Real-Time Gas Control System Implementation And Potential Integration In IoT

Kenan Alić   
Faculty of Information Technologies, University of “Džemal Bijedić” of MostarMostar, Bosnia and Herzegovina  
kenan.alic@edu.fit.ba

*Abstract*— This document contains a detailed description of the implementation of the real-time system for gas control in enclosed environment. Topics that are going to be discussed are software as well as hardware implementation of this real-time system. This document will also discuss the potential integration of this system in IoT projects as well as its potential usage in the real world. Hardware technologies that are used for implementation are ESP32 microcontroller, MOSFET transistor, DC fan from an old laptop and MQ-135 gas sensor. Hardware components are connected with combination of jumper cables and breadboard. Software technologies that are used in the project are Arduino IDE, which has support for ESP32 microcontroller, and C++ programming language, which allows us to write memory efficient and fast code. This document will describe how hardware and software can control air quality in real-time and its potential integration and use in Internet of Things technologies.

Keywords— Real-Time System, Interrupt Service Routine (ISR), Internet of Things (IoT), ESP32 microcontroller, MQ-135, C++, Arduino IDE

# Introduction

Proper air circulation in enclosed environment is important for our health, yet there are times when we can’t notice potentially dangerous gases like carbon monoxide, or effectively identify gas stove leaks. Conventional methods for controlling air quality, like ventilation, are not enough when it comes to managing gases in real-time that can’t be detected by humans. That’s where this real-time system for gas control steps in and prevents hazardous gases from accumulating and causing damage.

A real-time system is characterized by its ability to produce the expected result within a defined deadline (timeliness) and to coordinate independent clocks and operate together in unison (time synchronization) [1]. The brain of this system is ESP32 microcontroller that controls other components connected with jumper cables and breadboard, which are responsible for monitoring and regulating air quality. The programming language used in this project is C++ and it is uploaded to a microcontroller using an Arduino IDE which supports ESP32 boards via USB cable.

# Hardware implementation

One of the most important steps in this project was choosing a reliable platform such as ESP32. ESP32 microcontroller is chosen for its reliability, low power consumption, high level of integration, Wi-Fi and Bluetooth support [2] which can enable us to integrate this system in IoT. ESP32 represents the brain of this real-time system, and its main role is to control other components and send or receive data from them. The system is powered with 5 volts via USB cable that is connected to a laptop ensuring steady power supply as well as programming interface to ESP32 microcontroller. Other components in the system are connected to the GPIO pins of the ESP32 microcontroller using a combination of jumper cables and a breadboard. One of the challenges that was faced in this project was how to power and control a 5 volt DC fan when output voltages of GPIO pins are 3.3 volts and only pin that outputs 5 volts was VIN pin that can’t be controlled with ESP32 microcontroller. Solution for that problem was introduction of MOSFET transistor that acts as a switch, and it enables us to control output of VIN pin. MOSFET transistor has three pins:

* Gate controls switching operation by applying voltage and it allows or prevents current from flowing between drain and source [3].
* Drain allows current to flow from drain to source when MOSFET is turned on [3].
* Source is pin where current exits the MOSFET [3].

In the case of this system, the gate of the transistor is connected to GPIO pin 14 that will allow us to control 5 volts. Source of the transistor is connected to the ground pin on ESP32 microcontroller while drain is connected to DC fan that is powered by 5 volts as shown in *Figure 1*.

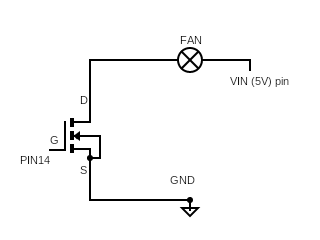


Figure 1

One key component of this real-time system is MQ-135 gas sensor which can detect concentration of various harmful gases.

This sensor has four pins:

* VCC pin is used to power the sensor with 5 volts and in our case, it is connected to VIN pin on ESP32 microcontroller.
* GND pin is connected to the ground pin on ESP32 microcontroller
* DO (digital out) pin is used to send high or low voltage based on threshold level of detected gas.
* AO (analog out) pin outputs an analog voltage signal that corresponds to the concentration of detected gas and it is connected to GPIO 34 pin on ESP32 that is designed to receive analog signal.

After properly connecting hardware components, shown in *Figure 2*, the real-time system is ready to be programmed.

A circuit board with wires

Description automatically generated

Figure 2

# Software implementation

After successfully connecting hardware components it is time to make them work together with code. To be able to upload code to ESP32 it is important to choose a proper development environment. For that purpose, Arduino IDE was chosen. The Arduino IDE supports the ESP32 microcontroller and allows easy uploading of code. The first step after installing Arduino IDE is adding ESP32 board manager in preferences and installing necessary packages which will enable us to select ESP32 board and upload code. The programming language used for programming ESP32 microcontroller is C++. It is developed by Bjarne Stroustrup as an extension to the C language [4].

C++ is object-oriented programming language that allows developers to make high performance and memory efficient applications [4], which makes it perfect choice for this project. After choosing a suitable programming language it was time to start programming an ESP32 microcontroller. The first thing that was done in this phase was definition of GPIO pins by using keyword “#define”. We defined two constants “fanPin” as number 14 which represents physical connection to DC fan, and “MQ135” as number 34 which represents physical connection to MQ-135 gas sensor.

After defining GPIO pins we defined more constants and variables:

* Constant “gasThreshold” represents the threshold value which is set to 1500 and if gas sensor reading exceeds that value the DC fan will start spinning.
* Constant “fanSpinDuration” represents the duration in milliseconds initialized to 10000, in which the fan would run when activated.
* Variable “sensorVal” is used to store the current reading from MQ135 gas sensor.
* Variable “fanStartTime” is used to store time when fan was activated and it is declared as volatile, which indicates that its value will be changed asynchronously, in our case within an interrupt service routine (ISR).
* Flag “fanSpinActive” indicates whether the fan is currently running and it is declared as volatile .
* Hardware timer pointer manages hardware timers on the ESP32 microcontroller and it is initialized to null.

Definition and initialization of variables and constants are shown in *Figure 3*.

A computer screen shot of a program code

Description automatically generated

Figure 3

Next step after declaring and initializing constants and variables is setting up GPIO pins in built in function called “setup()” which is called once when ESP32 is powered up. GPIO pins are set by calling function “pinMode()” which is called inside setup and it accepts two arguments:

* First argument that is passed in this function is GPIO pin number, in this case constants “fanPin” and “MQ135” that were defined previously in code.
* Second argument that is passed is pin state which can be “OUTPUT” or “INPUT”. In this case “fanPin” is set to “OUTPUT” which will allow pin to send signal needed to activate DC fan, and “MQ135” is set to “INPUT” which will enable system to receive analog signal sent from MQ-135 gas sensor.

To ensure the real-time aspect of this system, an interrupt service routine (ISR) was implemented. “An interrupt service routine (ISR) is a software routine that hardware invokes in response to an interrupt. ISR examines an interrupt and determines how to handle it. The ISR handles the interrupt, and then returns a logical interrupt value. If no further handling is required because the device is disabled or data is buffered, the ISR notifies the kernel with a return value. An ISR must perform very fast to avoid slowing down the operation of the device and the operation of all lower priority ISRs” [5]. In this code, function “onTimer()” was declared as an interrupt service routine (ISR) with attribute called “IRAM\_ATTR” which ensures that the function is placed in the Instruction RAM (IRAM) [2] of ESP32 microcontroller for fast execution. This interrupt service routine (ISR) is responsible for handling an operation crucial for turning on the DC fan:

* The analog value that represents the concentration of gases detected by MQ-135 sensor is read via “MQ135” GPIO pin, using function “analogRead()” , which accepts mentioned GPIO pin as an argument, and the value returned by that function is stored in “sensorVal” variable.
* The condition checks if “sensorVal” is greater than “gasThreshold”. If condition is fulfilled “fanPin” GPIO will be set to “HIGH” using built-in function “digitalWrite()”, which will allow DC fan to start spinning, and variable “fanSpinActive” will be set to true and “fanStartTime” will record current time in milliseconds using built-in function “millis()” which returns that value.

After setting up interrupt service routine (ISR) we need to attach it to interrupt within the setup function, which will allow this system to periodically check concentration of gas and respond in real-time. The first step was to set hardware timer on ESP32 by using function “timerBegin()” which accepts three arguments:

* First argument is the timer number and ESP32 microcontroller has four of them (0 to 3) [2].
* Second argument is prescaler value, which divides clock frequency by 80 to set the timer`s thick rate [2] .
* Third argument is boolean value which indicates whether the counter should go up (true) or down (false) [2].

After the hardware timer was set, the interrupt handler needs to be attached to it. That was achieved by using function “timerAttachInterrupt()” and it accepts hardware timer, reference to interrupt service routine (ISR) and a boolean value indicating whether the interrupt service routine should be called on the rising edge of the timer`s output [2]. The function “timerAlarmWrite()” is used to set the alarm value for timer and it has three arguments: hardware timer, alarm value in timer ticks that is set to 1000 and boolean value which specifies if alarm should automatically reload [2]. The final step in setting interrupt is enabling timer alarm, using the function “timerAlarmEnable()” which accepts timer as argument [2]. Implementation of interrupt service routine (ISR) as well as implementation of “setup()” functions is shown in *Figure 4.*

A screen shot of a computer code

Description automatically generated

Figure 4

The “loop()” function continuously checks if the DC fan is turned on, and if it is it will initialize a new variable called “currentMillis” to current time, which is used to calculate elapsed time by subtracting it with “fanStartTime”. After subtraction the difference is compared to “fanSpinDuration” and if it is greater or equal, the state of “fanPin” will be set to “LOW” and variable “fanSpinActive” will be set to fals which indicates that fan is turned off. Implementation of “loop()” function is shown on *Figure 5*.

A computer screen with white text and blue text

Description automatically generated

Figure 5

# Testing the Real-Time Gas Control System

In this phase the real-time system was tested to ensure that it works properly. To be able to test the real-time system the code needs to be compiled and uploaded to ESP32 microcontroller via USB cable, which also provides power. After uploading code, the system should be ready for testing. To simulate gas stove leakage, a lighter was used as a controlled source of gas to test responsiveness and efficiency of the system. When the gas threshold that was set to 1500 parts per million was exceeded the real-time system responded immediately and turned on the DC fan for 10 seconds after which it was turned off which demonstrates responsiveness. If the gas concentration is constantly above threshold the DC fan will not stop spinning until the gas concentration is below 1500 parts per million, which demonstrates efficiency of the real-time system. Demonstration of the system is available at: [Testing Video](https://www.youtube.com/shorts/qn2XoabOsRc).

# Potential integration in IoT

The Internet of Things (IoT) describes the network of physical objects “things” that are embedded with sensors, software, and other technologies for the purpose of connecting and exchanging data with other devices and systems over the interne [6]. IoT three-layer architecture is the common and generally known structure [7] comprised of:

* Perception layer is the IoT architecture’s physical layer [7]. This layer would be, in this case, the real-time system.
* Network layer is layer that distributes and stores data and it transfers data between application layer and perception layer [7].
* Application layer is in-charge of providing the customer with software resources [7].

One of the approaches to integrating and improving the system is to use firebase as network layer and web application as an application layer. In this case the values of detected gas concentration and state of DC fan could be stored and updated in firebase and shown in web applications frontend. Web application could also be used to set variables as well as notify user about potential gas leak.

Second approach could be upgrading the real-time system by introducing ESP32-CAM module and heat sensor as perception layer, python flask web server as network layer and web application as frontend. Role of ESP32-CAM module would be to take a picture of room, if the high temperature is detected, and send it to phyton flask server. Python flask server would have a role in hosting an machine learning model or artificial intelligence that will process the image and detect anomalies such as smoke or high heat. Another role of python flask server would be to notify user about potential danger and send information to frontend where user will be able to see picture and other sensor readings. Frontend could be used to set up thresholds as well as notify authorities in case of emergency.

# Conclusion

In this document it was discussed how hardware components such as ESP32, MOSFET transistor, DC fan, and MQ-135 gas sensor can work together to achieve good air quality. Software implementation using Arduino IDE and C++ enabled the system to perform in real-time with implementation of interrupt service routine (ISR) and interrupt. Testing demonstrated system`s ability to detect gas ensuring responsiveness and efficiency. The potential integration into IoT projects offers possibilities for remote monitoring and control using various technologies. In conclusion, this system provides affordable, scalable and efficient for managing air quality in enclosed environments.

# References

|  |  |
| --- | --- |
| [1] | Intel, "www.intel.com," Intel, [Online]. Available: https://www.intel.com/content/www/us/en/robotics/real-time-systems.html. |
| [2] | espressif, "www.espressif.com," espressif, [Online]. Available: https://docs.espressif.com. |
| [3] | A. S. S. a. K. C. Smith, Microelectronic Circuits 7th edition, New York, NY, USA: Oxford University Press, 2014. |
| [4] | w3school, "w3school.com," [Online]. Available: https://www.w3schools.com/cpp/cpp\_intro.asp. |
| [5] | P. Zhang, Industrial Control Technology, 1st ed, Amsterdam: Netherlands: Elsevier, 2008. |
| [6] | Oracle, "Internet of Things," Oracle, [Online]. Available: https://www.oracle.com/internet-of-things/. |
| [7] | GeeksforGeeks, "3 Layer IoT Architecture," GeeksforGeeks, [Online]. Available: https://www.geeksforgeeks.org/3-layer-iot-architecture/. |
| [8] | K. Alić, "Video Demonstration," [Online]. Available: https://www.youtube.com/shorts/qn2XoabOsRc. [Accessed 10 July 2024]. |