Rapid Design of Specific MPSoC prototype within FPGA

Emna Kallel
Electrical department
CES Research Unit, National school of Engineers of Sfax, Tunisia
kallelemna@yahoo.fr

Yassine Aoudni
Electrical department
CES Research Unit, National school of Engineers of Sfax, Tunisia
yassine.aoudni@gmail.com

Mohamed Abid
Electrical department
CES Research Unit, National school of Engineers of Sfax, Tunisia
mohamed.abid@enis.rnu.tn

***Abstract*—This paper presents an idea of automatic generation of SoC architecture from a functional specification. Starting from an application task graph, C code will be generated automatically. Then the user can add the specific code of each task. Compilation rules are used in order to have a correct task graph and c code. Finally, an automatic generation of hardware and software SoC architecture will be down to get a new model prototype of the FPGA based SOC platform. We demonstrate the effectiveness of the proposed idea by the implementation of CAGT software tool which can generate efficient and correct C code from a task graph. The generated code is compliant to the ANSI C standard thus can be accepted by most compilers.**

***Keywords*—automatic code generation, functional specification, Compilation rules, SoC architecture.**

I. INTRODUCTION

As most design decisions are taken in early design phases at high abstraction levels, there is a need for methodologies and support tools helping the designer to select the best design alternatives. Much work has been done in synthesizing the HW part of the system. However, most industrial embedded software is still created manually from the system specification [3]. It is desired that the embedded software could be generated automatically from the system specification. Several commercial behavioral or high-level synthesis tools are available, e.g., Cynthesizer by Forte Design Systems [5], CatapultC by Mentor Graphics [1], and NEC's CyberWorkBench [14]. Moreover, automatic code generation tools for embedded processors exist [6, 15]. Various design methodologies for designing embedded software are also available. Many of these starts from an abstract model of ([2], [10]). One methodology is to manually write both models and verify them for equivalence by checking for similar properties using techniques like bounded model checking [16]. However, this would require manual effort in model rewriting as well as re-verifying every time the architecture is modified. Another methodology would be verifying the functional specification first and then automatically refining it to an equivalent architecture model [13]. In this paper, we address this problem by describing a design method which can automatically generate C code from a system specification model and by proving its effectiveness by implementing automatic code generation software tool: Custom Architecture Generator Tool (CAGT).

Fig. 1 shows how a specification model is refined to an implementation level model. Each task in the specification is generated to a unique component in the architecture. The model generation algorithm has the specification model as input. It uses the compilation rules to generate a correct C code. The output consists in custom architecture like DSP, Custom instruction, and MPSoC that can be validated through simulation or verification.

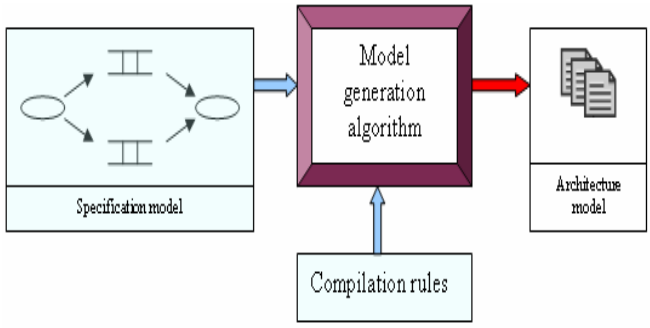

Figure 1: Architecture refinement

The rest of this paper is organized as follows: Section II gives an insight into the related work on software code generation in system design. Section III describes how the automatic software generation is integrated with the system level design method. Details of the code generation process are covered in Section III.A, Section III.B and Section III.C. Section IV.A describes the CAGT software. Section IV.B illustrates a GUI example with CAGT. Experimental results are shown in Section IV.C and Section V concludes this paper with a brief summary and an outlook on future work.

II. RELATED WORK

Some related work can be found on code generation for embedded software. There are approaches to automatic code generation from abstract models (e.g UML [4]), from graphical finite state machine design environments (e.g UML StateCharts [17]), or from synchronous programming languages (e.g Esterel) [7]. In [18], a software synthesis approach from a concurrent process specification through intermediate Petri-Net is given. The proposed method applies quasi-static scheduling to a set of Petri-Nets to produce a set of corresponding state machines, which are then mapped syntactically to the final software code. In [8], a way of combining static task scheduling and dynamic scheduling in software synthesis is proposed. In [12], a method for automatic generation of application-specific operating systems and corresponding application software for a target processor is given. While these approaches mainly focus on software scheduling issues, no efficient code generation method from system specification is described. In POLIS [11], software synthesis from Co-design Finite State Machine (CFSM) is presented. Code generation is performed in two steps: (1) transformation of the CFSM specification into a s-Graph and (2) translation of s-Graph into portable C code. A small customized operating system consisting of a scheduler and drivers for the I/O channels is used to correctly implement the run-time behavior of the input CFSMs. This work, however, is mainly for reactive real time systems and can't be easily extended to other more general frameworks. In [9], software code generation from a high-level model of operating system called SoCOS is presented. However, SoCOS requires its own proprietary simulation engine and it requires manual refinement to get the software code. In [3], software generation from SystemC SLDL based on the redefinition and overloading of SystemC class library elements is presented. Their approaches use the same SystemC code both for the system-level specification and for the target binary code generation. However, they have very strict requirements with regards to the input SystemC model (e.g no event wait/notify are allowed inside process and the processes lack a sensitivity list). Besides, there is no information regarding how the processes in the input SystemC model get scheduled in the final implementation, which is very important considering the real time requirements of the embedded software.

In this work, we aim to generate SoC architecture from a task graph description to accelerate embedded system conception and increase productivity. We also present the design process steps in which the architecture model is automatically generated from a compiled task graph. The approach is similar to that of Gajski [10], where a method of automatically creating embedded software from system level model written in system level design language (SLDL) is presented.

III. DESIGN METHOD

The overall design flow of the design methodology is based on (i) a task graph of the application, (ii) compilation of task graph using the compilation rules, (iii) code generation from a XML description of the compiled task graph after selecting parameters by the editor (iv) architecture model generation for SOC platforms. Fig. 2 shows our design method steps.

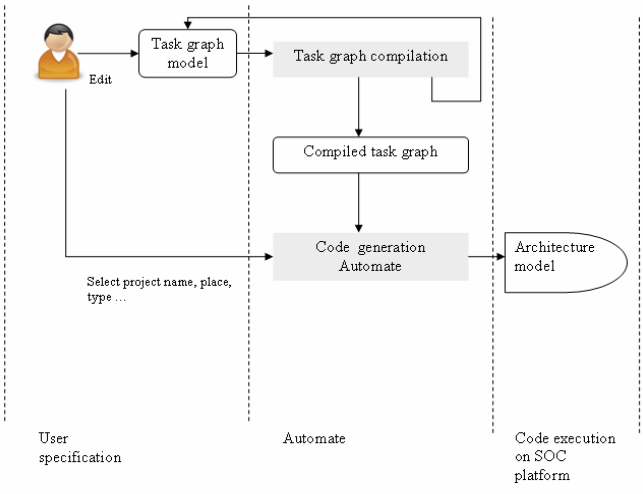

Figure 2: Our design method that generates code from a task graph model

A. *specification model*

The specification model gathers a set of concepts to specify the application part of a system. It expresses tasks and data dependencies of the application.

In our specification model, task management is handled by the operating system µC-OS II which runs on each mpSoC NIOS II processor. This operating system offers to the user the means to communicate between the various processes through communications tools (mutex, mailbox, etc). Tasks are synchronized by using routines to access the mutex and the mailbox core hardware. These functions are specific to the mutex and the mailbox core and directly manipulate low-level hardware. In our RTOS model, tasks have a set of parameters like name, size, priority, processor number, etc.

TABLE I shows some of Hardware Access Routines used in task synchronization.

TABLE I. HARDWARE ACCESS ROUTINES

Function Name	Description
altera_avalon_mailbox_pend()	Blocks waiting for a message to be in the mailbox.
altera_avalon_mailbox_post()	Posts a message to the mailbox.
altera_avalon_mutex_lock()	Locks the mutex. Will not return until it has successfully claimed the mutex.
altera_avalon_mutex_unlock()	Unlocks the mutex.

B. *Compilation rules*

The task graph compilation process verifies the description of tasks and processors. The main idea is that we correct the mistakes in the task graph by applying some rules in order to have a correct code generation.

Rules for C code generation are described in TABLE II.

TABLE II. TASK GRAPH COMPILATION RULES

Rules	Description
R1	The diagram must not be empty
R2	The diagram must not be composed by an only object
R3	Two different objects mustn't have the same name
R4	Two different objects mustn't have the same priority

The DiagramValidation function reflects the four rules for software code generation. Fig. 3 shows the code of DiagramValidation (LFDDiagram) function.

```
1: LFDDiagram ={FD1,FD 2,……,FD k}
2: boolean isvalid <- true;
3:  if isEmpty(LFDDiagram) then                                  }
4:  write(" the diagram is empty !");                            } R1
5:  isvalid <- false;                                            }
6: end if
7: for all FormDrawed FD in LFDDiagram do
8:    if  ((not (instanceof (FD,Tasks)) and                      }
9:    (orphelin(FD)))then                                        }
10:  write "Erreur : Orphan Object, the object                   } R2
11:  FD is orphan";                                              }
12:  isvalid <- false;                                           }
13:  end if
14: for i = indexOF(FD, LFDDiagram) TO k do
15:      if equals(FD, FDi ) then                                }
16:      write "Error: Name Duplication, D the                   }
17:      Objects F and FDi have the same name";                  } R3
18:       isvalid <- false;                                      }
19:       end if                                                 }
20:      if  (instanceof (FD,Tasks) and instanceof
21:       (FDi,Tasks) ) then
22:          if ( Priority(FD)= Priority(FDi)) then              }
23:          write "Warning: Priority Duplication,               }
24:          the Objects  FD and FDi have the                    } R4
25:          same  priority";                                    }
26:          isvalid <-false;                                    }
27:          end if
28:       end if
29:    end for
30: end for
```

Figure 3: DiagramValidation (LFDDiagram) code

The DiagramValidation function verifies the task graph created by the user. The input to DiagramValidation is the whole of objects created in the diagram "LFDDiagram".
TABLE III shows the several functions used in DiagramValidation.

TABLE III. DIAGRAMVALIDATION 'S FUNCTIONS

Functions	Description
isEmpty(LFD*Diagram)*	returns true if the vector "LFD*Diagram"* is empty
instanceof (*FD,Tasks*)	returns true if the object FD is an instance of the Tasks class
orphelin(FD)	returns true if the object FD is created alone
indexOF(FD,LFD*Diagram*)	returns the index of the object FD in the LFD*Diagram*
equals(FD, FD_i)	retruns true if the Object FD and FD_i have the same name
Priority(FD)	returns the priority of the object FD

The DiagramValidation function returns the value of the attribute *isvalid*. The diagram will be validated only when *isvalid* is true.

C. Code generation automate

After compiling the task graph for the considered application, the next step in our design method (see figure 2) consists in the automatic generation of the correspondent C code. This is an important step for extraction of performance parameters in terms of execution times, area values, and other non-functional properties.

Fig. 4 shows the code of generateCcode(nbprocessor) function which generates the code for the considered application.

```
1: Begin
2:     String S <-"";
3:     S <- S + includesListGenerator ();
4:     S <- S + constantsGenerator ();
5:     S <- S + structureGenerator ();
6:     S <- S + globalVariablesGenerator ();
7:   for i = 0 TO nbprocessor do
8:     S <- S + createAllTasks(i);
9:    end for
10:  return S;
11: end
```

Figure 4: generateCcode(nbprocessor) code

The input to generateCcode function is the diagram processors number selected by the user.
TABLE IV shows the several functions used in generateCcode function.

TABLE IV. GENERATECCODE'S FUNCTIONS

Functions	Description
includesListGenerator ()	for generation of includes list if exist
constantsGenerator ()	for generation of constants list if exist
structureGenerator ()	for generation of structure list if exist
globalVariablesGenerator ()	for generation of global variables list if exist
createAllTasks(i)	for the code task generation

IV. SOFTWARE IMPLEMENTATION: CUSTOM ARCHITECTURE GENERATOR TOOL

A. *Custom Architecture Generator Tool presentation*

We test the effectiveness of the proposed method by the *Custom Architecture Generator Tool (CAGT)*. The CAGT project seeks to develop techniques to aid in the development of reliable architecture based embedded systems using advanced development and verification systems. As shown in figure 5, CAGT offers a Graphic User Interface (GUI) for automatic generating processor architecture from a high specification level.

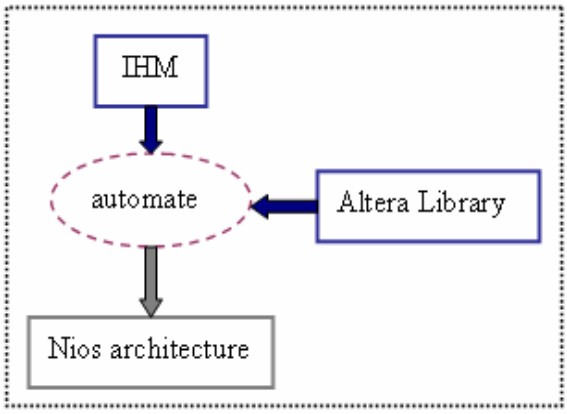

Figure 5: CAGT working strategy

The CAGT system has several principal modules.
TABLE V shows the functionalities of CAGT modules.

TABLE V. CAGT'S FUNCTIONALITIES

module	functionalities
Custom Instructions Integrator	Integrate Custom Instructions in the Nios processor
Template Converter	Converts a Template to NIOS Coprocessor
Statistics generator	▪ Calculates code complexity ▪ Calculates the number of templates or elementary operations
Generator of monoprocessor real time application	▪ Realizes a mono processor task diagram Generates the equivalent C code
Generator of multiprocessor real time application	▪ Realizes a multiprocessor task diagram ▪ Generates the equivalent C code

B. *Graphic User Interface of the multiprocessor real time application generation module*

The GUI construction is very important in the software design. In our tool, the aim is to design an ergonomic application (efficient and easy).
Fig. 6 illustrates the CAGT GUI of the multiprocessor real time application generation module.

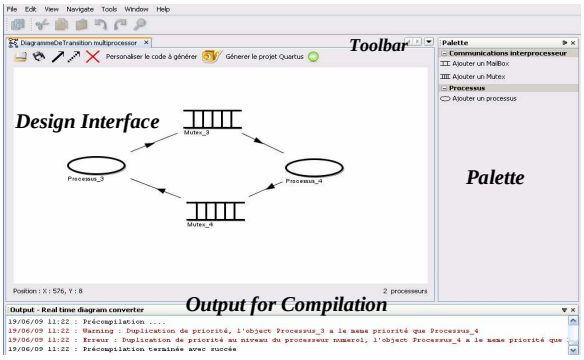

Figure 6: GUI of the multiprocessor real time application generation module

C. *Experimentation and results*

1) *Application: Two-Dimensional Fast Fourier Transform*

The entry point of our design method is a multiprocessor task graph designed and edited by the CAGT user. For the experimental evaluation of the CAGT software, we use the FFT example of a 4*256 matrix (matrix A).
This application is used to compute FFT of a complex sequence of size m, a power of 4, with "decimation-in-frequency decomposition" method. The output is in digit-reversed order. Each complex value is with interleaved 16-bit real and imaginary parts. An excerpt of such an application is illustrated by TABLE VI.

TABLE VI. 2D FFT APPLICATION PROCESSORS ROLES

parallel Processors	sequential processors	Priority	role
	Allocate_MatA	1	Allocate memory for Matrix A
	Allocate_MatB	2	Allocate memory for coefficient Matrix B
FFT_X1		3	Calculate the FFT of x1 (the line number 1 of matrix A)
FFT_X2		3	Calculate the FFT of x2 (the line number 2 of matrix A)
FFT_X3		3	Calculate the FFT of x3 (the line number 3 of matrix A)
FFT_X4		3	Calculate the FFT of x4 (the line number 4 of matrix A)
	Assembling	4	Assemble the FFT_X1, FFT_X2, FFT_X3 and FFT_X4 vectors to have finally the FFT of matrix A

The 2D FFT case study consists in tree steps:

- Step1: After allocating memory for matrix A and B, the "Allocate_MatA" and "Allocate_MatB" processors send to "FFT_X1", "FFT_X2", "FFT_X3" and "FFT_X4" processors respectively

the vectors x1[ ], x2[ ], x3[ ], and x4[ ] (the rows of matrix A) and the vectors w1[ ], w2[ ], w3[ ] and w4[ ] (the rows of coefficient matrix B), using hardware mailboxes.

- Step2: The "FFT_X1", "FFT_X2", "FFT_X3" and "FFT_X4" processors will establish in parallel the fft of each line of the matrix A. Each processor sends a message to the "Assembling" processor through a hardware mailbox, when it finishes.
- Step3: The fft of the vectors x1[ ], x2[ ], x3[ ], and x4[ ] will be assembled by the "Assembling" processor to obtain, finally, the fft of matrix A.

In the multiprocessor task graph design, CAGT user can, also, specify the priority of each processor and other processor parameters like the name, processor number, return type…

2) Results

After a correct task graph compilation, the code generation process creates the C code equivalent to the compiled task graph.

TABLE VII Illustrates the conception effort spent for realization of the complete 2D FFT application with and without using the CAGT software. It indicates a large difference between the conception time using CAGT and the manual conception time.

TABLE VII. CONCEPTION TIME WITH AND WITHOUT CAGT BY ALTERING THE PROCESSORS NUMBER

Processors number	1	2	4	7	8	10	15
Time conception with CAGT (day)	2	3	5	6	7	8	8
Time conception without CAGT (day)	7	10	26	43	55	70	90

Fig. 7 proves the important gain of time offered by CAGT. However the curve of conception time without CAGT is infinitely growing. Conceiving an application without using CAGT needs a very long time comparing when using CAGT. So, conception without using CAGT can cause an important loss in terms of cost.

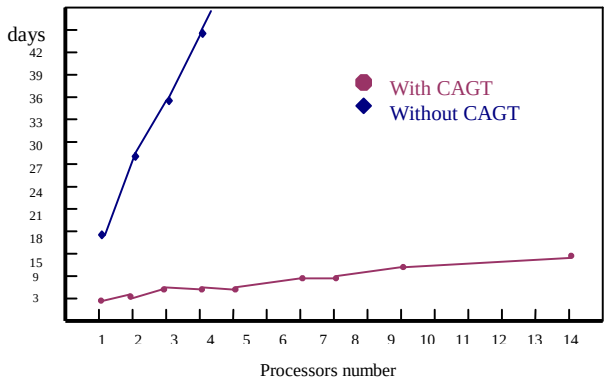

Figure 7: conception time of 2D FFT application with and without CAGT vs processors number

V. CONCLUSIONS AND FUTURE WORK

This paper presents a design method that, starting from a specification model, generates the corresponding SOC architecture code. For this purpose, we developed transformations from the application model to the compiled model and a C code generation.

This method is fully automatized and has been validated by implementing the CAGT software which can automatically generates efficient and correct C code for embedded system from his GUI.

As future work, we plan to optimize the generated architecture model in order to enhance the FPGA implementation of the produced design.

REFERENCES

[1] http://www.mentor.com.

[2] Rational. http://www.rational.com/uml/index.html.

[3] F. Herrera, H. Posadas, P. Snchez, and E. Villar. "Systematic embedded software generation from systemc. *Proceedings of Design Automation and Test in Europe", DATE*, 2003.

[4] Selo Sulistyo, and Andreas Prinz. "Recursive Modeling for Completed Code Generation", Proceedings of the 1st Workshop on Behaviour Modelling in Model- Driven Architecture, June 2009.

[5] http://www.forteds.com.

[6] Christian Haubelt, Thomas Schlichter, Joachim Keinert and Mike Meredith. "SystemCoDesigner: Automatic Design Space Exploration and Rapid Prototyping from Behavioral Models", DAC 2008, June 8–13, 2008, Anaheim, California, USA.

[7] F. Boussinot and R. de Simone. "The ESTEREL Language". In *Proceedings of the IEEE*, September 1991.

[8] J. Cortadella, "Task generation and compile time scheduling for mixed data-control embedded software". In *Proceedings of the Design Automation Conference*, June 2000.

[9] D. Desmet. "Operating system based software generation for system-on-chip". In *Proceedings of the Design Automation Conference*, June 2000.

[10] Haobo Yu, Rainer Doemer, and Daniel Gajski. "Automatic Software Generation for System Level Design", CECS Technical Report 03-18, May 14, 2003.

[11] F. Balarin "*Hardware-Software Co-design of Embedded Systems – The POLIS approach*". Kluwer Academic Publishers, January 1997.

[12] L. Gauthier. "Automatic generation and targeting of application-specific operating systems and embedded systems software", *IEEE Trans. on CAD*, November 2001.

[13] Daniel Gajski and Samar Abdi. "Automatic Generation of Equivalent Architecture Model from Functional Specification". DAC 2004, June 7–11, 2004, San Diego, California, USA.

[14] http://www.cyberworkbench.com.

[15] R. Leupers. "Code Optimization Techniques for EmbeddedProcessors – Methods, Algorithms, and Tools". Kluwer Academic Publishers, Nov. 2000.

[16] X. Chen, H. Hsieh, F. Balarin, and Y. Watanabe. "Case studies of model checking for embedded system designs". In Third International Conference on Application of Concurrency to System Design, pages 20–28, June 2003.

[17] A. Knapp and S. Merz, "Model checking and code generation for UML state machines and collaborations", Proc. 5th Workshop on Tools for System Design and Verification, Reisenburg, Germany, 2002, 59-64.

[18] B. Lin. "Software synthesis of process-based concurrent programs". In Proceedings of the Design Automation Conference, 1998.