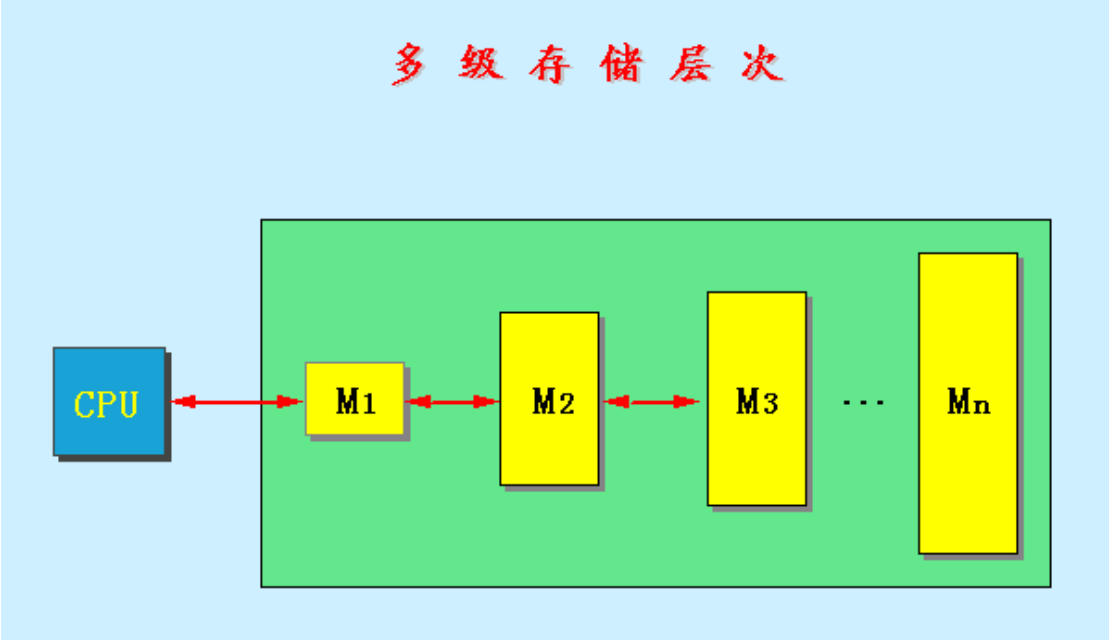
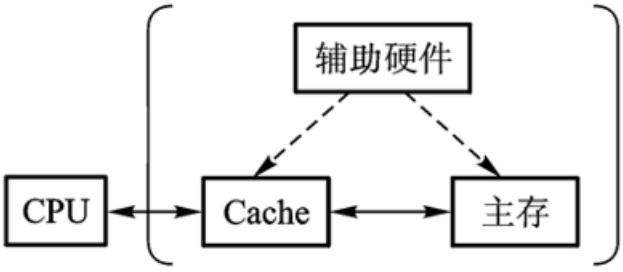
一、提升实验要求

1.1 实验描述

对于绝大多数程序而言，所访问的指令和数据在地址上不是均匀的,而是相对聚集的。包含时间局部性和空间局部性。前者指程序将用到的信息很可能就是现在使用的信息。后者指程序将用到的信息很可能与现在正在使用的信息在存储空间上是相邻的。多级存储器之间以块或页面传送数据，其目标是：从 CPU 来看,存储系统速度接近 M1 的速度，而容量和位价格都接近 Mn 的容量和价格。



本实验采用三级存储结构，即 CACHE -> 主存 -> 辅存。借助辅助硬件，使 CACHE 与主存形成一个有机整体，弥补主存速度不足.由硬件实现,对应用与系统程序员都透明。



“Cachc-主存”层次

“Cache－主存”与“主存－辅存”层次的区别

比较项目 \ 存储层次	“Cache －主存”层次	“主存－辅存”层次
目　　的	为了弥补主存速度的不足	为了弥补主存容量的不足
存储管理实现	主要由专用硬件实现	主要由软件实现
访问速度的比值(第一级和第二级)	几比一	几百比一
典型的块(页)大小	几十个字节	几百到几千个字节
CPU对第二级的访问方式	可直接访问	均通过第一级
失效时CPU是否切换	不切换	切换到其他进程

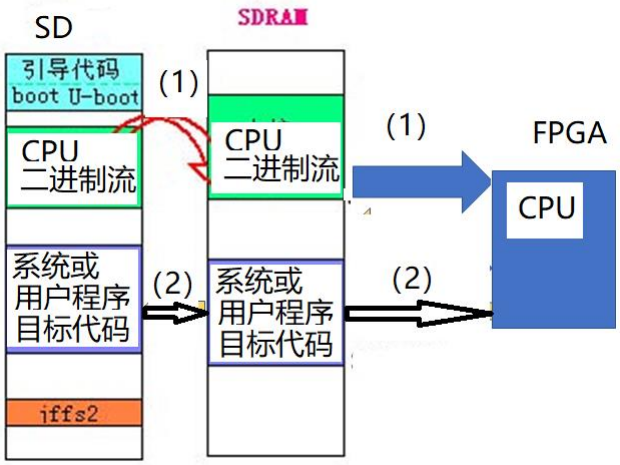
“Cache－主存”层次：弥补主存速度的不足。“主存－辅存”层次：弥补主存容量的不足。

其中，CACHE 具体使用 IMEM 来代替，主存使用 SDRAM DDR2 来代替，辅存使用 SD 来代替。

1.2 总体框架

数据流通路：

SD -> SDRAM DDR2 -> CACHE

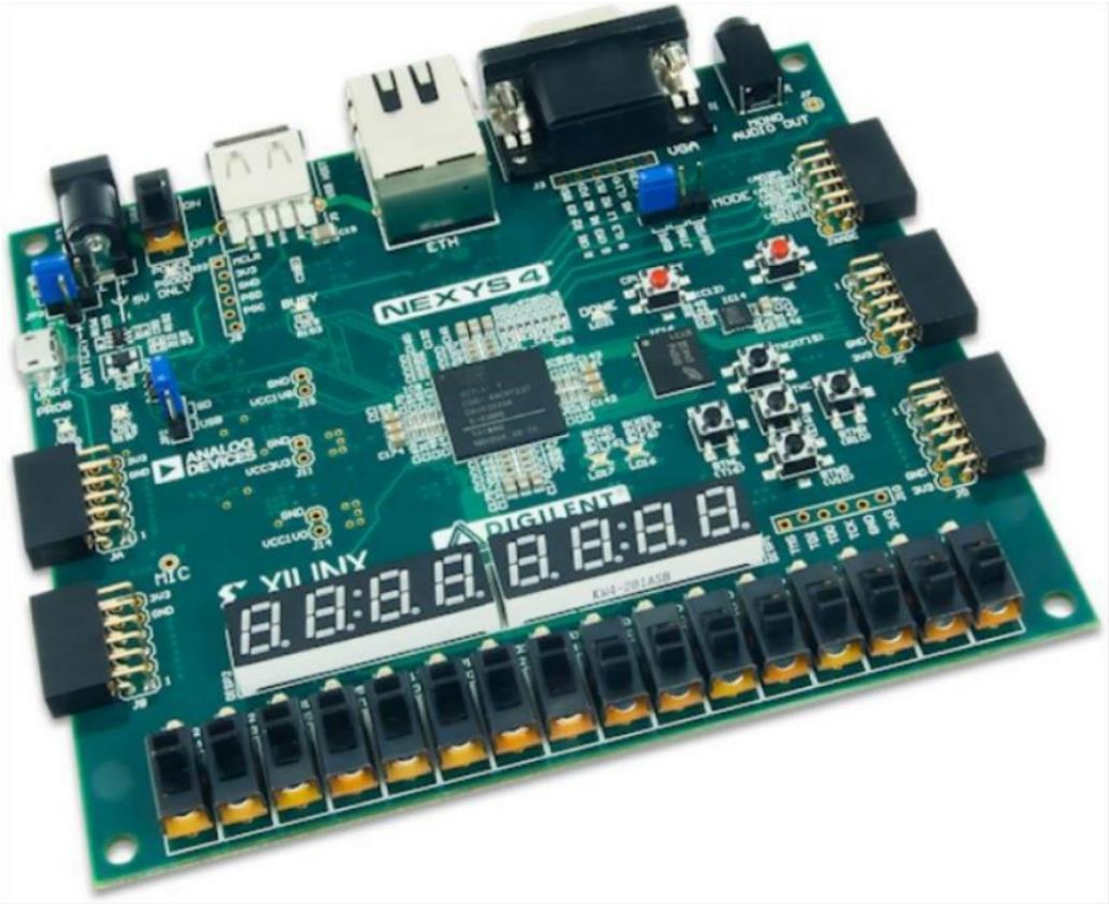


1.3 实验环境

软件环境：Vivado 2018.2

硬件配置：

1. Nexys4 DDR Artix-7 FPGA Board 开发板



2. 2GB microSD 卡一张

1.4 实验具体要求

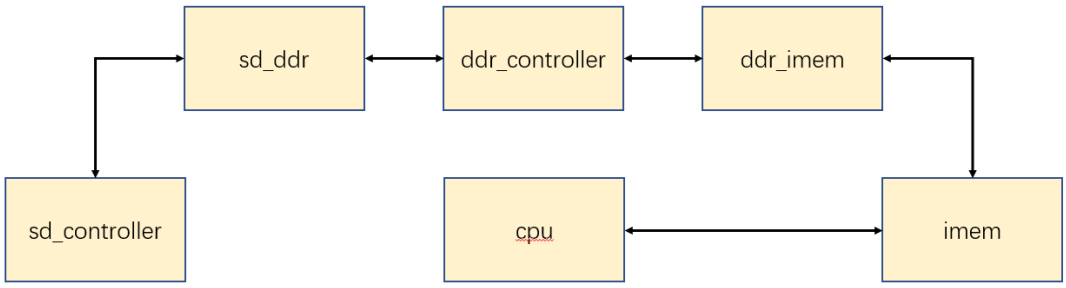
SD 卡中存放流水线 CPU 的二进制流，以及用户程序，N4 板上电自动完成如下的任务：

1） 采用跳线的方式，FPGA 自动从 SD 卡中获取流水线 CPU 的二进制流,并运行该二进制流，使 FPGA 成为 CPU。

2） CPU 再按照三级存储的方式访问 SDRAM，再由 SDRAM 从 SD 卡中把用户程序的目标代码调入到 SDRAM，再由 CPU 把 SDRAM 中的用户程序目标代码调入到片内 CACHE 加以运行。

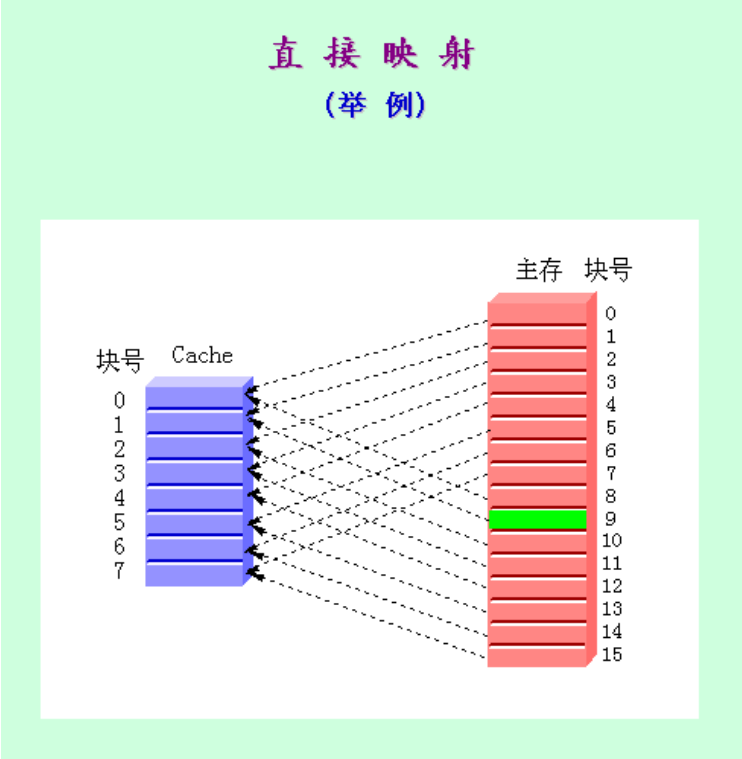
二、系统整体设计

2.1. 系统设计总体模块图



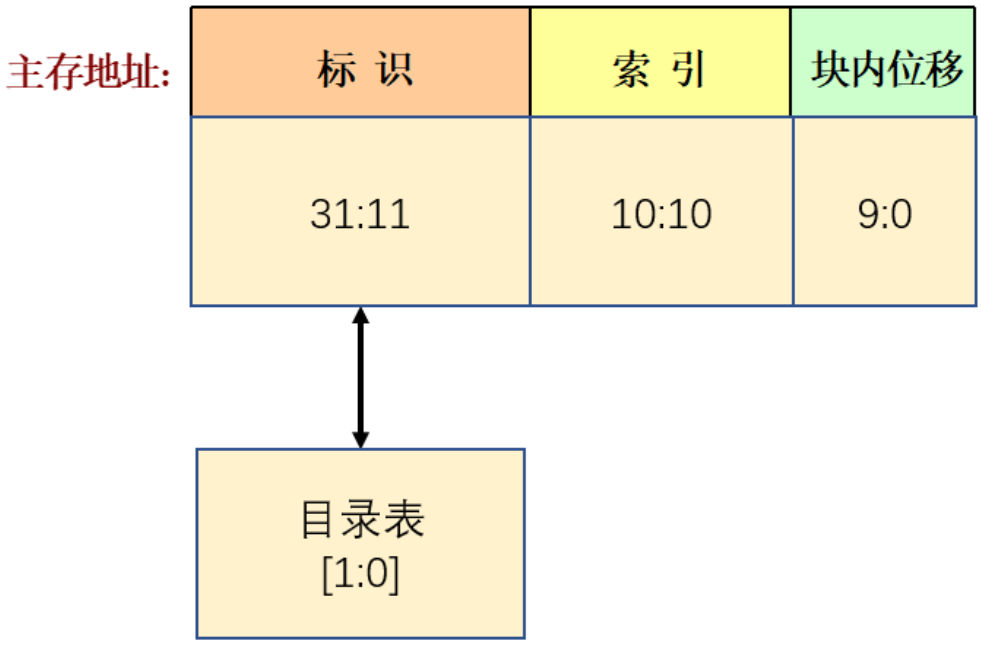
2.2. 三级存储子系统设计

为了简化模型，这里采用的映像规则是直接映像。



我设计的模型中，一个 CPU 地址对应 8bits 数据，一个 Cache 地址对应 32bits 数据。每一个 Cache 块的大小为 1024 个字，因此块内位移是 10 位。

两块 Cache 块，因此索引位是 1 位。
剩下的 21 位作为标识位。但是，出于简化模型的目的，这里假设主存的大小只有 4096 个字，因此只采用 31：30 两位作为标识来判断数据是否有效。将 31：30 上的两位数据送入目录表进行判断，如果结果是 0，表示不命中。
遇见不命中的情况，需要考虑替换算法。由于我采用的是直接映像，只有 1 块 Cache，直接替换目标 Cache 即可。
模型如下图所示：



2.3. 系统工作数据流说明

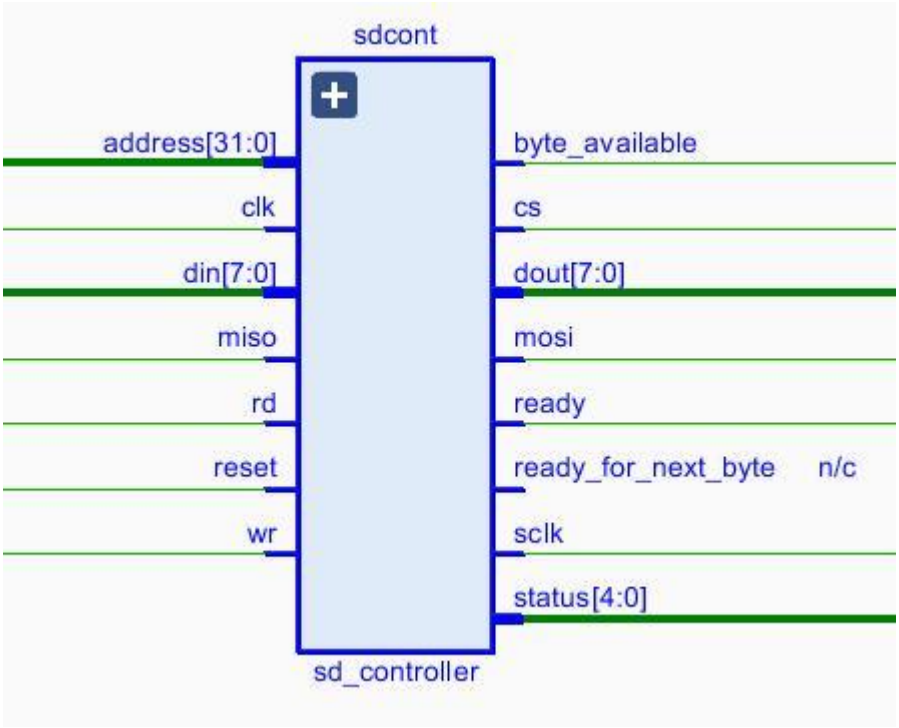
当 CPU 发起一次 Cache 访存时，Cache 块的控制器如果发现给定的内存地址不在 Cache 块上时，会发送信号通知 CPU，使得 CPU 自动进入阻塞的状态，等待 Cache 块的控制器发送一个就绪信号后才能继续执行。此时 Cache 块的控制器向 DDR 控制器发送读取数据请求，同时将自己陷入等待 DDR 控制器信号的状态。DDR 进一步向 SD 控制器发送数据请求信号，SD 控制器使能 SD 驱动来读取 SD 卡上指定扇区的数据。等到 SD 卡将指定扇区的数据读取完毕后发送信号通知 DDR 开始读取 SD 卡暂存在 SD_MEM 中的数据。DDR 数据就绪后发送信号通知 Cache 控制器开始读取 DDR 中的数据。Cache 数据就绪后，Cache 块的控制器发送数据就绪信号通知 CPU 进入正常运转状态。至此，数据流在三级存储中的整个过程结束。

三、具体实现方法

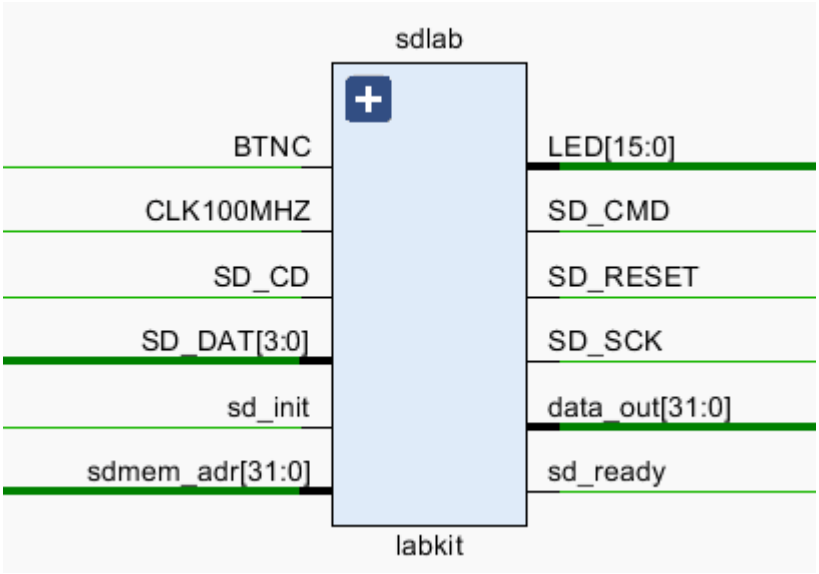
3.1 SD 模块

sd_controller 模块中使用 SPI 协议进行数据传输，每次与 SD 卡传输的最小单位是一个扇区，

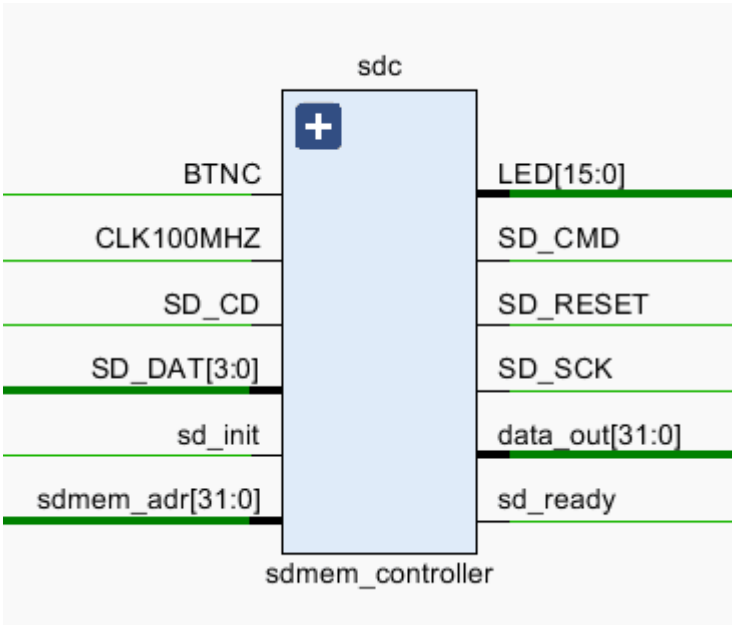
在数据传输的过程中，当 byte_available 信号有效时表示 dout[7:0]中的 8bits 数据需要被上一层模块取走。



Labkit 模块是 sd_controller 的上一层模块，将 sd_controller 的数据存入 sdmem 中。

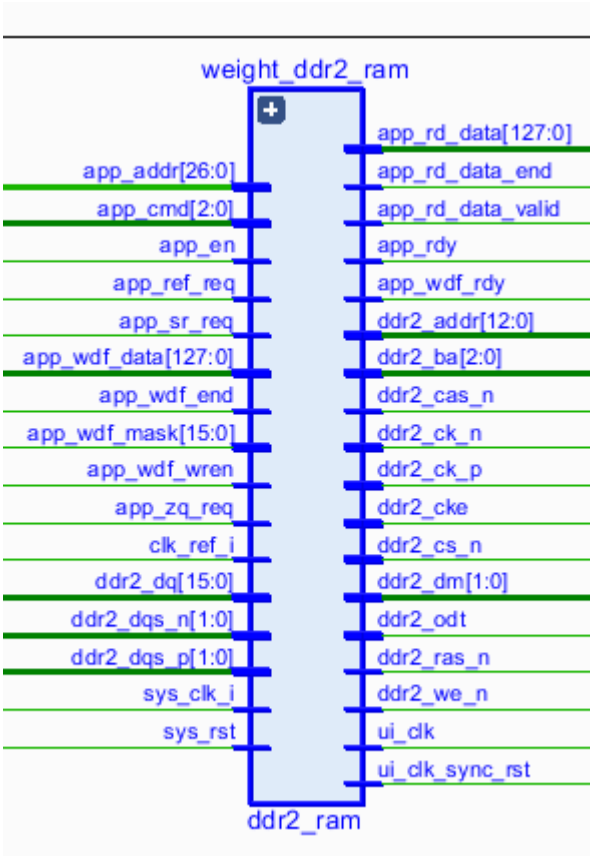


Sdmem_controller 将 labkit 封装成 sdmem，在 sd_init 信号发出后，开始使能 sd 驱动读取 sd 卡的数据。在 sd_ready 信号有效之后就可以直接把 sdmem_controller 当成 mem 使用，输入地址 sdmem_adr，获取数据 data_out。

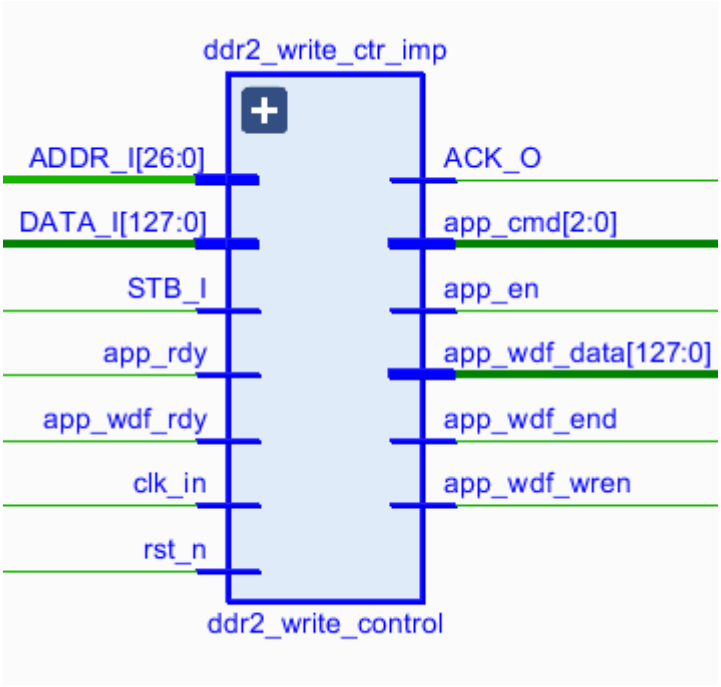


3.2 DDR 模块

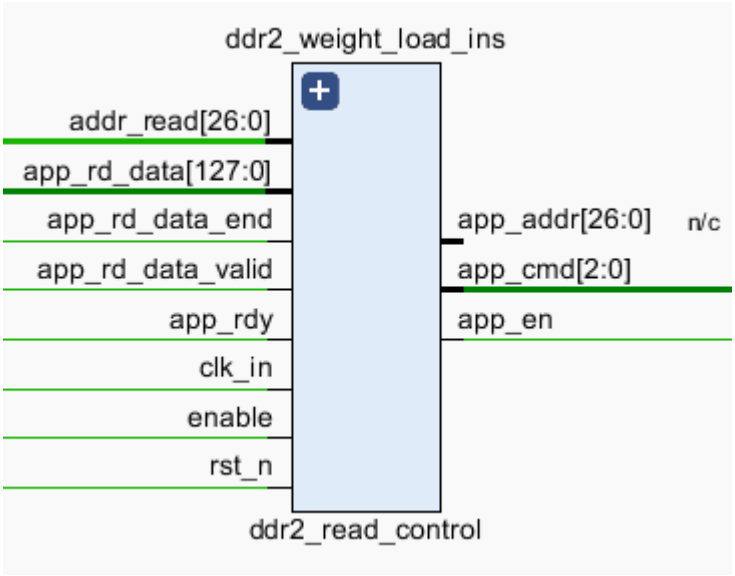
ddr2_ram，控制 ddr2 读写的 ip 核



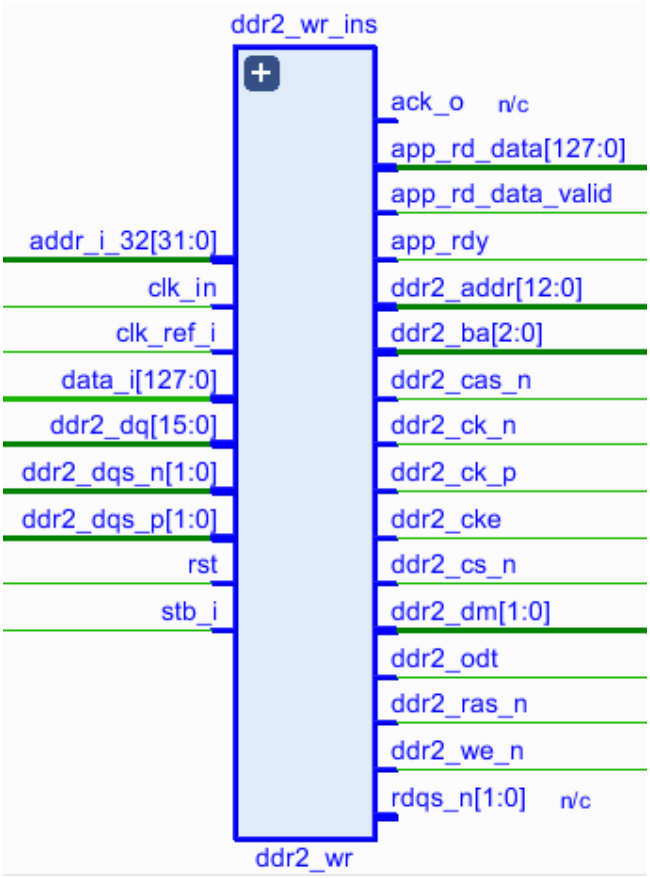
ddr2_write_control,控制 ddr2 写状态。



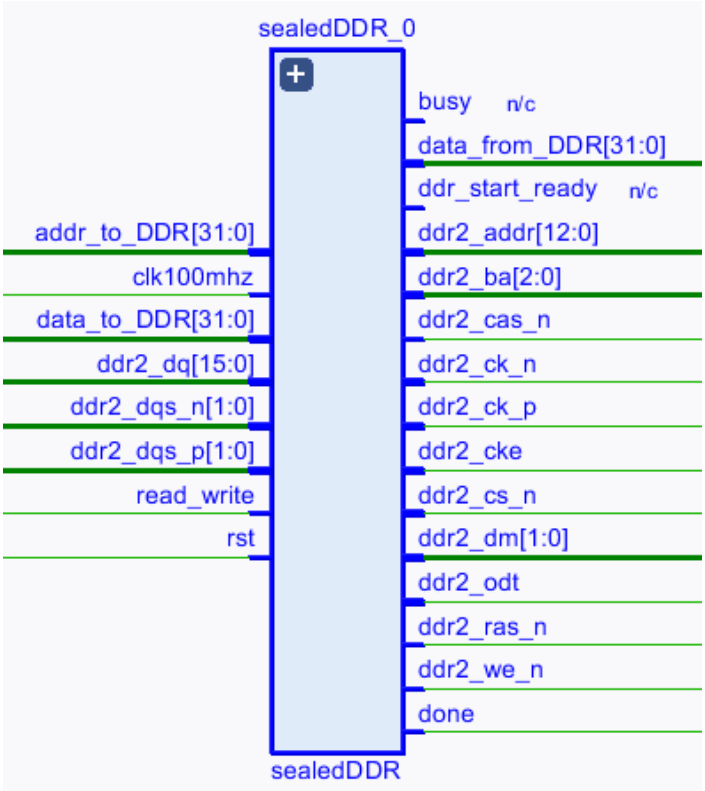
ddr2_read_control,控制 ddr2 读状态。



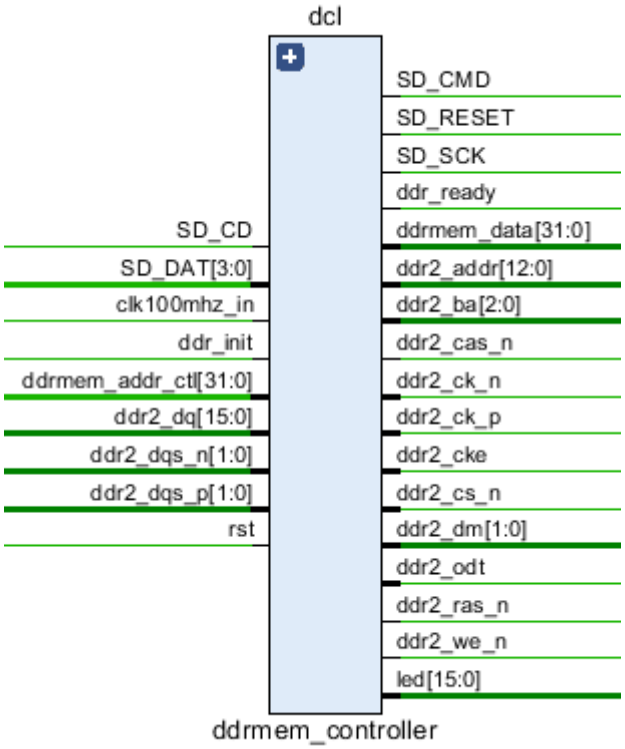
ddr2_wr,将上述三个模块合并起来，一个地址可以读写 128bits 数据。



sealedDDR 将 DDR2 读写控制器封装起来，通过判断 busy 和 done 信号来进行读写。地址为 addr_to_DDR,读出数据为 data_from_DDR,写入数据为 data_to_DDR

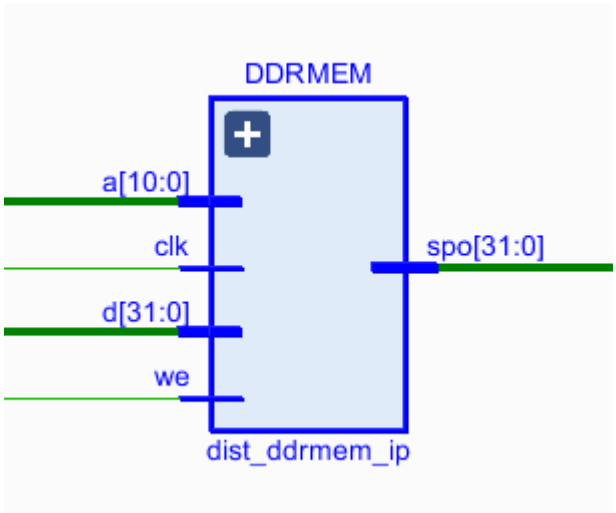


ddrmem_controller，将 sdmem_controller 与 seadDDR 链接起来，将数据从 DDR 读写到 Dist_ddrmem_ip 中，将整体封成一个与 Cache 功能类似的模块，供 CPU 使用



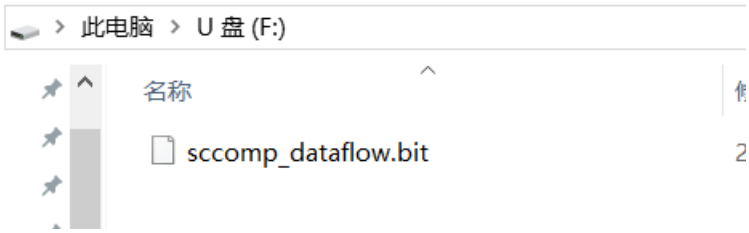
3.3 Cache 模块

Dist_ddrmem_ip，通过和 ddrmem_controller 进行通信来读取数据。

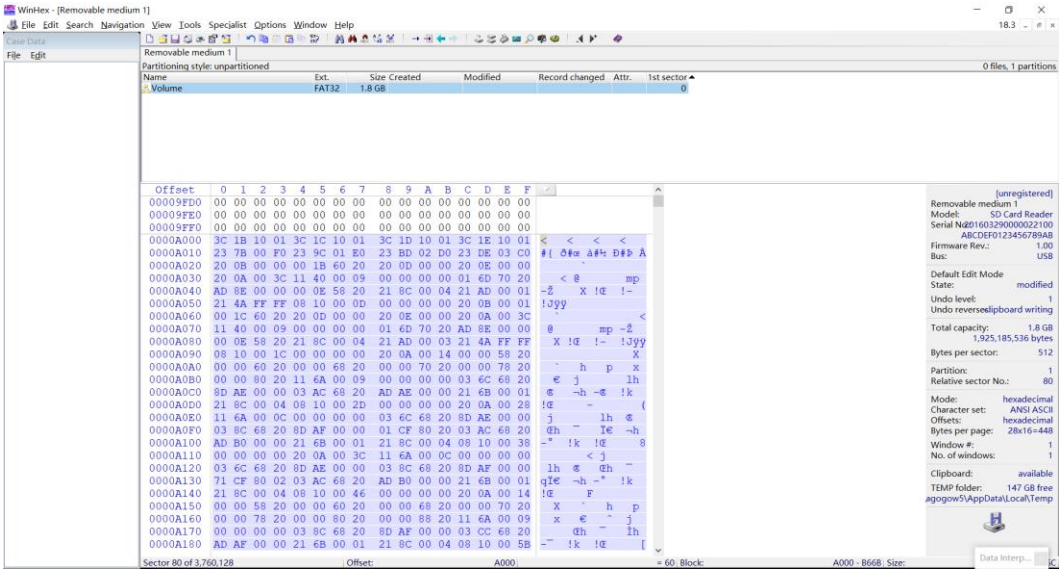


四、实际运行验证

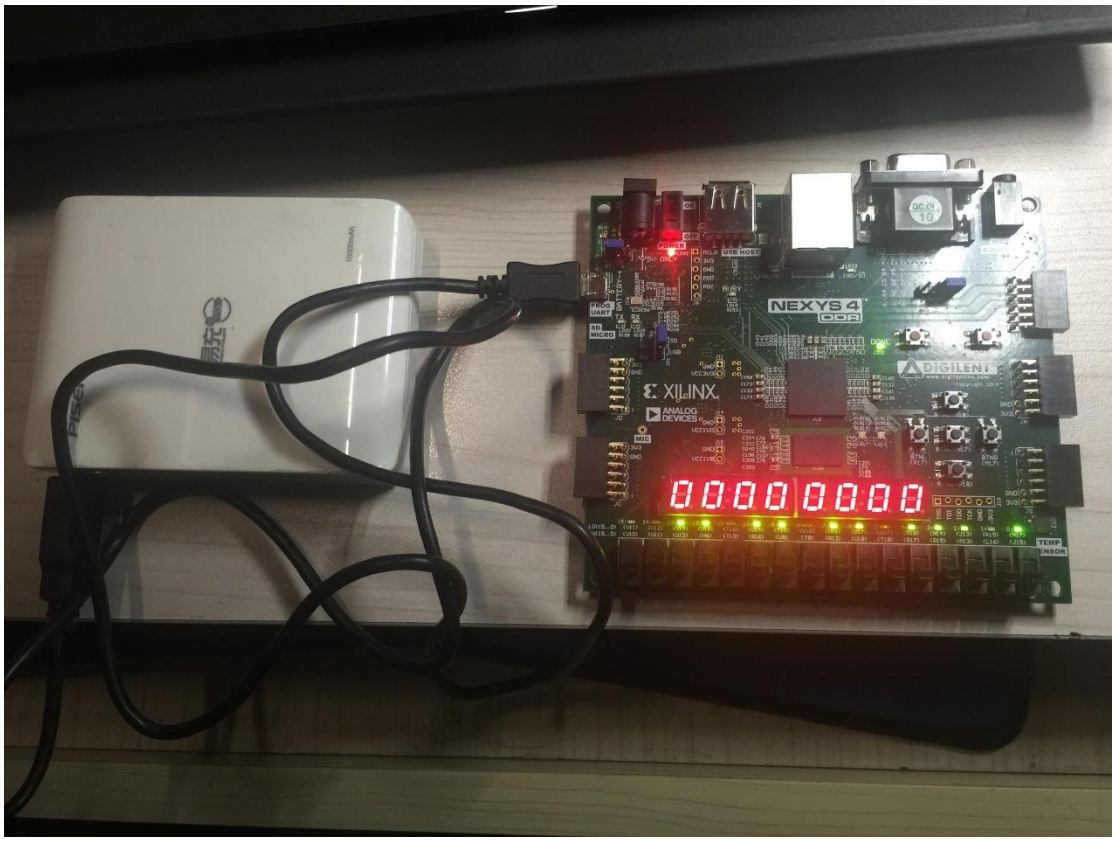
首先将 SD 卡格式化为 FAT32，然后将 CPU 的 bit 流文件存放在 SD 卡的根目录下。



使用 winhex 将 CPU 需要执行的指令写从 80 号扇区开始写入。



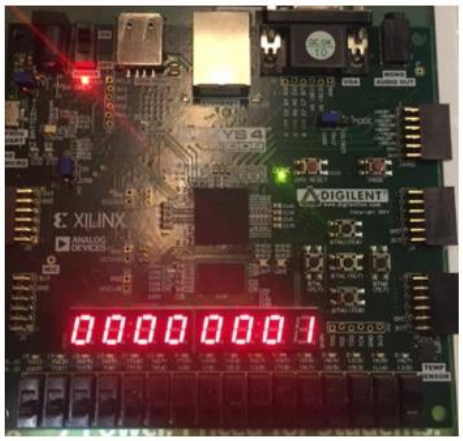
连接上充电宝，将 SD 卡插入卡槽，切换跳线，打开开关，运行状态如下：



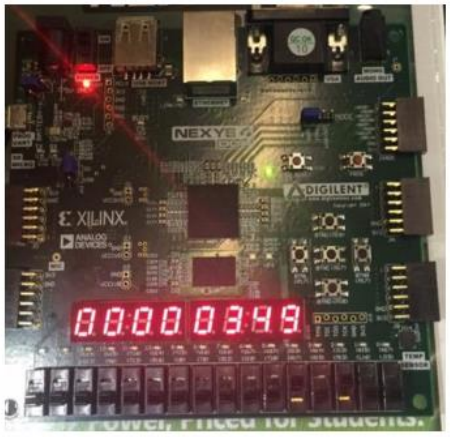
数组 C 的部分结果:

C[0]

C[1]

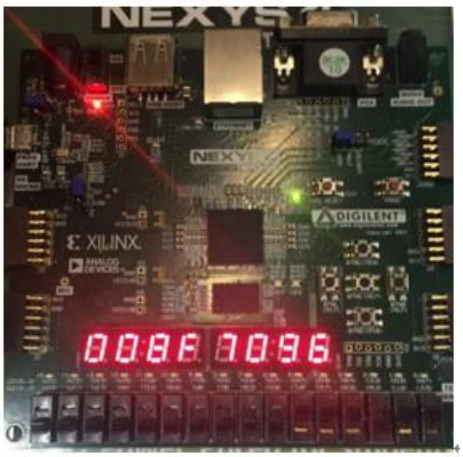


C[20]



C[40]

C[59]



数组 D 的部分结果:

D[0]

D[1]

D[20]

D[40]

D[59]

MARS 中数组 C 与 D 的结果

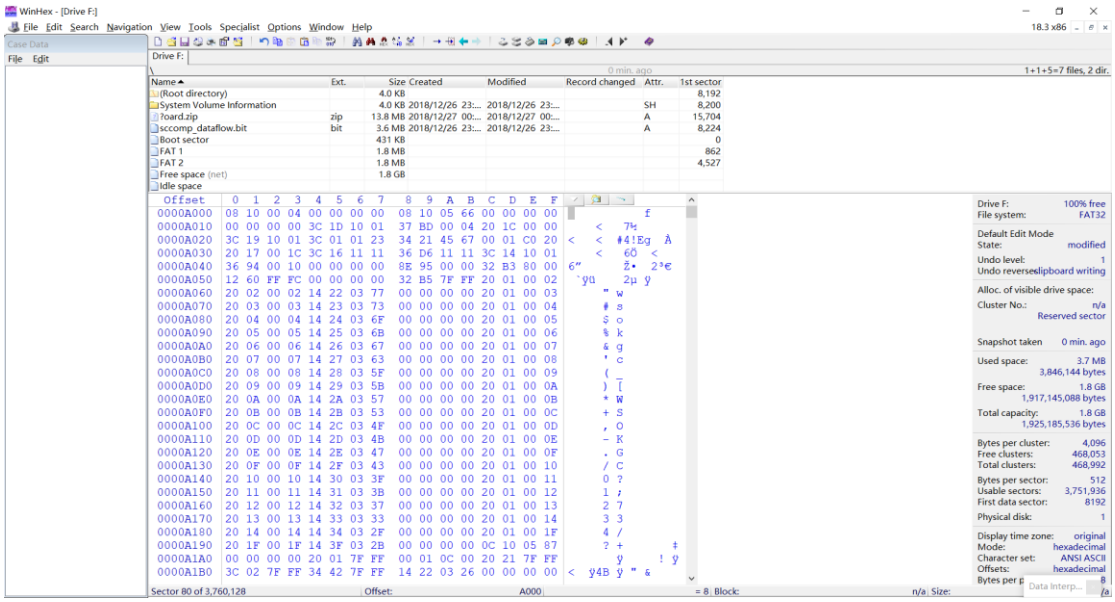
C:

0x100102c0	0x000012b5	0x00001360	0x0000140e	0x000014bf	0x00000000	0x00000001	0x00000003	0x00000006
0x100102e0	0x0000000a	0x0000000f	0x00000015	0x0000001c	0x00000024	0x0000002d	0x00000037	0x00000042
0x10010300	0x0000004e	0x0000005b	0x00000069	0x00000078	0x00000088	0x00000099	0x000000ab	0x000000be
0x10010320	0x00000349	0x0000039d	0x000003f5	0x00000451	0x000004b1	0x00000515	0x0000057d	0x000005e9
0x10010340	0x00000659	0x000006cd	0x00000745	0x000007c1	0x00000841	0x000008c5	0x0000094d	0x000009d9
0x10010360	0x00000a69	0x00000afd	0x00000b95	0x00000c31	0x001ecae4	0x0021f2b8	0x0025571a	0x0028fafe
0x10010380	0x002ce16a	0x00310d76	0x0035824c	0x003a4328	0x003f5358	0x0044b63c	0x004a6f46	0x005081fa
0x100103a0	0x0056f1ee	0x005dc2ca	0x0064f848	0x006c9634	0x0074a06c	0x007d1ae0	0x00860992	0x008f7096

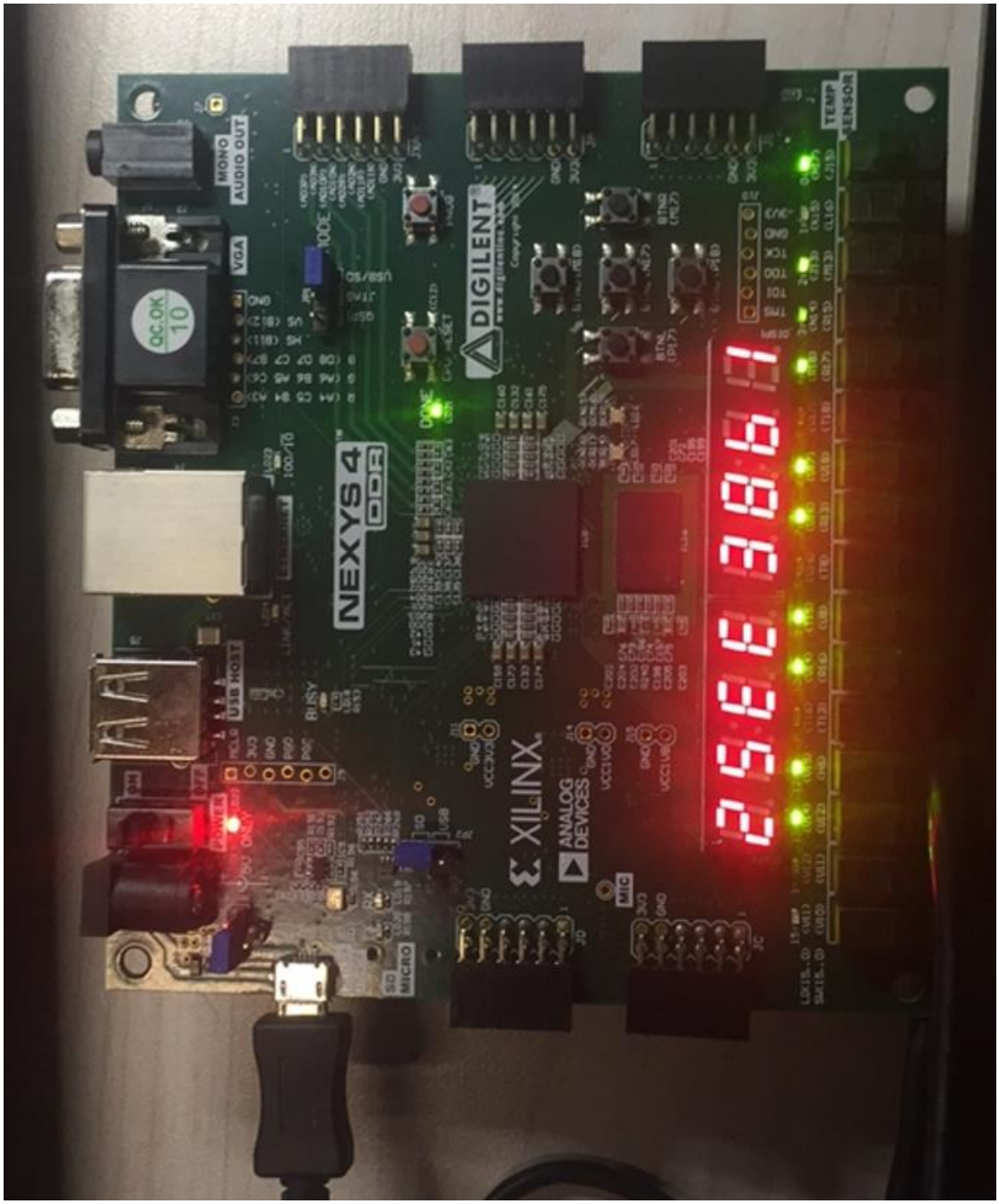
D:

0x100103c0	0x00000001	0x00000004	0x0000000a	0x00000013	0x0000001f	0x0000002e	0x00000040	0x00000055
0x100103e0	0x0000006d	0x00000088	0x000000a6	0x000000c7	0x000000eb	0x00000112	0x0000013c	0x00000169
0x10010400	0x00000199	0x000001cc	0x00000202	0x0000023b	0x0002b1e2	0x000342ab	0x0003e921	0x0004a754
0x10010420	0x00057f6c	0x000673a9	0x00078663	0x0008ba0a	0x000a1126	0x000b8e57	0x000d3455	0x000f05f0
0x10010440	0x00110610	0x001337b5	0x00159df7	0x00183c06	0x001b152a	0x001e2cc3	0x00218649	0x0025254c
0x10010460	0x280471d4	0x56a9f140	0x8b480d3c	0xc67776d2	0x08dc092e	0x532551ac	0xa60f1b10	0x0261fbe8
0x10010480	0x68f3e818	0xdaa8c590	0x5873042c	0xe35438be	0x7c5dbb42	0x24b1483c	0xdd81a540	0xa81348a4
0x100104a0	0x85bd045c	0x77e8b400	0x8013edfc	0x9fd0b7ea	0x00000000	0x00000000	0x00000000	0x00000000

将 SD 卡从卡槽中取出，打开 WinHEX 更换 SD 卡上 CPU 运行指令（使用上学期计算机组成原理课程中，验证 54 条通路的指令），依旧是从 80 号扇区开始读写。



再将 SD 卡插回卡槽，按下 rst，结果如下：

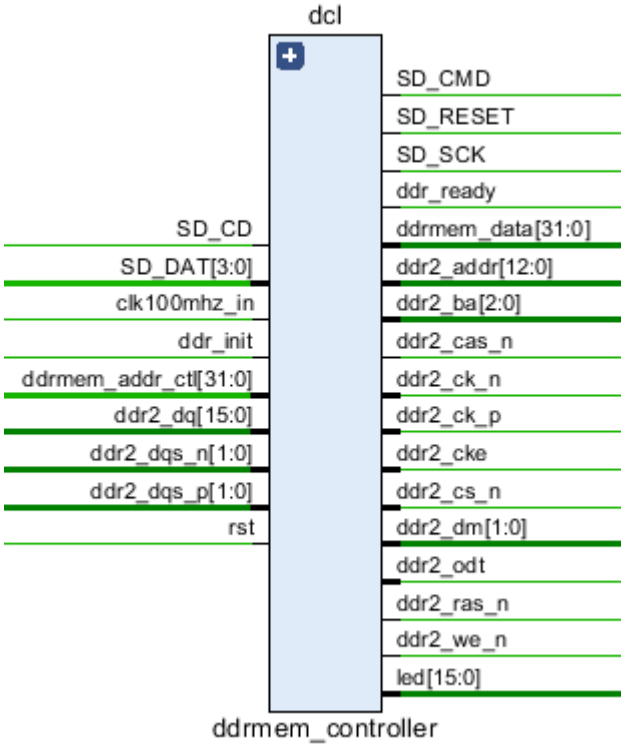


程序指令正常运行。
可以得出结论，三级存储系统能够正常运行。

五、源程序代码与说明

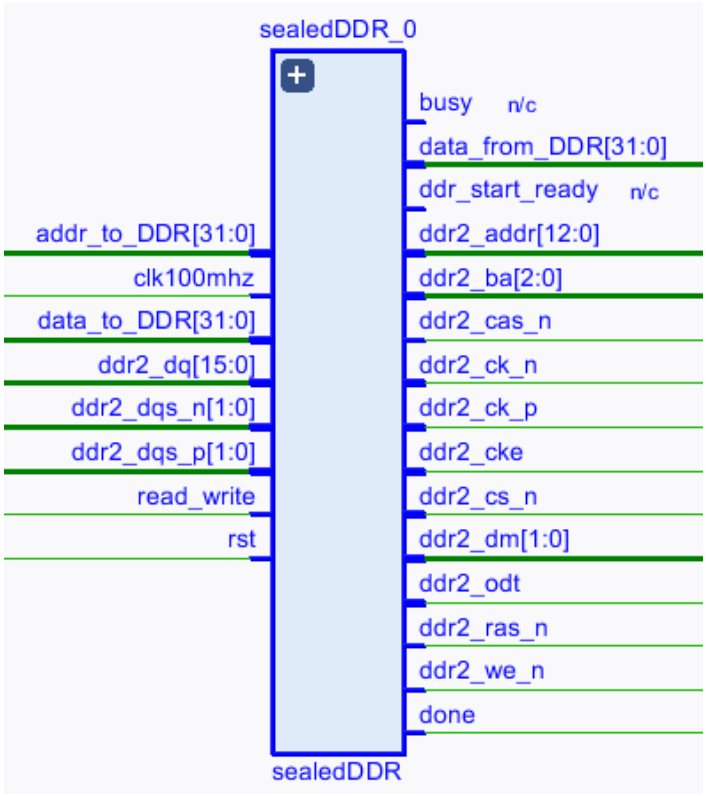
5.1 ddrmem_controller.v

将 sd 控制器，ddr 控制器以及 cache 封装起来，等效成一个 mem。



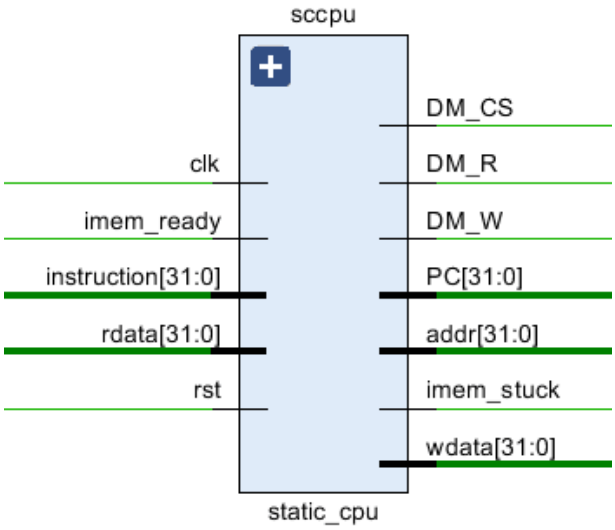
5.2 sealedDDR.v

将 DDR 的 ip，读写控制器封装起来。



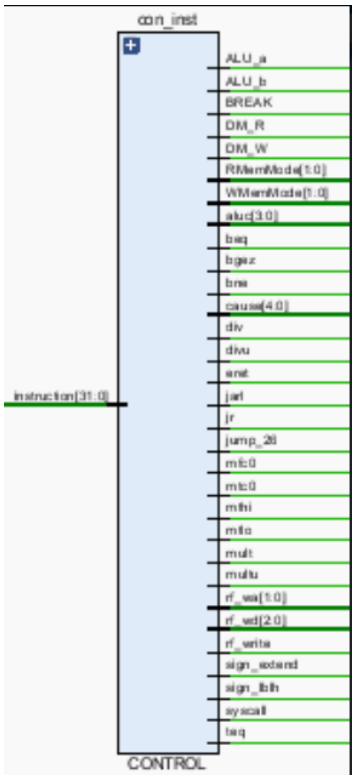
5.3 static_cpu.v

静态流水线 CPU 的主体部分。



5.4 CONTROL.v

静态流水线中信号控制器。



5.5 sccomp_dataflow.v

顶层结构，将三级存储系统与 CPU 结合起来。