Design of an Adaptive Motors Controller based on Fuzzy Logic using Behavioural Synthesis

Adel CHANGUEL(1), Robin ROLLAND(2), Ahmed Amine JERRAYA(1)

(1) TIMA/INPG, System-level Synthesis Group
(2) Centre Interuniversitaire de Micro-Électronique (CIME).
46, Avenue Felix Viallet, 38031 Grenoble Cédex1, FRANCE
Fax: (33) 76 47 38 14, E-mail: adel.changuel@imag.fr

Abstract

This paper combines two advanced technologies, High-level synthesis and Fuzzy control, for the design of an adaptive multiple motor speed controller. The obtained solution compares favourably with classic methods in terms of design quality. The use of Fuzzy control allows to implement an original architecture which is faster and smaller then classical solution based on PID. The use of High-level synthesis results in a drastic acceleration of the design process.

This paper presents the adaptive motor control application, the Fuzzy control approach and the results of the design using high-level synthesis.

1. Introduction

Motor controllers are in general implemented using a (Proportional Integral Derivative) PID algorithm. An explicit optimal solution for the robot arm control problem is proposed in [1]. Various implementations in an analogue technology have been proposed. However in this case we have an extra constraint of designing the full system in a pure digital technology.

When the Adaptive Motor Speed Controller (AMSC) is modelled using a PID algorithm [2], the obtained design has a relative long execution time. This is due to the complexity of the computation needed by the PID algorithm. In fact, this application has a reaction time constraint of 6 ms that must be respected. So, many PID circuits are needed to control 18 motors, and the design obtained will be relatively big in terms of number of gates. Thus, the implementation will be expensive.

To reduce the cost of the implementation, we propose to model the application using fuzzy logic [3]. The implementation of a fuzzy logic controller requests undeniably some abstraction efforts. This is not the only difficulty of the development of fuzzy logic into the control system. In fact, unlike models based on classic control algorithms, there is not an analytical method of evaluating the stability of the fuzzy logic controlled system. This is due to the non linearity in the fuzzy algorithm [4].

Fuzzy logic will not give a significant improvement when a process is modelled using classic methods. However, if the process is difficult to model, the question is to know if it is possible to have mathematical model. In this case, fuzzy logic may give quickly a simple solution. So, to control a complex process, the resort to fuzzy logic is in most cases justified. Another benefit of fuzzy control is that this kind of algorithm can be easily handled by high-level synthesis tools, called also a behavioural compiler [5]. In fact the algorithmic nature of fuzzy computations makes it easy to specify and synthesise the controller. The main contribution of this paper is the combination of these two advanced technologies - High-Level Synthesis and fuzzy control - in order to improve the design quality and productivity.

This paper is organised as follow. The next section illustrates the AMSC application. Section 3 deals with the fuzzy logic algorithm implemented. In the last section we discuss the architectural synthesis and FPGA prototyping of the AMSC.

2. The Adaptive Motors Speed Controller

The robot arm controller acts between a "host machine" and a robot arm that makes use of 18 stepper motors. The host fixes the trajectory of the arm. It sends the commands to the robot arm. The controller makes use of an adaptive speed algorithm to smooth the motion of the arm.

The principle of adaptative control and detailed discution of several system architectural solutions are given in a companion paper [6]. This paper focus on the hardware design of the AMSC for one spécific system architectural solution.

2.1. Functional description of the Adaptive Motor Speed Controller (AMSC)

The AMSC is embedded in an on-board system. It interacts both with the robot's host machine and a group of motors. It adjusts the speed of each of the motors according to the distance to cover. As shown in fig.1, the system is composed of many subsystems: a main computer, a dual-port memory, the AMSC and local send-

EURO-DAC '96 with EURO-VHDL '96
0-89791-848-7/96 $4.00

receive modules. The main computer computes the global motion of the robot and transforms it into specific elementary motions for each motor in the robot's coordinates. These elementary motions as well as some other motor characteristics are stored in the dual-port memory. For each motor, the AMSC loads from the dual port memory, the remaining distance to cover. Then, using a fuzzy logic based algorithm, the AMSC computes the corresponding speed and transmits it to the send module. The send module is a sort of timer that transforms the speed value into a pulse-width-modulated signal to supply the motor. The receive module memorises the distance done by the motor in order to subtract it from the remaining motions in the memory.

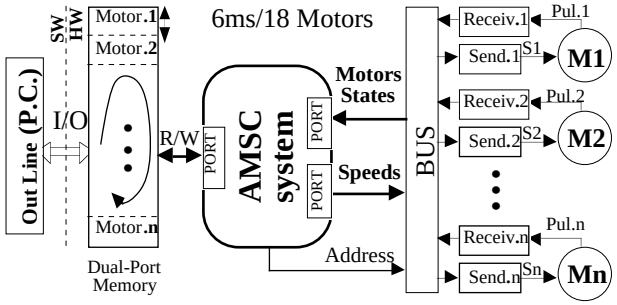


Figure 1. The global environment of the AMSC system

2.2. Real-time constraints

With this architecture, the AMSC will compute concurrently all the motor speeds. The total execution time is obtained by multiplying one motor execution time by the number of motors. In this case we assume that we have 18 motors to control at a reaction time less then 6 ms.

Using the classic solution based on a PID, the system modelling is complex due to parameter variation and the system has a little information on the task to do. For example to follow a curve, the travelling speed must be limited in order to avoid excessive speed increase and a brusque stop. In fact, for mechanical reasons, the change in speed should follow a smooth curve for acceleration and deceleration. All these constraints must be taken into account. When the computation time is short, the system can react at the correct moment.

2.3. Architectural analysis of the AMSC

Several architecture solutions are possible. In this case, the selected architecture must comply with the real-time constraints and a cost optimisation will be made. In addition, and because of integration constraints, the solution has to be fully digital. In the classic solution a digital PID structure would be used. Such a solution has been performed in [2]. It resulted in a 10.000 gates-chip with a computation time of 1 ms/motor. In order to control the 18 motors within the 6 ms, we would need 3 PID modules.

The other solution investigated in this paper is based on fuzzy control. In fact the function of the AMSC may be modelled as a set of rules depending on a set of input data.

3. The fuzzy logic controller

In this section we provide the details of the solution based on fuzzy Logic. The AMSC will be modelled and discussed using the specification of the fuzzy controller.

3.1. Basic principles

The basic idea in fuzzy logic [7] is that a value such as "speed is 100 laps/min" can be expressed as having a membership value (μ) in a fuzzy set. For example "speed is .8 slow" and "speed is .2 fast", where slow and fast are fuzzy sets and the values .8 and .2 represent the degree of membership in the respective sets [8].

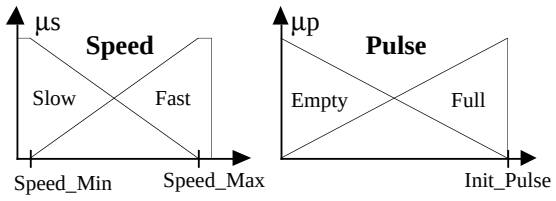
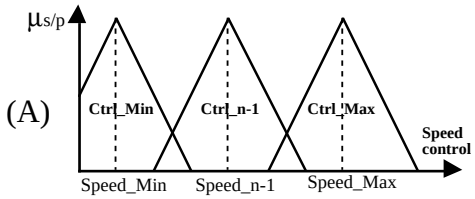


Figure 2. Frontend fuzzy sets

In addition to above fuzzy sets, each fuzzy logic controller incorporates:

***Sets of membership functions** to which the input facts belong in varying degrees. Typically, a set has one name such as Slow (Sl), Fast (Fa), Empty (Em) and Full (Fu). Each variable has its own set of membership functions. In the case of the AMSC, we need two variables: *speed* and *pulse*. The set of membership functions of speed are *slow* and *fast*. The variable pulse corresponds to the number of pulses that still have to be performed by the motor. The set of membership functions of pulse are *full* and *empty*. Fig.2 shows the sets of membership functions for the two variables, *Speed* and *Pulse* in a sample fuzzy logic application.



(B)
(1) if **Full** and **Slow** then **Speed_max**
(2) if **Empty** and **Fast** then **Speed_min**
(3) if **Full** and **Fast** then **Same_Speed**

Figure 3. (A) Backend fuzzy sets. (B) Rules.

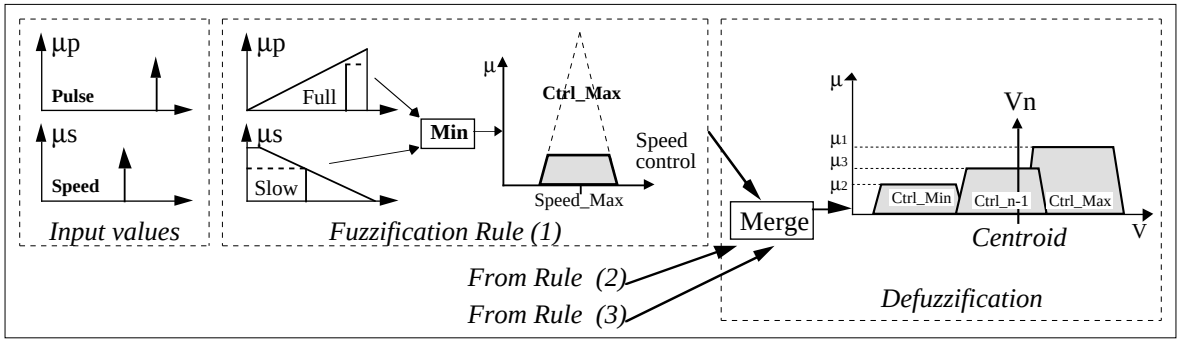


Figure 4. Fuzzy controller principles

***A rule base** which governs the behaviour of the fuzzy controller. Typically, these rules are in the form of "if ... then" statements such as "if Full and Slow thenSpeed_max", where Full and Slow represent the degree of membership in the membership functions. In the case of our example, the control is specified using the three rules of fig.3 (B).

3.2. Application to the adaptive AMSC.

The operations of the fuzzy controller are shown in fig.4. The fuzzy controller performs the three following functions:

(1) Fuzzification converts the input values for the variables into a fuzzy value. The operation includes a lot of searching in lookup tables and comparisons.

(2) Rule Application applies all the rules and produces a fuzzy output value. The operations include truncation and convolution of the membership functions.

(3) Defuzzification converts the fuzzy output value produced by the rules. The operations here include computing the centroid of the backend membership functions. Fig.5 shows the Defuzzification formula used to compute the output speed "Vn". This is the last step, an optimisation can be obtained by reducing unnecessary parameter, using normalising function to limit variation interval and by using a simple defuzzification function.

In the next section, we will illustrate architectural synthesis of the design.

$$\mathbf{Vn} = \frac{\mu_1 * \mathbf{Vmax} + \mu_2 * \mathbf{Vmin} + \mu_3 * \mathbf{Vn\text{-}1}}{\mu_1 + \mu_2 + \mu_3}$$

Figure 5. Defuzzification Formula

4. High level synthesis

Once the speed computation algorithm is elaborated, we proceed with simulations [9]. Firstly, the algorithms are described in Behavioural VHDL [10] and compiled in order to verify the behaviour in critical situations and to resolve same convergence problem. When correct functionality is assured, we begin the design synthesis. The architecture synthesis tool (AMICAL) and Logic synthesis tool (SYNOPSYS) are used for this purpose.

```
entity core  is
port (...);
end core ;
architecture core_arch of core  is
begin
   core1 : process
Variable  N , Adr_Ram,U1,U2,U3 : Bu_8bits;
Variable  V1,V2,V3,Tempo : Bu_8bits;
constant  Z : Bu_8bits := "00000000";
Begin
Adr_ram   := z ;
ReadRam(Adr_Ram,N);
  While (N /= Z) Loop
    Adr_ram := Mult10(N);
    ReadRam(Adr_Ram,U1);
    tempo := Z;
    Adr_ram := plus1(Adr_ram);
    .....
    ReceivePls(N,V1);
    moinw(U3,V3,tempo,V1,U3,V3);
    tempo := Diviw(U3,V3,U2,V2);
    tempo := shiftw2(tempo);
    writeram(Adr_ram,U3);
    Adr_ram := plus1(Adr_ram);
     --VN=(U1*V1+U2*V2+U3*VN)/(U1+U2+U3);
    writeram(Adr_ram,tempo);
    sendspeed(N,tempo);
    N := moin1(N);
  end loop;
end process core1;
end core_arch;
```

Figure 6. Incomplete VHDL description

4.1. Design flow

The architecture synthesis is realised using AMICAL, an architectural synthesis tool [11]. It starts with two kinds of information: a behavioural specification given in VHDL and an external functional unit library. During the different steps in the high level synthesis, the functional units are used as black boxes. The different steps involved in the synthesis process are: scheduling, allocation and architecture generation.

The library used by AMICAL can include complex functional units. The behavioural description accesses these function blocks through VHDL functions and procedure calls. For each procedure or function used, the library must include at least one functional unit able to execute the corresponding operation. As shown in fig.7, to synthesise the fuzzy logic algorithm, we need a library of functional units holding multiplication, division and ALU (+, -, shift, etc.). So, we specify and simulate separately these functional units in VHDL at the register transfer level

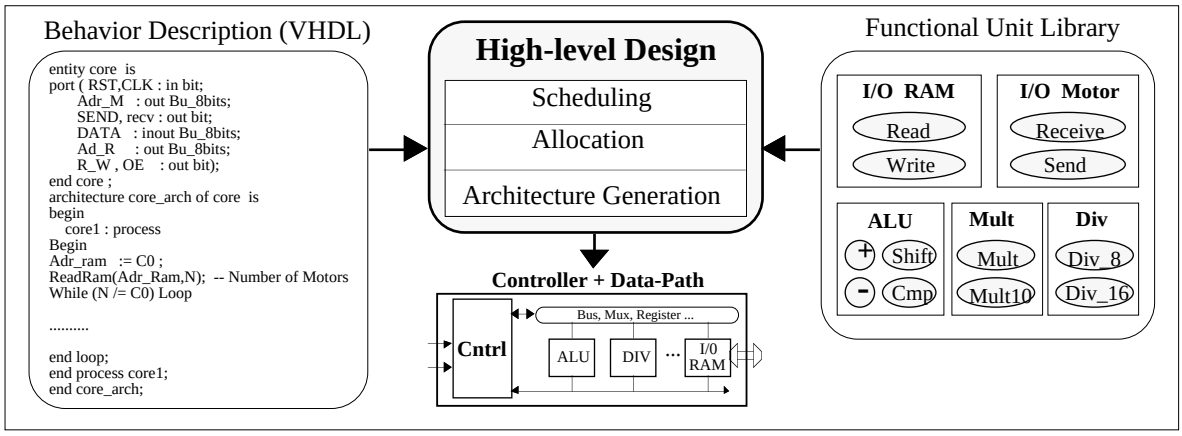


Figure 7. Architecture synthesis flow

(RTL). Next, according to their characteristics we specify them in the conceptual, implementation and high-level synthesis view. These specifications are used by AMICAL for the architectural synthesis.

The design is explicitly described by the VHDL code at the behaviour level. A brief VHDL description is given in fig.6. The simulation at the behaviour level has shown that 8-bit computation are sufficient to have an accurately result. In addition to the AMSC, the full VHDL description includes the SEND and RECEIVE modules in charge of communication with the motors. These are described in VHDL at Register-Transfer-Level and synthesised using the "Synopsys" design compiler.

As it can be seen in fig.1, the AMSC has an interface to the shared memory at the left hand and an interface to the motor current drive at the right hand. So, the AMSC system communicate in real time with these interface using an appropriate protocol. To allow this communication, we use two communication units, one for the RAM and the second for the motor interface. These are used as functional units (FU) executing communication operations. These FUs execute specific protocols synchronised by static clock cycles. The I/O Ram FU executes two operations: Read_ram and Write_ram. Each operation needs two clock cycles. The I/O motor FU consists of two operations: Send_speed and Receive_pulse

4.2. Architectural synthesis results

The architectural synthesis of the AMSC, using the I/O interfaces, generates an architecture with a controller of 39-states and an FSM with 108 transitions. It controls the data-path through 49 command lines. The data-path obtained after some interactive architectural transformations is made of:

- 5 functional units: ALU, Multiplier (8x8), Divisor (16/16), I/O RAM and I/O MOTOR.
- 9 Registers of 8-bits.
- 3 Mux(2), 5 Mux(3) and 8 Mux(4). Mux(i) is a multiplexer with "i" inputs.

Fig.8 shows the architecture of the AMSC produced by AMICAL. It is composed of a top controller, a set of functional units and a communication network. These last two form the data path. The communication network is composed of buses, multiplexes and registers. The network is build in order to allow the communication between registers, functional unit and the controller. The top controller sequences the operations executed by the functional units and the communication network

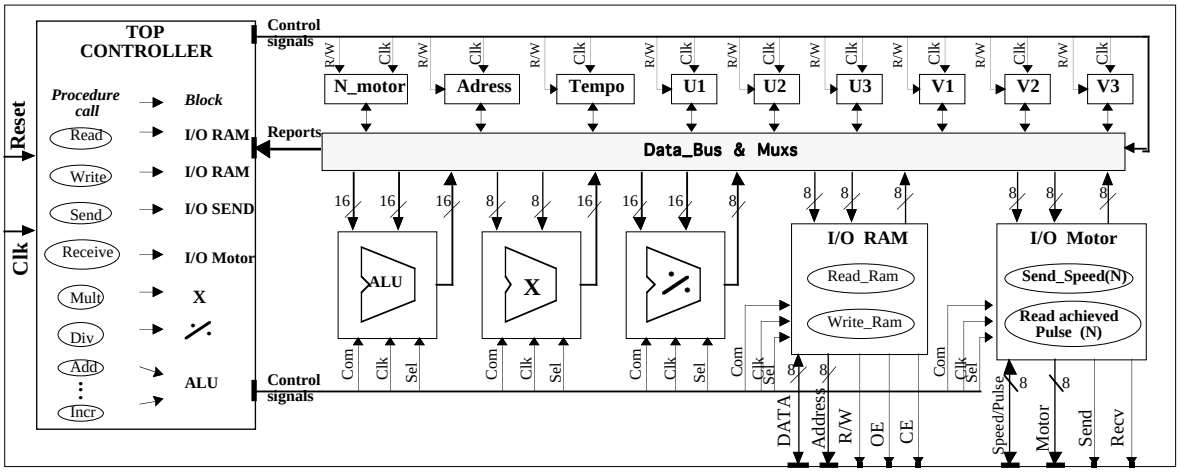


Figure 8. Architecture of the AMSC design

The controller is generated automatically during the synthesis process. The architecture includes several functional units that may run in parallel. The amount of parallelism is fixed during the synthesis process. The Data path includes 2 communication units: I/O-Ram and I/O-Motor. These communication units are not only connected with the data path, like the other components, but they are also connected to the external environment and are building the connection between the internal data path and the external pins of the circuit. Such components consist of two parts: the first part contains the interface to the data path and is synchronous. Only this interface must be considered by the high level synthesis system. The second part contains the external interface of the circuit. It contains all signals for the specific protocol. The interface component handles the complete protocol without further interaction with the controller or data path. It may be to complex that it contain internal registers and a local controller.

5. System integration

After architectural synthesis, we use a commercial logic synthesis tool and the Xilinx placing and routing tool to produce a prototype net-list. The final design is about 10.000 gates. The system is implemented on a platform supporting all sub-modules (fig.9). This platform is a standard PC-compatible computer with an extension FPGAs card. The AMSC is mapped on a Xilinx 4013 running at 4Mhz (FPGA1). The send and receive sub-modules with a motor models are mapped on a Xilinx 4005 (FPGA2) [12]. A 4K x 8-bit dual-port RAM is used as an interface between the out-line (P.C.) and the AMSC .

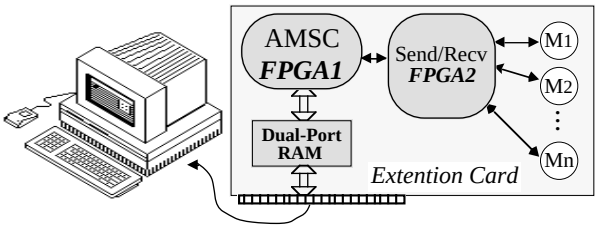


Figure 9. Implementation of all the system

The AMSC needs 120 clock cycles to run the fuzzy logic algorithm for one motors. Since the FPGA runs at 4 Mhz, the AMSC allocates 0,03 ms of computation time to each motor. The computation loop for 18 motors will be about 2160 cycles and takes 0.54 ms, which satisfies the real-time constrain. However the AMSC control up to 200 motors within the 6 ms initial reaction loop constraint.

The fig.10 illustrates the important improvement and the cost reduction using fuzzy logic compared to the PID algorithm. In fact, in this case fuzzy control solution allows to obtain a much fasten and smaller design than classic control solutions.

	PID	**Fuzzy Algorithm**
N° Cycle /motor	3000	120
Reaction Delay/motor	1 ms	0,03 ms
N° of Motors/6ms	6	200
N° of gates	10.000	10.000

Figure 10. Synthesis results

6. Conclusion

This paper discussed and illustrated the implementation of the AMSC using a fuzzy logic algorithm. This design shows the advantages of fuzzy logic to model a complex controller system. In fact, it ease the design since we avoid the complexity of a mathematical model. Since the system uses few rules, the implementation was very fast.

In this project the combination of two advanced technologies High-level synthesis and Fuzzy logic, allowed the implementation of a new efficient solution. The new design is much faster and smaller then classic solution based on PID.

References

[1] G. L. LOU and G. N. SARIDIS, "*L-Q* Design of PID Controllers for Robot Arms", IEEE Journal of Robotics and Automation, Vol. RA-1, N° 3, 152-159, September 1985.

[2] P.Kission, H.Ding, A.A.Jerraya "Structured Design Methodology for High-level Design" 31st ACM/IEEE Design Automation Conference, 1994.

[3] L.A.Zadeh, "Fuzzy Set," Information and Control. , (1965).

[4] M.C.Fritsch, E.Wendling, "Commande de processus: Les atouts de la logique floue" Technologies Internationales, pp 27-32, April 1994.

[5] Daniel Gajski, Nikil Dutt, Allen Wu and Steve Lin, High Level Synthesis. Kluwar Academic Publishers, 1992.

[6] M.Abid, A.Changuel, A.A.Jerraya, "Exploration of Hardware/Software Design Space through a Codesign of Robot Arm Controller" EURODAC, Genève 1996.

[7] G.C.Lee, "Fuzzy Logic in Control Systems: Fuzzy Logic Controller - Part I and II" IEEE Transaction on systems, Man and Cybernetics, pp. 404-435, March/April 1990.

[8] D.D.Gajski, "Towards achieving an 100-hour Design Cycle: A test Case" Technical Report #94-08 February 14, 1994.

[9] C.A.Valderrama, A.Changuel, P.V. Raghavan, M. Abid, T. Ben Ismail, A.A.Jerraya "A Unified Model for Co-simulation and Co-synthesis of Mixed Hardware/Software Systems" Proc. European Design & Test Conference (ED&TC), Paris, France, IEEE CS Press, March 1995.

[10] IEEE Standard VHDL Language Reference Manuel, 1988.

[11] A. A. Jerraya, H. Ding, P. Kission, M. Rahmouni, "Behavioral synthesis & Design Re-use with VHDL". Kluwer Academic Publishers, To appear 1996.

[12] Xilinx, The Programmable Gate Array Data Book, 1994.