Generic Architectures

The concept

`Generic Architecture` means we will define some SoC hardware architecture with few variable parameters. The main goal is to change number of Cpus or memory banks, but not main architecture like interconnect choice.

Implementation

An architecture is defined in a python class inherited from `Architecture` (defined in `soclib`). Method `architecture` will create contents of the SoC: components creation, connections, utilities (like MappingTable?).

You'll have to tell the architecture framework where is the lowest-level component, most probably the base interconnect. This must be archieved by calling `self.setBase(component)`.

Parameters

Anywhere in `architecture` method, you may call `self.getParam(name)` to get an architecture instance parameter.

- name is the parameter name
- default value is optionnal
    - user-specified value overrides default value
    - if default value is not given when calling `getParam`, user-specified argument becomes mandatory.
- default value must be specified in class-global dictionnary 'defaults'

Examples

Basic example

We may define a basic architecture with no parameters:

```
class OneLevel(GenericArchitecture):
    def architecture(self):
        vgmn = Vgmn( ?vgmn? )

        mips0 = Mips( ?mips0? )
        cache0 = Xcache( ?cache0? )

        # Cache ports connexion
        cache0.cache // mips0.cache
        vgmn.getTarget() // cache0.vci
```

```
        reset = Segment( ?reset?,
                         address = 0xbfc00000,
                         type = Cached )
        excep = Segment( ?excep?,
                         address = 0x80000080,
                         type = Cached )
        code = Segment( ?code?, type = Cached )
        data = Segment( ?data?, type = Uncached )
        ram = MultiRam( ?ram?, reset, excep, code, data )

        # Connections
        vgmn.getInit() // ram.vci

        # Declare base component
        self.setBase(vgmn)

        # Add configuration utilities
        self.setTimeBase()
        self.setConfig(?mapping_table?, MappingTable())
```

As you may want to refer to some objects later on, you may set attributes to self:

```
        ...
        self.ram = MultiRam( ?ram?, reset, code, data )
```

Parametrized example

We will define a one-level architecture using a Vgmn, with two parameters: number of Cpus (`ncpu`) and number of Ram components (`nram`). We will set `nram` default value to 1.

We'll add in `ram0` (which always exists) two more segments: `excep` and `reset`.

We'll those attributes in instance objet (accessible through `object.name`):

- `vgmn`: the interconnect
- `cpu`: a list of all the cpus
- `ram`: a list of all the ram chips
- `cram`: a list of all the cached ram segments
- `uram`: a list of all the uncached ram segments

This way, we'll be able to refer to `platform.ram[2]` or `platform.cpu[0]`.

```
class Parametrized(GenericArchitecture):
    defaults = {
        'nram': 1,
    }
    def architecture(self):
        self.vgmn = Vgmn( ?vgmn? )

        self.cpu = []
        for i in range(self.getParam('ncpu')):
            cpu = Mips( ?mips%d?%i )
            cache = Xcache( ?cache%d?%i )

            # Cache ports connexion
            cache.cache // cpu.cache
            vgmn.getTarget() // cache.vci
            self.cpu.append( cpu )

        self.ram = []
```

```
self.cram = []
self.uram = []
for i in range(self.getParam('nram')):
    cram = Segment( ?cram%d?%i, type = Cached )
    uram = Segment( ?uram%d?%i, type = Uncached )
    ram = MultiRam( ?ram%d?%i, cram, uram)

    vgmn.getInit() // ram.vci
    self.ram.append( ram )
    self.cram.append( cram )
    self.uram.append( uram )

self.reset = Segment( ?reset', type = Cached )
self.excep = Segment( ?excep', type = Cached )
self.ram[0].addSegment( self.reset, self.excep )

self.setBase( self.vgmn )

self.setTimeBase()
self.setConfig(?mapping_table?, MappingTable())
```

In this platform

- argument `nram` is optional with default value being 1
- arguments `ncpu` is mandatory