

论文分类号: 2

学校代码: 10708

学 号: 1206030



陕西科技大学

SHAANXI UNIVERSITY OF SCIENCE & TECHNOLOGY

硕 士 学 位 论 文

Thesis for Master's Degree

基于 Vivado 的频谱显示系统设计与实现

曹梦娜

指导教师姓名: 张俊涛 教授

学 科 名 称: 模式识别与智能系统

论文提交日期: 2015 年 3 月

论文答辩日期: 2015 年 6 月

学位授予单位: 陕西科技大学

陕西科技大学

申请工学硕士学位论文

论文题目：

基于 Vivado 的频谱显示系统设计与实现

学科门类：工学

一级学科：控制科学与工程

培养单位：电气与信息工程学院

硕士生：曹梦娜

导 师：张俊涛 教授

2015 年 6 月

DESIGN AND IMPLEMENTATION OF SPECTRUM DISPLAY SYSTEM BASED ON VIVADO

A Thesis Submitted to

Shaanxi University of Science and Technology

in Partial Fulfillment of the Requirements

for the Degree of

Master of Engineering Science

By

Mengna Cao

Supervisor: Prof. Juntao Zhang

June 2015

基于 Vivado 的频谱显示系统设计与实现

摘 要

信号频谱分析在通信、信号检测与分析、数字信号处理等工程中应用广泛。随着系统复杂度的增加,对于频谱分析的精确度及实时性等方面也提出了更高的要求。FFT (Fast Fourier Transformation) 是实现频谱的一种重要方法,因此设计一种基于 FFT 算法的实时频谱显示系统在数字信号处理中具有重要作用。课题在分析 FFT 算法的基础之上,利用基于 Vivado 的软硬件协同设计方法,以 Xilinx 公司的 Zynq 为开发平台,设计并实现了一种可以实时、可靠显示音频频谱的系统。主要工作有:

(1) 基于 Vivado HLS (High Level Synthesis) 高层次综合工具设计 FFT IP (Intellectual Property Core)。利用 HLS 工具对使用 C 语言编写的 FFT 算法进行仿真和 RTL 级验证,并对 FFT IP 在占用资源和处理数据速度方面进行优化,最终生成了一个 RTL (Register Transfer Level) 级的可以处理实数和复数的 FFT IP,可以在 Vivado 硬件开发中使用。

(2) 利用 Verilog 语言设计 OLED (Organic Light-Emitting Diode) IP。借助 Xilinx 提供的 IP 搭建 OLED 硬件系统,利用 SDK (Software Development Kit) 编写 OLED 驱动程序,将软硬件部分结合起来在 ZedBoard 开发板上验证 OLED IP 和驱动程序。

(3) 设计音频驱动程序。驱动是根据 Xilinx 提供的音频 IP 和 ZedBoard 开发板外围的音频芯片 ADAU1761 来进行设计的,使用音频 IP 搭建测试音频驱动系统,验证设计的音频驱动程序。

(4) 设计并实现频谱显示系统。利用 FFT IP、OLED IP、音频 IP 以及 Xilinx 提供的其它 IPs 搭建音频频谱显示系统,在 SDK 中编写整体系统的控制程序,在 OLED 上实时显示频谱。使用 HLS 生成的 FFT IP 设计频谱显示系统对单频正弦信号进行频谱处理,并和 Xilinx 的 FFT IP 设计的频谱显示系统的结果进行对比,分析两次实验的性能和资源消耗情况。

实验结果表明音频信号经过频谱显示系统处理之后,可以在 OLED 上实时显示音频信号频谱,证明设计的 FFT IP 是正确的,同时也表明软硬件协同设计方法能够充分利用 FPGA 的并行处理的能力和软件实现方式的灵活性,为实际应用中提供一种实用的设计方法。

关键词: FFT IP, Vivado, 软硬件协同设计, 高层次综合, FPGA

DESIGN AND IMPLEMENTATION OF SPECTRUM DISPLAY SYSTEM BASED ON VIVADO

ABSTRACT

Signal spectrum analysis is widely used in communication, signal detection and analysis, digital signal processing and so on. With the increase of the complexity of the system, higher requirements for the accuracy and real-time of the spectrum analysis is put forward. FFT is an important method for realizing frequency spectrum, so the design of a real time spectrum display system based on FFT algorithm is important in signal processing. Firstly the FFT algorithm is analyzed, and then a co-design of hardware and software methodology based on Vivado is proposed. The Zynq provided by Xilinx Company is the development platform. A real-time, reliable audio spectrum display system is designed and realized. The main work is summarized as follows:

(1) FFT IP is designed based on high-level synthesis tool Vivado HLS, HLS is used to inspect the simulation and RTL of the FFT algorithm compiled by using C language. The FFT IP is optimized in the occupation of resources and data processing speed, Finally a RTL level FFT IP is generated which can process real and complex data. The FFT IP can be used in Vivado hardware development.

(2) OLED IP is designed by using the Verilog language. The hardware system of OLED is built with the OLED IP and the other IPs provided by Xilinx. SDK is used to compile the driver of OLED. Finally the software and hardware parts are put together to verify the OLED IP and driver programs on ZedBoard development board.

(3) The Audio driver is designed according to the Audio IP provided by Xilinx and the external Auau1761 chip of ZedBoard development board. A hardware system is designed with Audio IP to test the driver of Audio IP.

(4) A spectrum display system is designed and realized. The spectrum display system is designed by using FFT IP, OLED IP, Audio IP and other IPs provided by Xilinx. The system's test program is written in SDK, and the spectrum is real time displayed on OLED. The system designed by using HLS FFT IP is used to process the single frequency sinusoidal signal , and the result

generated by it is compared with the result generated the other system which is designed by using Xilinx FFT IP. Finally two experimental performance and resource consumption are compared and analyzed.

The experimental result shows that the audio signal spectrum can be displayed in real time on the OLED after the audio signal processed by the spectrum display system. In other words the result proves the correctness of the design of FFT IP. At the same time it shows that the co-design of software and hardware methodology can make full use of the parallel processing ability of FPGA and the flexibility of software implementation. This co-design method provides a practical design method for practical application.

KEY WORDS : Fast Fourier Transformation IP, Vivado, Co-design of hardware and software, High Level Synthesis, Field-Programmable Gate Array

目录

摘 要	I
ABSTRACT	II
1 绪论	1
1.1 课题背景与意义	1
1.2 数字信号实现方式的发展	1
1.3 主要工作和章节安排	3
2 FFT 原理和开发工具	5
2.1 FFT 算法原理	5
2.1.1 直接计算 DFT 的特点	5
2.1.2 基-2 时域抽取法	7
2.2 IP 介绍	10
2.3 Vivado HLS 工具	11
2.3.1 HLS 工具开发原理	11
2.3.2 HLS 工具开发流程	11
2.3.3 HLS 应用领域	13
2.3.4 HLS 优化设计	14
2.4 本章小结	16
3 频谱显示系统设计	17
3.1 开发平台介绍	17
3.1.1 Xilinx Zynq 平台介绍	17
3.1.2 软硬件协同设计原理	18
3.1.3 硬件开发环境	20
3.1.4 软件开发环境	21
3.2 频谱显示系统设计	21
3.3 FFT IP 设计	22
3.3.1 FFT 算法	23
3.3.2 FFT 的 C 模型实现	26
3.3.3 HLS 对 FFT 进行优化	36
3.3.4 FFT 优化前后结果对比	38
3.4 OLED IP 设计	40
3.4.1 OLED IP 整体设计	40

3.4.2 OLED IP 实现.....	43
3.4.3 OLED IP 测试程序设计.....	47
3.5 音频驱动设计.....	50
3.5.1 音频控制器介绍.....	51
3.5.2 音频驱动设计.....	52
3.5.3 设计测试程序.....	54
3.6 本章小结.....	56
4 频谱显示系统实现.....	57
4.1 硬件系统实现.....	57
4.2 软件程序实现.....	59
4.3 系统测试及结果分析.....	62
4.3.1 测试特定频率的信号.....	63
4.3.2 测试 MP3 音频信号.....	64
4.3.3 结果分析.....	65
4.4 本章小结.....	66
5 总结与展望.....	67
5.1 全文工作总结.....	67
5.2 未来工作展望.....	68
致 谢.....	69
参考文献.....	70
附录 A: 采用 HLS FFT IP 设计的系统.....	73
附录 B: Real FFT IP 的组成.....	74
附录 C: 采用 Xilinx FFT IP 设计的系统.....	75
附录 D: 测试频谱系统的主程序中 main 函数代码.....	76
攻读学位期间发表的学术论文目录.....	77
原创性声明及关于学位论文使用授权的声明.....	78

1 绪论

1.1 课题背景与意义

数字信号处理技术从开始出现到现在一直处于持续发展的状态，其应用领域也越来越广泛，已经变成了通信系统、数字视音频应用以及数字信号系统中最强大的技术之一，数字信号处理技术发展的同时也使的这些应用领域得到了更进一步的发展。数字信号系统^[1]比模拟系统在实现中的改善体现在计算数据的精确性、实时性以及对外界的抵抗能力增强、灵活的设计思想以及可编程等方面。

FFT^[2]是数字信号处理中的一种重要算法，在频谱分析中有着重要的应用。随着技术发展，对于频谱显示单元也提出高精度、高实时性等设计需求，本课题就是从现代行业对数字信号处理中对频谱显示技术提出的要求提出使用 Vivado HLS 实现频谱的一种设计方法。

本课题以此为背景，利用 Vivado 设计套件为开发环境，设计 FFT IP 以及相关 IPs 用于实现音频信号的频谱生成，利用该 IP 实现实时显示音频频谱硬件系统的设计，利用软件编写硬件接口、控制程序以及驱动程序，最终在 ZedBoard 开发板上进行验证。

1.2 数字信号实现方式的发展

数字信号处理传统的实现方式主要有如下这三种：

(1) 通过软件方式实现。优点是设计灵活，价格低，但是存在的不足之处是处理速度不够，实时性差。

(2) 通过专用 FFT 芯片来实现^[3]。适合对于信号处理速度要求不高的系统，但是由于功能固定，因而很难扩展，是一个串行处理的过程，因而对数字信号处理产生了非常严重的瓶颈。

(3) 通过 FPGA^[3] (Field-Programmable Gate Array) 实现。利用其可以并行执行的特点，达到提高执行速度的目的，在实时性和灵活性方面都能够很好的满足设计要求，以花费更多的硬件资源为代价，且开发难度大，有时候如果需要在时序等性能方面进行提高，还必须花费更多的时间去重复仿真与优化。如果把使用软件设计的模块移植到可编程逻辑 FPGA 硬件中进行实现，还得根据对硬件逻辑提出的需求将软件设计进行相应的转换，浪费了人力和时间。

因为单一的实现方式存在着一定的缺点和不足之处，有时候难以满足对设计提出的要求，因而就有了将多种方式结合的想法，例如将 DSP 和 FPGA 结合起来，充分利用各自在实现中所具备的优势和善于解决的问题从而来提高设计效果。随着数字信号的进一步发展，逐渐有了软硬件协同这种设计理念并逐渐被很多人应用于实际的系统开发中。

软硬件协同设计方法^[4]是指采用统一的工具对需要实现的任务进行描述,然后利用软件与硬件实现各自相应的任务,并在同一个开发环境中设计并实现系统的集成开发,最终完成设计任务。可以利用对系统进行验证^[5]的方式来检测所完成系统的正确性和性能效果;并且软硬件协同能够跨越软硬件环境之间的代沟,对系统执行优化操作使得它的性能更加完善。软硬件协同设计方法^[6]是充分考虑系统整体设计任务的前提下,分析软硬件各自所具备的功能、适合完成的工作以及可以使用的资源,然后合理的划分软硬件以便使整个系统在一个最完美和协调的状态下工作,从而达到提高系统性能和满足要求的目。在系统实现的过程中,软件部分和硬件部分^[7]不是各自独立的,而是互相影响,两者之间的关系贯穿在实现的全部过程中。

以往使用软硬件协同^[8]方法设计具有某种功能的系统时,首先是硬件开发人员按照性能等方面的要求完成硬件部分,然后软件编程人员去完成软件部分,这两部分工作没有并行执行去完成,而是将整体的设计目标进行模块化,然后将其实现或者采用从上到下的方法完成设计任务。硬件部分和软件部分两者^[9]是独立并各自完成任务的,两个部分是孤立的,设计人员因为在设计时缺少沟通,没有到达真正意义上的协同,最终^[8]只有通过多次对功能的划分进行修改并多次试验才能使设计满足要求。

软硬件协同^[10]这种实现方式最大程度的利用了异构多核的这种特殊处理器所具备的系统设计优势。游余新^[11]研究了复杂 SoC (System on a Chip) 的软硬件协同解决方案,并分析了划分到软件的功能在不同处理器上依次执行时的结果,这种设计方式很大程度上完善了使用多核处理器架构实现系统时进行验证的效率问题;刘洋^[12]设计了一种在不同架构的多核加速器架构,充分利用片上多核技术并结合软硬件协同这种设计方法所具备的优势,能够对以往的文档解析这种架构进行优化操作;韩红蕾^[13]等人研究调度算法,参照调度并结合软硬件划分技术,在此基础上提出了一种启发式算法,该算法具有高效的特点。

沈淦松^[14]等人利用软硬件协同设计方法实现了对复杂图像的处理,此设计结构简单,使用起来非常灵活,且开发费用较低。张开锋^[15]等人设计并实现软硬件协同的进化方法,这种方法主要是将硬件进化思想加入到图像处理研究中,这种方法功能强大并且具有简单灵活的特点。Usman Ali 等人^[16]将软硬协同设计的思想应用到实时视频跟踪方案中,这种设计方法大大地提高了软件算法的处理速度。李平^[17]等人提出并验证了一种基于软硬件协同技术的 FPGA 芯片测试方法。

软硬件协同处理架构的重要之处^[18]是不同架构处理器之间采用何种通信模式,针对主中央处理器与 FPGA 两个部分之间应该采用哪种通信方法才能使得效果更好,不同的科研人员提出了不同的方法。G.Nguyen^[19]提出了两种接口实现微处理器与 FPGA 的通信:

紧、松耦合接口，紧耦合采用的是 dual-RAM 和 register，但是实现过程要求较长时间，松耦合利用改变微处理器中指令级别的总线，大大改善了紧耦合的不足之处。

新一代的 FPGA 朝着在单独器件上实现整个系统的方向发展，2009 年 Xilinx All Programmable Zynq-7000 SoC 器件的出现，为数字信号处理的发展注入了新的活力；Zynq-7000 的出现实现了“软件”和“硬件”协同设计以及调试，这样使得设计者可以更加灵活的在成本和性能之间均衡。

Vivado^[20]高层次综合工具（High Level Synthesis, HLS）是在 2012 年由 FPGA 的生产厂家之一的 Xilinx 公司新发布的一种用于开发的集成设计工具，为软硬件以及 FPGA 的发展提供了新的设计思路，并加快了 FPGA 的开发。新发展^[21]起来的软硬件协同设计方法是利用统一的语言方式对系统进行描述，在满足整体设计性能的前提下，研究并熟悉软件和硬件所具备的特征和体系结构，最大程度的找出两者之间潜在的并发性并发掘出一个最佳的功能划分方式^[21]，以便让系统能够在满足设计任务的条件下高效率地运行。

在 HLS 工具出现之前，对于通过 C、C++或是 System C 编写的代码模块若想使其通过硬件方式实现，就要求开发人员用 Verilog HDL 或是 VHDL 的方式重新描述出模块，这种转换过程速度很慢而且因为是手动的方式所以容易出错。HLS 工具解决了以往开发过程中出现的难度大问题，用户可以通过 HLS 工具设计流程开发 IP 并将其集中到自己的设计当中。Vivado HLS 工具对于使用 C、C++或 System C 代码建立的系统或者是模块的设计，能根据内置的编译器进行编译，并对它进行综合，然后按照性能的要求对其进行优化操作，并将 C 语言代码转换成硬件语言，直接提取出 RTL 级模型，也就是能产生实现硬件加速所需要的 Verilog HDL 或者是 VHDL 代码，用于 FPGA 的开发，不仅可以进行 System C 架构及仿真，进一步创建自己的测试平台，验证设计的架构行为和功能，与此同时还支持 AXI4 接口，可以直接在 Zynq 平台的硬件逻辑开发部分使用。

本课题提出的软硬件协同设计方法综合了软件实现的灵活性和 FPGA 实现的快速并行优势，因而具有结构简单，处理速度快，灵活性好等优点。

1.3 主要工作和章节安排

本课题旨在设计一个频谱显示系统，实现音频信号频谱的实时显示。主要过程包括：（1）利用 HLS 高层次综合工具设计 FFT IP，实现音频信号频谱的产生；（2）利用 Verilog 语言设计用于显示频谱信息的 OLED IP；（3）设计 ADAU1761 音频驱动程序；（4）设计并实现完整的频谱显示系统，以音频信号作为激励，利用 FFT IP 对音频信号进行处理，音频频谱在 Zynq 系列的 ZedBoard 开发板所提供的 OLED 显示屏上显示，验证频谱显示系统设计的正确性。

本论文后续章节安排如下：

第 2 章主要是对相关原理进行介绍：包括 FFT 算法原理和 HLS 开发原理。介绍基-2 时域抽取算法、IP 的概念以及高层次综合工具 HLS 的实现原理、开发流程以及对设计进行优化的方法。

第 3 章是关于频谱系统的整体设计以及各个子模块的设计。介绍 Zynq 开发平台的架构和基于该平台的软硬件协同设计开发原理；利用 HLS 设计一个 RTL 级 FFT IP，并对设计进行优化操作使其满足性能和资源占用等要求；利用 Vivado 设计 OLED IP 和 OLED 驱动程序并验证；设计并验证音频驱动程序。

第 4 章搭建完整的频谱显示系统。在 SDK（Software Development Kit）中编写硬件系统中各个模块相应的驱动程序及整体系统的测试程序。分别以 HLS 创建的 FFT IP 搭建的系统和使用 Xilinx 提供的 FFT IP 搭建的系统，将频谱信息在 ZedBoard 开发板上显示，对比验证系统的正确性。

第 5 章是全文工作总结和未来工作展望。对本次课题中完成的工作进行总结，分析频谱显示系统在设计过程中存在的不足之处，确定下一步的工作目标。

2 FFT 原理和开发工具

伴随着数字化这门技术的持续发展和成熟，数字信号处理也随之处于更进一步的发展中，目前已经深入到很多学科。FFT 算法是该领域一个比较重要的算法，而且应用广泛。FFT 之所以应用广泛，其主要原因^[22]之一是它可以将不容易在时域处理的信号转换到频域中按照要求对其处理，获得期望的信号类型或是成分，再经过 FFT 反变换就能够得到时域信号，给下一步应用做好准备。

FFT 主要应用领域如下：

(1) 频谱分析^[23]。分析各种运转设备整体或者部分运转情况的频谱，根据频谱信息可以了解机器的设计数据以及运转是否正常，而且如果检测的结果和预期的有出入，则频谱信息可以帮助工作人员找到故障可能出现的地方，方便进行维修处理从而保证设备正常运行。

(2) 滤波^[24]。滤波是 FFT 应用最广泛和最多的地方，使用 FFT 对信号处理后^[25]可以根据信号的特征对其进行滤波，滤除不需要的部分，再经过 FFT 的反变换就能够获得想要的信号成分。

2.1 FFT 算法原理

DFT (Discrete Fourier Transform) 运算变换特点^[26]是总运算量大小与进行变换的信号长度 N 有关，与 N 的平方呈正比的一种数学关系，即运算量会随着 N 的增大而迅速地增多，因此在实际应用中具有一定的局限性。在人们多次的研究中发现，DFT 算法^[27]在运算过程中具有潜在的特点，如周期性、对称性等，根据这些运算规律对 DFT 算法进行改进得到了一种高效的运算方法即 FFT，对 FFT 算法以及实现它的方法做分析和研究具有现实意义。

2.1.1 直接计算 DFT 的特点

假设有一个离散信号 $x(n)$ ，其长度是 N ，则 DFT 的实现过程如式 (1-1) 所示。

$$X(k) = DFT[x(n)] = \sum_{n=0}^{N-1} x(n)W_N^{nk} \quad k = 0, 1, \dots, N-1 \quad (1-1)$$

其中

$$W_N^{nk} = \exp(-j2\pi nk / N) \quad (1-2)$$

我们经常说的算法运算量是描述在某个运算过程中总的加法次数与总的乘法次数两者之和。其中： $x(n)$ 为时域信号； $X(k)$ 为 k 点处的频域值； W_N^{nk} 为旋转因子； $x(n)$ 、 $X(k)$ 、 W_N^{nk} 都是复数。经过研究发现对于变量 k ，如果直接按照 DFT^[28]表达式 (1-1) 进行，在

经过 N 次复乘以及 $(N-1)$ 次复加运算之后才能得到 N 点 $X(k)$ 结果值，那么可以很容易得出这样一个结论，即计算完整个过程共需要复乘运算 N^2 次，复加运算是 $N(N-1)$ 次。当 N 值很大^[29]的时候，需要的复乘与复加运算都等于 N^2 ，其执行次数之多很容易看出来。对于实时性要求高的处理设备会要求 CPU 运算速度很高，DFT 这种实现方式必然是一个很难解决的问题，工程上也很难实现。因此研究如何减少其运算量、达到高实时性就成为了一个很重要问题，随之出现了 FFT。

如果将长度 N 变成比较短的序列然后进行运算，那么 DFT 所需要的复乘运算次数就能减少很多，从而达到降低运算量的目的。另外从表达式 (1-1) 可以看出，旋转因子 $W_N^{nk} = \exp(-j2\pi nk/N)$ 存在潜在的周期性和对称性，利用这些特性可以进行化简操作。

周期性表现在

$$W_N^{m+IN} = e^{-j\frac{2\pi}{N}(m+IN)} = e^{-j\frac{2\pi}{N}m} = W_N^m \quad (1-3)$$

对称性表现在

$$W_N^{-m} = W_N^{N-m}, \quad W_N^{m+\frac{N}{2}} = -W_N^m \quad (1-4)$$

FFT 变换减少^[30]运算量的思想就是在很大程度上利用 W_N 所具备的周期性、对称性，其具体方式是：把变换区间为 N 的离散信号的 DFT 变换经过多次分解之后，转换为计算较短离散信号的 DFT，把短变换的 DFT 结果经过某种组合便能够获得原始 N 点离散信号的 DFT 结果。利用 DFT 运算量^[31]与 N^2 之间呈数学正比关系可以知道，当 N 减小时，在不影响结果的前提下，分解法使得运算量会大大降低，主要体现在速度方面。计算 $N=2^M$ 长度 DFT 变换总共需要复乘运算将会由原来 N^2 次降到 $(N/2)\log_2(N)$ 次，复加运算则是由原来的 $N(N-1)$ 次降为 $(N/2)\log_2(N)$ 次。

根据变换区间 N 与 2 的整次幂的关系能将 FFT 算法的实现分为两种^[32]： N 是 2 的整次幂类型，这种算法有常见的基-2 抽取方式、基-4 抽取方式以及综合两者产生的分裂基抽取方式；变换区间 N 不是 2 的整次幂，Winograd 是这种情况的代表性算法。采用硬件方法设计算法时，不仅要考虑算法可能的运算量大小之外，还应该将算法的复杂性和模块化考虑在其中，这也是两个非常重要的因素，在考虑的因素中，要优先考虑的是尽可能的使实现方式控制容易，其次才会考虑降低运算量的这个因素。目前利用 FPGA 设计 FFT 算法都是针对 N 是 2 的整数次幂的这种情况，第二种情况虽更具有研究意义，但不适合利用硬件方式来实现。

FFT 根据抽取规则的不同又可以分成两类^[33]：一类在时间域按照时间序列进行抽取的方法，即时间域抽取方式（Decimation in time, DIT）；另一类在频域对频域序列进行抽取的方法，也就是频率域抽取方式（Decimation in frequency, DIF）。

2.1.2 基-2 时域抽取法

对于一个满足 $N = 2^M$ 的离散信号 $x(n)$ ，所谓的时域抽取法即按照变量 n 的奇偶性来进行分解，得到两个长度都是 $(N/2)$ 的子序列，如式 (1-5) 和 (1-6) 所示。

$$x_1(r) = x(2r) \quad r = 0, 1, \dots, \frac{N}{2} - 1 \quad (1-5)$$

$$x_2(r) = x(2r+1) \quad r = 0, 1, \dots, \frac{N}{2} - 1 \quad (1-6)$$

则 $x(n)$ 的 DFT 为

$$\begin{aligned} X(k) &= \sum_{n=\text{偶数}} x(n)W_N^{kn} + \sum_{n=\text{奇数}} x(n)W_N^{kn} \\ &= \sum_{r=0}^{N/2-1} x(2r)W_N^{2kr} + \sum_{r=0}^{N/2-1} x(2r+1)W_N^{k(2r+1)} \\ &= \sum_{r=0}^{N/2-1} x_1(r)W_N^{2kr} + W_N^k \sum_{r=0}^{N/2-1} x_2(r)W_N^{2kr} \end{aligned} \quad (1-7)$$

由于

$$W_N^{2kr} = e^{-j\frac{2\pi}{N}2kr} = e^{-j\frac{2\pi}{N/2}kr} = W_{N/2}^{kr} \quad (1-8)$$

所以

$$X(k) = \sum_{r=0}^{N/2-1} x_1(r)W_{N/2}^{kr} + W_N^k \sum_{r=0}^{N/2-1} x_2(r)W_{N/2}^{kr} = X_1(k) + X_2(k) \quad k = 0, 1, \dots, N-1 \quad (1-9)$$

式 (1-9) 中的 $X_1(k)$ 与 $X_2(k)$ 分别是长度 $N/2$ 的 $x_1(r)$ 和 $x_2(r)$ 的 DFT 计算表达式，计算方程如式 (1-10) 和 (1-11) 所示，即

$$X_1(k) = \sum_{r=0}^{N/2-1} x_1(r)W_{N/2}^{kr} = DFT[x_1(r)]_{N/2} \quad (1-10)$$

$$X_2(k) = \sum_{r=0}^{N/2-1} x_2(r)W_{N/2}^{kr} = DFT[x_2(r)]_{N/2} \quad (1-11)$$

考虑 $X_1(k)$ 和 $X_2(k)$ 具有周期性，周期为 $N/2$ ，同时

$$W_N^{k+\frac{N}{2}} = -W_N^k \quad (1-12)$$

所以 $X(k)$ 可以分解成前后长度为 $(N/2)$ 的两段，也即式 (1-13) 和 (1-14) 所示。

$$X(k) = X_1(k) + W_N^k X_2(k) \quad k = 0, 1, \dots, \frac{N}{2} - 1 \quad (1-13)$$

$$X(k + \frac{N}{2}) = X_1(k) - W_N^k X_2(k) \quad k = 0, 1, \dots, \frac{N}{2} - 1 \quad (1-14)$$

利用这种方法便可以实现把 N 点 DFT 进行一次分解，结果得到两个 $(N/2)$ 点长度的 DFT，式 (1-13) 与 (1-14) 的计算过程如图 2-1 所示。

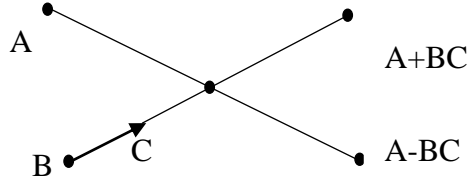


图 2-1 时域蝶形运算单元

Fig.2-1 The time domain butterfly unit

图 2-2 是进行一次时域分解的过程图，其中序列长度是 $N=2^3=8$ 。

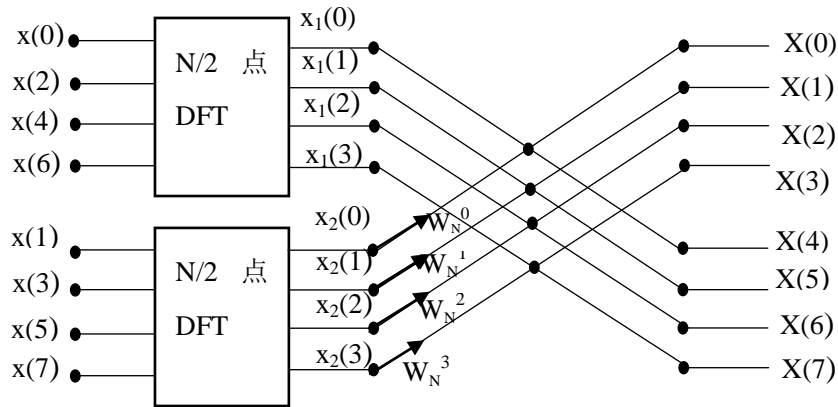


图 2-2 一次时域分解

Fig.2-2 One time decomposition

根据图 2-1 我们能够看出，执行任意的蝶形单元的计算包含一次复乘及两次复加；再根据图 2-2 可以看出在一次分解之后，一个变换点数是 N 的离散信号对应的 DFT 运算实现方式^[34]转换成了求取两个长度为 $N/2$ 的 DFT 运算和 $N/2$ 个蝶形单元结果值的方式。由前面的分析可以知道，求取长度 $N/2$ 序列的 DFT 值包含的复乘执行次数是 $(N/2)^2$ ，复加运算 $(N/2)(N/2-1)$ ，所以求取 N 点序列的 DFT 包含的复乘法运算次数是

$$2\left(\frac{N}{2}\right)^2 + \frac{N}{2} \approx \frac{N^2}{2} \quad (N \gg 1) \quad (1-15)$$

复数加法次数是

$$N \left(\frac{N}{2} - 1 \right) + \frac{2N}{2} = \frac{N^2}{2} \quad (N \gg 1) \quad (1-16)$$

可以发现在一次分解之后，计算的运算次数已经接近原计算运算量的 $(1/2)$ ，运算次数大大降低，如果 N 点序列在一次分解完成以后得到的 $(N/2)$ 依然为偶数就可以继续执行分解，直到不能再分解为止，8 点的 DFT 时域分解完整过程如图 2-3 所示。

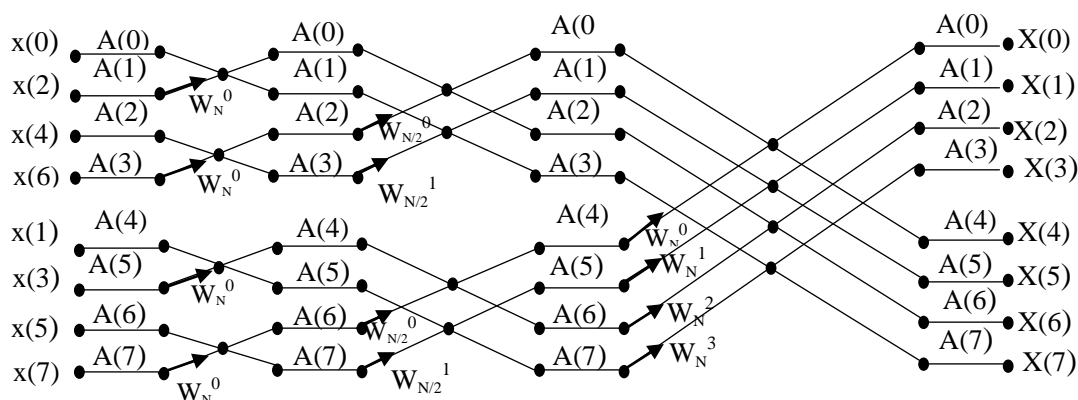


图 2-3 8 点完整时域分解图

Fig. 2-3 Flow chart of complete time-domain extraction of 8

图 2-3 中 A 表示数组，在这里的作用是存放输入序列值以及每级计算值。从图 2-3 可以知道，对于一个长度满足 $N=2^M$ 的序列，在执行 M 次的分解之后，能够将 N 点的 DFT 计算过程变成求 N 个长度都是 1 的离散信号的 DFT 运算和 M 级蝶形单元值的方式。1 点的 DFT 其实就是时域序列本身的变换结果，每级都存在 $(N/2)$ 个蝶形单元，且都包含 $(N/2)$ 次复乘与 N 次复加，因此 M 次分解需要执行的复乘次数为

$$C_M = \frac{N}{2} \cdot M = \frac{N}{2} \log_2 N \quad (1-17)$$

复加次数为

$$C_A = N \cdot M = N \log_2 N \quad (1-18)$$

当需要处理的点数较大时即 $N \gg 1$ 时， $N^2 \gg (N/2) \log_2 N$ ，由基-2 算法的计算特点可以知道，FFT 比 DFT 有更大的计算优势，运算量大大降低。以 1024 点的运算^[35]为例，若用 DFT 直接计算需要的乘法数是 $1024^2=1048576$ ，而采用基-2 FFT 算法，则需要 $(1024/2) \cdot \log_2 1024=5120$ 次，由此可见 FFT 算法的优势。

如图 2-4 给出的是按照 DFT 公式直接求取变换结果需要执行的乘法次数和使用 FFT 方式求取时需要的复数次数之间的曲线关系图。

根据图 2-4 中的曲线关系可以很容易得出的结论是 N 越大时，FFT 所隐含的优势也就更加突出。

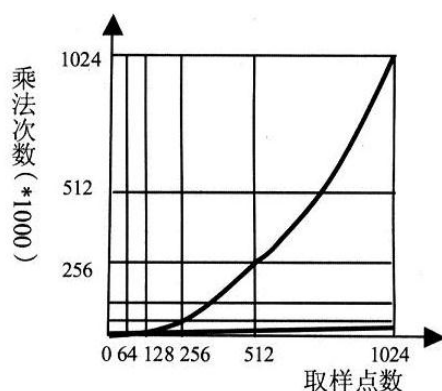


图 2-4 运算量对比图

Fig. 2-4 Computation comparison chart

从前面分析基-2 时域抽取法的过程可以看出，旋转因子在计算中起着非常关键的作用。由于旋转因子为 $W_N^m = \cos(2\pi m/N) - j\sin(2\pi m/N)$ ，为了求得旋转因子值，必须求取正余弦函数值。而当序列 N 很大的情况下，计算正余弦的运算量之大可想而知，其直接影响着 FFT 运算的速度，所以旋转因子的产生方式以及产生所占用的时间会直接影响整个运算速度。在实际应用中计算旋转因子主要有两种方式^[36]：一种方式是在计算过程中产生所要用的旋转因子系数值；另一种方式是事先根据要求将需要用到的旋转因子 $W_N^m, m=0,1,\dots,N/2-1$ ，计算好然后放在数组里存储，作为一个存储器，在代码执行过程中通过对存储器进行查找来获取需要的旋转因子值，这种方式的不足之处在于是花费了更多的存储空间来换取运算量的降低。

2.2 IP 介绍

IP^[37]，也即常说的知识产权核，指由第三方厂家所提供的、表现方式为逻辑单元、芯片设计能够被多次重复使用的模块。IP 一般都事先已经通过了设计验证，因为具有可重用特性，能够适用于不同的系统，因此开发人员利用 IP 设计系统将会在很大程度上减少设计时间。

FPGA IP^[38]是厂商和其他的第三方提前设计好的一些普遍适用的模块，然后根据各种类型的 FPGA 芯片的结构，从布局与布线等方面对设计实行优化操作之后得到的拥有自主知识产权且具备一定功能的模块。在开发 FPGA 大型系统时如果优先考虑使用 FPGA IP，将会大大减少开发时间、降低不必要的风险、开发成本也能够减少且能够改善性能和可靠性。

Xilinx 公司提供的 IP 是使用 HDL 编写而实现的大规模或者比较复杂的系统级模块，具有一定的功能，也具备对其进行修改重用的特性，它可以对所有的 Xilinx FPGA 芯片都得到最优化实现，与此时还支持 VHDL 或 Verilog HDL 文本级的功能仿真，可适用于规范的 EDA（Electronic Design Automation）工具中实现设计。

2.3 Vivado HLS 工具

Vivado HLS 工具^[39]的应用突破了以往开发 FPGA 时采用 HDL 文本语言设计系统造成的瓶颈,通过 C、C++、System C 可以对信号或系统实现直接建模,对不熟悉 HDL 编程语言的软件开发人员或者研究 FPGA 的硬件研发人员来说,都能够称为是 FPGA 的一类全新的设计理念和想法。

开发人员可以通过 HLS 开发工具^[20]完成以 All Programmable SoC 为基础、利用软硬件协同调试方式开发产品,让产品的开发周期缩短很多。开发人员在熟知软件和硬件各自的优缺点和适合的应用领域之后,再利用 HLS 设计工具对 FPGA 进行开发时效率就能够得到很大程度的提高^[40]。用户不再需要具有特别扎实的硬件知识和熟练 HDL 编程的能力就能够建立数学模型, HLS 在以后的 FPGA 开发中将占据着重要的地位和实用价值,对数字信号的发展起到了极大的推动作用,加速了 FPGA 在高性能信号和数据领域的推广。这种方法能降低 FPGA 的开发时间,并使效率得到提高,而且还可以在很大程度上利用已经存在的可靠软件代码^[39]进行设计。

2.3.1 HLS 工具开发原理

HLS 工具之所以在开发 FPGA 中占据着一定的优势,并成为未来 FPGA 开发中优先考虑的开发工具,主要是因为其开发简单,利用 C 语言便能够实现系统的设计,对于不太熟悉硬件开发的人员来说无疑是一个很好的选择。

使用 HLS 工具将 C 语言模型转换成硬件模型的思路,即 HLS 开发原理如下:

(1) 该工具能够直接从 C 语言描述的源代码中创建一个 RTL 级实现。HLS 工具能从 C 源代码的顶层获取控制命令和数据通道,再基于默认和用户应用的指令实现设计。

(2) HLS 通过调度和绑定过程将 C 代码映射到硬件逻辑。调度和绑定是 HLS 工具的核心,调度是用来确定操作将发生在哪个时钟周期,考虑控制、数据流和用户命令;绑定确定每个操作所使用的库单元,例如元件的延迟。例如使用 C 语言编写的数组, HLS 开发工具能利用内置的编译器将其直接转换成 RAM 寄存器,即数组对应的 RTL 实现就是寄存器,实现数组的目标可以是库里的任何存储器资源。

2.3.2 HLS 工具开发流程

Vivado HLS 设计工具具有帮助减少硬件系统功率消耗和设计成本的作用,而且还能够在设计性能方面进行提升,加快设计的实现速度; HLS 工具能够实现功能级和结构层两个方面的验证;在提取方面, HLS 可以提取出接口的数据类型、接口和类。

HLS 的优势体现在:

(1) 从相同的源代码描述中可以实现很多的设计,也就是说: 1) 进行较小、较快、优化的系统设计; 2) 可以对设计方法进行“探索”,从而找到一种最有利的解决思路。

(2) HLS 容易移植的功能，体现在这么几个方面：1) 处理器和 FPGA；2) 技术的转移；3) 减少开销；4) 降低功耗。

(3) HLS 命令的优先级体现在：1) 满足性能的要求（要求的延迟和吞吐量）；2) 降低延迟；3) 减少面积。

图 2-5 表示的是采取 Vivado HLS 进行设计时最普遍的一种开发过程。最关键的工作之一是利用 C 或者是 C++ 语言编写一个具备特定作用的函数和适用该设计的测试平台，该平台可以检测搭建的系统正确性和性能，然后再通过 C 仿真器对模型所具备的功能进行检验；假如结果满足设计提出的需求，便使用 Vivado HLS 工具把 C 模型转换为相应的 RTL 级模块。有了 RTL 模块之后，就可以通过 HLS 内置的仿真器来检验前面建立的架构还有它的功能^[41]。

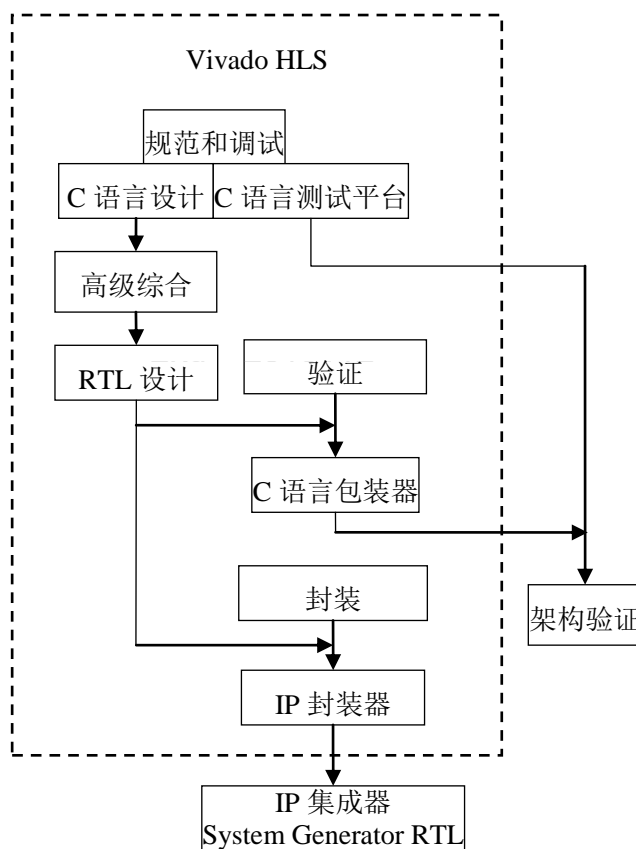


图 2-5 HLS 开发流程图

Fig. 2-5 Process of HLS development

HLS 工具关键的特点之一是利用 C 语言编写和描述算法或对系统设计提出的功能和性能要求进行编写和描述。C 语言描述系统和采用硬件结构实现系统设计的对应关系：

(1) 函数：所有 C 编写的代码都是由函数组成的，与硬件逻辑的设计思路是一样的，用函数表示设计的层次，从而使设计变得有条理；每个函数将会被翻译成一个 RTL 模块，用于表示 Verilog 模块或者 VHDL 实体。

(2) 参数: C 语言中顶层文件的参数用来定义 RTL 描述硬件设计时的接口的。

(3) 类型: C 语言代码定义变量时一定是具有属于自己的数据类型, 不同的数据类型占用的内存将会有所不同, 这都会导致翻译成硬件设计时对性能和面积的不同影响。

(4) 循环: 一般用 C 语言描述较为复杂的系统时函数都会包含循环, 这样就能够避免大量的代码; 对硬件而言, 对循环方法的不同处理将会对面积和性能产生不同影响。

(5) 数组: C 语言常用数组来存放数据, 一般对应着硬件设计的 IO 口, 所以它们直接影响着设备的 IO 端口, 有时会成为影响设计性能的瓶颈。

(6) 操作符: C 代码的操作符有时可能会要求共享, 通过共享可以控制面积或者指定硬件设计的实现来满足对设计性能的要求。Vivado HLS 尽可能最大限度的降低操作符的个数, 在满足约束添加的前提下, HLS 会自动寻找最小化的面积, 即尽可能使用最少的逻辑资源。

本课题中使用 C 语言编写的 FFT 算法就是从上面六个方面进行优化设计的。

2.3.3 HLS 应用领域

(1) 加速 FPGA 设计

若是 FPGA 项目的开发时间紧张, 但是对逻辑以及时序没有特别严格的要求, 或者是存在 FPGA 项目相应的已经通过验证的 C 程序算法, 那么利用 Vivado HLS 工具实现系统的设计就是一个很好的想法。HLS 工具开发项目的关键点之一就是利用 Matlab 和 C 进行建模操作, 实际用到 HLS 工具的时间是比较短的。若是想要设计一个新系统, 可以先利用 Matlab 仿真软件对自己建立的算法模型执行检验, 看其是否具备期望的功能, 如果满足就能把 Matlab 程序转换成 C、C++ 程序; 或者采用 C 直接编程然后通过 Matlab 进行仿真。算法无误之后就可以利用 HLS 开发工具对 RTL 级模型进行验证, 分析占用的逻辑资源以及速度并根据分析的结果做出相应的优化操作。如果结果不满足性能要求, 就进一步修改优化命令, 直到最终的综合结果满足期望的设计性能。

(2) 数据处理

Vivado HLS 工具可以识别并处理的数据类型有很多种, 主要有标准 C 整数类型、任意精度类型和浮点类型三种。

1) 通过任意精度的数据进行系统或者是模型的设计时, 可以改变通过 C 语言编写 FPGA 模型由于精度因素引起的不足。这意味着只要用户愿意, 可以^[42]在实现算法的设计工具中很好的利用这种工具, 而不是只能在以往所说的硬件环境下。所具备的明显优势在于对设计的算法进行验证时比通过硬件方式实现有数量级的改善, 即可以在改善算法运行速度的时候同时寻找算法中隐含的并行执行特性。可以设计 C 语言中所说的浮点型有两种: 单精度类型的数据, 其占用的数据存储宽度是 32 位; 双精度类型数据, 其占

用的数据存储宽度是 64 位；除此之外，还可以设计常用的整型类型，即占用的数据存储宽度都是整数位的字节。

2) C 中没有相应定义而且使用的数据存储空间位也非整数的一类数据，例如 7 位，24 位的这种数据类型。如果不对其进行定义，而直接进行综合的话，在综合的时候就会默认使用整数字节的寄存器型，这种情况就无法很好的利用 FPGA 逻辑资源。HLS 工具很好的解决了这一个问题，利用 Vivado HLS 内任意精度类型定义的数据，可以根据需要指定每一个整型或者是浮点型数据的位数，它可以识别并处理任何想要的位数数据并可以对其进行综合操作。

3) 尽管 FPGA 实现定点运算比浮点型运算要快，且面积更高效，但有时候也需要用浮点型来描述系统，因为定点数据位宽有限，需要深入分析才能更好的决定整个设计的中间位宽变化的模式，而浮点型数据比定点数据类型具有更大的数据动态范围，因而在许多实际的算法设计中它在数据类型方面体现了很大的优越性。

(3) IP 封装

IP 的封装是使用 Vivado HLS 工具进行 IP 设计的最后也是比较重要的一步，在进行 IP 的封装的前一步是 RTL 级实现。这两步是在 Vivado HLS 工具自带的编译器中完成的。Vivado HLS 工具进行 IP 封装^[43]时是以 IP-XACT 等多种可以让别的 Xilinx 工具识别并接受的类型格式输出，方便后面的工作中在 Vivado 开发环境搭建系统时使用。这种方式制作的 IP 能够被 FPGA 开发工具直接识别并在设计中使用，加快了系统设计的速度。

2.3.4 HLS 优化设计

HLS 工具可以对 C 代码执行优化命令，对功能、环路和数组的架构进行优化，以便制作更加优越的 RTL 级模型，很好的适应性能以及资源占用的提出的要求。首先是综合初始的设计，并分析限制性能和面积的原因，利用用户的指令改进初始的设计，以满足性能的要求。

HLS 提供了很多可以使用的优化命令对 C 语言设计的系统进行优化操作，使其在资源占用率、处理速度得到改善。HLS 优化策略的对象主要包括：改善延迟和吞吐量、循环、存储数据的数组以及面积的优化。

(1) 延迟和吞吐量

设计延迟是指从输入数据直到产生输出结果之间经历的周期数。

吞吐量是指在两个新输入数据之间的周期数。

延迟和吞吐量的关系：(1) 一般默认情况下，在没有并发的情况下，吞吐量与延迟是相同的；(2) 当前一组交易完成后，开始进行下一组新的交易，即当一组输入有了输出之后，进行下一组的输入，如此循环；(3) 吞吐量和延迟之间具有某种特定的关联。

Vivado HLS 默认情况下会尽量的较少延迟，将延迟降低到最小。HLS 工具默认的优化命令中，降低延迟的方法有：（1）对函数的处理：通过并行运算，可以减少操作，从而减少延迟。（2）对循环的处理：默认情况下，Vivado HLS 并不会调度循环使其自动的并行执行，想要使其并行执行循环，则必须使用数据流（dataflow）优化或者是循环展开（unroll）优化策略。（3）操作：Vivado HLS 允许在函数和循环内并行执行操作，用来将延迟降低到最小。

（2）优化流量

数据流（dataflow）优化：用于顶层函数的粗粒度优化，添加数据流优化命令能够实现改善吞吐量的要求。数据流优化可以使函数或者循环操作代码并行执行，可以提高整个设计的流量，但是这是以花费更多的逻辑资源为代价实现的，即在设计中使用额外的存储器件达到减少吞吐量的目的。

流水线（pipeline）优化：流水线优化是在操作符级上的细粒度优化。可以对循环和函数使用流水线操作，经过流水线操作处理后，允许并行执行函数和循环内的操作，流水线策略可以改善吞吐量，改善子功能或是循环的流量，减少吞吐量周期数。对于循环而言，当有多重循环嵌套时，流水线命令加在不同的地方将会导致不同优化结果。

（3）数组优化

数组是一种直观且有用的软件结构，能使设计者和阅读者很容易理解 C 算法所表达的代码含义，但是访问数组却成为了硬件性能上的瓶颈。HLS 工具在默认情况下会将 C 语言描述中的数组映射成硬件实现中的 RAM 存储器，HLS 将决定要实现的 RAM 哪些类型与存取和要求的类型的关系，可以利用资源指令隐含所用到存储器的状态。RAM 有单端口和双端口之分，如果是为了提高吞吐量，则使用双端口 RAM，反之则使用单端口 RAM，可以根据需要选择。

Vivado HLS 对数组的优化主要是“分割”（partition）和“重组”（reshape）。主要作用是：能够对 RAM 执行更好的优化配置使其状态更佳；改善存储器这种资源的实现方式使其更佳。

数组分割（partition）：数组分割是指将一个数组分割成更小的元素，数组分割可以分割成任何的维数，且所有的分割也可以使用相同的资源目标。如果没有指定分割的维数 dimension，则将其假定为 0。Vivado HLS 可以自动地将数组进行分割，以提高设计吞吐量。

数组重组：数组重组将分割的数组重新组合成一个单个的数组，数组重组命令选项和数组分割完全一样，但是数组重组命令会自动地将各个数组进行组合，得到一个单个元素。

（4）面积优化

例如最小化数据位宽，映射较小的数组到较大的数组以更好的利用存在的 RAM；控制设计的结构；内联命令能够直接影响 RTL 的层次和优化；控制功能调用层次和循环层次等来实现面积优化。

2.4 本章小结

本章主要介绍了 FFT 的应用以及 FFT 实现的原理，同时也对基-2 时频域抽取法和减少 FFT 运算量的途径和方法以及 IP 的概念、功能进行说明。详细介绍了 Vivado HLS 高层次综合工具所具备的功能、特点、开发流程，与此同时还将 Vivado HLS 所具有的优势和它的应用领域、在设计中使用 HLS 进行系统设计时的一些优化命令的使用和作用进行介绍。

3 频谱显示系统设计

3.1 开发平台介绍

3.1.1 Xilinx Zynq 平台介绍

软硬件协同这种设计理念需要处理器和 FPGA 的支持，而 Xilinx 公司提供的 Zynq-7000 系列开发平台是一款异构、多核处理器架构，是一款很适合这种协同设计理念的开发架构。内嵌有双核 ARM 处理器和 FPGA，这款 FPGA 结构体系^[45]是将处理器看成是核心部分，而将 FPGA 作为中心处理器的一个外围设备，在单芯片上能够提供软、硬件以及能够重新编程 IO 的功能。

简单来说，Xilinx Zynq 是一款低功耗、高性能及高容量的 FPGA 硬件可编程器件，用户可以根据自己的设计进行编程，搭建自己专用的全可编程 SoC。

Zynq 7000 系列的 ZedBoard 开发板由两部分组成：（1）处理器部分，在 Zynq-7000AP SoC 平台里被称为 Processing System，简称 PS；（2）FPGA 部分，被称为 Programmable Logic，简称 PL。

在 PS 部分封装了内存控制器和很大数量的外围设备，以便使 Cortex™-A9 双核处理器部分在 Zynq-7000 中能够彻底与 FPGA 部分独立起来，采用 ARM 处理器部分代替 FPGA 来控制运行。在开机启动时 ARM 处理器部分会控制独立于 FGPA 的其它各个部分的工作，这些都是在 FPGA 之前完成的，换句话说就是 ARM 部分可以独立运行，而 FPGA 部分可以不参与其中，FPGA 硬件部分可以用于扩展子系统。处理速度高至 1GHz 的 PS^[44]部分在 Zynq-7000 开发平台中起着控制整个系统的作用，不仅如此还能通过进一步扩展 NEON 协处理器和单精度浮点型单元对此进行提高和增强，而且处理丰富的外围设备是处理器与外部其他设备之间进行通信的接口，如 SPI 控制器、串口控制器(UART)、USB 控制器等。

PL 包裹 PS 部分，整个 PL 部分可视为 PS 的另一个具有可重配置特点的“外设”，通过 AXI 总线与 PS 连接。PL 部分提供了多达 500 万逻辑门的可编程逻辑单元，能够灵活地用于各种开发。FPGA 可编程逻辑部分则利用多口高性能接口与 PS 部分进行连接，从而达到 PS 与 FPGA 两者之间高带宽通信的目的。在 PL 部分，用户可根据需要，使用硬件描述语言，实现自定义逻辑，从而设计了具有特殊功能的“外设”。除此之外还允许 FPGA 按照需求自己定义 IO 接口，这些接口可定义为高速的用于通信的串行口。

本课题将使用 ZedBoard 开发板验证系统设计，介绍主要用到的外围设备。

（1）开关

ZedBoard 提供了 8 个 dip 封装的用户开关 SW0-SW7 用作输入信号，各个开关和芯片上的引脚对应关系如表 3-1 所示。

表 3-1 用户开关连接

Tab. 3-1 User switches connected

开关名	Zynq 对应引脚	开关名	Zynq 对应引脚
SW0	F22	SW4	H19
SW1	G22	SW5	H18
SW2	H22	SW6	H17
SW3	F21	SW7	M15

(2) USB 接口

ZedBoard 开发板提供了 3 个 Micro USB 接口，分别是 USB OTG、USB-UART 和 USB-JTAG 编程口，在本课题中用到了后两种接口。

1) USB-UART

USB-UART 在开发板上对应 J14 端口，是利用 Cypress 公司提供的 USB-UART 桥接芯片 CY7C64225 连接到 Zynq-7000 SoC 的 PS 部分 UART 外设，USB 连接器采用了 TE 的 MicroUSB 型连接器(1981584-1)。CY7C64225^[10]用的是 28 个引脚的 SSOP 封装模式，体积很小，而且它的内部集成了 USB 收发器、EEPROM、晶体振荡器、端接电阻以及电压调节等所有功能，极大减少了外围电路。

CY7C64225 支持 USB 2.0 Full-Speed 数据传输，速率达到 12Mb/s。Zynq-7000 SoC PS 部分 UART 接口（MIO[48:49]）供电电压为 1.8V，CY7C64225 芯片支持的电压为 3.3V 或者 5V，因此只有将串行信号（RXD/TXD）变成要求的电压才能够进行连接，该转变是通过电平转换器件（TXS0102）实现的。

2) USB-JTAG

USB-JTAG 对应于开发板的 J17 接口，支持高速传输，是以 Diligent USB 技术为基础，该模块已经集成到了 Vivado 设计套件中并且是完全支持该开发环境的。USB-JTAG 接口模块^[10]中包含 FTDI 公司制造的接口转换芯片 FT232HQ，该芯片是 48 脚的 QFP 封装模式，支持很多类型接口以及 USB 的转换操作，例如 JTAG，能够实现最大传输速率高至 480Mb/s。

3.1.2 软硬件协同设计原理

软硬件协同设计方法，也就是在研究整个系统及定义的基础上，利用软硬件两个部分同时设计并调试系统，其中包括软硬件功能的划分、协同开发和调试直至满足设计需求和任务。软硬件协同^[8]方法中，通过使用同一个工具描述想要实现的整个系统，并把任务进行划分使软硬件两个部分能并行的进行，同时充分利用软件与硬件两者之间的协

同性来完成和改善整体设计的效率的任务。使用软件和硬件协同的方法设计系统不仅能够减少系统的开发时间，而且还能得到更佳的实现结果。

完整的软硬件实现过程：首先使用软件设计程序，再利用综合器将软件编程语言转换为硬件实现中可以使用的 HDL 在硬件平台中将其实现，再将软件部分和硬件部分联合起来，实施协同仿真，即协同调试和分析整体设计性能。这种设计方式具备的优越性表现为：系统一旦有所更改或更新，只需要修改相对容易的软件描述语言就可以了。这种方法增加了系统的灵活性。普遍适用的软硬件设计过程如图 3-1 所示。

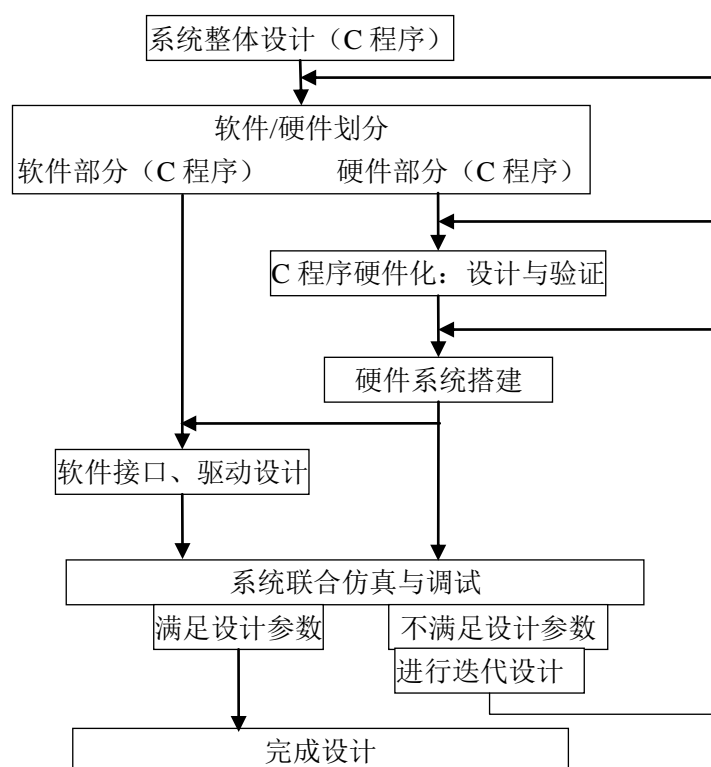


图 3-1 软硬件协同设计流程图

Fig. 3-1 Flow diagram of co-design of hardware and software

首先是根据系统的设计要求，用高级语言描述系统（这里用 C 语言实现），然后根据系统和各个设计模块的性能和资源进行软硬件的划分，即将描述系统的 C 语言分解为两部分：一部分是在软件处理器上执行的 C 程序，另一部分是将要在硬件部分执行的可以转换成硬件的 C 程序。从图 3-1 可以看出，软硬件的划分对系统的实现影响很大，如果可以做好软硬件的划分就可以减少迭代的次数，软硬件划分的一般原则：对于控制偏多或者串行性更强的任务用处理器来实现；硬件加速协处理器更适合执行一些并行性更强的任务。在硬件部分主要是完成系统的搭建，软件部分主要是编写接口、驱动程序和控制程序。将软硬件联合起来仿真和调试，假如性能符合要求就完成设计；假如不符合，就必须回到开始的步骤，例如功能划分、硬件的实现等步骤重新设计。

通过 Xilinx 公司提供的 Vivado HLS 工具能够将 C 语言编写的算法综合并转换成 Verilog 代码,能依照 C 语言算法直接创建 IP 并对设计的 IP 进行验证。这种方式不但能高效地实现设计,而且比传统使用 RTL 方式进行设计仿真时在速度方面高出好多。

3.1.3 硬件开发环境

在本课题中使用 Vivado 来实现硬件部分,各个模块单独进行验证,最终在 Vivado 中实现一个完整的系统进行音频频谱的显示。

Vivado^[40]开发环境是 Xilinx 公司新出的一套适用于可编程逻辑的开发软件,它不仅有一些能够高度集成的开发环境,还拥有新一代 IC 等级的设计工具,自带的这些资源是以共享和支撑可扩展的数据模型和常用的调试工具为基础而创建起来的。Vivado 设计套件同时也是一个以 AXI4 总线通信互联规范为基础的开放式的设计环境,设计能够根据客户的要求进行定制且实现过程适合业界的准则,不仅如此该套件还能够把各种类型的可编程逻辑器件联合起来进行扩展实现等同于 1 亿个 ASIC 门的系统设计。Vivado 套件使用层次化的可编程器件和布局管理器将执行的速度提高到了原来的 3-15 倍^[40]。

本课题采用 Vivado13.4 搭建硬件系统结构,用 Vivado SDK 实现软件的开发,最后利用软硬件协同的方式对系统进行调试。利用 Vivado 开发套件的实现过程如图 3-2 所示。

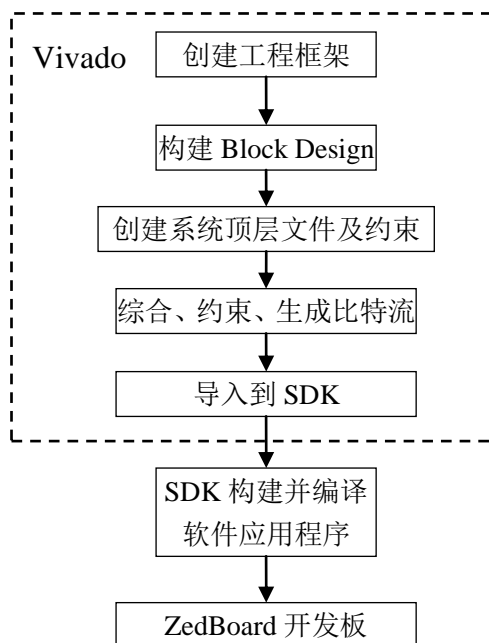


图 3-2 基于 Vivado 的设计流程图

Fig. 3-2 Flow diagram of design based on Vivado

在 Vivado 中进行硬件调试的步骤:建立 RTL 工程;在 RTL 代码中根据设计的实际情况实例化相应的调试 IP 或者在综合后添加相应的 IP 对相应的 Net 进行调试;下载比特流到目标器件进行测试分析。

3.1.4 软件开发环境

本设计中的软件部分是在 Vivado 设计套件中的 SDK 中进行完成的，SDK 能够识别使用 C 语言进行编写的代码，这部分主要是设计硬件部分所需要的接口程序，同时为硬件 IP 以及系统编写测试程序以及正常工作所必需的驱动程序。如果涉及操作系统还需要设计硬件所涉及的加速协处理器的驱动程序，这样软件才可以顺利的调度我们所设计的硬件协处理器，从而达到软硬件两部分联合设计和调试系统的目的。

本课题用 SDK 嵌入式软件开发环境编写 OLED 实现频谱显示的驱动程序和测试程序、音频控制部分所对应的驱动程序和测试程序、频谱显示系统的主测试程序。

3.2 频谱显示系统设计

本课题的目的是实现一个完整的音频频谱的显示系统，能够正确显示音频信号频谱信息。设计的系统中包含处理音频信号的 FFT IP，显示频谱信息的 OLED IP，控制音频信号的 Audio IP 等，首先对设计中包含的各个子模块分别进行验证，最终对整个系统进行验证。

通过软硬件协同这种方式设计并实现硬件系统、编写控制程序，完成软件和硬件各自的设计之后，把两个部分结合起来协同调试系统，由 PC 机随机播放一段音频信号，并由音频输出接口得到随机音频信息，在经过硬件 PL 部分的 FFT IP 处理后将频谱结果传送到 OLED 显示器进行频谱的实时显示。

完整的实时显示随机音频频谱的硬件系统结构如图 3-3 所示。

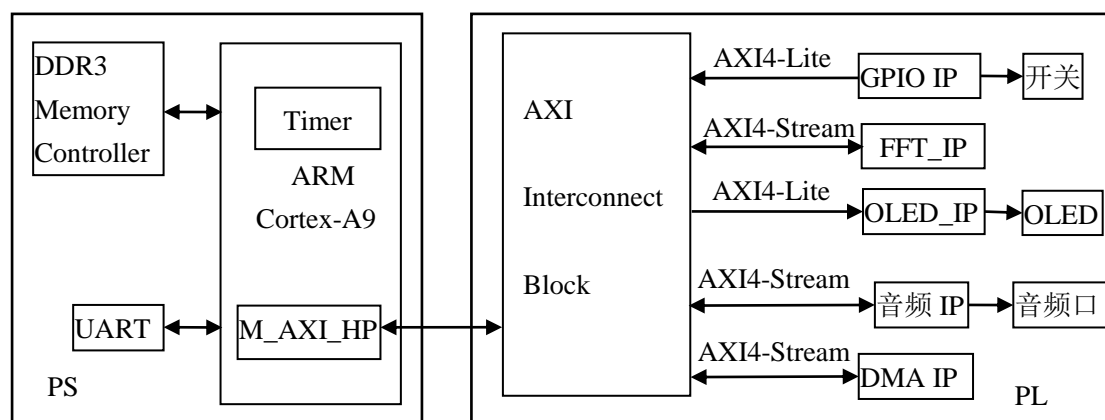


图 3-3 系统结构图

Fig. 3-3 Structure diagram of system

系统的实现过程：从 ZedBoard 开发板的音频接口获得电脑上播放的音频信号，音频信号通过高速的 AXI4-Stream 协议分别进行 FFT 处理和电脑音频信号的原样输出，这两种不同处理方式是由 AXI_GPIO_IP 采集开关的高低电平状态决定的，进行完信号的 FFT 处理之后，由处理器将获得信号的频谱信息送给由 OLED IP 控制的 OLED 显示器进行实

时的频谱显示，由于采集开关的状态以及控制 OLED 显示器的显示不需要很高的速度，因此 AXI_GPIO IP 和 OLED IP 采用低速的 AXI4-Lite 协议，这种处理方式既能实现目的又能节约硬件资源。

PL 硬件部分实现的主要功能：PL 部分通过 AXI Interconnect 模块实现 FFT IP、OLED IP、音频 IP、GPIO IP、DMA IP 等各个模块和 PS 部分的 AXI 接口相连接。

软件部分实现的功能：1) 为硬件部分的各个 IP 设计创建接口，编写 OLED IP、FFT IP、音频 Audio 的驱动程序使它们能够正确读写相关寄存器的值；2) 编写测试程序测试硬件设计的系统能否按照设计要求进行音频频谱的显示。

3.3 FFT IP 设计

传统的设计方式实现 FFT IP 需要考虑五个模块：

(1) 输入选择模块。存储单元输入级，将原始的输入数据以及蝶形单元的计算结果依据需要进行选择然后进行存储。

(2) 存储模块 RAM。在这里可以用来存放进行变换的所有输入数据、中间结果以及变换结果的最终输出。这个 RAM 是读数据与写数据的随机 RAM。

(3) 存储模块 ROM。该模块被用来存放蝶形运算所用到的各种旋转因子值，这些值可以在变换之前就计算好存放起来，方便在需要时直接访问并读取，该存储器在使用中不能写数据进去，为只读方式。

(4) 蝶形运算模块。是实现 FFT 变换的主要功能模块，运算结果存入存储单元 RAM。

(5) 时序控制模块：生成 FFT 运行中所有子模块需要的使能信号和控制信号，使所有的模块协调、有规律的运行，从而完成整个运算过程。

使用 HLS 综合工具不需要考虑这五个部分在硬件上如何实现，只需要用 C、C++、System C 等高级语言来描述 FFT 算法即可，HLS 工具负责将算法转换成 RTL 级实现，将用高级语言描述的算法中的各个模块按照一定的规则和传统的硬件实现所需模块进行一一的映射。本课题利用 HLS 工具将 C 语言编写的 FFT 算法打包生成一个 RTL 级 IP。

本课题采用基-2 流水数据流这种实现方法进行算法的设计，如图 3-4 所示是流水数据流 I/O 结构。

对于流水数据流^[45]I/O 这种实现结构，进行变换处理当前帧数据的同时可以加载下一帧处理需要用到的输入数据并输出上一帧变换处理的数据结果值，根据这个特点可以连续进行数据的输入，然后经过一定时间长度的延迟后就能获得连续的计算输出结果。

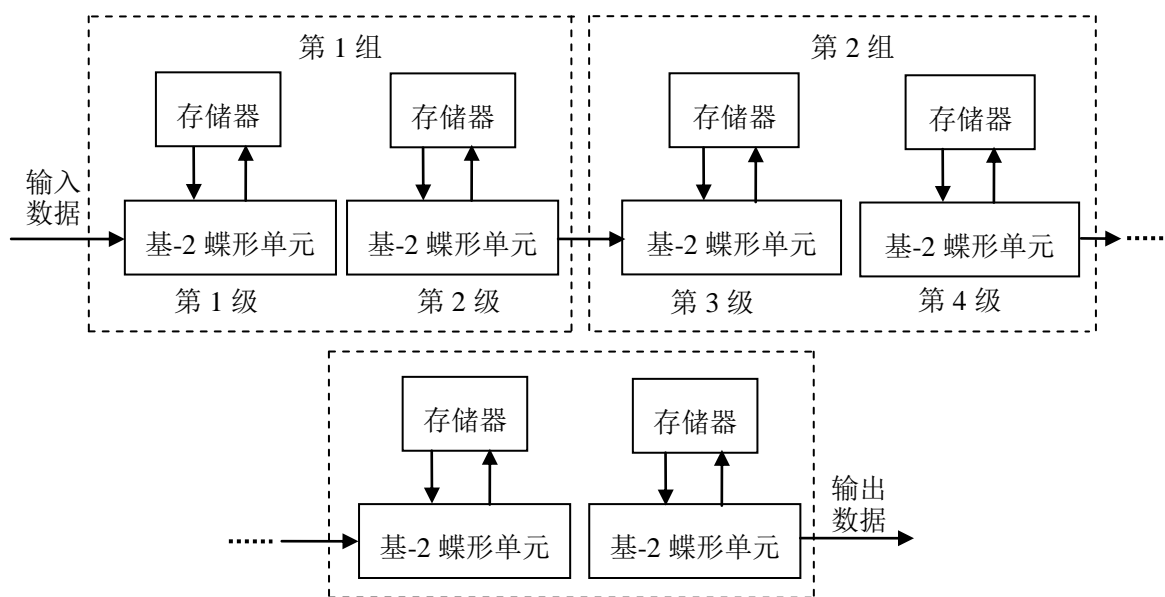


图 3-4 流水线的数据流 I/O 结构
Fig.3-4 Data flow I/O structure of pipelining

3.3.1 FFT 算法

根据 FFT 算法的基-2 时域分解方式我们知道：FFT 变换中用到三个核心的知识点即输入数据的倒序运算、 M 级递推运算和构造旋转因子。

(1) 倒序（变址）运算

当真正用到算法时，总是习惯按照人的思维也就是通常的自然排序法进行运算，所以为了计算时方便，必须把自然排序进行一个倒序处理才能方便实际的计算，自然排序和倒序排序在二进制上存在一定的规律。十进制数值增大一个单位 1 时，相应的操作是对十进制数对应的二进制表示的数值最低位执行加 1 运算，之后再把结果向左边执行进位，两种运算的结果是一样的。变址后的数与之不同，对 M 位二进制数的最高位执行加 1 操作，在这里是向次低位进位并依次类推，利用反方向进位加法的特点能够通过当前倒序数计算紧接着的下一个倒序数，进而实现变址操作。

利用反向进位加法实现倒序数的排序算法称作雷德（Rader）算法，其实现过程如图 3-5 所示。

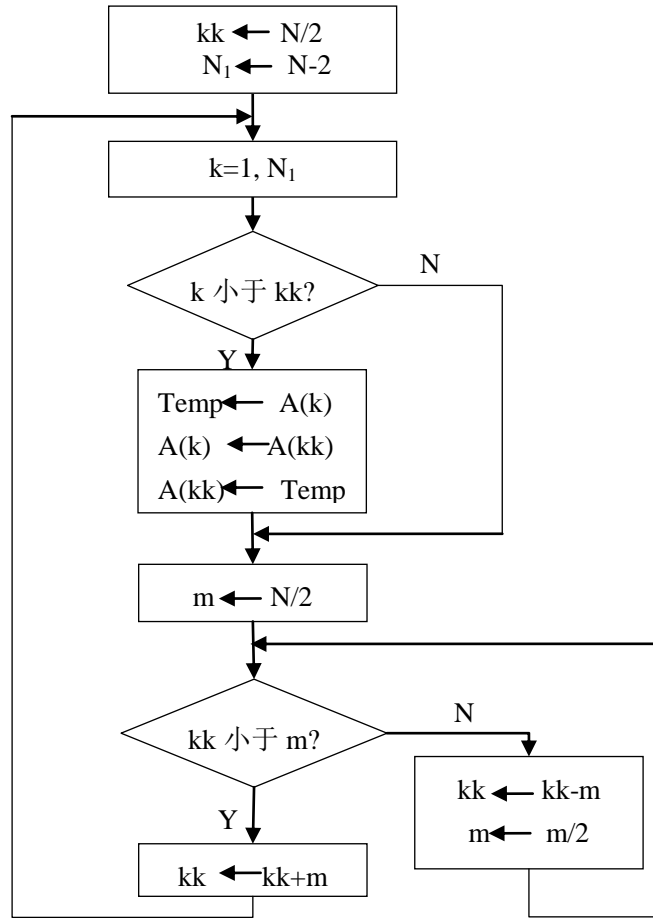


图 3-5 倒序程序框图

Fig. 3-5 Reverse program flow diagram

用 k 存放原输入序列顺序值, kk 存放当前倒序数的十进制数数值。对于 $N=2^M$, M 位二进制数最高位的十进制权值是 $N/2$, 且从左到右二进制的权值依次是 $N/4$, $N/8$, 直到最后 2, 1 结束。因此, 最高位加 1 相当于十进制运算 $kk+N/2$ 。如果最高位是 0 ($kk < N/2$), 则直接由 $kk+N/2$ 得到下一个倒序数; 如果最高位是 1, 即 ($kk \geq N/2$), 则先将最高位变成 0 (kk 赋值为 $kk-N/2$), 然后次高位加 1 ($kk+N/4$)。但次高位加 1 时, 同样判断 0、1 值, 如果为 0 ($kk < N/4$), 则直接加 1 (kk 赋值为 $kk+N/4$), 否则将次高位变成 0 (kk 赋值为 $kk-N/4$), 再判断下一位; 依此类推, 直到完成最高位加 1, 逢 2 向右进位的运算。

(2) M 级递推计算

M 级递推运算流程可以根据采用的 FFT 变换的原理得到, 其中本课题采用的基-2 时域抽取法中的递推运算流程如图 3-6 所示。

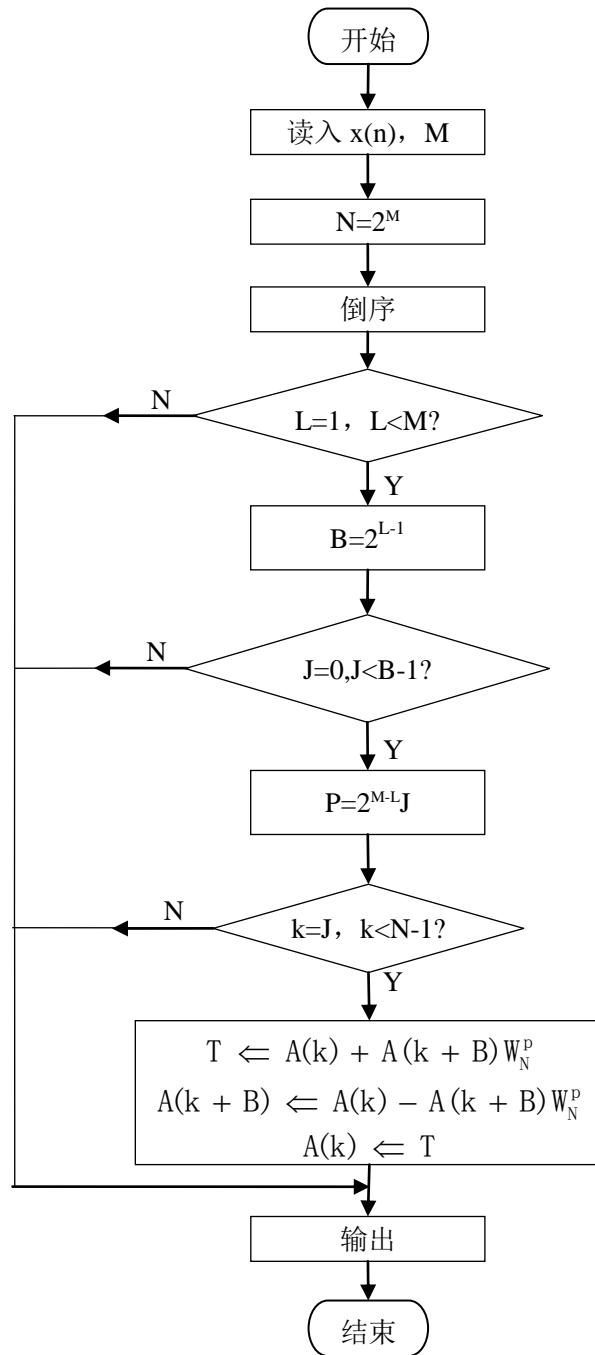


图 3-6 M 级递推运算流程图

Fig. 3-6 Flow chart of M stages recursive operation

所有的 M 级递推运算都是由外层，中间层和里层三个循环嵌套而构成的。最外面那层循环的作用是控制 M 轮设计的顺序逐步执行，中间层和里层循环共同对同一轮蝶形单元进行控制，而里层循环的作用对象是使用同一个旋转因子的蝶形计算模块，中间层的作用对象是不同类即使用不同旋转因子的蝶形单元。

(3) 构造旋转因子

本设计中采用提前计算好旋转因子系数并将它存储起来的方式，即查表的方式寻找想要的旋转因子系数。计算 1024 点的 FFT 时，为了节省时间使其在运行 C 语言 FFT 算法时可以直接调用，避免计算蝶形运算时才开始计算所需要的旋转因子导致的复杂过程。采用查表法，使用 Matlab 软件生成蝶形单元计算时需要的正弦系数和余弦系数两种旋转因子系数，并将生成的旋转因子保存到文档中。

根据 w_N 的对称性，因而只需要计算 $N/2$ 长度的 w_N 系数值就足够，旋转因子的正弦部分写入到 sin_data.dat 文件中，余弦部分写入到 cos_data.dat 文件中，生成旋转因子的 Matlab 代码如下所示：

```
N=1024;%1024 点数据
%%%%余弦旋转因子%%%
fid=fopen('cos_data.dat', 'w')
for i=0:(N/2)-1
a=cos(2*pi*i/N);
fprintf(fid, '%f, \n', a);
end
fclose(fid);
%%%正弦旋转因子%%%
fid=fopen('sin_data.dat', 'w');
for i=0:(N/2)-1
a=sin(2*pi*i/N);
fprintf(fid, '%f, \n', a);
end
fclose(fid);
```

3.3.2 FFT 的 C 模型实现

本节将使用 HLS 工具用 C 语言实现 FFT 算法，并对 FFT 算法进行优化，使性能和速度都有所提升，将 FFT 算法转换成硬件实现并创建 FFT IP，供在 Vivado 中搭建音频频谱显示系统时使用。内容主要包括创建新的工程、创建源文件、设计综合、创建仿真测试文件、运行协同仿真、对设计进行优化，优化的主要工作是将设计转换成流水线（Pipeline）结构、添加 dataflow 命令、定义数据位宽等。

FFT IP 的创建是以 FFT 算法原理为依据实现的。使用高级综合工具 Vivado HLS 将 C 语言综合为硬件描述语言，C 语言设计灵活，是熟知的一门软件语言，只要熟悉 C 语言就可以完成本设计，不用专门的去学习并使用 HDL 语言实现设计。

利用 HLS 设计 FFT IP 的主要步骤如下：

(1) 新建 Vivado 工程，File ->New Project，设置工程路径、工程名，利用 Vivado HLS 工具导入实现设计好的 FFT 的 C 源代码 fft.c、头文件 fft.h、旋转因子 sin_data.dat 和 cos_data.dat，如图 3-7 所示。

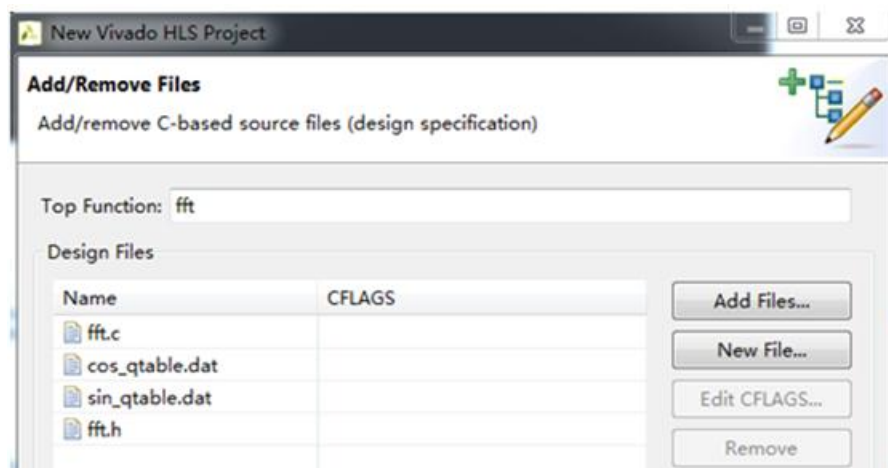


图 3-7 导入 fft 源程序

Fig. 3-7 Import the fft source program

1) fft.c 文件是指在 HLS 工具中被综合成硬件 HDL 描述语言的 FFT 算法。代码程序如下：

```
#include "fft.h"

/*****computing the cos twiddles*****/
float cos_lookup(int n){
    float cos_table[512]={
        #include "cos_data.dat "
    };
    return cos_table[n];
}

/*****computing the sin twiddles*****/
float sin_lookup(int n){
    float sin_table[512]={
        #include " sin_data.dat "
    };
    return sin_table[n];
}

/*****computing the twiddles*****/
compx twiddle_fft(int n)
{
```

```
    compx tmp;
    tmp.real=cos_lookup(n);
    tmp.imag=-sin_lookup(n);
    return tmp;
}
/*****complex multiply*****/
compx multiply(compx twiddle,compx data)
{
    compx tmp;
    float a,b,c,d,e,f,g;
    a=twiddle.real;
    b=twiddle.imag;
    c=data.real;
    d=data.imag;
    tmp.real=a*c-b*d;
    tmp.imag=a*d+b*c;
    return tmp;
}
/*****complex addition*****/
compx plus(compx a,compx b){
    compx tmp;
    tmp.real=a.real+b.real;
    tmp.imag=a.imag+b.imag;
    return tmp;
}
/*****complex subtraction*****/
compx minus(compx a,compx b){
    compx tmp;
    tmp.real=a.real-b.real;
    tmp.imag=a.imag-b.imag;
    return tmp;
}
void fft(int xin[FFT_SIZE],compx xout[FFT_SIZE])
```

```

{
    int kk;
    int k;
    int p;
    const int itp[9]={2,4,8,16,32,64,128,256,512}
    compx xout [10][FFT_SIZE];
    compx twd;
    compx tmp;
    int tmpl;
    int xin_t[FFT_SIZE];
    /*****address translation*****/
        int m;
        int tmpl;
//stage 0
copyadd: for(kk=0,k=0;k<FFT_SIZE-1;k++)
{
    if(k<kk){
        tmpl=xin_t[kk];
        xin_t[kk]=xin_t[k];
        xin_t[k]=tmpl;
    }
    m = FFT_SIZE/2;
    fft_label22:while(m<(kk+1))
    {
        kk = kk-m;
        m = m/2;
    }
    kk = kk+m;
}
//stage 1
Stage1_Loop:
AddTra_Loop:for(k=0;k<FFT_SIZE;k=k+2)
{

```

```

        xout[0][k].real=xin_t[k]+xin_t[k+1];
        xout[0][k].imag=0.0;
        xout[0][k+1].real=xin_t[k]-xin_t[k+1];
        xout[0][k+1].imag=0.0;
    }
    //stage 2
    outer_loop:
        for(p=1;p<10;p++)
            middle_loop:
                for(kk=0;kk<FFT_SIZE;kk=kk+2*itp[p-1])
                    inter_loop:for(k=0;k<itp[p-1];k++)
                        {
                            twd=twiddle_fft(k*FFT_SIZE/2*itp[p-1]);
                            tmp=multiply(twd,xout[p-1][k+kk+itp[p-1]]);
                            xout[p][k+kk]=plus(xout[p-1][k+kk],tmp);
                            xout[p][k+kk+itp[p-2]]=minus(xout[p-1][k+kk],tmp);
                        }

```

Stage0 的作用是实现变址运算，Stage1 和 Stage2 的作用是进行 $\log_2 1024=10$ 级递推运算。

2) 头文件 fft.h 的作用主要是用来声明 fft.c 程序中定义的函数、结构体类型的变量、部分宏定义以及所需要使用的系统头文件等。其代码如下：

```

#ifndef _FFT_H_
#define _FFT_H_
#define FFT_SIZE 1024
#define FFT_NUM FFT_SIZE/2
typedef struct
{
    float real;
    float imag;
}
compx;
#include <stdio.h>
#include <stdlib.h>

```



```
#include <math.h>
void fft(int xin[FFT_SIZE],compx xout[FFT_SIZE]);
#endif
```

(2) 导入测试数据 in.dat 以及验证算法的测试程序 fft_test.c，过程如图 3-8 所示。

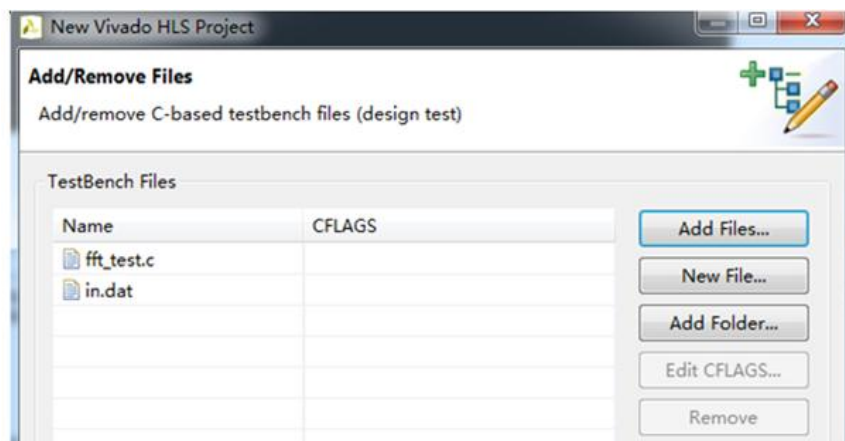


图 3-8 导入测试文件及数据

Fig. 3-8 Import test documents and data

1) fft_test.c 为测试 FFT 算法的测试程序，输入信号是 in.dat 文件中的数据，作为 FFT 运算的输入序列，FFT 运算后将运算结果打印到控制台。代码如下：

```
#include <stdio.h>
#include "fft.h"
int main(){
    int xin[FFT_SIZE];
    compx xout[FFT_SIZE];
    int i;
    FILE *fp;
    fp=fopen("in.dat","r");
    for (i=0; i<FFT_SIZE; i++){
        int tmp;
        fscanf(fp, "%d", &tmp);
        xin[i] = tmp;
    }
    fclose(fp);
    for(i=0;i<FFT_SIZE;i++){
        printf("%d ",xin[i]);
    }
}
```

```

fft(xin,xout);
printf("\n");
for(i=0;i<FFT_SIZE;i++){
    printf("i= %d      %f + %f j\n",i,xout[i].real,xout[i].imag);
}
}

```

2) in.dat 文件是输入的 1024 个整数, 该 1024 个数据是通过 Matlab 代码来产生的, 并保存为 in.dat 文件, 在测试程序中作为 FFT 运算的 1024 点输入。

生成输入数据 in.dat 文件的 Matlab 代码如下:

```

fid=fopen('in.dat', 'w');
fori=0:1023
    fprintf(fid, '%f\n', y);
end
fclose(fid);

```

(3) 进行硬件配置, 如图 3-9 所示。

时钟周期选择 10 (ns), part selection 选择为: ZedBoard, 芯片型号为 xc7z020clg484。

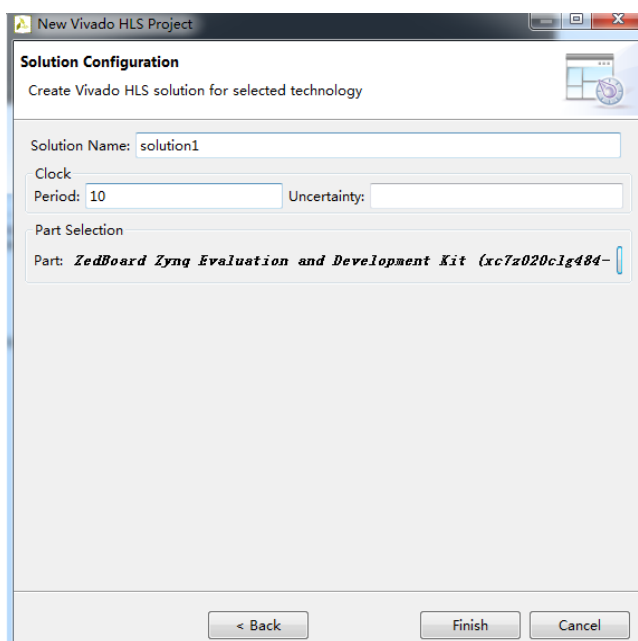


图 3-9 方案的硬件配置

Fig. 3-9 The hardware configuration of scheme

(4) 将 FFT 算法进行综合, 将其转换成 RTL 描述。

1) HLS 主界面主菜单下选择 Solution Synthesis→Active Solution 或者在工具栏内单击“综合”按钮开始综合过程。在综合的过程中, Console 窗口内打印出综合过程相关的信息。如果有错误则会在控制台报错, 根据错误提示进行修改, 如果没有错误则这些打

印信息表明 FFT 算法能够被综合,并且成功生成 System C、VHDL 和 Verilog 格式的 RTL 文件。

2) 图 3-10 所示是综合 FFT 算法后的关于延迟和吞吐量的信息,可以看出 1024 点的 FFT 占用的延迟和吞吐量信息以及各自的最小最大值,其中延迟的变化范围是 8725 到 10771,吞吐量变化范围是 8726 到 10772。

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
default	10.00	8.09	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
8725	10771	8726	10772	none

图 3-10 FFT 运算占用的延迟和吞吐量信息

Fig. 3-10 Information of delay and throughput of running the FFT operation

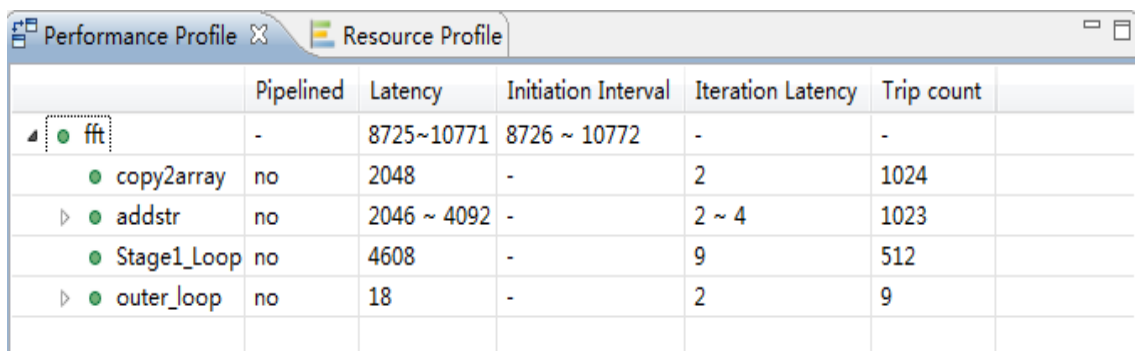
3) 图 3-11 所示是该设计所占用的硬件资源情况,使用的 BRAM_18K 为 69 个, DSP48E 为 21 个, FF 为 3020 个, LUT 为 5319 个。

Utilization Estimates				
Summary				
Name	BRAM_18K	DSP48E	FF	LUT
Expression	-	1	0	759
FIFO	-	-	-	-
Instance	-	20	2072	3952
Memory	69	-	0	0
Multiplexer	-	-	-	608
Register	-	-	948	-
ShiftMemory	-	-	-	-
Total	69	21	3020	5319
Available	280	220	106400	53200
Utilization (%)	24	9	2	9

图 3-11 FFT 设计所占用的各类资源

Fig. 3-11 All kinds of resources occupied of the FFT design

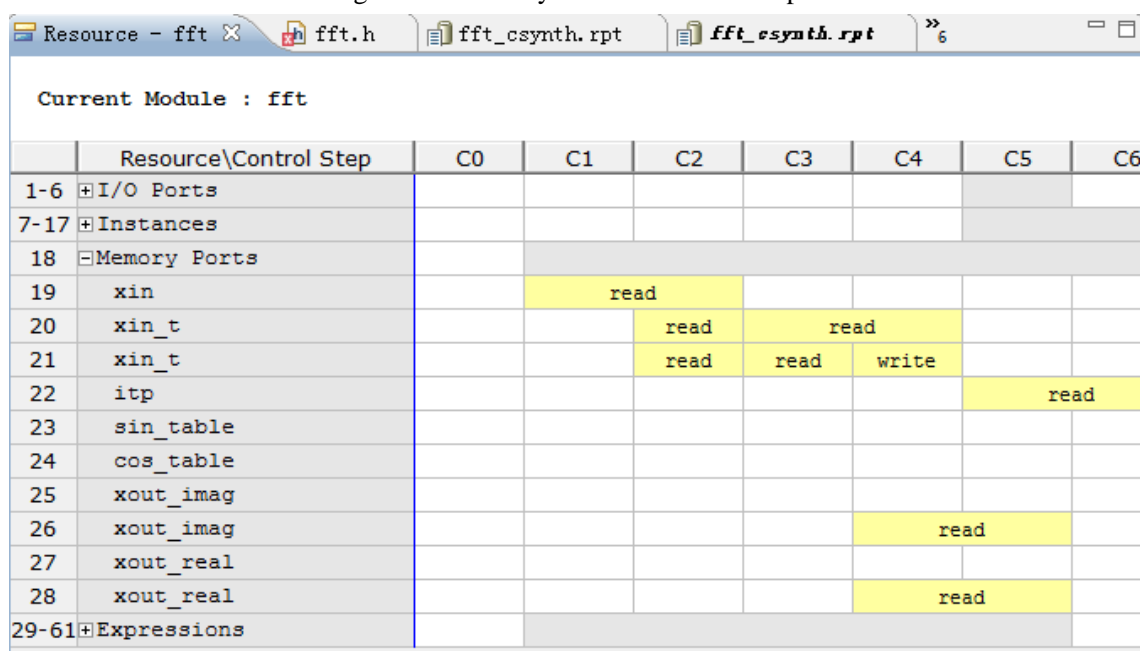
4) 查看性能分析报告,如图 3-12 所示和 3-13 所示,可以看出程序没有实现优化操作,比如 Pipelined 栏下面的“no”表示循环没有实现流水线操作,访问存储器没有执行优化操作。



	Pipelined	Latency	Initiation Interval	Iteration Latency	Trip count
fft	-	8725~10771	8726 ~ 10772	-	-
copy2array	no	2048	-	2	1024
addstr	no	2046 ~ 4092	-	2 ~ 4	1023
Stage1_Loop	no	4608	-	9	512
outer_loop	no	18	-	2	9

图 3-12 各部分占用的延迟信息

Fig. 3-12 The delay information of each part



Current Module : fft		C0	C1	C2	C3	C4	C5	C6
1-6	+I/O Ports							
7-17	+Instances							
18	-Memory Ports							
19	xin		read					
20	xin_t			read	read			
21	xin_t			read	read	write		
22	itp						read	
23	sin_table							
24	cos_table							
25	xout_imag							
26	xout_imag					read		
27	xout_real							
28	xout_real					read		
29-61	+Expressions							

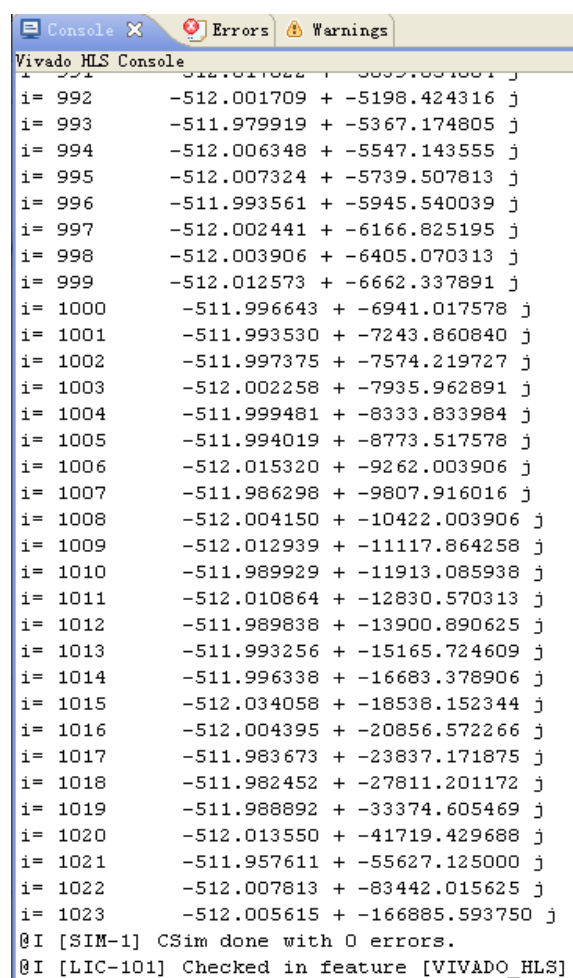
图 3-13 FFT 的存储器资源消耗

Fig. 3-13 Consumption of the memory resource of FFT

(4) 运行协同仿真，主要步骤有：

- 1) 在 HLS 界面下，选择 Solution ->Run C/RTL Co-simulation” 命令执行；
- 2) 出现 Warning 对话框，单击 OK；
- 3) 出现 C/RTL Co-simulation 对话框界面，不做任何更改，单击 OK；
- 4) 开始运行 RTL 协同仿真。

在运行协同仿真的过程中，Console（控制台）窗口内给出仿真过程中相关的信息。如图 3-14 所示的是 0 到 1023 共 1024 点数据经过 FFT 运算后在控制台打印的输出结果，这些打印信息来自 fft_test.c 文件，这里截取的是 992~1023 区间的计算结果。



```

Vivado HLS Console
i= 992      -512.001709 + -5198.424316 j
i= 993      -511.979919 + -5367.174805 j
i= 994      -512.006348 + -5547.143555 j
i= 995      -512.007324 + -5739.507813 j
i= 996      -511.993561 + -5945.540039 j
i= 997      -512.002441 + -6166.825195 j
i= 998      -512.003906 + -6405.070313 j
i= 999      -512.012573 + -6662.337891 j
i= 1000     -511.996643 + -6941.017578 j
i= 1001     -511.993530 + -7243.860840 j
i= 1002     -511.997375 + -7574.219727 j
i= 1003     -512.002258 + -7935.962891 j
i= 1004     -511.999481 + -8333.833984 j
i= 1005     -511.994019 + -8773.517578 j
i= 1006     -512.015320 + -9262.003906 j
i= 1007     -511.986298 + -9807.916016 j
i= 1008     -512.004150 + -10422.003906 j
i= 1009     -512.012939 + -11117.864258 j
i= 1010     -511.989929 + -11913.085938 j
i= 1011     -512.010864 + -12830.570313 j
i= 1012     -511.989838 + -13900.890625 j
i= 1013     -511.993256 + -15165.724609 j
i= 1014     -511.996338 + -16683.378906 j
i= 1015     -512.034058 + -18538.152344 j
i= 1016     -512.004395 + -20856.572266 j
i= 1017     -511.983673 + -23837.171875 j
i= 1018     -511.982452 + -27811.201172 j
i= 1019     -511.988892 + -33374.605469 j
i= 1020     -512.013550 + -41719.429688 j
i= 1021     -511.957611 + -55627.125000 j
i= 1022     -512.007813 + -83442.015625 j
i= 1023     -512.005615 + -166885.593750 j
@I [SIM-1] CSim done with 0 errors.
@I [LIC-101] Checked in feature [VIVADO_HLS]
    
```

图 3-14 打印出的测试信息

Fig. 3-14 Test information be printed out

(5) 利用 Matlab 软件验证 FFT 代码执行结果是否正确。

为了验证设计的 FFT 算法的正确性，将在 HLS 中进行协同仿真后在控制台输出的结果与在 Matlab 中调用 fft 函数处理的结果进行对比，如果两者结果完全一致，则表明自己设计的 FFT 算法是正确的。

验证首先要保证具有相同的输入数据，在此使用 Matlab 创建输入数据与 in.dat 完全相同的数据。在 Matlab 中调用其自带的 fft 函数，将与 in.dat 文件相同的输入数据输入到 fft 函数进行快速傅里叶计算，将输出的结果显示如图 3-15 所示。

```

x=0:1023; y=fft(x); z=y(993:1024); z.'
ans =
 1.0e+05 *
-0.0051 - 0.0520i
-0.0051 - 0.0537i
-0.0051 - 0.0555i
-0.0051 - 0.0574i
-0.0051 - 0.0595i
-0.0051 - 0.0617i
-0.0051 - 0.0641i
-0.0051 - 0.0666i
-0.0051 - 0.0694i
-0.0051 - 0.0724i
-0.0051 - 0.0757i
-0.0051 - 0.0794i
-0.0051 - 0.0833i
-0.0051 - 0.0877i
-0.0051 - 0.0926i
-0.0051 - 0.0981i
-0.0051 - 0.1042i
-0.0051 - 0.1112i
-0.0051 - 0.1191i
-0.0051 - 0.1283i
-0.0051 - 0.1390i
-0.0051 - 0.1517i
-0.0051 - 0.1668i
-0.0051 - 0.1854i
-0.0051 - 0.2086i
-0.0051 - 0.2384i
-0.0051 - 0.2781i
-0.0051 - 0.3337i
-0.0051 - 0.4172i
-0.0051 - 0.5563i
-0.0051 - 0.8344i
-0.0051 - 1.6689i

```

图 3-15 Matlab 软件调用 fft 函数计算的部分结果值

Fig. 3-15 Someresults calculated by fft function of Matlab software

因为计算结果数值比较大，因而同时提取了一个乘数因子即 10^5 ，计算的是 993 到 1024 区间的结果来和图 3-14 进行对比，因为精确度不一样，会有较小的误差，但整体对比可以证明算法设计是正确的。

3.3.3 HLS 对 FFT 进行优化

创建新的 solution，给 fft.c 文件添加优化命令，添加过程具体如下：

- (1) 在 Vivado HLS 主界面菜单下，选择 Project ->New Solution;
- (2) 出现 Solution Configuration 对话框，在该界面中选择 Copy existing directive from solution 前面的复选框，并且在其右侧得到下拉框中选择 Solution1，然后点击 Finish;
- (3) 打开 fft.c 文件，在其右侧窗口中单击 Directive 标签，选择需要添加优化命令的地方，例如下面即将添加的 copy2add，单击右键出现浮动窗口，在浮动窗口中选择 Insert Directive.....;
- (4) 在出现的 Vivado HLS Directive Editor 对话框界面的 Directive 一栏中选择需要的详细优化命令；
- (5) 具体的优化命令从三个方面考虑：粗优化、精优化、总线化，实现算法从串行运算到并行运算的转换，从而提高对数据的处理速度。

1) 粗优化命令。用上面介绍的方法在 copy2add、stage1_loop 位置处添加 pipeline 流水线优化命令，该命令的作用是使函数执行方式由串行执行变为全并行执行，对顶层函数 fft 添加 dataflow 数据流粗粒度优化操作命令，如图 3-16 中高亮所示。

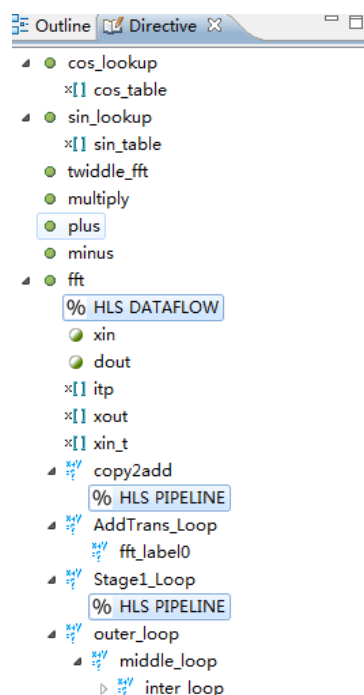


图 3-16 添加粗优化命令的函数

Fig. 3-16 Functions added the coarse optimization commands

2) 精优化命令。根据要处理数据的类型和位宽，进行优化，降低资源消耗。本次设计中输入数据为 16 位音频信息，因此可以将输入数据类型设置为 int16：即 16 位宽度的无符号整型数据，程序中的数据类型根据实际情况也设置成相应的类型。对数据类型添加的优化部分代码如图 3-17 所示。

```
void fft(int16 xin[FFT_SIZE], compx *dout)
{
    //,compx dout[FFT_SIZE]
    int16 kk;
    int16 k;
    int16 i;
    int16 p;
    const int16 itp[9]={2,4,8,16,32,64,128,256,512};
    compx xout[10][FFT_SIZE];
    compx twd;
    compx tmp;
    compx *tmp1;
    int16 xin_t[FFT_SIZE];
}
```

图 3-17 部分精优化代码

Fig. 3-17 Part of refined optimization codes

3) 顶层函数的端口总线设置。为了提高对大量数据的处理速度，采用 AXI4 Stream 的方式来进行数据的处理，即 CPU 通过使用 streaming 方式直接和 IP 通信。

关于端口的配置，如果没有为端口指定接口类型，接口的默认实现如图 3-18 所示，其中红色为不支持的接口，绿色为支持的接口。由图 3-18 可以看出 AXI4-Stream 的实现协议只有 ap_fifo 类型支持，因此将顶层函数的端口总线设置为 ap_fifo 类型。

Bus Interfaces															
AXI4			Argument Type	Variable			Pointer Variable			Array			Reference Variable		
Stream	Lite	Master		Pass-by-value			Pass-by-reference			Pass-by-reference			Pass-by-reference		
			Interface Type	I	IO	O	I	IO	O	I	IO	O	I	IO	O
			ap_none	D			D						D		
			ap_stable												
			ap_ack												
			ap_vld						D						D
			ap_ovld					D						D	
			ap_hs												
			ap_memory							D	D	D			
			ap_fifo												
			ap_bus												
			ap_ctrl_none												
			ap_ctrl_hs			D									
			ap_ctrl_chain												

Supported Interface

Unsupported Interface

图 3-18 数据类型和支持的端口

Fig. 3-18 The data type and the support of the port

执行端口优化操作如图 3-19 高亮部分所示，将输入变量 xin 和输出变量 dout 设置为 ap_fifo 类型。

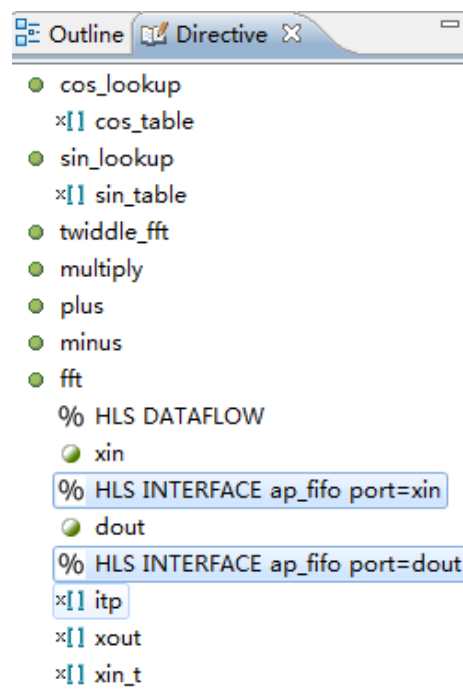


图 3-19 添加端口优化命令的函数

Fig. 3-19 Functions added the port optimization commands

3.3.4 FFT 优化前后结果对比

添加完优化命令后对比优化前后占用资源情况。

(1) 点击综合按钮，开始综合过程。综合完成后给出了优化后关于性能和资源消耗情况如图 3-20 所示。

Performance Estimates				Summary			
Timing (ns)				Name	BRAM_18K	DSP48E	FF
Summary				Expression	-	-	0
Clock	Target	Estimated	Uncertainty	FIFO	-	-	-
default	10.00	8.37	1.25	Instance	3	21	2515
Latency (clock cycles)				Memory	130	-	0
Summary				Multiplexer	-	-	-
Latency	Interval			Register	-	-	5
min	max	min	max	ShiftMemory	-	-	-
3072	5118	3073	5119	Total	133	21	2520
				Available	280	220	106400
				Utilization (%)	47	9	2

solution6 是优化之后的分析报告，solution1 是没有经过优化操作的分析报告。从图 3-21 可以看出 solution6 延迟从 solution1 的 8725 降低到了 3072；solution6 的资源占用除了 BRAM_18K 的资源比 solution1 多，FF 和 LUT 均比优化之前减低了很多，也就是用硬件资源换取速度。

(1) 将经过优化操作的 FFT 程序进行 C/RTL 联合仿真，测试生成的 RTL 代码的正确性，经过程序验证发现和优化前的仿真结果相同，表明优化命令没有更改程序的功能。

(2) 优化设计完成之后，将设计的 FFT 模型以 IP—XACT 格式导出并生成 FFT IP，该 FFT IP 可以在 Xilinx 公司的其它软件中直接使用。生成的 IP 是以.zip 后缀名显示，该 IP 可以在搭建整体测试系统时使用，只需要将该压缩文件加入到 Vivado 中就可以像使用库中自带的 IP 一样使用了，导出的 FFT IP 如图 3-22 所示。

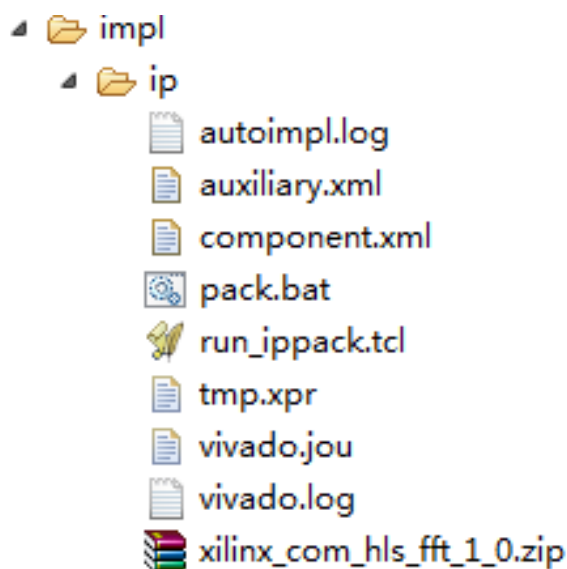


图 3-22 导出的 FFT IP

Fig. 3-22 The exported FFT IP

3.4 OLED IP 设计

OLED^[46] (Organic Light-Emitting Diode) 全称是有机发光二极管。ZedBoard 上使用的是 Inteltronic/Wisechip 公司的 OLED 显示模组 UG-2832HSWEG04。OLED 同时具备^[46]自发光、不需要光源照射、对比度较高、在厚度表现为更薄、敏感性更高以及适用的温度变化幅度比较大等特征，是平面显示领域的一个新发展。

3.4.1 OLED IP 整体设计

为了使 OLED 显示器模块能够正常的工作并显示数据，需要完成如下任务：搭建 Zynq 硬件系统、根据要求设计适合自己需求的 IP，并在软件部分即 PS 中设计 OLED 显示的驱动程序，该驱动程序可以参考 SSD1306 用户手册来进行设计实现。硬件系统是指

在硬件部分对需要用到的一些外围设备进行相应的设置以满足要求，并将这一部分当作处理器即 PS 的外设使用。

开发板 ZedBoard 上的 OLED 显示模组为 UG-2832HSWEG04。OLED 显示模组分辨率为 128*32，使用的驱动电路是由所罗门科技提供的 SSD1306 芯片。OLED 原理图如图 3-23 所示。

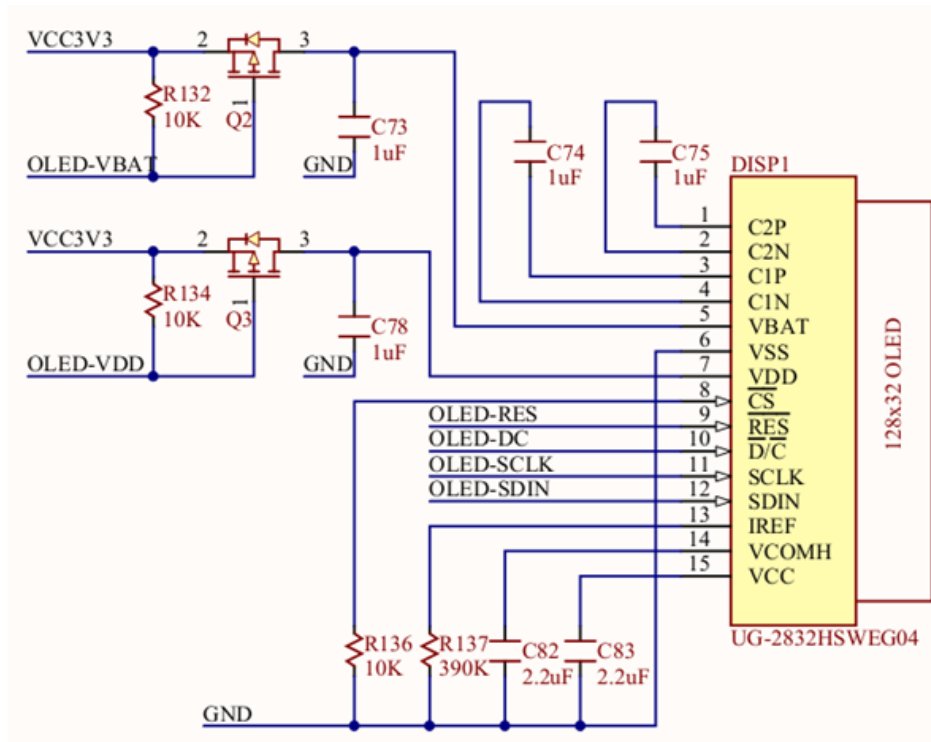


图 3-23 OLED 原理图

Fig. 3-23 Schematic diagram of OLED

ZedBoard 开发板在应用 OLED 时用到的控制方式是 SPI 模式，ZedBoard 通过 SPI 串行接口与 OLED 显示模组进行连接。这种模式用到的相关信号线与电源线如下：

CS: OLED 片选信号；

RST (RES)：硬复位 OLED；

D/C: 命令 / 数据位 (0, 表示进行命令的读写操作；1, 表示进行数据的读写)；

SCLK: 串行时钟线；

SDIN: 串行数据线；

VDD: 逻辑电路电源；

VBAT: 转换电路电源。

在 SPI 控制模式下，所有的数据长度都是 8 位宽，且是在上升沿时有效，即数据在 SCLK 的上升沿到达时从 SDIN 移入至 SSD1306，且低位在后，高位在前。为了使 OLED 显示器能够准确地显示信息，ZedBoard 利用 SPI 协议串行接口与 OLED 显示器中的显示

模组相连实现数据传输和通信，这个协议能够支持写模式，即数据和命令的传输可以利用该协议实现。其时序如下：当时钟的上升沿到达时获取数据信息，下降沿到达时改变数据信息。在 SPI 控制模式下使用写命令的时序图如图 3-24 所示。

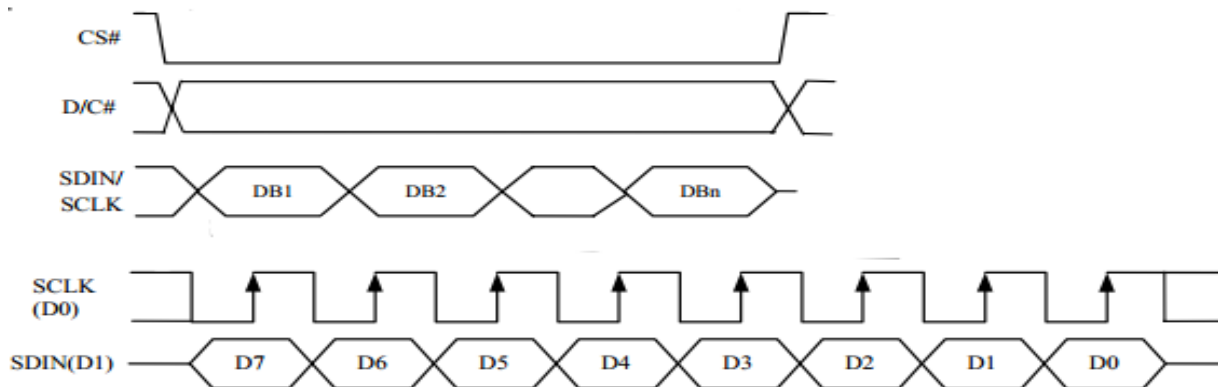


图 3-24 写操作时序

Fig. 3-24 Sequence of write operation

OLED 显示模块^[48]是连接在 ZedBoard 开发板的 PL 硬件部分即 FPGA 部分的，所以 ZedBoard 对于 OLED 显示器的控制操作就必须通过 PL 硬件间接控制才能够实现，所以需要将在硬件环境下设计的 OLED IP 所需要使用的引脚与 PS 端进行连接，再在 SDK 软件中编写 OLED 显示器驱动程序，具体的实现框图如图 3-25 所示。

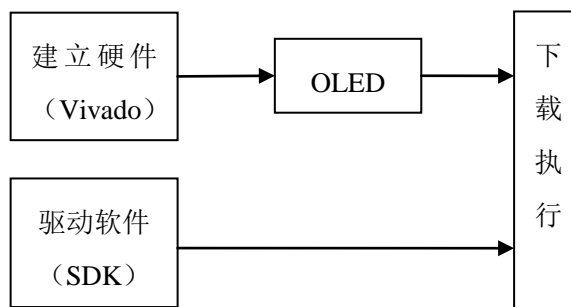


图 3-25 OLED IP 设计

Fig. 3-25 Design flow of OLED IP

ZedBoard 控制 OLED 显示器的主要方法：由于 OLED 驱动使用的 IO 接口有 6 个，包括控制信号和电源，因此设计的 OLED IP 就必须对这些控制信号以及电源信号的工作方式实现逻辑设计，并为了在开发板上正常工作，还需要对引脚添加一定的约束和定义，例如工作电压的定义，具体就是利用 AXI 总线将 OLED IP 与整个架构的 PS 端连接起来成为完整系统。

表 3-2 是表示的是 Zynq 与 OLED 显示模组之间的连线关系。

表 3-2 OLED 与 Zynq 引脚对应的连线
Tab. 3-2 Connections between OLED and Zynq pins

OLED 引脚编号	OLED 信号	Zynq-7000 引脚	功能说明
7	VDD	U12	逻辑电路电源供电
6	VSS	NC	地信号
15	VCC	NC	OLED 面板电源供电
13	IREF	U11	亮度调节参考电流
14	VCOMH	NC	COM 信号电压输出高电平
5	VBAT	NC	DC/DC 转换电路电源供电
3/4	C1P/C1N	U9	快速逆变电容正负端
1/2	C2P/C2N	NC	快速升压电容正负端
9	RES#	U10	控制器和驱动复位信号
8	CS#	AB12	片选
10	D/C#	AA12	数据命令控制
11	SCLK	AB4	串行输入时钟
12	SDIN	AB5	串行输入数据

相应的 OLED IP 系统设计如图 3-26 所示。

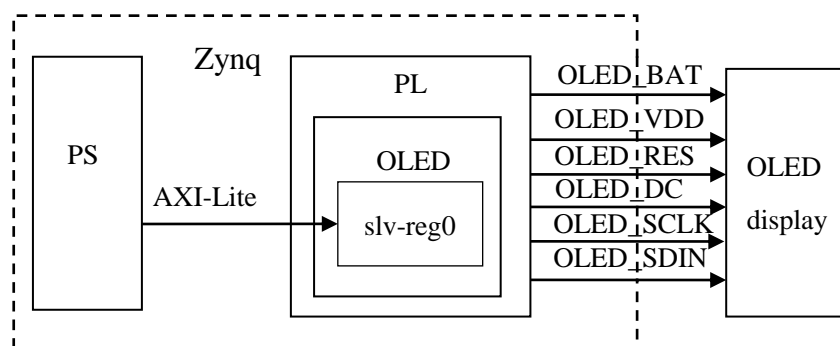


图 3-26 OLED 设计系统图

Fig. 3-26 Design diagram of OLED system

PL 部分中的 OLED 是在 Vivado 环境中自己设计的，它是在 AXI_GPIO IP 实例化产生的，因此只需要设计一个六位的寄存器来控制 OLED 的 SSD1306 控制器。

PS 控制设计的 IP 的时序和数据，这里 PL 的作用只是传输 PS 部分的控制命令。PS 部分的作用把 slv_reg0 设置为 IO 口，在控制的过程中主要是通过 slv_reg0 的最低 6 位数据进行控制。

3.4.2 OLED IP 实现

OLED IP 的实现过程就是根据 Vivado 的设计流程完成的，具体的步骤如下：

- (1) IP 使用 Vivado IDE 建立工程，并创建 OLED IP。

1) 启动 Vivado, 创建一个新的工程, 选择 RTL Project, 并将开发板选择为 ZedBoard, 完成创建过程。

2) 构建 OLED IP。在 Vivado 主界面执行 “Tools->Create and Package IP” 操作开始创建 IP, 点击 Next 继续执行。

3) 根据提示选择 AXI4 的工作方式, 在此我们选择 Create a new AXI4 peripheral, 然后单击 Next, 继续执行。

4) 在 Peripheral Details 中输入创建的 IP 的详细信息, 主要是设置名称, 在此设置为 oled_ip, 其他参数根据自己的需要修改或者是保持不变, 单击 Next 进入下一步。

5) 在 Add Interfaces 对话框中为 IP 添加 AXI4 总线接口的支持, 在此将 Interface Type 类型选为 Lite 型, 该接口专用于和元件内部的控制寄存器进行通信, 而且这种类型接口支持搭建简单和规模小的元件接口, 这种总线类型的特点是适合小批量数据处理, 简单控制的场合, 不支持大批量数据传输, 读写数据时每次仅仅是一个数据。

6) 在 Creation Peripheral 对话框中查看新建的 OLED IP 的信息, 但是此时创建的 OLED IP 不具有一个实际的功能, 需要对 OLED IP 修改用户端口和逻辑, 使其具有一个具体的功能, 选择 Edit IP 单选按钮, 点击 Finish 完成创建过程并进入编辑阶段。

7) 这时会有一个新的临时工程出现, Vivado 完成 IP 的创建及功能编辑之后会自行将其关闭并删除在系统中存在的相应文件。

8) 在 Source 窗口中双击 oled_ip_v1_0.v 顶层文件, 查看 SSD1306 用户书册编辑顶层文件。

在 “User to add ports here” 位置处添加文件

“output wire [5:0]OLED” ,

对端口进行例化,

“.OLED(OLED)” ,

完成后保存文件。

9) 双击打开 oled_ip_v1_0_S00_AXI.v 文件编辑文件。

在 “User to add ports here” 位置处添加文件

“output reg [5:0] OLED” ,

在 Add user logic here 处添加逻辑文件:

```
always@(*)
```

```
begin
```

```
    OLED<=slv_reg0[5:0];
```

```
end
```

添加完毕后保存文件。

10) 在 project manager 面板中, 可以查看创建的 OLED IP 的基本信息, 包括支持的芯片、端口、地址分配以及 IP 的模块化显示, 当这些信息没有错误之后, 就可以点击 Package IP 进行 IP 的封装操作, 此时临时界面将会自动关闭。

具有实际功能的 oled_ip 自动存放在 IP catalog 中的 AXI Peripheral 文件夹下面, 如图 3-27 所示。

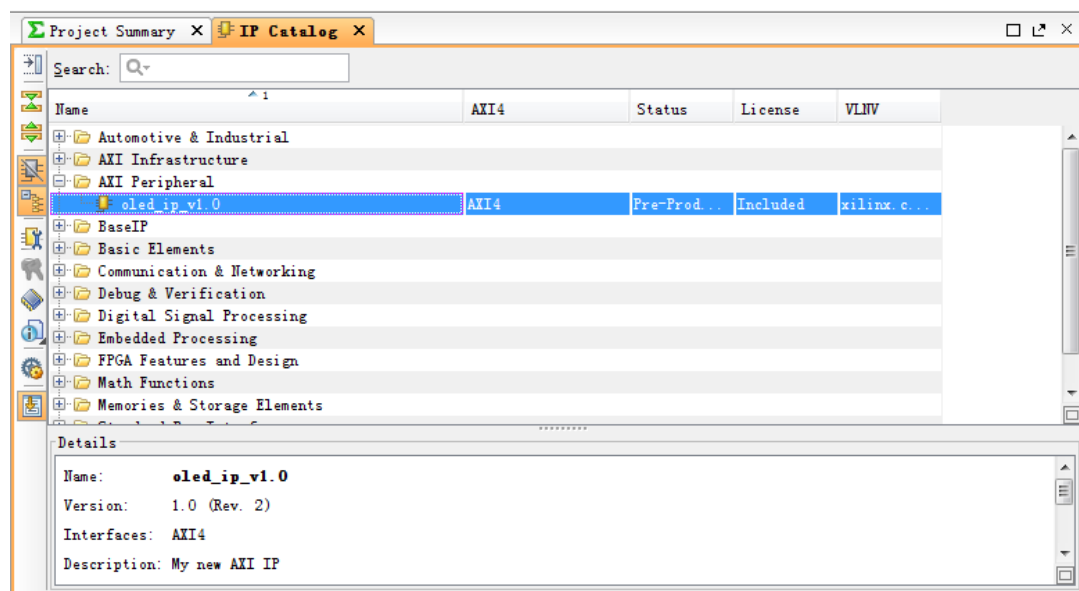


图 3-27 OLED IP 的存放位置

Fig. 3-27 OLED IP storage location

到此 OLED IP 的创建过程结束, 为了使其能够在 ZedBoard 开发板上运行, 还需要将 OLED IP 引脚分配到开发板引脚, 之后搭建系统来测试。

(2) 在 Vivado 中搭建一个测试系统, 测试之前创建的 OLED IP 是否能正常工作。主要是新建空白的 Block, 将需要的 IP 添加进来、同时添加 OLED 的约束文件 xdc, 生成硬件配置文件。

1) 在 Vivado 中新建一个空白的 Block, 命名为 Design_1, 这时会出现一个空白的画布, 在此画布中添加 IP, 搭建系统, IP 是搭建系统的最小单位。

2) 添加 IP 过程

a 首先添加 Zynq, 即 PS 端的 Zynq 7000 Processing System。IP 会以图形化的形式存在, 双击 Zynq IP 对其根据需要进行配置, 在这里我们选择 Zynq 系列的 ZedBoard 开发板, 本次设计配置相关接口:

MIO: Memory Interfaces: 勾掉 Quad SPI

IO peripheral: 选中 UART, 这个接口用于串口通信; GPIO 按照默认即可

application Processor Unit: 勾掉 Timer 0;

完成配置后单击 OK。单击 Run Block Automation 会自动将 Zynq IP 上 DDR 端口和 FIXED_IO 连接到顶层接口。

b 添加 OLED IP，该 IP 是自己创建的，不需要做任何更改。

3) 利用 IP 搭建系统进行连线。Vivado 提供自动连线功能，点击 Run Connect Automation 链接将自动将系统进行连线，并自动添加一些必要的 IP 到该 Block 中。在这里主要添加进来的有 Processor System Reset IP，这是 PS 正常工作所需要的时钟信号；AXI Interconnect IP，该 IP 是用来连接 PS 部分与 PL 部分的。连线完成后对 OLED IP 设置端口连线，将鼠标放置在 oled_ip 输出端口 out[5:0] 右击选择 Make External，完成将 OLED IP 输出口与顶层文件进行连接，重新布局，最终完成的 Block 如图 3-28 所示。

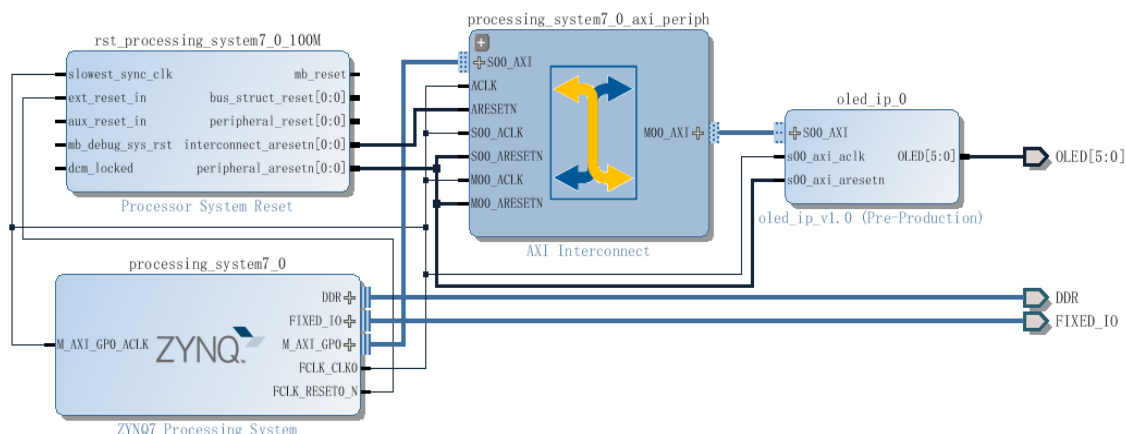


图 3-28 验证 OLED IP 的系统

Fig. 3-28 Validation system of OLED IP

4) 连线完成之后在 Address Editor 界面查看地址分配信息，如果没有自动分配成功，则需要手动实现地址的分配即选择需要分配地址的目标，右击选择 Auto Assign Address 即可完成地址的分配。

5) 检测 Block 的有效性，点击 Validate Design 项检测 Block 是否有效。如果设计正确，检测将会提示成功，否则根据错误处进行修改即可，最终检测无误后单击 OK 按钮完成检测。

6) 生成设计的 HDL 源代码和顶层文件，右击 Design_1.db 文件，选择 Generate Output Products 选项，直接点击 Generate 按钮就会生成 Block 的 HDL 源代码文件和相应端口的约束文件；类似的再次右击 Design_1.db，然后选择 Create HDL Wrapper 选项，在出现的对话框中选择 Let Vivado manage wrapper and auto-update，单击 OK 将会生成顶层文件。

7) 添加约束文件，即引脚分配和工作电压的设置。选择 source->constrs_1->Add Source, 弹出的对话框中选择“Create File...”, 命名为 oled_ip, 后缀名是.xdc, 单击 Finish 按钮完成创建过程。

查看硬件手册 SSD1306, 编写约束文件并保存。编写的约束文件必须与驱动中相关定义保持一致, 不然的话就会发生错误。

根据 OLED 原理图在这里进行如下电路与引脚分配:

```
#U11<->OLED-VBAT
```

```
#U12<->OLED-VDD
```

```
#U9<->OLED-RES
```

```
#U10<->OLED-DC
```

```
#AB12<->OLED-SCLK
```

```
#AA12<->OLED-SDIN
```

根据 SSD1306 和 ZedBoard, 进行引脚分配。

```
#OLED_DC0OLED[0]<->U10
```

```
#OLED_RES1OLED[1]<->U9
```

```
#OLED_SCLK2OLED[2]<->AB12
```

```
#OLED_SDIN3OLED[3]<->AA12
```

```
#OLED_VBAT4OLED[4]<->U11
```

```
#OLED_VDD5OLED[5]<->U12
```

9) 选择 generate bitstream 选项。根据提示执行, 生成比特流完成后必须选择 open implement design, 确保导出到软件 SDK 部分包含比特流文件, 否则导出时 bitstream 将不会被选中从而导致出错。除此之外导出的文件还包括 Block Design, 所以在导出到 SDK 之前必须将设计 Design_1.bd 打开。

在 Vivado 中的硬件部分设计就完成了, 下面利用 SDK 实现软件驱动和测试主程序等内容的设计。

3.4.3 OLED IP 测试程序设计

OLED 正常显示必须执行的操作是将显示屏进行初始化、传输数据还有命令、对显存进行设置等几个主要过程。

(1) 显示屏初始化

这个过程主要包括复位 SSD1306, 设计驱动 IC 初始化代码, 开启显示, 清零显存以及显示信息。

使 OLED 显示器正常工作需要执行下面这几个主要操作:

1) 设置 ZedBoard 与 OLED 模块相连接的 IO 接口。在这里设置与 OLED 显示模组连接的接口为输出接口。

2) 初始化 OLED 模块。实际上是对 OLED 中用到的相关寄存器执行初始化, 达到开启 OLED 显示的目的, 为正常显示字符以及数字等做好前期工作。

3) 正确显示字符和数字, 可以通过编写函数代码来实现, 即执行编写的函数代码, 把需要显示的信息传送至 OLED 来完成。

(2) 写数据和命令的实现

写数据和命令是根据 SCLK 时钟来实现的, 当该 SCLK 被设置为高电平时执行写数据, 为低电平时执行写命令, 数据和命令都是每次按照一个字节的方式进行传送的。实现传输一个字节数据的具体代码如下所示:

```
for(i=0;i<8;i++)
{Clr_OLED_SCLK;
if(data&0x80)
Set_OLED_SDIN;
else
Clr_OLED_SDIN;
Set_OLED_SCLK;
data<<=1;}
```

(3) 显存数据写入 SSD1306 存储器

实现此任务的过程是在 PS 端的内部创建一个 OLED 的 GRAM 存储器, 进行修改或者写入操作, 其实就是对 PS 端的 GRAM 进行操作, 即 SRAM, 操作完成以后就可以一次性地将 GRAM 中存放的数据写进到 SSD1306 中, 具体代码如下所示:

```
void OLED_Refresh_Gram(void){
u8 i, n;//定义无符号的 8 位整型 i, u
for(i=0;i<4;i++){
write_cmd(0xb0+i);//设置页地址
write_cmd(0x00);//列的低显示位置
write_cmd(0x10);//列的高显示位置
for (n=0;n<128;n++) write_data (OLED_GRAM[n][i]);}
}
```

(4) 编写程序测试 IP, 最终下载到 ZedBoard 上进行调试。这一步是在 SDK 开发工具中实现的。

1) 新建工程并对其命名, 选用空模板 `empty application`, 该工程用于实现 OLED 控制以及驱动程序等的设计。本次设计是用 C 语言编写软件控制代码的, 故而 `Language` 一项中选择 C 语言为目标代码, 设置完后继续后续操作。

2) 当工程新建之后添加控制和驱动的程序。执行 `OLED->src`, 右击选择 `import`, 选择 `general->file system`, 选择之前编写好的 `font.h`、`oled.h`、`oled.c`、`helloworld.c` 等文件。

对添加的文件说明如下: `oled.c` 就是 OLED 的驱动程序, 参考 SSD1306 编写; `font` 就是字库, 即控制 OLED 上的字符以何种方式进行显示才能够被 OLED 识别, 即能够保证显示的正确性; `helloworld` 是测试整个系统的主程序, 用来检验 OLED IP 的正确性。

3) 保存并编译工程, 根据提示进行修改直到编译成功。测试的运行是从 `helloworld.c` 文件的主函数 `main` 开始的。

4) 将 ZedBoard 与 PC 机进行连接, 用 USB 线连接 J14 和 J17 接口即可, 其中 ZedBoard 开发板上的 J14 接口是用来实现串口通信功能的, J17 接口被用来向开发板烧写由硬件部分生成的 bit 流文件。首次连接开发板需要等待操作系统为开发板安装需要的驱动, 或者是手动连接。

5) 打开开发板电源开始实现比特流文件的烧写, 选中 `Xilinx Tools->Program FPGA` 将硬件系统生成的比特流文件写入到 FPGA 中。如果开发板上的蓝灯由开始亮的状态变黑又再次亮起来说明比特流下载成功。

6) 配置 COM。在 SDK 主界面单击 `Run As->Run Configuration` 命令, 进行 COM 口的选择和串口波特率的设置, 此处将波特率设置为 115200。

7) 程序执行结果将在开发板 ZedBoard OLED 显示器上进行显示。选择 `Run As->Launch On Hardware (GDB)`, 执行过程中能在控制台观察到串口打印信息, 并在 OLED 中将结果显示出来。最终运行结果如图 3-29 所示。



图 3-29 OLED IP 测试结果

Fig. 3-29 The test result of OLED IP

OLED 的测试结果说明创建的 OLED IP 在开发板上能正确显示字母以及标点符号，说明设计的 OLED IP 以及驱动程序是正确的。利用 Vivado 开发工具实现了 OLED 硬件系统的搭建，并在 SDK 开发环境下采用 C 语言编写 OLED 的驱动和测试程序，成功设计 OLED IP 使其能正确显示。

3.5 音频驱动设计

ZedBoard 上总共有 4 个标准的音频接口：音频输入接口、音频输出接口、麦克风输入接口以及耳机接口。

开发板上使用 AD 公司的 CODEC 芯片 ADAU1761 与 Zynq-7000 SoC 的 IIS 接口相连。ADAU1761 是一款集成了数字音频信号处理的低功耗立体声 CDEC 芯片，支持 48KHz 立体声录音和回放、采样率的动态范围 8KHz 至 96KHz。ZedBoard 开发板上提供的音频口是由 ADAU1761 进行控制的，该芯片利用 ADI 公司提供的 SigmaStudio 工具配置为需要的音响效果、算法和增强功能优化音频。

ADAU1761 与 PL 连接，因此需要在 FPGA 上开发相应的控制器对 ADAU1761 进行控制。本次开发中在 FPGA 工程中添加 IIC 模块用于配置 ADAU1761 芯片，添加 IIS 模块用来实现音频信号的发送与接收任务。

主要特性：

- (1) SigmaDSP28-/56 位，50 个 MIPS 数字音频处理器；
- (2) 与 Sigma Studio 完全可编程图形工具；
- (3) 采样率从 8KHz 到 96KHz；
- (4) 低功耗：48KHz 时 7MW 记录，7MW 回放，消耗 1.8V；
- (5) 总共有 6 个模拟输入插脚，可配置为单端输入模式或者是差动输入模式；
- (6) 支持从麦克风输入立体声信号；
- (7) 模拟输出：2 微分立体声，2 单端音响，1 单声道耳机输出驱动；
- (8) 数字音频串行数据 I/O：音响和时分；
- (9) 输入锁相环时钟变换区间是从 8MHz 至 27MHz；
- (10) 模拟自动电平控制（ALC）；
- (11) 麦克风偏见参考电压；
- (12) 模拟和数字 I/O：1.8V 至 3.65V；
- (13) IIC 和 SPI 接口的控制；

ADAU1761 芯片与 Zynq-7000 SoC 的连接如表 3-3 所示。

表 3-3 音频接口信号功能描述

Tab. 3-3 Function descriptions of audio signal interface

信号	描述	Zynq-7000 引脚	ADAU1761 引脚
AC-ADR0	IIC Address bit 0/SPI Latch Signal	AB1	3
AC-ADR1	IIC Address bit 1/SPI Data Input	Y5	30
AC-MCLK	Master clock Input	AB2	2
AC-GPIO2	Digital Audio Bit Clock Input/out	AA6	28
AC-GPIO3	Digital Audio Left-Right Clock Input/Out	Y6	29
AC-GPIO0	Digital Audio Serial-Data DAC Input	Y8	27
AC-GPIO1	Digital Audio Serial-Data DAC Input	AA7	26
AC-SDA	IIC Serial Data interface	AB5	31
AC-SCK	IIC Serial Data interface	AB4	32

此次设计中，我们需要用到 ADAU1761 的音频输入和音频输出功能。因为音频接口在 PL 部分，PS 部分要对其进行控制只能通过 PL 部分进行间接控制，需要音频 IP 和驱动程序，音频 IP 已经存在，因此只需要根据音频 IP 和音频芯片设计音频驱动程序。

3.5.1 音频控制器介绍

图 3-30 所示是音频控制器 ADAU1761 与 ZedBoard 开发板之间引脚连线图。

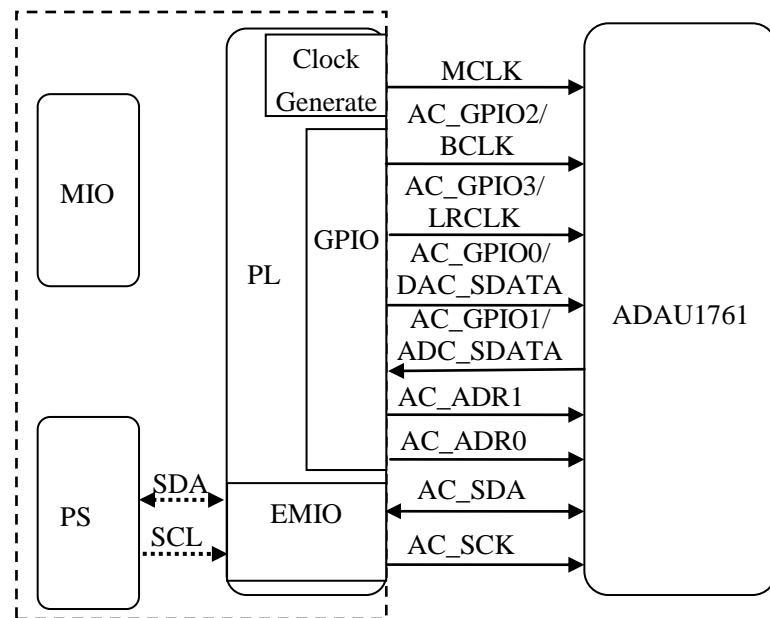


图 3-30 ADAU1761 IP 与 ZedBoard 连接图

Fig. 3-30 The connection chart between ADAU1761 IP and ZedBoard

根据图 3-30 显示的信息可以知道，利用 PS 端的 IIC 接口借助 EMIO 接口就能够将 IIC 控制总线与 ADAU1761 音频控制器进行连接，硬件 PL 端的 clock Generator 可以为 MCLK 提供时钟，其它端口则是由 GPIO 实现的。

3.5.2 音频驱动设计

由于 ZedBoard 开发板上的 ADAU1761 是挂载在 PL 上的，ADAU1761 的数据传输是通过 IIS 进行的，控制命令是通过 IIC 设计的，因此，在 PS 中控制 ADAU1761 就需要通过 IIC 和 IIS 对控制器进行控制，设计的相关驱动程序就是关于 IIC 和 IIS 接口的。

(1) IIC 总线

ADAU1761 支持 2 线串行同时能够驱动多个外设的微处理器总线。两个串行引脚数据 (SDA) 和串行时钟 (SCL) 携带 ADAU1761 和系统 IIC 主控制器之间的信息，在 IIC 模式下，ADAU1761 总是作为一个从设备，即意味着它不能启动数据传输。每个从设备靠仅有的一个地址来被识别。地址和 R/ \bar{W} 字节命令之间的关系如表 3-4 所示。

表 3-4 ADAU1761 IIC 地址与读写字节格式

Tab. 3-4 ADAU1761 IIC addresses and read/write byte format

Bit0	Bit1	Bit2	Bit3	Bit4	Bit5	Bit6	Bit7
0	1	1	1	0	ADDR1	ADDR0	R/ \bar{W}

IIC 写操作前七位是地址位，ADDR1 和 ADDR0 用于设置 IIC 地址的 Bit[5:6]两位。地址的 LSB 即 R/ \bar{W} ，被定义读或者写操作，逻辑电平 1 对应于读操作，逻辑电平 0 对应于一个写操作。

开始，IIC 总线上的每个设备处于空闲状态，并且检测 SDA 和 SCL 使其处于一个开始状态和正常的地址。当 SCL 保持高电平时，IIC 主控制器通过 SDA 从高逻辑到低逻辑的变换建立一个启动条件来启动数据传输，这表明地址或数据流跟随。

启动条件能够被连接到总线上每一个设备所响应，第一个被转换的位置是下一个 8 位 MSB，8 位是由 7 位地址位和 R/W 位构成的，当将第九位时钟脉冲位变成低数据线来识别发送地址的设备进行回应表示找到成功。第九个比特位作为一个标识位，同时总线上的其他设备从总线上撤回并返回到空闲状态。

IIC 总线部分函数介绍：

- 1) IicConfig(XPAR_XIICPS_0_DEVICE_ID)：配置 IIC 的数据结构；
- 2) XPAR_XIICPS_0_DEVICE_ID=0：IIC 设备在 PS 系统中的 ID 号，在 xparameters.h 定义为 0；
- 3) IicConfig(0)->Config=XIicPs_LookupConfig(0)：即查表，得到设备信息即 XIicPs_ConfigTable[0]={XPAR_PS7_I2C_1_DEVICE_ID, XPAR_PS7_I2C_1_BASEADDR, XPAR_PS7_I2C_1_I2C_CLK_FREQ_HZ}={0, 0xE0005000, 111111115};

4) Config Status=XIicPs_CfgInitialize(&Iic, Config, Config->BaseAddress): 初始化, 对 InstancePtr 写入从 Config 获取的相关值, 通过 XIicPs_GetOptions()函数完成初始化;

5) XIicPs_SetSclk(&Iic, IIC_SCLK_RATE): 初始化 IIC 设备的串行时钟频率. IIC_SCLK_RATE=400000//在 Audio.h 中定义。

(2) 锁相环

锁相环利用主时钟 MCLK 来产生需要的核心时钟信号, 锁相环的设置其实就是通过设置它所控制的 R 寄存器来实现的。考虑到 MCLK 的工作频率, 锁相环只有被定义成整数或分数模式才能运转, 且它适用的输入频率是从 8MHz 到 27MHz 这个区间。所有六个字节的锁相环控制寄存器必须写一个连续的写操作到控制端口。

1) 整数模式

当 MCLK 是整数 (R) 的形式作为多个锁相环才能够将其输出($1024 \times f_s$)。例如, 如果 MCLK=12.288MHz, $f_s=48\text{KHz}$, 然后需要锁相环输出= $1024 \times 48\text{KHz}=49.152\text{MHz}$, 在整数模式下, 设置 N 和 M 的值将被忽略, 则 $R=49.152\text{MHz}/12.288\text{MHz}=4$ 。

2) 分数模式

当 MCLK 是分数 ($R + (N/M)$) 的形式作为多个锁相环输出时使用分数模式。例如, 如果 MCLK=12MHz, $f_s=48\text{KHz}$, 然后需要锁相环输出= $1024 \times 48\text{KHz}=49.152\text{MHz}$, 则 $R + (N/M) = 49.152\text{MHz}/12\text{MHz} = 4 + (12/125)$ 。

(3) 音频实现中的主函数及过程描述

主要函数: AudioPllConfig()对音频编解码器的锁相环进行配置操作。

过程描述:

1) 初始化 IIC;

2) AudioWriteToReg(R0_CLOCK_CONTROL, 0x0E), 将一个字节的输入信息写到 Audio controller 的一个寄存器中, 这里的 R0_CLOCK_CONTROL=0x00 会在 (audio.h) 定义;

3) XIicPs_MasterSendPolled(), 发送数据到 FIFO, 等待被接收, 若接收失败则报错;

4) XIicPs_BusIsBusy(), 判断 IIC 状态是否忙, 如果忙就返回 true, 反之则返回 false;

5) 执行 XIicPs_MasterRecvPolled(), 表示接收数据, 如果接受成功则返回 True; 如果超时则返回错误超时信息。

(4) 音频接口

void AudioConfigureJacks()配置音频编码的各种寄存器、ADC、DAC、放大器、接受立体声输入线和推动立体声输出线, 部分函数如下:

VoidAudioConfigureJacks ()

```
{AudioWriteToReg (R4_RECORD_MIXER_LEFT_CONTROL_0 , 0x01); //enable
//mixer 1
```

```
AudioWriteToReg (R5_RECORD_MIXER_LEFT_CONTROL_1 0x07); //unmute //Left
//channel of line in into mxr 1 and set gain to 6 db
```

```
.....
```

```
AudioWriteToReg (R65_CLOCK_ENABLE_0, 0x7F); //Enable clocks
```

```
AudioWriteToReg (R66_CLOCK_ENABLE_1, 0x03); //Enable rest of clocks}
```

(5) void LineinLineoutConfig ()配置输入和输出接口、ADC、DAC、以及耳机输出方式，部分函数如下：

```
AudioWriteToReg (R17_CONVERTER_CONTROL_0, 0x06); //96 KHz
```

```
AudioWriteToReg (R64_SERIAL_PORT_SAMPLING_RATE, 0x06); //96 KHz
```

```
AudioWriteToReg (R19_ADC_CONTROL, 0x13);
```

```
.....
```

```
AudioWriteToReg
(R32_PLAYBACK_LINE_OUTPUT_RIGHT_VOLUME_CONTROL, 0xE6); //0 dB
}
```

(6) Xil_in32(), Xil_out32(): 这里表示的是 32 位宽的支持读写数据的寄存器，主要功能是把 IIS 上的数据利用寄存器读入到变量里面去，也可以把计算的输出结果值传送到寄存器。

3.5.3 设计测试程序

测试原理：采用软硬件协同设计的方法，通过电脑机产生的音频信号输入到 ZedBoard 开发板的音频输入接口，直接通过 DMA 的方式进入 ADAU1761 进行处理，声音信息通过开发板输出到音频输出接口。

音频驱动的具体测试过程：用音频线将开发板的音频输入接口和电脑的音频输出口相连，首先配置音频的 IIC、PLL 以及接口，根据开关 SW0 的不同高低电平状态进而执行不同操作，当 SW0 为高电平时，输出静音信号；当 SW0 位低电平状态时，输出原始的音频信号。处理信号的程序为：

```
u32DataL = Xil_In32(I2S_DATA_RX_L_REG); //采集左声道的信号
```

```
u32DataR = Xil_In32(I2S_DATA_RX_R_REG); //采集右声道的信号
```

```
sw_check = Xil_In32(XPAR_AXI_GPIO_0_BASEADDR+8); //检测开关的位置
```

```
if(sw_check == 0x03) // SW1=1 SW0=1 进入处理函数
```

```
{Sample_L = (u32DataL = 0);
```

```
Sample_R = (u32DataR = 0);
```



```

u32DataL = Sample_L;
u32DataR = Sample_R ;}
Xil_Out32(I2S_DATA_TX_L_REG, u32DataL); //将处理后的左声道信号输出到左声
//道输出寄存器
Xil_Out32(I2S_DATA_TX_R_REG, u32DataR); //将处理后的右声道信号输出到右声
//道的输出寄存器

```

音频 IP 硬件测试流程图如图 3-31 所示。

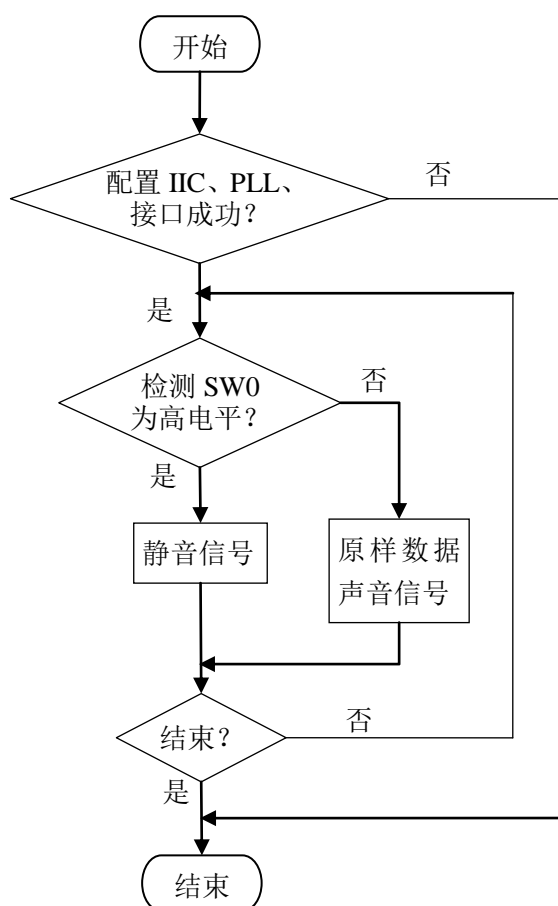


图 3-31 音频 IP 测试流程图

Fig. 3-31 The test flow chart of Audio IP

用 DMA 传输数据流程图如图 3-32 所示。

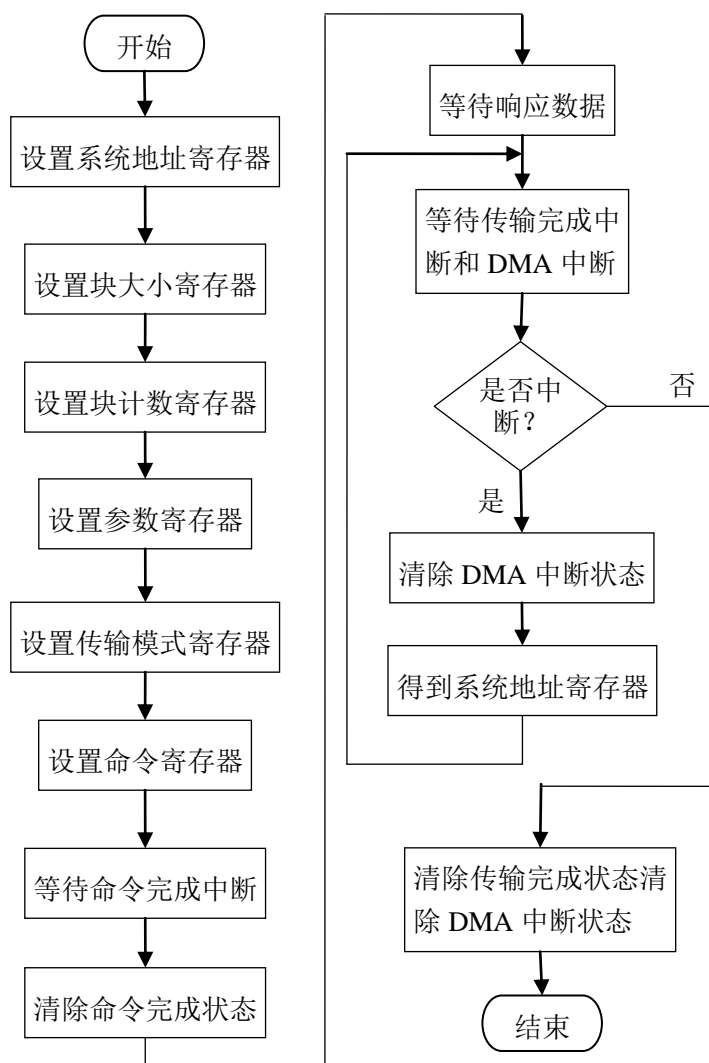


图 3-32 用 DMA 传输数据的流程图

Fig. 3-32 The flow chart of data transmission by using DMA

测试结果：当 SW0 的开关处于高电平时，从耳机中可以听到输出的声音为经过处理的音频信号，在测试试验中，输入原音频信号是带有高斯噪声的信号，经过处理之后声音信号变为静音信号；当 SW0 的开关处于低电平时，耳机中输出的信号为原始的含有噪声的音频信号，表明音频驱动能够正常工作。

3.6 本章小结

本章主要介绍了 Zynq 设计平台与基于 Vivado 的软硬件协同方法的具体设计原理，并对软硬件各自的开发环境和作用进行了简单的介绍。给出了整体频谱显示系统设计结构图，并介绍各个子模块的作用，根据整体设计要求分别设计了 FFT IP、OLED IP，并对其进行验证，同时根据音频芯片设计音频驱动程序并对设计的驱动进行验证。

4 频谱显示系统实现

整体系统的实现是在 Vivado 环境中完成的,系统测试是以软硬件协同设计的思想来进行的。整个测试过程分为硬件部分和软件部分,该过程使用 Vivado 集成开发环境创建硬件系统,使用软件开发工具 SDK 创建应用程序验证硬件功能,编写频谱显示系统的驱动程序以及测试程序,然后将硬件系统生成的比特流文件和软件系统编译的可执行程序烧写至开发板,利用软件编写的程序进行运行方式的控制,并且将音频频谱信息在 OLED 显示屏上进行显示。

测试过程共进行两次:第一次利用 HLS FFT IP 设计频谱显示系统对特定频率的正弦信号进行处理;第二次采用对比验证的方法,让两个测试系统中只有对音频信号处理的 FFT IP 不同,一个采用 HLS FFT IP 对音频信号进行处理并显示频谱,另一个是利用 Xilinx 公司的 FFT IP 对音频信号进行处理并显示频谱,两次实验结果进行对比,分析利用 Vivado HLS 设计的 FFT IP 和 Xilinx 公司的 FFT IP 对资源的使用情况。

4.1 硬件系统实现

在 Vivado 开发环境下实现硬件系统,利用 FFT IP、OLED IP、音频 IP 以及 Xilinx 提供的 IP 来实现完整的频谱显示系统,并通过观察在 OLED 显示屏上音频频谱是否能够正确显示来测试设计的系统。

硬件系统主要 IP 包括:

- (1) 自己创建的 IP: axi_gpio_0 IP、oled_IP_0 IP、Zed_audio_ctrl IP、Real FFT IP。
- (2) Xilinx 提供的 IP: Zynq IP、AXI_DMA IP、rst_processing_system7_0_100M IP, AXI Interconnect IP。

主要 IP 作用与配置:

Zynq IP: 主要配置 PS 部分与外设资源和时钟。其中时钟配置:整个系统的 PS 部分的时钟源由 ARM PLL 驱动,时钟频率约为 667MHz, PL 部分的时钟源由 IO PLL 驱动,时钟频率为 100MHz, Zynq IP 时钟频率配置如图 4-1 所示。

DMA IP: 配置直接内存管理的方式,它为内存和 AXI4-Stream 之间提供高带宽的直接存储器存取,其可选的分散/收集功能也从中央处理单元处理数据转移任务;

axi_gpio_0 IP: 配置一个寄存器用来存储开发板上开关的状态;

oled_IP_0 IP: 为了对 OLED 上的 SSD1306 控制器执行进行写操作,使用了一个 6 位寄存器;

Zed_audio_ctrl IP: 对 IIC 的控制方式以及 IIS 上执行音频数据的读写寄存器进行控制。

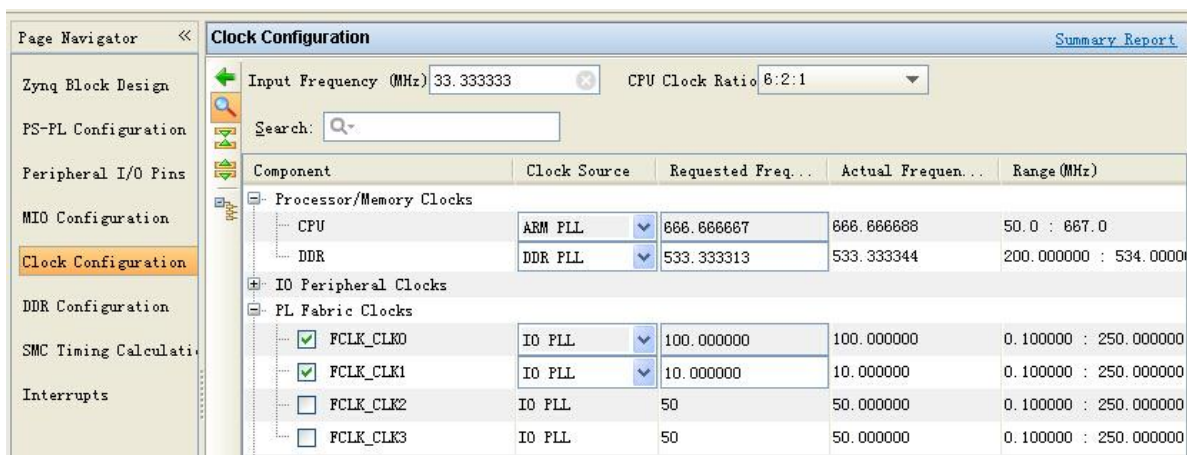


图 4-1 Zynq IP 的时钟频率配置

Fig. 4-1 The configuration of clock frequency of Zynq IP

(1) 利用 HLS 设计的 FFT IP 在 Vivado 开发环境中创建硬件系统。主要过程如下:

1) 在 Vivado 环境中创建新的工程, 其中开发板选项选择为 ZedBoard: xc7z020clg484-1。

2) 在 Vivado 工程中创建 Block design 设计, 添加 Zynq IP、fft_0 IP、AXI_DMA IP、axi_gpio_0 IP、OLED IP, Zed_audio_ctrl IP 并配置各个 IP, 进行连线操作, 最终完整的系统设计如附录 A 所示。

3) 验证设计的有效性, 如图 4-2 所示。

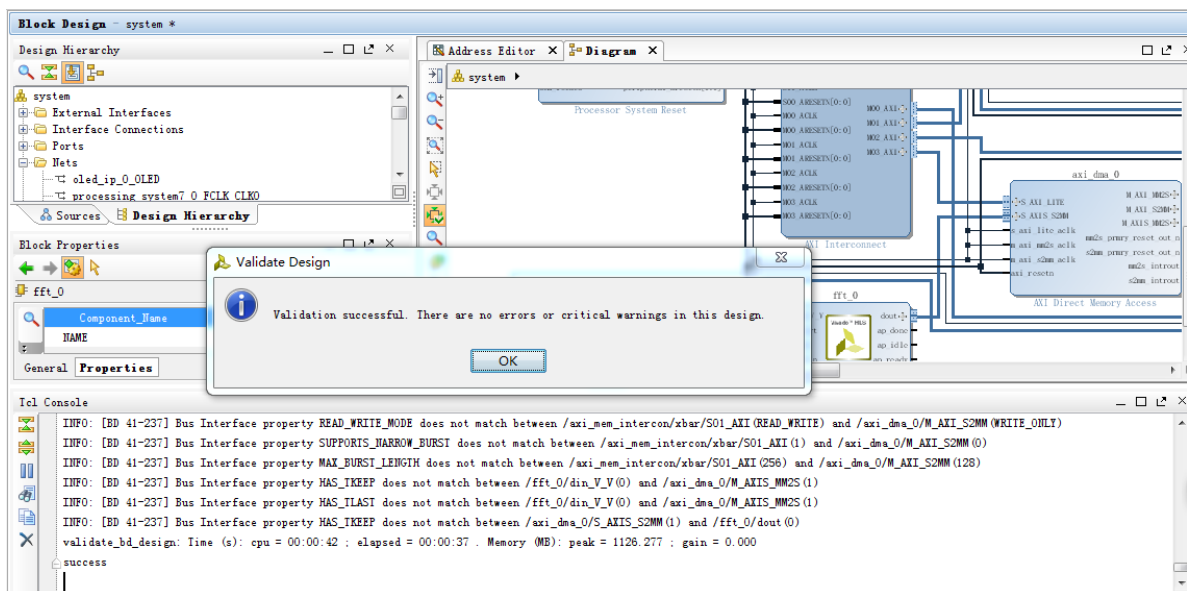


图 4-2 验证设计的有效性

Fig. 4-2 Validate Design

4) 添加 Audio 和 OLED 的约束, 主要作用是将 Audio IP 上的引脚和 OLED IP 的引脚分别与声卡芯片和 OLED 的引脚相连接以及设置引脚的电平约束。

OLED 的约束文件和 3.4 节创建 IP 时采用的约束文件一致。

Audio 的约束文件和 3.5 节部分的约束文件一致。

5) 生成 HDL 源文件和顶层文件。

6) 综合、实现并生成 bit 流文件。

7) 将板级支持包导出到 SDK 中进行开发。

(2) 利用 Xilinx 的 FFT IP 在 Vivado 中创建硬件环境。其中过程和 (1) 中的过程相同, 不同的模块是将自己设计的 fft_0 IP 替换为 Xilinx 的 Real FFT IP。

其中 Real FFT IP 由三个 IP 组成: HLS_real2xfft IP, XFFT IP、hls_xfft2real IP。

由于 Xilinx 的 FFT IP 只能处理复数, 而传过来的音频数据为实数, 因此需要在进行 FFT 处理前进行预处理能够提高数据的处理速度。并且 FFT 运算结果是一个倒序的排序输出, 为了按照自然顺序输出 FFT 运算结果, 需要在 FFT 处理后将处理结果进行排序处理。将 N 点的 FFT 变换结果转换成实数。

设计 HLS_real2xfft IP: 作用是采样 1024 点实数数据, 并将数据的实部和虚部发送至 N/2 点的 XFFT 进行后续处理。

设计 HLS_xfft2real IP: 后端 HLS 块执行倒序操作, 使输出数据以自然的顺序输出, 第一个输出对是将第 0 个和第 512 个 (纯实数) 频谱数据点分别打包成实数和虚数部分, 最后设计的 Real FFT IP 如附录 B 所示。

使用 Xilinx 提供的 FFT IP 设计的完整音频频谱显示硬件系统如附录 C 所示。

4.2 软件程序实现

两次对比试验的软件系统部分是相同的, 都是利用 SDK 软件将各个 IP 的驱动模块统一起来, 组成一个完整的软件测试系统, 进行音频频谱信息的显示, 主要过程如下:

(1) 在 SDK 中创建工程文件: 主要添加设计的源文件。源文件主要有 OLED 驱动程序、音频驱动以及 DMA 函数与 main 主函数。其中 main 主函数的作用是利用各个驱动使外设能够正常工作, 将软硬件中的各个资源统一起来协作完成目标任务。主测试程序的 main 函数的部分代码如附录 D 所示, 程序过程如图 4-3 所示。

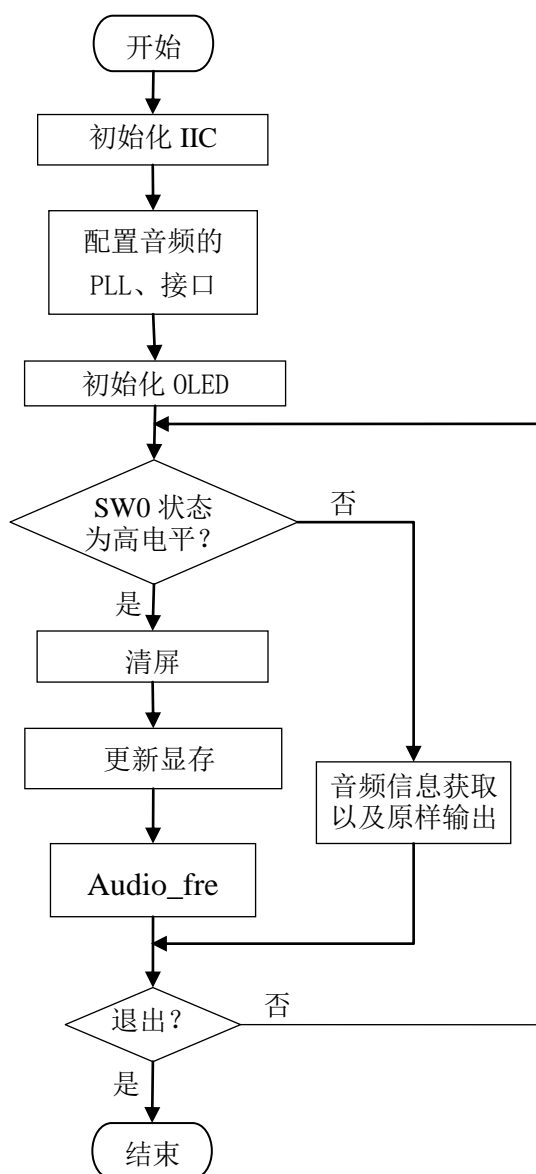


图 4-3 main 函数的执行流程图

Fig. 4-3 Execution flow chart of main function

main 函数调用音频驱动、OLED 驱动，完成音频控制器、SSD1306 控制器的初始化工作，通过读取 SW0 的开关不同的状态，执行两个不同的操作。

main 函数中调用的两个主要子函数 Audio Fre 和 Real_DMA 的流程图分别如图 4-4 和 4-5 所示。

Audio_Fre 函数的作用：当 SW0 是高电平时，从音频线上得到实时的音频数据，并以 DMA 的方式将数据传送给 PS，PS 从处理后的音频信息计算出信号的频谱信息，送给 OLED 显示器进行实时的显示并在 SDK 控制台的窗口中输出。

Real_DMA 函数的作用是：把获取的数据直接传输至 PS 部分。

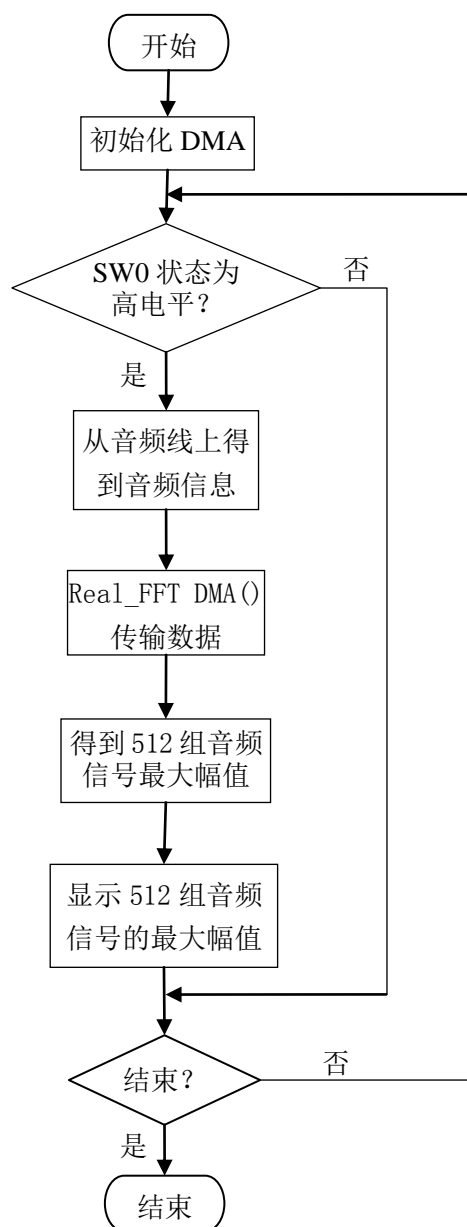


图 4-4 Audio_fre 函数的流程图

Fig. 4-4 Flow chart of Audio_fre function

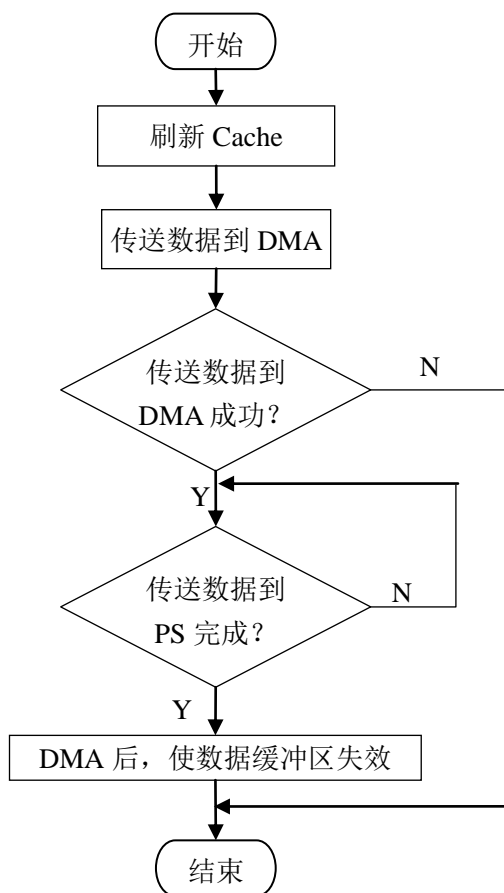


图 4-5 Real_DMA 函数的流程图

Fig. 4-5 Flow chart of Real_DMA function

(2) 函数代码编完后执行编译操作, 在此过程中会生成能在开发板上执行的后缀是.elf 的文件。

(3) 利用 USB 设备连接串口和 JTAG 接口进行串口通信和下载程序到开发板中。

(4) 选择 Xilinx tools->Program FPGA, 烧写 bit 流文件程序, 蓝灯 DONE 亮说明成功; 选择 run->run as->run as hardware 将软件程序下载到板子上。

(5) 使用音频线将电脑的音频输出口和开发板的音频输入接口相连接, 将耳机的插头与开发板的音频输出接口相连, 通过耳机使人耳直观的感受音频信号。

4.3 系统测试及结果分析

系统测试分为两个阶段: 第一阶段使用音频发生器产生特定频率的正弦音频信号, 使用利用 HLS FFT IP 实现的音频显示系统采集音频信号, 并进行处理和频谱的显示。

第二个阶段使用两个不同的 FFT IP 搭建两个频谱显示系统, 利用电脑上的播放器播放一段随机音频信号, 通过音频线输出到开发板, 并在 OLED 上显示频谱, 对比两次的结果分析自己创建的 FFT IP 以及 OLED IP 的正确性和资源占用情况。

4.3.1 测试特定频率的信号

在电脑上利用信号发生器软件 MyToneTest(版本为 1.27)产生特定频率的正弦信号，正弦信号经过音频线输入到 ZedBoard 开发板中进行 FFT 处理，之后在 OLED 显示器上显示。

产生的频率有 2000hz、5000hz、8000hz、10000hz、15000hz 和 20000hz 的正弦信号，这些信号都在人的耳朵的听觉范围之内。不同频率的正弦信号在 SDK 软件的控制台窗口中显示的结果以及 MyToneTest 软件中的设置方式如图 4-6 所示。

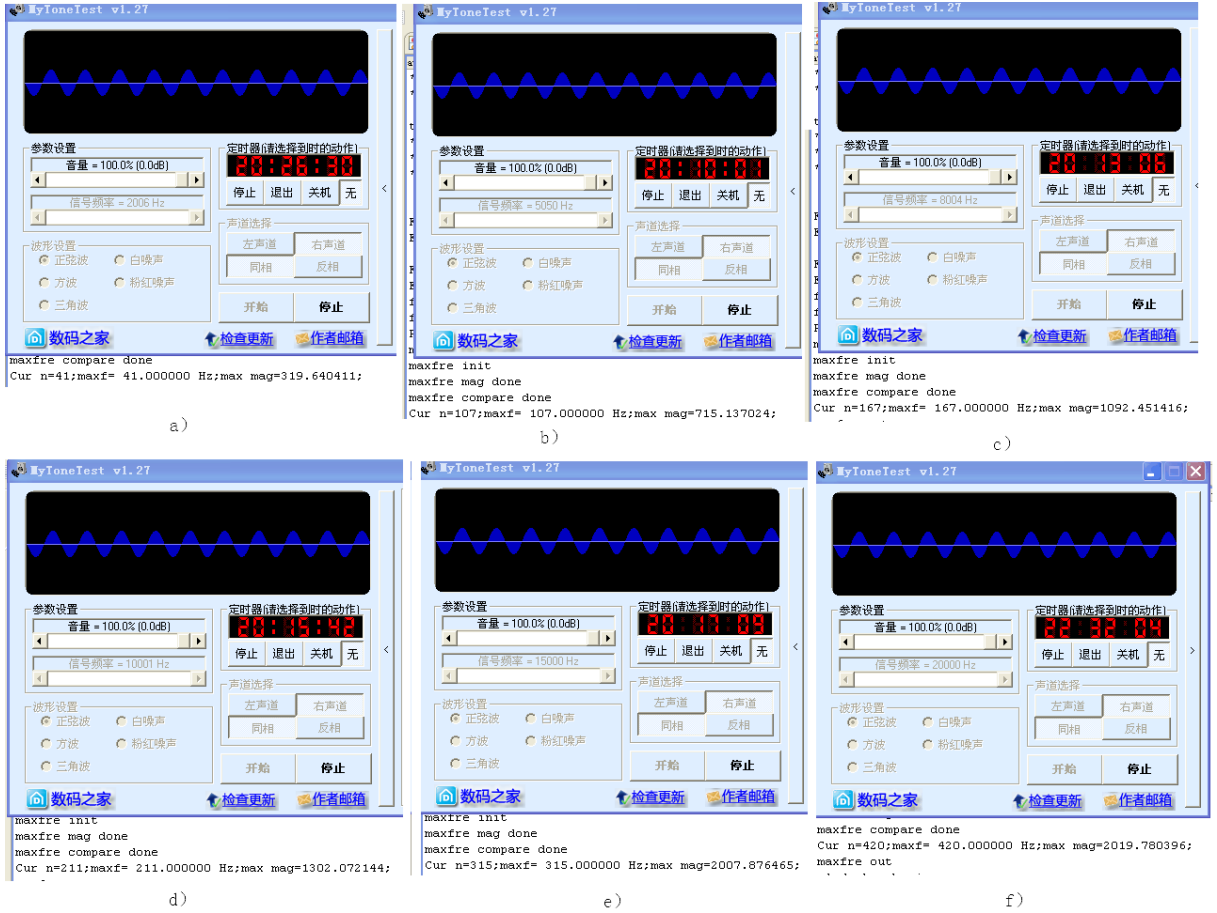


图 4-6 信号源频率和显示频率

Fig. 4-6 The frequency of signal source and display

表 4-1 为信号源的实际频率和 OLED 上显示的频率的对应表。

表 4-1 信号频率和显示频率的映射关系

Tab. 4-1 Mapping between signal source frequency and display frequency

信号源频率 (Hz)	2006	5005	8004	10000	15000	20000
显示的频率 (Hz)	41	107	167	211	315	420
比率 (信号源/显示)	48.9	46.7	47.9	47.4	47.6	47.6

这 6 个频率的信号在 OLED 显示器上显示结果如图 4-7 所示。

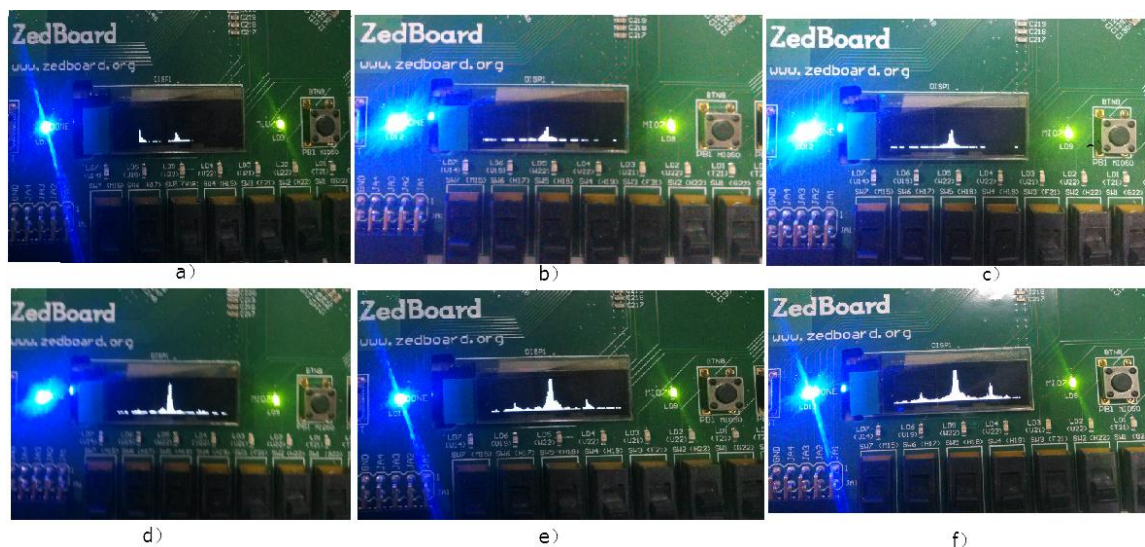


图 4-7 不同频率的正弦信号的频谱

Fig. 4-7 The spectrum of different frequency sine signal

4.3.2 测试 MP3 音频信号

本次测试使用电脑中播放的 MP3 信号作为使用不同 FFT IP 的两个系统的输入音频信号。

(1) 使用 HLS 生成 IP 的频谱显示系统：当 SW0 开关拨到高电平的位置时，可以看到 OLED 显示屏上显示当前播放音频信号的频谱，如图 4-8 所示。

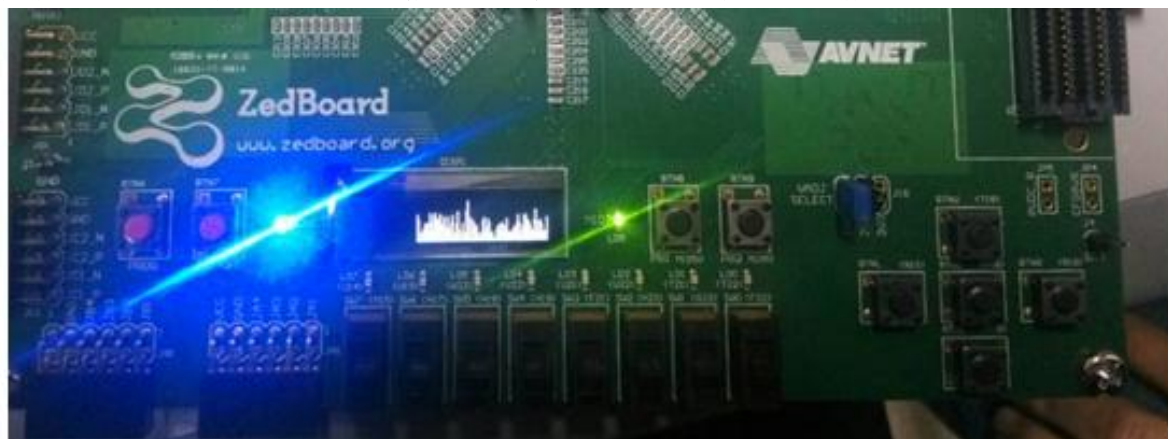


图 4-8 利用 HLS 创建的 FFT IP 的音频频谱显示

Fig. 4-8 The audio spectrum display by using HLS FFT IP

当 SW0 开关拨到低电平的位置时，可以从开发板子上所接的耳机中听到音频信号的声音。让这段音频单独从电脑的耳机输出口输出，对比两者的音频效果，是一致的。表明 ZedBoard 开发板不失真地采样电脑输出的音频信号。

(2) 使用 Xilinx FFT IP 生成的系统：此部分可以使用 (1) 的软件部分进行测试。当 SW0 开关处于高电平状态时，即对音频信号进行 FFT 处理，可以看到频谱信息在 OLED

上某一时刻的显示结果如图 4-9 所示；当 SW0 处于低电平时，从耳机中能够听到（1）中一样音频效果的音乐。

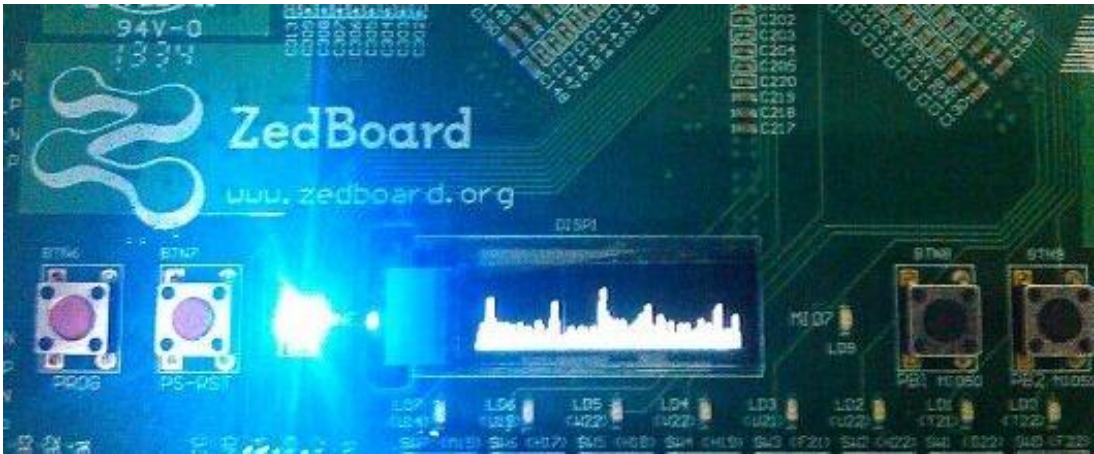


图 4-9 使用 XFFT IP 的音频频谱显示

Fig. 4-9 The audio spectrum display by using XFFT IP

4.3.3 结果分析

由图 4-6 可以看出，正弦信号的频率在 OLED 显示器的中间显示为单一的频率信号，由于在软件中设置频率在 OLED 中显示方式是：在最中间显示最大频率的音频信号，其它频率在两边依次均匀分布，因此正弦信号显示的频率图形与实际正弦波的频率图形保持一致；开发板的音频模块以 48KHz 的采样率采集电脑上音频信号，而 FFT 处理的点数为 1024 点，因此离散谱的基本频率为 $48K/1024$ 且等于 46.8，46.8 表示数字频率为 i 的时候，实际的模拟频率为 $46.8*i$ ，此处数字频率对应为开发板上显示的频率，模拟频率对应电脑中发生器产生的音频信号，与表 4-1 信号源与实际显示的频率之间的比率在去掉电噪声影响后几乎保持一致，说明频谱显示系统正确处理和显示正弦信号的频率。

所用的资源对比分析：图 4-10 所示为使用 HLS FFT IP 搭建测试系统综合后的硬件资源消耗情况，图 4-11 所示为使用 Xilinx FFT IP 搭建测试系统综合之后的资源消耗情况。

Resource	Estimation	Available	Utilizat...
FF	10174	106400	10
LUT	7408	53200	14
Memory LUT	871	17400	5
I/O	147	200	74
BRAM	11	140	8
DSP48	18	220	8
BUFG	2	32	6

图 4-10 使用 HLS FFT IP 综合后的资源消耗情况

Fig. 4-10 Resource consumption after comprehensive by using HLS FFT IP

Resource	Utilization	Available	Utilizat...
FF	9812	106400	9
LUT	6828	53200	13
Memory LUT	719	17400	4
I/O	17	200	9
BRAM	11	140	8
DSP48	18	220	8
BUFG	2	32	6

图 4-11 使用 Xilinx FFT IP 综合后的资源消耗情况

Fig. 4-11 Resource consumption after comprehensive by using Xilinx FFT IP

将硬件和软件协同起来在 ZedBoard 开发板上实现，结果表明使用 HLS 高层次综合工具设计的 FFT IP 能够完成对信号的频谱处理并输出频谱信息，同时也表明 OLED IP 也能够正常工作；对比图 4-10 和图 4-11，在实际资源消耗的“Estimation”项，使用自制 FFT IP 综合后消耗的 FF、LUT、以及 Memory LUT 资源比使用 Xilinx FFT IP 略微多了一些，表明 HLS FFT IP 性能接近 Xilinx 的 FFT IP。说明 HLS 高层次设计工具在实际应用中能够发挥重大的作用，在已有 IP 不能满足设计要求时，可以不使用传统 HDL 语言生成 IP 的方法，而是利用 Vivado HLS 工具进行算法设计并转化为 IP，可以提高生产效率。

4.3.4 根据频谱设计去噪系统

4.4 本章小结

本章主要采用软硬件协同设计的方法进行频谱显示系统的设计，使用高层次综合工具 HLS 创建的 FFT IP 联合在硬件开发环境 Vivado 下设计的各个子模块 OLED IP、Audio IP 等搭建了一个完整的音频显示系统，并对单频正弦信号进行 FFT 处理，结果表明对单一信号进行频谱处理的正确性。为了对比说明设计的 FFT IP 在整个开发板中的资源消耗情况，将使用 Xilinx 的 FFT IP 替代 HLS FFT IP 搭建系统，对两者的结果加以对比分析。在软件开发环境 SDK 中编写测试程序，测试 FFT IP 和系统总体设计的正确性，并在 ZedBoard 开发板上实时显示音频频谱，最终结果证明利用 HLS 设计的 FFT IP 以及 Vivado 下设计的频谱显示系统是正确的，在资源消耗上比 Xilinx 公司提供的 FFT IP 多一点。

5 总结与展望

5.1 全文工作总结

本课题是以数字信号的发展为研究背景，利用软硬件协同设计方法设计了一个频谱实时显示系统，以 Xilinx 公司的 Zynq-7000 为开发平台、Vivado 为开发环境，将 PC 上的音频信号传给 ZedBoard 开发板，之后将获取的音频信号通过开发板中硬件部分 HLS FFT IP 对音频进行 FFT 处理后送到 OLED 进行音频频谱的实时显示。本课题是以 FFT 频谱分析、OLED 显示技术为基础，搭建完整的音频显示系统，在 Vivado SDK 进行硬件系统的控制程序的编写，最终在 ZedBoard 开发板上进行验证。

现将课题的具体工作总结如下：

(1) 对课题研究背景意义及原理进行了介绍。首先对数字信号处理的发展动态进行了介绍，其次对 FFT 算法的基本原理和 Vivado HLS 开发工具进行了详细的说明。

(2) 利用 HLS 工具设计了 FFT IP。在硬件结构的实现中，本课题根据基-2 时域算法的特点利用 C 语言设计流水线数据流 FFT IP，这种 FFT IP 具有并行运算和流水线相结合的特点；经由 Vivado HLS 对设计的 FFT IP 综合并分析占用的资源，然后进行优化处理，减少 FFT 硬件资源消耗和数据处理时间；与 Matlab 中的 FFT 处理结果进行对比验证，证明算法的正确性，最终导出一个 RTL 级 IP，供硬件环境中使用。

(3) 基于软硬件协同方法设计了 OLED IP。为了正确显示频谱，本课题中利用 Vivado 开发环境设计了 OLED IP，利用 SDK 编写接口驱动程序，在 ZedBoard 开发板上通过 OLED 显示器上显示字母和标点符号验证设计的 OLED 驱动程序，结果证明设计的 OLED IP 能够准确显示字母以及编写的驱动程序是正确的。

(4) 设计了音频驱动程序。本次设计的频谱显示系统的激励信号是音频信号，需要音频控制器 Audio IP 控制音频信号，利用 SDK 设计音频驱动程序，搭建硬件系统测试设计的音频驱动程序，测试表明设计的音频驱动是正确的。

(5) 设计并实现了频谱显示系统的搭建。在 Vivado 集成开发环境中综合利用 FFT IP，OLED IP，Audio IP 以及 DMA 等 IP 搭建一个完整的音频频谱显示系统，综合并生成比特流文件，导出到 PS（软件设计）部分；利用 SDK 编写测试程序、驱动程序，最后将硬件的 bit 流文件和软件的可执行文件下载到 ZedBoard 开发板中实现；利用音频显示系统对单个频率的正弦音频信号进行测试并分析频谱显示的正确性；利用 Xilinx 提供的 XFFT IP 替换 HLS FFT IP，对比两次的显示结果并分析资源消耗。

理论和实际测试结果表明基于 Vivado 搭建的硬件频谱显示系统能够实现实时且正确显示音频信号频谱的功能，证明频谱显示系统设计是正确的。

5.2 未来工作展望

本课题利用软硬件协同设计的方法搭建频谱显示系统，实现在 OLED 上准确实时地显示音频信号的频谱。当然，设计中也存在着一些不足之处，需要在今后的研究中进一步改进和完善。

（1）本课题搭建的系统主要是单线程的工作，因而造成了实时性上的瓶颈，在未来的工作研究中希望多线程工作。

（2）本课题的实现中目前没有用到操作系统，因而没有体现出 ZedBoard 强大资源的优势，下一步的工作是希望能在 PS 部分用到 Linux 操作系统以及 API，提高软硬件设计的性能，而且还能通过从 PS 外设中读取音频信号，在 PL 中进行处理，进而充分利用 ZedBoard 丰富的硬件资源实现复杂的处理任务。

致 谢

时光飞逝，转眼间研究生学习也即结束，想想三年来研究生生活的经历，一切情景都显现在脑海里挥之不去，值得我用一颗感恩的心去铭记它。在这里我向那些给予我支持、鼓励和帮助的老师、同学和朋友送上我最美好的祝愿和最诚挚的感谢。

在这里，我要首先感谢我的导师—张俊涛教授，张老师渊博的知识、开阔的思维和严谨的治学态度深深的影响了我，在我论文的选题到定题以及最终文章的定稿，张老师都提出了宝贵的意见。生活中，张老师以一位长辈和亲人的身份来和我交流、关心我，当生活不如意时更是讲道理来指引我；张老师对我的教导我将铭记在心，这些教益终将成为我人生道路上的一盏亮灯陪伴我、指引我、激励我奋勇向前。在这里非常感谢张俊涛老师在硕士期间给予我的所有关心和指导。

我也非常感谢我的父母在我求学期间所给予的无限支持与包容，一直以来，他们的关怀与理解是我的精神支柱。当我学习遇到挫折和不顺心时，他们鼓励我，给我自信；当我取得一点点成就时，他们又使我明白戒躁戒燥的重要性；生活中他们总是尽可能的满足我但又不会让我产生虚荣心；他们对我奉献出了无私的爱与照顾是我不断前进的动力，他们的鼓励与关怀使我能专心完成学业。在此诚挚的感谢我的父母，你们辛苦了。

我要感谢我的室友，她们和我一起来到了这个陌生的校园，一起感受生活的酸甜苦辣，一起谈天说地畅想未来，生活中给予我陪伴和照顾，使我在校园的生活不再孤单。同时我也要感谢我的师弟师妹，我们大家在一起互相学习和帮助、互相监督，大家在一起营造了一个良好的学习氛围并为未来而拼搏共同进步。还有感谢同班同学，正是由他们这些可爱和可亲的人，让我感受到了学校生活的多姿多彩，在此对你们的关心和帮助表示感谢。

最后感谢我的母校—陕西科技大学，感谢她给了我一个继续深造的机会让我来完成自己的学业，在此祝愿母校的明天更加辉煌。

感谢各位评审老师，让我有机会审视自己的不足。在此祝福你们身体健康、生活如意。

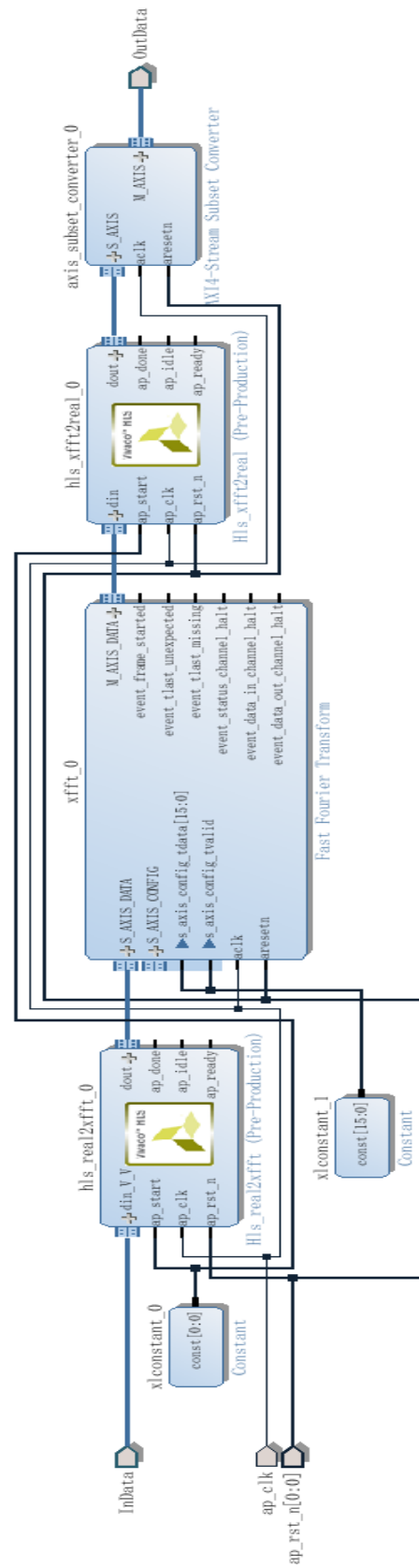
参考文献

- [1] 彭宇, 姜红兰, 杨智明等. 基于 DSP 和 FPGA 的通用数字信号处理系统[J]. 国外电子测量技术, 2013, 32 (1): 17-21.
- [2] 陈智, 王贵锋, 柳莺. 一种基于 IP Core 实现 FFT 变换的新方法[J]. 自动化与仪器仪表, 2012 (2): 163-164.
- [3] 张骥, 杨天凯. 基于 FPGA 的谐波检测装置设计[J]. 自动化应用, 2011 (12): 47-48, 52.
- [4] 于海, 姚启桂, 虞跃等. 基于 SoPC 的状态监测装置的嵌入式软硬件协同设计[J]. 现代电子技术, 2012, 35 (22): 1-3.
- [5] 李志军, 陈丽娟, 刘建霞等. 实现 SOPC 的嵌入式软硬件协同设计平台[J]. 单片机与嵌入式系统应用, 2011, 11 (5): 8-10.
- [6] 何巍, 贺飞, 顾明. 分布式嵌入式系统软硬件协同仿真平台[J]. 计算机工程与设计, 2014, 35 (5): 1607-1611.
- [7] 赵建勋, 王兆东. 软硬件协同设计算法的嵌入式人脸识别系统[J]. 计算机仿真, 2013, 30 (7): 371-374.
- [8] 杨宏璋, 王嘉. 用 C 语言建模辅助软硬件协同设计[J]. 信息技术, 2010, 34 (8): 62-64.
- [9] 杨守良, 程鹏宇. 基于 Nios 的 FFT 算法软硬件协同设计[J]. 电子设计工程, 2010, 18 (9): 158-160.
- [10] 陆佳华, 江舟, 马岷. 嵌入式系统软硬件协同设计实战指南[M]. 北京: 机械工业出版社, 2013: 154-159.
- [11] 游余新. 复杂 SOC 的软硬件协同验证解决方案[J]. 中国集成电路, 2013 (1): 68-73.
- [12] 刘洋. 基于片上多核的版面加速器系统优化研究[J]. 计算机应用研究, 2011, 28 (10): 3731-3734.
- [13] 韩红蕾, 刘文菊, 武继刚等. 多核片上系统的高效软硬件划分及调度算法[J]. 计算机工程与科学, 2011, 33 (9): 57-62.
- [14] 沈淦松, 叶玉堂, 刘霖等. FPGA 软硬件协同处理实时图像处理系统[J]. 光电工程, 2012, 39 (10): 143-150.
- [15] 张开峰. 软硬件协同仿真在图像滤波器演化设计中的应用[J]. 宇航学报, 2012, 33 (12): 1815-1822.
- [16] U.Ali, M.Malik. Hardware/software co-design of a real-time kernel based tracking system[J]. Systems Architecture, 2010, 56 (8): 317-326.

- [17] 李平, 廖永波, 阮爱武等. SoC 软硬件协同技术的 FPGA 芯片测试新方法[J]. 电子科技大学学报, 2009, 38 (5): 716-719.
- [18] 叶华, 武继刚. 软硬件协同设计复杂问题的计算模型和算法[J]. 电子科技大学学报, 2011, 40 (3) : 333-345.
- [19] G.Nguyen Thi Huong, Y.Na, S.Kim. Applying frame layout to hardware design in fpga for seamless support of cross calls in cpu-fpga coupling architecture[J]. Microprocessors and Microsystems, 2011, 35 (5): 462-472.
- [20] James Hrica. 利用赛灵思 Vivado HLS 实现浮点设计[J]. 今日电子, 2013, 01: 34-38.
- [21] 刘毅飞. 基于 Python 软硬件协同设计方法[J]. 现代电子技术, 2013, 36 (8): 76-78.
- [22] Ghissoni S , Costa E , Monteiro J , et al . Combination of constant matrix multiplication and gate-level approaches for are8 and power efficient hybrid radix-2 dit / fft realization[J]. Circuits and Systems, 2011: 567-570.
- [23] Abhishek Kesh. Implementation of fast Fourier transform on FPGA using Verilog HDL[D]. Kharagpur: Indian Institute of Technology, 2004.
- [24] Sarbishei, Radecka K. Analysis of Mean-Square-Error (MSE) for Fixed point FFT Units [J]. Circuits and Systems, 2011: 1732-1735.
- [25] Mateus Bf, Eduardo A, Csarda Costa. Martins design of power efficient butterflies from Radix-2 DIT FFT using adder compressors with a new XOR gate topology[C] . Analog Integrated Circuits and Signal Processing, 2012, 73(3): 945-954.
- [26] 张裕, 方康玲. 基于 FPGA 的通用 FFT 处理器的设计[J]. 计算机技术与发展, 2010, 20 (8): 87-90.
- [27] 杨晶, 康宁, 王元庆. 基于低成本 FPGA 的 FFT 设计实现[J]. 电子器件, 2013, 36 (4): 506-509.
- [28] 李大习. 基于 FPGA 的可配置 FFT IP 核实现研究[J]. 电子科技, 2014, 27 (6): 46-49.
- [29] 李欣, 刘峰, 龙腾. 定点 FFT 在 TS201 上的高效实现[J]. 北京理工大学学报, 2010, 30 (1): 88-91.
- [30] 任健, 高晓蓉. 基于 FPGA 的 FFT 处理器设计[J]. 现代电子技术, 2010, 33 (24): 142-144.
- [31] 李仕专, 李维涛, 姜全贤等. 一种基于并行计算的快速 FFT IP 核设计[J]. 计算机与数字工程, 2010 (04): 139-141.

- [32] 栗旭光, 何国经, 刘江涛. 基于 FPGA IP 核的 FFT 实现与改进[J]. 电子科技, 2013, 26 (12): 96-100.
- [33] 姜丽丽, 薛岩, 郭凤仪等. 基于 FPGA 的电力谐波分析仪的研制[J]. 电子技术, 2011, 37 (7): 100-102.
- [34] 梁赫西, 陈佑红, 郭朝霞. 基于 FPGA 的可配置 FFT_IFFT 处理器的设计与实现[J]. 电子技术应用, 2012, 38 (3): 57-59.
- [35] 童庆为, 陈建春. 基于 FPGA 的 FFT 算法硬件实现[J]. 电子科技, 2010, 23 (11): 113-115.
- [36] 万书芹, 阮园, 于宗光等. 混合 CORDIC 在分裂基 FFT 中的应用[J]. 计算机工程与应用, 2010, 46 (11): 73-76.
- [37] 马壮, 齐林, 马鹏阁等. 基于 FPGA IP 核的 FFT 实现[J]. 现代电子技术, 2009, 32 (7): 160-162.
- [38] 窦秀梅, 赵振纲. 基于 IP 核的 FPGA FFT 算法模块的设计与实现[J]. 无线电工程, 2008, 38 (8): 29-31.
- [39] 丛秋波. Vivado 设计套件将可编程系统集成度和实施速度提升 4 倍[J]. 电子设计技术, 2012, 19 (6): 22-24.
- [40] 何宾. Xilinx All Programmable Zynq-7000 SOC 设计指南[M]. 北京: 清华大学出版社, 2013.
- [41] 21ic 官方微博. 赛灵思 Vivado 开启 “All Programmable” 新征程[EB/OL].
<http://www.21ic.com/wz/CRXDQWHFA/20120501/20120501.htm>, 2014-08-15.
- [42] 思文. Vivado 设计套件将速度提高四倍[N]. 中国电子报, 2013-06-18 (007).
- [43] Xilinx. Vivado Design Suite Tutorial High-Level Synthesis[EB/OL].
http://www.xilinx.com/support/documentation/sw_manuals/xilinx2012_4/ug871-vivado-high-level-synthesis-tutorial.pdf, 2014-08-15.
- [44] 杨晓安, 罗杰, 包文博. 基于 Xilinx Zynq 的物距测量系统设计与实现[J]. 现代电子技术, 2014, 37 (15): 123-126.
- [45] 宋广南. 多通道 DBF 专用测试模拟器的设计与实现[D]. 南京: 南京理工大学, 2013.
- [46] 邢艳芳, 张延冬. 基于 Zynq 的 OLED 驱动设计[J]. 液晶与显示, 2014 (2): 224-228.

附录 B: Real FFT IP 的组成



附录 D：测试频谱系统的主程序中 main 函数代码

```

int main(void)
{
    int sw_check;//定义一个存储开关状态的变量
    Xint16 audio_data;
    //=====audio=====
    IicConfig(XPAR_XIICPS_0_DEVICE_ID); //配置IIC的数据结构
    AudioPllConfig();//配置Audio编码的锁相环
    AudioConfigureJacks();//使能HP接口.
    LineinLineoutConfig();//配置音频输入输出接口
    printf("audio initialized done\n\r");//提示音频初始化完成
    //=====oled=====
    Xil_Out32(OLED_BASE_ADDR,0xff);
    printf("oled initializing\n\r");//开始初始化OLED
    OLED_Init();//初始化液晶
    printf("oled initialized done\n\r");//OLED初始化完成
    //=====
    printf("PROG ing \n\r");
    sw_check = Xil_In32(XPAR_AXI_GPIO_0_BASEADDR+8);//读取开关状态
    while(1)
    {
        sw_check = Xil_In32(XPAR_AXI_GPIO_0_BASEADDR+8);
        if(sw_check == 0x01) //SW0高电平时OLED显示频谱信息
        {
            xil_printf("executing audio_fre()!\n\r");
            OLED_Clear();
            OLED_Refresh_Gram();
            audio_fre();}
        else
        {
            get_audio(&audio_data);//SW0为低时从音频输出口接收音频信号
            xil_printf("current audio_data is: %d \n\r", audio_data);//提示当前的音频信号的大小
            audio_orignal();//输出音频信号
            xil_printf("done!\n\r");} //提示完成
    }
    return 0;
}

```

攻读学位期间发表的学术论文目录

- [1] 张俊涛, 曹梦娜. 基于 LabVIEW 的宽频时变信号阈值去噪系统设计[J]. 陕西科技大学学报, 2014, 32 (6) : 161-164.

原创性声明及关于学位论文使用授权的声明

原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的科研成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律责任由本人承担。

论文作者签名：_____ 日期：2015 年 月

关于学位论文使用授权的声明

本人完全了解陕西科技大学有关保留、使用学位论文的规定，同意学校保留或向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅；本人授权陕西科技大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或其他复制手段保存论文和汇编本学位论文。同时授权中国科学技术信息研究所将本学位论文收录到《中国学位论文全文数据库》，并通过网络向社会公众提供信息服务。

（保密论文在解密后应遵守此规定）

论文作者签名：_____ 导师签名：_____ 日期：2015 年 月