基于紫光同创 PGL22G 的 RISC-V 处理器设计——UniRISCV

王韬；陈岩；陈子昂

第一部分　设计概述

1.1 设计目的

本系统的设计目的是在紫光同创 PGL22G 开发板上实现基于 RISC-V 指令集架构的软核处理器，充分利用板卡硬件资源以提高 CPU 运行性能，并且在软件生态上对设计进行适配和支持。

1.2 应用领域

UniRISCV 是一款面向 FPGA，基于 Verilog 语言设计的 RISC-V 软核，可用于信号处理、SOPC 系统控制等领域。一方面，UniRISCV 软核相比于 CPU 硬核成本更低，可移植性更好；另一方面，UniRISCV 软核能以较小的资源消耗为代价帮助 FPGA 实现复杂的逻辑控制和计算，提高开发效率，降低开发成本。

1.3 主要技术特点

（1）32 位数据和指令宽度。

（2）支持 RV32IMZicsr 指令集，通过 RISC-V 指令兼容性测试。

（3）可以运行 C 语言程序。

（4）支持 AXI4 总线和中断。

（5）支持通过串口更新程序。

（6）容易移植到任何 FPGA 平台。

1.4 UniRISCV 关键性能指标

（1）Coremark: 2.50 Coremark/Mhz (紫光 PGL22L 开发板，50Mhz)

2.91 Coremark/Mhz (参考设计，Arty A7 开发板，25Mhz)

（2）支持 RISC-V 整数(I)、乘除(M)和 CSR 指令(Z)扩展(RV32IMZicsr)

（3）支持用户、管理和机器模式权限级别。

（4）基本的 MMU 支持。

（5）支持指令/数据缓存、AXI 总线接口或紧耦合内存。

1.5 主要创新点

（1）五级可配置流水线，支持 MMU 和 CSR。

（2）支持嵌入式软件开发平台 SDK，便于开发。

（3）模块化实现，各模块功能划分合理，逻辑清晰。

（4）提供 Verilog 和 System-C / Verilator 的 Testbench。

第二部分　系统组成及功能说明

2.1 整体介绍

本系统名称为 UniRISCV，分为硬件和软件两部分。

硬件方面，SoC 基于 UltraEmbedded 的开源 RISC-V 内核 riscv[1]设计的，分为内核和外设两部分。内核由取值、译码、执行等模块构成，实现了基本的操作，内核支持指令和数据 Cache；提供 64KB 的片上缓存 TCM，用于存储裸机代码或者存放 bootloader 启动代码，内核与 TCM 之间通过 AXI4 高速片上总线互连；提供 UART，SPI，TIMER，GPIO 等多种外设，内核与外设通过 AXI-Lite 高速片上总线互连；提供中断管脚用于外部中断控制器的输入。

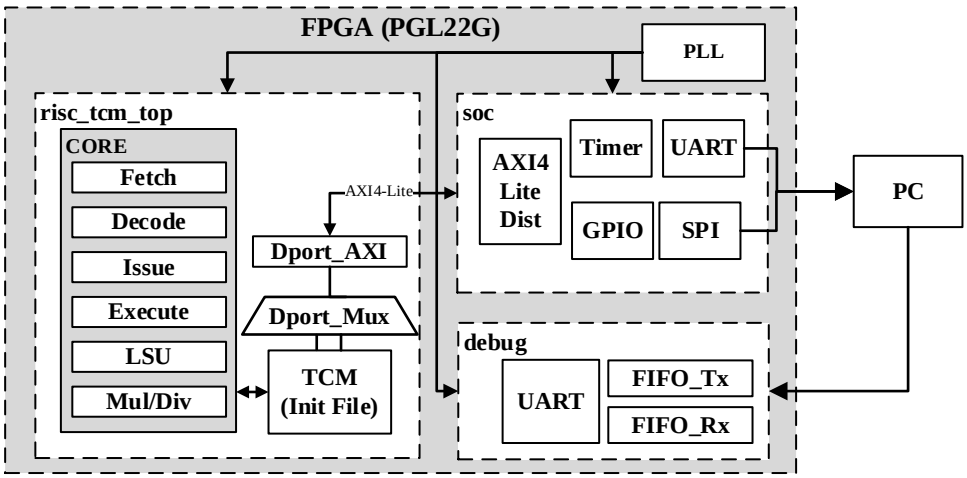

图 2.1 系统硬件构成

软件方面，UniRISCV 具有硬件对应的软件接口，提供了标准库函数的实现。初次之外，提供了 UART 和 GPIO 的软件驱动函数和 Coremark 跑分程序的移植，。

给出芯片或系统整体框图，各子模块标注清楚，并进行整体的文字说明，需要表达出各模块之间的关系。

2.2 各模块介绍

（1）内核模块 riscv_tcm_top

内核模块包括两部分：核心处理单元 CPU 和紧耦合内存 TCM，其中 CPU 根据参考开源设计文档所示，分为 IF、ID、EX、MEM、WB 五级流水线；TCM 用于高速内存访问和缓存，用于存储程序和数据，基地址为 0x0000_0000，大小为 64K。

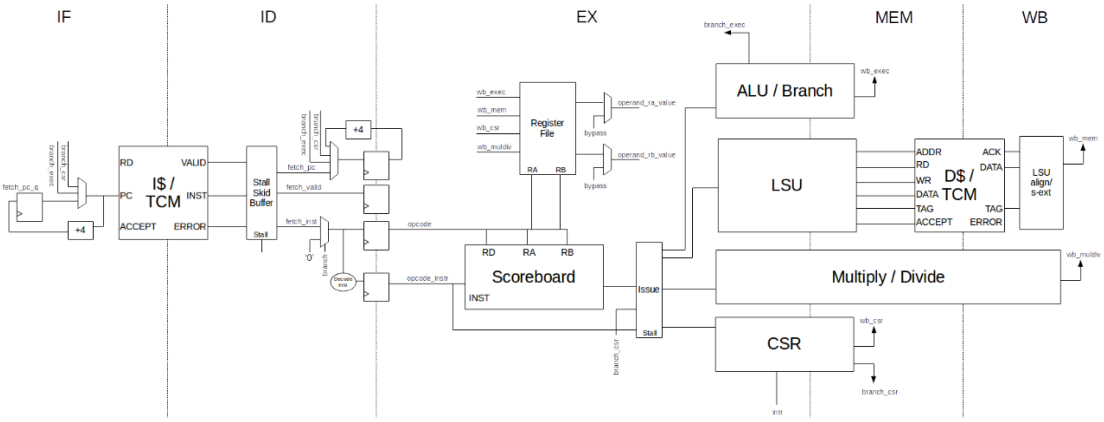

图 2.2 riscv 内核结构

（2）外设模块 core_soc

该模块用于 AXI-Lite 总线仲裁和外设的连接。其中，外设模块有中断控制器、串口、SPI、GPIO 等。axilite_dist 模块根据总线地址的[26:24]位的数据，对个外设的访问进行仲裁。

表 1 外设地址与对应文件

外设	基地址	对应文件
UART	0x92000000	uart_lite.v
SPI	0x93000000	spi_lite.v
GPIO	0x94000000	gpio.v
TIMER	0x91000000	timer.v
IRQ_CTRL	0x90000000	irq_ctrl.v

（3）调试模块 dbg_bridge

dbg_bridge 模块实质上是一个串口模块，通过 AXI4 总线连接到内核中的 TCM，用于在线下载和更新程序。

第三部分　完成情况及性能参数

3.1 完成情况

Ultraembedded 的参考设计提供了两种内核，分别为 riscv 和 riscv_tcm。前者具有两个 AXI4 Master 接口分别用于访问指令和访问数据及外设，后者使用一个 AXI4-Lite 接口用于连接外设，还提供了一个 AXI4 Slave 接口用于将程序提供串口在线下载到 TCM 中。为了初期验证阶段的简洁性，本系统使用 riscv_tcm 作为原型，将启动程序直接烧录在 TCM 中执行指令。同时挂载 UART，GPIO，SPI 以及 TIMER 等外设模块，用于系统验证。

UniRISCV 目前具备并已经完成测试的硬件功能：串口、GPIO、Coremark 跑分测试。其中，串口模块波特率可以配置，实验中选取的参数为 115200。串口实验输出如下图所示，软件界面为 C 程序 printf("Hello RISC-V!\n"); printf("Pango PGL22G\n\n");烧写硬件完成后，点击复位按键，处理器通过 AXI4-Lite 总线将数据写入串口模块 UART，再通过板子上的 Micro USB 端口输出。

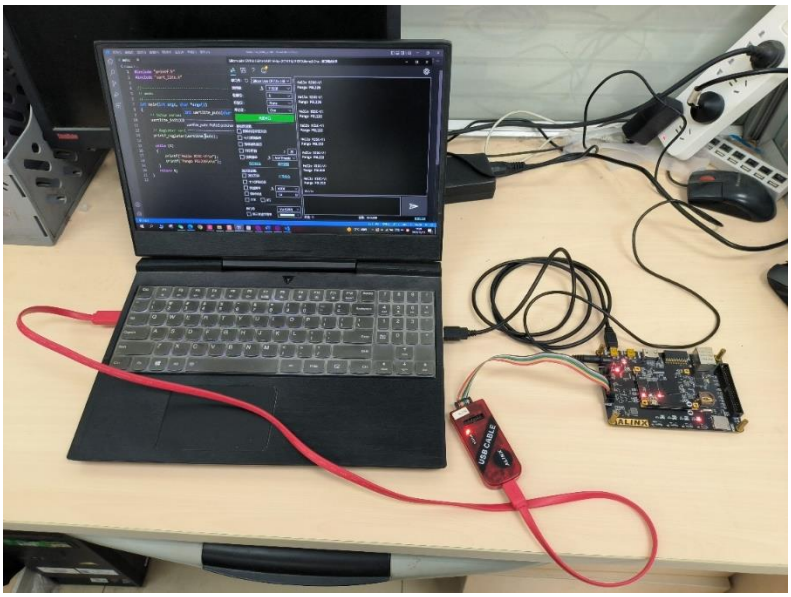

图 3.1 串口输出

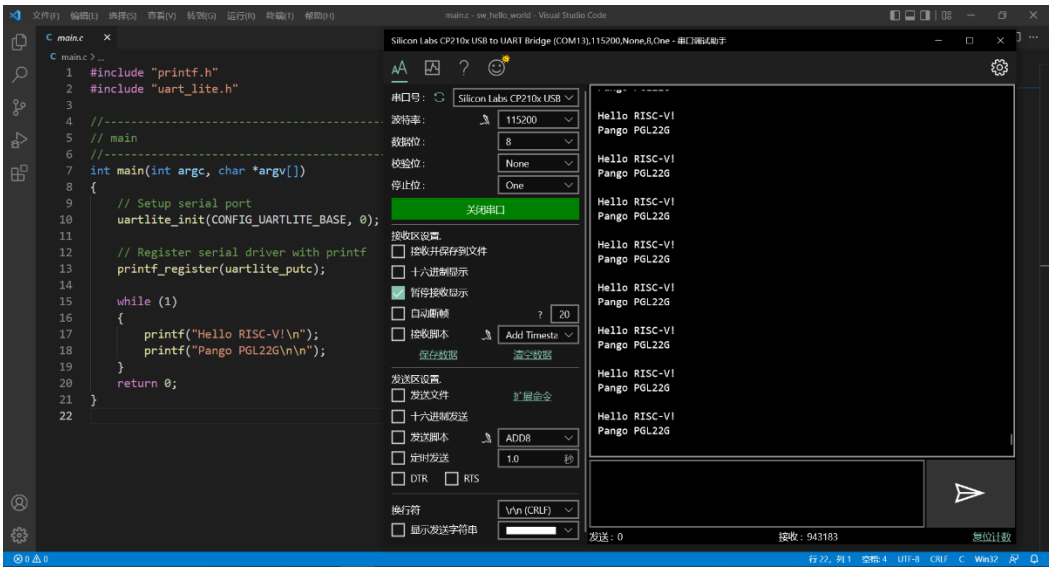

图 3.2 PC 端串口接收界面

软件上，为了便于嵌入式软件程序的编写、下载和调试，提供了基于 Eclipse 的 SDK。只需要恰当新建项目，配置 build 工具和 riscv-toolchain 编译工具链的路径，便可以交互式地利用 SDK 实现 C 语言软件程序的编译，若再配置 OpenOCD 和对应的硬件接口，可以实现 JTAG 调试功能。该 SDK 参考了其他器件的 RISC-V 设计的软件开发平台，通过修改 Eclipse 平台的 Java 插件实现。

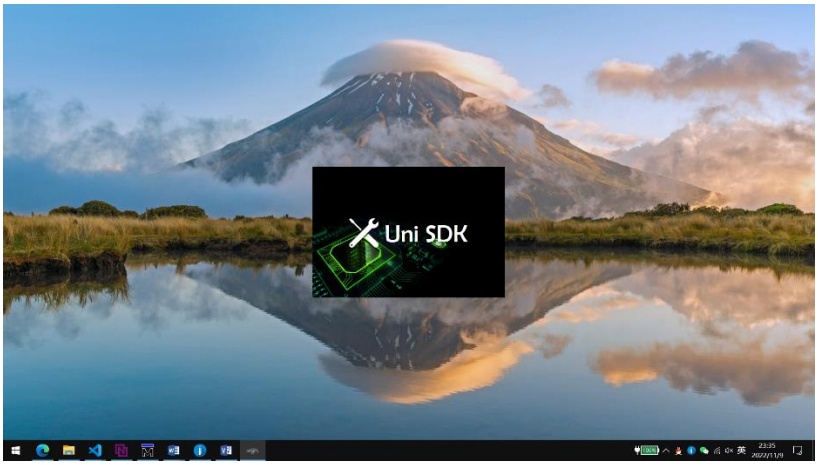

图 3.3 Uni SDK 启动界面

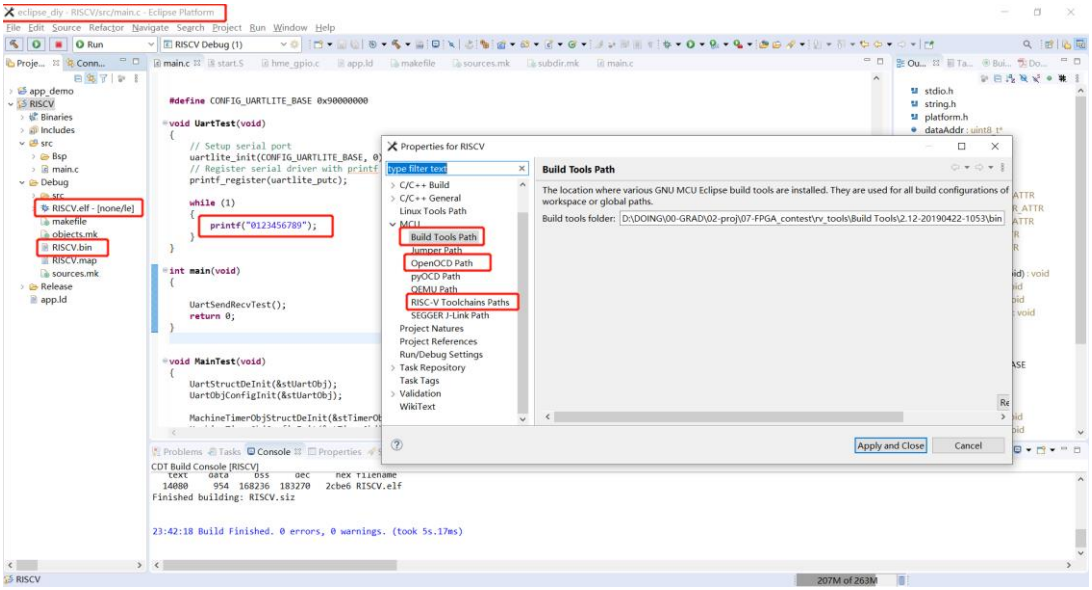

图 3.4 Uni SDK 工作界面

3.2 性能参数

（1）Coremark 跑分

目前 UniRISCV 在 PGL22G 的 FPGA 平台（时钟 50MHz）上运行 CoreMark 跑分程序的结果为 2.50CoreMark/MHz。具体输出内容如下图所示，设置 Iteration 为 2000 次，共运行了 16s，得到每秒迭代次数 Iteration/Sec：125，最终的 Coremark 跑分结果为：125/50 = 2.50 Coremark/Mhz。

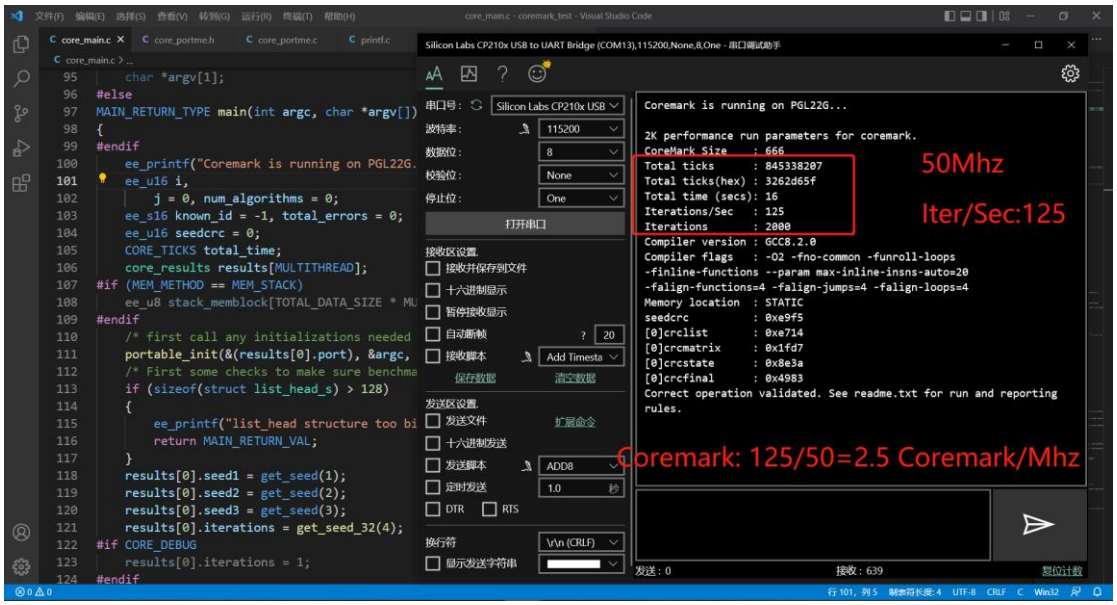

图 3.5 Coremark 跑分测试

（2）FPGA 资源利用量

资源消耗方面，UniRISCV 处理器设计在 PGL22G 上使用了 6844 个 LUT，

4244 个触发器和 32 个 DRM18K 的 RAM。组合和时序逻辑资源利用率均在 40%以下，面积上有着很好的表现，无时序违例。

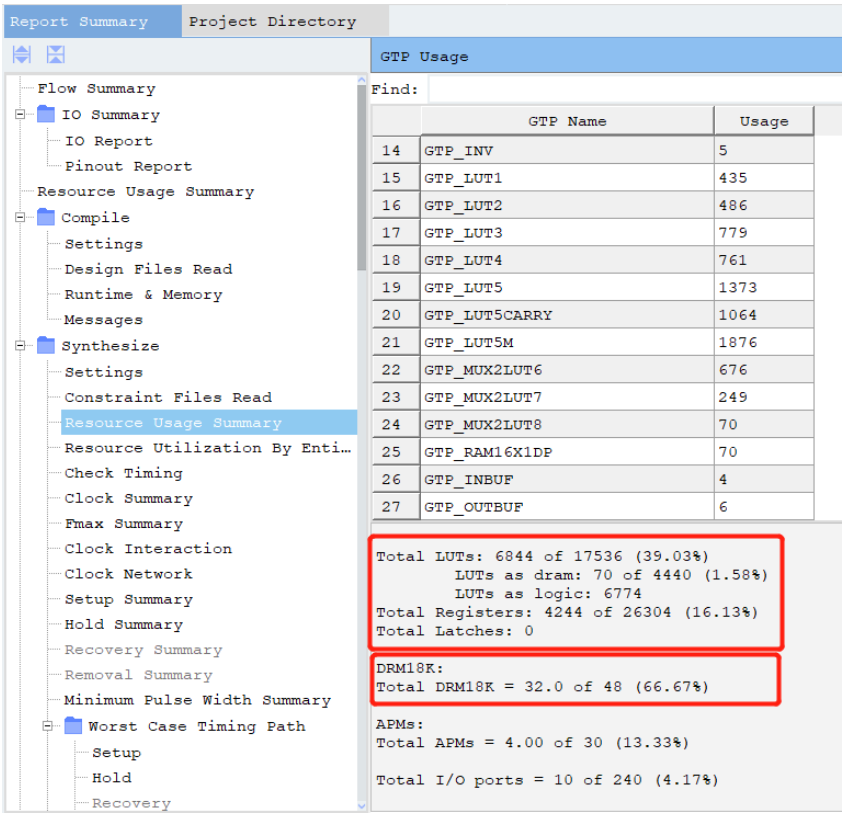

图 3.6 FPGA 资源使用情况

第四部分　总结

4.1 系统总结

目前，UniRISCV 在 PGL22G 平台上实现了基本的 MCU 功能：C 程序编译和运行、串口和 Coremark 跑分，Coremark 实现 2.50Mhz 的跑分结果，性能较为良好。

4.2 可扩展之处

（1）Ultraembedded 提供了升级版内核设计 biriscv，支持 dual issue，支持 MMU，根据提供的设计文档跑分达到 Coremark: 4.1 CoreMark/MHz。如果有机会继续开发，可以考虑替换为性能更高的 biriscv。

（2）在紫光 PGL22G 开发板上移植 biriscv 时，可以成功综合，资源利用率低于 60%，但是在 Place&Route 阶段不收敛，该问题有待解决。

4.3 心得体会

（1）紫光 Pango Design Suite【可能】不支持 RAM 初始化语句$readmemh。

具体问题：例化 RTL 设计的 RAM 用于存储二进制程序和数据，behavior 仿真可以正常读出，Post Synthesis 仿真时，数据读出错误，二者不匹配。

最终解决方案：使用紫光自带的 DRAM IP，用 init file 读取 dat 文件实现。

（2）开源的内核设计数不胜数，移植案例也有很多，本次比赛在开源内核的比较和选择上耗费了大多数精力，实际功能的调试和实现时间十分有限，所以只实现了基本的 C 语言编译、串口发送和接收和 Coremark 跑分的功能。目前未实现的功能包括但不限于：Flash 软件程序烧写和 bootloader 实现；FreeRTOS 移植和 Micropython 环境移植等等，目前只是一个基础和开源 MCU 移植验证，希望假以时日，未来可以实现更多的指令集，完成更多功能。

（3）Coremark 跑分程序参考了 tinyriscv[3]的移植过程，计算公式为 Iteration/Sec 值除以工作频率，将 Iteration 从 10000 调节至 2000 后，分数得到了 0.1 左右的提升。

第五部分　参考文献

[1] https://github.com/ultraembedded/riscv

[2] https://github.com/ultraembedded/biriscv

[3] https://gitee.com/liangkangnan/tinyriscv

附录

工具链概述

本系统提供了完整的 gcc 工具链，实现 C 语言程序的预处理、编译、汇编和链接，生成软核可执行的 elf 文件和 bin 文件。同时，本系统还提供了一个 Python 脚本，从而将 bin 文件转换为 Pango Design Suite 中 DRM Ram IP 可读取的 RAM 初始化文件。

软件编译示例

下面以 coremark 跑分软件程序为例，介绍软件编译整体流程。

将 riscv32im gcc 工具链“opt”目录拷贝到 linux 系统根目录，运行以下命令将工具链添加到系统变量：

export PATH=$PATH:/opt/riscv32im/bin

使用 cd 命令进入软件程序目录，运行以下命令编译软件项目：

make

编译后在软件程序目录下的“build.riscv.coremark_test”目录内可以找到 coremark_test 文件，该文件即为 elf 文件。

使用 cd 命令进入“build.riscv.coremark_test”目录，使用以下命令将 elf 文件转换为 bin 文件：

/opt/riscv32im/bin/riscv32-unknown-elf-objcopy -O binary coremark_test coremark_test.bin

执行后将在同一目录找到 coremark_test.bin 文件。将此文件改名为 software.bin，并将 conv_bin_to_txt.py 脚本复制到同一目录下，执行 Python 脚本，将 software.bin 转换为 software.bin.dat 文件。此文件为十六进制文本文件，可作为 Pango Design Suite 中 DRM Ram IP 可读取的 RAM 初始化文件，从而将软件随硬件一同烧写至开发板。