Component description

This component is a caba utility component. This is a class which

- will handle most of the simple cases of VCI targets
- may handle many segments with different address ranges
- handle atomic operations for you
- will let you define the behaviour through callbacks

It has a few template parameters:

- VCI parameters, the standart ones
- `default_target`: Whether it will send back an error response if the component is not authoritative for received address
- `support_llsc`: Wheter atomic operations should be supported.

It has a few instance parameters:

- VCI target port to handle
- List of segments to handle, in the same form that the return value from `getSegmentList()` in the MappingTable
- fifo size, the amount of VCI words that may be served in a pipeline fashion
    - setting to 1 means only one vci request word will be handled at a time, there will be no pipelining
    - setting to more than 1 will enable pipelining

If default target, this component filters the requests and directly returns an error if address is out of handled segments.

If atomic operations are enabled, this component filters atomic operations and calls on_write callback only if atomic write is successful.

Component usage

Let's suppose we implement a VCI target component `MyComponent`.

Let's have a target VCI port and a FSM handler.

Let's also have two callbacks for read and write behaviour:

```
template<typename vci_param>
class MyComponent
    : soclib::caba::BaseModule
{
    // Import some definitions from vci_param
    typedef typename vci_param::addr_t vci_addr_t;
    typedef typename vci_param::data_t vci_data_t;

    sc_in<bool> p_resetn;
    sc_in<bool> p_clk;
    soclib::caba::VciTarget<vci_param> p_vci;
    soclib::caba::VciTargetFsm<vci_param> m_vci_fsm;

    // Callbacks, see below for parameters' meanings
    bool on_write( size_t seg, vci_addr_t addr, vci_data_t data, int be);
    bool on_read( size_t seg, vci_addr_t addr, vci_data_t &data );

    void transition();
    void genMoore();

    ...

public:
    MyComponent(
        sc_module_name name,
        const soclib::common::MappingTable &mt,
        const soclib::common::IntTab &ident );
};
```

Initialization

In the constructor, we must initialize the target fsm and its callbacks:

```
template <typename vci_param>
MyComponent<vci_param>::MyComponent(
    sc_module_name name,
    const soclib::common::MappingTable &mt,
    const soclib::common::IntTab &ident )
    : p_resetn("resetn"),
      p_clk("clk"),
      p_vci("vci"),       // Constructor for the vci port, give it a name
      m_vci_fsm(p_vci, mt.getSegmentList(ident), 1), // Constructor for the FSM, with 1 request
      ... // other constructors
{
    // Constructor code
    // Set callbacks
    m_vci_fsm.on_read_write(on_read, on_write);

    // Like any other Caba module:
    SC_METHOD(transition);
    dont_initialize();
    sensitive << p_clk.pos();

    SC_METHOD(genMoore);
    dont_initialize();
    sensitive << p_clk.neg();
}
```

Now for transition and Moore generation, we must also call the fsm:

```
template <typename vci_param>
void MyComponent<vci_param>::transition()
{
    if ( ! p_resetn.read() ) {
        m_vci_fsm.reset();
        // Other code specific to MyComponent
        return;
    }

    m_vci_fsm.transition();

    // Other code specific to MyComponent
}

template <typename vci_param>
void MyComponent<vci_param>::genMoore()
{
    m_vci_fsm.genMoore();

    // Other code specific to MyComponent
}
```

Callbacks

On read function

This function is called when a read request comes from VCI port.

If the request is a multi-word request, callback will be called once per word.

```
bool on_read( size_t seg, vci_addr_t addr, vci_data_t &data )
```

return value
: true if this request is valid, false if not. If false, an error value is returned in response packet for this word.

seg
: index of the served segment in the list passed in constructor. As segments caracteristics are copied, you may retrieve them with other functions, see below

addr
: offset in segment, this value is the vci-requested address minus the hit segment base address.

data
: data value to return in response packet

On write function

This function is called when a write request comes from VCI port.

If the request is a multi-word request, callback will be called once per word.

```
bool on_write(size_t seg, vci_addr_t addr, vci_data_t data, int be)
```

return value
: true if this request is valid, false if not. If false, an error value is returned in response packet for this word.

seg
: index of the served segment in the list passed in constructor. As segments caracteristics are copied, you may retrieve them with other functions, see below

addr
: offset in segment, this value is the vci-requested address minus the hit segment base address.

data
: data value received in the command packet

be
: byte-enable field from the request

Other functions

`getSize( size_t segment_index )`
: get the size of the handled segment

`getBase( size_t segment_index )`
: get the base address of the handled segment

`getEnd( size_t segment_index )`
: get the end address of the handled segment (addresses above are not valid for this segment)

`getName( size_t segment_index )`
: get the name of the handled segment, for debugging purposes

`nbSegments()`
: get the cardinal of handled segments set

`currentSourceId()`
: get the srcid of the currently served request