1. 数据结构

DFS, BFS

最小生成树：Prim algorithm, Kruskal algorithm

关键路径：DFS

最短路径：Dijkstra algorithm, Floyd algorithm

1. 离散数学：

Graph isomorphism: NAUTY, NP problem.

Euler Circuits: Chinese postman problem

Hamilton Circuits: Traveling salesperson problem (approximation algorithm)

Graph coloring

Expander: PageRank.

3． 消息式内存

第一步：Riscv simulator（西数）, 调研下哪个好用，可以往上加IP核模拟器（互联，消息式内存），有可能毕业论文只能做到第一步。

第二步：在板卡上实现某个应用（PageRank或其他），也可能用上增加特色指令。

第三步：

1. 题目

Chisel 实现一个 多个riscv处理器核并发处理图计算（PageRank），消息式内存（local & remote memory），总线为DOCE协议。进一步工作：对memory controler 进行优化。副产品：提供一个riscv核的simulator。

1. 疑问

需要做64位的吗？

1. HCPF:

· 只是加了几个数组，访问频率少的数据才可以通过旁路

· 需要修改软件，透明度不够

修改意见：

· 增加profile，统计热词（调查一下pagerank算法）

· 修改测试环境，修改AXI接口为SRAM接口

· 探索Type的作用

· //调度对不同bank，不同core的访问

1. DOCE:

· 每个node内存地址和内存空间怎么安排

· 哪些部分是实现在PL上

· 如何修改MAC层和物理层的

· 如何在X86上部署DOCE

· cache coherence? DEOI会解决吗

每个node可以共享的区域已经提前确定，这个区域，可以读share，不能写share，无cache coherence

修改意见：

增加share状态

1. 想法

* 刚开始为了设计的透明度，我计划用读操作代替HCPF的写操作，此时就要用乱序的核。但ROB和LS 模块的大小是固定的，不可能支持高并发的读操作，所以还是要使用写操作。这样，乱序和顺序就没有影响了，鉴于rocket core有更好的支持，决定选择七月更新的rocket core。
* 如果为写操作，高并发的请求就没必要通过cache，可以将HCPF作为一个IO设备加入rocket chip上
* 对于图应用，在struct粒度上的访存是随机的，HCPF应该支持不同size的访存。我们可以让size由用户决定或者根据profile来决定

1. 在IJ上搭建chisel3环境

* 参考：<http://www.cnblogs.com/wjcdx/p/9595516.html>
* Scala 要选择2.11版本
* 亦可将编译好的chisel.jar文件作为库

1. Chisel笔记

* fliped interface 不能和正常interface连接
* Queue的reset方法：



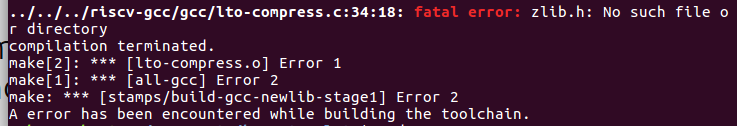
* 生成CPP的函数verilogToCpp（还不会用）
* Higher function: scan, map, fold, reduce

1. 搭建boom2-template: <https://github.com/riscv-boom/boom-template>

在这一步完成了编译链及一系列编程工具的安装，verilator的使用，C++ emulator的生成

使用BoomConfig

* 安装依赖：texinfo, gawk, device-tree-compiler, bison, flex, libexpat1-dev, libncurses5-dev, babeltrace, libmpc-dev, zlib1g-dev, autoconf, automake, libtool
* sudo apt-get install autoconf automake autotools-dev curl device-tree-compiler libmpc-dev libmpfr-dev libgmp-dev gawk build-essential bison flex texinfo gperf libtool patchutils bc zlib1g-dev libexpat1-dev pkg-config
* libusb-1.x not found
* jdk 8u191 : openjdk-8-jdk !!! do not use headless
* 出现错误：



1. 将boom2部署在zynq上

<https://github.com/ucb-bar/fpga-zynq>

<https://github.com/donggyukim/fpga-zynq>

rocket chip 版本：boom2晚于ucb-bar晚于donggyukim

donggyukim使用boom1，boomzynqconfig





ucb-bar使用ZynqConfig

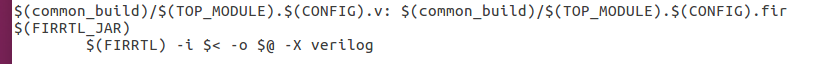






ECC (at least as far as it concerns PC users) is used to detect errors that occur in DRAM due to environmental effects

* 如何生成verilog



* 如何生成bit以及constrain文件

需要板卡的定义以及顶层文件（rocket\_wrapper.v）

在common文件夹中有顶层文件

在相应板卡文件中定义了板卡信息，调用common 文件夹中的Makefrage文件生成bit.bin：make fpga-images-zybo/boot.bin

* FPGA内部资源如何调用（内存和接口）

该部分在TCL中已定义，在zynq\_wrapper.v文件中调用了这些的接口

* 跑操作系统的流程

亦可使用riscv-fesvr进行arm核与riscv 核之间的通信

* 存在的问题

Boom1的上版文件调用的库不在最新的rocket chip中

* 在top文件中，chip有一个adapter模块，接受PS传来的AXI控制信号，转化成serial信号，传给target。我怀疑target就是内核。因为target向主机打印东西的时候，是serial信号？
* 尝试用ucb bar的rocket chip以及顶层文件加入boom2，根据donggyukim的代码进行修改:
  + 尝试了boom2.1.1对应的rocket chip和相应模块的版本，可能因为chisel版本过高，导致与project\_zynq一样的代码（TOP Configs）在该版本下不能跑过。如果把chisel3版本调为project\_zynq的版本，无法编译rocket chip
  + 尝试boom2.1.0(此版本的chisel3和project\_zynq一致)，还是出现option[T]非T的报错。从option[t]中得到t可使用get或者map方法
* 目前可用的内核是boom1与rocket（已调整到最新版本）
* 乱序的CU并没有用，倾向于用rocket
* delete verilog\_srcs in Makefrag and zynq\_rocketchip.tcl
* error:

Failed to construct terminal; falling back to unsupported

java.lang.NumberFormatException: For input string: "0x100"

solution: use sbt 0.13.16. It miss a packet

1. 如何在rocket chip上增加自己的模块

<https://github.com/ucb-bar/project-template>

* 在哪里增加，有何限制

增加mmio映射的寄存器：

[https://github.com/ucb-bar/project-template#adding-an-mmio-peripheral](https://github.com/ucb-bar/project-template%23adding-an-mmio-peripheral)

三种选择，加协处理器（每个core最多加三个，不推荐），作为MMIO的IO设备，直接修改boom代码

* 增加后如何编译

通过java将boom重新打包成jar文件，firrtl文件verilog文件

生成C++ simulator: Verilogverilator处理 C++

* 怎么debug

Using tester method

1. **poke** to set input port and state values
2. **step** to execute the circuit one time unit
3. **peek** to read port and state values, and
4. **expect** to compare peeked circuit values to expected arguments

* 测试方案

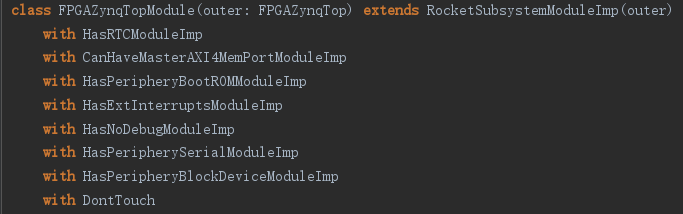
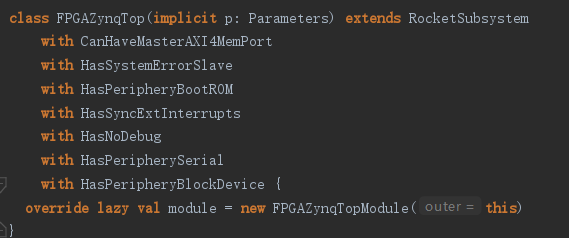
1. 研究rocket代码

* 框架
  + 在顶层文件中创建了core和adapter模块



Core通过adapter将信息写到arm核内，进而打印到屏幕上，adapter通过tile link与core相连

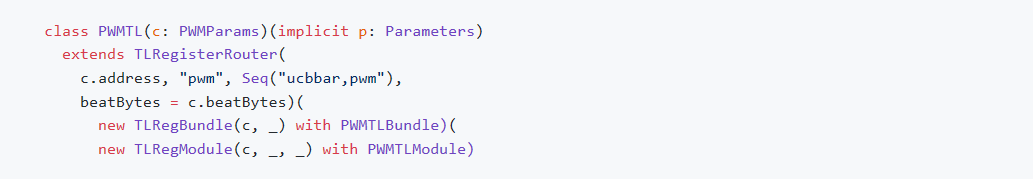
* + Core module和moduleImp模块，包含AXI接口，block device controller，Boot Rom（包含BTD，系统从0x10040启动），外部中断



* + HasPeripherySerial与HasPeripheryBlockDevice，HasPeripheryBootROM
* 如何增加mmio device
  + 通过TileLink 与core相连，调用TLRegisterRouter
  + 需要a parameter case class for the configuration settings, a bundle trait with the extra top-level ports, and a module implementation containing the actual RTL



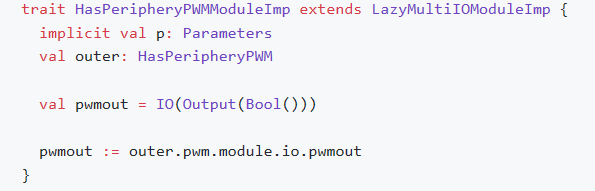
* + 最后扩展reg router，将参数，接口，模块定义作为参数传入



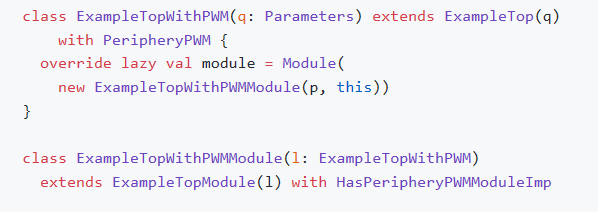
* + 将模块挂载在板卡上
    - LazyModule trait runs setup code that must execute before all the hardware gets elaborated
    - involves connecting the peripheral's TileLink node to the MMIO crossbar



* + The module implementation trait is where we instantiate our PWM module and connect it to the rest of the SoC。ModuleImp模块就是负责外围连线



* + mix our traits into the system as a whole



* 增加DMA
  + 
  + The ExtBundle contains the signals we connect off-chip that we get data from
  + The DMADevice also has a Tilelink client port that we connect into the L1-L2 crossbar through the front-side buffer (fsb)
  + The sourceId variable given in the TLClientNode instantiation determines the range of ids that can be used in acquire messages from this device. Since we specified [0, 1) as our range, only the ID 0 can be used.
* 内存空间的架构
* 控制寄存器内存地址设为unmapped，测试程序放在unmapped地址空间
* memory如何增加新的接口，或者增加axi interconnection
* 如何嵌入DOCE模块，verilog代码和chisel混合编译

1. 设计并完成支持不同访存size的HCPF模块，并debug，跑PageRank算法（寒假前完成）
2. 性能测试
3. 嵌入DOCE模块，在两个板卡上跑PageRank算法
4. 无HCPF与有HCPF进行性能比较（中期报告前完成）
5. 做优化（想办法减少模块间的交互，例如将状态寄存器上移，因为实际情况下，访问HCRF可能周期很长。或者让硬件猜访存的粒度），跑PageRank算法
6. 性能比较

四月初进行中期检查

五月初（五一结束后）第一轮查重

三天后进行第二次查重

查重通过后进行评阅

五月10+号进行答辩（讲10min，问5min）

可以先提供开题报告的题目和关键词

**计划表（自己看，不交）**

现在 - 2019/1/31 完成访存加速器的代码编辑与调试

2019/1/31 - 2019/2/28 完成PageRank软件代码并测试性能

2019/2/28 - 2019/3/31 在Rocket chip上启动Linux系统，并加入DoCE模块

2019/3/31 - 2019/4/30 测试性能并完成毕业论文