VciXcacheWrapper / VciVcacheWrapper

1) Functional Description

These 2 hardware components are generic cache controllers, fully compliant with the VCI advanced protocol. They can be used to interface a - single instruction issue - 32 bits RISC processor (such as Mips32, Sparc V8, Xilinx microBlaze, Altera Nios, or PPC 405) to a VCI based multi-processor system. They act directly as a wrapper for any ISS (Instruction Set Simulator) respecting the generic cache/processor interface defined in source:trunk/soclib/soclib/iss/iss2/include/iss2.h?.

Each cache controller implements two separated instruction and data caches, sharing the same VCI interface.

The VciVcacheWrapper component (in short Vcache) has the same functionnalities as the VciXcacheWapper component (in short Xcache), but it implements the SoCLib generic MMU.

General features

The general characteristics are the following

- The VCI DATA field must have 4 bytes,
- The VCI ADDRESS field must have 32 bits (when there is no MMU),
- The VCI PLEN field must be large enough to represent 4*(N+2), where N is the number of words in a cache line.
- The VCI ERROR field has 1 bit.
- The number of lines must be a power of 2, and cannot be larger than 1024.
- The number of words must be a power of 2, and cannot be larger than 32.
- The number of ways must be a power of 2, and cannot be larger than 16.

According to the VCI advanced specification, these components use one word VCI command packets for Read bursts, and accept one word VCI response packets for Write bursts.

The Xcache implements a very simple write buffer, and do NOT start a new VCI transaction until the previous transaction is completed. Therefore, it doesn't use the VCI PKTID and TRDID fields.

The Vcache controller implements a more advanced write buffer, supporting several simultaneous VCI transactions. It uses the VCI TRDID field to support this functinnality.

Instruction Cache

- It is read-only.
- It uses the Mapping Table to support uncached segments.
- In case of read MISS, or read uncached, the processor is stalled until the missing instruction is available.
- The two VCI transactions generated by the Instruction cache are
    - ♦ read burst corresponding to a missing cache line,
    - ♦ one word read, when the corresponding address is uncached.

Data Cache

- The write policy is WRITE-THROUGH (the data is immediately written in memory, and the cache is

updated only in case of HIT).
- The Data cache contains a write buffer, and builds a burst when there are successive write requests in the same cache line.
- It uses the Mapping Table to support uncached segments.
- The Data Cache supports the following requests : Read, Write, Linked load, and Store Conditional
- The Data cache accepts a line invalidate command.
- Three types of VCI transactions can be generated by the data cache:
    - ♦ read burst, corresponding to a missing cache line,
    - ♦ one word transaction, corresponding to an uncached read, a linked load, or a store conditional.
    - ♦ write burst of variable length (no larger than a cache line)
- The processor is stalled in case of cached read MISS, in case of uncached read, or in case of write, if the write buffer is full.

Generic MMU

The Vcache and CC_Vcache components implement a generic MMU, that can be used by all the single instruction issue 32 bits processors available in the SoCLib platform. This MMU performs both the logical address to physical address translation, and access rights checking.

- The logical address is 32 bits.
- The physical address (VCI address) is up to 40 bits.
- It uses a two level, hierarchical, page table.
- Two page sizes are supported : 4 Kbytes, or 2 Mbytes.
- Separated TLBs are used for instruction and data addresses.
- The TLB misses are handled by hardware (hardwired table-walk).

A more detailed specification of the generic MMU can be find on the Web site of the TSAR project : ?https://www-asim.lip6.fr/trac/tsar/wiki/VirtualMemory

2) CABA Implementation

Xcache

- Usage : source:trunk/soclib/soclib/module/internal_component/vci_xcache_wrapper/caba/metadata/vci_xcache_wrapper.sd?
- interface : source:trunk/soclib/soclib/module/internal_component/vci_xcache_wrapper/caba/source/include/vci_xcache_wrapper
- implementation : source:trunk/soclib/soclib/module/internal_component/vci_xcache_wrapper/caba/source/src/vci_xcache_wrapper.cpp

Vcache

- Usage : source:trunk/soclib/soclib/module/internal_component/vci_vcache_wrapper/caba/metadata/vci_vcache_wrapper.sd?
- interface : source:trunk/soclib/soclib/module/internal_component/vci_vcache_wrapper/caba/source/include/vci_vcache_wrapper
- implementation : source:trunk/soclib/soclib/module/internal_component/vci_vcache_wrapper/caba/source/src/vci_vcache_wrapper.cpp

CABA template parameters

Both the Xcache and Vcache components have two template parameters, defining respectively the width of the

various VCI signals, and the instanciated ISS.

```
template<typename vci_param, typename iss_t>
```

CABA constructor parameters

Xcache

```
VciXcacheWrapper(
    sc_module_name insname,
    int proc_id,
    const soclib::common::MappingTable &mt,
    const soclib::common::IntTab &index,
    size_t icache_ways,  // number of ways per associative set (instruction cache)
    size_t icache_sets,  // number of  associative sets (instruction cache)
    size_t icache_words,  // number of words per line (instruction cache)
    size_t dcache_ways,  // number of ways per associative set (data cache)
    size_t dcache_sets,  // number of  associative sets (data cache)
    size_t dcache_words);  // number of words per line (data cache)
```

Vcache

```
VciVcacheWrapper(
    sc_module_name insname,
    int proc_id,
    const soclib::common::MappingTable &mt,
    const soclib::common::IntTab &index,
    size_t itlb_ways,  // number of ways per associative set (instruction TLB)
    size_t itlb_sets,  // number of associative sets (instruction TLB)
    size_t dtlb_ways,  // number of ways per associative sets (data TLB)
    size_t dtlb_sets,  // number of associative sets (data TLB)
    size_t icache_ways,  // number of ways per associative set (instruction cache)
    size_t icache_sets,  // number of  associative sets (instruction cache)
    size_t icache_words,  // number of words per line (instruction cache)
    size_t dcache_ways,  // number of ways per associative set (data cache)
    size_t dcache_sets,  // number of  associative sets (data cache)
    size_t dcache_words);  // number of words per line (data cache)
```

CABA ports

Xcache & **Vcache**

- sc_in<bool> **p_resetn** : Global system reset
- sc_in<bool> **p_clk** : Global system clock
- soclib::caba::VciInitiator<vci_param> **p_vci** : The VCI port
- sc_in<bool> **p_irq[iss_t::n_irq] : IRQ ports, depending on the processor type**

3) TLM-DT Implementation

- interface : source:trunk/soclib/soclib/module/internal_component/vci_xcache_wrapper/tlmdt/source/include/vci_xcache_wrappe
- implementation : source:trunk/soclib/soclib/module/internal_component/vci_xcache_wrapper/tlmdt/source/src/vci_xcache_wrapper.cp

TLM-DT template parameters

All these components have two template parameters, defining respectively the width of the various VCI signals, and the instanciated ISS.

```
template<typename vci_param, typename iss_t>
```

TLM-DT constructor parameters

Xcache

```
VciXcacheWrapper(
    sc_module_name insname,
    int proc_id,
    const soclib::common::MappingTable &mt,
    const soclib::common::IntTab &index,
    size_t icache_ways,  // number of ways per associative set (instruction cache)
    size_t icache_sets,  // number of  associative sets (instruction cache)
    size_t icache_words,  // number of words per line (instruction cache)
    size_t dcache_ways,  // number of ways per associative set (data cache)
    size_t dcache_sets,  // number of  associative sets (data cache)
    size_t dcache_words,  // number of words per line (data cache)
    size_t time_quantum, // optional : maximal number of cycles between two consecutive mess
    size_t simulation_time); //optional : number of cycles of simulation (default = std::num
```

TLM-DT ports

Xcache

- **p_vci** : VCI initiator port
- **p_irq[iss_t::n_irq]** : IRQ ports, depending on the processor type