Mapping Info Structure

Dsx-vm needs to have a way to communicate with the operating system to specify how to map code, data and tasks 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 in the xml file has to be respected. This file describing the mapping information contains two main descriptions:

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

The mapping_info element

The two descriptions are regrouped in a single `mapping_info` element, which contains the following attributes:

- `name`: the name of the mapping described.
- `signature`: a way to check that we are handling a mapping info structure. It must be equal to "0xdeadbeef".
- `clusters`: the number of clusters in 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. the 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 element(s):

- 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* (with *n* being the number of cluster). The cluster index makes the correspondance with the cluster id in the architecture.

The cluster element contains the following element(s):

- 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. The name is local to the cluster, i.e. two different clusters can have two pseg with the same name.
- `base`: base address 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 (with *n* being the number of processor). The index corresponds to the processor id in the architecture.

The proc element contains the following element(s):

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

The irq (interrupt request) element

This element describes the routing of interruption signals on the hardware. It contains the following attributes:

- `type`: The type of interrupt request. It can currently 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 (triggers a context-switch). The channel attribute must be equal to the processor id.
    - `ISR_TTY`: handles the interruptions emaning from the tty
    - `ISR_DMA`: handles the dma interruptions
    - `ISR_IOC`: handles the block device interruptions
    - `ISR_TIMER`: handle user timer interruptions

The periph (peripheral) element

This element contains the following attributes:

- `type`: type of the peripheral. It can be one of :
    - `IOC` : block device peripheral
    - `TTY`: tty device peripheral
    - `TIM`: timer device peripheral

    - ♦ DMA: dma device peripheral
    - ♦ FBF: frame buffer peripheral
    - ♦ NIC: nic buffer peripheral
    - ♦ IOB: io bridge peripheral
- psegname: name of the physical segment of the peripheral. This name must be one of the pseg elements name, and whose type is PERI. This is used to make the association between the physical segment declared in the architecture and the peripheral
- channels: the number of channels in the peripheral (one channel usually corresponds to one processor)

The globalset (global set) element

This element contains no attribute, and contains the following element(s):

- 0 to *n* vseg element(s)

The vseg (virtual segment) element

This element contains the following attributes:

- name: name of the virtual segment. This name is used only for clarity and has no functional use.
- vbase: base address of the virtual segment in the final binary. It must be page size (0x1000 or 4Ko) 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 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. This field is not necessarily required (TODO : à vérifier si c'est nécessaire et dans ce cas pourquoi)

The vseg element contains the following element(s):

- 1 to *n* vobj element(s). Note: One vseg can contain at most 1 vobj whose type is ELF (for practical purpose, a vobj of type ELF will always be the single vobj of its vseg). This is because the virtual address of the vobj is the one of the vseg + the size (and alignment) of previous vobj; besides the virtual base address of the ELF vobj must be the one of the section as defined in the .ld file. Thus, if two vobj of type ELF were in the same vseg, only one could have the vbase address of the vseg.

The vobj (virtual object) element

This element contains the following attributes:

- name: the name of the virtual object. This is used to obtain the virtual base address of the vobj: this name is a parameter of the vobj_get_vbase syscall along with the vspace name. Two vobj elements can't have the same name inside a vspace, even if they are in different vseg elements.
- type: type of the virtual object. It can be one of the following:

- ELF: describes an elf section. The system has to load the elf described by the binpath in this segment. This type has two constraint :
  - The vobj must be the only one vobj in its virtual segment (vseg)
  - The vbase of its vseg must be equal to the virtual address of the section defined in the ldscript. This is because of hard-coded jumps.
- PTAB: memory space reservation for the page table. Each vspace must have one vobj of this type. The system will have to initialize the page tables with the information contained in this file.
- PERI: type of vseg used for peripheral mapping. No particular action is performed by the operating system on this type of segment.
- MWMR: describes an mwmr channel. This vobj will be appropriately initialized by the operating system.
- LOCK: describes a lock. This vobj will be appropriately initialized by the operating system.
- BARRIER: describes a barrier. This vobj will be appropriately initialized by the operating system.
- BUFFER: describes a buffer. No particular action is performed by the operating system on this type of segment.
- BLOB: Binary Large OBject. The system has to load the file pointed to by the binpath in this segment.

- length: size of the virtual object in bytes. The sum of the length of all vobj defines 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 next aligned address.
- binpath: path to the file to load in this segment, if any. It is used only for ELF and BLOB segments, and must empty for other segment types. This path can be relative or absolute.

The vspaceset (virtual space set) element

The vspaceset element has no attribute and contains the following element(s):

- 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 virtual space (a vspace corresponds to one application). The operating system will load at the top of this section a table containing for each task the first function to execute. This table is called the task function entry table.

The vspace element contains the following element(s):

- 0 to *n* vseg element(s)
- 0 to *n* task element(s) (Remarque : que se passe-t-il si 0 task ? accès impossible ? accès possible de toutes les tâches ? autre ?)

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 for the task, 0 otherwise.
- `startid`: indicates the index of the function to execute in the task function entry table of the corresponding vspace.
- `usefbdma`: (optional) if set (with value 1), indicates that the task uses a frame buffer with dma accesses (`fb_write` function).