VCI protocol adaptations for SoCLib

VCI has some ambiguous features, and other are unusable. Here we'll try to clarify the VCI extensions made in the SoCLib context.

Endianness

VCI is quite unclear about endianness interpretation. SoCLib defines some rules on top of VCI:

- VCI addresses words. LSBs of addresses are non-significant. i.e. on a 32-data-bits VCI network, ADDR[1:0] are useless, and can be kept 0.
- VCI subword addressing is done through BE field
    - BE[0] is associated to DATA[7:0]
    - BE[1] is associated to DATA[15:8]
    - BE[2] is associated to DATA[23:16]
    - BE[3] is associated to DATA[31:24]
- DATA Words are little endian so considering an (aligned) address @ of a word in DATA,
    - byte at address @ is in DATA[7:0]
    - byte at address @+1 is in DATA[15:8]
    - byte at address @+2 is in DATA[23:16]
    - byte at address @+3 is in DATA[31:24]

Peripherals considerations

It's up to the peripherals to define their endianness, i.e. the association between lower addresses and lower-significant-bits.

As our convention aligns lower addressable byte to LSBs of DATA, thus is Little-Endian, it may be easier to have little-endian-only peripherals and use them seamlessly from any CPU with encapsulated accesses.

Memory considerations

Memory is endianness-agnostic. It serves reading and writing from/to memory words (in the VCI way defined above)

Cache considerations

VciXCache does not care about endianness of attached processor BUT asserts the VCI definitions above.

When accessing memory with a big-endian processor, it's up to the processor to swap the word from LE to BE.

Atomic Operations

VCI defines a `LOCKED_READ` operation. Unfortunately, it can't be properly used with a NoC-based interconnect as putting the reservation on the "read" operation:

- Can't guarantee the absence of livelocks
- Necessitates the addition of a timeout mechanism to workaround software doing locked read without associated write.

Therefore, SoCLib uses the "NOOP" VCI command code (redundant with CMDVAL = 0) to introduce the "Store conditional" mechanism.

A "STORE_COND" response uses the RDATA field for STORE completion:

`RDATA == 0`
: STORE_COND was atomic

`RDATA == 1`
: STORE_COND was not atomic, a complete `LOCKED_READ` / `STORE_COND` cycle must be done again.