DESIGN OF A NEURON CONTROLLER FOR CLIMATIZED BROILER HOUSES

ALISON ZILLE LOPES

2009

ALISON ZILLE LOPES

DESIGN OF A NEURON CONTROLLER FOR CLIMATIZED BROILER HOUSES

Dissertation presented to Federal University of Lavras as partial fulfillment of the requirements of Graduate program in Systems Engineering, Systems Modeling and Analysis concentration area, to obtain Master's degree.

Adviser

Prof. DSc. Tadayuki Yanagi Junior

Co-Advisers

Prof. DSc. Wilian Soares Lacerda

Prof. DSc. Giovanni Francisco Rabelo

LAVRAS

MINAS GERAIS - BRAZIL

2009

Ficha Catalográfica Preparada pela Divisão de Processos Técnicos da Biblioteca Central da UFLA

Lopes, Alison Zille.
Design of a neuron controller for climatized broiler houses / Alison Zille Lopes. – Lavras : UFLA, 2009.
140 p. : il.

Dissertação (Mestrado) – Universidade Federal de Lavras, 2009.
Orientador: Tadayuki Yanagi Junior
Bibliografia.

1. Poultry. 2. Thermal confort. 3. Environment control 4. Embedded System. 5. Artificial neural network I. Universidade Federal de Lavras. II. Título.

CDD – 620.0042
– 636.50831

ALISON ZILLE LOPES

DESIGN OF A NEURON CONTROLLER FOR CLIMATIZED BROILER HOUSES

Dissertation presented to Federal University of Lavras as partial fulfillment of the requirements of Graduate program in Systems Engineering, Systems Modeling and Analysis concentration area, to obtain Master's degree.

APPROVED in February 19, 2009

Prof. DSc. Wilian Soares Lacerda	UFLA
Prof. DSc. Giovanni Francisco Rabelo	UFLA
Prof. DSc. André Vital Saúde	UFLA
Prof. DSc. Sérgio Zolnier	UFV

Prof. DSc. Tadayuki Yanagi Junior
UFLA
(Adviser)

LAVRAS
MINAS GERAIS - BRAZIL

ACKNOWLEDGEMENTS

This dissertation would not have been written without the support, confidence and encouragement of my family and friends. The freedom offered by my family, allow me to follow my own steps. My friends assumed an important role in this point of my life, mainly Crysttian Arantes Paixão who encouraged me to join him in a graduate course. The love, support and constant patience of my girlfriend Isis were my motivation to go ahead.

I would like to thank the Federal University of Lavras (UFLA), through the Engineering Department, for the opportunity and support during the master's degree course; and to CAPES for providing financial support.

I wish to express my gratitude to Prof. Tadayuki Yanagi Junior for the opportunity to work in this challenge and for the guidance, friendship, dedication, confidence and the exemplar professionalism showed during this research.

I am very grateful to Prof. Wilian Soares Lacerda for everything I learned with him, for his dedication and, above all, for his trust in my capacities, showing encouragement and friendship.

I would also like to thank Prof. Giovanni Francisco Rabelo for the attention and constant dedication, mainly, during the period where he was the coordinator of the Graduate Program.

I thank all professors connected to the Systems Engineering program for the suggestions and lessons. I also thank the course colleagues, ever present, and the engineering department staff, ever friendly and attentively.

Finally, I thank everybody that contributed for this important personal conquest.

Thank you very much!

SUMMARY

LIST OF ABBREVIATIONS, ACRONYMS AND SYMBOLS

ΔT_{90}	Body temperature rise after 90-min exposures to the thermal conditions, ºC;
A/D	Analog/digital;
AC	Alternate current;
ANN	Artificial Neural Network;
CMOS	Complementary metal oxide semiconductor;
CRC	Cyclic redundancy check;
DB7-DB0	Eight bits LCD module data line, where DB means data bit and the number refers to the bit position;
DC	Direct current;
DE	Data enable pin of MAX 485;
DI	Data input pin of MAX485;
E1	LCD half-up chip enable signal;
E2	LCD half-down chip enable signal;
EEPROM	Electrically erasable programmable read-only memory;
GND	Ground;
I/O	Input/output;
I^2C	Inter-integrated circuit;
IBM-PC	International Business Machines- personal computer;
LCD	Liquid crystal display;
LED	Light emitting diode;

LED-	Negative pole of LCD back light power supply;
LED+	Positive pole of LCD back light power supply;
MLP	*Multilayer perceptron*;
MSE	Mean square error;
MSSP	Master synchronous serial port;
NC	Normally closed;
NO	Normally open;
NPN	Negative-positive-negative;
PIC	Peripheral interface controller;
PTAT	Proportional to absolute temperature;
PWM	Pulse-width modulation;
R/W	Read/write;
R^2	Determination coefficient;
RAM	Random access memory;
RE	Complementary reception enable pin of MAX 485;
RH	Relative humidity, %;
RH_{linear}	Linear relative humidity, %;
RH_{true}	True relative humidity, %;
RISC	Reduced instruction set computer;
RO	Reception output of MAX485;
ROM	Read only memory;
RS	Register select;

RX	Reception;
SCK	Serial clock;
SO_{RH}	The two bytes of sensor output for relative humidity calculation;
SO_T	The two bytes of sensor output for air temperature calculation;
SPI	Serial peripheral interface;
SRAM	Static random access memory;
T_{air}	Air temperature, ºC;
T_{dp}	Dew point temperature, ºC;
T_r	Rectal temperature, ºC;
T_{wb}	Wet bulb temperature, ºC;
THVI	Temperature-humidity-velocity index;
TX	Transmission;
USART	Universal synchronous asynchronous receiver transmitter;
V	Air velocity, m s^{-1};
VCC	Collector common voltage;
VDD	Voltage drain drain, it is the positive pole;
V_i	Air velocity after turn on ventilation stage i – i equal to 1, 2 or 3;
Vo	LCD Contrast adjust;
VSS	Voltage source source, it is the same as ground;
W	Weight vector;
W_1	Weight of the edge connected between air temperature input and a hidden layer neuron or connected between the neuron 1 and the output neuron;

W_2 Weight of the edge connected between air velocity input and a hidden layer neuron or connected between the neuron 2 and the output neuron;

W_3 Weight of the edge connected between relative humidity input and a hidden layer neuron or connected between the neuron 3 and the output neuron;

GENERAL ABSTRACT

LOPES, Alison Zille. **Design of a neuron controller for climatized broiler houses.** 2009. 140p. Dissertation (Master's degree in Systems Engineering – Systems Modeling and Analysis) Federal University of Lavras, Lavras, MG. *

The objective of the present research was to develop a low cost neuron controller for climatized broiler houses. The controller was projected to operate with four control stages, three of ventilation and one of evaporative cooling, as a function of three acquisition points of temperature and relative humidity. The hardware was constructed starting from peripheral interface controllers (PICs), capable of being divided into control module and sensor modules. Artificial neural networks (ANNs) were trained and validated for the prediction of the rectal temperature (T_r) as a function of the thermal conditions (air temperature, T_{air}; relative humidity, RH; and air velocity, V), thus making possible the decision making in the activation of the control stages. The architecture chosen for that purpose was the *multilayer perceptron* (MLP), where 139 data, obtained from the literature, were used in the training (94) and validation (45). The selected MLP with three inputs (T_{air}, RH and V), three neurons in the hidden layer and an output (T_r) presented excellent results, providing estimates with an average percentile error of 0.78% for the training set and 1.02% for the validation set. In the control module, the MLP inputs are obtained starting from a request to the sensor modules. The system developed presented low development cost, stable behavior and absence of random switching under simulated thermal conditions. The neuron controller showed as consistent, seeking comfortable conditions adequate for production and exhibiting the potential of artificial neural network applications to the solution of real problems related to poultry production.

* Guidance Committee: Prof. DSc. Tadayuki Yanagi Junior – UFLA (Adviser), Prof. DSc. Wilian Soares Lacerda – UFLA e Prof. DSc. Giovanni Francisco Rabelo – UFLA

RESUMO GERAL

LOPES, Alison Zille. **Desenvolvimento de um neuro-controlador para galpões climatizados de frangos de corte.** 2009. 140p. Dissertação (Mestrado em Engenharia de Sistema – Modelagem e Análise de sitemas) Universidade Federal de Lavras, Lavras, MG. *

Objetivou-se com a presente pesquisa desenvolver um neuro-controlador de baixo custo para galpões climatizados de frangos de corte. O controlador foi projetado para operar com quatro estágios de controle, três de ventilação e um de resfriamento evaporativo, em função de três pontos de aquisição de temperatura e umidade relativa. O hardware foi construído a partir de *peripheral interface controllers* (PICs), podendo ser dividido em módulo de controle e módulos sensores. Redes neurais artificiais (RNAs) foram treinadas e validadas para a predição da temperatura retal (T_r) em função das condições térmicas (temperatura do ar, T_{ar}; umidade relativa, UR; e velocidade do ar, V), possibilitando assim, a tomada de decisão no acionamento dos estágios de controle. A arquitetura escolhida para esse propósito foi a *multilayer perceptron* (MLP), onde 139 dados, obtidos da literatura, foram usados no treinamento (94) e validação (45). A MLP selecionada com três entradas (T_{ar}, UR e V), três neurônios na camada escondida e uma saída (T_r); apresentou excelentes resultados, proporcionando estimativas com erro percentual médio de 0,78 % para o conjunto de treinamento e 1,02 % para o conjunto de validação. No módulo de controle, as entradas da MLP são obtidas a partir de requisição aos módulos sensores. O sistema desenvolvido apresentou baixo custo de desenvolvimento, comportamento estável e ausência de chaveamentos aleatórios perante condições térmicas simuladas. O neuro-controlador mostrou-se consistente, buscando por condições de conforto adequadas a produção e exibindo a potencialidade da aplicação de redes neurais artificiais na solução de problemas reais relacionados à produção avícola.

* Comitê Orientador: Prof. Dr. Tadayuki Yanagi Junior – UFLA (Orientador), Prof. Dr. Wilian Soares Lacerda – UFLA e Prof. Dr. Giovanni Francisco Rabelo – UFLA

CHAPTER 1

Initial considerations

1 Introdution

Broiler production, having exceptional performance in Brazilian agribusiness, has been using in recent years, new technologies seeking to increase its competitiveness in the market. With a growing internal and external demand, Brazil is today the third largest producer of poultry meat, the largest exporter (Neves, 2008) and it approaches to Europe and United States in the area of production technology (Daniel, 2006). The search for productivity increases has become accentuated in the country with the opening of new markets, and today it is involved in many aspects, from genetic improvement programs to the construction of high technology poultry houses, besides more efficient sanitary control, among others.

Based on the fact that the meteorological factors can favor or harm the animal behavior, facilitating or inhibiting the productive and reproductive processes, currently environmental management has been thoroughly diffused. Most of the national farms use some type of climatization system with a differentiated technological level, which maintain the temperature and relative humidity inside the poultry house within intervals more appropriate for an adequate production. As a result, control becomes necessary for reduction of thermal environment variation to the which the birds are subjected and to prevent potential catastrophes due to extreme conditions. The thermal environment inside of a broiler house can be controlled manually or electronically. In both systems, the essential component is the sensor, which

captures the environment variables at certain time intervals. The electronic control systems have the advantage of quickly responding to changes, saving time and labor, and being able to be adjusted according to climate condition changes (Sukati, 2004).

A control system has the function of provide responses to a particular input in agreement with its transfer function. For many systems this function presents high complexity which hinders the control through conventional procedures. In these cases, the artificial neural networks can be used with success, due to their universal mapping and learning capacity characteristics (Hunt & Sbarbaro 1991; Fukuda & Shibata, 1992).

The artificial neural networks can be embedded in microcontrollers (Corrêa et al., 1999), thus, the use of microcomputers can be eliminated, making a more compact, cheap and energetically economical electric system, besides guaranteeing higher portability.

2 Interaction Between Thermal Environment and Broiler Comfort

Basically, broilers need appropriate feeding, protection against climatic variability and an aseptic environment. It is impossible to simply prioritize one of those three elements, because all are critical factors in survival and productivity maintenance. However, the reduction of the climatic effects on the animals involves a higher number of variables, and it is the area in which broiler production has more control opportunities providing higher habitability and performance (Donald et al., 2001).

The characterization of the thermal animal environment involves the effects of temperature, humidity, radiation and wind; and it can be done through a single variable called effective temperature. For a certain range of effective environmental temperature, the animal maintains its body temperature approximately constant, with minimum thermoregulator mechanism effort. It is

what is called the area of thermal comfort or thermoneutrality, in which there is no sensation of heat or cold and the performance of the animal in any activity is optimized (Baêta & Souza, 1997).

Under thermal stress conditions, the birds try to compensate their reduced ability to dissipate latent heat activating the responsible physiologic processes for the dissipation of heat to the external environment (Moura, 1998). The birds adopt a differentiated behavior, spreading their wings and maintaining them away from the body, increasing their body surface area, facilitating heat loss by convection. When the air temperature approaches to bird's core temperature, around 41 °C, the efficiency of the sensible heat exchange decreases. At that point, the latent heat loss driven for water evaporation process by the respiratory tract becomes more important. As higher the air vapor pressure deficit as easier is the latent heat loss. The increase of the breathing movements is only efficient when the environmental relative humidity is at levels relatively lower than 70%. When the evaporative exchanges are no longer effective for heat loss, the birds enter in heat prostration, possibly leading to death (Moura, 2001).

The main broiler characteristic used by researchers in the determination of the termoneutrality zone is age. In the termoneutral zone, the body temperature of adult chickens is maintained between 41.2 °C and 42.2 °C (Tao & Xin, 2003). According to Medeiros et al. (2005), in general, the thermal comfort range for adult broilers, raised in conventional houses is from 18 to 28 °C, with relative humidity varying from 50 to 70% and air speed around 1.0 to 2.5 m s^{-1}. However, Medeiros (2001) relates that the maximum productivity of broilers subjected to the Brazilian environmental conditions are obtained when the air temperature is in the range from 21 to 27 °C, with 50 to 70% relative humidity and air velocity from 0.5 to 1.5 m s^{-1}. In a general way, adult birds, with five weeks of age, tolerate temperatures above 27 °C, without problems

with the relative humidity level to which they are being submitted. However, environmental temperatures higher than 32 °C and relative humidities higher than 75% are severely stressing to the birds (Moura, 2001).

3 Thermal Environment Control

Many of the problems found in confinement production are caused by the difficulty of maintaining the thermal environment stable and within the adequate comfort levels for each breed. Within that context, the need for thermal environment control emerges, in order to minimize the impact of meteorological conditions on the animals.

According to Sukati (2004), there are many parameters which need to be monitored in a poultry house, such as temperature, relative humidity and air movement, being critical parameters and relevant for automatic control of other parameters.

The control of the thermal environment can be done basically in two ways, manual or automatic. Currently, the manual control still survives in spite of the need for intensive work, frequent visits to the houses and inherent inefficiency in the lack of knowledge of the duration of the current or extreme conditions (Czarick & Lacy, 1994). Automated climatization systems are capable of controlling multiple variables in real time (Czarick & Lacy, 1994), although most of the studies and available systems are only based on temperature (Mitchell & Hamrita, 1999), being able to present adaptive characteristics according to the project of the decision making system.

An automated climatization system is developed with the objective of maintaining certain environmental parameter values within a group of pre-established values. Sensors, hardware and software are the basis of such systems. The sensors are responsible for acquiring and quantifing the current state of the parameters to be evaluated (Zazueta et al., 1991). The hardware is

composed by microcomputers or microcontrollers installed in the houses, or for personal computers, moved away from the poultry houses, connected to the rest of the system through wires, making the monitoring of different poultry houses possible simultaneously (Sukati, 2004). The software is the responsible component for the decision making in a control system. The information captured by the sensor is transmitted to the software, that compares it with reference values and, from then, makes decisions as to which actions the controller needs to put in practice (Sukati, 2004). Nowadays, the most modern poultry facilities present climatization systems whose decision making is done through heuristic methods.

4 General Objectives

In present work, the objective was to develop and to validate an artificial neural network, *multilayer perceptron*, for the estimate of the rectal temperature of broilers as a function of thermal environment variables (Chapter 2); and, using it as support in decision making, to develop a low cost embedded system for monitoring and control of the thermal environment inside of climatized broiler houses, based on microcontrollers (Chapter 3).

5 References

BAÊTA, F. C.; SOUZA, C. F. **Ambiência em edificações rurais:** conforto animal. Viçosa: UFV, 1997. 246 p.

CORRÊA, G. R.; TEIXEIRA, E. P.; FARIA, E. B. Implementação em *hardware* de um controlador neural para robô. In: CONGRESSO BRASILEIRO DE REDES NEURAIS, 4., 1999, São José dos Campos. **Anais...** São José dos Campos: ITA, 1999. p. 350-355.

CZARICK, M.; LACY, M. Environmental controllers. **University of Georgia**, Athens, v. 6, n. 11, Nov. 1994. Disponível em:

<http://www.engr.uga.edu/service/extension/ventilation/tips/1994/v6n11.html>. Acesso em: 18 jul. 2007.

DANIEL, I. **A corrida pelo frango perfeito.** São Paulo: Agência de Notícias Brasil-Árabe, 2006. Disponível em: <http://www.anba.com.br/especial.php?id=298>. Acesso em: 17 jul. 2007.

DONALD, J.; ECKMAN, M.; SIMPSON, G. The impact of environmental management on broiler performance. **The Alabama Poultry Engineering and Economics**, Auburn, n. 10, mar. 2001. Disponível em: <http://www.aces.edu/poultryventilation/>. Acesso em: 22 jul. 2007.

FUKUDA, T.; SHIBATA, T. Theory and applications of neural networks for industrial control systems. **IEEE Transactions on Industrial Electronics**, Nagoya, v. 39, n. 6, p. 472-489, Dec. 1992.

HUNT, K.J.; SBARBARO, D. Neural networks for nonlinear internal model control. **IEE PROCEEDINGS-D**, Glasgow, v. 138, n. 5, p. 431-438, Sept. 1991.

MEDEIROS, C. M. **Ajuste de modelos e determinação de índice térmico ambiental de produtividade para frangos de corte.** 2001. 115 p. Tese (Doutorado em Engenharia Agrícola) – Universidade Federal de Viçosa, Viçosa, MG.

MEDEIROS, C. M.; BAÊTA, F. C.; OLIVEIRA, F. M.; TINÔCO, I. F. F.; ALBINO, L. F. T.; CECON, P. R. Efeitos da temperatura, umidade relativa e velocidade do ar em frangos de corte. **Engenharia na Agricultura**, Viçosa, v. 13, n. 4, p. 277-286, out./dez. 2005.

MITCHELL, B.; HAMRITA, T. K. Poultry environment and production control and optimization: A summary of where we are and where we want to go. **Transaction of the ASAE**, Saint Joseph, v. 42, n. 2, p. 479-483. 1999.

MOURA, D. J. de. Ambiência na avicultura de corte. In: SILVA, I. J. O. **Ambiência na produção de aves em clima tropical.** Piracicaba: NUPEA – ESALQ/USP, 2001. v. 2, p. 75-149.

MOURA, D. J. de. **Avaliação da deficiência térmica de instalações avícolas com diferentes orientações, sombreamento e ventilação.** 1998. 204 p. Tese (Doutorado em Engenharia Agrícola) – Universidade Estadual de Campinas, Campinas, SP.

NEVES, T. **Brasil lidera na exportação de frango e é o 3º maior produtor.** 2008. Disponível em: <http://avicultura.com.pt/index.php?option=com_content&task=view&id=587&Itemid=59>. Acesso em 22 dez. 2008.

SUKATI, N. E. **Environmental control systems in poultry houses**. 2004. 27 p. Monograph (BSc degree in Agricultural Engineering) – University of KwaZulu, Pietermaritzburg, KwaZulu-Natal.

TAO, X.; XIN, H. Acute synergistic effects of air temperature, humidity, and velocity on homeostasis of market–size broilers. **Transactions of the ASAE**, Saint Joseph, v. 46, n. 2, p. 491-497, Dec. 2003.

ZAZUETA, F. S.; BUCKLIN, R.; JONES, P. H.; SMAJSTRLA, A. G.. **Basic concept in environmental computer control of agricultural systems**. Gainesvillle: University of Florida, 1991. 10 p. Disponível em: <http://edis.ifas.ufl.edu/BODY_AE003>. Acesso em: 24 jul. 2007.

CHAPTER 2

Predicting rectal temperature of broiler chickens with artificial neural network

1 Abstract

Poultry production, facing modernization and increasing competitiveness, shows itself to be enterprising in the adoption of new technologies which enable increased productivity. Knowing that poultry productivity and rectal temperature (T_r) are affected by environmental conditions, this research was done with the objective of developing and evaluating artificial neural networks (ANNs) for the prediction of T_r as a function of thermal conditions (air temperature, T_{air}; relative humidity, RH; and air velocity, V). The *multilayer perceptron* (MLP) architecture was used, in which 139 data obtained from the literature were used in training (94) and validation (45). Selected MLP presented excellent results, providing estimates with an average percentile error of 0.78% for the training process and 1.02% for the validation process. Thus, ANNs constitute an appropriate and promising methodology to solve problems related to poultry production.

KEYWORDS: poultry production, thermal comfort, artificial intelligence

Predição da temperatura retal de frangos de corte com redes neurais artificiais

2 Resumo

A avicultura de corte, frente à modernização e à crescente competitividade, mostra-se empreendedora na adoção de novas tecnologias que possibilitem o aumento da produtividade. Sendo a produtividade das aves, bem como a temperatura retal (T_r), afetada pelas condições ambientais, objetivou-se com a presente pesquisa desenvolver e avaliar redes neurais artificiais (RNAs) para a predição da T_r em função das condições térmicas (temperatura do ar, T_{ar}; umidade relativa do ar, UR; e velocidade do ar, V). A arquitetura escolhida para esse propósito foi a *multilayer perceptron* (MLP), no qual 139 dados, obtidos da literatura, foram usados no treinamento (94) e validação (45). A MLP selecionada apresentou excelentes resultados, proporcionando estimativas com erro percentual médio de 0,78 % para o conjunto de treinamento e 1,02 % para o conjunto de validação. Dessa forma, as RNAs constituem uma metodologia adequada e promissora para solucionar problemas reais relacionados à produção avícola.

PALAVRAS-CHAVE: produção avícola, conforto térmico, inteligência artificial

3 Introduction

Poultry production, having exceptional performance in Brazilian agribusiness, has employed new technologies to enhance its competitiveness. The opening of new markets accentuated the search for productivity, involving both breeding program as well as the construction of high-tech poultry houses.

To get the most productivity in poultry production, it is essential that the thermal environment presents appropriate comfort levels, allowing broilers to express their maximum genetic potential. When thermal environment conditions inside a poultry house are out of the comfort limits, the environment becomes uncomfortable, requiring some physiological adjustments of the animal body to maintain its homeothermy, be to retain or dissipate heat (Curtis, 1983). As the thermal environment becomes increasingly stressful, the animal body perceives the risk to life and ceases to prioritize production and reproduction, focusing only on their survival (Medeiros et al., 2005).

The main effects of homeothermic animals exposure to heat are changes in the normal standard of rectal temperature (T_r). Thus, rectal temperature can be considered the best isolated criterion to judge heat tolerance (Campos et al., 2004) and it is an important indicator of efficiency in the homeothermy maintenance facing the thermal environment. Rectal temperature can also be used to assess heat stress impacts (Spiers et al., 2004). The average rectal temperature of broiler chickens is around 41.5 °C, ranging from 40.6 to 43.0 °C, and the upper safety limit to maintain their survival is equal to 45 °C (Medeiros, 2001).

Artificial neural networks (ANNs), a technique inspired by the operation and structure of biological neurons, are trained through the execution of standards across the network, enabling to identify relationships between variables without prior knowledge (Roush et al., 1997). Mathematically, the

ANNs are universal approximators that carry out the mapping between two spaces of variables (Hornik et al., 1989).

Currently, ANNs have been applied in several areas of knowledge, and, in general, its use is linked to the search for patterns and techniques of temporal forecasts for making decisions. Moreover, the ANNs are a technique that can be implemented in microcontrollers, producing systems with high performance and low cost systems (Lima et al., 2000; Shen et al., 2005).

Specifically in poultry production, Roush et al. (1997) used models based on ANNs as a noninvasive way to identify broiler susceptibility to ascites. Similarly, ANN performance in detection of pulmonary hypertension syndrome suggests a new selection method for broilers not prone to this disease (Roush & Wideman, 2000; Roush et al., 2001).

Roush et al. (2006), searching for a broiler growth model, verified the great predictive ability of ANNs, observing little or no overestimation of observed values. Additionally, there are still researches related to analysis of feed intake and weight gain (Vandegrift et al., 2003), detection of dirt in eggs (Mertens et al., 2005), incubation process tracking (Piaia, 2005; Salle, 2005), breeding parameter estimates (Salle et al., 2001), broilers production management (Reali, 2004), among others.

In this context, this research aimed at to develop and to evaluate the use of ANNs for estimating broiler chicken rectal temperature, having as input variables air temperature (T_{air}), relative humidity (RH) and air velocity (V).

4 Material and Methods

4.1 Artificial neural network development

ANNs were implemented and evaluated in an IBM-PC compatible computer, using the ANN Toolbox 0.4.2 in the Scilab 4.1.1 environment (Scilab, 2008), both distributed free through the Internet. The *multilayer perceptron*

(MLP) architecture was used with an input layer (3 inputs), a hidden layer (3 neurons) and an output layer (1 neuron). T_{air}, RH and V are the MLP's inputs and T_r is the output (Figure 1).

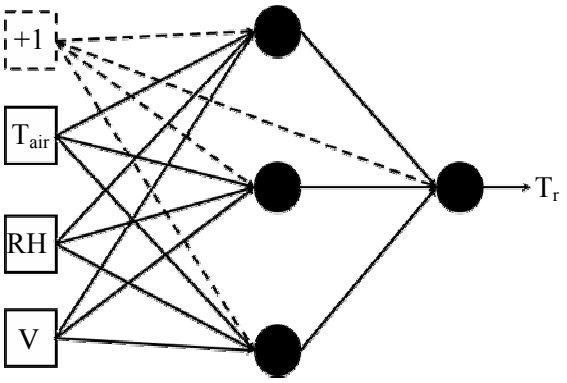


FIGURE 1 *Multilayer perceptron* (MLP) architecture with 3 inputs (air temperature, T_{air}; relative humidity, RH; and air velocity, V), 3 neurons in the hidden layer and an output (rectal temperature, T_r)

4.2 Training

Training data set was composed of 94 observations obtained from several researches (Kettlewell et al., 1997; Yahav et al., 1997; Lacey & Hamrita, 2000; Sevegnani, 2000; Sandercock et al., 2001; Yahav et al., 2001; Tao & Xin, 2003; Lin et al., 2005; Medeiros et al., 2005; Abu-Dieyeh, 2006; Silva et al., 2007), in which broilers, with more than 21 days of age, were subjected to different thermal comfort and discomfort conditions.

In these studies, the thermal environment was characterized by T_{air}, RH, V, and the response variable was the T_r. The variables T_{air}, RH and V ranged from 10 to 41ºC, 20 to 90% and 0 to 4 m s^{-1}, respectively. In 25 data pairs, air velocity was considered equal to 0 m s^{-1} due to the authors not having specified its value. However, in 7 data pairs the existence of a minimum renewed flow was informed; thus, the value of 0.2 m s^{-1} was adopted for V.

Data used in neural network development refer to following broiler breeds: Arbor Acres (Lin et al., 2005), Avian Farm (Sevegnani, 2000; Medeiros et al., 2005), Cobb (Yahav et al., 1997, 2001), ISA Vedette (Abu-Dieyeh, 2006), Ross (Tao & Xin, 2003; Silva et al., 2007) and not specified (Kettlewell et al., 1997; Lacey & Hamrita, 2000; Sandercock et al., 2001). The development of ANNs from data of different broiler breeds is justifiable because this methodology can be applied to thermal environment controllers of poultry houses that can receive various animal breeds.

Significant differences of magnitude can be observed in input variables, making the training process inefficient, since these variations can surmount important trends in data set (Bishop, 1995; Aminian & Aminian, 2000). Thus, it became necessary to pre-process the input data through their standardization in the interval between 0 and 1. For that, T_{air}, RH and V have been divided by 50, 100 and 6, respectively. In addition, the training process is only possible if output values also belong to the range between 0 to 1; therefore, T_r was also divided by 50, similarly T_{air}.

The backpropagation algorithm was used in the training, adopting a learning rate equal to 1 and a momentum factor equal to 0.7. The selection of a high learning rate, despite accelerating the convergence of training process, introduces variations on the reduction of the cost function (Dreyfus, 2005). However, the introduction of the momentum factor minimize this effect (Kasabov, 1996), allowing high-speed convergence without affecting the network performance during the solution process. The mean square error (MSE) cost function was adopted. The weights were initiated randomly with values in the [−1, 1] interval, being the bias equal to 0.

Knowing that the random initial weights interfere in the final training result, five training instances were created aimed to select the one that had best performance. In each instance, the examples were executed 100,000 epochs

through the MLP, computing the weights with smallest MSE during the training process and saving them for later estimation. Thus, at the end of the training process, the acquisition of five distinct ANNs was possible.

4.3 Validation

The validation process was accomplished using 45 observations obtained from diverse studies published in the literature that evaluated broiler chickens with age over 21 days (Furlan et al., 2000; Medeiros, 2001; Borges et al., 2004; Souza et al., 2005; Toyomizu et al., 2005; Al-Fataftah & Abu-Dieyeh, 2007).

Data used included several broiler breeds – Arbor Acres (Toyomizu et al., 2005), Avian Farm (Medeiros, 2001), Cobb (Borges et al., 2004; Toyomizu et al., 2005), Hubbard (Souza et al., 2005), ISA Vedette (Al-Fataftah & Abu-Dieyeh, 2007) and not specified (Furlan et al., 2000) – were submitted to different thermal environments.

Similar to the training data set, the validation data set was made up of the input variables T_{air}, RH, V, and by the response variable T_r. The variables T_{air}, RH and V ranged from 16 to 40ºC, 20 to 90 % and 0 to 3 m s^{-1}, respectively. Air velocity was not specified in 15 data pairs, considering, in these cases, V equal to 0 m s^{-1}. In 7 data pairs, despite omission of information, it was known that there was a minimum air flow, adopting V equal to 0.2 m s^{-1}.

4.4 Comparison with mathematical models

In order to perform a comparative analysis with models proposed in the literature, MLP was confronted with equations adjusted by Medeiros (2001) and Tao & Xin (2003). Medeiros (2001), evaluating Avian Farm male broilers chickens (22 to 42 days-old) fit a surface response model (equation 1) to estimate T_r as a function of T_{air}, RH and V.

$$T_r = 46.102818 - 0.425395\ T_{air} - 0.031012\ RH + 0.118907\ V + 0.009092\ T_{air}^2 + 0.00013\ RH^2 + 0.0263\ V^2 + 0.000893\ T_{air}\ RH - 0.006944\ T_{air}\ V - 0.000661\ RH\ V \quad (1)$$

Equation 1 was fit for the intervals of 16 to 36 ºC, 20 to 90% and 0 to 3 m s^{-1} for T_{air}, RH and V, respectively.

Studying the effect of severe heat stress on broiler chickens, Tao & Xin (2003) developed a temperature-humidity-velocity index (THVI) (equation 2). Based on THVI, equation 3 was developed to estimate the body temperature increase after 90-min exposure to the thermal conditions (ΔT_{90}). Equation 2 and equation 3 were developed through the combination of three T_{air} (35, 38 and 41 ºC) values, two dew point temperature (T_{dp}) (19.4 and 26.1 ºC) values and three V values (0.2, 0.6 and 1.2 m s^{-1}). These equations are only valid for the interval between lower and upper limits of each considered variable. The wet bulb temperature (T_{wb}), used in equation 2, is calculated as a function of T_{air} and T_{dp} or RH through the methodology proposed by Wilhelm (1976).

$$ITUV = (0.85\ T_{air} + 0.15\ T_{wb})\ V^{-0.058} \quad (2)$$

$$\Delta T_{90} = 0.39\ THVI - 12.22 \quad (3)$$

Tao & Xin (2003) did not propose a model for the direct estimate of T_r. So, ΔT_{90} calculated by equation 3 was increased by 41.7 (midpoint of thermoneutral zone considered in the same study) in order to obtain the desired T_r value.

Finally, based on the training and validation data sets, T_r values estimated by MLP were confronted with those calculated by models proposed by Medeiros (2001) and Tao & Xin (2003). Comparative analysis through the mean

square error (MSE), determination coefficient (R^2), mean percent error with its standard deviation, and maximum percent error was done.

5 Results and Discussion

The number of inputs and outputs of an artificial neural network is a characteristic of each studied problem, while the number of hidden layers and their neurons is dependent on the complexity, constituting a characteristic of the project (Bishop, 1995). A hidden layer with 3 neurons was sufficient for an efficient modeling of the problem. A superior number of neurons improves the mapping capacity of neural networks. However, it is necessary to be aware of the possibility of incurring in *overfitting*. Besides, considering the computational cost, the increase of neuron numbers without *overfitting* does not reduce the error significantly.

Table 1 shows the results of the different training instances that originated five distinct ANNs. The chosen model was the ANN 4 (in bold) because it presented the lowest mean square error (MSE) and the highest coefficient of determination (R^2).

TABLE 1 Information of artificial neural networks (ANNs) adjusted to estimate rectal temperature

ANN	Iterations	MSE	R^2
1	99,999	0.4315	0.8793
2	99,904	0.4584	0.8638
3	99,704	0.4559	0.8652
4	**99,986**	**0.4193**	**0.8860**
5	99,925	0.4470	0.87047

The learning rate and the momentum factor were selected according to the recommendation that these parameters should be comprised in the [0, 1] interval (Hu & Hwang, 2002) and that the momentum factor value is usually between 0.5 and 0.9 (Salehfar & Benson, 1998). The learning rate and the momentum factor are the result of an empirical trial and error process, due to their values depending on the studied problem.

Knowing that the learning rate usually varies during the training process (Bishop, 1995; Rao, 1995; Hu & Hwang, 2002), it was tried to adjust it automatically through the *bold driver* technique (Bishop, 1995). However, the process showed to be quite slow and it did not reduce the error when applied to ANN 4.

Despite the variations, the training process presented fast convergence without affecting the solution process. Figure 2 illustrates the training graph of ANN 4, which spent 99,986 epochs to reach a training error (MSE) of 0.4193.

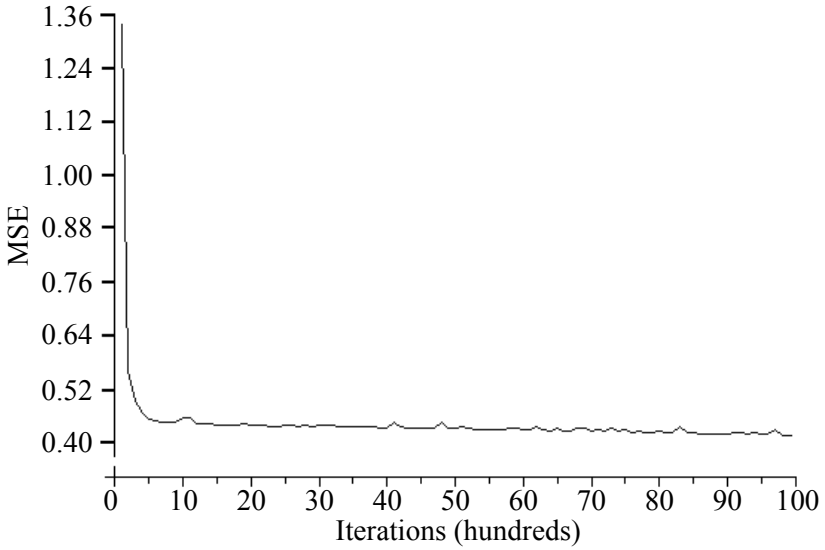


FIGURE 2 Evolution of mean square error (MSE) during the training process of the artificial neural network (ANN) 4

The final weights of the fourth training instance are shown in Table 2, being possible to observe the bias and the weight vector represented by W. In the hidden layer, W_1, W_2 and W_3 are the weights of the edges connected to T_{air}, V and RH inputs, respectively. In the output neuron, W_1, W_2 and W_3 are the weights of the edges that leave the neurons 1, 2 and 3, respectively.

TABLE 2 Weights resulting from the artificial neural network training process

neurons	bias	W_1	W_2	W_3
1	-0.5827986	1.4174650	-3.8484132	-6.5232177
2	1.7943517	1.1100434	-4.0314207	-7.6660550
3	-12.1465030	11.9241280	- 0.3432754	1.1697885
output	1.5810874	5.2696701	- 1.4707425	6.3003672

Figure 3 shows the functional relation among observed values (training data – Figure 3A – and validation data – Figure 3B) and data estimated by selected MLP. The points are very close to tendency line, suggesting that ANN 4

is able to represent the observed data efficiently, which is also confirmed through the high value of the coefficient of determination (R^2).

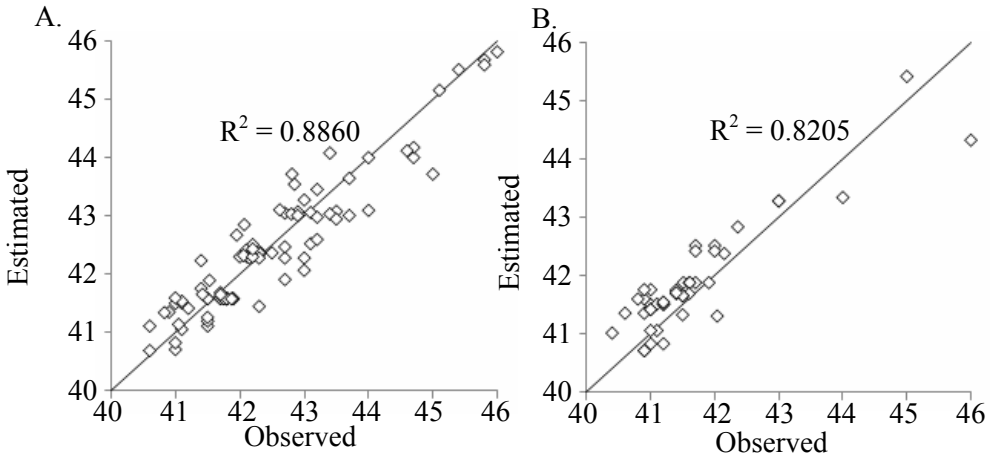


FIGURE 3 Functional relation among observed and estimated (*Multilayer perceptron*) rectal temperature for training (A) and validation data (B).

The neural network training adjustment presented mean percent error equal to 0.78±0.6%, observing a maximum error of 2.81%. Validation process got excellent results: R^2 of 0.8205 and MSE of 0.5179. Mean percent error of validation process was equal to 1.02±0.68%, verifying a maximum value of 3.61%.

Figure 4A exhibits T_r values estimated by neural network as a function of T_{air} ranging from 10 to 40 ºC, RH ranging from 20 to 90% and V equal to 1.5 m s^{-1}. Likewise, Figure 4B shows the profile of the rectal temperature estimate, fixing RH at 55% and varying T_{air} between 10 and 40 ºC and V between 0 and 4 m s^{-1}. Figure 4C shows T_r values estimated in function of RH ranging from 20 to 90%, V ranging from 0 to 4 m s^{-1} and T_{air} equal to 26 ºC. The profiles observed in Figure 4 are according to the expected physiological responses of broiler

chickens, and they agree with results observed in other studies, such as Medeiros (2001) and Tao & Xin (2003).

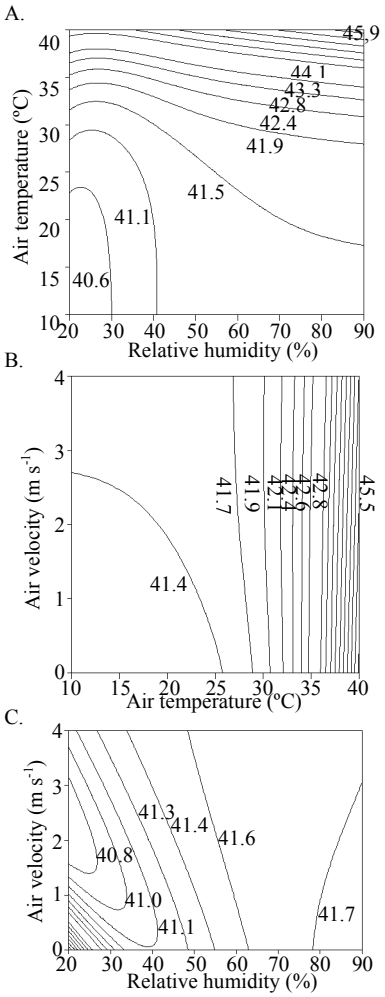


FIGURE 4 Rectal temperature estimated (ºC) as a function of the variation in air temperature and relative humidity with air velocity equal to 1.5 m s^{-1} (A), as a function of the variation in air temperature and air velocity with relative humidity equal to 55% (B) and as a function of the variation of relative humidity and air velocity with air temperature equal to 26 ºC (C).

Table 3 shows the comparative analysis between MLP and the method proposed by Medeiros (2001) (equation 1). The comparisons were accomplished based on data of training and validation, extrapolating as well as respecting the limits of the model proposed by Medeiros (2001). Comparisons using just the average data presented in Medeiros (2001) were also done.

TABLE 3 Comparative analysis between the model developed by Medeiros (2001) and the artificial neural network (ANN)

		MSE	R^2	E_{mean} (%)	E_{max} (%)
Training	ANN*	0.4193	0.8860	0.78 ± 0.60	2.81
	equation 1*	1.5241	0.7885	3.32 ± 1.23	7.30
	ANN	0.4635	0.7519	0.88 ± 0.64	2.81
	equation 1	1.4228	0.6630	3.11 ± 1.23	7.30
Validation	ANN*	0.5179	0.8205	1.02 ± 0.68	3.61
	equation 1*	1.1416	0.8168	2.45 ± 1.10	7.35
	ANN	0.5196	0.7754	1.02 ± 0.69	3.61
	equation 1	1.1216	0.7641	2.41 ± 1.09	7.35
Means (Medeiros, 2001)	ANN	0.4406	0.7679	0.91 ± 0.56	2.14
	equation 1	0.9472	0.8675	2.23 ± 0.51	3.03

E_{mean} and E_{max} are, respectively, percentages of mean and maximum error.
* indicates the extrapolation of the considered interval in equation 1.

The artificial neural network presented better results than equation 1, besides being a model of more wide scope. Even for the average values used for the equation 1 adjustment, ANN provided better results. In spite of coefficient of determination value being superior for equation 1 (0.8675 against 0.7679), MLP presented smaller MSE values, mean and maximum percent errors.

Table 4 shows the comparative analysis between neural networks and the mathematical model developed by Tao & Xin (2003). So, ΔT_{90} (equation 3) was increased by 41.7 ºC, as discussed previously. It is important to observe that there is an application limitation of this model when V is equal to 0 m s^{-1}, due to the occurrence of a division by zero in the calculation of THVI (equation 2).

Thus, the comparisons were accomplished based on both training and validation data, excluding the data pairs in which V is equal to 0 m s^{-1}. Besides these, there are comparasions using, from the total data, only those which belong within the scope of equation 3 and only present in the work done by Tao & Xin (2003).

TABLE 4 Comparative analysis between the model developed by Tao & Xin (2003) and the artificial neural network (ANN)

		MSE	R^2	E_{main} (%)	E_{max} (%)
Training	ANN	0.3534	0.9164	0.66 ± 0.50	2.01
	Equation 3	2.2788	0.7416	4.25 ± 3.47	9.54
Validation	ANN	0.4822	0.8449	1.02 ± 0.58	2.14
	Equation 3	2.6799	0.6317	5.12 ± 4.11	13.70
Scope	ANN	0.4090	0.8930	0.75 ± 0.57	2.01
	Equation 3	0.6993	0.6950	1.16 ± 1.15	4.48
Means (Tao & Xin, 2003)	ANN	0.3895	0.8989	0.71 ± 0.55	2.01
	Equation 3	0.5063	0.8236	0.89 ± 0.71	2.55

E_{mean} and E_{max} are, respectively, percentages of mean and maximum error.

Again, ANN furnished better results than the mathematical model, in spite of being a more encompassing model, even for the average of the data used in the fitting of Tao & Xin (2003) model.

6 Conclusion

Artificial neural networks adjusted for prediction of broiler chickens rectal temperature as a function of air temperature, relative humidity and air velocity were trained and validated, presenting statistical indexes adequate to applications related to poultry production.

7 Acknowledgements

To CAPES and FAPEMIG for the financial support.

8 References

ABU-DIEYEH, Z. M. M. Effect of chronic heat stress and long-term feed restriction on broiler performance. **International Journal of Poultry Science**, Lasani, v. 5, n. 2, p. 185-190, Feb. 2006.

AL-FATAFTAH, A. R. A.; ABU-DIEYEH, Z. M. M. Effect of chronic heat stress on broiler performance in Jordan. **International Journal of Poultry Science**, Lasani, v. 6, n. 1, p. 64-70, Jan. 2007.

AMINIAN, M.; AMINIAN, F. Neural-network based analog circuit fault diagnosis using wavelet transform as preprocessor. **IEEE Transactions on Circuits and Systems II**, Piscataway, v. 47, n. 2, p. 185-190, Feb. 2000.

BISHOP, C. M. **Neural networks for pattern recognition**. Oxford: Clarendon, 1995. 504 p.

BORGES, S. A.; SILVA, A. V. F. da.; MAJORKA, A.; HOOGE, D. M.; CUMMINGS, K. R. Physiological responses of broiler chickens to heat stress and dietary electrolyte balance (sodium plus potassium minus chloride, milliequivalents per kilogram). **Poultry Science**, Champaign, v. 83, n. 9, p. 1551-1558, Sept. 2004.

CAMPOS, O. F. de.; CUNHA, D. N. F. V. da.; PEREIRA, J. C.; JUNQUEIRA, M. M.; MARTUSCELLO, J. A.; PIRES, M. F. A.; LIZIEIRE, R. S. Utilização de diferentes abrigos para bezerros de rebanhos leiteiros em condições tropicais durante a época das águas: temperatura retal, freqüência respiratória e consumo de água. In: REUNIÃO ANUAL DA SOCIEDADE BRASILEIRA DE ZOOTECNIA, 41., 2004, Campo Grande. **Anais...** Campo Grande: Sociedade Brasileira de Zootecnia, 2004. p. 1-5.

CURTIS, S. E. **Environmental management in animal agriculture**. Iowa: Iowa State University, 1983. 409 p.

DREYFUS, G. **Neural networks**: methodology and applications. Berlin: Springer, 2005. 508 p.

FURLAN, R. L.; MACARI, M. SECATO, E. R.; GUERREIRO, J. R. Air velocity and exposure time to ventilation affect body surface and rectal temperature of broiler chickens. **Applied Poultry Research**, Savoy, v. 9, n. 1, p. 1-5, Mar./May 2000.

HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. **Neural Network**, Oxford, v. 2, n. 5, p. 359-366, July 1989.

HU, Y. H.; HWANG, J. N. **Handbook of neural network signal processing**. Boca Raton: CRC, 2002. 410 p.

KASABOV, N. K. **Foundations of neural networks, fuzzy systems and knowledge engineering**. Cambridge: MIT, 1996. 568 p.

KETTLEWELL, P. J.; MITCHELL, M. A.; MEEKS, I. R. An implantable radio-telemetry system for remote monitoring of heart rate and deep body temperature in poultry. **Computers and Electronics in Agriculture**, Amsterdam, v. 17, n. 2, p. 161-175, May 1997.

LACEY, B.; HAMRITA, T. K. Monitoring deep body temperature responses of broilers using biotelemetry. **Applied Poultry Research**, Savoy, v. 9, n. 1, p. 6-12, Mar./May 2000.

LIMA, J. C.; MEDEIROS, A.; CANALLI, V.M. ANTUNES, F.; REIS, F. S. dos. A PIC controller for grid connected PV system. In: IEEE INTERNATIONAL POWER ELETRONIC CONGRESS, 7., 2000, Acapulco, **Proceedings…**, Acapulco: CIEP, 2000, p. 307-311.

LIN, H.; ZHANG, H. F.; GU, X. H.; ZHANG, Z. Y.; BUYSE, J.; DECUYPERE, E. Thermoregulation responses of broiler chickens to humidity at different ambient temperatures. II. four weeks of age. **Poultry Science**, Champaign, v. 84, n. 8, p. 1173-1178, Aug. 2005.

MEDEIROS, C. M. **Ajuste de modelos e determinação de índice térmico ambiental de produtividade para frangos de corte**. 2001. 115 p. Tese (Doutorado em Engenharia Agrícola) – Universidade Federal de Viçosa, Viçosa, MG.

MEDEIROS, C. M.; BAÊTA, F. C.; OLIVEIRA, F. M.; TINÔCO, I. F. F.; ALBINO, L. F. T.; CECON, P. R. Efeitos da temperatura, umidade relativa e velocidade do ar em frangos de corte. **Engenharia na Agricultura**, Viçosa, v. 13, n. 4, p. 277–286, out./dez. 2005.

MERTENS, K.; DE KETELAERE, B.; KAMERS, B.; BAMELIS, F. R.; KEMPS, B. J.; VERHOELST, E. M.; DE BAERDEMAEKER, J. G.; DECUYPERE, E. M. Dirt detection on brown eggs by means of color computer vision. **Poultry Science**, Champaign, v. 84, n. 7, p. 1653–1659, June 2005.

PIAIA, J. C. Z. **Aplicação da inteligência artificial no monitoramento do processo de incubação**. 2005. 90 p. Dissertação (Mestrado em Engenharia Química) – Universidade Federal de Santa Catarina, Florianópolis, SC.

RAO, V. B. **C++ Neural Networks and Fuzzy Logic**. 2. ed. New York: M & T Books, 1995. 549 p

REALI, E. H. **Utilização de inteligência artificial (redes neurais artificiais) no gerenciamento da produção de frangos de corte**. 2004. 127 p. Dissertação (Mestrado em Ciências Veterinárias) – Universidade Federal do Rio Grande do Sul, Porto Alegre, RS.

ROUSH, W. B.; CRAVENER, T. L.; KIRBY, Y. K.; WIDEMAN JR., R. F. Probabilistic neural network prediction of ascites in broilers based on minimally invasive physiological factors. **Poultry Science**, Champaign, v. 76, n. 11, p. 1513-1516, June 1997.

ROUSH, W. B.; WIDEMAN JR., R. F. Evaluation of broiler growth velocity and acceleration in relation to pulmonary hypertension syndrome. **Poultry Science**, Champaign, v. 79, n. 2, p. 180-191, Sept. 2000.

ROUSH, W. B.; WIDEMAN JR., R. F.; CAHANER, A.; DEEB, N.; CRAVENER, T. L. Minimal number of chicken daily growth velocities for artificial neural network detection of pulmonary hypertension syndrome (PHS). **Poultry Science**, Champaign, v. 80, n. 3, p. 254-259, Mar. 2001.

ROUSH, W. B.; DOZIER III, W. A.; BRANTON, S. L. Comparison of Gompertz and neural network models of broiler growth. **Poultry Science**, Champaign, v. 85, n. 4, p. 794-797, Nov. 2006.

SALEHFAR, H.; BENSON, S. A. Electric utility coal quality analysis using artificial neural network techniques. **Neurocomputing**, Savoy, v. 23, n. 1/3, p. 195-206, Dec. 1998

SALLE, C. T. P.; GUAHYBA, A. S.; WALD, V. B.; SILVA, A. B.; SALLE, F. O.; FALLAVENA, L. C. B. Uso de redes neurais artificiais para estimar parâmetros de produção de galinhas reprodutoras pesadas em recria. **Revista Brasileira de Ciência Avícola**, Campinas, v. 3, n.3, p. 257-264, set./dez. 2001.

SALLE, F. O. **Utilização de inteligência artificial (redes neurais artificiais) no gerenciamento do incubatório de uma empresa avícola do sul do Brasil**. 2005. 82 p. Dissertação (Mestrado em Ciências Veterinárias) – Universidade Federal do Rio Grande do Sul, Porto Alegre, RS.

SANDERCOCK, D. A.; HUNTER, R. R.; NUTE, G. R.; MITCHELL, M. A. HOCKING, P. M. Acute heat stress-induced alterations in blood acid-base status and skeletal muscle membrane integrity in broiler chickens at two ages: implications for meat quality. **Poultry Science**, Champaign, v. 80, n. 3, p. 418-425, Mar. 2001.

SCILAB. INRIA. 1989-2008. **Apresenta informações sobre o SCILAB, plataforma para computação numérica**. Disponível em: <http://www.scilab.org>. Acessado em: 17 jun. 2008.

SEVEGNANI, K. B. **Avaliação dos efeitos fisiológicos causados pela ventilação artificial em frangos de corte, em dispositivos de simulação climática**. 2000. 106 p. Tese (Doutorado em Engenharia Agrícola) – Universidade Estadual de Campinas, Campinas, SP.

SHEN, W. X.; CHAU, K. T.; CHAN, C. C.; LO, E. W. C. Neural network-based residual capacity indicator for nickel-metal hydride batteries in electric vehicles. **IEEE Transactions on Vehicular Technology**, Hong Kong, v. 54, n. 5, p. 1705-1712, Sept. 2005.

SILVA, M. A. N. da.; BARBOSA FILHO, J. A. D.; SILVA, C. J. M. da.; ROSÁRIO, M. F. do.; SILVA, I. J. O. da.; COELHO, A. A. D.; SAVINO, V. J. M. Avaliação do estresse térmico em condição simulada de transporte de frangos de corte. **Revista Brasileira de Zootecnia**, Viçosa, v. 36, n. 4, p. 1126-1130, jul./ago. 2007.

SOUZA, B. B. de.; BERTECHINI, A. G.; TEIXEIRA, A. S.; LIMA, J. A. F.; CONTE, A. J. Efeito do nível energético e da suplementação com cloretos de potássio e de amônia na dieta sobre as respostas fisiológicas e o desempenho de frangos de corte no verão. **Ciências Agrotécnica**, Lavras, v. 29, n. 1, p. 185-192, jan./fev. 2005.

SPIERS, D. E.; SPAIN, J. N.; SAMPSON, J. D.; RHOADS, R. P. Use of physiological parameters to predict milk yield and feed intake in heat-stressedd airy cows. **Jounal of Thermal Biology**, Pergamon, v. 29, n. 7-8, p. 759–764, Oct./Dec. 2004.

TAO, X.; XIN, H. Acute synergistic effects of air temperature, humidity, and velocity on homeostasis of market–size broilers. **Transactions of the ASAE**, Saint Joseph, v. 46, n. 2, p. 491-497, Dec. 2003.

TOYOMIZU, M.; TOKUDA, M.; MUJAHID, A.; AKIBA, Y. Progressive alteration to core temperature, respiration and blood acid-base balance in broiler chickens exposed to acute heat stress. **The Journal of Poultry Science**, Saint Joseph, v. 42, n. 2, p. 110-118, Nov. 2005.

VANDEGRIFT, K. J.; CRAVENER, T. L.; HULET, R. M.; ROUSH, W. B. Analysis of the nonlinear dynamics of daily broiler growth and feed intake. **Poultry Science**, Champaign, v. 82, n. 4, p. 1091-1099, Feb. 2003.

WILHELM, L. R. Numerical calculation of psychrometric properties in SI units, **Transactions of the ASAE**, Saint Joseph, v. 19, n. 2, p. 318-325, Apr. 1976.

YAHAV, S.; STRASCHNOW, A.; PLAVNIK, I.; HURWITZ, S. Blood system response of chickens to changes in environmental temperature. **Poultry Science**, Champaign, v. 76, n. 4, p. 627-633, Apr. 1997.

YAHAV, S.; STRASCHNOW, A.; VAX, E.; RAZPAKOVSKI, V.; SHINDER, D. Air velocity alters broiler performance under harsh environmental conditions. **Poultry Science**, Champaign, v. 80, n. 6, p. 724-726, June 2001.

CHAPTER 3

Design of a neuron controller for climatized broiler houses

1 Abstract

The objective of the present research was to develop an environmental controller to be applied in climatized poultry houses based on a *multilayer perceptron* (MLP) artificial neural network and peripheral interface controllers (PICs). The MLP, responsible for the prediction of the rectal temperature (T_r) as a function of the thermal conditions (air temperature, T_{air}; relative humidity, RH; and air velocity, V), was used in the decision making, where the ventilation and cooling systems activation depend on the estimated T_r value. The hardware, divided in control and sensor modules, was developed to operate with four control stages (three of ventilation and one of evaporative cooling) and up to three acquisition points of T_{air} and RH. The neuron controller developed was evaluated under laboratory conditions. The PICs made possible the construction of a low cost robust system. The neuron controller being capable of translating thermal environment variables into comfort conditions, presenting suitable behavior in the search for adequate thermal conditions for poultry production.

KEYWORDS: embedded system, artificial neural network, environment control

Desenvolvimento de um neuro-controlador para galpões climatizados de frangos de corte

2 Resumo

Objetivou-se com a presente pesquisa, desenvolver um neuro-controlador para o controle do ambiente térmico em galpões avícolas climatizados com base em uma rede neural artificial *multilayer perceptron* (MLP) e *peripheral interface controllers* (PICs). A MLP, responsável pela predição da temperatura retal (T_r) em função das condições térmicas (temperatura do ar, T_{ar}; umidade relativa, UR; e velocidade do ar, V), foi utilizada na tomada de decisão, onde o acionamento dos sistemas de ventilação e resfriamento depende do valor estimado de T_r. O hardware, dividido em módulo de controle e módulos sensores, foi desenvolvido para operar com quatro estágios de controle (três de ventilação e um de resfriamento evaporativo) e até três pontos de aquisição de T_{ar} e UR. O neuro-controlador desenvolvido foi avaliado sob condições laboratoriais. Os PICs possibilitaram a construção de um sistema robusto e de baixo custo. Sendo o neuro-controlador capaz de traduzir variáveis do ambiente térmico em condições de conforto, apresentando comportamento condizente a busca por condições térmicas adequadas a produção avícola.

PALAVRAS-CHAVE: sistema embarcado, rede neural artificial, controle do ambiente

3 Introduction

Significant losses in world poultry production during summer time (Furlan, 2000) and the fast growth of aviculture in hot climate countries (Abu-Dieyeh, 2006), denote the importance of methods capable of relieving or reducing the caloric stress. In that sense, climatization stands out as a strategic solution, where ventilation and cooling systems promote adequate thermal environment inside of facilities (Baeta & Souza, 1997; Bueno, 2004).

However, the control of a climatization system is a complex process, since its objective is the maintenance of the thermal environment at adequate comfort levels for production (Padilha et al., 2001). Inside the facilities, thermal comfort, the state of animal welfare within the thermal environment, is influenced, mainly, by the air temperature (T_{air}), relative humidity (RH) and air velocity (V) (Yahav et al., 2001; Tao & Xin, 2003). In spite of that, even today, the climatization systems continue only being controlled through T_{air} and, in some cases, RH (Pereira & Nääs, 2005).

The most modern broiler production facilities have climatization system control strategies based on heuristic methods, which makes the use of intelligent systems attractive (Padilha et al., 2001). In that context, having high computacional power and capacity to approach non-linear functions, the artificial neural networks (ANNs), a technique inspired by the operation of the biological neuron (Roush et al., 1997), stand out for their popularization in control activities (Turkoglu, 2007; Kumar et al., 2008) and possibility of implementation in microcontrollers (Chung & Lee, 2008; Bayindir et al., 2009).

In recent years, due to increasing processing power and flexibility, linked to the integration of peripherals, low cost and programming simplicity, the microcontrollers have acquired a position of prominence in the most diverse areas, including those renounly computer based (Frankowiak et al., 2005; Amirante et al., 2008). Furthermore, more and more, researchers in their

association with ANNs, look for the construction of more low cost robust systems. Chung & Lee (2008), for instance, seeking to improve air quality inside vehicles, produced an air quality control system using an ANN embedded in a microcontroller, reducing the error in gas detection. Kumar et al. (2008), intending to control the speed of direct current motors, affirmed that the ANNs are of easy implementation in microcontrollers, producing a more compact code and better response than the conventional methods.

Specifically related to animal thermal comfort, the rectal temperature (T_r) is known to be one of the main physiologic responses for the evaluation of thermal stress (Spiers et al., 2004). Based on the aforementioned, the objective of the present research was to develop a microcontrolled climatization system for broiler raising houses in which the decision making is based on the T_r, estimate via an ANN.

4 The System

4.1 Hardware

The hardware developed for the climatization system control, based on the PIC (peripheral interface controller) family microcontrollers, is basically made up of two parts: control module and sensor modules. A general view of the hardware is presented in block diagram form in Figure 1.

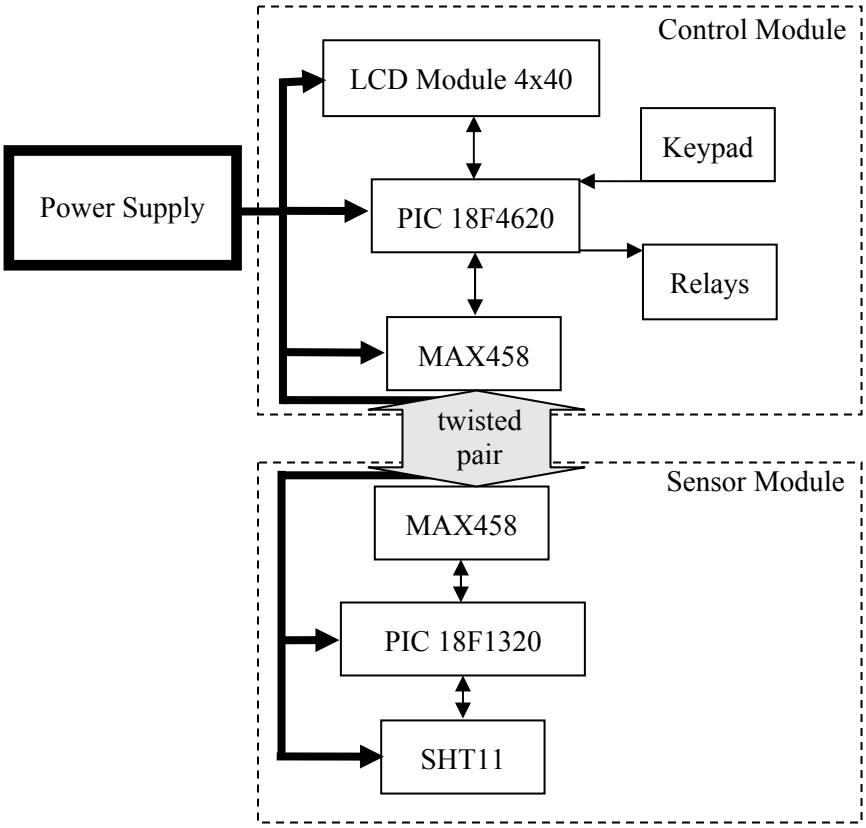


FIGURE 1 Simplified block diagram for the hardware of the climatization control system

The hardware was built starting from the idea of dividing the control in four stages, three of ventilation and one of evaporative cooling. Four relays, controlled by the PIC 18F4620 microcontroller, are responsible for the activation of the different stages. The PIC 18F4620, embedded in the control module, is responsible for the control of the whole system, from equipment activation to sensor module communication.

The relationship between PIC 18F4620 and 18F1320 is a master-slave relationship. The communication between the control module, PIC 18F4620,

and the sensor modules, PIC 18F1320, is done via MAX485 through a twisted pair cable (four pairs). Up to three air temperature (T_{air}) and relative humidity (RH) sensor modules are supported by the hardware. The sensor modules are fed by the same twisted pair cable used for communication with the control module. The voltage and consumption current of the microcontrollers, liquid crystal display (LCD), MAX485 transceivers and SHT11 sensors are obtained from the same power supply (5 Vcc, 700 mA).

The LCD serves for visualization of system options, typed values, state of the system, as well as sensor module responses. Light emitting diodes (LEDs) are used to indicate connected modules and activated stages.

As the air velocity sensors usually are very expensive, the option was the inclusion of a numeric keypad for manual set of V by the user. Thus, an anemometer can be used for measuring V at each ventilation stage and it can further be informed to the system. It is important to observe that the hardware was built to make possible the connection of an air velocity sensor module, as well as a curtain control module, however, the construction of these will be further developed. In Figure 2, the components that compose the hardware can be observed.

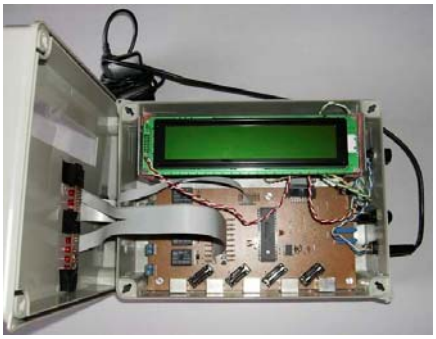
A.  B. 

FIGURE 2 Internal view of the (A) control module and (B) the temperature and humidity sensor module and an example of its printed circuit board with the respective components

4.1.1 PIC 18F4620 and 18F1320 microcontrollers

Microcontrollers are complete computers in a single chip, incorporating all the basic components of a personal computer on a much smaller scale. In functional terms, a microcontroller is a programmable chip used in the control of a process or system (Turkoglu, 2007; Wilmshurst, 2007).

The PICs are integrated circuits based on the complementary metal oxide semiconductor (CMOS) technology (Bayindir et al., 2009). The integration of this technology to the Harvard architecture and the reduced instruction set computer (RISC) imparts low energy consumption, high flexibility and high performance to the PIC microcontrollers (Wilmshurst, 2007).

The PICs stand out for their popularization, having different devices for different applications. The PIC 18F family works with 16 bit instructions, advanced RISC architecture, 32 level instruction stack, multiple internal and external interruptions. It presents the highest performance among the 8 bit microcontrollers produced by Microchip. Besides, it has become popular among new developers and those who want to program in C language (Microchip, 2008a).

In view of this, the microcontrollers PIC 18F4620, control module, and the PIC 18F1320, sensor modules, were selected for the present research. In adition, the size of the program memory justifies the selection of those devices.

The PIC 18F4620 is a 40 pin microcontroller that operates at a clock frequency of up to 40 MHz. It has 64 KB of Flash memory as program memory and 3968 bytes of static random access memory (SRAM) and 1KB of electrically erasable programmable read-only memory (EEPROM) as data memory. It also has 36 input/output (I/O) pins, 13 analog/digital (A/D) channels, 2 Capture\Compare\Pulse width modulation modules (CCPMs), 2 comparators, 4 timers, master synchronous serial port (MSSP) module and universal synchronous asynchronous receiver transmitter (USART) module. The MSSP

module supports the serial peripheral interface (SPI) and the inter-integrated circuit (I^2C) communication standards. The USART module supports RS485, RS232 and LIN/J2602 (Microchip, 2008c).

The PIC 18F1320 has 18 pins, 8 KB Flash memory, 256 bytes of SRAM and EEPROM memory, operating at a frequency of up to 40 MHz. It also has 16 I/O pins, 7 A/D channels, CCPM, 4 timers and USART module that supports RS485, RS232 and LIN 1.2 (Microchip, 2008b).

4.1.2 MAX485 transceiver

The MAX485 transceiver is an integrated circuit, manufactured by Maxim Integrated Products, that implements the RS485 standard (Maxim, 2008). This standard provides a quite robust form of multipoint communication that is being widely used in the industry for systems control and data transfer (Zhenyao et al., 2006).

The MAX485 is a serial half-duplex communication transceiver, where the logical level is determined by the tension difference between the A and B pins (Figure 3). When A is positive and B negative, the logical level is 1; the inverse implies in logical level 0. This manner of operation, known as differential, characterizes the RS485 standard. The main advantage of differential transmission is the robustness to noise and interference. If noise is introduced in the line, it is induced in the two wires in a way that the difference between A and B of that interference tends to be almost null. Thus, the maximum distance can be up to 1200 m, being, however, dependent on the transmission rate. The RE and DE pins, usually connected together, are responsible for the change between transmission and reception. The DI pin is the transmission (TX) input, while RO is the reception (RX) output. The transceiver is powered, VCC (voltage common collector) and GND (ground), by a direct current (DC) 5V power supply, with current consumption between 120 μA and

500 μA (Maxim, 2008). Besides the pinout, Figure 3 exhibits the integration of MAX485 in the hardware.

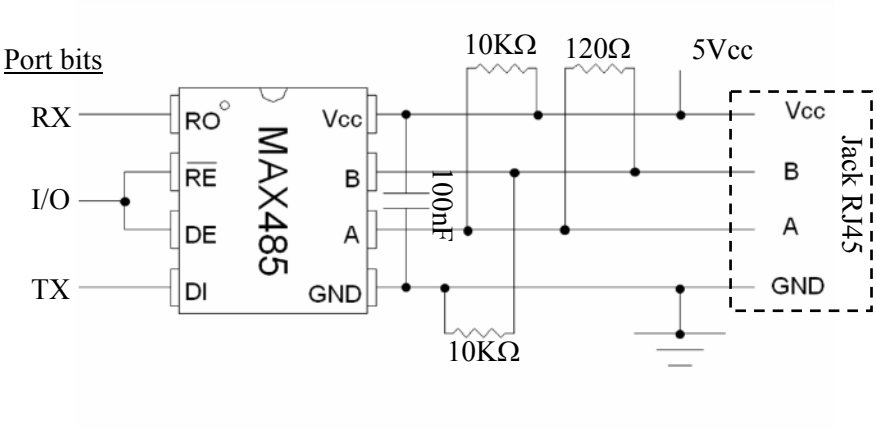


FIGURE 3 MAX485 transceiver: pinout and mounting in the hardware

To minimize interference between the wires connected to A and B, a twisted pair cable should be utilized. In Figure 3, the RJ45 Jack serves the connection of the twisted pair cable. To avoid that the logical level of the data bus (twisted pair) becomes undefined, all the devices in reception mode, a pull-up resistor should be added to pin A and a pull-down to pin B. The reflection is avoided adding resistors of equal value, 120 Ω, between A and B at the end points (Maxim, 2008).

4.1.3 SHT11 sensor

The SHT11, manufactured by Sensirion, is a humidity and temperature sensor module in a single chip with calibrated digital output. The precision is ±3% for RH and ±0.4 ºC for T_{air} (25 ºC). The application of the CMOS industrial process and the CMOSens® technology guarantees high reliability and stability. The device includes a capacitive polymer as a humidity sensor element and a

bandgap PTAT (proportional to absolute temperature) temperature sensor. Both coupled directly to a 14 bit analog to digital converter and a serial interface circuit on the same chip. As a result, superior signal quality, a fast response time and insensitivity to external disturbances can be reach at a quite competitive price (Sensirion, 2007).

Each SHT11 is calibrated in a precision climatic chamber. The calibration coefficients are programmed in the calibration memory of the sensor. Those coefficients are used internally during the measurements to calibrate the signals of the sensor. Furthermore, the sensor presents an internal cyclic redundancy check (CRC) generator, where each transmission is assured by an 8 bit checksum (Sensirion, 2007). Assuring that wrong data can be detected and eliminated.

The serial communication through only two pins and the internal voltage regulator allow easy and fast integration. The SCK (serial clock) serves the synchronization of the communication between SHT11 and microcontroller. DATA is used for transmission in both directions. A pull-up resistor is necessary to maintain DATA at a high logical level. The power supply pins, VCC and GND, can be coupled to a 100 nF rectifier capacitor and require a voltage between 2.4 and 5.5 V. SHT11 is a quite compact device and of low consumption. Figure 4 outlines the form of how SHT11 should be connected to the microcontroller and the power source.

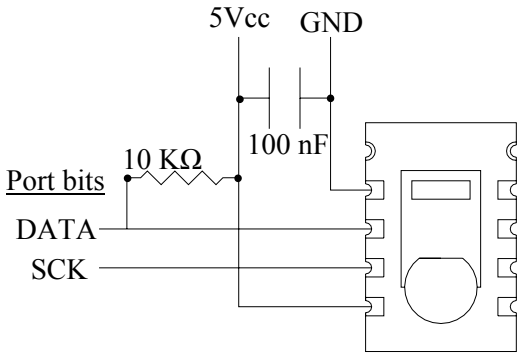


FIGURE 4 Integration between SHT11, power supply and microcontroller

Knowing about the internal conditions of a poultry facility, where dust and ammonia are present in the air, a membrane should be used to protect the sensor. In choosing the membrane, absorbent materials should be avoided, since they would introduce errors in the measured values. Thus, for better sensor response time, the air volume after the membrane should be reduced (Sensirion, 2007). Considering this, a polyester membrane was adapted in a way to minimally influence the sensor response time.

4.1.4 Liquid crystal display (LCD)

The LCD display was one of the technologies that made the current technological revolution possible. Being an essential part in various electronic equipments, such as cellular telephones, laptops, electronic calendars, among others. Now, LCD displays controlled by microcontrollers are widely used due to their low energy consumption and graphic flexibility (Wilmshurst, 2007).

The HD44780 microcontroller, produced by Hitachi, especially developed to control alphanumeric LCD modules, became an informal standard. For possessing a simple interface it is connected easily to general purpose

microprocessors and microcontrollers (Wilmshurst, 2007). The display is a 4x40, four lines by forty columns, alphanumeric LCD module based on the HD44780 standard. The need for exhibition screens with options and to show temperature and humidity simultaneously from up to three different sensor modules justifies the choice of a display of large proportions (4x40).

The LCD module can exhibit up to 160 different characters, which are stored in its character ROM (read only memory). It even presents some functions such as clearing the display, to move cursor, cursor on/off, among others (Bayindir & Ates, 2007). Just a 5V power supply is necessary for its operation. Each character is generated by switching some of dots in a 5x7 matrix. Pin connecting of the LCD is shown concisely in Table 1.

TABLE 1 Pin connecting of the 4X40 LCD module

Pin No.	Symbol	Function
1-8	DB7-DB0	Data line
9	E1	Half-up chip enable signal
10	R/W	Read/write
11	RS	Register select
12	Vo	LCD contrast adjust
13	VSS	Ground
14	VDD	+5V
15	E2	Half-down chip enable signal
16	-	Not connected
17	LED+	Backlight power supply + (5V)
18	LED-	Backlight power supply -

Normally, the LCD modules present eight pins for data, three control pins and three power pins. However, in the 4x40 LCD module four control pins exist. This is because, unlike most of the LCD displays which present a single pin for signal enabling, it presents E1 as enabling signal for the superior half and E2 for the inferior. If the other pins were not in common, half would work as

completely independent modules, where E1 (pin 9) is used to begin transfer of data or characters among the superior module and the data line, and E2 (pin 15) would complete the same task for the inferior module. When the register select (RS), defined as pin 11, is low, the data bytes transferred to the display are behaved as commands and data bytes read from the display indicates its status. By setting line as high, character data or status information is taken from its registers. Pin 10 is a Read/Write (R/W) pin that can be set low in order to write commands or character data to the module, or set high to read character data. The pins from 1 to 8 are used for the data line. Pins 13 and 14 are the power supply lines, where VDD (Voltage Drain Drain) should be connected to the positive pole and VSS (Voltage Source Source) to the ground. Vo (pin 12) should be connected a variable voltage source for the display contrast adjust. Pins 17 and 18 are the backlight power supply lines (Powertip, 2008).

4.1.5 Relays

The relays are eletromechanical components capable of controlling external circuits of high current or tension starting from small current or tension. The relay has its construction based on a metallic contact that opens up or closes under the influence of electromagnetic field induced in a coil in its interior (Bates, 2006). This way, when the relay coil is supplied by an electric current, the metallic contact is attracted, opening (Normally Closed – NC) or closing (Normally Open – NO) the contact.

The JQC-3F-C model (Shenle Relay); a relay made up of a pole and two contacts, one NO and one NC; was chosen for equipment activation. The coil work tension of that relay is 5V, where the tension and maximum current between pole and contact can be up to 120 V and 10A, respectively. The current between pole and contact should be used in fan activation or exhaust fan motor

contators, or the contator of the evaporative cooling system pump. Just the JQC-3F-C NO contact was used.

In spite of the coil working with 5V and the microcontroller pins being capable to offer 5V outputs, the direct connection of the coil for relay control is not possible. That is due to the fact that, at most, 25mA of current can be drained from each pin (Microchip, 2008b), not being enough for the closing of the contact. Being such, BC337 (Philips) transistors were used as intermediaries in activation of the relays. BC337 is a general purpose NPN (negative-positive-negative) transistor (Philips, 1999). A simplified outline of its connection with the relay and the microcontroller pin can be seen in Figure 5, including the LED indicative of activation. The 1N4007 diode should be installed, inverse polarization, between the terminals of the coil to avoid current escape.

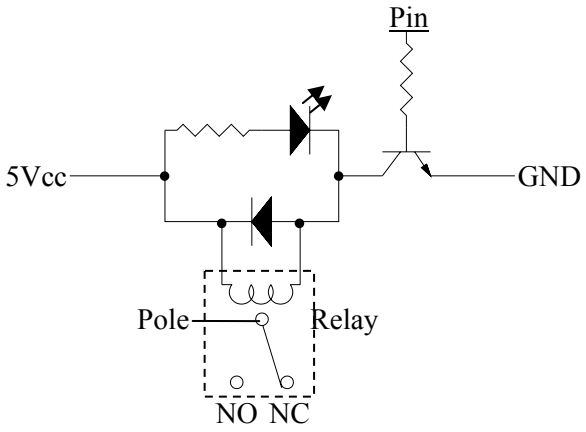


FIGURE 5 Schematic drawing of the connection between BC337 microcontroller, and relay, including activation indicator LED

4.1.6 Numeric keypad

A keypad is based on switches, however resource consumption becomes elevated in the case of each switch being connected to one pin. Switches are extensively used as a user interface (Wilmshurst, 2007). Therefore, the numeric keypad used in the hardware presents better resource use, since the switches are layed out in a matrix. The twelve-key keypad corresponds to a 4x3 matrix, where only seven pins of the microcontroller are used in the connection. Figure 6 exhibits the diagram of the keypad connection.

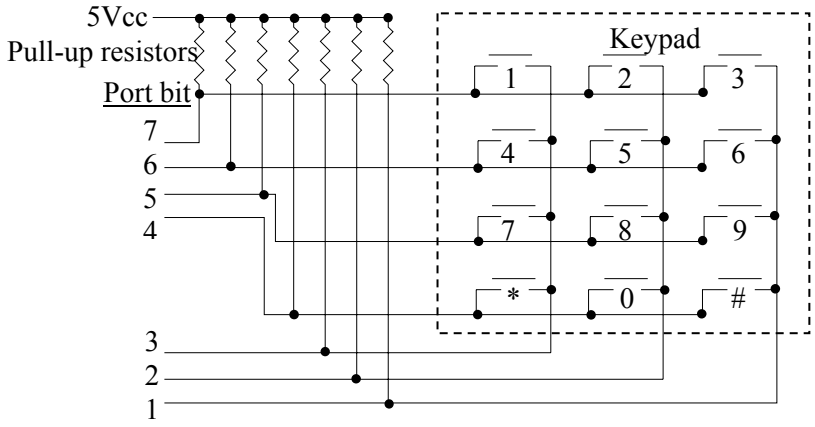


FIGURE 6 Keypad connection diagram

4.1.7 Power supply

Electrical power is produced and distributed with a sinusoidal waveform, alternating current (AC). However, electronic equipment usually works with a DC power supply (Bayindir & Ates, 2007). The hardware was all projected to work with a 5V DC source. Therefore, a cellular charger that provides 5V DC and maximum current of 700mA, switched automatically between 110 and 220V AC, was adapted as a power supply for the hardware. The control module

hardware and the sensor modules are fed by the same source. For this reason, fuses were inserted for protection of the hardware and the source. For each sensor module, a 20AG 0.1A glass fuse was inserted close to the connection point, and another 20AG 0.6A glass fuse, for the complete control system soon after the power supply.

4.2 Software

The flowchart of the main program is concisely presented in Figure 7. At the initialization, the memory position Start (h ‘00 ') is attributed to the microcontroller program counter. It is the program counter that stores the next instruction to be executed by the microcontroller. Soon afterwards, the PIC 18F4620 microcontroller, initializes the LCD module, leaving the device ready for writing. The timer0 is also initialized, being used in the counting of time.

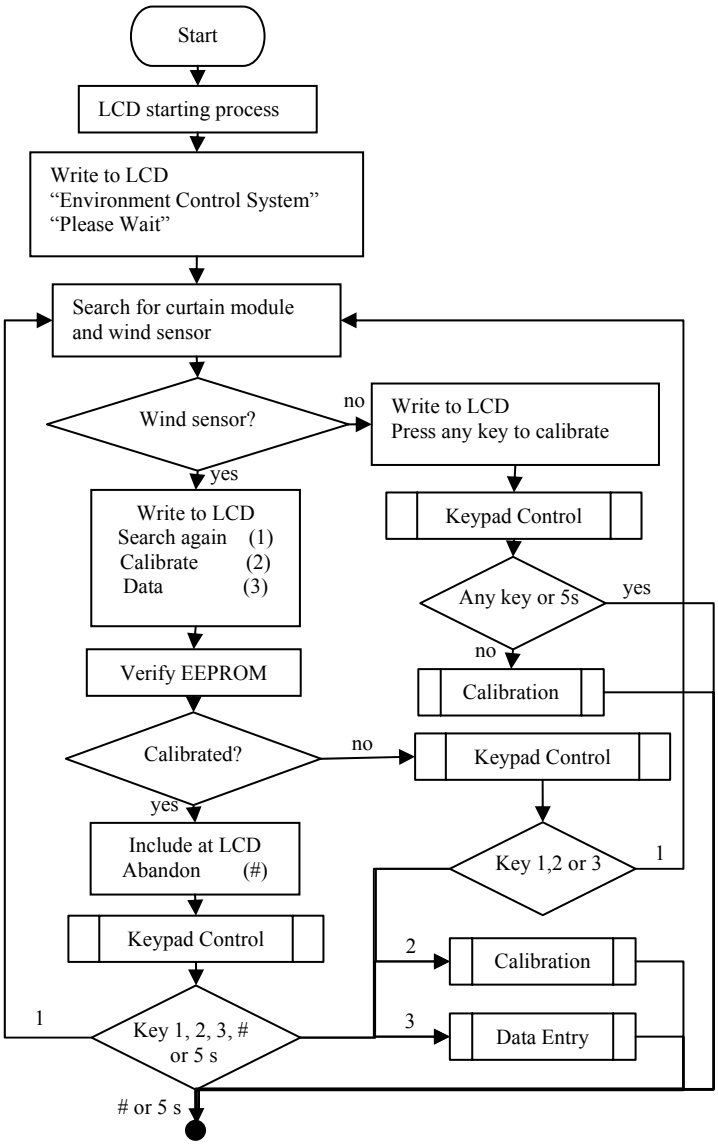


FIGURE 7 Flowchart of the control module program (…continue…)

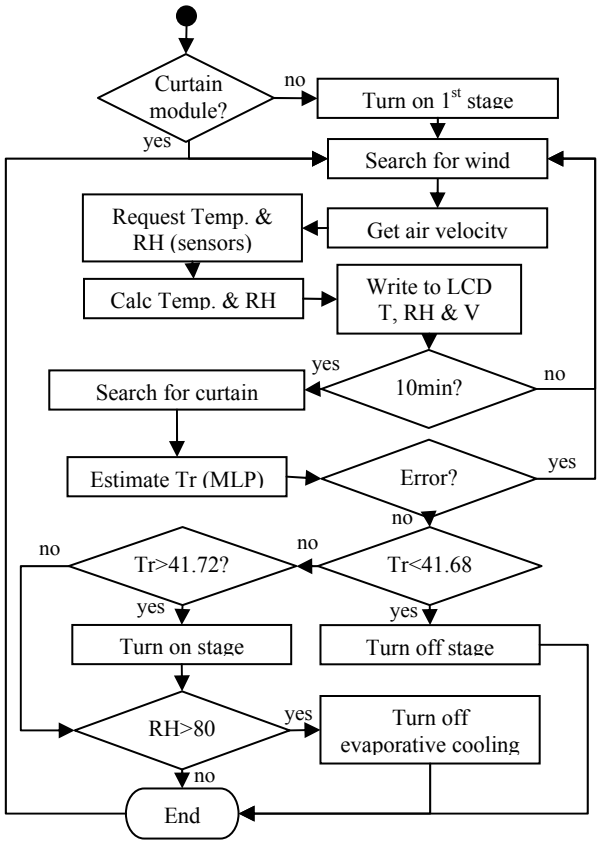


FIGURE 7 Continuation

The sentence “Environment Control System” and the message “Please wait”, which indicates that the system is busy, are written on the LCD. Three communication attempts through USART, previously configured for the RS485 standard, are performed to search for curtain control module, being the process indicated on LCD. Similar procedure is done for the air velocity (V) sensor module. The search for the hard modules lasts about three seconds. The curtain control module response, received through USART, is the state of the curtain (open or closed) which it is stored in a variable. Having a response from the V sensor module, also received by USART, a setting menu is exhibited. Setting

menu is specific for the use of the V sensor. If this function is not accessed the system works normally once the existence of the sensor guarantees the operation without need for setting. In case the curtain is present and open, a message for closing is sent through USART before the activation. One minute after the activation of each stage, V is measured. Thus, V is stored in EEPROM as a byte. This byte should be received with redundancy for higher reliability, where the equality among the bytes will be verified. In case of error, sensor not found or three consecutive equality verification errors, the system returns to the point where the system looks for the V sensor, exhibiting the "Please Wait" message.

If the search for the V sensor module fails, the system exhibits on the LCD a menu for manual setting. In general, that menu presents the options '1' seek V sensor again, '2' setting of the system and '3' direct input of the V values. If data of valid precedence, $V_1<V_2<V_3$ (V_i, V after activation of stage i – i equal to 1, 2 or 3), are found in EEPROM; the '#' option is included to quit the menu and to continue the execution. Furthermore, the menu has a maximum duration of five seconds before it continues automatically. When valid data does not exist in EEPROM, the system waits until one of the option keys is pressed.

The setting carried out by the user when typing '2' begins by verifying the state of the curtains, in case the curtain control module is present. Open curtains result in their closing by the system. The objective of the setting is to make it possible for the user to work the ventilation stages gradually as it measures and inserts the V values in the system. As such, the first stage is activated and a field is enabled for the user to insert the respective V value. The confirmation of the value by the user results in it being recorded in EEPROM. In the same way, the process repeats until the last V value, regarding the activation of the third ventilation stage, is confirmed and recorded in EEPROM. If the precedence among the values recorded in EEPROM does not check, the system

re-starts the setting process. The precedence being correct, the V values are exhibited on the LCD, asking for user confirmation.

After the correct input of data in EEPROM, or a V sensor being connected, the acquisition and control phase begins. When the curtain control module is not detected, the first ventilation stage is activated to guarantee the air quality, promoting the necessary changes. Even if the system were adjusted manually, every sensoring cycle will look for the air velocity sensor, making possible its connection without the need of restarting the system. If the V sensor module is present and the system was still not adjusted, as the ventilation stages become activated, the V values become stored in EEPROM. The value received from the sensor module or read from EEPROM is converted to m s^{-1}, division by ten, and attributed the first position of three data vectors of the thermal environment. There are three vectors, because there are up to three sensors of air temperature and humidity. In the vectors, second and third positions are, respectively, air temperature (T_{air}) and relative humidity (RH).

The T_{air} and RH acquisition happens with the request made to the modules based in the SHT11 sensor. The requests are made independently, being followed by the reception of six bytes, three regarding T_{air} and three RH. The first two (SO_T for T_{air} or SO_{RH} for RH) are data bytes and the last is the checksum. The checksum, generated by SHT11, is used for integrity verification of the bytes of the measured variables. Thus, if any error occurs, the bytes are transformed in T_{air} values in degrees Celsius (equation 1) and RH in percentage (equation 3). In the equation 3, the linearity of equation 2 is eliminated with the contribution of the T_{air}. T_{air} and RH values of each sensor are put in different vectors of the three previously mentioned. T_{air}, RH and V values are written on the LCD, one vector per line. When a humidity and temperature sensor module does not respond or three checksum errors occur, the line of LCD where the data

would be displayed is erased. The acquisition, as well as the LCD updating, happens every minute.

$$T_{air} = 0.01\ SO_T - 40 \quad (1)$$

$$RH_{linear} = -4 + 0.0405\ SO_{RH} - 2.8\times10^{-6}\ (SO_{RH})^2 \quad (2)$$

$$RH_{true} = (T_{air} - 25)\ (0.01+8\times10^{-5}\ SO_{RH})+RH_{linear} \quad (3)$$

Where:

T_{air} – air temperature (ºC);

RH_{linear} – linear relative humidity (%);

RH_{true} – true relative humidity (%);

SO_T – sensor output for temperature (integer);

SO_{RH} – sensor output for relative humidity (integer).

The ventilation and evaporative cooling stage control is accomplished based on T_r estimate by the artificial neural network (ANN) described in chapter 2. ANN is a *multilayer perceptron* (MLP) with three inputs (T_{air}, RH and V), three neurons in the hidden layer and an output (T_r) (Figure 8).

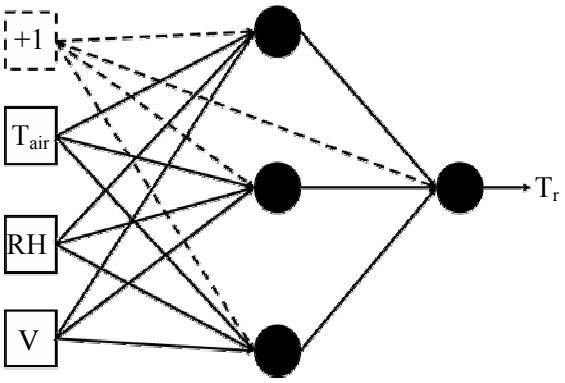


FIGURE 8 Architecture of *multilayer perceptron* MLP with 3 inputs (air temperature, T_{air}; air relative humidity, RH; and air velocity, V), 3 neurons in the hidden layer and an output (rectal temperature, T_r)

The MLP was adjusted through the backpropagation algorithm, reaching, after 99,986 epochs, a mean square error of training and validation of 0.4193 and 0.5179, respectively, and the weights listed in Table 2. The bias and the vector weights represented by the letter W can be observed. In the hidden layer W_1, W_2 and W_3 are the weights of the edges that are connected to the T_{air}, V and RH inputs, respectively. In the output neuron, W_1, W_2 and W_3 are the weights of the edges that leave neurons 1, 2 and 3, respectively.

TABLE 2 Weights resulting from the artificial neural network training process

neurons	bias	W_1	W_2	W_3
1	-0.5827986	1.4174650	-3.8484132	-6.5232177
2	1.7943517	1.1100434	-4.0314207	-7.6660550
3	-12.1465030	11.9241280	- 0.3432754	1.1697885
output	1.5810874	5.2696701	- 1.4707425	6.3003672

In this way, MLP, starting from the vectors containing the values of T_{air}, RH and V, is capable of estimating T_r. The highest of estimated T_r values is used

as criterion in decision making. The process repeats itself every ten minutes, followed by the climatization system control. The choice of ten minutes is justified because it is time needed for verifying air speed changes on rectal temperature (T_r) of broilers (Furlan et al., 2000).

The T_r of 41.7 ºC, average body temperature point of adult broiler chickens in the thermoneutral zone (Tao & Xin, 2003), was considered as limit between comfort and beginning of caloric stress. However, wanting to avoid activation followed by a possible deactivation, a tolerance of 0.02 ºC was assumed. Therefore, if the highest rectal temperature (T_r) value is less than 41.68 ºC, a control stage is disabled. Regarding the presence of curtain control module and just one stage of ventilation activated, the stage is disabled and the curtains are open. If only the first stage activated and no curtain module exists, no operation is accomplished. The activation of stages happens when T_r is higher than 41.72 ºC. Evaporative cooling stage is only activated when the relative humidity (RH) is smaller than 70%, avoiding illegitimate activations. If RH is higher than 80% the evaporative cooling will be turned off.

The software produced for the RH and T_{air} sensor modules, besides the control and communication functions with SHT11 sensor, implements the communication with the control module. After initializing variables and communication with SHT11, the microcontroller, PIC 18F1320, waits for a specific pair of bytes, sent by the control module, for the beginning of the T_{air} and RH acquisition process. If the pair is received, the microcontroller sends SHT11 the request for acquisition of T_{air} (byte 0x03). After the reception of the bytes regarding T_{air}, or its failure, the microcontroller makes the request for RH (byte 0x05). Having RH bytes, not having errors in their reception or those of T_{air}, the transfer begins to the control module. First, three bytes are sent regarding T_{air}, two of data and one of checksum, later, in the same way, three regarding RH. Following, when a flaw occurs in the data verification with the

use of the checksum, the sensor module receives two acquisition requests in a row, resulting in the reset of SHT11. In case of error during the reception, by PIC18F1320, of the T_{air} or RH bytes coming from SHT11, the connection between sensor and microcontroller receives a reset sequence. The flowchart of the software embedded in PIC 18F1320 is illustrated in Figure 9.

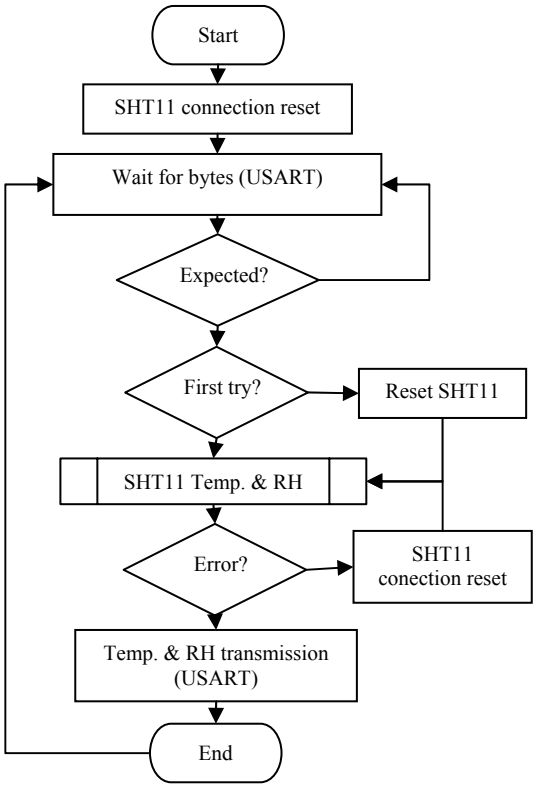


FIGURE 9 Flowchart of the sensor module program

The control module and sensor modules programs were implemented in C language (APPENDIX A, B, C and D). The watch dog timer, whose purpose is the restart in case of lock-up, is used in both programs. The control module program offers the possibility of integration of the air velocity sensor and curtain control modules, without any source code modification. The control module and sensor modules are exhibited in the Figure 10.

FIGURE 10 Control module and sensor modules

5 System Test

For the system test the sensors were conditioned in a duct, where alterations were produced in thermal environment, T_{air} and RH. The 10 cm diameter duct was connected to a 54x33x22 cm (length x width x height) compartment, in which there was the possibility of heating the air. The humidification and the cooling were accomplished through moistened porous material introduced in a traverse section of the duct. Inside the compartment, two fans promoted air circulation inside the duct, guaranteeing the mixture of the air. T_{air} and RH calibrated sensors were placed close to the developed sensor modules for verifying its functioning.

Thus, the operation and stability of the system was verified, making possible the detection and correction of errors, not observed during the implementation of the software. Indeed, sensor response time was checked. During the procedure, the T_{air} and RH varied from 23.1 to 40.1 °C e 47.0 to 95.1 %, respectively.

6 Results and Discussion

In this study, the neuron controller operation for climatized broiler houses was evaluated only under laboratory conditions. However, it was possible to observe the capacity, aggregated by the use of the artificial neural network, in relating the environmental variables (T_{air}, RH and V) and translating them to comfort conditions. Considering, for instance, RH superior to 60% inside the broiler house and first stage activated providing an air velocity (V) of 0.5 m s^{-1}, the T_{air} at which the next stage will be activated can vary from 24.6 to 28 °C, being dependent on the RH value. When all the considered variables are within the thermal comfort limits, T_{air} between 21 and 26 °C, RH from 50 to 70% and V between 0.5 and 1.5 m s^{-1} (Medeiros, 2001), activations do not occur. An example of system profile during the test phase is illustrated in Figure 11. The V values inserted in the control module as the expected air velocity after activation of the first, second and third ventilation stages were, respectively, 0.5, 1 and 1.5 m s^{-1}.

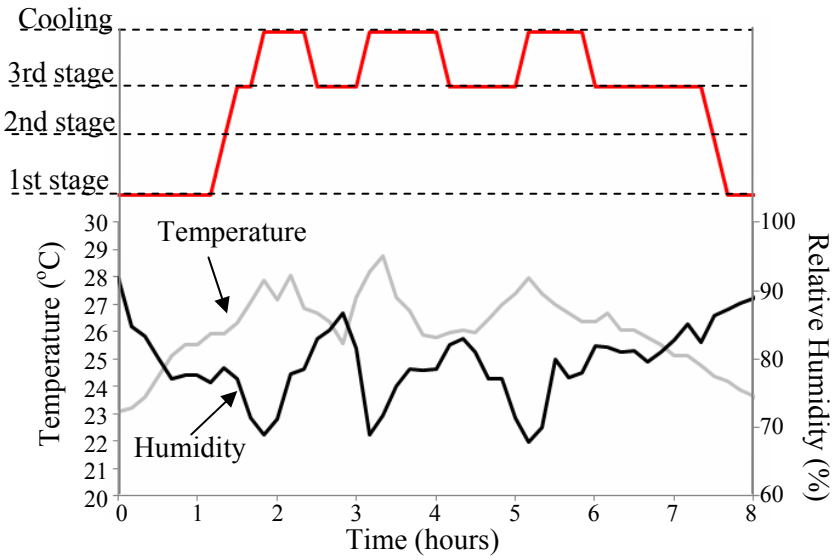


FIGURE 11 System profile exposed to changes in thermal environment during the test phase

In Figure 11, given the absence of the curtain control module, it can be observed that the first ventilation stage was turned on during the entire period. The activation of the second stage, T_{air} 26.0 °C and RH 76.7%, was followed by the activation of the third, T_{air} 26.0 °C and RH 78.7%, once stress conditions were maintained. The activation of the evaporative cooling happened when the T_{air} was approximately 28.0 °C and RH inferior to 70%, being turned off when the RH was superior to 80%. The period considered as caloric stress had a duration of 6 hours, with average T_{air}, RH and V values equal to, respectively, 26.6 °C, 78% and 1.5 m s^{-1}. With T_{air} of 24.8 °C and RH of 82.4%, the gradual deactivation of the ventilation stages began. Being such, the system showed solid behavior for the search for adequate comfort levels. In addition, the system showed stable during the test phase.

The PICs provided fast implementation and facilitated the experimental process due its superior features such as changing software, low cost, easy programmable and usage elasticity. The use of PIC microcontrollers provided the construction of a quite compact low cost system.

The SHT 11 sensor is easily integrated to PICs and presents an adequate response time. When the neuron-controller system developed was tested in the laboratory, simulating similar gradual variation of air temperature and relative humidity as reported by Corria & Nääs (2001) and Damasceno (2008) in field studies, the delay introduced by the material used in the device protection was negligible.

7 Conclusion

The developed neuron controller, based on a MLP (*multilayer perceptron*) artificial neural network and PIC (peripheral interface controller) microcontrollers, resulted in a robust, flexible and low cost system for the control of climatized broiler houses. The neuron controller was shown capable of translating air temperature, relative humidity and air velocity into comfort conditions. Its behavior, at a laboratorial scale, was also suitable for the search for adequate thermal conditions for poultry production.

8 Acknowledgements

To CAPES and FAPEMIG for financial support. To SENSIRION for furnishing the SHT11 sensor samples.

9 References

ABU-DIEYEH, Z. M. M. Effect of chronic heat stress and long-term feed restriction on broiler performance. **International Journal of Poultry Science**, Lasani, v. 5, n. 2, p. 185-190, Feb. 2006.

AMIRANTE, R.; INNONE, A.; CATALANO, L. A. Boosted PWM open loop control of hydraulic proportional valves. **Energy Conversion and Management**, Oxford, v. 49, n. 8, p. 2225-2236, Aug. 2008.

BAÊTA, F. C.; SOUZA, C. F. **Ambiência em edificações rurais:** conforto animal. Viçosa: UFV, 1997. 246 p.

BATES, M. **Interfacing PIC microcontrollers embedded design by interactive simulation**. Oxford: Elsevier, 2006. 313 p.

BAYINDIR, R.; ATES, H. Low-cost and high sensitively microcontroller based control unit for a friction welding machine. **Journal of Material Processing Technology**, Oxford, v. 189, n. 1/3, p. 126-131, Jan. 2007.

BAYINDIR, R.; SAGIROGLU, S., COLAK, I. An intelligent power factor corrector for power system using artificial neural networks. **Electric Power Systems Research**, Oxford, v. 79, n. 1, p. 152-160, Jan. 2009.

BUENO, L. G. F. **Avaliação da eficiência energética e do conforto térmico em instalações de frangos de corte**. 2004. 100 p. Dissertação (Mestrado em Engenharia Agrícola) – Universidade Estadual de Campinas, Campinas, SP.

CHUNG, W. Y.; LEE, S. C. A selective AQS system with artificial neural network in automobile. **Sensors and Actuators B: Chemical**, New York, v. 130, n. 1, p. 258-263, Mar. 2008.

CORRIA, M. E.; NÄÄS, I. A. Adapting a tunnel ventilation for high density broiler production: a case study. In: INTERNATIONAL LIVESTOCK ENVIRONMENT SYMPOSIUM, 6, 2001, Louisville, **Proceedings…**, Louisvile: ASAE, 2001, p. 461-467.

DAMASCENO, F. A. **Ajuste de modelos e determinação de índice térmico ambiental de produtividade para frangos de corte**. 2008. 220 p. Dissertação (Mestrado em Engenharia Agrícola) – Universidade Federal de Lavras, Lavras, MG.

FRANKOWIAK, M. R.; GROSVENOR, R. I.; PRICKETT, P. W. A Petri-net based distributed monitoring system using PIC microcontrollers. **Microprocessors and Microsystems**, Amsterdam, v. 29, n. 5, p. 189-196, June 2005.

FURLAN, R. L.; MACARI, M. SECATO, E. R.; GUERREIRO, J. R. Air velocity and exposure time to ventilation affect body surface and rectal temperature of broiler chickens. **Applied Poultry Research**, Savoy, v. 9, n. 1, p. 1-5, Mar./May 2000.

KUMAR, N. S.; SADASIVAM, V., ASAN SUKRIYA, H. M.; BALAKRISHNAN, S. Design of low cost universal artificial neuron controller for chopper fed embedded DC drives. **Applied Soft Computing**, New York, v. 8, n. 1, p. 1637-1642, Jan. 2008.

MAXIM. **Low-power, slew-rate-limited rs-485/rs-422 transceivers**. 2008. 19 p. Disponível em: < http://datasheets.maxim-ic.com/en/ds/MAX1487-MAX491.pdf> Acesso em: 28 jan. 2008.

MEDEIROS, C. M. **Ajuste de modelos e determinação de índice térmico ambiental de produtividade para frangos de corte**. 2001. 115 p. Tese (Doutorado em Engenharia Agrícola) – Universidade Federal de Viçosa, Viçosa, MG.

MICROCHIP. **8-bit PIC microcontroller solutions**. 2008a. 16 p. Disponível em: <www.microchip.com/8bit> Acesso em: 6 dez. 2008.

MICROCHIP. **PIC18F1220/1320 data sheet:** 18/20/28-pin high-performance, enhanced flash microcontrollers with 10-bit a/d and nanowatt technology. 2008b. 308 p. Disponível em: < ww1.microchip.com/downloads/en/DeviceDoc/39605f.pdf > Acesso em: 11 jan. 2008.

MICROCHIP. **PIC18F2525/2620/4525/4620 data sheet:** 28/40/44-pin, enhanced flash microcontrollers with 10-bit a/d and nanowatt technology. 2008c. 412 p. Disponível em: < ww1.microchip.com/downloads/en/DeviceDoc/39626e.pdf > Acesso em: 10 jan. 2008.

PADILHA, A. S.; FARRET, F. A.; POPOV, V. A. Neurofuzzy controller in automated climatization for poultry houses. In: ANNUAL CONFERENCE OF THE IEEE INDUSTRIAL ELECTRONIC SOCIETY, 27, 2001, Denver, **Proceedings…**, Denver: IECON'01, 2001, p. 70-75.

PEREIRA, D. F.; NÄÄS, I. A. Estimativa do conforto de matrizes de frango de corte baseada em análise do comportamento de preferência térmica. **Engenharia Agrícola**, Jaboticabal, v. 25, n. 2, p. 315-321, maio/ago. 2005.

PHILIPS. **BC337 NPN general purpose transistor:** Datasheet. 9 p. 1999. Disponível em: < http://www.datasheet4u.com/download.php?id=128246> Acesso em: 15 jun. 2008.

POWERTIP. Powertip Technology Corporation. 2000 – 2005. **Apresenta informações sobre o módulo LCD PC4004A**. 2008. Disponível em: <http://www.powertip.com.tw/products_2.php?product_id=1171055369&area_idbk=1170985616> Acesso em: 17 dez. 2008.

ROUSH, W. B.; CRAVENER, T. L.; KIRBY, Y. K.; WIDEMAN JR., R. F. Probabilistic neural network prediction of ascites in broilers based on minimally invasive physiological factors. **Poultry Science**, Champaign, v. 76, n. 11, p. 1513-1516, June 1997.

SENSIRION. **Datasheet SHT1x (SHT10, SHT11, SHT15):** humidity and temperature sensor. 2007. 11 p. Disponível em: <http://www.sensirion.com/en/pdf/product_information/Datasheet-humidity-sensor-SHT1x.pdf> Acesso em: 12 out. 2007.

SPIERS, D. E.; SPAIN, J. N.; SAMPSON, J. D.; RHOADS, R. P. Use of physiological parameters to predict milk yield and feed intake in heat-stressedd dairy cows. **Jounal of Thermal Biology**, Pergamon, v. 29, n. 7-8, p. 759-764, Oct./Dec. 2004.

TAO, X.; XIN, H. Acute synergistic effects of air temperature, humidity, and velocity on homeostasis of market–size broilers. **Transactions of the ASAE**, Saint Joseph, v. 46, n. 2, p. 491–497, Dec. 2003.

TURKOGLU, I. Hardware implementation of varicap diode's ANN model using PIC microcontrollers. **Sensors and Actuators A: Physical**, New York, v. 138, n. 2, p. 288-293, May 2007.

WILMSHURST, T. **Designing embedeed systems with PIC microcontrollers:** principles and applications. Oxford: Elsevier, 2007. 583 p.

YAHAV, S.; STRASCHNOW, A.; PLAVNIK, I.; HURWITZ, S. Blood system response of chickens to changes in environmental temperature. **Poultry Science**, Champaign, v. 80, n. 6, p. 724-726, June 2001.

ZHENYAO, Z.; JIYANG, D.; ZHONG, C.; ZHIKAI, C.; QINGYIN, J. Design of interface software for a distributed system with mixed varied intelligent meters. In: INTERNATIONAL CONFERENCE ON CONTROL, AUTOMATION, ROBOTICS AND VISION, 8, 2006, Singapore, **Proceedings…**, Singapore: ICARCV, 2006. 4 p.

CHAPTER 4

Final considerations

1 General Conclusions

The artificial neural networks (ANNs) adjusted for the prediction of rectal temperature of broiler chickens as a function of air temperature, relative humidity and air velocity were trained and validated, presenting statistical indexes of fitting adequate to applications related to poultry production. Specifically in the thermal environment control activity inside the climatized broiler houses, the application of ANNs was shown to be a promising technique, making the integration of thermal environment variables in decision making possible. Furthermore, ANN's are easily implemented in microcontrollers, producing simple and compact code. The developed neuron controller presented a stable behavior; without random switching; and suitable for the search for adequate comfort levels. The use of PIC microcontrollers made possible the construction of a low cost robust control system.

2 Future Works

The development of V sensor module in the present research was restricted by the high cost of air velocity (V) sensor. Therefore, the development of a V sensor module, or even the sensor, continues as a proposal for future works, as well as the development of the curtain control module.

The scarcity of experimental data in the literature resulted in some limitations or generalizations for this research. As ANNs produce quite compact

codes, it would be possible to develop and to implement ANNs for different broiler breeds, which its selection could be made via numeric keypad.

The possibilities for future works are countless due to flexibility of the developed hardware, going from this to the change of technique used in the control or increment of other physiologic or productive parameters. Besides, small improvements in the hardware can bring high flexibility gains, field system tests, followed by comparison with commercial control systems, assumes a prominent role in the continuity of the present research.

APPENDICES

APPENDIX A - Control module main program

```
1: /* CCS control module source code.
2: /*
3: * Name: main.c
4: * Copyright:
5: * Author: Alison Zille Lopes
6: * Date: 27/06/08 14:27
7: * Description: control module main program
8: */
9:
10: #include <18f4620.h>
11: #use delay(clock=20000000)
12: #use rs232 (baud = 9600,xmit = pin_c6,rcv = pin_c7,
enable = pin_c5)
13: #include <math.h>
14: #include <string.h>
15: #include <stdlib.h>
16: #include "lcd.h"
17: #include "keypad.h"
18: #fuses HS,WDT128
19:
20: //-------------------------------------------------------
21: // Control Commands
22: //-------------------------------------------------------
23:
24: #define SHT11_1 0x30
25: #define SHT11_2 0x31
26: #define SHT11_3 0x32
27: #define WIND 0x33
28: #define CURTAIN 0x34
29: #define OPEN_CURTAIN 0x35
30: #define CLOSE_CURTAIN 0x36
31:
32:
33: //-------------------------------------------------------
34: // Equipments LED and/or activation
35: //-------------------------------------------------------
36:
37: #define CUR_LED pin_c2
38: #define STAGE_1 pin_c3
39: #define STAGE_2 pin_d0
40: #define STAGE_3 pin_d1
41: #define NOZZLE pin_d2
42:
```

```
43: //-------------------------------------------------------
44: // Sensor LED
45: //-------------------------------------------------------
46:
47: #define TH_LED_1 pin_e1
48: #define TH_LED_2 pin_e2
49: #define TH_LED_3 pin_c0
50: #define WIND_LED pin_c1
51:
52:
53: //-------------------------------------------------------
54: // START DEFINITIONS OF SENSOR FUNCTIONS
55: //-------------------------------------------------------
56:
57: //-------------------------------------------------------
58: float sht11_temp(long int temp)
59: //-------------------------------------------------------
60: // calculates temperature [°C]
61: // input : temp [Ticks] (14 bit)
62: // output: temp [°C]
63: {
64:   float t_C;
65:   t_C = (temp*0.01) - 40; //calc. Temperature from
ticks to [IC]
66:   return t_C;
67: }
68:
69: //-------------------------------------------------------
70: float sht11_humi(long int humi,float temp)
71: //-------------------------------------------------------
72: // calculates humidity [%RH]
73: // input : humi [Ticks] (12 bit)
74: // output: humi [%RH]
75: {
76:   const float C1=-4.0; // for 12 Bit
77:   const float C2= 0.0405; // for 12 Bit
78:   const float C3=-0.0000028; // for 12 Bit
79:   const float T1=0.01; // for 14 Bit @ 5V
80:   const float T2=0.00008; // for 14 Bit @ 5V
81:   float rh; //= *p_humidity; // rh: Humidity [Ticks] 12
Bit
82:   float rh_lin; // rh_lin: Humidity linear
83:   float rh_true; // rh_true: Temperature compensated
humidity
84:   rh = humi;
85:   rh_lin=C3*rh*rh + C2*rh + C1; //calc. Humidity from
ticks to [%RH]
86:   rh_true=(temp-25)*(T1+T2*rh)+rh_lin; //calc.
Temperature compensated humidity [%
87:   if(rh_true>100)rh_true=100; //cut if the value is
outside of
88:   if(rh_true<0.1)rh_true=0.1; //the physical possible
range
```

```
89:    return rh_true; //return humidity[%RH]
90: }
91:
92: //--------------------------------------------------
93: boolean sht11_crc8(unsigned int hi, unsigned int lo,
unsigned int checksum,boolean
94: //--------------------------------------------------
95: {
96:    // verifies if it the crc transmited and the data
received match
97:    // input: received data (MSB [hi] and LSB [lo]) and
type of data
98:    // (1 - temperature or 0 - humidity)
99:    // output: 1 if the crc match with the data
100:
101:    unsigned int aux,i,mask; // aux - auxiliar variable
to store the CRC for the
102:    // calculation procedure
103:    // i - counter used in the for-loop
104:    // mask - used before the xor operation to prevent
105:    // the effect only on the most significant bit.
106:
107:    #bit bit0 = aux.0 // access the bit 0
108:
109:    if(m_type) // if the received data is temperature
110:    {
111:      aux = 0x53;
112:    }
113:    else // received data is humidity
114:    {
115:      aux = 0xF5;
116:    }
117:
118:    // bitwise method to calculate the CRC
119:
120:    for(i=0;i<8;i++)
121:    {
122:      mask = 0x80 & aux; // mask to catch only the 7th
bit
123:      aux = mask ^ hi; // xor on bit 7
124:      rotate_left(&aux,1); // rotate aux left
125:      rotate_left(&hi,1); // rotate hi left
126:      if(bit0) // if bit0 == 1
127:      {
128:        aux = aux ^ 0x30; // invert bit 4 and 5
129:      }
130:    }
131:
132:    for(i=0;i<8;i++)
133:    {
134:      mask = 0x80 & aux; // mask to catch only the 7th
bit
135:      aux = mask ^ lo; // xor on bit 7
```

```
136:     rotate_left(&aux,1); // rotate aux left
137:     rotate_left(&lo,1); // rotate hi left
138:     if(bit0) // if bit0 == 1
139:     {
140:       aux = aux ^ 0x30; // invert bit 4 and 5
141:     }
142:   }
143:   if(checksum == aux)
144:   {
145:     return 1; // return 1 if checksum match
146:   }
147:   else
148:   {
149:     return 0; // return 0 in case of mismatch
150:   }
151: }
152:
153:
154: //---------------------------------------------------
155: // END DEFINITIONS OF SENSOR FUNCTIONS
156: //---------------------------------------------------
157:
158:
159: //---------------------------------------------------
160: // START DEFINITIONS OF ARTIFICIAL NEURAL NETWORK
FUNCTIONS
161: //---------------------------------------------------
162:
163: //---------------------------------------------------
164: float ann_log_activ(float x)
165: //---------------------------------------------------
166: // Artificial Neural Network activation function
167: {
168:   float y; //activation function output
169:   y = 1/(1+exp(-x)); //activation function
170:   return y; //return output
171: }
172:
173: //---------------------------------------------------
174: float ann_FF_run(float environment[])
175: //---------------------------------------------------
176: // estimates rectal temperature [°C]
177: // inputs: temperature, humidity and air velocity
178: // output: rectal temperature [°C]
179: {
180:   float temp, result,x[3];
181:   x[0] = environment[0]/6;
182:   x[1] = environment[1]/50;
183:   x[2] = environment[2]/100;
184:   temp = -0.5827986 - 3.8484132*x[0] + 1.417465*x[1] -
6.5232177*x[2];
185:   temp = ann_log_activ(temp);
186:   result = 1.5810874 + 5.2696701*temp;
```

```
187:   temp = 1.7943517 - 4.0314207*x[0] + 1.1100434*x[1] -
7.666055*x[2];
188:   temp = ann_log_activ(temp);
189:   result = result - 1.4707425*temp;
190:   temp = - 12.146503 - 0.3432754*x[0] + 11.924128*x[1]
+ 1.1697885*x[2];
191:   temp = ann_log_activ(temp);
192:   result = result + 6.3003672*temp;
193:   result = ann_log_activ(result);
194:   result = result*50;
195:   return result; // return output
196: }
197:
198: //-----------------------------------------------------
199: // END DEFINITIONS OF ARTIFICIAL NEURAL NETWORK
FUNCTIONS
200: //-----------------------------------------------------
201:
202:
203: //-----------------------------------------------------
204: boolean calibrate(boolean av_module,boolean av_levels)
205: //-----------------------------------------------------
206: // calibrates the system
207: // av_module: if true the calibration is made with air
velocity sensor
208: // av_levels: if true the system is already calibrated
209: {
210:   // variables
211:   char key; // pressed key
212:   float av; // air velocity
213:   unsigned int i; // one index or auxiliar variable
214:   unsigned int air, check; // air velocity and comfirm
byte (redundancy)
215:   char line[41]; // corresponds to one LCD line
216:
217:   again: // in case of error or user choice restart
here
218:
219:   restart_wdt(); // restart watch dog timer
220:
221:   // deactive ventilation stages
222:   output_low(STAGE_1);
223:   output_low(STAGE_2);
224:   output_low(STAGE_3);
225:
226:   //clear LCD
227:   WriteCmdHalfUpLCD(ERASE);
228:   WriteCmdHalfDownLCD(ERASE);
229:   while(BusyHalfDownLCD());
230:   while(BusyHalfUpLCD());
231:
232:
233:   SetDDRamAddrHalfUp(0x0E); // changes cursor position
```

```
234:    while(BusyHalfUpLCD());
235:    strcpy(line,"Calibration"); // atributse one string
to the line
236:    putsHalfUpLCD(line); // sends line to LCD
237:    while(BusyHalfUpLCD());
238:
239:    WriteCmdHalfUpLCD(HOME); // returns cursor to
initial position
240:    while(BusyHalfUpLCD());
241:    SetDDRamAddrHalfUp(0x29); // changes cursor position
242:    while(BusyHalfUpLCD());
243:    sprintf(line,"[%c] 1st stage: ",0xA5); // atributes
one string to the line
244:    putsHalfUpLCD(line); // sends line to LCD
245:    while(BusyHalfUpLCD());
246:
247:    strcpy(line,"[ ] 2sd stage:"); // atributes one
string to the line
248:    putsHalfDownLCD(line); // sends line to LCD
249:    while(BusyHalfDownLCD());
250:
251:    WriteCmdHalfDownLCD(HOME); // returns cursor to
initial position
252:    while(BusyHalfDownLCD());
253:    SetDDRamAddrHalfDown(0x29); // changes cursor
position
254:    while(BusyHalfDownLCD());
255:    strcpy(line,"[ ] 3rd stage:"); // atributes one
string to the line
256:    putsHalfDownLCD(line); // sends line to LCD
257:    while(BusyHalfDownLCD());
258:
259:
260:    if(av_module) // if air velocity sensor is available
261:    {
262:      output_high(STAGE_1); // activates ventilation
first stage
263:
264:      for(i=0;i<20;i++) // waits approximately 1 minute
265:      {
266:        set_timer0(0); // timer0 is set (prescale value
must be 1:256)
267:        while(get_timer0()<58594) // 3 seconds
268:        {
269:          restart_wdt();
270:        }
271:      }
272:      i = 0;
273:      again_0:
274:      restart_wdt();
275:      for(i=0;i<3;i++)
276:      {
277:        putc(0xFF);
```

```
278:        delay_us(10);
279:        putc(WIND);
280:        set_timer0(0); // timer0 is set (1:256 prescale
value)
281:        while(get_timer0()<9766)
282:        {
283:          if(kbhit()) // if receive, sensor is ok
284:          {
285:            goto ok_av1;
286:          }
287:        }
288:        restart_wdt();
289:      }
290:      return 0; // error
291:
292:      ok_av1:
293:
294:      restart_wdt();
295:
296:      getc();
297:
298:      // in case of error during reception
299:      set_timer0(0);
300:      while((get_timer0()<1954)&&(!kbhit())); // waits
100ms for a byte
301:      if(!kbhit())
302:      {
303:        i++;
304:        if(i>2) // more than two errors
305:        {
306:          return 0; // return false
307:        }
308:        goto again_0; // try again
309:      }
310:
311:      getc();
312:
313:      // in case of error during reception
314:      set_timer0(0);
315:      while((get_timer0()<1954)&&(!kbhit())); // waits
100ms for a byte
316:      if(!kbhit())
317:      {
318:        i++;
319:        if(i>2) // more than two errors
320:        {
321:          return 0; // return false
322:        }
323:        goto again_0; // try again
324:      }
325:
326:      air = getc(); // receives air velocity data
327:
```

```
328:
329:      // in case of error during reception
330:      set_timer0(0);
331:      while((get_timer0()<1954)&&(!kbhit())); // waits
100ms for a byte
332:      if(!kbhit())
333:      {
334:        i++;
335:        if(i>2) // more than two errors
336:        {
337:          return 0; // return false
338:        }
339:        goto again_0; // try again
340:      }
341:
342:      check = getc(); // receives air velocity
redundancy
343:
344:      if(air!=check) // air velocity data and the
redundacy do not match
345:      {
346:        i++;
347:        if(i>2) // more than two errors
348:        {
349:          return 0; // return false
350:        }
351:        goto again_0; // try again
352:      }
353:
354:      write_eeprom(0,air); // data save
355:      av = air*0.1; // calculates of real air velocity
356:      sprintf(line,"%01.1f m/s",av); // passes air
velocity to string
357:      putsHalfUpLCD(line); // prints the string at LCD
358:      while(BusyHalfUpLCD());
359:
360:      WriteCmdHalfUpLCD(HOME); // returns cursor to
initial position
361:      while(BusyHalfUpLCD());
362:      SetDDRamAddrHalfUp(0x29); // changes cursor
position
363:      while(BusyHalfUpLCD());
364:      WriteCmdHalfUpLCD(SHIFT_CUR_RIGHT); // shifts
cursor one position right
365:      while(BusyHalfUpLCD());
366:      WriteDataHalfUpLCD(' '); // dismarks 1st stage
367:      while(BusyHalfUpLCD());
368:
369:      WriteCmdHalfDownLCD(HOME); // returns cursor to
initial position
370:      while(BusyHalfDownLCD());
371:      SetDDRamAddrHalfDown(0x00); // makes sure that it
is 0 address
```

```
372:      while(BusyHalfDownLCD());
373:      WriteCmdHalfDownLCD(SHIFT_CUR_RIGHT); // shifts
cursor one position right
374:      while(BusyHalfDownLCD());
375:      WriteDataHalfDownLCD(0xA5); // marks 2nd stage
376:      while(BusyHalfDownLCD());
377:      for(i=0;i<13;i++) // shifts cursor to the entry
point
378:      {
379:        WriteCmdHalfDownLCD(SHIFT_CUR_RIGHT);
380:        while(BusyHalfDownLCD());
381:      }
382:
383:
384:      output_high(STAGE_2); // activates ventilation
second stage
385:
386:      for(i=0;i<20;i++) // waits approximately 1 minute
387:      {
388:        set_timer0(0); // timer0 is set (1:256 prescale
value)
389:        while(get_timer0()<58594) // 3 seconds
390:        {
391:          restart_wdt();
392:        }
393:      }
394:      i = 0;
395:      again_1:
396:      restart_wdt();
397:      for(i=0;i<3;i++)
398:      {
399:        putc(0xFF);
400:        delay_us(10);
401:        putc(WIND);
402:        set_timer0(0); // timer0 is set (1:256 prescale
value)
403:        while(get_timer0()<9766)
404:        {
405:          if(kbhit()) // if receive, sensor is ok
406:          {
407:            goto ok_av2;
408:          }
409:          restart_wdt();
410:        }
411:      }
412:      return 0; // error
413:
414:      ok_av2:
415:
416:      restart_wdt();
417:
418:      getc();
419:
```

```
420:      // in case of error during reception
421:      set_timer0(0);
422:      while((get_timer0()<1954)&&(!kbhit())); // waits
100ms for a byte
423:      if(!kbhit())
424:      {
425:        i++;
426:        if(i>2) // more than two errors
427:        {
428:          return 0; // return false
429:        }
430:        goto again_1; // try again
431:      }
432:
433:      getc();
434:
435:      // in case of error during reception
436:      set_timer0(0);
437:      while((get_timer0()<1954)&&(!kbhit())); // waits
100ms for a byte
438:      if(!kbhit())
439:      {
440:        i++;
441:        if(i>2) // more than two errors
442:        {
443:          return 0; // return false
444:        }
445:        goto again_1; // try again
446:      }
447:
448:      air = getc(); // receives air velocity data
449:
450:
451:      // in case of error during reception
452:      set_timer0(0);
453:      while((get_timer0()<1954)&&(!kbhit())); // waits
100ms for a byte
454:      if(!kbhit())
455:      {
456:        i++;
457:        if(i>2) // more than two errors
458:        {
459:          return 0; // return false
460:        }
461:        goto again_1; // try again
462:      }
463:
464:      check = getc(); // receives air velocity
redundancy
465:
466:      if(air!=check) // air velocity data and the
redundacy do not match
467:      {
```

```
468:        i++;
469:        if(i>2) // more than two errors
470:        {
471:          return 0; // return false
472:        }
473:        goto again_1; // try again
474:      }
475:
476:      write_eeprom(1,air); // data save
477:      av = air*0.1; // calculates of real air velocity
478:      sprintf(line,"%01.1f m/s",av); // passes air
velocity to string
479:      putsHalfDownLCD(line); // prints the string at LCD
480:      while(BusyHalfDownLCD());
481:
482:      WriteCmdHalfDownLCD(HOME); // returns cursor to
initial position
483:      while(BusyHalfDownLCD());
484:      SetDDRamAddrHalfDown(0x00); // makes sure it is 0
address
485:      while(BusyHalfDownLCD());
486:      WriteCmdHalfDownLCD(SHIFT_CUR_RIGHT); // shifts
cursor one position right
487:      while(BusyHalfDownLCD());
488:      WriteDataHalfDownLCD(' '); // dismarks 2nd stage
489:      while(BusyHalfDownLCD());
490:
491:      WriteCmdHalfDownLCD(HOME); // returns cursor to
initial position
492:      while(BusyHalfDownLCD());
493:      SetDDRamAddrHalfDown(0x29); // changes cursor
position
494:      while(BusyHalfDownLCD());
495:      WriteCmdHalfDownLCD(SHIFT_CUR_RIGHT); // shifts
cursor one position right
496:      while(BusyHalfDownLCD());
497:      WriteDataHalfDownLCD(0xA5); // markes 3nd stage
498:      while(BusyHalfDownLCD());
499:
500:      for(i=0;i<13;i++) // shifts cursor to the entry
point
501:      {
502:        WriteCmdHalfDownLCD(SHIFT_CUR_RIGHT);
503:        while(BusyHalfDownLCD());
504:      }
505:
506:      output_high(STAGE_3); // activates ventilation
third stage
507:
508:      for(i=0;i<20;i++) // waits approximately 1 minute
509:      {
510:        set_timer0(0); // timer0 is set (1:256 prescale
value)
```

```
511:        while(get_timer0()<58594) // 3 seconds
512:        {
513:          restart_wdt();
514:        }
515:      }
516:
517:      i = 0;
518:      again_2:
519:      restart_wdt();
520:      for(i=0;i<3;i++)
521:      {
522:        putc(0xFF);
523:        delay_us(10);
524:        putc(WIND);
525:        set_timer0(0); // timer0 is set (1:256 prescale
value)
526:        while(get_timer0()<9766)
527:        {
528:          if(kbhit()) // if receive, sensor is ok
529:          {
530:            goto ok_av3;
531:          }
532:          restart_wdt();
533:        }
534:      }
535:      return 0; // error
536:
537:      ok_av3:
538:
539:      restart_wdt();
540:
541:      getc();
542:
543:      // in case of error during reception
544:      set_timer0(0);
545:      while((get_timer0()<1954)&&(!kbhit())); // waits
100ms for a byte
546:      if(!kbhit())
547:      {
548:        i++;
549:        if(i>2) // more than two errors
550:        {
551:          return 0; // return false
552:        }
553:        goto again_2; // try again
554:      }
555:
556:      getc();
557:
558:      // in case of error during reception
559:      set_timer0(0);
560:      while((get_timer0()<1954)&&(!kbhit())); // waits
100ms for a byte
```

```
561:      if(!kbhit())
562:      {
563:        i++;
564:        if(i>2) // more than two errors
565:        {
566:          return 0; // return false
567:        }
568:        goto again_2; // try again
569:      }
570:
571:      air = getc(); // receives air velocity data
572:
573:
574:      // in case of error during reception
575:      set_timer0(0);
576:      while((get_timer0()<1954)&&(!kbhit())); // waits
100ms for a byte
577:      if(!kbhit())
578:      {
579:        i++;
580:        if(i>2) // more than two errors
581:        {
582:          return 0; // return false
583:        }
584:        goto again_2; // try again
585:      }
586:
587:      check = getc(); // receives air velocity
redundancy
588:
589:      if(air!=check) // air velocity data and the
redundacy do not match
590:      {
591:        i++;
592:        if(i>2) // more than two errors
593:        {
594:          return 0; // return false
595:        }
596:        goto again_2; // try again
597:      }
598:      write_eeprom(2,air); // data save
599:      av = air*0.1; // calculates of real air velocity
600:      sprintf(line,"%01.1f m/s",av); // passes air
velocity to string
601:      putsHalfDownLCD(line); // prints the string at LCD
602:      while(BusyHalfDownLCD());
603:    }
604:    else // if av_module not found
605:    {
606:      WriteCmdHalfUpLCD(CURSOR_ON&BLINK_ON); // displays
blinking cursor
607:      while(BusyHalfUpLCD());
```

```
608:      output_high(STAGE_1); // activates vetilation
first stage
609:      if(av_levels) // if the system was calibrated
610:      {
611:        sprintf(line,"%02d",read_eeprom(0)); // reads
1st stage data
612:
613:        // displays 1st stage air velocity value at LCD
614:        WriteDataHalfUpLCD(line[0]);
615:        while(BusyHalfUpLCD());
616:        WriteDataHalfUpLCD('.');
617:        while(BusyHalfUpLCD());
618:        WriteDataHalfUpLCD(line[1]);
619:        while(BusyHalfUpLCD());
620:
621:        // puts cursor at data entry position
622:        WriteCmdHalfUpLCD(SHIFT_CUR_LEFT);
623:        while(BusyHalfUpLCD());
624:        WriteCmdHalfUpLCD(SHIFT_CUR_LEFT);
625:        while(BusyHalfUpLCD());
626:        WriteCmdHalfUpLCD(SHIFT_CUR_LEFT);
627:        while(BusyHalfUpLCD());
628:      }
629:      else
630:      {
631:        line[0] = null;
632:        line[1] = null;
633:        line[2] = null;
634:      }
635:      i = 0;
636:      while(i<2)
637:      {
638:        key = keypad(); // receives pressed key
639:        if((key!='*')&&(key!='#')) // if key is a number
640:        {
641:          line[i] = key;
642:          WriteDataHalfUpLCD(key); // displays pressed
number at LCD
643:          while(BusyHalfUpLCD());
644:          if (!i) // if it is the first character
645:          {
646:            WriteDataHalfUpLCD('.'); // displays float
point
647:            while(BusyHalfUpLCD());
648:            i++;
649:          }
650:          else // otherwise (i!=0)
651:          {
652:            WriteCmdHalfUpLCD(SHIFT_CUR_LEFT); // shifts
cursor backward
653:            while(BusyHalfUpLCD());
654:          }
655:        }
```

```
656:        else if((key=='*')&&i) // if * is pressed and
i>0
657:        {
658:          WriteCmdHalfUpLCD(SHIFT_CUR_LEFT); // shifts
cursor backward
659:          while(BusyHalfUpLCD());
660:          WriteCmdHalfUpLCD(SHIFT_CUR_LEFT); // shifts
cursor backward
661:          while(BusyHalfUpLCD());
662:          i--;
663:        }
664:        else if((key=='#')&&(line[i])&&(!i)) // if
there's a number in line[0]
665:        { // (i==0)
666:          WriteCmdHalfUpLCD(SHIFT_CUR_RIGHT); // shifts
cursor forward
667:          while(BusyHalfUpLCD());
668:          WriteCmdHalfUpLCD(SHIFT_CUR_RIGHT); // shifts
cursor forward
669:          while(BusyHalfUpLCD());
670:          i++;
671:        }
672:        else if((key=='#')&&(line[i])&&i) // if there's
a number in line[1]
673:        { // (i==1)
674:          write_eeprom(0,atoi(line)); // receives
calibration information
675:          WriteCmdHalfUpLCD(CURSOR_OFF&BLINK_OFF); //
cursor and blink off
676:          i++;
677:          while(BusyHalfUpLCD());
678:        }
679:      }
680:
681:      WriteCmdHalfUpLCD(HOME); // returns cursor to
initial position
682:      while(BusyHalfUpLCD());
683:      SetDDRamAddrHalfUp(0x29); // changes cursor
position
684:      while(BusyHalfUpLCD());
685:      WriteCmdHalfUpLCD(SHIFT_CUR_RIGHT); // shifts
cursor right
686:      while(BusyHalfUpLCD());
687:      WriteDataHalfUpLCD(' '); // dismarks first stage
688:      while(BusyHalfUpLCD());
689:
690:      WriteCmdHalfDownLCD(HOME); // returns cursor to
initial position
691:      while(BusyHalfDownLCD());
692:      SetDDRamAddrHalfDown(0x00); // makes sure it is 0
address
693:      while(BusyHalfDownLCD());
```

```
694:      WriteCmdHalfDownLCD(SHIFT_CUR_RIGHT); // shifts
cursor right
695:      while(BusyHalfDownLCD());
696:      WriteDataHalfDownLCD(0xA5); // marks second stage
697:      while(BusyHalfDownLCD());
698:      for(i=0;i<13;i++) // shifts cursor to the entry
point
699:      {
700:        WriteCmdHalfDownLCD(SHIFT_CUR_RIGHT);
701:        while(BusyHalfDownLCD());
702:      }
703:      WriteCmdHalfDownLCD(CURSOR_ON&BLINK_ON); //
displays blinking cursor
704:      while(BusyHalfDownLCD());
705:
706:      output_high(STAGE_2); // activates ventilation
second stage
707:
708:      if(av_levels) // if the system was calibrated
709:      {
710:        sprintf(line,"%02d",read_eeprom(1)); // reads
2nd stage data
711:
712:        // displays 2nd stage air velocity value at LCD
713:        WriteDataHalfDownLCD(line[0]);
714:        while(BusyHalfDownLCD());
715:        WriteDataHalfDownLCD('.');
716:        while(BusyHalfDownLCD());
717:        WriteDataHalfDownLCD(line[1]);
718:        while(BusyHalfDownLCD());
719:
720:        // puts cursor at data entry position
721:        WriteCmdHalfDownLCD(SHIFT_CUR_LEFT);
722:        while(BusyHalfDownLCD());
723:        WriteCmdHalfDownLCD(SHIFT_CUR_LEFT);
724:        while(BusyHalfDownLCD());
725:        WriteCmdHalfDownLCD(SHIFT_CUR_LEFT);
726:        while(BusyHalfDownLCD());
727:      }
728:      else
729:      {
730:        line[0] = null;
731:        line[1] = null;
732:        line[2] = null;
733:      }
734:
735:      i = 0;
736:      while(i<2)
737:      {
738:        key = keypad(); // receives pressed key
739:        if((key!='*')&&(key!='#')) // if key is a number
740:        {
741:          line[i] = key;
```

```
742:          WriteDataHalfDownLCD(key); // displays pressed
number at LCD
743:          while(BusyHalfDownLCD());
744:          if (!i) // if it is the first character
745:          {
746:            WriteDataHalfDownLCD('.'); // displays float
point
747:            while(BusyHalfDownLCD());
748:            i++;
749:          }
750:          else // otherwise (i!=0)
751:          {
752:            WriteCmdHalfDownLCD(SHIFT_CUR_LEFT); //
shifts cursor backward
753:            while(BusyHalfDownLCD());
754:          }
755:        }
756:        else if((key=='*')&&i) // if * is pressed and
i>0
757:        {
758:          WriteCmdHalfDownLCD(SHIFT_CUR_LEFT); // shifts
cursor backward
759:          while(BusyHalfDownLCD());
760:          WriteCmdHalfDownLCD(SHIFT_CUR_LEFT); // shifts
cursor backward
761:          while(BusyHalfDownLCD());
762:          i--;
763:        }
764:        else if((key=='#')&&(line[i])&&(!i)) // if
there's a number in line[0]
765:        { // (i==0)
766:          WriteCmdHalfDownLCD(SHIFT_CUR_RIGHT); //
shifts cursor forward
767:          while(BusyHalfDownLCD());
768:          WriteCmdHalfDownLCD(SHIFT_CUR_RIGHT); //
shifts cursor forward
769:          while(BusyHalfDownLCD());
770:          i++;
771:        }
772:        else if((key=='#')&&(line[i])&&i) // if there's
a number in line[1]
773:        { // (i==1)
774:          write_eeprom(1,atoi(line)); // receives
calibration information
775:          WriteCmdHalfDownLCD(CURSOR_OFF&BLINK_OFF); //
cursor and blink off
776:          i++;
777:          while(BusyHalfDownLCD());
778:        }
779:      }
780:
781:      WriteCmdHalfDownLCD(HOME); // returns cursor to
initial position
```

```
782:      while(BusyHalfDownLCD());
783:      SetDDRamAddrHalfDown(0x00); // makes sure it is 0
address
784:      while(BusyHalfDownLCD());
785:      WriteCmdHalfDownLCD(SHIFT_CUR_RIGHT); // shifts
cursor right
786:      while(BusyHalfDownLCD());
787:      WriteDataHalfDownLCD(' '); // dismarks second
stage
788:      while(BusyHalfDownLCD());
789:
790:      WriteCmdHalfDownLCD(HOME); // returns cursor to
initial position
791:      while(BusyHalfDownLCD());
792:      SetDDRamAddrHalfDown(0x29); // changes cursor
position
793:      while(BusyHalfDownLCD());
794:      WriteCmdHalfDownLCD(SHIFT_CUR_RIGHT); // shifts
cursor right
795:      while(BusyHalfDownLCD());
796:      WriteDataHalfDownLCD(0xA5); // marks third stage
797:      while(BusyHalfDownLCD());
798:
799:      for(i=0;i<13;i++) // shifts cursor to the entry
point
800:      {
801:        WriteCmdHalfDownLCD(SHIFT_CUR_RIGHT);
802:        while(BusyHalfDownLCD());
803:      }
804:      WriteCmdHalfDownLCD(CURSOR_ON&BLINK_ON); //
display blinking cursor
805:      while(BusyHalfDownLCD());
806:
807:      output_high(STAGE_3); // activates ventilation
third stage
808:
809:      if(av_levels) // if the system was calibrated
810:      {
811:        // displays 3rd stage air velocity value at LCD
812:        sprintf(line,"%02d",read_eeprom(2));
813:        WriteDataHalfDownLCD(line[0]);
814:        while(BusyHalfDownLCD());
815:        WriteDataHalfDownLCD('.');
816:        while(BusyHalfDownLCD());
817:        WriteDataHalfDownLCD(line[1]);
818:        while(BusyHalfDownLCD());
819:
820:        // puts cursor at data entry position
821:        WriteCmdHalfDownLCD(SHIFT_CUR_LEFT);
822:        while(BusyHalfDownLCD());
823:        WriteCmdHalfDownLCD(SHIFT_CUR_LEFT);
824:        while(BusyHalfDownLCD());
825:        WriteCmdHalfDownLCD(SHIFT_CUR_LEFT);
```

```
826:       while(BusyHalfDownLCD());
827:     }
828:     else
829:     {
830:       line[0] = null;
831:       line[1] = null;
832:       line[2] = null;
833:     }
834:
835:     i = 0;
836:     while(i<2)
837:     {
838:       key = keypad(); // receives pressed key
839:       if((key!='*')&&(key!='#')) // if key is a number
840:       {
841:         line[i] = key;
842:         WriteDataHalfDownLCD(key); // displays pressed
number at LCD
843:         while(BusyHalfDownLCD());
844:         if (!i) // if it is the first character
845:         {
846:           WriteDataHalfDownLCD('.'); // displays float
point
847:           while(BusyHalfDownLCD());
848:           i++;
849:         }
850:         else // otherwise (i!=0)
851:         {
852:           WriteCmdHalfDownLCD(SHIFT_CUR_LEFT); //
shifts cursor backward
853:           while(BusyHalfDownLCD());
854:         }
855:       }
856:       else if((key=='*')&&i) // if * is pressed and
i>0
857:       {
858:         WriteCmdHalfDownLCD(SHIFT_CUR_LEFT); // shifts
cursor backward
859:         while(BusyHalfDownLCD());
860:         WriteCmdHalfDownLCD(SHIFT_CUR_LEFT); // shifts
cursor backward
861:         while(BusyHalfDownLCD());
862:         i--;
863:       }
864:       else if((key=='#')&&(line[i])&&(!i)) // if
there's a number in line[0]
865:       { // (i==0)
866:         WriteCmdHalfDownLCD(SHIFT_CUR_RIGHT); //
shifts cursor forward
867:         while(BusyHalfDownLCD());
868:         WriteCmdHalfDownLCD(SHIFT_CUR_RIGHT); //
shifts cursor forward
869:         while(BusyHalfDownLCD());
```

```
870:         i++;
871:       }
872:       else if((key=='#')&&(line[i])&&i) // if there's
a number in line[1]
873:       { // (i==1)
874:         write_eeprom(2,atoi(line)); // receives
calibration information
875:         WriteCmdHalfDownLCD(CURSOR_OFF&BLINK_OFF); //
cursor and blink off
876:         i++;
877:         while(BusyHalfDownLCD());
878:       }
879:     }
880:   }
881:
882:   //deactivates ventilation stages
883:   output_low(STAGE_1);
884:   output_low(STAGE_2);
885:   output_low(STAGE_3);
886:
887:   if((read_eeprom(0)<read_eeprom(1))&
(read_eeprom(1)<read_eeprom(2))) // if calibrate
888:   {
889:     av_levels = 1; // successful
890:   }
891:   else // if an error occur
892:   {
893:     av_levels = 0; // fails
894:     goto again; // try again
895:   }
896:   if(av_levels&&(!av_module)) // if no errors and
sensor not found
897:   {
898:     WriteCmdHalfUpLCD(ERASE); // cleans LCD half up
899:     WriteCmdHalfDownLCD(ERASE); // cleans LCD half
down
900:     strcpy(line,"Ventilation Stages"); // sets string
(line)
901:     while(BusyHalfDownLCD());
902:     while(BusyHalfUpLCD());
903:     for(i=0;i<12;i++) // adjust cursor position
904:     {
905:       WriteCmdHalfUpLCD(SHIFT_CUR_RIGHT);
906:       while(BusyHalfUpLCD());
907:     }
908:     putsHalfUpLCD(line); // prints the previous string
909:     while(BusyHalfUpLCD());
910:     strcpy(line," 1st 2nd 3rd"); // sets string (line)
911:     WriteCmdHalfUpLCD(HOME); // returns cursor to
initial position
912:     while(BusyHalfUpLCD());
913:     SetDDRamAddrHalfUp(0x29); // sets cursor position
914:     while(BusyHalfUpLCD());
```

```
915:      putsHalfUpLCD(line); // prints the previous string
916:
917:      av = read_eeprom(0)*0.1; // read the 1st stage
stored air velocity
918:      sprintf(line," %01.1f m/s",av); // turns air
velocity value into a string
919:      putsHalfDownLCD(line); // prints the string
920:      while(BusyHalfDownLCD());
921:      av = read_eeprom(1)*0.1; // read the 2nd stage
stored air velocity
922:      sprintf(line," %01.1f m/s",av); // turns air
velocity value into a string
923:      putsHalfDownLCD(line); // prints the string
924:      av = read_eeprom(2)*0.1; // read the 3rd stage
stored air velocity
925:      sprintf(line," %01.1f m/s",av); // turns air
velocity value into a string
926:      putsHalfDownLCD(line); // prints the string
927:      while(BusyHalfDownLCD());
928:
929:      strcpy(line," [*] Cancel [#] Confirm"); // sets
string (line)
930:      WriteCmdHalfDownLCD(HOME); // returns cursor to
initial position
931:      while(BusyHalfDownLCD());
932:      SetDDRamAddrHalfDown(0x29); // changes cursor
position
933:      while(BusyHalfDownLCD());
934:      putsHalfDownLCD(line); // displays the string
(line) at LCD
935:      while(BusyHalfDownLCD());
936:      key = 0;
937:      while((key!='*')&&(key!='#')) // verifies the
pressed key
938:      {
939:        key = keypad();
940:        if(key == '*') // if key is *, do again
941:        {
942:          goto again;
943:        }
944:      }
945:    }
946:    restart_wdt();
947:    return 1; // no error
948: }
949:
950: //-----------------------------------------------------
951: boolean data_entry(boolean av_levels)
952: //-----------------------------------------------------
953: // informes, directily, the values of air velocity to
the system
954: // av_levels: if true there is valid values
955: {
```

```
956:    // variables
957:    char key; // pressed key
958:    float av; // air velocity
959:    int i; // index or auxiliar variable
960:    char line[41]; // corresponds to one LCD line
961:
962:    again: // in case of error or user choice restart
here
963:
964:    restart_wdt();
965:
966:    //clear LCD
967:    WriteCmdHalfUpLCD(ERASE);
968:    WriteCmdHalfDownLCD(ERASE);
969:    while(BusyHalfDownLCD());
970:    while(BusyHalfUpLCD());
971:
972:    WriteCmdHalfUpLCD(SHIFT_CUR_RIGHT); //shifts cursor
right
973:    while(BusyHalfUpLCD());
974:    strcpy(line,"Stages of ventilation and air
velocity"); // sets string (line)
975:    putsHalfUpLCD(line); // prints line at LCD
976:    while(BusyHalfUpLCD());
977:    WriteCmdHalfUpLCD(HOME); // sends cursor to initial
position
978:    sprintf(line,"[%c] 1st stage: ",0xA5); // sets
string (line)
979:    while(BusyHalfUpLCD());
980:    SetDDRamAddrHalfUp(0x29); // changes cursor position
981:    while(BusyHalfUpLCD());
982:    putsHalfUpLCD(line); // prints line at LCD
983:    while(BusyHalfUpLCD());
984:
985:    strcpy(line,"[ ] 2sd stage:"); // sets string (line)
986:    putsHalfDownLCD(line); // prints line at LCD
987:    while(BusyHalfDownLCD());
988:
989:    WriteCmdHalfDownLCD(HOME); // sends cursor to
initial position
990:    strcpy(line,"[ ] 3rd stage:"); // sets string (line)
991:    while(BusyHalfDownLCD());
992:    SetDDRamAddrHalfDown(0x29); // changes cursor
position
993:    while(BusyHalfDownLCD());
994:    putsHalfDownLCD(line); // prints line at LCD
995:    while(BusyHalfDownLCD());
996:
997:    WriteCmdHalfUpLCD(CURSOR_ON&BLINK_ON); // displays
blinking cursor
998:    while(BusyHalfUpLCD());
999:
```

```
1000:   if(av_levels) // ventilation stages was set at
least one time
1001:   {
1002:     sprintf(line,"%02d",read_eeprom(0)); // reads 1st
stage data
1003:
1004:     // displays 1st stage air velocity value at LCD
1005:     WriteDataHalfUpLCD(line[0]);
1006:     while(BusyHalfUpLCD());
1007:     WriteDataHalfUpLCD('.');
1008:     while(BusyHalfUpLCD());
1009:     WriteDataHalfUpLCD(line[1]);
1010:     while(BusyHalfUpLCD());
1011:
1012:     // puts cursor at data entry position
1013:     WriteCmdHalfUpLCD(SHIFT_CUR_LEFT);
1014:     while(BusyHalfUpLCD());
1015:     WriteCmdHalfUpLCD(SHIFT_CUR_LEFT);
1016:     while(BusyHalfUpLCD());
1017:     WriteCmdHalfUpLCD(SHIFT_CUR_LEFT);
1018:     while(BusyHalfUpLCD());
1019:   }
1020:   else
1021:   {
1022:     line[0] = null;
1023:     line[1] = null;
1024:     line[2] = null;
1025:   }
1026:   i = 0;
1027:   while(i<2)
1028:   {
1029:     key = keypad(); // receives pressed key
1030:     if((key!='*')&&(key!='#')) // if key is a number
1031:     {
1032:       line[i] = key;
1033:       WriteDataHalfUpLCD(key); // displays pressed
number at LCD
1034:       while(BusyHalfUpLCD());
1035:       if (!i) // if it is the first character
1036:       {
1037:         WriteDataHalfUpLCD('.'); // displays float
point
1038:         while(BusyHalfUpLCD());
1039:         i++;
1040:       }
1041:       else // otherwise (i!=0)
1042:       {
1043:         WriteCmdHalfUpLCD(SHIFT_CUR_LEFT); // shifts
cursor backward
1044:         while(BusyHalfUpLCD());
1045:       }
1046:     }
```

```
1047:     else if((key=='*')&&i) // if * is pressed and
i==1
1048:     {
1049:       WriteCmdHalfUpLCD(SHIFT_CUR_LEFT); // shifts
cursor backward
1050:       while(BusyHalfUpLCD());
1051:       WriteCmdHalfUpLCD(SHIFT_CUR_LEFT); // shifts
cursor backward
1052:       while(BusyHalfUpLCD());
1053:       i--;
1054:     }
1055:     else if((key=='#')&&(line[i])&&(!i)) // if
there's a number in line[0] and
1056:     { // (i==0)
1057:       WriteCmdHalfUpLCD(SHIFT_CUR_RIGHT); // shifts
cursor forward
1058:       while(BusyHalfUpLCD());
1059:       WriteCmdHalfUpLCD(SHIFT_CUR_RIGHT); // shifts
cursor forward
1060:       while(BusyHalfUpLCD());
1061:       i++;
1062:     }
1063:     else if((key=='#')&&(line[i])&&i) // if there's a
number in line[1] and
1064:     { // (i==1)
1065:       write_eeprom(0,atoi(line)); // receives data
information
1066:       WriteCmdHalfUpLCD(CURSOR_OFF&BLINK_OFF); //
cursor and blink off
1067:       i++;
1068:       while(BusyHalfUpLCD());
1069:     }
1070:   }
1071:
1072:   WriteCmdHalfUpLCD(HOME); // returns cursor to
initial position
1073:   while(BusyHalfUpLCD());
1074:   SetDDRamAddrHalfUp(0x29); // changes cursor
position
1075:   while(BusyHalfUpLCD());
1076:   WriteCmdHalfUpLCD(SHIFT_CUR_RIGHT); // shifts
cursor right
1077:   while(BusyHalfUpLCD());
1078:   WriteDataHalfUpLCD(' '); // dismarks 1st stage
1079:
1080:   WriteCmdHalfDownLCD(HOME); // returns cursor to
initial position
1081:   while(BusyHalfDownLCD());
1082:   SetDDRamAddrHalfDown(0x00); // makes sure it is 0
address
1083:   while(BusyHalfDownLCD());
1084:   WriteCmdHalfDownLCD(SHIFT_CUR_RIGHT); // shifts
cursor right
```

```
1085:    while(BusyHalfDownLCD());
1086:    WriteDataHalfDownLCD(0xA5); // marks 2nd stage
1087:    while(BusyHalfDownLCD());
1088:
1089:    for(i=0;i<13;i++) // shifts cursor to the entry
point
1090:    {
1091:      WriteCmdHalfDownLCD(SHIFT_CUR_RIGHT);
1092:      while(BusyHalfDownLCD());
1093:    }
1094:    WriteCmdHalfDownLCD(CURSOR_ON&BLINK_ON); //
displays blinking cursor
1095:    while(BusyHalfDownLCD());
1096:
1097:    if(av_levels) // ventilation stages was set at
least one time
1098:    {
1099:      sprintf(line,"%02d",read_eeprom(1)); // reads 2nd
stage data
1100:
1101:      // displays 2nd stage air velocity value at LCD
1102:      WriteDataHalfDownLCD(line[0]);
1103:      while(BusyHalfDownLCD());
1104:      WriteDataHalfDownLCD('.');
1105:      while(BusyHalfDownLCD());
1106:      WriteDataHalfDownLCD(line[1]);
1107:      while(BusyHalfDownLCD());
1108:
1109:      // puts cursor at data entry position
1110:      WriteCmdHalfDownLCD(SHIFT_CUR_LEFT);
1111:      while(BusyHalfDownLCD());
1112:      WriteCmdHalfDownLCD(SHIFT_CUR_LEFT);
1113:      while(BusyHalfDownLCD());
1114:      WriteCmdHalfDownLCD(SHIFT_CUR_LEFT);
1115:      while(BusyHalfDownLCD());
1116:    }
1117:    else
1118:    {
1119:      line[0] = null;
1120:      line[1] = null;
1121:      line[2] = null;
1122:    }
1123:
1124:    i = 0;
1125:    while(i<2)
1126:    {
1127:      key = keypad(); // receives pressed key
1128:      if((key!='*')&&(key!='#')) // if key is a number
1129:      {
1130:        line[i] = key;
1131:        WriteDataHalfDownLCD(key); // displays pressed
number at LCD
1132:        while(BusyHalfDownLCD());
```

```
1133:       if (!i) // if it is the first character
1134:       {
1135:         WriteDataHalfDownLCD('.'); // displays float
point
1136:         while(BusyHalfDownLCD());
1137:         i++;
1138:       }
1139:       else // otherwise (i!=0)
1140:       {
1141:         WriteCmdHalfDownLCD(SHIFT_CUR_LEFT); //
shifts cursor backward
1142:         while(BusyHalfDownLCD());
1143:       }
1144:     }
1145:     else if((key=='*')&&i) // if * is pressed and
i==1
1146:     {
1147:       WriteCmdHalfDownLCD(SHIFT_CUR_LEFT); // shifts
cursor backward
1148:       while(BusyHalfDownLCD());
1149:       WriteCmdHalfDownLCD(SHIFT_CUR_LEFT); // shifts
cursor backward
1150:       while(BusyHalfDownLCD());
1151:       i--;
1152:     }
1153:     else if((key=='#')&&(line[i])&&(!i)) // if
there's a number in line[0] and
1154:     { // (i==0)
1155:       WriteCmdHalfDownLCD(SHIFT_CUR_RIGHT); // shifts
cursor forward
1156:       while(BusyHalfDownLCD());
1157:       WriteCmdHalfDownLCD(SHIFT_CUR_RIGHT); // shifts
cursor forward
1158:       while(BusyHalfDownLCD());
1159:       i++;
1160:     }
1161:     else if((key=='#')&&(line[i])&&i) // if there's a
number in line[1] and
1162:     { // (i==1)
1163:       write_eeprom(1,atoi(line)); // receives data
information
1164:       WriteCmdHalfDownLCD(CURSOR_OFF&BLINK_OFF); //
cursor and blink off
1165:       i++;
1166:       while(BusyHalfDownLCD());
1167:     }
1168:   }
1169:
1170:   WriteCmdHalfDownLCD(HOME); // returns cursor to
initial position
1171:   while(BusyHalfDownLCD());
1172:   SetDDRamAddrHalfDown(0x00); // makes sure it is 0
address
```

```
1173:    while(BusyHalfDownLCD());
1174:    WriteCmdHalfDownLCD(SHIFT_CUR_RIGHT); // shifts
cursor right
1175:    while(BusyHalfDownLCD());
1176:    WriteDataHalfDownLCD(' '); // dismarks 2nd stage
1177:    while(BusyHalfDownLCD());
1178:
1179:    WriteCmdHalfDownLCD(HOME); // returns cursor to
initial position
1180:    while(BusyHalfDownLCD());
1181:    SetDDRamAddrHalfDown(0x29); // changes cursor
position
1182:    while(BusyHalfDownLCD());
1183:    WriteCmdHalfDownLCD(SHIFT_CUR_RIGHT); // shifts
cursor right
1184:    while(BusyHalfDownLCD());
1185:    WriteDataHalfDownLCD(0xA5); // marks 3rd stage
1186:    while(BusyHalfDownLCD());
1187:
1188:    for(i=0;i<13;i++) // shifts cursor to the entry
point
1189:    {
1190:      WriteCmdHalfDownLCD(SHIFT_CUR_RIGHT);
1191:      while(BusyHalfDownLCD());
1192:    }
1193:    WriteCmdHalfDownLCD(CURSOR_ON&BLINK_ON); //
displays blinking cursor
1194:    while(BusyHalfDownLCD());
1195:
1196:    if(av_levels) // ventilation stages was set at
least one time
1197:    {
1198:      sprintf(line,"%02d",read_eeprom(2)); // reads 3rd
stage data
1199:
1200:      // displays 3rd stage air velocity value at LCD
1201:      WriteDataHalfDownLCD(line[0]);
1202:      while(BusyHalfDownLCD());
1203:      WriteDataHalfDownLCD('.');
1204:      while(BusyHalfDownLCD());
1205:      WriteDataHalfDownLCD(line[1]);
1206:      while(BusyHalfDownLCD());
1207:
1208:      // puts cursor at data entry position
1209:      WriteCmdHalfDownLCD(SHIFT_CUR_LEFT);
1210:      while(BusyHalfDownLCD());
1211:      WriteCmdHalfDownLCD(SHIFT_CUR_LEFT);
1212:      while(BusyHalfDownLCD());
1213:      WriteCmdHalfDownLCD(SHIFT_CUR_LEFT);
1214:      while(BusyHalfDownLCD());
1215:    }
1216:    else
1217:    {
```

```
1218:     line[0] = null;
1219:     line[1] = null;
1220:     line[2] = null;
1221:   }
1222:
1223:   i = 0;
1224:   while(i<2)
1225:   {
1226:     key = keypad(); // receives pressed key
1227:     if((key!='*')&&(key!='#')) // if key is a number
1228:     {
1229:       line[i] = key;
1230:       WriteDataHalfDownLCD(key); // displays pressed
number at LCD
1231:       while(BusyHalfDownLCD());
1232:       if (!i) // if it is the first character
1233:       {
1234:         WriteDataHalfDownLCD('.'); // displays float
point
1235:         while(BusyHalfDownLCD());
1236:         i++;
1237:       }
1238:       else // otherwise (i!=0)
1239:       {
1240:         WriteCmdHalfDownLCD(SHIFT_CUR_LEFT); //
shifts cursor backward
1241:         while(BusyHalfDownLCD());
1242:       }
1243:     }
1244:     else if((key=='*')&&i) // if * is pressed and
i==1
1245:     {
1246:       WriteCmdHalfDownLCD(SHIFT_CUR_LEFT); // shifts
cursor backward
1247:       while(BusyHalfDownLCD());
1248:       WriteCmdHalfDownLCD(SHIFT_CUR_LEFT); // shifts
cursor backward
1249:       while(BusyHalfDownLCD());
1250:       i--;
1251:     }
1252:     else if((key=='#')&&(line[i])&&(!i)) // if
there's a number in line[0] and
1253:     { // (i==0)
1254:       WriteCmdHalfDownLCD(SHIFT_CUR_RIGHT); // shifts
cursor forward
1255:       while(BusyHalfDownLCD());
1256:       WriteCmdHalfDownLCD(SHIFT_CUR_RIGHT); // shifts
cursor forward
1257:       while(BusyHalfDownLCD());
1258:       i++;
1259:     }
1260:     else if((key=='#')&&(line[i])&&i) // if there's a
number in line[1] and
```

```
1261:      { // (i==1)
1262:        write_eeprom(2,atoi(line)); // receives data
information
1263:        WriteCmdHalfDownLCD(CURSOR_OFF&BLINK_OFF); //
cursor and blink off
1264:        i++;
1265:        while(BusyHalfDownLCD());
1266:      }
1267:    }
1268:
1269:    // if levels are correctily ajusted
1270:    if((read_eeprom(0)<read_eeprom(1))&
(read_eeprom(1)<read_eeprom(2)))
1271:    {
1272:      av_levels = 1; // stage data ok
1273:
1274:      // clean LCD
1275:      WriteCmdHalfUpLCD(ERASE);
1276:      WriteCmdHalfDownLCD(ERASE);
1277:      while(BusyHalfDownLCD());
1278:      while(BusyHalfUpLCD());
1279:
1280:      strcpy(line,"Ventilation Stages"); // sets line
new value
1281:
1282:      for(i=0;i<12;i++) // adjusts cursor position
1283:      {
1284:        WriteCmdHalfUpLCD(SHIFT_CUR_RIGHT);
1285:        while(BusyHalfUpLCD());
1286:      }
1287:      putsHalfUpLCD(line); // shows line at LCD
1288:      while(BusyHalfUpLCD());
1289:      strcpy(line," 1st 2nd 3rd"); // sets line new
value
1290:      WriteCmdHalfUpLCD(HOME); // returns cursor to
initial position
1291:      while(BusyHalfUpLCD());
1292:      SetDDRamAddrHalfUp(0x29); // changes cursor
position
1293:      while(BusyHalfUpLCD());
1294:      putsHalfUpLCD(line); // prints line at LCD
1295:      while(BusyHalfUpLCD());
1296:      av = read_eeprom(0)*0.1; // extracts air velocity
value from stored data
1297:      sprintf(line," %01.1f m/s",av); // puts air
velocity in a string
1298:      putsHalfDownLCD(line); // prints the string at
LCD
1299:      while(BusyHalfDownLCD());
1300:      av = read_eeprom(1)*0.1; // extracts air velocity
value from stored data
1301:      sprintf(line," %01.1f m/s",av); // puts air
velocity in a string
```

```
1302:     putsHalfDownLCD(line); // prints the string at
LCD
1303:     while(BusyHalfDownLCD());
1304:     av = read_eeprom(2)*0.1; // extracts air velocity
value from stored data
1305:     sprintf(line," %01.1f m/s",av); // puts air
velocity in a string
1306:     putsHalfDownLCD(line); // prints the string at
LCD
1307:     while(BusyHalfDownLCD());
1308:
1309:     strcpy(line,"[*] Cancel [#] Confirm"); // choice
informantion at "
1310:     WriteCmdHalfDownLCD(HOME); // returns cursor to
initial position
1311:     while(BusyHalfDownLCD());
1312:     SetDDRamAddrHalfDown(0x29); // changes cursor
position
1313:     while(BusyHalfDownLCD());
1314:
1315:     for(i=0;i<5;i++) // adjusts cursor position
1316:     {
1317:       WriteCmdHalfDownLCD(SHIFT_CUR_RIGHT);
1318:       while(BusyHalfDownLCD());
1319:     }
1320:
1321:     putsHalfDownLCD(line); // prints choice
information at LCD
1322:     while(BusyHalfDownLCD());
1323:     key = 0;
1324:     while((key!='*')&&(key!='#')) // verifies the
pressed key
1325:     {
1326:       key = keypad(); // catchs the pressed key
1327:       if(key == '*') // if key is *, do again
1328:       {
1329:         goto again;
1330:       }
1331:     }
1332:   }
1333:   else // if an error occur
1334:   {
1335:     av_levels = 0; // fails
1336:     goto again; // try again
1337:   }
1338:   restart_wdt();
1339:   return 1;
1340: }
1341:
1342:
1343: //-----------------------------------------------------
1344: void main()
1345: //-----------------------------------------------------
```

```
1346: {
1347:    // air velocity variables
1348:    boolean av_module, av_levels, av_count,
curtain_module;
1349:    unsigned int curtain_state;
1350:
1351:    // control variables and environment variables
1352:    float result,humidity;
1353:    int stage,time,av;
1354:    float environment_0[3], environment_1[3],
environment_2[3], rectal[3];
1355:
1356:    unsigned long int timer_aux;
1357:
1358:    // temperature and humidity variables
1359:    unsigned long int temp,humi;
1360:
1361:    // reception variable
1362:    unsigned int hi,lo,checksum,opt,i,error;
1363:
1364:    char line[41];
1365:
1366:    //-------------------------------------------------
1367:    // DISPLAY INITIALIZATION
1368:    //-------------------------------------------------
1369:
1370:    OpenLCD(EIGHT_BIT & LINES_5X7);
1371:    WriteCmdHalfUpLCD(CURSOR_OFF&BLINK_OFF);
1372:    WriteCmdHalfDownLCD(CURSOR_OFF&BLINK_OFF);
1373:    while(BusyHalfDownLCD());
1374:    while(BusyHalfUpLCD());
1375:
1376:    //-------------------------------------------------
1377:    //-------------------------------------------------
1378:
1379:    setup_wdt(WDT_ON); // watch dog timer
initialization
1380:    restart_wdt();
1381:    setup_timer_0(RTCC_INTERNAL|RTCC_DIV_256); // timer
initialization (1:256)
1382:
1383:    //-------------------------------------------------
1384:    // SYSTEM INITIALIZATION
1385:    //-------------------------------------------------
1386:
1387:    SetDDRamAddrHalfUp(0x07); // adjusts cursor
position to centralize the string
1388:    strcpy(line,"Environment Control System"); // puts
one string at "line"
1389:    putsHalfUpLCD(line); // prints line at LCD
1390:    while(BusyHalfUpLCD());
1391:    WriteCmdHalfUpLCD(HOME); // returns to initial
position
```

```
1392:    while(BusyHalfUpLCD());
1393:    SetDDRamAddrHalfUp(0x00); // makes sure that it is
0 address
1394:
1395:    begin:
1396:    SetDDRamAddrHalfDown(0x0D); // changes cursor
position
1397:    while(BusyHalfDownLCD());
1398:    strcpy(line,"Please wait"); // sets new string at
"line"
1399:    putsHalfDownLCD(line); // prints the string at LCD
1400:    while(BusyHalfDownLCD());
1401:
1402:    curtain_module = 0; // sets as false (curtain)
1403:
1404:    for(i=0;i<3;i++)
1405:    {
1406:      putc(0xFF); // code validation sequence
1407:      delay_us(20);
1408:      putc(CURTAIN); // request curtain state
1409:      set_timer0(0); // timer0 is set
1410:      WriteDataHalfDownLCD('.'); // shows that the
system is working
1411:      while(BusyHalfDownLCD());
1412:      restart_wdt();
1413:      while(get_timer0()<9766)
1414:      {
1415:        if(kbhit()) // if receive, there is a curtain
module
1416:        {
1417:          curtain_module = 1; // curtain module found
1418:          output_high(CUR_LED);
1419:          break;
1420:        }
1421:      }
1422:      restart_wdt();
1423:      if(curtain_module) // if curtain module found
1424:      {
1425:        //discharge usart
1426:        getc();
1427:
1428:        // preventing reception errors
1429:        set_timer0(0);
1430:        while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
1431:
1432:        if(!kbhit()) // if there is no data in the
USART
1433:        {
1434:          curtain_module = 0; // there is no curtain
module
1435:          continue; // finish actual iteration
1436:        }
```

```
1437:
1438:         getc();
1439:
1440:         // preventing reception errors
1441:         set_timer0(0);
1442:         while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
1443:
1444:         if(!kbhit()) // if there is no data in the
USART
1445:         {
1446:           curtain_module = 0; // there is no curtain
module
1447:           continue; // finish actual iteration
1448:         }
1449:
1450:         curtain_state = getc();
1451:         break;
1452:       }
1453:     }
1454:
1455:     restart_wdt();
1456:
1457:     WriteCmdHalfDownLCD(ERASE);
1458:     while(BusyHalfDownLCD());
1459:
1460:     av_search: // return point if one error occur or
user wants search
1461:
1462:     // in case of an error during calibration with
sensor (deactivates ventilation
1463:     output_low(STAGE_1); // deactivates ventilation
first stage
1464:     output_low(STAGE_2); // deactivates ventilation
second stage
1465:     output_low(STAGE_3); // deactivates ventilation
third stage
1466:
1467:     SetDDRamAddrHalfDown(0x0D); // changes cursor
position
1468:     while(BusyHalfDownLCD());
1469:     strcpy(line,"Please wait"); // sets new string at
"line"
1470:     putsHalfDownLCD(line); // prints the string at LCD
1471:     while(BusyHalfDownLCD());
1472:
1473:     av_module = 0; // sets as false (air sensor)
1474:
1475:     for(i=0;i<3;i++)
1476:     {
1477:       putc(0xFF); // validation sequence
1478:       delay_us(20);
1479:       putc(WIND);
```

```
1480:     set_timer0(0); // timer0 is set (1:256 prescale
value)
1481:     WriteDataHalfDownLCD('.'); // shows that the
system is working
1482:     while(BusyHalfDownLCD());
1483:     restart_wdt();
1484:     while(get_timer0()<9766)
1485:     {
1486:       if(kbhit()) // if receive, sensor is ok
1487:       {
1488:         av_module = 1; // air velocity sensor found
1489:         output_high(WIND_LED);
1490:         break;
1491:       }
1492:     }
1493:     restart_wdt();
1494:     if(av_module) // if sensor module found
1495:     {
1496:       //discharge usart
1497:       getc();
1498:
1499:       // preventing reception errors
1500:       set_timer0(0);
1501:       while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
1502:
1503:       if(!kbhit()) // if there is no data in the
USART
1504:       {
1505:         av_module = 0; // there is no air velocity
sensor
1506:         continue; // finish actual iteration
1507:       }
1508:
1509:       getc();
1510:
1511:       // preventing reception errors
1512:       set_timer0(0);
1513:       while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
1514:
1515:       if(!kbhit()) // if there is no data in the
USART
1516:       {
1517:         av_module = 0; // there is no air velocity
sensor
1518:         continue; // finish actual iteration
1519:       }
1520:
1521:       getc();
1522:
1523:       // preventing reception errors
1524:       set_timer0(0);
```

```
1525:       while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
1526:
1527:       if(!kbhit()) // if there is no data in the
USART
1528:       {
1529:         av_module = 0; // there is no air velocity
sensor
1530:         continue; // finish actual iteration
1531:       }
1532:
1533:       getc();
1534:       break;
1535:     }
1536:   }
1537:
1538:   restart_wdt();
1539:
1540:   av_levels = 0; // puts av_levels as false
1541:
1542:   // verifies if there is stored values of
ventilation stages
1543:   if((read_eeprom(0)<read_eeprom(1))&&
(read_eeprom(1)<read_eeprom(2)))
1544:   {
1545:     av_levels = 1; // puts av_levels as true
1546:   }
1547:
1548:   // cleans LCD screen
1549:   WriteCmdHalfUpLCD(ERASE);
1550:   WriteCmdHalfDownLCD(ERASE);
1551:   while(BusyHalfDownLCD());
1552:   while(BusyHalfUpLCD());
1553:
1554:   restart_wdt();
1555:
1556:   // if air velocity sensor was found
1557:   if(av_module)
1558:   {
1559:     SetDDRamAddrHalfUp(0x07); // adjust cursor
position
1560:     strcpy(line,"Air velocity sensor found!"); //
sets new string into "line"
1561:     while(BusyHalfUpLCD());
1562:     putsHalfUpLCD(line); // shows the string (line)
at LCD
1563:     while(BusyHalfUpLCD());
1564:
1565:     WriteCmdHalfUpLCD(HOME); // returns cursor to
initial position
1566:     while(BusyHalfUpLCD());
1567:     SetDDRamAddrHalfUp(0x29); // sets cursor position
1568:
```

```
1569:     // shows at LCD information about the importance
of calibration procedure
1570:     strcpy(line,"The system calibration is
recommended to"); // sets one string
1571:     while(BusyHalfUpLCD());
1572:     putsHalfUpLCD(line); // prints the string (line)
1573:     strcpy(line,"prevent sensor failures."); // sets
one string at "line"
1574:     putsHalfDownLCD(line); // prints the string
(line)
1575:     while(BusyHalfDownLCD());
1576:
1577:     // shows at LCD what to do if the user wants to
calibrate the system
1578:     WriteCmdHalfDownLCD(HOME); // returns cursor to
initial position
1579:     while(BusyHalfDownLCD());
1580:     SetDDRamAddrHalfDown(0x29); // sets cursor
position
1581:     strcpy(line," [Hit any key to calibrate the
system]"); // sets one string at
1582:     while(BusyHalfDownLCD());
1583:     putsHalfDownLCD(line); // prints the string
(line)
1584:
1585:     output_low(ROW_1);
1586:     output_low(ROW_2);
1587:     output_low(ROW_3);
1588:     output_low(ROW_4);
1589:     opt = 0;
1590:     for(i=0;i<5;i++) // waits five seconds for a key
1591:     {
1592:       set_timer0(0);
1593:       while((get_timer0()<19531 )&&(!opt))
1594:       {
1595:         if((!input(COLUMN_1))||(!input(COLUMN_2))||
(!input(COLUMN_3)))
1596:         {
1597:           opt = 1; // detection of a pressed key
1598:         }
1599:         restart_wdt();
1600:       }
1601:       if(opt)
1602:       {
1603:         while((curtain_module)&&
(curtain_state!=CLOSE_CURTAIN)) // close curtain
1604:         {
1605:           putc(0xFF);
1606:           delay_us(20);
1607:           putc(CLOSE_CURTAIN);
1608:           getc();
1609:           getc();
1610:           curtain_state = getc();
```

```
1611:          }
1612:          av_levels = calibrate(av_module,av_levels);
// calibrate result
1613:          if(!av_levels) // if calibration error
1614:          {
1615:            // cleans LCD sreen
1616:            WriteCmdHalfUpLCD(ERASE);
1617:            WriteCmdHalfDownLCD(ERASE);
1618:            while(BusyHalfDownLCD());
1619:            while(BusyHalfUpLCD());
1620:
1621:            SetDDRamAddrHalfUp(0x2C); // sets cursor
position
1622:            strcpy(line,"ERROR: sensor problem!"); //
sets one string at "line"
1623:            while(BusyHalfUpLCD());
1624:            putsHalfDownLCD(line); // prints the string
(line)
1625:            while(BusyHalfDownLCD());
1626:            restart_wdt();
1627:            goto av_search; // restart and search
sensor again
1628:          }
1629:          break;
1630:        }
1631:      }
1632:    }
1633:    else // sensor not found
1634:    {
1635:      SetDDRamAddrHalfUp(0x05); // adjusts cursor
position
1636:      strcpy(line,"Air velocity sensor not found!"); //
sets new string at "line"
1637:      while(BusyHalfUpLCD());
1638:      putsHalfUpLCD(line); // prints line at LCD
1639:      while(BusyHalfUpLCD());
1640:      WriteCmdHalfUpLCD(HOME); // returns cursor to
initial position
1641:      while(BusyHalfUpLCD());
1642:      SetDDRamAddrHalfUp(0x29); // changes cursor
position
1643:      while(BusyHalfUpLCD());
1644:      strcpy(line,"Choose the next step"); // sets new
string at "line"
1645:
1646:      for(i=0;i<10;i++) //adjusts cursor position
1647:      {
1648:        WriteCmdHalfUpLCD(SHIFT_CUR_RIGHT);
1649:        while(BusyHalfUpLCD());
1650:      }
1651:
1652:      putsHalfUpLCD(line); // prints line at LCD
1653:      while(BusyHalfUpLCD());
```

```
1654:     strcpy(line," [1] search again [2] calibrate");
// sets new string at
1655:     putsHalfDownLCD(line); // prints line at LCD
1656:     while(BusyHalfDownLCD());
1657:     WriteCmdHalfDownLCD(HOME); // returns cursor to
initial position
1658:     while(BusyHalfDownLCD());
1659:     SetDDRamAddrHalfDown(0x29); // changes cursor
position
1660:     while(BusyHalfDownLCD());
1661:     strcpy(line," [3] data entry"); // sets new
string at line
1662:     putsHalfDownLCD(line); // prints line at LCD
1663:     while(BusyHalfDownLCD());
1664:
1665:     restart_wdt();
1666:
1667:     if(av_levels) // if there are stored values of
ventilation
1668:     {
1669:       strcpy(line," [#] abandon"); // sets new string
at line
1670:       while(BusyHalfDownLCD());
1671:       putsHalfDownLCD(line); // prints line at LCD
1672:
1673:       opt = 0;
1674:       for(i=0;i<5;i++) // waits during 5 seconds one
choice is made
1675:       {
1676:         set_timer0(0);
1677:         output_low(ROW_1);
1678:         output_low(ROW_2);
1679:         output_low(ROW_3);
1680:         output_low(ROW_4);
1681:         while((get_timer0()<19531)&&(opt!='#')&&
(opt!='1')&&(opt!='2')&&(opt!='3'))
1682:         {
1683:           restart_wdt();
1684:           if((!input(COLUMN_1))||(!input(COLUMN_2))||
(!input(COLUMN_3))) // one
1685:           {
1686:             timer_aux = get_timer0(); // gets timer0
value
1687:             opt = keypad(); // gets pressed key
1688:             set_timer0((timer_aux+1)); // restore the
timer0 value with increment
1689:           }
1690:         }
1691:         if((opt!='#')&&(opt!='1')&&(opt!='2')&&
(opt!='3'))
1692:         {
1693:           opt = 0;
1694:         }
```

```
1695:           else
1696:           {
1697:             break;
1698:           }
1699:         }
1700:         switch (opt)
1701:         {
1702:           case '1': // case 1 - search for sensor again
1703:             WriteCmdHalfUpLCD(ERASE);
1704:             WriteCmdHalfDownLCD(ERASE);
1705:             while(BusyHalfDownLCD());
1706:             while(BusyHalfUpLCD());
1707:             goto av_search;
1708:             break;
1709:           case '2': // case 2 - calibrate the system
1710:             while((curtain_module)&&
(curtain_state!=CLOSE_CURTAIN)) // close curtain
1711:             {
1712:               putc(0xFF);
1713:               putc(CLOSE_CURTAIN);
1714:               getc();
1715:               getc();
1716:               curtain_state = getc();
1717:             }
1718:             av_levels = calibrate(av_module,av_levels);
1719:             break;
1720:           case '3': // case 3 - sets data values
directily
1721:             av_levels = data_entry(av_levels);
1722:             break;
1723:         }
1724:       }
1725:       else
1726:       {
1727:         opt = 0;
1728:         while((opt!='1')&(opt!='2')&(opt!='3')) //waits
the user choice
1729:         {
1730:           opt = keypad();
1731:         }
1732:         switch (opt)
1733:         {
1734:           case '1': // case 1 - search sensor again
1735:             WriteCmdHalfUpLCD(ERASE);
1736:             WriteCmdHalfDownLCD(ERASE);
1737:             while(BusyHalfDownLCD());
1738:             goto av_search;
1739:             break;
1740:           case '2': // case 2 - calibrate the system
1741:             av_levels = calibrate(av_module,av_levels);
1742:             break;
1743:           case '3': // case 3 - sets data values
directily
```

```
1744:            av_levels = data_entry(av_levels);
1745:            break;
1746:         }
1747:       }
1748:    }
1749:    restart_wdt();
1750:
1751:    //-------------------------------------------------
1752:    //-------------------------------------------------
1753:    stage = 0;
1754:    if(!curtain_module) //verifies if there is a
curtain module
1755:    {
1756:      output_high(STAGE_1); // in negative case
activates first level of ventilation
1757:      stage = 1;
1758:    }
1759:
1760:    // cleans LCD screen
1761:    WriteCmdHalfUpLCD(ERASE);
1762:    WriteCmdHalfDownLCD(ERASE);
1763:    while(BusyHalfDownLCD());
1764:    while(BusyHalfUpLCD());
1765:    // writes at LCD the label of the data that will be
displayed
1766:    strcpy(line,"Sector Temp. (");
1767:    putsHalfUpLCD(line);
1768:    while(BusyHalfUpLCD());
1769:    WriteDataHalfUpLCD(0xDF);
1770:    while(BusyHalfUpLCD());
1771:    strcpy(line,"C) RH (%) AV (m/s)");
1772:    putsHalfUpLCD(line);
1773:    while(BusyHalfUpLCD());
1774:    time = 10; // variable time initialization
1775:    av_count = 0;
1776:    while(1)
1777:    {
1778:      error = 0;
1779:      again_0:
1780:      av_module = 0; // sets as false (air sensor)
1781:      restart_wdt();
1782:      if(error<3) // if the number of errors is less
than 3
1783:      {
1784:        for(i=0;i<3;i++)
1785:        {
1786:          putc(0xFF);
1787:          delay_us(20);
1788:          putc(WIND);
1789:          set_timer0(0); // timer0 is set (1:256
prescale value)
1790:          while(get_timer0()<9766)
1791:          {
```

```
1792:           if(kbhit()) // if receive, sensor is ok
1793:           {
1794:             av_module = 1; // air velocity sensor
found
1795:             break;
1796:           }
1797:         }
1798:         restart_wdt();
1799:         if(av_module) // if sensor module found
1800:         {
1801:           //get air velocity;
1802:           getc();
1803:
1804:           // preventing reception errors
1805:           set_timer0(0);
1806:           while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
1807:           if(!kbhit()) // if there is no byte in the
USART
1808:           {
1809:             av_module = 0; // there is no air
velocity sensor
1810:             continue; // finish actual iteration
1811:           }
1812:
1813:           getc();
1814:
1815:           // preventing reception errors
1816:           set_timer0(0);
1817:           while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
1818:           if(!kbhit()) // if there is no byte in the
USART
1819:           {
1820:             av_module = 0; // there is no air
velocity sensor
1821:             continue; // finish actual iteration
1822:           }
1823:
1824:           av = getc(); // sensor response (byte of
air velocity)
1825:
1826:           // preventing reception errors
1827:           set_timer0(0);
1828:           while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
1829:           if(!kbhit()) // if there is no byte in the
USART
1830:           {
1831:             av_module = 0; // there is no air
velocity sensor
1832:             continue; // finish actual iteration
1833:           }
```

```
1834:
1835:             checksum = getc(); // air velocity
redundacy
1836:
1837:             restart_wdt();
1838:
1839:             if(checksum!=av) // air velocity byte and
redundacy don't match
1840:             {
1841:               error++;
1842:               goto again_0;
1843:             }
1844:             environment_0[0] = av*0.1; // air velocity
conversion (m/s)
1845:             environment_1[0] = environment_0[0];
1846:             environment_2[0] = environment_0[0];
1847:             output_high(WIND_LED);
1848:             break;
1849:           }
1850:         }
1851:       }
1852:       restart_wdt();
1853:       if(!av_module) // if sensor not found
1854:       {
1855:         output_low(WIND_LED);
1856:         if(stage == 0) // if stage egual to zero
1857:         {
1858:           if(curtain_state==OPEN_CURTAIN) // if curtain
is open
1859:           {
1860:             environment_0[0] = 0.2;
1861:           }
1862:           else // if curtain is close
1863:           {
1864:             environment_0[0] = 0;
1865:           }
1866:           environment_1[0] = environment_0[0];
1867:           environment_2[0] = environment_0[0];
1868:         }
1869:         else if(stage == 1) // if first stage
1870:         {
1871:           environment_0[0] = read_eeprom(0)*0.1; // air
velocity (m/s)
1872:           environment_1[0] = environment_0[0];
1873:           environment_2[0] = environment_0[0];
1874:         }
1875:         else if(stage == 2) // if second stage
1876:         {
1877:           environment_0[0] = read_eeprom(1)*0.1; // air
velocity (m/s)
1878:           environment_1[0] = environment_0[0];
1879:           environment_2[0] = environment_0[0];
1880:         }
```

```
1881:       else if(stage == 3) // if third stage
1882:       {
1883:         environment_0[0] = read_eeprom(2)*0.1; // air
velocity (m/s)
1884:         environment_1[0] = environment_0[0];
1885:         environment_2[0] = environment_0[0];
1886:       }
1887:     }
1888:     else
1889:     {
1890:       if((!av_levels)&&(stage==av_count)) // if there
is no calibration data
1891:       {
1892:         write_eeprom(av_count,av);
1893:         av_count++;
1894:       }
1895:       else if(av_count>2)
1896:       {
1897:         av_levels = 1;
1898:       }
1899:     }
1900:
1901:     //-----------------------------------------------
1902:     // START OF MEASURE PROCEDURES OF FIRST SENSOR
MODULE
1903:     //-----------------------------------------------
1904:
1905:     SetDDRamAddrHalfUp(0x29);
1906:     while(BusyHalfUpLCD());
1907:     environment_0[1] = null;
1908:     environment_0[2] = null;
1909:     error = 0;
1910:     again_1:
1911:     restart_wdt();
1912:
1913:     // bytes elimination
1914:     if(error)
1915:     {
1916:       set_timer0(0);
1917:       while(get_timer0()<1954)
1918:       {
1919:         if(kbhit())
1920:         {
1921:           getc();
1922:         }
1923:       }
1924:       restart_wdt();
1925:     }
1926:
1927:     if(error<3)
1928:     {
1929:       for(i=0;i<3;i++)
1930:       {
```

```
1931:          putc(0xFF);
1932:          delay_us(20);
1933:          putc(SHT11_1);
1934:          set_timer0(0); // timer0 is set (1:256
prescale value)
1935:          while(get_timer0()<9766)
1936:          {
1937:            if(kbhit()) // if receive, sensor is ok
1938:            {
1939:              goto ahead_1;
1940:            }
1941:          }
1942:          restart_wdt();
1943:        }
1944:
1945:        output_low(TH_LED_1);
1946:        strcpy(line," ");
1947:        putsHalfUpLCD(line);
1948:        while(BusyHalfUpLCD());
1949:        goto error_1;
1950:
1951:        ahead_1:
1952:
1953:        restart_wdt();
1954:
1955:        getc();
1956:
1957:        // preventing reception errors
1958:        set_timer0(0);
1959:        while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
1960:        if(!kbhit()) // if there is no byte in the
USART
1961:        {
1962:          error++;
1963:          goto again_1; // try again
1964:        }
1965:
1966:        getc();
1967:
1968:        // preventing reception errors
1969:        set_timer0(0);
1970:        while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
1971:        if(!kbhit()) // if there is no byte in the
USART
1972:        {
1973:          error++;
1974:          goto again_1; // try again
1975:        }
1976:
1977:        hi = getc(); // Temp. MSB
1978:
```

```
1979:       // preventing reception errors
1980:       set_timer0(0);
1981:       while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
1982:       if(!kbhit()) // if there is no byte in the
USART
1983:       {
1984:         error++;
1985:         goto again_1; // try again
1986:       }
1987:
1988:       lo = getc(); // Temp. LSB
1989:
1990:       // preventing reception errors
1991:       set_timer0(0);
1992:       while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
1993:       if(!kbhit()) // if there is no byte in the
USART
1994:       {
1995:         error++;
1996:         goto again_1; // try again
1997:       }
1998:
1999:       checksum = getc();
2000:
2001:       temp = hi;
2002:       temp = temp<<8;
2003:       temp = temp|lo;
2004:
2005:       if(sht11_crc8(hi,lo,checksum,1))
2006:       {
2007:         error++;
2008:         goto again_1;
2009:       }
2010:
2011:       set_timer0(0);
2012:       while((get_timer0()<1954)&&(!kbhit()));
2013:       if(!kbhit())
2014:       {
2015:         error++;
2016:         goto again_1;
2017:       }
2018:
2019:       hi = getc(); // RH MSB
2020:
2021:       // preventing reception errors
2022:       set_timer0(0);
2023:       while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
2024:       if(!kbhit()) // if there is no byte in the
USART
2025:       {
```

```
2026:         error++;
2027:         goto again_1; // try again
2028:       }
2029:
2030:       lo = getc(); //RH LSB
2031:
2032:       // preventing reception errors
2033:       set_timer0(0);
2034:       while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
2035:       if(!kbhit()) // if there is no byte in the
USART
2036:       {
2037:         error++;
2038:         goto again_1; // try again
2039:       }
2040:
2041:       checksum = getc();
2042:
2043:       humi = hi;
2044:       humi = humi<<8;
2045:       humi = humi|lo;
2046:
2047:       if(sht11_crc8(hi,lo,checksum,0))
2048:       {
2049:         error++;
2050:         goto again_1;
2051:       }
2052:       environment_0[1] = sht11_temp(temp); // Temp
calculation
2053:       environment_0[2] =
sht11_humi(humi,environment_0[1]); // RH calculation
2054:       output_high(TH_LED_1);
2055:
2056:       // writes Temp., RH and air velocity to LCD
2057:       sprintf(line," 1 %2.1f ",environment_0[1]);
2058:       putsHalfUpLCD(line);
2059:       while(BusyHalfUpLCD());
2060:       sprintf(line, "%2.1f ",environment_0[2]);
2061:       putsHalfUpLCD(line);
2062:       while(BusyHalfUpLCD());
2063:       sprintf(line,"%1.1f",environment_0[0]);
2064:       putsHalfUpLCD(line);
2065:       while(BusyHalfUpLCD());
2066:     }
2067:     else
2068:     {
2069:       output_low(TH_LED_1);
2070:       strcpy(line," ");
2071:       putsHalfUpLCD(line);
2072:       while(BusyHalfUpLCD());
2073:     }
2074:     error_1:
```

```
2075:     WriteCmdHalfUpLCD(HOME);
2076:     while(BusyHalfUpLCD());
2077:     SetDDRamAddrHalfUp(0x29);
2078:     while(BusyHalfUpLCD());
2079:     restart_wdt();
2080:
2081:     //-----------------------------------------------
2082:     // END OF MEASURE PROCEDURES OF FIRST SENSOR
MODULE
2083:     //-----------------------------------------------
2084:
2085:
2086:     //-----------------------------------------------
2087:     // START OF MEASURE PROCEDURES OF SECOND SENSOR
MODULE
2088:     //-----------------------------------------------
2089:     WriteCmdHalfDownLCD(HOME);
2090:     while(BusyHalfDownLCD());
2091:     SetDDRamAddrHalfDown(0x00);
2092:     while(BusyHalfDownLCD());
2093:     environment_1[1] = null;
2094:     environment_1[2] = null;
2095:     error = 0;
2096:     again_2:
2097:     restart_wdt();
2098:
2099:     // bytes elimination
2100:     if(error)
2101:     {
2102:       set_timer0(0);
2103:       while(get_timer0()<1954)
2104:       {
2105:         if(kbhit())
2106:         {
2107:           getc();
2108:         }
2109:       }
2110:       restart_wdt();
2111:     }
2112:
2113:     if(error<3)
2114:     {
2115:       for(i=0;i<3;i++)
2116:       {
2117:         putc(0xFF);
2118:         delay_us(20);
2119:         putc(SHT11_2);
2120:         set_timer0(0); // timer0 is set (1:256
prescale value)
2121:         while(get_timer0()<9766)
2122:         {
2123:           if(kbhit()) // if receive, sensor is ok
2124:           {
```

```
2125:             goto ahead_2;
2126:           }
2127:         }
2128:         restart_wdt();
2129:       }
2130:       output_low(TH_LED_2);
2131:       strcpy(line," ");
2132:       putsHalfDownLCD(line);
2133:       while(BusyHalfDownLCD());
2134:       goto error_2;
2135:
2136:       ahead_2:
2137:
2138:       restart_wdt();
2139:
2140:       getc();
2141:
2142:       // preventing reception errors
2143:       set_timer0(0);
2144:       while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
2145:       if(!kbhit()) // if there is no byte in the
USART
2146:       {
2147:         error++;
2148:         goto again_2; // try again
2149:       }
2150:
2151:       getc();
2152:
2153:       // preventing reception errors
2154:       set_timer0(0);
2155:       while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
2156:       if(!kbhit()) // if there is no byte in the
USART
2157:       {
2158:         error++;
2159:         goto again_2; // try again
2160:       }
2161:
2162:       hi = getc(); // Temp. MSB
2163:
2164:       // preventing reception errors
2165:       set_timer0(0);
2166:       while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
2167:       if(!kbhit()) // if there is no byte in the
USART
2168:       {
2169:         error++;
2170:         goto again_2; // try again
2171:       }
```

```
2172:
2173:       lo = getc(); // Temp. LSB
2174:
2175:       // preventing reception errors
2176:       set_timer0(0);
2177:       while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
2178:       if(!kbhit()) // if there is no byte in the
USART
2179:       {
2180:         error++;
2181:         goto again_2; // try again
2182:       }
2183:
2184:       checksum = getc();
2185:
2186:       temp = hi;
2187:       temp = temp<<8;
2188:       temp = temp|lo;
2189:
2190:       if(sht11_crc8(hi,lo,checksum,1))
2191:       {
2192:         error++;
2193:         goto again_2;
2194:       }
2195:
2196:       // preventing reception errors
2197:       set_timer0(0);
2198:       while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
2199:       if(!kbhit()) // if there is no byte in the
USART
2200:       {
2201:         error++;
2202:         goto again_2; // try again
2203:       }
2204:
2205:       hi = getc(); // RH MSB
2206:
2207:       // preventing reception errors
2208:       set_timer0(0);
2209:       while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
2210:       if(!kbhit()) // if there is no byte in the
USART
2211:       {
2212:         error++;
2213:         goto again_2; // try again
2214:       }
2215:
2216:       lo = getc(); //RH LSB
2217:
2218:       // preventing reception errors
```

```
2219:       set_timer0(0);
2220:       while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
2221:       if(!kbhit()) // if there is no byte in the
USART
2222:       {
2223:         error++;
2224:         goto again_2; // try again
2225:       }
2226:
2227:       checksum = getc();
2228:
2229:       humi = hi;
2230:       humi = humi<<8;
2231:       humi = humi|lo;
2232:
2233:       if(sht11_crc8(hi,lo,checksum,0))
2234:       {
2235:         error++;
2236:         goto again_2;
2237:       }
2238:
2239:       environment_1[1] = sht11_temp(temp); // Temp.
calculation
2240:       environment_1[2] =
sht11_humi(humi,environment_1[1]); // RH calculation
2241:       output_high(TH_LED_2);
2242:
2243:       // writes Temp., RH and V to LCD
2244:       sprintf(line," 2 %2.1f ",environment_1[1]);
2245:       putsHalfDownLCD(line);
2246:       while(BusyHalfDownLCD());
2247:       sprintf(line,"%2.1f ",environment_1[2]);
2248:       putsHalfDownLCD(line);
2249:       while(BusyHalfDownLCD());
2250:       sprintf(line,"%1.1f",environment_1[0]);
2251:       putsHalfDownLCD(line);
2252:       while(BusyHalfDownLCD());
2253:     }
2254:     else
2255:     {
2256:       output_low(TH_LED_2);
2257:       strcpy(line," ");
2258:       putsHalfDownLCD(line);
2259:       while(BusyHalfDownLCD());
2260:     }
2261:     error_2:
2262:     restart_wdt();
2263:
2264:     //-----------------------------------------------
2265:     // END OF MEASURE PROCEDURES OF SECOND SENSOR
MODULE
2266:     //-----------------------------------------------
```

```
2267:
2268:
2269:      //-----------------------------------------------
2270:      // START OF MEASURE PROCEDURES OF THIRD SENSOR
MODULE
2271:      //-----------------------------------------------
2272:      WriteCmdHalfDownLCD(HOME);
2273:      while(BusyHalfDownLCD());
2274:      SetDDRamAddrHalfDown(0x29);
2275:      while(BusyHalfDownLCD());
2276:      environment_2[1] = null;
2277:      environment_2[2] = null;
2278:      error = 0;
2279:      again_3:
2280:      restart_wdt();
2281:
2282:      // bytes elimination
2283:      if(error)
2284:      {
2285:        set_timer0(0);
2286:        while(get_timer0()<1954)
2287:        {
2288:          if(kbhit())
2289:          {
2290:            getc();
2291:          }
2292:        }
2293:        restart_wdt();
2294:      }
2295:
2296:      if(error<3)
2297:      {
2298:        for(i=0;i<3;i++)
2299:        {
2300:          putc(0xFF);
2301:          delay_us(20);
2302:          putc(SHT11_3);
2303:          set_timer0(0); // timer0 is set (1:256
prescale value)
2304:          while(get_timer0()<9766)
2305:          {
2306:            if(kbhit()) // if receive, sensor is ok
2307:            {
2308:              goto ahead_3;
2309:            }
2310:          }
2311:          restart_wdt();
2312:        }
2313:        output_low(TH_LED_3);
2314:        strcpy(line," ");
2315:        putsHalfDownLCD(line);
2316:        while(BusyHalfDownLCD());
2317:        goto error_3;
```

```
2318:
2319:       ahead_3:
2320:
2321:       restart_wdt();
2322:
2323:       getc();
2324:
2325:       // preventing reception errors
2326:       set_timer0(0);
2327:       while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
2328:       if(!kbhit()) // if there is no byte in the
USART
2329:       {
2330:         error++;
2331:         goto again_3; // try again
2332:       }
2333:
2334:       getc();
2335:
2336:       // preventing reception errors
2337:       set_timer0(0);
2338:       while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
2339:       if(!kbhit()) // if there is no byte in the
USART
2340:       {
2341:         error++;
2342:         goto again_3; // try again
2343:       }
2344:
2345:       hi = getc(); // Temp. MSB
2346:
2347:       // preventing reception errors
2348:       set_timer0(0);
2349:       while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
2350:       if(!kbhit()) // if there is no byte in the
USART
2351:       {
2352:         error++;
2353:         goto again_3; // try again
2354:       }
2355:
2356:       lo = getc(); // Temp. LSB
2357:
2358:       // preventing reception errors
2359:       set_timer0(0);
2360:       while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
2361:       if(!kbhit()) // if there is no byte in the
USART
2362:       {
```

```
2363:         error++;
2364:         goto again_3; // try again
2365:       }
2366:
2367:       checksum = getc(); // Temp. checksum
2368:
2369:       temp = hi;
2370:       temp = temp<<8;
2371:       temp = temp|lo;
2372:
2373:       if(sht11_crc8(hi,lo,checksum,1))
2374:       {
2375:         error++;
2376:         goto again_3;
2377:       }
2378:
2379:       // preventing reception errors
2380:       set_timer0(0);
2381:       while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
2382:       if(!kbhit()) // if there is no byte in the
USART
2383:       {
2384:         error++;
2385:         goto again_3; // try again
2386:       }
2387:
2388:       hi = getc(); // RH MSB
2389:
2390:       // preventing reception errors
2391:       set_timer0(0);
2392:       while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
2393:       if(!kbhit()) // if there is no byte in the
USART
2394:       {
2395:         error++;
2396:         goto again_3; // try again
2397:       }
2398:
2399:       lo = getc(); // RH LSB
2400:
2401:       // preventing reception errors
2402:       set_timer0(0);
2403:       while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
2404:       if(!kbhit()) // if there is no byte in the
USART
2405:       {
2406:         error++;
2407:         goto again_3; // try again
2408:       }
2409:
```

```
2410:         checksum = getc(); // RH checksum
2411:
2412:         humi = hi;
2413:         humi = humi<<8;
2414:         humi = humi|lo;
2415:
2416:         if(sht11_crc8(hi,lo,checksum,0))
2417:         {
2418:           error++;
2419:           goto again_3;
2420:         }
2421:
2422:         restart_wdt();
2423:
2424:         environment_2[1] = sht11_temp(temp);// Temp.
calculation
2425:         environment_2[2] =
sht11_humi(humi,environment_2[1]); // RH Calculation
2426:         output_high(TH_LED_3);
2427:
2428:         // writes Temp., RH and V to LCD
2429:         sprintf(line," 3 %2.1f ",environment_2[1]);
2430:         putsHalfDownLCD(line);
2431:         while(BusyHalfDownLCD());
2432:         sprintf(line,"%2.1f ",environment_2[2]);
2433:         putsHalfDownLCD(line);
2434:         while(BusyHalfDownLCD());
2435:         sprintf(line,"%1.1f",environment_2[0]);
2436:         putsHalfDownLCD(line);
2437:         while(BusyHalfDownLCD());
2438:       }
2439:       else
2440:       {
2441:         output_low(TH_LED_3);
2442:         strcpy(line," ");
2443:         putsHalfDownLCD(line);
2444:         while(BusyHalfDownLCD());
2445:       }
2446:       error_3:
2447:       restart_wdt();
2448:       //-------------------------------------------------
2449:       // END OF MEASURE PROCEDURES OF THIRD SENSOR
MODULE
2450:       //-------------------------------------------------
2451:
2452:       if(time>9)
2453:       {
2454:         rectal[0] = null;
2455:         rectal[1] = null;
2456:         rectal[2] = null;
2457:
2458:         // Rectal temperature estimation
2459:         if(environment_0[1]!=null)
```

```
2460:         {
2461:           rectal[0] = ann_FF_run(environment_0);
2462:         }
2463:         if(environment_1[1]!=null)
2464:         {
2465:           rectal[1] = ann_FF_run(environment_1);
2466:         }
2467:         if(environment_2[2]!=null)
2468:         {
2469:           rectal[2] = ann_FF_run(environment_2);
2470:         }
2471:
2472:         //-----------------------------------------------
2473:         // selection of the high value of rectal
temperature
2474:         //-----------------------------------------------
2475:         result = rectal[0];
2476:
2477:         if(((result<rectal[1])||(result==null))&&
(rectal[1]!=null))
2478:         {
2479:           result = rectal[1];
2480:         }
2481:         if(((result<rectal[2])||(result==null))&&
(rectal[2]!=null))
2482:         {
2483:           result = rectal[2];
2484:         }
2485:
2486:         //-----------------------------------------------
2487:         //-----------------------------------------------
2488:
2489:         //estimation error
2490:         if(result!=null)
2491:         {
2492:           time = 0;
2493:         }
2494:         else
2495:         {
2496:           error = 0;
2497:           goto again_1;
2498:         }
2499:
2500:         //-----------------------------------------------
2501:         // geting the high value of RH
2502:         //-----------------------------------------------
2503:
2504:         humidity = environment_0[2];
2505:
2506:         if(((humidity<environment_1[2])||
(humidity==null))&&(environment_1[2]!=null))
2507:         {
2508:           humidity = environment_1[2];
```

```
2509:         }
2510:         if(((humidity<environment_2[2])||
(humidity==null))&&(environment_2[2]!=null))
2511:         {
2512:           humidity = environment_2[2];
2513:         }
2514:         //---------------------------------------------
2515:         //---------------------------------------------
2516:
2517:
2518:         restart_wdt();
2519:
2520:         if(result<41.68) //if rectal temperature is
less than 41.68
2521:         {
2522:           set_timer0(0);
2523:           curtain_module = 1;
2524:           error_4:
2525:           restart_wdt();
2526:
2527:           if((stage == 1)&&(curtain_module)) // if 1st
stage and curtain
2528:           {
2529:             for(i=0;i<3;i++)
2530:             {
2531:               putc(0xFF);
2532:               putc(OPEN_CURTAIN); // open curtain
2533:               set_timer0(0); // timer0 is set (1:256
prescale value)
2534:               while(get_timer0()<9766)
2535:               {
2536:                 if(kbhit()) // if receive, curtain
module is ok
2537:                 {
2538:                   goto ahead_4;
2539:                 }
2540:               }
2541:               restart_wdt();
2542:             }
2543:             curtain_module = 0;
2544:             output_low(CUR_LED); // turn off curtain
LED
2545:             goto error_4;
2546:
2547:             ahead_4:
2548:
2549:             restart_wdt();
2550:
2551:             getc();
2552:
2553:             // preventing reception errors
2554:             set_timer0(0);
```

```
2555:           while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
2556:           if(!kbhit()) // if there is no byte in the
USART
2557:           {
2558:             // curtain module error
2559:             curtain_module = 0;
2560:             output_low(CUR_LED); // turn off curtain
LED
2561:             goto error_4;
2562:           }
2563:
2564:           getc();
2565:
2566:           // preventing reception errors
2567:           set_timer0(0);
2568:           while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
2569:           if(!kbhit()) // if there is no byte in the
USART
2570:           {
2571:             // curtain module error
2572:             curtain_module = 0;
2573:             output_low(CUR_LED); // turn off curtain
LED
2574:             goto error_4;
2575:           }
2576:
2577:           curtain_state = getc(); // curtain state
2578:           output_high(CUR_LED); // turn on curtain
LED
2579:
2580:           output_low(STAGE_1); // deactivates 1st
stage
2581:           stage--;
2582:         }
2583:         else if(stage == 2) // if 2nd stage
2584:         {
2585:           output_low(STAGE_2); // deactivates 2nd
stage
2586:           stage--;
2587:         }
2588:         else if(stage == 3) // if 3rd stage
2589:         {
2590:           output_low(STAGE_3); // deactivates 3rd
stage
2591:           stage--;
2592:         }
2593:         else if(stage == 4) // if 4th stage
(evaporative cooling)
2594:         {
2595:           output_low(NOZZLE); // deactivates
evaporative cooling
```

```
2596:           stage--;
2597:         }
2598:         else if(curtain_module&&
(curtain_state==CLOSE_CURTAIN))
2599:         {
2600:           for(i=0;i<3;i++)
2601:           {
2602:             putc(0xFF);
2603:             putc(OPEN_CURTAIN); // open curtain
2604:             set_timer0(0); // timer0 is set (1:256
prescale value)
2605:             while(get_timer0()<9766)
2606:             {
2607:               if(kbhit()) // if receive, curtain
module is ok
2608:               {
2609:                 goto ahead_5;
2610:               }
2611:             }
2612:             restart_wdt();
2613:           }
2614:           curtain_module = 0;
2615:           output_low(CUR_LED);
2616:           goto error_5;
2617:
2618:           ahead_5:
2619:
2620:           restart_wdt();
2621:
2622:           getc();
2623:
2624:           // preventing reception errors
2625:           set_timer0(0);
2626:           while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
2627:           if(!kbhit()) // if there is no byte in the
USART
2628:           {
2629:             // curtain module error
2630:             curtain_module = 0;
2631:             goto error_5;
2632:           }
2633:
2634:           getc();
2635:
2636:           // preventing reception errors
2637:           set_timer0(0);
2638:           while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
2639:           if(!kbhit()) // if there is no byte in the
USART
2640:           {
2641:             // curtain module error
```

```
2642:             curtain_module = 0;
2643:             goto error_5;
2644:           }
2645:
2646:           curtain_state = getc(); // curtain state
2647:           output_high(CUR_LED); // turn on curtain
LED
2648:
2649:           error_5:
2650:           if(!curtain_module)
2651:           {
2652:             output_low(CUR_LED);
2653:             output_high(STAGE_1);
2654:             stage++;
2655:           }
2656:         }
2657:       }
2658:       else if(result>41.72) // if rectal temperature
is more than 41.72
2659:       {
2660:         restart_wdt();
2661:         if(!stage)
2662:         {
2663:           for(i=0;i<3;i++)
2664:           {
2665:             putc(0xFF);
2666:             putc(CLOSE_CURTAIN);
2667:             set_timer0(0); // timer0 is set (1:256
prescale value)
2668:             while(get_timer0()<9766)
2669:             {
2670:               if(kbhit()) // if receive, sensor is ok
2671:               {
2672:                 goto ahead_6;
2673:               }
2674:             }
2675:             restart_wdt();
2676:           }
2677:           curtain_module = 0;
2678:           output_low(CUR_LED); // turn off curtain
LED
2679:           goto error_6;
2680:
2681:           ahead_6:
2682:
2683:           restart_wdt();
2684:
2685:           getc();
2686:
2687:           // preventing reception errors
2688:           set_timer0(0);
2689:           while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
```

```
2690:           if(!kbhit()) // if there is no byte in the
USART
2691:           {
2692:             // curtain module error
2693:             curtain_module = 0;
2694:             output_low(CUR_LED); // turn off curtain
LED
2695:             goto error_6;
2696:           }
2697:
2698:           getc();
2699:
2700:           // preventing reception errors
2701:           set_timer0(0);
2702:           while((get_timer0()<1954)&&(!kbhit())); //
waits 100ms for a byte
2703:           if(!kbhit()) // if there is no byte in the
USART
2704:           {
2705:             // curtain module error
2706:             curtain_module = 0;
2707:             output_low(CUR_LED); // turn off curtain
LED
2708:             goto error_6;
2709:           }
2710:
2711:           curtain_state = getc();
2712:           error_6:
2713:           output_high(STAGE_1);
2714:           stage++;
2715:         }
2716:         else if(stage == 1) // if 1st stage
2717:         {
2718:           output_high(STAGE_2); // activates 2nd
stage
2719:           stage++;
2720:         }
2721:         else if(stage == 2) // if 2nd stage
2722:         {
2723:           output_high(STAGE_3); // activates 3rd
stage
2724:           stage++;
2725:         }
2726:         else if((stage == 3)&&(humidity<70)) // if
3rd stage and humidity
2727:         { // less than 70 %
2728:           output_high(NOZZLE); // activates
evaporative cooling
2729:           stage++;
2730:         }
2731:         else if((stage == 4)&&(humidity>=80)) //if
humidity equal or more than 80
2732:         {
```

```
2733:           output_low(NOZZLE); // deactivates
evaporative cooling
2734:           stage--;
2735:         }
2736:       }
2737:       else if((humidity>=80)&&(stage==4)) // if 4th
stage and humidity >= 80%
2738:       {
2739:         output_low(NOZZLE); // deactivates
evaporative cooling
2740:         stage--;
2741:       }
2742:       else if((stage==0)&&
(curtain_state==CLOSE_CURTAIN)) // no ventilation
2743:       { // and closed curtain
2744:         output_high(STAGE_1); // activates 1st stage
2745:         stage++;
2746:       }
2747:     }
2748:     for(i=0;i<20;i++) //waits 1 minute
2749:     {
2750:       set_timer0(0);
2751:       while(get_timer0()<58594) // 3 seconds
2752:       {
2753:         restart_wdt();
2754:       }
2755:     }
2756:     time++;
2757:   }
2758: }
```

APPENDIX B - Keypad routines

```
1: /* CCS keypad routines.
2: /*
3: * Name: keypad.h
4: * Copyright: free
5: * Author: Alison Zille Lopes
6: * Date: 25/05/08 12:18
7: * Description: keypad control
8: */
9:
10:
11: /* Keypad rows definition
12: could be changed to adapt to the hardware project */
13:
14: #define ROW_1 pin_a0 /* keypad first row */
15: #define ROW_2 pin_a4 /* keypad second row */
16: #define ROW_3 pin_a5 /* keypad third row */
17: #define ROW_4 pin_e0 /* keypad fourth row */
18:
19: /* Keypad columns definition
20: could be changed to adapt to the hardware project */
21:
22: #define COLUMN_1 pin_a3 /* keypad first column */
23: #define COLUMN_2 pin_a2 /* keypad second column */
24: #define COLUMN_3 pin_a1 /* keypad third column */
25:
26: char keypad()
27: {
28:   char key;
29:   output_low(ROW_1);
30:   output_low(ROW_2);
31:   output_low(ROW_3);
32:   output_low(ROW_4);
33:   while((input(COLUMN_1))&(input(COLUMN_2))&
(input(COLUMN_3)))
34:   {
35:     restart_wdt();
36:   }
37:
38:   if(!input(COLUMN_1))
39:   {
40:     set_tris_a(0b110001);
41:     output_low(COLUMN_1);
42:     if(!input(ROW_1))
43:     {
44:       key = '1';
45:     }
46:     else if(!input(ROW_2))
47:     {
```

```
48:       key = '4';
49:     }
50:     else if(!input(ROW_3))
51:     {
52:       key = '7';
53:     }
54:     else if(!input(ROW_4))
55:     {
56:       key = '*';
57:     }
58:   }
59:   else if(!input(COLUMN_2))
60:   {
61:     set_tris_a(0b110001);
62:     output_low(COLUMN_2);
63:     if(!input(ROW_1))
64:     {
65:       key = '2';
66:     }
67:     else if(!input(ROW_2))
68:     {
69:       key = '5';
70:     }
71:     else if(!input(ROW_3))
72:     {
73:       key = '8';
74:     }
75:     else if(!input(ROW_4))
76:     {
77:       key = '0';
78:     }
79:   }
80:   else if(!input(COLUMN_3))
81:   {
82:     set_tris_a(0b110001);
83:     output_low(COLUMN_3);
84:     if(!input(ROW_1))
85:     {
86:       key = '3';
87:     }
88:     else if(!input(ROW_2))
89:     {
90:       key = '6';
91:     }
92:       else if(!input(ROW_3))
93:       {
94:         key = '9';
95:       }
96:       else if(!input(ROW_4))
97:       {
98:         key = '#';
99:       }
100:   }
```

```
101:    set_tris_a(0b001110);
102:    output_low(ROW_1);
103:    output_low(ROW_2);
104:    output_low(ROW_3);
105:    output_low(ROW_4);
106:    while((!input(COLUMN_1))||(!input(COLUMN_2))||
(!input(COLUMN_3)))
107:    {
108:      restart_wdt();
109:    }
110:    return key;
111: }
```

APPENDIX C - LCD peripheral routines

```
1: /* CCS LCD peripheral routines.
2: /*
3: * Name: lcd.h
4: * Copyright: free
5: * Author: Alison Zille Lopes
6: * Date: 24/05/08 02:53
7: * Description:
8: */
9:
10: /* DATA_LINE 8 bits comands */
11:
12: #define WriteDataPort(x) output_b(x) /* Write at data
port */
13: #define ReadDataPort() input_b() /* Read at data port
*/
14:
15: /* CTRL_PORT defines the port where the control lines
are connected.
16: * These are just samples, change to match your
application.
17: */
18:
19: #define E1_PIN pin_d7 /* PORT for E1 */
20: #define RW_PIN pin_d6 /* PORT for RW */
21: #define RS_PIN pin_d5 /* PORT for RS */
22: #define E2_PIN pin_d4 /* PORT for E2 */
23:
24:
25: /* Display ON/OFF Control defines */
26: #define ERASE 0b00000001 /* Display clean */
27: #define HOME 0b00000011 /* Cursor Home */
28: #define DON 0b00001111 /* Display on */
29: #define DOFF 0b00001011 /* Display off */
30: #define CURSOR_ON 0b00001111 /* Cursor on */
31: #define CURSOR_OFF 0b00001101 /* Cursor off */
32: #define BLINK_ON 0b00001111 /* Cursor Blink */
33: #define BLINK_OFF 0b00001110 /* Cursor No Blink */
34:
35: /* Cursor or Display Shift defines */
36: #define SHIFT_CUR_LEFT 0b00010011 /* Cursor shifts to
the left */
37: #define SHIFT_CUR_RIGHT 0b00010111 /* Cursor shifts to
the right */
38: #define SHIFT_DISP_LEFT 0b00011011 /* Display shifts to
the left */
39: #define SHIFT_DISP_RIGHT 0b00011111 /* Display shifts
to the right */
40:
```

```
41: /* Function Set defines */
42: #define FOUR_BIT 0b00101111 /* 4-bit Interface */
43: #define EIGHT_BIT 0b00111111 /* 8-bit Interface */
44: #define LINE_5X7 0b00110011 /* 5x7 characters, single
line */
45: #define LINE_5X10 0b00110111 /* 5x10 characters */
46: #define LINES_5X7 0b00111011 /* 5x7 characters,
multiple line */
47:
48: /*-----------------------------------------------------*/
49: /* Is the half-up screen controller busy? */
50: unsigned char BusyHalfUpLCD(void)
51: /*-----------------------------------------------------*/
52: {
53:   output_high(RW_PIN); // Set the control bits for read
54:   output_low(RS_PIN);
55:   delay_us(1);
56:   output_high(E1_PIN); // Clock in the command
57:   delay_us(1);
58:   if(input(pin_b7)) // Read bit 7 (busy bit)
59:   { // If high
60:     output_low(E1_PIN); // Reset clock line
61:     output_low(RW_PIN); // Reset control line
62:     return 1; // Return TRUE
63:   }
64:   else // Bit 7 low
65:   {
66:     output_low(E1_PIN); // Reset clock line
67:     output_low(RW_PIN); // Reset control line
68:     return 0; // Return FALSE
69:   }
70: }
71:
72: /*-----------------------------------------------------*/
73: /* Is the half-down screen controller busy? */
74: unsigned char BusyHalfDownLCD(void)
75: /*-----------------------------------------------------*/
76: {
77:   output_high(RW_PIN); // Set the control bits for read
78:   output_low(RS_PIN);
79:   delay_us(1);
80:   output_high(E2_PIN); // Clock in the command
81:   delay_us(1);
82:   if(input(pin_b7)) // Read bit 7 (busy bit)
83:   { // If high
84:     output_low(E2_PIN); // Reset clock line
85:     output_low(RW_PIN); // Reset control line
86:     return 1; // Return TRUE
87:   }
88:   else // Bit 7 low
89:   {
90:     output_low(E2_PIN); // Reset clock line
91:     output_low(RW_PIN); // Reset control line
```

```
92:     return 0; // Return FALSE
93:   }
94: }
95:
96: /*-------------------------------------------------*/
97: /* Write a command to the half-up screen controller */
98: void WriteCmdHalfUpLCD(unsigned char cmd)
99: /*-------------------------------------------------*/
100: {
101:   output_low(RW_PIN); // Set the control signals
102:   output_low(RS_PIN); // for sending a command
103:   WriteDataPort(cmd); // Write command to data port
104:   delay_us(1);
105:   output_high(E1_PIN); // Clock the command in
106:   delay_us(1);
107:   output_low(E1_PIN);
108:   delay_us(1);
109: }
110:
111: /*-------------------------------------------------*/
112: /* Write a command to the half-down screen controller
*/
113: void WriteCmdHalfDownLCD(unsigned char cmd)
114: /*-------------------------------------------------*/
115: {
116:   output_low(RW_PIN); // Set the control signals
117:   output_low(RS_PIN); // for sending a command
118:   WriteDataPort(cmd); // Write command to data port
119:   delay_us(1);
120:   output_high(E2_PIN); // Clock the command in
121:   delay_us(1);
122:   output_low(E2_PIN);
123:   delay_us(1);
124: }
125:
126: /*-------------------------------------------------*/
127: /* Write data to the half-up screen */
128: void WriteDataHalfUpLCD(char data)
129: /*-------------------------------------------------*/
130: {
131:   output_high(RS_PIN); // Set control bits
132:   output_low(RW_PIN);
133:   WriteDataPort(data); // Write data to port
134:   delay_us(1);
135:   output_high(E1_PIN); // Clock data into LCD
136:   delay_us(1);
137:   output_low(E1_PIN);
138:   output_low(RS_PIN); // Reset control bits
139: }
140:
141: /*-------------------------------------------------*/
142: /* Write data to the half-down screen */
143: void WriteDataHalfDownLCD(char data)
```

```
144: /*-------------------------------------------------*/
145: {
146:   output_high(RS_PIN); // Set control bits
147:   output_low(RW_PIN);
148:   WriteDataPort(data); // Write data to port
149:   delay_us(1);
150:   output_high(E2_PIN); // Clock data into LCD
151:   delay_us(1);
152:   output_low(E2_PIN);
153:   output_low(RS_PIN); // Reset control bits
154: }
155:
156: /*-------------------------------------------------*/
157: /* Write one string on half-up LCD driver screen */
158: void putsHalfUpLCD(char *buffer)
159: /*-------------------------------------------------*/
160: {
161:   while(*buffer) // Write data to LCD up to null
162:   {
163:     while(BusyHalfUpLCD()); // Wait while LCD is busy
164:     WriteDataHalfUpLCD(*buffer); // Write character to
LCD
165:     buffer++; // Increment buffer
166:   }
167: }
168:
169: /*-------------------------------------------------*/
170: /* Write one string on half-down LCD driver screen */
171: void putsHalfDownLCD(char *buffer)
172: /*-------------------------------------------------
*/
173: {
174:   while(*buffer) // Write data to LCD up to null
175:   {
176:     while(BusyHalfDownLCD()); // Wait while LCD is
busy
177:     WriteDataHalfDownLCD(*buffer); // Write character
to LCD
178:     buffer++; // Increment buffer
179:   }
180: }
181:
182: /*-------------------------------------------------*/
183: /* Set the half-up display data address */
184: void SetDDRamAddrHalfUp(unsigned char DDaddr)
185: /*-------------------------------------------------*/
186: {
187:   output_low(RW_PIN); // Set the control bits
188:   output_low(RS_PIN);
189:   WriteDataPort(DDaddr | 0b10000000); // Write cmd and
address to port
190:   delay_us(1);
191:   output_high(E1_PIN); // Clock the cmd and address in
```

```
192:   delay_us(1);
193:   output_low(E1_PIN);
194:   delay_us(1);
195: }
196:
197: /*-------------------------------------------------*/
198: /* Set the half-down display data address */
199: void SetDDRamAddrHalfDown(unsigned char DDaddr)
200: /*-------------------------------------------------*/
201: {
202:   output_low(RW_PIN); // Set the control bits
203:   output_low(RS_PIN);
204:   WriteDataPort(DDaddr | 0b10000000); // Write cmd and
address to port
205:   delay_us(1);
206:   output_high(E2_PIN); // Clock the cmd and address in
207:   delay_us(1);
208:   output_low(E2_PIN);
209:   delay_us(1);
210: }
211:
212: /*-------------------------------------------------*/
213: /* Set the half-up caracter generator address */
214: void SetCGRamAddrHalfUp(unsigned char CGaddr)
215: /*-------------------------------------------------*/
216: {
217:   WriteDataPort(CGaddr | 0b01000000); // Write cmd and
address to port
218:   output_low(RW_PIN); // Set control signals
219:   output_low(RS_PIN);
220:   delay_us(1);
221:   output_high(E1_PIN); // Clock cmd and address in
222:   delay_us(1);
223:   output_low(E1_PIN);
224:   delay_us(1);
225: }
226:
227: /*-------------------------------------------------*/
228: /* Set the half-down caracter generator address */
229: void SetCGRamAddrHalfDown(unsigned char CGaddr)
230: /*-------------------------------------------------*/
231: {
232:   WriteDataPort(CGaddr | 0b01000000); // Write cmd and
address to port
233:   output_low(RW_PIN); // Set control signals
234:   output_low(RS_PIN);
235:   delay_us(1);
236:   output_high(E2_PIN); // Clock cmd and address in
237:   delay_us(1);
238:   output_low(E2_PIN);
239:   delay_us(1);
240: }
241:
```

```
242: /*-------------------------------------------------*/
243: /* Read data address from half-up screen controller */
244: unsigned char ReadAddrHalfUpLCD(void)
245: /*-------------------------------------------------*/
246: {
247:   char data; // Holds the data retrieved from the LCD
248:   output_high(RW_PIN); // Set control bits for the
read
249:   output_low(RS_PIN);
250:   delay_us(1);
251:   output_high(E1_PIN); // Clock data out of the LCD
controller
252:   delay_us(1);
253:   data = ReadDataPort(); // Save the data in the
register
254:   output_low(E1_PIN);
255:   output_low(RW_PIN); // Reset the control bits
256:   return (data&0x7f); // Return the address, Mask off
the busy bit
257: }
258:
259: /*-------------------------------------------------*/
260: /* Read data address from half-down screen controller
*/
261: unsigned char ReadAddrHalfDownLCD(void)
262: /*-------------------------------------------------*/
263: {
264:   char data; // Holds the data retrieved from the LCD
265:   output_high(RW_PIN); // Set control bits for the
read
266:   output_low(RS_PIN);
267:   delay_us(1);
268:   output_high(E2_PIN); // Clock data out of the LCD
controller
269:   delay_us(1);
270:   data = ReadDataPort(); // Save the data in the
register
271:   output_low(E2_PIN);
272:   output_low(RW_PIN); // Reset the control bits
273: return (data&0x7f); // Return the address, Mask off
the busy bit
274: }
275:
276: /*-------------------------------------------------*/
277: /* Read data from half-up screen controller */
278: char ReadDataHalfUpLCD(void)
279: /*-------------------------------------------------*/
280: {
281:   char data; // Holds the data retrieved from the LCD
282:   output_high(RW_PIN); // Set control bits for the
read
283:   output_high(RS_PIN);
284:   delay_us(1);
```

```
285:    output_high(E1_PIN); // Clock data out of the LCD
controller
286:    delay_us(1);
287:    data = ReadDataPort(); // Read the data
288:    output_low(E1_PIN);
289:    output_low(RW_PIN); // Reset the control bits
290:    output_low(RS_PIN);
291:    return (data); // Return the data
292: }
293:
294: /*-----------------------------------------------*/
295: /* Read data from half-down screen controller */
296: unsigned char ReadDataHalfDownLCD(void)
297: /*-----------------------------------------------*/
298: {
299:    char data; // Holds the data retrieved from the LCD
300:    output_high(RW_PIN); // Set control bits for the
read
301:    output_high(RS_PIN);
302:    delay_us(1);
303:    output_high(E2_PIN); // Clock data out of the LCD
controller
304:    delay_us(1);
305:    data = ReadDataPort(); // Read the data
306:    output_low(E2_PIN);
307:    output_low(RW_PIN); // Reset the control bits
308:    output_low(RS_PIN);
309:    return (data); // Return the data
310: }
311:
312: /*-----------------------------------------------*/
313: /* Configure PIC I/O pins and initialize LCD
controllers */
314: void OpenLCD(unsigned char lcdtype)
315: /*-----------------------------------------------*/
316: {
317:    set_tris_b(0xFF);
318:    output_low(RW_PIN); // R/W pin made low
319:    output_low(RS_PIN); // Register select pin made low
320:    output_low(E1_PIN); // Clock pin made low
321:    output_low(E2_PIN);
322:    delay_ms(30); // Delay for 30ms to allow for LCD
Power on
323:    WriteDataPort(0b00110000); // Function set cmd(8-bit
interface)
324:
325:    // Half Up
326:
327:    output_high(E1_PIN); // Clock the cmd in
328:    delay_ms(5);
329:    output_low(E1_PIN);
330:    delay_ms(5); // Delay for 5ms
331:    output_high(E1_PIN); // Clock the cmd in
```

```
332:    delay_us(1);
333:    output_low(E1_PIN);
334:    delay_us(1); // Delay for 1ms
335:    output_high(E1_PIN); // Clock the cmd in
336:    delay_us(1);
337:    output_low(E1_PIN);
338:    delay_us(1); // Delay for 1ms
339:
340:    // Half Down
341:
342:    output_high(E2_PIN); // Clock the cmd in
343:    delay_ms(5);
344:    output_low(E2_PIN);
345:    delay_ms(5); // Delay for 5ms
346:    output_high(E2_PIN); // Clock the cmd in
347:    delay_us(1);
348:    output_low(E2_PIN);
349:    delay_us(1); // Delay for 1ms
350:    output_high(E2_PIN); // Clock the cmd in
351:    delay_us(1);
352:    output_low(E2_PIN);
353:    delay_us(1); // Delay for 1ms
354:
355:    // Set data interface width, # lines, font
356:    while(BusyHalfUpLCD()); // Wait if LCD busy
357:    WriteCmdHalfUpLCD(lcdtype); // Function set cmd
358:    while(BusyHalfDownLCD()); // Wait if LCD busy
359:    WriteCmdHalfDownLCD(lcdtype); // Function set cmd
360:
361:    // Turn the display on then off
362:    while(BusyHalfUpLCD()); // Wait if LCD busy
363:    WriteCmdHalfUpLCD(DOFF&CURSOR_OFF&BLINK_OFF); //
Display OFF/Blink OFF
364:    while(BusyHalfDownLCD()); // Wait if LCD busy
365:    WriteCmdHalfDownLCD(DOFF&CURSOR_OFF&BLINK_OFF); //
Display OFF/Blink OFF
366:    while(BusyHalfUpLCD()); // Wait if LCD busy
367:    WriteCmdHalfUpLCD(DON&CURSOR_ON&BLINK_ON); //
Display ON/Blink ON
368:    while(BusyHalfDownLCD()); // Wait if LCD busy
369:    WriteCmdHalfDownLCD(DON&CURSOR_ON&BLINK_ON); //
Display ON/Blink ON
370:
371:    // Clear display
372:    while(BusyHalfUpLCD()); // Wait if LCD busy
373:    WriteCmdHalfUpLCD(ERASE); // Clear display
374:    while(BusyHalfDownLCD()); // Wait if LCD busy
375:    WriteCmdHalfDownLCD(ERASE); // Clear display
376:
377:    // Set entry mode inc, no shift
378:    while(BusyHalfUpLCD()); // Wait if LCD busy
379:    WriteCmdHalfUpLCD(SHIFT_CUR_LEFT); // Entry Mode
380:    while(BusyHalfDownLCD()); // Wait if LCD busy
```

```
381:    WriteCmdHalfDownLCD(SHIFT_CUR_LEFT); // Entry Mode
382:
383:    // Set DD Ram address to 0
384:    while(BusyHalfUpLCD()); // Wait if LCD busy
385:    SetDDRamAddrHalfUp(0); // Set Display data ram
address to 0
386:    while(BusyHalfDownLCD()); // Wait if LCD busy
387:    SetDDRamAddrHalfDown(0); // Set Display data ram
address to 0
388:    while(BusyHalfUpLCD());
389:    while(BusyHalfDownLCD());
390: }
```

APPENDIX D - Sensor module program

```
1: /*******************************************************
2: Name: sensor.c
3: Copyright: private
4: Author: Alison Zille Lopes
5: Date: 07/16/08 02:44
6: Description: The source code for the sensor module,
sht11 based, was developed
7: as a part of an environment control system for poutry
houses.
8: *******************************************************/
9:  #include <18f1320.h>
10: #use delay(clock=20000000)
11: #use rs232 (baud = 9600,xmit = pin_b1,rcv =
pin_b4,enable = pin_b0,restart_wdt)
12: #use standard_io (A)
13: #fuses HS,WDT128
14:
15: #case
16:
17: //-------------------------------------------------------
18: // modul-var
19: //-------------------------------------------------------
20:
21: enum {TEMP,HUMI};
22: #define DATA PIN_A0
23: #define SCK PIN_A1
24: #define noACK 0
25: #define ACK 1
26:
27: //-------------------------------------------------------
28: // modul-cmd
29: //-------------------------------------------------------
30:
31: #define MEASURE 0xFF31
32:
33: // adr command r/w
34: #define STATUS_REG_W 0x06 // 000 0011 0
35: #define STATUS_REG_R 0x07 // 000 0011 1
36: #define MEASURE_TEMP 0x03 // 000 0001 1
37: #define MEASURE_HUMI 0x05 // 000 0010 1
38: #define RESET 0x1E // 000 1111 0
39:
40:
41:
42: //-------------------------------------------------------
43: char s_write_byte(unsigned char value)
44: //-------------------------------------------------------
```

```
45: // writes a byte on the Sensibus and checks the
acknowledge
46: {
47:   unsigned int i,error=0;
48:   for(i=0x80;i>0;i/=2) //shift bit for masking
49:   {
50:     if(i & value)
51:     {
52:       output_high(DATA); //masking value with i , write
to SENSI-BUS
53:       output_high(DATA);
54:     }
55:     else
56:     {
57:       output_low(DATA);
58:       output_low(DATA);
59:     }
60:     delay_us(5);
61:     output_high(SCK); //clk for SENSI-BUS
62:     delay_us(5);
63:     output_low(SCK);
64:   }
65:   input(DATA);
66:   input(DATA);
67:   delay_us(5);
68:   output_high(SCK); //clk #9 for ack
69:   delay_us(5);
70:   output_low(SCK);
71:   error = input(DATA); //check ack (DATA will be pulled
down by SHT11)
72:   error = input(DATA); //check ack (DATA will be pulled
down by SHT11)
73:   return error; //error=1 in case of no acknowledge
74: }
75:
76:
77: //-----------------------------------------------------
78: char s_read_byte(unsigned char ack)
79: //-----------------------------------------------------
80: // reads a byte form the Sensibus and gives an
acknowledge in case of "ack=1"
81: {
82:   unsigned int i,val=0;
83:   input(DATA);
84:   input(DATA);
85:   for(i=0x80;i>0;i/=2) //shift bit for masking
86:   {
87:     delay_us(5);
88:     output_high(SCK); //clk for SENSI-BUS
89:     if(input(DATA)) val=(val | i); //read bit
90:     delay_us(5);
91:     output_low(SCK);
92:   }
```

```
93:   output_bit(DATA,(!ack)); //in case of "ack==1" pull
down DATA-Line
94:   output_bit(DATA,(!ack)); //in case of "ack==1" pull
down DATA-Line
95:   delay_us(5);
96:   output_high(SCK); //clk #9 for ack
97:   delay_us(5);
98:   output_low(SCK);
99:   input(DATA);
100:   input(DATA);
101:   delay_us(5);
102:   return val;
103: }
104:
105:
106: //-----------------------------------------------------
107: void s_transstart(void)
108: //-----------------------------------------------------
109: // generates a transmission start
110: //         _____         ________
111: // DATA:      |_______|
112: //
113: // SCK : ___|   |___|   |______
114: {
115:   output_high(DATA);
116:   output_high(DATA);
117:   output_low(SCK); //Initial state
118:   delay_us(5);
119:   output_high(SCK);
120:   delay_us(5);
121:   output_low(DATA);
122:   output_low(DATA);
123:   delay_us(5);
124:   output_low(SCK);
125:   delay_us(5);
126:   output_high(SCK);
127:   delay_us(5);
128:   output_high(DATA);
129:   output_high(DATA);
130:   delay_us(5);
131:   output_low(SCK);
132: }
133:
134:
135: //-----------------------------------------------------
136: void s_connectionreset(void)
137: //-----------------------------------------------------
138: // communication reset: DATA-line=1 and at least 9 SCK
cycles followed by transstart
139: //         _____________________________________________         __
140: // DATA:                                              |___|
141: //
142: // SCK : _|‾|_|‾|_|‾|_|‾|_|‾|_|‾|_|‾|_|‾|_|‾|_|‾|_|‾|_|‾|_|‾|_
```

```
143: {
144:   unsigned char i;
145:   output_high(DATA);
146:   output_high(DATA);
147:   output_low(SCK); //Initial state
148:   for(i=0;i<9;i++) //9 SCK cycles
149:   {
150:     delay_us(5);
151:     output_high(SCK);
152:     delay_us(5);
153:     output_low(SCK);
154:   }
155:   s_transstart(); //transmission start
156: }
157:
158:
159: //-----------------------------------------------------
160: char s_measure(unsigned char *hi,unsigned char *lo,
unsigned char *p_checksum, unsigned char mode)
161: //-----------------------------------------------------
162: // makes a measurement (humidity/temperature) with
checksum
163: {
164:   unsigned char error=0;
165:   s_transstart(); //transmission start
166:   switch(mode)//send command to sensor
167:   {
168:     case TEMP : error+=s_write_byte(MEASURE_TEMP);
break;
169:     case HUMI : error+=s_write_byte(MEASURE_HUMI);
break;
170:     default : break;
171:   }
172:   input(DATA);
173:   delay_us(5);
174:   while(input(DATA)); //wait until sensor has finished
the measurement
175:   input(DATA);
176:   input(DATA);
177:   *hi =s_read_byte(ACK); //read the first byte (MSB)
178:   *lo = s_read_byte(ACK); //read the second byte (LSB)
179:   *p_checksum =s_read_byte(noACK); //read checksum
180:   return error;
181: }
182:
183:
184: //-----------------------------------------------------
185: void main()
186: //-----------------------------------------------------
187: {
188:   unsigned char t_hi,t_lo,t_checksum,h_hi,h_lo,
h_checksum,error,aux,count;
189:   unsigned long int opt;
```

```
190:    error = 0;
191:    count = 0;
192:    setup_wdt(WDT_ON);
193:    delay_ms(20);
194:    s_connectionreset();
195:    count = 2;
196:    while(1)
197:    {
198:      opt = 0;
199:      restart_wdt();
200:      while (opt!=MEASURE)
201:      {
202:        aux = getc();
203:        opt = opt<<8;
204:        opt = opt & 0xFF00;
205:        opt = opt | aux;
206:        count++;
207:      }
208:      if(count<3)
209:      {
210:        s_write_byte(RESET);
211:        delay_ms(20);
212:      }
213:      count = 0;
214:
215:      try_again:
216:      restart_wdt();
217:      error+=s_measure(&t_hi,&t_lo,&t_checksum,TEMP);
//measure temperature
218:      restart_wdt();
219:      error+=s_measure(&h_hi,&h_lo,&h_checksum,HUMI);
//measure humidity
220:      if(error)
221:      {
222:        s_connectionreset();
223:        error = 0;
224:        goto try_again;
225:      }
226:
227:      // send temperature
228:      putc(0x00);
229:      putc(0x00);
230:
231:      delay_us(20);
232:      putc(t_hi);
233:      delay_us(10);
234:      putc(t_lo);
235:      delay_us(10);
236:      putc(t_checksum);
237:      delay_us(50);
238:
239:      restart_wdt();
240:      // send humidity
```

```
241:
242:     putc(h_hi);
243:     delay_us(10);
244:     putc(h_lo);
245:     delay_us(10);
246:     putc(h_checksum);
247:   }
248: }
```