Mapping Info Structure

Dsx-vm needs to have a way to communicate with the operating system to specify how to map code, data and taks on the architecture. To this end, an xml data structure has been defined, and is presented below. This xml file is produced by Dsx-vm and read by the operating system and used in particular for page tables generation and thread mapping on processors.

TODO: hypothesis: clustered_architecture, etc.

Document XML Structure

In order to simplify the parsing of the document the order between elements has to be respected. This document contain two main descriptions:

- a minimal description of the hardware : all elements described must exist on the targeted architecture. Non described element will not be used by the soft.
- a description of the soft mapping on the hardware

The mapping_info element

The two descriptions are regrouped in the `mapping_info` element, which contain the following attributes:

- `name`: the name of the mapping described
- `signature`: a way to check that we are handling a mapping info structure, must be equal to "0xdeadbeef"
- `clusters`: the number of clusters of the targeted architecture
- `vpsaces`: the number of vspaces defined. This number is typically equal to the number of applications

The `mapping_info` element contains the following elements:

- a `clusterset` element, describing the targeted hardware
- a `globalset` element, describing the global segments, i.e. segments replicated in all virtual spaces (typically the kernel code)
- a `vspaceset` element, describing the mapping for all virtual spaces

The clusterset (cluster set) element

This element contains no attribute, and contains the following elements:

- 1 to *n* `cluster` element(s)

The cluster element

This element contains the following attribute:

- `index`: the cluster index, or *id*. It is unique for each cluster and must range from 0 to *n - 1* (n being the number of cluster). The cluster index make the correspondance with the cluster id in the architecture.

The cluster element contains the following elements:

- 0 to *n* `pseg` element(s)

- 0 to *n* `proc` element(s).
- 0 to *n* `periph` element(s)

The pseg (physical segment) element

This element contains the following attributes:

- `name`: name of the segment. The name of the segment is not related to any segment name in the architecture. It is used to associate virtual segments on physical segments inside the mapping info file.
- `base`: base of the physical segment.
- `type`: type of the segment. It can be one of `RAM`, `ROM`, `PERI`
- `length`: size of the segment

The proc (processor) element

This element contains the following attribute:

- `index`: the processor index, or *id*. It is unique for each processor and must range from 0 to *n - 1* inside the cluster (vrai ! c'est pas un id global.). The index corresponds to the processor id in the architecture.

The proc element contains the following element:

- 0 to *n* `irq` element(s).

The irq (interrupt request) element

This element describe the routing of the interruptions signals on the hardware. It contains the following attributes:

- `type`: The type of interrupt request. Currently can only be `HARD`.
- `icuid`: the index of the interrupt entry in the hardware icu or xicu.
- `channel`: the peripheral channel from which the interruption is connected (for peripherals with only one channel, it must be set to zero).
- `isr`: the interrupt sub-routine types:
    - `ISR_SWITCH`: Used to schedule the tasks of a processor, channel attribute must be equal to the processor id.
    - `ISR_TTY`: handle the irq emaning from the tty
    - `ISR_DMA`: handle the dma irq
    - `ISR_IOC`: handle the block device interruption
    - `ISR_TIMER`: handle user timer interruption

The periph (peripheral) element

This element contains the following attributes:

- `type`: type of the peripheral. It can be one of :
    - `PERIPH_TYPE_IOC` : block device peripheral
    - `PPERIPH_TYPE_TTY`: tty device peripheral
    - `PPERIPH_TYPE_TIM`: timer device peripheral
    - `PPERIPH_TYPE_DMA`: dma device peripheral
    - `PPERIPH_TYPE_FBF`: frame buffer peripheral
    - `PPERIPH_TYPE_NIC`: nic buffer peripheral

    - ♦ PPERIPH_TYPE_IOB: io bridge peripheral
- psegname: name of the physical segment of the peripheral. This name must be one of the pseg elements name. (pourquoi nom du pseg et pas nom du vseg ?)
- channels: the number of channels in the peripheral (one channel usually corresponds to one processor; an exception is for ttys where one channel is used for the operating system)

The globalset (global set) element

This element contains no attribute, and contains the following elements:

- 0 to *n* vseg element(s)

The vseg (virtual segment) element

This element contains the following attributes:

- name: name of the virtual segment (utilisé quelque part ? Non)
- vbase: base address of the virtual segment in the final binary. Must be page size (4096) aligned.
- psegname: name of the physical segment in which this virtual segment will be mapped. This name must be one of the pseg elements name.
- clusterid: cluster id in which the the psegname have been defined.
- mode: specific properties of the segment. The value is of the form [C_][X_][W_][U_] e.g. C_W_. Each character indicates if the property is selected (letter) or not (underscore). The properties are the following:
    - ♦ C: the segment is cached
    - ♦ X: the segment is executable
    - ♦ W: the segment is writable
    - ♦ U: the segment is accessible is user mode
- ident: (optional) is 1 if the segment is an identity segment, 0 otherwise. An identity virtual segment is a virtual segment whose base address is the same as the one of the physical segment it is mapped on. Generally used for the mapping of vseg on peripherals.

The vseg element contains the following elements:

- 1 to *n* vobj element(s)

The vobj (virtual object) element

This element contains the following attributes:

- name: the name of the virtual object. *utilisé ?* pour obtenir les ressources (mwmr, barrier,..) en mode utilisateur grace a l'appel system vobj_get_vbase(stdio.c).
- type: type of the virtual object. It can be one of the following:
    - ♦ ELF: describe an elf section.
    - ♦ PTAB: reserve memory space for the page table, each vspace must have one vobj of this type.
    - ♦ PERI: for peripheral mapping.
    - ♦ MWMR: describe an mwmr channel, this vobj will be apropriatly initialised by the operating system
    - ♦ LOCK: describe a lock, this vobj will be apropriatly initialised by the operating system
    - ♦ BARIERR: describe a barrier, this vobj will be apropriatly initialised by the operating system
    - ♦ BUFFER: describe a buffer. The content of this vobj is not initialised
- length: size of the virtual object in bytes. The sum of the length of all vobj define the length of the vseg.

- `align`: (optional) logarithm in base 2 of the physical alignemnt required for this segment. For example, a value of 13 means that the segment must be aligned on 0x2000. Be careful when using this attribute, since the size of the vobj will be incremented until the
- `binpath`: path to the file to load in this segment, if any (ELF vobj). This path can be relative or absolute.

The vspaceset (virtual space set) element

The vspaceset element has no attribute and contains the following elements:

- 0 to *n* `vspace` element(s)

The vspace (virtual space) element

This element contains the following attributes:

- `name`: name of the virtual space.
- `startname`: name of the data section of the application (a vspace correspond to an application). This section will contain at his top the table of task function entry.

The vspace element contains the following elements:

- 0 to *n* `vseg` element(s)
- 0 to *n* `task` element(s)

The task element

This element contains the following attributes:

- `name`: name of the task.
- `clusterid`: cluster id in the targeted architecture of the processor on which to execute the task.
- `proclocid`: processor local id on which to execute the task.
- `stackname`: name of the vobj on which to place the task's stack.
- `usetty`: if 1 the task a tty will be reserved to the task, 0 otherwise.
- `startid`: indicate the index in the `startname` vobj for this task function entry.