Vérification du protocole Peripheral VCI

Ce document présente les composants de vérification dynamique permettant de s'assurer que le protocole Peripheral VCI est respecté durant une simulation. 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 un des deux fichiers soclib\module\verification_component\pvci\caba\source\include\pvci_assert.h, soclib\module\verification_component\pvci\caba\source\include\pvci_filter.h, voire les deux fichiers pour créer les composants de vérification qui sont soit montés en série, soit sont de simples observateurs. Ces composants de conformité vérifient le « Handshake protocol » du Peripheral VCI.

Cette vérification est à rapprocher de composants comme soclib::caba::VciPiInitiatorWrapper défini dans soclib\module\network_component\vci_pi_initiator_wrapper\caba\source\include\vci_pi_initiator_wrapper.h.

Débogage et messages d'erreurs

Les messages d'erreur identifient le nombre de paquets traités depuis le début de la simulation et la cellule du paquet associée à l'erreur. Ces nombres sont une aide au déboguage.

Une utilisation sous gdb lorsque peu de PVciFilter ont été instantiés est :

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

suivi de

```
> break PVciFilter::acquireCell 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. PVciFilter et PVciAssert sont utilisés de la même manière sous gdb.

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

ERROR : Protocol Error "message" on PVciFilter001 for the packet 100, the cell 2 issued from request!!!

où message est parmi :

- The peripheral protocol does not accept "bubble" in a VCI command packet
  - L'erreur a lieu lorsque le signal val n'est pas maintenu sur le composant initiateur alors que l'acknowledgement n'a pas eu lieu de la part du composant cible.
- The peripheral protocol does not accept "change" during acknowledgement
  - L'erreur a lieu lorsqu'un des signaux address, be, cmd, data, eop n'est pas constant sur le composant initiateur alors que l'acknowledgement n'a pas eu lieu de la part du composant cible.
- Violation in the peripheral protocol : the reset command had no effect
  - L'erreur a lieu lorsque le reset n'a pas effectivement repositionné les champs cmdack et val à 0 dans la limite des 8 cycles nécessaires à cet effet. Ce nombre de 8 cycles peut être spécifié statiquement par l'intermédiaire de la méthode setDefaultReset.
- The peripheral protocol accepts only VCI BE corresponding to known formats

- L'erreur a lieu lorsque le signal be n'est pas parmi les valeurs acceptées par le protocole, notamment en default mode (par opposition au free-BE mode) lorsqu'il autorise des lignes non restreintes à des cases contiguës.

- The peripheral protocol accepts only increasing addresses in a given packet
  - L'erreur a lieu lorsque les adresses ne sont pas croissantes sur des cellules d'un même paquet. Le protocole PVCI impose que l'adresse de la cellule suivante soit CELLSIZE + l'adresse de la cellule courante et que l'adresse de la cellule courante soit alignée en conformité avec les bornes de la cellule.

Le composant PVciFilter

Ce composant de vérification est monté en série dans une architecture existante bâtie autour du protocole VCI. Il capture ses données en fin de front descendant après que toutes les méthodes genMoore des composants aient terminé leur travail et notre composant effectue la vérification durant le front montant.

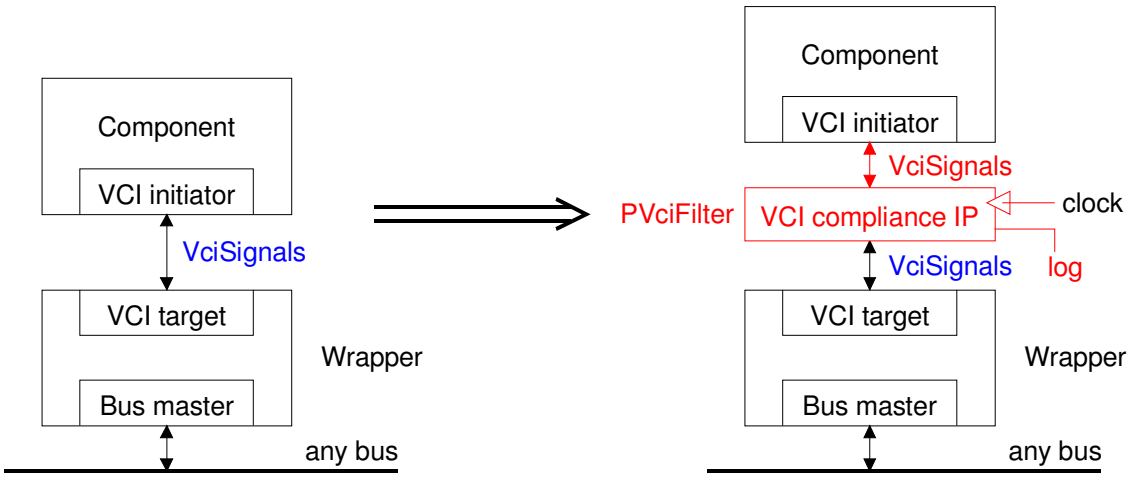

Figure 1 : composant de vérification monté en série

L'horloge de PVciFilter n'est pas forcément globale mais doit être commune avec le composant initiateur et le composant cible.

L'interface caba de ces composants est la suivante. Lorsqu'un utilisateur souhaite introduire un composant de vérification sur un signal de type Peripheral 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 PVciFilter : public soclib::caba::BaseModule {
 public:
  struct In {
    In(const std::string &name);
    void operator()(VciSignals<vci_param> &sig);
  };

  struct Out {
    Out(const std::string &name);
    void operator()(VciSignals<vci_param> &sig);
  };

  sc_in<bool> p_clk;
  In p_in;
  Out p_out;

 protected:
  SC_HAS_PROCESS(PVciFilter);

 public:
  PVciFilter(sc_module_name insname);
```

```
// sc_main implementation

VciFstComponent<MyVciParams> vciFst("VciFstComponent");
VciSndComponent<MyVciParams> vciSnd("VciSndComponent");
sc_clock clk("Clock", 1, 0.5, 0.0);
soclib::caba::VciSignals<MyVciParams> vciSignals("VciSignals");

#ifdef SOCLIB_VERIF
std::ofstream log_file("verif.log");
soclib::caba::VciSignals<MyVciParams>
                                vciSignalsVerif("VciSignals_verif");
soclib::caba::PVciFilter<MyVciParams> vciFilter("VciFilter_verif");
vciFilter.p_clk(clk);
vciFilter.setLogOut(log_file);
vciFilter.p_in(vciSignals);
vciFilter.p_out(vciSignalsVerif);
// optional
vciFilter.setDefaultResult(8);
vciFilter.setDefaultMode();
#else
  VciSignals<MyVciParams>& vciSignalsVerif = vciSignals;
#endif

vciFst.p_clk(clk);
vciFst.p_vci(vciSignals);
vciSnd.p_clk(clk);
vciSnd.p_vci(vciSignalsVerif);
```

```
  void reset(); // process sensitive to p_resetn.pos()
  void copy(); // process that copies inputs to outputs
  void transition(); // process that does
          // the verification on the rising edge of p_clk

   // configuration methods
     // set max cycle number before reset is effective
  void setDefaultReset(int uDefaultReset);
     // set the error file
  void setLogOut(std::ostream& out);
     // assume that PVCI is in default mode for be
  void setDefaultMode();
     // assume that PVCI is in free mode for be
  void setFreeMode();
};
```

Le composant PVciAssert

Ce composant de vérification se contente d'observer le contenu d'un VciSignals sur chaque front montant d'horloge suivant le schéma de la Figure 2. Du point de vue fonctionnel il effectue les mêmes vérifications que PVciFilter, mais il est moins intrusif.

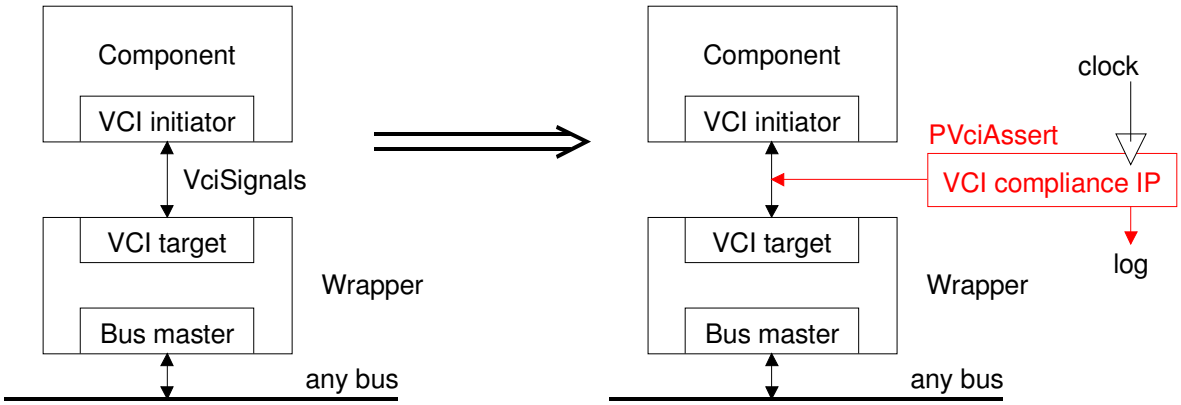

Figure 2 : composant de vérification observateur

Les composants PVciAssert sont garantis ne pas perturber le comportement du système. En particulier, ils n'ont pas de capacité de filtrer. La vérification de la cohérence a lieu sur chaque front montant d'horloge. Ces composants n'ont qu'une sortie qui est le fichier de log pour enregistrer les erreurs détectées.

L'horloge de PVciAssert, comme celle de PVciFilter, n'est pas forcément globale mais doit être commune avec le composant initiateur et le composant cible, afin de ne pas manquer des évènements sur les entrées.

L'interface caba de ces composants est la suivante. Lorsqu'un utilisateur souhaite introduire un composant de vérification sur un signal de type Peripheral 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 PVciAssert : public soclib::caba::BaseModule {
  public:
   VciSignals<vci_param>& observedSignals;
   sc_in<bool> p_clk;

  protected:
   SC_HAS_PROCESS(PVciAssert);

  public:
   PVciAssert(VciSignals<vci_param>& ref,
      sc_module_name insname);

   void reset(); // process sensitive to p_resetn.pos()
   void transition(); // process that does
           // the verification on the rising edge of p_clk

    // configuration methods
     // set max cycle number before reset is effective
   void setDefaultReset(int uDefaultReset);
     // set the error file
   void setLogOut(std::ostream& out);
     // assume that PVCI is in default mode for be
   void setDefaultMode();
     // assume that PVCI is in free mode for be
   void setFreeMode();
};
```

```
// sc_main implementation

VciFstComponent<MyVciParams> vciFst("VciFstComponent");
VciSndComponent<MyVciParams> vciSnd("VciSndComponent");
sc_clock clk("Clock", 1, 0.5, 0.0);
soclib::caba::VciSignals<MyVciParams> vciSignals("VciSignals");

#ifdef SOCLIB_VERIF
std:auto_ptr<BasicVciAssert<MyVciParams> > pvciAssert;
std::ofstream log_file("verif.log");
pvciAssert = new PVciAssert<MyVciParams>(vciSignals, "vciAssert");
pvciAssert->p_clk(clk);
pvciAssert->setLogOut(log_file);
// optional
pvciAssert->setDefaultResult(8);
pvciAssert->setDefaultMode();
#endif

vciFst.p_clk(clk);
vciFst.p_vci(vciSignals);
vciSnd.p_clk(clk);
vciSnd.p_vci(vciSignal);
```

Conclusion

L'introduction de composants de vérification a été mise en place pour ne pas perturber le système. Néanmoins elle ralentit les temps de simulation. Il est donc important de pouvoir la désactiver ou de ne vérifier le protocole VCI qu'entre certains composants. Cette désactivation est mise en œuvre :

- soit 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.
- soit en ne créant les composants de vérification que si spécifié par l'utilisateur sur la ligne de commande.