VciFdAccess Functional Description

This VCI component is both a target and an initiator.

- Addressing as a target allows to configure it for a transfer.
- Initiator will do the transfer

There is only one access context handled at a time, but fd can be changed for each operation, many fd can be open at the same time.

An IRQ is optionally asserted when transfer is finished.

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

Memory region layout

- `FD_ACCESS_FD` File descriptor for the transfer
- `FD_ACCESS_BUFFER` Buffer (in SoC memory) for the transfer
- `FD_ACCESS_SIZE` Size of the transfer
- `FD_ACCESS_HOW` Type of open() operation
- `FD_ACCESS_WHENCE` Type of offset when changing file pointer
- `FD_ACCESS_OP` Type of operation, writing here initiates the operation.
  This register goes back to FD_ACCESS_NOOP when operation is finished.
- `FD_ACCESS_RETVAL` Return value
- `FD_ACCESS_ERRNO` Errno
- `FD_ACCESS_IRQ_ENABLE` Boolean enabling the IRQ line
- `FD_ACCESS_MODE` (aliases with WHENCE) Mode for open() with a O_CREAT

Operations codes

- `FD_ACCESS_NOOP` Nothing
- `FD_ACCESS_OPEN` open()
  path: `FD_ACCESS_BUFFER`
  `FD_ACCESS_SIZE` must contain the length of path
  returned fd is in `FD_ACCESS_RETVAL`
- `FD_ACCESS_CLOSE` close()
- `FD_ACCESS_READ` read()

- `FD_ACCESS_WRITE` write()
- `FD_ACCESS_LSEEK` lseek()

Component usage

For extensibility issues, you should access this component using globally-defined offsets.

You should include file `soclib/fd_access.h` from your software, it defines `FD_ACCESS_FD`, `FD_ACCESS_BUFFER`, ...

Sample code:

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

static const void* fd_base = 0xc0000000;

int open( const char *path, const int how, const int mode )
{
        soclib_io_set(fd_base, FD_ACCESS_BUFFER, (uint32_t)path);
        soclib_io_set(fd_base, FD_ACCESS_SIZE, strlen(path));
        soclib_io_set(fd_base, FD_ACCESS_HOW, how);
        soclib_io_set(fd_base, FD_ACCESS_MODE, mode);
        soclib_io_set(fd_base, FD_ACCESS_OP, FD_ACCESS_OPEN);
        while (soclib_io_get(fd_base, FD_ACCESS_OP))
                ;
        errno = soclib_io_get(fd_base, FD_ACCESS_ERRNO);
        return soclib_io_get(fd_base, FD_ACCESS_RETVAL);
}

int close( const int fd )
{
        soclib_io_set(fd_base, FD_ACCESS_FD, fd);
        soclib_io_set(fd_base, FD_ACCESS_OP, FD_ACCESS_CLOSE);
        while (soclib_io_get(fd_base, FD_ACCESS_OP))
                ;
        errno = soclib_io_get(fd_base, FD_ACCESS_ERRNO);
        return soclib_io_get(fd_base, FD_ACCESS_RETVAL);
}

// Beware your caches could contain stale memory image after this call
int read( const int fd, const void *buffer, const size_t len )
{
        soclib_io_set(fd_base, FD_ACCESS_FD, fd);
        soclib_io_set(fd_base, FD_ACCESS_BUFFER, (uint32_t)buffer);
        soclib_io_set(fd_base, FD_ACCESS_SIZE, len);
        soclib_io_set(fd_base, FD_ACCESS_OP, FD_ACCESS_READ);
        while (soclib_io_get(fd_base, FD_ACCESS_OP))
                ;
        errno = soclib_io_get(fd_base, FD_ACCESS_ERRNO);
        return soclib_io_get(fd_base, FD_ACCESS_RETVAL);
}

int write( const int fd, const void *buffer, const size_t len )
{
        soclib_io_set(fd_base, FD_ACCESS_FD, fd);
        soclib_io_set(fd_base, FD_ACCESS_BUFFER, (uint32_t)buffer);
        soclib_io_set(fd_base, FD_ACCESS_SIZE, len);
        soclib_io_set(fd_base, FD_ACCESS_OP, FD_ACCESS_WRITE);
        while (soclib_io_get(fd_base, FD_ACCESS_OP))
                ;
        errno = soclib_io_get(fd_base, FD_ACCESS_ERRNO);
```

```
        return soclib_io_get(fd_base, FD_ACCESS_RETVAL);
}

int lseek( const int fd, const off_t offset, int whence )
{
        soclib_io_set(fd_base, FD_ACCESS_FD, fd);
        soclib_io_set(fd_base, FD_ACCESS_SIZE, offset);
        soclib_io_set(fd_base, FD_ACCESS_WHENCE, whence);
        soclib_io_set(fd_base, FD_ACCESS_OP, FD_ACCESS_LSEEK);
        while (soclib_io_get(fd_base, FD_ACCESS_OP))
                ;
        errno = soclib_io_get(fd_base, FD_ACCESS_ERRNO);
        return soclib_io_get(fd_base, FD_ACCESS_RETVAL);
}
```

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

Component definition

Available in source:trunk/soclib/module/connectivity_component/vci_fd_access/caba/metadata/vci_fd_access.sd

Usage

VciFdAccess has no other parameter than VCI ones, it may be used like others, see SoclibCc/VciParameters

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

VciFdAccess CABA Implementation

The caba implementation is in

- source:trunk/soclib/module/connectivity_component/vci_fd_access/caba/source/include/vci_fd_access.h
- source:trunk/soclib/module/connectivity_component/vci_fd_access/caba/source/src/vci_fd_access.cpp

Template parameters:

- The VCI parameters

Constructor parameters

```
VciFdAccess(
     sc_module_name name,   //  Component Name
     const soclib::common::IntTab & index,  // Target index
     const soclib::common::MappingTable &mt )   // MappingTable
```

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