SocLib Components General Index

VciBlockDevice

1) Functional Description

This VCI component 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, limited to 2^{41} bytes. An IRQ is optionally asserted when transfer is finished.

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

It contains the following memory-mapped registers:

- `BLOCK_DEVICE_BUFFER` Physical address of the buffer in SoC memory
- `BLOCK_DEVICE_COUNT` Count of blocks to transfer
- `BLOCK_DEVICE_LBA` Base sector for transfer
- `BLOCK_DEVICE_OP` Type of operation, writing here initiates the operation.
  This register goes back to BLOCK_DEVICE_NOOP when operation is finished.
- `BLOCK_DEVICE_STATUS` State of the transfer, -1 on failure
- `BLOCK_DEVICE_IRQ_ENABLE` Boolean enabling the IRQ line
- `BLOCK_DEVICE_SIZE` Number of blocks addressable in the controller (read-only)

The following operations codes are defined:

- `BLOCK_DEVICE_NOOP` Nothing
- `BLOCK_DEVICE_READ` read()
- `BLOCK_DEVICE_WRITE` write()

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

Sample code:

```
#include "soclib/block_device.h"

static const void* bd_base = 0xc0000000;


int block_read( const size_t lba, void *buffer, const size_t len )
{
```

```
        soclib_io_set(bd_base, BLOCK_DEVICE_LBA, lba);
        soclib_io_set(bd_base, BLOCK_DEVICE_BUFFER, (uint32_t)buffer);
        soclib_io_set(bd_base, BLOCK_DEVICE_COUNT, len);
        soclib_io_set(bd_base, BLOCK_DEVICE_OP, BLOCK_DEVICE_READ);
        while (soclib_io_get(bd_base, BLOCK_DEVICE_OP))
                ;
        return soclib_io_get(bd_base, BLOCK_DEVICE_STATUS);
}

int block_write( const size_t lba, const void *buffer, const size_t len )
{
        soclib_io_set(bd_base, BLOCK_DEVICE_LBA, lba);
        soclib_io_set(bd_base, BLOCK_DEVICE_BUFFER, (uint32_t)buffer);
        soclib_io_set(bd_base, BLOCK_DEVICE_COUNT, len);
        soclib_io_set(bd_base, BLOCK_DEVICE_OP, BLOCK_DEVICE_WRITE);
        while (soclib_io_get(bd_base, BLOCK_DEVICE_OP))
                ;
        return soclib_io_get(bd_base, BLOCK_DEVICE_STATUS);
}

uint32_t block_size()
{
        return soclib_io_get(bd_base, BLOCK_DEVICE_SIZE);
}
```

(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.h
- 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::IntTab & index,  // Target index
     const soclib::common::MappingTable &mt,  // MappingTable
     const std::string &filename ) // mapped file, may be a host block device
```

CABA Ports

- sc_in<bool> **p_resetn** : Global system reset
- sc_in<bool> **p_clk** : Global system clock

- soclib::caba::VciTarget<vci_param> **p_vci_target** : The VCI target port
- soclib::caba::VciInitiator<vci_param> **p_vci_initiator** : The VCI initiator port
- sc_out<bool> **p_irq** : Interrupt port

4) TLM-T Implementation

The TLM-T implementation is not yet available.