E:\学习文档\大三上\计算机组成与设计\流水线\设计文档\总框架.emf

说明：

整个实验框架由图上几部分组成：Clock模块、流水线CPU模块、Fpga板子显示模块、Cache及内存DDR模块、Spi Flash模块、VGA显示模块、键盘模块、代码装载模块、Cpu Interface连接模块，以及后期可能会添加的USB连接模块与网络连接模块等（目前代码中未涉及，但可能的拓展）。

* **Clock模块：**Clock control 通过调用IP核Clock Generator生成DDR时钟以及CPU时钟，并在本模块中进行分频得到其他的时钟频率用以控制其他模块的时钟，以及提供调试用的慢速时钟（可以在7段数码管肉眼看到PC变化的程度）。
* **流水线CPU模块：**此次实验**主要的实现模块**，其余模块可参考之前的代码，也可自行修改实现。此模块主要实现一个流水线CPU，具体功能在本文档中会详细进行描述，此处不再赘述。
* **Fpga板子显示模块：**通过7段数码管显示PC、指令、寄存器数据等，通过LED灯显示DDR2读取暂停（stall）、VGA显示暂停等状态，用以检测异常状态以及进行Debug。（其中PC、指令显示建议显示MEM段的PC以及指令，原因是寄存器状态的改变、异常的停顿等都在MEM阶段或之后，显示MEM段更好查看其真正导致BUG的指令）
* **Cache及内存DDR模块：**逻辑上（假设不考虑代码从Flash到DDR2的装载过程）CPU开始执行我们的代码时对数据及指令进行访问的模块，CPU从指令Cache读取指令，从数据Cache读取数据，若Cache缺失或需要进行写回则访问DDR2模块，也就是内存模块。因此归纳来说，这里就是通俗所说的，Cache与内存。
* **Spi Flash模块：**Fpga板子上提供的串口通信的Flash存储模块，CPU执行的（不考虑装载代码时）我们的代码未读入DDR2时存储的模块。将源代码编译并转化成.mcs代码，并通过使用Vivado的烧录接口，将.mcs代码固化到Fpga板子上的Flash存储中，通过提供的Flash驱动代码进行访问。
* **VGA显示模块：**将从CPU输出到显存的数据通过VGA模块转化数据格式并显示到显示屏上。不过此处是没有真正显存（没有使用Fpga板子上的存储空间）的，只是将输出的数据直接显示到屏幕。（后期会加入显存）
* **键盘模块：**对于来自键盘的输入输出进行处理，并传输到CPU中进行键盘中断的处理，早期实验无需实现，在后序实验中可以进行陆续实现。
* **代码装载模块：**存储了CPU装载Flash中的代码到DDR2的执行代码，装载代码是在CPU中执行，将Flash中代码陆续读入CPU再写入DDR2，最后跳转到Flash代码中的入口地址，开始执行需要执行的真正的代码。其接口类似Cache，为双端接口，一端为数据口，用于读写数据，一端为指令口用于读指令。通过IP核配置，将loader代码（loader.coe为处理后的特殊文件）写入配置出的Block RAM中，CPU通过之前所说读写口对该代码进行读写。
* **Cpu Interface连接模块：**一者用于地址的转换，CPU对于不同的地址的访问对应的接到各模块中（判断地址，设置使能或读出数据等）；二者将各部分的信号以及数据进行互连。

代码执行流程说明：

1. 正常执行流程：CPU将PC设置为装载代码的起始地址，开始从Loader\_mem中读取装载代码，装载代码将Flash中的代码读入CPU，并写入Cache中（Cache写满会进行写回，将代码写入DDR2相应的空间中），装载代码执行完时，将PC设置为Flash代码的起始地址，开始执行写入Cache及DDR2中的代码；执行过程中，若要访存的话根据访存地址分别对Cache/DDR2、VGA、Keyboard等进行访存，直到该代码执行完毕。
2. 其他代码可执行流程：由于loader\_mem的接口类似于Cache，因此可以直接向loader\_mem导入需要执行的代码，并在通过该接口读写数据与读出指令，栈区可以设置在Cache/DDR中（当然也可以设置在loader\_mem中）。

注：第2号执行流程存在两类问题，一者数据可能不通过Cache，因此如果自己写了Cache则无法对Cache进行测试。二者loader\_mem由于存放的是装载程序，装载程序较小，因此设置loader\_mem的空间较小（根据ip核设置我们是byte mem[0x2000][4]），较大程序装不下，或者存储数据空间不够。