1. 关于页表的问题：

页表的引用计数移动到页表权能内部的额外域上。它记录本页表被cons到了多少个槽位中这样，Pgreg就彻底消失了，这样方便得多。X86这个port和c66x这个port暂时扔下不管它。以后再说吧，事情太多没时间做这个的。

1. 关于权能的引用计数：

权能的引用计数移动到根权能内部。这样，不存在链状反应了。

Type\_Ref改成Type\_Root

Parent改成Parent\_Refcnt。

如果这个权能是root，那么parent里面放的就是refcnt，否则就是指针。

这样一个权能的delegation次数限制就被解除了。

1. 关于三种MPU上的页表实现

MPU有三个种类。

|  |  |  |
| --- | --- | --- |
|  | 单核 | 多核 |
| 子区域型MPU | 目前A7M的实现即可，有指向顶层的指针和一系列制约。 | 都是动态填充的，静态页优先替换动态页，替换完了才替换静态页。页表只有小于等于子区域的数量才可映射。 |
| 基址长度型MPU | 采取大虚拟页表的形式。整个页表就是一体化的，然后事实上采用堆管理各个子区域，只是虚拟出页表的假象。 | 都是动态填充的，静态页优先替换动态页，替换完了才替换静态页。页表只有小于等于子区域的数量才可映射，另外每一个页都是一个区域（与区块型一致，我们放弃了这好处） |
| 区块型MPU | 有指向顶层的指针和一系列制约。无页（不仅是无页目录）才可以解除页目录映射。 | 都是动态填充的，静态页优先替换动态页，替换完了才替换静态页。页表只有小于等于子区域的数量才可映射，另外每一个页都是一个区域。 |

1. 变更CortexM为A7M。全部的宏、标识符、文件名称都要换掉。

另外除去STM32的HAL依赖。

对Cortex-M增加IACCVIOL的动态页支持。动态支持这样实现：一旦发生IACCVIOL，我们知道那个地址就在用户态的栈里面。但是我们并不知道栈是否安全，也即内核是否能够访问它。此时，对栈做页表查询即可，判断那位置内核是否能够访问。如果在页表中找到了那个地方，而且那地方登录在了可访问内存区里面，我们就知道内核可以安全访问那个地址了。但是，对于生成器，我们要求第一个代码区和第一个数据区总是静态分配的。

1. 对于Cortex-M，当前我们仅考虑单核的情况，暂不考虑多核。这是由于多核下我们不可能保持那个对顶层指针，一个核的static mapping也无法强制施加给其他核，因此即便是static的region也一定是从handler来填充的。因为UNSTKERR无论如何也无法给出正确的地址，在多核上我们就只有这样了：令第一个线程在进入时LDR PC所在的address和SP所在的address，强制MPU完成载入，然后再跳转运行，对每个核都是这样。中间可能发生stkerr，我们直接忽略即可，事实上我们也是这么做的。

单核下，对于ARMv8-M，还可以采取更灵活的页表组织方法充分利用其MPU的细粒度分配能力。

1. RVM虚拟机的支持。究竟要不要支持，要怎么支持，还有posix。

现在已经决定了，RVM与RME不合并成一个工程。

要不要例子程序，例子程序怎么做，需要额外决定。

例子程序是手写还是？或者说，我们已经过了提供例子程序的日子了？

1. 现在的分配器已经决定不去猜测各个分区的大小而由用户手动指定。但是，如果能够调动compiler driver知晓分区的大小就太好了。这必然是未来的方向，也是我们全套中间件策略的必经之路。我们永远要记得简化用户的使用体验。越简单，越好。
2. 在分配器中我们已经决定不支持共享内存。共享内存可以说十分麻烦，有凑整问题，还要考虑到attribute是否一致的问题。有鉴于此，我们暂时把这个问题留给用户，或者说留给上级配置器解决，因为上级配置器更加具备全局观，能够从一个更高的角度来判断谁和谁应该如何通信。
3. 在分配器中支持内核权能。当前是不支持的。
4. 在分配器中支持追加SDRAM内存。当前是不支持的。
5. 在分配器中支持以前的工程的迁移。已经被生成的工程可以被再度读取并重新生成文件。这种状况下，我们不去动用户的文件，因为它们可能已经被大幅度修改过