Chapter

Real-Time FPGA-Based Systems to Remote Monitoring

J. Guadalupe Velásquez-Aguilar, Outmane Oubram and Luis Cisneros-Villalobos

Abstract

Some industrial and laboratory applications such as control, monitoring, test and measurements, and automation require real-time systems for their development. Embedded systems for acquisition and processing often require the participation of the embedded operating system and therefore are necessary techniques that can accelerate software execution. The latest field-programmable gate arrays' (FPGA) technology has blurred the distinction between hardware and software with embedded processors that allow the development of Systems-on-a-Chip (SoC) running on operating systems. The widespread adoption of wireless technologies such as Bluetooth, ZigBee, and Wi-Fi in the last years has facilitated the use of these technologies to the development of real-time monitoring applications that combined with FPGA devices which has the advantages of low cost, flexibility, and scalability as compared with other commercial systems.

Keywords: real-time monitoring, wireless FPGA-based controllers

1. Introduction

In many of the industrial and laboratory systems, especially in control and monitoring tasks, hardware is used in a loop (**Figure 1**).

In the diagram shown above, information about the physical environment is obtained through sensors that respond to a physical stimulus (light, heat, pressure, magnetism, acceleration, stress) and that are designed so that the information acquired is transformed into an electrical signal proportional to the changes. Frequently, the electrical signal obtained from the sensor has noise or interference, so signal conditioning is necessary, which is achieved through some processing operations such as amplification, linearization, compensation, and filtering. Analog-to-digital converters (ADC) are used to sample and hold charge, thereby converting the analog circuit current/voltage into a digital value. Without encoding, sensors are useful in analog control systems, but for the use in digital control and monitoring systems, encoding is critical. Real-time embedded systems therefore require digital encoding of all sensor inputs, with the exception of subsystems, which are all analog [1].

One of the main tasks of embedded systems is the processing and interpretation of information that arrives from the outside. An embedded system is a combination of hardware and software that is specifically designed for a particular function. In most cases, an embedded system is used to replace an application specific

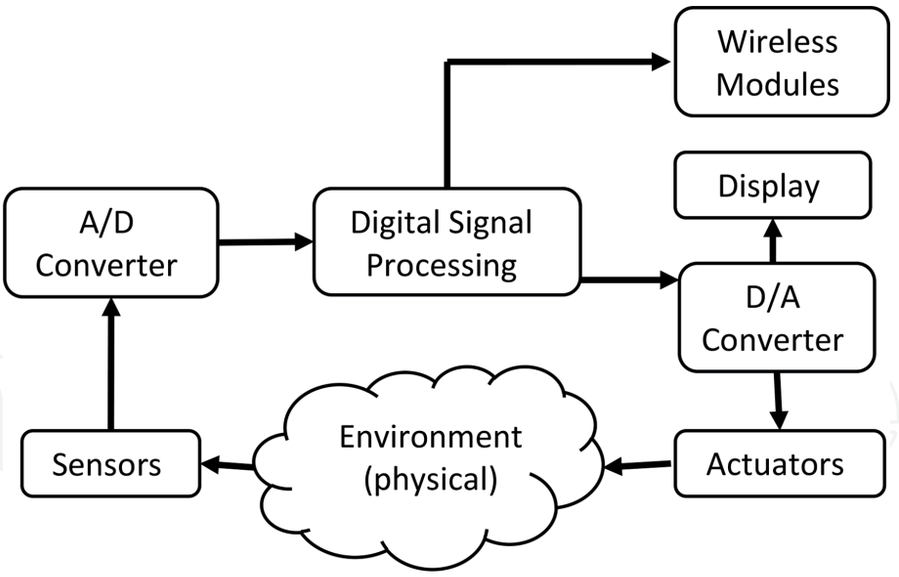


Figure 1.
Real-time processing system hardware in the loop.

electronics in consumer products. By doing so, most of the systems functionality is encapsulated in the firmware that runs the system, and it is possible to change and upgrade the system by changing the firmware, while keeping the hardware same [2]. When embedded systems are board-based, it is fairly straightforward to select the proper components, integrate them with software, and ship the product.

In the mid-1990s, the development of embedded systems evolved with the concept of ASIC technology, changing the philosophy of systems based on a chip-set to a concept System-on-a-Chip (SoC) based on embedded cores. The term SoC defines an integrated circuit (IC) designed by joining multiple independent VLSI models to provide full functionality for an application. Each model is predesigned with complex functions known as cores that serve to a variety of applications. Cores can use a library of components designed by intellectual property (IP) companies or by self in house. The chip used for the system may contain combinations of cores that are generally available in the form of a synthesizable high-level description language (HDL), as Verilog/VHDL, or optimized transistor-level design. Some examples of core-based SoC include high-end microprocessors, GPS positioning for autonomous vehicles, smartphone, and even PC-on-a-Chip [3].

Nowadays, embedded systems are made on SoC. The SoC can include several heterogeneous subsystems, including specific hardware components and sophisticated interconnects (**Figure 2**).

Often in systems used in industrial and laboratory applications for control, monitoring, testing and measurement, and automation, data acquisition (DAQ) subsystem is the first stage. The main purpose of DAQ is to measure physical phenomena, converting the analog signal into a digital signal, and then send or save the data collected for further analysis. An important point to consider is the problems of output conversion into a digital format, as well as to high accuracy and speed conversion methods used. In addition, if the application requires simultaneously capturing several signals, the DAQ must be of the multichannel type and will need a central processor, which will control the channeling and organization of data acquisition for further displays or its use in control systems. The methods to be used in multichannel data acquisition depend on the control and measurement tasks and directly influence the structure and functionalities of the DAQ. In a modern system, the measurement and control sensors can be set up in different ways; the most used are:

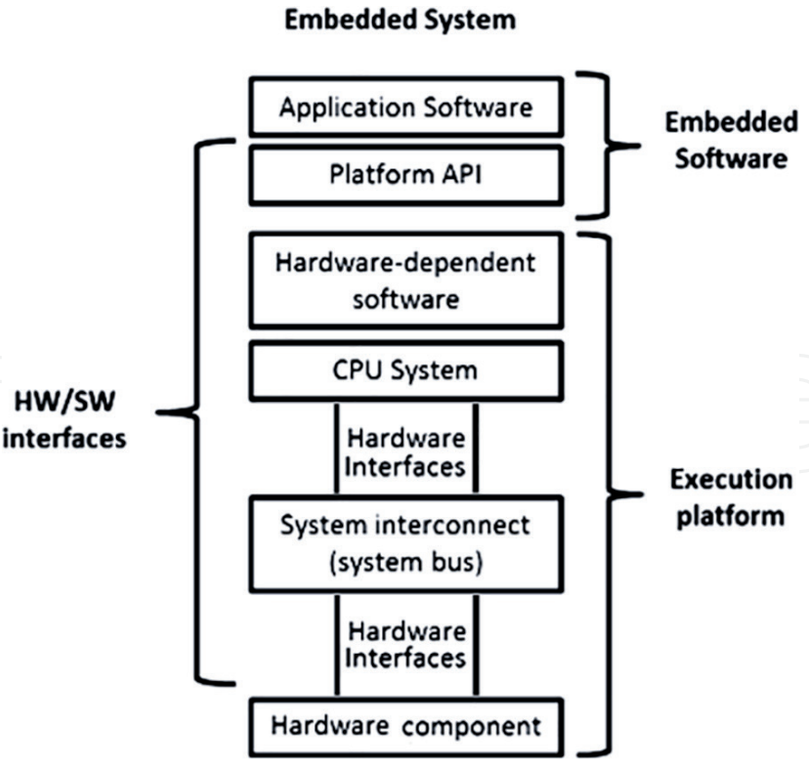


Figure 2.
Embedded system architecture. In addition to hardware, a SoC includes classic application software- and hardware-dependent software that must be co-designed with hardware interfaces. The API hides hardware details such as interrupt controllers or memory and I/O systems [4].

- Methods that use time-division channeling, which perform sensor multiplexing, that is, the time is shared by each sensor in the data acquisition.

- Methods using space-division channeling, based on simultaneous data acquisition from all the sensors.

In both cases, access to information at any time depends on the control and measurement tasks used [5].

Commercial DAQ cards are differentiated by their viabilities such as sampling frequency, scale of acquired signal, power, and requirements but are generally high in cost, and they need a PC at the collection site. Embedded systems to data acquisition often require the participation of the embedded operating system. The modern on-board FPGA can not only overcome the deficiency of the microcontroller unit (MCU) or the digital signal processor (DSP) and meet the requirements of system for real-time and synchronization but also for embedded applications using SoC FPGA platforms with the high level coordination, versatility, and full-stacked operative system [6].

2. Wireless communications standard protocols

At present, the most used standard protocols in communication in wireless sensor networks (WSN) are IEEE 802.15.1 Bluetooth, IEEE 802.15.4, IEEE 802.15.4/a ZigBee, and IEEE 802.11 Wi-Fi. The following describes these protocols:

2.1 Bluetooth technology

IEEE 802.15.1 protocol is an economical and secure wireless communication standard, used to exchange information between devices through a short-range radio

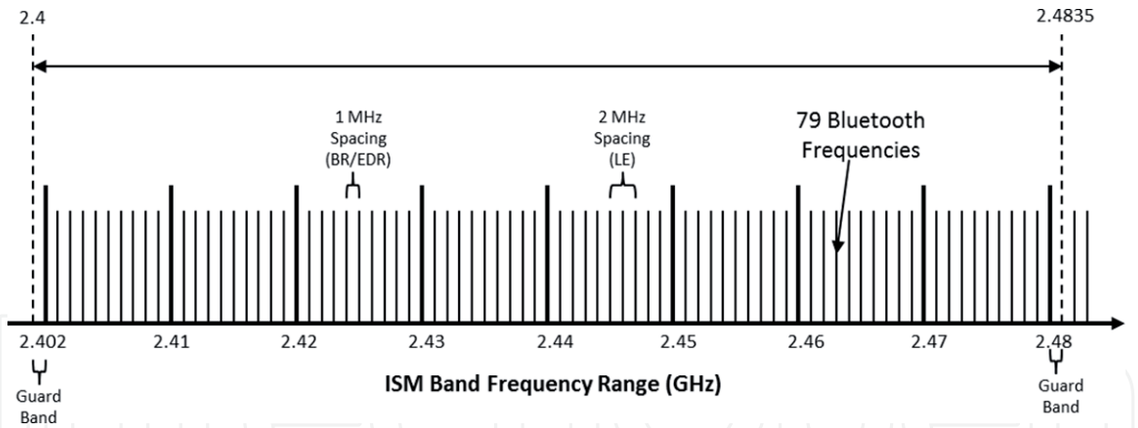


Figure 3.
*Bluetooth frequency bands and RF channels. Each RF channel is ordered in channel number n as follows: f = 2402 + n MHz, where n = 0, …, 78 (BR/EDR) and f = 2402 + n*2 MHz, with n = 0, …, 39 (LE).*

frequency; it was invented in 1994 by a group of engineers of the Ericsson Company. The original idea of Bluetooth was to eliminate the need for a cable connection between devices by connecting them over short distances (up to 100 m). Bluetooth operates with industrial, scientific, and medical frequencies (ISM), from 2.4 to 2.4835 GHz starting at 2.402 GHz. Bluetooth devices can be configured to operate in two ways:

1. Basic and Enhanced Data Rates (BR/EDR) transmissions, where 79 radio frequency (RF) channels with 1 MHz spacing are used. This configuration uses frequency-hopping spread spectrum (FHSS) scheme, at a nominal rate of 1600 hop per second.

2. Low Energy (LE) mode, where only 40 RF channels with 2 MHz spacing are available and adaptive frequency hopping (AFH) is used (**Figure 3**) [7, 8].

Since its appearance, Bluetooth protocol has continuously evolved, so there are several versions that are differentiated with a number. Bluetooth versions 1.0–3.0 are known as Bluetooth Classic category and originally supported a maximum data rate of 721 kbps. This is referred to as Basic Rate (BR). The Bluetooth 2.0 EDR specification added support for data rates up to 2.1 Mbps. This is referred to as Enhanced Data Rate (EDR). The Bluetooth 3.0 High Speed (HS) specification enhanced it even further to 24 Mbps. Bluetooth Low Energy (BLE) is a new category that include versions 4.0 and 5.0. Geared toward applications requiring low power consumption, BLE returns to a lower data throughput of 1 Mbps using the GFSK modulation scheme. The Bluetooth 4.0 specification did not add any additional data rates; it only reduced the current consumption to enable low-energy devices. In Bluetooth 5.0, in addition to low power consumption, four different data rates are offered to accommodate a variety of transmission ranges: 2 Mbps, 1 Mbps, 500 kbps, and 125 kbps. The lower data rate of 125 kbps was added to compensate for the increase in transmission range [9].

Bluetooth module generally consists of four components: radio transceiver, baseband/link controller, link manager, and a host controller interface (HCI) [8]. HCI is the interface to access the Bluetooth module setup from the host. Bluetooth communication is based on the following two network topologies:

1. Piconet: It consists of one master and up to seven slaves (**Figure 4a**).

2. Scatternet (combination or two or more piconets) (**Figure 4b**): It is formed when two or more piconets come together by sharing a device. Scatternets help

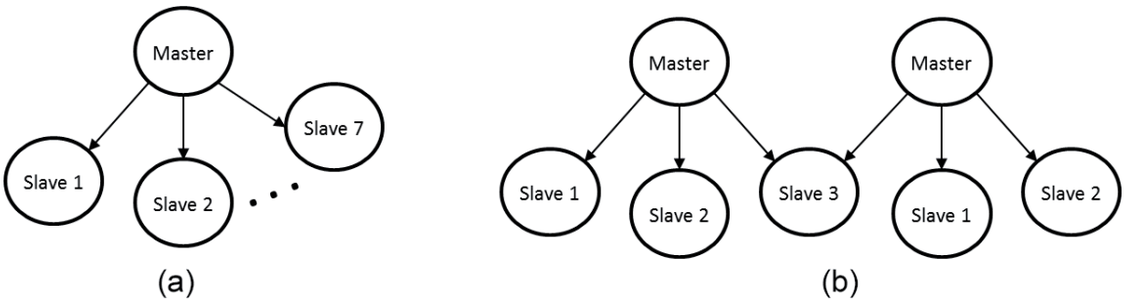


Figure 4.
Bluetooth network topologies. (a) Piconet. (b) Scatternet.

to extend the number of Bluetooth devices that can communicate with each other. They allow more than seven devices to communicate with each other [10].

2.2 ZigBee technology

ZigBee, also known as IEEE 802.15.4, was initially conceived in 1998, standardized in 2003, and finally revised in 2006; it is a low power standard for short-range communications between wireless devices. ZigBee is classified as a wireless personal area network (WPAN). ZigBee devices operate in one of three bands: 868 MHz (Europe), 915 MHz (North America), and 2.4 GHz (worldwide). The 2.4 GHz band is the most used by the ZigBee transceivers and uses offset quadrature phase-shift keying (OQPSK) modulation stream. This type of modulation, which is a derivation of traditional QPSK, is used for requiring less transmission power and achieving the same or better performance than similar ones. OQPSK modulation combined with the use of a 5 MHz channel bandwidth allows devices to reach a data rate of up to 250 kbits/s efficiently [11]. The IEEE 802.15.4 has three different operation modes (**Figure 5**):

1. Personal area network coordinator (ZigBee coordinator, ZC): It is the principal controller of the PAN. This device identifies the network, and in it the configurations that allow other devices to be associated are made. ZC function is to act as ZigBee Router (ZR) once the network is formed. ZC is a full-functional device (FFD) that implements the full protocol stack; it can operate with or without beacon mode. The beacon mode of operation is used when data packets must be sent within an allowable delay, such as in monitoring and control applications. The beaconless mode is suitable for applications where data is only sent when an event occurs, that is, there is no continuity in sending information such as motion detection. In a cluster-tree network, all ZRs will receive beacons from their parents and send their own beacons to synchronize the nodes that belong to their clusters.

2. Local Coordinator (ZigBee Router, ZR): This device must be associated with a ZC or with another ZR previously associated with a network, because it does not create its own network. ZR is a full-functional device (FFD) that implements the full protocol stack. This device participates in multi-hop routing of message in mesh and cluster-tree networks (in the latter case they are also called cluster heads (CHs)). ZR provides synchronization services through beacon transmission.

3. End device (ZigBee end device, ZED): It is a device that does not implement the previous functionalities and should associate with a ZC or ZR before interacting with the network. In ZigBee, it is just a sensor/actuator node; it can be a reduced function device (RFD) that implements a reduced subset of the protocol stack [12].

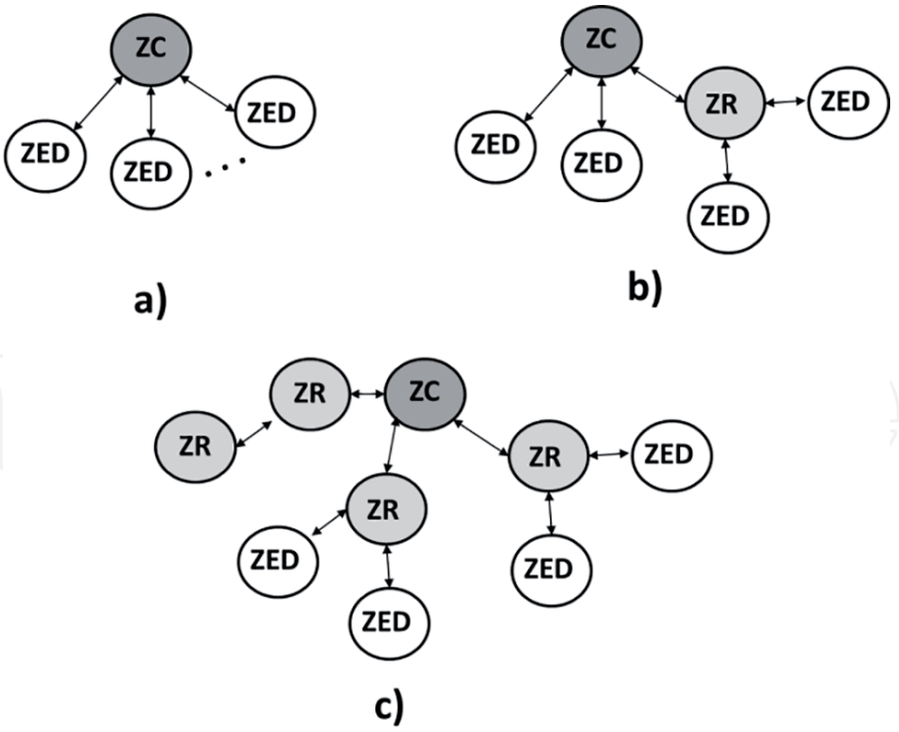


Figure 5.
ZigBee network topologies. (a) Star topology contains a unique node that operates as ZC, which establishes the PAN identifier. The identifier should not be used by any other ZigBee network in the vicinity. Also in the star topology, the communication is centralized, so each device (FFD or RFD) joining the network and willing to communicate with other devices must send its data to the ZC, which sends it to the adequate destination. (b) Mesh topology includes a ZC that identifies the entire network. Communication in this topology is decentralized, so each node can communicate directly with any other node within its radio. (c) In cluster tree topology, there is a single routing path between any pair of nodes, and there is a distributed synchronization mechanism (IEEE 802.15.4 beacon-enabled mode). There is only one ZC that identifies the entire network and one ZR per cluster. Any of the FFDs can act as a ZR that provides synchronization services to other devices and ZRs [13].

It is important to consider some operational considerations that may be presented by topologies for traditional wireless sensor networks (WSN). If you choose to use the star topology, you should keep in mind (a) that the sensor node selected as ZC will quickly consume its battery and (b) that the coverage of an IEEE 802.15.4/ZigBee cluster is very limited when addressing a large-scale WSN, leading to a scalability problem. On the other hand, the mesh topology enables enhanced networking, but it induces additional complexity to provide end-to-end connectivity between all nodes in the network. Therefore, unlike the star topology, the mesh topology can be more energy efficient, since the communication process does not depend on a particular node [14].

2.3 Wi-fi technology

Wi-Fi is the name given by the Wi-Fi Alliance [15] to the IEEE 802.11 suite of standards. 802.11 defined the initial standard for wireless local area networks (WLANs).

The evolution of Wi-Fi technology has focused on increasing speed, lower latency, and better user experiences in a multitude of environments and with a variety of device types. Wi-Fi Alliance has introduced generational names to devices and product descriptions. The latest generation of Wi-Fi devices, based on the 802.11ax standard, is known as Wi-Fi devices 6. If the device contains 802.11 ac, 5 GHz technology is known as Wi-Fi 5, or if the device uses technology 802.11n, 2.4 GHz is known as Wi-Fi 4 [16]. Generations of Wi-Fi prior to Wi-Fi 4 will not be assigned names. Most of devices available in the market today are identified as Wi-Fi 5.

DOI: http://dx.doi.org/10.5772/intechopen.89629

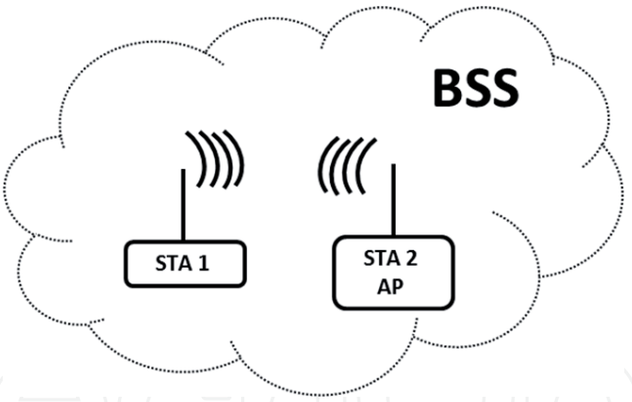


Figure 6.
BSS controlled by a single coordination function (CF). The CF determines when a STA transmits and when it receives.

Wi-Fi is a physical layer/link interface, as is Ethernet. A wireless station (STA) can be a personal computer (PC), a laptop, a personal digital assistant (PDA), or phone. When two or more STAs are connected wirelessly, they form a Basic Service Set (BSS) (**Figure 6**). This is the basic component of a Wi-Fi network [17].

Wi-Fi has two different operation modes: infrastructure mode and ad hoc mode. Each one uses the BSS, but they yield different network topologies.

1. Ad hoc mode: Wireless stations communicate directly with one another, with a peer-to-peer network model. A BSS operating in ad hoc mode is isolated, that is, there is no connection to other Wi-Fi or wired LAN networks. The utility of this network is in situations that demand a quick setup in places where there is no network infrastructure.

2. Infrastructure mode: This mode requires the BSS to contain a wireless access point (AP). An AP is an STA with additional functionality that allows extending access to wired networks for clients of a wireless network. Any wireless device that tries to join the BSS must first be associated with the AP. A distribution system (DS) is generated when an AP provides access to its associated STAs. The DS can allow communication between APs as shown in **Figure 7**.

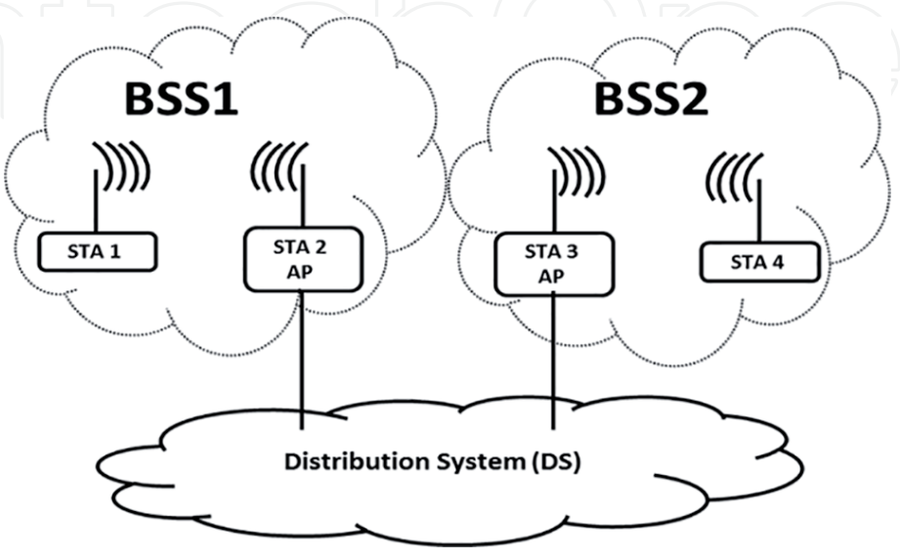


Figure 7.
All wireless communication to or from an associated STA goes through an AP. This type of setup is similar to the "star topology" used in wired networks.

2.3.1 Services specified by IEEE 802.11

The IEEE 802.11 standard does not define any specific implementations. Instead, nine services are specified that all implementations must provide; these are:

2.3.1.1 Station services (SS)

Authentication - The STA must identify itself to the AP before it can access network services.

De-authentication - This service voids an existing authentication.

Privacy - An STA must be able to encrypt the frame to protect the message content to be transmitted, so that only the recipient can read it.

MAC service data unit (MSDU) delivery - An MSDU is a data frame that must be transmitted to the proper destination.

2.3.1.2 Distribution system services (DSS)

A STA that functions as an AP must implement the following services:

Association - This service establishes an AP/STA mapping after mutually agreeable authentication has taken place between the two wireless stations. A STA can only associate with one AP at a time.

Re-association - This service allows you to change the current association from one AP to another AP.

Disassociation - This service voids a current association.

Distribution - This service handles delivery of MSDUs within the distribution system.

Integration - This service is the bridge function between wireless and wired networks. MSDU handles the delivery of between the distribution system and a wired LAN [17].

3. Hardware description

The elements used for the realization of the proposed system are shown in **Figure 8**. The platform is composed of four components: the FPGA board that includes A/D converter and three wireless interface Bluetooth, XBee (ZigBee protocol-based), and Wi-Fi module. The wireless modules provide the FPGA device the capacity to communicate with other system or the Internet.

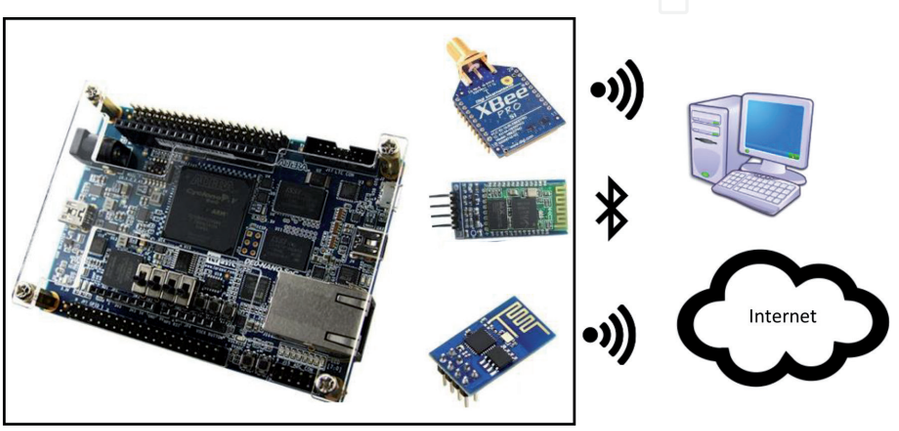


Figure 8.
Hardware components used for the real-time monitoring system.

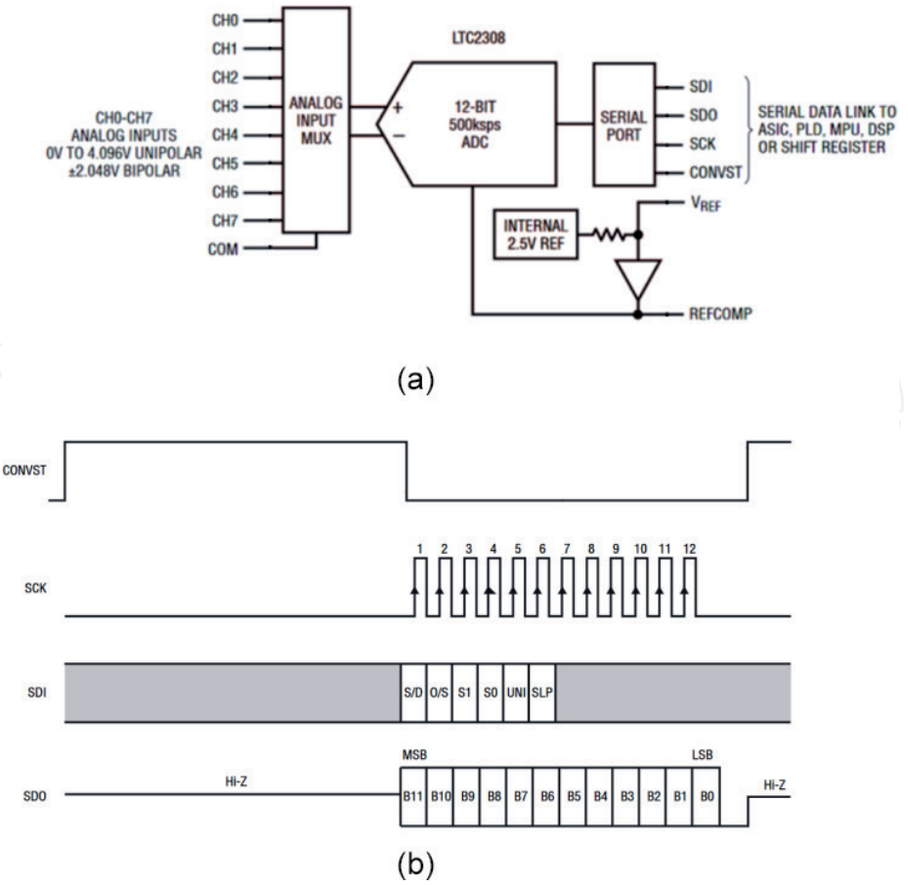


Figure 9.
(a) Block diagram LTC2308 device. Eight analog input and operation modes can be programmed by a 6-bit DIN word through SDI terminal. (b) Timing with a long pulse. The configuration signals are S/D can be single-ended/differential-bit; O/S can be odd/sing-bit; S1 and S0 addressing select bit; UNI can be unipolar/bipolar and SLP active sleep mode [19].

3.1 Analog/digital converter

The A/D converter chip used is the integrated circuit (IC) LTC2308, Linear Technology, whose characteristics are low noise and power consumption, up to 500 Kbps, 8-channel, 12-bit, and SPI/MICROWIRE compatible serial interface. The internal conversion clock allows the external serial output data clock (SCK) to operate at any frequency up to 40 MHz [18]. **Figure 9** shows the block diagram of ADC.

3.2 FPGA device

Cyclone V FPGA. The Intel FPGA Cyclone V SE 5CSEMA4U23C6N device has Dual ARM Cortex-A9 MPCore with Core Sight System on Chip (SoC), integrated circuit Cyclone V SE FPGA, with 40 K logic elements, maximum CPU clock frequency 925 MHz, 224 18×19 multipliers, and 5761 kb embedded memory (**Figure 10**).

3.3 Wireless modules

3.3.1 XBee Pro S1

The Digi XBee series modules implement the IEEE 802.15.4 radio and ZigBee networking protocol for its physical layer and MAC. Outdoor transmission distances to 0–90 meters depending on power output and environmental characteristics. XBee devices work in ISM 2.4 GHz frequency bands having a serial interface data

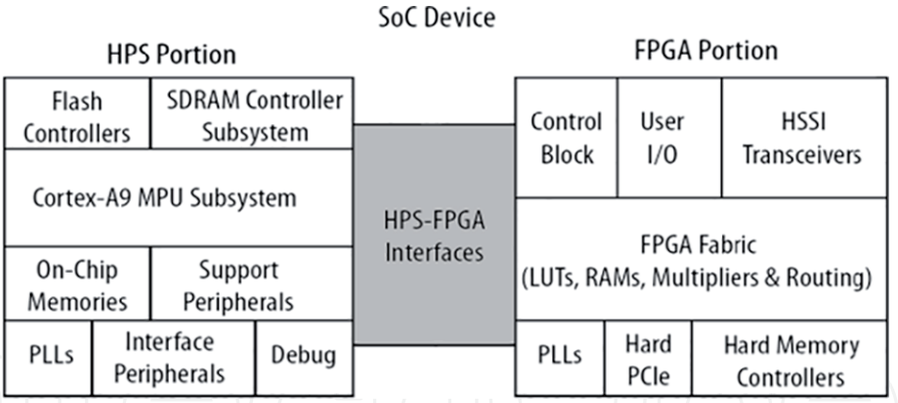


Figure 10.
Cyclone V SoC device block diagram is composed of two distinct portions: A dual-core ARM cortex-A9 hard processor system (HPS) and an FPGA. The cortex-A9 processor has two 32-bit CPUs and associated subsystems on the Intel Cyclone V SoC chip, where hardware circuits can be implemented, which reduce the size of the board and increase the performance of the developed system [20].

rate from 1200 bps to 250Kbps. The following are the supported network topologies: point-to-point, point-to-multipoint, and peer-to-peer.

3.3.2 HC-06 Bluetooth 2.0 EDR module

This module is a serial interface converter to Bluetooth adapter. HC-06 has a 2.4GHz digital wireless transceiver, low power consumption, an EDR module, the change range of modulation depth: 2Mbps–3Mbps, and standard HCI Port (UART or USB), and it can work at the low voltage (3.1–4.2 V). The module can be set by AT commands and have two modes, master and slave, but the mode cannot be switched during the process of communication. Serial baud rate is 1200–1,382,400 bps [21].

3.3.3 ESP8266 Wi-Fi module.

This module implements TCP/IP and full 802.11 b/g/n (support 2.4 GHz, up to 72.2 Mbps) WLAN MAC protocol. It can perform either as a stand-alone application or as the slave to a host MCU, so it supports Basic Service Set (BSS) STA and SoftAP operations under the distributed control function (DCF). ESP8266 includes a CPU Tensilica L106 32-bit processor, and it has peripheral interfaces: UART, SDIO, SPI, I2C, I2S, and IR. Power management is handled with minimum host interaction to minimize active duty period. ESP8266EX can be applied to any microcontroller design as a Wi-Fi adaptor through SPI/SDIO or UART interfaces [22].

4. System architectures

The design of an FPGA-based remote monitoring system architectures is show in **Figure 11**. The resultant design is implemented in VHDL and block diagrams; it is validated in co-simulation environment, and finally, it is tested in a real-time application to monitoring an electric signal.

There are three important features to consider before starting the development system: first, the nature of the feedback signal. If the sensor which measures the variable to be monitored has an analog nature, it is necessary to use an analog-to-digital converter (ADC) which has an output with a fixed bit width. Second: in order to avoid performing arithmetic operations between signals of different bit

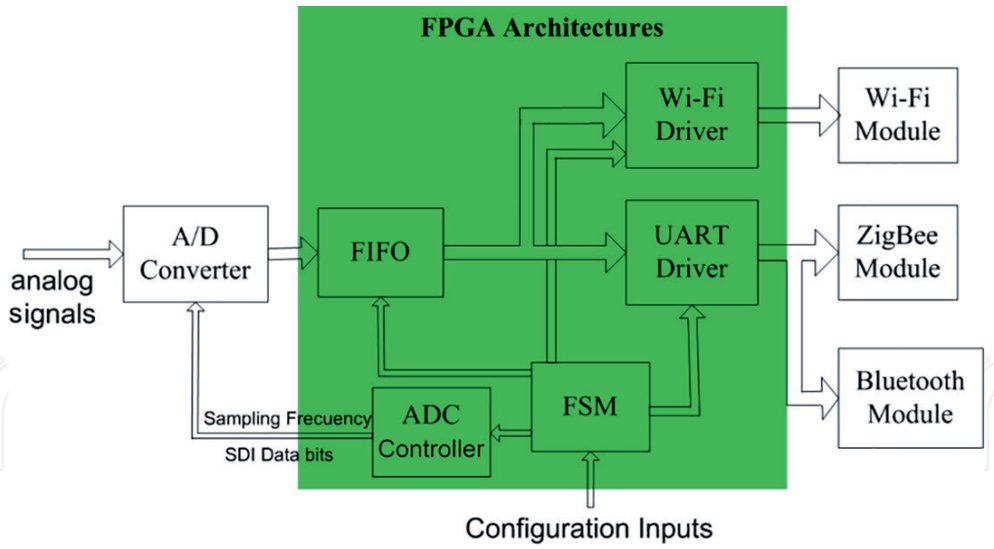


Figure 11.
Block diagram of architectures implemented on FPGA. This module comprises five blocks: ADC controller, FIFO memory, Wi-fi, UART drivers, and a finite state machine (FSM).

width, it is strongly recommended that the operations have the same bit width as the measured variable. Finally, the system output must be congruent with the bus width wireless interface.

4.1 A/D converter controller

The ADC LTC2308 operates on a 12-cycle operational frame, as shown in **Figure 9b**. ADC has four wires to control and communicate with the FPGA: SCLK, CS, DIN, and DOUT. The SCLK and CS signals are used to control the ADC. SCLK is the signal clock for the ADC. The CS signal serves as chip select for the ADC chip. The DIN and DOUT wires are used for transferring addresses and data between the two chips (ADC and FPGA). The FPGA uses the DIN connection to provide the address (3 bits in length) of the next channel requested for conversion. The DOUT connection is used by the ADC to send the digital value (12 bits long) of the converted signal to the FPGA. Both DIN and DOUT are sent in a serial manner at a rate of 1 bit per SCLK cycle [23].

In the case of our working example, SPI controller was developed to control the conversion process. A long CONVST pulse is used. **Figure 9b** shows time diagram to programming ADC. According to the diagram, "the conversions are initiated by a rising edge on the CONVST input. Once a conversion cycle has begun, it cannot be restarted. Between conversions, a 6-bit input word (DIN) at the SDI input configures the MUX and programs various modes of operation. As the DIN bits are shifted in, data from the previous conversion is shifted out on SDO. After the 6 bits of the DIN word have been shifted in, the ADC begins acquiring the analog input in preparation for the next conversion as the rest of the data is shifted out" [19]. **Figure 12** shows the block diagram architecture corresponding to SPI controller.

4.2 FIFO architecture

A dual-clock First-In First-Out (FIFO) buffer was used to cross data between the two different clock domains: sampling frequency A/D converter (from 1 to 25 MHz) and transmission rate (from 9600 to 921,600 bps), **Figure 13**. In the systems' clock frequency domain, the serialized outputs are continuously stored in 12 bits shift

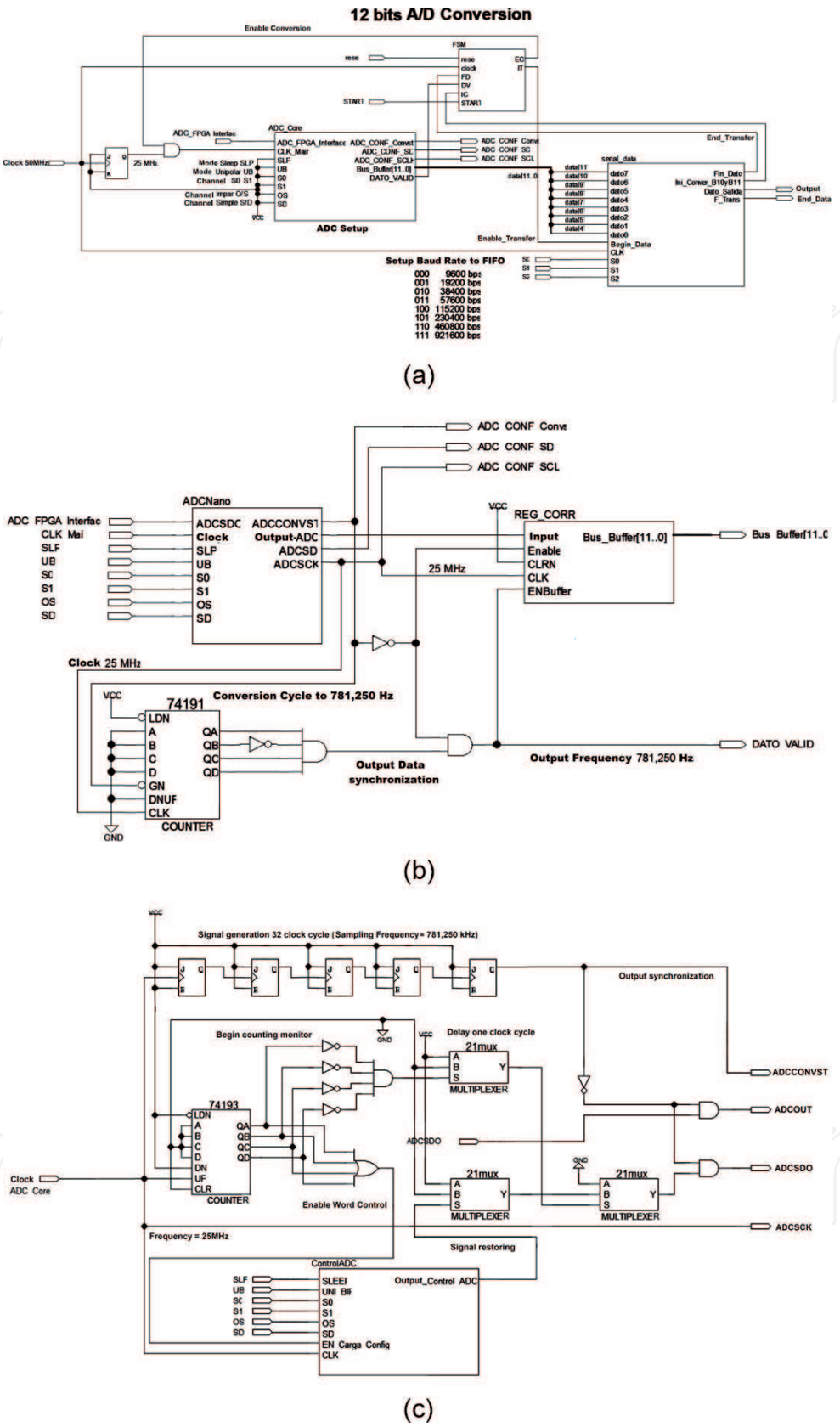


Figure 12.
SPI controller architecture. (a) 12 bits a/D conversion general architecture. (b) ADC_Core architecture. (c) ADC_Nano architecture generates signal control to ADC. The 4-bit counter counts 16 cycles in high for the acquisition of the signal and 16 cycles in low for the sending of the 12 output bits parallel to the configuration instruction for the next sample. The control ADC architecture is based on shift register.

register, before they will be sent to FIFO buffer. The finite state machine (FSM) FIFO, in the system controller, wait until collected data of the last active channel will be sent through wireless module, before starting a new acquisition.

DOI: http://dx.doi.org/10.5772/intechopen.89629

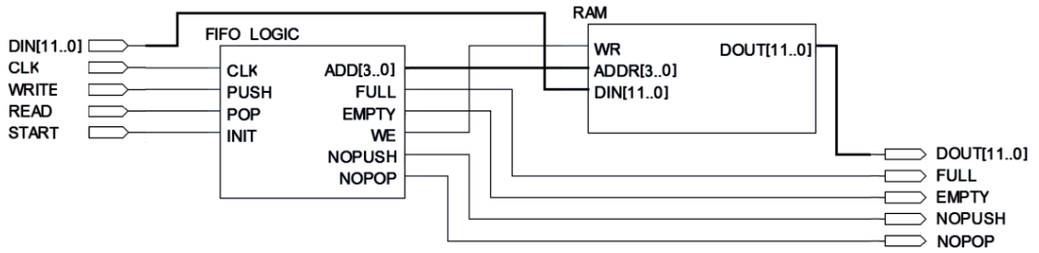


Figure 13.
Dual-clock FIFO architecture. Two counters are used to addressing the data to read and write operations. RAM of 12-bit and 16 words is used to store data.

The following source code corresponds to the FIFO_LOGIC and RAM entities of the design.

Code 1. FIFO_LOGIC.vhd [24].

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY FIFO_LOGIC IS
    PORT (
        CLK, PUSH, POP, INIT:           IN      STD_LOGIC;
        ADD:                            OUT     STD_LOGIC_VECTOR(3 DOWNTO 0);
        FULL, EMPTY, WE, NOPUSH, NOPOP: BUFFER  STD_LOGIC);
END ENTITY FIFO_LOGIC;

ARCHITECTURE BEHAVIOR OF FIFO_LOGIC IS
    SIGNAL WPTR, RPTR:  STD_LOGIC_VECTOR (3 DOWNTO 0);
    SIGNAL LASTOP:      STD_LOGIC;
BEGIN
    SEQUENTIAL: PROCESS (CLK)
    BEGIN
        IF (CLK'EVENT AND CLK = '1') THEN
            IF (INIT = '1') THEN
                WPTR <= (OTHERS => '0');
                RPTR <= (OTHERS => '0');
                LASTOP <= '0';
            ELSIF (POP = '1' AND EMPTY = '0') THEN          -- POP
                RPTR <= RPTR + 1;
                LASTOP <= '0';
                ELSIF (PUSH = '1' AND FULL = '0') THEN      -- PUSH
                    WPTR <= WPTR + 1;
                    LASTOP <= '1';
                END IF;
        END IF;
    END PROCESS SEQUENTIAL;

    COMBINATIONAL: PROCESS (PUSH, POP, WPTR, RPTR, LASTOP, FULL, EMPTY)
    BEGIN
        IF (RPTR = WPTR) THEN
            IF (LASTOP = '1') THEN
                FULL <= '1';
                EMPTY <= '0';
            ELSE
                FULL <= '0';
                EMPTY <= '1';
            END IF;
        ELSE
            FULL <= '0';
            EMPTY <= '0';
        END IF;

        IF (POP = '0' AND PUSH = '0') THEN      -- NO OPERATION --
            ADD <= RPTR;
            WE <= '0';
            NOPUSH <= '0';
            NOPOP <= '0';
        ELSIF (POP = '0' AND PUSH = '1') THEN   -- PUSH ONLY --
            ADD <= WPTR;
            NOPOP <= '0';
            IF (FULL = '0') THEN                -- VALID WRITE CONDITION --
                WE <= '1';
                NOPUSH <= '0';
            ELSE                                -- NO WRITE CONDITION --
                WE <= '0';
                NOPUSH <= '1';
            END IF;
            ELSIF (POP = '1' AND PUSH = '0') THEN -- POP ONLY --
                ADD <= RPTR;
                NOPUSH <= '0';
                WE <= '0';
                IF (EMPTY = '0') THEN           -- VALID READ CONDITION --
                    NOPOP <= '0';
                ELSE
                    NOPOP <= '1';               -- NO READ CONDITION --
                END IF;
            ELSE                                -- PUSH AND POP AT SAME TIME -
                IF (EMPTY = '0') THEN           -- VALID POP --
                    ADD <= RPTR;
                    WE <= '0';
                    NOPUSH <= '1';
                    NOPOP <= '0';
                ELSE
                    ADD <= WPTR;
                    WE <= '1';
                    NOPUSH <= '0';
                    NOPOP <= '1';
                END IF;
            END IF;
    END PROCESS COMBINATIONAL;
```

Code 2. RAM_16.vhd.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY RAM_16 IS
    PORT
    (
        WR:     IN STD_LOGIC;
        ADDR:   IN STD_LOGIC_VECTOR (3 DOWNTO 0)
        DIN:    IN STD_LOGIC_VECTOR (11 DOWNTO 0
        DOUT:   OUT STD_LOGIC_VECTOR (11 DOWNTO
    );
END ENTITY RAM_16;

ARCHITECTURE BEHAVIOR OF RAM_16 IS
SUBTYPE WORD IS STD_LOGIC_VECTOR (11 DOWNTO 0);
TYPE MEMORY IS ARRAY (0 TO 15) OF WORD;
SIGNAL RAM16: MEMORY;
BEGIN
    PROCESS (WR, DIN, ADDR)
    VARIABLE RAM_ADDR_IN: INTEGER RANGE 0 TO 15;
    BEGIN
        RAM_ADDR_IN := CONV_INTEGER (ADDR);
        IF (WR = '1') THEN
            RAM16 (RAM_ADDR_IN) <= DIN;
        END IF;
        DOUT <= RAM16(RAM_ADDR_IN);
    END PROCESS;
END ARCHITECTURE BEHAVIOR;
```

4.3 UART driver

Serial communications depend on the two UART devices (the FPGA architecture and the wireless module) to be configured with compatible settings: baud rate, parity, control (start and stop bits), and data bits (**Figure 14**).

In this system, a general port input/output (GPIO) is used to send serial data. Subsystem architecture (**Figure 15**) is used to set the baud rate in the output. UART interface will read out the data when it is filled in the FIFO and send to the host

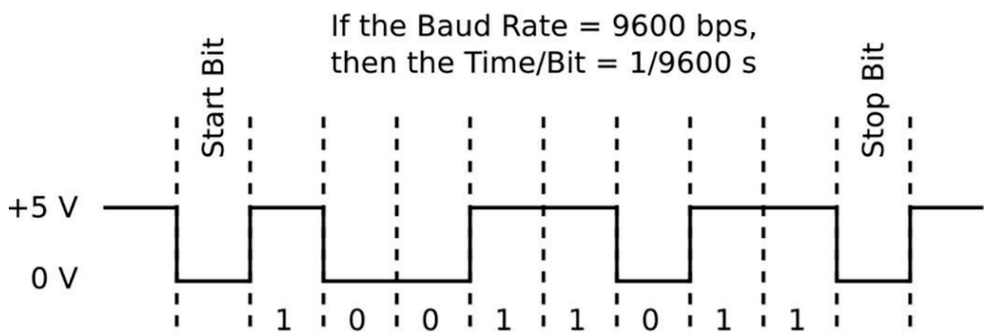


Figure 14.
UART data packet has data format structure: Data bits, parity, and stop bits. In the graph, the data 0x9B (decimal number "155," ASCII character "ø") is transmitted through the wireless module with format: 8-N-1 [25].

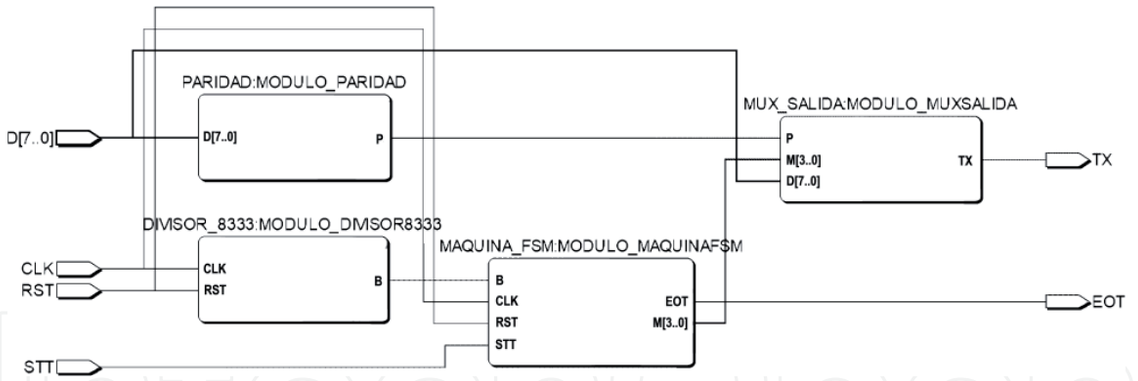


Figure 15.
UART driver diagram. Serial transmission uses baud rate module (DIVISOR_8333). MAQUINAFSM together with MUXSALIDA sends data from FIFO to serial data in the transmission format. The parity is verified with PARIDAD.

through the wireless link (Bluetooth or XBee modules), and finally the data can be displayed in the host with software application.

Code 3. DIVISOR_8333.vhd.

```
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY DIVISOR_8333 IS
  PORT(
  RST : IN  STD_LOGIC;
  CLK : IN  STD_LOGIC;
  B   : OUT STD_LOGIC
  );
END DIVISOR_8333;

ARCHITECTURE BEHAVIOR OF DIVISOR_8333 IS
SIGNAL QN, QP, A : STD_LOGIC_VECTOR(13 DOWNTO 0);
BEGIN
  A <= (OTHERS => '0');
  COMBINACIONAL:PROCESS(QP,A)
  BEGIN
    IF(QP = A) THEN
      B  <= '1';
      QN <= "01010001010111";
    ELSE
      B  <= '0';
      QN <= QP - 1;
    END IF;
  END PROCESS COMBINACIONAL;

  SECUENCIAL:PROCESS(RST,CLK)
  BEGIN
    IF(RST = '1')THEN
      QP <= (OTHERS => '0');
    ELSIF(CLK'EVENT AND CLK = '1')THEN
      QP <= QN;
    END IF;
  END PROCESS SECUENCIAL;
END BEHAVIOR;
```

Code 4. MUX_SALIDA.vhd.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY MUX_SALIDA IS
  PORT(
  M  : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
  D  : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
  P  : IN  STD_LOGIC;
  TX : OUT STD_LOGIC
  );
END MUX_SALIDA;

ARCHITECTURE SELECCION OF MUX_SALIDA IS
BEGIN
  PROCESS(M,D,P)
  BEGIN
    CASE M IS
      WHEN "0000" => TX <= '0';
      WHEN "0001" => TX <= D(0);
      WHEN "0010" => TX <= D(1);
      WHEN "0011" => TX <= D(2);
      WHEN "0100" => TX <= D(3);
      WHEN "0101" => TX <= D(4);
      WHEN "0110" => TX <= D(5);
      WHEN "0111" => TX <= D(6);
      WHEN "1000" => TX <= D(7);
      WHEN "1001" => TX <= P;
      WHEN OTHERS => TX <= '1';
    END CASE;
  END PROCESS;
END SELECCION;
```

Code 5. MAQUINA_FSM.vhd.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY MAQUINA_FSM IS
  PORT(
  RST : IN  STD_LOGIC;
  CLK : IN  STD_LOGIC;
  B   : IN  STD_LOGIC;
  STT : IN  STD_LOGIC;
  EOT : OUT STD_LOGIC;
  M   : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
  );
END MAQUINA_FSM;

ARCHITECTURE CONTROL OF MAQUINA_FSM IS
SIGNAL QN, QP : STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
  COMBINACIONAL:PROCESS(QP,STT,B)
  BEGIN
    CASE QP IS
        WHEN "0000" =>  IF (STT = '0') THEN QN <= QP;
                        ELSE QN <= "0001";
                        END IF;
                        EOT <= '1'; M <= "1111";
        WHEN "0001" =>  IF (B = '0') THEN QN <= QP;
                        ELSE QN <= "0010";
                        END IF;
                        EOT <= '0'; M <= "1111";
        WHEN "0010" =>  IF (B = '0') THEN QN <= QP;
                        ELSE QN <= "0011";
                        END IF;
                        EOT <= '0'; M <= "0000";
        WHEN "0011" =>  IF (B = '0') THEN QN <= QP;
                        ELSE QN <= "0100";
                        END IF;
                        EOT <= '0'; M <= "0001";
```

```
        WHEN "0100" =>  IF (B = '0') THEN QN <= QP;
                        ELSE QN <= "0101";
                        END IF;
                        EOT <= '0'; M <= "0010";
        WHEN "0101" =>  IF (B = '0') THEN QN <= QP;
                        ELSE QN <= "0110";
                        END IF;
                        EOT <= '0'; M <= "0011";
        WHEN "0110" =>  IF (B = '0') THEN QN <= QP;
                        ELSE QN <= "0111";
                        END IF;
                        EOT <= '0'; M <= "0100";
        WHEN "0111" =>  IF (B = '0') THEN QN <= QP;
                        ELSE QN <= "1000";
                        END IF;
                        EOT <= '0'; M <= "0101";
        WHEN "1000" =>  IF (B = '0') THEN QN <= QP;
                        ELSE QN <= "1001";
                        END IF;
                        EOT <= '0'; M <= "0110";
        WHEN "1001" =>  IF (B = '0') THEN QN <= QP;
                        ELSE QN <= "1010";
                        END IF;
                        EOT <= '0'; M <= "0111";
        WHEN "1010" =>  IF (B = '0') THEN QN <= QP;
                        ELSE QN <= "1011";
                        END IF;
                        EOT <= '0'; M <= "1000";
        WHEN "1011" =>  IF (B = '0') THEN QN <= QP;
                        ELSE QN <= "1100";
                        END IF;
                        EOT <= '0'; M <= "1001";
       WHEN OTHERS =>   QN  <= "0000";
                        EOT <= '0'; M <= "1111";
      END CASE;
    END PROCESS COMBINACIONAL;

    SECUENCIAL:PROCESS(RST,CLK)
    BEGIN
      IF(RST = '1')THEN
        QP <= (OTHERS => '0');
      ELSIF(CLK'EVENT AND CLK = '1')THEN
        QP <= QN;
      END IF;
    END PROCESS SECUENCIAL;
END CONTROL;
```

Code 6. PARIDAD.vhd.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY PARIDAD IS
  PORT(
  D : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
  P : OUT STD_LOGIC
  );
END PARIDAD;

ARCHITECTURE SIMPLE OF PARIDAD IS
BEGIN
  P <= ((D(0) XOR D(1)) XOR (D(2) XOR D(3))) XOR ((D(4) XOR D(5)) XOR (D(6) XOR D(7)));
END SIMPLE;
```

4.4 Bluetooth and XBee modules

The wireless modules are configured through AT commands. Command strings have the form ATxx (where xx is the name of a setting). The mode for both is slave to receive data from UART driver architecture. Bluetooth can be set to baud rate from 9600 to 921,600 bps. XBee can be set to baud rate from 9600 to 250,000 bps. Terminal software like Tera Term [26] can be used to have an initial configuration of the devices. Any USB to TTL converter, for example, PL2303HX device or similar, can be used.

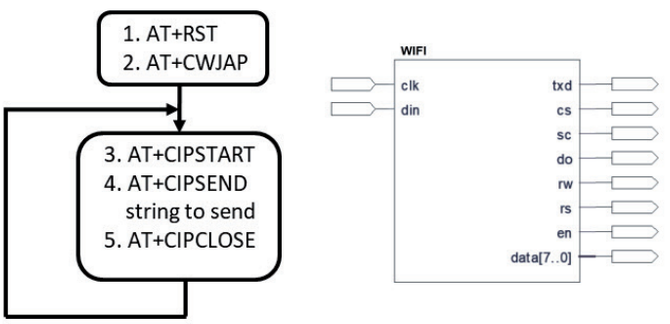
In the case of Bluetooth, the module only needs to be connected to the Rx of module to Tx of USB-TTL converter and Tx of module. It is necessary to connect ground and Vcc. HC-06 module is permanently configured to be slaved, and it

is always in AT mode when not paired to any other device. AT commands can be founded in datasheets [27].

XBee configuration needs a test utility (XCTU) to enable users to interact with radio frequency (RF) devices through a graphical interface. The application includes built-in tools that make it easy to set up, configure, and test RF devices. The software can be downloaded from the Internet [28].

4.5 Wi-fi driver

ESP8266 Wi-Fi module is used to transmit the sensor data wirelessly to the Wi-Fi modem at the other end with Internet connection. ESP8266 can be initialized using a set of AT commands. Initialization process includes (a) verifying the communication between ESP8266 module and FPGA architecture (RST command) and (b) searching for a Wi-Fi network within its range and connecting to it, with the required credentials (CWJAP command). Sending process includes (a) setting the Wi-Fi module as a TCP/IP client (CIPSTART command); (b) transmitting data involves communication with cloud server using IP address (CIPSEND command). Address IP of the server is required to access the data from personal computing devices such laptop, tablet, and smartphone. **Figure 16** shows the AT command sequence and a block of Wi-Fi architecture:



(a) (b)

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY WIFI IS
    PORT (
      CLK   : IN  STD_LOGIC;
      TXD   : OUT STD_LOGIC;
      CS    : OUT STD_LOGIC;
      SC    : OUT STD_LOGIC;
      DO    : OUT STD_LOGIC;
      DIN   : IN  STD_LOGIC;
      RW    : OUT STD_LOGIC;
      RS    : OUT STD_LOGIC;
      EN    : OUT STD_LOGIC;
      DATA  : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END WIFI;

ARCHITECTURE BEHAVIOR OF WIFI IS

TYPE ARR IS ARRAY (1 TO 167) OF STD_LOGIC_VECTOR(7 DOWNTO 0);

CONSTANT STR: ARR:= ( X"41",X"41",X"54",X"2B",X"52",X"53",X"54",X"0D",X"0A",                                    -- ATT+RST <CR+LF>
X"41",X"54",X"2B",X"43",X"57",X"4A",X"41",X"50",X"3D",                                                          -- AT+CWJAP=
X"22",X"4C",X"61",X"62",X"5F",X"49",X"45",X"45",X"5F",X"69",X"6E",X"76",X"69",X"74",X"61",X"64",X"6F",X"22",X"2C",   -- "Lab_IEE_invitado",
X"22",X"45",X"4C",X"45",X"43",X"54",X"52",X"49",X"43",X"41",X"22",X"0D",X"0A",                                   -- "ELECTRICA" <CR+LF>
X"41",X"54",X"2B",X"43",X"49",X"50",X"53",X"54",X"41",X"52",X"54",X"3D",X"22",X"54",X"43",X"50",X"22",X"2C",    -- AT+CIPSTART="TCP",
X"22",X"31",X"38",X"34",X"2E",X"31",X"30",X"36",X"2E",X"31",X"35",X"33",X"2E",X"31",X"34",X"39",X"22",X"2C",X"38",X"30",X"0D",X"0A", -- "184.106.153.149",80
X"41",X"54",X"2B",X"43",X"49",X"50",X"53",X"45",X"4E",X"44",X"3D",X"34",X"39",X"0D",X"0A",                     -- AT+CIPSEND=49 <CR+LF>
X"47",X"45",X"54",X"20",X"2F",X"75",X"70",X"64",X"61",X"74",X"65",X"3F",X"61",X"70",X"69",X"5F",X"6B",X"65",X"79",X"3D",-- GET /update?api_key=
X"51",X"48",X"53",X"36",X"45",X"34",X"48",X"35",X"57",X"34",X"4D",X"42",X"55",X"39",X"5A",X"34",X"26",X"66",X"69",X"65",X"6C",X"64",X"31",X"3D",X"31",X"0D",X"0A
X"41",X"54",X"2B",X"43",X"49",X"50",X"43",X"4C",X"4F",X"53",X"45",X"0D",X"0A");                                -- AT+CIPCLOSE <CR+LF>
```

(c)

Figure 16.
AT command sequence. (a) Flow diagram of WEB connection [27]. (b) Block WI-FI driver module. (c) Code for AT command definition inside FPGA; green letters are comments about hexadecimal data.

DOI: http://dx.doi.org/10.5772/intechopen.89629

5. Experimental results on graphical user interface (GUI)

An analog signal is generated by the function generator to test the system, and the final data sent to the PC or WEB page is observed. **Figures 17**, **18** and **19** show the corresponding practical wave and storage wave.

The GUI (**Figure 18**) was made using Java Eclipse Oxygen [29] and serial communication libraries (jSerialComm). jSerialComm is a Java library designed to provide a platform-independent way to access standard serial ports without requiring external

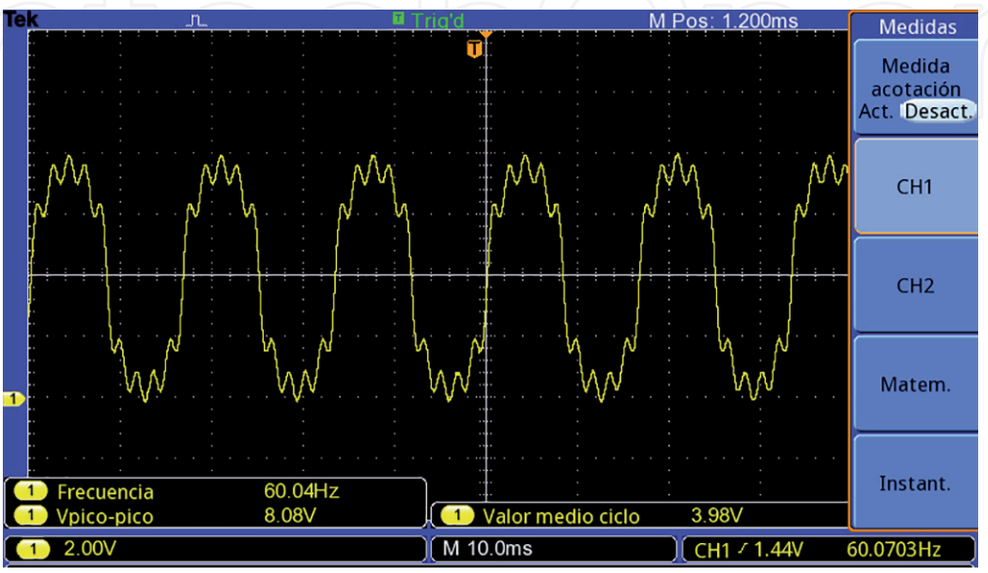


Figure 17.
Measurements of real signal sent to the host and WEB page. The signal has an offset = 3.98Vdc and 8.06Vpp and frequency of 60 Hz with harmonics of 3rd, 5th, 7th, and 9th. This signal is obtained from digital oscilloscope.

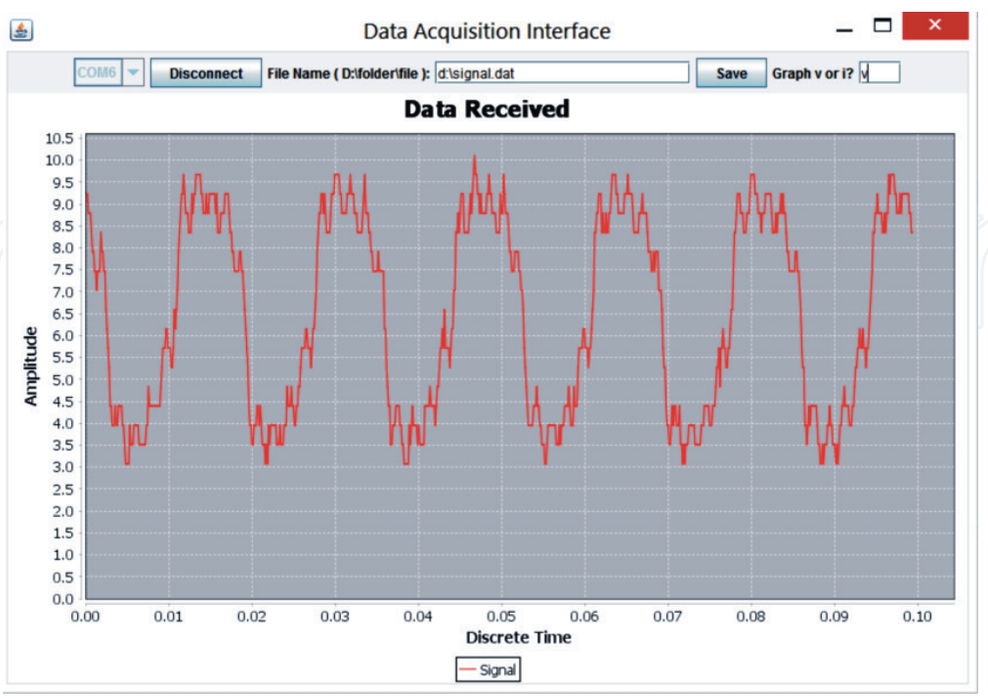


Figure 18.
Data received from the remote DAQ system (Bluetooth or XBee module) using GUI development. Each cycle is represented by 133 samples (sampling frequency = 8 kHz). The UART baud rate is 115,200 bps.

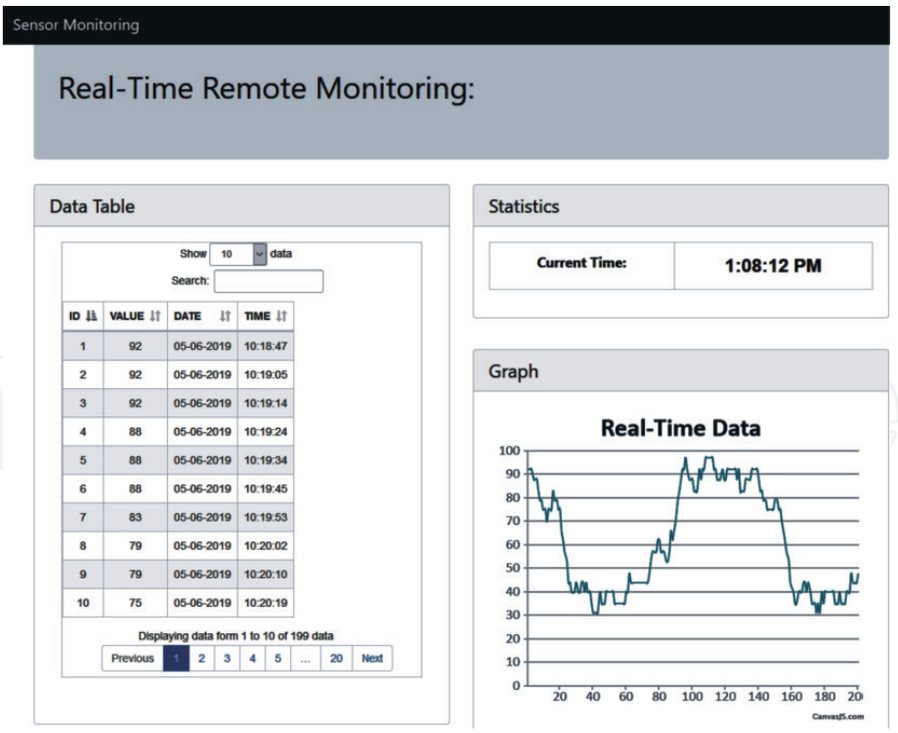


Figure 19.
Data sent to WEB page through Wi-fi module. The WEB page was made on a XAMPP package that includes apache WEB server, MySQL, and PHP [31].

libraries, native code, or any other tools. It is meant as an alternative to Rx-Tx and the (deprecated) Java Communications API, with increased ease of use, an enhanced support for timeouts, and the ability to open multiple ports simultaneously [30].

6. Conclusions

This chapter described a data acquisition system based on FPGA. Several architectures to ADC controller, UART communication, FIFO memory, and Wi-Fi configuration process were made to develop the system. Experiments show that the system can convert the analog signals to digital signal and send to host computer to Java GUI or WEB page in real-time. The data can be acquired by using custom sampling frequency and baud rate. The entire system is designed to be simple, stable, and low cost.

DOI: http://dx.doi.org/10.5772/intechopen.89629

Author details

J. Guadalupe Velásquez-Aguilar*, Outmane Oubram and Luis Cisneros-Villalobos
Faculty of Chemical Sciences and Engineering, Department of Electrical Engineering, Autonomous University of the State of Morelos, Cuernavaca, Morelos, México

*Address all correspondence to: jgpeva@uaem.mx

References

[1] Siewert S, Pratt J, editors. Real-time embedded components and system with linux and RTOS. Mercury Learning and Information LLC; 2016. 483 p. ISBN: 978-1942270041

[2] Mohit A editor. Embedded system design: Introduction to SoC system architecture. Learning Bytes Publishing; 2016. 214 p. ISBN: 978-0997297201

[3] Rajsuman R editor. System-on-a-Chip: Design and Test. Artech House; 2000. 294 p. ISBN: 978-1580531078

[4] Jerraya AA, Mint WW. Hardware/software interface codesign for embedded systems. Computer, IEEE. 2005;**38**:63-69

[5] Kirianaki N, Yurish S, Shpak N, Deynega V. Data Acquisition and Signal Processing for Smart Sensors. John Wiley & Sons Ltd; 2002. 291 p. ISBN: 0-470843179

[6] Velásquez-AguilarJG, Aquino-RobleroF, Limón-MendozaM, Cisneros-VillalobosL, Zamudio-Lara A. Multi-channel data acquisition and wireless communication FPGA-based system, to real-time remote monitoring. In: 2017 International Conference on Mechatronics, Electronics and Automotive Engineering (ICMEAE); 21-24 November 2017; Cuernavaca, México. IEEE; 2017. pp. 181-186

[7] Collotta M, Pau G, Salerno VM, Scatá G, editors. Wireless Sensor Networks to Improve Road Monitoring. IntechOpen; 2012. pp. 323-346. DOI: 10.5772/48505.ch15

[8] National Instruments. Introduction to Bluetooth Device Testing: From Theory to Transmitter and Receiver Measurements. Available from: http://download.ni.com/evaluation/rf/intro_to_bluetooth_test.pdf

[9] Symmetry Electronics, Bluetooth 1.0 vs 2.0,vs 3.0 vs 4.0 vs 5.0—How They Compare. Available from: https://www.semiconductorstore.com/blog/2018/Bluetooth-1-0-vs-2-0-vs-3-0-vs-4-0-vs-5-0-How-They-Differ-Symmetry-Blog/3147

[10] Gupta NK, editor. Inside Bluetooth Low Energy. Artech House; 2016. 458 p. ISBN: 978-1630810894

[11] National Instruments. The Basic of ZigBee Transmitter Testing. Available from: www.ni.com

[12] Jaiswal L, Kaur J, Singh G. Performance analysis of backoff exponent behaviour at MAC layer in ZigBee sensor networks. International Journal of Computer Applications. 2012;**57**(22)

[13] Cunha A, Koubâa A, Severino R, Alves M. Open-ZB: An open-source implementation of the IEEE 802.15.4/ZigBee protocol stack on TinyOS. Available from: https://www.cister.isep.ipp.pt/docs/open_zb__an_open_source_implementation_of_the_ieee_802_15_4_zigbee_protocol_stack_on_tinyos/381/view.pdf

[14] Tennina S et al. editors. IEEE 802.15.4 and ZigBee as enabling Technologies for Low-Power Wireless Systems with Quality-of-Service Constraints. Springer; 2013. 173 p. DOI: 10.1007/978-3-642-37368-8

[15] Available from: https://www.wi-fi.org

[16] Wi-Fi Alliance. Generational Wi-Fi User Guide. Available from: https://www.wi-fi.org/file/generational-wi-fi-user-guide

[17] Rabbit Web site. An Introduction to Wi-Fi. Available

from: http://ftp1.digi.com/support/documentation/0190170_b.pdf

[18] Terasic, DE0-Nano-SoC User Manual. Available from: https://media.digikey.com/pdf/Data%20Sheets/Terasic%20Technologies/DE0-Nano-SoC_UM.pdf

[19] LTC2308 Datasheets, Linear Technology Corporation, 2007

[20] Cyclone V. Hard Processor System Technical Reference Manual, Intel FPGA; 2018. cv_5v4 | 2019.06.14

[21] HC-06 Datasheets, Guangzhou HC Information Technology Co. Ltd. 2011. Available from: https://www.olimex.com/Products/Components/RF/BLUETOOTH-SERIAL-HC-06/resources/hc06.pdf

[22] Espressif. ESP8285 Datasheet. Available from: https://www.espressif.com/sites/default/files/documentation/0a-esp8285_datasheet_en.pdf

[23] Altera, Using the DE0-Nano ADC Controller. Available from: ftp://ftp.intel.com/Pub/fpgaup/.../Using_DE0-Nano_ADC.pdf

[24] Stroud CE. First-In First-Out (FIFO) Control Logic VHDL Modeling Example, ECE Department, Auburn University. Available from: http://www.eng.auburn.edu/~strouce/class/elec4200/vhdlmods.pdf

[25] Digi International Inc. XBee-PRO 900/ DigiMesh 900 OEM RF Modules. Available from: https://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-900-Manual.pdf

[26] Available from: https://tera-term.en.lo4d.com/windows

[27] Espressif. ESP8266 AT Command Examples. 2017. Available from: https://espressif.com/sites/default/files/.../4b-esp8266_at_command_examples_en.pdf

[28] Available from: https://www.digi.com/products/iot-platform/xctu#productsupport-utilities

[29] Available from: https://www.eclipse.org/

[30] Available from: https://fazecast.github.io/jSerialComm/

[31] Available from: https://sourceforge.net/projects/xampp/