ADL Database Reference

This document describes the ADL's database layout. This is generated from an ADL input file via the **make-db** command. Either a Perl or an XML database may be generated.

Author: Brian Kahne
Contact: bkahne@freescale.com

Table of Contents

1 General Overview

An ADL database consists of a Perl or XML file describing all of the information contained within an ADL description. To generate such a database, use the **make-db** command:

```
make-db --type=<perl|xml> <source>
```

The output is generated to standard out. If Perl is selected, the resulting database can be directly evaluated by the Perl interpreter. The XML version may be parsed by any standard XML parser.

The Perl and XML formats contain the same information and posses the same structure. They differ only in syntax.

1.1 Perl Format

The Perl database consists of a series of nested Perl data structures. In most cases, these consist of hash tables. For named items, such as registers and instructions, a hash table will consist of keys identifying the named item, e.g. register names. In the Database Layout section, these types are identified as **"Hash (name => contents)"**. The value for each key will then generally consist of another hash, with keys representing the parameters of the item. For example:

```
regs = {
  PC => {
    width = 32,
```

```
    ...
  },
  ...
}
```

Simple data types are represented using native Perl types, where possible, e.g. an integer will parse as an integer. In situations where large numbers are possible, such as for reset values and masks, the data will be stored as a string. Code blocks, such as for specifying instruction behavior, are also stored as strings, with embedded newlines.

1.2 XML Format

As with the Perl database, the XML database is hierarchically structured. Since XML does not have a concept of hashes, the equivalent within XML is to use a tag with a name attribute, e.g.:

```
<register name="PC"> ... data ... </register>
```

Parameters use a tag surrounding their data:

```
<width><int>32</int></width>
```

Data types are identified by tags:

- `<int>`: Integer. Data is contained between tags.
- `<str>`: String. Multi-line strings are stored using **CDATA** blocks.
- `<true/>`: Boolean true. Singular tag with no data.
- `<false/>`: Boolean true. Singular tag with no data.

1.3 Register Usage Information

Several items within the database, such as instructions and helpers, contain usage information about register usage: Which registers or register-files are read, which are written, etc. This information is presented as a list of strings, where each string encodes usage information about a single resource (register or register-file).

The format consists of the following: **resource-name index mask qualifiers**

- **resource-name**: The name of the register or register-file being accessed.
- **index**: Only present if a register-file. Format: `( <start-expr> [ , <end-expr> ] )`. Each expression may be either an instruction field, constant, expression containing either constants or instruction fields, or for a helper function, `[parm #n]`, where **n** is the index of the parameter in the function's parameter list, used to index into the register file.
- **mask**: If a partial access which may be statically determined, has the format `[mask <hex-value>]`. This is a hex mask specifying the portion of the

register accessed. If the database was generated with `--used-reg-fields`, then the mask will be omitted and named instruction fields will be specified (where possible), using the format `.field`.

- **qualifiers**:
  - `/p`: Partial access, but the portion accessed is specified dynamically and cannot thus be statically determined.
  - `/d`: Delayed access. The write's effects will be delayed by one or more instructions.
  - `/D`: Direct write. The write will not be renamed or tracked if used in a model with micro-architectural knowledge.
  - `?`: Conditional access. The read or write of this resource lies within a conditional block whose condition could not be statically determined to be true or false.

2 Database Layout

This section describes the structure of the database, including what each element represents.

cores:

Hash (name => core contents). Describes a core in the system. A core consists of various processor resources, such as registers, plus various instructions.

doc: String (optional). The documentation string for the core.

type: String (optional). An optional type attribute for the core.

bit_endianness: String (optional). "little" or "big" for little-endian or big-endian.

blocks: Array(string) (optional). Lists all functional-unit blocks in the core.

type_declarations: Array(string) (optional). Lists any C declarations made in the core scope.

parallel_execution: Integer (optional). If non-zero, lists the number of instructions executed in parallel in the core. This is only applicable for VLIW architectures; superscalar pipelines, since they act as sequential machines, would omit this.

RaMask: Hash. Describes how a real address is generated for a memory operation.

initial: String. Hex value of the initial mask.

constant: Boolean (optional). If true, mask is always constant.

watch: String (optional). Code describing how a non-constant real address mask is modified.

RaMask: Hash. Describes how an effective address is generated for a memory operation.

initial: String. Hex value of the initial mask.

constant: Boolean (optional). If true, mask is always constant.

watch: String (optional). Code describing how a non-constant effective address mask is modified.

regs: Hash (name => contents). Describes a register in the core.

doc: String (optional). The documentation string for the register.

width: Integer. The width of the register, in bits.

offset: Integer (optional). Starting bit index.

serial: Boolean (optional). If true, implies that, for a parallel architecture, this register is written serially and not in parallel, i.e. writes take effect immediately.

pseudo: Boolean (optional). If true, the register does not have its own data contents. Writes and reads are handled via hooks which modify other resources.

enumerated: Hash (value(string) => string). The register has values which have special meanings. This hash maps integer values to description strings.

fields: Hash (name => contents). Lists any fields this register may have.

doc: String (optional). The documentation string for the register field.

indexed: Hash. The register field is indexed, which means that it is replicated across the entire register.

width: Integer. The width, in bits, of the field.

count: Integer. The number of instances of the field.

bits: Array(integer). The bit range of the field. Two integers. Order is dependent upon endianness of the core.

readonly: Boolean (optional). True if the field is read-only.

reserved: Boolean (optional). True if the field is reserved, which means it consist of unimplemented bits which will always read 0.

attributes: Hash (name (string) => value (string)). Register attributes.

reserved_mask: String (optional). The numerical value of the register's reserve mask (unimplemented bits which read 0).

readonly_mask: String (optional). The numerical value of the register's read-only mask.

attributes: Hash (name (string) => value (string)). Register attributes.

reset: String (optional). The reset value or text of the function called to reset the register.

shared: Integer (optional). 1 or 0. Non-zero implies that the register is shared by other cores in the system.

read: Hash. Behavior on a register read or alias information. If a function, then the content is a string describing the read behavior.

alias: String (optional). If an alias, then the name of the target of the alias for a read operation.

alias_index: Integer (optional). If the alias target is a register file, the target element of the register file.

write: Hash. Behavior on a register write or alias information. If a function, then the content is a string describing the write behavior.

ignored: Boolean (optional). If true, the write is ignored.

alias: String (optional). If an alias, then the name of the target of the alias for a write operation.

alias_index: Integer (optional). If the alias target is a register file, the target element of the register file.

regfiles: Hash (name => contents). Describes a register in the core.

doc: String (optional). The documentation string for the register.

width: Integer. The width of the register, in bits.

offset: Integer (optional). Starting bit index.

serial: Boolean (optional). If true, implies that, for a parallel architecture, this register is written serially and not in parallel, i.e. writes take effect immediately.

pseudo: Boolean (optional). If true, the register does not have its own data contents. Writes and reads are handled via hooks which modify other resources.

enumerated: Hash (value(string) => string). The register has values which have special meanings. This hash maps integer values to description strings.

fields: Hash (name => contents). Lists any fields this register may have.

doc: String (optional). The documentation string for the register field.

indexed: Hash. The register field is indexed, which means that it is replicated across the entire register.

width: Integer. The width, in bits, of the field.

count: Integer. The number of instances of the field.

bits: Pair(integer). The bit range of the field. Two integers. Order is dependent upon endianness of the core.

readonly: Boolean (optional). True if the field is read-only.

reserved: Boolean (optional). True if the field is reserved, which means it consist of unimplemented bits which will always read 0.

attributes: Hash (name (string) => value (string)). Register attributes.

reserved_mask: String (optional). The numerical value of the register's reserve mask (unimplemented bits which read 0).

readonly_mask: String (optional). The numerical value of the register's read-only mask.

attributes: Hash (name (string) => value (string)). Register attributes.

reset: String (optional). The reset value or text of the function called to reset the register.

size: Integer. The number of elements in the register file.

prefix: String (optional). Prefix string used to identify an operand for this register file in an assembly string.

shared: Integer (optional). 1 or 0. Non-zero implies that the register is shared by other cores in the system.

entries: Hash (integer => contents). For a sparse register file, lists register entries, keyed by register-file index.

syntax: String (optional). A name identifying the register or register-file plus the index, if applicable.

reg: String (optional). If the element maps to a register, then the name of the target register.

read: String (optional). Code executed on a read.

write: Hash. Behavior of a write. If a hook function exists, then consists simply of the text of the write behavior.

ignored: Boolean (optional). True if the write is ignored.

read: Hash. Behavior on a register read or alias information. If a function, then the content is a string describing the read behavior.

alias: Hash. Information about the register alias for reads.

reg: Name of the register-file target.

map: Hash (integer => integer) (optional). If the alias consists of a per-element mapping, this maps alias indexes to target indexes.

write: Hash ("read" => contents | string). Behavior on a register write or alias information. If a function, then the content is a string describing the write behavior.

ignored: Boolean (optional). If true, the write is ignored.

alias: Hash. Information about the register alias for writes.

reg: Name of the register-file target.

map: Hash (integer => integer) (optional). If the alias consists of a per-element mapping, this maps alias indexes to target indexes.

relocations: Hash(name => contents) (optional). Lists linker relocations for the core.

doc: String (optional). The documentation string for the relocation.

value: Integer. The relocation default vale.

width: Integer. The relocation width, in bits.

instrfields: Hash(name => contents) (optional). Lists all instruction fields in the design.

doc: String (optional). The documentation string for the instruction field.

syntax: String (optional). Syntax string for the field, for when a complex field exists.

attributes: Hash (name (string) => value (string)). Instruction attributes.

alias: String (optional). Name of the target instruction field, if an alias.

bits: Pair(integer) (optional). Bit range for the instruction field.

pseudo: Boolean (optional). True if this is a pseudo field (not mapped to fixed bits).

prefix: Boolean (optional). If true, this instruction field may be used in prefix instructions.

width: Integer (optional). Width of the instruction field. Relevant if this is a pseudo field, not mapped to fixed bits.

size: Integer (optional). The full, computed width of a field, including any extra prefix bits.

shift: Integer (optional). Number of bits to shift an operand value to the left when used by an instruction.

indexed: Integer (optional). For prefix fields, describes the number of bits to be allocated to each instruction in a packet from this field.

mask: String. The field mask, in hex. This has a bit set for each bit in the field.

blocks: Array(string) (optional). Lists all function-unit blocks that this instruction field is associated with.

table: Hash (index => contents) (optional). If instruction field values are computed from a table, then this lists the table contents, mapped by index entry. Contents are either a simple integer value (for a one-dimensional table), an array of integers for a two-dimensional table, or "reserved" to represent an illegal value.

enumerated: Hash (value(string) => string) (optional). The register has values which have special meanings. This hash maps integer values to description strings.

addr: String(abs|pc) (optional). If the instruction field contents are used as an address, this specifies program-counter relative or absolute addressing.

display: String(hex|dec|name) (optional). Specifies how the field should be display during disassemble. If **name**, then the prefix from the associated register file, or entry name (if a sparse register file) is used.

type: String(regfile|memory|immed|ident) (optional). Specifies the type of the instruction field, i.e. the type of resource associated with the field.

overlay: Boolean (optional). If true, this field may overlap with other fields in the instruction.

signed: Boolean (optional). If true, the value of the field should be considered a signed value.

inverted: Boolean (optional). If true, the value is inverted before being used by the instruction.

assembler: Boolean (optional). If true, the field is not considered a direct operand by the assembler, but is set later by various hooks.

action: String (optional). Any behavior associated with this instruction field.

value: Integer (optional). The default value for the field.

implements: String (optional). The name of the field which this field implements. This is relevant for pseudo fields; this is the concrete field implementing a particular pseudo field.

reloc: String (optional). The name of the relocation associated with this field.

valid_ranges: List(pair(integer)). Lists valid ranges for this field.

valid_masks: List(pair(integer)). Lists valid (mask,value) pairs for this field.

ref: String (optional). Resources referenced by this fielde: register file or memory.

fields: Array (contents). Each element describes a nested field used to construct the outer field value.

String: A field name.

Pair(integer): Range of bits from the outer field.

Hash (field => bit value used).

instrfields: Lists any nested instruction fields.

size: Integer (optional). The full, computed width of a field, including any extra prefix bits.

fields: Refer to the fields description.

bits: Pair(integer) (optional). Bit range for the instruction field.

table: Refer to the description for table.

instrs: Hash(name => contents). Lists all instructions in the core.

width: Integer (optional). The full width of the instruction, taking into account any prefix bits.

fetch_width: Integer (optional). The fetched width of the instruction. This excludes bits contributed by a prefix instruction.

doc: String (optional). The documentation string for the instruction.

syntax: String (optional). Syntax string for the instruction. Field codes, e.g. `%f` are replaced with the respective operand name.

block: String (optional). Functional unit with which the instruction is associated.

attributes: Hash (name (string) => value (string)). Instruction attributes.

fields: Hash (name (string) => value (string|integer)). Lists all fields within the instruction. Keys are the field names. Values are integers for opcode fields, `reserved` to indicate a reserve field, or a string to store an expression, such as for an assembler short-hand.

pseudo: Boolean (optional). If true, the instruction is a pseudo instruction. Pseudo instructions act as templates for other instructions, such as to describe which portion is extracted from a prefix instruction. They do not contain any semantics themselves and are not directly encoded in any instruction tables.

aliases: If this instruction is an alias, this contains information about the alias(es). An alias may have multiple targets if it is just an assembler shorthand (thus acting as a form of macro). For real aliases (those understood by models), only one target is allowed. In the hash, each key is the name of the target instruction and values are a hash consisting of the following keys:

sources: List ("source" => hash) (optional). Lists any instruction fields which are used to identify source registers.

field: String. The name of the instruction field in the target instruction.

value: String. The name of the field or expression in the alias.

destinations: List ("destination" => hash) (optional). Lists any instruction fields which are used to identify destination registers.

field: String. The name of the instruction field in the target instruction.

value: String. The name of the field or expression in the alias.

miscs: List ("misc" => hash) (optional). Any remaining fields not directly used as a source or target.

field: String. The name of the instruction field in the target instruction.

value: String. The name of the field or expression in the alias.

parent_action: String (optional). This contains the alias target's action code with all operands replaced by the alias's operands.

extracted-doc-op: String (optional). This contains the target's operation, if it was extracted automatically from an instruction's action, via the use of the `doc_op` label.

action: String (optional). The instruction's semantics.

assembler: Boolean (optional). If true, the instruction is used only by the assembler, for modifying the behavior of other instructions' encodings. It is not encoded within any instruction tables used by an actual model.

parent: String (optional). If a nested instruction, this specifies the name of the parent instruction.

prefix: Boolean (optional). If true, this is a prefix instruction. It contributes encoding bits towards the instructions in a VLIW packet.

disassemble: String (true|false|prefix) (optional). Indicate how to handle an instruction field by default, unless otherwise specified by an instruction's syntax string.

true:
Disassemble normally.

false:
Do not disassemble. This is overridden by an instruction's syntax, if specified.

prefix:
Disassemble in front of an instruction's mnemonic. This occurs even if an instruction's syntax string also specifies it. This is useful for implementing instruction prefix syntax without explicit `%p` fields in all syntax strings. To support assembly, the user would normally mark this field as an `assembler` field and then set it via assembler instructions and the `pre_asm` hook.

extracted-doc-op: String (optional). This contains the instruction's operation, if it was extracted automatically from an instruction's action, via the use of the `doc_op` label.

doc-op: String (optional). This contains the instruction's operation if it was specified explicitly.

type: String (optional). If the instruction inherits its structure from another instruction, this lists the master instruction.

inputs: List (string) (optional). Lists source registers or register files. Refer to Register Usage Information for more information on the format.

outputs: List (string) (optional). Lists target registers or register files. Refer to Register Usage Information for more information on the format.

input_mems: List (string) (optional). List any memories read by the instruction.

output_mems: List (string) (optional). List any memories written by the instruction.

input_caches: List (string) (optional). List any caches read by the instruction.

output_caches: List (string) (optional). List any caches written by the instruction.

helpers: List (string) (optional). List any core-level helper functions used by the instruction.

raises_exception: Boolean (optional). If true, the instruction may raise an explicit exception.

subinstrs: Hash (name => contents) (optional). List all sub-instructions in the design.

doc: String (optional). The documentation string for the instruction.

block: String (optional). Functional unit with which the instruction is associated.

attributes: Hash (name (string) => value (string)). Instruction attributes.

syntax: String (optional). Syntax string for the instruction. Field codes, e.g. `%f` are replaced with the respective operand name.

fields: Hash (name (string) => value (string|integer)). Lists all fields within the instruction. Keys are the field names. Values are integers for opcode fields, `reserved` to indicate a reserve field, or a string to store an expression, such as for an assembler short-hand.

inputs: List (string) (optional). Lists source registers or register files. Refer to Register Usage Information for more information on the format.

outputs: List (string) (optional). Lists target registers or register files. Refer to Register Usage Information for more information on the format.

input_mems: List (string) (optional). List any memories read by the instruction.

output_mems: List (string) (optional). List any memories written by the instruction.

input_caches: List (string) (optional). List any caches read by the instruction.

output_caches: List (string) (optional). List any caches written by the instruction.

helpers: List (string) (optional). List any core-level helper functions used by the instruction.

raises_exception: Boolean (optional). If true, the instruction may raise an explicit exception.

exceptions: Hash (name => contents) (optional). List all exceptions in the core.

doc: String (optional). The documentation string for the exception.

attributes: Hash (name (string) => value (string)). Exception attributes.

fields: Hash (name => contents) (optional). List any fields for this exception.

doc: String (optional). The documentation string for the exception field.

attributes: Hash (name (string) => value (string)). Exception-field attributes.

bits: Integer. Bit width of the field.

contexts: Hash (name => contents) (optional). List any contexts in the core.

doc: String (optional). The documentation string for the context.

attributes: Hash (name (string) => value (string)). Context attributes.

thread: Boolean (optional). True if the context is considered to be context. ADL considers a context to be a thread if it contains the program counter register.

active: String (optional). The hook function for determining whether a context is active.

num_contexts: Integer. The number of instances of this context.

regs: List(string). All registers associated with this context.

regfiles: List(string). All register files associated with this context.

caches: Hash (name => contents). Describes all caches in the core.

doc: String (optional). The documentation string for the context.

attributes: Hash (name (string) => value (string)). Context attributes.

level: Integer. Cache level. A value of one implies the first cache in the system (closest to the fetch/data unit).

type: String (instr|data|unified). Cache type.

size: Integer. Total cache size, in bytes.

line_size: Integer. Line size of cache, in bytes.

fully_assoc: Boolean (optional). True implies that this is a fully associative cache.

set_assoc: Integer (optional). If present, specifies cache set associativity.

setfields: Hash (name => contents). List all fields which are defined for each set of the cache.

doc: String (optional). The documentation string for the exception field.

attributes: Hash (name (string) => value (string)). Exception-field attributes.

bits: Integer. Bit width of the field.

wayfields: Hash (name => contents). List all fields which are defined for each way of the cache.

doc: String (optional). The documentation string for the exception field.

attributes: Hash (name (string) => value (string)). Exception-field attributes.

bits: Integer. Bit width of the field.

enable-predicate: String (optional). The cache's enable predicate function.

hit: String (optional). Code to be executed on a cache hit.

miss: String (optional). Code to be executed on a cache miss.

invalidate-line: String (optional). Code to be executed when a line is invalidated.

read-line: String (optional). Code to be executed when a line is read.

replace: String (optional). Custom replacement code for the cache.

line-access: String (optional). Code to be executed when a line is hit or loaded into the cache.

writethrough-predicate: String (optional). The predicate to specify whether the cache is write-through or write-back.

number-of-sets: String (optional). Function to specify number of cache sets, if dynamic.

number-of-ways: String (optional). Function to specify number of cache ways per set, if dynamic.

eventbuses: Hash (name => contents). List all event buses (inter-core communication channels) defined for the core.

doc: String (optional). The documentation string for the event bus.

attributes: Hash (name (string) => value (string)). Event bus attributes.

fields: Hash (name => contents) (optional). List any fields for this event bus.

doc: String (optional). The documentation string for the event bus field.

attributes: Hash (name (string) => value (string)). Event bus field attributes.

bits: Integer. Bit width of the field.

action: String (optional). Action to be taken when an event bus receives data.

memories: Hash (name => contents). List all local memories for the core.

instr_mem: Boolean. If true, this memory is used only for instruction fetches.

prefetch_mem: Boolean (optional). If true, this memory serves as a prefetch
buffer for instructions.

size: Integer. Size of memory, in bytes.

addr_unit: Integer. Unit of addressing, in bytes. One means that the memory is byte addressable.

parent: Pair(string,integer). (optional). Name of parent memory, if this memory is memory-mapped within another memory, and offset within parent memory.

read: String (optional). Code to be executed on a memory read, such as to modify the returned memory contents.

write: String (optional). Code to be executed on a memory write.

instr_read: String (optional). Code to be executed on an instruction fetch.

mmu: Hash. Describes the MMU in the core, if one is present.

lookups: Hash (name => contents). List all TLBs in the MMU.

priority: Integer. List the priority of this TLB (order in which TLBs are searched for a translation).

page-size: Hash. Describes the page size for this TLB.

const: Boolean (optional). True if page sizes are constant.

size: Integer (optional). Size of the page, for constant-size TLBs.

bitsize: Boolean (optional). True means that the page size is interpreted as

`(2^(sizescale*size+sizeoffset) << sizeshift)` bytes.

leftmask: Boolean (optional). True means that the page size is interpreted as a mask with 1's specifying the bits of the address which are ignored, shifted by sizeshift.

scale: Integer (optional). The page-size scaling factor, in bits.

field: String (optional). MMU field used as the page-size source.

offset: Integer (optional). Page-size offset value.

shift: Integer (optional). Page-size shit value, in bits.

entries: Integer. Number of entries in the TLB.

fully_assoc: Boolean (optional). True implies that this is a fully associative cache.

set_assoc: Integer (optional). If present, specifies cache set associativity.

setfields: Hash (name => contents). List all fields which are defined for each set of the TLB.

doc: String (optional). The documentation string for the exception field.

attributes: Hash (name (string) => value (string)). Exception-field attributes.

bits: Integer. Bit width of the field.

wayfields: Hash (name => contents). List all fields which are defined for each way of the TLB.

doc: String (optional). The documentation string for the exception field.

attributes: Hash (name (string) => value (string)). Exception-field attributes.

bits: Integer. Bit width of the field.

tests: List(string) (optional). Lists test expressions used for determining a hit in the TLB, if only one set of tests exists.

testsets: Hash (name => contents). Lists test sets, if more than one set of tests exists.

enable: String (optional). Enable predicate for the test set.

tests: List(string). List test expressions for the TLB test set.

exec-perm: String (optional). Code to be executed to check execution permission.

load-perm: String (optional). Code to be executed to check data load permission.

store-perm: String (optional). Code to be executed to check data store permission.

valid: String (optional). Predicate to determine if a TLB entry is valid.

hit: String (optional). Code to be executed on a TLB hit.

miss: String (optional). Code to be executed on a TLB miss.

reset: String (optional). Code to be executed on a reset, to configure a TLB.

instr-enable: String. Code to determine if the MMU is enabled for an instruction fetch.

data-enable: String. Code to determine if the MMMU is enabled for a data access.

instr-miss: String. Code to be executed on an instruction miss in the MMU.

data-miss: String. Code to be executed on a data miss in the MMU.

multi-hit: String. Code to be executed if a hit occurs in more than one TLB.

misaligned-write: String. Code to be executed if a data-write access occurs which is misaligned.

misaligned-read: String. Code to be executed if a data-read access occurs which is misaligned.

post-write: String. Code to be executed after a data-write occurs.

post-read: String. Code to be executed after a data-read occurs.

core-level-hooks: Lists various hook functions associated with the core.

decode-miss: String (optional). Code to be executed on a decode miss.

pre-cycle: String (optional). Code to be executed once per cycle, at the beginning of the cycle.

post-cycle: String (optional). Code to be executed once per cycle, at the end of the cycle.

pre-fetch: String (optional). Code to be executed immediately before an instruction fetch.

post-fetch: String (optional). Code to be executed immediately after an instruction fetch.

post-exec: String (optional). Code to be executed immediately after an instruction has been executed.

post-asm: String (optional). Code to be executed by the assembler immediately after an instruction has been assembled from its operands.

post-packet-asm: String (optional). Code to be executed by the assembler after a packet of instructions has been assembled.

post-packet: String (optional). Code to be executed after a packet of instructions has been executed.

active-watch: String (optional). Predicate to determine if the core is currently active or halted.

instr-table-watch: String (optional). Code which determines the current instruction table currently in effect.

groups: Hash (name => contents). Lists all groups defined in the core.

type: String (instr|reg|regfile|parm|instrfield|none). Group type.

items: List(string). List of all items in the group.

parms: Hash (name => contents). List all architectural parameters in the core.

doc: String (optional). The documentation string for the parameter.

value: String. The default value for the parameter.

options: List(string). List of valid values for the parameter.

watch: String (optional). Code to be executed to determine the current value for the parameter.

constant: Boolean (optional). If true, the parameter is constant and may not be changed by action code.

asm_config: Hash (optional). List information about the assembler configuration.

comments: List(string) (optional). List prefixes used to denote the start of a comment.

line_comments: List(string) (optional). List characters used to denote the start of a single-line comment.

line_seps: List(string) (optional). List additional instruction separator characters besides newline.

group_seps: List(pair(string) | string) (optional). Lists group separators. If the item is a pair, the first element starts a group, the second element indicates the end of a group.

instrtables: List(string) (optional). List tables associated with this assembler.

parms: Hash(name => contents). List assembler parameters.

value: Integer (optional). Default value for the parameter.

type: String. The parameter type.

helpers: Hash (name => contents) (optional). List all helper methods in the core.

action: String. The code for the helper function.

inputs: List (string) (optional). Lists source registers or register files. Refer to Register Usage Information for more information on the format.

outputs: List (string) (optional). Lists target registers or register files. Refer to Register Usage Information for more information on the format.

input_mems: List (string) (optional). List any memories read by the helper.

output_mems: List (string) (optional). List any memories written by the helper.

input_caches: List (string) (optional). List any caches read by the helper.

output_caches: List (string) (optional). List any caches written by the helper.

helpers: List (string) (optional). List any core-level helper functions used by the helper.

raises_exception: Boolean (optional). If true, the helper may raise an explicit exception.

systems:

Hash (name => system contents). Lists all systems defined in the ADL description. A system contains other cores or systems, plus resources shared between these child systems or cores. In Perl, the contents are a hash. The keys are the system names, values are the core contents.

regs: Registers in the system. Refer to regs definition.

regfiles: Register files in the system. Refer to the regfiles definition.

caches: Caches in the system. Refer to the caches definition.

mmulookups: MMU TLBs in the system. Refer to the mmulookup definition.

memories: Memories in the system. Refer to the memories definition.

eventbuses: Event buses in the system. Refer to the eventbuses definition.

core: Cores in the system. Refer to the core definition.

Generated on: 2018/02/28 15:17:11 MST.