计算机系统结构实验

总结报告

实验题目：操作系统移植与应用编写

学号：1752723

姓名：王松森

指导教师：秦国峰

日期：2020-6-10

目 录

[一.实验环境部署与硬件配置说明 3](#_Toc42848291)

[二.实验流程 3](#_Toc42848292)

[三. OPENMIPS CPU改造 3](#_Toc42848293)

[3.1.CPU的总体结构 3](#_Toc42848294)

[四. 操作系统移植 7](#_Toc42848295)

[4.1.概述 7](#_Toc42848296)

[4.2.移植过程 8](#_Toc42848297)

[五. 应用程序编写 15](#_Toc42848298)

[六. 总结与体会 21](#_Toc42848299)

[七. 参考文献 22](#_Toc42848300)

# 一.实验环境部署与硬件配置说明

本次实验是在Windows10操作系统下，利用Vivado 2018.2设计套件的集成环境，基于XILINX公司的NEXYS4 DDR板子进行的操作系统移植与引用程序编写实验，辅以Modelsim进行仿真验证。

由于疫情影响，条件所限，本实验没有采用外设模块进行应用编写，而是充分利用了板上DDR2，数码管，LED等来进行应用程序的设计与实现。

# 二.实验流程

根据实验要求，我们首先需要改进我们上学期设计的简单MIPS流水线CPU，进行下板验证；

然后我们需要利用CPU移植任意操作系统，并在此操作系统上编写自己所需的应用程序，下板呈现效果。

我的实现过程大致如下，后续报告展开也将依据此流程进行阐述：

1.首先在Vivado中创建项目并根据已有的OPENMIPS动态流水线 CPU进行适当改造，然后通过编写test bench文件进行测试，联合Modelsim仿真以查看波形是否正确，从而查错和纠错；然后仿真正确后将管脚的约束文件加入，编译下板；

2.由于自行设计的MIPS CPU过于简单，缺少MMU内存管理单元等移植操作系统必需的部件，且在指令集上也有所不足，无法真正兼容和移植操作系统，因此本实验选用XILINX公司经过优化，可以嵌入在FPGA中的RISC处理器软核—— MicroBlaze嵌入式软核来进行操作系统的移植工作；

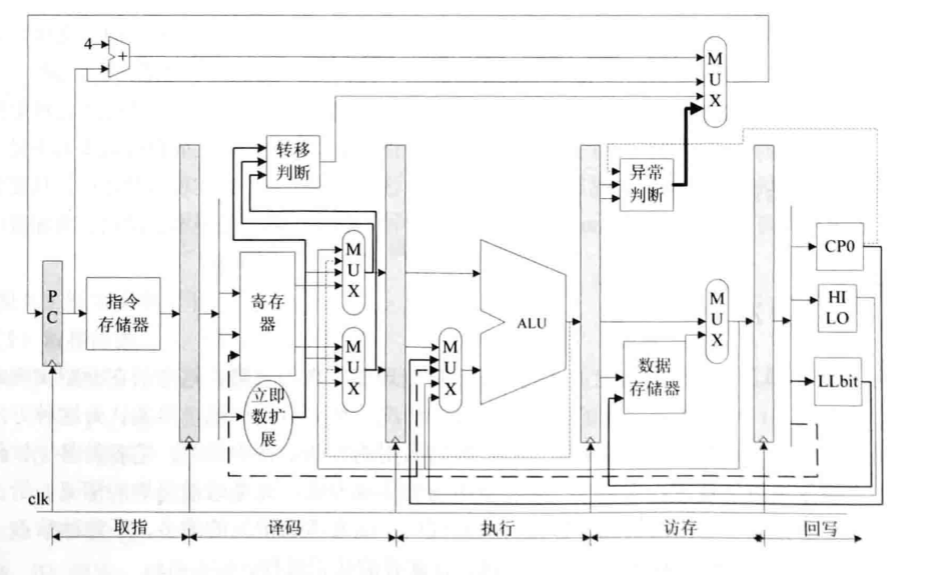
3.然后我经过筛选选择了架构较为简洁的FreeRTOS 操作系统来完成移植。根据官网介绍，该操作系统能够支持MicroBlaze软核，只需将FreeRTOS源代码以及其中支持MicroBlaze软核编译环境的端口文件导入项目，即可利用软核带有的GNU工具链进行编译，从而完成移植工作；

4.移植操作系统结束后，我们可以在Xilinx SDK环境中按照FreeRTOS操作系统的规范编写我们所需实现的应用程序，并引入板级支持包(bsp)中相应板载模块的API函数，进行DDR2、LED等的应用程序编写，并重新编译下载执行，整个实验到此结束。

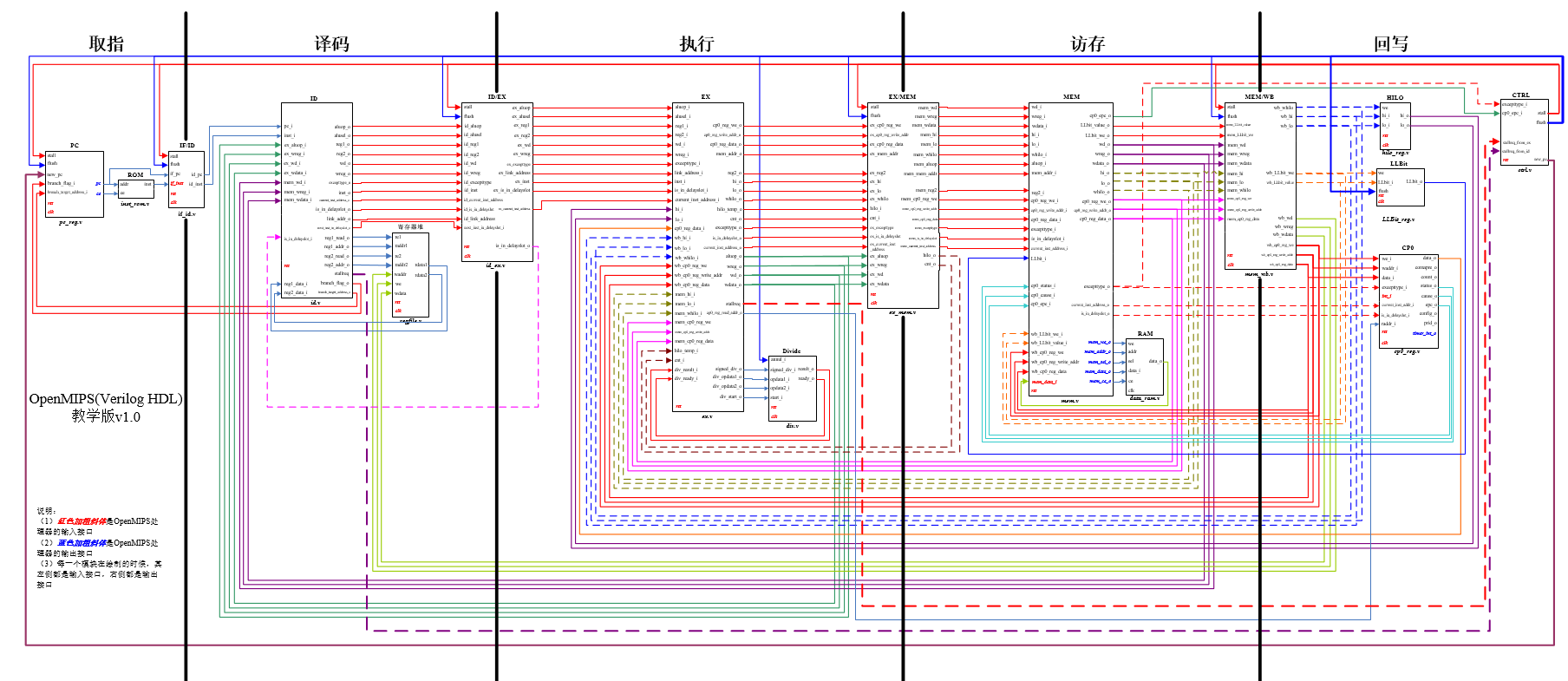
# 三. OPENMIPS CPU改造

## 3.1.CPU的总体结构

我们将要实现的CPU总体框架如下图所示：



具体的线路连接图如下(线路在实现过程中进行了改进，因此本图仅做最初参考之用)：



【说明】上图的流水线CPU主要包含取址、译码、执行、访存和写回5个大部件，并将控制器及状态机融入其中，各部件各司其职，通力合作完成指令的执行。图中还包含IF\ID等中间存储部件，用于在各周期之间暂存数据结果。具体如下：

1. 取址：IF部件。通过PC计算得到的指令地址从CPU外部的指令寄存器中取出指令；

2.译码：ID部件。这一部件主要是识别出指令所需的运算、取得所需的源操作数和要写入的寄存器地址；

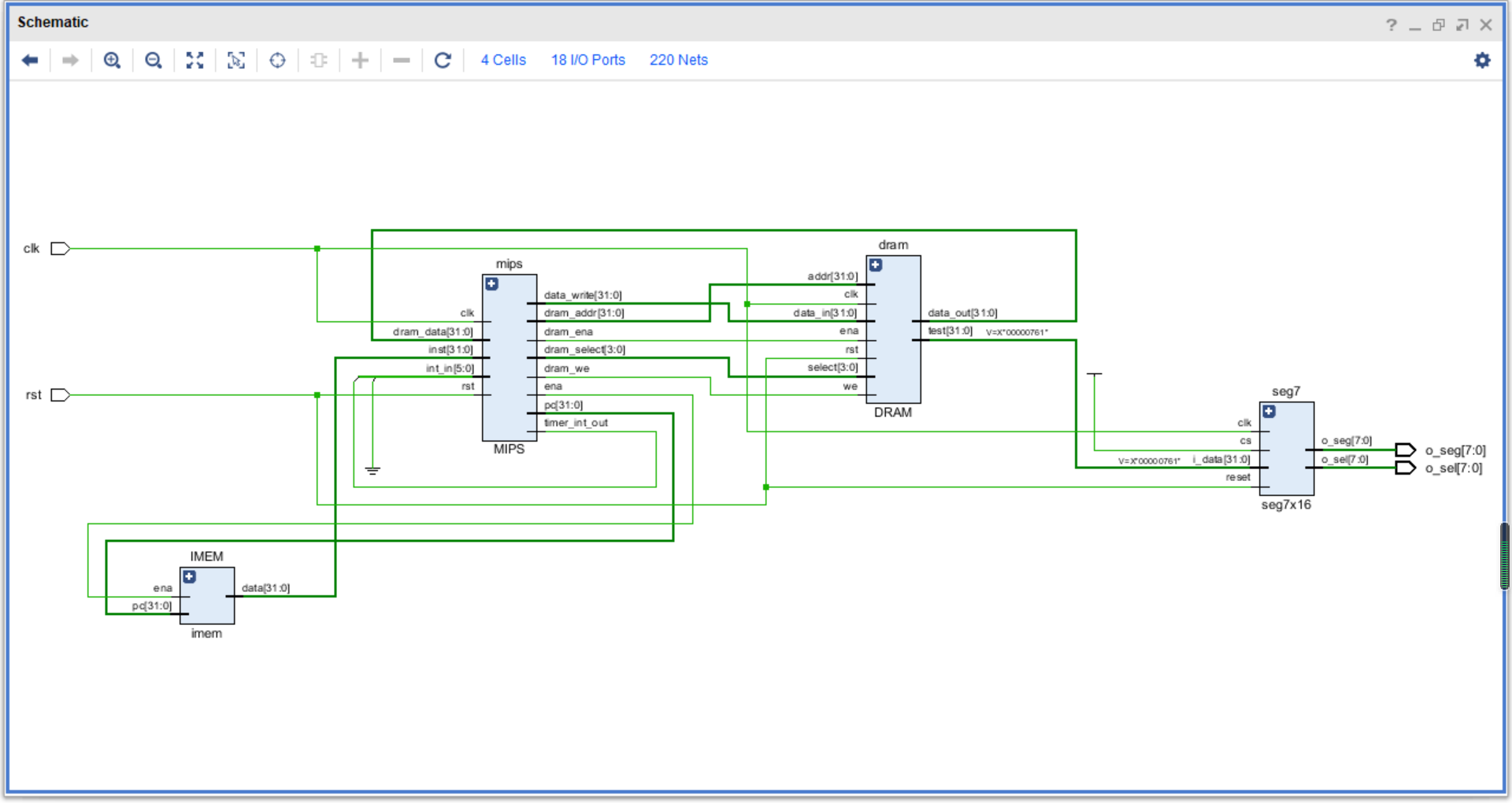
3.执行：EX部件。根据译码所得的信息和源操作数，进行运算并给出结果。

4.访存：MEM部件。对于跳转和读写指令，需要在这一阶段读写内存，其余指令空闲；

5.写回：MEM\WB部件。这里把要写回regfile的数据写回相应地址。

除此之外,该CPU还设计有比较完善的异常处理指令集，能够胜任比较复杂的应用编写。

最终实现的RTL结构图如下：



支持的指令集

该CPU所支持的指令集采用MIPS32 Release1指令集，具体包括的指令如下：

1.逻辑操作指令

AND、ANDI、OR、ORI、XORI、NOR、LUI

2.移位操作指令

SLL、SLLV、SRA、SRAV、SRL、SRLV

3.移动操作指令

MOVN、MOVZ、MFHI、MTHI、MFLO、MTLO

4.算数操作指令

ADD、ADDI、ADDIU、ADDU、SUB、SUBU、CLO、CLZ、SLT、

SLTI、SLTIU、SLTU、MUL、MULT、MULTU、MADD、MADDU、

MSUB、MSUBU、DIV、DIVU

5.转移指令

JR、JALR、J、JAL、B、BAL、BEQ、BEGZ、BEGZAL、BGTZ、

BLEZ、BLTZ、BLTZAL、BNE

6.加载存储指令

LB、LBU、LH、LHU、LL、LW、LWL、LWR、SB、SC、SH、SW、

SWL、SWR

7.协处理器(CP0)访问指令

MTC0、MFC0

8.异常相关指令

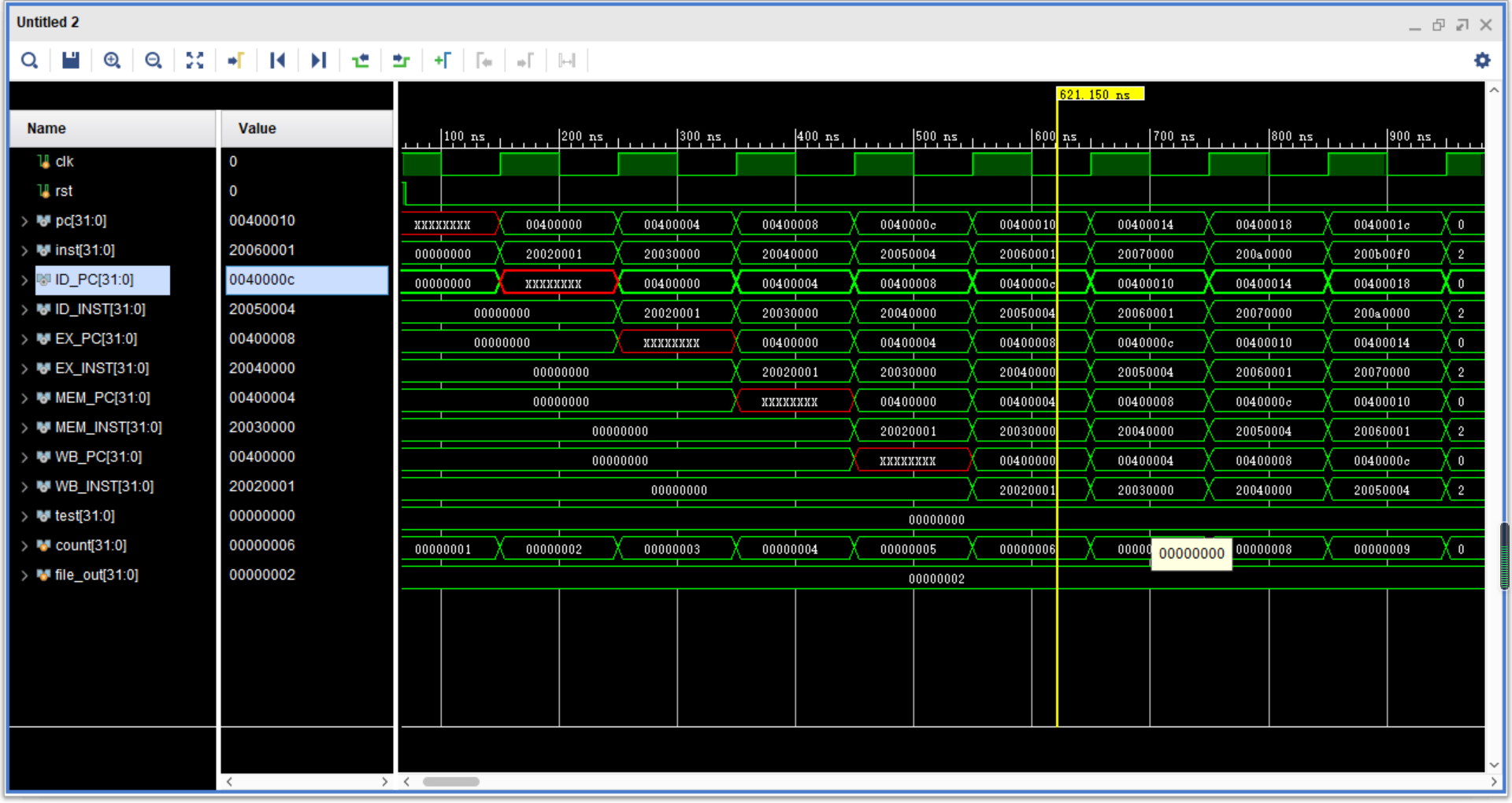
TEQ、TGE、TGEU、TLT、TLTU、TNE、TEQI、TGEI、TGEIU、

TLTI、TLTIU、TNEI

其他还包括NOP(空指令)等。

其余部分已经在前面报告中详述，这里不再赘述，以下是仿真波形以及下板测试的结果展示：

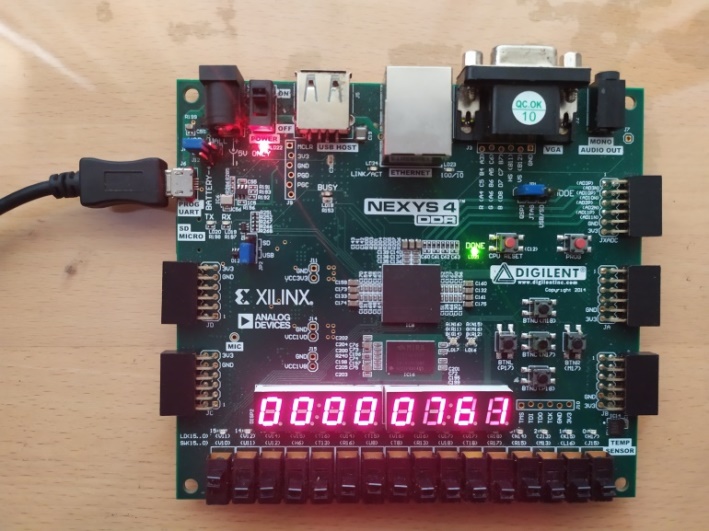
主要测试了各阶段的pc及指令值的流水作业情况，同时将存储器以及regfiles中的数值输出到文件，具体波形如下所示：



输出的寄存器等部分结果如下：



下板结果如下：

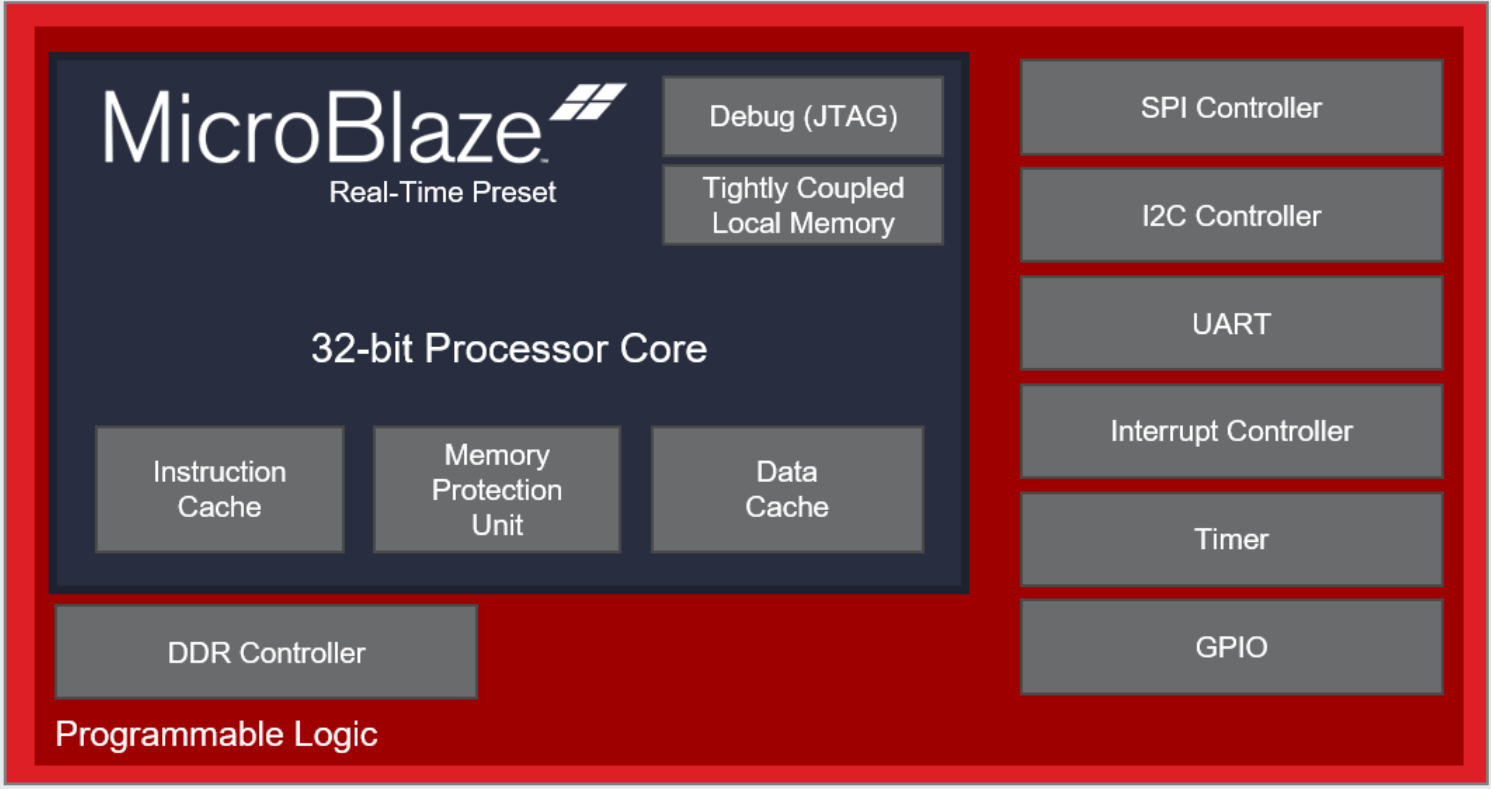


# 四. 操作系统移植

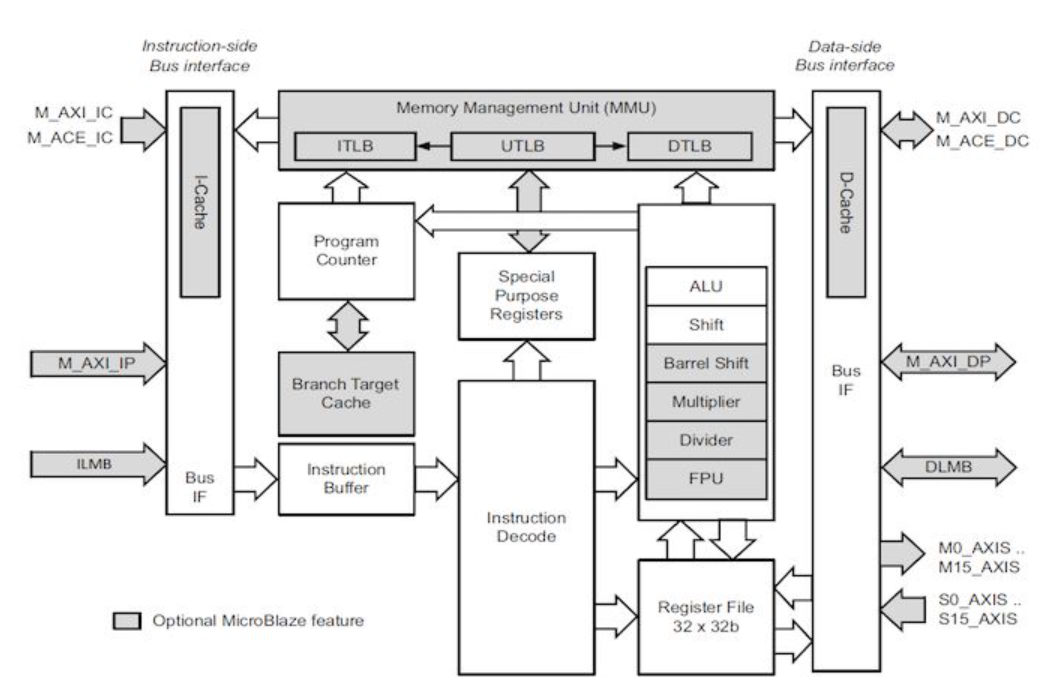
## 4.1.概述

如前所述，由于自制CPU的局限性，不足以支撑XILINX系列FPGA的嵌入式开发。因此我根据XILINX官网推荐，选用了Artix-7系列可支持的[MicroBlaze™ 处理器内核](https://china.xilinx.com/products/design-tools/microblaze.html)进行嵌入式的开发，该软核具有运行速度快、占用资源少、可配置性强等优点，广泛应用于通信、军事、高端消费市场等领域。

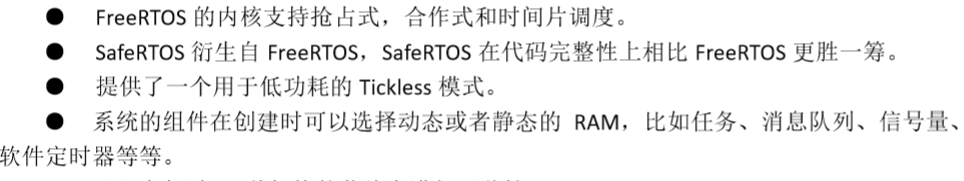
MicroBlaze™ 是基于 Xilinx FPGA 的 32 位 / 64 位 RISC Harvard 架构软处理器。它支持各种高级架构选项，如 AXI 接口、内存管理单元 (MMU)、指令与数据端缓存、可配置流水线深度以及浮点单元 (FPU)。该软核处理器包含了针对RTOS上的确定性实时处理的预置配置，主要如下图所示：

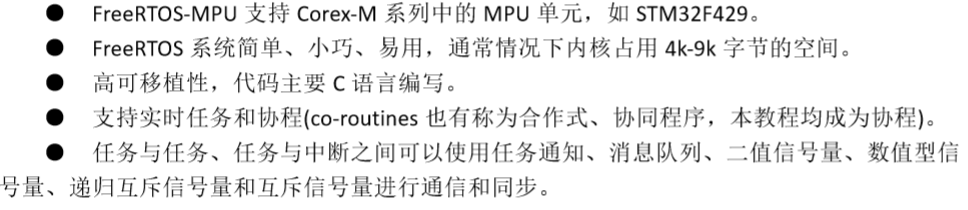


该软核处理器的核心架构如下图：

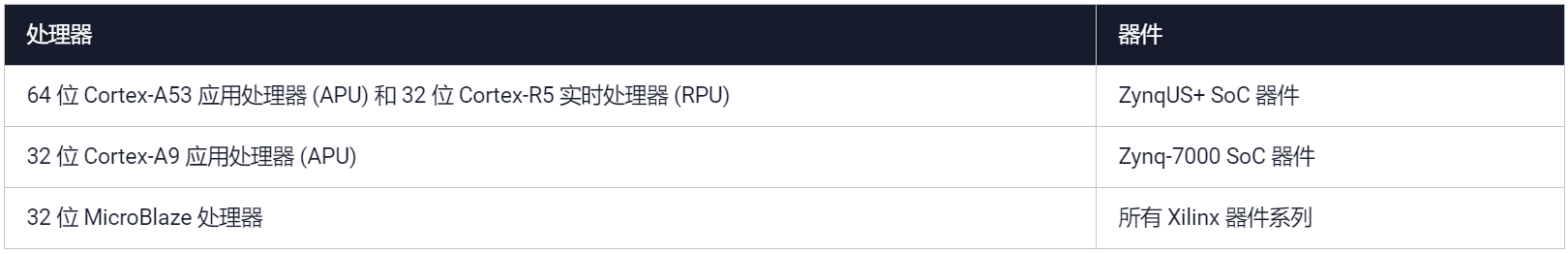


操作系统方面，考虑到移植的方便性以及源码获取的简便性，我选择了FreeRTOS操作系统进行移植。选用实时操作系统(RTOS)的原因在于其简单和轻量级的特性，同时它有最低的中断时延和接触线程切换时延。而FreeRTOS操作系统十分的小巧易用，可裁剪，完全免费，已经被移植到了许多不同的微处理器上，且它经过了专业的开发，质量控制得到了严格保证，运行非常稳健。该操作系统的特点部分如下：





同时，FreeRTOS操作系统内核端口已经针对XILINX公司的处理器提供了支持，如下图所示；且XILINX公司方面也提供了方便FreeRTOS嵌入式开发的板级支持包。因此移植该操作系统会更加简便。

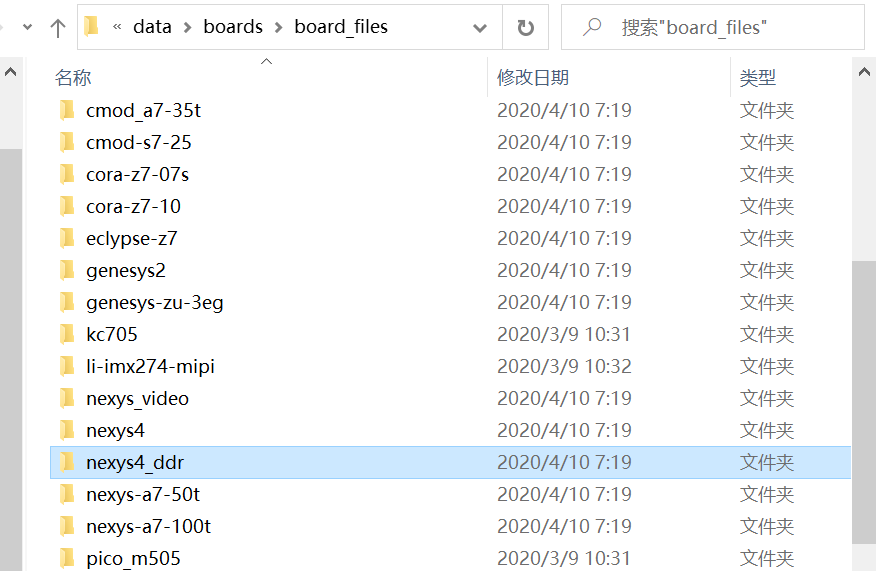


综合以上考虑，我完成了基于MicroBlaze软核的FreeRTOS操作系统移植工作。

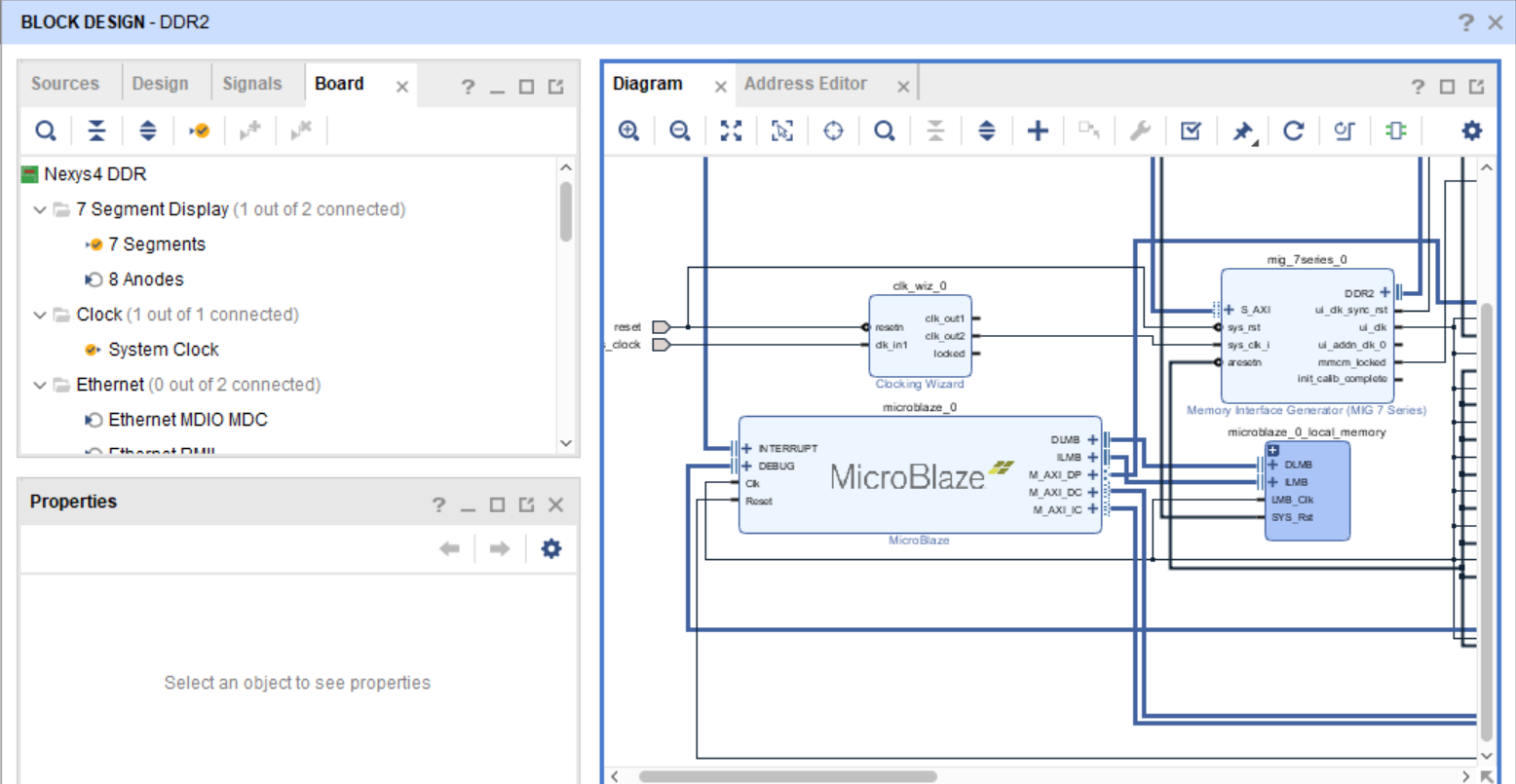
## 4.2.移植过程

移植FreeRTOS的过程包括硬件设计以及软件实现两部分。

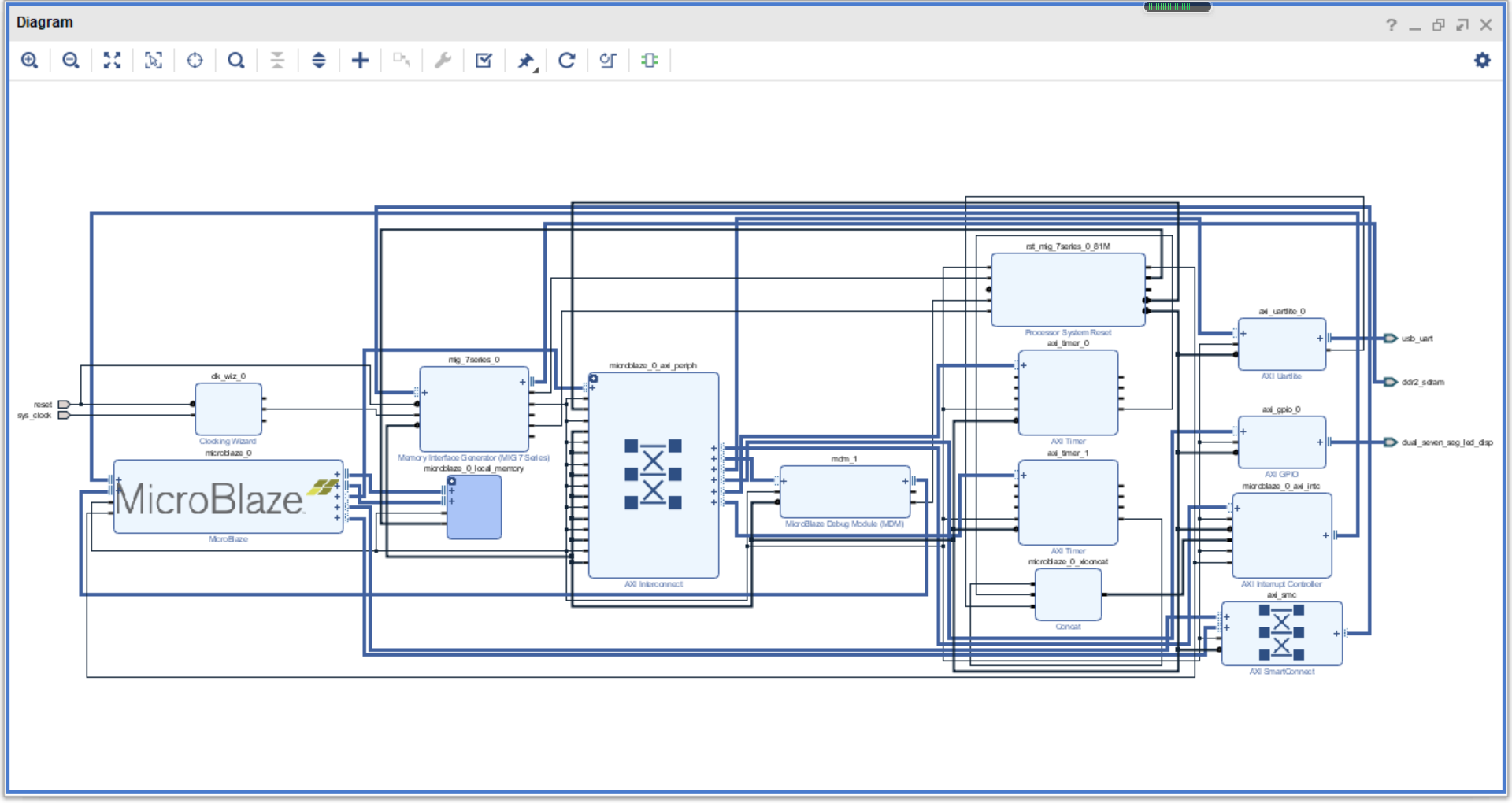
首先需要按需搭建硬件系统，以便支持操作系统的运作。搭建的具体过程此处不再赘述，准备工作中需要下载好支持NEXYS4 DDR的板卡文件，安装在对应的文件夹路径(D:\Xilinx\Vivado\2018.2\data\boards\board\_files)中即可：



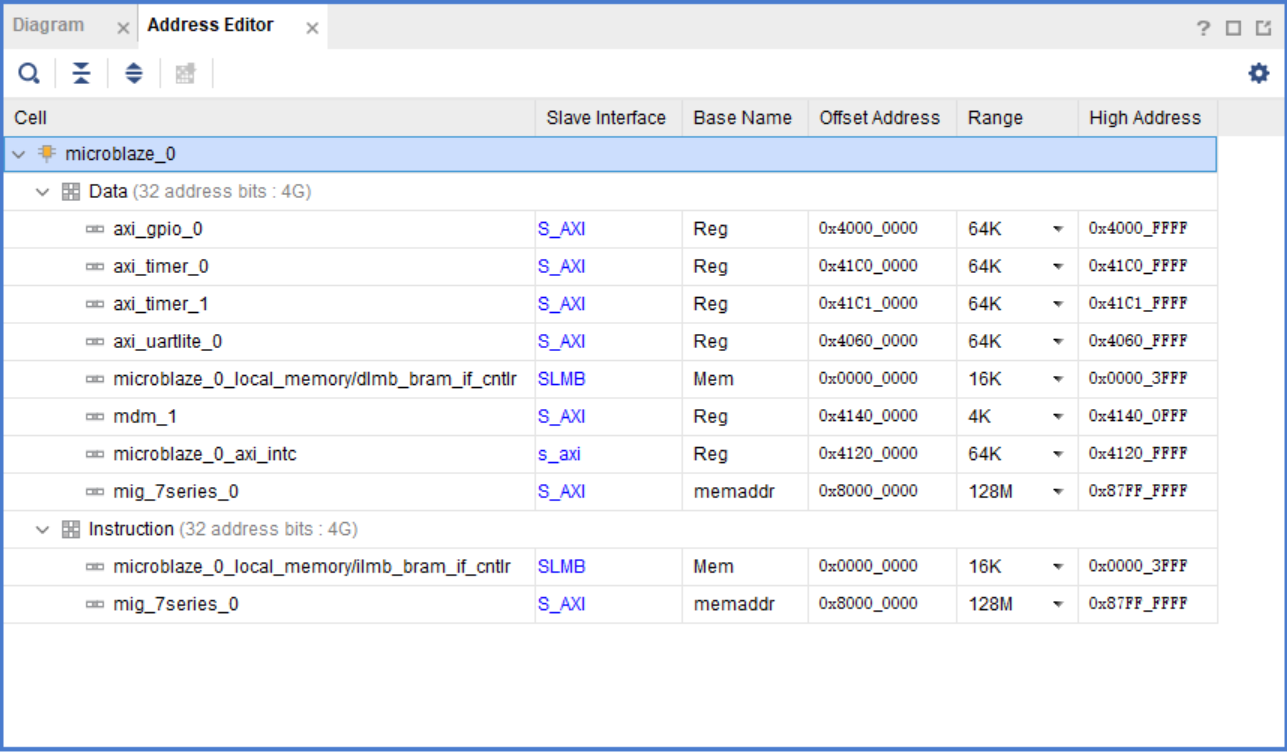
然后可以通过Vivado自带的Block Design窗口进行可视化的硬件搭建过程——只需将所需的板卡设计资源以及其他所需的IP核等拖入窗口，完成手动或自动连线操作即可：



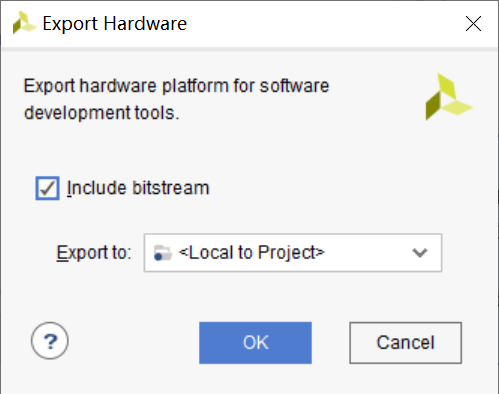
FreeRTOS所需的IP核包括软核MicroBlaze，系统时钟，与电脑端通信等的UART端口，以及AXI Timer这类时钟模块；我在设计时又根据需要自行添加了DDR2，GPIO模块等，方便后续软件部分实现应用程序的开发。实现后的总体架构如下：



通过编译自动分配管脚与各IP核地址之后，我们可以在地址分配部分看到最终不同模块所分配的地址范围以及空间大小：



选择导出硬件工程以及生成的比特流：

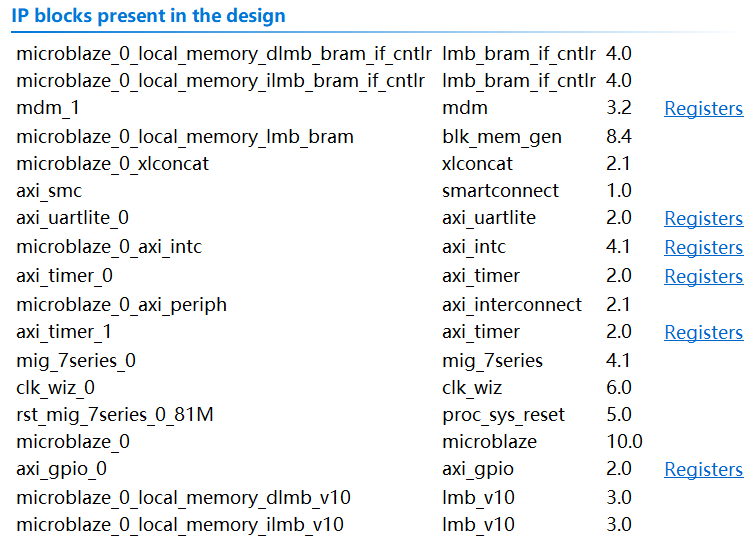
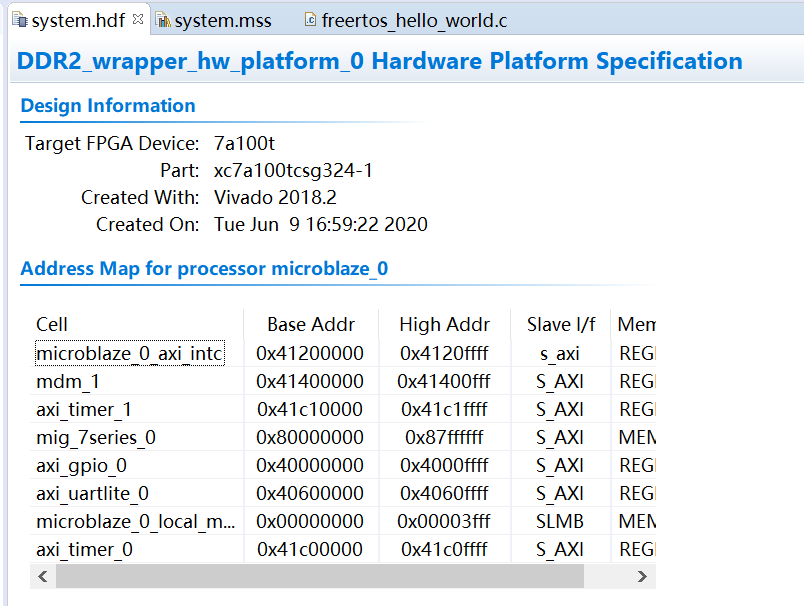


硬件工程设计部分到此结束。

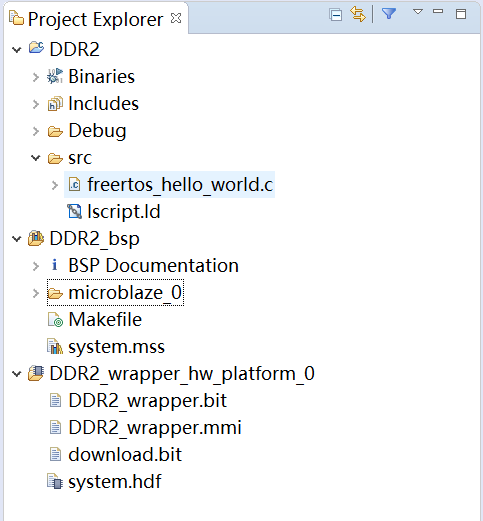
------------------------------------------------------

软件实现部分将利用XILINX SDK环境进行开发和调试。SDK是基于Eclipse环境进行开发的，所以具有较好的亲和力，无需学习过多的软件使用方法。

我们可以在生成的system.hdf文件中看到各模块的地址分配信息以及设计中的所有IP核：

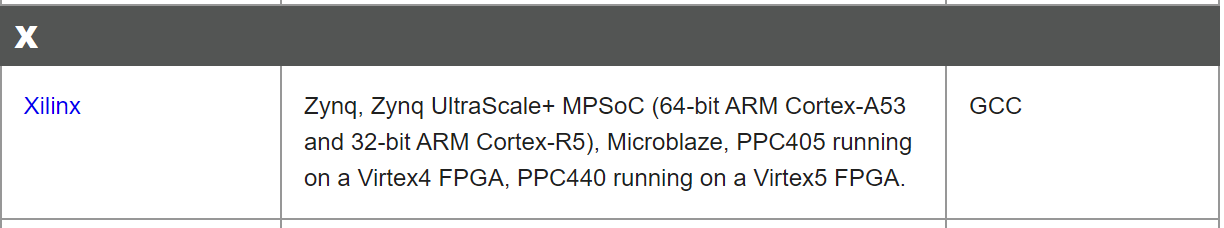


创建一个项目，所生成的项目目录结构如下：

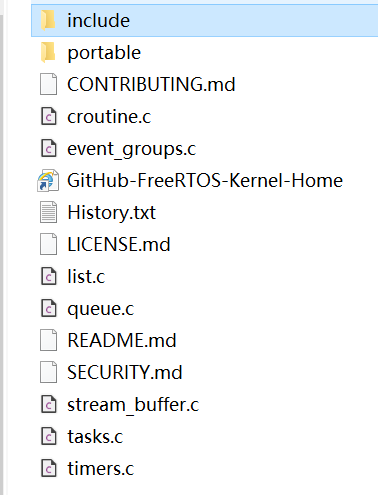


接下来进行具体的移植操作：

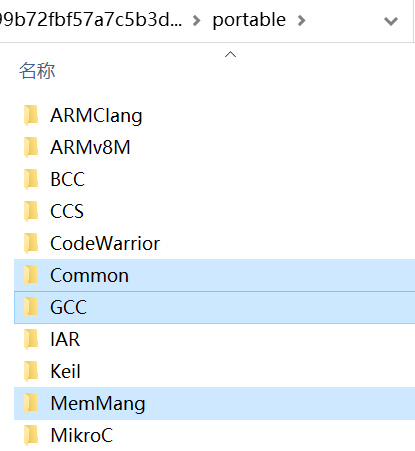
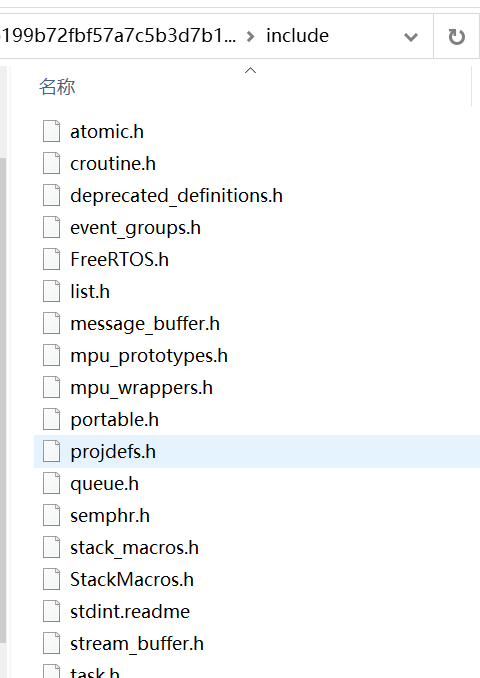
首先根据FreeRTOS操作系统官网提供的支持设备的端口，我们看到XILINX系列支持Microblaze家族的端口，支持GCC编译。



我从官网下载了FreeRTOSv10.0.0的源码，根据上述信息，我分析了源码的目录结构：



排除掉无关文件后，外部的c文件是列表、队列、定时器等FreeRTOS操作系统的源码部分，include文件夹中则是相关的头文件，在应用程序编写过程中，所需的API定义可以在其中找到；下图中，左图为include文件夹，右图是portable文件夹。



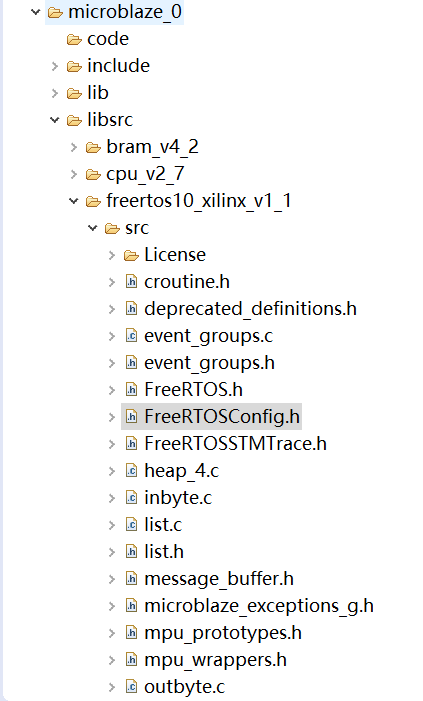
而portable文件夹则是根据具体MCU类型设定的端口文件，根据SDK中自带的FreeRTOS板级支持包，我找到了portable文件夹中必需的几个文件夹，如上面右图所示。

由于FreeRTOS支持XILINX SDK MicroBlaze自带的GCC编译器，所以我们选择GCC文件夹下的MicroBlaze端口文件，用于建立将操作系统架设在MicroBlaze软核上的桥梁，如下图。

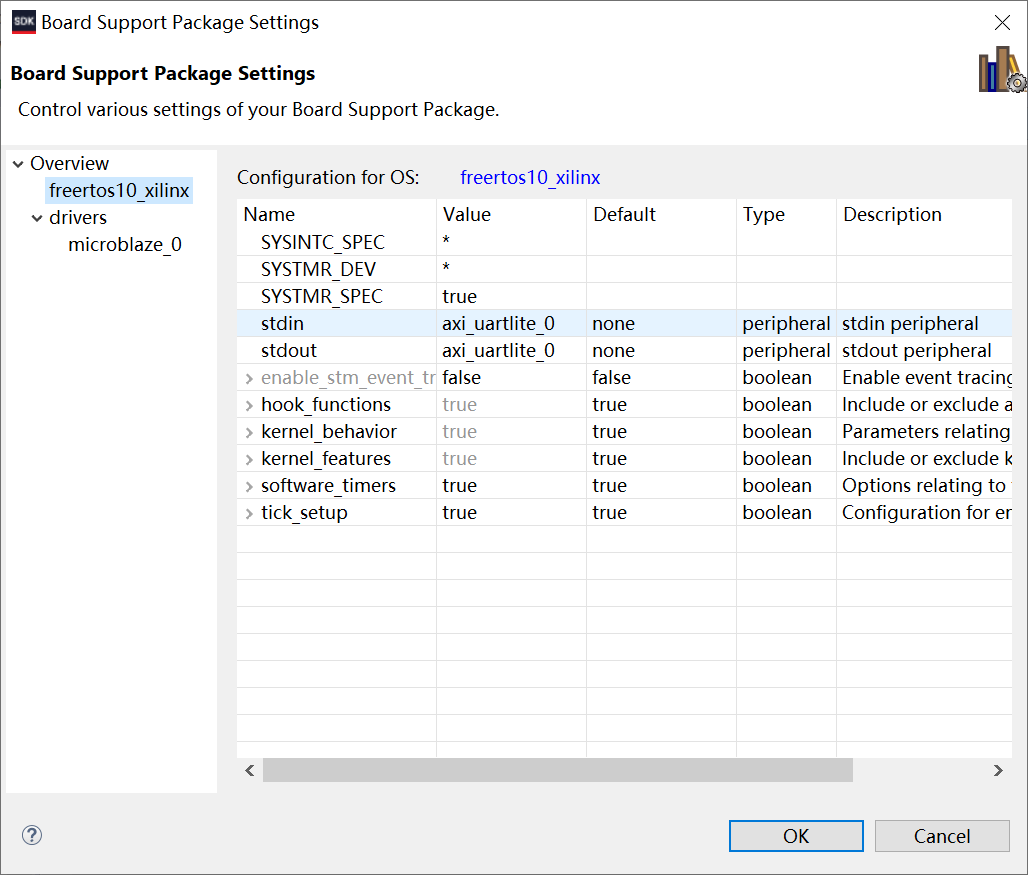
除此之外，内存管理MemMang文件中我选择了head\_4.c的内存管理策略；以及MPU相关的c文件等。



然后将精简出来的文件、文件夹按照原有支持包中FreeRTOS要求导入自己所需版本的FreeRTOS对应文件到bsp中microblaze目录下，不必对源代码做过多改动和前期编译工作，最终生成的目录如下；

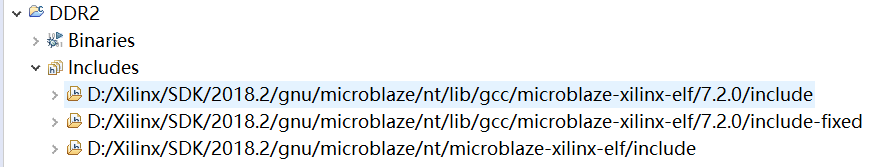


同时可以在配置文件FreeRTOSConfig.h中对FreeRTOS操作系统进行裁剪，如下图：

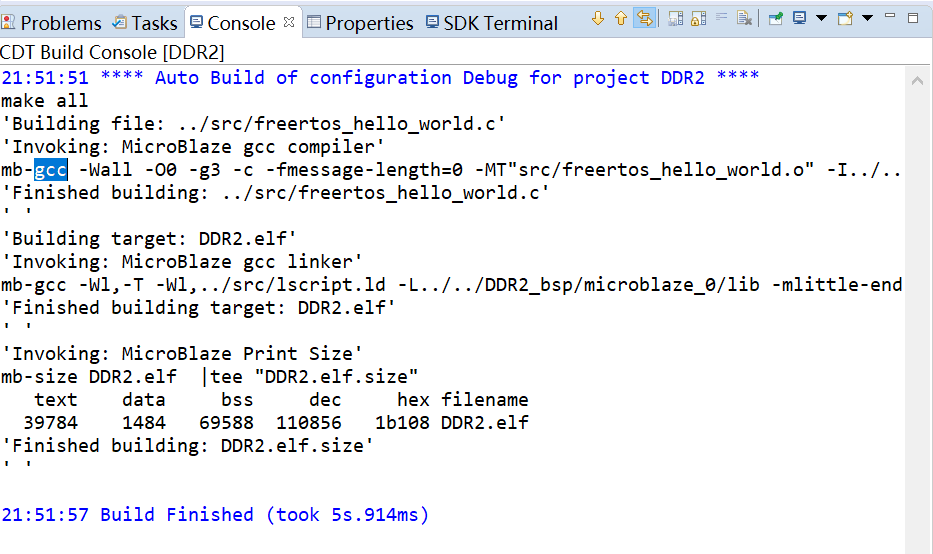


由于SDK 由GNU工具链（GNU编译器集合，GNU调试器）提供支持，因此只需要通过该GNU工具链对整个软核连同FreeRTOS一同编译即可完成移植工作，接下来只需要进行应用编写即可。

编译工具链包含在项目之中，如下图所示：



保存修改后只需重新build项目，即会自动进行编译工作，将嵌入式的C语言项目编译成比特流，方便下板：



到此，操作系统的移植操作就基本完成了。

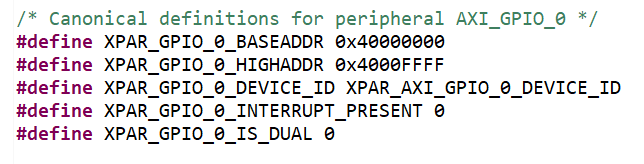
# 五. 应用程序编写

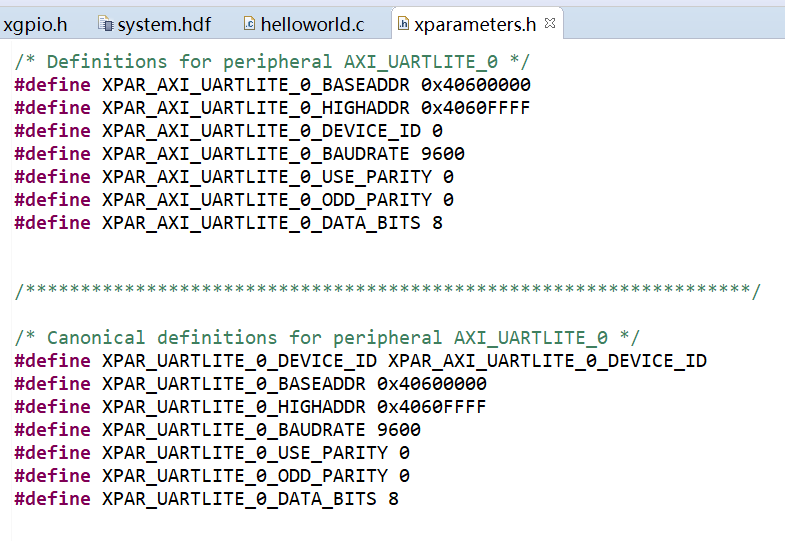
移植操作系统结束，就可以正式地进行应用程序的编写了。为了熟悉C语言操作外设的方法，首先我尝试了直接使用MicroBlaze进行应用程序的编写。

以下是通过GPIO控制LED实现流水灯的程序。只需要按照GPIO初始化、设置电平高低以及串口输出的函数顺序编写代码。由于在这里程序将依照主函数顺序执行各个子函数，因此编码较为容易，只需关注xparaments.h中的设备号、连接频率等即可。



下图所示即为控制LED所需的设备号，以及串口连接时的频率限制：





在完成上述MicroBlaze应用程序的编写之后，我就转而进行同样应用程序在FreeRTOS上的编写。根据FreeRTOS的手册，应用程序主函数部分将不再顺序执行，而是需要创建任务并指定优先级，应用程序可以创建多个任务。由于这里使用的微处理器仅有一个核，因此在任意给定时间，实际上只会有一个任务被执行，其他任务将会保存上次执行状态并休眠。

大致流程是：先创建任务，使用的是xTaskCreate() API 函数，并为它们指定优先级；由于这里采用的是FreeRTOS中的**固定优先级抢占式调度策略**，所以调度器总是在所有处于就绪态的任务中选择具有高优先级的任务来执行，阻塞态等其他状态的任务将不被纳入考虑范围，可以采用vTaskDelay() API函数来让任务进入阻塞态。

FreeRTOS 本身并没有限定设定的优先级的数量。在FreeRTOSConfig.h 文件中设定的编译时配置常量configMAX\_PRIORITIES 的值，即是最多可具有的优先级数目，优先级此时可设定为0到(configMAX\_PRIORITES–1)之间的整数。如下图所示：



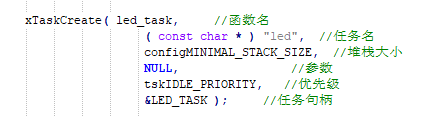
与此同时，任意数量的任务可以共享同一个优先级，当选中的优先级上具有不止一个任务时，调度器会让这些任务按照**时间片轮转**的策略轮流执行，时间片是根据系统心跳中断来划分的，时间片的长度通过心跳中断的频率进行设定，频率设定如下图：



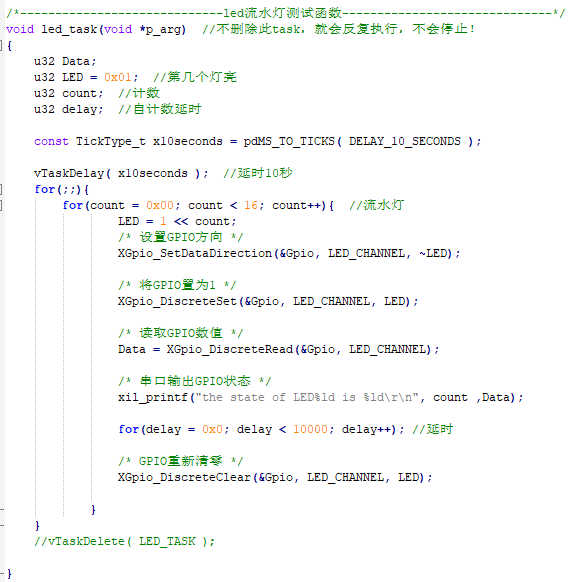
最后利用vTaskDelete()函数删除结束的任务，回收内存空间。否则该任务将可能再度上台运行。

理论分析完毕，现在来进行具体编码。

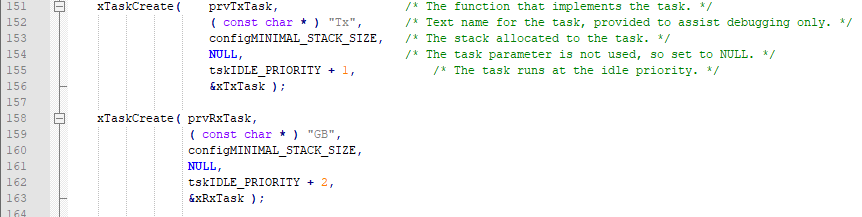
首先创建一个LED流水灯的任务，处于0优先级，便于优先执行其他任务：



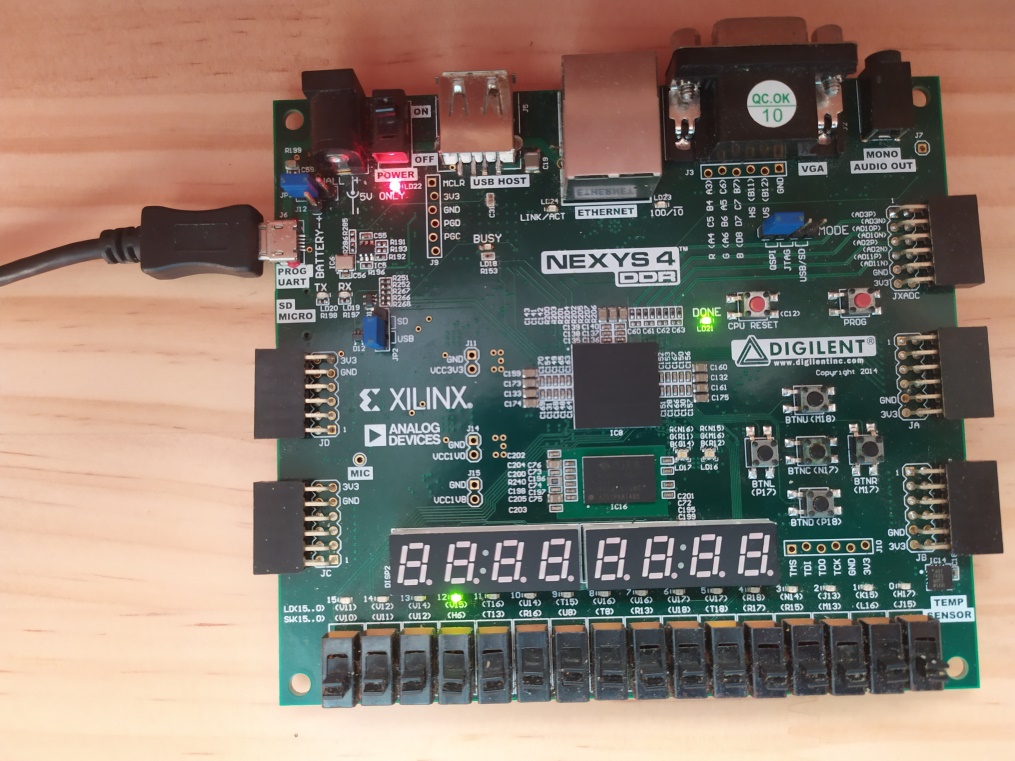
以下是该任务的子函数，仿照前面MicroBlaze流水灯函数写即可(通过vTaskDelay()阻塞其执行)：



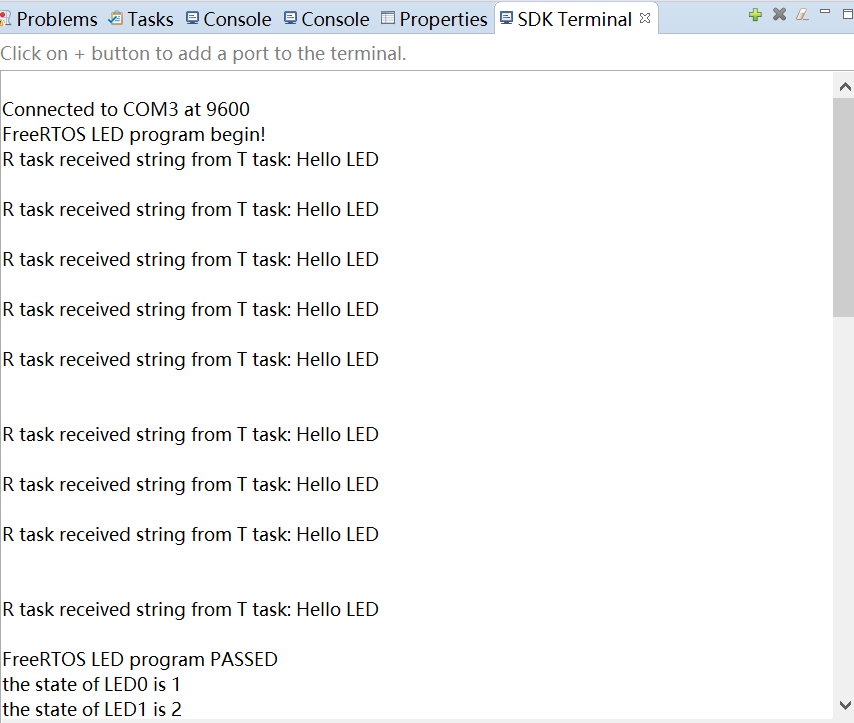
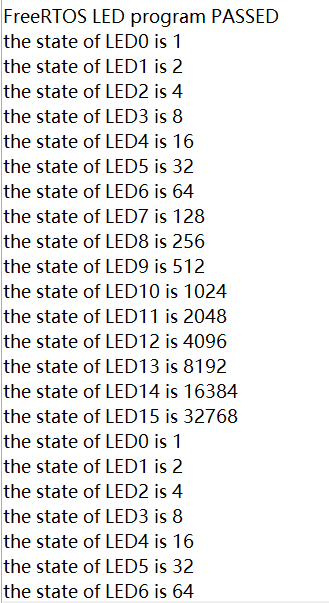
其余两个任务是一个前10s完成的Tx，Rx有关队列的交互式任务，优先级等设定如下：



运行结果如下图。前10s完成Tx-Rx互传任务，然后流水灯任务在10s之后开始无限执行：



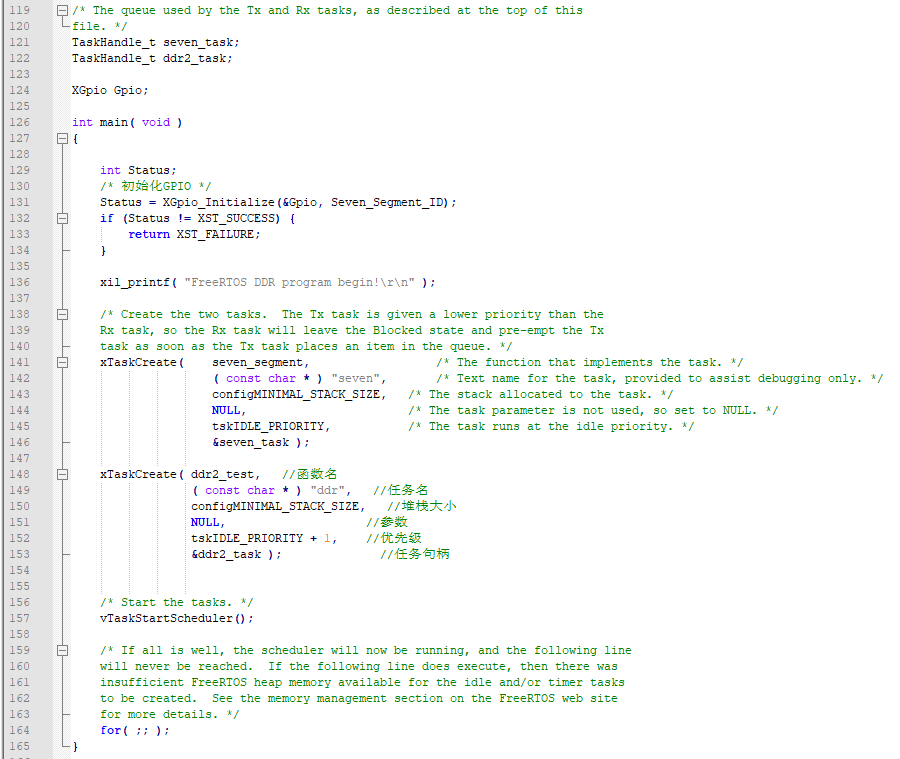
串口终端返回的结果如下：

第一个应用程序到此结束。

-------------------------------------------------------------------------

然后第二个应用程序是有关对DDR2进行读写和7段数码管的显示。同样是先在MicroBlaze上直接编写代码，然后仿写到FreeRTOS中，最终代码如下：



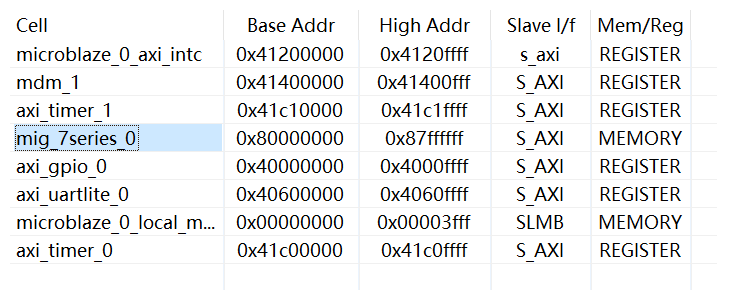
任务函数如下：

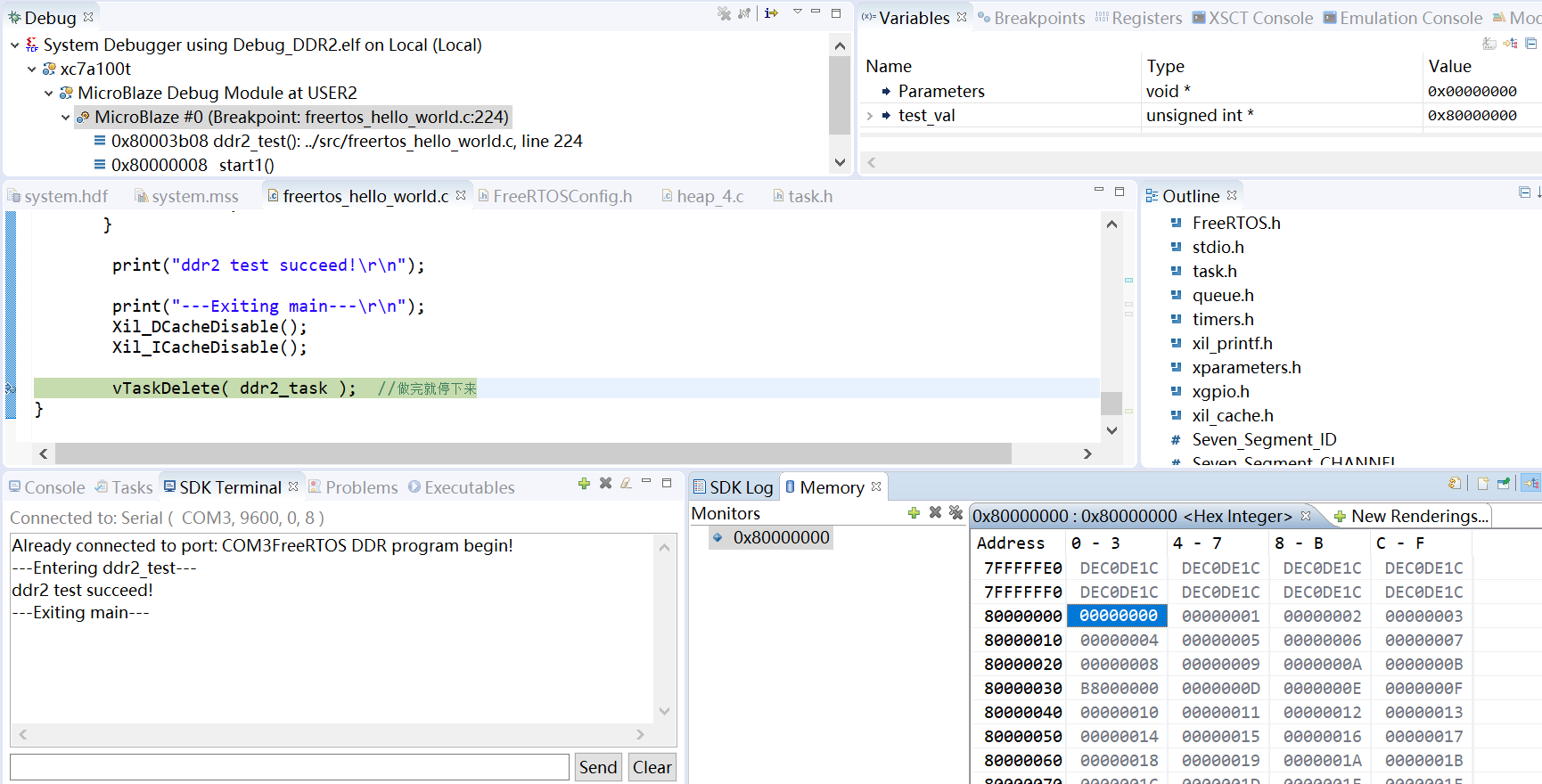




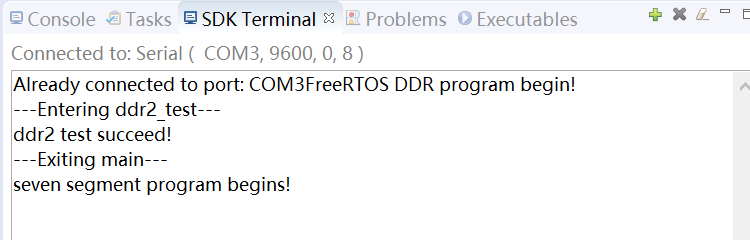
可见DDR2任务具有更高优先级，故先行完成，然后数码管开始蛇形流动，调试过程如下：

由调试断点可知程序首先进入DDR2任务执行，由于从system.hdf我们可以看出来(下图)，DDR2的基地址是0x80000000,所以调试时memory部分可以看到从0x80000000地址写入的递增序列以及SDK Terminal中串口返回的输出信息：

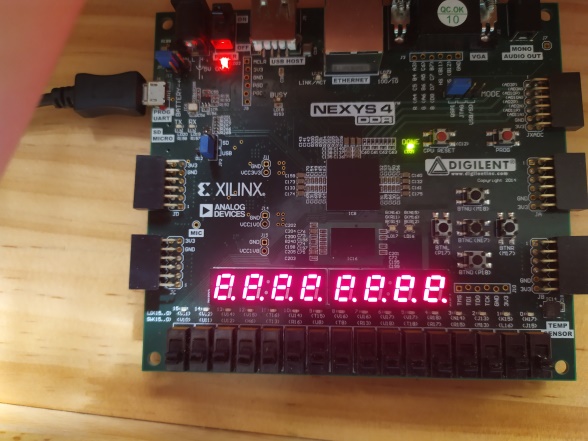


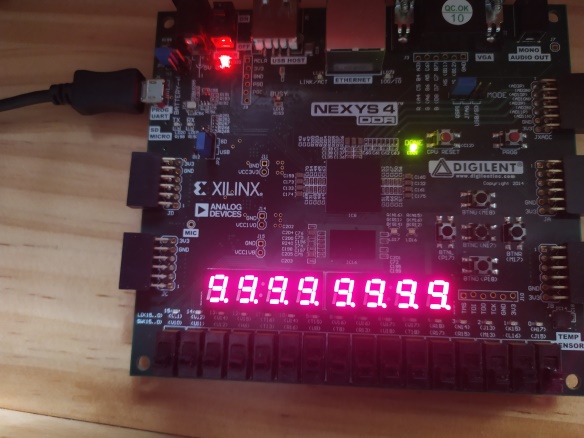
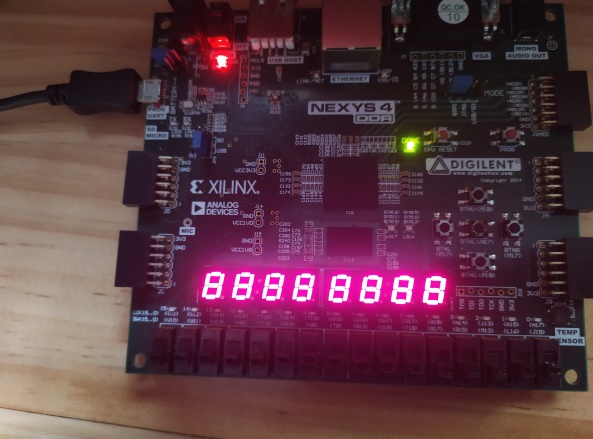


继续执行，可见数码管开始流动，调试口输出结果如下：



板子状态如下：

# 六. 总结与体会

通过本次实验，我对于系统结构课内所学知识有了更深的理解和体会，同时对MIPS CPU的设计和理解也都更进一步；我学会了利用XILINX提供的软核进行操作系统移植与应用程序编写，真正体会到了FPGA在工业应用中的使用便利之处和其强大的性能，也真正感受到了嵌入式应用开发的过程和方法，对于嵌入式有了更大的兴趣和更深的理解，为今后在硬件开发领域的研究打下了良好的基础。

# 七. 参考文献

[1] XILINX FPGA软核MicroBlaze开发资料汇总分享[Z].

<https://hifpga.com/%E9%97%AE%E9%A2%98/36676/xilinx-fpga%E8%BD%AF%E6%A0%B8microblaze%E5%BC%80%E5%8F%91%E8%B5%84%E6%96%99%E6%B1%87%E6%80%BB%E5%88%86%E4%BA%AB/>.

[2] [john199112](http://www.digilent.com.cn/usercenter/e5b294b70c9647dcf804d7baa1903918.html).迪芝伦社区[Z].<http://www.digilent.com.cn/community>. 2017-12-18.

[3] 米联客博文-FreeRTOS 移植到 Xilinx 的 Microblaze[Z].

<https://www.uisrc.com/portal.php?mod=list&catid=59&page=1>.

[4] FreeRTOS官网--FreeRTOS Kernel Ports[Z]. <https://www.freertos.org/RTOS_ports.html>.

[5] XILINX官网[Z].

<https://china.xilinx.com/products/design-tools/embedded-software.html#overview>.

[6] yiwenbo. FreeRTOS(一)系统移植配置[Z].

<https://www.cnblogs.com/yiwenbo/p/10459125.html>.

[7] Frank~\_~FPGA. MicroBlaze测试DDR3[Z].

<https://www.cnblogs.com/yiwenbo/p/10459125.html.2018-10-27>

[8] 正点原子团队. FreeRTOS开发手册[M].广州星翼电子科技有限公司.

[9] Zou Changjun. FREERTOS 实时内核实用指南[M].