Design Pattern Based Approach for Embedded System Design

Yassine Manai, Joseph Haggège, Mohamed Benrejeb
LA.R.A, Ecole Nationale d'Ingénieurs de Tunis, BP 37, le Belvédère, 1002 Tunis, Tunisie
yacine_manai@yahoo.fr, joseph.haggege@enit.rnu.tn, mohamed.benrejeb@enit.rnu.tn

Abstract **—This paper deals with a proposed strategy for embedded system design based on the design pattern approach and the concept of object-oriented, supported by Unified Modeling Language (UML). It benefits from a set of methodologies as co-design approach, design pattern approach, the language of unified models and abstraction of information.**
The new view in design strategy lays on a proposed design pattern we call Unified Structure. Further, it examines the specification and the integration of fuzzy controllers on a DSP board, in order to control the speed and the torque of a DC drive. For an experimental validation of this controller a simulation is carried out in order to generate the code to be integrated on the DSP board.

Keywords — ***Embedded system, design pattern, UML, unified structure, integration on DSP, source code generation.***

I. Introduction

The embedded systems architecture, based on elementary decomposition of the system, can be considered as a set of modules, each one being managed by a smart cell. Each cell, supervised by a main processor, is the implementation of a processor which controls the operation of the corresponding module. This architecture is called "*smarts cells structure*". Cell represents a revolutionary extension of conventional microprocessor architecture and organization [13].

The aim of this paper is to propose an integration strategy of a fuzzy controller based on the approach of design pattern and the object-oriented concept for the use of a Digital Signal Processor (DSP).

The proposed integration strategy is based on a vision of embedded systems design which considers the embedded systems as interconnected smart cells. The structuration of these cells has a unified characteristic for all the cells of the system.

This paper is organized as follows: the design pattern approach is presented as well as the unified model language UML; then, is examined the embedded system design process, based on the design partitioning into two parts a hardware part and software one which will be integrated in an advanced phase of the design process. Thereafter, the embedded systems design methodology, based on the design patterns approach, is developed. Subsequently, an illustration of the embedded system design vision is treated and, at this level, the design pattern entitled *unified structure* is proposed. A study case is then discussed in order to design a fuzzy logic controller by exploiting the approach of the designs patterns. For an experimental validation of this controller, a simulation is achieved in order to generate the code to be integrated on a DSP board.

II. Development of Embedded System Design Strategy

In this section, the implementation of an embedded system design strategy, by the use of the design pattern approach, is examined. Furthermore, this approach illustrates the use of an object-oriented language like UML for the patterns development.

A. *Methodology of embedded system design*

The development of an embedded system design process is considered and a methodology based on the design patterns implementation and the exploitation of UML language proposed.

1) *Design methodology*

Figure 1 illustrates the principle of the embedded systems design process.

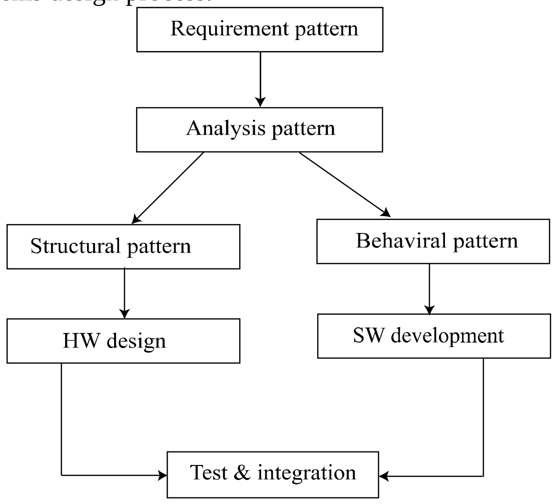

Figure 1. Process of embedded system design

Indeed, in the first phase the requirement definition is carried out by developing the *requirement pattern*. The second phase consists in the analysis by developing the *analysis pattern* which divides the system into structural and behavioral parts. The *structural pattern* enables us to implement the design hardware of the system using an embedded cells library, whereas the *behavioral pattern* leads to software development of the embedded system software. Finally, integration between hardware (HW) and software (SW) leads to the final realisation of the system. The major problem in the design process lays in synchronisation and integration between HW and SW components.

In this paper, the requirement pattern model is inspired by the requirement pattern model used by Gamma [12, 13], well-known in the design pattern world, and classified as follows:

[Name and classification]: the name consist in describing the pattern and classification gives the pattern nature: structural or behavioral.

[Intention]: the intention describes the problem targeted by the pattern.

[Structure]: the structure gives the pattern structure.

[Constraints]: the constraints describe the operating conditions and the pattern applicability fields.

[Behavior]: the behavior allows to describe the different dataflow which manage the design pattern operation represented by corresponding UML diagrams.

[Consequences]: the consequences describe the objectives they supported by such pattern.

2) Requirement pattern

The development of the requirement pattern consists of the abstraction of information encapsulated in a smart cell model.

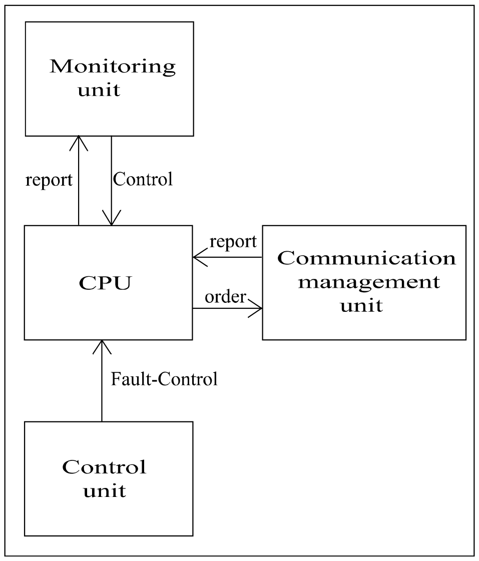

Figure 2. Elementary Unified Structure

The proposal in this domain is the use of a design pattern called *unified structure* which allows modeling the characteristics of an embedded system starting with the elementary components until the macroscopic system.

It is about unification in the vision of the control devices architecture. Any control device is seen as a unified structure or a network of interconnected unified structure. The proposed unified structure is based on four basic components: the Central Processing Unit (CPU) which the communications components manage the communication with inputs and outputs, the monitoring unit which allows the monitoring of the unified structure and the control component that control the errors. The unified structure is seen as being a main processor, figure 2; it is on this level that the programs are executed; each program is stored in an internal memory. The processor can take various realisations, a DSP or a microcontroller for examples.

The monitoring unit called the monitor, placed in the internal memory of the central calculator represents the Real Time Operating System (RTOS) of the unified structure; it allows the operation management of the unified structure.

The control unit contains an array gathering a system errors which can be appear and put the system in an unspecified state as well as the corresponding correction cycle.

Lastly, the communication unit allows the communication management of the unified structure with the external world. This unit can contain a great number of input and/or output modules.

In this paper, the systems, using only one unified structure are considered.

We describe now the design pattern corresponding to the proposed unified structure.

While referring to the introduction on the design pattern described higher, the model of the requirement pattern consists of four parts: the pattern name, the concerned problem, the solution and the sequences which are involved into this solution. The solution concerns the description of the pattern structure and the system constraints, while the sequences relates to the behavioral model description in the dataflow form and to application consequences of this pattern.

B. Requirement pattern model

[Name and classification]: the name of design pattern is *unified structure*; this design pattern has structural type.

[Intention]: the intention is an abstraction of the constitutive components of an embedded system and encapsulation of the methods and arguments by an object-oriented technique using UML.

[Structure]: it is composed, as indicated in figure 2, of four elements: a Central Processing Unit (CPU), a communication management unit, monitoring unit and control unit.

[Constraints]: it describes the operating conditions and the pattern applicability domains.

[Consequences]: the application of this pattern allows the easy re-use of this model, whereas, the abstraction supported by the requirement pattern, because of the object-oriented aspect that it supports, allows the data abstraction which generates the generic pattern in relation to the systems and what increases the applicability field of the requirement pattern for the unified structure. The unified structure, allows as before unifying the vision of architectures design of the embedded systems.

[Behavior]: it allows describing the data different flows which manage the operation of the unified structure represented by corresponding UML diagrams. Two families of data flows are distinguish: the entity data flow family and the functionalities data flow one. The first family concerns the creation and states of unified structure, whereas the second family data flow devoted to management of the unified structure functionalities. Initially, the state data flow of a unified structure is presented, and then the data flow of management of the unified structure functionalities are approached, once implemented.

- First data flow: state flow. The state flow chart, figure 2, allows to describe the various states where the unified structure can be. It acts of determining the data flow of the unit which manages the actions progress in the unified structure, the monitor. In the monitor program, it is necessary to take in consideration of the following constraints:

1. It allows the startup unified structure, initialisation, operation, the idle state, the standby state and the stopped state. Thus, it manages the various states in which the system can be, including extreme cases.
2. The monitor allows the management of the unified structure communication with the external world.
3. It manages the priority of the controls and messages as well as the standby state in the unified structure, i.e. it manages the multitasking.
4. It identifies the system errors in collaboration with the control unit. When the error identified, it must execute the suitable correction cycle.

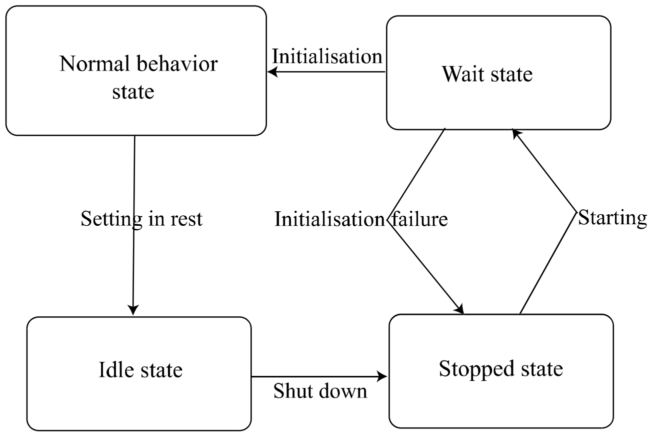

Figure 3. State diagram of unified struture

Thus, the monitor operation can be described graphically using UML diagrams, namely the state – transition diagram and the activity diagram.

The following states are distinguished: the normal behavior state, the stopped state, the wait state and idle state. The state diagram, figure 3, allows describing these various states of the system.

- Second Data flow: the main program flow. It is about the program which manages the tasks progress in the unified structure, figure 4, the UML activity diagram is used to describe this function.

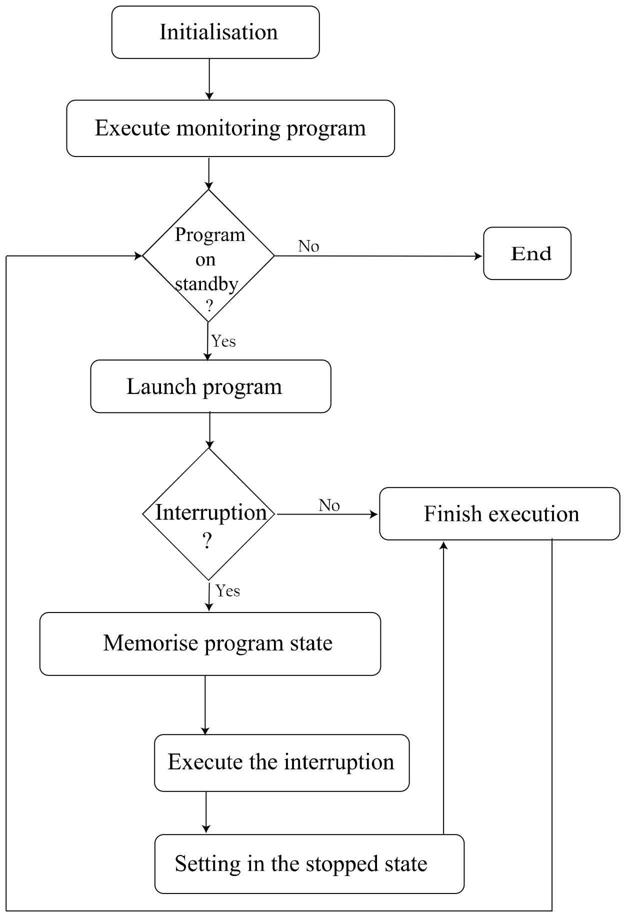

Figure 4. Programs execution

The central computer allows executing the programs which are classified by set of priorities by the unit of monitoring. The central computer can be in various states according to events which occur. During the execution of such a program, it can occur: an intervention by the monitor which requires the program to stop to execute a high priority instruction. When a system error occurs, the monitor saves the system state, launches a correction cycle according to the error, then takes again the program starting from the stopped point.

- Third data flow: communication management flow. This data flow allows the management of the input/output flow.

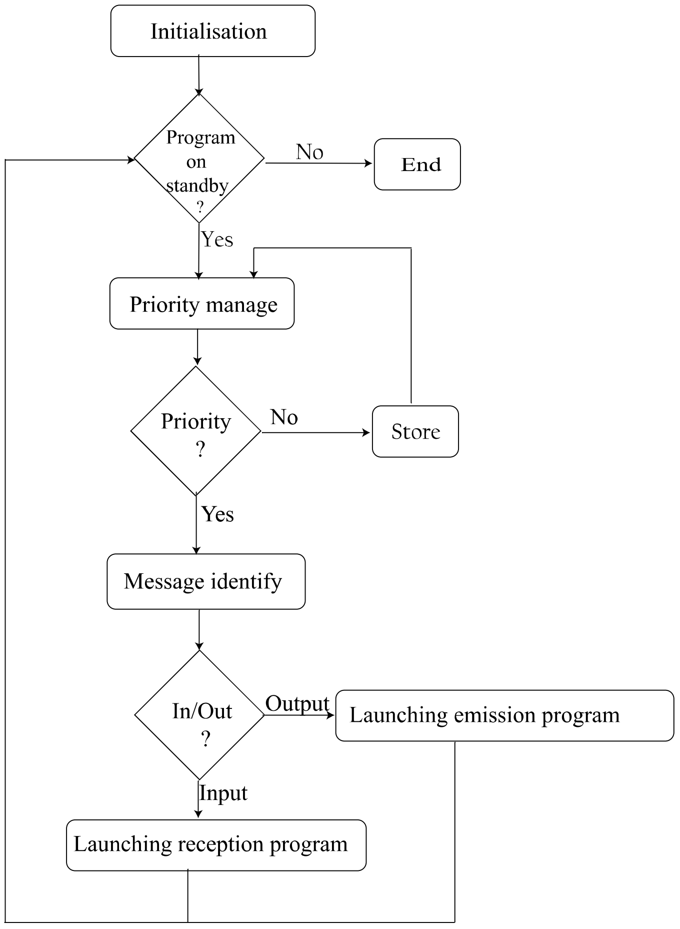

Figure 5. Input/Output flow management

The communication interface includes the interfacing components with the unified structure, i.e. the input/output flow mandatory circulates through this component. The monitor allows, by the means of a program, to manage this interfacing flow. First, the monitor identifies the flow nature: input or output; if flow is an input flow, then launch the reception program so not launch the emission program. An output flow is identified from a message sent by the central processor to the monitor. To identify an input flow, the communication interface sends a message to the monitor to inform it of the signal entering to the system. The monitor receives various messages, affects to it a priority level and decides suitable action to take: either it answers the message by launching the corresponding program, or it puts it into a queue.
In figure 5, the data flow graph, responsible for the communication management in the unified structure, is presented. The UML activity diagram is used to describe the communication sequences.

- Fourth data flow: errors management flow. The errors are managed by the control unit. When an error occurs, it will be detected by the program controller, and will be executed according to the following flow diagram, figure 6.

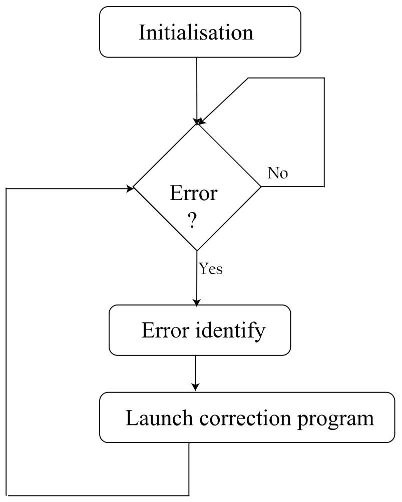

Figure 6. Search of errors

III. Case study

In this section, we propose to apply the unified structure technique to control the speed w and the torque C_{em} of a DC drive.

The controller architecture is given by the figure 4. The objective of the programmable control device, based on DSP TMS320LF2812, is to allow the implementation of a fuzzy logic control law [2, 3, 6] to control the speed of a DC drive and to limit the current. The control device will be modelled as a unified structure, applying the proposed approach of the designs pattern developed higher.
We propose to model the DSP fuzzy logic control device by the means of the unified structure; its structure is given by figure 7.

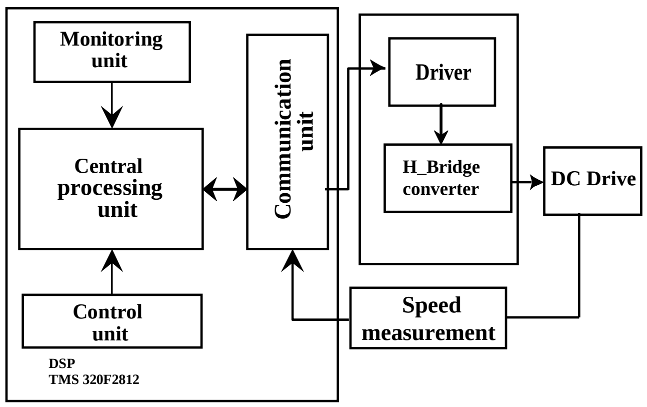

Figure 7. Control structure

The model of this controller, with the requirement pattern, is as follows:

[Name and classification]: [FuzzyController; structural pattern].

[Intention]: abstraction of the designed controller containing fuzzy logic and encapsulation of the methods and the arguments by the object-oriented technique, supported by UML.

[Structure]: the fuzzy controller unified structure has two inputs, the error ε and the delta_error $\Delta\varepsilon$, and one output control signal Δu is given by figure 8.

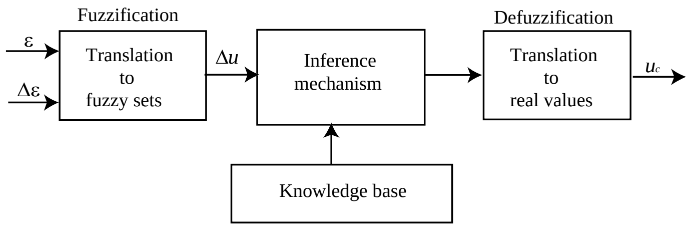

Figure 8. Fuzzy logic controller structure

By examining the unified structure during the design of such a controller, the inference mechanism is seen as the central processing unit of the design pattern, the fuzzification as an input communication interface and the defuzzification as an output interface.

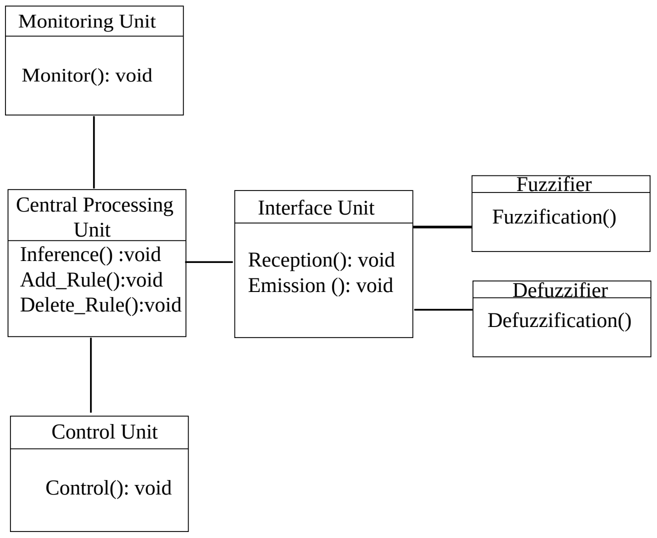

Figure 9. Control Unified Structure

Design pattern implementation of the fuzzy controller, according to the unified structures technique, figure 9, is presented as follows:

- the basic classes definition: *central processing unit class*, admits the following functions:

- inference mechanism method: *Inference*();
- addition rule method: *Add_Rule()*;
- suppression rule method: *Delete_Rule().*

- the second class: *fuzzifier class*, contains the methods:

- fuzzification method: *Fuzzification();*
- input reception method: *Reception();*
- output emission method: *Emission().*

-the third class: *defuzzifier class*, contains the method:

- defuzzification method : *Defuzzification().*

- the fourth class *monitor class*, allows managing the operation progress of the system using the method:

- monitoring method: *Monitor().*

- the last class, *control class*, controls the system against the errors with the method:

- control method: *Control().*

[Constraints]: Knowledge Base

The establishment of the relation between the inputs and the outputs of the system represents the significant part of the fuzzy controller. This relation must be correctly implemented in order to obtain the desired performances of the fuzzy logic controlled system.

The setting of the rules is managed by the *Add_Rule()* methods and *Delete_Rule()* of the central processing unit.

Table 1 shows the base of rules R_i chosen in this study.

TABLE I. Inference

$\Delta\varepsilon$ \ ε	GN	MN	ZE	MP	GP
GN	ZE	PP	MP	GP	GP
MN	PN	ZE	PP	MP	GP
ZE	MN	PN	ZE	PP	MP
MP	GN	MN	PN	ZE	PP
GP	GN	GN	MN	PN	ZE

[Behavior]: The behavior part in the model of the requirement pattern for the unified structure is described by the methods Fuzzification(), Inference() and Defuzzification(). These methods are illustrated as follows:

- Fuzzification

This fuzzy logic controller development phase is managed by the module *Fuzzifier class*. Initially, the reception method of the input is launched, thereafter the method of membership functions is launched to transform the data in the linguistic field. The cycle starts again if there is new input. This algorithm, implemented by the *Fuzzification()* method, belongs to the *Fuzzifier class* of the communication interface.

The membership functions are chosen of triangular form; this choice is justified by the simplicity of implementation of these functions on a digital computer.

- Inference mecanism

The sum-product inference method is adopted in the inference mechanism. This method is given by the following algorithm, where:

o_i *is operation i*; c_i *is condition i*; R_i *is rule i*;

- condition membership function evaluation :

$$\mu_{C_i} = \begin{cases} \mu_{O_i}(\varepsilon) \times \mu_{O_i}(\Delta\varepsilon) & \text{if the condition is verified} \\ 0 & \text{otherwise} \end{cases}$$

- membership function of the rule R_i computing :

$$\mu_{R_i}(\Delta u) = \mu_{C_i} \times \mu_{O_i}(\Delta u) \quad i = 1, 2, ..., m$$

- resultant membership function calculation :

$$\mu_{RES}(\Delta u) = \frac{1}{m}\left[\mu_{R_1}(\Delta u) + \mu_{R_2}(\Delta u) + \ldots + \mu_{R_m}(\Delta u)\right] (1)$$

The Inference() method of the central processor is developed according this algorithm. Thus, the value of the control signal variation to be applied to the engine is obtained. This quantity is then defuzzified into a real value. The main program is responsible of the organisation of the tasks execution progress in the unified structure. This program is given by the flow chart described in the preceding sections.
To examine the flow management supported by the monitor, figure 10 illustrate the principle of flow management in fuzzy logic controller.

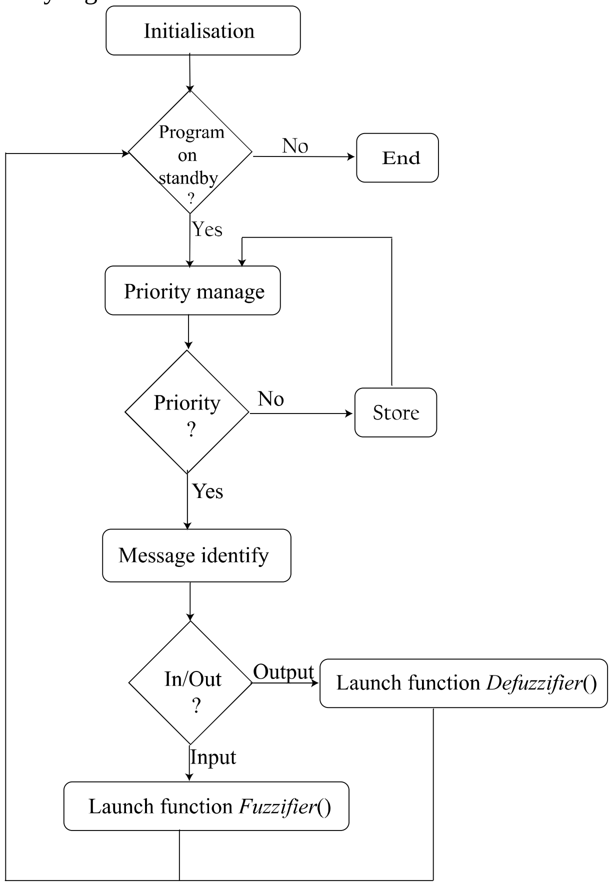

Figure 10. Management of fuzzification and defuzzification flow

- **Defuzzification:**
In this last stage, the results obtained by the inferences rules must be defuzzified to convert them into concrete numerical values.

To achieve this goal, the centroid method is used; it is given by the following expression:

$$u_c = \frac{\sum_{u=1}^{m} \mu_{RES}(\Delta u) \cdot \Delta u}{\sum_{u=1}^{m} \mu_{RES}(\Delta u)} \quad (2)$$

Where *m* is the number of rules contained in the rules base.

IV. Experiental Validation

In order to design a fuzzy logic controller following stages are taken into account:

- choosing the inputs and outputs variables;
- defining inference rules of and the memberships functions;
- developing inference mechanism;
- choosing a defuzzification strategy.

A Proportional Integral (PI) control law, given by equation 3 is considered:

$$\Delta u_k = k_p(\varepsilon_k - \varepsilon_{k-1}) + k_p \frac{T}{\tau_i} \varepsilon_k \quad (3)$$

where k_p *is* proportional gain; τ_i *is* integral constant

A. Application to D.C. drive

The proposed fuzzy controller of studied system combines a conventional controller with a fuzzy one by adaptation of the virtual set point [2]. The developed simulation blocks include PI-fuzzy controller block, a DC-PWM block, a DC drive model block, a speed measurement block and an H-Bridge converter block, figure 11.

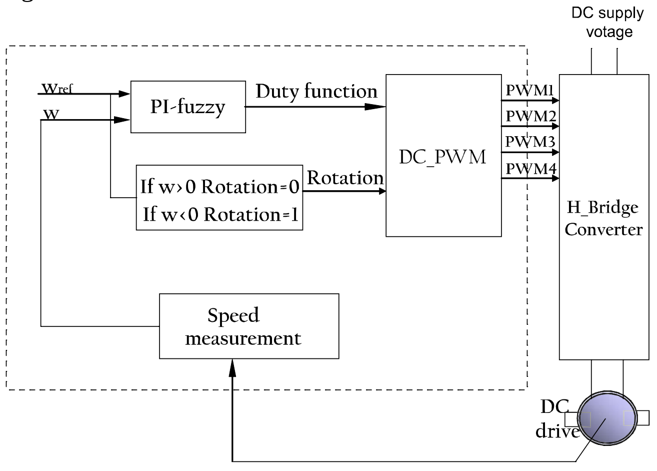

Figure 11. Composite PI-Fuzzy Control of speed

The figure 12 shows the behavior of the controlled system with a composite controller PI–fuzzy at the time of the application of order speed $w_{ref} = 150 rad.s^{-1}$, which become $w_{ref} = 300 rad.s^{-1}$ at $t = 40s$ .The resistant torque is $C_r = 10\, N.m$ as initial value which become

$C_r = 50\ N.m$ from $t = 50s$. The advantage of this control that is the speed does not present any overshot.

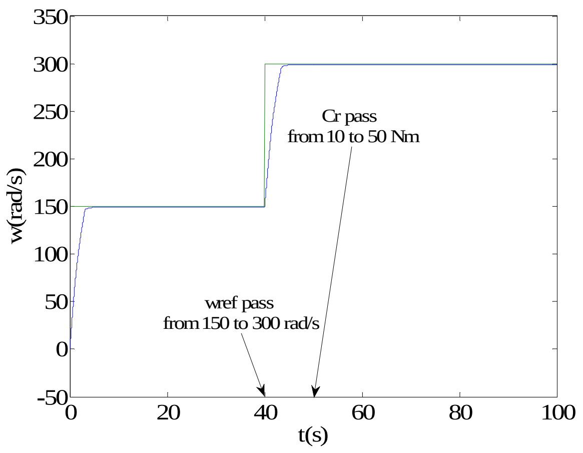

Figure 12. Answer of speed

Figure 13 shows the behavior of the current. The starting current of the system is less than 40 A, at the time of application of the resistant torque the current present a surpassing and his value stabilises to 45 A.

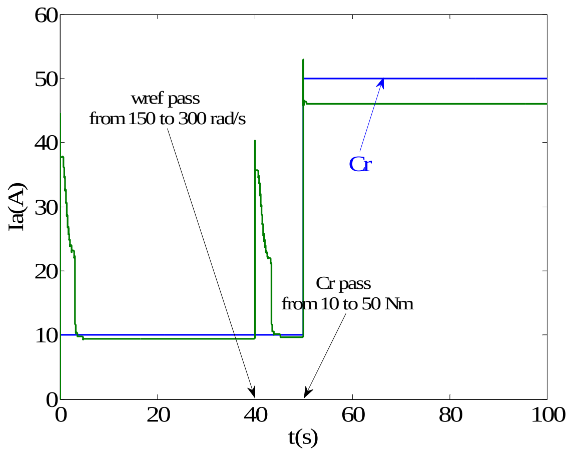

Figure 13. Current behavior

To integrate the code in DSP board, an example of this code generated for the initialisation part is given by the following listing:

```
            .def _DSP28x_usDelay
            .sect "ramfuncs"
            .global  __DSP28x_usDelay
_DSP28x_usDelay:
   SUB    ACC,#1
    BF   _DSP28x_usDelay,GEQ   ; Loop if ACC >= 0
      LRETR
;There is a 9/10 cycle overhead and each loop
;takes five cycles. The LoopCount is given by
;the following formula:
;DELAY_CPU_CYCLES = 9 + 5*LoopCount
;LoopCount = (DELAY_CPU_CYCLES - 9) / 5
.def _DSP28x_usDelay
```

B. *Results*

To determine the behaviour of the system showing the different states and events, the Gantt diagram, figure 14, allows to illustrate the different tasks according to the number of cycles T, with examining the different states in which the system can be.

The following states can be identified:

- The *normal behavior state*, during which the system determines the control signal and sends it to the driver to adjust the speed of the DC drive by the means of the communication interface. The computing of the fuzzy logic control signal is executed as follow: fuzzification of the error, fuzzification of the error variation, constraints establishment, mechanism of inference and defuzzification. The setting of the rules base is independent of the fuzzification.
- The *power turn-on management*: which allows optimising the power consumption, the stopped state is entered when the initialisation of the system fails.
- The *errors management*: during which the system makes the error identification, this is achieved during a cycle, and consequently, the monitor launches the cycle of correction during two cycles or three cycles according to the error category.
- The *priority management* is necessary during the management of a message flow of various places. Initially, the monitor receives the new message; if it has a high priority, it is sent to the processing unit to be processed, if not, it is stored into a queue.
- The *Identification of the messages* in order to launch the suitable program: after the reception of the message in a cycle, the monitor identifies the message into the same period; if the message is a reception order then the monitor launches the program of reception which lasts two cycles, if the message is an emission order, the monitor carries out the emission program during two machine cycles.

V. CONCLUSION

The presented works are related to the design and the integration of an intelligent controller on a DSP. A strategy of integration of a controller based on the concept of design pattern by using a digital signal processor was developed and implemented. The proposed strategy is based on the partitioning of the design in two parts: a hardware part and a software part which will be integrated in an advanced phase of the process of design. In order to simplify the re-use of the control components, a vision of embedded systems design based on the introduction of a unified structure is illustrated. This vision consists with the implementation of a design pattern called unified structure, responsible for the modeling of control device in an embedded system. The approaches using this principle are based on the use of the designs pattern. A case study based on the fuzzy logic control of a DC drive is discussed in order to validate the approach of design, based on the unified structure. Lastly, the experimental validation of the integration of the control device on DSP board was made.

VI. References

[1] F. Betin, "Energie Electrique et Systèmes Associés", CREA – UPRES 2002.

[2] P. Borne, J. Rozinoer, J. Y. Dieulot, "Introduction à la commande floue", Technip, Paris, 1998.

[3] H. Bühler, "Réglage par logique floue", Presses Polytechniques et Universitaire Romandes, Lausanne, 1994.

[4] J. Y. Haggege, "Sur La Synthèse de processus complexe par des méthodes neuro-floues", Thèse de Doctorat, Enit, Tunis, 2003.

[5] D. Dubois, H. Prade, "Fuzzy sets for intelligent systems", Morgan Kaufman Publishers, San Mateo, 1993.

[6] E. Mamdani, "*An experiment in linguistic synthesis with a fuzzy logic controller*", International Journal on Man Machine Studies, 7, pp. 1-13, 1975.

[7] T.Takagi and M.Sugeno, "*Fuzzy Identification of Systems and Its Applications to Modelling and Control*", IEEE Transactions on Systems, Man and Cybernetics, vol. SMC-15, N° 1, 1985.

[8] G. Majauskas, V. Stuikys, "*Application of Design Patterns for Hardware Design*", DAC 2003, Anaheim, California, pp. 48-53.

[9] R. Ernest, "*Codesign of embedded system : Status and Trends*". IEEE Design & Test, pp. 45–54, 1998.

[10] M. Broy, "Requirements Engineering for Embedded Systems", Proc. First Workshop Formal Design of Safety Critical Embedded Systems (FemSys), Apr. 1997.

[11] Z. Balanyi and R. Ferenc, "*Mining Design Patterns from C++ Source Code*", Proceedings of the International Conference on Software Maintenance (ICSM'03), IEEE 2003.

[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "*Design Patterns : Elements of Reusable Object-Oriented Software*", Addison-Wesley Pub Co, 1995.

[13] J. A. Kahle, M. N. Day, and D. Shippy, "*Introduction to the Cell multiprocessor*", IBM Journal of research and Developemnt, Volume 49, Number 4/5, 2005

[14] S. Konrad and B.H.C. Cheng, "*Requirements Patterns for Embedded Systems*", Proc. IEEE Joint International Conference Requirements Eng. (RE '02), Sept. 2002, Washington, DC, USA.

[15] S. Konrad, B.H.C. Cheng, and L.A. Campbell, "*Object Analysis Patterns for Embedded Systems*", Technical Report MSU-CSE-04-29, Computer Science and Eng., Michigan State Univ., East Lansing, Oct. 2004.

[16] F. Pospiech, S. Olsen, "*Embedded Software in the SoC World. How HdS Helps to Face the HW and SW Design Challenge*", IEEE 2003 CIC Conference, pp 653-558 ;

[17] B.P. Douglass, "Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks and Patterns", Addison-Wesley, Cambridge, 1999.

[18] C. Alexender, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King and S. Angel, "A Pattern Language ", Oxford University Press, Oxford, 1977.

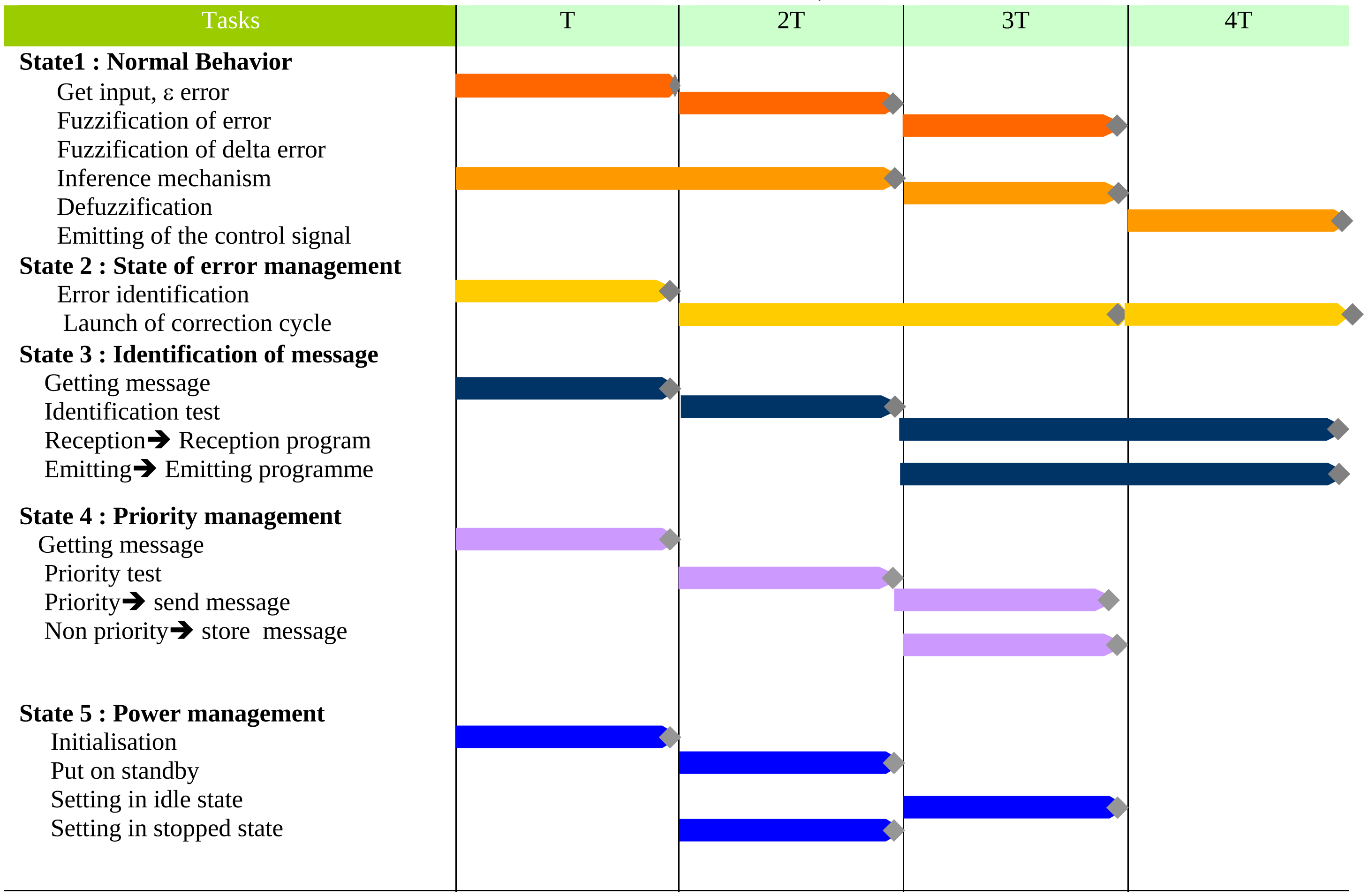

Figure 14. Gantt Diagram of operating state of unified structure of PI-fuzzy controller