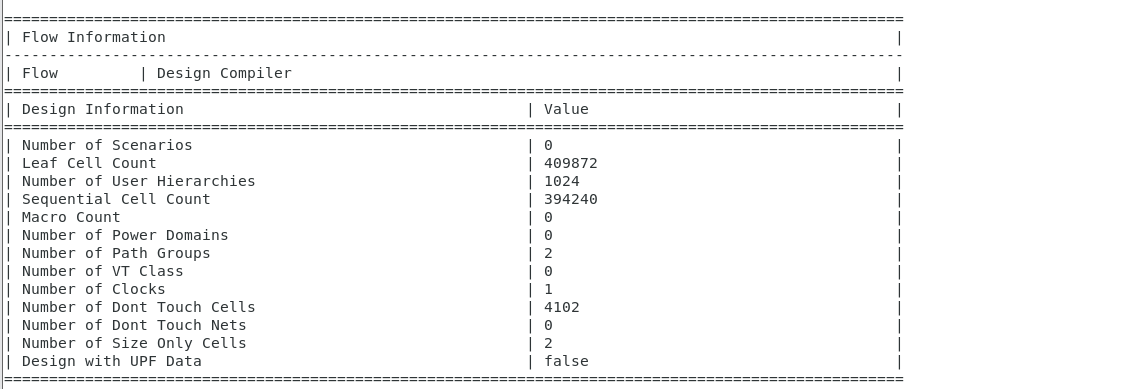
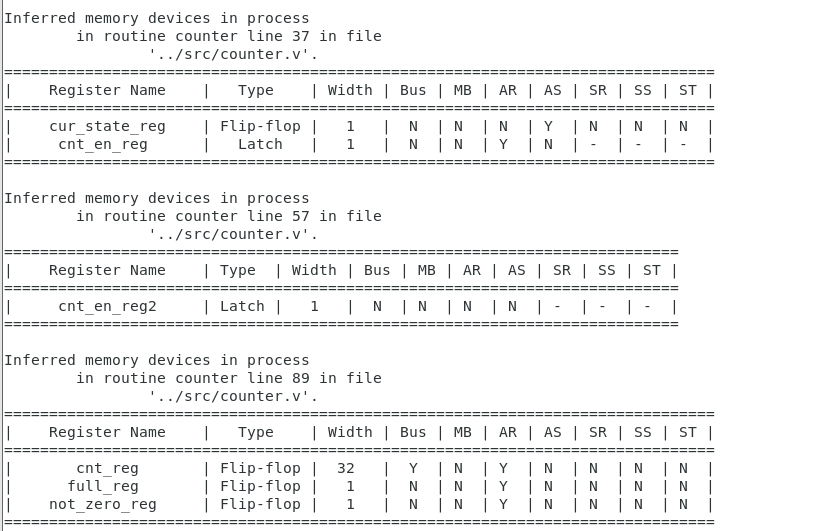


图5.11 切换inv = 0 进行FFT运算。

## ASIC综合





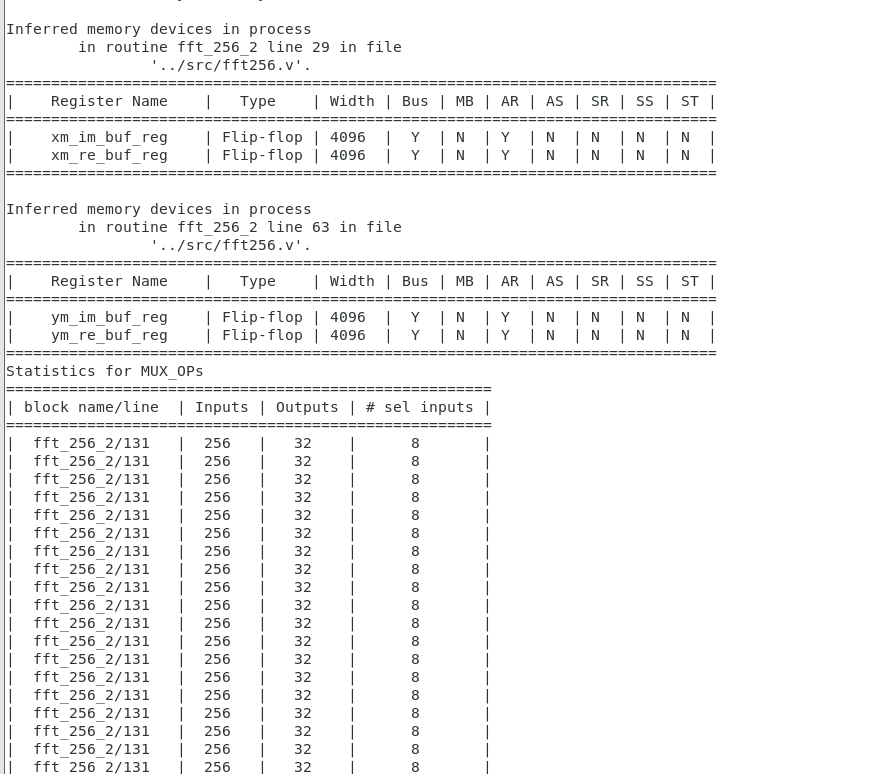


图5.12 综合结果

**Number of Scenarios:** 0，表示在当前设计中没有任何多场景分析（multi-scenario analysis）。多场景分析通常用于考虑不同的工作条件（如电压、温度等）。

**Leaf Cell Count:** 409872，表示设计中有 409872 个叶子单元（leaf cells）。叶子单元是指设计中最底层的逻辑单元，如标准单元库中的与门、或门等。

**Number of User Hierarchies:** 1024，表示设计中有 1024 个用户定义的层次结构。层次结构通常用于模块化设计。

**Sequential Cell Count:** 394240，表示设计中有 394240 个时序单元（如触发器、锁存器等）。这个数字较高，表明设计中包含大量的时序逻辑。

**Macro Count:** 0，表示设计中没有使用宏单元（如存储器、PLL 等）。宏单元通常是预定义的复杂功能块。

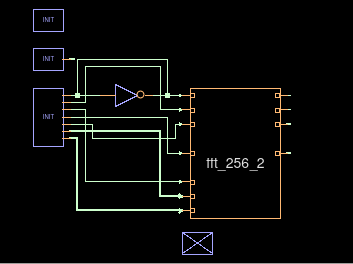


图5.13 生成的网表结构

# 总结

## 工程总结

1. **工程流程**

* 设计采用了标准编译仿真和综合流程，使用Vcs和Verdi进行verilog的编译和仿真。 Design Compiler 将 RTL 代码转换为门级网表。单时钟设计和无电源域管理简化了设计，但可能限制了性能和功耗优化。

1. **FFT 算法实现**

* 256 点 FFT 通过蝶形运算实现，流水线结构提高了吞吐量，但需要大量时序单元支持。设计中未使用多电压和多阈值电压技术，可能更注重性能而非功耗优化。

1. **改进方向**
2. **性能优化**：
   * 引入多时钟域或异步设计，以提高吞吐量。
   * 优化关键路径，减少时序瓶颈。
3. **功耗优化**：
   * 引入多电压域或动态电压频率调节（DVFS）。
   * 使用多阈值电压单元，在性能和功耗之间取得平衡。
4. **资源优化**：
   * 减少“Don't Touch Cells”数量，允许综合工具进行更多优化。
   * 优化存储器使用，减少面积和功耗。

## 个人总结

通过本次 FFT-256 点大作业，我深入掌握了数字信号处理算法的硬件实现方法，并熟悉了基于 VCS、Verdi 和 Design Compiler（DC）的联合仿真与综合流程。这次实践让我在硬件设计、仿真调试和性能优化方面积累了宝贵的经验，以下是我的主要收获和体会：

**1. 硬件设计能力的提升**

* **FFT 算法实现**：通过实现 Radix-2 蝶形运算模块，我深入理解了 FFT 算法的分治思想和硬件实现细节，包括位反转、旋转因子计算和多级流水线设计。
* **模块化设计**：将 FFT-256 点模块拆分为多个子模块（如计数器、蝶形运算单元等），提高了代码的可读性和可维护性。
* **资源优化**：通过合理设计数据路径和控制逻辑，优化了硬件资源的使用，减少了面积和功耗。

**2. 仿真与调试技能的提升**

* **VCS 前仿真**：使用 VCS 进行功能仿真，验证了设计的正确性，并通过波形分析定位了逻辑错误。
* **Verdi 调试**：利用 Verdi 的强大波形查看功能，快速定位了时序问题和数据路径错误，提高了调试效率。