Research Article

A State-Based Modeling Approach for Efficient Performance Evaluation of Embedded System Architectures at Transaction Level

Anthony Barreteau, Sébastien Le Nours, and Olivier Pasquier

IREENA, EA1770, Université de Nantes, Polytech-Nantes, rue C. Pauc, Bât. IRESTE, BP 50609, 44000 Nantes, France

Correspondence should be addressed to Anthony Barreteau, anthony.barreteau@univ-nantes.fr

Received 29 June 2011; Revised 7 October 2011; Accepted 20 November 2011

Academic Editor: Philippe Coussy

Copyright © 2012 Anthony Barreteau et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Abstract models are necessary to assist system architects in the evaluation process of hardware/software architectures and to cope with the still increasing complexity of embedded systems. Efficient methods are required to create reliable models of system architectures and to allow early performance evaluation and fast exploration of the design space. In this paper, we present a specific transaction level modeling approach for performance evaluation of hardware/software architectures. This approach relies on a generic execution model that exhibits light modeling effort. Created models are used to evaluate by simulation expected processing and memory resources according to various architectures. The proposed execution model relies on a specific computation method defined to improve the simulation speed of transaction level models. The benefits of the proposed approach are highlighted through two case studies. The first case study is a didactic example illustrating the modeling approach. In this example, a simulation speed-up by a factor of 7,62 is achieved by using the proposed computation method. The second case study concerns the analysis of a communication receiver supporting part of the physical layer of the LTE protocol. In this case study, architecture exploration is led in order to improve the allocation of processing functions.

1. Introduction

In the consumer domain, current trends in embedded systems design are related to integration of high-performance applications and improvement of communication capabilities and mobility. Such functionalities have big influence on system architectures, significantly rising complexity of software, and hardware resources implemented. Typically, hardware resources are organized as multicore platforms consisting of a set of modules like fully programmable processor cores, standard interface modules, memories, and dedicated hardware blocks. Advances in chip technology will allow more resources to be integrated. Consequently, massively parallel architectures clustered by application category will be adopted [1]. Furthermore, in order to improve scalability of such platforms, network infrastructure represents a convenient solution to replace bus-based communication.

In this context, the process of system architecting consists in optimally defining allocation of system applications on platform resources and fixing characteristics of processing, communication, and memory resources according to functional and nonfunctional requirements. Functional requirements express what the designer wishes to implement whereas nonfunctional requirements are used to correctly tune parameters of related architecture. Typical nonfunctional requirements under consideration for embedded systems are timing constraints, power consumption, and cost. Exploration of the design space is led according to these requirements to identify potential architectures. Performances of candidate architectures are then evaluated and compared. In order to maintain short design time, fast exploration of the design space, and reliable evaluation of nonfunctional properties early in the development process

have then become mandatory to avoid costly design iterations. Due to increasing system complexity, evaluation of architecture performances calls for specific methods and tools to assist system architects in creating reliable models.

As reported in [2], the principles of the Y-chart model are usually followed for creation of models for performance evaluation of architectures. Following this approach, a model of the *application* is *mapped* onto a model of the considered *platform* and the resulting description is then evaluated through simulation or analytical methods. Analytical methods are used to perform formal analysis on architecture models. As stated in [3], these methods fit well if deterministic or worstcase behavior is a reasonable assumption for the architecture under evaluation. Simulation approaches rely on execution of a model of the architecture under evaluation with respect to a given set of stimuli. Compared to analytical approaches, simulation methods are required to investigate dynamic and nondeterministic effects in the system model. Simulation results are obtained in order to compare performances of limited set of candidate architectures, as illustrated in approaches presented in [4, 5]. The definition of efficient simulation-based approaches targets light modeling effort and improved simulation speed.

Simulation speed and accuracy are directly related to the level of abstraction considered to model the system architecture. On both application and platform sides, modeling of computation and modeling of communication can be strongly separated and defined at various abstraction levels. Among simulation-based approaches, the Transaction Level Modeling (TLM) approach has recently received wide interest in industrial and research communities in order to improve system design and its productivity [6]. This modeling approach provides facilities to hide unnecessary details of computation and communication (pins, wires, clock, etc.). The different levels of abstraction considered in transaction level models are classified according to time accuracy and granularity of computation and communication [6, 7]. Most of recent simulation-based approaches for performance evaluation rely on languages such as SpecC [8] or SystemC [9] to provide executable specifications of architectures, notably through the TLM2.0 standard promoted by OSCI [10]. Examples of recent TLM approaches are described in [6, 11, 12]. In such approaches, early evaluation of architecture performances is typically performed with transaction level models incorporating approximated time annotations about computation and communication. Architecture models are then simulated to evaluate usage of resources with respect to a given set of stimuli. However, TLM still lacks reference models used to facilitate creation and manipulation of performance models of system architectures and to provide light modeling effort. Besides, the achievable simulation speed of transaction level models is still limited by the amount of required transactions and integration of nonfunctional properties in performance models can significantly reduce simulation speed due to additional properties included. A quantitative analysis of the speed-accuracy tradeoff is presented in [13] through different case studies and different modeling styles.

This paper presents an approach for creation of efficient transaction level models for performance evaluation of system architectures. Compared to existing works, the main contribution is about a generic execution model used to capture evolution of nonfunctional properties assessed for performance evaluation. This execution model serves as a basic instance to create approximately timed models and it can be parameterized in order to evaluate various configurations of system architectures. Furthermore, it relies on a specific computation method proposed to significantly reduce the amount of required transactions during model execution and, consequently, to improve the simulation speed. This computation method is based on the decoupling between the description of model evolution, which is driven by transactions, and the description of nonfunctional properties. This separation of concerns leads to reducing the number of events in transaction level models. Simulation speedup can then be achieved by reducing the number of context switches between modules during model simulation. The proposed execution model and the related computation method have been implemented in a specific modeling framework based on the SystemC language. The considered modeling approach provides fast evaluation of architecture performances and then allows efficient exploration of architectures. The benefits of this approach are highlighted through two case studies. The modeling approach and the generic execution model are first illustrated through a didactic example. Then, the approach is illustrated through the analysis of two possible architectures of a communication receiver based on the Long Term Evolution (LTE) protocol.

The remainder of this paper is structured as follows. Section 2 analyzes the related modeling and simulation approaches for evaluation of performances of embedded systems. In Section 3, the proposed modeling approach is presented and related notations are defined. In Section 4, we describe the proposed generic execution model. The computation method used to improve the simulation speed of models is detailed. Also, we describe the implementation in a specific simulation framework. Section 5 highlights the benefits of the contributions through two separated case studies. Finally conclusions are drawn in Section 6.

2. Related Work

Performance evaluation of embedded systems has been approached in many ways at different levels of abstraction. A good survey of various methods, tools, and environments for early design space exploration is presented in [3]. Typically, performance models aims at capturing characteristics of architectures and they are used to gain reliable data of resource usage. For this purpose, performance evaluation can be performed without considering a complete description of the application. In simulation-based approaches, this abstraction enables efficient simulation speed and favors early performance evaluation. Workload models are then defined to represent computation and communication loads applications caused on platforms when executed. Workload models are mapped onto platform models and the resulting

architecture models are simulated to obtain performance data. Related works mainly differ according to the way application and platform models are created and combined.

The technique called trace-driven simulation has been proposed for performance analysis of architectures in [14]. Following this technique, the execution of the platform model is driven by traces from the execution of the application model. A trace represents the communication and computation workloads imposed to a specific resource of the platform. In this approach, application is described in the Kahn Process Network (KPN) model to expose parallelism. Platform model is made of processing resources and communication interfaces. Each processing element is described by the number of cycles each instruction takes when executed and communication is characterized by buffer size and transfer delay. The Sesame approach [4] extends this simulation technique by introducing the concept of virtual processors. Virtual processor is used to map a trace to a transaction level model of the platform. In this approach, candidate architectures are first selected using analytical modeling and multiobjective optimization according to parameters such as processing capacities, power consumption, and cost. Potential solutions are then simulated at transaction level using SystemC. A similar approach is considered in [5].

Trace-driven simulation is also addressed in the TAPES approach [15]. In this approach, traces abstract the description of functionalities for each resource and they are defined as a sequence of processing delays interleaved with transactions. Depending on the allocation decision, each processing resource of the architecture contains one or more traces that are related to different processing sequences. Shared resources like memory or bus imply generation of additional delays as consequence of competing accesses. Architecture specification is then translated in SystemC and obtained description is simulated by triggering traces required for processing particular data in the respective resources of the architecture.

Approaches presented in [16, 17] describe the combined use of UML2 and SystemC for performance evaluation. Approach presented in [16] gets a strong emphasis on streaming data embedded systems. A specific metamodel is defined to guide designers in the creation process of application and platform models. UML2 activity diagram and class diagram are used to capture workload and platform models. Stereotypes of the UML2 MARTE profile [18] are used for nonfunctional properties and allocation description. Once allocation defined, SystemC description is generated automatically and simulated to obtain performance data. In [17], system requirements are captured as a layered model defined at the service level. Workload models are mapped onto the platform models and the resulting system model is simulated at transaction level to obtain performance data. A specific attention is paid about the way workload models are obtained and three load extraction techniques are proposed: analytical, measurement based, and source code based.

The proposed design framework in [19] aims at evaluating nonfunctional properties such as power consumption and temperature. In this approach, description of temporal

behavior is done through a model called communication dependency graph. This model represents a probabilistic quantification of temporal aspects of computation as well as an abstract representation of the control flow of each component. This description is completed by models of nonfunctional properties characterizing the behavior of dynamic power management. Simulation is then performed in SystemC to obtain an evaluation of time evolution of power consumption.

Our approach mainly differs from the above as to the way system architecture is modeled and models of workload are defined. In our approach, architecture specification is done graphically through a specific activity diagram notation. The behavior related to each elementary activity is captured in a state-action table. So, in our approach, models of workload are expressed as finite-state machines in order to describe the influence of application when executed on platform resources. The resulting architecture model is then automatically generated in SystemC to allow simulation and performance assessment. Compared to the related works, a specific attention is paid in order to reduce the time required to create models. In our works, the architecture model relies on a generic execution model proposed to facilitate the capture of the architecture behavior and the related properties. A specific method is also defined to improve the simulation time of models. The proposed modeling approach and the performance evaluation method are close to one presented in [15]. Similarly, our modeling approach considers the description of architecture properties in the form of traces for each resource. However, compared to [15], the description of the architecture model is significantly improved by the use of the proposed execution model. This reference model differs from one presented in [16] because it is not limited to streaming data application. Compared to approaches presented in [4, 16] the architecture model relies on a state diagram notation which allows modeling time-dependent behavior. A similar notation is also adopted in [17, 19] but architecture model does not rely on any reference model and requires specific development. Then, the aim of the contribution is to provide a reference model to build performance models of architectures with light modeling effort. Furthermore, this reference model makes use of the SystemC language to improve simulation speed of created performance models.

3. Considered Modeling Approach for Performance Evaluation of System Architectures

3.1. Graphical Notation. The modeling approach presented in this section aims at creating models in order to evaluate resources composing system architectures. As previously discussed, model of system architecture does not require complete description of system functionalities. In the considered approach, the architecture model combines the structural description of the system application and the description of nonfunctional properties relevant to considered hardware and software resources. The utilization

of resources is described as sequences of processing delays interleaved with exchanged transactions. This approach is illustrated in Figure 1.

The lower part of Figure 1 depicts a typical platform made of communication nodes, memories, and processing resources. Processing resources are classified as processors and dedicated hardware resources. In Figure 1, F_{11} , F_{12} , and F_2 represent the functions of the system application. They are allocated on the processing resources P_1 and P_2 to form the system architecture. For clarity reason, communications and memory accesses induced by this allocation are not represented. The upper part of the figure depicts the structural and the behavioral modeling of the system architecture. The structural description is based on an activity diagram notation inspired from [20]. This notation is close to the UML2 activity diagram. Each activity A_i represents a function, or a set of functions, allocated on a processing resource of the platform. As for example, activity A_{11} models the execution of function F_{11} on processor P_1 . Relations M_i between activities are represented by single arrow links. Transactions are exchanged through relations and they are defined as data transfer or synchronization between activities. Following the adopted notation, transactions are exchanged in conformity with the rendezvous protocol. The graphical notation adopted for the description of activity behavior is close to the Statechart [21]. Behavior related to each elementary activity models the usage of resources by each function of the system application. Behavior exhibits waiting conditions on input transactions and production of output transactions. In the notation adopted one important point is about the meaning of temporal dependencies. Here, transitions between states s_i are expressed as waiting transactions, or logical conditions on internal data. A specific data value may be a time variable which evolves naturally. This data is denoted by t in Figure 1. The amount of processing and memory resources used is expressed according to the allocation of functions. In Figure 1, the use of processing resources due to the execution of function F_2 on P_2 is modeled by the evolution of the parameter denoted by Cc_{A2} . For example, Cc_{A2} can be defined as an analytical expression to give the number of operations related to the execution of function F_2 . Value of Cc_{A2} can be influenced by data associated to the transaction received through relation M_3 .

3.2. Formal Definition. A notation similar to [16] is adopted to define more formally elements of the performance model. We define a system architecture SA modeled according to the considered approach as a tuple

$$SA = (A, M, TC), \quad (1)$$

where A is the set of activities that compose the architecture model, M is the set of relations connecting activities, and TC is the set of timing constraints related to activities. Due to the communication protocol considered in our approach, no data is stored through relations. This implies that activities can also be used to model specific communication and memory effects such as latency, throughput, or bus

TABLE 1: State-action table for specification of activities.

Current state	Next state		Actions	
	Condition	State	Condition	Action
s_0	M_3	s_1	—	$Cc_{A2} = 0$
s_1	$k < N$ AND $t = T_j$	s_0	—	$Cc_{A2} = Cc_{s1}$
	$k = N$ AND $t = T_j$	s_2	—	$Cc_{A2} = Cc_{s2}$
s_2	$t = T_k$	s_3	—	$Cc_{A2} = Cc_{s3}$
s_3	$t = T_l$	s_0	—	M_4

contentions due to competing accesses. Relations are uni-directional and each relation $M_i \in M$ is defined as

$$M_i = (A_{src}, A_{dst}), \quad (2)$$

where $A_{src} \in A$ corresponds to the emitting activity and $A_{dst} \in A$ is the receiving activity. An activity $A_i \in A$ is defined as

$$A_i = (S, M_i, M_o, F), \quad (3)$$

where S is the set of states used for activity description, $M_i \in M$ is the set of input relations, $M_o \in M$ is the set of output relations, and F is the set of transitions used to describe the evolution of A_i . States $s_i \in S$ decompose the evolution of activity in terms of time intervals. In the following, we consider that these intervals only exhibit the use of processing resources. Additional properties such as memory resources could also be addressed. A transition $F_i \in F$ is then defined as

$$F_i = (E, M_{out}, Cc), \quad (4)$$

where E is the set of conditions implying a transition of state, $M_{out} \in M_o$ is the set of output transactions related to the considered transition, and Cc is the computational complexity inferred by the state s_i following the transition F_i . Additional properties such as the memory cost could also be considered. Cc can be influenced by data associated to transactions. Conditions $E_i \in E$ can be defined as a combination of waiting conditions on a set of input relations, time conditions, and logical conditions.

Based on this set of rules, the behavior of an activity can be captured using a state-action table notation as defined in [22]. Compared to a finite state machine sensitive to the evolution of a clock signal, this table gives the evolution of activities defined at the transaction level. The description related to activity A_2 in Figure 1 is presented in Table 1.

The first column specifies the set of current states. The second column specifies the next states and the conditions under which the activity will move to those states. The third column specifies assignment of properties under study and production of output transactions. Other actions are not depicted in Table 1 for clarity reason. Conditions under which these assignments occur are also included. This means that state-action table is able to capture both Moore and Mealy types of finite state machines. It should be noted

FIGURE 1: Considered modeling approach for performance evaluation of system architectures.

that assignment of properties is done during current state whereas production of transactions is done only on state transition.

In our approach, state-action tables are used to capture the behavior and the time properties related to each elementary activity. As a result, captured behavior and related time properties depend on the considered allocation of the application.

3.3. Temporal Behavior. Using languages as SystemC, evolution of the model can be analyzed according to the simulated time supported by the simulator. In the following, the simulated time is denoted by t_s . The obtained evolution for activity A_2 is depicted in Figure 2 for internal parameter N set to 3.

As depicted in Figure 2, Cc_{s1} operations are first executed for a duration set to T_j after reception of transaction M_3 . Once N transactions received ($N = 3$ in this case), Cc_{s2} operations are executed for a duration set to T_k . The production of transaction M_4 is done once state s_3 finished, after a duration set to T_l . Such an observation gives

FIGURE 2: Time evolution of parameter Cc_{A2} according to the simulated time.

indication about the way processing resources are used when architecture executes.

The time properties used are directly influenced by the characteristics of the processing resources and by the characteristics of the communication nodes used for transaction exchange. These properties could be provided by estimations, profiling existing codes, or source code analysis, as illustrated in [17]. In the following the illustration of the modeling

approach will be done considering estimations given by analytical expressions.

Furthermore, the temporal behavior related to each activity is relevant to the function allocation. In case of a single processor architecture, functions are executed sequentially or according to a specific scheduling policy. In case of a multiprocessor architecture, behaviors should express the available parallelism to execute functions. In the following, the different allocations will be obtained by modifying the behavior related to each activity.

Moreover, in the notation adopted in Figure 1, TC_{A1} and TC_{A2} denote the time constraints to be met by the activities A_1 and A_2 . TC_{A2} expresses the time constraint to satisfy when function F_2 is executed by P_2 . In Figure 2, durations T_j , T_k , and T_l are set in order to meet TC_{A2} . TC_{A1} is the time constraint to satisfy when F_{11} and F_{12} are successively executed by P_1 . Values considered for TC_{A11} and TC_{A12} can be modified in order to consider different time repartitions for the execution of F_{11} and F_{12} .

Following this modeling approach, resulting model incorporates evolution of quantitative properties defined analytically and relevant to the use of processing resources, communication nodes, and memories. Using languages as SystemC, created models can then be simulated to evaluate the time evolution of performances obtained for a given set of stimuli. Various platform configurations and function allocations can be compared considering different descriptions of activities. In the following, a generic execution model is proposed to efficiently capture the behavior of activities and then the evolution of nonfunctional properties assessed. This reference model facilitates creation of transaction level models for performance evaluation. State-action tables are then used to parameterize instances of the generic execution model.

4. Proposed Generic Execution Model for Performance Evaluation

4.1. Behavioral Description of Proposed Generic Execution Model. A generic execution model is proposed to describe the behavior of activities and then to easily build architecture models. This execution model expresses the reception and production of transactions and the evolution of resources utilization. Its description can be adapted according to the number of input and output relations and according to the specification captured in the associated state-action table. The proposed execution model is illustrated in Figure 3 for the case of i input relations and j output relations.

As depicted in Figure 3, behavior exhibits two states. State *Waiting* is for reception of transactions. Selection of input relation is expressed through parameter $Select_M_i$. Once a transaction has been received through a relation M_{li} activity goes to state *Performance analysis*. When $Select_M_i$ is set to 0 no transaction is waited. The time conditions related to activity evolution are represented by parameter T_s . T_s is updated during model execution and its value can be influenced by data associated to transactions. This is represented by action *ComputeAfter* M_i and it is depicted

FIGURE 3: Proposed execution model for activity description.

in Figure 3 by symbol (*). Time conditions are evaluated according to the simulated time t_s . The parameter $Select_M_i$ is used to select the output relation when a transaction is produced. When $Select_M_i$ is set to 0 no transaction is produced. $Select_M_i$ and $Select_M_j$ are successively updated in state *Waiting* during model execution to meet the specified behavior.

Evolution of assessed property Cc_A is done in state *Performance analysis*. Successive values are denoted in Figure 3 by Cc_{si} . These values can be evaluated in zero time according to the simulated time t_s . This means that no SystemC wait primitives are used, leading to no thread context switches. Resulting observations correspond to values Cc_{si} and associated timestamps, denoted in Figure 3 by $T_o^{(i)}$. Timestamps are also local variables of the activity and their values are considered relatively to what we call the observed time, denoted in Figure 3 by t_o . The observed time t_o is a local time used for property evolution, whereas t_s is the simulated time used as a reference by the simulator. Using this technique, the evolution of considered property Cc_A can be computed locally between successive transactions. Details are given in the next section about how this computation technique can significantly improve simulation time of model execution.

The application of this modeling style to the activity specified in Table 1 is illustrated in Figure 4.

Figure 4 depicts a specific instance of the generic execution model for one input relation and one output relation. As indicated in Table 1, the activity evolution depends on relation M_3 , logical conditions, and specific time conditions. The evolution obtained for activity A_2 using the execution model is illustrated in Figure 5. The reception and production of transactions are depicted according to the simulated time t_s . Evolution of property assessed is represented according to the observed time t_o .

The Figure 5(a) depicts the evolution of activity A_2 when transactions occur and according to the simulated time

FIGURE 4: Application of proposed execution model to a specific activity description.

FIGURE 5: Evolution of activity A_2 according to (a) the simulation time t_s and (b) the observed time t_o .

t_s . Values of $Select_M_1$, $Select_M_2$, and T_s are successively updated to meet specified behavior. t_{s0} denotes the current simulated time. The evolution of Cc_{A2} according to the observed time t_o is represented in the Figure 5(b). Once transactions received, successive values of Cc_{A2} and timestamps are computed relatively to the arrival time of transaction. For example, in Figure 5, when the third transaction is received successive values of Cc_{A2} and timestamps values $T_o^{(6)}$, $T_o^{(7)}$, and $T_o^{(8)}$ are defined locally. The evolution of Cc_{A2} between the reception of the third transaction through relation M_3 and the production through M_4 does not imply

use of SystemC wait primitive and evolution is then obtained in zero time according to the simulated time t_s .

Next section details this computation method and how it can be applied to improve the simulation speed of performance models.

4.2. Proposed Computation Method of Nonfunctional Properties of System Architectures. As previously discussed, the simulation speed of transaction level models can be significantly improved by avoiding context switches between threads. The computation method described in this section relies on the same principle as temporal decoupling supported by the loosely timed coding style defined by OSCI. Using this coding style, parts of the model are permitted to run ahead in a local time until they reach the point when they need to synchronize with the rest of the model. The proposed method can be seen as an application of this principle to create efficient performance models. This method makes it possible to minimize the number of transactions required for the description of properties assessed for evaluation of performances. Figure 6 illustrates the application of proposed computation method to the example considered in Figure 1.

Figure 6 depicts two possible modeling approaches. The upper part of the figure corresponds to a description with 3 successive transactions. Delays between successive transactions are denoted by Δt_1 and Δt_2 . In this so-called transaction-based modeling approach, the property Cc_{A2} evolves each time a transaction is received and a similar observation to one depicted in Figure 2 can be obtained. The lower part of the figure considers the description of the activity A_2 considering application of the computation method. Here, we focus only on explanation of the computation method and the application of the generic execution model is not fully represented in Figure 6 for clarity reason. Compared to the situation depicted in the upper part of the figure, only one transaction occurs and the content of the transaction is defined at higher granularity. However, the evolution of property Cc_{A2} can be preserved by considering a decoupling with the evolution of activity A_2 . In that case, duration T_s corresponds to the time elapsed between the first transaction and the production of transaction through relation M_4 . This value is locally computed relatively to the arrival time of input transaction M_3 and it defines the next output event. In Figure 6, this is denoted by action *ComputeAfter* M_1 . The time condition is evaluated during state s_0 according to the simulated time t_s . This computation supposes estimates about values of Δt_1 and Δt_2 .

The evolution of property Cc_{A2} between two external events is done during state s_0 . Successive values, denoted in Figure 6 by Cc_{si} , are evaluated in zero time according to the simulated time. This means that no SystemC wait primitives are used, leading to no thread context switches. Resulting observations correspond to values Cc_{si} and associated timestamps $T_o^{(i)}$. Timestamps values are considered relatively to what we call the observed time, denoted in Figure 6 by t_o . Using this technique, evolution of the considered property can then be computed locally between successive

FIGURE 6: Comparison of two modeling approaches (a) a transaction-based modeling approach and (b) a state-based modeling approach.

FIGURE 7: Evolution of property Cc_{A2} considering (a) a transaction-based modeling approach and (b) the proposed state-based modeling approach.

transactions. Compared to the previous transaction-based approach, the second modeling approach can be considered as a state-based approach. Assessed properties are then locally computed in the same state which reduces the number of required transactions. Figure 7 represents time evolution of property Cc_{A2} considering the two modeling approaches illustrated in Figure 6.

The Figure 7(a) illustrates the time evolution of property Cc_{A2} with 3 successive input transactions. During simulation of the model each transaction implies a thread context switch between activities and Cc_{A2} evolves according to the simulated time t_s . On the Figure 7(b), successive values of property Cc_{A2} and associated timestamps are computed

at the reception of transaction M_3 . Evolution is depicted according to the observed time t_o . Improved simulation time is achieved due to the amount of context switches avoided. More generally, we can consider that when the number of transaction is reduced by a factor of N , a simulation speedup by the same factor can be achieved. This assumption has been verified through various experimentations presented in [23]. Here the achievable simulation speedup is illustrated in the first case study considered in this paper.

The proposed generic execution model makes use of this computation method to provide improved simulation time of performance models. In order to validate this modeling style, we have considered the implementation

FIGURE 8: Graphical modeling in the CoFluent Studio framework to implement the proposed execution model.

of the proposed execution model in a specific modeling framework.

4.3. Implementation of the Generic Execution Model in a Specific Framework. The proposed execution model has been implemented in the framework CoFluent Studio [24]. This environment supports creation of transaction level models of system applications and architectures. Graphical models captured and associated codes are automatically translated in a SystemC description. This description is then executed to analyze models and to assess performances. We used the so-called *Timed-Behavioral Modeling* part of this framework to create models following the considered approach. Figure 8 illustrates a possible graphical modeling to implement the proposed execution model. It corresponds to the specific case illustrated in Figure 4 with one input relation and one output relation.

In Figure 8, the selection of input relation is represented by an alternative statement. Selection is denoted by parameter *Select_Min*. Transactions are received through relation M_3 . Parameter T_s is expressed through the duration of the operation *OpPerformanceAnalysis*. This operation is described in a sequential C/C++ code to define computation of local variables and display. The Algorithm 1 example corresponds to part of required instructions to obtain observations depicted on the Figure 7(b).

The procedure *CurrentUserTime* is used in CoFluent Studio to get the current simulated time. In our case, it is used to get the reception time of transactions and then to compute values of timestamps. The procedure *CofDisplay*

is used to display variables in a $Y = f(X)$ chart. In our case, it is used to display studied properties according to the observed time. The keyword *OpDuration* defines the duration of operation *OpPerformanceAnalysis*. It is evaluated according to the simulated time supported by the SystemC simulator. Successive values of Cc_{A2} and timestamps can be provided by estimations and they can be computed according to data associated to transactions. This implementation can be extended to the case of multiple input and output relations. It should be noted that this modeling approach is not limited to a specific environment and it could be applied to other SystemC-based frameworks. In the following, we consider application of this approach to the analysis of two specific case studies.

5. Experiments

5.1. Pipelined FFT Case Study. We aim at illustrating application of the proposed generic execution model and the computation method previously presented in Section 4 through a didactic case study. Application considered here is about a Fast Fourier Transform (FFT) algorithm which is widely used in digital signal processing. A pipeline architecture based on dedicated hardware resources is analyzed. A possible implementation of this architecture is described in [25]. An 8-point FFT is described in order to easily illustrate the proposed modeling approach. The created performance model enables to estimate resource utilization and the computation method is used to reduce the simulation time required to obtain reliable information. Figure 9 represents the pipeline architecture and the related performance model.

The lower part of Figure 9 shows the 3-stage pipeline architecture analyzed. This architecture enables to simultaneously perform transform calculations on a current frame of 8 complex symbols, load input data for the next frame of data, and unload the 8 output complex symbols of the previous frame. Each pipeline stage implements its own memory banks to store input and intermediate data. Each one is made of a processing unit that enables to perform arithmetic operations (addition, subtraction, and multiplication) on two complex numbers. Each processing unit performs an equivalent of 10 arithmetic operations on 2 real numbers at each iteration and four iterations of each processing unit are required to perform one transform calculation on 8 points. The number of clock cycles to perform each iteration is related to the way logic resources are implemented. The clock frequency of each processing unit is defined according to the expected data throughput and calculation latency. The upper part of Figure 9 gives the structural description of the proposed performance model. The behavior of each activity is specified to describe the resource utilization for each stage of the architecture. Each activity is described using the proposed generic execution model with one input and one output as presented in Figure 4. Time constraints are denoted by TC_{Stage1} , TC_{Stage2} , and TC_{Stage3} .

```

{
  switch (s)
  {
    {...}
    case s1:
      ts0 = CurrentUserTime(ns);
      To = ts0;
      CofDisplay('to=%f ns, CcA2=%f op/s', To, 0);
      CofDisplay('to=%f ns, CcA2=%f op/s', To, Ccs1);
      To = ts0 + Tj;
      CofDisplay('to=%f ns, CcA2=%f op/s', To, Ccs1);
      CofDisplay('to=%f ns, CcA2=%f op/s', To, 0);
      Select_Min = 1;
      Select_Mout = 0;
      OpDuration = Tj;
      if(k < N)s = s0;
      else {s = s2; k + +;}
      break;
    case s2:
      {...}
  }
}

```

ALGORITHM 1

FIGURE 9: Performance model of a 3-stage pipeline architecture.

A first performance model has been defined by using the state-based modeling approach presented in Section 4. At the abstraction level considered, a transaction is made of 8 complex symbols. Table 2 gives the specification of the activity *Stage3*.

s_0 is the waiting state for reception of transactions through relation *InputStage3*. During processing state s_1 , Cc_{s1} operations are executed. T_{Proc} represents the computation duration. T_{Idle} represents the delay between two iterations of the processing unit. For one 8-point FFT execution, four iterations of s_1 and s_2 are required to analyze the resource utilization for this pipeline stage. Considering the previously presented computation method, the evolution instants of Cc_{Stage3} are computed locally according to the arrival time of transactions. A transaction made of 8 complex symbols

is finally produced through relation *OutputSymbol*. The specifications of *Stage1* and *Stage2* activities are described in the same way. Three states are needed to describe reception of transactions through the relations *InputSymbol* and *InputStage2*, evolution of assessed properties Cc_{Stage1} and Cc_{Stage2} and production of output transactions through the relations *InputStage2* and *InputStage3*. The specifications of each activity described with state-action tables are then used to parameterize the generic execution model given in Figure 4.

Figures 10 and 11 depict possible observations obtained from the simulation of the performance model described with the CoFluent Studio tool.

Figure 10 shows the resource utilization and the time evolution of the computational complexity per time unit for the third stage of pipeline to perform one 8-point FFT. In this

TABLE 2: Behavior of Stage3 activity described by using the state-based modeling approach.

Current state	Next state		Actions	
	Condition	State	Condition	Action
s_0	InputStage3	s_1	—	$Cc_{Stage3} = 0$
s_1	$t = T_{Proc}$	s_2	—	$Cc_{Stage3} = Cc_{s1}$
s_2	$t = T_{Idle}$ AND $k < 4$	s_1	—	$Cc_{Stage3} = 0$
	$t = T_{Idle}$ AND $k = 4$	s_0	—	OutputSymbol

FIGURE 10: Time evolution of the computational complexity (in MOPS) of the third pipeline stage.

FIGURE 11: Time evolution of the global computational complexity (in MOPS) of the pipeline architecture with configuration considered.

example a data throughput of 100 Mega complex symbols per second is considered. Then, an 8-point FFT is to be performed every 80 ns. During this period, four iterations of each processing unit are executed. To meet the expected data throughput, time constraints TC_{Stage1} , TC_{Stage2} , and TC_{Stage3} are set to 20 ns. Observations depicted in Figure 10 are obtained with a computation duration T_{Proc} set to 15 ns. A computational complexity per time unit of 666 MOPS is also observed.

Figure 11 presents simulation results obtained to observe evolution of the global computational complexity related

to the three stages of the pipeline. Three successive 8-point FFT are executed in order to observe the influence of simultaneous processing.

The lower part of Figure 11 gives information about the occupation of processing units on each pipeline stage. The upper part of Figure 11 enables to analyze the resource utilization according to the computation duration applied on each pipeline stage. For each stage T_{Proc} is fixed to 15 ns and T_{Idle} to 5 ns. A maximal computational complexity per time unit of 2 GOPS is observed with this timing configuration. This information enables the designer to deduce the number of logic resources required for the architecture. The simulation time to execute the performance model for one 8-point FFT took about 50 μ s on a 2.66 GHz Intel Core2 duo machine. This simulation is fast enough to compare different architecture configurations and to analyze different tradeoffs between data throughput and resource utilization.

A second performance model has been defined at a lower level of granularity following a transaction-based modeling approach as presented in Section 4. The goal is to evaluate the execution speedup obtained with the first model by reducing the number of context switches during simulation. At the data granularity considered, a transaction corresponds to one complex symbol. The second model initiates 32 transactions, whereas 4 transactions were required in the first model. Properties Cc_{Stage1} , Cc_{Stage2} , and Cc_{Stage3} evolve also each time a transaction is received. The same results previously presented with the first model can be observed. For the second performance model, the simulation speed is significantly improved by reducing the numbers of transactions required and the related context switches between threads. The first performance model enables to obtain a simulation speedup of 7.62. A minor difference with the theoretical factor speedup of 8 is due to weak effect of algorithm added to obtain the same observation about resource utilization. In [23], it is shown how the measured simulation speedup evolves linearly with the reduction factor of thread switches.

5.2. LTE Receiver System Case Study. This section highlights application of the proposed generic execution model to favor creation of transaction level models and to perform architecture exploration. The case study considered here concerns the creation of a transaction level model for analysis of processing functions involved at the physical layer of a communication receiver implementing part of the LTE protocol. This protocol is considered for next

FIGURE 12: Activity diagram of the LTE receiver studied.

generation of mobile radio access [26]. Associated baseband architecture demands high computational complexity under real-time constraints and multiprocessor implementation is then required [27]. The aim of the proposed model is a comparison of performances achieved with two potential architectures. Information obtained by simulation enables to identify and compare the amount of logic resources and memory capacity required for potential architectures. Figure 12 describes the structural description of the studied LTE receiver, captured with the activity diagram notation adopted.

A single-input single-output (SISO) configuration is analyzed. Figure 12 depicts the activities proposed for performance analysis of baseband functions of the LTE receiver [28], namely, OFDM demodulation, channel estimation, equalization, symbol demapping, turbo decoding, and transport block (TB) reassembling. Different configurations exist for a LTE subframe. The relation *LTESymbol* represents one OFDM symbol and processing of a symbol takes 71428 ns [29]. An OFDM demodulation is performed on data associated to the relation *LTESymbol*. It is typically performed using FFT algorithm. In a LTE subframe known complex symbols, called pilots and denoted by *PilotSymbol*, are inserted at a distance of 4 and 3 OFDM symbols from one another to facilitate channel estimation. Data associated to relation *DataSymbol* are equalized to compensate effects of propagation channel. The *Symbol Demapper* activity represents interface between symbol level processing and bit processing. Channel decoding is performed through turbo decoder algorithm. The activity *TBReassembly* receives binary data block through the relation *SegmentedBlock*. Data blocks are then transmitted through relation *TransportBlock* to the Medium Access Control (MAC) layer each 1 ms when 14 OFDM symbols have been received and processed by the different functions related to the physical layer.

Figure 13 shows the performance model defined to compare the studied architectures.

The lower part of Figure 13 depicts the two studied architectures. Architecture I corresponds to an heterogeneous architecture. P_1 is considered as a processor and P_2

as a dedicated hardware resource. Architecture II consists in implementing each function as a set of dedicated hardware resources denoted by P_3 . The upper part of the figure depicts the model to analyze and compare performances obtained with these two architectures. Time constraints are denoted in Figure 13 by $TC_{\text{OFDMDemod}}$, $TC_{\text{ChanEstim}}$, $TC_{\text{Equalizer}}$, and $TC_{\text{TurboDecoder}}$. TC_{P1} is the time constraint related to the sequential execution of the functions allocated on P_1 . Its value is set to the reception period of input transactions, which is equal to 71428 ns. The behavior of each activity is specified in order to evaluate the memory cost and the computational complexity each function causes on the resources when executed. Table 3 illustrates the specified behavior to analyze the computational complexity related to the channel estimation function.

States s_0 and s_2 represent waiting states on relation *Pilot-Symbol*. Operations performed for one channel estimation iteration are performed during states s_1 and s_3 . Cc_{s1} and Cc_{s3} correspond to the number of arithmetic operations executed during these processing states. According to the OFDM symbol received, 4 or 3 iterations of channel estimation are required. T_{Proc} is the computation duration of processing states. It is fixed according to the time constraint $TC_{\text{ChanEstim}}$. The behaviors related to the other activities have also been specified with state-action tables. Based on a detailed analysis of resources required for each function, we have defined analytical expressions to give relations between functional parameters related to the different configurations of LTE subframes and the resulting computational complexity in terms of arithmetic operations [30].

Each activity has been captured by using the proposed generic execution model. Activities *ChannelEstimator*, *SymbolDemapper*, *TurboDecoder*, and *TBReassembly* have been captured using the same description given in Figure 4. Activity *OFDMDemodulator* corresponds to the case of two output relations, whereas activity *Equalizer* has been described with two input relations. A design test environment has been defined to produce LTE subframes with different configurations. We captured the model in the CoFluent Studio tool. Each activity was described in a way

FIGURE 13: Proposed model to analyze performances obtained with different architectures.

TABLE 3: State-action table for specification of the *ChannelEstimator* activity.

Current state	Next state		Actions
	Condition	State	Action
s_0	PilotSymbol	s_1	$Cc_{Estim} = 0$
s_1	$t = T_{Proc}$ AND $k < 4$	s_1	$Cc_{Estim} = Cc_{s1}$ EstimatedChannelFactor
	$t = T_{Proc}$ AND $k = 4$	s_2	
s_2	PilotSymbol	s_3	$Cc_{Estim} = 0$
s_3	$t = T_{Proc}$ AND $k < 3$	s_3	$Cc_{Estim} = Cc_{s3}$ EstimatedChannelFactor
	$t = T_{Proc}$ AND $k = 3$	s_0	

similar to one presented in Section 4. Modeling effort was then significantly reduced due to the adoption of a generic model. The model instances are parameterized according to the specifications defined in state-action tables. The complete architecture model corresponds to 3842 lines of SystemC code, with 22% automatically generated by the tool. The rest of the code is sequential C/C++ code defined for computation and display of properties studied. Table 4 shows the time constraints considered in the following for the simulation of the two architectures studied.

These values have been set to guarantee processing of a LTE subframe each 1 ms. In case of architecture I, OFDM demodulation, channel estimation, and equalization functions are executed sequentially on the processor P_1 . $TC_{OFDMDemod}$, $TC_{Equalizer}$, and $TC_{ChanEstim}$ are fixed to meet time constraint TC_{P1} and to limit the computational complexity per time unit required by processor P_1 . $TC_{TurboDecoder}$ is also fixed to limit the computational complexity per time unit required by P_2 .

In case of architecture II, each function can be executed simultaneously. $TC_{OFDMDemod}$ is equal to the reception period of input transaction *LTESymbol*. $TC_{Equalizer}$ and $TC_{ChanEstim}$ are defined to perform a maximum of 4 iterations of channel estimation and equalization during OFDM demodulation. $TC_{TurboDecoder}$ is set to perform turbo decoding of 1 or 2 blocks of data during equalization.

Figure 14 shows the observations obtained for architecture I with the time constraints set. Results observed correspond to the reception of a LTE subframe with the following configuration: number of resource blocks allocated: 12, size of FFT: 512, modulation scheme: QPSK, and number of iterations of turbo decoder: 5.

The Figure 14(a) depicts the evolution of the computational complexity per time unit on processor P_1 . The Figure 14(b) shows the evolution of the computational complexity per time unit on P_2 to perform operations related to turbo decoding. A maximum of 1,6 GOPS is estimated for processor P_1 . With the time constraints set, it is mainly

TABLE 4: Time constraints applied for each activity according to architectures.

Architecture	$TC_{OFDM\text{Demod}}$	$TC_{Chan\text{Estim}}$	$TC_{Equalizer}$	$TC_{Turbo\text{Decoder}}$
I	$0,2 * TC_{P1}$	$0,12 * TC_{P1}$	$0,04 * TC_{P1}$	$0,08 * TC_{P1}$
II	71428 ns	14285 ns	14285 ns	7142 ns

FIGURE 14: Evolution of the computational complexity per time unit of (a) processor P_1 and (b) P_2 .

FIGURE 15: Evolution of the estimated memory cost (in Kbit) for architecture I.

due to OFDM demodulation. A maximum of 221 GOPS is estimated to perform turbo decoding. Figure 15 describes the evolution of the memory cost associated to the symbol demapper and transport block reassembly functions.

The memory cost evolves each time a transaction is received or sent by one of the three functions studied. Instants (1) correspond to the evolution of the memory cost after reception of a packet of data by the symbol demapper function. The amount of data stored in memory increases each time a packet of data is received by the symbol demapper. At instant (2), the complete LTE subframe has been processed at the physical layer level and the resulting data packet is produced. The maximum value achieved with the considered LTE subframe is estimated to be 1920 bits.

FIGURE 16: Evolution of the computational complexity per time unit for architecture II.

Figure 16 illustrates the observation obtained with time constraints considered for architecture II and for the LTE subframe configuration evaluated previously.

The upper part of Figure 16 shows evolution of the computational complexity per time unit for P_3 . The lower part depicts the resource utilization of P_3 . A maximum computational complexity per time unit of 177,812 GOPS is observed when turbo decoding and OFDM demodulation are performed simultaneously.

Table 5 summarizes the simulation results obtained for architectures I and II. The maximal computational complexity metric is considered here because it directly impacts area and energy consumption of the two candidate architectures. With architecture II, we note that simultaneous execution of functions on the processing resources enables to reduce the maximum computational complexity from 223,382 GOPS to 177,812 GOPS. For architecture I, the resource usage metric enables to express the percentage of time used by P_1 to execute each function. For architecture II, the resource usage metric expresses the usage of each dedicated resource for processing one OFDM symbol.

Observations given in Figures 14, 15, and 16 are used to estimate expected resources of the architecture. Similar observations can be obtained for different subframe configurations. The simulation time to execute the performance model for 1000 input frames took 11 s on a 2.66 GHz Intel Core2 duo machine. This is fast enough for performing performance evaluation and for simulating multiple configurations of architectures. Time properties and quantitative

TABLE 5: Maximal computational complexity and utilization of the processing resources observed for the two architectures considered.

	Architecture I				Architecture II	
	P_1		P_2		P_3	
	Maximal computational complexity (GOPS)	Resource usage (%)	Maximal computational complexity (GOPS)	Resource usage (%)	Maximal computational complexity (GOPS)	Resource usage (%)
OFDMDemodulator	1,612	20	—	—	0,322	100
ChannelEstimator	0,406	48	—	—	0,244	80
Equalizer	0,151	12	—	—	0,03	80
TurboDecoder	—	—	221,77	8	177,49	10
Application	1,612	80	221,77	8	177,812	—

properties defined for each activity can be easily modified to evaluate and compare various architectures.

The main benefit of the presented approach comes from the adoption of a generic modeling style. The modeling effort is then significantly reduced. We estimate that the creation of the complete model of the LTE Receiver architecture took less than 4 hours. Once the model created, parameters can easily be modified to address different architectures and simulation time is fast enough to allow exploration. In the presented modeling approach the simulation method used makes possible to run ahead evolution of the studied properties in a local time until activities need to synchronize. This favours creation of models at higher abstraction level. Synchronization points are defined by transactions exhibited in the architecture model. Then, this approach is sensitive to estimates related to each function and further works should be led in order to evaluate the simulation speed-accuracy tradeoff.

6. Conclusion

Abstract models of system architectures represent a convenient solution to maintain the design complexity of embedded systems and to enable architecting of complex hardware and software resources. In this paper, we have presented a state-based modeling approach for the creation of transaction level models for performance evaluation. According to this approach, system architecture is modeled as an activity diagram and description of activities incorporates properties relevant to resources used. The presented contribution is about a generic execution model defined in order to facilitate the creation of performance models. This model relies on a specific computation method to significantly reduce the simulation time of performance models. The experimentation of this modeling style has been illustrated through the use of the framework CoFluent Studio. However, the approach is not limited to this specific environment and it could be applied to other SystemC-based frameworks. Further research is directed towards validation of estimates provided by simulation and applying the same modeling principle to other nonfunctional properties such as dynamic power consumption.

References

- [1] C. H. van Berkel, "Multi-core for mobile phones," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE '09)*, pp. 1260–1265, April 2009.
- [2] D. Densmore, R. Passerone, and A. Sangiovanni-Vincentelli, "A platform-based taxonomy for ESL design," *IEEE Design and Test of Computers*, vol. 23, no. 5, pp. 359–373, 2006.
- [3] M. Gries, "Methods for evaluating and covering the design space during early design development," *Integration, the VLSI Journal*, vol. 38, no. 2, pp. 131–183, 2004.
- [4] A. D. Pimentel, C. Erbas, and S. Polstra, "A systematic approach to exploring embedded system architectures at multiple abstraction levels," *IEEE Transactions on Computers*, vol. 55, no. 2, pp. 99–111, 2006.
- [5] C. Haubelt, J. Falk, J. Keinert et al., "A systemC-based design methodology for digital signal processing systems," *EURASIP Journal on Embedded Systems*, vol. 2007, Article ID 47580, 22 pages, 2007.
- [6] F. Ghenassia, *Transaction-Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems*, Springer, 2005.
- [7] L. Gai and D. Gajski, "Transaction level modeling: an overview," in *Proceedings of the 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codeign and System Synthesis (CODES+ISSS '03)*, pp. 19–24, Newport Beach, Calif, USA, October 2003.
- [8] D. D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, and S. Zhao, *SpecC: Specification Language and Design Methodology*, Kluwer Academic, Dodrecht, The Netherlands, 2000.
- [9] Open SystemC Initiative (OSCI), "Functional specification for SystemC 2.0," <http://www.systemc.org>.
- [10] Open SystemC Initiative TLM Working Group, Transaction Level Modeling Standard 2 (TLM 2), June 2008.
- [11] R. Dömer, A. Gerstlauer, J. Peng et al., "System-on-chip environment: a SpecC-based framework for heterogeneous MPSoC design," *EURASIP Journal on Embedded Systems*, vol. 2008, no. 1, Article ID 647953, 2008.
- [12] Soclib: a modelisation and simulation platform for system on chip design, 2003, <http://www.soclib.fr>.
- [13] G. Schirner and R. Dömer, "Quantitative analysis of the speed/accuracy trade-off in transaction level modeling," *Transactions on Embedded Computing Systems*, vol. 8, no. 1, article 4, 2008.
- [14] P. Lieveise, P. van Der Wolf, K. Vissers, and E. F. Deprettere, "A methodology for architecture exploration of heterogeneous

- signal processing systems,” *The Journal of VLSI Signal Processing*, vol. 29, no. 3, pp. 197–207, 2001.
- [15] T. Wild, A. Herkersdorf, and G. Y. Lee, “TAPES-Trace-based architecture performance evaluation with SystemC,” *Design Automation for Embedded Systems*, vol. 10, no. 2-3, pp. 157–179, 2005.
 - [16] T. Arpinen, E. Salminen, T. D. Hämäläinen, and M. Hännikäinen, “Performance evaluation of UML2-modeled embedded streaming applications with system-level simulation,” *EURASIP Journal on Embedded Systems*, vol. 2009, Article ID 826296, 16 pages, 2009.
 - [17] J. Kreku, M. Hoppari, T. Kestilä et al., “Combining UML2 application and systemC platform modelling for performance evaluation of real-time embedded systems,” *EURASIP Journal on Embedded Systems*, vol. 2008, Article ID 712329, 18 pages, 2008.
 - [18] Object Management Group (OMG), “A UML profile for MARTE, beta 1 specification,” August 2007.
 - [19] A. Viehl, B. Sander, O. Bringmann, and W. Rosenstiel, “Integrated requirement evaluation of non-functional system-on-chip properties,” in *Proceedings of the Forum on Specification, Verification and Design Languages (FDL’08)*, pp. 105–110, Stuttgart, Germany, September 2008.
 - [20] J. P. Calvez, *Embedded Real-Time Systems: A Specification and Design Methodology*, John Wiley & Sons, New York, NY, USA, 1993.
 - [21] D. Harel and M. Politi, *Modeling Reactive Systems with Statechart*, McGraw-Hill, 1993.
 - [22] D. D. Gajski, *Principles of Digital Design*, Prentice-Hall, 1996.
 - [23] S. Le Nours, A. Barreateau, and O. Pasquier, “Modeling technique for simulation time speed-up of performance computation in transaction level models,” in *Proceedings of the Forum of specification and Design Languages (FDL ’10)*, vol. 2010, pp. 136–141, Southampton, UK, September 2010.
 - [24] CoFluent Design, <http://www.cofluentdesign.com/>.
 - [25] Xilinx. LogiCORE Fast Fourier Transform v7.1., http://www.xilinx.com/support/documentation/ip_documentation/xfft_ds260.pdf.
 - [26] E. Dahlman, S. Parkvall, J. Skold, and P. Beming, *3G Evolution, HSPA and LTE for Mobile Broadband*, Academic Press, 2008.
 - [27] C. Jalier, D. Lattard, A. A. Jerraya, G. Sassatelli, P. Benoit, and L. Torres, “Heterogeneous vs homogeneous MPSoC approaches for a mobile LTE modem,” in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE ’10)*, pp. 184–189, Dresden, Germany, March 2010.
 - [28] 3GPP. TS 36.201, Evolved Universal Terrestrial Radio Access (E-UTRA); LTE physical layer; General description, March 2010, <http://www.3gpp.org/ftp/Specs/html-info/36201.htm>.
 - [29] 3GPP. TS 36.211 V9.1.0, Evolved Universal Terrestrial Radio Access (E-UTRA); Physical channels and modulation, April 2010, <http://www.3gpp.org/ftp/specs/html-info/36211.htm>.
 - [30] J. Berkmann, C. Carbonelli, F. Dietrich, C. Drewes, and W. Xu, “On 3 G LTE terminal implementation—standard, algorithms, complexities and challenges,” in *Proceedings of the International Wireless Communications and Mobile Computing Conference (IWCMC ’08)*, pp. 970–975, August 2008.