A New hybrid approach of Embedded Application Validation

Mostefa Belarbi
Ibn Khaldoun University of Tiaret – Algeria.
mbelarbi@mail.univ-tiaret.dz

Abstract-- **In this paper, we suggest certain approach of validation of an embedded application. The presented methodology is divided on several layers, we can transit from one level to another level by refining. The first level represents abstraction associated to the application and the lowest level concerns FPGA implementation. The methodology is illustrated by an application based on GMM model and its hardware implementation.**

Index Terms-Embedded application, Refinement, Verification, Invariant, B formalism, hardware implementation.

1 Introduction

It seems relevant and important to study properties of design levels of embedded application in formal context. This context allows to:

- work in independent manner from specific design methods and actors of these designs.
- perform execution time analysis for particular target.
- investigate verification interface between software and hardware, this context is relevant when we have to deal with deep problems of embedded problems industries.

In this paper, we propose an approach which enables us to validate the implementation of embedded systems from a refinement point of view. Implementation validation is necessary when using the V cycle or when we have to add new elements (functions) to an existing system (notion of incrementality) [13][**15**].
In approaches like UML, SDL[1], the system is defined independently from the hardware implementation. In addition it is necessary to validate the implementation generated by taking into account the hardware properties. The validation methods coupled to these approaches have used analytic methods and heuristics [17][21][16][10]. Our approach of validation can be considered as complementary to the pervious methods because we deal with implementation directly.
Our method concerns the horizontal and vertical validation between design layers. We refer us to formal method like VDM, B**[**6**][**2**],** communicating timed automata [1][5][4][09**]** to structure embedded systems design process. We qualify our method as novel because we associate a certain analatical mathematic representation with formal computer science formalisms and there fore we try to fill gap between two approaches.

This paper presents an approach in order to validate an implementation of an embedded application. The approach is hierarchical and allows extracting models for each level of design process. The transition from one level to another level is realised by abstracting (omitting) some details of the concerned level. There fore we need some criterion to validate the transformation.

The extraction method of models is divided on two levels : The first level of modelling consists of establishing set of equations representing probabilistic model based in this paper on GMM model (Gaussian Mixture Models**)** [14]. The second level consists of hardware part : in our case it is represented by FPGA architecture [14**][**8**].**

The case study GMM provides good performances and interesting properties as classifier, there fore dealing with its real-time characteristics[Mam] must improve its applicability.

This paper in divided on several sections, the section two presents the general method used during embedded application design. The section three presents the based proof formalism used to validate applications stages and give definitions of refinement notion. The section four presents case study which illustrates the new point of view of refinement methodology. We conclude by some observations which give future directions of the present work.

2 Hardware Design

The design process of a digital FPGA consists of the following stages : logic design and simulation, placement and routing, design rules check and finally prototype production.

The design of IC is divided in two parts called “HDL software development” and “hardware prototyping”. The HDL software development includes two stages : behavioural stage and structural stage. The hardeware prototyping is a physical stage.

Behavioural stage: in the behavioural stage, the studied applications are fist written at system level using an HDL. The most common used HDL’s are Verilog or Very high speed integrated circuit hardware description language (VHDL). VHDL allows the possibility of modelling the application

behaviour, regardless of target technology and implementation. VHDL is not suited for modelling or describing power or analog elements, where continuous time is required. MATLAB/Simulink tool [ ] is used for the modelling and simulation of control algorithms.
Power system blockset are used for the modelling and simulation of power system. The same models will be reused all over this methodology to check the IC functionality.
After functional validation at system level description, integrated control block properties must be studied and chosen by the designer. Thus, data binary format (word length, fixed or floating point), mathematical and numerical calculation methods (mathematical operators, integration) or quantification effects due to used binary format, must be defined. These choices are based on accuracy of estimated digital values. These parameters must be also defined to minimize digital estimation errors, which could decrease control performances. At the end of this behavioural stage, all digital properties must be defined, after functional validation by mixed (analog-digital) simulations.

Structural stage: at this stage, the application must be partitioned into functional blocks. Each block is described into a lower abstraction level as in precedent stage, taking into account the digital properties previously defined. The description of the digital IC is closer to hardware implementation. The final goal of this stage is to obtain a Register Transfer Logic (RTL) model, which must be checked against by mixed simulation, reusing high level analog MATLAB/Simulink models written in behavioural stage for power analog elements. The major results of this stage are, first, the functionality of the VHDL described application and secondly, the additional information, such as length of the operations in term of clock cycles number. The final RTL digital model must be written according to the synthesis aspect, optimizing hardware resources and timing of the final digital IC.

Physical stage: The RTL, VHDL model obtained at the end of structural stage, is independent of the target technology. The rapid hardware prototyping used method changes depending on the aim of the final implementation. The evolution of FPGA in term of integration capabilities, low costs and re-programmability using static SRAM technology leads the designer to think of FPGA prototyping. For theses reasons, to reduce lead time and initial prototyping cost, we propose in methodology to implement and validate, first, on an experimental test bench, a FPGA prototype, programmed with the RTL final description of the structural stage. Another interest of FPGA prototyping is component re-programability.

3 Proof-based development based on B formalism.

3.1 Event-based modelling

Event-driven approach [2*,7*] is based on the B notation [4]. A formal model is characterized by a (finite) list x of state variables possibly modified by a (finite) list of events, an invariant I(x) states some properties that must always be satisfied by the variables x and maintained by the activation of the events.

Definition 1 : Generalised substitution
Generalised substitutions are borrowed from B notation. They provide a way to express the transformations of the values of the state of a formal model. In its simple form, x:= E(x), a generalised substitution looks like an assignment statement. In this construct, x denotes a vector built on the set of state variables of the model, and E(x) a vector of expressions of the same size as the vector x. The interpretation we shall give here to this statement is not however that of an assignment statement. We interpret it as a logical simultaneous substitution of each variable of the vector x by the corresponding expression of the vector E(x). There exists a more general form of generalised substitution. It is denoted by the construct x : $P(x_0,x)$. This is to be read ;" x is modified in such a way that the predicate $P(x_0,x)$ holds", where x denotes the new value of the vector whereas x_0 denotes its old value.

Definition 2 Events and BeforeAfter predicates
An event is essentially made of two parts : a guard, which is a predicate built on the state variables, and an action, which is a generalised substitution. An event can take one of the forms shown in the table below. In these constructs, evt is an identifier : this is the event name. The first event is not guarded: it is thus always enabled. The guard of the other events, which states the necessary condition for these events to occur, is represented by G(x) in the second case, and by ∃ t. G(t,x) in the third one. The latter defines a non-deterministic event where t represents a vector of distinct local variables. The, so-called, before-after predicate B A(x,x') associated with each event shape, describes the event as a logical predicate expressing the relationship linking the values of the state variables just before(x) and just after(x') the event "execution".

Event	**Before-after Predicate B A(x,x')**
Evt = **begin** x : P(x0,x) **end**	P(x,x')
Evt = **select** G(x) **then** x : Q(x0,x) **end**	G(x) ∧ Q(x,x')
Evt = **any t where** G(t,x) **then** x :	∃ t (G(t,x) ∧ R(x,x',t))

R(x0,x,t) **end**	

Proof obligations are produced from events in order to state that the invariant condition I(x) is preserved. We next give the general rule to be proved. It follows immediately from the very definition of the before-after predicate, B A(x,x') of each event :

$I(x) \wedge B\ A(x,x') \Rightarrow I(x')$

3.2 Model refinement

The refinement of a formal model allows us to enrich a model in a step by step approach. Refinement provides a way to construct stronger invariants and also to add details in a model. It is used to transform an abstract model in a more concrete version by modifying the state description. This is essentially done by extending the list of state variables (possibly suppression some of them), by refining each abstract event into a corresponding concrete version, and by adding new events. The abstract state variables, x, and the concrete ones, y, are linked together by means of a, so-called, gluing invariant J(x,y). A number of proof obligations ensure that (1) each abstract event is correctly refined by its corresponding concrete version, (2) each new event refines skip, (3) no new event take control for ever, and (4) relative deadlock freeness is preserved.

Definition 3 : Refinement

We suppose that an abstract model AM with variables x and invariant I(x) is refined by concrete model CM with variables y and gluing invariant J(x,y). If B AA(x,x') and B AC(x,x') are respectively the abstract and concret before-after predicates of the same event, we have to prove the following statement :

$I(x) \wedge J(x,y) \wedge B\ AC(y,y') \Rightarrow \exists\ x'.\ (B\ AA(x,x') \wedge J(x',y'))$

This says that under the abstract invariant I(x) and the concret one J(x,y), a concrete step B AC(y,y') can be simulated ($\exists$ x') by an abstract one BAA(x,x') in such a way that gluing invariant J(x',y') is preserved. A new event with before-after predicate B A(y,y') must refine skip (x'=x). This leads to the following statement to prove:

$I(x) \wedge J(x,y) \wedge B\ A(y,y') \Rightarrow J(x,y')$

Moreover, we must prove that a variant V(y) is decreased by each new event (this is to guarantee that an abstract step may occur). We have thus to prove the following for each new event with befire-after predicate B A (y,y') :

$I(x) \wedge J(x,y) \wedge B\ A(y,y') \Rightarrow V(y') < V(y)$

Finally, we must prove that the concrete model does not introduce more deadlocks than the abstract one. This is formalized by means of the following proof obligation :

$I(x) \wedge J(x,y) \wedge grds(AM) \Rightarrow grds(C\ M)$

Where grds(AM) stands for the disjunction of the guards of the events of the abstract model, and grds(C M) stands for the disjunction of the guards of the events of the concrete one.

4 Incremental development of Gaussian Mixture Models Classifier

In this section, we present several models of the GMM case-study. The case study used in this paper is the task of a pattern recognition algorithm which consists of setting a decision rule, which optimally partitions the data space into c regions, one for each class Ck. A pattern classifier generates a class label for an unknown feature vector $x \in R^d$ from a discrete set of previously learned classes. The Gaussian mixture models (GMM) classifier is used to illustrate the validation methodology presented in this paper.

4.1 First model

GMM is classified as a semi-parametric density estimation method since it defines a general class of functional forms for the density model. In this mixture model, a probability density function is expressed as a linear combination of basis functions. The interesting property of GMM is that the training procedure is done independently for each class by constructing a Gaussian mixture of a given class. In GMM, a classifier is constructed by evaluating the posterior probability of an unknown input pattern x belonging to a given class Ck expressed as $\varphi(C_k|x)$

$\varphi(C_k|x) = \varphi(C_k)\ \varphi(x|C_k) / \varphi(x)$

Where $\varphi(C_k)$ is the frequency of a given training sample in the data-set and the unconditional density $\varphi(x)$ is given by

$\varphi(x) = \sum \varphi(x|Ck)\ \varphi(Ck).$

The class conditional densities $\varphi(x|Ck)$ can expressed by

$\varphi(x|Ck) = \sum j = 1 \text{ to } M\ \varphi(j)\ \varphi(x|j)$

Where $\varphi(j)$ are the mixing coefficients of the component density functions $\varphi(x|j)$. Each mixture component is defined by a Gaussian parametric distribution in d dimensional space

$$\varphi(x|j) = \exp\{ -1/2(x-\mu j)^T \Sigma j^{-1} (x-\mu j) \} / (2\pi)^{d/2} |\Sigma j|^{1/2}$$

The parameters to be estimated are the mixing coefficients $\varphi(j)$, the covariance matrix Σj, and the mean vector μj.

4.2 The second model

One important task when considering GMM hardware complexity is to estimate the number of memory units storing the parameters of the

Gaussian models. In this context new set of parameters : constant Kj and a triangular matrix Gj are defined and used instead of $\varphi(C_k)$, $\varphi(j)$, $|\sum_j| ½$ and $|\sum_j|^{-1}$

The new coefficients Kj and Gj are given by :

$K_i = \varphi(C_k)\ \varphi(j) /(2\pi)^{d/2} |\sum_j| ½$

$G_j^T G_j = ½ \sum_j^{-1}$

Gj is a triangular matrix introduced in order to reduce the complexity.
Consequently , we suppose that :
$z_j = [(x - \mu_j)^T G_j] [(x - \mu_j)^T G_j]^T$

the equation $\varphi(x|C_k)\ \varphi(C_k) = \varphi(C_k) \sum j=1 \text{ to } M\ \varphi(j)\ \varphi(x|j)$

$\varphi(x|C_k)\ \varphi(C_k) = \sum_{j=1..M} K_j \exp\{-z_j\}$ (*)

Thus the calculation of (*) can be divided into three steps : evaluation of parameters zj, the exponential calculation, and the multiplication with constant Kj.

4.3 The third model

The GMM processor includes a serial parallel

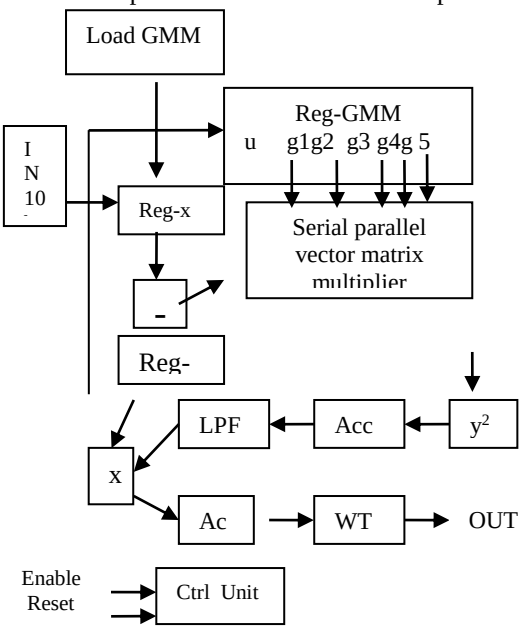

Fig. 1 GMM processor

vector matrix multiplier, a square and multiplier units, a LPF unit, a winner-takes-all (WTA) circuit, and two accumulators. A 10-bit data bus is used in order to load the GMM parameters and the test vector data using the load signals (Fig.1).

State processor = Enable | Reset

Serial parallel vector matrix multiplier = (-, Reg-GMM, Reg-x)
Reg-x = (input, bus-10 bits)
Reg-GMM = (load, GMM)
Reg-k = (input, bus-10 bits)
Y2 = (affectation, SPVMM)
Acc1 = (affectation, Y2)
LPF = (affectation, Acc)
Acc2 = (x, LPF, Reg-k)
WTA = (Affectation, Acc2)
OUTPUT = (affectation, WTA)

4.4 The fourth model

One the major problems when implementing the GMM classifier is related to the complexity of the exponential function calculation. To approximate the expentional function using linear piecewise function LPF. The process of building the classifier using LPF can be divided into two major phases : the training and the test procedure. The parameters of the classifier are selected during the training phase. We first obtain the parameters of the mixture models such as μ and $\sum$ based on the training data-set. The same training data are fed on the classifier in which the exponential function is replaced by LPF, while the other parameters of the Gaussian model such as the mean vector and the covariance matrix are kept unchanged. Next, the parameters of LPF are optimised such that the mismatch between decision boundaries in the LPF-based GMM and the original GMM is minimized.

5 Conclusion

The present paper suggests new approach of incremental validation of embedded application. The notion of refining is explored comparing to its initial definition used in formal methods like VDM, B formalism, etc….
We suggest to combine these new approaches in order to deal with functional et temporal properties to be verified on embedded application. Combining formalism like B or IF2C provide us the possibility to deal horizontal and vertical during design-implementation process of embedded application.

6 References

[1] R. Allur, D.L. Dill, “A theory of timed automata”, *Theoretical Computer Science*. 1994, Vol. 126, pp. 183-235.

[2] J.-R. Abrial. The B book – Assigning Programs to Meanings. Cambridge University Press, 1996.

[3] .A.-B. Alkhodre, J.-P Babau, J.-J Schwarz : Modelling of real-time constraints using SDL for embedded systems design. In IEE Control and Computing System, London, August,2002.

[4] M.belarbi, J.-P.Babau and J.-J.Schwarz : “Temporal Verification of Real-Time Multitasking Application Properties Based on Communicating Timed Automata”. The eighth IEEE Symposium on Distributed Simulation

and Real-Time Applications. DS-RT'04, Budapest, October, 21-23, 2004.

[5] M. Bozga, S. Graf and L. Mounier, "Automated validation of distributed software using the IF environment", IEEE International Symposium on Network Computing and Applications (NCA 2001), Electronic Notes in Theoritical Computer Science 55, N° 3, Cambridge, 08-10 October, 2001

[6] D. Cansell, C. Tanougast and Y. Berviller : " Integration of the proof process in the design of a microelectronic architecture for Bitrate measurement instrumentation of transport stream program MPEG-2 DVB-T. April 11, 2003.

[7] G. Durrieu, O. Laurent, C. Seguin and V. Wiels : Automatic test case generation for critical embedded systems. DASIA 2004 (Data Systems in Aerospace) June 28-30 2004, Nice, France.

[8] S. Kumar, G.T Ram Das and V. Subrahmanyam : " VLSI Design methodology for hardware prototyping of integrated direct torque control of induction motor drieves. International Journal of Electrics and Power engineering 2(4) 262-270, 2008.

[9] K.G. Larsen, P. Pettersson, and W. Yi: «UPPAAL in a Nutshell », in Journal on Software Tools for Technology Transfer, 1(1-2): pp 134-152, October 1997.

[10] J.W.Layland : Scheduling algorithms for multiprogramming in a hard real time environment. Association for Computing Machinary, 20(1), 46-61(1973).

[11] Z. Mammeri : Expression et dérivation des contraintes temporelles dans les applications temps réel. APII-JESA. 1998, Vol.32, N° 5-6, p. 609 à 644.

[12] R. Riemenschneider : A simplified method for establishing the correctness of architectural refinements. SRI, Working paper. http://www.csl.sri.com/papers/simplified.

[13] P. Pop, P. Eles and Z. Peng : "Schedulability-Driven communication synthesis for time triggered embedded systems". Real-Time Systems, 26, 297-325, 2004.

[14] M. SHI and A. BERMARK : " An efficient digital VLSI implementation of gaussian mixture models-based classifier". IEEE Transactions on very large integration (VLSI) systems, Vol. 14, N°. 9, September 2006.

[15] W. Zheng, J. Chong, C. Pinello, S. Kanajan and A. Sangiovanni-Vincentelli : " Extensible and Scalable time triggered scheduling ". ACSD 2005: 132-141. Fifth Int. Conf. on App. System Design. 6-9 June 2005, St Malo, France.