2 Fuzzy logic software and hardware

2.1 Introduction

With the growing acceptance of fuzzy logic control, more and more fuzzy logic software and hardware products have appeared in the market. As stated in Chapter 1, one of the main subjects of this report is testing and evaluating the recently released fuzzy logic products of **SGS-Thomson**. These fuzzy logic products consist of the **WARP Software Development Tool** (SDT) and the **WARP Application Development Board** (ADB). The WARP SDT is a tool to develop fuzzy controller software. It runs on PC under Ms Windows. As a Fuzzy Logic Controller (FLC) is designed, the WARP ADB can be connected to the PC by a RS232-C interface cable. Next, the development tool can be used to download any FLC onto the WARP board. The purpose of the board is to control real-time processes, and can be active *without* the assistance of a digital computer.

In order to evaluate the WARP SDT and WARP ADB, one should compare both these products with similar products. Therefore, first a short overview of fuzzy logic software and hardware is given. In Section 2.2, software development tools, not only is discussed WARP SDT (Section 2.2.1), but attention is paid to another software environment, FULDEK (Section 2.2.2), as well. The reason is that FULDEK offers many useful features, from which one can be applied for rule-base reduction (see Section 3.4). In Section 2.3 the features and limitations of the WARP ADB are reviewed. The last Section 2.4 deals with using the board in a real-time system.

The purpose of **fuzzy logic software** tools, mostly used for the development of fuzzy **controllers**, is to take over the implementation problem, and leave only the design problem to the user. Most of the tools can simulate and debug the designed fuzzy systems and are able to generate executable code in for example C-language. Jamshidi *et al.* [1993, 1995] have presented a survey on both fuzzy logic software and hardware. Table 2.1 shows a brief comparison of some recently available fuzzy logic software.

Program	Computer Media	Developer(s)	Special features
TIL-Shell	PC	Togai InfraLogic, Irvine, CA	C code generator (Windows)
FLCG	Macintosh	Univ. New Mexico, Albuquerque, NM	Fuzzy logic C code generator
FIDE	PC	Aptronix, Inc., Palo Alto, CA	Fuzzy logic simulation and code generation (Windows)
fuzzyTECH	PC	Inform Software, Aachen, Germany	Fuzzy logic simulation and code generation (Windows)
FL_Control	PC	Texas A&M U. College Stn., TX	Adaptive and non-adaptive fuzzy controllers
FUDGE	PC	Motorola, Inc., Austin, TX	Fuzzy logic simulation and code generation (Windows)
FULDEK	PC	Bell Helicopter Textron, Inc., Fort Worth, TX	**Complete** FLC simulation and code & **rule** generation (Windows)
WARP SDT	PC	SGS-Thomson, Catania, Italy	Fuzzy logic simulation and code generation (Windows)
O'INCA	PC	Intel. Machines, Inc. Sunnyvale, CA	Fuzzy-neural network simulation environment

Table 2.1: A partial list of fuzzy logic software

Next, an overview of **fuzzy logic hardware** will be presented, but first the reason is given why it exists, why special hardware boards with fuzzy logic chips have been developed.

A digital computer is useful for e.g. design, simulation and debugging of a FLC, but it does not achieve **quick** fuzzy information processing [Yamakawa, 1988] contrary to one's expectations. Yamakawa [1988] claims that there are two reasons for this. The first is the 'bus organized off-chip hardware'. The operating speed of a digital computer is determined by the rate of data transfer but not by the switching speed of transistors used in the system. Therefore, it is desirable that all the building blocks (e.g. CPU, ROM and RAM) are integrated as a semiconductor in monolithic form. The second reason is the architecture of the digital computer, which is based on the 'von Neumann stored-program concept'. The concept completes the fuzzy information processing by repeatedly fetching and executing programs stored in the memory. Although this approach guarantees saving transistors, for real-time fuzzy processing it is preferable to adopt a partly or even a purely parallel signal processing for high-speed operations. In Section 5.4.3 the execution time of FLC's running on the WARP ADB is compared to software FLC's running on a digital computer (PC).

Another great advantage of using a embedded FLC can be mentioned. In most cases a PC or other digital computer is used for the design and simulation of a fuzzy controller, but after the design stage on PC, the fuzzy hardware board is disconnected from the PC and works ‘stand alone’. This means a reduction of material and costs.

Table 2.2 presents an overview on available fuzzy logic hardware boards. The column Program gives the program that is the designing tool, used to develop the FLC before it is downloaded onto a certain board. All programs run on PC, except for SBFC that runs on the board itself. Note that both Table 2.1 and 2.2 do not present exhaustive surveys.

Program	Developer(s)	Hardware
SBFC/TIL-Shell	Togai InfraLogic, Irvine, CA	Single Board Fuzzy Controller board + A/D & D/A chips
Fuzzy micro controller	NeuraLogix, Palo Alto, CA	PC board for RT control
FIDE	Aptronix, Inc., Palo Alto, CA	Board with Motorola chips
fuzzyTECH	Inform Software, Aachen, Germany	Board with Intel chips
WARP SDT	SGS-Thomson, Catania, Italy	Board with WARP chip
FUDGE	Motorola, Inc., Austin, TX	Board with standard Motorola chips

Table 2.2: A partial listing of fuzzy logic hardware

The SBFC board uses the standard Togai fuzzy chip, which can handle 256 inputs, 128 outputs and up to 1000 rules. The board can evaluate up to 200.000 fuzzy rules per second.

2.2 Software development tools

2.2.1 WARP S.D.T. Fuzzy Studio

As mentioned, WARP SDT, called **Fuzzy Studio**, is an environment for easy implementation of fuzzy controllers, and is meant to be used in combination with the WARP ADB. The code generator feature has the purpose to create executable code for simulation. This section provides a global overview of the possibilities offered and how one can work with Fuzzy Studio. For more detailed information about Fuzzy Studio is referred to [SGS-Thomson, 1994a].

Fuzzy Studio requires an Intel type 80386 processor or higher, 2 Mbytes RAM memory, mouse and Windows 3.1 or higher. If the system meets up with these requirements, WARP SDT can be set up.

To start Fuzzy Studio, one has to start up windows and open the 'W.A.R.P.-SDT' group window by double clicking the related icon in the 'Program Manager' window. Next, one has to double click in the 'W.A.R.P.-SDT' group window, the icon with the same name. Now, Fuzzy Studio is started and after a few second the main window is shown. Figure 2.1 describes the Fuzzy Studio main window of a certain project, called 'test1'. Under the projects name, the number of input variables, output variables and rules defined for this specific project are indicated. The different items can be found in the menu bar. The items can be activated either by clicking with the mouse or by pressing the 'ALT' key in combination with the underlined letter of the menu item. Some menu items have drop-down menus of their own, and some of these have further sub-menus. One by one the items of the menu bar will be discussed.

Figure 2.1: Fuzzy Studio main window of project 'test1'

- **File:**

In this drop-down menu no special commands can be found. The commands are about opening (**New** and **Open**) and saving (**Save** and **Save as**) of project files. As one may expect, the **Exit** command can be found here as well.

- **Edit:**

This drop-down menu contains two commands. The command **Variable** opens the *variable editor* (see Figure 2.2), while **Rule** opens the *rule editor* (see Figure 2.3). All variables and membership functions have to be defined in the *variable editor* and can be used to construct the fuzzy rules in the *rule editor*. Obviously, one should first define the variables and then the rules. In both editors definitions are not accepted as they exceed the limitations of the WARP ADB (see Section 2.3). Without a mouse neither of the editors can be used.

The *variable editor* permits one to define the boundaries of the universe of discourse, the resolution (configurable to 6 or 7 bit for input and fixed to 10 bit for the output; see Section 2.3) and the membership functions for each input and output variable of the FLC. Moreover, the variable editor allows a symbolic name to be specified in order to recognize a variable. The membership functions associated to that variable can be defined by calling the *membership function editor* from the *variable editor*, by clicking the 'MF' button. The *membership function editor* allows the user to graphically define the membership functions in any possible shape by using mouse. Once defined the variable, the variable editor shows all characteristic data, the membership functions and the total number of defined variables, as shown in Figure 2.2..

Figure 2.2: The vafiable editor window of project `test1´

The *rule editor* allows the user to define the rule-base by mouse. A rule is valid when it has at most 4 antecedents and 8 consequents and at least 1 antecedent and 1 consequent. The antecedents are connected by either 'OR' or 'AND', being the fuzzy logic union and intersection operation, respectively. One is allowed to use the 'NOT' (complement) operation in combination with any membership function. In the lower part of the rule editor window (see Figure 2.3) the defined rules are listed and just above the number of defined rules is indicated. By clicking the 'New' button a new rule can be added to the rule-base. A new rule can be constructed by simply clicking on the command buttons (e.g. 'If', 'Then', 'And', 'Or', 'Is', and 'IsNot') and the listed variables and membership functions in the right order. As the 'If'-part of a rule is defined, off course one can only chose from the defined the input variables, and likewise, as the 'Then'-part of a rule is defined, one can only chose from the defined output variables. In the rule editor window of Figure 2.3 one can see that a new rule is being defined.

Figure 2.3: The rule editor window of project `test1'

- **Compile:**

This item starts the process of translating the data originated by the editors into a data stream to be loaded onto the WARP memories, thus generating the machine code. The WARP memories are automatically optimized by the compiler.

- **Debugger:**

The **debugger** item starts the debugger. The debugger window will appear on the screen. The debugger allows the user to verify if the fuzzy project is correct and to perform an optimization and tuning of the process behaviour. The debugger can either provide the output values or execute the computation while monitoring the fuzzy computing values. The step-by-step analysis can be performed monitoring the only rules related to any selected output.

- **Exporter:**

The items in this drop-down menu allow the user to translate the in Fuzzy Studio defined FLC into either a Matlab, C or WARP-SIM file, so simulations can be done in one of these three environments.

- **Board:**

The items that can be chosen in this drop-down menu, are to communicate and control the WARP ADB. **Port Setting** allows the user to define to which port the WARP board is connected by the RS232-C cable. Clicking on **Download** makes a sub-menu popping up. **WARP Download** provides macine code file, generted by the **Compiler** item, to be downloaded into the memories of the WARP board and **EPROM Download** accomplishes dowloading of the compiled FLC into the EPROM's. Obviously, **EPROM to WARP** downloads the data stored in the EPROM's into the WARP chip. For the differences between the WARP memories and the EPROM chips is referred to Section 2.3. If **EPROM Verification** is chosen the contents of the EPROM's will be checked. If the EPROM's are correctly downloaded, all data matches with those ones of the current project. **Pattern Test** allows the user to provide values for the input variables, after what the WARP chip is ordered to calculate the corresponding values of the output variables. **Stream test** generates a stream of values of the output variables corresponding to a stream given by the user, consisting of input combinations for the FLC.

2.2.2 FULDEK

FULDEK, standing for FUzzy Logic DEvelopment Kit, is a more complete fuzzy logic tool than WARP SDT Fuzzy Studio. The program requires PC with at least 2 Mbytes RAM memory and Windows 3.1 or higher. An 80286 processor or higher and a math co-processor are recommended. In this sub-section all menu's and items are not discussed, because FULDEK is beyond the scope of this work. However, of all fuzzy logic SDT's, FULDEK offers probably the most extensive set of possibilities. Here, the features of FULDEK are shortly reviewed, so one can form a idea what useful options are possible in fuzzy logic software, besides these that are offered by Fuzzy Studio. It is noted that there is no specially developed fuzzy control hardware cooperating with FULDEK. Real-time control has to be performed on PC by generated code.

A first distinguishing feature of FULDEK is that within this environment complete simulations can be accomplished. FULDEK allows the user to define both a linear model of any arbitrary system and external disturbances injected to this system. The for this system designed fuzzy controller can be linked to the linear model. After the step time, final time and initial values are defined, a closed loop simulation can be performed and the results are shown in graphs.

The FULDEK code generator can generate code for a large set of computer languages (ADA, BASIC, C and FORTRAN). Furthermore, Matlab code can be generated. Although real-time control can not be performed by Matlab, Matlab code is extremely useful for simulation if one deals with (highly) NON-linear systems. In that case FULDEK itself can not be used for simulation.

An interesting feature (see Section 3.4) is FULDEK's **Autorule** function. This function provides a complete fuzzy system using observed data. The observed data are input/output combinations of any 'black-box' system. Therefore, this function make it possible to identify and model a non-linear system by a fuzzy system. For every variable the number of membership functions can be chosen. The more membership functions are chosen, the more precise the observed data is approximated, but the larger the rule set. A comparison between the data points and fuzzy system makes validation of the approximation possible.

The **Map** option makes visualisation of a fuzzy system possible. The control behaviour can be plotted in both 3-D, showing the control surface, and 2-D, giving a so-called contour map. Phase-plane plots are also possible. Implementation is facilitated by allowing using standard membership functions, e.g. singleton, rectangular, triangular, trapezoidal and Gaussian. Different inference methods and different defuzzification methods can be chosen.

2.3 WARP Application Development Board

In this section the WARP board and its possibilities will be discussed. The heart of the board is the Weight Associative Rule Processor. The most important features of this chip are summarized in Table 2.3 [SGS-Thomson, 1994c].

Number of inputs	Up to 16 configurable variables
Number of outputs	Up to 16 variables
Input data resolution	Configurable to 6 or 7 bit
Output data resolution	10 bit
Antecedent's MF's number	Up to 16 for an input variable (configurable from 8 up to 16)
Consequent's MF's number	Up to 128 for all output variables
Standard rule format	Up to 4 anticedents, 1 consequent or subsets
Number of rules	Up to 256
Inference method	MAX-DOT

Table 2.3: WARP configuration settings

In the table above one can notice that the resolution of the input variables is configurable. In the 7 bit setting, the membership functions use twice as much memory as in the 6 bit setting, and therefore less membership functions in total can be used. There are 4 membership function memory blocks. Each of them can be configured in 5 ways, explained by Table 2.4.

Numbers of input	Data resolution	Number of MF's for each input variable
1	7 bit	16

Numbers of input	Data resolution	Number of MF's for each input variable
2	7 bit	8
2	6 bit	16
3	6 bit	2x8 + 1x16
4	6 bit	8

Table 2.4: Different input variable configuration per memory block

If one wants to use the maximum of 16 input variables, then every variable must be of the 6 bit resolution and just up to 8 membership funtions can be associated to each variable.

In Appendix A.4.3 the MAX-DOT inference method is described. In the formulas realizing the different defuzzification methods it does not make any difference whether one uses the MAX-DOT or the MAX-MIN inference method [Driankov *et al.*,1993]. However, the first method costs considerably less computation time, and therefore most fuzzy chips apply this method.

About the used WARP chip defuzzification method a few more words should be said, since the manual [SGS-Thomson, 1994c] explains the procedure, but is unclear which method this represents. From the description in the user manual of the processor, one can easily conclude that it is the 'centre of sums' method (see Appendix A.4.4) that is used. The defuzzification method is described by the following equation:

$$X_{crisp} = \frac{\sum_{1}^{n} A_i X_i}{\sum_{1}^{n} A_i} \tag{2.1}$$

where n is the number of membership functions that determine the output (thus the rule 'truth'-value, calculated by the MAX-DOT inference method, is greater than zero), A_i is the weighted area of the consequents membership function and X_i is the centroid of each of these. X_{crisp} is the crisp value of the output variable. In Figure 2.4 an example is presented in the case of two membership functions determining the output, i.e. $n = 2$.

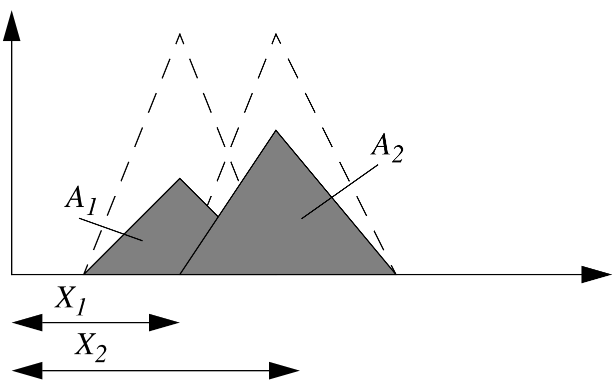

Figure 2.4: An example of a MAX-DOT inference conclusion

If equation (2.1) is applied to Figure 2.4 one notices that the overlap of both membership functions is taken in account twice. Therefore, the method that determines the crisp output is in fact the ‘centre of sums’ method. Interesting to mention is that one might prefer the ‘centre of sums’ method. (Driankov *et al.* [1993] are in favour of this method.)

In the last part of this section, the board itself is reviewed. In Figure 2.5 the board is represented by a schematic diagram.

.

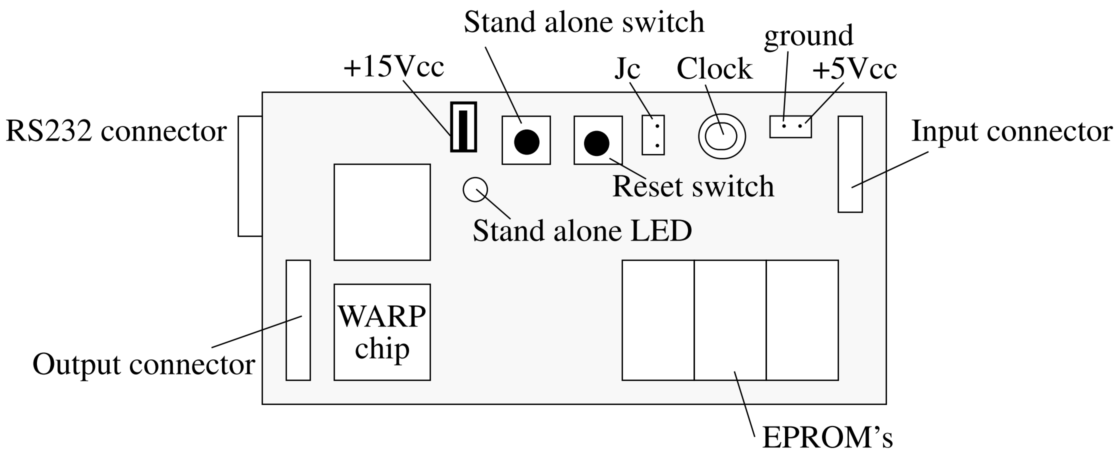

Figure 2.5: Schematic diagram of WARP ADB

It may be obvious that ‘+5Vcc’ and ‘ground’ are the connectors for the power supply.

If the board is used in combination with the WARP SDT, e.g. for downloading a designed program onto the board, the PC and the board should be connected by an RS232-C connection cable.

After switching off the power supply of the board the WARP memories are erased. Therefore EPROM’s have been placed on the board. The EPROM’s (3 chips) can contain one complete WARP memory. If a FLC has to be downloaded onto the EPROM’s, the ‘+15Vcc’ should be connected.

The EPROM’s can be removed from their sockets and therefore, one can build a library of FLC’s memorized in different sets of EPROM’s. As the power supply is switched off, the EPROM’s in the sockets can be simply replaced by another set of EPROM’s containing another project. However, pay attention that the EPROM’s are allways put on the board in the same order as their programming order.

To transfer a FLC from a programmed set of EPROM’s into the memory of the WARP chip, one has to follow the next procedure. Connect the 5V power supply. Now, by pressing the ‘Reset switch’ the EPROM’s are downloaded into the WARP chip. After these actions, one can start the WARP ADB by pressing the ‘Stand alone switch’. The board is now working by itself, what is indicated by the green ‘Stand alone LED’, which will be activated. During successful downloading from PC onto WARP chip or EPROM’s this green LED will be blinking.

If jumper ‘Jc’ is closed the board works by its built-in 24Mhz clock. If ‘Jc’ is open, the board uses the clock signal connected to the coaxial connecter ‘Clock’. The board can deal with clock frequencies up to 40 Mhz.

The 14 pins ‘Input connector’ and the 20 pins ‘Output connector’ will be discussed in the next section.

2.4 WARP A.D.B. in a real-time system

In a real-time control system, the WARP board has to be connected with the system to be controlled and be put in the ‘stand alone’ mode . After a specific start pulse, that must be generated at the desired sampling time, the WARP board will read the input variable(s) and will calculate and generate the control signal(s). As shown in Figure 2.5 the board has just one input connector and one output connector. In case more than one input variable is defined, the input variables have to be offered to board in a serial way and if more than one output is generated, the outputs have to be read one by one. This means that in these cases an input Inter-Face (IF) and an output interface is needed, respectively.

First, the input acquisition is discussed. Figure 2.6 describes this phase.

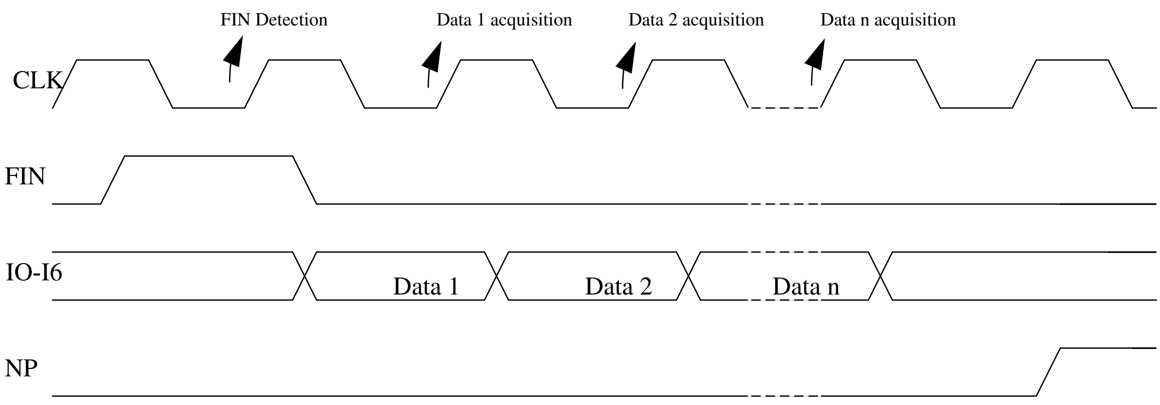

Figure 2.6: Graphical representation of the acquisition phase

In Figure 2.6 ‘CLK’ is the internal or external clock signal. Usually, one will connect an external clock, because the input IF, the WARP board and the output IF have to be synchronized. Notice that the board works with positive-edge triggered timing.

In this figure ‘I0-I6’ represent the 7 bit input variable (pin 1 to 7 of the input connector). Notice that the board only can deal with digital input data. If ‘I0-I6’ has the value zero, the WARP board reads it as the lower boundary of the universe of discourse. The upper boundary is the value 127 in case of a 7 bit variable and 64 for a 6 bit variable. The remaining values are equally distributed over the universe of discourse. Notice that the input value has to be normalized between 0 and 127 for a 7 bit variable and between 0 and 64 for a 6 bit variable.

‘FIN’ is the label of pin 11 of the input connector and ‘NP’ is the label of pin 17 of the output connector. (for a full description of the pin assignments of both connectors see [SGS-Thomson, 1994b]).

‘NP’ is low means that a new cycle of acquisition, calculation and output generation can be started. To start the acquisition phase FIN has be high for one positive-edge of the clock. The next positive-edge the value ‘Data 1’ of the thirst input variable has to be ready on ‘I0-I6’. The following positive-edge ‘Data 2’ has to be ready, etc. The board will read as many values as the number of input variables defined for the specific controller in the WARP memories. After the board acquired the last value, ‘NP’ is set to high and the calculation of the output signals is started.

The user of the board has to develop a digital circuit to perform the above mentioned actions. In Section 5.? an example of an input interface is discussed.

The output-generation procedure is described by Figure 2.7.

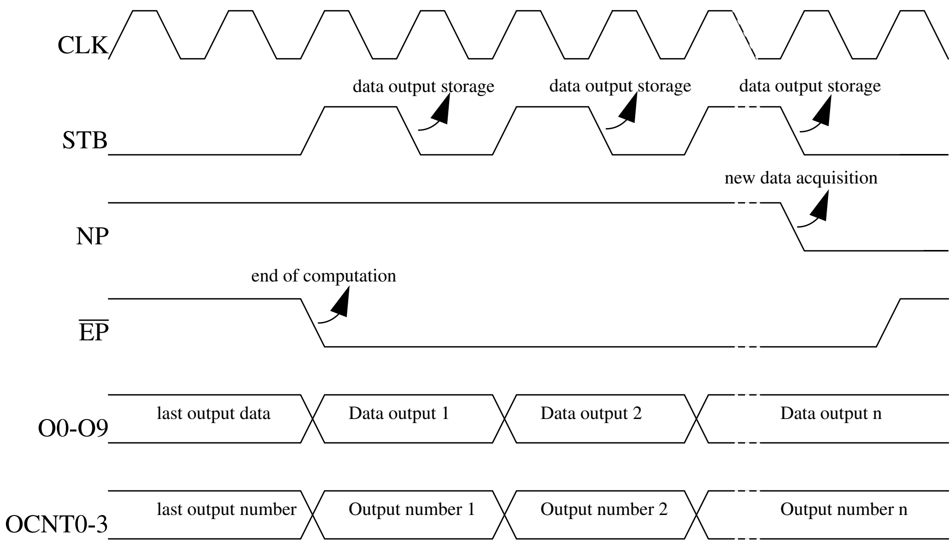

Figure 2.7: Graphical representation of the output generation

As the calculation is completed the ‘$\overline{EP}$’ signal (pin 17) is set low. From now on, at every negative-edge of the ‘STB’ signal (pin 18) the next ‘Output number’ (‘OCNT0-9, pin 1 to 4) and its value (‘O0-O9’, pin 5 to 14) can be obtained. After the last output variable is generated, ‘NP’ is set to low to indicate that the board is ready for the next acquisition phase. Notice that the output ‘O0-O9’ holds the value of the last output variable. Again, zero represents the lower boundary of the universe of discourse and 2^{10}-1 =1023 the upper boundary.

A digital circuit is needed to extract the output signals from the WARP board. Before the control signal is fed to the system a denormalisation or D/A conversion might be needed.

Figure 2.8 shows a block diagram of the WARP board in a real-time control system. The input IF might have to accomplish more functions than normalisation and multiplexing. The sensor signals might need to be converted from analog to digital and filtering of the sensors might be necessary.

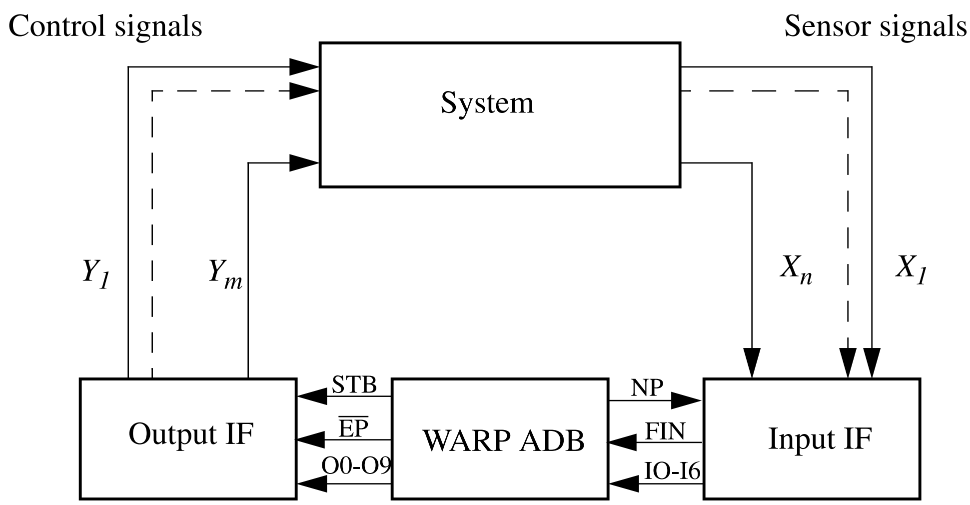

Figure 2.8: Block diagram of WARP ADB in a real-time system

In the manual [SGS-Thomson, 1994c] an example is given of the execution time for one complete cycle. If the downloaded FLC has 16 inputs, 8 outputs and 28 rules and the board works at a frequency of 40 Mhz, the elapsed time after one cycle is 810 ns. That means that for this situation in one second 34,567,901 rules can be evaluated, considerably more than the SBFC board of Togai is capable to. In section 5.? some practical results of measurements of the WARP board execution time can be found.

One has to realize that on one WARP board several FLC's can be active at the same time. One simple example is given to explain this point. Suppose 3 FLC's are needed to control a certain industrial process. All of the FLC's have 2 inputs. Two fuzzy controllers have 1 output and 25 rules, the remaining one has 2 outputs and 50 rules. The realisation is graphically represented by Figure 2.9.

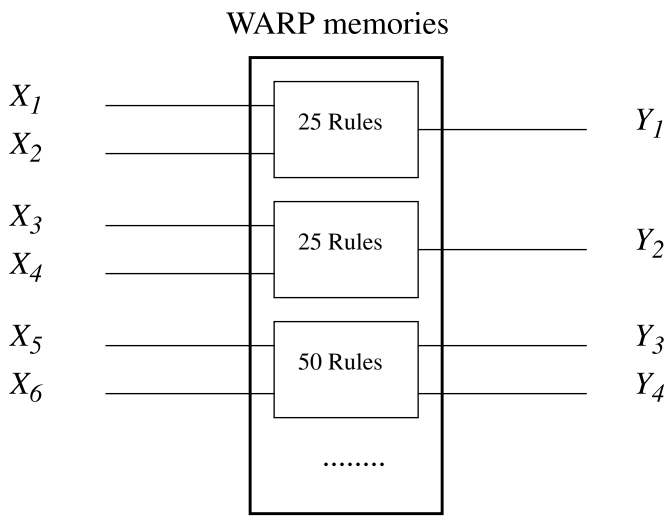

Figure 2.9: An example of a multiple FLC structure

Figure 2.9 shows that one WARP ADB can deal with this particular control problem. Notice that the capabilities of the WARP board are even just partially used, i.e. just 6 of the 16 inputs, 5 of the 16 outputs and 100 of the 256 rules.