IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.021 ⁒ Peer-reviewed & Refereed journal ⁒ Vol. 12, Issue 12, December 2024

DOI: 10.17148/IJIREEICE.2024.121205

A visualized, self-regulating, easily expandable and low-cost system, for simultaneous measuring and control of visible and infrared lighting, temperature, humidity and time duration of the above parameters' values of a greenhouse or industrial environment, using VHDL and FPGAs

Dr Evangelos I. Dimitriadis[1], Ioannis Vourvoulakis[2], Leonidas Dimitriadis[3], Xenofon Dimitriadis[4]

Department of Computer, Informatics and Telecommunications Engineering,

International Hellenic University, End of Magnisias Str, 62124 Serres Greece, edimitriad@sch.gr[1]

Assistant Professor Department of Computer, Informatics and Telecommunications Engineering,

International Hellenic University, End of Magnisias Str, 62124 Serres Greece, jvourv@ihu.gr[2]

Undergraduate Student, Department of Information and Electronic Engineering, International Hellenic University,

57400, Sindos Thessaloniki, Greece, leonidasdimitriadis635@gmail.com[3]

B.Sc. Agriculture, Serres Greece, xedimitr@otenet.gr[4]

Abstract: We present here an FPGA-based system, capable of simultaneous measuring and control of visible and infrared lighting, temperature and humidity values of a greenhouse or industrial environment. The system uses DE10-Lite FPGA board with four sensors connected to it and in conjunction with the easy way of expanding both its software and hardware by adding more sensors, in order to cover larger areas, it is considered to be of low cost. All sensors and time values act as input. Sensors' range values and time duration for each of them are initially set by the programmer. The program used in this work converts analog inputs to digital values and displays corresponding voltage measurements in seven-segment displays of the board. A series of processes is activated upon system is set to ON, in order to achieve checking and control of parameter values. Blue LEDs and corresponding control systems are activated if related sensors' values become less than lower critical values set by programmer, while red LEDs and corresponding control systems are activated for sensors' values overcoming upper critical values. Especially for visible light values, step motor for opening or closing curtains is also activated and FPGA's board LEDs and buzzer connected to it are also ON, when upper critical value is exceeded. Yellow LEDs are activated for each parameter exceeding time set values. Finally an alarm level system turns on corresponding LEDs, depending on the number of parameter values that simultaneously exceed range values set by programmer.

Keywords: Sensors, Self-regulating system, FPGA, VHDL, Buzzer, LEDs

INTRODUCTION

It is well known that FPGAs have attracted attention of researchers in the recent years, for industrial as well as other applications. [1-13] FPGAs have the main advantage of combining software and hardware, thus enabling hardware programming for a series of applications. The most used languages for FPGAs' programing are VHDL and Verilog and VHDL is the one used in our work.
Another interesting application field of FPGAs are greenhouse and industrial environments parameters control. We found out that in spite of all work done concerning FPGAs, there are a few projects [14-16] concerning greenhouse control systems. Presented works use expensive and complicated systems, or control only two basic parameters. We present here an FPGA-based system able to measure and control four parameters, including visible and infrared light, temperature and humidity of the inspected area. Our work provides a system which can simultaneously measure and control the above

ISSN (O) 2321-2004, ISSN (P) 2321-5526
International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering
Impact Factor 8.021 ⁘ Peer-reviewed & Refereed journal ⁘ Vol. 12, Issue 12, December 2024
DOI: 10.17148/IJIREEICE.2024.121205

parameters and keep greenhouse or industrial area in a self-regulating condition, according to parameters' values set by the programmer. He can have a visualized view of parameters' measured values and also of the alarm level that the inspected area happens to be in. The algorithms used here can restore parameter values to desired ones by activating corresponding control systems, whenever one or more parameters are out of programmer's set range values. Another benefit of our system is that it can be easily expanded using more of the same kind sensors, in order to cover larger areas of inspection or alternative sensors (i.e. ultraviolet sensors) to measure other physical quantities. Additionally its cost is remarkably low.

Design overview and operation of the system

It is known[17,18] that visible light (400–780 nm) is critical for driving the photosynthesis process of greenhouse plants. Red and blue light are more efficient for photosynthesis, with the peak efficiency occurring at around 625 nm, but near-infrared spectrum (780–2500 nm) have little contributions. The plant consists for the most part of water, so it is quickly warmed up by infrared radiation, especially above 1200 nm. It is obvious from the above statements that both visible and infrared light are of great importance in modern greenhouse environments, hence two of our system sensors measuring and controlling their values. The other two sensors are involved with temperature and air humidity of the inspected area. Figure1 presents device overview and operational units of our system, using FPGA DE10-Lite board, while Figure2 presents circuit diagram of the system.

It is obvious from both of the above figures that our system, except from DE10-Lite FPGA board, contains also some basic circuit parts. The first one at the left top is visible light control system. Next to it there is a buzzer circuit containing a transistor and diode and it is connected in I/O pins of the FPGA board. It is the circuit that controls buzzer's operation. The transistor is used for amplifying signal bit 1 sent by the I/O pins of FPGA board, in order to provide sufficient voltage supply for buzzer operation.

At the right top we can see the four sensors used here, each of them connected to its voltage divider and acting as input to Analog to Digital Converter (ADC) pins of the FPGA board. We used NTC thermistor 10K, BPW 34 PIN IR photo-diode, GL5516 Light Dependent Resistor LDR 5MM and HR 202 humidity module. Below sensors' circuit part, exist three control parts. IR light, temperature and humidity control systems. At the right bottom we can see the alarm level system control, which informs user about the number of sensors' values being out of set range. Finally at the bottom of the system we can see the step motor with its control module, used for opening or closing curtains of the greenhouse, in order to achieve higher or lower visible light illuminance values, respectively. All sensors' values control circuits contain three LEDs, blue, red and yellow, corresponding to external control systems, which are activated when they receive bit 1 from the FPGA board.

Our system starts operating as soon as power supply +5V is applied to all sensors and the VHDL program is sent via USB Blaster interface, to FPGA chip, reading at first input voltages from all sensors' voltage dividers, with simultaneous start of time measurement. The analogue input voltages are converted to digital and presented in seven-segment displays using precision of two decimal digits. We used an SW switch of the FPGA board in order to present analog voltages in couples, infrared-visible and temperature-humidity. Needless to mention that the system receives input voltage values periodically, ensuring continuous voltage change monitoring.

Consequently, thirteen controls are simultaneously put in use. Each sensor is involved with three controls, leading to a total number of twelve controls for all sensors. First one is checking whether input voltage value is equal or lower than lower critical voltage value of 0.2V, set by the programmer, and, if this is true, then bit 1 is sent to control system1, represented by blue external LED. All sensors' range values are similarly set for convenience reasons. Second control is relevant to whether input voltage value exceeds upper critical voltage value of 1.0V and, if this is true, then bit 1 is sent to control system2, represented by red external LED. It must be mentioned that if input voltage values are greater than 0.2V and smaller than 1.0V, then bit 0 is sent to control systems 1 and 2 and both of them are deactivated. It is a great advantage of our system that the above critical voltage values can be set by the programmer, providing the ability of using our system in a variety of applications.

Third control is involved with time duration of each sensor's range values. A critical time-duration value is also set by the programmer and, if it is exceeded, then bit 1 is sent to control system3, represented by yellow external LED. In this work we used critical time value of a few minutes, in order to obtain a fast check of our system's successful operation and this value is the same for all sensors for convenience reasons.

The system can accept different sensor range values for different corresponding time- duration values, after making a small change in sensors' control algorithms. This is one of our system advantages mentioned earlier and in conjunction with the easy way of adding more sensors for monitoring larger areas, it offers the ability of a variety of applications for our system.

Finally, the thirteenth control activated at the same time as all the above sensors' controls is our system's alarm level control. It uses an algorithm process to periodically check the number of sensors whose values are simultaneously out of range, set by the programmer. The above number could take one of the following values 0, 1, 2, 3 or 4, leading to green, blue, white, yellow or red LED lighting up respectively, in the alarm level control area of our system, shown in Figures

DOI: 10.17148/IJIREEICE.2024.121205

1 and 2. It is obvious that every LED that lights up has received bit 1 from FPGA's GPIO pins, while simultaneously all others received bit 0.

All the above controls are periodically operated as long as the system is at the ON state. The system goes to OFF state if external circuit voltage supply is OFF or if FPGA board is unplugged from USB Blaster, or both of them.

It must be mentioned here that, concerning visible light control, except from blue, red and yellow LEDs output, we use simultaneously a step motor system with its corresponding algorithm process, for opening or closing covering area equipment, such as curtains, depending on illuminance values. When blue LED lights up, suggesting visible light values less than appropriate ones, step motor starts opening the greenhouse's curtains with the opposite process taking place if values exceed desired ones. Additionally, when visible light values exceed those set by the programmer, all FPGA's board LEDs light up and buzzer starts sounding.

It is obvious that similar systems are used to regulate all other physical quantities' values, monitored by four sensors, whenever they are out of set range values. Bit 0 or 1 could be sent to an infrared lamp system for turning it OFF or ON, if infrared values of the inspected area are higher or lower, respectively, than those set by the programmer. Similarly bit 0 or 1 can be sent to an air heating system for turning it OFF or ON, if temperature values of the inspected area are higher or lower, respectively, than those set by the programmer.

Finally, an air drying system could control, similarly to the above systems, air humidity values. All the above controls are related to corresponding algorithm processes, which are intended to achieve the goal of self-regulation of our system. The second goal of visualized system holds true, since system user has a consolidated view of the monitored physical quantities' values of the supervised area, with simultaneous check of all control systems behavior and operation.

We must mention here, some indicative measured values using our four sensors. Visible light illuminance of 650 lux resulted in 0.2V input value from LDR divider, while 3500 lux value resulted in 1.0V input value. Concerning IR light, 0.51 V input value was measured using a red, near-infrared laser of 5mW power for lighting IR photodiode, resulting in an approximate light intensity of 71.42 mW/cm^2. Temperature of 22^o C, resulted in an input voltage of 0.16V from the NTC divider, while 0.9V were measured for 45^o C. Finally, the humidity module HR202 resulted in 4.5V input value for an air humidity of 45%. All the above, make clear that depending on the inspected area and the physical quantities' range values existing there, we are enforced to make the appropriate changes for the resistance values used in the sensors' dividers, in order to sufficiently monitor input values.

Figures 3,4,5,6 and 7 present our system's operations mentioned above.

Figure 3 shows the system in red alarm condition operation, because all sensors values are out of set range. Red LED in the alarm system control unit is ON. Visible light, infrared and temperature inputs are below lower critical value (blue LEDs ON), while humidity input is above upper critical value (red LED ON). Consequently, alarm level system has red LED ON. Seven-segment displays present 0.04V and 0.18V input values for infrared and visible light respectively, both below 0.2V set as lower critical value. If we want to view in seven-segment displays the input values of temperature and humidity, SW0 switch of the FPGA board must go to ON state. Step motor starts rotating clockwise.

Figure 4 presents the system in red alarm condition operation again, but the difference is that visible light input values are now above upper critical value of 1V. We observe input value of 1.83V due to lighting LDR with the red, near-infrared laser beam. As we mentioned above, all FPGA board LEDs are ON and buzzer sounds. Step motor starts rotating counter-clockwise.

Figure 5 presents the system in yellow alarm condition operation, because one of four sensors input value, in this case visible light, is in the set range (0.23V) due to lighting LDR sensor. It is worth mentioning that step motor stops rotating.

Figure 6 presents our system in white alarm condition operation, meaning that two of four sensors' input values are in the set range. Both LDR and IR photodiode are lit and present input voltage values of 0.29V and 0.50V, respectively.

Figure 7 shows another important operation of our system. It is time duration of set range input values for all sensors. It is set to 1min for convenience reasons for all sensors, hence four yellow LEDs, placed in four sensors' control systems, are ON, informing system user that he can change one or more sensors' set range input values. VHDL program used here, gives the ability of setting different time duration input range values for one or more sensors, without the need of programmer's intervention each time a change is made. Yellow LED in each sensor control system could be used then just to remind user about the next input value time duration period.

Programing the system

We used Quartus Prime Lite Edition 21.1.1 to create the VHDL programs of our system.

It must be mentioned here that before proceeding with the VHDL programming of our system, we had to set a series of parameters controlling the operation of DE10-Lite FPGA's Analog to Digital Converter (ADC). This converter plays a very important role in the whole system operation, since it converts the analogue input voltages from all sensors connected to FPGA board to digital values, acting as main input of the system. The files created by the above ADC parameters setting are imported into the final project of our system. We also integrated specific settings, which give us the ability of having four input channels, corresponding to four sensors used in this work.

DOI: 10.17148/IJIREEICE.2024.121205

The overall program is shown in the appendix of this paper. Figure 8 presents the basic flowchart of our system's operation, which contains the main algorithmic procedures used here. The first check is whether the system is at the ON or OFF condition. If this holds true, four sensors' control systems start sending analog input voltage values to Analog to Digital Converter (ADC) of the FPGA board, with simultaneous time measurement. We use the appropriate processes to convert each sensor's input value to its corresponding digital one and display the analog values in seven-segment displays in pairs of two sensors, by using SW0 switch of the FPGA board to alternate between pairs. Processes of binary to BCD conversion are also incorporated here for each sensor.
Figure 8 consequently presents thirteen processes operating simultaneously in parallel. Twelve of them are related to the three controls taking place for each one of four sensors used in our system. They measure and control minimum and maximum input values and also time duration of the set range values for each sensor, thus controlling visible light, infrared light, temperature and humidity values of the greenhouse or industrial area inspected by the system. A set of LEDs, blue, red and yellow, act as output for each sensor control system, representing the corresponding control systems that are being activated in each case. Blue LED turns ON if input voltage value is lower than the minimum set critical value. Red LED respectively, turns ON if input value is higher than the maximum set value and yellow LED turns ON to inform user that the time duration of the set range values for each sensor is exceeded. Especially for visible light, additional algorithmic process is activated concerning step motor operation, to control the opening or closing of curtains or roof covering of the inspected area, as we discuss below for Figure 9.
Finally, the thirteenth process of Figure 8 is involved with the alarm system control, which shows the number of sensor values being simultaneously out of range, whether they exceed maximum or minimum of set values, as we discuss below for Figure 10.
Figure 9 presents a more detailed flowchart view of visible light controls. Needless to say that except from the step motor and buzzer operation, all the controls presented in this figure are the same for all sensors used here. It presents controls of minimum, maximum and time duration of the set range values and activation of corresponding LEDs. It also shows the activation of step motor system rotating, clockwise for lower than minimum set input values and counterclockwise for higher than maximum set input values, thus increasing or decreasing visible light illuminance, respectively, by opening or closing curtains or the roof covering of the inspected area. Finally, Figure 9 shows that all FPGA board LEDS simultaneously with the buzzer system are ON, if input visible light illuminance value is higher than maximum set value.
Figure 10 shows a flowchart of the alarm system control operation, which in conjunction with all other control systems, completes the visualized view of system's condition. The above alarm system, also shown in Figures 1 and 2, uses five different color LEDs, green, blue, white, yellow and red. Its main function is to present the alarm level of the system after checking algorithmically the number of sensors' input values that are simultaneously out of programmer set range, by using a serial control algorithm. If all sensors' input values are within set range, there is no alarm for the system and only green LED is ON by receiving bit1 from the FPGA board. If one sensor input value is out of range then only blue LED is ON. Similarly only white LED is ON if two sensors' input values are simultaneously out of set range, and only yellow LED is ON if three sensors' input values are at the same time, out of set range. Finally, red alarm condition is reached and only red LED is ON, in the case of all four sensors' input values being simultaneously out of set range.
If one or more sensors' input values turn into set range, the system alarm level subsequently lowers.

CONCLUSION

An FPGA-based system is presented here for measuring, displaying and controlling four sensors' input values. Visible light illuminance, infrared light intensity, temperature and humidity of an inspected area are measured and controlled, in order to maintain their values within programmer's set range values. Our system is also capable of setting and controlling time duration values for all the above parameters. Corresponding control systems and respective LEDs are activated whenever one or more sensor input values are lower than minimum set values or higher than maximum set values. Another control system that is activated as soon as our system starts operating is the alarm level control system, which informs user about the number of sensors' input values that are simultaneously out of set range. Our system's main advantages are based on its low cost, easy expansion for larger area use or alternative sensor incorporation, and the ability to visualize and self-regulate the system. Our system is programed using VHDL language and can be implemented in a variety of inspected areas, such as greenhouses, or industrial areas.

REFERENCES

1. M. Yussup, M.M. Ibrahim, L. Lombigit, N.A.A. Rahman and M.R.M. Zin, "Implementation of data acquisition interface using on-board field-programmable gate array (FPGA) universal serial bus (USB) link", Advanc. in Nuclear Research and Energy Development, AIP Conf. Proc. 1584, 69-72 (2014), doi: 10.1063/1.4866106
2. A. Gujar, International Journal of Computer Science and Information Technologies, "Image Encryption using AES Algorithm based on FPGA", vol 5, (5), 2014, 6853-6859

DOI: 10.17148/IJIREEICE.2024.121205

3. S. Singh, A.K. Saini, R. Saini, I.J. Image, Graphics and Signal Processing, “Interfacing the Analog Camera with FPGA Board for Real-time Video Acquisition” 2014, 4, 32-38, DOI: 10.5815/ijigsp.2014.04.04
4. S. Muthukrishnan and R. Priyadharsini, International Journal of Computer Science and Mobile Computing, “32-Bit RISC and DSP System Design in an FPGA” vol 3, issue 12, Dec. 2014, pg. 361-368
5. L. Chen, Y. Chang, L. Yan, IEEE Transactions on Geoscience and Remote Sensing, “On-orbit real-time variational image destriping: FPGA architecture and implementation”, 10.1109/tgrs.2022.3140428, 2022, pp. 1-1
6. S. Yarlagadda, S. Kaza, A. Tummala, E. Babu, R. Prabhakar, Information Technology in Industry, “The reduction of Crosstalk in VLSI due to parallel bus structure using Data Compression Bus Encoding technique implemented on Artix 7 FPGA Architecture”, 10.17762/itii.v9i1.151, 2021, Vol 9 (1), pp. 456-460
7. C. Du, Y. Yamaguchi, Electronics, “High-Level Synthesis Design for Stencil Computations on FPGA with High Bandwidth Memory”, 2020, 9(8), 1275; https://doi.org/10.3390/electronics9081275
8. X. Hao, C. Lin and Q. Wu, Electronics, “A Parallel Timing Synchronization Structure in Real-Time High Transmission Capacity Wireless Communication Systems”, 2020, 9(4), 652; https://doi.org/10.3390/electronics9040652
9. P. A. Bawiskar, R.K. Agrawal, International Journal of Innovative Research in Science, Engineering and Technology, “FPGA Based Home Security System” vol.4, issue 12, Dec. 2015, p. 12865-12869, DOI: 10.15680/IJIRSET.2015.0412139
10. K. Saroch, A. Sharma, IOSR Journal of Electronics and Communication Engineering, “FPGA Based System Login Security Lock Design Using Finite State Machine” vol 5, issue 3, Mar.-Apr. 2013, pp 70-75
11. R.S. Parikh, Int. Journal of Engineering Research and Application, “Alarm System Implementation on Field Programmable Gate Array” vol 8, issue 1, Jan. 2018, pp 01-04.
12. E. I. Dimitriadis and L. Dimitriadis, “A Simple, Low Cost and Multiple Input Alarm System, Functioning as Finite State Machine (FSM), Using VHDL and FPGAs”, Journal of Active and Passive Electronic Devices, vol. 17, pp. 307–315, 2024.
13. E. I. Dimitriadis and L. Dimitriadis, “A g-sensor based alarm system, for multiple tilt sensor applications, using VHDL and FPGAs”, Journal of Active and Passive Electronic Devices, vol. 18, pp. 119-129, 2024.
14. L. Bajer and O. Krejcar, “Design and realization of low cost control for greenhouse environment with remote control”, IFAC-PapersOnLine, vol. 48-4, pp. 368-373, 2015.
15. N. Thirer and I. Uchansky, “An FPGA based computer system for greenhouse control”, Athens Journal of Sciences, vol. 2, pp. 23-32, 2015.
16. C. Lachouri, K. Mansouri, A. Belmeguenai and M. Lafifi, “FPGA implementation of adaptive neuro-fuzzy inference systems controller for greenhouse climate”, International Journal of Advanced Computer Science and Applications, vol. 7, no. 1, pp. 261-266, 2016.
17. A. Kavga, F. Evangelopoulou, C. Koulopoulou, M. Zografou and I. Lycoskoufis, “Effects of infrared radiation (IR) on growth parameters of eggplant cultivation and greenhouse energy efficiency”. Acta Hortic. 1296, pp.203-210, 2020DOI: 10.17660/ActaHortic.2020.1296.26
18. T. Goldammer, “Greenhouse Management. A Guide to Operations and Technology”, Apex Publishers, ISBN (13): 979-8-89342-168-2, 2024

Figure captions

Figure 1: Device overview and operational units of the system presented in this work, using FPGA DE10-Lite board.
Figure 2: Circuit diagram of the system used in this work.
Figure 3: System at red alarm condition, with 4 sensors’ values out of range.
Figure 4: System at red alarm condition, with 4 sensors’ values out of range and visible light maximum value exceeded.
Figure 5: System at yellow alarm condition, with 3 sensors’ values out of range.
Figure 6: System at white alarm condition, with 2 sensors’ values out of range.
Figure 7: Time values set by the programmer have been exceeded. All sensors control systems have yellow LEDs ON.
Figure 8: Flowchart of system operation.
Figure 9: Visible light control system flowchart. Same for three other sensors, except for step motor and buzzer-led activation.
Figure 10: Alarm level system control flowchart.

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.021 ⁙ Peer-reviewed & Refereed journal ⁙ Vol. 12, Issue 12, December 2024

DOI: 10.17148/IJIREEICE.2024.121205

Figures

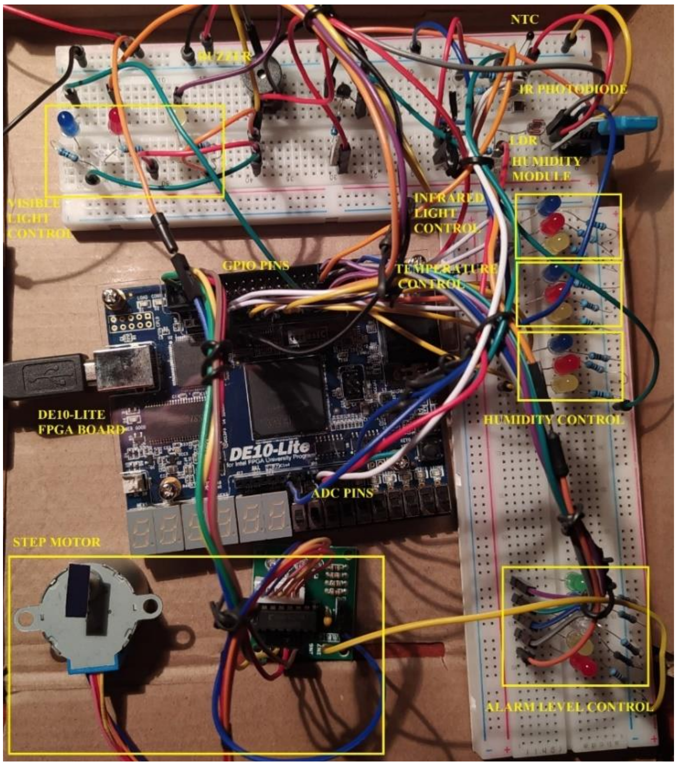

Figure 1

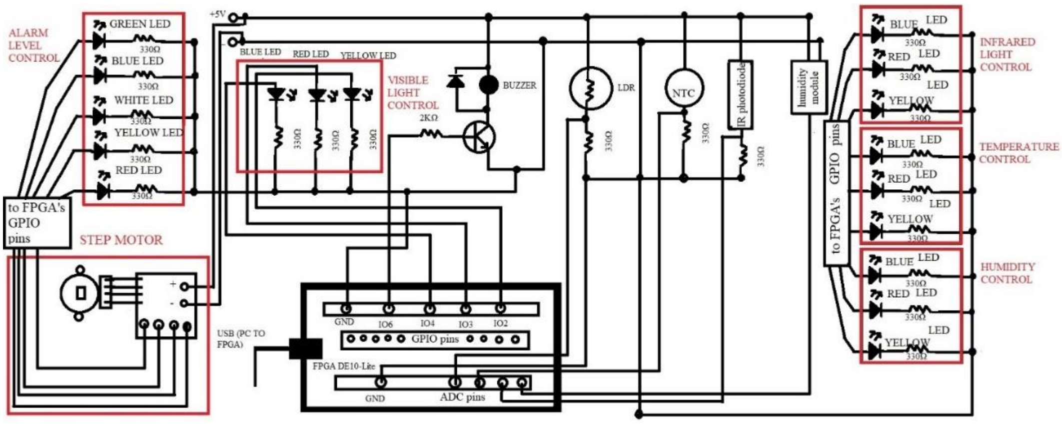

Figure 2

DOI: 10.17148/IJIREEICE.2024.121205

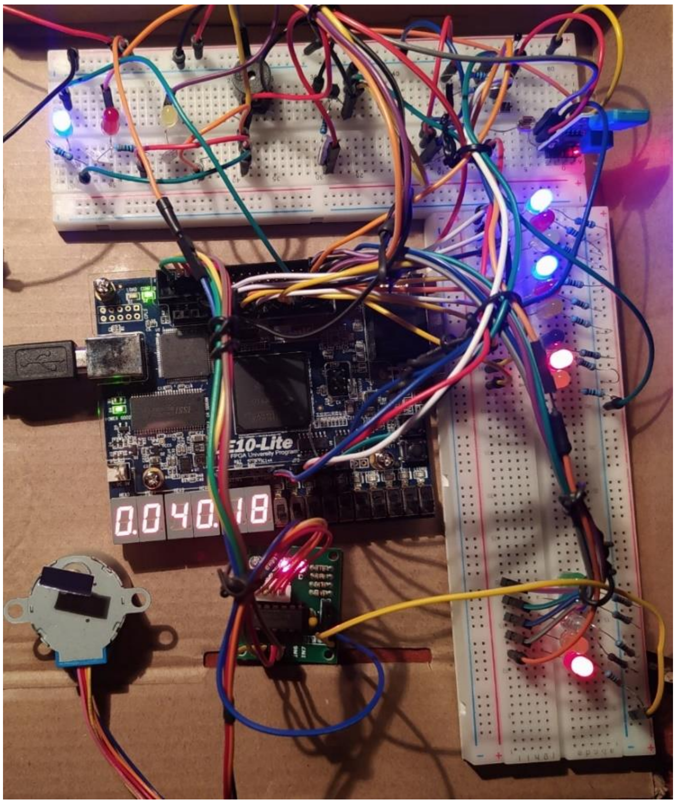

Figure 3

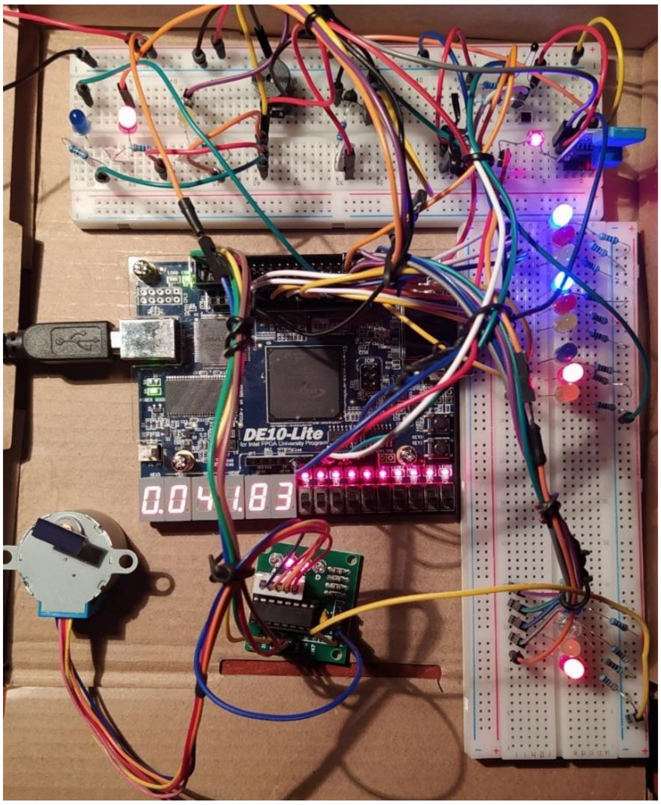

Figure 4

DOI: 10.17148/IJIREEICE.2024.121205

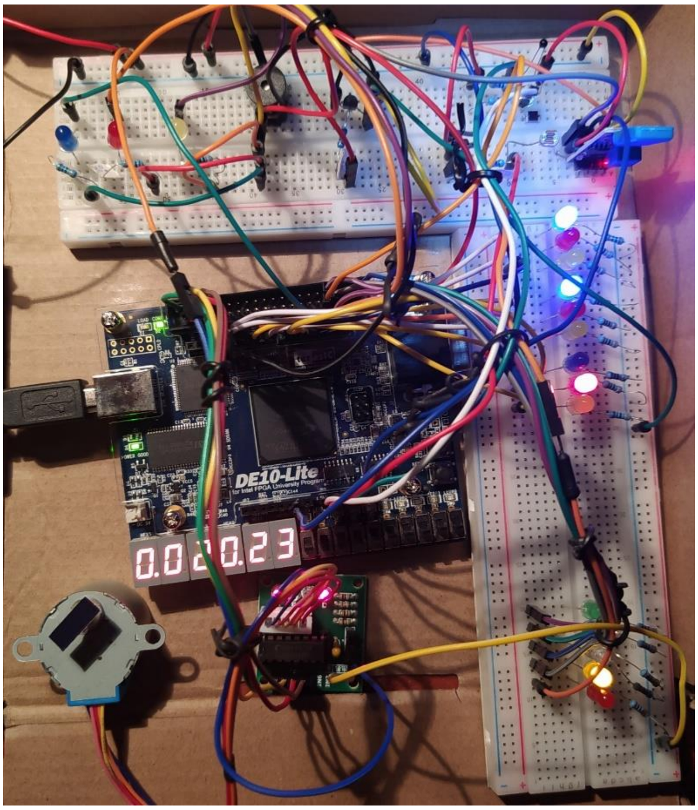

Figure 5

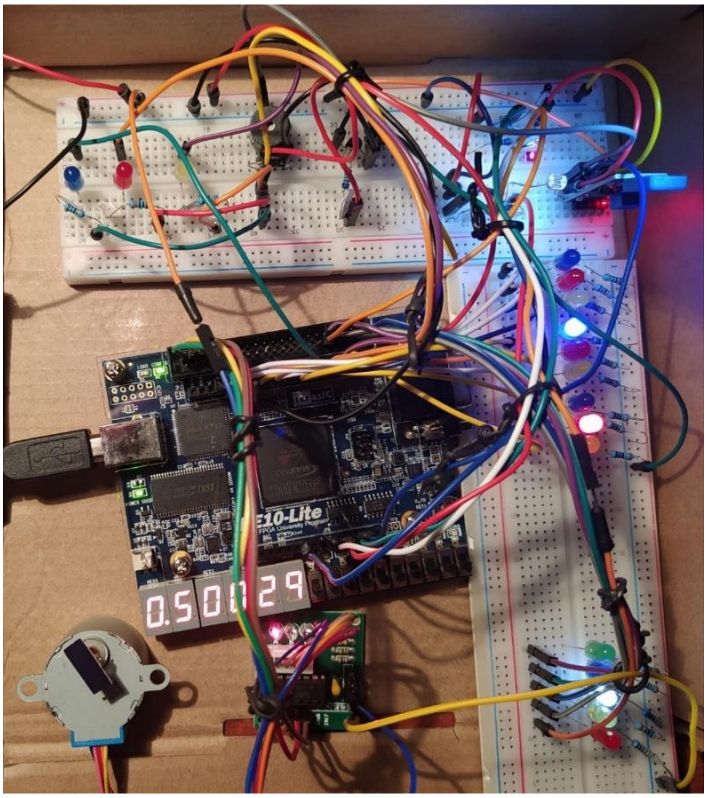

Figure 6

DOI: 10.17148/IJIREEICE.2024.121205

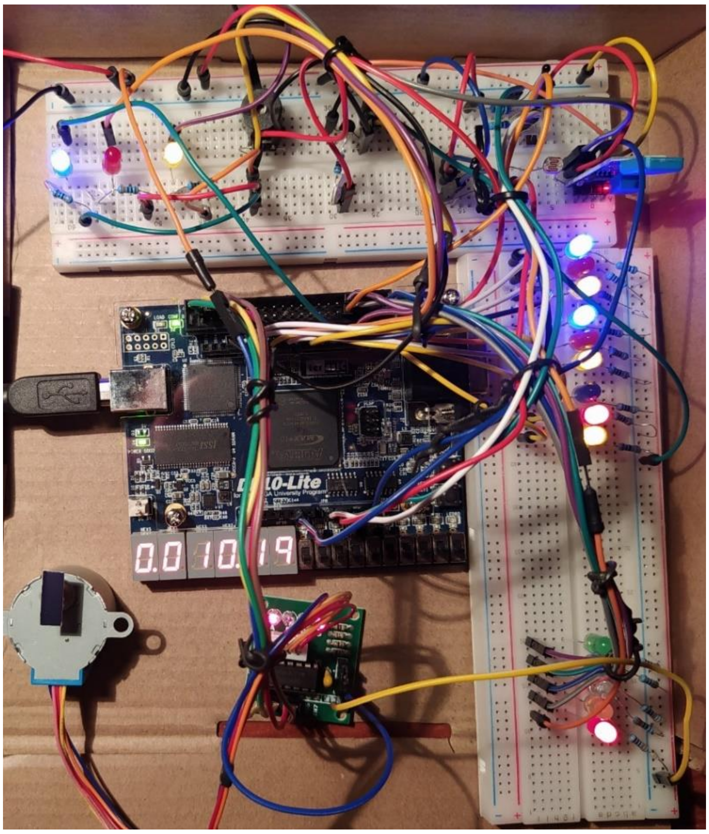

Figure 7

DOI: 10.17148/IJIREEICE.2024.121205

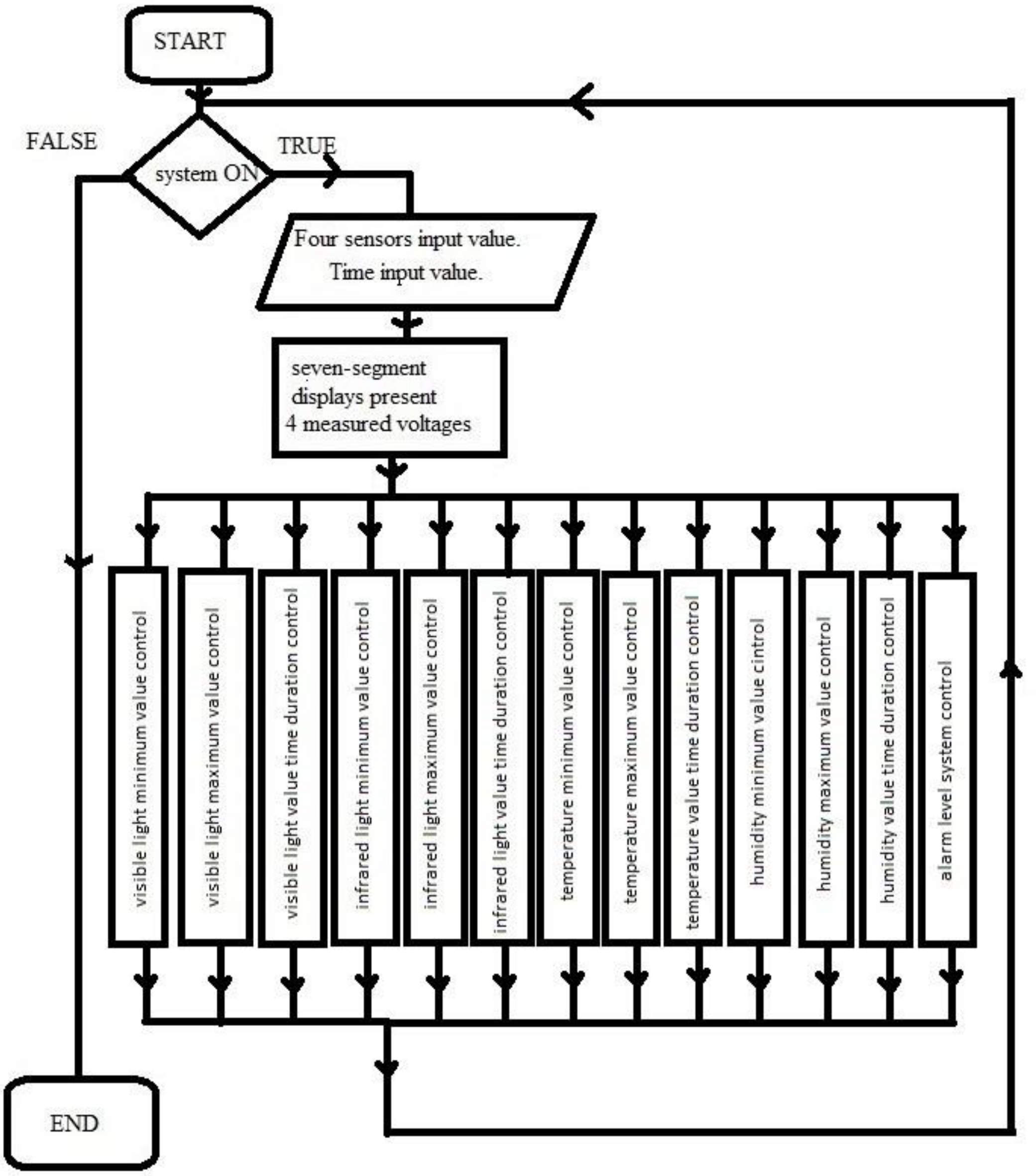

Figure 8

DOI: 10.17148/IJIREEICE.2024.121205

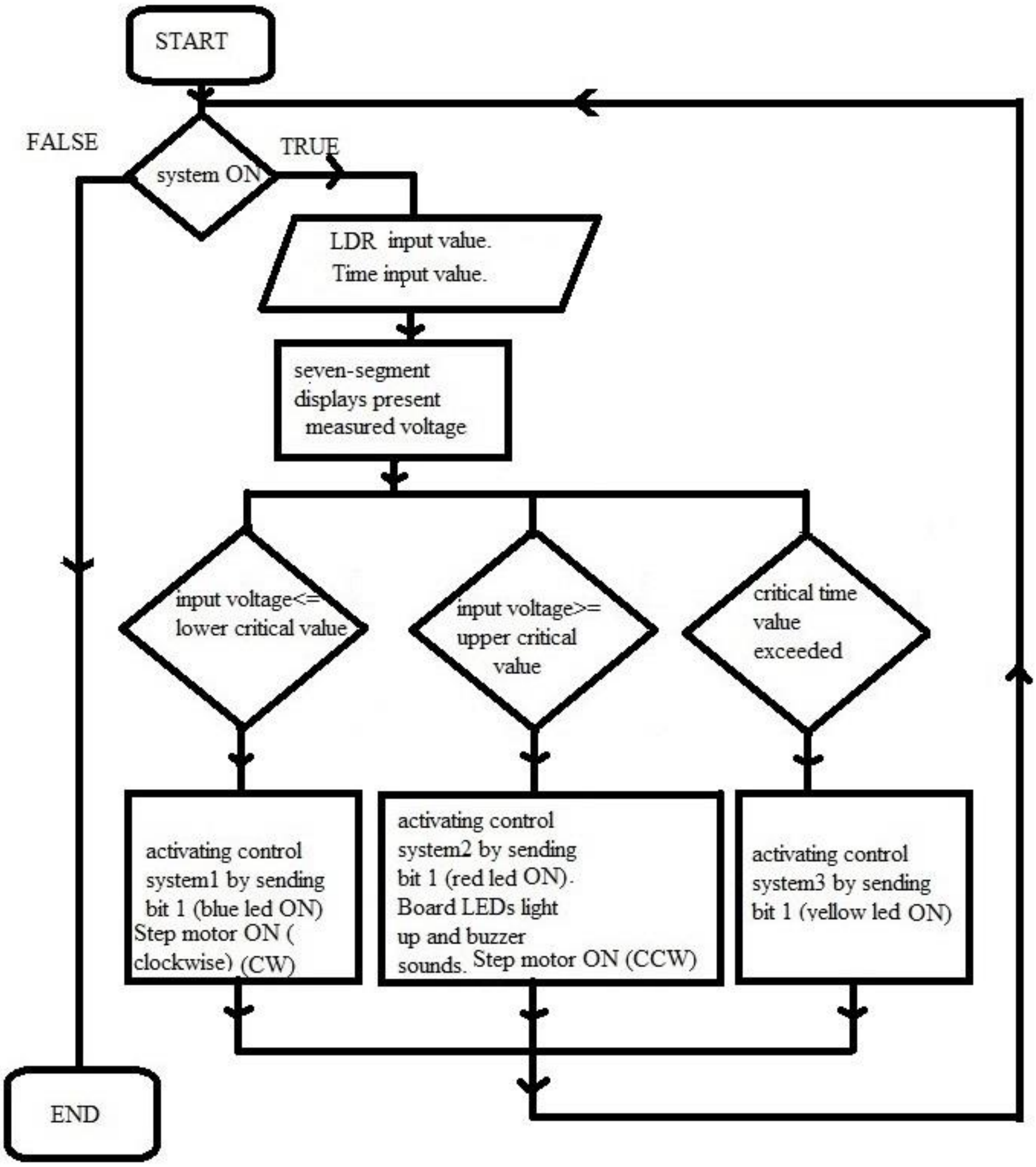

Figure 9

DOI: 10.17148/IJIREEICE.2024.121205

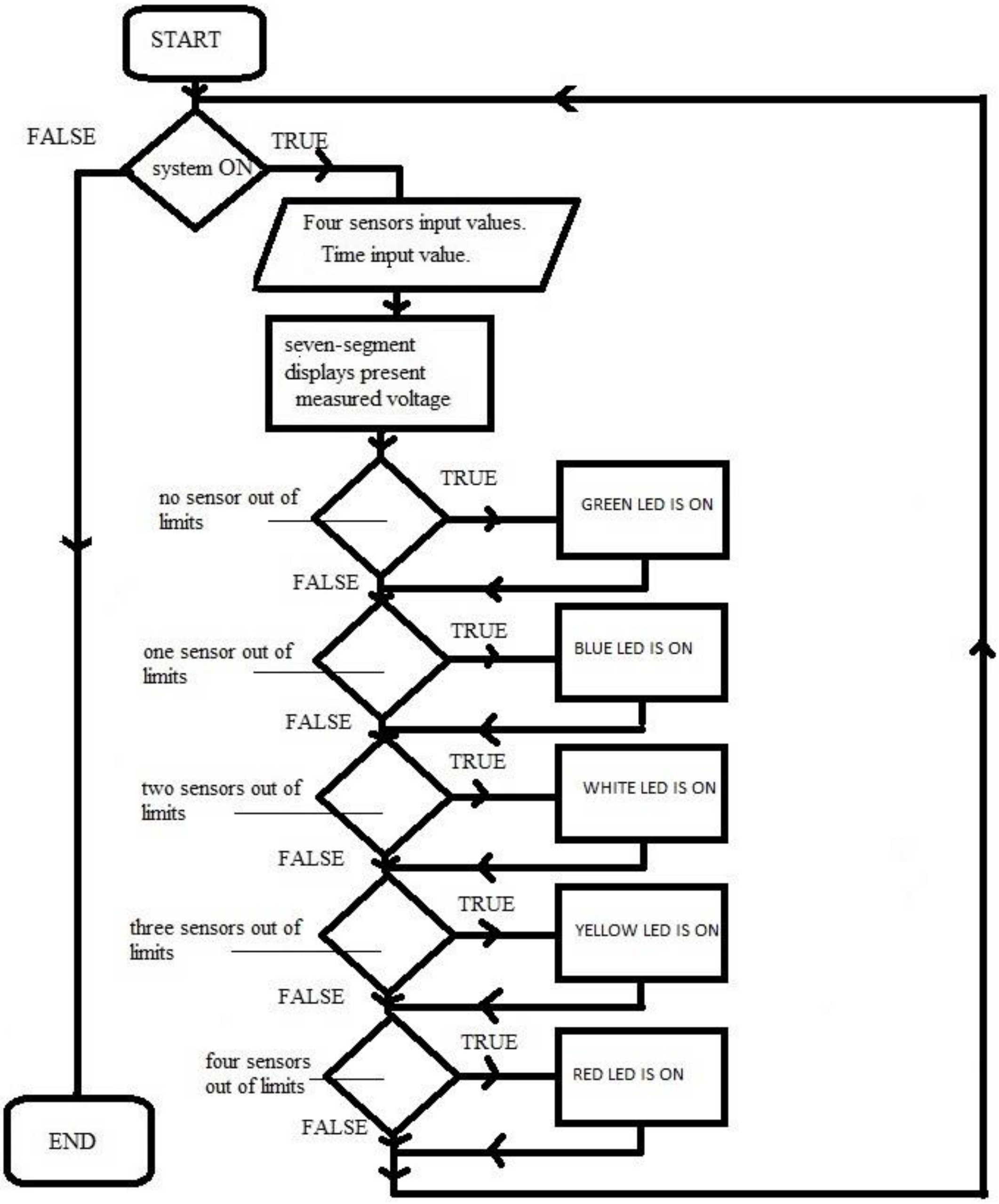

Figure 10

DOI: 10.17148/IJIREEICE.2024.121205

APPENDIX

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.ALL;    -- step = step + 1
entity DE10_Lite_ADC_sensors is
generic(ClockFrequencyHz : integer:=50000000;
wait_count : natural := 1250000);  -- 50000000=1sec
wait time for the stepper
port
(rst : in std_logic; --SW8
coils : out std_logic_vector(3 downto 0); -- connected
to IN1..IN4
nRst    : in std_logic; -- Negative reset
Seconds : inout integer;
led1: out std_logic;
led2: out std_logic;
led3: out std_logic;
led4: out std_logic;
led5: out std_logic;
led6: out std_logic;
led7: out std_logic;
led8: out std_logic;
led9: out std_logic;
led10: out std_logic;
led_blue: buffer std_logic;
led_red: buffer std_logic;
led_yellow: buffer std_logic;
led_blue_out: out std_logic;
led_red_out: out std_logic;
led_yellow_out: out std_logic;
led_blue_infr: buffer std_logic;
led_red_infr: buffer std_logic;
led_yellow_infr: buffer std_logic;
led_blue_out_infr: out std_logic;
led_red_out_infr: out std_logic;
led_yellow_out_infr: out std_logic;
led_blue_temp: buffer std_logic;
led_red_temp: buffer std_logic;
led_yellow_temp: buffer std_logic;
led_blue_out_temp: out std_logic;
led_red_out_temp: out std_logic;
led_yellow_out_temp: out std_logic;
led_blue_hum: buffer std_logic;
led_red_hum: buffer std_logic;
led_yellow_hum: buffer std_logic;
led_blue_out_hum: out std_logic;
led_red_out_hum: out std_logic;
led_yellow_out_hum: out std_logic;
buzzer:out std_logic;
Vr: buffer integer;
Vinfr: buffer integer;
Vtemp: buffer integer;
Vhum: buffer integer;
d2abuf :buffer integer range 0 to 9;
d1abuf :buffer integer range 0 to 9;
d0abuf :buffer integer range 0 to 9;
d2bbuf :buffer integer range 0 to 9;
d1bbuf :buffer integer range 0 to 9;
d0bbuf :buffer integer range 0 to 9;
d2cbuf :buffer integer range 0 to 9;
d1cbuf :buffer integer range 0 to 9;
d0cbuf :buffer integer range 0 to 9;
d2dbuf :buffer integer range 0 to 9;
d1dbuf :buffer integer range 0 to 9;
d0dbuf :buffer integer range 0 to 9;
SW0 : in std_logic;
red_led_alarm : out std_logic;
yellow_led_alarm: out std_logic;
white_led_alarm: out std_logic;
blue_led_alarm: out std_logic;
green_led_no_alarm: out std_logic;
red_led_alarm_buff : buffer std_logic;
yellow_led_alarm_buff: buffer std_logic;
white_led_alarm_buff: buffer std_logic;
blue_led_alarm_buff: buffer std_logic;
green_led_no_alarm_buff: buffer std_logic;
-- Clocks
ADC_CLK_10: in std_logic;
MAX10_CLK1_50: in std_logic;
MAX10_CLK2_50: in std_logic;
-- KEYs
KEY: in std_logic_vector(1 downto 0);
-- HEX
HEX0: out std_logic_vector(7 downto 0);
HEX1: out std_logic_vector(7 downto 0);
HEX2: out std_logic_vector(7 downto 0);
HEX3: out std_logic_vector(7 downto 0);
HEX4: out std_logic_vector(7 downto 0);
HEX5: out std_logic_vector(7 downto 0);
ARDUINO_IO: inout std_logic_vector(15 downto 0);
ARDUINO_RESET_N: inout std_logic);
-- GPIO
--GPIO: inout std_logic_vector(35 downto 0));
end entity;
architecture DE10_Lite_ADC_sensors_Arch of
DE10_Lite_ADC_sensors is
-- Analog to Digital Converter IP core
component myADC is
port(clk_clk: in std_logic := 'X';
modular_adc_0_command_valid: in std_logic := 'X';
modular_adc_0_command_channel: in std_logic_vec-
tor(4 downto 0) := (others => 'X');
modular_adc_0_command_startofpacket: in std_logic
:= 'X';
modular_adc_0_command_endofpacket: in std_logic
:= 'X';
modular_adc_0_command_ready: out std_logic;
modular_adc_0_response_valid: out std_logic;
modular_adc_0_response_channel: out std_logic_vec-
tor(4 downto 0);
```

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526
International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering
Impact Factor 8.021 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 12, Issue 12, December 2024
DOI: 10.17148/IJIREEICE.2024.121205

```
modular_adc_0_response_data: out std_logic_vec-
tor(11 downto 0);
modular_adc_0_response_startofpacket: out std_logic;
modular_adc_0_response_endofpacket: out std_logic;
reset_reset_n: in std_logic
);
end component myADC;
signal modular_adc_0_command_valid: std_logic;
signal modular_adc_0_command_channel:
std_logic_vector(4 downto 0);
signal modular_adc_0_command_startofpacket:
std_logic;
signal modular_adc_0_command_endofpacket:
std_logic;
signal modular_adc_0_command_ready: std_logic;
signal modular_adc_0_response_valid: std_logic;
signal modular_adc_0_response_channel:
std_logic_vector(4 downto 0);
signal modular_adc_0_response_data: std_logic_vec-
tor(11 downto 0);
signal modular_adc_0_response_startofpacket:
std_logic;
signal modular_adc_0_response_endofpacket:
std_logic;
signal clk_clk: std_logic;
signal reset_reset_n: std_logic;
type state_machines is (sm0, sm1, sm2, sm3, sm4);
signal sm: state_machines;
-- signals to store conversion results
signal ADCIN1, ADCIN2, ADCIN3, ADCIN4:
std_logic_vector(11 downto 0);
signal AD1, AD2, AD3, AD4: std_logic_vector(11
downto 0);
-- signals for BCD digits
signal digit2a, digit1a, digit0a: std_logic_vector(3
downto 0);
signal digit2b, digit1b, digit0b: std_logic_vector(3
downto 0);
signal digit2c, digit1c, digit0c: std_logic_vector(3
downto 0);
signal digit2d, digit1d, digit0d: std_logic_vector(3
downto 0);
signal digit5, digit4, digit3, digit2, digit1, digit0:
std_logic_vector(3 downto 0);
-- signal to determine how fast the
-- 7-seg displays will be updated
signal cnt: integer;
signal state_LED: std_logic;
signal state_Vr: integer;
signal state_Vinfr: integer;
signal state_Vtemp: integer;
signal state_Vhum: integer;
signal Ticks : integer;
-- signal for step motor control
signal count : natural range 0 to wait_count;
begin
-- ADC port map
adc1: myADC port map
(modular_adc_0_command_valid => modu-
lar_adc_0_command_valid,
modular_adc_0_command_channel => modu-
lar_adc_0_command_channel,
modular_adc_0_command_startofpacket => modu-
lar_adc_0_command_startofpacket,
modular_adc_0_command_endofpacket => modu-
lar_adc_0_command_endofpacket,
modular_adc_0_command_ready => modu-
lar_adc_0_command_ready,
modular_adc_0_response_valid => modular_adc_0_re-
sponse_valid,
modular_adc_0_response_channel => modu-
lar_adc_0_response_channel,
modular_adc_0_response_data => modular_adc_0_re-
sponse_data,
modular_adc_0_response_startofpacket => modu-
lar_adc_0_response_startofpacket,
modular_adc_0_response_endofpacket => modu-
lar_adc_0_response_endofpacket,
clk_clk => clk_clk,
reset_reset_n => reset_reset_n
);
clk_clk <= MAX10_CLK1_50;
reset_reset_n <= KEY(0);
-- process for reading new samples
p1: process(reset_reset_n, clk_clk)
begin
if reset_reset_n = '0' then
        sm <= sm0;
elsif rising_edge(clk_clk) then
        case sm is
                when sm0 =>
                sm <= sm1;
        modular_adc_0_command_valid <= '1';
        modular_adc_0_command_channel <=
"00001";
        when sm1 =>
    if modular_adc_0_response_valid = '1' then
        modular_adc_0_command_channel <=
"00010";
        ADCIN4 <= modular_adc_0_response_data;
                sm <= sm2;
                        end if;
                when sm2 =>
if modular_adc_0_response_valid = '1' then
modular_adc_0_command_channel <= "00011";
ADCIN1 <= modular_adc_0_response_data;
                    sm <= sm3;
                end if;
                when sm3 =>
if modular_adc_0_response_valid = '1' then
modular_adc_0_command_channel <= "00100";
ADCIN2 <= modular_adc_0_response_data;
                sm <= sm4;
                end if;
                when sm4 =>
if modular_adc_0_response_valid = '1' then
```

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering
Impact Factor 8.021 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 12, Issue 12, December 2024
DOI: 10.17148/IJIREEICE.2024.121205

```
            modular_adc_0_command_channel <=
"00001";
            ADCIN3 <= modular_adc_0_response_data;
                        sm <= sm1;
                        end if;
                        when others =>
            end case;
end if;
end process;
-- process for conversion from binary to BCD (first an-
alog voltage)
p2: process(AD1, d2abuf, d1abuf, d0abuf)
variable vin: integer;
variable d2, d1, d0: integer;
begin
vin := to_integer(unsigned(std_logic_vector(to_un-
signed(to_integer(unsigned(AD1)) * 500,
32))(31 downto 12)));
d2 := vin / 100;
d1 := vin mod 100 / 10;
d0 := ((vin mod 100) mod 10);
digit2a <= std_logic_vector(to_unsigned(d2, 4));
digit1a <= std_logic_vector(to_unsigned(d1, 4));
digit0a <= std_logic_vector(to_unsigned(d0, 4));
d2abuf<= d2;
d1abuf<= d1;
d0abuf<= d0;
end process;
-- process for conversion from binary to BCD (second
analog voltage)
p3: process(AD2,d2bbuf,d1bbuf,d0bbuf)
variable vin: integer;
variable d2, d1, d0: integer;
begin
vin := to_integer(unsigned(std_logic_vector(to_un-
signed(to_integer(unsigned(AD2)) * 500,
32))(31 downto 12)));
d2 := vin / 100;
d1 := vin mod 100 / 10;
d0 := ((vin mod 100) mod 10);
digit2b <= std_logic_vector(to_unsigned(d2, 4));
digit1b <= std_logic_vector(to_unsigned(d1, 4));
digit0b <= std_logic_vector(to_unsigned(d0, 4));
d2bbuf<= d2;
d1bbuf<= d1;
d0bbuf<= d0;
end process;
-- process for conversion from binary to BCD (third
analog voltage)
p5: process(AD3,d2cbuf,d1cbuf,d0cbuf)
variable vin: integer;
variable d2, d1, d0: integer;
begin
vin := to_integer(unsigned(std_logic_vector(to_un-
signed(to_integer(unsigned(AD3)) * 500,
32))(31 downto 12)));
d2 := vin / 100;
d1 := vin mod 100 / 10;
d0 := ((vin mod 100) mod 10);
digit2c <= std_logic_vector(to_unsigned(d2, 4));
digit1c <= std_logic_vector(to_unsigned(d1, 4));
digit0c <= std_logic_vector(to_unsigned(d0, 4));
d2cbuf<= d2;
d1cbuf<= d1;
d0cbuf<= d0;
end process;
-- process for conversion from binary to BCD (fourth
analog voltage)
p6: process(AD4,d2dbuf,d1dbuf,d0dbuf)
variable vin: integer;
variable d2, d1, d0: integer;
begin
vin := to_integer(unsigned(std_logic_vector(to_un-
signed(to_integer(unsigned(AD4)) * 500,
32))(31 downto 12)));
d2 := vin / 100;
d1 := vin mod 100 / 10;
d0 := ((vin mod 100) mod 10);
digit2d <= std_logic_vector(to_unsigned(d2, 4));
digit1d <= std_logic_vector(to_unsigned(d1, 4));
digit0d <= std_logic_vector(to_unsigned(d0, 4));
d2dbuf<= d2;
d1dbuf<= d1;
d0dbuf<= d0;
end process;
state_Vr<= (d2bbuf*100)+(d1bbuf*10)+(d0bbuf);
Vr<= state_Vr;
state_Vinfr<= (d2abuf*100)+(d1abuf*10)+(d0abuf);
Vinfr<= state_Vinfr;
state_Vtemp<= (d2cbuf*100)+(d1cbuf*10)+(d0cbuf);
Vtemp<= state_Vtemp;
state_Vhum<= (d2dbuf*100)+(d1dbuf*10)+(d0dbuf);
Vhum<= state_Vhum;

-- determine how fast the 7-seg displays will be up-
dated
p4: process(reset_reset_n, clk_clk)
begin
if reset_reset_n = '0' then
cnt <= 0;
elsif rising_edge(clk_clk) then
if cnt < 20_000_000 then
cnt <= cnt + 1;
else
cnt <= 0;
AD1 <= ADCIN1;
AD2 <= ADCIN2;
AD3 <= ADCIN3;
AD4 <= ADCIN4;
end if;
end if;
end process;
--time-seconds
process(MAX10_CLK1_50) is
   begin
      if rising_edge(MAX10_CLK1_50) then
```

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering
Impact Factor 8.021 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 12, Issue 12, December 2024
DOI: 10.17148/IJIREEICE.2024.121205

```
    -- If the negative reset signal is active
        if nRst = '0' then
            Ticks   <= 0;
            Seconds <= 0;
        else
            -- True once every second
            if Ticks = ClockFrequencyHz - 1 then
                Ticks <= 0;
                    Seconds <= Seconds + 1;
            else
                Ticks <= Ticks + 1;
            end if;
        end if;
      end if;
   end process;
 --critical Vr value exceeded buzzer sounds
Process(MAX10_CLK1_50,Vr)
variable i : integer := 0;
BEGIN
IF Vr>=100 THEN
if MAX10_CLK1_50'event and MAX10_CLK1_50 =
'1' then
if i <= 50000000 then
i := i + 1;
buzzer <= '1';
elsif i > 50000000 and i < 100000000 then
i := i + 1;
buzzer <= '0';
elsif i = 100000000 then
i := 0;
end if;
end if;
end if;
end process;
--critical visible light Vr value exceeded board LEDs
and external red led lights up
process(state_LED,led_red,Vr)
begin
  IF Vr>=100 THEN
    led_red <= '1';
            state_LED <= '1';
    else
    led_red <= '0';
            state_LED <= '0';
  end if;
end process;
led_red_out<= led_red;
led1 <= state_LED;
led2 <= state_LED;
led3 <= state_LED;
led4 <= state_LED;
led5 <= state_LED;
led6 <= state_LED;
led7 <= state_LED;
led8 <= state_LED;
led9 <= state_LED;
led10 <= state_LED;
--Visible light Vr lower than low critical value then ex-
ternal blue led lights up
process(led_blue,Vr)
begin
  IF Vr<=20 THEN
    led_blue <= '1';
    else
    led_blue <= '0';
  end if;
end process;
led_blue_out<= led_blue;
--Time-seconds exceeds critical value for visible light
lighting then yellow led lights up
process(led_yellow,Seconds)
begin
  IF Seconds>=60 THEN
    led_yellow <= '1';
    else
    led_yellow <= '0';
  end if;
end process;
led_yellow_out<= led_yellow;
--critical Vinfr value exceeded board LEDs and exter-
nal red led light up
process(led_red_infr,Vinfr)
begin
  IF Vinfr>=100 THEN
    led_red_infr <= '1';
    else
    led_red_infr <= '0';
  end if;
end process;
led_red_out_infr<= led_red_infr;
--Vinfr lower than low critical value then external blue
led lights up
process(led_blue_infr,Vinfr)
begin
  IF Vinfr<=20 THEN
    led_blue_infr <= '1';
    else
    led_blue_infr <= '0';
  end if;
end process;
led_blue_out_infr<= led_blue_infr;
--Time-seconds exceeds critical value for infrared
lighting then yellow led lights up
process(led_yellow_infr,Seconds)
begin
  IF Seconds>=60 THEN
    led_yellow_infr <= '1';
    else
    led_yellow_infr <= '0';
  end if;
end process;
led_yellow_out_infr<= led_yellow_infr;
--critical Vtemp value exceeded board LEDs and exter-
nal red led light up
process(led_red_temp,Vtemp)
```

DOI: 10.17148/IJIREEICE.2024.121205

```
begin
 IF Vtemp>=100 THEN
  led_red_temp <= '1';
  else
  led_red_temp <= '0';
 end if;
end process;
led_red_out_temp<= led_red_temp;
--Vtemp lower than low critical value then external
blue led lights up
process(led_blue_temp,Vtemp)
begin
 IF Vtemp<=20 THEN
  led_blue_temp <= '1';
  else
  led_blue_temp <= '0';
 end if;
end process;
led_blue_out_temp<= led_blue_temp;
--Time-seconds exceeds critical value for this tempera-
ture then yellow led lights up
process(led_yellow_temp,Seconds)
begin
 IF Seconds>=60 THEN
  led_yellow_temp <= '1';
  else
  led_yellow_temp <= '0';
 end if;
end process;
led_yellow_out_temp<= led_yellow_temp;
--critical Vhum value exceeded board LEDs and exter-
nal red led light up
process(led_red_hum,Vhum)
begin
 IF Vhum>=100 THEN
  led_red_hum <= '1';
  else
  led_red_hum <= '0';
 end if;
end process;
led_red_out_hum<= led_red_hum;
--Vhum lower than low critical value then external
blue led lights up
process(led_blue_hum,Vhum)
begin
 IF Vhum<=20 THEN
  led_blue_hum <= '1';
  else
  led_blue_hum <= '0';
 end if;
end process;
led_blue_out_hum<= led_blue_hum;
--Time-seconds exceeds critical value for this humidity
then yellow led lights up
process(led_yellow_hum,Seconds)
begin
 IF Seconds>=60 THEN
  led_yellow_hum <= '1';
  else
  led_yellow_hum <= '0';
 end if;
end process;
led_yellow_out_hum<= led_yellow_hum;
--alarm level
process ( red_led_alarm_buff,yel-
low_led_alarm_buff,white_led_alarm_buff,
blue_led_alarm_buff,green_led_no_alarm_buff,Vr,Vin
fr,Vtemp,Vhum)
begin
IF (Vr<100 AND Vr>20) AND (Vinfr<100 AND
Vinfr>20) AND
(Vtemp<100 AND Vtemp>20) AND (Vhum<100
AND Vhum>20) THEN
green_led_no_alarm_buff <= '1';
blue_led_alarm_buff <= '0';
white_led_alarm_buff <= '0';
yellow_led_alarm_buff <= '0';
red_led_alarm_buff <= '0';
elsif ((Vr>=100 OR Vr<=20) AND (Vinfr<100 AND
Vinfr>20) AND
(Vtemp<100 AND Vtemp>20) AND (Vhum<100
AND Vhum>20)) OR
((Vr<100 AND Vr>20) AND (Vinfr>=100 OR
Vinfr<=20) AND
(Vtemp<100 AND Vtemp>20) AND (Vhum<100
AND Vhum>20)) OR
((Vr<100 AND Vr>20) AND (Vinfr<100 AND
Vinfr>20) AND
(Vtemp>=100 OR Vtemp<=20) AND (Vhum<100
AND Vhum>20)) OR
((Vr<100 AND Vr>20) AND (Vinfr<100 AND
Vinfr>20) AND
(Vtemp<100 AND Vtemp>20) AND (Vhum>=100 OR
Vhum<=20))
THEN
blue_led_alarm_buff <= '1';
green_led_no_alarm_buff <= '0';
white_led_alarm_buff <= '0';
yellow_led_alarm_buff <= '0';
red_led_alarm_buff <= '0';
elsif ((Vr>=100 OR Vr<=20) AND (Vinfr>=100 OR
Vinfr<=20) AND
(Vtemp<100 AND Vtemp>20) AND (Vhum<100
AND Vhum>20)) OR
((Vr>=100 OR Vr<=20) AND (Vinfr<100 AND
Vinfr>20) AND
(Vtemp>=100 OR Vtemp<=20) AND (Vhum<100
AND Vhum>20)) OR
((Vr>=100 OR Vr<=20) AND (Vinfr<100 AND
Vinfr>20) AND
(Vhum>=100 OR Vhum<=20)) OR ((Vr<100 AND
Vr>20) AND
(Vinfr>=100 OR Vinfr<=20) AND (Vtemp>=100 OR
Vtemp<=20) AND
(Vhum<100 AND Vhum>20))
```

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering
Impact Factor 8.021 ⁘ Peer-reviewed & Refereed journal ⁘ Vol. 12, Issue 12, December 2024
DOI: 10.17148/IJIREEICE.2024.121205

```
OR ((Vr<100 AND Vr>20) AND (Vinfr>=100 OR
Vinfr<=20) AND
(Vtemp<100 AND Vtemp>20) AND (Vhum>=100 OR
Vhum<=20)) OR
((Vr<100 AND Vr>20) AND (Vinfr<100 AND
Vinfr>20) AND
(Vtemp>=100 OR Vtemp<=20) AND (Vhum>=100
OR Vhum<=20))
 THEN
white_led_alarm_buff <= '1';
green_led_no_alarm_buff <= '0';
blue_led_alarm_buff <= '0';
yellow_led_alarm_buff <= '0';
red_led_alarm_buff <= '0';
elsif ((Vr>=100 OR Vr<=20) AND (Vinfr>=100 OR
Vinfr<=20) AND
(Vtemp>=100 OR Vtemp<=20) AND (Vhum<100
AND Vhum>20)) OR
((Vr>=100 OR Vr<=20) AND (Vinfr<100 AND
Vinfr>20) AND
(Vtemp>=100 OR Vtemp<=20) AND (Vhum>=100
OR Vhum<=20)) OR
((Vr<100 AND Vr>20) AND (Vinfr>=100 OR
Vinfr<=20) AND
(Vtemp>=100 OR Vtemp<=20) AND (Vhum>=100
OR Vhum<=20)) OR
((Vr>=100 OR Vr<=20) AND (Vinfr>=100 OR
Vinfr<=20) AND
(Vtemp<100 AND Vtemp>20) AND (Vhum>=100 OR
Vhum<=20))
 THEN
yellow_led_alarm_buff <= '1';
green_led_no_alarm_buff <= '0';
blue_led_alarm_buff <= '0';
white_led_alarm_buff <= '0';
red_led_alarm_buff <= '0';
elsif ((Vr>=100 OR Vr<=20) AND (Vinfr>=100 OR
Vinfr<=20) AND
(Vtemp>=100 OR Vtemp<=20) AND (Vhum>=100
OR Vhum<=20)) THEN
red_led_alarm_buff <= '1';
green_led_no_alarm_buff <= '0';
blue_led_alarm_buff <= '0';
white_led_alarm_buff <= '0';
yellow_led_alarm_buff <= '0';
end if;
end process;
red_led_alarm <= red_led_alarm_buff;
yellow_led_alarm <= yellow_led_alarm_buff;
white_led_alarm <= white_led_alarm_buff;
blue_led_alarm <= blue_led_alarm_buff;
green_led_no_alarm <= green_led_no_alarm_buff;
MICROSTEP_PROC : process(MAX10_CLK1_50,
rst, Vr)
      variable step : std_logic_vector(0 to 2) := "111";
   begin
      if rst = '1' then
         coils <= "0000";
```

```
         -- we start with a step
           count <= wait_count;

      elsif rising_edge(MAX10_CLK1_50) then
         if (count < wait_count) then
            -- wait for the next micro step

            count <= count + 1;
         else
            -- perfom a single micro step
            count <= 0;
            if (Vr<=20) then
               step := step + 1;
            elsif (Vr>=100) then
               step := step - 1;
            end if;
            case step is
               when "000" => coils <= "0001";
               when "001" => coils <= "0011";
               when "010" => coils <= "0010";
               when "011" => coils <= "0110";
               when "100" => coils <= "0100";
               when "101" => coils <= "1100";
               when "110" => coils <= "1000";
               when "111" => coils <= "1001";
               when others => coils <= "0000";
            end case;
         end if;
      end if;
   end process;
process( digit2a,digit1a,digit0a, digit2c, digit1c,
digit0c, SW0,
           digit2b,digit1b,digit0b, digit2d, digit1d,
digit0d)
begin
 IF SW0='0' THEN
 digit5 <= digit2a;
 digit4 <= digit1a;
 digit3 <= digit0a;
 digit2 <= digit2b;
 digit1 <= digit1b;
 digit0 <= digit0b;
 elsif SW0='1' THEN
 digit5 <= digit2c;
 digit4 <= digit1c;
 digit3 <= digit0c;
 digit2 <= digit2d;
 digit1 <= digit1d;
 digit0 <= digit0d;
 end if;
end process;
 WITH digit5 SELECT
HEX5 <= "01000000" WHEN "0000", -- display 0
"01111001" WHEN "0001", -- display 1
"00100100" WHEN "0010", -- display 2
"00110000" WHEN "0011", -- display 3
"00011001" WHEN "0100", -- display 4
"00010010" WHEN "0101", -- display 5
```

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering
Impact Factor 8.021 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 12, Issue 12, December 2024
DOI: 10.17148/IJIREEICE.2024.121205

```
"00000011" WHEN "0110", -- display 6
"01111000" WHEN "0111", -- display 7
"00000000" WHEN "1000", -- display 8
"00011000" WHEN "1001", -- display 9
"01111111" WHEN OTHERS; -- blank display
WITH digit4 SELECT
HEX4 <= "11000000" WHEN "0000", -- display 0
"11111001" WHEN "0001", -- display 1
"10100100" WHEN "0010", -- display 2
"10110000" WHEN "0011", -- display 3
"10011001" WHEN "0100", -- display 4
"10010010" WHEN "0101", -- display 5
"10000011" WHEN "0110", -- display 6
"11111000" WHEN "0111", -- display 7
"10000000" WHEN "1000", -- display 8
"10011000" WHEN "1001", -- display 9
"11111111" WHEN OTHERS; -- blank display
WITH digit3 SELECT
HEX3 <= "11000000" WHEN "0000", -- display 0
"11111001" WHEN "0001", -- display 1
"10100100" WHEN "0010", -- display 2
"10110000" WHEN "0011", -- display 3
"10011001" WHEN "0100", -- display 4
"10010010" WHEN "0101", -- display 5
"10000011" WHEN "0110", -- display 6
"11111000" WHEN "0111", -- display 7
"10000000" WHEN "1000", -- display 8
"10011000" WHEN "1001", -- display 9
"11111111" WHEN OTHERS; -- blank display
 WITH digit2 SELECT
HEX2 <= "01000000" WHEN "0000", -- display 0
"01111001" WHEN "0001", -- display 1
"00100100" WHEN "0010", -- display 2
"00110000" WHEN "0011", -- display 3
"00011001" WHEN "0100", -- display 4
"00010010" WHEN "0101", -- display 5
"00000011" WHEN "0110", -- display 6
"01111000" WHEN "0111", -- display 7
"00000000" WHEN "1000", -- display 8
"00011000" WHEN "1001", -- display 9
"01111111" WHEN OTHERS; -- blank display
WITH digit1 SELECT
HEX1 <= "11000000" WHEN "0000", -- display 0
"11111001" WHEN "0001", -- display 1
"10100100" WHEN "0010", -- display 2
"10110000" WHEN "0011", -- display 3
"10011001" WHEN "0100", -- display 4
"10010010" WHEN "0101", -- display 5
"10000011" WHEN "0110", -- display 6
"11111000" WHEN "0111", -- display 7
"10000000" WHEN "1000", -- display 8
"10011000" WHEN "1001", -- display 9
"11111111" WHEN OTHERS; -- blank display
WITH digit0 SELECT
HEX0 <= "11000000" WHEN "0000", -- display 0
"11111001" WHEN "0001", -- display 1
"10100100" WHEN "0010", -- display 2
"10110000" WHEN "0011", -- display 3
"10011001" WHEN "0100", -- display 4
"10010010" WHEN "0101", -- display 5
"10000011" WHEN "0110", -- display 6
"11111000" WHEN "0111", -- display 7
"10000000" WHEN "1000", -- display 8
"10011000" WHEN "1001", -- display 9
"11111111" WHEN OTHERS; -- blank display
end architecture;
```