

```

#!/usr/bin/env python

import dsx
import soclib

def VgmnNoirqMono():
    pf = soclib.Architecture(cell_size = 4,
                             plen_size = 8,
                             addr_size = 32,
                             rerror_size = 1,
                             clen_size = 1,
                             rflag_size = 1,
                             srcid_size = 8,
                             pktid_size = 1,
                             trdid_size = 1,
                             wrplen_size = 1
                             )

    pf.create('common:mapping_table',
              'mapping_table',
              addr_bits = [8],
              srcid_bits = [8],
              cacheability_mask = 0xc00000)
    pf.create('common:loader', 'loader')

    vgmn = pf.create('caba:vci_vgmn', 'vgmn0', *** remplir ***)

    cpu = pf.create('caba:vci_xcache_wrapper', 'cpu0',
                    iss_t = "common:mips32el",
                    ident = 0,
                    icache_ways = 1,
                    icache_sets = 16,
                    icache_words = 8,
                    dcache_ways = 1,
                    dcache_sets = 16,
                    dcache_words = 8)

    vgmn.to_initiator.new() // cpu.vci

# Generation des coprocesseurs, commentée pour l'instant
# # Ici, on récupère l'implémentation matérielle de tg, qui va
# # nous permettre d'instancier le coprocesseur et son controlleur
#
# tg = dsx.TaskModel.getByName('tg').getImpl(soclib.HwTask)
# # La création nous retourne les deux composants créées.
# ctrl, coproc = tg.instanciate(arch, 'tg0')
# # Il reste à donner une adresse au controlleur, et le connecter
# # à l'interconnect (attention il a deux ports)
# ctrl.addSegment('tg_ctrl', 0x70400000, 0x100, False)
# ctrl.vci_initiator // vgmn.to_initiator.new()
# ctrl.vci_target // vgmn.to_target.new()
#
# # pareil avec ramdac
# ramdac = dsx.TaskModel.getByName('ramdac').getImpl(soclib.HwTask)
# ctrl, coproc = ramdac.instanciate(arch, 'ramdac0')
# ctrl.addSegment('ramdac_ctrl', 0x71400000, 0x100, False)
# ctrl.vci_initiator // vgmn.to_initiator.new()
# ctrl.vci_target // vgmn.to_target.new()

for i in range(2):
    ram = pf.create('caba:vci_ram', 'ram%d%i' % i)
    base = 0x10000000*i+0x10000000
    ram.addSegment('cram%d%i' % i, base, 0x100000, True)
    ram.addSegment('uram%d%i' % i, base + 0x400000, 0x100000, False)
    ram.vci // vgmn.to_target.new()
    ram.addSegment('boot', *** remplir ***) # Mips boot address, 0x1000 octets, cacheable

```

```

    ram.addSegment('excep', *** remplir ***) # Mips exception address, 0x1000 octets, cachea

    tty = pf.create('caba:vci_multi_tty', 'tty0', names = ['tty0'])
    tty.addSegment('tty0', 0x90000000, 0x20, False)
    tty.vci // vgmN.to_target.new()

    return pf

# This is a python quirk to generate the platform
# if this file is directly called, but only export
# methods if imported from somewhere else

if __name__ == '__main__':
    VgmNNoirqMono().generate(soclib.PfDriver())

```