Model based design flow for implementing an Anti-Collision Radar detection system

Imran Rafiq Quadri, Yassin El Hillali, Samy Meftali, Jean-Luc Dekeyser

Model based design flow for implementing an Anti-Collision Radar detection system

Imran Rafiq Quadri*, Yassin ElHillali†, Samy Meftali* and Jean-Luc Dekeyser*
*INRIA Lille Nord Europe - LIFL - USTL - CNRS, 40 avenue Halley, 59650 Villeneuve d'Ascq, FRANCE
Email:{Firstname.Lastname}@inria.fr
†IEMN-DOAE UVHC Le Mont Houy 59313 Valenciennes, FRANCE
Email:Yassin.ElHillali@univ-valenciennes.fr

I. Introduction

In order to ensure and increase the safety and reliability of transport systems, these systems are becoming more and more intelligent. They integrate more and more sensors and communication systems. Each of these functionalities can be implemented on a System-on-Chip (SoC). These functionalities are carried out by massive computations. As the number of integrated functionalities increase in the transport systems, the design and implementation complexity also increases at a tremendous rate. Implementation of these functionalities can be carried out either via FPGAs or DSP (digital signal processors) platforms. FPGAs are considered the ideal choice as they accelerate the computations by executing the algorithms in a parallel manner.

Actually the initial steps in existing design flows are the development of the complex algorithms; and then their manual implementation, which are daunting tasks. Change in the scale of the algorithm requires starting the implementation over from scratch which results in increase design time. Normally these steps are carried out by teams of different domains which could result in compatibility issues.

We propose a high abstraction level design methodology for implementation of these algorithms in a graphical manner. The advantages offered by our approach aim to reduce time to make up or time to market. Changes in the nature of the algorithm can be easily carried out due to their graphical nature and the code can be generated automatically rapidly. Afterwards the implementation can be carried out on the target FPGA platforms.

Model Driven Engineering (MDE) is an emerging domain and can be seen as a High Level Design Flow in order to resolve the issues related to SoC co-design. MDE enables system level (application/architecture) modeling at a high specification level allowing several abstraction stages (i.e. IRs). Thus a system can be viewed globally or from a specific point of view of the system, allowing to separate the system model into parts according to relations between system concepts defined at different abstraction stages. This Separation of Views (SoV) allows a designer to focus on a domain aspect related to an abstraction stage thus permitting a transition from solution space to problem space. Using a graphical modeling language i.e. UML (Unified Modeling Language) for system description increases the system comprehensibility. This allows designers to provide high-level descriptions of the system that easily illustrate the internal concepts (task/data parallelism, data dependencies and hierarchy). These specifications can be reused, modified or extended due to their graphical nature. Finally MDE's *model transformations* allow to generate executable models (or executable code) from high level models bridging the gap between these models and execution platforms.

Gaspard [1],[2] as shown in Figure.1 is an MDE-based SoC Co-design framework dedicated to parallel hardware and software. It is based on the UML MARTE profile [3] that allows to model real-time and embedded systems; and allows to move from high level MARTE specifications to different execution platforms such as RTL synthesis in FPGAs [4]. Gaspard exploits the inherent *parallelism* included in repetitive constructions of hardware elements or regular constructions such as application loops. The applications targeted by Gaspard also focus on a specific application domain, that of intensive data-parallel applications.

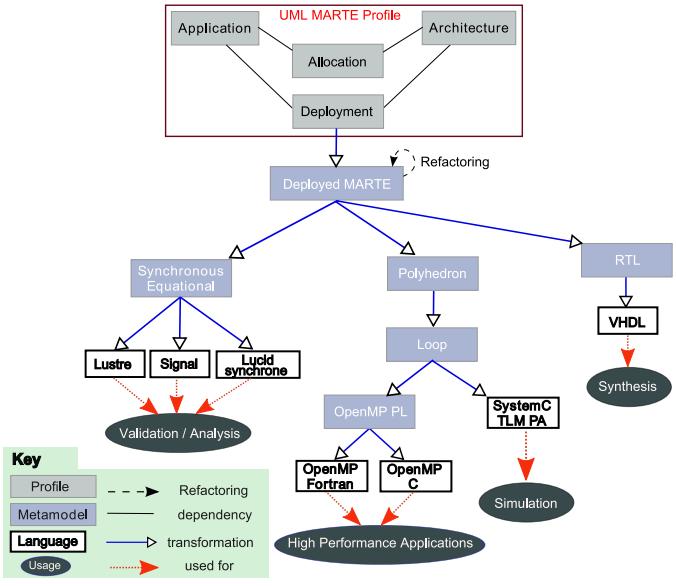


Figure.1: A global view of the Gaspard framework

In this paper we present the modeling and final implementation of a key integral part of an anti-collision radar detection

system. This part is based on delay estimation using a correlation algorithm. This part is modeled at an high abstraction level using the MARTE profile in the Gaspard framework. Afterwards using the model transformations present in our design flow, we have generated the necessary RTL level code for synthesis on a target FPGA using commercial tools. An important point to observe is that the final code generation from the high level models usually is carried out in a few seconds resulting in a huge save in the overall system conception development time.

II. Case Study: Delay estimation correlation module

Correlation algorithms are among the type of digital processing largely employed in DSP (digital signal processing) based systems. They offer a large applicability range such as linear phase and stability. A correlation algorithm normally takes some input data values and compute an output which is then multiplied by a set of coefficients. Afterwards the result of this multiplication is added together to produce the final output. While a software implementation can be utilized for implementing this functionality, the correlation functionality will be sequentially executed. Where as a hardware implementation allows the correlation functions to be executed in a parallel manner and thus increases the processing speed.

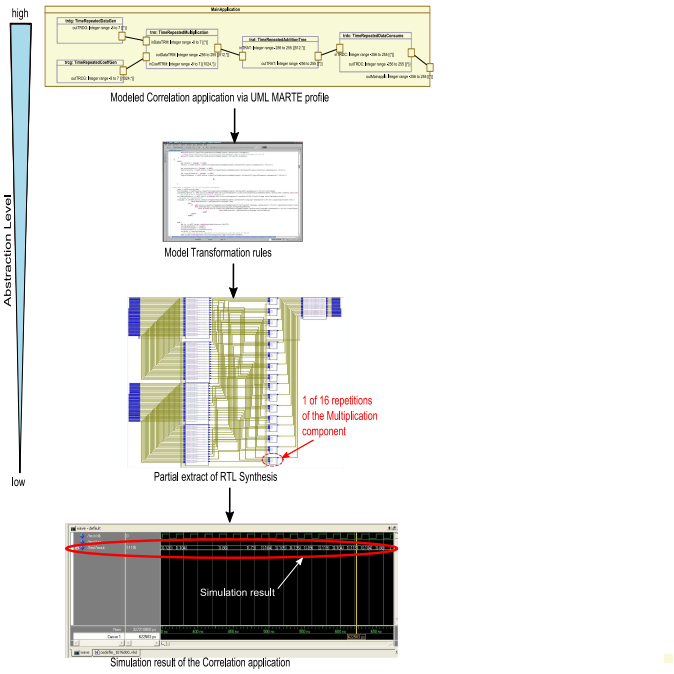


Figure.2: An overviw of our complete design flow

A. Algorithm

We propose to study a case where our radar uses a PRBS (Pseudorandom binary sequence) of length of 127 chips. In order to produce a computation result, the algorithm requires 64 multiplications between the 127 elements of the reference code and the last 127 received samples. The result of this multiplication produces 64 data elements. The sum of these 64 data elements produces the final result. This result can be sent as input to other parts of the Anti-Collision radar detection system.

The mathematical expression of the DECM is described as:

$$C_{cy}(j) = \frac{1}{N}\sum_{i=0}^{N-1} c(i)\cdot y(i+j) \quad (1)$$

Where $c(i)$ represents reference code, $y(i+j)$ the received signal and N the length of the referenced code. In this case stydy, the normalization $\frac{1}{127}$ does not improves the detection quality itself, we consider the following expression:

$$C_{cy}(j) = \sum_{i=0}^{126} c(i)\cdot y(i+j) \quad (2)$$

Figure.2 shows the global overview of our design flow. The figure only shows the top hierarchical level of our correlation application. Once the application has been completely modeled using the UML MARTE profile, the model transformations allow to move from high level models to lower detailed models which add detailed information and concepts related to RTL semantics. Once the final low level RTL model is generated, the *model to text transformations* can be invoked which allow to convert the models into user specified Hardware description language. In our case, we generated the VHDL source code for the modeled application, however, Verilog code can be easily generated as well. The automatic generation of the code allows to convert the application into a hardware functionality, i.e., a hardware accelerator for final FPGA implementation.

The application contains temporal as well as spatial dimensions which can be easily expressed in our design flow. Similarly, *task parallelism* and *data parallelism* can be specified at the high abstraction levels, and the generated HDL code expresses the parallelism specified at the modeling level. The generated HDL code has been synthesized and simulated on Xilinx Virtex II-Pro XC2VP30 as well as Stratix II 2S180 FPGAs in order to validate our design methodology.

1) High level view: Figure.3 shows the top level of our modeled DCEM module. The instance *trm* of the component *TimeRepeatedMultiplication* determines the multiplications while instance *trat* of component *TimeRepeatedAdditionTree* determines the sum.

The instance *trdg* of component *TimeRepeatedDataGen* produces the data values for the received signal while the instance *trcg* of component *TimeRepeatedCoeffGen* produces the reference code. For the received signal, as the port *out_TRDG* of *trdg* has an infinite flow of data; it has a shape specification of {*}. As only the 4 MSB (most significant bits) of the input signal are retained, we utilize the type INTEGER RANGE -8 TO 7 for this signal. The reference code is initially composed of 127 samples, and in order to standardize this input port with a power of 2; we add a 128th element in the reference code. The value of this element is neutral and does not effect the final computation result. The reference code can have different values: from a range of -1 to 1 where 0 allows to encode the

added element. Therefore type of the port *out_TRCG* of *trcg* is INTEGER RANGE -1 TO 1 with a shape of {128,*}. The choice to model the reference code in the form of a flow in time (on the basis of * in its dimension) permits to modify the code during execution of the algorithm. Figure.3 shows the top level view of the filter application.

The output *out_TRAT* port of the instance *trat* also indicates infinite flow as illustrated by its dimension {*}. However, the algorithm allows us to specify that the maximum value of the output (type of the output port of *trat*) will be between -4096 and 4095; hence the primitive type associated is INTEGER RANGE -4096 TO 4095. In order to standardize the system output, another component could be created between *trat* and *trdc* to convert the output into INTEGER. This step has not been taken in this case study. This value is then sent out to the radar.

2) Modeling of the Multiplication step: The modeling of the modeling step is shown in Figure.4 by means of two components. The component *TimeRepeatedMultiplication* expresses the repetition in time while *RepeatedMultiplication* expresses the repetition in space. At the level of *TimeRepeatedMultiplication*, the port *inDataM* of the instance *rm* is composed of an array of 128 data elements and is constructed by a sliding window in time of length 127. This sliding window is expressed via the *Tiler* connected to port *inDataTRM* which expresses the data dependency between the two ports. The component *RepeatedMultiplication* realizes 64 multiplications between the data on the ports *inDataM* and *inCoeffM* by means of the repeated instantiations of the elementary component *Multiplication*.

3) Modeling of the Addition step: Figure.5 represents the component which realizes the addition of 128 data elements. The input port *inAdditionTree* of this tree has a dimension of {128} while the output port *outAdditionTree* is a scalar: shape of {1}.

The addition computation has been decomposed in a tree, with each stage of this tree carrying out partial additions. The dimensions of the ports between each stage in this pipeline of tasks reduce by a factor of 2 ($128 \rightarrow 64 \ldots 2 \rightarrow 1$). Figure.6 represents the first stage, which realizes a partial addition of sixty four elements on an input port and produces thirty two elements on its output port. The computation task *Addition* is repeated thirty two times and is elementary in nature.

In conclusion, we have presented a novel design methodology to model complex intensive data-parallel applications. The modeling is carried out using the UML graphical language and the MARTE standard. Afterwards, automatic code generation can be carried out via MDE tools and technologies. Finally the code can be synthesized and implemented on a target FPGA.

B. Simulation Results

Figure.7 and Figure.8 shows the simulation results of our DCEM.

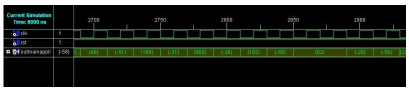


Figure.7: *Peak 1*

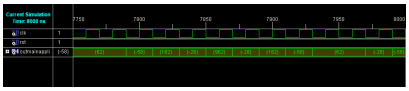


Figure.8: *Peak 2*

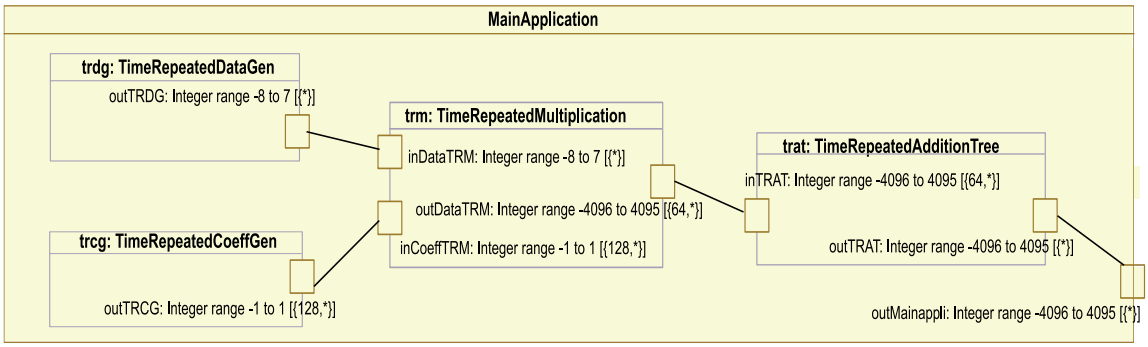
REFERENCES

[1] INRIA DaRT team, "GASPARD SoC Framework," 2009, http://www.gaspard2.org/.

[2] A. Gamatié and S. Le Beux and É. Piel and A. Etien and R. B. Atitallah and P. Marquet and J.-L Dekeyser, "A model driven design framework for high performance embedded systems," INRIA, Research Report RR-6614, 2008, http://hal.inria.fr/inria-00311115/en.

[3] OMG, "Modeling and analysis of real-time and embedded systems (MARTE)," http://www.omgmarte.org/.

[4] I.-R. Quadri, S. Meftali, and J.-L. Dekeyser, "A model driven design flow for fpgas supporting partial reconfiguration," *International Journal of Reconfigurable Computing*, 2009, Hindawi Publishing Corporation, Tentative publication date : June 2009.


MainApplication
trdg: TimeRepeatedDataGen
outTRDG: Integer range -8 to 7 [{*}]
trm: TimeRepeatedMultiplication
inDataTRM: Integer range -8 to 7 [{*}]
outDataTRM: Integer range -4096 to 4095 [{64,*}]
inCoeffTRM: Integer range -1 to 1 [{128,*}]
trat: TimeRepeatedAdditionTree
inTRAT: Integer range -4096 to 4095 [{64,*}]
outTRAT: Integer range -4096 to 4095 [{*}]
trcg: TimeRepeatedCoeffGen
outTRCG: Integer range -1 to 1 [{128,*}]
outMainappli: Integer range -4096 to 4095 [{*}]

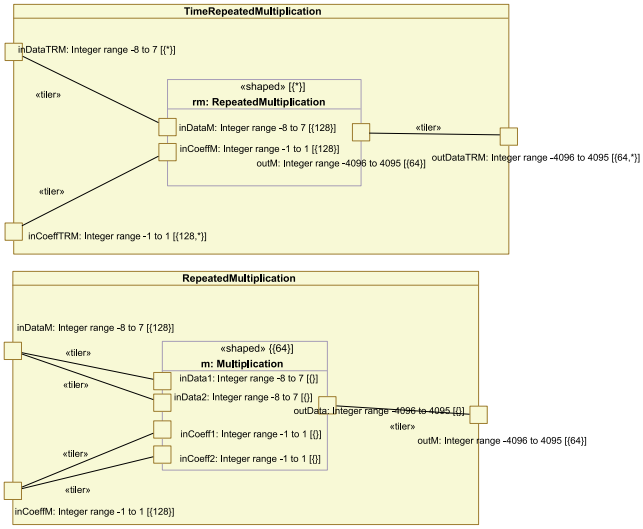


Figure.4: *Modeling of the Multiplication stage*

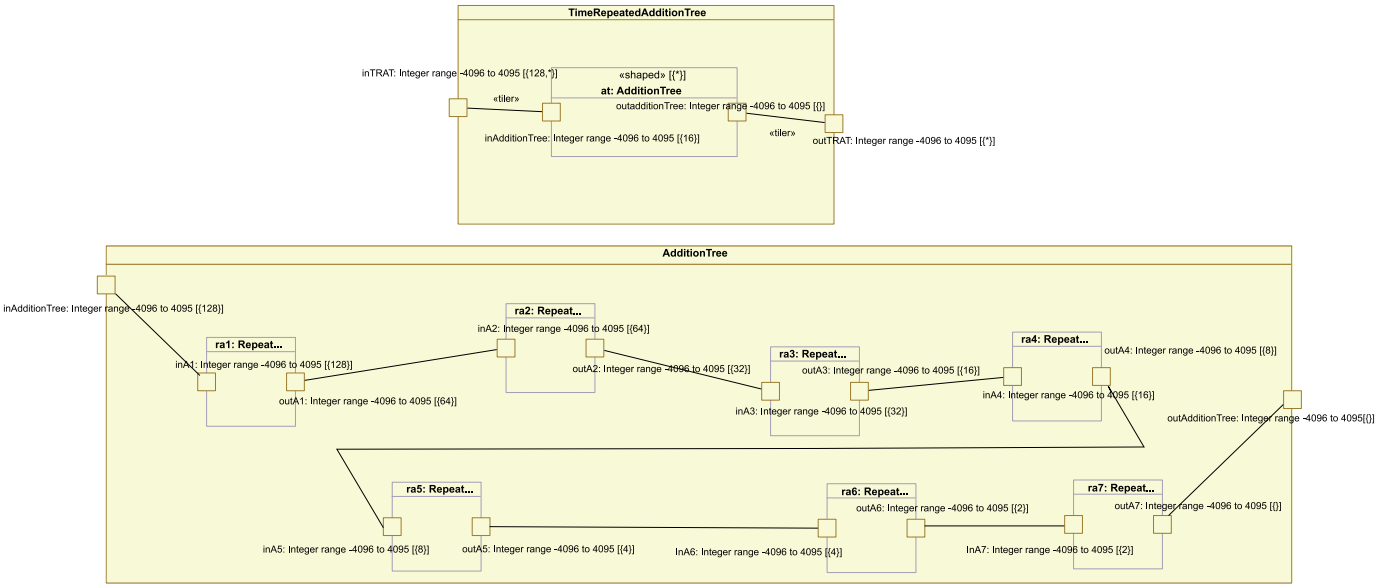


Figure.5: *The Addition tree*

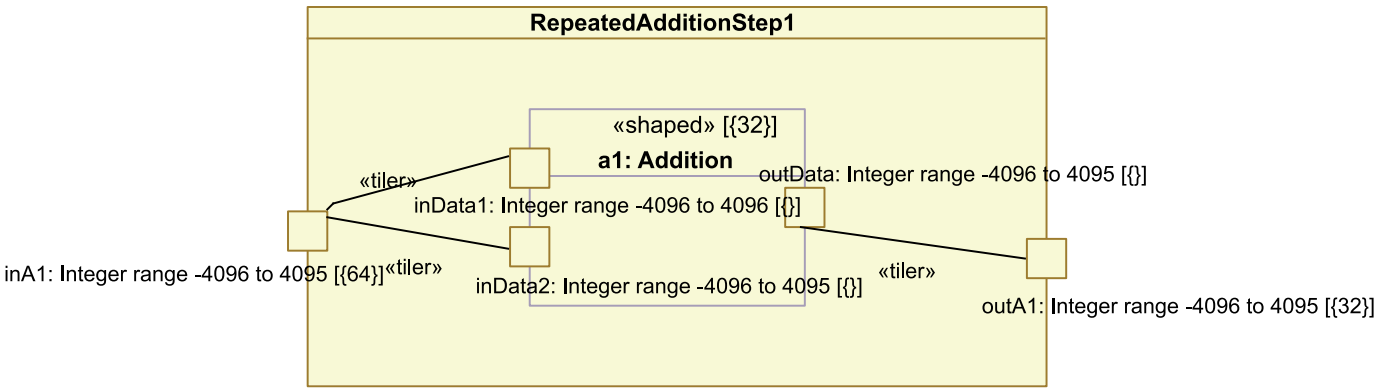


Figure.6: *An addition step*