J. Software Engineering & Applications, 2010, 3, 525-535
doi:10.4236/jsea.2010.36060 Published Online June 2010 (http://www.SciRP.org/journal/jsea)

New Approach for Hardware/Software Embedded System Conception Based on the Use of Design Patterns

Yassine Manai, Joseph Haggège, Mohamed Benrejeb

LA.R.A, Ecole Nationale d'Ingénieurs de Tunis, Tunis, Tunisie.
Email: yacine.manai@gmail.com, {joseph.haggege, mohamed.benrejeb}@enit.rnu.tn

Received February 5th, 2010; revised April 25th, 2010; accepted April 27th, 2010.

ABSTRACT

This paper deals with a new hardware/software embedded system design methodology based on design pattern approach by development of a new design tool called smartcell. Three main constraints of embedded systems design process are investigated: *the complexity, the partitioning between hardware and software aspects and the reusability. Two intermediate models are carried out in order to solve the complexity problem. The partitioning problem deals with the proposed hardware/software partitioning algorithm based on Ant Colony Optimisation. The reusability problem is resolved by synthesis of intellectual property blocks. Specification and integration of an intelligent controller on heterogeneous platform are considered to illustrate the proposed approach.*

Keywords: *Embedded Systems, Design Patterns, Smartcell, Hardware/Software Partitioning, Intellectual Property*

1. Introduction

There are two main orientations in embedded system research, the technological field and the methodological one [1]. The first is characterized by the increasing revolution in integration, the second tries to develop the embedded system design process by examining new design tools in order to front the complexity of embedded systems. There are three main problems during system design: the complexity, the hardware/software (HW/SW) partitioning and the reusability.

To simplify the design process, designers are recurring to raise the abstraction level, from Register Transfer Level (RTL) to system level. As a consequence, a gap between application development and architecture synthesis appears. In order to solve this problem, many frameworks are developed like transactional environments between application development and architecture synthesis [2,3], or many design tools are developed in order to improve embedded system performances [4,5]. In the domain of control system processor implementation, architecture and design framework for processor, solutions have been developed for linear time invariant (LTI) control and embedded real time control applications [6-8]. In [9], a design methodology based on a transactional model which is inserted between the application and the architecture is presented. In this way, the application is refined in an intermediate level which contains the architecture parameters. From this level, the implementation step is achieved in order to generate the RTL architecture.

Our contribution to resolve the complexity problem consists to develop two intermediate environments in order to minimize the gap between application development and architecture synthesis.

The second problem is the hardware/software partitioning. The HW/SW co-design is evolved in a way to automate all phases of design flow coming from physical phase to design one passing through the HW/SW partitioning and synthesis phases [10]. Our contribution to resolve HW/SW partitioning problem, based on ant colony algorithm development, presented in [11]. The work of [12] considers the hardware/software partitioning problem of the embedded system design of reconfigurable architecture. An automatic hardware/software partitioning methodology is proposed in order to develop the dynamically reconfigurable architecture. First, the system specification is developed with the *SyncChart* formalism based on the *Esterel* language. Next, the proposed partitioning method is applied, and the generated (C, Java) code is implemented on the heterogeneous target. To give a reusable solution of hardware/software partition-

ing, this paper presents a solution based on *Composite* design pattern development.

The third problem is the reusability in design process. Design patterns [13] have been operated in order to develop reusable design tools in different engineering fields. Many researches in this field are performed. The work of [14] developed an object analysis pattern for embedded system, further; a requirement pattern with design pattern approach was developed in [15]. A wrapper design pattern for adapting the behaviour of the soft IPs was proposed in [16]. The reusability of Intellectual Property (IP) blocks have been performed extensively for design hardware applications and IP blocks synthesis [16-18]. The development of IP blocks based on design pattern use *Unified Modelling Language* (UML) as specification language, [19] present design pattern modelling in UML. Many researches are performed for the reusability problem in order to develop new design tools that encapsulate all co-design phases in order to implement intellectual property (IP) blocks. One attempt proposed in [20,21] have as aim to develop the *smartcell* design tools in order to implement HW and SW IP blocks for heterogeneous platforms. This *smartcell* is developed with design pattern approach and oriented-object concept based on UML language. Our contribution to resolve the reusability problem consists in the synthesis of IP blocks for hardware and software solutions from direct acyclic graph (DAC). The proposed approach examines the *Builder* design pattern to produce IP blocks.

The remainder of this paper is organized as follows. Section 2 introduces the proposed hardware/software approach for embedded system design. A case study is discussed in next section which validates the proposed approach by design of induction motor controller system. The conclusion and the future works are presented in the last section of this paper.

2. The Proposed Hardware/Sotware Approach

2.1 Requirements of Proposed Approach

Three main problems are targeted by this paper: the first concerns the complexity mastering of embedded system; our contribution is to raise the abstraction level by investigating an object-oriented approach with the design pattern concept. The second is the reusability of IP blocks in order to minimize the time-to-market. Finally, the hardware-software partitioning is solved with a proposed algorithm, based on ant colony optimisation, in order to optimise task's deadline of a direct acyclic graph that models the embedded system. Further, this paper demonstrates the use of a design pattern concept for all phases in design flow.

The proposed hardware/software embedded system design process is presented in **Figure 1**. The proposed co-design flow operates in two levels, the system level and the smartcell one. First step consists to decompose the embedded system in a set of subsystems. Each subsystem is developed in the smartcell level.

The proposed approach considers the smartcell as a design agent that encapsulates the design process composed by specification, application development, architecture synthesis, the HW/SW partitioning, integration and validation phase. In the system level, we model the embedded system by the "smartcell system level", which have the following actions: the decomposition of the main system into subsystems, HW/SW partitioning process, the integration and the global validation of the main system.

In the second level of abstraction, each subsystem is modelled with a smartcell which have the following steps: the application development, the architecture synthesis, and the hardware/software partitioning.

2.2 Complexity Problem

The first step for smartcell system level is the decomposition of the system into a set of subsystems. We develop the design pattern smartcell Factory in order to do this. The decomposition's automation is guaranteed with this design pattern. Its intent is to allow an interface for creating a family of dependent objects without need to specify their concrete classes. The global system is decomposed into four subsystems, the input, the output, the physical subsystem and the controller one. Each one of these subsystems is managed by a smartcell (e.g., SCell_ Input in **Figure 2**).

We define the following actions: the application development, the architecture synthesis the communication management and the HW/SW partitioning. We define for each action an actor modelled with a class diagram in UML. Each actor has four missions corresponding to its smartcell.

Figure 2 presents the smartcell Factory. To implement each subsystem, the design pattern Factory_Method allows making use of the subsystem structure. It is named also virtual constructor and it defines an interface for creating an object instantiated from subclasses (concrete classes) [13]. The design pattern combined with abstract factory in order to decompose the global system into a class of subsystems.

2.2.1 Application Development

The application development is composed of two phases, the application modelled and the direct acyclic graph (DAG) development. First, the application model is developed with state space approach in order to extract the main block of the disturbances blocks. Second, the DAG is developed with the proposed MAC_Builder environment which builds application's graph. The application

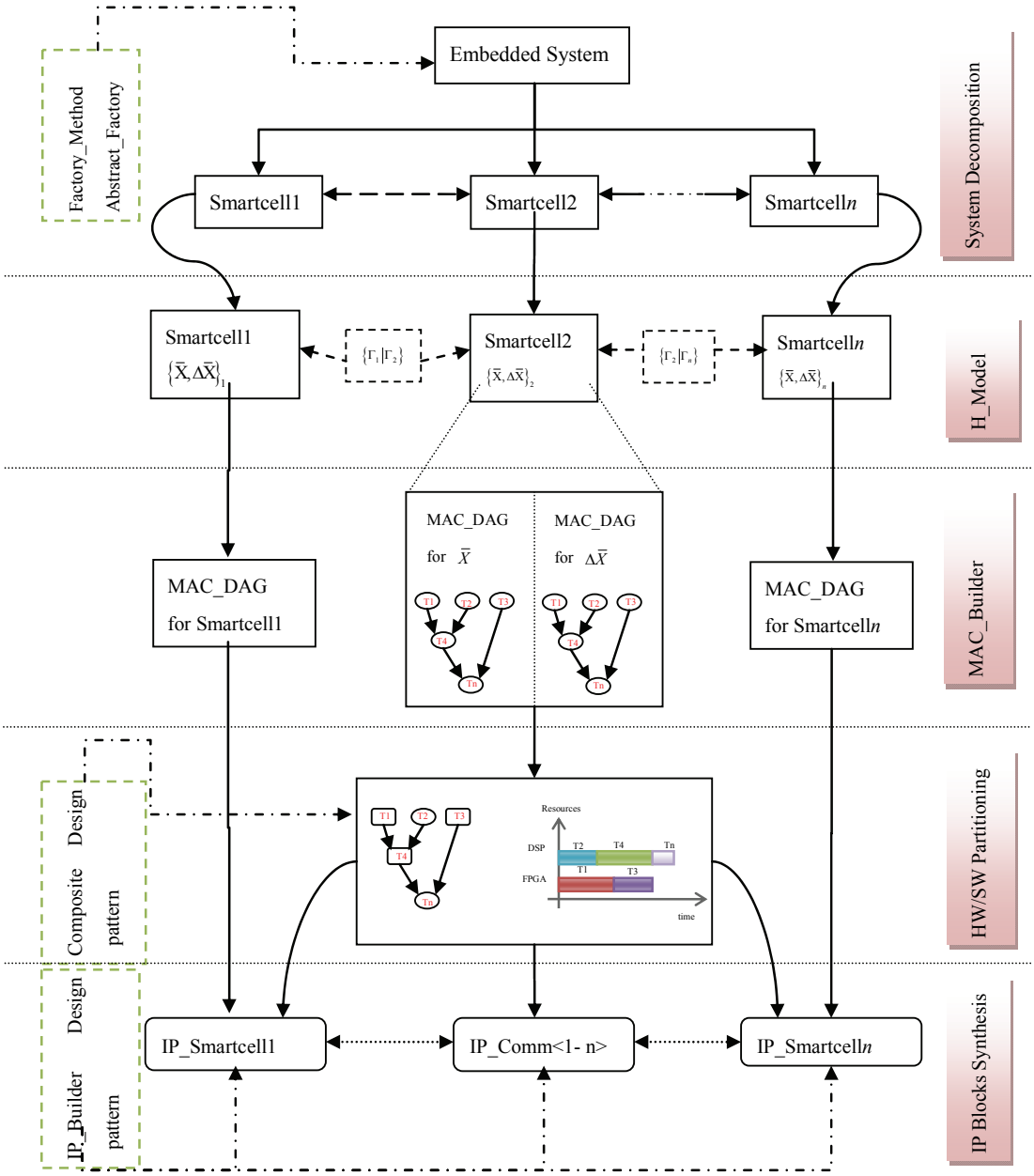

Figure 1. Proposed design process

development consists to following functionalities:

- the analytic model development,
- the MAC_Builder development.

1) The Analytic Model H

In this section, we present the analytic model corresponding to smartcell design pattern. This model allows developing the mathematical representation of subsystems with state space approach in order to characterise the corresponding subsystem.

The proposed analytic model H encapsulates the necessary information in order to carry out the smartcell. This model is a hybrid model that comports heterogeneous elements, presented by the Equation (1).

$$H = \{\bar{X}, \Delta\bar{X}, \Gamma, Y\} \quad (1)$$

where $\bar{X}$ is the nominal system model, $\Delta\bar{X}$ is the distur-bances model, $Y = \bigcup_{i=0}^{n} y_i$: the monitoring system, $\Gamma = \bigcup_{i=0}^{n} f_i$: the communication protocol system.

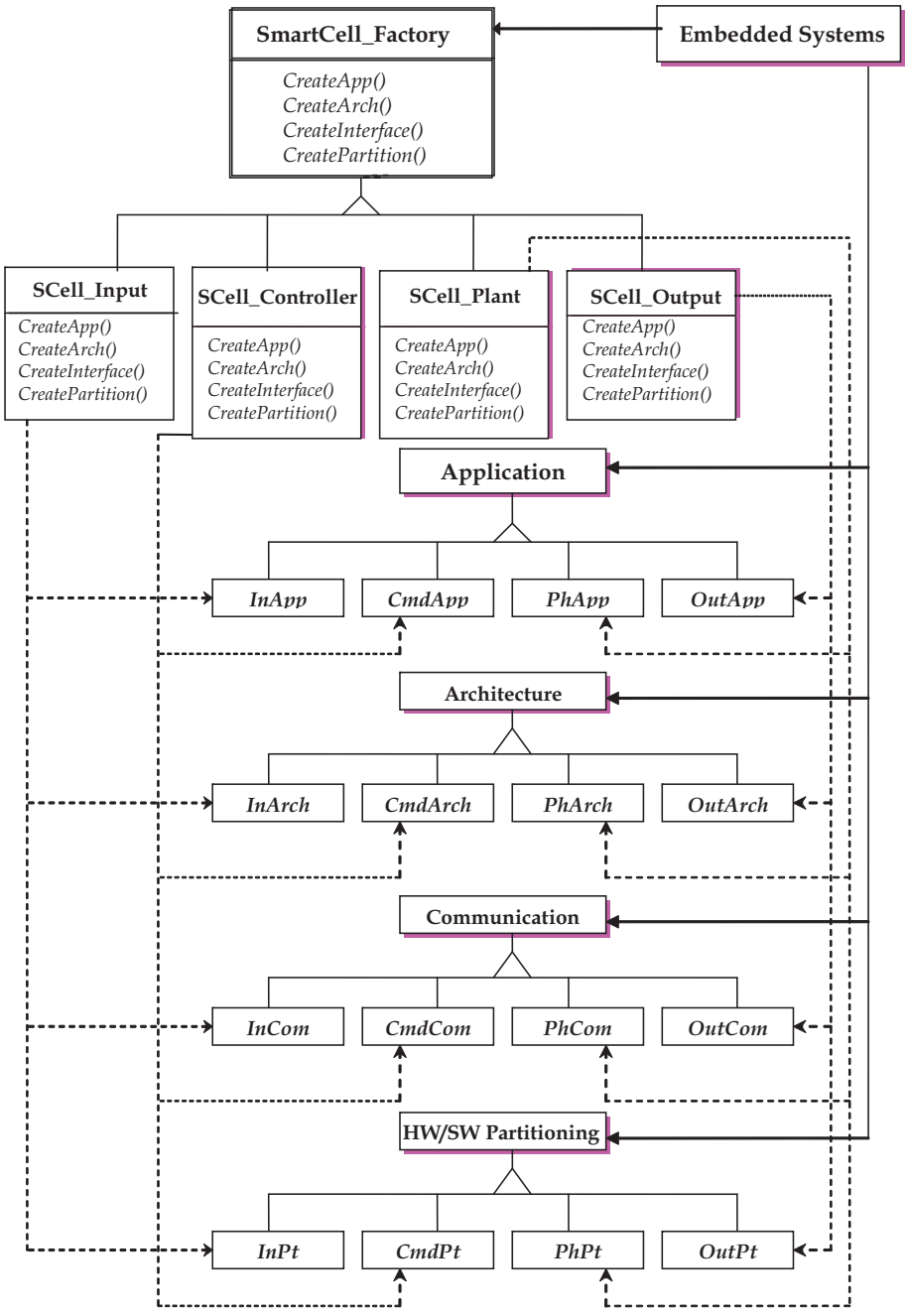

Figure 2. System decomposition with smartcell factory

In this equation, $\overline{X}$ encapsulate the system model described in state space in addition to state vector, input vector and output vector. The $\Delta\overline{X}$ function represents the disturbances applied on the system. This function is represented with sensitivity functions in order to model the disturbances.

The communication protocols are encapsulated with the Γ function, and the fault-handler control laws to be integrated in target architecture are encapsulated with Y function. Three phases must be distinguished for analytical model H; first, we elaborate the model with transfer function of smartcell. Next, we transform each transfer function in the state space using of *compagnon* form. Third phase consists in determination of the smartcell with delta representation.

The Strategy design pattern is developed in order to simplify the control law coding, and the choice of the correspondent control law for application. The control

law chooses is carried out through activation of *ControlLaw*() function of *Model* class. This function activates the *ControlLaw*() function of AbstractLaw class. Considering a linear system defined by Equation (2),

$$\begin{bmatrix} Y(P) \\ U(P) \end{bmatrix} = \begin{bmatrix} FT_{11} & FT_{12} & FT_{13} & FT_{14} \\ FT_{21} & FT_{22} & FT_{23} & FT_{24} \end{bmatrix} \times \begin{bmatrix} R(P) \\ D_i(P) \\ D_o(P) \\ D_m(p) \end{bmatrix} \quad (2)$$

we define an operator that allow extracting the i^{th} column of matrix (FT), we note this operator $X(.)$. To extract the main part of system, *i.e.* the output vector $\mathrm{Y}(P)$ and the control vector $\mathrm{U}(P)$ each one in function of reference vector $R(P)$, we apply the operator $X(.)$ to first column of equation, and we obtain:

$$X(1)(FT) = \begin{bmatrix} FT_{11} & 0 & 0 & 0 \\ FT_{21} & 0 & 0 & 0 \end{bmatrix} \quad (3)$$

Otherwise, the $X(.)$ operator allows extracting the disturbance information, like input or output system disturbance, the $\Delta\overline{X}$ function of H model, is given with $X(.)$ operator as follow,

$$\Delta\overline{X} = \{\bigcup_{i=2}^{4} x(i)(FT)\}.\begin{bmatrix} R(P) \\ D_i(p) \\ D_o(P) \\ D_m(p) \end{bmatrix} \quad (4)$$

where $$\left\{\bigcup_{i=2}^{4} \chi(i)(FT)\right\} = \begin{bmatrix} 0 & \Delta\overline{X}_{12} & \Delta\overline{X}_{13} & \Delta\overline{X}_{14} \\ 0 & \Delta\overline{X}_{22} & \Delta\overline{X}_{23} & \Delta\overline{X}_{24} \end{bmatrix}$$

The second phase of H model is the transformation of transfer functions in state space. The main part of the system is given by Equation (5):

$$\overline{X} = \begin{bmatrix} Y(p) \\ U(p) \end{bmatrix} = \begin{bmatrix} \overline{X}_{11} \\ \overline{X}_{21} \end{bmatrix}\left[R(p)\right] = \frac{\begin{bmatrix} G(p)C(p) \\ C(p) \end{bmatrix}}{1+G(p)C(p)} \quad (5)$$

Each component of $\overline{X}$ vector is described with state space approach, as follow:

$$\overline{X}_{ij} = \begin{bmatrix} X_{ij}[k+1] \\ y_{ij}[k] \end{bmatrix} = \begin{bmatrix} A_{ij} & B_{ij} \\ C_{ij} & D_{ij} \end{bmatrix}\begin{bmatrix} X_{ij}[k] \\ U_{ij}[k] \end{bmatrix}; i = 1,2 \text{ and } j = 1 \quad (6)$$

where, the matrix A, B, C and D are given with a canonical representation like *compagnon* form. To model the disturbances parts of smartcell, we compute each $\Delta\overline{X}_{ij}$ vector as presented in Equation (7):

$$\Delta\overline{X}_{ij} = \begin{bmatrix} X_{ij}[k+1] \\ y_{ij}[k] \end{bmatrix} = \begin{bmatrix} A_{ij} & B_{ij} \\ C_{ij} & D_{ij} \end{bmatrix}\begin{bmatrix} X_{ij}[k] \\ U_{ij}[k] \end{bmatrix}; \quad (7)$$
$$i = 1,2 \text{ and } j = 2,\ldots,4$$

Then, we can develop the discrete model with delta operator, defined by Equation (8):

$$\delta(f(t)) \equiv \delta f[k] = f(k+1) - f(k) \quad (8)$$

Through delta representation of system, we can develop the reccurent equations as describe by Equation (9).

$$y[k] = c_1x_1[k] + c_1x_1[k] + \cdots + c_nx_n[k] + c_uu[k] \quad (9)$$

where,

$$\begin{aligned} x_t &= x_1 + x_2 + \cdots + x_n \\ x_1[k+1] &= x_1[k] + a_1x_2[k] \\ x_2[k+1] &= x_2[k] + a_2x_3[k] \\ &\vdots \\ x_{n\text{-}1}[k+1] &= x_{n\text{-}1}[k] + a_{n\text{-}1}x_n[k] \\ x_n[k+1] &= x_n[k] - a_nx_t + a_nu[k] \end{aligned}$$

2) The MAC Builder Model

An embedded system is modelled with a set of task graphs. Each task graph is composed by a set of nodes each one representing a task, and a set of edges that links between nodes. Each task can be implemented with software IP or hardware IP. An important property that characterizes the task graph building is the node granularity. There are three categories of granularity: the fine, the gross and the variable granularity. This paper introduces a new approach to building a task graph, that model an embedded system, based on a MAC_Operation granularity.

The MAC operations are composed of arithmetic operations, multiply and accumulate. The MAC builder environment consists in building graphs task from recurrent equations given by H model. Consider a recurrent equation; we can transform this in the list of MAC operations, for example, a fourth order linear system can modelled with MAC operation environment as present **Figure 3**. We use 13 MAC units for this system development. T_0 and T_N are fictive tasks, which indicate the start and end point respectively.

After modelling with task graph, the next step consists to realize the tasks partitioning into hardware and software targets. Indeed, the partitioning phase comports two main stages, space allocation and times scheduling.

Consider the fourth order linear system. The scheduling tasks of this system conduct to result presented in **Figure 4**. In this example, the *time execute* of one MAC operation is taken equal to 3 cycles.

The proposed MAC_Builder environment can be used to determine a task graph corresponding to any other type of system. For example, consider a non linear system given by the following equations:

$$y[0] = \sqrt{\frac{2}{3}} \times \left(u[0] \times \cos(i) + u[1] \times \cos(j) + u[2] \times \cos(k)\right);$$

$$y[1]=\sqrt{\frac{2}{3}}\times\left(-u[0]\times\sin(i)-u[1]\times\sin(j)-u[2]\times\sin(k)\right);$$

This system uses sinusoidal functions. In order to implement these functions, we develop new operations called MAC_cos and MAC_sin.

Consider the "cos" function development for example. First, we determine the approximation of "cos" by a polynomial of degree 12 on [0, π/4],

$$\cos x \approx 1-\frac{x^2}{2}+C_1x^4+C_2x^6+C_3x^8+C_4x^{10}+C_5x^{12} \quad \text{where,}$$

$C_i, i=\{1,\ldots,5\}$ are the given constants. The proposed task graph of "cos" function is given in **Figure 5**.

The register R is initially loaded by the C5 constant. Six iterations are needed to compute the function "cos". The last example demonstrates how we can apply the proposed MAC_Builder for non linear applications.

For a generic aspect of a proposed approach, a "Composite" design pattern is carried out in order to building a task graph corresponding to this subsystem. The next section presents the hardware/software partitioning by use of this design pattern.

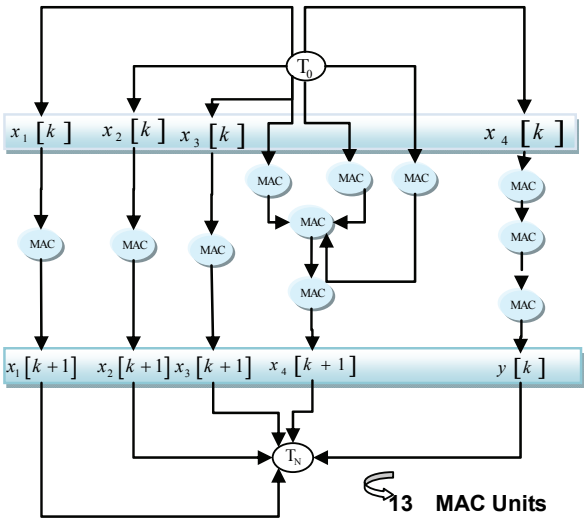

Figure 3. Task graph of four order system

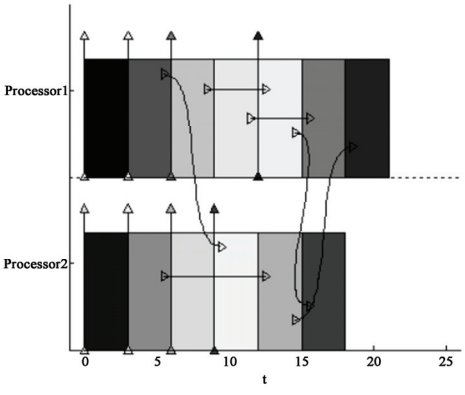

Figure 4. Scheduling in two processors

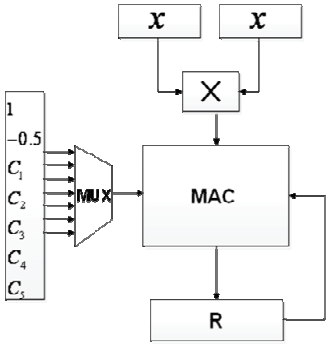

Figure 5. MAC_cos operation

2.3 Hardware/Software Partitioning

The hardware/software partitioning problem consists to respect a deadline of tasks in direct acyclic graph. The optimisation of this factor is function of the parallelism between tasks, and the good management of allocation tasks to hardware and software targets.

The partitioning problem is an NP-complete problem which it hasn't a polynomial resolution algorithm, but we can verify in polynomial time if S is a solution (S is a proposition of resolution).

2.3.1 MAC Operation as an Estimation Unit

An embedded system modelled with a smartcell, can be designed with state space models. This search examines the determination of MAC operation unit as an elementary block to represent a granularity of embedded system. The state vector, for example, can be represented with the MAC operation structure from its recurrent function.

2.3.2 Problem Formulation

Consider an embedded system modelled with a task graph G = {E, V}, E is edges set which rely two nodes and V is a set of nodes. Each node is defined with a start execution date and end of execution date.

An embedded system is a set of smartcells each one is modelled with a state space representation. For each smartcell, state vector is programmed with a recursive functions based on MAC operation.

Each node of task graph has a list of parameters, the time execution in DSP, the time execution in FPGA, and the silicon area. **Figure 6** presents the task graph parameteri-sation. Later on estimation parameters, we apply the proposed algorithm. Each node can be implemented either on DSP board or on FPGA one, then the complexity is equal to 2^n if n is the number of node.

The design pattern Composite is used for hardware/software partitioning problem formulation as a task graph. All successors' tasks are viewed as children tasks in relation to precedent task. The last task is viewed as a leaf by

the Composite design pattern. **Figure 7** present this design pattern.

2.4 IP Blocks Reusability

After the hardware/software partitioning phase, the next step in design process is to synthesise the intellectual property IP blocks. We distinguish two families of IP blocks, the Soft IP and the Hard IP. **Figure 8** presents the

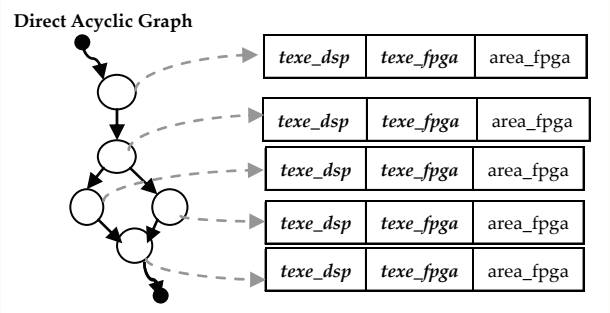

Figure 6. Resources estimation

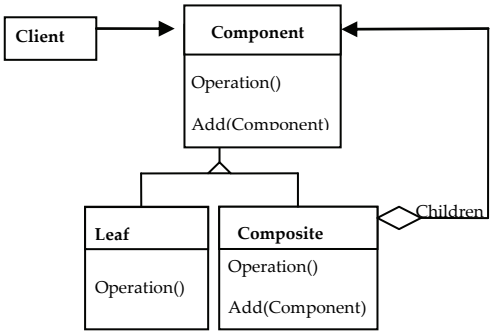

Figure 7. Composite design pattern

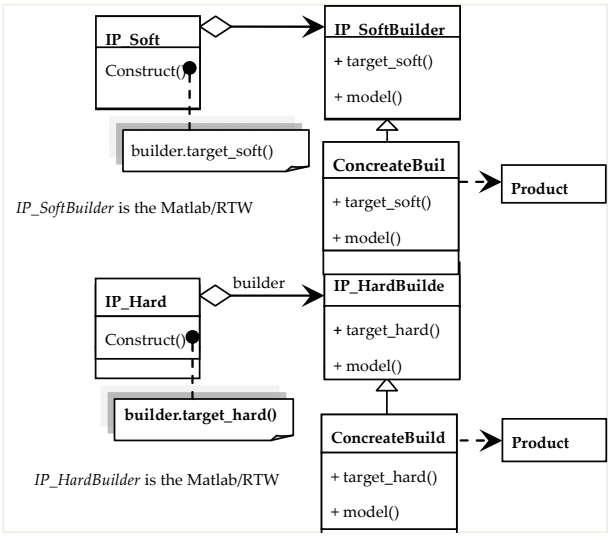

Figure 8. Hardware & software IP blocks

in the development of C/C++ code to be implemented in proposed IP design pattern. The synthesis of soft IP consists software target like DSP. In the other hand, for hard IP, we use the VHDL/Verilog code to be implemented in hardware target like FPGA. Each control law allows generating the C/C++ code in floating or fixed point implementation. This research investigates the development of IP soft and IP hard in order to synthesis the architecture of controller systems implemented in heterogeneous hardware/software target.

2.4.1 The IP Soft Development

The soft IPs blocks reusability in the design process, led us to introduce improvements in the development process of these blocks by investigating the aptitudes of design pattern approach. The IP soft development consists to convert a state space representation into a C/C++ file which can be implemented on software target. The "Builder" design pattern assumes the building of complex object by the specification of its type. The building details are hidden to user. The main motivation to use "Builder" design pattern is to simplify the code generation for building a complex object. The Builder pattern encapsulates the composite objects building, because this action is hard, repetitive and complex.

2.4.2 The IP Hard Development

The hardware synthesis of an application consists in the generation of VHDL/Verilog code to be implemented on target. We investigate two kinds of hardware architecture, FPGA and ASIC circuits. As seen for IP soft development, the IP hard development consist to model a subsystem with the state space approach and coding this model with corresponding hardware language. Each IP hard represent one MAC operation generated with MAC builder environment. We distinguish two kinds of MAC operation implementation, either hardware or software.

3. Case Study

This section presents the design of a control system in such a way that justify how our approach can be applied, in order to implement a hardware/software solution of embedded system by use of IP blocks.

The studied system, given in **Figure 9**, is an induction machine and we intend to implement its speed control system with our proposed approach. Then, we present the development of Hard/Soft IP blocks, and the HW/SW partitioning of this embedded system with *smartcell* design approach.

The induction machine control system is carried out with park transformation technique. Two blocks are developed with S-function, park_dq_abc function, and park_abc_dq fucntion. The variable measurement is carried out with estimator. The dynamic of the study system is presented in **Figure 10**.

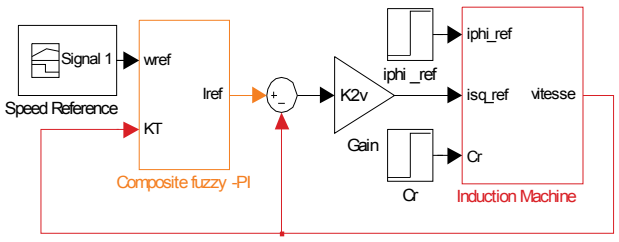

Figure 9. Induction machine control system

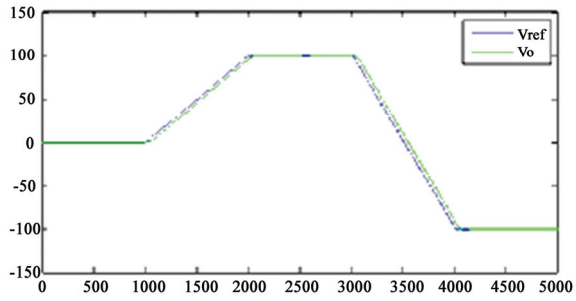

Figure 10. Speed response

3.1 Complexity Problem

We decompose the global system into four subsystems. The first subsystem composed with the input vector that contains the speed reference, the courant reference and the load torque. The control subsystem contains the flow controller, the torque controller and the speed controller. The second subsystem is the induction machine model composed with park transformation modules and induction machine model. The output subsystem contains the output vector, the courant estimator, and the speed estimator.

The control system is modelled as a smartcell in order to apply the proposed approach. First, the system is decomposed into four subsystems presented before with the SmartcellFactory design pattern. The development of task graph that model the embedded system is carried out in two phases, the H_model determination and the MAC_Builder development. From given recurrent equation we develop the task graph correspondent to each subsystem. The synthesis of Hard/Soft IP blocks is developed with VHDL and C/C++ code respectively by the mean of "Builder" design pattern. The HW/SW partitioning is carried out after development of each task graph with "composite" design pattern. The generated C/C++ code is implemented in DSP TMS320F2812 target, whereas the VHDL/Verilog code is integrated in FPGA Spartan 3 target.

The system decomposition is made with a developed abstract factory design pattern, the SmartCellFactory. This design pattern assumes the system decomposition and gives four main missions to each subsystem, the application development, the architecture synthesis, the hardware/software partitioning and the communication management.

Given that in oriented object concept the object creation is based on constructor function, the smartcell tool uses the Factory Method design pattern so that this function supports the heritage management by the mean of virtual property.

Indeed, the smartcell investigates the couple {*Abstract_Factory, Factory_Method*} design patterns in order to assume the decomposition process with oriented object approach. **Listing 1** presents the proposed SmartCellFactory that decomposes the initial system specification into a set of subsystems and presents the control subsystem development.

3.1.1 Analytic Model H of System

Consider the speed control system of induction drive. To extract the system from $\bar{X}$ vector, we apply the $\chi(.)$ operator

$$\bar{X} = \begin{bmatrix} Y(p) \\ U(p) \end{bmatrix} = \begin{bmatrix} \bar{X}_{11} \\ \bar{X}_{21} \end{bmatrix} \left[R(p) \right] \tag{10}$$

The process is modelled with transfer function $G(p) = e^{-\tau p}\bar{G}(p)$, $G(p)$ is the system model

```
/* System decomposition with    SmartCellFactory */


Class SmartCellFactory
{
Public :
      virtual Application CreateApp() const
      {return new Application ;}
      virtual Architecture CreateArch() const
      {return new Architecture;}
       virtual Communication CreateInterface() const
      {return new Communication ;}
      virtual Partitionnement CreatePartition() const
      {return new Partitionnement ;}
} ;
SmartCellControl*
SmartCell.Design ::CreateApplication(SmartCell.Factory          &
factory)
{
    SmartCellControl                                              *
UnifiedStructure=factory.CreateU.S() ;
```

Listing 1. Embedded system decomposition

$$\left[\bar{X}_{11}\right]=e^{-\tau p}\frac{\bar{G}(p)C(p)}{1+\bar{G}(p)C(p)} \quad (11)$$

The time delay represent the duration between control signal sending and its reception by the physical system. Its expression is approximated with first order Taylor series, then,

$$\left[\bar{X}_{11}\right]=\frac{1-\tau p/2}{1+\tau p/2}\times\frac{\bar{G}(p)C(p)}{1+\bar{G}(p)C(p)} \quad (12)$$

This fractional equation of $\bar{X}_{11}$ has the form,

$$\left[\bar{X}_{11}\right]=\frac{B(p)}{A(p)}=\frac{b_n p^n+b_{n-1}p^{n-1}+\cdots+b_1 p+b_0}{a_n p^n+a_{n-1}p^{n-1}+\cdots+a_1 p+a_0} \quad (13)$$

where $A(p)$ and $B(p)$ are polynomials that have n as maximum order. The next step consists to transform this equation with delta operator in order to discrete it. From this model, we can develop the corresponding recurrent equations.

3.1.2 MAC_Builder Environment

From recurrent equations given by the H model, the proposed MAC_Builder environment allows task graph development with MAC granularity. We propose to develop a task graph for the Park transformation function presented in **Listing 2**.

This function use trigonometric functions as sine function. We have developed a MAC structure corresponding to sinusoidal functions called MAC_sin and MAC_cos in order to develop a task graph corresponding to nonlinear elements. The **Figure 5** illustrates the proposed structure of MAC_cos. The task graph corresponding to Park Transformation function is given by **Figure 11**.

The process starts by computing the trigonometric functions of input vector by use of MAC_cos and MAC_sin functions; next we compute the output vector through investigation of elementary MAC operations.

3.2 Hardware/Software Partitioning

After the task graph development with MAC_Builder environment, the next step of proposed approach is the partitioning of these tasks between hardware and software platforms. With the aim to apply this approach on the induction motor control system, we can implement the developed task graph with the Composite design pattern. The advantage of this approach is to give an object which we can be reusable for several embedded system application just by modifying some parameters. In fact, each task is modelled with Composite design pattern given by **Figure 7**.

In order to affect tasks to hardware/software targets, we have developed a partitioning algorithm based on ant colony optimisation. We use the following notation, the visibility between two nodes of task graph and the pheromone's constants given by matrixes (h_{nn}) and (τ_{nn}) respectively:

$$(\eta_{nn})=\begin{bmatrix}\eta_{11} & \eta_{12} & \cdots & \eta_{1n}\\ \eta_{21} & \eta_{22} & \cdots & \eta_{2n}\\ \vdots & \vdots & & \vdots\\ \eta_{n1} & \eta_{n2} & \cdots & \eta_{nn}\end{bmatrix} \qquad (\tau_{nn})=\begin{bmatrix}\tau_{11} & \tau_{12} & \cdots & \tau_{1n}\\ \tau_{21} & \tau_{22} & \cdots & \tau_{2n}\\ \vdots & \vdots & & \vdots\\ \tau_{n1} & \tau_{n2} & \cdots & \tau_{nn}\end{bmatrix}$$

The transition rule is computed by a probability given by Equation (14), where α *and* β are parameters that control the visibility and pheromone respectively.

$$p_{ij}^{k}(t)=\frac{\left[\tau_{ij}(t)\right]^{\alpha}\cdot\left[\eta_{ij}\right]^{\beta}}{\sum_{l\notin S_k}\left[\tau_{il}(t)\right]^{\alpha}\cdot\left[\eta_{il}\right]^{\beta}} \quad (14)$$

The pheromone matrix is updated by the function, $\tau_{ij}(t)=\rho\times\tau_{ij}(t)+(1-\rho)\times\Delta\tau_{ij}$ where $0\prec\rho\prec 1$ and $\Delta\tau_{ij}(t)=\begin{cases}\dfrac{Q}{L(t)} & si\ \ (i,j)\in T\ \ (t)\\ 0 & si\ \ (i,j)\notin T\ \ (t)\end{cases}$; Q constant.

3.3 Complexity Problem

This section presents the synthesis of a hardware IP

```
void park_abc_dq_Outputs_wrapper
(const real_T *u, real_T *y)
{
const double pi=3.1416;
double i, j, k;
i=0; j=0; k=0;
   i=u[3];
   j=u[3]- 2*pi/3;
```

Listing 2. Park transformation function

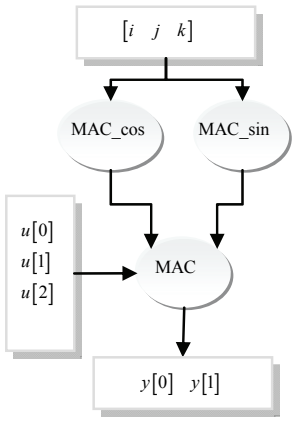

Figure 11. Park transformation DAG_MAC

block that implements the fuzzy controller in FPGA target. **Figure 12** illustrates three signals, the *error*, the *delta_error* and the *control* signal.

The application of partitioning algorithm contributes to affect the tasks into hardware aspect or software one. **Figure 13** present an obtained result for this phase. Further, a part of the logic circuit of this IP hard is presented in **Figure 14**.

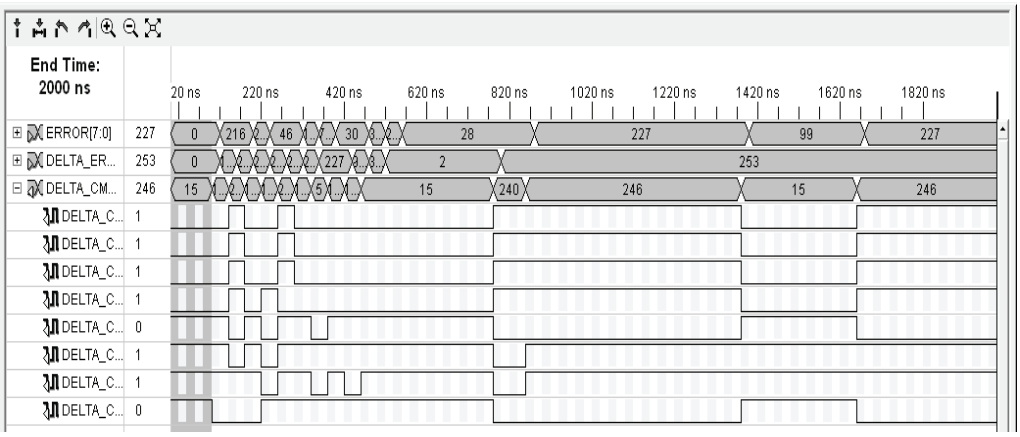

Figure 12. Simulation of fuzzy control system

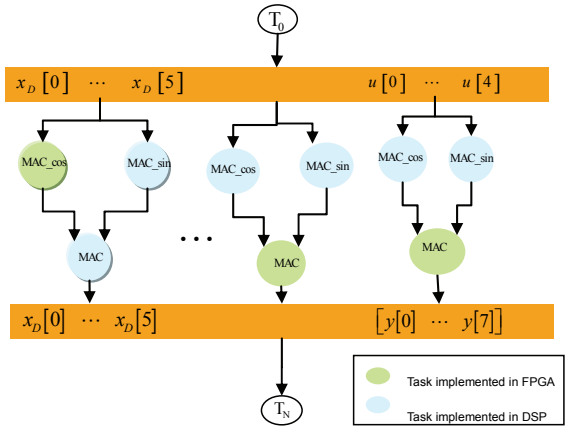

Figure 13. HW/SW partitioning

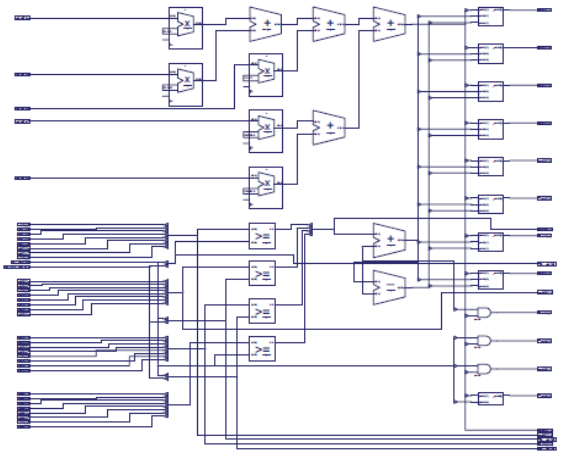

Figure 14. Logic circuit of fuzzy control system

4. Conclusions

In this work we carried out a multilevel design flow for embedded system through investigating the design pattern concept. In system level, the system decomposition is realised with the *Smartcell_Factory* design pattern. In the second level, each smartcell realises the model development, the DAG development, the hardware/software partitioning and the IP_hard/IP_soft blocks synthesis.

Three problems are resolved: the complexity, the hardware/software partitioning, and the reusability. Indeed, two intermediate models are developed in order to model the subsystem and to develop the task graph, the *H_model* that encapsulates the principal and the disturbances information of system in addition to communication protocol and control laws. The *MAC_Builder* is the second environment that allows developing a task graph corresponding to subsystem from the recurrent equation given by the *H_model*. The hardware/software partitioning problem deals with proposed *Component* design pattern which model the task graph given by the *MAC_Builder* in order to simplify the application of the proposed Ant Colony Optimisation algorithm. The IP blocks reusability is carried out through the result given by the partitioning phase, the IP_soft blocks are developed with C/C++ language and the IP_hard blocks are developed with VHDL/Verilog language.

As future work, the development of paradigm environment to execute the proposed approach is very required. In addition, we propose the development of complex control laws as fuzzy or neuronal control by the mean of the proposed *MAC_Builder* environment.

REFERENCES

[1] R. C. Dorf, "Systems, Controls, Embedded Systems, Energy, and Machines," Taylor & Francis, New York, 2006, pp. 486-511.

[2] K. Virk and J. Madsen, "A System-Level Multiprocessor System-on-Chip Modeling Framework," *Proceedings of SOC*, 2004.

[3] A. D. Pimentel and C. Erbas, "A Systematic Approach to Exploring Embedded System Architectures at Multiple Abstraction Levels," *IEEE Transactions on Computer*, Vol. 55, No. 2, February 2006, pp. 99-112.

[4] B. Zhou, W. Qiu and C. Peng, "An Operaing System Framework for Reconfigurable Systems," *Proceedings of CIT*, Salt Lake, 2005.

[5] S. Pasricha, N. Dutt and M. B. Romdhane, "Using TLM for Exploring Bus-Based SoC Communication Architectures," *Proceedings of ASAP*, Atlantic, 2005.

[6] R. Cumplido, S. Jones, R. M. Goodall and S. Bateman, "A High Performance Processor for Embedded Real-Time Control," *IEEE Transactions on Control Systems Technology*, Vol. 13, No. 3, May 2005, pp. 485-492.

[7] X. Wu, V. A. Chouliaras, J. L. Nunez and R. M. Goodall, "A Novel DS Control System Processor and its VLSI Implem-Entation," *IEEE Transactions on Very Large Scale Integration* (*VLSI*) *Systems*, Vol. 16, No. 3, March 2008, pp. 217-228.

[8] D. L. Sancho-Pradel and R. M. Goodall, "Targeted Processing for Real-Time Embedded Mechatronic Systems," *Control Engineering Practice*, Vol. 15, 2007, pp. 363-375.

[9] Y. Atat and N. E. Zergainoh, "Automatic Code Generation for MPSoC Platform Starting From Simulink/Matlab: New Approach to Bridge the Gap between Algorithm and Architecture Design," *Conference of ICTTA*, Bali Island, 2008.

[10] G. Wang, W. Gong and R. Kastner, "Application Partitioning on Programmable Platforms Using the Ant Colony Optimization," *Journal of Embedded Computing*, Vol. 2, No. 1, 2005, pp. 1-18.

[11] Y. Manai, J. Haggège and M. Benrejeb, "HW/SW Partitioning in Embedded System Conception Using Design Pattern Approach," *Conference of JTEA*, Hammamet, 2008.

[12] K. B. Chehida, "Méthodologie de Partitionnement Logiciel/Matériel Pour Plateformes Reconfigurables Dynami-Quement," PhD Thesis, Université de Nice-Sophia Antipolis, France, 2004.

[13] E. Gamma, *et al*., "Design Patterns: Elements of Reusable Object-Oriented Software," Addison-Wesley, Massachusetts, 1995.

[14] S. Konrad, H. C. Cheng and L. A. Campbell, "Object Analysis Patterns for Embedded Systems," *IEEE Transactions on Software Engineering*, Vol. 30, No. 12, December 2004, pp. 970-990.

[15] S. Konrad and B. Cheng, "Requirement Pattern for Embedded System," *Proceedings of the IEEE Joint International Conference on Requirements Engineering*, Atlanta, 2002.

[16] R. Damasevicius, G. Majauskas and V. Stulikys, "Application of Design Patterns for Hardware Design," *Proceedings of DAC*, Anaheim, 2-6 June 2003, pp. 48-53.

[17] F. Rincon, F. Moya and J. Barba, "Model Reuse through Hardware Design Patterns," *Proceedings of Design, Automation, and Test in Europe*, 2005.

[18] P. Coussy, *et al*., "Constrained Algorithmic IP Design for System-on-Chip," *Integration, the VLSI Journal*, Vol. 40, No. 2, 2007, pp. 94-105.

[19] J. K. Mak, C. S. Choy and D. P. Lun, "Precise Modeling of Design Patterns in UML," *Proceedings of International Conference on Software Engineering*, 2004.

[20] Y. Manai, J. Haggège and M. Benrejeb, "PI-Fuzzy Controller Conception with Design Pattern Based Approach," 14*th IEEE International Conference on Electronics, Circuits and Systems*, Marrakech, 2007, pp. 483-489.

[21] Y. Manai, "Contribution à la conception et la synthèse d'architecture de systèmes embarqués utilisant des plates-formes hétérogènes," Ph.D. Dissertation, Ecole Nationale d'Ingénieurs de Tunis, Tunisia, 2009.

