From high level MPSoC description to SystemC code generation

Rabie Ben Atitallah, Eric Piel, Julien Taillard, Smail Niar, Jean Luc Dekeyser

INRIA-FUTURS, DaRT project – Parc Scientifique de la Haute Borne – 40 avenue Halley – 59650 Villeneuve d'Ascq – FRANCE

ABSTRACT

In this paper, we present an efficient Multi-Processor Systems-on-Chip (MPSoC) design flow. It is based on a Model-Driven Engineering (MDE) approach. A compilation chain has been developed to transform the high abstraction level into both Cycle Accurate Bit Accurate (CABA) and Timed Programmer View PVT SystemC simulation. We use the standard MARTE profile to represent MPSoC systems. This representation separates the application, the hardware architecture and the corresponding allocation. Later, through several model to model transformations, we succeed to generate SystemC code of the modeled MPSoC system.

KEYWORDS: MPSoC, MDE, Code generation, SystemC simulation

1   Introduction

MPSoC architecture has become an unavoidable part of designing embedded systems dedicated to applications that require intensive parallel computations. The most important design challenge in such systems consist in solving the huge architectural space solution and evaluating the corresponding alternatives. MPSoC systems need new development methodology to reduce the complexity of design space exploration and to increase the engineers productivity.

In this scopen, we propose a new design flow dedicated to MPSoC based on *Model-Driven Engineering* [Pla07] (MDE). This methodology is centered around two concepts: model and transformation. Data and their structures are represented in models, while the computation is done by transformations. Models contain information structured according to the metamodel they conform to. In our framework, models are used to

Figure 1: The part of our compilation chain for SystemC generation. The five top models are aggregated to form the MPSoC model.

represent the system (application, architecture, and allocation). Transformations are employed to move from an abstract model to a detailed model. The set of transformations forms the compilation chain. In our case, this chain converts the platform-independent MPSoC model into a platform-dependent. In our case you obtained a SystemC simulation code. Nevertheless, any other HDL (such as verilog or VHDL) can be easily supported. The Fig. 1 depicts the proposed design flow, next sections will detail the proposed transformation from the high level description language to SystemC (smail) description code.
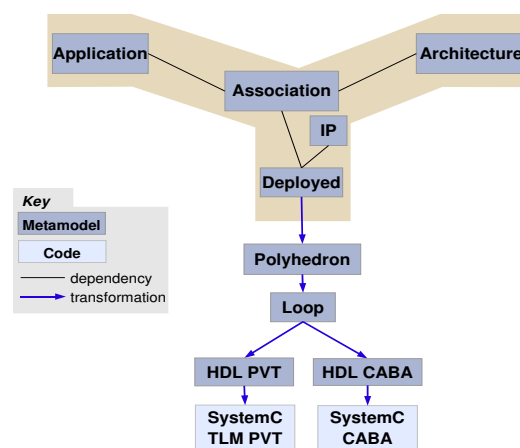
---

[1]E-mail: {benatita,piel,taillard,niar,dekeyser}@lifl.fr

2 MARTE: a profile for embedded systems modeling

MARTE (Modeling and Analysis of Real-Time and Embedded systems) [RSG[+]05] is a standard proposal of the Object Management Group (OMG). The primary aim of MARTE is to add capabilities to the Unified Modeling Language (UML) for real-time and embedded systems concepts modeling. UML provides the framework into which the needed concepts are plugged. The MARTE profile enhances possibility to model software, hardware and relations between them. It also provides extensions to make performance and scheduling analysis and to take into account platform services.

A strong advantage of using the MARTE profile for MPSoC modeling is the particular concept of *factorization*, both for hardware architecture and application. With the semantic introduced by the ARRAY-OL model of computation [Bou07], factorization provides a mechanism that expresses the parallelism of the system in a compact way. As shown in Fig. 2, the multiplicity syntax specifies repetitions in the hardware architecture (e.g., `ProcessingUnits[(4)]`) as well as in the application (e.g., `Dct[(11,9)]`). In the same way, *allocation* syntax expresses the distribution of tasks over the processing units.
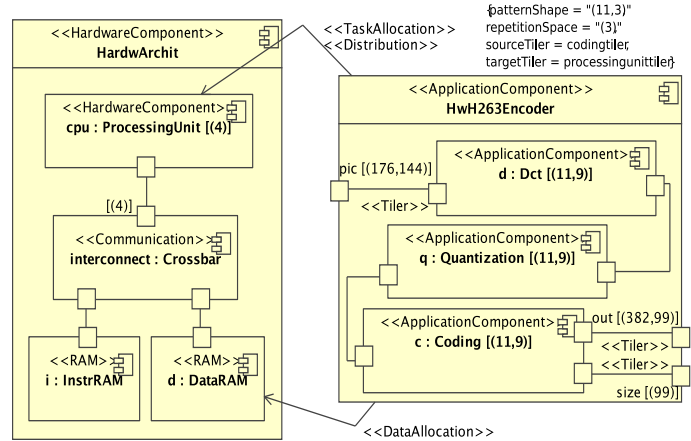
Figure 2: Global view of the H.263 application on a 4 processors MPSoC.

3 Deployment Profile

To transform the high abstraction level models into simulation code, very detailed deployment information must be provided. In particular, each elementary component must be linked to an existing code. For this purpose, a *deployment* profile is introduced. A key point in our methodology is to facilitate Intellectual Property block (IP) reuse. Therefore great care was taken to allow usage of IP libraries and to keep the MPSoC model independent from the compilation target. In this profile, we introduce the concept of *AbstractImplementation* which expresses hardware or software functionality, independently of the compilation target. It contains one or several *Implementations*, each one being used to define a specific implementation at a given simulation level and programming language. Fig. 3 shows an example of an *AbstractImplementation* of a MIPS processor which contains two *Implementations* at the CABA and PVT levels written in SystemC.

Using the *ImplementedBy* syntax, the designer can select the adequate IP for each hardware and software component. The *Implementation* which fits best the target will be used to reify the component during the compilation phase. This automatic selection allows to generate the (Smail enlever exact) same SoC model at different simulation levels. For automatic code generation, we use the concept of *CodeFile* to specify the code and compilation options required.
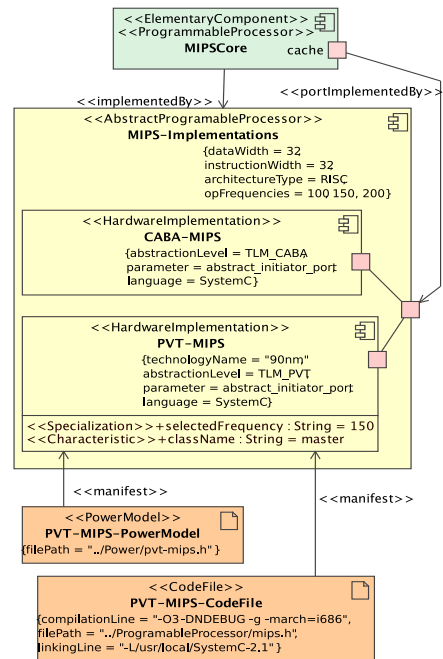
Figure 3: Deployment example of a MIPS processor.

Specializations and *Characteristics* can be added by the designer in order to pass information respectively to the generated code and to the transformation.

Using this methodology, we were able to target the PVT and CABA levels in SystemC from the same design. System description at these two levels is detailed in [ANMD07]. Based on SystemC version 2.1 and the TLM library, hardware components were specified. Similarly, software components were specified. Several implementations can also be provided for each software components, for instance this allows to target functional simulation levels or to generate an hardware accelerator.

4 Polyhedron Model

From the distribution information given in the MPSoC model, a polyhedron is automatically generated for each task repetition which is placed on processors. Polyhedrons are parameterized by a processor number ($p0$). It allows to have a compact polyhedron for each processor. From the distribution modeled in Fig. 2, the following polyhedron (Fig. 4) is generated. The `p` variables are the processor indexes (used as parameter), the `x` variables are the task indexes.

When the application is distributed over different processors, that is processors explicitly represented by different components in the hardware model, an additional rule is executed. The appli-

$$\begin{cases} p0 \leq 0, 3 - p0 \leq 0 \\ -4 * mh0 - p0 + 1 * q0 + 1 * q1 + 0 * d0 = 0 \\ -16 * ms0 - x0 + 2 * q0 + 0 * q1 + 1 * d0 = 0 \\ -16 * ms1 - x1 + 0 * q0 + 2 * q1 + 0 * d0 = 0 \\ q0 \leq 0, 7 - q0 \leq 0 \\ q1 \leq 0, 7 - q1 \leq 0 \\ d0 \leq 0, 1 - d0 \leq 0 \\ d1 \leq 0, 1 - d1 \leq 0 \\ x0 \leq 0, 15 - x0 \leq 0 \\ x1 \leq 0, 15 - x1 \leq 0 \end{cases}$$

Figure 4: Generated integer polyhedron formula for the example distribution.

cation can be seen as a tree, where the main component is the root, each sub-component is a branch, and the elementary components are the leaves of the tree. In this rule, the tree is cut and moved onto the different processor sets, so that each leaf is placed on the processor the designer had allocated it. The branches which have some leaves placed on a processor and other leaves placed on an other processor are duplicated and copied on both processors. This transformation permits to generate one code per group of processors in case the system is heterogeneous.

5 Loop Model

To exploit this polyhedron into our generated code, control loops have to be generated. Scanning polyhedron is a classical problem, existing tools already exist. CLooG [Bas04] (Chunky Loop Generator), written by Cedric Bastoul, handles this problem. Loops scanning the polyhedron are generated by CLooG. The MDE transformation is responsible for selecting each polyhedron, passing it to CLooG, interpreting the result and generating models of loops from those results. From the polyhedron (Fig. 4), the loops illustrated in Fig. 5 are generated.

```
for (d1=0; d<1; d++){
 for (q1=0; q1<7; q1++){
  for (x1=0; x1<7; x1++){
   for(d0=MAX(0,-14), d0<MIN(1,15), d0++){
    q0 = -q1+4*mh0+p0
    If (MOD(d1+2*q1-x1,16) == 0){
      ms1 = (d1+2*q1-x1)/16
      x0 = -2*q1+d0+8*mh0+2*p0
      ! Call to the task
    }
   }
  }
 }
}
```

Figure 5: Control loop generated by CLooG.

6 SystemC Code Generation

From the Loop model, the SystemC code generation phase is started. For the hardware part, this step consists of creating the hierarchical structure of the system, instantiating the IP block (proces-

```
MainArchitecture::MainArchitecture( sc_module_name module_name, int index):
sc_module (module_name)
{
//Component Instanciation
actuator_pointer = new actuator("actuator");
sensor_pointer = new sensor("sensor");
MultiBankMemory_pointer = new MultiBankMemory(« mbm", index);
simple_bus_pointer = new simple_bus<9,9>("simple_bus");
MultiProcessingUnit_pointer = new MultiProcessingUnit("MultiProcessingUnit", index);

//Connector Instanciation
for(int j=0; j<8;j++){ Reshape_91a829[j]= new channel_type("reshape");}
for(int j=0; j<8;j++){Reshape_b73194[j]= new channel_type("reshape");}
for(int j=0; j<1;j++){Reshape_1180cbd[j]= new channel_type("reshape");}
for(int j=0; j<1;j++){Reshape_1ec6a9a[j]= new channel_type("reshape");}

//Connector Description and connection
//Reshape processing
int reptitionSpaceI[1];
int arrayInI[1];
int repetitionSpace[1] = {1} ;
int arrayIn[1] = {9} ;
int arrayOut[0] = {} ;
int originMatrix_In[1]={8};
int pavingMatrix_In[1][1] = {{1}};
int originMatrix_Ou[1]={0};

for(reptitionSpaceI[0]=0; reptitionSpaceI[0]<repetitionSpace[0];reptitionSpaceI++){
  for(int i = 0 ; i < 1 ; i++) {
    // Origin
    arrayInI_Reshape[i] = originMatrix_In[i];
     // Paving
    for(int j = 0 ; j < 1 ; j++)
    arrayInI[i] += reptitionSpaceI[j] * pavingMatrix_In[i][j];
  }
simple_bus_pointer->slave_port[arrayInI[0]]->bind(*actuator_pointer->target_port );
}
```

Figure 6: Example of generated SsytemC code

sors, caches, interconnection networks...) from the appropriate hardware libraries and connecting components together. The objects used between the communicating modules depend on the target abstraction level (Fig. 6). For instance, at the PVT level, transactions are performed through channels [Gro03] instead of signals as used at the CABA level. The concept of *Specializations* introduced by the *deployment* profile is used to add timing information to the IP block. Thus, allowing performance estimation. For example, for the memory module, read and write time accesses are specified. Moreover, in order to generate simulation code with precise energy estimation, a hardware component can be associated to an energy consumption model.

For the application part, the code generation step consists of synchronizing tasks over processors, computing data addresses and calling tasks (FFT, FIR, DCT...) from the appropriate software libraries. When an Instruction Set Simulator (ISS) is used for processors simulation, tasks are compiled for the target processor. After generating the SystemC code which represents our MPSoC system, the simulation is executed in order to obtain performance and energy estimation. Thus a decision can be taken about the adequacy between architecture, application and the allocation of tasks.

7 Conclusion

We have presented an MPSoC design flow based on Model-Driven Engineering approach. The proposed approach allows SystemC code generation at the CABA and PVT levels from the same high level description. In particular, MDE transformations help to factorize the work for several simulation levels and also has the advantage of being very flexible to adapt the compilation to new technologies such as VHDL implementation on FPGA or synchronous language code generation. Future work in this domain will encompass interoperability between different simulation levels, allowing simula-

tion at a given level even if not all the IPs have an implementation at this simulation level. We will also be working on the automatization of Design Space Exploration so that simulation results can control the compilation chain target.

References

[ANMD07] Rabie Ben Atitallah, Smail Niar, Samy Meftali, and Jean-Luc Dekeyser. An MPSoC performance estimation framework using transaction level modeling. In *The 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, Daegu, Korea, August 2007.

[Bas04] C. Bastoul. Code generation in the polyhedral model is easier than you think. In *PACT'13 IEEE International Conference on Parallel Architecture and Compilation Techniques*, pages 7–16, Juan-les-Pins, september 2004.

[Bou07] Pierre Boulet. Array-OL revisited, multidimensional intensive signal processing specification. Technical Report RR-6113, February 2007.

[Gro03] Thorsten Groetker et al. *System Design with SystemC*. Kluwer, 2003.

[Pla07] Planet MDE. Model Driven Engineering, 2007. `http://planetmde.org`.

[RSG+05] L. Rioux, T. Saunier, S. Gerard, A. Radermacher, R. de Simone, T. Gautier, Y. Sorel, J. Forget, J.-L. Dekeyser, A. Cuccuru, C. Dumoulin, and C. Andre. MARTE: A new profile RFP for the modeling and analysis of real-time embedded systems. In *UML-SoC'05, DAC 2005 Workshop UML for SoC Design*, Anaheim, CA, June 2005.