SocLib Components General Index

VciXcacheWrapper / VciVcacheWrapper / VciCcXcacheWrapper / VciCcVcacheWrapper

1) Functional Description

These 4 hardware components are generic cache controllers, fully compliant with the VCI advanced protocol. They can be used to interface any - 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:root/trunk/soclib/soclib/lib/iss2/include/iss2.h.

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

- The VciXcacheWrapper (in short Xcache) replace the previous VciXcache component. It has an higher simulation speed, and supports associativity (for both the instruction and data caches).
- The VciVcacheWrapper (in short Vcache) has the same functionnalities as the Xcache, and implements a generic paged MMU (see below).
- The VciCcacheWrapper (in short CC_Xcache) has the same functionnalities as the Xcache, and implement a directory-based cache coherence protocol (see below).
- The VciCcVcachewrapper supports both the generic MMU and the cache coherence.

	No Virtual memory	With Virtual Memory
No Cache Coherence	VciXcacheWrapper	VciVcacheWrapper
With Cache Coherence	VciCcXcacheWrapper	VciCcVcacheWrapper

General features

The general characteristics are the following

- The VCI DATA field must have 32 bits,
- The VCI ADDRESS field must have 32 bits (when there is no MMU),
- 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 associativity levels 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 MISS, and accept one word VCI response packets for Write bursts. In order to garanty the memory consistency, these cache controllers do NOT start a new VCI transaction until the previous transaction is completed. Therefore, they do NOT use the VCI PKTID and TRDID fields.

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 only VCI transaction generated by the Instruction cache is a read burst corresponding to a missing cache line.

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 of fixed length, corresponding to a cached read MISS,
    - ♦ 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 is 36 bits (or less).
- It is implemented as a two level, hierarchical, page table.
- Both first & second level page table contains 1024 entries.
- Two page sizes are supported : 4 Kbytes, or 4 Mbytes.
- Separated TLBs are used for instruction and data addresses.
- Separated TLBS are used for 4 Kbytes and 4 Mbytes pages (accessed in parallel).
- The TLB misses are handled by hardware (hardwired table-walk).
- An execution context is defined by the value stored in the PTPR (Page Table Pointer Register).
- Any context switch flush both the instruction & data TLBs.

The page page descriptor format is 32 bits:

ET	Entry Type	2 bits
C	Cachable	1 bit
W	Writable	1 bit
X	eXecutable	1 bit
U	User 'access in user mode allowed	1 bit
G	Global (not invalidate by TLB flush	1 bit
D	Dirty (page has been modified)	1 bit
PPN	Physical Page Number	24 bits

The generic MMU defines 10 registers, that can be accessed by the software through the generic cache/proccessor interface defined in source:root/trunk/soclib/soclib/lib/iss2/include/iss2.h

PTPR	set Page Table Pointer Register	Write
TLB_EN	activates Data & Instruction TLBs	Write
ICACHE_FLUSH	flush Instruction Cache	Write
DCACHE_FLUSH	flush Data Cache	Write

ITLB_INVAL	Instruction TLB line invalidate	Write
DTLB_INVAL	Data TLB line invalidate	Write
ICACHE_INVAL	Instruction Cache line invalidate	Write
DCACHE_INVAL	Data Cache line invalidate	Write
BAD_VADDR	Bad Virtual Address Register	Read
ERR_TYPE	Exception type Register	Read

Both the instruction & data caches are accessed with physical addresses.

In the Vcache and CC_Vcache components, the cachability (for both instruction & data accesses) can be defined by software - on a per-logical-page basis) through the cacheability attribut contained in each page descriptor. But the cachability can also be controlled (on a per-physical-segment basis) through the mapping table.

Cache Coherence

The CC_Xcache & CC_Vcache components implement a directory-based cache coherence protocol. The global memory directory itself should be implemented in a dedicated memory controller such as the VciMemCache component. The cache coherence protocol is strongly simplified by the WRITE-THROUGH policy and is implemented by three types of packets. The CC_Xcache (or CC_Vcache) component has one VCI target port, and can receive UPDATE or INVALIDATE packets, from the memory controller. When the CC-Xcache (or CC_Vcache) component discard a cache line (due to a cache line replacement following a MISS), it signals this change by a CLEANUP packet sent to the cache controller. All those *coherence* packets are implemented as VCI write packets to dedicated memory mapped registers.

- an UPDATE packet (memory controller to cache) has N+2 words : the first word contains the the line index of the modified cache line. The second word contains the index of the first modified word in the line. The N following words contain the N data values.
- an INVALIDATE packet (memory controller to cache) has 1 word : it contains the line index of the modified cache line.
- a CLEANUP packet (cache to memory controller) has 1 word : it contains the line index of the discarded cache line.

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

CC_Xcache

- Usage :
  source:trunk/soclib/soclib/module/internal_component/vci_cc_xcache_wrapper/caba/metadata/vci_cc_xcache_wrapp
- interface :
  source:trunk/soclib/soclib/module/internal_component/vci_cc_xcache_wrapper/caba/source/include/vci_cc_xcache_v
- implementation :
  source:trunk/soclib/soclib/module/internal_component/vci_cc_xcache_wrapper/caba/source/src/vci_cc_xcache_wrap

CC_Vcache

- Usage :
  source:trunk/soclib/soclib/module/internal_component/vci_cc_vcache_wrapper/caba/metadata/vci_cc_vcache_wrapp
- interface :
  source:trunk/soclib/soclib/module/internal_component/vci_cc_vcache_wrapper/caba/source/include/vci_cc_vcache_v
- implementation :
  source:trunk/soclib/soclib/module/internal_component/vci_cc_vcache_wrapper/caba/source/src/vci_cc_vcache_wrap

CABA template parameters

All these 4 component 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_sets,  // number of  associative sets (instruction cache)
    size_t icache_words,  // number of words per line (instruction cache)
    size_t icache_ways,  // number of ways per associative set (instruction cache)
    size_t dcache_sets,  // number of  associative sets (data cache)
    size_t dcache_words,  // number of words per line (data cache)
    size_t dcache_ways );  // number of ways per associative set (data cache)
```

Vcache

```
VciVcacheWrapper(
    sc_module_name insname,
    int proc_id,
    const soclib::common::MappingTable &mt,
    const soclib::common::IntTab &index,
    size_t itlb_m_ways,  // number of ways per associative sets (instruction TLB/M)
    size_t itlb_m_sets,  // number of associative sets (instruction TLB/M)
    size_t itlb_k_ways,  // number of ways per associative sets (instruction TLB/K)
    size_t itlb_k_sets,   // number of associative sets (instruction TLB/K)
    size_t dtlb_m_ways,  // number of ways per associative sets (data TLB/M)
    size_t dtlb_m_sets,  // number of associative sets (data TLB/M)
    size_t dtlb_k_ways,  // number of ways per associative sets (data TLB/K)
    size_t dtlb_k_sets,  // number of associative sets (data TLB/K)
    size_t icache_sets,  // number of  associative sets (instruction cache)
    size_t icache_words,  // number of words per line (instruction cache)
```

```
    size_t icache_ways,  // number of ways per associative set (instruction cache)
    size_t dcache_sets,  // number of  associative sets (data cache)
    size_t dcache_words,  // number of words per line (data cache)
    size_t dcache_ways );  // number of ways per associative set (data cache)
```

CC_Xcache

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

CC_Vcache

```
VciCcVcacheWrapper(
    sc_module_name insname,
    int proc_id,
    const soclib::common::MappingTable &mt,
    const soclib::common::IntTab &initiator_index,
    const soclib::common::IntTab &target_index,
    size_t itlb_m_ways,  // number of ways per associative sets (instruction TLB/M)
    size_t itlb_m_sets,  // number of associative sets (instruction TLB/M)
    size_t itlb_k_ways,  // number of ways per associative sets (instruction TLB/K)
    size_t itlb_k_sets,   // number of associative sets (instruction TLB/K)
    size_t dtlb_m_ways,  // number of ways per associative sets (data TLB/M)
    size_t dtlb_m_sets,  // number of associative sets (data TLB/M)
    size_t dtlb_k_ways,  // number of ways per associative sets (data TLB/K)
    size_t dtlb_k_sets,  // number of associative sets (data TLB/K)
    size_t icache_sets,  // number of  associative sets (instruction cache)
    size_t icache_words,  // number of words per line (instruction cache)
    size_t icache_ways,  // number of ways per associative set (instruction cache)
    size_t dcache_sets,  // number of  associative sets (data cache)
    size_t dcache_words,  // number of words per line (data cache)
    size_t dcache_ways );  // number of ways per associative set (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

CC_Xcache & **CC_Vcache**

- sc_in<bool> **p_resetn** : Global system reset
- sc_in<bool> **p_clk** : Global system clock
- soclib::caba::VciInitiator<vci_param> **p_vci_ini** : The VCI initiator port
- soclib::caba::VciTarget<vci_param> **p_vci_tgt** : The VCI target port

3) TLM-T Implementation

- interface :
  source:trunk/soclib/soclib/module/internal_component/vci_xcache_wrapper/tlmt/source/include/vci_xcache_wrapper
- implementation :
  source:trunk/soclib/soclib/module/internal_component/vci_xcache_wrapper/tlmt/source/src/vci_xcache_wrapper.cpp

TLM-T template parameters

TLM-T constructor parameters

TLM-T ports