Chapitre II : Introduction aux systèmes embarqués

1. Les systèmes embarqués

Définition (1)

Embedded system : tout système conçu pour résoudre un problème ou une tâche spécifique mais n'est pas un ordinateur d'usage général.

Utilisent généralement un microprocesseur combiné avec d'autres matériels et logiciel pour résoudre un problème de calcul spécifique.

Système électronique et informatique autonome possédant pas des entrées-sorties standards.

Le système matériel et l'application sont intimement liés et noyés dans le matériel et ne sont pas discernables comme dans un environnement de travail classique de type PC.

Définition (2)

N'est pas visible en tant que tel, mais est intégré dans un équipement doté d'une autre fonction; on dit aussi que le système est enfoui, ce qui traduit plus fidèlement le terme anglais «embedded». Une faible barrière existe entre les systèmes embarqués et les systèmes temps réels (un logiciel embarqué n'a pas forcément de contraintes temps réel). La conception de ces systèmes est fiable (avions, système de freinage ABS) à cause de leur utilisations dans des domaines à fortes contraintes mais également parce que l'accès au logiciel est souvent difficile une fois le système fabriqué.

Définition (3)

Les microprocesseurs s'étendent depuis de simples microcontrôleurs 8 bits aux 64-bit le plus rapidement et les plus sophistiqués. Le logiciel système inclus s'étend d'un petit directeur à un grand logiciel d'exploitation en temps réel (RTOS) avec une interface utilisateur graphique (GUI). Typiquement, le logiciel système inclus doit répondre aux événements d'une manière déterministe et devrait toujours être opérationnel. Les systèmes embarqués couvrent aussi bien les commandes de navigation et de commande de trafic aérien qu'un simple agenda électronique de poche.

Les types de systèmes embarqués

Calcul général : Jeu vidéo.

Contrôle de systèmes en Temps Réel : Système de navigation aérien.

Traitement du signal : Radar, Sonar,

Transmission d'information et commutation : Téléphone, internent.

Quelques exemples

Équipement mobile et bureautiques : Répondeurs, Copieurs, Téléphone portable, Imprimante. Équipement dans le bâtiment : Ascenseurs, escalators, Système de surveillance, Contrôle d'accès, Systèmes d'éclairage.

Équipement de production : Productions automatisées, Systèmes de commande d'énergie, équipements de stockage,

Transport : Avionique, Trains, Automobiles (+ de 100 processeurs), Contrôle de navigation,

Communications : Satellites, GPS, Téléphonie mobile,

Historique (récent) Fin des années 1940: Le processeur Whirlwind du MIT est conçu pour des applications temps réel A l origine pour contrôler un simulateur de vol. Le premier microprocesseur est l Intel 4004 au début des années La calculatrice HP-35 utilise plusieurs circuits pour implémenter un microprocesseur en 1972.

Les automobiles utilisent des systèmes de contrôle du moteur avec microprocesseurs depuis les années Contrôle du mélange fuel/air, gestion du moteur, etc. Modes de fonctionnement multiples: démarrage, croisière, montées, etc. Baisse des émissions polluantes, consommation optimisée.

Caractéristiques d'un système embarqué

(1) **Fonctionnement en Temps Réel : (Réactivité) :** des opérations de calcul doivent être faites en réponse à un événement extérieur (interruption matérielle). La validité d'un résultat dépend du moment où il est délivré, (deadlines). Rater une échéance peut causer une erreur de fonctionnement. La plus part des systèmes sont «multirate» : traitement d'informations à différents rythmes.
(2) **Faible encombrement, poids et consommation :** Consommation électrique minimisée, Difficulté de packaging (analogique, numérique et RF), Batterie de 8 heures et plus (PC portable : 2 heures). Environnement sévère (Température, vibrations, variations d'alimentation, interférences RF, corrosion, eau, feu, radiations, etc.), Le système n'évolue pas dans un environnement contrôlé (évolutions des caractéristiques des composants en fonction de l'environnement).
(3) **Coût, sûreté et sécurité :** Le système doit toujours fonctionner correctement (faible coût et une redondance minimale), Sûreté de fonctionnement du logiciel (système opérationnel même quand un composant électronique «lâche». Beaucoup de systèmes embarqués sont fabriqués en grande série et doivent avoir des prix de revient extrêmement faibles

Les systèmes embarqués et le temps réel

Un système embarqué doit généralement respecter des contraintes temporelles fortes (Hard Real Time). On y trouve enfoui un système d'exploitation ou un noyau Temps Réel (Real Time Operating System, RTOS).

Un système est dit Temps Réel lorsque l'information après acquisition et traitement reste encore pertinente. Cela veut dire que dans le cas d'une information arrivant de façon périodique (interruption), les temps d'acquisition et de traitement doivent rester inférieurs à la période de rafraîchissement de cette information. Ne pas mélanger Temps Réel et rapidité de calcul du système donc puissance du processeur

Le fonctionnement temps réel

Les opérations doivent être faites avec des échéances (deadlines) précises**.**

Hard real time : le manquement des échéances provoque une faute

Soft real time : le manquement des échéances cause des dégradations de performances.

La plupart des systèmes sont multi-rate : Les opérations doivent être gérées à des vitesses (très) différentes.

Spécifications non fonctionnelles

Beaucoup de systèmes sont des systèmes à production de masse qui doivent avoir des coûts de fabrication bas, Mémoire limitée, puissance du microprocesseur, etc. La consommation est un facteur critique pour les systèmes fonctionnant sur piles (ou batteries). Une consommation excessive augmente le coût même en cas d'alimentation par le secteur.

Lien entre le matériel et le logiciel «le codesign»

Objectif : intégrer un système dans un même composant (single chip). On parle aussi de système sur silicium SoC (System on Chip) ou SoPC (System on Programmable Chip), (loi empirique de Moore).

Codesign hardware / software

Le codesign dans la méthodologie de conception d'un système embarqué est de plus en plus utilisé. Le codesign permet de concevoir en même temps à la fois le matériel et le logiciel pour une fonctionnalité à implémenter. Cela est maintenant possible avec les niveaux d'intégration offerts dans les circuits logiques programmables. Le codesign permet de repousser le plus loin possible dans la conception du système les choix matériels à faire contrairement à l'approche classique où les choix matériels sont faits en premier lieu.

Les étapes du Codesign

Spécifications : liste des fonctionnalités du système de façon abstraite.

Modélisation : conceptualisation et affinement des spécifications produisant un modèle du matériel et du logiciel.

Partitionnement : partage logiciel matériel. Synthèse et optimisation compilation logicielle.

Validation : co-simulation**.**

Synthèse matérielle et Intégration : rassemblement des différents modules**.**

Tests d'intégration : vérification du fonctionnement.

Avantages du codesign

- **Amélioration des performances :** parallélisme, algorithmes distribués, architecture spécialisée, etc**.**
- **Reconfiguration statique ou dynamique en cours de fonctionnement.**
- **Indépendance vis à vis des évolutions technologiques des circuits logiques programmables.**
- **Mise à profit des améliorations des outils de conception fournis par les fabricants de circuits logiques.**

2. Langages de description d'architecture (AADL, systemC)

2.1. AADL :

Le langage AADL (Architecture analysis & design langage) pour analyse de l'architecture et langage de conception : est un langage émergeant développé sous l'autorité de la SAE (Society of Automotive Engineers) pour répondre aux besoins spéciaux des systèmes embarqués temps réel critiques tels que les systèmes avioniques.

Elle est conçue pour permettre la génération de systèmes exécutables et l'expression des caractéristiques des éléments du système ainsi que sa traduction en langage de programmation. Elle est dédiée spécialement pour la description des éléments matériels et logiciels des systèmes embarqués sous forme de plusieurs représentations :

• texte

• XML

• notation graphique

• profil UML 2, et représentation en UML 1.4.

La description d'une architecture en AADL consiste essentiellement à la représentation d'une architecture dynamique du logiciel associée et liée à une plateforme d'exécution.

En pratique, elle se décrit à l'aide de composants hiérarchisés (ils peuvent contenir d'autres composants) et interconnectés. Les composants communiquant entre eux par des liens fonctionnels (flots de données et de contrôle) et par des liens physiques (bus, mémoire, réseau,…).

Des informations telles que les exigences temporelles, les modèles de fautes et d'erreurs, le partitionnement temporel et spatial ou les besoins de sûreté et de certification peuvent être modélisés et associés aux composants à l'aide de propriétés.

a. Les concepts AADL:

-Concept de composant.

Les composants sont les éléments de base d'une architecture AADL. Ils sont hiérarchisés, composés et interconnectés.

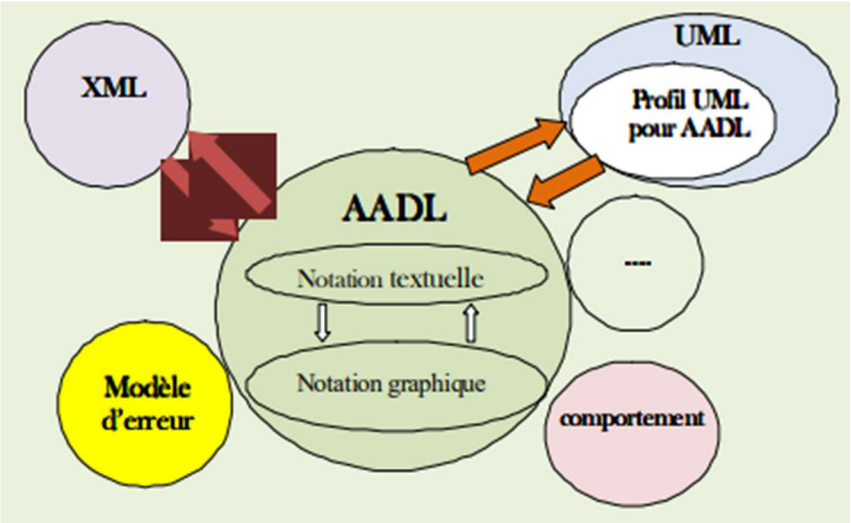
Ils appartiennent à l'une des catégories prédéfinies :

-catégorie composite

-catégories logicielles

-catégories plate-forme

Catégories de composants. dans laquelle on trouve : Les threads ; les processus, les sous programmes ; les bus ; les mémoires ; les groupes de threads ; les systèmes, les données ; les périphériques et les processeurs.



Flot de données UML / AADL

On peut trouver aussi des Catégories prédéfinies dans lesquelles on doit spécifier : Le Type et la spécification de l'interface externe l'Implémentation.

La spécification du contenu se fait par une instance. L'instanciation d'un type ou d'une implémentation se fait par : type ; implémentation et instance. Un type peut hériter d'un autre type et une implémentation peut hériter d'une autre implémentation.

-**Concept de caractéristique :**

Une « caractéristique » (feature) est un élément de type AADL de composant qui spécifie comment ce composant s'interface avec les autres composants du système.

Il y a quatre catégories de caractéristiques :

- Les ports ;
- Les accès à un sous-composant ;
- Les sous-programmes ;

- Les paramètres.

Port :

Un « port » représente une interface de communication pour échanger des données ou des événements entre composants.

Trois directions sont possibles :

- port entrant (in) ;
- port sortant (out) ;
- port bidirectionnel (in out).

Trois types de ports :

- port donnée (data port) ;
- port événement (event port) ;
- port événement-donnée (event data port).

-**Concept de propriété.**

Une « propriété » (property) fournit une information sur un composant, une caractéristique, une connexion, un mode ou un appel de sous-programmes. Une propriété est définie par : un nom ; un type et une valeur.

Le type de propriété définit l'ensemble des valeurs acceptables. On peut définir de nouvelles propriétés dans un property set. Les Propriétés représentent des ensembles de propriétés pré déclarés. Deux ensembles de propriétés sont pré déclarés :

- AADL_Properties définit des propriétés communes à toutes les spécifications AADL;
- AADL_Project définit des types et des constantes énumérés, dont les valeurs peuvent différer d'un projet à l'autre.

Ces deux ensembles de propriétés font implicitement partie de toute spécification AADL.

Les types et constantes de AADL_Project peuvent être modifiées.

-**Concept de Flux.**

Pouvoir spécifier des flux de bout en bout permet des analyses temporelles, de fiabilité, de propagation d'erreur, de qualité de service, etc.

La description de flux complets est composée de plusieurs sortes de déclarations :

- Spécification de flux ;
- Implémentation de flux ;
- Déclaration de bout en bout.

-**Concept de Paquet.**

Un paquet (package) fournit le moyen d'organiser les descriptions en introduisant des espaces de noms. Un paquet peut contenir des types de composants ; des implémentations de composants ou des librairies annexes.

Le contenu de la section public est visible hors du paquet et le contenu de la section private n'est pas visible hors du paquet.

- **Les Annexes.**

Une annexe permet d'utiliser des déclarations exprimées dans un autre langage qu'AADL.

L'usage principal est le support de nouvelles méthodes d'analyse ou de description de nouveaux aspects. Une clause annexe peut être utilisée dans un type ou une implémentation de composants, par contre une bibliothèque d'annexes peut être déclarée dans un paquet.

b. AADL comme profil UML :

AADL comme profil UML se compose d'un ensemble de (voir tableau):

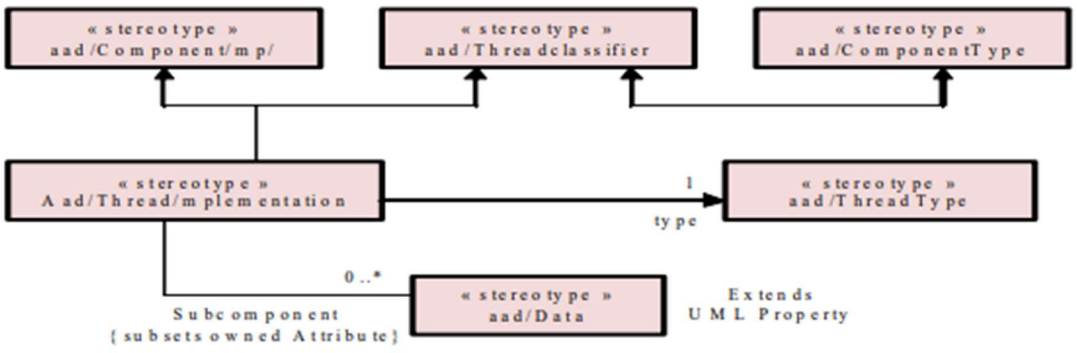
-différents types de stéréotypes : stéréotypes abstraits (entités conceptuelles : espace de nommage, type et implémentations de composants, features, etc.) et stéréotypes concrets (Entités AADL, implémentation et type système, port de données connections de données…).

-Contraintes : qui assurent la consistance et la cohérence et limitent les options de composition

-Propriétés relatives aux stéréotypes : ce sont les propriétés standard AADL.

AADL Concept	UML Profile
Component Type	Stereotype aadl/Thread Type,... Extend UML Class
Feature	Stereotype aadl/dataPort,... Extend UML Property
Component Implementation	Stereotype aadl/Thread/mplementation,... Extend UML Class
Subcomponent	Stereotype aadl/Thread ,... Extend UML Property
Package, Property Set	Stereotype aadl/Package, aadl/PropertySet Extend UML Package
Connection	Stereotype aadl/dataConnection,... Extend UML Association

Eléments du profil UML



Les contraintes du profil UML

Translation DSL – profil UML :

Un pont entre les 2 approches doit passer par une translation entre les modèles AADL conformément au méta-modèle AADL méta-model et par des modèles AADL conformément à AADL comme profil UML.

Il Permet notamment l'utilisation d'outils matures d'UML pour le développent de modèles, et l'utilisation des techniques spécifiques d'analyse d'AADL basées sur le méta-modèle AADL.

Il a comme objectifs principaux d'activer rapidement un outil commercial qui supporte AADL et d'accélérer la transition d'AADL dans l'industrie.

Le Processus de transformation suit les étapes suivantes :

1. créer le modèle AADL en utilisant un outil UML

2. traduire au méta modèle AADL

3. exécuter des analyses (exemple dans OSATE)

4. traduire au profil UML

5. Raffiner le modèle UML

2. SystemC

2.1 A PROPOS DE SYSTEMC

SystemC est, comme Verilog et VHDL, un langage de description de matériel. C'est le fruit de contributions de plusieurs sociétés. En 1989, Synopsys met son outil commercial Scenic dans le domaine libre, et crée la version 0.9 de SystemC. Une première contribution de Frontier Design donne lieu à la version 1.0, et une autre de CoWare aboutit en 2000 à la version 1.1, première version officielle de SystemC. L'OSCI (Open SystemC Initiative) est alors crée, rassemblant une multitude de sociétés et laboratoires de recherche. Cette organisation a fusionné en 2011 avec Accellera, une association de normalisation de standards pour le design des systèmes électroniques. C'est maintenant Accellera Systems Initiative qui est en charge de diffuser, promouvoir et rédiger les spécifications de SystemC.

Les spécifications de SystemC ont été étendues en 2001 à la modélisation de systèmes abstraits (de très haut niveau, avant partitionnement matériel / logiciel), aboutissant à la version 2.0. En 2005, l'IEEE a approuvé une version de référence de SystemC, appelée IEEE1666-2005 et correspondant à la version 2.2.0 de SystemC.

En 2011 une révision du standard est approuvée, elle intègre entre autre la modélisation transactionnelle de TLM-2.0. Elle est appelée IEEE1666-2011 et est téléchargeable directement sur le site de l'IEEE.

Ce cours n'aborde pas en détail la modélisation transactionnelle.Les concepts abordés s'appliquent ainsi aux versions 2.2.0 et 2.3.0 de SystemC.

Bien que libre, la bibliothèque SystemC ne peut être téléchargée que depuis le site d'Accelera. Elle a été installée sur les machines du département COMELEC, dans le répertoire /comelec/softs/opt/systemc/systemc-2.3.0 et a été compilée pour les architectures x86 64bits sur Linux. Nous verrons plus loin comment l'utiliser.

Objectifs Et Utilité De Systemc

SystemC a pour objectif de modéliser des systèmes numériques matériels et logiciels à l'aide de C++. Il permet donc de modéliser non seulement des systèmes matériels, mais aussi des systèmes logiciels, mixtes ou non-partitionnés (où on ne sait pas encore ce qui sera fait en logiciel, et ce qui sera fait en matériel).

La modélisation d'un système complet passe par des niveaux comportementaux plus détaillés que ceux abordés précédemment.

La description de ces niveaux se fait à l'aide des termes suivants :

- BCA (Bus Cycle Accurate) : s'applique à l'interface d'un modèle.

Il signifie que la modélisation des transactions sur l'interface sont correctes au cycle près.

Un modèle BCA n'apporte aucune information sur les bits (signaux) de l'interface.

- BA (Bit Accurate) : s'applique à l'interface d'un modèle et à la fonctionnalité d'un modèle.

Il signifie que la modélisation des transactions sur l'interface est BCA, et porte aussi sur les signaux de l'interface. La modélisation est précise au bit (fil) près.

Le terme PCA est aussi appelé CABA (cycle accurate / bit accurate).

- UTF (UnTimed Functional) : s'applique à l'interface et à la fonctionnalité d'un modèle.

Le modèle ne comporte aucune notion de durée d'exécution, mais seulement un ordre éventuel dans l'exécution des événements.

Chaque événement s'exécute en un temps nul. Seul compte l'ordonnancement des événements.

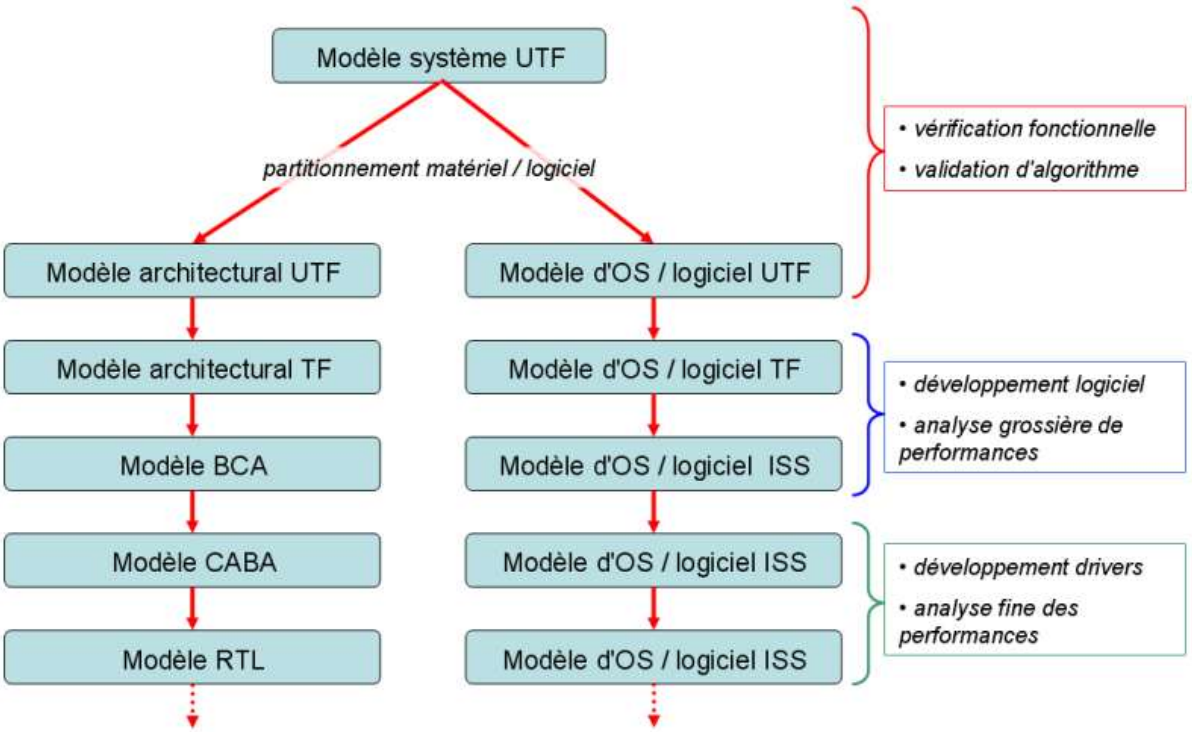
- TF (Timed Functional) : s'applique à l'interface et à la fonctionnalité d'un modèle.

Le modèle comporte des notion de durée (temps d'exécution des processus, latence, temps de propagation, ...)

- RTL (Register Transfert Level) : s'applique à l'interface et à la fonctionnalité d'un modèle matériel.

Chaque bit, chaque cycle, chaque registre du système est modélisé.

Le processus de modélisation d'un SoC se décompose alors ainsi :



Développement d'un système : raffinement des modèles matériels / logiciels

L'un des inconvénients majeurs des flots habituels vient de la multitude des langages mis en oeuvre :

- les modèles de haut niveaux (algorithmiques, UTF) sont le plus souvent écrits en C / C++ / Matlab
- puis le partitionnement matériel / logiciel est effectué, le plus souvent en C / C++ (ou dans un langage spécialisé)
- un modèle matériel transactionnel (TF) est alors produit, qui sert de modèle de référence.

C'est un exécutable modélisant le comportement du système matériel. Il est écrit le plus souvent en C / C++ ou dans un langage propriétaire.
Le développement du logiciel se fait à partir de ce modèle-là.

- ce modèle est alors manuellement transcrit dans HDL (Verilog / VHDL),
- puis raffiné jusqu'à la synthèse. Ce modèle (HDL) est alors le centre du développement, et devient, en pratique, la référence.

Les changements de comportement du modèle HDL ne sont souvent pas reportés dans le modèle C/C++.

- une fois le module matériel arrivé au stade CABA, un autre modèle C / C++ de référence est produit (manuellement) incorporant les éventuelles modifications qui ont eu lieu tout au long du cycle de développement.
- à chaque étape, les environnements de test doivent eux aussi être transcodés.

Toutes ces transcriptions manuelles introduisent des erreurs. L'objectif principal de SystemC est de maintenir un seul et même langage d'un bout à l'autre du flot de conception.

STRUCTURE D'UN MODÈLE SYSTEM

Les différents composants structurels de SystemC seront étudiés en détail plus tard. Mais voici d'ores et déjà un aperçu rapide de la façon dont est construit un modèle SystemC d'un système numérique.

Hiérarchie

Comme en Verilog ou VHDL, un système est une hiérarchie d'objets. Généralement de sont des modules imbriqués et/ou des processus. Les modules communiquent entre eux par des canaux.

Le plus haut de la hiérarchie d'un système complet (module à tester + testbench) n'est pas un module top (comme on en a l'habitude en Verilog/VHDL), mais la fonction SystemC sc_main (l'équivalent du main des programmes en C).

Modules

Un ***module*** en SystemC est composé d'autres modules, de canaux de communication entre ces modules (signaux, ou canaux plus abstraits), et éventuellement de processus.

Ports : 

Un module possède un ou plusieurs ***ports***. Les ports sont juste des points d'entrée ou de sortie, qui ne font rien de particulier.
Par contre, les ports doivent déclarer les fonctions qui seront utilisées pour communiquer à travers eux. Exemples :

- un port en entrée destiné à être relié à un signal normal déclarera qu'il utilise la fonction read des signaux.
- un port similaire mais bidirectionnel déclarera qu'il utilisera les fonctions read et write
- un port destiné à être relié à une fifo (un canal de communication abstrait, de haut niveau), déclarera selon le côté de la fifo où il st sensé se trouver, qu'il utilisera les fonction read, nb_read (read non bloquant), num_available, write, nb_write, num_free...

La déclaration des fonctions qu'il va utiliser est appelée ***interface***.

Interfaces : 

Une interface est une déclaration des fonctions (ou "méthodes", pour prendre la terminologie C++) qui pourront être utilisées à travers les ports d'un module. Une interface ne contient pas de code, c'est seulement une ***déclaration de fonctions***.
Les interfaces permettent au compilateur de détecter très tôt le branchement d'un port à un canal qui ne lui est pas adapté.

Canaux : 

Les canaux sont les moyens de communication entre les modules. Ils peuvent être basique et concrets (signaux), ou plus évolués / plus abstraits (fifo, réseau ethernet, ...). Ils peuvent aussi contenir d'autres canaux, voire même des modules si ce sont des canaux de très haut niveau.

Les canaux contiennent (entre autres) l'implémentation du code C++ déclaré dans les interfaces. On dit qu'ils ***implémentent*** une interface.

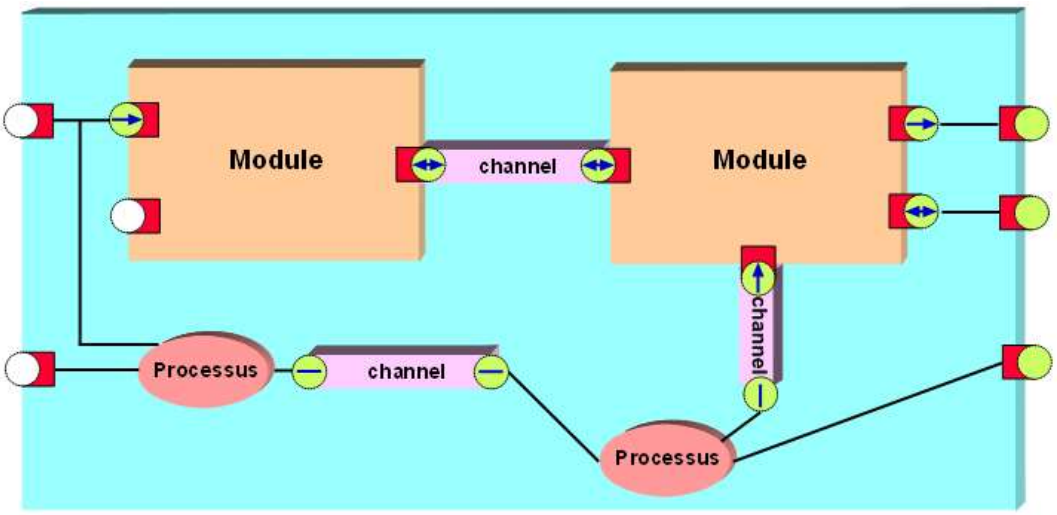
On branche un canal à un module par l'intermédiaire d'un port, ce port devant déclarer l'interface implémentée par le canal. L'intérieur du module peut alors accéder au canal en appelant les fonctions déclarées dans l'interface. De façon imagée, les ports publient une offre d'embauche (l'interface) et les canaux aptes à y répondre font le boulot. L'interface est donc ce qui lie les ports et les canaux en décrivant comment l'information transite par les ports. Ce qui est symbolisé dans le schéma ci-dessous par l'image  (qui combine port et interface). Tout ce dont a besoin un module pour pouvoir utiliser un port est de connaître l'interface qu'il publie. Autrement dit, on peut modifier à loisir le contenu d'un canal, le raffiner progressivement, sans avoir à toucher quoi que ce soit d'autre. Il suffit que son interface reste la même.

Processus :

Les processus en SystemC sont similaire à ceux de Verilog et VHDL. Ils décrivent une fonctionnalité, un comportement. Un processus ne doit pas être appelé directement; c'est le moteur de simulation SystemC qui se charge de l'appeler (le déclencher) sur certains événements particuliers : ceux qui sont dans sa liste des sensibilité.

Les processus peuvent éventuellement communiquer directement avec les canaux. Ils n'ont pas besoin de port pour cela, ils appellent directement les méthodes du canal.

En résumé, l'architecture d'un module en SystemC ressemble à ceci :



Organisation structurelle en SystemC

EN RÉSUMÉ

SystemC est une bibliothèque C++, qui fournit des classes C++ adaptées à la modélisation de matériel ainsi qu'un moteur de simulation événementiel rapide.

Il permet de garder un même langage d'un bout à l'autre du flot de conception, excepté peut-être à la toute fin (mais on a alors des moyens de s'assurer de la cohérence des modèles).

Un modèle SystemC est écrit en C++, c'est généralement un module.

Les modules sont les briques structurelles de base. Ils instancient d'autres modules, ainsi que des canaux et des processus. Ils communiquent avec l'extérieur par des ports. La communication proprement dite se fait au travers de canaux reliés au port.

Pour qu'un canal soit relié à un port, les deux doivent avoir la même interface (déclaration de fonctions). L'implémentation de ces fonctions est faite dans le canal.