```
#!/usr/bin/env python

from dsx.hardware import *
from soclib import *

class ClusteredNoirqMulti(SoclibGenericArchitecture):
    """
    Parameters:
     - min_latency: Vgmn's latency
     - cpus: Cpus per cluster in a list
     - rams: Rams per cluster in a list

    Exports:
     - vgmn: global vgmn interconnect
     - clust[]: cluster's interconnect; contains:
       - cpu[]: cpu list for cluster
       - cram[]: cached ram list for cluster
       - uram[]: uncached ram list for cluster
       - locks: ramlocks
     - cpu[]: global cpu list
     - cram[]: global cached ram list
     - uram[]: global uncached ram list
     - locks: global ramlocks list
    """
    def cluster(self, clustno, ncpu, nram):
        ic = LocalCrossbar('lc%d'%clustno)

        cpu = []
        cache = []
        for n in range(ncpu):
            c = Mips('mips%d_%d'%(clustno,n))
            xc = Xcache('cache%d_%d'%(clustno,n))
            c.cache // xc.cache

            cpu.append(c)
            cache.append(xc)

            xc.vci // ic.getTarget()

        ic.cpu = cpu
        ic.cache = cache

        ram = []
        uram = []
        cram = []
        for n in range(nram):
            cr = Segment('cram%d'%clustno, __remplir__)
            ur = Segment('uram%d'%clustno, __remplir__)
            uram.append(ur)
            cram.append(cr)
            r = MultiRam('ram%d'%clustno, cr, ur)

            r.vci // ic.getInit()
            ram.append(r)
        ic.ram = ram
        ic.uram = uram
        ic.cram = cram

        ic.locks = Locks('locks%d'%clustno)
        ic.locks.vci // ic.getInit()

        return ic

    def architecture(self):
        self.vgmn = Vgmn('vgmn', self.getParam(__remplir__))
        self.setBase(self.vgmn)
```

```
        nram = self.getParam('rams')
        ncpu = self.getParam('cpus')
        assert(len(nram) == len(ncpu))

        ncluster = len(nram)

        self.clust = []
        for n, nc, nr in zip(range(ncluster), ncpu, nram):
            ic = self.cluster(n, nc, nr)
            self.clust.append(ic)
            ic.__remplir__ // self.vgmn.getBoth()

        self.clust[0].ram[0].addSegment(
            Segment('reset', Cached, addr=0xbfc00000),
            Segment('excep', Cached, addr=0x80000080))

        # export global lists
        add = lambda x,y: x+y
        self.ram = reduce(add, [self.clust[i].ram for i in range(ncluster)])
        self.uram = reduce(add, [self.clust[i].uram for i in range(ncluster)])
        self.cram = reduce(add, [self.clust[i].cram for i in range(ncluster)])
        self.cpu = reduce(add, [self.clust[i].cpu for i in range(ncluster)])
        self.locks = [self.clust[i].locks for i in range(ncluster)]

# Ceci permet de tester l'architecture:
# Le bout de code dans le if ne sera exécuté que si vous faites ./clustered_noirq_multi.py
# et pas si vous importez clustered_noirq_multi depuis une autre description.

if __name__ == '__main__':
    hard = ClusteredNoirqMulti( cpus = [1, 2, 2, 1],
                                rams = [1, 1, 1, 1],
                                min_latency = 10 )
    hard.dispTree()
    print hard.cpu
    print hard.ram
    print hard.cram
    print hard.uram
    print hard.locks
    hard.generate(Caba())
```