Comparative Analysis and Practical Implementation of the ESP32 Microcontroller Module for the Internet of Things

Conference Paper · September 2017

DOI: 10.1109/ITECHA.2017.8101926

CITATIONS
33

READS
5,593

3 authors:

Alexander Maier
Johnson Matthey
1 PUBLICATION **33** CITATIONS
SEE PROFILE

Andrew Sharp
Glyndwr University
2 PUBLICATIONS **34** CITATIONS
SEE PROFILE

Yuriy Vagapov
Glyndwr University
89 PUBLICATIONS **224** CITATIONS
SEE PROFILE

Comparative Analysis and Practical Implementation of the ESP32 Microcontroller Module for the Internet of Things

Alexander Maier, Andrew Sharp, Yuriy Vagapov
School of Applied Science, Computing and Engineering,
Glyndwr University, Plas Coch, Mold Road,
Wrexham, LL11 2AW, UK

***Abstract*—This paper discusses the Espressif Systems latest product ESP32 designed for Internet of Things and embedded system related projects. The ESP32 is a low-cost, low-power system on a chip series of microcontrollers with Wi-Fi and Bluetooth capabilities and a highly integrated structure powered by a dual-core Tensilica Xtensa LX6 microprocessor. This paper provides a comparative analysis of the ESP32 with some other market competitors and introduces the microcontroller specification, features and programming details. A portable, wireless oscilloscope based on the ESP-WROOM-32 and a mobile application is described in detail as an example of successful practical implementation of the device.**

Keywords—ESP32, Internet of Things, Wi-Fi, oscilloscope

I. INTRODUCTION

The Internet of Things (IoT) market has rapidly expanded over the last few years following the increased demand in communication and control for various devices and gadgets. The main requirement applied for modern IoT devices is to provide an effective connectivity to ensure reliable remote communication and data transfer in a wireless environment. This concept for IoT devices is called 6A (Anything, Anytime, Anyone, Anyplace, Any service, and Any network) [1].

The IoT technology significantly impacts on behaviour and lifestyle of people in both working and domestic environments. The advanced communication capabilities dramatically reshape properties and operation of industrial automation and manufacturing, business and process management, intelligent transportation and logistics, etc. In terms of domestic applications, IoT enhances home automation and introduces new, communication based technologies such as domotics, assisted living, e-health and e-learning, etc. [2].

In order to develop the IoT further and expand the area of its applications, a powerful, low-cost and low-power solutions for the IoT devices are required. Another requirement for an IoT device is to have a small form-factor; the smaller size and weight of the device the wider the area of its applications. Each IoT based unit comprises of a microcontroller (µC) and a wireless commutation module (usually WiFi), or a combination of both in one. A large variety of modules and µC are already on the market and widely used for the design and development of IoT devices. These are Xbee, WhizFi, certain Arduino boards, etc. However, most of currently offered devices are either quite expensive or large in terms of weight and size. Moreover, very few modules are open source devices and have no restriction in the operation purpose. A new device QFN48 called ESP32 will be released in the market by Espressif Systems in September 2016 to replace the previous µC ESP8266. ESP32 device is a powerful microcontroller with build in Wi-Fi and Bluetooth®, designed to be a perfect solution for IoT devices [3].

This paper presents a detailed comparison of the ESP32 to its competitors including competitive analysis of its technical features and functions. The paper also describes how to program ESP32 and discusses an example application demonstrating a practical implementation of the microcontroller.

II. COMPARATIVE ANALYSIS OF ESP32

Along with the release of ESP32, Espressif Systems also offers a corresponding module ESP-WROOM-32. Despite the small size (25.5 x 18.0 x 2.8mm) it is very easy to use the

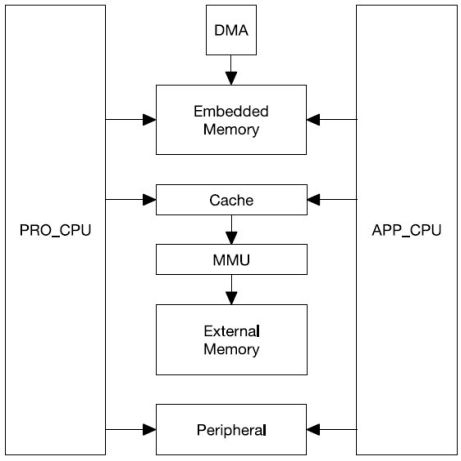

Fig. 1. ESP32 system structure.

978-1-5090-4815-1/17/$31.00 ©2017 IEEE

TABLE I. MICROCONTROLLES FOR THE IOT DESIGN [4]-[7]

Chip (Module)	ESP32 (ESP-WROOM-32)	ESP8266 (ESP8266-12E)	CC32 (CC3220MODSF)	Xbee (XB2B-WFPS-001)
Details:				
CPU	Tensilica Xtensa LX6 32 bit Dual-Core at 160/240 MHz	Tensilica LX106 32 bit at 80 MHz (up to 160 MHz)	ARM Cortex-M4 at 80 MHz	N/A
SRAM	520 KB	36 KB available	256 KB	N/A
FLASH	2MB (max. 64MB)	4MB (max. 16MB)	1MB (max. 32MB)	N/A
Voltage	2.2V to 3.6V	3.0V to 3.6V	2.3V to 3.6V	3.14V to 3.46V
Operating Current	80 mA average	80 mA average	N/A	N/A
Programmable	Free (C, C++, Lua, etc.)	Free (C, C++, Lua, etc.)	C (SimpleLink SDK)	AT and API commands
Open source	Yes	Yes	No	No
Connectivity:				
Wi-Fi	802.11 b/g/n	802.11 b/g/n	802.11 b/g/n	802.11 b/g/n
Bluetooth®	4.2 BR/EDR + BLE	-	-	-
UART	3	2	2	1
I/O:				
GPIO	32	17	21	10
SPI	4	2	1	1
I2C	2	1	1	-
PWM	8	-	6	-
ADC	18 (12-bit)	1 (10-bit)	4 (12-bit)	4 (12-bit)
DAC	2 (8-bit)	-	-	-
Size	25.5 x 18.0 x 2.8 mm	24.0 x 16.0 x 3.0 mm	20.5 x 17.5 x 2.5 mm	24.0 x 22.0 x 3.0 mm
Prize	£8	£5	£16	£23

module due to integrated components such as antenna, oscillator and flash. Similar modules for other microcontrollers are often used for tests and prototypes or by hobbyists. Table I compares some of those in detail [4]-[7].

The table shows the details of 4 modules and µC used for the design of IoT devices. Actually, the variety of modules and microcontrollers for IoT are much bigger but most of them have the same problems related to size, performance and price. For example, the boards like RTLDuino are open source and can handle complex tasks on their own unlike the Xbee, but they are quite large in terms of size. On the other hand, ESP32 QFN48, compared to other microcontrollers, is a very small component having a size of just 5mm x 5mm. Due to the published circuit of the module ESP-WROOM-32 it is easy to integrate ESP32 onto a custom PCB and design a space saving device. The board ESP32-DevKitC is a bread-board friendly, ready to use solution for testing and educational purposes. ESP8266, ESP32 predecessor, was extremely popular for the design in many IoT related projects, however, ESP32 is a better solution which can be implemented in more complex projects.

III. ESP32 TECHNICAL DETAILS AND FUNCTIONS

A. System and Memory

ESP32 is a dual-core system with two Harvard Architecture Xtensa LX6 CPUs. All embedded memory, external memory and peripherals are located on the data bus and/or the instruction bus of these CPUs. The microcontroller has two cores – PRO_CPU for protocol and APP_CPU for application, however, the purposes of those are not fixed. The address space for both data and instruction bus is 4GB and the peripheral address space is 512KB. Moreover, the embedded memories are 448KB ROM, 520KB SRAM and two 8KB RTC memory. The external memory supports up to four times 16MB Flash [4].

B. Clock and Timer

ESP32 can use either he internal Phase Lock Loop (PLL) of 320MHz or an external crystal. It is also possible to use an oscillating circuit as a clock source at 2-40MHz to generate the master clock CPU_CLK for both CPU cores. This clock can be as high as 160MHz for high performance or lower to

reduce the power consumption. All other clocks, like the APB_CLK for peripherals are driven by the master clock. In addition, there are several low power clocks like the internal RTC_CLK with a default frequency of 150kHz and the option to adjust it for deep sleep modes. There are four 64-bit timers for generic purposes with 16-bit prescalers with a range from 2 to 65536. Each timer uses the APB clock, usually at 80MHz. Those timers can count either up or down, be frozen and trigger events. Besides 4 generic timers there are also timers to drive the PWM controller. There are 8 high speed and 8 low speed PWM channels, each driven by four timers [4].

C. Block Diagram and Functions

ESP32 microcontroller structure is designed to operate under the following protocols – TCP/IP, full 802.11 b/g/n/e/i WLAN MAC, and Wi-Fi Direct specification. The microcontroller can provide Basic Service Set (BSS) STA and SoftAP operations under the Distributed Control Function (DCF) protocol. It is also support P2P group operation compliant with the latest Wi-Fi P2P protocol. Thus, it can operate as a station and be connected to the internet or server and access point in order to provide a user interface to, for example, smartphone running a mobile application [8].

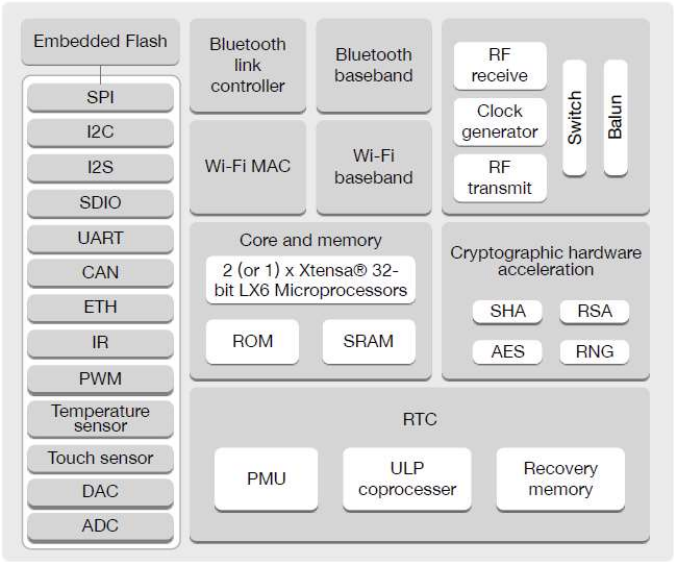

Fig. 2. Function block diagram [8].

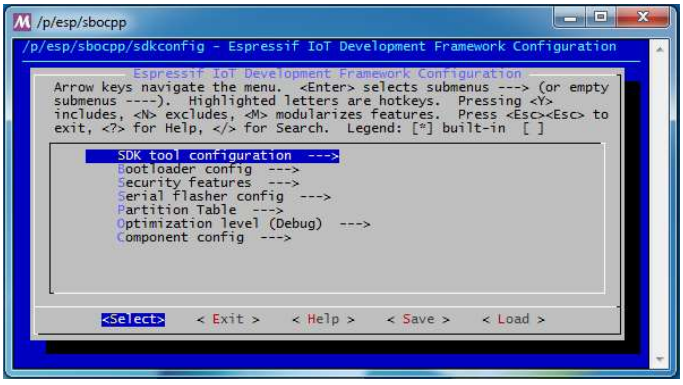

Fig. 3. ESP-IDF configuration menu.

The microcontroller supports v4.2 BR/EDR and BLE Bluetooth which fits the current standard and is capable to operate at a speed up to 4 Mbps. ESP32 can operate under various power modes – active mode (the chip radio is working) and modem-sleep mode (CPU is fully operational but Wi-Fi and Bluetooth is powered off). Furthermore, there are light and deep-sleep modes, where either both or only one CPU are operating at a lower performance. The GPIOs include two 12-bit ADCs with 18 channels in sum. Those can be configured for 9-bit, 10-bit and 12-bit resolutions with an attenuation of −0dB, −6dB or −11dB for different input ranges. One ADC channel is connected to the integrated hall sensor in order to detect magnetic fields, whereas another to the temperature sensor with the range from −40°C to 125°C to monitor the chip temperature. Besides the ADCs there are also two 8-bit DACs to convert the digital signals into analogue voltage signal outputs. Ten of the GPIOs are capable to sense capacitive variations and can be used for touch sensors. Since those are high sensitive relatively small pads can be used. Moreover, ESP32 provides a number of interfaces: an Ethernet MAC Interface, one SD/SDIO/MMC Host Controller, three UART interfaces up to 5Mbps, two I2C bus interfaces with standard and fast mode, two I2C interfaces with a frequency of 10kHz up to 10MHz, an 8-channel infrared remote controller and an 8-channel pulse counter. The PWM controller can be used to drive digital motors or generate digital waveforms. Three SPIs can be used in slave or master mode with a clock up to 80MHz. [8].

D. Programming the ESP32

The real-time operating system on ESP32 is FreeRTOS. It is open source, designed for embedded systems and provides basic functions to the higher-level applications. The core functions are memory management, task management and API synchronization [9].

The usual way to program the ESP32 is using the ESP-IDF, Espressif Systems Internet of Things development framework, which is available on their GitHub repository. The ESP-IDF was developed for Linux, thus a Linux terminal is required in order to execute the bash files. However, it possible to develop in Windows by using MSYS2. This software provides a Linux terminal in Windows. Furthermore, the ESP-IDF-Template is required order to start an ESP32 project. It includes all necessary files for a successful compilation, which are part of an individual project and not included in the ESP-IDF.

The ESP-IDF provides a visual configuration menu accessible by the command “make menuconfig” which is the only graphical menu (Fig. 3). All other operations such as compiling or flashing take place by executing simple commands. Therefore, the open source IDE Eclipse provides great support for Makefile project. A project should be configured in order to use the xtensa-esp32-elf-gcc compiler and refer the ESP-IDF for enabling autocomplete and debug features, which are essential for proper program development.

It is also possible to flash ESP32 out of Eclipse without terminal opening anymore.

The common language for programming ESP32 is C, thus most API libraries are also provided in C. However, the microcontroller can be also easily programmed in C++. Some Arduino libraries can be used under C++ programming option, although some changes might be required. Neil Kolban, an engineer from Texas provides plenty of C++ libraries in his GitHub repository for the ESP32 APIs. Since this chip is open source everyone can develop an "operating system" for the ESP32, thus there are also solutions on the Internet to program it in LUA, JavaScript, etc.

IV. Example Application of ESP32

The variety of application of ESP32 is not limited to common IoT projects, such as controlling sockets and lights remotely in order to build a smart-home. The following example is a project called "smartphone based oscilloscope" aimed to build a prototype of a portable, wireless oscilloscope based on the ESP-WROOM-32 as the hardware core and a smartphone application as the display and control unit.

At the first stage of the project, the build-in ADC was analysed in order to verify the accuracy of the conversion. A voltage from 0.0V to 3.5V was applied to the ADC input in steps of 0.1V and the measured values were recorded. This test was repeated for different settings. The best result was delivered at 12 bit resolution with an attenuation of −6dB (Fig. 4). These tests had also shown that the ADC input range does not begin at 0.0V but at 0.17V instead. Thus, the decision was made to use the range from 0.2V to 1.9V for this project. Moreover, the average frequency was approximate 44.5 kHz at 10-bit and 12-bit resolution. It was slightly higher at 9 bit resolution approximate 45.9 kHz but that frequency gain was not big enough to take the loss of resolution.

This provides a maximal frequency limit for the input signal of this project of approximate 20 kHz since the sampling frequency must be at least two times greater than the signal frequency in order to sample the signal reliably. The frequency of 20kHz is actually not suitable for an oscilloscope but it is good enough for a first prototype and an excellent opportunity to test ESP32 capabilities.

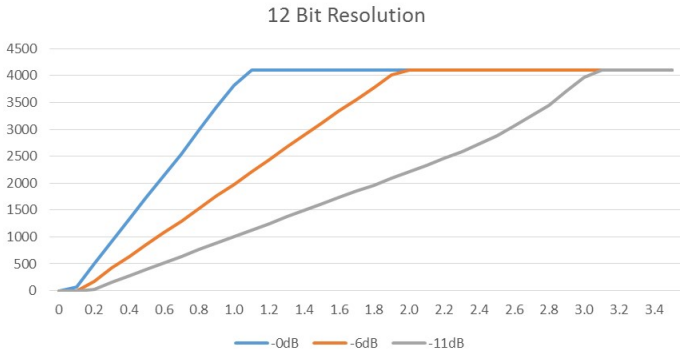

Fig. 4. ADC linearity test at 12 bit resolution.

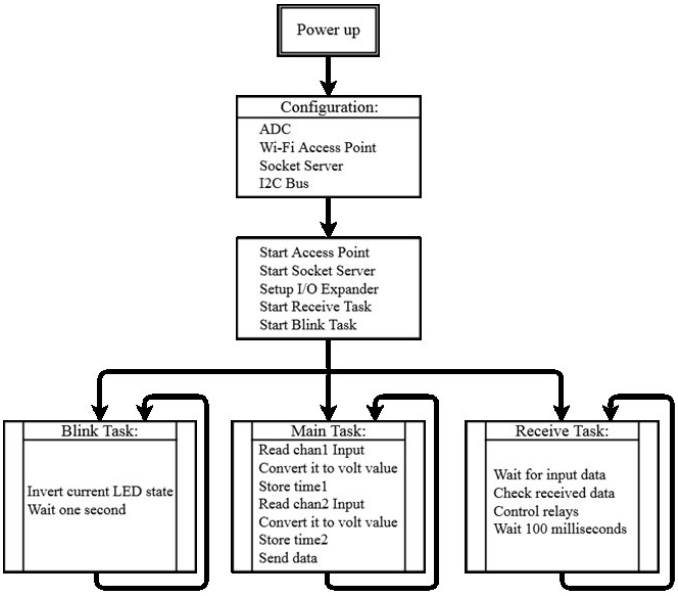

Fig. 5. Microcontroller flowchart.

The chosen language was C++ because Adafruit provides a C++ library for the MCP23017 I/O expander, which was used to control the relays, and no such library exists in C. C++ implements also classes for good structured program. The simplified program flowchart is shown in the Fig. 5.

The simplified flowchart illustrates the general program structure. At the initial stage, the microcontroller must be configured to provide the settings for the ADC, I2C bus and APIs required for the communication between Wi-Fi module and smartphone application. The settings for the ADC comprised the resolution, attenuation and the channel. In order to conduct this procedure, two instances of the created class ADC_am were instantiated, one for each input channel and the parameter were passed to the constructer. The I2C master should be defined and initiated to use the I/O Expander. This includes setting of the GPIOs to use as SCL and SDA as well as the clock frequency moreover defining a GPIO, in this project GPIO_NUM_17 as MCP23017 reset pin. Upon completion of the procedure above, some classes have been instantiated for Wi-Fi connection and socket server. Since the data is continually streamed to the application a socket connection will performs better than a HTTP request method. The socket connection also needs to be established once and causes less data overhead.

Once the configuration is completed the Wi-Fi access point starts with the pre-set SSID and WPA2 password also an event handler is passed to the Wi-Fi object in order to handle connections. The socket server configured for the port 8001 starts as well at this point.

A float variable ch1 stores the voltage reading of the first input channel. However, in order to store the voltage it must run created scaleAndRound() function of the sbo class and pass the data received from chan1.read() along with the channel number. This is necessary because chan1.read() returns a value from 0 to 4095 which equals to a certain

```
float ch1 = sbo.scaleAndRound(chan1.read(),1);
int ms1 = system_get_time();
//ESP_LOGI("ADC", "Chan1 read = %f time: %d", ch1, ms1);
float ch2 = sbo.scaleAndRound(chan2.read(),2);
int ms2 = system_get_time();
//ESP_LOGI("ADC", "Chan2 read = %f time: %d", ch2, ms2);

//vTaskDelay(1000/portTICK_PERIOD_MS);

char outputData[100];
sprintf(outputData, "%d,%f,%d,%f", ms1,ch1,ms2,ch2);
if(socketServer.connectedCount()==1){

            socketServer.sendData(outputData);

} else {
    vTaskDelay(10/portTICK_PERIOD_MS);
}
```

Fig. 6. ESP32 main task listing.

```
void receiveTask(void *param){
    uint8_t data;
    while(1){
        if(socketServer.connectedCount()>0){
            gpio_set_level(GPIO_NUM_4,1);
            int erg = socketServer.receive_cpp(&data,8);
            if(erg){
                sbo.executeCmd(data);
                ESP_LOGI("Received", "Data = %d", data);
            }
        } else{
            gpio_set_level(GPIO_NUM_4,0);
        }
    vTaskDelay(100/portTICK_PERIOD_MS);
    }
}
```

Fig. 7. ESP32 receive task listing.

voltage depending on the scaling factor used at that channel. This is shown in Fig. 6.

Depending on the passed channel number the scale is set to the scaling factor of the channel one or channel two. Two variables defaultVal and valPerVolt are implemented to convert the measured value to an actual voltage. Since the channel input ground potential at the middle of the ADC input range, defaultVal stores the value equals to ground potential. The dependency between input value and voltage is stored in valPerVolt. Thus, a variable res represents the voltage at the ADC channel input. To get the voltage at the actual PCB channel input it is necessary to multiply res with the current scaling factor of this channel. To limit the decimals to two the function round() of the math.h library was used. Since it rounds any given value to an integer it was necessary to multiply it with 100.0 before and to divide it by 100.0 after that step in order to keep two decimals. Finally, the formatted value get returned the function output.

Along with the voltage value the main task stores the time after reading the input. More precisely this is the time in microseconds which the microcontroller runs since the power up. This is necessary to let the application know how much time passed since the last input to plot the traces correctly. 4 values are combined to a string or char array, which are separated by commas and sent to the application if a connection is established. If there is no connection available a delay of 10 milliseconds will be built in to prevent triggering the watchdog reset.

The blink task is very simple GPIO_NUM_5 is set the value of a Boolean variable, this variable is then inverted and the task waits one second until it repeats. This gives a visual indication that ESP32 is still running and no error has occurred.

The receive task is shown in Fig. 7. If a client is connected to the socket server then this task lights up the connection LED to show that a connection is established. Upon the completion of the connection establishing, the task waits for incoming data. As soon as data are received it is passed to the exucuteCmd() function of the sbo object and printed on the serial monitor for debugging purpose. The execute functions checks first if the incoming char is a digit or a character. Digits are the commands for the channel one, whereas the characters are for the channel two. Depending on the values, it will be passed on to the corresponding switch statement. The cases inside that switch statement execute the command by setting the relays along with the associated scaling factor.

If there is no communication between the Wi-Fi module and the application then the previously defined GPIO_NUM_4 which is powering the connection LED will be set low. A delay of 100 milliseconds prevents the watchdog reset from triggering.

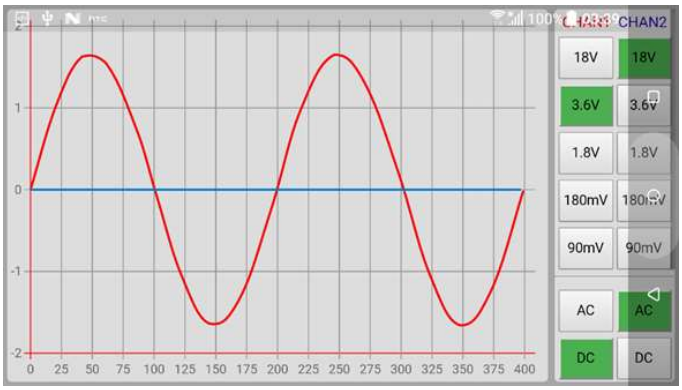

Fig. 8. Application output.

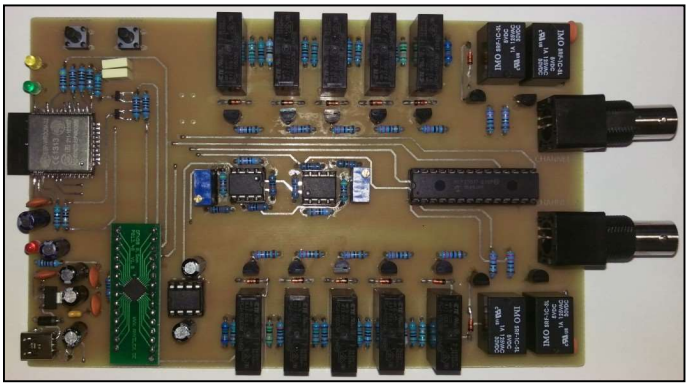

Fig. 9. PCB layout.

The application output is illustrated in Fig. 8. This app has been developed in Visual Studio using the Cordova framework in order to develop it for multiple OS. Cordova is a common open source framework for hybrid cross-platform applications.

The PCB designed and manufactured for this project contains ESP-WROOM-32, a 3.3V and −3.3V power supply and an USB-UART-Bridge for programming. Moreover, two input channels include relays for the AC/DC/GND setting and to set different scaling factors controlled by a MCP23017 I/O-Expander along with a LM538 for each channel for scaling and offset purposes.

V. Conclusion

This paper discusses a new ESP32 system on a chip series with Wi-Fi and Bluetooth. A detailed comparison of several IoT related modules has been provided to highlight the ESP32 microcontroller technical parameters and functions. An example of the microcontroller application has been presented and discussed in order to demonstrate practical implementation of this new component.

It has been shown that ESP32 is the excellent option for IoT devices due to the performance properties and price. The microcontroller is available in various form-factors. The bread board friendly version ESP32-DevKitC is a perfect solution for hobbyist and educational purposes, the ESP-WROOM-32 module provides a small solder friendly footage whereas the ESP32 QFN48 is the option for industrial manufactures and small sized solutions.

ESP32 performs much better than its predecessor ESP8266 widely used in a large variety of IoT applications. The excellent performance of the microcontroller is achieved due to dual core structure and a significant extension of the operational features. The microcontroller operating system FreeRTOS is open source software providing a great support for real time applications. Thus, it is expected that ESP32 will play a major role in design of future IoT systems and embedded projects.

References

[1] S. Li, L.D. Xu, and S. Zhao, “The Internet of things: A survey,” *Information Systems Frontiers*, vol. 17, no. 2, pp. 243–259, April 2015.

[2] L. Atzori, A. Iera, and G. Morabito, “The Internet of things: A survey,” *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010.

[3] K.J. Singh, and D.S. Kapoor, “Create your own Internet of Things: A survey of IoT platforms,” *IEEE Consumer Electronics Magazine*, vol. 6, no. 2, pp. 56-68, April 2017.

[4] Espressif Systems. (2017, May 4). espressif.com [Online]. Available: https://espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf. [Accessed 10 May 2017].

[5] AI-Thinker Team. (2015). mintbox.in [Online]. Available: https://mintbox.in/media/esp-12e.pdf

[6] Texas Instruments. (2017, March). ti.com [Online]. Available: http://www.ti.com/lit/ds/symlink/cc3220mod.pdf

[7] Digi International Inc. (2015). digi.com [Online]. Available: https://www.digi.com/pdf/ds_xbeewifis6b.pdf

[8] Espressif Systems. (2017, April 11) espressif.com [Online]. Available: https://espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

[9] N. Kolban, *Kolban’s Book on ESP32*, USA: Leanpub, 2017.