VciBlockDevice

1) Functional Description

This component emulates a disk controller, that can perform data transfers between a file belonging to the host system and a buffer in the memory of the virtual system. The file name is an argument of the constructor, as well as the block size (in bytes), and the simulated access latency (in cycles). As most block devices, this VCI component has a DMA capability, and is both a target and an initiator.

- It is addressed as a target to be configured for a transfer.
- It is acting as an initiator to do the transfer

There is only one block device handled by this component. Both read and write transfers are supported. An IRQ is optionally asserted when transfer is completed.

This hardware component checks for segmentation violation, and can be used as a default target.

It contains 9 memory-mapped, 32 bits registers:

- **BLOCK_DEVICE_BUFFER** (read/write)

Physical address 32 LSB bits of the source (or destination) buffer in SoC memory.

- **BLOCK_DEVICE_BUFFER_EXT** (read/write)

Physical address 32 MSB bits of the source (or destination) buffer in SoC memory. This register is only used in platform where the physical address width is larger than 32 bits.

- **BLOCK_DEVICE_COUNT** (read/write)

Number of blocks to be transfered.

- **BLOCK_DEVICE_LBA** (read/write)

Logical Base Address (index of the first block in the block device)

- **BLOCK_DEVICE_OP** (write only)

Type of operation, writing here initiates the operation.This register goes back to BLOCK_DEVICE_NOOP when operation is finished. The following operations codes are defined:

`BLOCK_DEVICE_NOOP`	No operation
`BLOCK_DEVICE_READ`	Transfer from block device to memory
`BLOCK_DEVICE_WRITE`	Transfer from memory to block device

- **BLOCK_DEVICE_STATUS** (read only)

Transfer status: Reading this register while not BUSY resets its value to IDLE, and acknowledge the IRQ. Value may be one of :

```
BLOCK_DEVICE_IDLE
BLOCK_DEVICE_BUSY
BLOCK_DEVICE_READ_SUCCESS
BLOCK_DEVICE_WRITE_SUCCESS
BLOCK_DEVICE_READ_ERROR
BLOCK_DEVICE_WRITE_ERROR
```

Writing any value to this register makes a soft reset: if the status is not IDLE, the on-going transfer is stopped as soon as possible, and the status is reset to IDLE.

- **BLOCK_DEVICE_IRQ_ENABLE** (read/write)

Boolean enabling the IRQ line

- **BLOCK_DEVICE_SIZE** (read only)

Number of blocks addressable in the block device

- **BLOCK_DEVICE_BLOCK_SIZE** (read only)

Block size (in bytes)

When the component is not IDLE, only the BLOCK_DEVICE_STATUS register can be accessed by software: Any read/write access to any other register returns an error.

For extensibility issues, you should access this component using globally-defined offsets. You should include file `soclib/block_device.h` that defines `BLOCK_DEVICE_COUNT`, `BLOCK_DEVICE_READ`, ...

Sample code: Please see reference implementation in source:trunk/soclib/soclib/platform/topcells/caba-vgmn-block_device-mips32el

(add -I/path/to/soclib/include to your compilation command-line)

2) Component definition & usage

source:trunk/soclib/soclib/module/connectivity_component/vci_block_device/caba/metadata/vci_block_device.sd?

See SoclibCc/VciParameters

```
Uses( 'vci_block_device', **vci_parameters )
```

3) CABA Implementation

CABA sources

- interface : source:trunk/soclib/soclib/module/connectivity_component/vci_block_device/caba/source/include/vci_block_device.
- implementation : source:trunk/soclib/soclib/module/connectivity_component/vci_block_device/caba/source/src/vci_block_device.cpp?

CABA Constructor parameters

```
VciBlockDevice(
     sc_module_name name,   //  Component Name
     const soclib::common::MappingTable &mt, // MappingTable
     const soclib::common::IntTab &srcid,    // Initiator index
     const soclib::common::IntTab &tgtid,    // Target index
     const std::string &filename, // mapped file, may be a host block device
     const uint32_t block_size = 512, // block size in bytes
     const uint32_t latency = 0);  // initial access time (number of cycles)
```

CABA Ports

- **p_resetn** : Global system reset
- **p_clk** : Global system clock
- **p_vci_target** : The VCI target port
- **p_vci_initiator** : The VCI initiator port
- **p_irq** : Interrupt port

4) TLM-DT Implementation

TLM-DT sources

- interface : source:trunk/soclib/soclib/module/connectivity_component/vci_block_device/tlmdt/source/include/vci_block_device
- implementation : source:trunk/soclib/soclib/module/connectivity_component/vci_block_device/tlmdt/source/src/vci_block_device.cpp

TLM-DT Constructor parameters

```
VciBlockDevice(
     sc_module_name name,   //  Component Name
     const soclib::common::MappingTable &mt, // MappingTable
     const soclib::common::IntTab &srcid,    // Initiator index
     const soclib::common::IntTab &tgtid,    // Target index
     const std::string &filename, // mapped file, may be a host block device
     const uint32_t block_size = 512, // block size in bytes
     const uint32_t latency = 0);  // initial access time (number of cycles)
```

TLM-DT Ports

- **p_vci_target** : The VCI target port
- **p_vci_initiator** : The VCI initiator port
- **p_irq** : Interrupt port