

**电子技术程序设计报告**

院 系 人工智能与自动化学院

专业班级 人 工 智 能 2304 班

姓 名 杜 辰 宇

学 号 U 2 0 2 3 1 5 2 6 5

指导教师 程 晶 晶

2024 年 12 月 27 日

目录

[1正文 - 2 -](#_Toc186187671)

[1.1嘉立创EDA绘制的XC7A35T-1FTG256C开发板原理图 - 2 -](#_Toc186187672)

[1.2 XC7A35T-1FTG256C的特性、资源及可以替代的国产FPGA - 6 -](#_Toc186187673)

[1.2.1XC7A35T-1FTG256C的特性 - 6 -](#_Toc186187674)

[1.2.2XC7A35T-1FTG256CFPGA包含的资源 - 7 -](#_Toc186187675)

[1.2.3引脚定义注意事项 - 8 -](#_Toc186187676)

[1.2.4国产FPGA替代型号及采购渠道 - 9 -](#_Toc186187677)

[1.3如何在电子工程师群体中推广使用嘉立创EDA，替代AltiumDesignerEDA工具软件？ - 11 -](#_Toc186187678)

[1.3.1. 突出嘉立创EDA的核心优势 - 11 -](#_Toc186187679)

[1.3.2强化云端服务和协作功能 - 12 -](#_Toc186187680)

[1.3.3社区和用户支持和合作伙伴关系 - 12 -](#_Toc186187681)

[1.4不能用开源软件替换收费的 Vivado 软件的原因 - 14 -](#_Toc186187682)

[1.5数字电路实验报告 - 16 -](#_Toc186187683)

[1.5.1组合逻辑电路中的数值比较器实验 - 16 -](#_Toc186187684)

[1.5.2时序逻辑电路中的同步四位二进制加计数器实验 - 19 -](#_Toc186187685)

[1.5.3存储器电路中的存储器FIFO实验 - 26 -](#_Toc186187686)

[1.6基于 Sobel 算子的边缘检测的原理 - 35 -](#_Toc186187687)

[1.6.1原理概述 - 35 -](#_Toc186187688)

[1.6.2详细原理 - 35 -](#_Toc186187689)

[1.6.3 总结 - 36 -](#_Toc186187690)

[1.7 FPGA实现边缘检测图像预处理具有较好实时性的原因 - 37 -](#_Toc186187691)

[1.7.1 FPGA实现边缘检测的并行处理能力 - 37 -](#_Toc186187692)

[1.7.2流水线设计 - 38 -](#_Toc186187693)

[1.7.3并行I/O操作 - 38 -](#_Toc186187694)

[1.7.4定制化硬件逻辑 - 38 -](#_Toc186187695)

[2. 致谢 - 39 -](#_Toc186187696)

1正文

1.1嘉立创EDA绘制的XC7A35T-1FTG256C开发板原理图

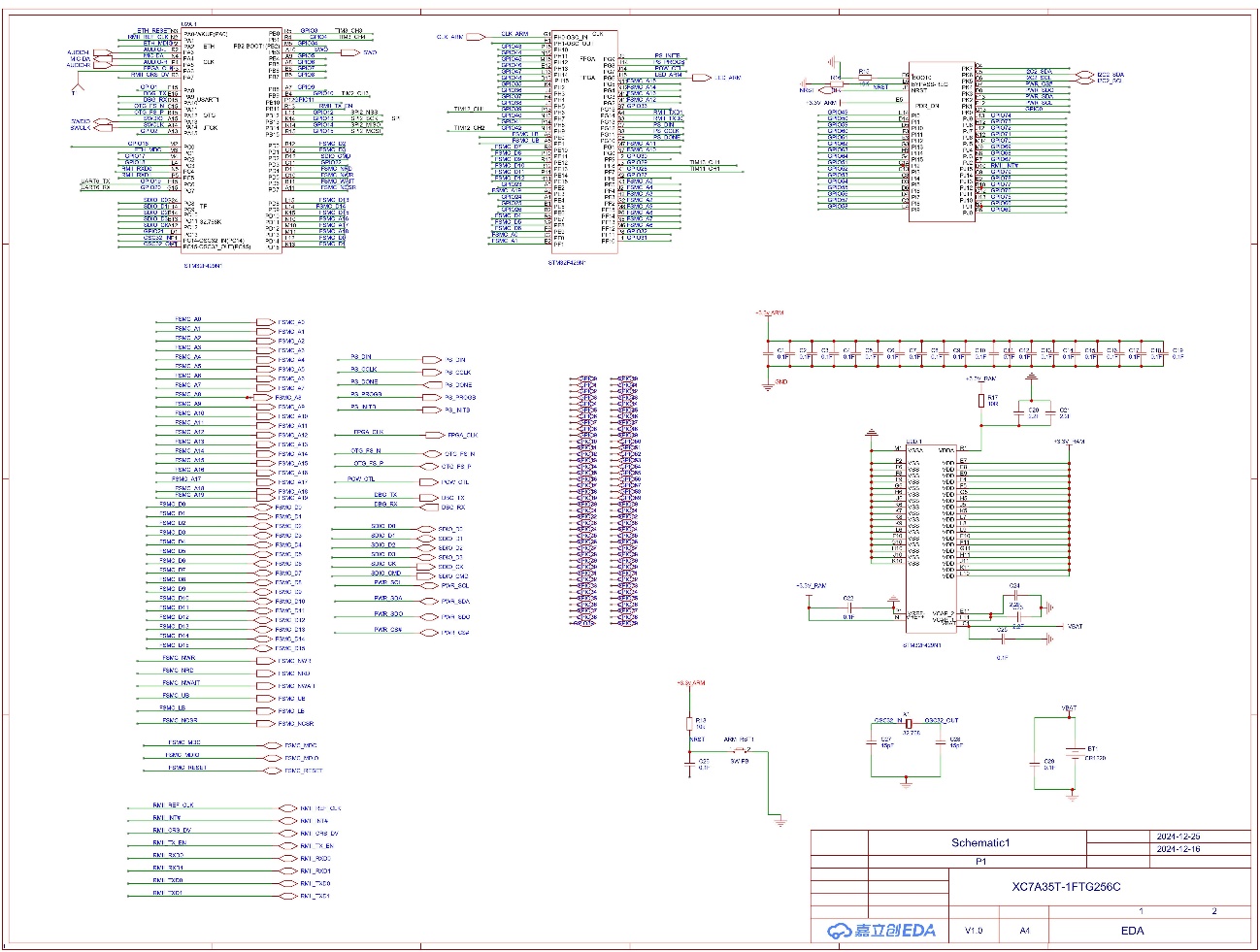
****

图1-1 XC7A35T-1FTG256C开发板原理图第一张

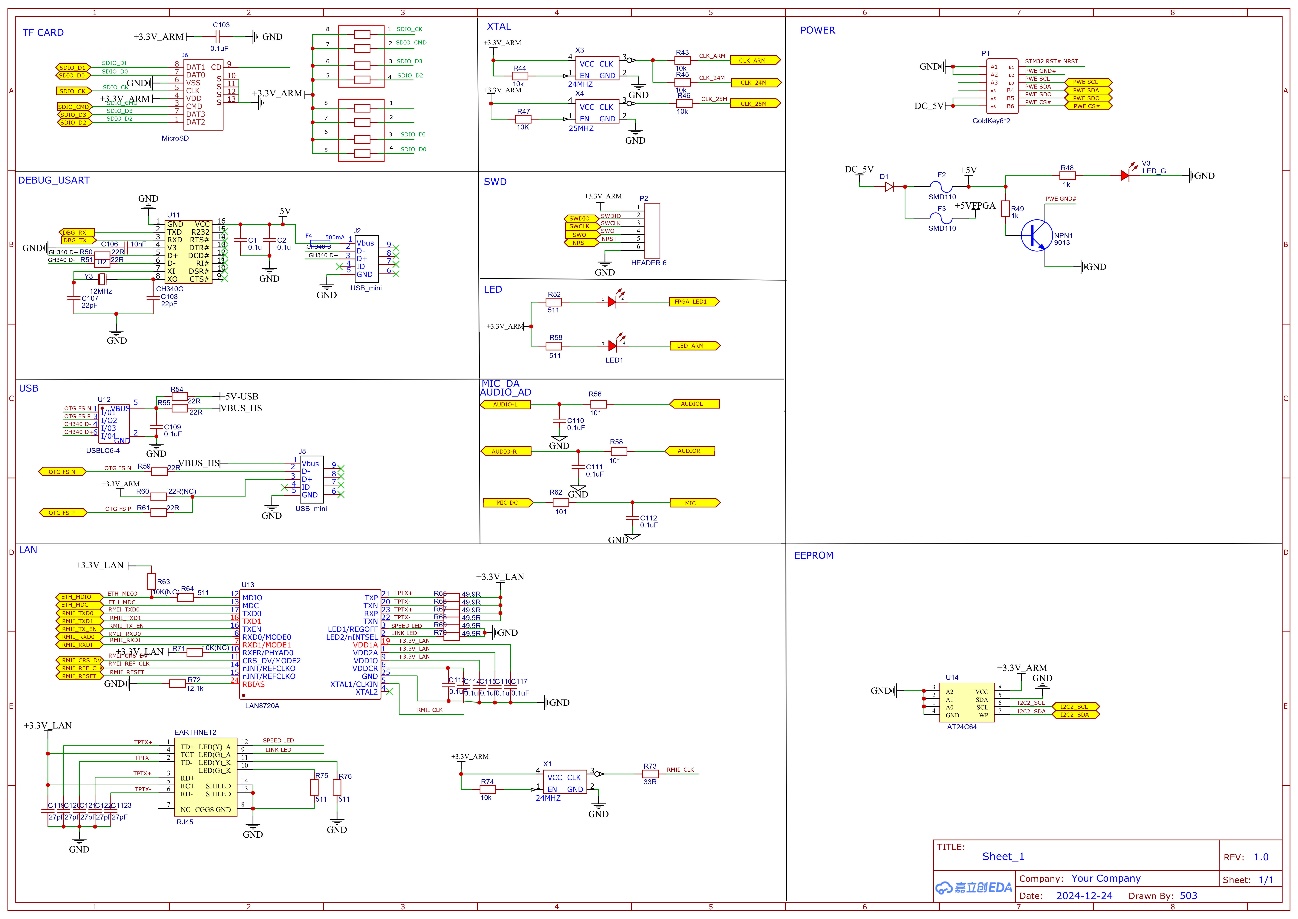
****

图1-2 XC7A35T-1FTG256C开发板原理图第二张

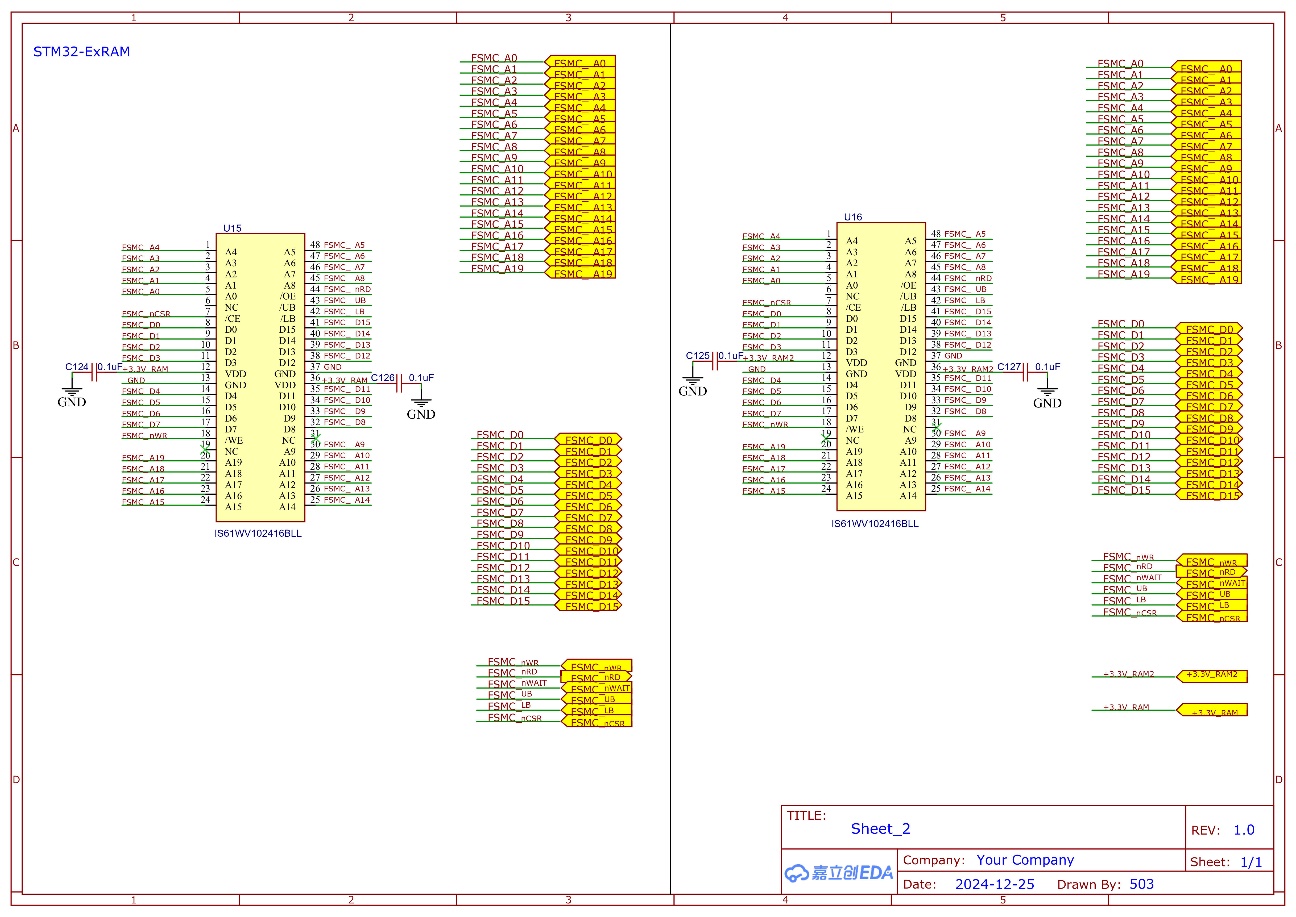
****

图1-3 XC7A35T-1FTG256C开发板原理图第三张

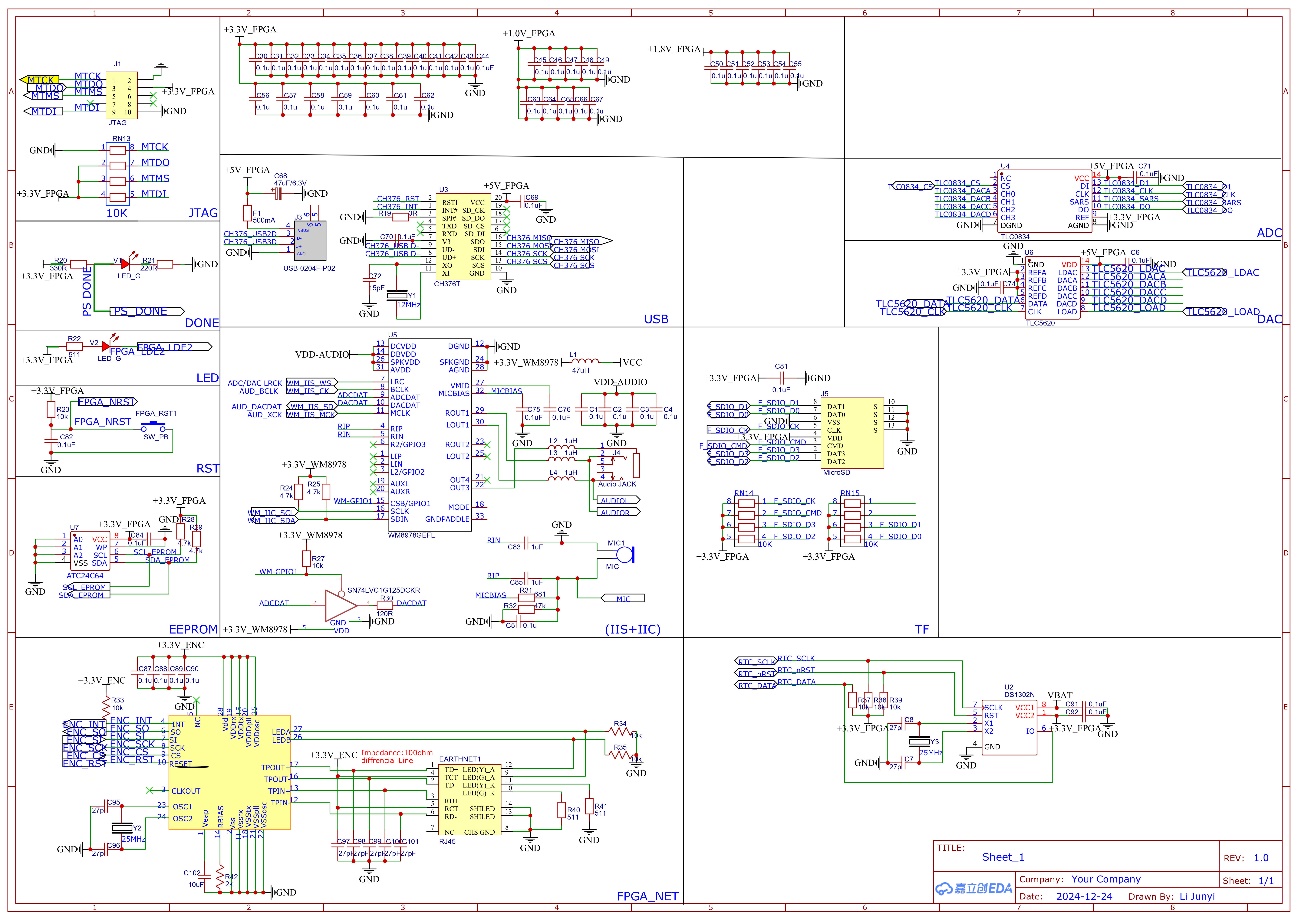
****

图1-4 XC7A35T-1FTG256C开发板原理图第四张

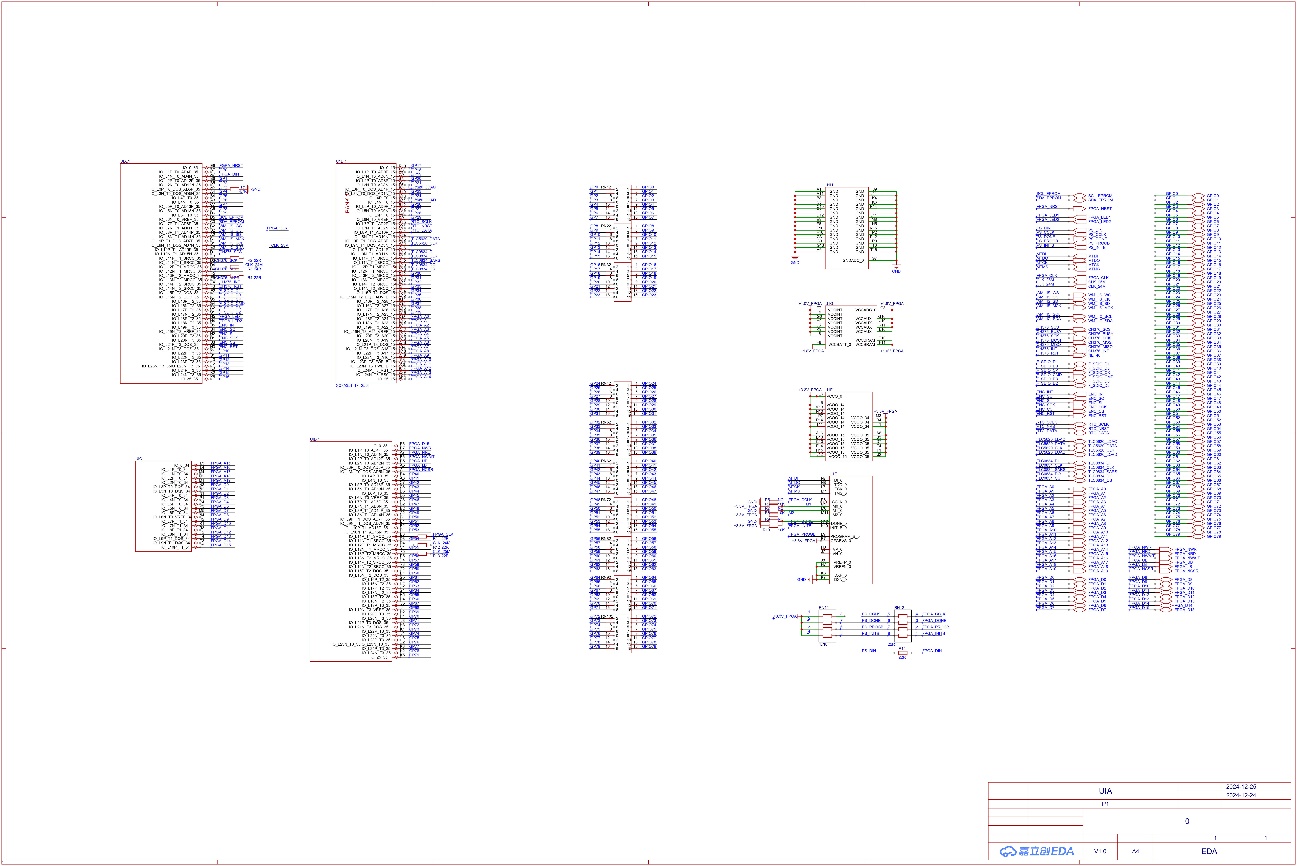
****

图1-5 XC7A35T-1FTG256C开发板原理图第五张

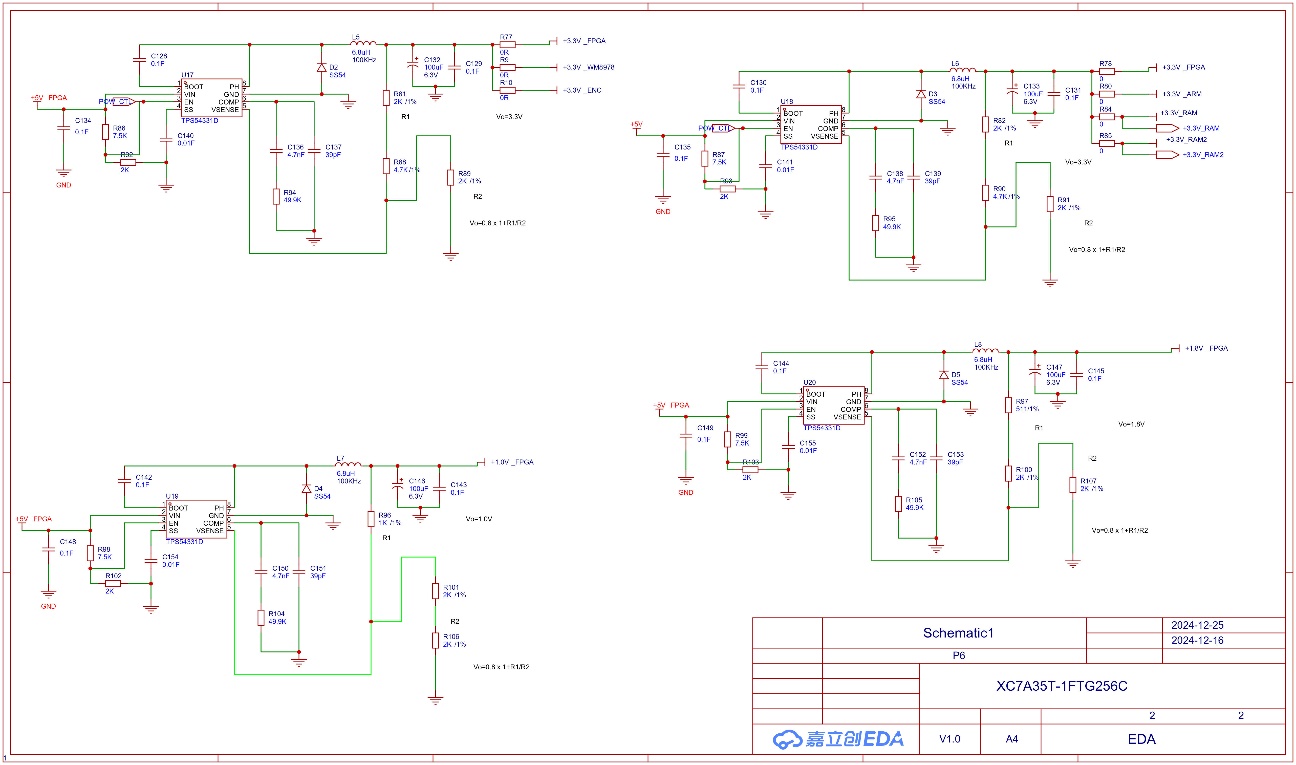
****

图1-6 XC7A35T-1FTG256C开发板原理图第六张

1.2 XC7A35T-1FTG256C的特性、资源及可以替代的国产FPGA

XC7A35T-1FTG256C是赛灵思（Xilinx）公司推出的一款基于Artix-7系列的高性能FPGA。该芯片以其高度集成和灵活性，广泛应用于通信、工业控制、消费电子等领域。

1.2.1XC7A35T-1FTG256C的特性

XC7A35T-1FTG256CFPGA以其领先的技术特性在高性能计算、高速数据传输和灵活的设计应用方面脱颖而出。这款FPGA采用了先进的28nm工艺，提供了卓越的计算能力和低功耗运行，支持多种低功耗模式以适应不同的应用场景,处理高速数据通信方面表现出色。此外，XC7A35T-1FTG256CFPGA还具备强大的安全性能，包括集成的AES加密和SHA-256认证，确保了数据传输的安全性。

1.2.1.1性能特性

高性能计算能力：凭借大量的逻辑单元和DSPSlices，XC7A35T-1FTG256C能够处理复杂的计算任务，适合高性能计算应用。

高速串行通信：通过集成的GT收发器，支持高达12.5Gbps的数据传输速率，适合高速数据通信。

1.2.1.2低功耗特性

先进工艺技术：采用28nm工艺技术，实现高性能与低功耗的平衡。

低功耗模式：支持多种低功耗模式，包括动态功耗管理，以适应不同的功耗需求。

1.2.1.3灵活性和可扩展性

I/O配置灵活性：支持多种I/O标准和丰富的I/O引脚，使得设计可以灵活适应不同的接口需求。

可扩展的内存接口：支持多种类型的内存接口，包括DDR3，为系统设计提供可扩展的内存解决方案。

1.2.1.4集成度和集成解决方案

集成的处理器支持：支持MicroBlaze™软处理器，提供裸机、freeRTOS和Linux支持，便于实现复杂的控制逻辑。

集成的安全特性：集成AES加密和SHA-256认证，为数据传输提供安全保障。

1.2.2XC7A35T-1FTG256CFPGA包含的资源

XC7A35T-1FTG256CFPGA以其卓越的性能、高度集成的特性和灵活的设计能力在众多FPGA产品中脱颖而出。这款设备不仅搭载了先进的28nm工艺技术，而且在处理速度、功耗控制、I/O接口多样性以及系统集成等方面展现出了显著的优势。其特性集中体现在高效的逻辑单元配置、强大的DSP处理能力、丰富的内存资源、灵活的I/O选项以及精确的时钟管理等方面，这些特性共同赋予了XC7A35T-1FTG256CFPGA在高速数据处理、复杂逻辑实现和多标准I/O支持等方面的强大竞争力。

1.2.2.1逻辑资源

逻辑单元（LogicCells）：拥有35,000个逻辑单元，基于6输入查找表（LUTs），为复杂的组合逻辑提供强大的支持。

内存资源

块RAM（BlockRAM）：提供640Kb的块RAM，这些双端口RAM资源可用于数据缓存、帧缓冲等高速存储需求。

分布式RAM（DistributedRAM）：拥有96Kb的分布式RAM资源，分布在FPGA内部，便于实现局部数据存储和处理。

数字信号处理资源

DSPSlices：包含90个DSPSlices，每个DSPSlice包含48位乘法器和累加器，适合进行数字信号处理和浮点运算。

1.2.2.2 I/O资源

I/O引脚：提供170个I/O引脚，支持多种不同的I/O标准，包括LVDS、HSTL、LVCMOS等，以适应各种外部接口需求。

高速串行I/O：支持GT收发器，实现最高12.5Gbps的数据传输速率，适用于高速串行通信。

1.2.2.3时钟和电源管理资源

时钟管理单元（CMT）：集成5个时钟管理单元，包括锁相环（PLL）和混合模式时钟管理器（MMCM），提供精确的时钟控制。

电源管理：支持多种电压级别的电源输入，包括核心电压（1.0V）和I/O电压（3.3V），以及专用的VCCO电压，为不同功能的I/O提供电源。

1.2.2.4其他特殊资源

模拟数字转换器（XADC）：集成双12位模数转换器，提供片上温度和电压监控。

PCIExpress接口：支持PCIExpress（PCIe）x8Gen3端点和根端口设计，实现高速数据传输。

安全特性：包括AES加密和SHA-256认证，增强数据安全性。

1.2.3引脚定义注意事项

在进行XC7A35T-1FTG256CFPGA的引脚定义时，关键在于确保电源稳定性、信号完整性、电磁兼容性以及热管理。这些因素共同决定了FPGA的性能和可靠性，需要在设计初期就给予充分考虑和精确配置。

1.电压等级匹配：确保外部电路的电压等级与FPGA引脚的电压等级相匹配，防止电压不匹配导致的损坏。

2.信号完整性与电源完整性：对于高速信号，需考虑信号完整性（SI）和电源完整性（PI）设计，以减少信号反射和串扰。

3.电磁兼容性：设计时应考虑电磁兼容性（EMC）要求，通过合理的布局和布线减少电磁干扰。

4.热管理：由于FPGA在运行中会产生热量，需要考虑散热设计，如使用合适的散热片和风扇。

5.引脚分配：合理分配引脚，避免将关键信号和噪声敏感信号放置在相邻位置，减少串扰。

1.2.4国产FPGA替代型号及采购渠道

随着国产FPGA技术的快速发展，已有多家国内厂商推出了性能相当的替代产品。**几种替代型号及其特点**：

1.复旦微FMQL45T900：复旦微是国内FPGA领域技术较为领先的公司之一，其亿门级FPGA芯片基于28nm工艺制程，是国内最早研制成功的亿门级FPGA芯片，且目前已经实现了量产销售。复旦微还推出了自主研发的EDA设计工具Procise，界面友好、功能强大且简单易用。

2.智多晶Seagull系列：智多晶（西安）提供的Seagull1000、Seagull2000和Seagull5000系列，覆盖了从低功耗到高性能的不同应用需求。这些系列提供了不同逻辑单元数量的选项，工艺从0.162um到40nm不等，频率从322MHz到更高频率，满足不同性能需求。

**采购渠道**：

国产FPGA的采购渠道多样，主要包括：

1.原厂直销：直接从复旦微或智多晶等原厂购买，可以获得最正规的产品和技术支持。

2.授权分销商：通过原厂授权的分销商购买，如Digi-Key、Mouser等，这些分销商通常能提供快速的物流和良好的售后服务。

3.在线电商平台：在阿里巴巴、京东、天猫等电商平台上，也有众多供应商提供国产FPGA产品。

1.3如何在电子工程师群体中推广使用嘉立创EDA，替代AltiumDesignerEDA工具软件？

1.3.1. 突出嘉立创EDA的核心优势

在推广嘉立创EDA以替代AltiumDesignerEDA工具软件的过程中，突出嘉立创EDA的核心优势是至关重要的。嘉立创EDA以其全面的服务和用户友好的设计，赢得了电子工程师的青睐。

要让客户认识到嘉立创EDA提供的一站式服务，集成了原理图设计、PCB设计的功能，并拥有丰富的器件库和封装库。这种一站式服务极大地便利了学生、企业和个人工程师的科创活动和生产需求。用户可以直接在立创商城采购所需器件，实现从设计到生产的无缝对接，这为工程师节省了宝贵的时间和资源。

同时要突出嘉立创EDA的交互逻辑与市场上流行的AD或protel相似，甚至更为简洁，使得新用户能够迅速上手，降低了学习成本。这种易用性大大降低了用户的学习曲线，使得即使是初学者也能在短时间内掌握基本操作。

此外，嘉立创EDA提供了免费版本，这对于预算有限的小型企业和个人工程师来说是一个很大的吸引力。同时，嘉立创还提供了每月包邮免费打样的福利，这可以帮助用户在实际生产前验证设计，降低了试错成本。

嘉立创EDA的定制化能力也是其核心优势之一。用户可以根据自己的需求自定义元件库和宏命令，这种灵活性使得嘉立创EDA能够适应各种特定的设计需求。

最后，嘉立创EDA依托于嘉立创集团强大的产业链优势，为用户提供了从设计到制造的一站式服务。这种产业链优势不仅为用户提供了快捷的PCB打样和SMT贴片生产服务，还通过立创商城为用户提供了丰富的元器件选择，进一步增强了嘉立创EDA的市场竞争力。

1.3.2强化云端服务和协作功能

在电子工程师群体中推广嘉立创EDA替代AltiumDesignerEDA工具软件时，强化云端服务和协作功能是至关重要的一环。以下是具体的推广策略：

要让客户认识到嘉立创EDA提供的云端服务极大地提高了工作效率。云计算技术支持多人在线协作，如使用在线文档编辑平台，员工可以共同编辑、查看和修改文档，实现实时协同办公。这种协作方式不仅打破了地理限制，还使得团队成员能够随时随地访问项目文件和数据，从而提高工作效率和响应速度。

同时要推广嘉立创EDA突出的云端协作能力。它提供了实时聊天、讨论区和文件共享功能，团队成员可以随时沟通、分享文件，并进行实时协作。这种即时的沟通和信息共享机制，对于快速解决问题和加快项目进度至关重要。

再者，嘉立创EDA的云端服务还包括项目管理工具，如任务分配、进度跟踪等功能，使得团队成员可以随时查看项目进度、分配任务，保证项目顺利进行。这些工具有助于项目经理和团队领导更好地控制项目进度，确保按时交付。

云端服务提供的数据安全保障也是宣传的方面。云计算通过身份验证和权限设置来确保数据的安全性。企业可以限制对机密数据的访问，只向那些拥有密码和安全访问权限的人提供访问权限。这对于保护知识产权和敏感设计数据尤为重要。

嘉立创EDA的云端服务还支持多项目管理，允许团队在同一平台上管理多个项目，提高工作效率。这种集中化的管理方式有助于团队成员更好地理解项目间的依赖关系和资源分配，从而优化工作流程。

1.3.3社区和用户支持和合作伙伴关系

在电子工程师群体中推广嘉立创EDA替代AltiumDesignerEDA工具软件时，社区和用户支持以及合作伙伴关系是两个重要的方面。以下是这两个方面的详细阐述：

嘉立创EDA构建了一个活跃的社区环境，为用户提供了一个交流和解决问题的平台。这个社区不仅增强了用户的粘性，而且为新用户提供了一个学习的平台。用户可以在社区中分享经验、讨论问题，并从其他用户和嘉立创的专家那里获得帮助。这种社区支持极大地促进了知识的共享和问题的快速解决，同时也增强了用户对嘉立创EDA的忠诚度。此外，嘉立创EDA提供了专业客服团队，快速解决用户问题，响应用户需求与建议，践行着从嘉立创EDA成立之初就确立的使命：用简约、高效的国产EDA工具，助力工程师专注创造与创新。

嘉立创EDA在合作伙伴关系方面的努力也是其成功的关键。嘉立创EDA积极与国内外知名企业和科研机构开展合作，共同推动EDA技术的发展和应用。这种合作不仅提升了嘉立创EDA的品牌影响力，也为用户带来了更多的资源和技术支持。例如，嘉立创EDA与全国500多所高校开展课程应用、电子竞赛、实验室建设、工程素质训练等项目合作，受到教师与学生的一致好评。同时，已经有近500家企业选择使用嘉立创EDA，既满足了企业设计需求，也大大提升了企业生产效率。此外，嘉立创EDA还与华为等企业合作，共同推动EDA软件的创新，得益于嘉立创EDA的云原生产品特性，便于团队协作和项目管理，以及集中管理更多的元器件库和模型，提升用户设计效率。

通过强化社区和用户支持以及建立强大的合作伙伴关系，嘉立创EDA能够更好地服务于电子工程师群体，并逐步替代AltiumDesigner成为首选的EDA工具软件。

1.4不能用开源软件替换收费的 Vivado 软件的原因

在探索FPGA设计的广阔天地时，Vivado不仅仅是一款工具，它是Xilinx赋予工程师的一把瑞士军刀，集专业性、性能优化、高级功能集成、技术支持和社区支持以及更新维护于一身的全能伙伴。它的存在，让设计师们在面对复杂的硬件设计挑战时，能够更加从容不迫，游刃有余。

Vivado的设计流程经过精心优化，它能够处理那些高密度、高难度的设计任务，让设计师们能够将更多的逻辑集成到FPGA中，从而降低系统的成本和功耗。这种专业性和性能优化不仅体现在其全面的设计套件中，还包括了高效的逻辑综合、仿真以及调试工具。Vivado通过智能地应用时钟门控技术和其他功耗降低策略，帮助设计者创建出功耗更低、效率更高的FPGA设计。此外，Vivado的性能优化还体现在其对设计流程的深度优化上，如综合、实现和验证的效率提升，以及改进的时序分析工具的提供。

Vivado的高级功能和集成能力为用户提供了一个强大的设计平台。它集成了IP集成、高层次综合、系统级设计和嵌入式开发等高级功能，使得设计者能够快速实现C语言算法IP的ESL设计，以及重用的标准算法和RTLIP封装技术。Vivado的集成化方法解决了传统FPGA设计中的集成瓶颈问题，提高了模块和系统验证的仿真速度，硬件协仿真性能也得到了显著提升。Vivado还支持设计输入在传统HDL如VHDL和Verilog中进行，同时引入了机器学习算法，以实现更佳的时序收敛。

在技术支持和社区支持方面，Vivado展现出了显著的优势。Xilinx公司提供了全面的技术支持服务，包括在线知识库、技术文档、故障排除指南以及直接联系技术支持专家的渠道。这种官方支持确保了用户在遇到复杂问题时能够得到及时和专业的帮助，这对于保证项目按时完成和系统稳定运行非常重要。同时，Vivado的社区支持网络为用户提供了一个宝贵的资源，使得用户可以在这里找到产品信息、知识文章，与其他产品用户和专家快速协作，以及轻松联系赛灵思支持专家。

Vivado的更新和维护策略确保了软件的持续更新和改进，为用户提供了最新的技术和功能，同时也确保了软件的稳定性和可靠性。Xilinx公司对Vivado的持续投入保证了软件能够不断适应和支持最新的FPGA技术和器件，每个新版本都带来了性能的提升和新功能的增加。这种定期的更新不仅确保了Vivado用户能够利用最新的技术进行设计，而且也保证了与最新器件的兼容性，使得设计工作能够紧跟技术发展的步伐。Xilinx还提供了详尽的用户指南和文档，帮助用户了解如何安装、配置和使用Vivado，以及如何利用新版本中的功能。这些文档为用户提供了宝贵的资源，帮助他们最大限度地利用Vivado的能力。同时，Xilinx的技术支持社区也为用户之间的交流和协作提供了平台，使得用户可以在这里找到产品信息、知识文章，与其他产品用户和专家快速协作，以及轻松联系赛灵思支持专家。

随着技术的不断进步，Vivado也在不断进化，它不仅仅是一个工具，更是设计师们创新旅程中的得力助手。在这个快速变化的世界里，Vivado以其强大的功能和不断的更新，确保了设计师们能够站在技术的最前沿，创造出更加卓越和高效的设计。让我们期待Vivado未来带来更多的惊喜，也期待它继续在FPGA设计领域中发光发热。

1.5数字电路实验报告

1.5.1组合逻辑电路中的数值比较器实验

实验器件简介：在数字系统中，特别是在计算机中常需要对两个数的大小进行比较。数值比较器就是对两个二进制数A和B进行比较的逻辑电路，比较结果有A > B、A < B和A = B三种情况。

1.5.1.1代码注释以及Bin文件生成

数据分配器代码以及解释如下：

|  |
| --- |
| *01* ***`timescale 1ns/1ps*** *//时间单位是1纳秒（1ns），时间精度是1皮秒（1ps）。*  *02*  *03* ***module compare\_3 (y,a,b);*** *//定义测试台模块*  *04*  *05*  ***parameter N = 4;*** *//参数化位宽，默认为3*  *06*  ***input [N-1:0] a ;*** *//数据a*  *07* ***input [N-1:0] b ;*** *//数据b*  *08*  ***output [2:0] y ;*** *//比较结果输出，y[2]代表a<b，y[1]代表a=b，y[0]代表a>b*  *09*  *10*  ***reg [2:0] y ;*** *//声明了一个3位宽的寄存器y，用来存储比较两个输入信号a和b的结果。*  *11*  *12* ***always @ (a or b)begin***  *// 对输入的a、b进行比较*  *13*  ***if (a > b)*** *//当a > b 时*  *14*  ***y <= 3'b001;*** *//输出 y = 3'b001*  *15* ***else if (a == b)*** *//当a = b 时*  *16*  ***y <= 3'b010;*** *//输出 y = 3'b010*  *17*  ***else*** *//当a < b 时*  *18*  ***y <= 3'b100;*** *//输出 y = 3'b100*  *19* ***end***  *20*  *21 endmodule* |

**Bin文件生成：**在Vivado中创建一个新的FPGA工程，创建Verilog文件并编写上述逻辑代码。运行综合工具，检查代码是否有错误。在Vivado的“Flow Navigator”中，找到并点击“Generate Bitstream”按钮，勾选生成bin选项。然后设置FPGA的输入输出管脚约束。在Vivado中生成Bitstream即可生成bin文件

1.5.1.2仿真测试方法的说明

仿真代码以及解释如下：

*01* ***timescale 1ns/1ps***

*//这行设置了仿真的时间单位和时间精度。时间单位是1纳秒（1ns），时间精度是1皮秒（1ps）。*

***02 module compare\_3\_tb;***

*//定义了一个名为compare\_3\_tb的测试台模块。*

***04 reg [3:0]a ,b;***

*//声明了两个4位的寄存器a和b，用于存储输入信号。*

***05 wire [2:0] y;***

*声明了一个3位的线网y，用于连接到被测试模块的输出。*

***07 initial begin***

*//开始一个初始块，这是仿真开始时执行的代码块。*

***08 a = 0;***

*//将寄存器a初始化为0。*

***09 b = 0;***

*//将寄存器b初始化为0。*

***10 fork***

*//关键字fork开始一个并行块，允许在同一个时间点开始多个进程。*

***11 repeat(20) #40 a = a + 1;***

*//重复执行20次，每次循环后等待40时间单位（这里是40皮秒），然后将a的值增加1。*

***12 repeat(25) #10 b = b + 2;***

*//重复执行25次，每次循环后等待10时间单位（这里是10皮秒），然后将b的值增加2。*

***13 join***

*//关键字join结束fork块，所有并行进程在此汇合。*

***14 $stop;***

*//停止仿真。*

***15 end***

*//结束初始块。*

***16 compare\_3 #(.N(4))***

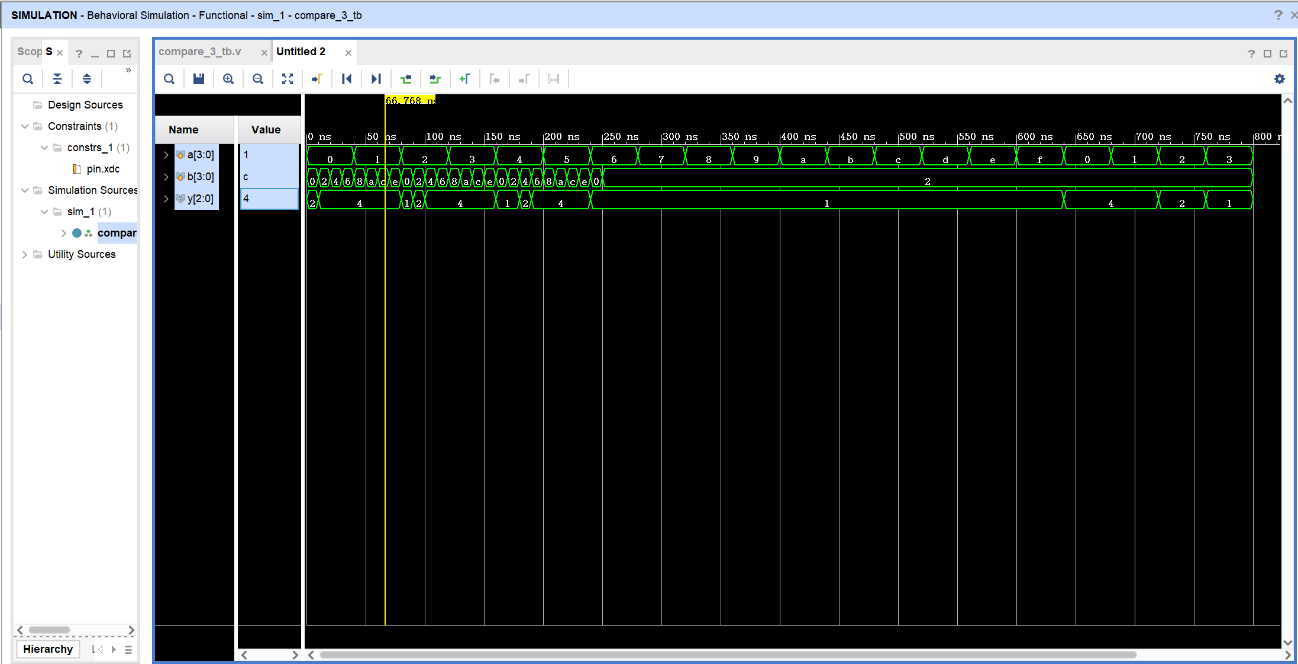
*实例化名为compare\_3的模块，并指定参数N为4。*

***17 compare\_3\_inst( .y (y), .a (a), .b (b) );***

*将实例化的模块compare\_3命名为compare\_3\_inst，并连接其端口。y是输出，a和b是输入。*

***18 endmodule***

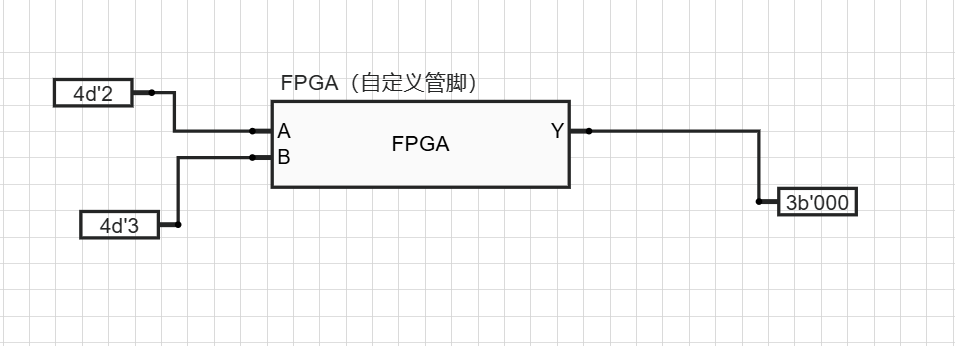
仿真结果如下：



从仿真结果中可以看出，a > b , a < b , a=b 三种情况下 y的输出，符合设计结果。

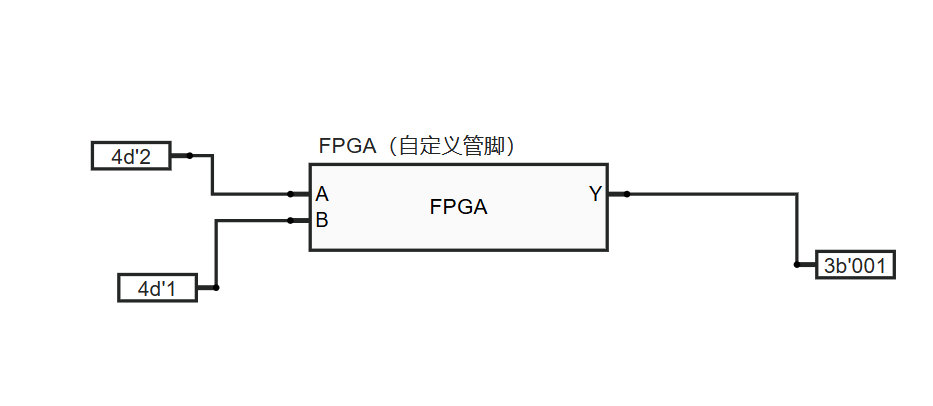
1.5.1.3远程云端实验方法的说明

登录远程云端平台，确定板卡连接无误的情况下，进入实验面板，在右侧器件面板的逻辑器件一栏中，找到自定义FPGA、对应的输入输出向实验面板图纸中央拖拽，同时将外围器件与逻辑器件的输入输出端口相连接，可以绘制如下图的电路实验模型图，位输入作为输入信号，位输出作为监测输出信号。这是一个8输入3输出的逻辑器件，使用2个4bit位输入，1个3bit位输出。

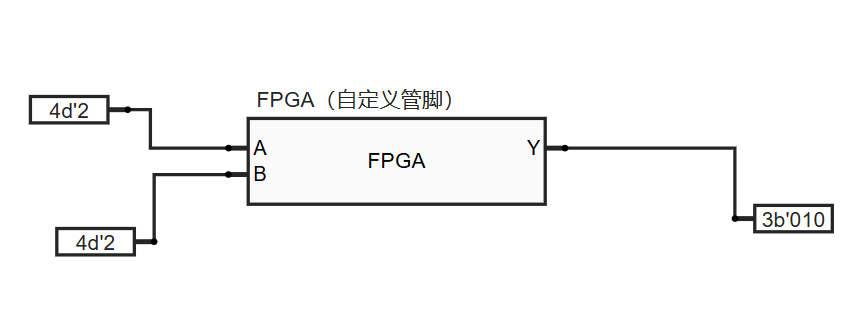


利用bin文件进行FPGA烧结后调整输入数据，观测输出数据。下面为a > b , a < b , a=b 三种情况下的输出。

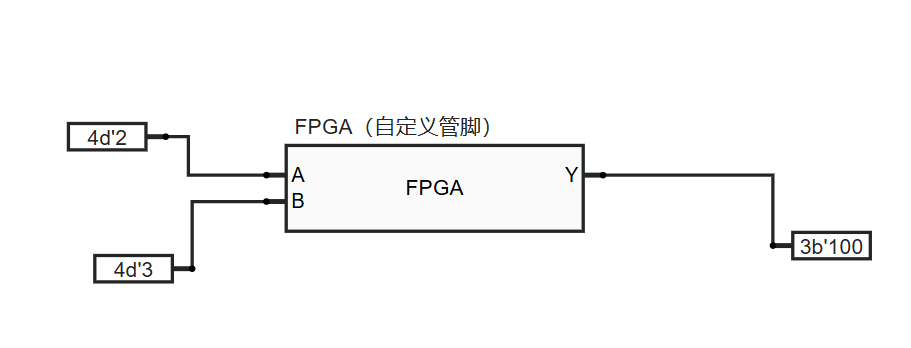
a>b时：



a=b时：



a<b时：



1.5.2时序逻辑电路中的同步四位二进制加计数器实验

实验器件简介：4位同步二进制加计数器74HC161，原理图如图3.1所示，该计数器具有两个高电平使能端CET和CEP，低电平异步清零端CR，低电平同步置位端PE，四位并行置数端D，四位数据输出端Q和一位进位信号TC。

图示

描述已自动生成

图3.1 74LVC161计数器

1.5.2.1代码注释以及Bin文件生成

数据分配器代码以及解释如下：

*01* ***module counter74HC161 (***

*02* ***input cep, cet, pe\_n, cp, cr\_n,*** *//输入端口声明 cr\_n为清零*

*03*  ***input [3:0] d,*** *//并行数据输入*

*04* ***output tc,*** *//进位输出*

*05* ***output reg [3:0] q*** *//数据输出端口及变量的数据 类型声明*

*06*  ***);***

*07* ***wire ce;*** *//中间变量声明*

*08* ***assign ce= cep & cet; //ce=1****时，计数器计数*

*09* ***assign tc = cet & q[3] & q[2] & q[1] & q[0];****//产生进位输出信号*

*10*

*11* ***always @(posedge cp, negedge cr\_n)***

*12* ***if ( !cr\_n ) q<= 4'b0000;*** *//实现异步清零功能*

*13* ***else if ( !pe\_n ) q <= d;***  *//pe\_n=0, 同步装入输入数据*

*14* ***else if ( ce ) q <= q+1'b1;***  *//加1计数*

*15*  ***else q <= q;*** *//输出保持不变*

*16*

*17* ***endmodule***

此计数器的功能为：

1. **异步清零（cr\_n为低电平）**：

不论其他输入信号的状态如何，只要cr\_n为低（0），计数器q会被立即清零到4'b0000。

1. **同步装载数据（pe\_n为低电平）**：

当pe\_n为低（0），并且cp有上升沿时，计数器q会被同步装载上输入的数据d。

1. **计数使能（ce为高电平）**：

ce是cep和cet的逻辑与结果，只有当cep和cet都为高（1）时，ce才为高。在ce为高的情况下，每当cp产生上升沿时，计数器q会加1。

1. **正常计数（无上述条件触发）**：

如果上述条件都不满足，计数器q保持当前值不变。

1. **进位输出（tc）**：

当cet为高电平，并且q的所有位（q[3]、q[2]、q[1]、q[0]）都为高（1），即计数器达到最大值9（二进制1001），tc会产生一个高电平的进位信号，表示计数器溢出。

**Bin文件生成：**在Vivado中创建一个新的FPGA工程，创建Verilog文件并编写上述逻辑代码。运行综合工具，检查代码是否有错误。在Vivado的“Flow Navigator”中，找到并点击“Generate Bitstream”按钮，勾选生成bin选项。然后设置FPGA的输入输出管脚约束。在Vivado中生成Bitstream即可生成bin文件

1.5.2.2仿真测试方法的说明

仿真代码如下：

*01 `timescale 1ns****/****1ps*

*02*

*03* ***module*** *counter74HC161\_tb****();***

*04* ***reg*** *cep****,*** *cet****,*** *pe\_n****,*** *cp****,*** *cr\_n****;*** *//输入端口声明*

*05* ***reg******[****3****:****0****]*** *d****;*** *//并行数据输入*

*06* ***wire*** *tc****;*** *//进位输出*

*07* ***wire******[****3****:****0****]*** *q****;*** *//数据输出端口及变量的数据类型声明*

*08*

*09 counter74HC161 TEST* ***(***

*10* ***.****cep****(****cep****),***

*11* ***.****cet****(****cet****),***

*12* ***.****pe\_n****(****pe\_n****),***

*13* ***.****cp****(****cp****),***

*14* ***.****cr\_n****(****cr\_n****),*** *//输入端口声明*

*15* ***.****d****(****d****),*** *//并行数据输入*

*16* ***.****tc****(****tc****),*** *//进位输出*

*17* ***.****q****(****q****)***

*18* ***);*** *//数据输出端口及变量的数据类型声明*

*19*

*20* ***initial******fork***

*21 cp* ***=*** *0****;***

*22 cr\_n* ***=*** *1****;******#****15 cr\_n* ***=*** *0****;******#****25 cr\_n* ***=*** *1****;***

*23 pe\_n* ***=*** *1****;******#****25 pe\_n* ***=*** *0****;******#****35 pe\_n* ***=****1****;***

*24 d* ***=*** *4'b1100****;***

*25 cep* ***=*** *0****;******#*** *28 cep* ***=*** *1****;******#*** *250 cep* ***=*** *0****;***

*26 cet* ***=*** *0****;******#*** *30 cet* ***=*** *1****;******#*** *235 cet* ***=*** *0****;***

*27* ***#****1000 $stop****;***

*28* ***join***

*29*

*30* ***always******#****10 cp* ***=******~****cp****;***

*31*

*32* ***endmodule***

代码解释如下：

**timescale 1ns/1ps：**

设置仿真的时间单位和时间精度。1ns（纳秒）是仿真的时间单位，1ps（皮秒）是仿真的时间精度。

**module counter74HC161\_tb();**：

定义了一个名为counter74HC161\_tb的测试平台模块。

**reg cep, cet, pe\_n, cp, cr\_n;**：

声明了5个寄存器类型的输入信号，分别对应74HC161的控制引脚。cep和cet是时钟使能端，pe\_n是预置输入端，cp是时钟输入端，cr\_n是复位输入端。

**reg [3:0] d;**：

声明了一个4位宽的寄存器d，用于并行数据输入。

**wire tc;**：

声明了一个线网类型的信号tc，用于接收计数器的进位输出。

**wire [3:0] q;**：

声明了一个4位宽的线网q，用于接收计数器的数据输出。

**counter74HC161 TEST (...);**：

实例化了counter74HC161模块，并将测试平台中的信号连接到该模块的对应引脚。

**initial fork ... join**：

这是一个初始块，用于在仿真开始时执行代码。fork和join关键字用于创建并行执行的代码块

**cp = 0;**：

初始化时钟信号cp为0。

**cr\_n = 1; #15 cr\_n = 0; #25 cr\_n = 1;**：

复位信号cr\_n在仿真开始后15ns设置为0（复位），在40ns后设置回1（解除复位）。

**pe\_n = 1; #25 pe\_n = 0; #35 pe\_n =1;**：

预置信号pe\_n在仿真开始后25ns设置为0（预置），在60ns后设置回1（取消预置）。

**d = 4'b1100;**：

并行数据输入d被初始化为二进制值1100。

**cep = 0; # 28 cep = 1; # 250 cep = 0;**：

时钟使能cep在仿真开始后28ns设置为1，然后在278ns设置回0。

**cet = 0; # 30 cet = 1; # 235 cet = 0;**：

时钟使能cet在仿真开始后30ns设置为1，然后在265ns设置回0。

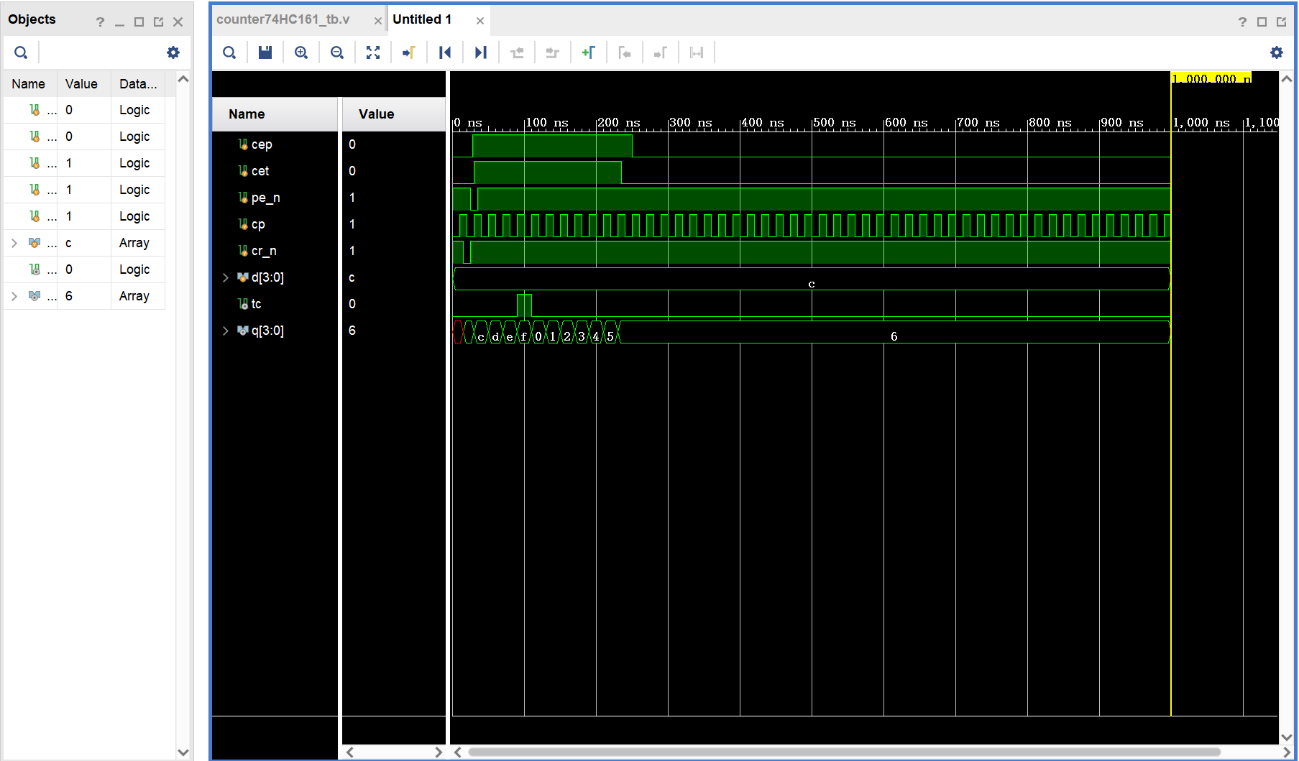
**#1000 $stop;**：

在仿真1000ns后停止仿真。

**always #10 cp = ~cp;**：

这是一个始终块，用于每隔10ns翻转时钟信号cp的状态，模拟时钟信号的周期性变化。

仿真结果如下：

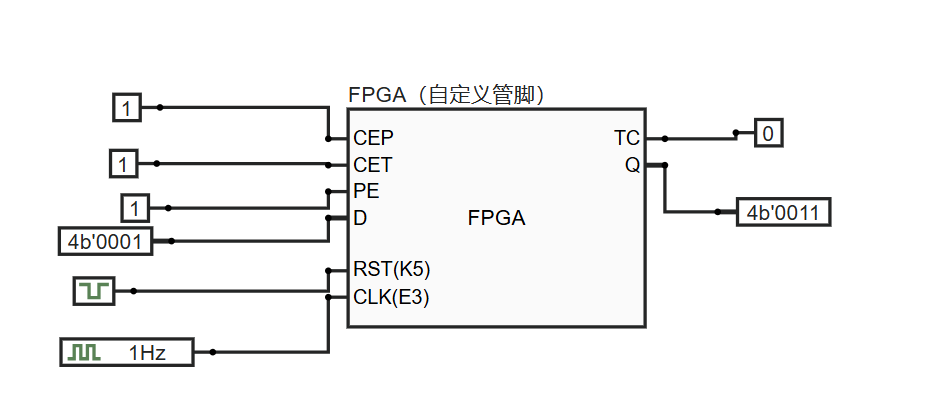


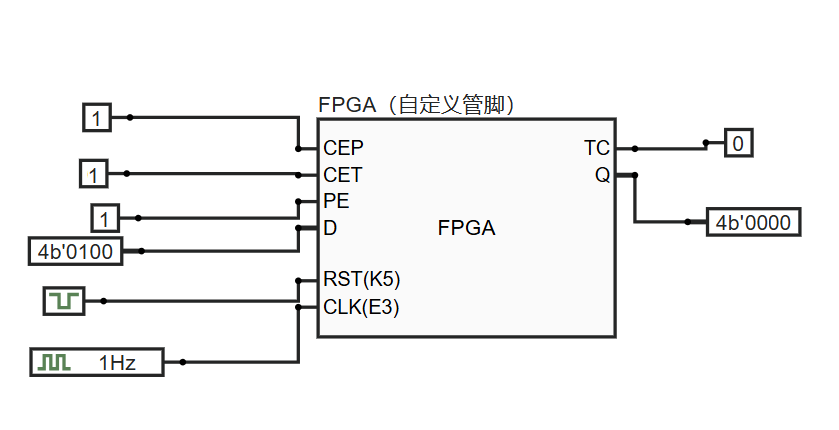
从仿真波形可以看出，如在15ns时刻，电路具有异步清零信号cr\_n（低电平有效），优先级最高；如在30ns时刻，当清零信号无效时，同步置位信号pe\_n（低电平有效），在时钟信号cp上升沿时将外部d的信号赋值给计数器状态信号q；如在30ns~90ns之间，在清零和置位信号均无效时，使能信号cep和cet高电平有效，时钟信号cp上升沿，计数器加1计数；如在90ns时刻，tc为四位二进制计数器进位信号，当cet、q3、q2、q1和q0均为高电平时，进位信号tc为1；如在250ns时刻，当cep或cet信号有一个为无效信号，计数器保持。

1.5.2.3远程云端实验方法的说明

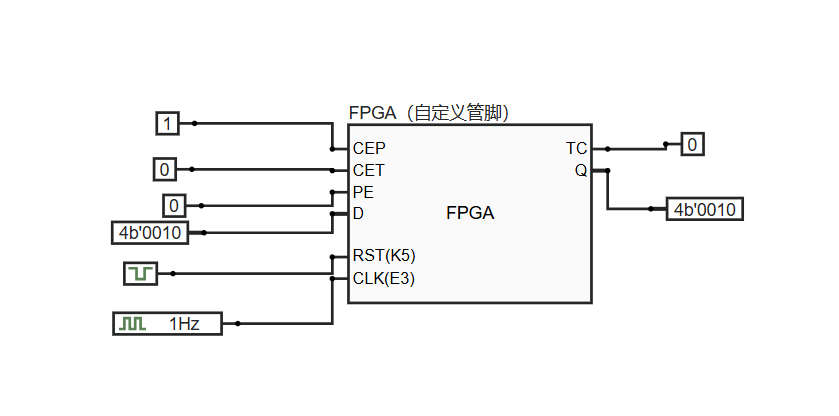
登录远程云端平台，确定板卡连接无误的情况下，进入实验面板，在右侧器件面板的逻辑器件一栏中，找到自定义FPGA、对应的输入输出以及时钟信号向实验面板图纸中央拖拽，同时将外围器件与逻辑器件的输入输出端口相连接，可以绘制如下图的电路实验模型图，位输入作为输入信号，位输出作为监测输出信号。这是一个带有时钟复位的7输入5输出的逻辑器件，3个1bit位输入作为输入信号CEP、CET、PE，1个4bit位输入信号D，1个时钟输入clk，一个下降沿复位RST，1个1bit位输出信号tc和1个4bit位输出Q。

利用bin文件进行FPGA烧结后调整输入数据，观测输出数据。

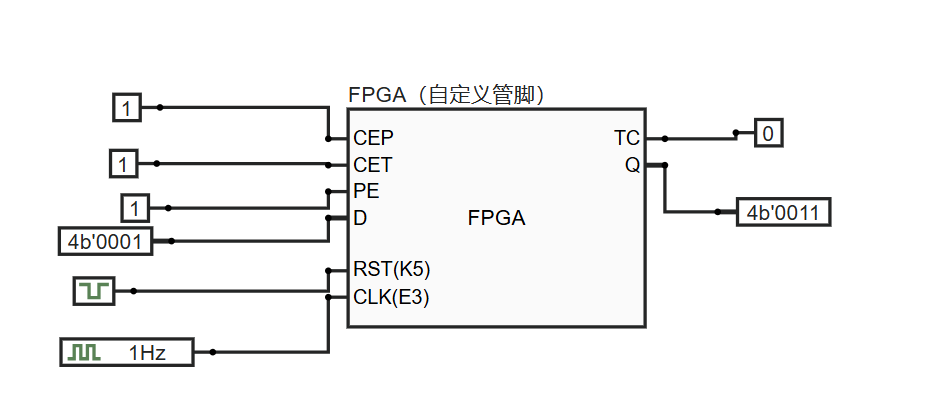
1. 异步清零（cr\_n为低电平，即RST为下降沿）：

↓↓

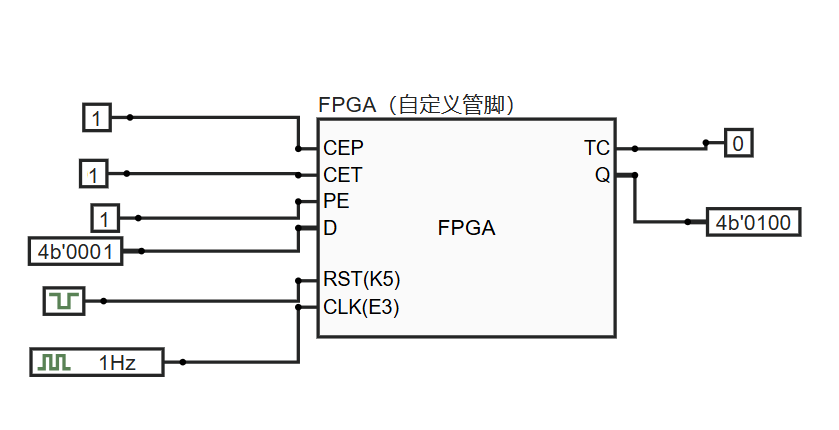
2. 同步装载数据（pe\_n为低电平）：



3. 计数使能（ce为高电平，即CEP,CET同时为高电平）：



↓↓



1.5.3存储器电路中的存储器FIFO实验

FIFO（First In First Out）即先进先出存储器，其特点是最先写进的数据最先从出口读出。本节将实现一个宽8位深32的同步FIFO，并通过远程实验平台验证。

1.5.3.1代码注释及bin文件生成

代码如下：

*01 `timescale 1ns****/****1ps*

*02*

*03* ***module*** *my\_fifo* ***(*** *clock****,*** *aclr****,*** *data****,*** *rdreq****,*** *wrreq****,*** *//input*

*04 q****,*** *empty****,*** *full****,*** *usedw //output*

*05* ***);***

*06* ***input*** *clock****;*** *//系统时钟*

*07* ***input*** *aclr****;*** *//系统复位，异步高有效*

*08* ***input******[****7****:****0****]*** *data****;*** *//写数据*

*09* ***input*** *rdreq****;*** *//读请求*

*10* ***input*** *wrreq****;*** *//写请求*

*11* ***output*** *empty****;*** *//空标志信号*

*12* ***output*** *full****;*** *//满标志信号*

*13* ***output******reg[****7****:****0****]*** *q****;*** *//读出数据*

*14* ***output******reg[****4****:****0****]*** *usedw****;*** *//可用资源量*

*15*

*16* ***reg[****7****:****0****]*** *fifo\_ram****[****0****:****31****];*** *//存储器矩阵*

*17* ***reg******[****4****:****0****]*** *rdaddr****;*** *//读地址指针*

*18* ***reg******[****4****:****0****]*** *wraddr****;*** *//写地址指针*

*19*

*20 //FIFO读操作*

*21* ***always******@******(posedge*** *clock****,******posedge*** *aclr****)******begin***

*22* ***if(****aclr****)******begin***

*23 q* ***<=*** *8'h00****;***

*24* ***end***

*25* ***else******if(*** *rdreq* ***==*** *1'b1* ***&&*** *empty* ***==*** *1'b0* ***)******begin***

*26 q* ***<=*** *fifo\_ram****[****rdaddr****];*** *//读出对应地址数据*

*27* ***end***

*28* ***else******if******(*** *rdreq* ***&&*** *wrreq* ***)******begin***

*29 q* ***<=*** *fifo\_ram****[****rdaddr****];*** *//读出对应地址数据*

*30* ***end***

*31* ***end***

*32*

*33 //FIFO写操作*

*34* ***always******@******(posedge*** *clock****,******posedge*** *aclr****)******begin***

*35* ***if(****aclr****)******begin***

*36* ***;***

*37* ***end***

*38* ***else******if(*** *wrreq* ***==*** *1'b1* ***&&*** *full* ***==*** *1'b0****)******begin***

*39 fifo\_ram****[****wraddr****]******<=*** *data****;*** *//将数据写入相应地址*

*40* ***end***

*41* ***else******if******(*** *rdreq* ***&&*** *wrreq* ***)******begin***

*42 fifo\_ram****[****wraddr****]******<=*** *data****;*** *//将数据写入相应地址*

*43* ***end***

*44* ***end***

*45 //指针操作*

*46* ***always******@******(posedge*** *clock****,******posedge*** *aclr****)******begin***

*47* ***if(****aclr****)******begin***

*48 rdaddr* ***<=*** *4'd0****;***

*49 wraddr* ***<=*** *4'd0****;***

*50* ***end***

*51* ***else******begin***

*52 wraddr* ***<=******((****wrreq* ***&&******!****full****)||(****wrreq* ***&&*** *rdreq****))******?*** *wraddr* ***+*** *1* ***:*** *wraddr****;***

*53 rdaddr* ***<=******((****rdreq* ***&&******!****empty****)||(****wrreq* ***&&*** *rdreq****))******?*** *rdaddr* ***+*** *1* ***:*** *rdaddr****;***

*54* ***end***

*55* ***end***

*56 //计算存储器资源*

*57* ***always******@******(posedge*** *clock****,******posedge*** *aclr****)******begin***

*58* ***if(****aclr****)******begin***

*59 usedw* ***<=*** *5'd0****;***

*60* ***end***

*61* ***else******begin***

*62* ***case******(******{****wrreq****,****rdreq****}******)***

*63 2'b00****:*** *usedw* ***<=*** *usedw****;***

*64 2'b01****:*** *usedw* ***<=******(****usedw* ***==*** *5'd0****)******?*** *5'b00000* ***:*** *usedw* ***-*** *1****;***

*65 2'b10****:*** *usedw* ***<=******(****usedw* ***==*** *5'd31****)******?*** *5'b11111* ***:*** *usedw* ***+*** *1****;***

*66 2'b11****:*** *usedw* ***<=*** *usedw****;***

*67* ***default:****usedw* ***<=*** *usedw****;***

*68* ***endcase***

*69* ***end***

*70* ***end***

*71*

*72 //空标志*

*73* ***assign*** *empty* ***=******(*** *usedw* ***==*** *5'd0* ***)?*** *1'b1****:*** *1'b0****;***

*74 //满标志*

*75* ***assign*** *full* ***=******(*** *usedw* ***==*** *5'd31* ***)?*** *1'b1****:*** *1'b0****;***

*76*

*77* ***endmodule***

代码解释：

*module my\_fifo ( clock, aclr, data, rdreq, wrreq,*

*q, empty, full, usedw );*

1.这里定义了FIFO的内部存储器fifo\_ram，它是一个8位宽、32字节深的数组。同时定义了两个5位的寄存器rdaddr和wraddr，分别用于跟踪读和写的地址指针。

*reg[7:0] fifo\_ram[0:31]; //存储器矩阵*

*reg [4:0] rdaddr; //读地址指针*

*reg [4:0] wraddr; //写地址指针*

2.这部分代码处理FIFO的读操作。在时钟上升沿或复位信号激活时，如果复位信号aclr为高，则清空输出数据q。如果读请求rdreq为高且FIFO不为空，则从fifo\_ram中读取数据到q。

*always @ (posedge clock, posedge aclr) begin*

*if(aclr) q <= 8'h00;*

*else if(rdreq == 1'b1 && empty == 1'b0) q <= fifo\_ram[rdaddr];*

*else if (rdreq && wrreq) q <= fifo\_ram[rdaddr];*

*end*

3.这部分代码处理FIFO的写操作。在时钟上升沿或复位信号激活时，如果复位信号aclr为高，则不执行任何操作。如果写请求wrreq为高且FIFO未满，则将输入数据data写入fifo\_ram。

*always @ (posedge clock, posedge aclr) begin*

*if(aclr);*

*else if(wrreq == 1'b1 && full == 1'b0) fifo\_ram[wraddr] <= data;*

*else if (rdreq && wrreq) fifo\_ram[wraddr] <= data;*

*end*

4.这部分代码更新读和写的地址指针。在复位时，两个指针都被重置为0。在正常操作中，根据读写请求和FIFO的满空状态，指针相应地增加。

*always @ (posedge clock, posedge aclr) begin*

*if(aclr) begin*

*rdaddr <= 4'd0;*

*wraddr <= 4'd0;*

*end*

*else begin*

*wraddr <= ((wrreq && !full)||(wrreq && rdreq)) ? wraddr + 1 : wraddr;*

*rdaddr <= ((rdreq && !empty)||(wrreq && rdreq)) ? rdaddr + 1 : rdaddr;*

*end*

*end*

5. 这部分代码更新读和写的地址指针。在复位时，两个指针都被重置为0。在正常操作中，根据读写请求和FIFO的满空状态，指针相应地增加。

*always @ (posedge clock, posedge aclr) begin*

*if(aclr) usedw <= 5'd0;*

*else case ({wrreq,rdreq})*

*2'b00: usedw <= usedw;*

*2'b01: usedw <= (usedw == 5'd0) ? 5'b00000 : usedw - 1;*

*2'b10: usedw <= (usedw == 5'd31) ? 5'b11111 : usedw + 1;*

*2'b11: usedw <= usedw;*

*default:usedw <= usedw;*

*endcase*

*end*

6.这部分代码计算FIFO中已使用的存储空间数量usedw。在复位时，usedw被清零。在正常操作中，根据读写请求的组合，usedw相应地增加或减少。

*assign empty = ( usedw == 5'd0 )? 1'b1: 1'b0;*

*assign full = ( usedw == 5'd31 )? 1'b1: 1'b0;*

7.这部分代码定义了FIFO的空和满标志信号。empty在usedw为0时为高，表示FIFO为空；full在usedw为31（最大值）时为高，表示FIFO已满。

**Bin文件生成：**在Vivado中创建一个新的FPGA工程，创建Verilog文件并编写上述逻辑代码。运行综合工具，检查代码是否有错误。在Vivado的“Flow Navigator”中，找到并点击“Generate Bitstream”按钮，勾选生成bin选项。然后设置FPGA的输入输出管脚约束。在Vivado中生成Bitstream即可生成bin文件

**烧写验证：将bin文件烧写到FPGA并进行功能验证。**

仿真代码如下：

*`timescale 1ns/1ps*

*module my\_fifo\_tb();*

*reg clock; //系统时钟*

*reg aclr; //系统复位，异步高有效*

*reg [7:0] data; //写数据*

*reg rdreq; //读请求*

*reg wrreq; //写请求*

*wire empty; //空标志信号*

*wire full; //满标志信号*

*wire [7:0] q; //读出数据*

*wire [4:0] usedw; //可用资源量*

*initial begin*

*clock = 0;//sys\_clk*

*aclr = 0;//reset*

*data = 8'd0;*

*rdreq = 0;*

*wrreq = 0;*

*#200*

*aclr = 1;//reset*

*#200*

*fork*

*aclr = 0;*

*wrreq = 1;*

*repeat(31) #40 data = data + 3;*

*join*

*wrreq = 0;*

*#100*

*rdreq = 1;*

*#1280*

*rdreq = 0;*

*#100*

*$stop;*

*end*

*my\_fifo my\_fifo\_inst(*

*.clock(clock),*

*.aclr(aclr),*

*.data(data),*

*.rdreq(rdreq),*

*.wrreq(wrreq), //input*

*.q(q),*

*.empty(empty),*

*.full(full),*

*.usedw(usedw)//output*

*);*

*always #20 clock = ~clock;*

*endmodule*

代码解释：

*`timescale 1ns/1ps*

*module my\_fifo\_tb();*

1.这行代码设置了仿真的时间单位为1纳秒（ns），时间精度为1皮秒（ps），并声明了名为my\_fifo\_tb的测试平台模块。

*reg clock; //系统时钟*

*reg aclr; //系统复位，异步高有效*

*reg [7:0] data; //写数据*

*reg rdreq; //读请求*

*reg wrreq; //写请求*

*wire empty; //空标志信号*

*wire full; //满标志信号*

*wire [7:0] q; //读出数据*

*wire [4:0] usedw; //可用资源量*

2.这部分代码声明了测试平台中使用的寄存器（reg）和线网（wire）类型信号。reg类型的信号可以被赋值，而wire类型的信号通常用于连接模块间的信号。

初始块定义了测试序列：

*initial begin*

*clock = 0;*

*aclr = 0;*

*data = 8'd0;*

*rdreq = 0;*

*wrreq = 0;*

*#200*

*aclr = 1;*

*#200*

*fork*

*aclr = 0;*

*wrreq = 1;*

*repeat(31) #40 data = data + 3;*

*join*

*wrreq = 0;*

*#100*

*rdreq = 1;*

*#1280*

*rdreq = 0;*

*#100*

*$stop;*

*End*

3.初始化所有控制信号为0。

200纳秒后，激活复位信号aclr。

再过200纳秒，复位结束，开始写操作：设置写请求wrreq为1，并循环3次， 每次循环将data增加3。

写操作完成后，关闭写请求wrreq。

100纳秒后，开始读操作：设置读请求rdreq为1。

1280纳秒后，关闭读请求rdreq。

再过100纳秒，停止仿真。

*my\_fifo my\_fifo\_inst(*

*.clock(clock),*

*.aclr(aclr),*

*.data(data),*

*.rdreq(rdreq),*

*.wrreq(wrreq), //input*

*.q(q),*

*.empty(empty),*

*.full(full),*

*.usedw(usedw)//output*

*);*

4. 这部分代码实例化了my\_fifo模块，并将测试平台中的信号连接到FIFO模块的对应端口。

*always #20 clock = ~clock;*

5.这个always块用于生成时钟信号。每隔20纳秒，时钟信号clock的状态会翻转一次，模拟一个周期为40纳秒的时钟信号。

*my\_fifo my\_fifo\_inst(*

*.clock(clock),*

*.aclr(aclr),*

*.data(data),*

*.rdreq(rdreq),*

*.wrreq(wrreq), //input*

*.q(q),*

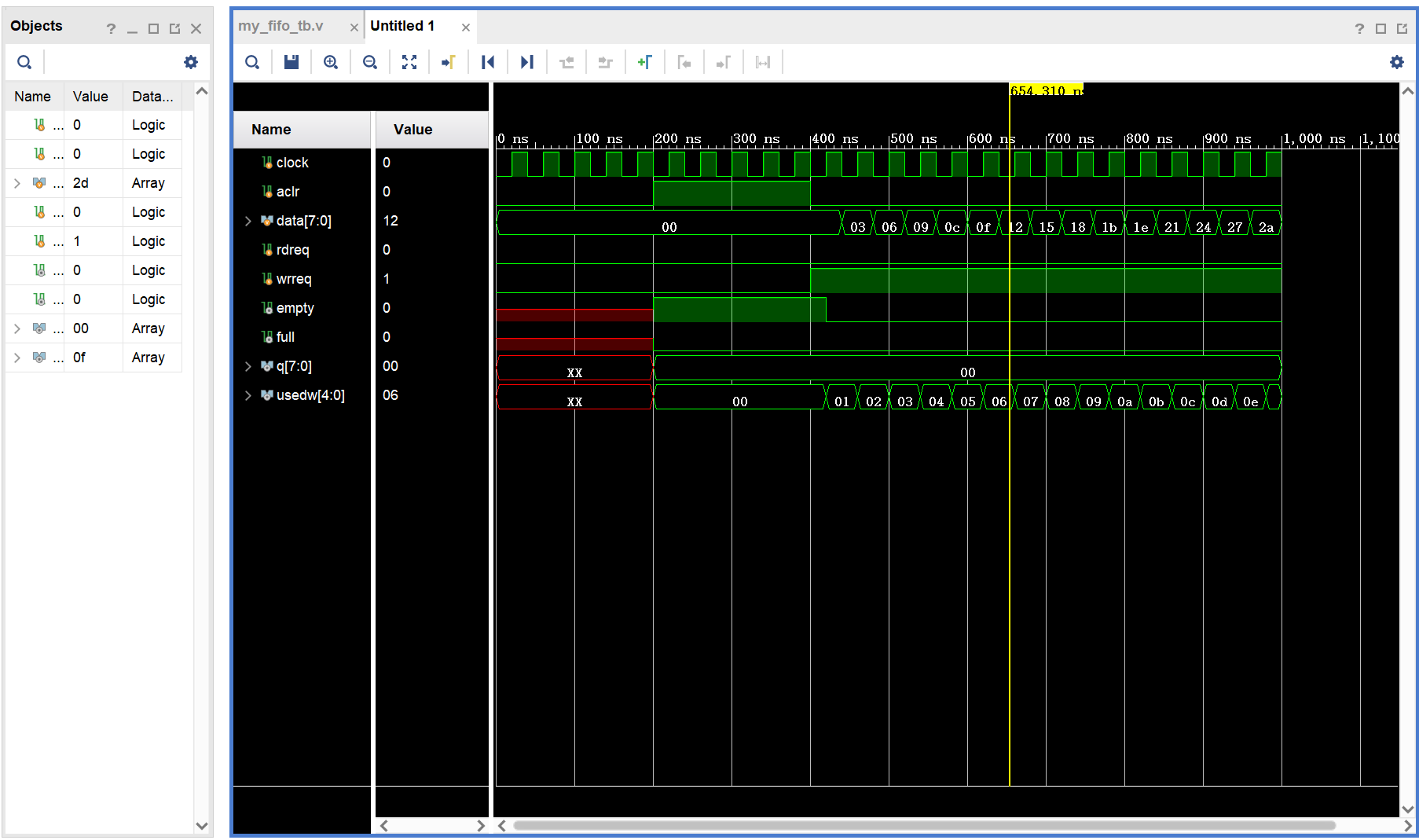
*.empty(empty),*

*.full(full),*

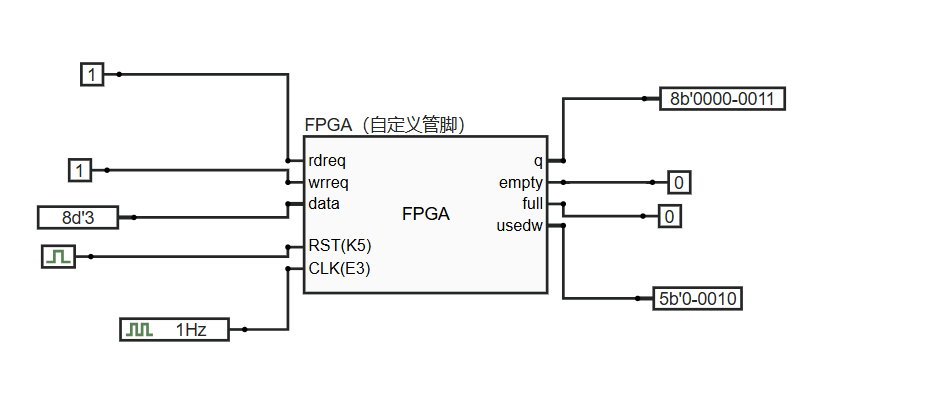
*.usedw(usedw)//output*

*);*

6.这部分代码实例化了my\_fifo模块，并将其端口连接到测试平台的相应信号。

仿真结果如下：

1.5.3.3远程云端实验方法的说明

登录远程云端平台，确定板卡连接无误的情况下，进入实验面板，在右侧器件面板的逻辑器件一栏中，找到自定义FPGA、对应的输入输出以及时钟信号向实验面板图纸中央拖拽，同时将外围器件与逻辑器件的输入输出端口相连接，可以绘制如下图的电路实验模型图，位输入作为输入信号，位输出作为监测输出信号。****

**主要功能和组件：**

**模块声明：**my\_fifo模块有输入端口clock（时钟信号）、aclr（异步复位信号）、data（8位数据输入）、rdreq（读请求信号）、wrreq（写请求信号），以及输出端口q（8位数据输出）、empty（空标志信号）、full（满标志信号）和usedw（表示FIFO中已使用的空间数量）。

**存储结构：**使用一个8位宽、32字节深的数组fifo\_ram作为FIFO的存储空间。

**读写操作：**

读操作：当有读请求（rdreq）且FIFO不为空（empty）时，从fifo\_ram中读取数据并赋值给输出q。

写操作：当有写请求（wrreq）且FIFO未满（full）时，将输入数据data写入fifo\_ram。

**地址指针：**使用两个5位的寄存器rdaddr和wraddr分别作为读和写的地址指针。

它们在每次读写操作后更新，以指向下一个要读写的位置。

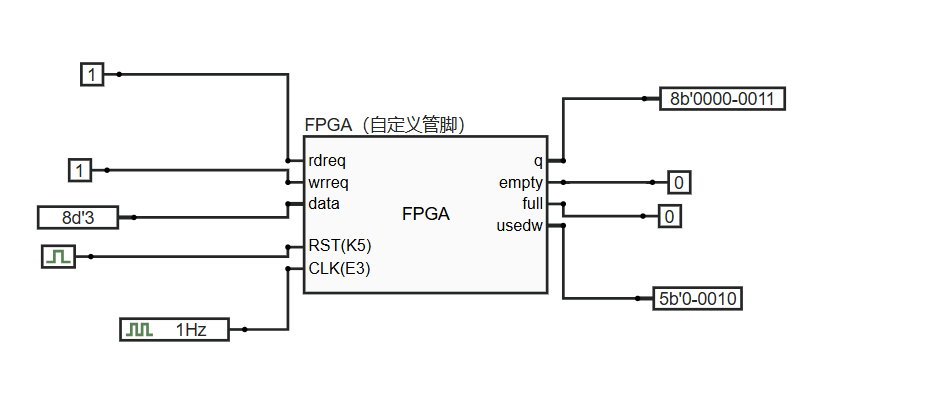
资源计算：usedw寄存器用于跟踪FIFO中已使用的空间数量。它在读写操作时更新，以反映FIFO的当前状态。

**空满标志：**empty和full标志通过比较usedw的值来确定FIFO是否为空或满。

**复位操作：**当复位信号aclr（RST）被激活时，FIFO的内容和所有控制寄存器

（包括q、rdaddr、wraddr和usedw）都被清零。

利用bin文件进行FPGA烧结后调整输入数据，观测输出数据。

****

1.6基于 Sobel 算子的边缘检测的原理

1.6.1原理概述

基于Sobel算子的边缘检测原理主要涉及对图像中亮度变化的识别，这一过程是通过分析图像中每个像素点及其周围像素的亮度差异来实现的。在某个像素点，梯度的方向指向亮度变化最快的方向，而梯度的大小则表示变化的剧烈程度。Sobel算子通过计算图像中每个像素点的梯度并将其与阈值进行比较来识别这些亮度的突变的区域，即可判断边缘。

1.6.2详细原理

1.6.2.1梯度概念

在图像处理中，边缘可以被定义为图像亮度变化最显著的地方。梯度是一个矢量，指向图像中亮度变化最大的方向，其大小表示变化的剧烈程度。因此，边缘检测本质上是寻找图像亮度梯度最大的区域。

1.6.2.2 Sobel算子

Sobel算子是一种用于边缘检测的离散微分算子，它通过计算图像亮度的空间梯度来识别边缘。Sobel算子包含两个3x3的卷积核（或滤波器），一个用于检测水平方向的边缘，另一个用于检测垂直方向的边缘。

1.6.2.3 水平和垂直梯度

**水平梯度（Gx）**：使用水平方向的Sobel模板与图像进行卷积，计算每个像素点在水平方向上的梯度幅度。水平模板如下所示：

**垂直梯度（Gy）**：使用垂直方向的Sobel模板与图像进行卷积，计算每个像素点在垂直方向上的梯度幅度。垂直模板如下所示：

1.6.2.4 梯度幅度和方向

**梯度幅度（G）**：通过计算水平和垂直梯度的平方和的平方根，可以得到每个像素点的梯度幅度，即：,这个值表示了在该点亮度变化的总量。

但此步骤运算较为复杂，涉及平方和开根运算，FPGA并不擅长此类运算，通常，为了提高效率，使用不开平方的近似值，但这样做会损失精度，迫不得已的时候可以如下这样：。

**梯度方向（θ）**：梯度的方向可以通过反正切函数计算得到，即：,这个角度表示了亮度变化最快的方向。

1.6.2.5 设置阈值并进行边缘检测

**非极大值抑制**：在计算出每个像素点的梯度幅度和方向后，Sobel算子还会进行非极大值抑制，以消除非边缘点。这一步骤通过比较当前像素点的梯度幅度与其相邻像素点在梯度方向上的梯度幅度来实现，只保留梯度幅度最大的点。

**双阈值检测**：为了进一步确定哪些梯度幅度表示真实的边缘，Sobel算子使用双阈值方法。设置一个高阈值和一个低阈值，高于高阈值的点被确定为强边缘，低于低阈值的点被排除，介于两者之间的点则根据它们是否与强边缘点相连来决定是否保留。

1.6.3 总结

Sobel算子的边缘检测原理基于图像亮度的空间梯度，通过计算水平和垂直方向上的梯度幅度和方向来识别边缘。这种方法简单有效，能够快速识别图像中的边缘，尽管它可能不如一些更复杂的边缘检测算子精确，但在许多应用中已经足够使用。

1.7 FPGA实现边缘检测图像预处理具有较好实时性的原因

Sobel算子作为一种广泛使用的边缘检测方法，其在FPGA平台上的实现显著提升了处理的实时性。FPGA的并行处理能力、流水线设计、并行I/O操作以及定制化硬件逻辑是其在图像预处理任务中表现卓越的关键因素。这些特性使得FPGA能够快速、高效地处理大量图像数据，满足实时或近实时处理的需求。以下是对FPGA在实现Sobel边缘检测中所展现的这些优势的详细探讨。

1.7.1 FPGA实现边缘检测的并行处理能力

在数字图像处理领域，边缘检测是识别图像中亮度变化明显区域的关键步骤，这些区域通常对应于物体的轮廓。Sobel算子作为一种流行的边缘检测方法，其核心在于计算图像亮度函数的梯度，从而识别出边缘。在传统的处理器上，这一过程通常是顺序执行的，而在FPGA上，由于其并行处理的特性，可以显著提高边缘检测的效率。

FPGA的并行处理能力主要体现在其能够同时处理多个数据点。以Sobel边缘检测为例，算法需要对图像的每个像素点应用3x3的模板进行卷积操作。在FPGA中，可以设计多个这样的模板处理单元并行工作，每个单元负责一部分图像区域的边缘检测，实现对整个图像的并行处理。这种并行处理方式使得FPGA在处理大规模图像数据时具有明显的速度优势。

在FPGA设计中，通过实例化移位寄存器模块，如shift\_register\_2taps，实现了两行数据的缓存。这种设计允许FPGA同时访问当前行和前一行的数据，为3x3模板的卷积操作提供了必要的数据。通过将两个移位寄存器级联，可以连续地保存两行数据，并且能够通过taps1x和taps0x端口访问这些数据。这种行缓存机制是并行处理的基础，因为它允许算法在处理当前像素时，同时为下一个像素准备数据。

在Sobel边缘检测模块sobel中，定义了多个寄存器用于存储3x3矩阵的像素值。这些寄存器的更新和梯度的计算都是在时钟信号的每个上升沿同时进行的，这体现了FPGA的并行处理特性。通过这种方式，FPGA能够快速地对每个像素块进行梯度计算，而不需要等待前一个像素的处理完成。

1.7.2流水线设计

流水线设计是FPGA提升处理速度的关键技术之一。在FPGA中，流水线技术通过将复杂的计算任务分解成多个阶段，每个阶段可以并行执行，从而提高处理速度。例如，在Sobel边缘检测算法中，梯度的计算可以分解为多个步骤：首先计算水平和垂直方向的梯度（Gx和Gy），然后计算梯度的幅度G，最后将G与阈值比较以输出二值图像结果。通过在FPGA中实现流水线，这些步骤可以并行进行，每个步骤由不同的硬件逻辑单元执行，从而显著减少总体的计算延迟。这种设计使得FPGA在处理图像数据时，能够以极高的效率并行处理多个像素点的边缘检测，实现实时或近实时的图像处理。

1.7.3并行I/O操作

FPGA的并行I/O操作能力是其另一个显著优势。FPGA的I/O单元可以同时进行数据的输入和输出操作，支持高速数据传输。这使得FPGA在处理外部设备的数据交互时也能保持高效率。在图像处理中，这意味着FPGA可以同时从传感器接收图像数据，并输出处理后的结果到显示设备或其他处理单元，而不需要等待I/O操作的完成。这种并行I/O操作减少了数据传输的时间延迟，提高了整体的实时性。

1.7.4定制化硬件逻辑

FPGA的定制化硬件逻辑是其核心优势之一。FPGA允许设计者根据特定算法的需求设计硬件逻辑，这意味着它可以针对这些任务进行优化，从而实现更高的效率。在Sobel边缘检测中，FPGA可以被编程为实现特定的梯度计算逻辑，这种硬件级别的优化可以显著提高算法的执行速度和性能。与传统的通用处理器相比，FPGA的定制逻辑可以减少不必要的计算和内存访问，提高性能功耗比。此外，FPGA的可编程性还允许系统的功能通过重新编程FPGA来进行升级，而不需要物理更换硬件，这为算法的迭代和优化提供了极大的灵活性。

2. 致谢

感谢老师这段时间的培养教育，是您拓展丰富，形式多样的授课让我接触到了电子技术的多彩领域，安排这样用心良苦的课程设计也让我学到了很多实践知识。同时也要感谢助教在我有疑惑时的耐心解答，才能让这次电子技术课程设计顺利完成。