System Specification with the SPF Model

J-Ph.Diguet, G.Gogniat, P.Danielo, J-L.Philippe
Lester, UBS University,
Lorient, France
guy.gogniat@univ-ubs.fr

M.Auguin
I3S, University of Nice,
Sophia Antipolis,France
Michel.Auguin@i3s.unice.fr

Abstract

In this paper we propose a specification intermediate model in order to handle a seamless global design of complex and heterogeneous digital systems. This model is based on three views: System, Process, Function and targets control-oriented and signal processing applications. The model exhibits all the information (e.g. operators, controls, multi-dimensional operands) required to estimate memory, control and computation units and to partition the application onto an heterogeneous architecture.

1. Introduction

Multimedia and telecommunication applications involve generally data intensive computations embedded in event-driven system functions with hard real-time constraints. The first step prior to the design of an application is to write a behavioral specification. In practice, a single formalism cannot be used to describe a whole application due to its heterogeneous behavior [1],[2],[15]. Therefore, traditional design approaches consider generally a fragmented design flow. The decomposition of the design flow results from the implementation target and/or from the type of information handled by the application. If a part of the application is prefigured to be implemented on a software component (hardware component respectively), it is described for example as a C or C++ (VHDL respectively) program. Codesign approaches attempt to avoid this preliminary hardware/software technology-based design flow splitting using an unified language (e.g., SystemC, ECL). However, the type of information handled by the application may lead to a semantic-based splitting of the design flow. For example, the control dominated part of the application may be described using an extended finite state machine or related models [3],[4],[5] whereas the data dominated part is described using a data flow model [5],[6]. Next each part is designed by adhoc approaches and in a final step the synthesized sub-systems are integrated to form the whole system. This fragmented design flow is particularly error prone and leads to an increase of the total design cost. To alleviate this problem, some initiatives attempt to define an unified modeling (e.g., *charts, SPI) and design framework, generally based on cosimulation (e.g., Ptolemy, CoWare, VCC from Cadence). However, these approaches do not support really a seamless complete design flow from high level specification to the generation of VHDL for synthesis and assembly code for processors. Indeed, one of the main steps in system design is partitioning the application functionalities in order to derive an architecture that matches real-time constraints and minimizes the total area and energy consumption. Since the efficiency of partitioning is greatly dependent on the granularity chosen to describe the application [7], the system specification must facilitate a hierarchical modeling. Furthermore, a complete system can be conceived as a set of communicating processes [1] where each process can have a data dependent and/or a control dependent execution flow. The system specification would be able to represent both the process level concurrency and the conditional execution flow within a process. Partitioning methods have to deal with these different models of computation: at the upper level it is mainly a schedulability problem [8],[9],[10] whereas at the process level it is a conditional data flow partitioning problem [11],[12]. Partitioning imposes that each level of the hierarchy is characterized by abstract information (e.g. execution times) derived from potential implementations of the functions involved at this level. This implementation-dependent characterization of the

hierarchy may be derived from previous designs, from hard or soft IPs, or from estimates (at the system [13] or architectural [14] levels). Getting reliable estimates is a challenging task since their accuracy has a direct impact on the quality of the resulting system architecture. Whereas a coarse or medium grain specification is enough for partitioning, estimation requires a fine grain description [13], [14]. Since we focus on system design, this fine grain specification must be also architecture-independent in order to estimate characteristics of both hardware and software implementations. However, the design flow starts with a system design space exploration where accurate estimates are not required for all parts of the system: for example a function that has a neglected impact on the total system cost and execution time. Consequently, our aim is to define a system specification formalism which can be used as support for the main steps involved in a complete system and architecture design flow.

In the following, section 2 introduces the main characteristics of the SPF model and gives the motivations of our work. Section 3 presents the three views composing the model and precises the links between these views. Section 4 details each view. Through an exemple, we illustrate in section 4 the utilisation of the SPF model at the Process and Function views. Finally we conclude.

2. The SPF model

In this paper we propose an intermediate model of specification (the SPF model) in order to handle a seamless global design of complex and heterogeneous digital systems. Different methods and tools are currently under development in our laboratory: performance and resource estimation methods [13],[14], system level partitioning [12]. Based on the SPF framework, these methods can be applied gradually to the different levels of the hierarchy within each view: System, Process and Function (see Fig 1).

The first aim of the SPF (System Process Function) model is to specify digital applications at a high level of abstraction in order to perform design space exploration. This exploration is performed gradually from a system level to an architectural level. At the system level any assumptions are made on the final target architecture. This is a key feature for heterogeneous applications since the underlying architecture can be computation, control, data-transfer oriented or a mix. The SPF model is based on six basic concepts which are concurrency, behavioral hierarchy, communication, synchronization and time.

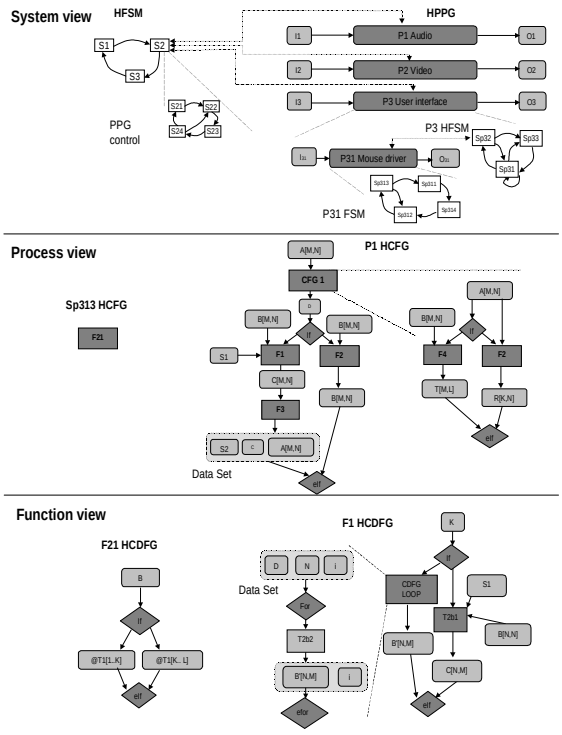


Figure 1• The SPF model

Concurrency exhibits parallel computations or data-accesses which may be needed from the process level to the operation level to meet the real-time constraints. Behavioral hierarchy enables to master the complexity since related functionalities are merged together and localized effects are abstracted away at higher levels. Through the hierarchy, designers can manage the great amount of processing and control implied in modern digital systems. Communication allows exchanges of data and control information between application tasks. Synchronization is necessary to represent execution dependencies between concurrent processes whereas temporal attributes are necessary to specify real-time constraints at various granularity levels. (e.g. periodic execution, deadline constraint). Hence, the targeted properties of the formalism are:

• Executable or simulable specification model.

• Well suited to describe an application as a set of concurrent processes where each process is data or control dominated or a mix.

• Hierarchical model to master the complexity of systems composed of interrelated subsystems and to give a higher flexibility to partitioning and estimation tools.

• Support for system and architecture design space exploration through the exhibition of

potential parallelism, mutual exclusion, control structures and data accesses.

• Efficient generation of hardware (e.g. VHDL) and software (e.g. C) models including communication.

The SPF Model constitutes the entry point of our codesign framework and enables to specify efficiently complex digital systems (e.g. signal processing and control-oriented systems). The model gives all the information (e.g. operators, multi-dimensional operands) required to estimate memory, control and computation units and to partition the application onto an heterogeneous architecture.

3. The SPF views

The SPF model is a graphical model. It is based on three complementary views : System, Process and Function (Fig 1). Starting from the System view the specification is progressively refined to the Function view. More details are given as the designer proceeds from the System view to the Function view. Such a decomposition is required to perform an efficient design space exploration. Once the architecture is designed, the System view will correspond to the overall architecture while the Function view will correspond to hardware and/or software components. A view enables a hierarchical representation according to several levels. When the finest granularity can't be split anymore using the view model, a transition towards the next view is performed. The System view is the highest specification level which emphasizes the different configurations of the system to design (e.g. set top box). This view exhibits all the concurrent processes to run in a given configuration (e.g. G729, MPEG2, AC3, USB) and the control structure needed to fire them (Fig 2).

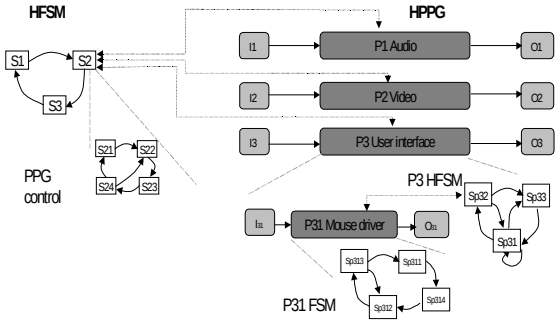


Figure 2• The system view

The top level system view is defined as a finite state machine where a state is associated with a set of data-flows. A finite state machine can be hierarchical if it involves multiple Parallel Processes (PPG). For example in Fig 2 the PPG FSM controls the processes active within the S2 configuration. A hierarchy (HPPG) is necessary if processes are modeled as sub-systems. For example, in Fig 2 process P3 is split into a FSM that controls the execution of the process P31. Thus, a system is split until a discrete event model is no more required. Typically the smallest specified FSM can represent a communication protocol with hard real time constraints whereas the highest level FSM represents the different configurations of the system.

The Process view is data-flow oriented and allows each process involved in the system to be refined. According to the standard specifications a process is seen as a hierarchical graph of functions (e.g. autocorrelation or prefiltering for G729) that are connected through control structures and that exchange data. Data can be multi-dimensional or scalar. In case of multi-dimensional data, addressing modes are explicitly represented (through the use of attributes associated with data nodes) in order to permit memory unit estimation. Control structures are mainly loop and test. Parallelism between functions is also highlighted. Within this view, the smallest processing node is a function which is naturally defined in a standard block-diagram (Fig 3).

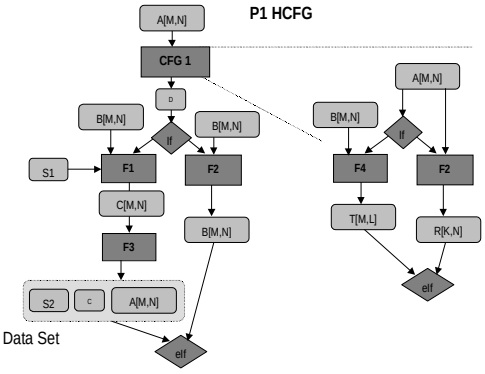


Figure 3• The process view

The Function view describes every function associated with the processes. In this view a function is described with elementary operations (or hierarchical control data flow graph of elementary operations) which can be represented at different levels of granularity (e.g. primitive operators *add*, *mac* but also *dct*, *acs* depending on designer's choices [13]). Control structures, and data exchanges involved in a function are also represented (Fig 4).

The graphical representation of the SPF model

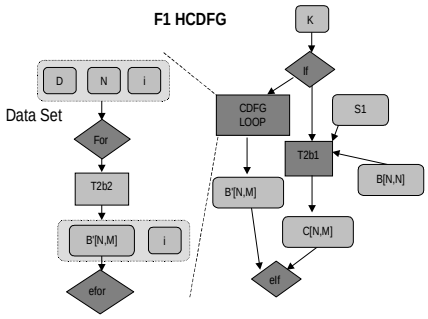


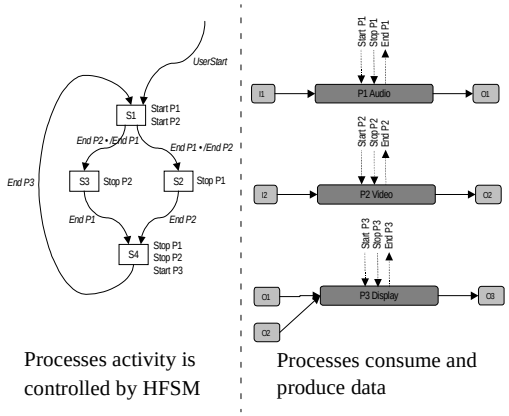
Figure 4• The function view

allows to zoom down and up from the System to the Function views in a homogeneous way. Using this feature and the hierarchy of each view, the designers can easily travel through the system description.

4. Model of each view

4.1. The System view

The model for the System view is state and activity oriented. We use a HFSM to represent all the (sub-)states of the system. Transitions between states occur on events which are triggered by the external environment (e.g. user interface at the top level, signal protocol at the lower levels) or by the processes (e.g. End of a process) of the system. In the HFSM model, nodes (S_X) represent states and edges (E_X) represent transitions between states. In a FSM a single node can be active at the same time, but each node can fired several processes in the case of concurrency. The scheduling between the processes is handled by the FSMs. For example in Fig 5 the state S1 trig-



Processes activity is controlled by HFSM

Processes consume and produce data

Figure 5• FSM & Process links

gers processes P1 and P2. When one of the two processes ends the FSM evolves to the state S2 or S3. Finally in State S4 the processes P1 and P2 are idle and P3 is triggered. Hence, the FSM synchronises the evolution between concurrent processes. Edges between states correspond to events generated by the processes and the environment. The processes are modeled through a Hierarchical Parallel Process Graph (HPPG) representation. This representation is activity-oriented: the processes consume and produce data (e.g. audio frame in the case of G729). Processes are described independently since synchronization are made through the HFSM. Process and data-transfer operations are described by nodes (P_X and D_X). Note that, at the System view, processes may be hierarchical and so described also as sub-systems with the same model. The scheduling between processes is clearly defined in this view. After the estimation step, each process of the system is characterized with its performances (i.e. speed, area and consumption). Data are represented with nodes in order to extract a data access graph to estimate and optimize memory structures. Futhermore, in case of multi-dimensional data each addressing mode is also represented by a graph to estimate the address generator units. Indeed in video applications memory structure is generally the main issue of the design.

4.2. The Process view

The model for the Process view is activity-oriented, using a Hierarchical Control Flow Graph (HCFG). The HCFG is composed of nodes representing CFG, control, data and function. These nodes are connected with edges that correspond to functional dependencies. The edges do not need attribute since all data are held by nodes. A CFG node corresponds to a hierachical node. At the lowest level, only control, data and function nodes are permitted. Control nodes can represent structures like (Fig 6): loop (i.e. for, do while, while) and switch (if, switch). The control nodes have predefined graphical patterns where the conditions are represented through a CDFG graph in order to estimate corresponding ressources. Till the function nodes can be split in other function nodes the hierarchy is exhibited (e.g., in Fig 7 function Main is decomposed in functions A to D). But the computational details of each function are defined in the Function view. This decomposition enables to define a specific model for each view. In order to guaranty consistency between hierarchical levels in a view all input and output data are automatically duplicated when a node is refined (for example in Fig 7 nodes A, B, G1, G2, C, G1# and G2# are duplicated from

```
(note : Short G1, G2, G3[] : global variables )

void MAIN( Short A, Short B, Short *C ) {
Short C_I = 2 ;
Short D_I ;
FCT_A( A, G1, G2, &C_I) ;
FCT_B( C_I, B, &G2 ) ;
FCT_C( C_I, G2, &D_I ) ;
FCT_D( D_I, B, &G1 ) ;
*C = C_I;
}
void FCT_A( Short A, Short D, Short E, Short *C ) {
*C = *C * D * E ;
switch (A) {
                case 1 : *C = A + *C ;
                case 2 : *C = A - *C ;
                case 3 : *C = A * *C ;
                case 4 : *C = A / *C ;
}
}
void FCT_B( Short G, Short B , Short *E ) {
        Short I = G ;
        while ( I < 10 && B != 0 ) {
                E = E * 3 ;
        }
}
void FCT_C( Short C, Short E, Short *F ) {
        Short SC, C2 ;
        if (C < 2) {SC = -1;
                        if (C == 1) C2 = 2;
                        else C2 = 1;}
        else {          SC = 1;
                        if (C == 3) C2 = 1;
                        else C2 = 2; }
        *F = SC*E*G3[C2-1]<<3;
}
void FCT_D( Short C, Short B, Short *D ) {
        D = C - 2 - B ;
        do {
        D = D * 2 ;
        } while ( D > B )
}
```

See Fig 7 for the representation

See Fig 8 for the representation

Figure 6• C benchmark

function Main into the lower level). Moreover, data are characterized by a single assignation in order to handle life span of each variable in the application. This information is necessary to estimate memory unit. Hence, an index is associated with each variable to representent the number of write accesses to the data. For example a data A that is written three times will appear A#1, A#2 and A#3.

4.3. The Function view

The model for the Function view is a Hierarchical Control Data Flow Graph (HCDFG). The HCDFG is composed of CDFG, control, computation and data nodes. Alike the Process view the CDFG node is used to represent the hierarchy of the graph. Hence, at the lowest level only control, computation and data are allowed. This is the most refined view of the application. Control nodes represent control structures of the function (e.g. loop, test). A CDFG node can be labeled as a loop to hide the loop body at the upper level. By this way we can chose between keeping the loop or attening the CDFG in order to unroll the loop. Data can also be merged together in Data Set to bring clarity to the graphical representation. Moreover, Data Set description can be very powerful for memory size optimization during the codesign flow.

5. Example: C benchmark

To illustrate the SPF model we consider a C program (Fig 6) including common control structures (i.e. switch, while, if, do-while). For this

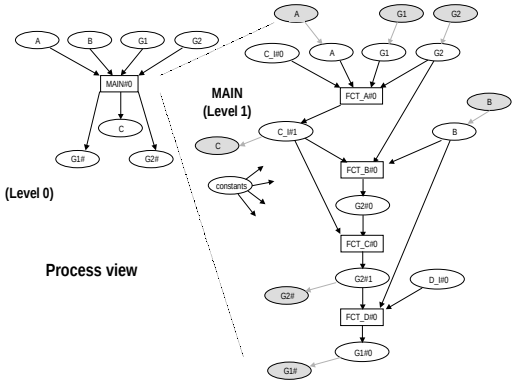


Figure 7• The process view: main

example Process and Function view are only illustrated since we are currently developing our codesign tools at this level. The main function is composed of four functions. Hence in the Process view this decomposition is explicitly represented through the hierarchy (Fig 7). We can notice the duplication of input and output data nodes in order to guaranty the consistency of the graph at each level transition.

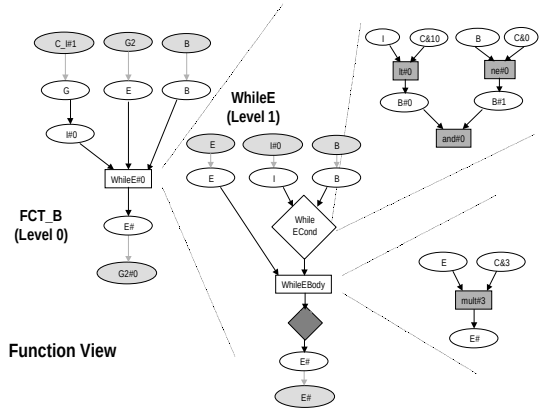


Figure 8• The function view: FCT_B

In Fig 8 the function FCT_B is represented. The main characteristics of this example is the use of the hierarchy to abstract local information. The condition of the control structure is represented by a CDFG in order to be able to perform estima-

tion on it since the evaluation of the condition requires computational resources. Each node of the graph is also labelled with specific attributes in order to bring information to the codesign tools.
Since, until now we have modelized only part of a whole application, we plan in the near future to model a complete wireless system with the SPF model in order to evaluate the capacity of this model to manage complex application and to validate our estimation and partitioning tools.

6. Conclusion

After attentive examination of available specification models we have elaborated a new model gathering the features we need to achieve the following objectives:
• Handle every kind of control-oriented and computational system.
• Enable to focus on specific parts of the system depending on their impact on the final architecture.
• Refine the model with specific attributes in order to get all the information needed to do an efficient design space exploration.
Hence, this model enable to specify and represent heterogeneous applications with the aim to apply simulation, estimation, partitioning and synthesisable VHDL model generation methods. Each node has numerous attributes filled up firstly by the designer, then by the code parser tool and then incrementally by the above method tools while the codesign flow proceeds. This model is suited to codesign approaches which include control-oriented (unpredictable events) and data flow (digital signal processing) parts. The view and hierarchy progressive refinement of the specification make the codesign flow tractable for real digital applications. The SPF model is currently under development in our laboratory. It is built on Java in order to facilitate its portability. Till now the input language translated to the Process and Function views is C. The designer has to describe the System and the Process views firstly because it is humanly tractable and because it is based mainly on the designer’s skill. On the contrary the Function view, which is a tedious and error prone task, results from a parsing step of an input language (e.g. C).

7. References

[1] L. Lavagno, E. Sentovich, ECL: a specification environment for system-level design, In Proc. DAC'99, New Orleans, Louisina, pp 511-516.
[2] D. Ziegenbein, R. Ernst, K. Richter, J. Teich, L. Thiele, Combining multiple models of computation for scheduling and allocation, In Proceedings of CODES'98, March 1998, pp 9-13.
[3] N. Halbwachs, Synchronous programming of rective systems, Kluwer Academic Publishers, 1993.
[4] F. Balarin et al., Hardware-software co-design of embedded systems: the POLIS approach, Kluwer Academic Publishers, 1997.
[5] S. Edwards, L. Lavagno, E. A. Lee, A. Sangiovanni-Vincentelli, Design of embedded systems: formal models, validation and synthesis, Proceedings of the IEEE, vol. 85, no. 3, March 19997, pp 366-390
[6] J. Buck, E.E. Lee, The token flow model, In Proceedings of Data Flow Workshop, Hamilton Island, Australia, May 1992.
[7] J. Henkel, R. Ernst, The interplay of run-time estimation and granularity in HW/SW partitioning, In Proceedings of 4th Workshop on Hardware/Software Codesign, Pittsburg, Pensylvania, March 1996, pp 52-58.
[8] T.Y. Yen, W. Wolf, Hardware-software co-synthesis of distributed embedded systems, Kluwer Academic Publishers, 1996.
[9] B. Dave, N. Jha, CASPER: concurrent hardware-software co-synthesis of hard real-time aperiodic and periodic specifications of embedded system architectures, In Proceedings of DATE'98, Paris, 1998, pp 118-124.
[10] F. Balarin, A. Sangiovanni-Vincentelli, Schedule validation for embedded reactive real-time systems, In Proceedings of DAC'97, Anaheim, California, 1997.
[11] P. Eles, K. Kunchcinski, Z. Peng, A. Doboli, P. Pop, Scheduling of conditional process graphs for the synthesis of embedded systems, In Proceedings of DATE'98, Paris, February 1998, pp 132-138.
[12] M. Auguin, L. Bianco, L. Capella, E. Gresset, Partitioning Conditional Data Flow Graphs for Embedded System Design, In proceedings of ASAP'2000, Boston, July 2000."
[13] H.Thomas, J-PhDiguet, and J-L.Philippe, "A methodology for an application profiling at a system level," in Ieee Work. on Signal Processing Systems (SiPS) Design and Implementation, Taiwan, Oct. 1999.
[14] S.Bilavarn, G.Gogniat, and J-L.Philippe, "A hardware-software codesign methodology for heterogeneous archi- tecture estimation," in Int. Conf. on Signal Proc. Appli. & Techno., Orlando, USA, Nov. 1999.
[15] A.A. Jerraya, M. Romdhani, Ph. Le Marrec, F. Hessel, P. Coste, C. Valderrama, G.F. Marchioro, J.M.Daveau, N.-E. Zergainoh,"Multilanguage Specification for System Design and Codesign", TIMA RR-02-98/12 ; chapter in "System-level Synthesis", NATO ASI 1998 edited by A. Jerraya and J. Mermet, Kluwer Academic Publishers, 1999.