

# Vérification du protocole VCI

Ce document présente les composants de vérification dynamique permettant de s'assurer que le protocole VCI est respecté durant une simulation transactionnelle « tlm ». Lorsque ce n'est pas le cas les erreurs identifiées sont écrites dans un fichier de log spécifié par l'utilisateur.

Cette vérification est mise en œuvre en incluant le fichier `soclib/module/verification_component/tlm/source/include/vci_filtered_ports.h`. La vérification est alors effectuée lorsqu'un paquet est envoyé et lorsqu'il est reçu.

## Débogage et messages d'erreurs

Les messages d'erreur identifient le nombre de paquets traités depuis le début de la simulation. Ces nombres sont une aide au débogage.

Une utilisation sous gdb lorsque peu de `VciFilteredInitiator` et de `VciFilteredTarget` ont été instanciés est :

```
> break VciFilteredInitiator::writeError
> run
> p m_nb_packets
102
```

suivi de

```
> break VciFilteredInitiator::check_command if m_nb_packets == 101
> run
```

Il suffit ensuite d'avancer pas à pas de manière à comprendre l'origine de l'erreur, éventuellement en activant des points d'arrêt sur d'autres processus.

Le format des messages d'erreur est de la forme :

```
ERROR : Protocol Error "message" on VciFilteredInitiator001 for the packet 102 issued from request !!!
ERROR : Protocol Error "message" on VciFilteredInitiator 001 for the packet 51 issued from response !!!
```

où message est parmi :

- The fields `nwords` and `contig` should be constant during the reception of chain of packets
  - L'erreur a lieu lorsque `nwords` ou `contig` ne sont pas constants durant la réception de tous les paquets composant une chaîne.
- Contiguous packets should have a length within the form  $2^n$ 
  - L'erreur a lieu lorsque `contig` = true et que `nwords` n'est pas une puissance de 2.
- The response packet does not correspond to any request
  - L'erreur a lieu lorsqu'une réponse arrive alors qu'aucune requête ne lui correspond, ce qui se produit lorsque le paquet reçu ne correspond à aucun paquet reçu mais non encore traité.
- The response packet has not the expected length
  - L'erreur a lieu lorsqu'une réponse arrive alors que la requête qui devrait lui correspondre ne lui correspond pas, en particulier sur le critère de la longueur. La raison est soit un problème de correspondance, soit le fait que le paquet-réponse soit composé de trop ou pas assez de cellule.

## Le composant VciFilteredInitiator

Ce composant de vérification s'utilise de la même manière que le composant `VciInitiator` et remplace ce dernier pour effectuer les vérifications ci-dessus. Il vérifie les données émises par la méthode

check\_command avant l'appel à la méthode send et vérifie les réponses par la méthode check\_command avant l'appel au call\_back.

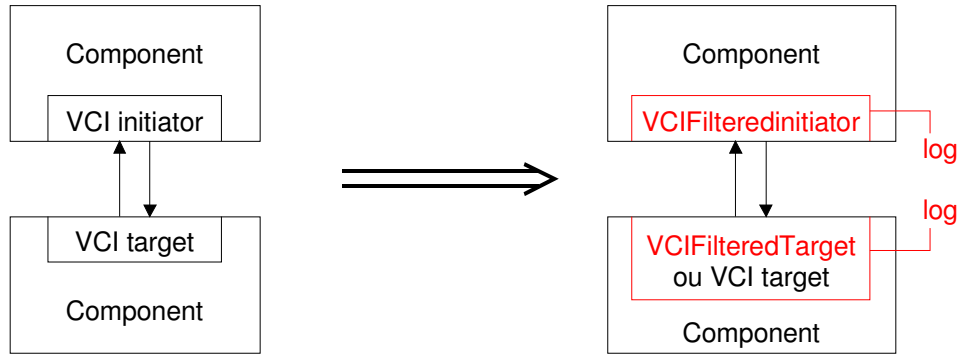


Figure 1 : Filtrage des requêtes et réponses au niveau initiateur

L'interface tlmt de ces composants est la suivante. Lorsqu'un utilisateur souhaite introduire un composant de vérification sur un signal de type VCI, il introduit à l'élaboration le code en rouge et en italique (sur la partie droite). La vérification est alors automatique.

```
template<typename vci_param>
class VciFilteredInitiator : public VciInitiator<vci_param> {
    class verif_callback
    : public tlmt_callback_base<vci_rsp_packet<vci_param>> > {
    private:
        VciFilteredInitiator* initiator;
        tlmt_callback_base<vci_rsp_packet<vci_param>>* call_back;

    public:
        verif_callback(VciFilteredInitiator* initiator_src,
            tlmt_callback_base<vci_rsp_packet<vci_param>>* src)
            : initiator(initiator_src), call_back(src) {}
        virtual void operator()(param_t param,
            const tlmt_time &time)
        { initiator->check_response(initiator->rsp_in);
            return (*call_back)(param, time, m_data);
        }
    };

    std::ostream* m_log_file;
    verif_callback m_call_back;

public:
    VciFilteredInitiator(const std::string &name, callback_t cb,
        tlmt_thread_context *opt_ref = NULL)
        : VciInitiator<vci_param>(name, &m_call_back, opt_ref),
          m_call_back(this, cb) {}

    void check_response(vci_rsp_packet<vci_param>* pkt);
    void check_command(vci_cmd_packet<vci_param>* pkt);
    void send(vci_cmd_packet<vci_param>* pkt,
        const tlmt_core::tlmt_time &time)
    { check_command(pkt);
        return VciInitiator<vci_param>::send(pkt, time);
    }
};
```

// sc\_main implementation

```
#include <tlmt>
#include "tlmt_base_module.h"
#ifdef SOCLIB_VERIF
#include "vci_filtered_ports.h"
#else
#include "vci_ports.h"
#endif

namespace soclib { namespace tlmt {

template <typename vci_param>
class VciSimpleInitiator : public BaseModule {
    ...
protected:
    SC_HAS_PROCESS(VciSimpleInitiator);

public:
#ifdef SOCLIB_VERIF
    VciFilteredInitiator<vci_param> p_vci;
#else
    VciInitiator<vci_param> p_vci;
#endif

    VciSimpleInitiator(sc_core::sc_module_name name, ...);
    void rspReceived(vci_rsp_packet<vci_param>* pkt,
        const tlmt_time& time, void* private_data);
    void behavior();
};

soclib::tlmt::VciSimpleInitiator<MyVciParams> vciFst("FstComponent");

#ifdef SOCLIB_VERIF
std::ofstream log_file("verif.log");
vciFst.p_vci.setLogOut(log_file);
#endif

soclib::tlmt::VciSimpleTarget<MyVciParams> vciSnd("SndComponent");
vciFst.p_vci(vciSnd.p_vci);
vciSnd.p_vci(vciFst.p_vci);
```

## Le composant VciFilteredTarget

Ce composant de vérification s'utilise de la même manière que le composant VciInitiator et remplace ce dernier pour effectuer les vérifications ci-dessus. Il vérifie les données émises par la méthode check\_command avant l'appel à la méthode send et vérifie les réponses par la méthode check\_command avant l'appel au call\_back.

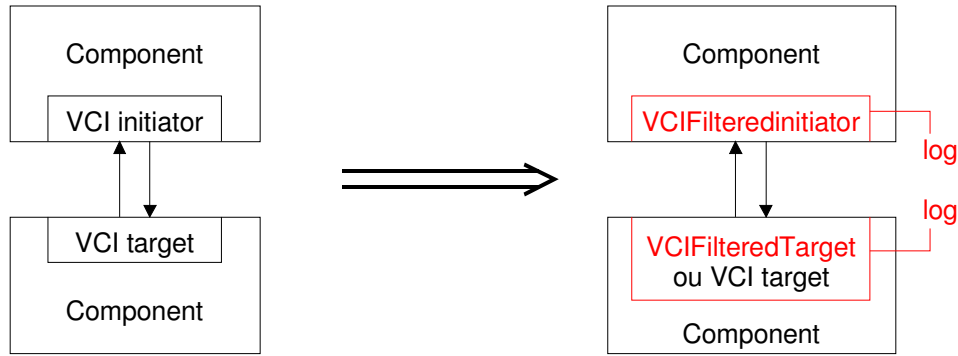


Figure 2 : Filtrage des requêtes et réponses au niveau initiateur

L'interface tlmt de ces composants est la suivante. Lorsqu'un utilisateur souhaite introduire un composant de vérification sur un signal de type VCI, il introduit à l'élaboration le code en rouge et en italique (sur la partie droite). La vérification est alors automatique.

```

template<typename vci_param>
class VciFilteredTarget : public VciTarget<vci_param> {
    class verif_callback
    : public tlmt_callback_base<vci_cmd_packet<vci_param>> {
    private:
        VciFilteredTarget* target;
        tlmt_callback_base<vci_cmd_packet<vci_param>>* call_back;

    public:
        verif_callback(VciFilteredTarget* target_src,
            tlmt_callback_base<vci_cmd_packet<vci_param>>* src)
            : target(target_src), call_back(src) {}
        virtual void operator()(param_t param,
            const tlmt_time &time)
        { target->check_command(target->cmd_in);
            return (*call_back)(param, time, m_data);
        }
    };

    std::ostream* m_log_file;
    verif_callback m_call_back;

public:
    VciFilteredTarget(const std::string &name, callback_t cb,
        tlmt_thread_context *opt_ref = NULL)
        : VciTarget<vci_param>(name, &m_call_back, opt_ref),
          m_call_back(this, cb) {}

    void check_command(vci_cmd_packet<vci_param>* pkt);
    void check_response(vci_rsp_packet<vci_param>* pkt);
    void send(vci_rsp_packet<vci_param>* pkt,
        const tlmt_core::tlmt_time &time)
    { check_response(pkt);
        return VciTarget<vci_param>::send(pkt, time);
    }
};

// sc_main implementation

#include <tlmt>
#include "tlmt_base_module.h"
#ifdef SOCLIB_VERIF
#include "vci_filtered_ports.h"
#else
#include "vci_ports.h"
#endif

namespace soclib { namespace tlmt {

template <typename vci_param>
class VciSimpleTarget : public BaseModule {
    ...
protected:
    SC_HAS_PROCESS(VciSimpleTarget);

public:
#ifdef SOCLIB_VERIF
    VciFilteredTarget<vci_param> p_vci;
#else
    VciTarget<vci_param> p_vci;
#endif

    VciSimpleTarget(sc_core::sc_module_name name, ...);
    void cmdReceived(vci_cmd_packet<vci_param>* pkt,
        const tlmt_time& time, void* private_data);
};

soclib::tlmt::VciSimpleInitiator<MyVciParams> vciFst("FstComponent");
soclib::tlmt::VciSimpleTarget<MyVciParams> vciSnd("SndComponent");
#ifdef SOCLIB_VERIF
std::ofstream log_file("verif.log");
vciSnd.p_vci.setLogOut(log_file);
#endif
vciFst.p_vci(vciSnd.p_vci);
vciSnd.p_vci(vciFst.p_vci);

```

## Conclusion

Les ports vérifiant le protocole de communication ont été mis en place pour ne pas perturber le système. Il est néanmoins important de pouvoir désactiver cette vérification ou de ne vérifier le protocole VCI qu'entre certains composants. Cette désactivation est principalement mise en œuvre en plaçant le code de vérification sous condition que la macro SOCLIB\_VERIF est définie et en proposant plusieurs modes de compilation pour le système : un mode avec g++ -O2 ... et un mode avec g++ -DSOCLIB\_VERIF.