Implementing Fuzzy Control Systems Using VHDL and Statecharts

Valentina Salapura, Volker Hamann

{valentina,volker}@vlsivie.tuwien.ac.at

Technische Universität Wien
Treitlstraße 1–182–2
A–1040 Wien
AUSTRIA

Abstract

In this paper, we propose an approach for designing fuzzy controllers. To reduce design time, we employ two high-level design methods: VHDL and VHDL-based logic synthesis, and Statecharts with a VHDL backend for graphical design description. A fuzzifier and a defuzzifier parts of a fuzzy control system are captured in VHDL, as these parts perform complex arithmetical operations. A rule base of the controller is described in Statecharts, and then is translated into VHDL. A complete description of the system is assembled in VHDL, and is synthesized using VHDL-based logic synthesis. The efficiency of the generated hardware is explored for FPGA technology.

1 Introduction

Today, still only a few full custom or semi-custom integrated fuzzy controllers exist and most of them are assembled from standard cells at the gate level. Our design approach presented here is a high-level one. The usage of high-level modeling methodologies for modeling fuzzy controllers reduces development time significantly, making rapid design of custom fuzzy hardware possible.

VHDL [5] for design capture and VHDL-based logic synthesis are an efficient method for designing complex hardware. However, for describing regular structures like finite-state machines, a different approach is more appropriate. For describing such structures, we employ Statecharts. In addition, a commercial tool based on Statecharts incorporates a VHDL generation facility for generating synthesizable code.

We have employed the Statecharts formalism for capturing a rule base of a fuzzy control system. Using this high-level approach, design time is reduced significantly, and different designs of the rule base can be explored in the short time. The fuzzifier and defuzzifier parts of the system, which incorporate a lot of mathematical computations, are described in VHDL as a hand-coded design. We assemble the fuzzy control system, and synthesize a gate level description for field-programmable gate array (FPGA) technology.

This paper is organized as follows: in section 2 we give a brief introduction to fuzzy controllers. Section 3 lists design choices we have made, and section 4 brings a description of the design flow. Section 5 shows how is a rule base of a fuzzy controller described in Statecharts. A fuzzifier and a defuzzifier of the controller are modeled in VHDL, as presented in section 6, and section 7 brings the translation of the rule base to VHDL. Section 8 outlines the testbench generation. The developed model is synthesized and implemented as sketched in section 9. We draw our conclusions in section 10.

2 Fuzzy controllers

In this section, we introduce a simple controller for a gas heater system, which serves as an example to the various steps involved when designing a fuzzy controller. The controller is sensitive to two input variables delivered by sensors: the room temperature and, assuming there is no stable gas quality, the current gas heating value. The controller's output influences the valve opening angle for the gas supply via an active element, e.g., a servo motor.

A membership function gives the degree of membership of an input value to every fuzzy set. The input may belong to more than one fuzzy set. In the example application, the sensed room temperature is divided into three overlapping fuzzy sets called "cool", "tepid", and "warm". A similar classification and membership assignment are made for the gas quality, namely "poor", "medium" and "high".

In figure 1, we show the calculation of the output variable *valve angle* when concrete values are assigned to the inputs

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

temperature (denoted i_1) and *quality* (denoted i_2). The value i_1 is a member of fuzzy sets "cool" and "tepid" with membership degrees $f_1^1(i_1)$ and $f_1^2(i_1)$. For the value i_2, only the fuzzy set "medium" is relevant with a membership degree $f_2^2(i_2)$.

The fuzzified input variables are inputs to the rule base. The sample controller uses nine rules, displayed in table 1, to find an output fuzzy set. Rule 32, for example, is interpreted in full English text as:

`if` *temperature* `is` *tepid* `and` *gas quality* `is` *medium* `then` *valve angle* `is` *half*

In our example, the rule R21 selects the output fuzzy set "large" with the membership degree $min(f_1^1(i_1), f_2^2(i_2))$ and the rule R32 the output fuzzy set "half" with the membership degree $min(f_1^2(i_1), f_2^2(i_2))$.

The membership degrees delimit the output set defining an area. The solution fuzzy set is composed by combining all such areas, and taking the ordinate of the gravitational center of the area.

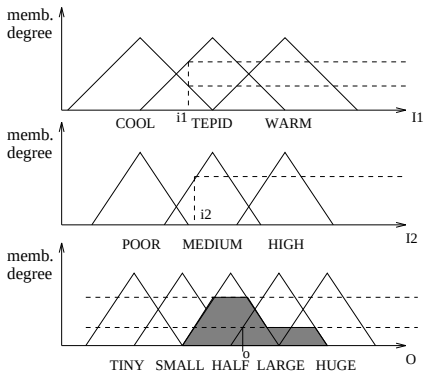

Figure 1. Input and output fuzzy sets.

Rule	Temperature	Gas quality	Valve angle
R11	COOL	POOR	HUGE
R21	COOL	MEDIUM	LARGE
R22	TEPID	POOR	LARGE
R31	COOL	HIGH	HALF
R32	TEPID	MEDIUM	HALF
R41	TEPID	HIGH	SMALL
R42	WARM	POOR	SMALL
R51	WARM	HIGH	TINY
R52	WARM	MEDIUM	TINY

Table 1. Fuzzy rule base for the gas heater control.

3 Design choices

We have implemented a dedicated fuzzy chip, capable of performing stand alone operations, rather then to extend some general purpose processor with fuzzy instructions. Our decision is influenced by several facts:

- a dedicated chip offers the highest execution speed,
- design time is short, due to the usage of synthesis and high-level design tools,
- low cost of real-estate.

For designing a rule base of a controller, we have used a high-level modeling approach instead of encoding it directly in VHDL. The reasons for this choice are:

- a high-level design approach reduces the design time,
- global design functionality is evaluated in a short time,
- different design choices are quickly explored,
- edit-compile-debug cycle is fastened [4],
- the design is represented graphically in a natural and understandable way,
- translation of a high-level model into VHDL is supported, so that VHDL-based logic synthesis can be employed.

According to Costa et al. [1], in describing a fuzzy controller in VHDL as hand design, the most time consuming process is writing a synthesizable VHDL description at the register-transfer level. The usage of high-level design tools reduces these efforts.

For high-level modeling, we have selected Statecharts, because of

- existence of tools supporting system modeling in Statecharts,
- support for validating Statecharts models,
- support for converting Statecharts models in VHDL,
- efficiency of Statecharts for hardware modeling.

Modeling a rule base of a fuzzy control system in Statecharts is efficient and simple, because of the rule base regularity. The graphical presentation of a rule base is easily understandable, and easier to survey and adapt than a VHDL description. Additionally, the built-in simulator of the front-end tool supports the quick validation of the actual model. Using the Statecharts formalism we were able to explore several different rule bases in a short time. Simulating the Statecharts model under a Statecharts simulation environment, we have fine-tuned the rules, and then picked the best rule base for hardware implementation.

A fuzzifier and a defuzzifier of a fuzzy control system perform mathematical operations, such as multiplications and divisions. For describing such parts, direct encoding in VHDL at the register-transfer level is more efficient. In

VHDL, the minimal number of multipliers, adders and dividers can be instantiated, resulting in a more efficient realization.

We have used field-programmable gate arrays (FPGAs) as hardware platform, because they are complex enough to be used for rapid prototyping of complex systems, and for the final implementation of a chip, if only a small number of pieces is needed. They offer more flexibility than ASICs, as when a design is not needed any more, the chip can be reprogrammed for a new hardware.

4 Design flow and environment

The rule base and the fuzzifier and the defuzzifier are modeled in parallel and separately in different design environments. For designing the rule base we have used the Statecharts-based SPeeDCHART design tool [2], whereas for validating the fuzzifier and the defuzzifier, and for integrated system simulation we employ the Synopsys VHDL system simulator [8].

The Statecharts model is converted to a VHDL code using the export facility of SPeeDCHART. The VHDL models of all system parts are integrated, and further system simulation and debugging are performed under the VHDL environment.

We have used the Synopsys VHDL design analyzer/FPGA compiler [7] for logic synthesis. Additionally, we have used the XSI Xilinx/Synopsys interface [11] and X-BLOX as cell generator (XACT 5.1). The target technology is Xilinx XC4000 FPGAs [10].

5 A Statechart model of the rule base

Each rule of the base has the form:

$$\texttt{if } p_1 \texttt{ and } p_2 \texttt{ and } \ldots \texttt{ and } p_n \texttt{ then } C$$

where p_i, $1 \leq i \leq n$ denotes the premises of this fuzzy rule, and C is its conclusion. As soon as all fuzzified inputs are available, the rule is applied.

We represent every rule with a substate `rule_ij`, as shown in figure 2. The state C_{ij} is entered as soon as all input variables are available, and if all premises of the rule are satisfied. The membership degree of the output fuzzy set is calculated in this state, by finding the minimum of the premises' membership degrees.

Rules with the same conclusion are grouped. If any conclusion of this subset of rules is reached, the transition to the state O_i is taken. The conclusive membership degree of the class i is calculated as the maximum of all conclusive membership degrees of the rules in the group.

A part of the Statecharts description of the rule base for the gas heater application is shown as hierarchical decomposition in figure 3, produced by the SPeeDCHART tool. Five statecharts, one for each rule group, activate the output fuzzy sets. Rules of the same class are described at the next lower hierarchical level. The membership degrees of the result sets are calculated as indicated.

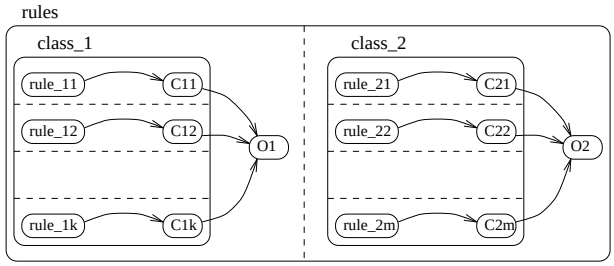

Figure 2. Statecharts model of a rule base.

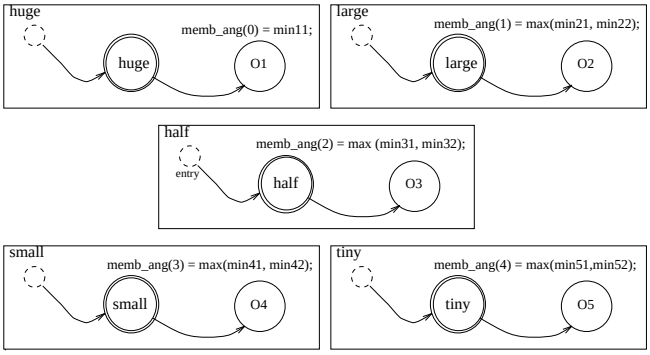

Figure 3. Statecharts model of the rule base of the gas heater controller.

6 VHDL models of the fuzzifier and the defuzzifier

While the rule base is described in Statecharts, the fuzzifier and the defuzzifier used in our fuzzy controller have been described in hand-optimized VHDL.

Inputs to the fuzzifier are input variables (8 bit wide), and outputs are vectors containing membership degrees for all input fuzzy sets. To each input fuzzy variable corresponds one output vector, whose range depends on the number of fuzzy sets (number of fuzzy sets multiplied by 8).

A part of the VHDL description of a fuzzifier is given in figure 4. In the architecture body, membership degrees of the input signal are calculated for each fuzzy set. In an early version, we have added inputs for changing the ranges of fuzzy sets. So, we could change the ranges of fuzzy sets without having to recompile the source code. Once the ranges of the input sets are determined, these inputs are removed from the entity declaration, and are declared as constants.

The generated hardware contains several comparators, an adder and a multiplier. To handle several input variables, two approaches are possible: a) for each input variable, a fuzzifier handling one input variable is instantiated, or b) a single component is used for fuzzification of all input variables,

```
Proc_fuzzifier: PROCESS (in_var)
   BEGIN
IF reset = '1' THEN
    memb_var => (Others <= '0');
ELSIF clock'EVENT AND (clock = '1') THEN
  IF (in_var < tm1m) THEN
    tmp :=conv_std_logic_vector(unsigned(unsigned(in_var) - unsigned(start)) * unsigned(delta), 8);
    memb_var(7 downto 0) <= tmp;
    memb_var(set_nr*8-1 downto 8) <= (Others => '0');
  ELSIF (in_var < tm2m) THEN
    tmp := conv_std_logic_vector(unsigned(unsigned(in_var) - unsigned(tm1m)) * unsigned(delta), 8);
    memb_var(7 downto 0) <= conv_std_logic_vector(1 - unsigned(tmp), 8);
    memb_var(15 downto 8) <= tmp;
    memb_var(set_nr*8-1 downto 16) <= (Others => '0');
  ELSIF (in_var < tm3m) THEN
        ...
END PROCESS;
```

Figure 4. Part of the VHDL description of a fuzzifier for one input variable.

```
Add_and_divide: PROCESS(clock, reset)
   BEGIN
IF reset = '1' THEN
   nomin_reg <= (Others => '0');  denomin_reg <= (Others => '0');
ELSIF clock'EVENT AND clock = '1' THEN
  IF (present_step = First) THEN
    nomin_reg <= unsigned(nomin_reg) + unsigned(conv_std_logic_vector
                 (unsigned(memb_var(1*8-1 downto (1-1)*8)) * unsigned(an1m), 16));
    denomin_reg <= unsigned(denomin_reg) + unsigned(memb_var(1*8-1 downto (1-1)*8));
   ELSIF (present_step = Second) THEN
          ...
   ELSIF (present_step = Divide) THEN
     out_var <= nomin_reg / denomin_reg;
   END IF; END IF;
END PROCESS;
```

Figure 5. Part of the VHDL description of a defuzzifier. The division unit is implemented as a function.

multiplexed in time. The decision which approach to use depends on area and performance constraints.

The defuzzifier is modeled in a similar way. Inputs to the defuzzifier are vectors, one for each controlling variable. The range of the vector is determined by the number of the solution fuzzy sets – it contains an 8 bit membership degree for each output fuzzy set. The defuzzifier determines the gravitational center of the solution fuzzy set, and delivers the results as an 8 bit output control signal.

In implementing the defuzzifier, we have used the method proposed by Yager [3]. This approach requires $2N$ additions, N multiplications, and one division, where N is the number of output fuzzy sets of a single output variable. We have reduced the hardware to two adders, one multiplier, and one divider. The adder and the multiplier are multiplexed in time to perform $2N$ additions and N multiplications.

In our application with one output variable and five fuzzy sets, we have to perform five additions and multiplications, before the division can start. The division unit is implemented to be fast, at the cost of chip area. We have implemented a 17 by 9 bit division as eight subtractions performed successively.

A part of the VHDL description of the defuzzifier is listed in figure 5. The input is a membership set vector (`memb_var`) and the output is the control variable (`out_var`) 8 bit encoded. A finite state machine controls five successive additions and multiplications, and then starts the dividing unit.

7 A VHDL description of the rule base

The high-level description of the rule base in Statecharts is translated to synthesizable VHDL code. The conversion is performed because of the following reasons: having all parts of a system in the same language makes the verification of the whole system possible, and logic-level synthesis tools do not support synthesis from a graphical description.

A part of the code generated from the Statecharts description of the rule base is given in figure 6: This code evaluates rule R21 from the rule base. In process `Rule_21`, the calculation of the minimal membership degrees of the premises

```
Rule_21: PROCESS (memb_temp, memb_qual, state_large_1)
    BEGIN
       CASE state_large_1 IS
          WHEN S21b =>
             min21<= min(memb_temp(COOL), memb_qual(MEDIUM));
          WHEN others => null;
       END CASE; END PROCESS;
Out_calc: PROCESS(..., min21, min22, out_large, ...)
    BEGIN
       CASE out_large IS
          WHEN o2 =>
             memb_ang(1) <= max( min21, min22 );
          WHEN others => null;
       END CASE; END PROCESS;
```

Figure 6. A part of the VHDL description of the rule base.

of rule 21 is performed. Rule R21 together with rule R22 belongs to the rule group with the conclusion "large". In process `Out_calc`, the maximum calculation for this rule group is performed.

The generated VHDL code is easy to read, and is similar to hand coded VHDL code. A difference is the usage of `case` statements for encoding rules, instead of an `if - then` statement, which may be a more natural choice in the coding process. From the hardware efficiency point of view, both coding styles result in the same hardware realization. Another difference is the partitioning of each rule in distinct processes.

A graphical presentation of a rule base is much easier to understand, as well as to model, than VHDL. The usage of a high-level modeling approach reduces the design time significantly.

8 Simulation and testbench creation

Statecharts are not only suitable for designing state-based reactive systems themselves but can also be used for specifying their working environment and thus may serve as a basis for testbench development. Instead of developing case studies and/or test vectors, the designer creates a model of the environment and steps down the design process in parallel with the controller itself.

The testbench as well as the controller are modeled in the same way. Like for the controller development, Statecharts can be used as a design front-end for the rule base testbench. The controller is then simulated directly in Statecharts or – after VHDL generation – in the VHDL environment.

There is a significant improvement in design time and fault coverage compared to standard testing methods. Using the described method, we were able to implement our design controller in a couple of days. The most time consuming process was the encoding of the fuzzifier and defuzzifier in VHDL. Especially when designing the rule base, the interactive testing and observation of the external behavior is eased considerably.

The proposed method has several advantages:

- the designer is obliged to spend more thought on the environment of the system, almost as much as in a design method based on formal semantics. As he does so in an early design phase, the costs of searching for an optimal solution can be kept small,
- high level design tools encourage the use of Statecharts models not only for the design itself but also for simulation testbenches,
- when reaching the FPGA prototype level, it is much cheaper to test the system prototype by connecting it to another FPGA prototype of the system environment. Especially in an industrial environment where expensive machines are controlled by electronic devices, erroneous behavior in the integration test stage may have disastrous consequences.

9 Synthesis and technology mapping

After design verification, the Synopsys design compiler [7] is used to perform logic synthesis. The result of synthesis is a gate-level description of the controller.

Table 2 shows the results of synthesis of the example fuzzy controller with two input and one output variables, and a rule base consisting of nine rules. The number of fuzzy sets of input variables is three, and for the output variable is five. The resolution of all variables is 8 bits.

The implementation of the fuzzifier instantiates two fuzzification units, one for each input variable. Each unit contains three comparators, an adder and a multiplier. The number of comparators depends on the number of input fuzzy sets. If area constraints are critical, only one fuzzification unit can be used, multiplexed in time, for fuzzifying several input variables. In our design, we have used a separate unit for each input variable. The realization is scalable to any number of variables.

Synopsys	Fuzzifier	Rule Base	Defuzzifier
Number of gates	1756	1062	2025
Number of CLBs	136	72	193
Time delay (ns)	4.75	6.24	4.75

Table 2. Synthesis results. Number of gates is given for the LSI 10k library.

The defuzzifier calculates one output variable for five fuzzy sets. Thus, this implementation performs five multiplications and additions, multiplexed in time, and one division. The implementation contains two adders, a multiplier, and a divider. We have implemented a fast divider unit at the cost of chip area.

A defuzzifier for handling several output variables can be implemented on different ways: a single defuzzifier can be multiplexed in time, separate defuzzifiers can be used for each output variable, or some combination of two. The selection of the hardware configuration depends on the area and performance constraints of the particular design.

We have compiled our models targeting two technologies: Xilinx FPGAs and the LSI 10k ASIC library. For FPGA implementation, the measure area efficiency is expressed in number of CLBs (configurable logic blocks), whereas for ASIC in number of gates.

Comparing our results with the ones of Costa et al. [1] for a simple problem (a fuzzy controller with 2 inputs and one output, a rule base with seven rules, and 8 bit resolution) our implementation is significantly smaller. For the implementation of a fuzzy controller of comparable complexity, we have needed only 4843 gates, versus 9392 gates used in the implementation by Costa. Costa et al. have described their model in VHDL as hand design.

Surman et al. [6] have implemented a fuzzy controller in FPGAs using hand-optimized CLB-level functional mapping. The design shows better results in CLB count, but involves more design effort and is thus more error-prone. Full-custom reconfigurable solutions such as presented by Watanabe [9] make heavy use of repetitive structure elements, but are only feasible in a high number of pieces, offering no flexibility.

10 Conclusion

In this paper, we have presented a modeling approach for describing fuzzy controllers using Statecharts and VHDL. We have outlined a modeling scheme for designing a rule base of a fuzzy controller using Statecharts, whereas the fuzzifier and defuzzifier parts of the system were coded in VHDL.

Using the Statecharts formalism for capturing the rule base of a system has been shown to be powerful. The rule base was captured and verified by simulation using the SPeeDCHART design and simulation environment. A simulation under a graphical environment makes it easy to see which rules are applied. The Statecharts description of the rule base is translated into VHDL, and integrated with the hand-coded VHDL code realizing fuzzification and defuzzification components. Finally, the system model is synthesized to logic level and mapped to FPGAs.

The most distinctive problem coming up when using the introduced method is the lack of back-annotation facilities between Statecharts and VHDL. Once the VHDL code has been generated, there is no way back to the Statecharts level.

One of the advantages of the proposed method is the simplicity of constructing and experimentally fine-tuning the rule base. Taking the standard language VHDL as backend opens all facilities to make use of synthesis tools. Thus, we receive all benefits of rapid prototyping and shortening time-to-market requirements.

11 Acknowledgment

The authors wish to thank to Michael K. Gschwind for his help with various issues related to VHDL and optimization for FPGAs, and in preparation of this article.

References

[1] A. Costa, A. de Gloria, P. Faraboschi, A. Pagni, and G. Rizzoto. Hardware solutions for fuzzy control. *Proceedings of the IEEE*, 83(3):422–434, March 1995.

[2] S. Dart. *SPeeDCHART Reference Manual*. SPEED S.A., Neuchâtel, Switzerland, 1993.

[3] M. Figueiredo, F. Gomide, A. Rocha, and R. Yager. Comparison of Yager's level set method for fuzzy logic control with Mamdani's and Larsen's methods. *IEEE Transactions on Fuzzy Systems*, 1(2):156–159, May 1993.

[4] D. D. Gajski, L. Ramachandran, P. Fung, S. Narayan, and F. Vahid. 100-hour design cycle: A test case. In *Proc. of the European Design Automation Conference Euro-DAC*, Grenoble, France, September 1994. ACM.

[5] IEEE. *IEEE Standard VHDL Language Reference Manual*. IEEE, NY, 1988. IEEE Standard 1076-1987.

[6] H. Surmann, A. Ungering, and K. Goser. Optimized fuzzy controller architecture for field programmable gate arrays. In S. Verlag, editor, *Field - Programmable Gate Arrays*, number 705 in Lecture Notes in Computer Science, pages 124–133, 1993.

[7] Synopsys. *Design Compiler Family Reference*. Synopsys Inc., Mountain View, April 1995. Version 3.3a.

[8] Synopsys. *System Simulator Reference Manual*. Synopsys, Inc., Mountain View, April 1995. Version 3.3a.

[9] H. Watanabe, W. Dettloff, and K. Yount. A VLSI fuzzy logic controller with reconfigurable, cascadable architecture. *IEEE Journal of Solid-State Circuits*, 25(2):376–382, April 1994.

[10] Xilinx. *The Programmable Logic Data Book*. Xilinx, Inc., San Jose, CA, 1994.

[11] Xilinx. *XACT Xilinx Synopsys Interface FPGA User Guide*. Xilinx, Inc., San Jose, CA, December 1994.