Democratic and Popular Republic of Algeria

Ministry of Higher Education and Scientific Research

university saad dahlab of blida

Faculty of Technology

Department of Electronics

Final year project thesis

presented by

GAROUDJA Lyes

&

HAOUHACHE Mohamed El_Hadi

With the requirements of the Master's degree in Automatic Control

Theme

68HC12 based fuzzy logic controller for an inverted pendulum

Under the supervision of : Boualem KAZED

academic year 2012-2013

ACKNOWLEDGEMENT

I thank Allah for helping and guiding us for the completion of this work.

Our first and foremost regards are addressed to **Prof. Kazed Boualem**, Dept. of Electrical Engineering for assigning us this project. His supportive nature, continuous guidance and constructive ideas were really valuable during every stage of this project.
Then, we express our deep thanks to my family, especially our parents, for their patience and moral support during the most important stages of this work.
Finally, we sincerely convey our gratitude to the students and all the staff members of the Department of Electrical Engineering.

HC12

المتحكم الدقيق HC12، المنطق المبهم

Résumé : Au nom de Dieu le miséricordieux, prière et paix soient sur ses Prophètes, avec la louange de Dieu il a été possible de réaliser ce projet de fin d'études intitulé: contrôle d'un pendule inversé par la logique floue dans le microcontrôleur HC12. Ce travail consiste à implémenter un contrôleur flou pour stabiliser la position d'un pendule inversé, l'une des caractéristiques principales du microcontrôleur 68HC12 est la présence, dans son jeu d'instructions, de fonctions particulières propres aux calculs intervenant dans l'implémentation d'un contrôleur flou.

Mots clés : microcontrôleur 68HC12, logique floue, pendule inversé

Abstract :

In the name of God the merciful, prayer and peace be upon His prophets, this final year project study entitled: 68HC12 based Fuzzy Logic Controller for an inverted pendulum, was completed. The aim of this project is to implement a fuzzy controller to stabilize the position of an inverted pendulum, one of the most important features of the 68HC12 microcontroller is the presence of special hardware instructions specifically designed to implement a fuzzy logic controller.

Keywords: 68HC12 microcontroller, fuzzy logic, inverted pendulum.

Acronyms and Abbreviations List

ADC: Analog to Digital Converter.

COP: Computer Operating Properly watchdog.

CPU: Central Processing Unit.

DAC: Digital to Analog Converter.

DC: Direct Current.

Derror: the Difference between two consecutive errors.

DIR: DIRect.

EMIND: smaller two unsigned 16 bit values in accumulator D.

EMACS: Extended Multiply and ACcumulate Signed 16 bit to 32 bit.

EMAXM: larger of two unsigned 16 bit values in Memory.

ETBL: Extended Table Lookup and interpolate.

EXT: extended.

FLC: Fuzzy Logic Controller.

IDX: Indexed.

INH: Inherent.

LIM: Lite Integration Module.

LQFP: Low profile Quad Flat Pack.

LSB: Least Significant Bit.

MAX: MAXimum.

MAXM: larger of two unsigned 8 bit values in Memory

MCU: Microcontroller Unit.

MEM: Membership.

MIN: Minimum.

MINA: smaller two unsigned 8 bit values in accumulator A.

MSB: Most Significant Bit.

N: Negative.

NB: Negative Big.

NRZ: Non Return to Zero.

P: Positive.

PB: Positive Big.

PLL: Phase Locked Loop.

PRO32CIDTJ1328t1 0 0 1 149.06 711.94 TmE8 RodCIDTJ1328t1 0 0 1 1495111.94 TmE8uc.94urig.

Contents table

Figures List

Tables list

Introduction

The inverted pendulum is among the most difficult systems to control in the field of control engineering, due to its relative complexity, it is widely used as test bench to analyze the performance of different types of controllers. In order to ensure the stability of the pendulum, two DC motors will make a mobile platform move forward and backward with a variable speed, this will be calculated in an embedded system based on a dedicated microcontroller. The control algorithm will be implemented using the fuzzy logic approach. In this project we have been asked to implement a fuzzy logic controller to stabilize the position of an inverted pendulum using a special microcontroller for this purpose. This microcontroller has the advantages of having special fuzzy logic instructions (MEM, REV Etc…) within its instruction set. The HC12compact board is a universal microcontroller module based on this microcontroller. Furthermore this board has many other peripherals that can be used to communicate with different kinds of analog or digital systems. For applications that require extensive memory storage this module is equipped with additional static and EEPROM memory. One of the most important features of this module is the availability of a bootloader preprogrammed on the microcontroller, this is not only used to download the compiled program on the chip but also enable us to monitor the content of different parts of the memory.

The role of the HC12 is to provide the control signal to another board, based on another microcontroller (ATMEGA 328), which will be used to transform this control signal in terms of duty cycle that will supply the power stage with the right signal to both the motors to make the pendulum on stable vertical position. The required current for the motors is supplied through a double H-bridge, connected so that a single PWM signal is necessary to rotate the motors in one direction or the other. This has been made possible by using a Schmidt trigger type inverter which provides the opposite PWM signal used as the second input signal for the H-Bridge.

Chapter1 Microcontroller MC68HC812A4, HC12 compact and Arduino(UNO)

1.1 Introduction :

The goal of this chapter is to describe the MC68HC812A4 microcontroller, the HC12 compact and the Arduino UNO features.

1.2 MC68HC812A4 microcontroller

1.2.1 Introduction

The MC68HC812A4 microcontroller unit (MCU) is a 16-bit device composed of standard on-chip peripheral modules connected by an inter-module bus.

Modules include:

• 16-bit central processor unit (CPU12).

• Lite Integration Module (LIM).

• Two asynchronous serial communications interfaces (SCI0 and SCI1).

• Serial peripheral interface (SPI).

• Timer and pulse accumulator module.

• 8-bit analog-to-digital converter (ATD).

• 1-Kbyte random-access memory (RAM).

• 4-Kbyte electrically erasable, programmable read-only memory (EEPROM).

• Memory expansion logic with chip selects, key wakeup ports, and a phase-locked loop (PLL).

1.2.2 Features

Features of the MC68HC812A4 include:

• Low-power, high-speed M68HC12 CPU.

• Power-saving stop and wait modes.

• Memory:

– 1024-byte RAM.

– 4096-byte EEPROM.

– On-chip memory mapping allows expansion to more than 5-Mbyte address space.

• Single-wire background debug mode.

• Non-multiplexed address and data buses.

• Seven programmable chip-selects with clock stretching (expanded modes).

• 8-channel, enhanced 16-bit timer with programmable prescaler :

– All channels configurable as input capture or output compare.

– Flexible choice of clock source.

• 16-bit pulse accumulator.

• Real-time interrupt circuit.

• Computer operating properly (COP) watchdog.

• Clock monitor.

• Phase-locked loop (PLL).

• Two enhanced asynchronous non-return-to-zero (NRZ) serial communication interfaces (SCI).

• Enhanced synchronous serial peripheral interface (SPI).

• 8-channel, 8-bit analog-to-digital converter (ATD).

• Up to 24 key wakeup lines with interrupt capability.

• Available in 112-lead low-profile quad flat pack (LQFP) packaging.

The figure 1.1 show the MC68HC812A4 block diagram:

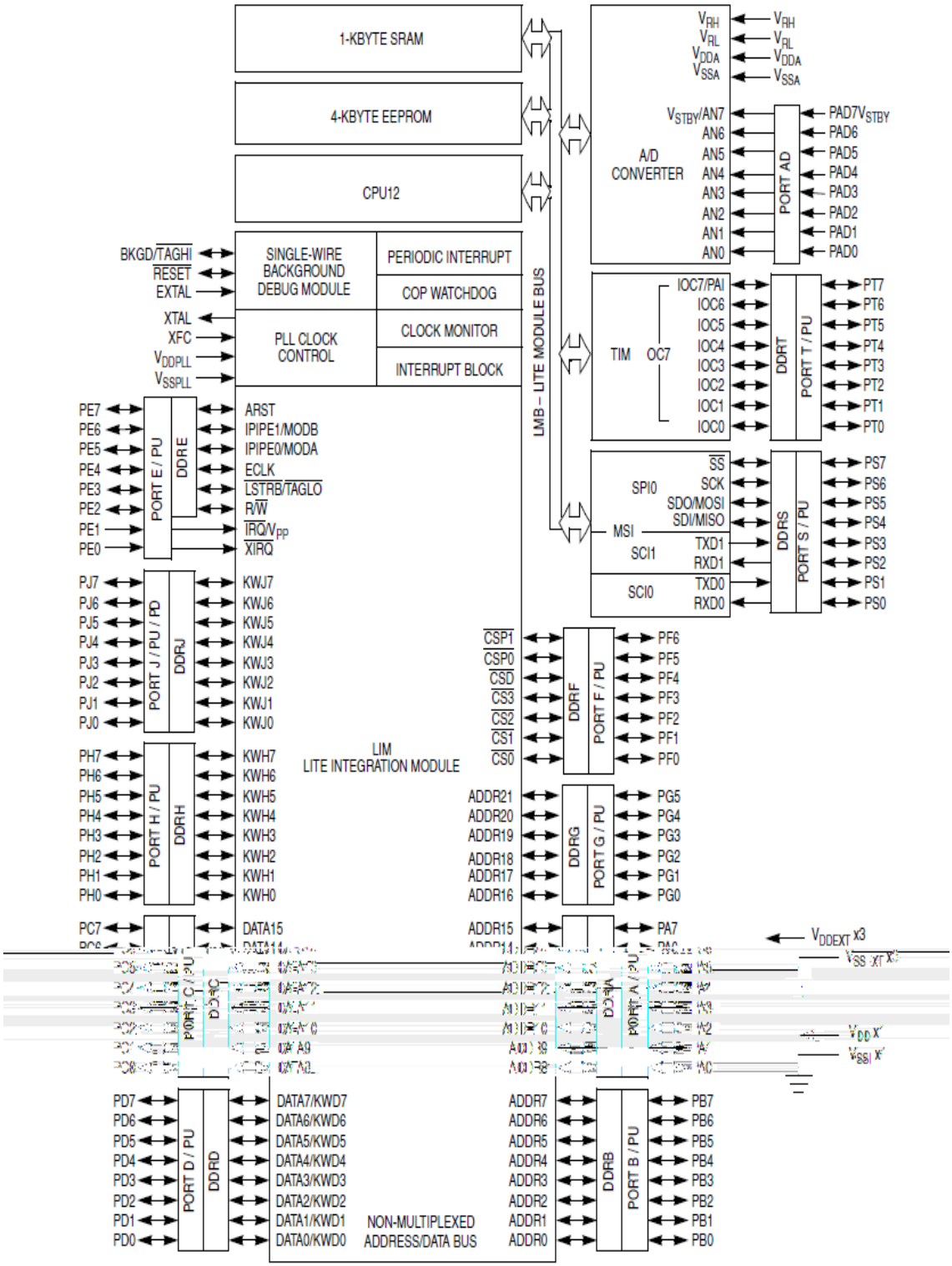


Figure 1. 1 MC68HC812A4 Block diagram.

1.2.3 Signal Descriptions

The MC68HC812A4 is available in a 112-lead low-profile quad flat pack (LQFP). The pin assignments are shown in Figure 1.2. Most pins perform two or more functions, as described in Table 1.1, Individual ports are cross referenced in Table 1.2.

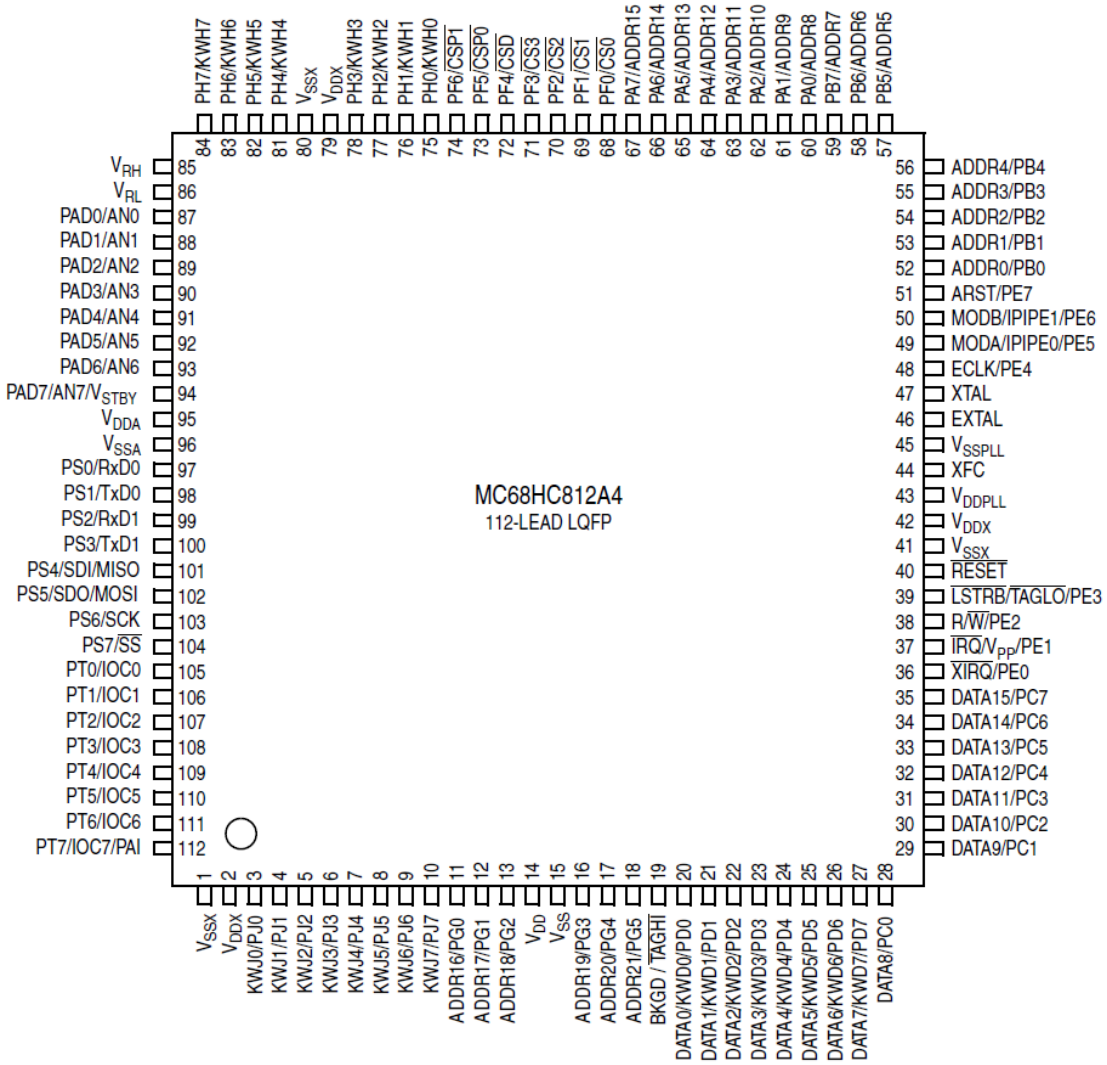


Figure 1. 2 Pin assignments.

Table 1. 1 Pin Descriptions.

Pin	Port	Description
VDD, VSS	—	Operating voltage and ground for the MCU
VRH, VRL	—	Reference voltages for the ADC
AVDD, AVSS	—	Operating voltage and ground for the ADC
VDDPLL,VSSPLL	—	Power and ground for PLL clock control
VSTBY	Port AD	RAM standby power input
XTAL, EXTAL	—	Input pins for either a crystal or a CMOS compatible clock
$\overline{XIRQ}$	PE0	Asynchronous, non-maskable external interrupt request input
IRQ	PE1	Asynchronous, maskable external interrupt request input with selectable falling-edge triggering or low-level triggering
R/$\overline{W}$	PE2	Expansion bus data direction indicator General-purpose I/O; read/write in expanded modes
$\overline{LSTRB}$	PE3	Low byte strobe (0 = low byte valid) General-purpose I/O
ECLK	PE4	Timing reference output for external bus clock (normally, half the crystal frequency) General-purpose I/O
BKGD	—	Mode-select pin determines initial operating mode of the MCU after reset
MODA	PE5	Mode-select input determines initial operating mode of the MCU after reset
MODB	PE6	Mode-select input determines initial operating mode of the MCU after reset
IPIPE0	PE5	Instruction queue tracking signals for development systems
IPIPE1	PE6	
ARST	PE7	Alternate active-high reset input General-purpose I/O
XFC	—	Loop filter pin for controlled damping of PLL VCO loop

$\overline{RESET}$	—	Active-low bidirectional control signal; input initializes MCU to known startup state; output when COP or clock monitor causes a reset
ADDR15–ADDR8	Port A	Single-chip modes: general-purpose I/O Expanded modes: external bus pins Port D in narrow data bus mode: general-purpose I/O or key wakeup port
ADDR7–ADDR0	Port B	
DATA15–DATA8	Port C	
DATA7–DATA0	Port D	
ADDR21,ADDR16	Port G	Memory expansion and general-purpose I/O
BKGD	—	Single-wire background debug pin Mode-select pin that determines special or normal operating mode after reset
KWD7–KWD0	Port D	Key wakeup pins that can generate interrupt requests on high-to-low transitions General-purpose I/O
KWH7–KWH0	Port H	
KWJ7–KWJ0	Port J	Key wakeup pins that can generate interrupt requests on any transition General-purpose I/O
RxD0	PS0	Receive pin for SCI0
TxD0	PS1	Transmit pin for SCI0
RxD1	PS2	Receive pin for SCI1
TxD1	PS3	Transmit pin for SCI
SDI/MISO	PS4	Master in/slave out pin for SPI
SDO/MOSI	PS5	Master out/slave in pin for SPI
SCK	PS6	Serial clock for SPI
$\overline{SS}$	PS7	Slave select output for SPI in master mode; slave select input in slave mode
IOC7–IOC0	Port T	Input capture or output

Table 1. 2 Port Descriptions.

Port	**Direction**	**Function**
Port A	I/O	Single-chip modes: general-purpose I/O Expanded modes: external address bus ADDR15–ADDR8
Port B	I/O	Single-chip modes: general-purpose I/O Expanded modes: external address bus ADDR7–ADDR0
Port C	I/O	Single-chip modes: general-purpose I/O Expanded wide modes: external data bus DATA15–DATA8 Expanded narrow modes: external data bus DATA15–DATA8/DATA7–DATA0
Port D	I/O	Single-chip and expanded narrow modes: general-purpose I/O External data bus DATA7–DATA0 in expanded wide mode
Port E	I/O and I(2)	External interrupt request inputs, mode select inputs, bus control signals General-purpose I/O
Port F	I/O	Chip select General-purpose I/O
Port G	I/O	Memory expansion General-purpose I/O
Port H	I/O	Key wakeup General-purpose I/O
Port J	I/O	Key wakeup General-purpose I/O
Port S	I/O	SCI and SPI ports General-purpose I/O
Port T	I/O	Timer port General-purpose I/O
Port AD	I	ADC port General-purpose input

1.2.4 Central processor unit (CPU12)

CPU12 registers are an integral part of the CPU and are not addressed as if they were memory locations See Figure 1.3.

7 A 0 | 7 B 0 15 D 0	8-BIT ACCUMULATORS A AND B ,16-BIT ACCUMULATOR D
15 x 0	INDEX REGISTER X
15 y 0	INDEX REGISTER Y
15 SP 0	STACK POINTER
15 PC 0	PROGRAM COUNTER

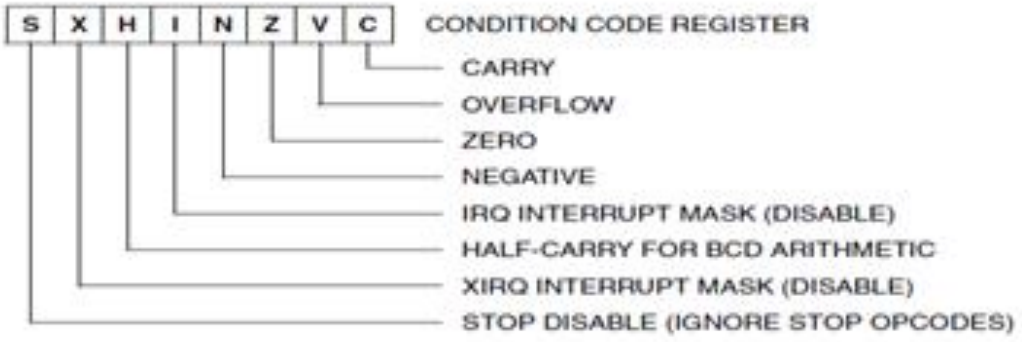


Figure 1. 3 Programming Model.

1.2.5 Addressing modes

Addressing modes determine how the CPU accesses memory locations to be operated upon. The CPU12 includes all of the addressing modes of the M68HC11 CPU as well as several new forms of indexed addressing. Table 1.3 is a summary of the available addressing modes.

Table 1. 3 HC12 addressing modes.

Addressing Mode	**Source Format**	**Abbreviation**	Inherent INST INH Operands (if any) are
Inherent	INST	INH	Operands (if any) are in CPU registers.
Immediate	INST *#opr8i* or INST *#opr16i*	IMM	Operand is included in instruction stream. 8- or 16-bit size implied by context
Direct	INST *opr8a*	DIR	Operand is the lower 8 bits of an address in the range $0000–$00FF.
Extended	INST *opr16a*	EXT	Operand is a 16-bit address
Relative	INST *rel8* or INST *rel16*	REL	An 8-bit or 16-bit relative offset from the current pc is supplied in the instruction.
Indexed (5-bit offset)	INST *oprx5,xysp*	IDX	5-bit signed constant offset from x, y, sp, or pc
Indexed (auto pre-decrement)	INST *oprx3,–xys*	IDX	Auto pre-decrement x, y, or sp by 1 ~ 8
Indexed (auto pre-increment)	INST *oprx3,+xys*	IDX	Auto pre-increment x, y, or sp by 1 ~ 8
Indexed (auto post-decrement)	INST *oprx3,xys–*	IDX	Auto post-decrement x, y, or sp by 1 ~ 8
Indexed (auto post-increment)	INST *oprx3,xys+*	IDX	Auto post-increment x, y, or sp by 1 ~ 8

Indexed (accumulator offset)	INST *abd,xysp*	IDX	Indexed with 8-bit (A or B) or 16-bit (D) accumulator offset from x, y, sp, or pc
Indexed (9-bit offset)	INST *oprx9,xysp*	IDX1	9-bit signed constant offset from x, y, sp, or pc (lower 8-bits of offset in one extension byte)
Indexed (16-bit offset)	INST *oprx16,xysp*	IDX2	16-bit constant offset from x, y, sp, or pc (16-bit offset in two extension bytes)
Indexed-indirect (16-bit offset)	INST [*oprx16,xysp*]	[IDX2]	Pointer to operand is found at... 16-bit constant offset from x, y, sp, or pc (16-bit offset in two extension bytes)
Indexed-indirect (D accumulator offset)	INST [D,*xysp*]	[D,IDX]	Pointer to operand is found at... x, y, sp, or pc plus the value in D

1.3 HC12Compact

1.3.1 Introduction

HC12compact is a universal microcontroller module based on the Motorola MC68HC812A4 MCU.

In addition to the on-chip features of the MCU itself, the following peripherals are available on the HC12compact:

- 512 KB Flash-Memory and 256 KB (optional: 1024 KB) RAM.
- Real Time Clock (battery backed).
- Analog/Digital-Converter (12 bit, 11 channels).
- Digital/Analog-Converter (12 bit, 2 channels).
- CAN Controller.
- RS232 interface driver.
- Beeper.
- Indicator LED.

Key features of the HC12compact unit are:

- compact design.
- low power consumption.
- easy handling.
- comprehensive software support available (Monitor, C-Compiler, BDM-Debugger etc…).

1.3.2 Package Contents

The base version (stock code HC12CO/1) of HC12compact is equipped as follows:

- Ready-to-use controller board with 256 KB RAM and 512 KB Flash.
- None of the options are populated (order RTC, ADC, DAC and CAN separately, but always together with the board).
- The header connectors at both edges of the board are not mounted (so the user may solder them up- or downward, depending on the application).
- User Manual and software on CD-ROM.
- TwinPEEKs Monitor Program, residing in the internal EEPROM of the HC12.
- Serial cable with Sub-D9 connector (PC side).

Figure 1.4 and 1.5 show the parts location diagram and jumpers location and the figure 1.6 shows the connectors:

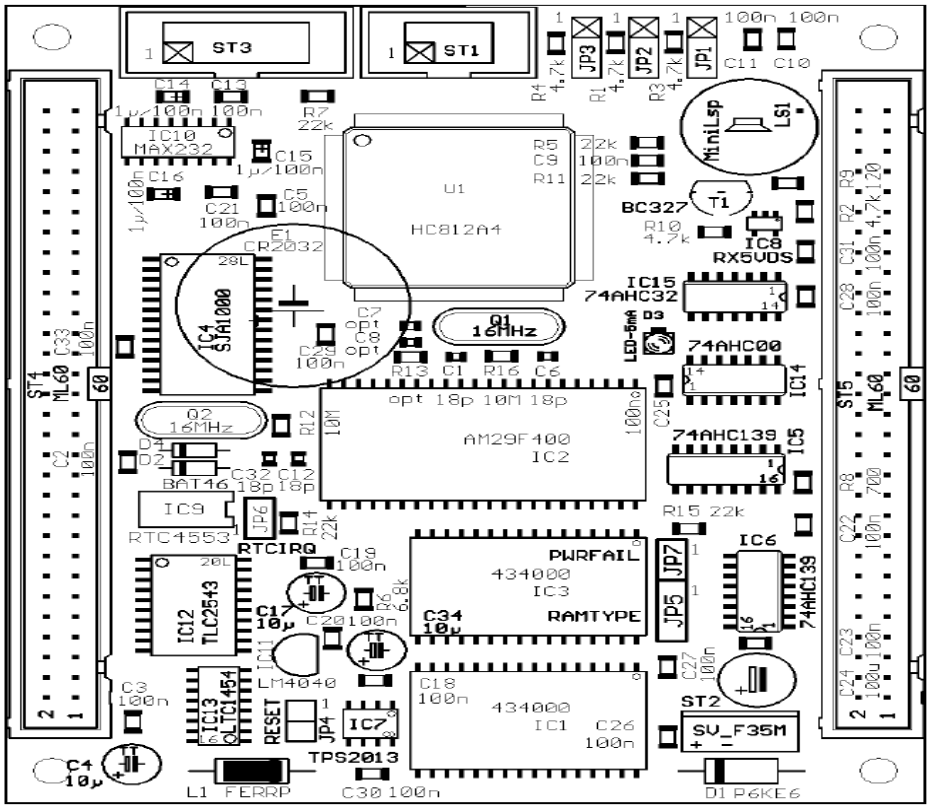


Figure 1. 4 Parts Location Diagram.

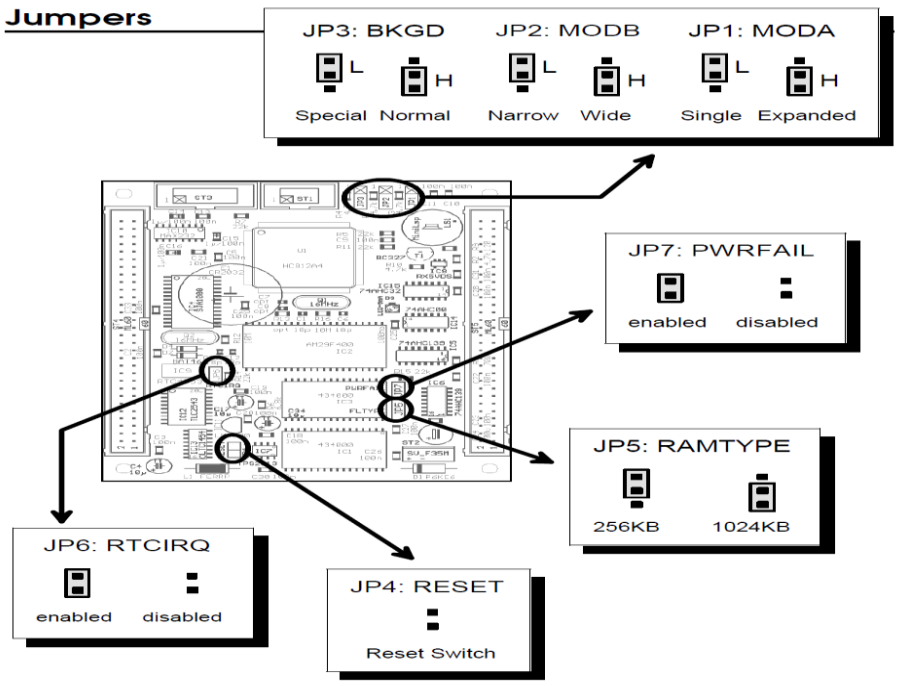


Figure 1. 5 Jumper locations.

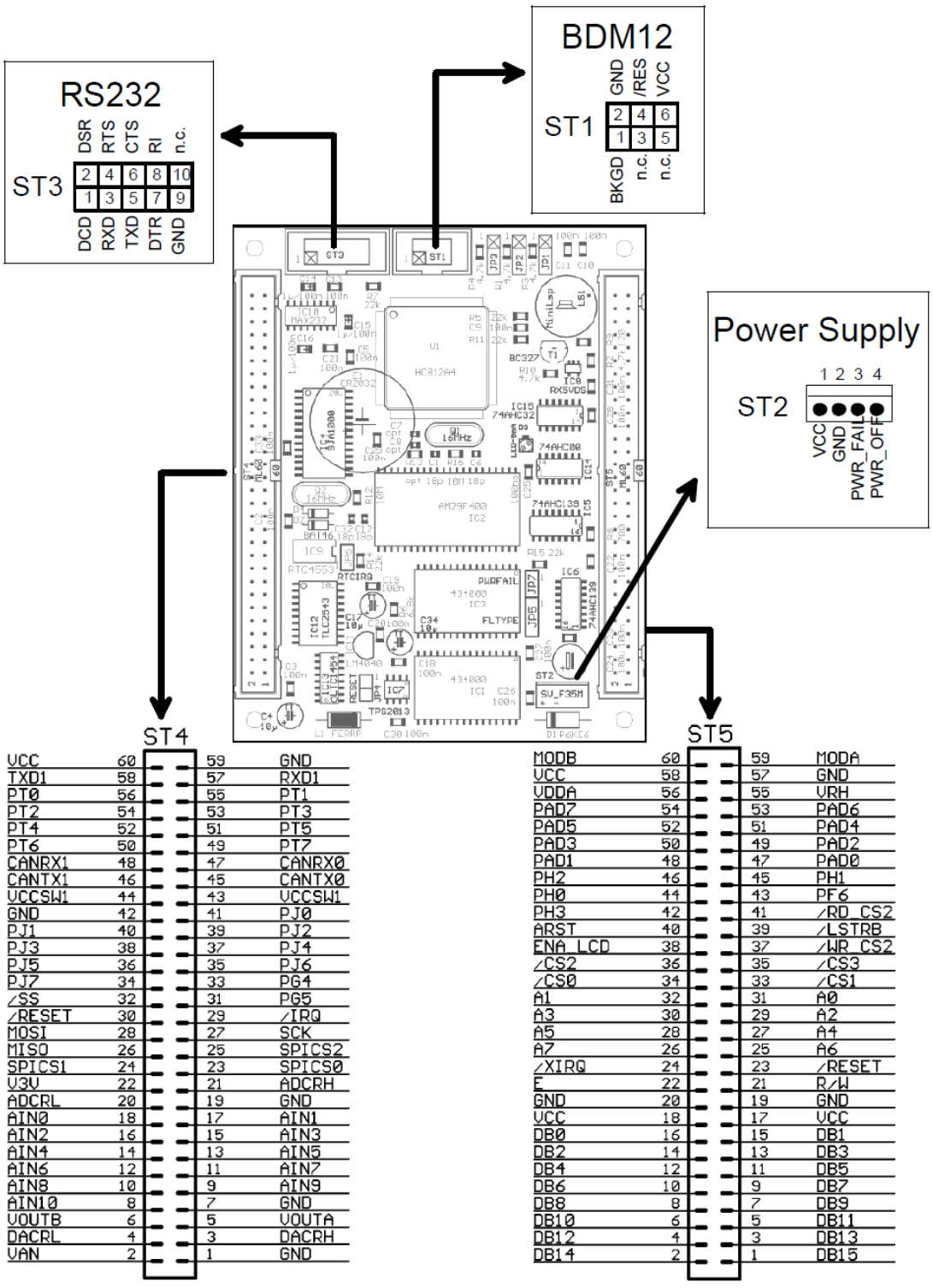


Figure 1. 6 Connectors.

1.3.3 Monitor Program Twin Peeks

a. Introduction

The monitor program TwinPEEKs is useful to load and execute user programs and to view and modify memory locations. TwinPEEKs resides in the internal EEPROM of the HC12. The whole 4 KB EEPROM area plus a region of about 512 bytes of RAM are reserved for TwinPEEKs.

The user program will be loaded into an external (RAM or Flash) memory space.

TwinPEEKs on the HC12compact communicates with a host PC via the first serial interface (SCI0).

Communication parameters are:

- 19200 Baud,
- 8N1,
- No protocol.

-Notation:

All numbers are in hex format, a monitor command consists of a single character, followed by a list of arguments. The argument list may contain up to six items, separated by a space or comma.

The address space is 64 KB, so addresses have a maximum length of 4 digits. This address space of the HC12 contains all ports and control registers.

b. Monitor Commands

The list bellow shows the commands and their arguments:

Table 1. 4 Twinpeeks commands

A	Display ADC results
D [<SADR> [<EADR>]]	Displays the memory from <SADR> to <EADR> ,that mean from start address to end address
E [<SADR>]	Edit memory starting at <SADR>
F <SADR> <EADR> <BY>	Fills memory from <SADR> to <EADR> with <BY>
G [<SADR>]	Go to Address <SADR>
H	Displays a list of available commands
I	Info on Flash device ID

L	Loads S-Records into memory
M <SADR> <EADR> <ADR2>	Copies the memory area from <SADR> to <EADR> to another area starting at <ADR2>
P	Displays the currently selected program page number
T [<HH> <MM> <SS> <DD> <MM> <YY>]	The command displays the RTC time when used w/o argument. Of course, the optional RTC chip must be present
V <CH0> <CH2>	The optional DAC has two channels which may be adjusted with this command. The values range from $0000 to $0FFF (12 bits). $0000 equates 0V, $0FFF equates 4.095V.
X [<SECTOR>]	Deletes a single Flash memory sector

The following table shows the sector addresses of the Am29F400T and the corresponding program page numbers:

Table 1. 5 the sector addresses of the Am29F400T.

Sector Address	**Sector Size**	**Memory Area**	**PPAGE**
00	64KB	00000–0FFFF	00-03
08	64KB	10000–1FFFF	04-07
10	64KB	20000–2FFFF	08-0B
18	64KB	30000–3FFFF	0C-0F
20	64KB	40000–4FFFF	10-13
28	64KB	50000–5FFFF	14-17
30	64KB	60000–6FFFF	18-1B
38	32KB	70000–77FFF	1C-1D
3C	8KB	78000-79FFF	1E (lower half)
3D	8KB	7A000-7BFFF	1E (upper half)
3E	16KB	7C000-7FFFF	1F

1.3.4 Serial communication interface module SCI1

The serial communications interface (SCI) allows asynchronous serial communications with peripheral devices and other MCUs.

a Features

-13-bit baud rate selection

-Programmable 8-bit or 9-bit data format

-Separately enabled transmitter and receiver

-flags with interrupt-generation capability:

- Transmitter empty

-Transmission complete

- Receiver full.

b The register map

1-SCI 1 Baud Rate Register High (SC1BDH) : $00C8.

2-SCI 1 Baud Rate Register low (SC1BDl) : $00C9.

3-SCI 1 control register 1 (SC1CR1) : $00CA.

4-SCI 1 control register 2 (SC1CR2) : $00CB.

5-SCI 1 status register 1 (SC1SR1) : $00CC.

6-SCI 1 status register 2 (SC1SR2) : $00CD.

7-SCI 1 data register high (SC1DRH) : $00CE.

8-SCI 1 data register low (SC1DRl) : $00CF.

c External Pin Descriptions

-TXD Pin

TXD is the SCI transmitter pin. TXD is available for general-purpose I/O when it is not configured for transmitter operation.

-RXD Pin

TXD is the SCI transmitter pin. TXD is available for general-purpose I/O when it is not configured for transmitter operation.

d Example

This code is intended to use SCI1 to serially transmit number from the hc12 compact to a computer; the transmission is performed at a baud rate of 9600.

```
                        //serial port SCI1//
SC1BDL=0x34;            //baud rate =9600//
SC1CR1=0x00;            // Initialize for 8-bit Data format//
SC1CR2=0x08;            //transmitter enabled //
asm(" LDAA $00cc ");   //step1 for clear TDRE//
asm(" STD $00ce ");    //step2 for clear TDRE//
while(1)

{

If (SC1SR1==0x80)    // set the Transmit Data Register Empty Flag//
SC1DRL=0x31;   //send the number 0x31 (1 in ASCII code) by the sci1 port //
}
```

1.3.5 Analog-to-Digital Converter (ATD)

The analog-to-digital converter (ATD) is an 8-channel, 8-bit, multiplexed-input, successive approximation analog-to-digital converter, accurate to±1 least significant bit (LSB). It does not require external sample and hold circuits The ATD converter timing can be synchronized to the system P-clock. The ATD module consists of a 16-word (32-byte) memory-mapped control register block used for control, testing, and configuration.

a Features

Features of the ATD module include:

• Eight multiplexed input channels.
• Multiplexed-input successive approximation.
• 8-bit resolution.
• Single or continuous conversion.
• Conversion complete flag with CPU interrupt request.
• Selectable ATD clock.

b Register Map

1-ATD control registers 0 (ATDCTL0) ($0060).

2-ATD control register 1 (ATDCTL1) ($0061).

3-ATD control register 2 (ATDCTL2) ($0062).

4-ATD control registers 3 (ATDCTL3) ($0063).

5-ATD control registers 4 (ATDCTL4) ($0064).

6-ATD control registers 5 (ATDCTL5) ($0065).

7-ATD status registers 1 (ATDSTAT1) ($0066).

8-ATD status registers 2 (ATDSTAT2) ($0067).

9-ATD test registers 1 (ATDCTEST1) ($0068).

10-ATD test registers 2 (ATDCTEST2) ($0069).

11-port AD data input register (PORTAD) ($006F).

12-ATD result register0 (ADR0H) ($0070).

13-ATD result register1 (ADR1H) ($0072).

14-ATD result register2 (ADR2H) ($0074).

15-ATD result register3 (ADR3H) ($0076).

16-ATD result register4 (ADR4H) ($0078).

17-ATD result register5 (ADR5H) ($007A).

18-ATD result register6 (ADR6H) ($007C).

19-ATD result register7 (ADR7H) ($007E).

c General-Purpose Ports

Port AD is an input-only port. When the ATD is enabled, port AD is the analog input port for the ATD.

Setting the ATD power-up bit, ADPU, in ATD control register 2 enables the ATD.

Port AD is available for general-purpose input when the ATD is disabled. Clearing the ADPU bit disables the ATD.

d Example

Using the ATD to Measure a Potentiometer Signal:

This example allows the student to utilize the ATD on the HC12 to measure a potentiometer signal output and save it in memory ($1000):

```
                        //ATD//
ATDCTL2=0x82;     // enable ATD and set ASCIE
for ( i = 0; i < 20; ++i );   //delay //
ATDCTL3 = 0x00;    //reset//
ATDCTL4 = 0x01;   // 4 ATD clock periods and a prescaler =4 //
ATDCTL5 = 0x01;   // Conversions of a single input channel 1( PAD1) //

While (1)
    {
     while (ATDCTL2==0x83)
          {
           asm("ldaa $0072");   //voltage of PAD1//
           asm("staa $1000");   //save tension in memory//

          }

    }
```

1.4 Arduino (UNO)

1.4.1 Introduction

Arduino is a flexible programmable hardware platform designed for artists, designers, tinkerers, and the makers of things. Arduino is small, blue circuit board, mythically taking its name from a local pub in Italy. Central to the Arduino interface board, shown in Figure 1.7, is an onboard microcontroller think of it as a small computer on a chip.
This microcontroller comes from a company called Atmel and the chip is known as an AVR, it is slow in modern terms, running at only 16 MHz with an 8-bit core, and has a very limited amount of available memory, with 32 kilobytes of storage and 2 kilobytes of random access memory.

The Arduino development environment is free for all to use and will run on just about any kind of computer that supports Java.

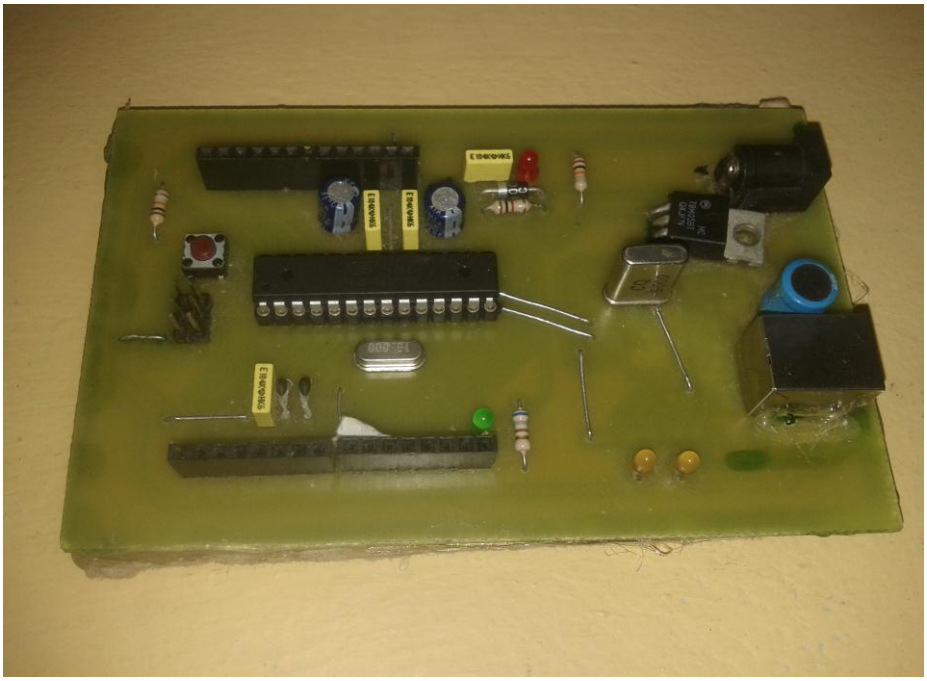


Figure1. 7 The Arduino Uno interface board

1.4.2 Some simple command of Arduino

In order to write an Arduino program, many commands can be used:

a pinMode:

This command, which goes in the setup () function, is used to set the direction of a digital I/O pin.

Set the pin to OUTPUT if the pin is driving and LED, motor or other device. Set the pin to INPUT if the pin is reading a switch or other sensor.

On power up or reset, all pins default to inputs.

This example sets pin 2 to an output and pin 3 to an input:

```
void setup()
{
pinMode(2,OUTPUT);
pinMode(3,INPUT);
}
```

b Serial.print

The Serial.print command lets we see what's going on inside the Arduino from your computer. For example, we can see the result of a math operation to determine if we are getting the right number. For the command to work, the command Serial.begin(9600) must be placed in the setup() function. After the program is uploaded, you must open the Serial Monitor window to see the response.

Here is a brief program to check if your board is alive and connected to the PC:

```
void setup()
{
Serial.begin(9600);//set the baud rate  to 9600
 Serial.println ("Hello World"); //    display (Hello World) in the Serial Monitor window
}
```

c digitalWrite

This command sets an I/O pin high (+5V) or low (0V).

Use the pinMode() command in the setup() function to set the pin to an output.

Example:

```
digitalWrite(2,HIGH); // sets pin 2 to +5 volts
digitalWrite(2,LOW); // sets pin 2 to zero volts
```

d analogWrite()

The function analogWrite() will allow us to access the pulse width modulation hardware on the Arduino microcontroller. The basic syntax for this function follows:

analogWrite(pin, duty cycle). In using this function we need to give it two pieces of information. The first is the pin number, which can only include one of the following pins on the Arduino Uno: 3, 5, 6, 9, 10, and 11. These are marked as PWM on the interface board. The second bit of information is the duty cycle expressed as an 8-bit numerical integer ranging between the values of 0 and 255,the value 0 corresponds to off or 0% duty cycle and 255 is basically full on or 100% duty cycle.

Example:

```
 analogWrite(5, 127). // generate a PWM signal with 50percent duty cycle in the pin 5.
```

e delay

delay pauses the program for a specified number of milliseconds. Since most interactions with the world involve timing, this is an essential instruction. The delay can be for 0 to 4,294,967,295 msec. This code snippet turns on pin 2 for 1 second.

```
digitalWrite(2,HIGH); // pin 2 high (LED on)
delay(1000); // wait 1000 ms
digitalWrite(2,LOW); // pin 2 low (LED off)
```

And many other command that we can use in order to write our program in the Arduino like: map(), constrain() etc...

Note that Arduino support the c/c ++ instructions like: if, for, while, goto ....etc.

1.5 CONCLUSION

The HC12 module and its microcontroller MC68HC812A4 are powerful equipment because of its great advantages of fuzzy logic control instruction ,various type of addressing mode ,much register ,much memory capacity …

Using the Arduino UNO with theirs easier instructions make the control algorithm and the application more effective.

Chapter 2 Implementation in HC12 microcontroller of fuzzy sets, fuzzy logic and fuzzy control

2.1 Introduction

Over the past few years, the use of fuzzy set theory, or fuzzy logic, in control systems has been gaining widespread popularity, especially in Japan.

From as early as the mid-1970s, Japanese scientists have been instrumental in transforming the theory of fuzzy logic into a technological realization. Today, fuzzy logic-based control systems, or simply fuzzy logic controllers (FLCs), can be found in a growing number of products, from washing machines to speedboats, from air condition units to hand-held autofocus cameras. The success of fuzzy logic controllers is mainly due to their ability to cope with knowledge represented in a linguistic form instead of representation in the conventional mathematical framework.

Control engineers have traditionally relied on mathematical models for their designs.

This is the fundamental concept that provided the motivation for fuzzy logic and is formulated by Lofti Zadeh, the founder of fuzzy set theory, as the Principle of Incompatibility.

Real-world problems can be extremely complex and complex systems are inherently fuzzy. The main advantage of fuzzy logic controllers is their ability to incorporate experience, intuition and heuristics into the system instead of relying on mathematical models. This makes them more effective in applications where existing models are ill-defined and not reliable enough.

2.2 Historical review

Fuzzy logic was introduced in 1965 by Lofti Zadeh in his paper "Fuzzy Sets". Zadeh and others continued to develop fuzzy logic at that time. The idea of fuzzy sets and fuzzy logic were not accepted well within academic circles because some of the underlying mathematics had not yet been explored. The applications of fuzzy logic were slow to develop because of this, except in the east. In Japan specifically fuzzy logic was fully accepted and implemented in products simply because fuzzy logic worked, regardless of whether mathematicians agreed or not. The success of many fuzzy logic based products in Japan in the early 80s led to a revival in fuzzy logic in the USA in the late 80s. Since that time America has been catching up with the east in the area of fuzzy logic.

2.3 Fuzzy sets and fuzzy logic

Classical set theory was founded by the German mathematician Georg Cantor (1845– 1918). In the theory, a universe of discourse U, is defined as a collection of objects all having the same characteristics. A classical set is then a collection of a number of those elements. The member elements of a classical set belong to the set 100 per cent.

Other elements in the universe of discourse, which are non-member elements of the set, are not related to the set at all. A definitive boundary can be drawn for the set, as depicted in Figure 2.1.

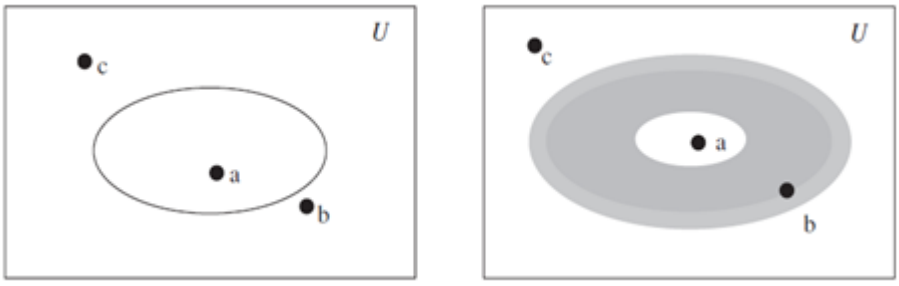


Figure2. 1 *(a)* Classical/crisp set boundary; (b) fuzzy set boundary

A classical set can be denoted by A = {x ∈ U | P(x)} where the elements of A have the property P, and U is the universe of discourse. The characteristic function μA(x): U → {0, 1} is defined as '0' if x is not an element of A and '1' if x is an element of A. Here, U contains only two elements, '1' and '0'. Therefore, an element x, in the universe of discourse is either a member of set A or not a member of set A.

There is ambiguity about membership for example, consider the set ADULT, which contains elements classified by the variable AGE, it can be said that an element with AGE = '5' would not be a member of the set whereas an element with AGE = '45' would be. The question which arises is: where can a sharp and discrete line be drawn in order to separate members from non-members? At AGE = '18'? By doing so, it means that elements with AGE = '17.9' are not members of the set ADULT but those with AGE = '18.1' are. This system is obviously not realistic to model the definition of an ADULT human. Simple problems such as this one embody the notion behind Zadeh's Principle of Incompatibility.

In fuzzy set theory, the concept of characteristic function is extended into a more generalized form, known as membership function: μA(x):U→[0, 1]. While a characteristic function exists in a two-element set of {0, 1}, a membership function can take up any value between the unit interval [0, 1] (note that curly brackets are used to represent discrete membership while square brackets are used to represent continuous membership). The set which is defined by this extended membership function is called a fuzzy set. In contrast, a classical set which is defined by the two-element characteristic function, as described earlier, is called a crisp set. Fuzzy set theory essentially extends the concept of sets to encompass vagueness. Membership to a set is no longer a matter of 'true' or 'false', '1' or '0', but a matter of degree. The degree of membership becomes important. The boundary of a fuzzy set is shown in Figure 2.1(b). While point (a) is a member of the fuzzy set and point (c) is not a member, the membership of point (b) is ambiguous as it falls on the boundary. The concept of membership function is used to define the extent to which a point on the boundary belongs to the set.

A fuzzy set F can be defined by the set of tuples F = {(μF(x), x) | x ∈ U}.

Zadeh proposed a notation for describing fuzzy sets whereby '+' denotes enumeration and '/' denotes a tuple. Therefore, the fuzzy set F becomes:

$F = \int_U \mu_F(x)/x$: For a continuous universe U.

Or:

$F = \sum_{x \in U} \mu_F(x)/x$: For a discrete universe U.

Returning to the earlier example, an element with AGE = '18.1' may now be assigned with the membership degree to the set ADULT of, say, 1.0.

An element of AGE = '17.9' may then have a membership degree of 0.8 instead of 0.

Such gradual change in the degree of membership provides a better representation of the real world. However, the exact shape of the membership function is very subjective and depends on the designer and the context of the application. While set operations such as complement, union and intersection are straightforward definitions in classical set theory, their interpretation is more complicated in fuzzy set theory due to the graded attribute of membership functions. Zadeh proposed the following fuzzy set operation definitions as an extension to the classical operations:

-Complement: $\forall x \in X$: $\mu_{\bar{A}} = 1-\mu_A$.

-Union: $\forall x \in X$: $\mu_{A \cup B}(x) = \max [\mu_A(x), \mu_B(x)]$.

- Intersection: $\forall x \in X$: $\mu (A \cap B)(x) = \min [\mu_A(x), \mu_B(x)]$.

These definitions form the foundations of the basics of fuzzy logic theory.

The relationship between an element in the universe of discourse and a fuzzy set is defined by its membership function.

2.3.1 Types of membership functions

Figures 2.2, 2.3 and 2.4 show various types of membership functions which are commonly used in fuzzy set theory. The choice of shape depends on the individual application.

a triangular function:

$$F(x) = \begin{cases} 0 & x < a \\ \frac{x-\alpha}{\beta-\alpha} & \alpha \le x < \beta \\ \frac{\gamma-x}{\gamma-\beta} & \beta \le x < \gamma \\ 0 & x \ge \gamma \end{cases}$$

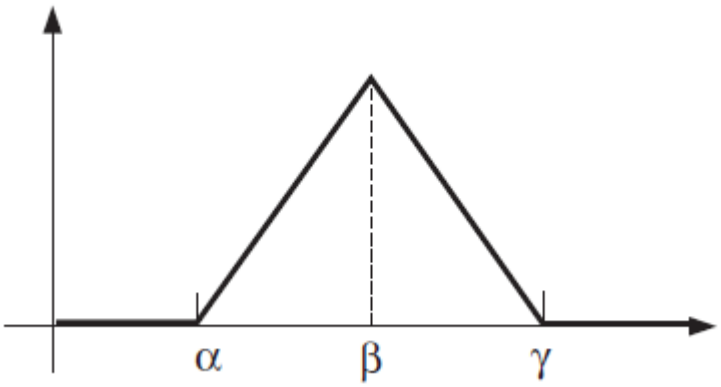


Figure2. 2 triangular function.

b trapezoidal function:

$$F(x) = \begin{cases} 0 & x < \alpha \\ \frac{x-\alpha}{\beta-\alpha} & \alpha \le x < \beta \\ 1 & \beta \le x < \gamma \\ \frac{\lambda-x}{\lambda-\gamma} & \gamma \le x < \lambda \\ 0 & x \ge \lambda \end{cases}$$

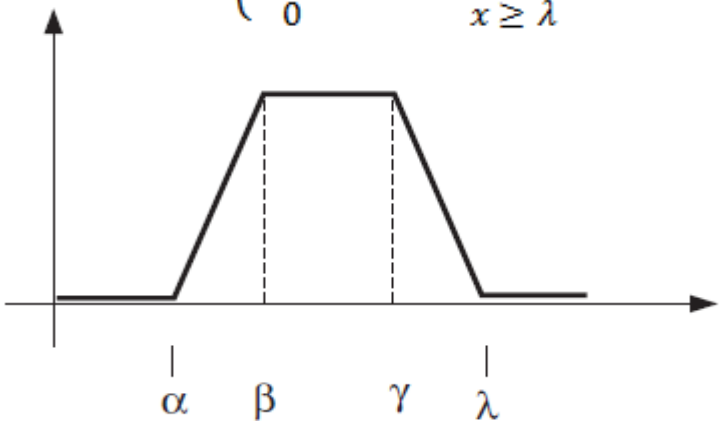


Figure2. 3 Trapezoidal function.

c Gaussian function:

$$F(x) = e^{-\frac{1}{2}(\frac{x-c}{\sigma})^2}$$

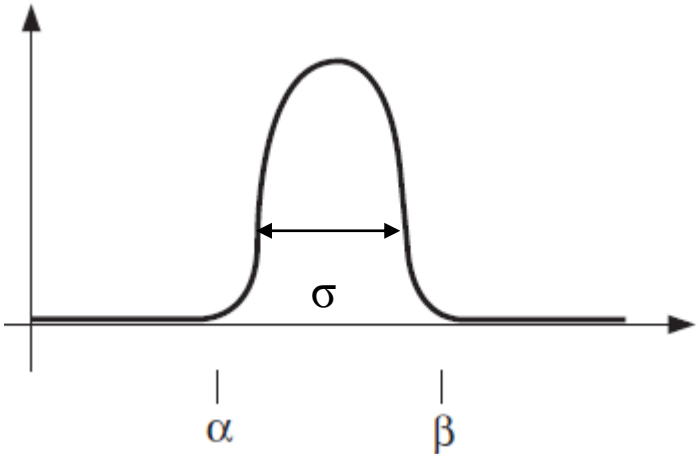


Figure2. 4 Gaussian function.

With: $c = \frac{\alpha+\beta}{2}$ (the average value).

σ: standard deviation .

2.3.2 Linguistic variables

The concept of a linguistic variable, a term which is later used to describe the inputs and outputs of the FLC, is the foundation of fuzzy logic control systems. A conventional variable is numerical and precise. It is not capable of supporting the vagueness in fuzzy set theory. By definition, a linguistic variable is made up of words, sentences or artificial languages which are less precise than numbers. It provides the means of approximate characterization of complex or ill-defined phenomena.

For example, 'AGE' is a linguistic variable whose values may be the fuzzy sets 'YOUNG' and 'OLD'.

A more common example in fuzzy control would be the linguistic variable 'ERROR', which may have linguistic values such as 'POSITIVE', 'ZERO' and 'NEGATIVE'.

2.3.3 Fuzzy control system

Figure 2.5 shows the block diagram of a typical fuzzy logic controller (FLC). There are five principal elements to a fuzzy logic controller:

-Fuzzification module (fuzzifier).

-Knowledge base.

-Rule base.

-Inference engine.

-Defuzzification module (defuzzifier).

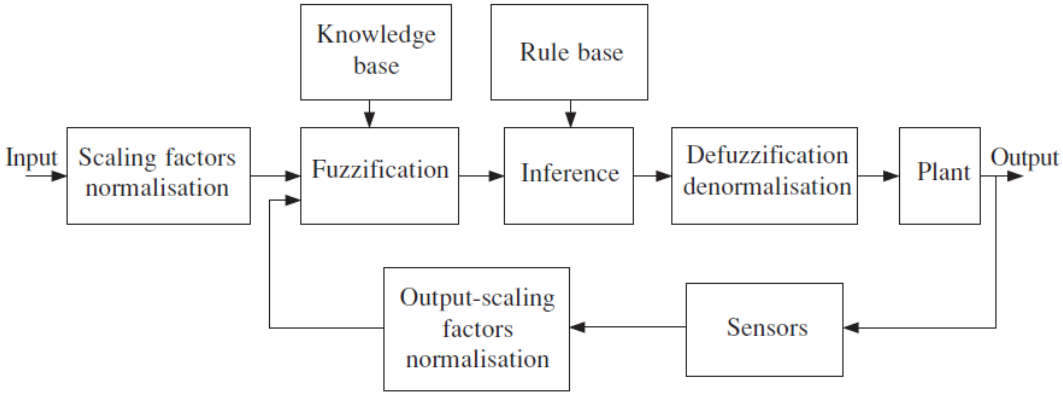


Figure2. 5 Block diagram of a typical fuzzy logic controller.

a Fuzzifier:

The fuzzification module converts the crisp values of the control inputs into fuzzy values, so that they are compatible with the fuzzy set representation in the rule base.

b Knowledge base :

The knowledge base consists of a database of the plant. It provides all the necessary definitions for the fuzzification process such as membership functions, fuzzy set representation of the input–output variables and the mapping functions between the physical and fuzzy domain.

c Rule base:

The rule base is essentially the control strategy of the system. It is usually obtained from expert knowledge or heuristics and expressed as a set of (IF-THEN) rules. The rules are based on the fuzzy inference concept and the antecedents and consequents are associated with linguistic variables.

For example:

IF error (e) is Positive Big (PB) THEN output (u) is Negative Big (NB)

Rule antecedent Rule consequent

Error (e) and output (u) are linguistic variables while Positive Big (PB) and Negative Big (NB) are the linguistic values.

d Defuzzifier:

The diagram in Fig. 2.6 shows the membership functions related to a typical fuzzy controller's output variable defined over its universe of discourse. The FLC will process the input data and map the output to one or more of these linguistic values.

Depending on the conditions, the membership functions of the linguistic values may be clipped. Figure 2.7 shows an output condition with two significant (clipped above zero) output linguistic values. The union of the membership functions forms the fuzzy output value of the controller.

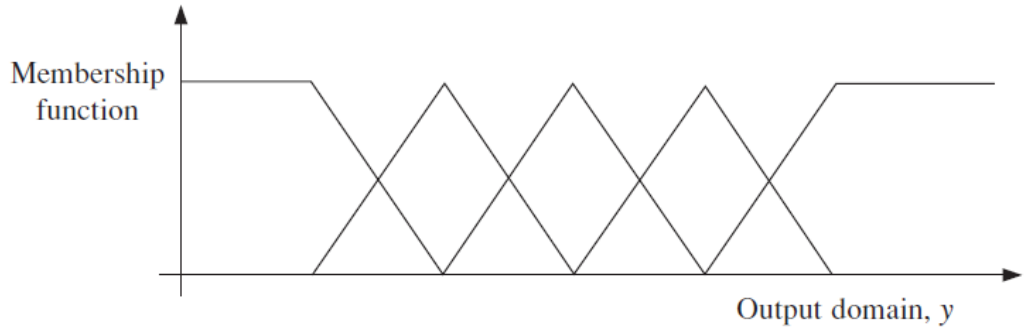


Figure2.6 Membership functions of the output linguistic values.

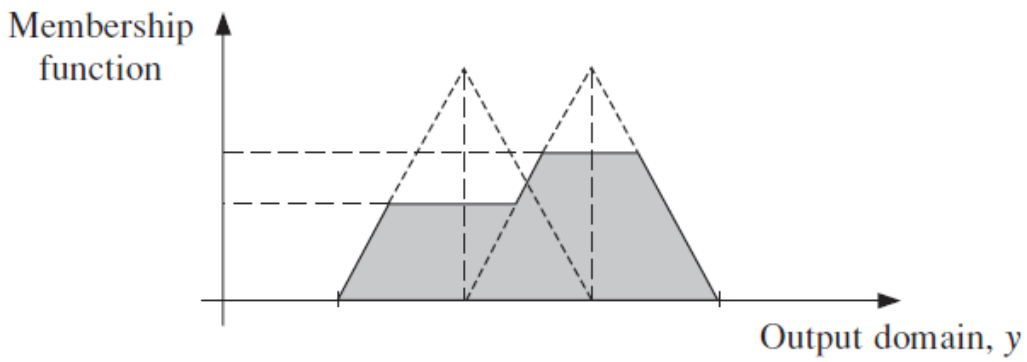


Figure2.7 Possible distribution of an output condition.

This is represented by the shaded area in Figure 2.7 and is expressed by the fuzzy set equation:

$$S = \bigcup_{i=1}^{k} S_i \ , \mu_s(y) = \max_i[\mu_{s_i}(y)] \quad i = 1,2, \dots k$$

Where:

S is the union of all the output linguistic values.

S_i is an output linguistic value with clipped membership function.

k is the total number of output linguistic values defined in the universe of discourse.

In most cases, the fuzzy output value S has very little practical use as most applications require non-fuzzy (crisp) control actions. Therefore, it is necessary to produce a crisp value to represent the possibility distribution of the output. The mathematical procedure of converting fuzzy values into crisp values is known as 'defuzzification'.

A number of defuzzification methods have been suggested. The choice of defuzzification methods usually depends on the application. The most defuzzification method used in fuzzy logic control is the weighted average method; it is only valid for symmetrical membership functions.

Each membership function is assigned with a weighting, which is the output point where the membership value is maximum.

The defuzzification process can be expressed by:

$$f(y) = \frac{\sum \mu(y).y}{\sum \mu(y)}$$

Where:

f(*y*) is the crisp output value.

y is the weight.

2.3.4 Types of fuzzy logic controllers

There are many types of FLC and the most popular are :

-MAMDANI.

-SUGENO.

These two types have the same methods of fuzzification and they differ in the inference and in the defuzzification.

a Mamdani Type Fuzzy Logic Controller:

The inference block of this type is based on the minimum (MIN) or the product (PROD) of the fuzzification's results.

The defuzzification output is calculated by: $f(y) = \frac{\sum \mu_{ci}(y).w_i}{\sum \mu_{ci}(y)}$

Where:

f(*y*) is the crisp output value.

$\mu_{ci}(y)$ is the maximum of all the rules with consequent c_i .

w_i is the weight.

b SUGENO Type Fuzzy Logic Controller:

The inference block is denoted by the product (PROD) of the fuzzifier outputs.

In this type the output membership functions are singleton (crisp), so the output value is calculated by:

$$f(y) = \frac{\sum \mu_{ci}(y).y_i}{\sum \mu_{ci}(y)}$$

Where:

f(*y*) is the crisp output value.

$\mu_{ci}(y)$ are the inference results.

y_i are the singleton value of the output membership.

2.4 Fuzzy Logic in HC12 compact

The CPU12 includes four instructions that perform specific fuzzy logic tasks. In addition, several other instructions are especially useful in fuzzy logic programs.

The four fuzzy logic instructions are:

• MEM (determine grade of membership), which evaluates trapezoidal membership functions.

• REV (fuzzy logic rule evaluation) and REVW (fuzzy logic rule evaluation weighted), which perform unweighted or weighted MIN-MAX rule evaluation

• WAV (weighted average), which performs weighted average defuzzification on singleton output membership functions.

Other instructions that are useful for custom fuzzy logic programs include:

• MINA (place smaller of two unsigned 8-bit values in accumulator A)

• EMIND (place smaller of two unsigned 16-bit values in accumulator D)

• MAXM (place larger of two unsigned 8-bit values in memory)

• EMAXM (place larger of two unsigned 16-bit values in memory)

• TBL (table lookup and interpolate)

• ETBL (extended table lookup and interpolate)

• EMACS (extended multiply and accumulate signed 16-bit by 16-bit to 32-bit)

For higher resolution fuzzy programs, the fast extended precision math instructions in the CPU12 are also beneficial.

Flexible indexed addressing modes help simplify access to fuzzy logic data structures stored as lists or tabular data structures in memory.

A microcontroller-based fuzzy logic control system has two parts:

• A fuzzy inference kernel which is executed periodically to determine system outputs based on current system inputs.

• A knowledge base which contains membership functions and rules.

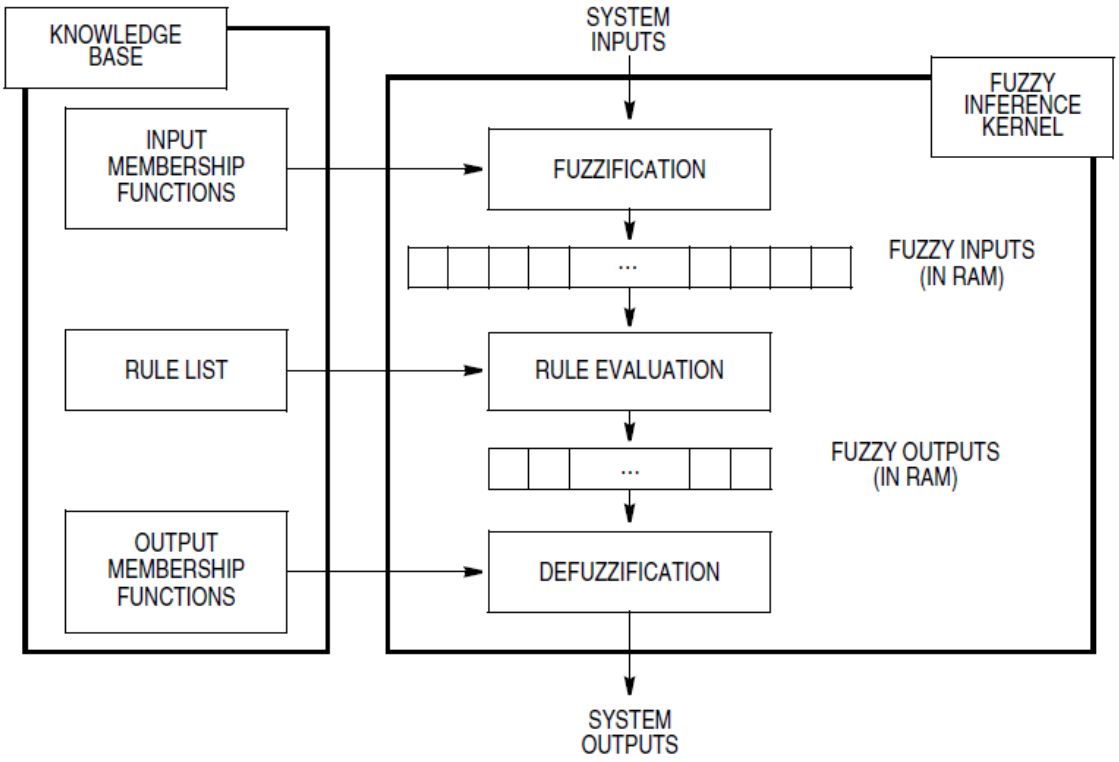


Figure 2.8 Block Diagram of a Fuzzy Logic System.

The knowledge base can be developed by an application expert without any microcontroller programming experience. Membership functions are simply expressions of the expert's understanding of the linguistic terms that describe the system to be controlled. Rules are ordinary language statements that describe the actions a human expert would take to solve the application problem. Rules and membership functions can be reduced to relatively simple data structures (the knowledge base) stored in non-volatile memory. A fuzzy inference kernel can be written by a programmer who does not know how the application system works. The only thing the programmer needs to do with knowledge base information is store it in the memory locations used by the kernel.

2.4.1 Fuzzification (MEM)

During the fuzzification step, the current system input values are compared against stored input membership functions to determine the degree to which each label of each system input is true. This is accomplished by finding the y value for the current input value on a trapezoidal membership function for each label of each system input.

The MEM instruction in the CPU12 performs this calculation for one label of one system input. To perform the complete fuzzification task for a system, several MEM instructions must be executed, usually in a program loop structure. Figure 2.9 shows a system of three input membership functions, one for each label of the system input.

The x-axis of all three membership functions represents the range of possible values of the system input. The vertical line through all three membership functions represents a specific system input value. The y-axis represents degree of truth and varies from completely false ($00 or 0 percent) to completely true ($FF or 100 percent).

The y-value where the vertical line intersects each of the membership functions is the degree to which the current input value matches the associated label for this system input. For example, the expression "temperature is warm" is 25 percent true ($40). The value $40 is stored to a random-access memory (RAM) location and is called a fuzzy input (in this case, the fuzzy input for "temperature is warm"). There is a RAM location for each fuzzy input (for each label of each system input). When the fuzzification step begins, the current value of the system input is in an accumulator of the CPU12, one index register points to the first membership function definition in the knowledge base, and a second index register points to the first fuzzy input in RAM. As each fuzzy input is calculated by executing a MEM instruction, the result is stored to the fuzzy input and both pointers are updated automatically to point to the locations associated with the next fuzzy input. The MEM instruction takes care of everything except counting the number of labels per system input and loading the current value of any subsequent system nputs. The end result of the fuzzification step is a table of fuzzy inputs representing current system conditions.

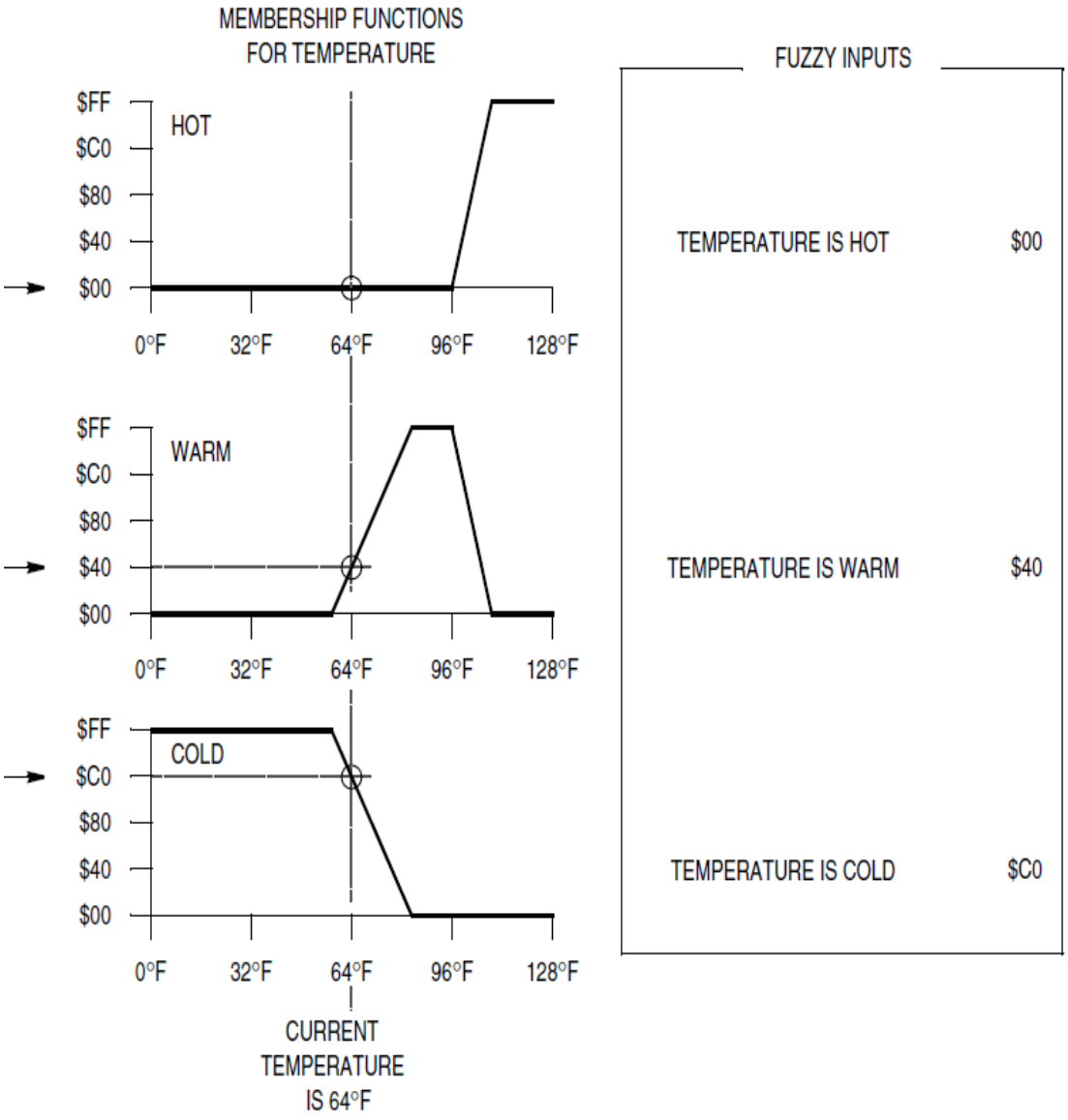


Figure 2.9 Fuzzification mecanism with the HC12 microcontroller.

2.4.2 Rule Evaluation (REV and REVW)

Rule evaluation is the central element of a fuzzy logic inference program. This step processes a list of rules from the knowledge base using current fuzzy input values from RAM to produce a list of fuzzy outputs in RAM. These fuzzy outputs can be thought of as raw suggestions for what the system output should be in response to the current input conditions. Before the results can be applied, the fuzzy outputs must be further processed, or defuzzified, to produce a single output value that represents the combined effect of all of the fuzzy outputs. The CPU12 offers two variations of rule evaluation instructions. The REV instruction provides for unweighted rules (all rules are considered to be equally important). The REVW instruction is similar but allows each rule to have a separate weighting factor which is stored in a separate parallel data structure in the knowledge base. In addition to the weights, the two rule evaluation instructions also differ in the way rules are encoded into the knowledge base.

An example of a typical rule is:

If temperature is warm and pressure is high, then heat is (should be) off.

The antecedent portion of the rule is a statement of input conditions and the consequent portion of the rule is a statement of output actions. The antecedent portion of a rule is made up of one or more (in this case two) antecedents connected by a fuzzy and operator. Each antecedent expression consists of the name of a system input, followed by a label name. The label must be defined by a membership function in the knowledge base. Each antecedent expression corresponds to one of the fuzzy inputs in RAM. Since and is the only operator allowed to connect antecedent expressions, there is no need to include these in the encoded rule. The antecedents can be encoded as a simple list of pointers to (or addresses of) the fuzzy inputs to which they refer.

The consequent portion of a rule is made up of one or more (in this case one) consequents. Each consequent expression consists of the name of a system output, followed by is, and followed by a label name. Each consequent expression corresponds to a specific fuzzy output in RAM. Consequents for a rule can be encoded as a simple list of pointers to (or addresses of) the fuzzy outputs to which they refer.

The complete rules are stored in the knowledge base as a list of pointers or addresses of fuzzy inputs and fuzzy outputs. For the rule evaluation logic to work, there must be some means of knowing which pointers refer to fuzzy inputs and which refer to fuzzy outputs.

There also must be a way to know when the last rule in the system has been reached. The important method used in the CPU12, is to mark the end of the rule list with a reserved value, and separate antecedents and consequents with another reserved value. This permits any number of rules, and allows each rule to have any number of antecedents and consequents, subject to the limits imposed by availability of system memory. Two mathematical operations take place during rule evaluation.

The fuzzy and operator corresponds to the mathematical minimum operation and the fuzzy or operation corresponds to the mathematical maximum operation. The fuzzy and is used to connect antecedents within a rule.

The fuzzy or is implied between successive rules. Before evaluating any rules, all fuzzy outputs are set to zero (meaning not true at all).

As each rule is evaluated, the smallest (minimum) antecedent is taken to be the overall truth of the rule. This rule truth value is applied to each consequent of the rule (by storing this value to the corresponding fuzzy output) unless the fuzzy output is already larger (maximum).

2.4.3 Defuzzification (WAV)

The final step in the fuzzy logic program combines the raw fuzzy outputs into a composite system output. Unlike the trapezoidal shapes used for inputs, the CPU12 typically uses singletons for output membership functions. As with the inputs, the x-axis represents the range of possible values for a system output. Singleton membership functions consist of the x-axis position for a label of the system output. Fuzzy outputs correspond to the y-axis height of the corresponding output membership function.

The WAV instruction calculates the numerator and denominator sums for weighted average of the fuzzy outputs according to the formula:

$$\text{System Output} = \frac{\sum_{i=1}^{n} S_i . F_i}{\sum_{i=1}^{n} F_i}$$

Where n is the number of labels of a system output, S_i are the singleton positions from the knowledge base, and F_i are fuzzy outputs from RAM. The final divide is performed with a separate EDIV instruction placed immediately after the WAV instruction.

Before executing WAV, an accumulator must be loaded with the number of iterations (n), one index register must be pointed at the list of singleton positions in the knowledge base, and a second index register must be pointed at the list of fuzzy outputs in RAM.

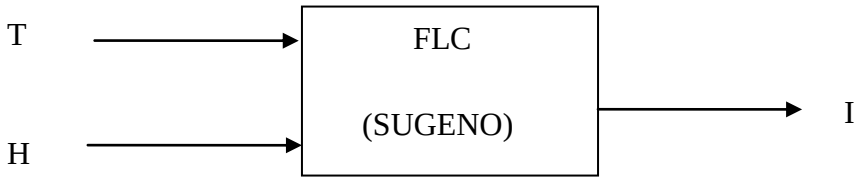
If the system has more than one system output, the WAV instruction is executed once for each system output.

2.4.4 Example

The aim of this example is to control the greenhouse windows depending on temperature and humidity inside it.

Our fuzzy logic controller is denoted by two inputs (the temperature (T) and the humidity (H)) and one output (the current (I)).

Its type is SUGENO fuzzy logic controller.



-The inputs:

Temperature (T) = {cold (C), hot (H), U_T=[17, 20], trapezoidal shape figure 2.10}

Humidity (H) = {dry (D), damp (Da), U_H=[54 ,60],trapezoidal shape figure 2.11}.

-The output:

u(t)={off, on, 2, 3, singleton, figure 2.12}.

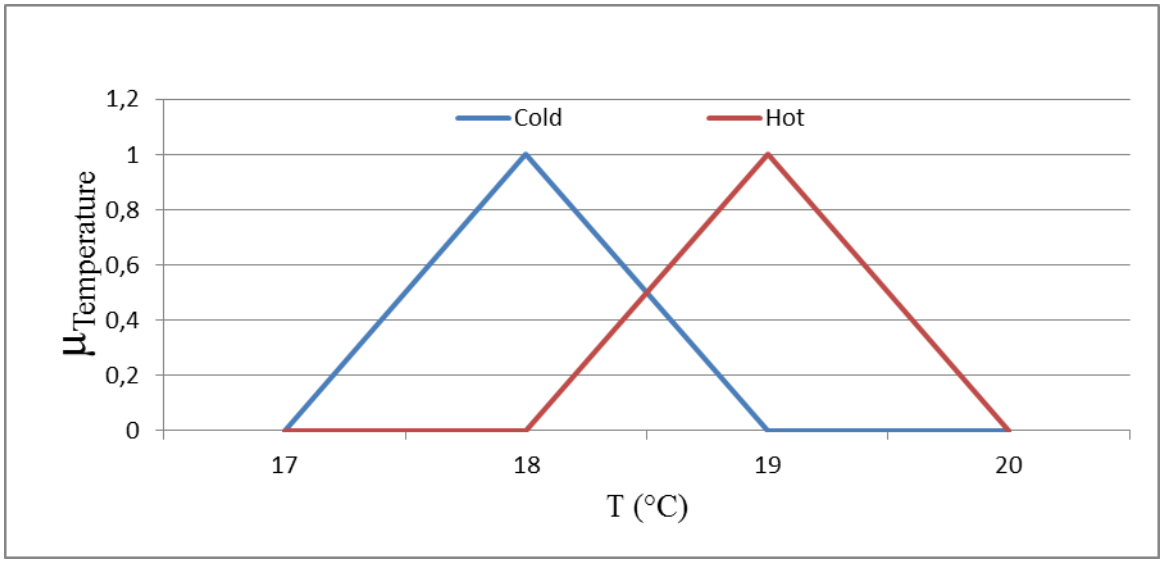


Figure2. 10 Temperature membership functions.

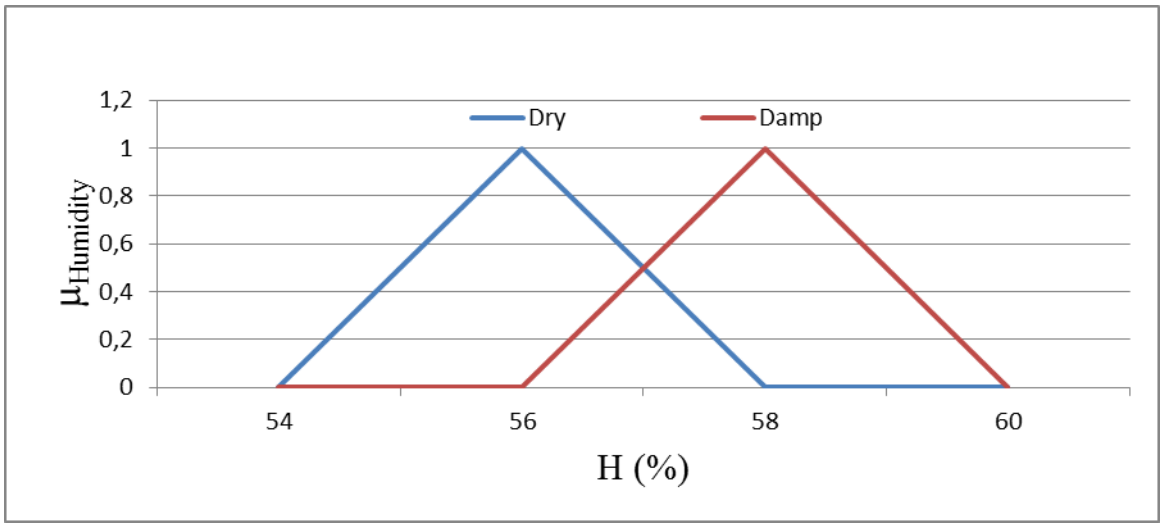


Figure2. 11 Humidity membership functions

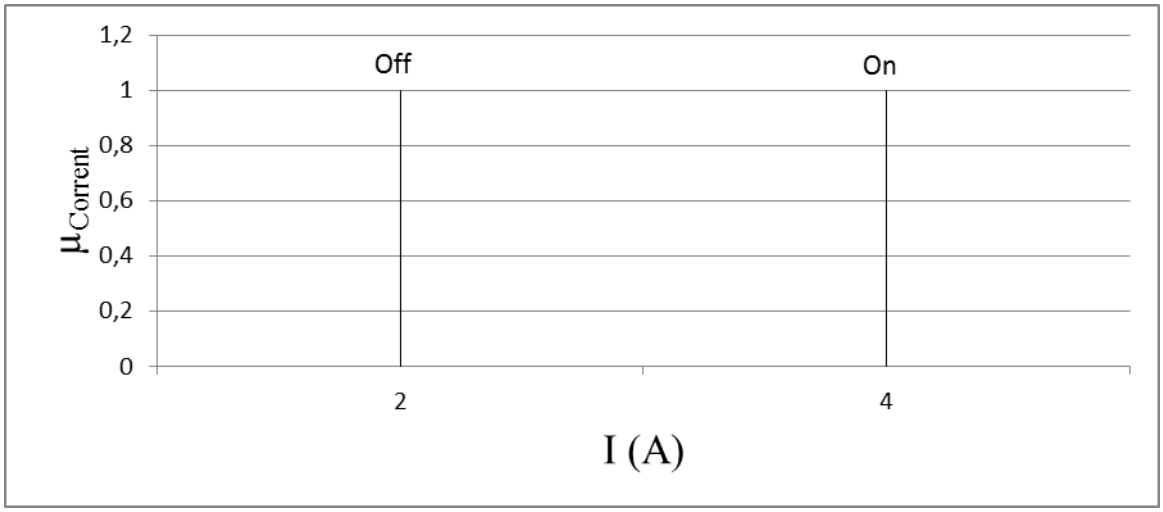


Figure2.12 Current output.

Inference mechanism is: min max**.**

Inference matrix is summarized in the following table:

Table 2.1 Inference matrix.

T / H	Cold (C)	Hot (H)
Dry (D)	Off	Off
Damp(Da)	Off	On

-That means that we have four rules:

Rule 1: if T is Cold and H is Dry than the Current Off.

Rule 2: if T is Cold and H is Damp than the Current Off.

Rule 3: if T is Hot and H is Dry than the Current Off.

Rule 4: if T is Hot and H is Damp than the Current On.

-The program in hc12 compact will be showing in the annex B

2.5 CONCLUSION

The HC12 compact module is a special microcontroller using to implement fuzzy logic algorithm by a special fuzzy logic instructions as MEM, REV, and WAV.

Chapter 3 Theoretical study of the inverted pendulum and project equipment

3.1 Introduction

The aim of this chapter is to make the mechanical study of an inverted pendulum in order to show its nonlinearity and instability and to show all project components.

3.2 Modeling the inverted pendulum

An inverted pendulum is a classic control problem. The process is nonlinear and unstable with one input signal and several output signals.
The aim is to balance a pendulum vertically on a motor driven wagon

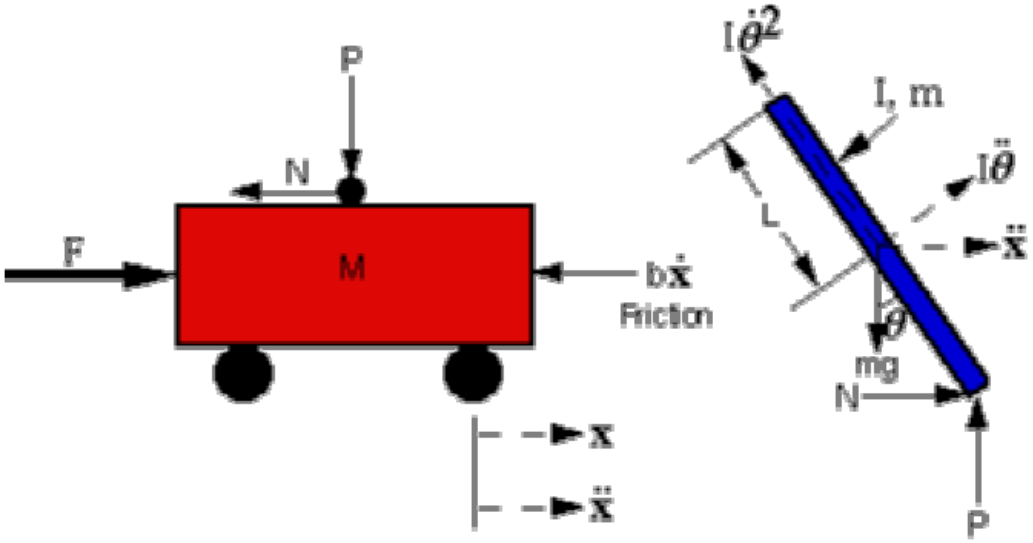


Figure3. 1 inverted pendulum's force representation.

With:

M: platform weight.

m: pendulum weight.

x:position.

Θ:pendulum angle.

Summing the forces in the Free Body Diagram of the car in the horizontal direction, we get the following equation of motion:

$$M\ddot{x} + b\dot{x} + N = F \ldots\ldots (3.1)$$

The force exerted in the horizontal direction due to the moment on the pendulum is determined as follows:

$$\tau = r\ X\ F = I\ddot{\theta} \qquad \ldots\ldots\ldots (3.2)$$

$$F = \frac{I\ddot{\theta}}{r} = mI\ddot{\theta} \qquad \ldots\ldots\ldots (3.3)$$

Component of this force in the direction of N is: $mI\ddot{\theta}\ cos\theta$

The component of the centripetal force acting along the horizontal axis is as follows:

$$F = \frac{I\dot{\theta}^2}{r} = mI\dot{\theta}^2 = mI\dot{\theta}^2 \qquad \ldots\ldots\ldots (3.4)$$

Component of this force in the direction of N is $mI\dot{\theta}^2 sin\theta$

Summing the forces in the Free Body Diagram of the pendulum in the horizontal Direction, we get an equation for N:

$$N = mx + mI\ddot{\theta}\ cos\ddot{\theta} + mI\dot{\theta}^2 sin\theta \quad \ldots\ldots (3.5)$$

If we substitute this equation (3.2) into the first equation (3.1) we get the first equation of motion for this system:

$$(M + m)\ddot{x} + b\dot{x} + mI\ddot{\theta}\ cos\theta - mI\dot{\theta}^2 sin\theta = F \ldots (3.6)$$

To get the second equation of motion, sum the forces perpendicular to the pendulum. This axis is chosen to simplify mathematical complexity. Solving the system along this axis ends up saving a lot of algebra. Just as the previous equation is obtained, the vertical components of those forces are considered here to get the following equation:

$$Psin\theta + Ncos\theta - mgsin\theta = mI\ddot{\theta} + m\ddot{x}\ cos\theta \ldots (3.7)$$

To get rid of the P and N terms in the equation above, sum the moments around the centroid of the pendulum to get the following equation:

$$-PIsin\theta - NIcos\theta = I\ddot{\theta} \ldots \text{(3.8)}$$

Combining these last two equations, we get the second dynamic equation:

$$(I + mI^2)\ddot{\theta} + mgIsin\theta = -mI\ddot{x}cos\theta \ldots \text{(3.9)}$$

The set of equations completely defining the dynamics of the inverted pendulum are:

$$(M + m)\ddot{x} + b\dot{x} + mI\ddot{\theta}\,cos\theta - mI\dot{\theta}^2 sin\theta = F \ldots \text{(3.6)}$$

$$(I + mI^2)\ddot{\theta} + mgIsin\theta = -mI\ddot{x}cos\theta \ldots \text{(3.9)}$$

These two equations are non-linear and need to be linearized for the operating range. Since the pendulum is being stabilized at an unstable equilibrium position, which is π radians from the stable equilibrium position, this set of equations should be linearized about $\theta = \pi$.

Assume that: $\theta = \pi + \emptyset$., (where $\emptyset$ represents a small angle from the vertical upward direction). Therefore $\cos\theta = -1$, $\sin\theta = -\emptyset$, and $\dot{\theta}^2 = 0$.

After linearization the two equations of motion become (u represents the input):

$$(M + m)\ddot{x} + b\dot{x} - mI\ddot{\emptyset} = u \ldots \text{(3.10)}$$

$$(I + mI^2)\ddot{\emptyset} - mgI\emptyset = mI\ddot{x} \ldots \text{(3.11)}$$

To obtain the transfer function of the linearized system equations analytically, we must first take the Laplace transform of the system equations.

The Laplace transforms are:

$$(M + m)S^2X(S) + bSX(S) - mIS^2\emptyset(S) = u(S)$$

$$(I + mI^2)S^2\emptyset(S) - mgI\emptyset(S) = mIS^2X(S)$$

So we obtain:

$$X(S) = \left(\frac{I + mI^2}{mI} - \frac{g}{S^2}\right)\emptyset(S)$$

Then, substituting into the second equation will yield:

$$(M + m)\left[\frac{I + mI^2}{mI} - \frac{g}{S^2}\right]\emptyset(S)S^2 + b\left[\frac{I + mI^2}{mI} - \frac{g}{S^2}\right]\emptyset(S).S - mIS^2\emptyset(S) = U(S)$$

Re-arranging, the transfer function is:

$$\frac{\emptyset(S)}{U(S)} = \frac{\frac{mI}{q}S^2}{S^4 + \frac{b(I+mI^2)}{q}S^3 - \frac{mgl(M+m)}{q}S^2 - \frac{bmgl}{q}S}$$

where :

$q = (M+m)(I+mI^2) - (mI)^2$

From the transfer function above it can be seen that there is both a pole and a zero at the origin. These can be canceled and the transfer function becomes:

$$\frac{\emptyset(S)}{U(S)} = \frac{\frac{mI}{q}S}{S^3 + \frac{b(I+mI^2)}{q}S^2 - \frac{mgl(M+m)}{q}S - \frac{bmgl}{q}}$$

The transfer function can thus be simplified as:

$$\frac{\emptyset(S)}{U(S)} = \frac{mI.s}{q.s^3 + b(I+mI^2)s^2 - mgl(M+m)s - bmgl}$$

3.3 Project Hardware tools

3.3.1 Introduction

In this section we will describe the hardware part of our project. As stated above the aim of this work is to test an FLC controller applied to an inverted pendulum. The proposed setup consists of a cart moving forward and backward on top of which the axis of the pendulum will be fixed. The moving platform is equipped with two DC motors and a caster wheel as shown in figure 3.2. To supply these motors with the right power, at each instant, a double H-Bridge whose inputs are connected to an arduino type board that is used to generate the PWM signals used to control the speed and direction of both motors. This board gets the duty cycle of these PWMs from the HC12 board after their computation by the FLC controller. The position angle of the pendulum is measured through a potentiometer fixed so as to minimize friction and distortion.

Figure3. 2 the robot

3.3.2 Mechanical and electrical tools

The overall hardware setup consists of the following parts:

a HC12 compact

It is a controller module from MOTOROLA which represents the controller part and we used it to implement the FLC algorithm.

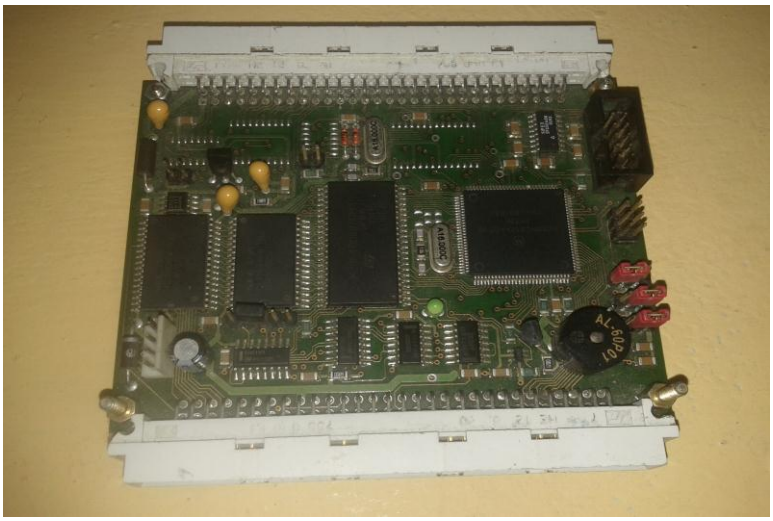


Figure3. 3 HC12 compact

b The Arduino UNO interface board

This is an ARDUINO type board used here just to generate the PWM signals to the DC motor.

Figure3. 4 Arduino UNO

c EMG30 DC motors

The EGM30 (encoder, motor, gearbox 30:1) is a 12v motor fully equipped with encoders and a 30:1 reduction gearbox.

It is ideal for small or medium robotic applications, providing cost effective drive and feedback for the user. It also includes a standard noise suppression capacitor across the motor windings.

Figure 3. 5 EMG30 DC motor.

- Specification

- Rated voltage 12v.
- Rated torque 1.5kg/cm.
- Rated speed 170rpm.
- Rated current 530mA.
- No load speed 216.
- No load current 150mA.
- Stall Current 2.5A.
- Rated output 4.22W.
- Encoder counts per output shaft turn 360.

d Potentiometer

Potentiometers are good position sensors. In our project we used potentiometer in order to measure, at each instant, the pendulum position. A potentiometer is a type of variable resistor that is used in circuits having low power. They are used to divide voltage and they come with three terminals.

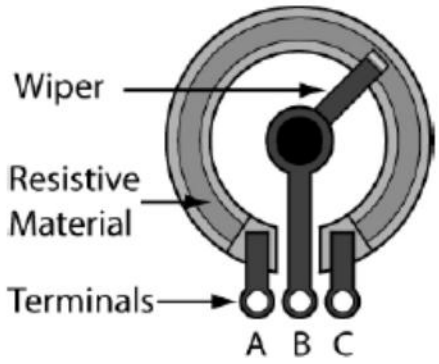


Figure3. 6 Basic Potentiometer Construction

The resistance between A and C is constant and the value of the potentiometer. The resistance between A and B, B and C changes according to the position of the wiper.

e The robot platform

This the mobile part on which we make the inverted pendulum, it's made of:

- Two wheels driven by the EMG30 dc motor rotate forward and backward depending on the control signal

- A caster wheel.

Figure3. 7 Robot platforme

f Battery

A 12 volt battery of 4.5 amps to supply the two dc motors through the power board and control boards (HC12 and Arduino).

g Arduino UNO power board

Between the controller (hc12 microcontroller) and the actuators (the EMG 30 dc motors) we need a power board ,the following schematic represents the power board used in the project which is made of L298 dual full bridge driver.

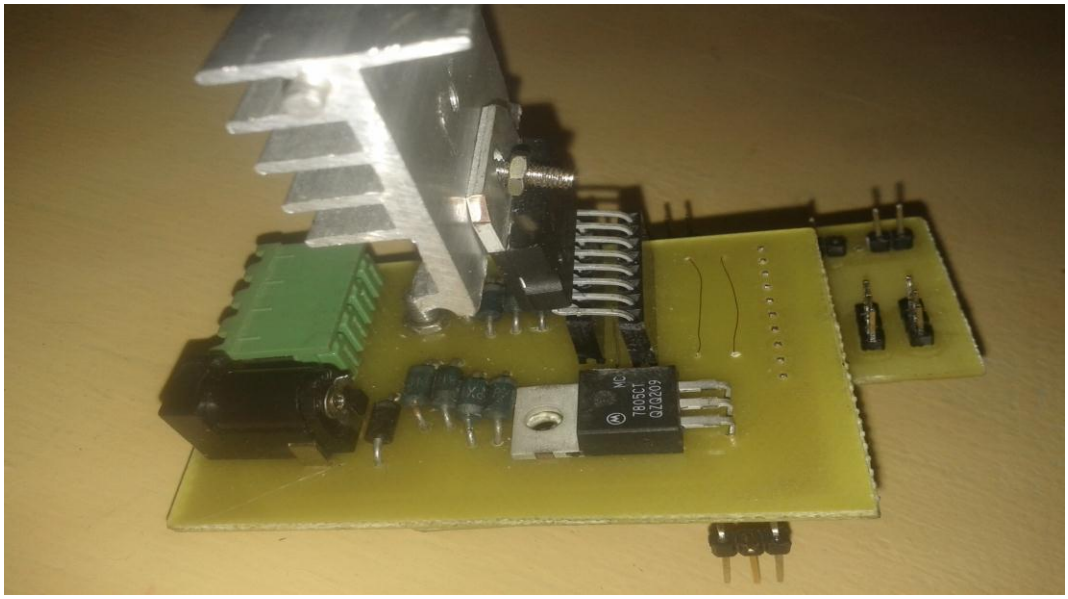


Figure 3. 8 Arduino UNO power board

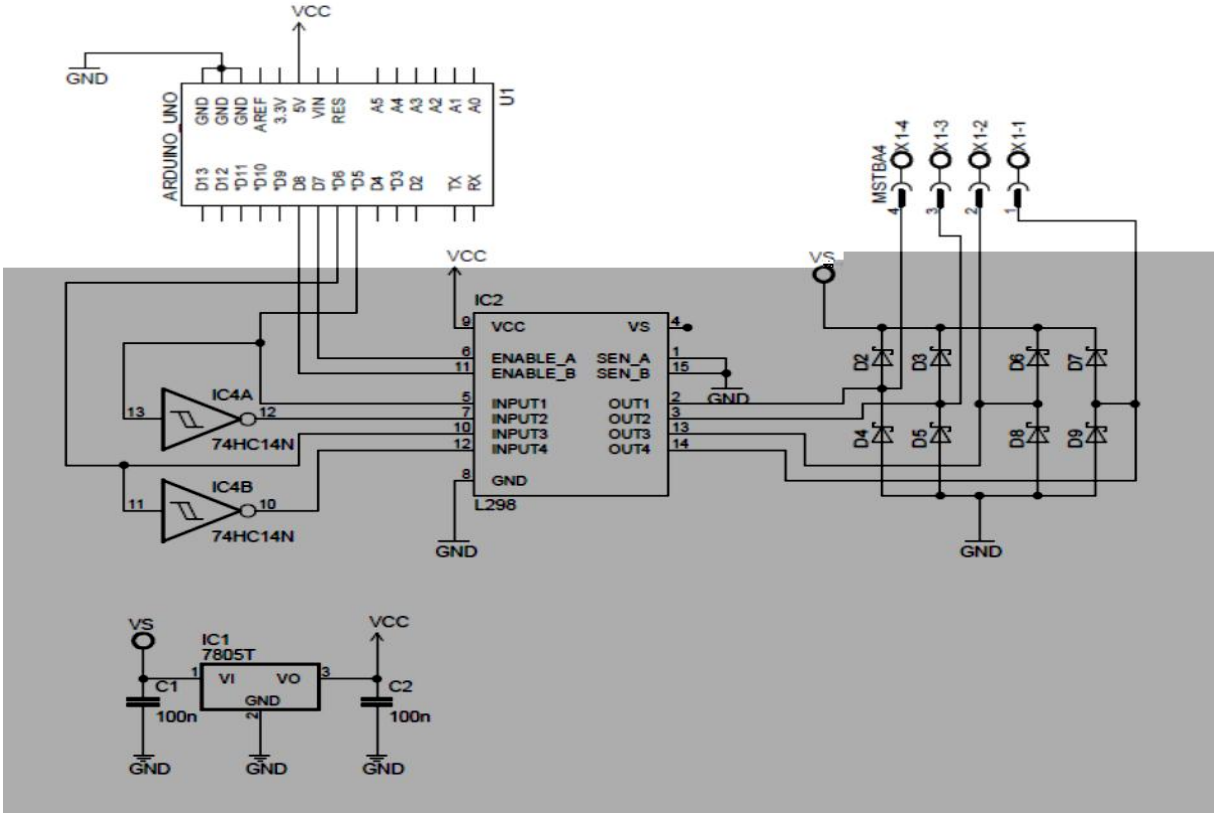


Figure 3.9 power board schematic.

- The L298 is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors.

Two enable inputs are provided to enable or disable the device independently of the input signals.

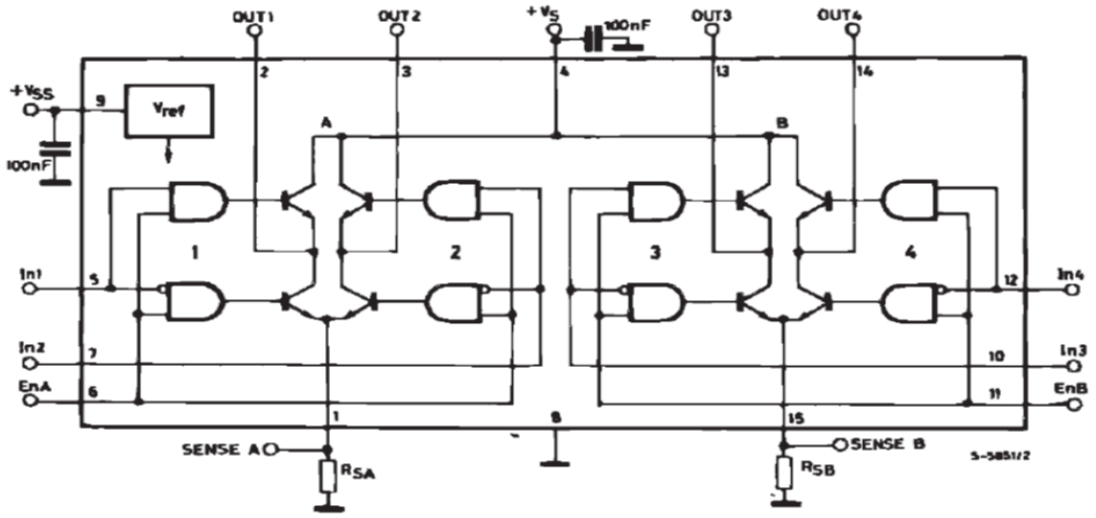


Figure 3. 10 The block diagram of the L298.

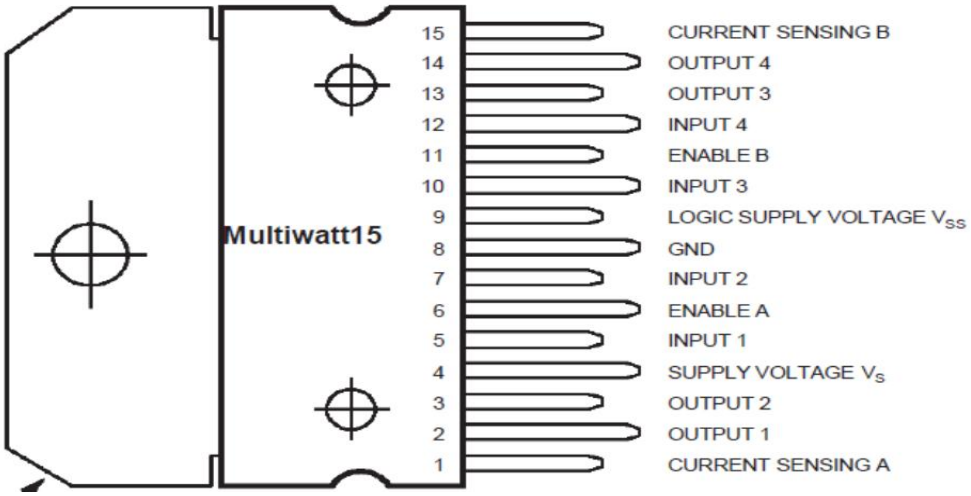


Figure 3. 11 The Pin connections of L298.

3.4 Conclusion

To achieve the pendulum stability, many tools and equipment were used as the platform and its component, the Arduino power board, a potentiometer sensor, an Arduino UNO microcontroller, a HC12 compact module microcontroller, Two EMG30 DC motors.

Chapter 4 Project synthesis and results analyzes

4.1 Introduction

The most important part of our project will be discussed in this chapter. This will include the experimental results and the problems we faced to make the overall system working together.

4.2 Choice of the FLC algorithm parameters

The FLC parameters have chosen as follows. First we choose the following configuration:

-inputs:

error={negative(N),zero(Z),positive(P),U∈[-7,7],trapezoidal}.

derror={negative(N),zero(Z),positive(P),U∈[-9,9],trapezoidal}.

-output:

control={negative big(NB), negative(N),zero(Z),positive(P), positive big(PB),-5, -2,0,2,5,signeleton}.

The inference mechanism: min-max.

Table4. 1 Inference matrix.

error / derror	Negative (N)	Zero(Z)	Positive (P)
Negative (N)	NB	PB	PB
Zero(Z)	N	Z	P
Positive (P)	NB	NB	NB

That means that we have nine rules:

Rule 1:if error is negative (N) and derror is negative (N) then the of control signal is negative big.(NB)

Rule 2:if error is negative (N) and derror is zero(Z) than the signal of control is negative (N).

Rule 3:if error is negative (N) and derror is positive(P) than the signal of control is negative big (NB).

Rule 4:if error is zero (Z) and derror is negative(N) than the signal of control is positive big(PB).

Rule 5:if error is zero (Z) and derror is zero(Z) than the signal of control is zero(Z).

Rule 6:if error is zero (Z) and derror is positive(P) than the signal of control is negative big(NB).

Rule 7:if error is positive (P) and derror is negative (N)than the signal of control is positive big(PB).

Rule 8:if error is positive(P) and derror is zero(Z) than the signal of control is positive(P).

Rule 9:if error is positive (P) and derror is positive(P) than the signal of control is negative big.

4.3 design of the fuzzy logic controller in hc12 compact

As we say in chapter two, we used the fuzzy logic instruction of HC12 to build the fuzzy logic controller like: MEM, REV, etc ….

- The first step(knowledge base) :

-The error: is defined by deference between the reference voltage and the accelerometer voltage.

-The derror: is defined by deference between the new value and the last value of error.

Then in hc12 we denote membership functions, in this step we crash with a problem of the negative value in the discourse universe because in the microprocessor the range of discourse universe can varies from $00 to $FF

To solve this problem we add a positive value to the value of discourse universe so we obtain a positive discourse universe.

-For the output, the same problem as the input (negative values) and the same way to solve this problem.

- The second step (fuzzification)

We put the first input address in an accumulator, the first membership function definition in the knowledge base is pointed by an index register and the first fuzzy input in RAM is pointed by another index register, finally we use the MEM instruction.

- The third step (rule evaluation):

in the RAM we define the nine rules, than we point the first index register at the first rule address and the second index register at the fuzzification output address ,we load the accumulator A with $FF than finally we used REV instruction.

- The final step (the defuzzification):

We point the first index register with the address of fuzzy out(output of rules evaluation step), the second index register with the address of the output singleton ,than we use WAV and EDIV instructions in order to obtain the result of deffuzification .

4.4 Using ARDUINO UNO as generator of PWM signal

After the implementation of the algorithm in HC12 microcontroller we send the control signal using one of the general purpose ports (Inputs/Outputs).

The ARDUINO UNO received this signal then we tried to find relation between it and the duty cycle of the PWM signal.

Many tests were done to improve the system response.

4.5 Analyze of the results

When we made the first test we observe that the controller give a wrong results because a small variation in the pendulum doesn't change the robot speed so the robot cannot check the pendulum stability, because of that we change the controller design (choose another universe of discourse and another membership functions).
In the second test we choose as a universe of discourse the following configuration: the error : U∈ [-5,5], the derror : U∈ [-7,7] and the control : U∈ {-2,-1,0,1,2}. For this configuration and after many tests we found that the system response did not give a really good response, but better than the first results when the variation of 10° change the robot speed but always the system response isn't enough.
For the third test made the following configuration:
the error: U∈ [-1, 1], the derror: U∈ [-1.5, 1.5] and the control: U∈ {-2,-1, 0, 1, 2}. After all those tests, we can consider the third fuzzy logic controller as a good controller for our system due to its results.

Conclusion

The main task of our project was to implement a fuzzy logic type controller on an embedded system. After having carried out many experiments on the constructed mobile platform we have come to the following conclusions; Even if from the theoretical point of view everything seems quite simple using simulation tools, this is not the case when it comes to implement the same results on an actual hardware. Indeed many problems appear especially when dealing with mechanical construction, where friction and other misalignment of some mechanical parts are sources of many difficulties which make this implementation even more complex. As stated previously, the problem of the well-known inverted pendulum has been the subject of the present thesis. Despite the fact that the same problem has been extensively studied by many scientists we did not come across a similar architecture as the one we adopted. This choice comes from the fact that when using fuzzy logic it not necessary to have an analytical model of the process under control, and our aim was to test this assumption on an experimental platform. This does not mean that very good results can obtained easily but at the price of the time it takes to test many possibilities before the final choice of the controller parameters (universe of discourse, membership functions …).

In our case we have been able to keep the proposed the pendulum in a near vertical position. Considering the difficulties we stated above it would have very difficult to expect a perfect result. The most important result that we are sure about is that this project has provided us considerable experience not only on the practical issues that are faced when it comes to design a hardware system but also gave us more confidence for eventually getting involved in similar projects in the future.

As an extension for the present work we have to insist on the improvement of the mechanical aspect of the hardware platform and using more powerful motors, this should make it less difficult to test similar algorithms using the same electronics and firmware that implements the actual controller.

Annexe A

-CPU Registers:

-Accumulators A and B:

Accumulators A and B are general-purpose 8-bit accumulators that contain operands and results of arithmetic calculations or data manipulations.

Bit 7	6	5	4	3	2	1	Bit 0
A7	A6	A5	A4	A3	A2	A1	A0

Reset: Unaffected by reset

Bit 7	6	5	4	3	2	1	Bit 0
A7	A6	A5	A4	A3	A2	A1	A0

Reset: Unaffected by reset

-Accumulators D:

Accumulator D is the concatenation of accumulators A and B. Some instructions treat the combination of these two 8-bit accumulators as a 16-bit double accumulator.

-index register X and Y:

Index registers X and Y are used for indexed addressing. Indexed addressing adds the value in an index register to a constant or to the value in an accumulator to form the effective address of the operand.

Index registers X and Y can also serve as temporary data storage locations stack pointer:

The stack pointer (SP) contains the last stack address used. The CPU12 supports an automatic program stack that is used to save system context during subroutine calls and interrupts.

The stack pointer can also serve as a temporary data storage location or as an index register for indexed addressing.

-program counter

The program counter contains the address of the next instruction to be executed.

The program counter can also serve as an index register in all indexed addressing modes except autoincrement and autodecrement.

-condition code register:

	Bit 7	6	5	4	3	2	1	Bit 0
	S	X	H	I	N	Z	V	C
Reset:	1	1	u	1	u	u	u	u

u = Unaffected

S — Stop Disable Bit.

Setting the S bit disables the STOP instruction.

X — XIRQ Interrupt Mask Bit.

Setting the X bit masks interrupt requests from the XIRQ pin.

H — Half-Carry Flag.

The H flag is used only for BCD arithmetic operations. It is set when an ABA, ADD, or ADC instruction produces a carry from bit 3 of accumulator A. The DAA instruction uses the H flag and the C flag to adjust the result to correct BCD format.

I — Interrupt Mask Bit.

Setting the I bit disables maskable interrupt sources.

N — Negative Flag.

The N flag is set when the result of an operation is less than 0.

Z — Zero Flag.

The Z flag is set when the result of an operation is all 0s.

V — two's Complement Overflow Flag.

The V flag is set when a two's complement overflow occurs.

C — Carry/Borrow Flag.

The C flag is set when an addition or subtraction operation produces a carry or borrows.

-SCI PORT:

-SCI 1 Baud Rate Register High (SC1BDH): $00C8

	Bit 7	6	5	4	3	2	1	Bit 0
	BTST	BSPL	BRLD	SBR12	SBR11	SBR10	SBR9	SBR8
Reset:	0	0	0	0	0	0	0	0

-SCI 1 Baud Rate Register low (SC1BDl): $00C9

	Bit 7	6	5	4	3	2	1	Bit 0
	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0
Reset:	0	0	0	0	0	1	0	0

-SCI 1 control register 1 (SC1CR1): $00CA

	Bit 7	6	5	4	3	2	1	Bit 0
	LOOPS	WOMS	RSRC	M	WAKE	ILT	PE	PT
Reset:	0	0	0	0	0	0	0	0

-SCI 1 control register 2 (SC1CR2) : $00CB

	Bit 7	6	5	4	3	2	1	Bit 0
	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
Reset:	0	0	0	0	0	0	0	0

-SCI 1 status register 1 (SC1SR1): $00CC

	Bit 7	6	5	4	3	2	1	Bit 0
Read	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
Write								
Reset:	1	1	0	0	0	0	0	0

-SCI 1 status register 2 (SC1SR2): $00CD

	Bit 7	6	5	4	3	2	1	Bit 0
Read	0	0	0	0	0	0	0	RAF
Write								
Reset:	0	0	0	0	0	0	0	0

-SCI 1 data register high (SC1DRH) : $00CE

	Bit 7	6	5	4	3	2	1	Bit 0
Read	R8	T8	0	0	0	0	0	RAF
Write								
Reset:	unaffected by reset							

-SCI 1 data register low (SC1DRl) : $00CF

	Bit 7	6	5	4	3	2	1	Bit 0
Read	R7	R6	R5	R4	R3	R2	R1	R0
Write	T7	T6	T5	T4	T3	T2	T1	T0
Reset:	unaffected by reset							

-ATD:

-ATD control registers 0 (ATDCTL0) ($0060)

	Bit 7	6	5	4	3	2	1	Bit 0
Read	0	0	0	0	0	0	0	0
Reset:	0	0	0	0	0	0	0	0

-ATD control register 1 (ATDCTL1) ($0061)

	Bit 7	6	5	4	3	2	1	Bit 0
Read	0	0	0	0	0	0	0	0
Reset:	0	0	0	0	0	0	0	0

-ATD control register 2 (ATDCTL2) ($0062)

	Bit 7	6	5	4	3	2	1	Bit 0
Read	ADPU	AFFC	AWAI	0	0	0	ASCIE	ASCIF
Reset:	0	0	0	0	0	0	0	0

-ATD control registers 3 (ATDCTL3) ($0063)

	Bit 7	6	5	4	3	2	1	Bit 0
Read	0	0	0	0	0	0	FRZ1	FRZ0
Reset:	0	0	0	0	0	0	0	0

-ATD control registers 4 (ATDCTL4) ($0064)

	Bit 7	6	5	4	3	2	1	Bit 0
Read	0	SMP1	SMP0	PRS4	PRS3	PRS2	PRS1	PRS0
Reset:	0	0	0	0	0	0	0	1

-ATD control registers 5 (ATDCTL5) ($0065)

	Bit 7	6	5	4	3	2	1	Bit 0
Read	0	S8CM	SCAN	MULT	CD	CC	CB	CA
Reset:	0	0	0	0	0	0	0	0

-ATD status registers 1 (ATDSTAT1) ($0066)

	Bit 7	6	5	4	3	2	1	Bit 0
Read	SCF	0	0	0	0	CC2	CC1	CC0
Reset:	0	0	0	0	0	0	0	0

-ATD status registers 2 (ATDSTAT2) ($0067)

	Bit 7	6	5	4	3	2	1	Bit 0
Read	CCF7	CCF6	CCF5	CCF4	CCF3	CCF2	CCF1	CCF0
Reset:	0	0	0	0	0	0	0	0

-ATD test registers 1 (ATDCTEST1) ($0068)

	Bit 7	6	5	4	3	2	1	Bit 0
Read	SAR9	SAR8	SAR7	SAR6	SAR5	SAR4	SAR3	SAR2
Reset:	0	0	0	0	0	0	0	0

-ATD test registers 2 (ATDCTEST2) ($0069)

	Bit 7	6	5	4	3	2	1	Bit 0
Read	SAR1	SAR0	RST	TSTOUT	TST3	TST2	TST1	TST0
Reset:	0	0	0	0	0	0	0	0

-port AD data input register (PORTAD) ($006F)

	Bit 7	6	5	4	3	2	1	Bit 0
Read	PAD7	PAD6	PAD5	PAD4	PAD3	PAD2	PAD1	PAD0
Reset:	0	0	0	0	0	0	0	0

-ATD result register0 (ADR0H) ($0070)

	Bit 7	6	5	4	3	2	1	0
Read	ADRxH7	ADRxH6	ADRxH5	ADRxH4	ADRxH3	ADRxH2	ADRxH1	ADRxH0
Reset:	0	0	0	0	0	0	0	0

-ATD result register1 (ADR1H) ($0072)

	Bit 7	6	5	4	3	2	1	Bit 0
Read	ADRxH7	ADRxH6	ADRxH5	ADRxH4	ADRxH3	ADRxH2	ADRxH1	ADRxH0
Reset:	0	0	0	0	0	0	0	0

-ATD result register2 (ADR2H) ($0074)

	Bit 7	6	5	4	3	2	1	Bit 0
Read	ADRxH7	ADRxH6	ADRxH5	ADRxH4	ADRxH3	ADRxH2	ADRxH1	ADRxH0
Reset:	0	0	0	0	0	0	0	0

-ATD result register3 (ADR3H) ($0076)

	Bit 7	6	5	4	3	2	1	Bit 0
Read	ADRxH7	ADRxH6	ADRxH5	ADRxH4	ADRxH3	ADRxH2	ADRxH1	ADRxH0
Reset:	0	0	0	0	0	0	0	0

-ATD result register4 (ADR4H) ($0078)

	Bit 7	6	5	4	3	2	1	Bit 0
Read	ADRxH7	ADRxH6	ADRxH5	ADRxH4	ADRxH3	ADRxH2	ADRxH1	ADRxH0
Reset:	0	0	0	0	0	0	0	0

-ATD result register5 (ADR5H) ($007A)

	Bit 7	6	5	4	3	2	1	Bit 0
Read	ADRxH7	ADRxH6	ADRxH5	ADRxH4	ADRxH3	ADRxH2	ADRxH1	ADRxH0
Reset:	0	0	0	0	0	0	0	0

-ATD result register6 (ADR6H) ($007C)

	Bit 7	6	5	4	3	2	1	Bit 0
Read	ADRxH7	ADRxH6	ADRxH5	ADRxH4	ADRxH3	ADRxH2	ADRxH1	ADRxH0
Reset:	0	0	0	0	0	0	0	0

-ATD result register7 (ADR7H) ($007E)

	Bit 7	6	5	4	3	2	1	Bit 0
Read	ADRxH7	ADRxH6	ADRxH5	ADRxH4	ADRxH3	ADRxH2	ADRxH1	ADRxH0
Reset:	0	0	0	0	0	0	0	0

-Port AD Data Register ($006F)

	Bit 7	6	5	4	3	2	1	Bit 0
Read	PAD7	PAD6	PAD5	PAD4	PAD3	PAD2	PAD1	PAD0
Reset:	0	0	0	0	0	0	0	0

-Port J Data Direction Register (DDRJ) ($0029)

	Bit 7	6	5	4	3	2	1	Bit 0
Read	DDRJ7	DDRJ6	DDRJ5	DDRJ4	DDRJ3	DDRJ2	DDRJ1	DDRJ0
Reset:	0	0	0	0	0	0	0	0

-Port J Data Register (PORTJ) ($0028)

	Bit 7	6	5	4	3	2	1	Bit 0
Read	PJ7	PJ6	PJ5	PJ4	PJ3	PJ2	PJ1	PJ0
Reset:	0	0	0	0	0	0	0	0

-Port H Data Direction Register ($0025) (DDRH)

	Bit 7	6	5	4	3	2	1	Bit 0
Read	DDRH7	DDRH6	DDRH5	DDRH4	DDRH3	DDRH2	DDRH1	DDRH0
Reset:	0	0	0	0	0	0	0	0

-Port H Data Register (PORTH) ($0024)

	Bit 7	6	5	4	3	2	1	Bit 0
Read	PH7	PH6	PH5	PH4	PH3	PH2	PH1	PH0
Reset:	0	0	0	0	0	0	0	0

Annexe B

-fuzzy logic implementation example:

```
//1-temperature//
//1st input Cold //
asm("LDAB #$11");
asm("STAB $1000");
asm("LDAB #$13");
asm("STAB $1001");
asm("LDAB #$ff");
asm("STAB $1002");
asm("LDAB #$ff");
asm("STAB $1003");
//1st input Hot //
asm("LDAB #$12");
asm("STAB $1004");
asm("LDAB #$14");
asm("STAB $1005");
asm("LDAB #$ff");
asm("STAB $1006");
asm("LDAB #$ff");
asm("STAB $1007");
//2-humidité//
//1st input Dry //
asm("LDAB #$35");
asm("STAB $1008");
asm("LDAB #$3a");
asm("STAB $1009");
asm("LDAB #$7f");
```

```
asm("STAB $100a");
asm("LDAB #$7f");
asm("STAB $100b");
//2nd input Damp //
asm("LDAB #$38");
asm("STAB $100c");
asm("LDAB #$3c");
asm("STAB $100d");
asm("LDAB #$7f");
asm("STAB $100e");
asm("LDAB #$7f");
asm("STAB $100f");
asm("LDAA #$12");    //the input T=18°C//
asm("LDX #$1000");    //temperature fuzzification //
asm("LDY #$1020");
asm("LDAB #2");
asm("loop_temp: MEM");
asm("DBNE B,loop_temp");
asm("LDAA #$37");   // the input H=55% //
asm("LDAB #2");         //humidity fuzzification //
asm("loop_temp1: MEM");
asm("DBNE B,loop_temp1");
//rule evaluation//
asm("ldab #4");
asm("RULE_EVAL: CLR 1,Y+");
asm(" DBNE B,RULE_EVAL");
//rules //
//first rule//
asm("ldy #$1020");
asm("ldaa #$00");
asm("staa $1050");
asm("ldaa #$02");
asm("staa $1051");
asm("ldaa #$fe");
```

```
asm("staa $1052");
asm("ldaa #$04");
asm("staa $1053");
asm("ldaa #$fe");
asm("staa $1054");
//second rule//
asm("ldaa #$00");
asm("staa $1055");
asm("ldaa #$03");
asm("staa $1056");
asm("ldaa #$fe");
asm("staa $1057");
asm("ldaa #$05");
asm("staa $1058");
asm("ldaa #$fe");
asm("staa $1059");
//third rule//
asm("ldaa #$01");
asm("staa $105a");
asm("ldaa #$02");
asm("staa $105b");
asm("ldaa #$fe");
asm("staa $105c");
asm("ldaa #$06");
asm("staa $105d");
asm("ldaa #$fe");
asm("staa $105e");
//fourth rule//
asm("ldaa #$01");
asm("staa $105f");
asm("ldaa #$03");
asm("staa $1060");
asm("ldaa #$fe");
asm("staa $1061");
```

```
asm("ldaa #$07");
asm("staa $1062");
asm("ldaa #$ff");
asm("staa $1063");
asm("ldx #$1050");
asm("ldy #$1020");
asm("ldaa #$ff");
asm("rev");
//current//
asm("ldaa #$02");
asm("staa $1070");
asm("ldaa #$02");
asm("staa $1071");
asm("ldaa #$02");
asm("staa $1072");
asm("ldaa #$03");
asm("staa $1073");
//defuzzification //
asm("ldy #$1024")
asm("ldx #$1070");
asm("ldab #4");
asm("wav");
asm("ediv");
asm("sty $1080");//current value//
}
```

Bibliography

[1] Freescale semiconductor: 'MC68HC812A4 DATASHEET', Freescale semiconductor, 2005/2006.

[2] ELMICRO COMPUTER: 'HC12compact user manual', ELMICRO COMPUTER, 2008.

[3] Guanrong Chen, Trung Tat Pham: 'Introduction to fuzzy set, Fuzzy logic and Fuzzy control systems', CRC Press, 2001.

[4] Khalil Sultan: 'Inverted Pendulum, Analysis, Design and Implementation', 2002.