DOI: 10.2478/sbeef-2021-0003

APPLICATIONS MODELS USING BEAGLEBONE FPGA

S. SOARE[1], G. PREDUSCA[2], E. DIACONU[2]
[1]IDEMIA, Bucharest, [2]Department of Electronics, Telecommunications and Energy Engineering, Valahia University of Targoviste
E-mail: steven.edg@gmail.com, gabriel.predusca@valahia.ro, emy_diaconu@yahoo.com

Abstract. ***In this paper it is proposed to offer solutions using the BeagleBone Black FPGA development board, together with a series of electronic devices: LEDs, ultrasonic sensors, proximity sensors. The applications were developed using various existing programming languages such as: JavaScript, Python, Ruby but also using various commands specific to the Linux operating system. Also, the entire work was implemented using an online programmable cloud environment, called Cloud9 IDE that allows accessing and modifying applications developed at will from any location with an active internet connection.***

Keywords: FPGA, BeagleBone, PIR, Cloud9 IDE

1. INTRODUCTION

One way to increase the reliability of electronic equipment is to reduce the number of connections and hence the idea of making electronic System-on-Chip (SoC) [1].

Embedded Systems have covered almost all aspects of modern life: GSM, digital camera, GPS, printers, various automations, routers, network equipment, vehicles, etc.

Such a system can be realized in several technologies. The most used in the production of SoC are two technologies: Standard Cells (SC) and Programmable Logic Device (PLD).

PLDs are integrated circuits that contain many logic gates that can be connected through connections and can be programmed to implement any logic function.

Programmable Gate Array (FPGA) are arrays of programmable logic gates in the electric field and have greater complexity than PLD [2].

Any application implemented with Complex Programmable Logic Device (CPLD) or FPGA circuits can have its operating algorithm modified, without the need to replace hardware, but only insert, modify or remove certain lines of code from the program in the device's memory [1].

2. BEAGLEBONE PLATFORM

A development board is a type of minicomputer based on a microprocessor and the minimum logic necessary for a user to become familiar with the technologies and features offered by it to finally learn to program and use it in various projects. In contrast to a typical computer, a development board does not have dedicated hardware to operate, instead they are open-source platforms compatible with many technologies and programmable environments on the market [3].

BeagleBone is a "single board computer" development board produced by Texas instruments in collaboration with Digi-Key and Newark element 14. The design of the board is open-source and was built in this way to demonstrate the capabilities of SoC (system-on-a-chip) OMAP353 (Figure 1) [4-7].

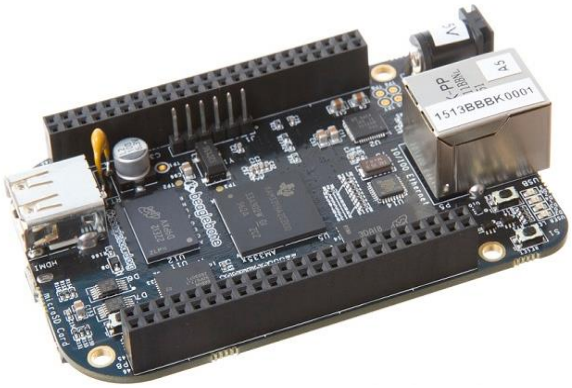

Figure 1. BeagleBone black rev. C platform

The SoC includes a Cortex-A8 processor (Sitara AM3358BZCZ100) that can run Linux, Minix, FreeBSD, OpenBSD, RISC OS, Symbian or even Android.

The board features a TMS320C64X + DSP (Digital Signal Processor) for audio and video decoding and a PowerVR SGX530 GPU (Graphic Processing Unit). The card also has a microSD slot, a USB port, an RS-232 serial connection and 2 stereo 3.5 mm input/output jacks for the audio part.

The storage memory is present through a PoP (package on package) chip that includes up to 4GB of flash memory and 512 MB of RAM. The board uses up to 2W and can be powered directly via the USB provided or using a separate power supply of up to 5V. Due to the low power consumption, the board does not require separate cooling.

The processor behind the BeagleBone platform is an ARM (Advanced RISC Machine) processor, a family of

RISC (Reduced Instruction Set Computing) processors configured for different electronic environments.

The main components of the platform are:
• TPS65217C PMIC - provides power management to various components;
• SMSC Ethernet PHY - physical network interface;
• HDMI Framer - provides HDMI or DVI-D connection to a display using an adapter;
• 5VDC - main 5V power supply;
• 10/100 Ethernet - RJ45 type connector that connects to a LAN;
• Serial Debug - serial port for debugging;
• USB Client - is a USB type connector to a PC and represents the secondary power supply;
• BOOT Switch - can be used to "force" a boot sequence from a microSD card;
• User LEDs - 4 blue LEDs that can be used by the user;
• Reset Button - used to reset the processor;
• microSD slot - the place where a memory card can be inserted;
• microHDMI - connector that interfaces with a display;
• USB Host - can be connected to various USB interfaces such as: Wi-fi, Bluetooth, Keyboard, Mouse, etc (Figure 2).

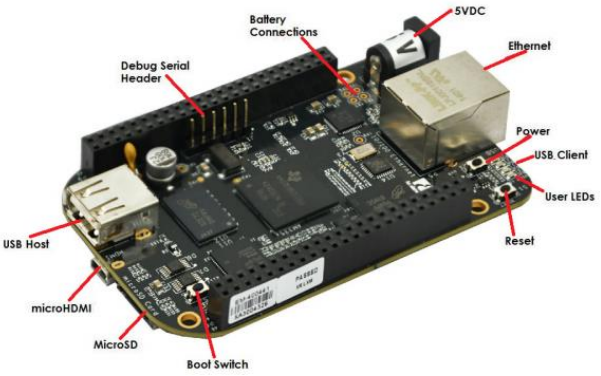

Figure 2. Location of connectors, LEDs and switches

The BeagleBone platform offers extraordinary Input/Output capabilities, each digital I/O pin having 8 different modes of operation, including GPIO (General-Purpose Input/Output):
• 65 digital I / O;
• 8 PWMs (Pulse-Width Modulation) and 4 clock signals;
• 7 analog inputs (1.8V);
• 4 UART (Universal Asynchronous Receiver/ Transmitter);
• 2 I2C ports (Inter-Integrated Circuit);
• 2 SPI (Serial Peripheral Interface) ports.

3. HC-SR04 ULTRASONIC SENSOR

The ultrasonic module offered by the HC-SR04 sensor offers a non-contact measurement function that can help up to 400 centimetres, its accuracy being very good, it can detect objects in its vicinity up to 3 millimetres (Figure 3) [8].

Figure 3. HC-SR04 ultrasonic sensor

The sensor includes an ultrasonic transmitter and a control and data reception circuit. Powering the sensor with a current for 10 microseconds creates an impulse to the "trigger" input and thus starts the scan, then the module will send 8 sound cycles at a frequency of 40 kHz to generate an echo. The echo represents the distance to a distant object (Figure 4). The distance to the object can be calculated by measuring the time interval between sending the signal and receiving the signal echo. The calculation formula is as follows:

Distance = (maximum response time x received signal speed (340m /s))/2.

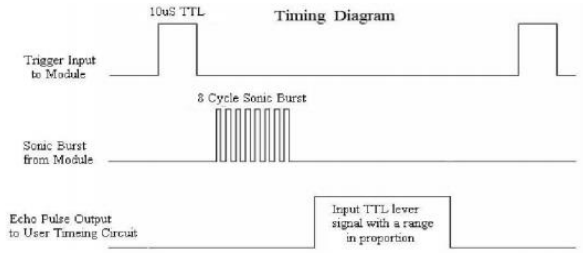

Figure 4. How the ultrasonic sensor works.

4. PIR (PASSIVE INFRARED) SENSOR

The HM300303 infrared sensor is used to detect the presence of nearby objects (Figure 5). It is mainly used in automatic light off / on systems in a room. It supports a higher supply voltage, it can detect objects up to 5 meters and has a sensitivity range of up to 110 degrees. The white cover works like a divergent lens which allows the sensor to detect objects and people over a much wider range. The sensor "reads" data at an adjustable time interval by the user, the interval is between 0.3 seconds and about 5 minutes.
Technical specifications [9, 6]:
• Supply voltage - between 5V and 20V;
• Current: 65mA;
• TTL digital output: 3.3V/0V;
• Adjustable delay: 0.3sec - 5min;
• Range: 110 degrees, 5 meters;
• Trigger L / H;
• Operating temperature: $-15^0 \ldots +70^0$ Celsius.

Figure 5. Motion detection sensor (PIR)

5. INSTALLING THE BEAGLEBONE PLATFORM

The first step in developing applications using the BeagleBone platform was to install it, which consisted of a simplistic process, but which required a multitude of steps.

The card can be connected to any computer via a mini-usb connection. As soon as the card is recognized as a device by the operating system used, it is necessary to install the corresponding drivers. To do this, access the ip 192.168.7.2, ip that hosts a web page with what the host has on the board. The host can be accessed regardless of whether there is an internet connection using any web browser (Figure 6) [10].

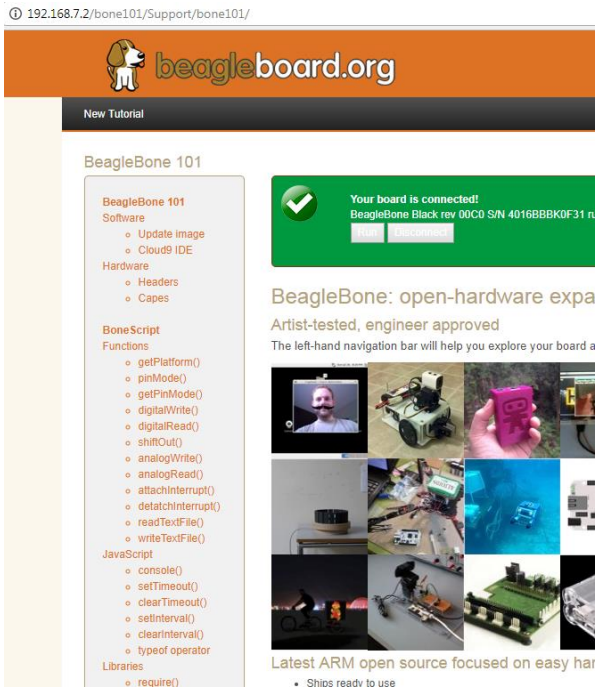

Figure 6. BeagleBone platform setup

Installing the drivers that will interface between BeagleBone and the used PC takes about 10 minutes, the process being intuitive (figure 7).

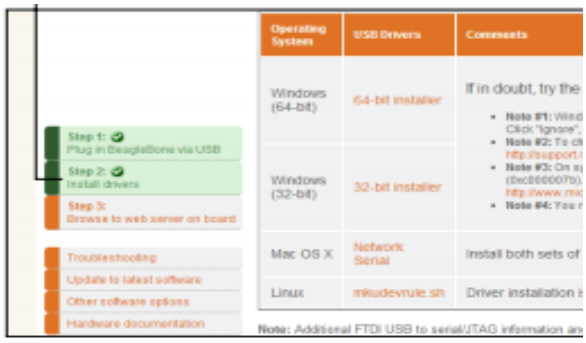

Figure 7. Steps for installing BeagleBone platform drivers.

The second step to be able to use the BeagleBone platform to develop an application is represented by the update of the Linux distribution running on the board. Linux was created as a "free" operating system for personal computers.

BeagleBone comes pre-installed with a version of the Linux distribution called Angstrom, but I chose to use a distribution called Debian because it is currently the distribution with the widest software support in embedded systems.

To change the Linux distribution on the BeagleBone board you need some additional software as well as a stable internet connection and a microSD card reader [11-14]. The process is a laborious one that requires the following steps:
- download distributions compatible with the BeagleBone board can be found at the link http://beagleboard.org/latest-images.
- using 7-zip software to extract the .img file. The image will be written on a microSD card. To do this I used a card reader adapter plus Win32 Disk Imager software.
- I used Win32 Disk Imager to write the image.
- After loading the image on the microSD card, it is inserted in BeagleBone, the card slot.
- To replace the Angstrom distribution on the BeagleBone board eMMC, the "flashing" process must be initialized, this is done by pressing the boot button for 10 seconds which will start the installation of the Debian distribution and the approximate duration is 45 minutes.
- After installing the Linux distribution, accessing the operating system through an SSH (Secure Shell) client and installing the libraries necessary to run the code used in applications. To access the BeagleBone platform, it must be connected to a local router, after which we need to identify its IP.
- After finding out the ip, in my case 192.168.1.xxx, I used an SSH client, Bitvise Client software because in addition to the SHH connection with the card it also offers an FTP connection (File Transfer Protocol) that allows me to see the contents of directories in the Linux distribution.
- After accessing the operating system, it is necessary to install certain libraries that help to run applications on the platform; libraries will contain compilers for the following programming languages: python, JavaScript, c ++, and several modules that will interface with these

languages, including examples including bonescript and socket io. Required library installation commands:

- **curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -**
- **sudo apt-get install -y nodejs** - > Javascript compiler setup
- **yum install gcc-c++ make** -> C++ compiler setup
- **sudo apt-get install python2.7** -> Python compiler setup
- **sudo npm install socket.io** -> Socket IO library setup
- **npm install bonescript** -> Bonescript library setup

Because both browsing and compiling plus writing code in a Linux distribution is a painstaking and difficult process to master, I chose to use a more elegant solution, namely connecting the BeagleBone Black platform to a cloud service that uses a "user" interface. friendly "and which can be accessed without the help of an SSH client directly from the browser (Figure 8).

Figure 8. Cloud9 IDE interface

Cloud9 IDE is an integrated open-source online development environment. It supports hundreds of programming languages such as: C, C ++, Ruby, Perl, Python, Javascript in combination with Node.js and Go. The biggest advantage of using this service is that it helps users start coding projects directly by providing pre-installed "workspaces", connecting with other people doing similar projects, and offering options for web-development such as "Live Preview" and "Browser Compatibility Testing" [15].

Some features of the interface are:
• "built-in" terminal that can run unix and npm commands;
• automatic completion of code and identifiers;
• multiple sliders for simultaneous editing of several processes;
• JavaScript language analysis in "Realtime" mode;
• the possibility to transform functions from other languages into JavaScript functions;
• error detection by line warnings that require debugging;
• a multitude of themes and an easy-to-change appearance;
• predefined buttons;
• built-in image editor;
• the ability to load projects through the "drag and drop" option;
• support for the following code archives: GitHub, Bitbucket, Mercurial, GiT, FTP servers;
• support for "deployment" platforms: Heroku, Joyent, OpenShift, Windows Azure.

After installing all the necessary scripts, access the platform by filling in the ip address of the BeagleBone Black board followed by the port through which the communication is made. In my case 192.168.1.xxx:3000 (the ip assigned by the router to connect to the platform in the online environment) but the platform also works offline once installed in the Linux version through the 192.168.x.x:3000 IP (Figure 9).

Figure 9. BeagleBone platform access

6. APPLICATIONS

6.1 Application 1: Hello World!!! / Blink LED on-board / Binary representation using on-board LEDs.

A "Hello World" type application for the BeagleBone Black platform consists in programming the USR0, USR1, USR2, USR3 LEDs (Figure 10).

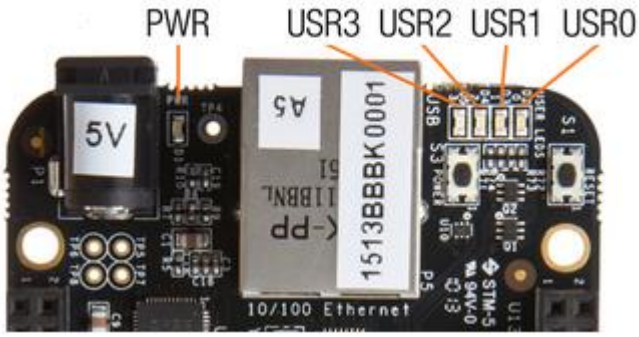

Figure 10. LED programming

The programming language used is Javascript and the library I called is called Bonescript. The source code used is as follows (Figure 11).

```
var b=require('bonescript');
b.pinMode('USR0', 'out');
b.digitalWrite('USR0', 1);

console.log('LED-ul este aprins');
```

```
bash - "beaglebor ×   Primele\ Aplicatii/ ×
Run   Run Co   Command:   Primele\ Aplic
debugger listening on port 15454
LED-ul este aprins
```

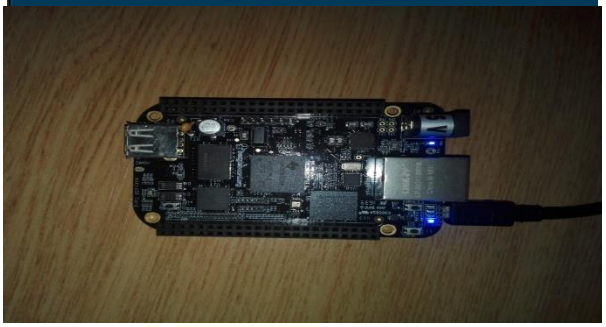

Figure 11. First application

The next step was to create a code that would make one of the LEDs turn off and on at a predefined time interval. Javascript code run through the Cloud 9 platform being (Figure 12):

```
var b = require('bonescript');
// Crearea unei variabile numite led, ce apeleaza led-ul U
var led = "USR3";
// Initializarea led-ului ca un OUTPUT
b.pinMode(led, b.OUTPUT);
// Crearea unei variabile de stare, ce memoreaza starea ac
var state = b.LOW;
// Setarea LED ca LOW (stins)
b.digitalWrite(led, state);
// Executarea functiei de intrerupator o data la o secunda
setInterval(toggle, 1000);
// Functie ce seteaza starea led-ului ca HIGH (aprins) or
function toggle() {
    if(state == b.LOW) state = b.HIGH;

 else state = b.LOW;

 b.digitalWrite(led, state); // scrierea noii stari a led-
 console.log('LED is blinking');
}
```

Figure 12. Javascript code for application 1

Console output (Figure 13):

```
bash - "beaglebone"
Stop   Run
LED is blinking
LED is blinking
LED is blinking
LED is blinking
LED is blinking
```

Figure 13. Console output

The code can be easily modified to turn on and off all the LEDs simultaneously at set time intervals as in Figure 14.

```
var b = require('bonescript');
 var state=b.LOW
b.pinMode('USR0', b.OUTPUT);
b.pinMode('USR1', b.OUTPUT);
b.pinMode('USR2', b.OUTPUT);
b.pinMode('USR3', b.OUTPUT);
b.digitalWrite('USR0', b.HIGH);
b.digitalWrite('USR1', b.HIGH);
b.digitalWrite('USR2', b.HIGH);
b.digitalWrite('USR3', b.HIGH);
setInterval(toggle, 2000);

function toggle() {
    if(state == b.LOW) state = b.HIGH;
    else state = b.LOW;
    b.digitalWrite("USR3", state);
    b.digitalWrite("USR0", state);
    console.log('LEDs are blinking');
}
```

Figure 14. LED on/off simultaneously

A more complex application using on-board LEDs was to represent the numbers from 0 to 15 in binary using each LED as a bit. USR3 = 20, USR2 = 21, USR1 = 22, USR0 = 23. The application was developed in the Ruby language and the Ruby code run through the Cloud9 platform (Figure 15):

```
#!/usr/bin/env ruby
require 'beaglebone'
include Beaglebone

# Crearea unui obiect pentru fiecare LED
led1 = GPIOPin.new(:USR0, :OUT)
led2 = GPIOPin.new(:USR1, :OUT)
led3 = GPIOPin.new(:USR2, :OUT)
led4 = GPIOPin.new(:USR3, :OUT)

#RESET - Functie ce aduce led-urile la starea initiala
1.times do
 [led1,led2,led3,led4].each do |led|
  led.digital_write(:LOW)
 end
end
sleep 1

#0001 - Reprezentarea lui 1 in binar
1.times do
  [led4].each do |led|
     led.digital_write(:HIGH)
   end
  end
sleep 0.25

#RESET
1.times do
 [led1,led2,led3,led4].each do |led|
  led.digital_write(:LOW)
 end
end
sleep 3

#0010 - Reprezentarea lui 2 in binar.
1.times do
  [led3].each do |led|
  led.digital_write(:HIGH)
 end
end
sleep 3
```

Figure 15. Representation of numbers in binary using LEDs.

The LEDs start from the LOW state, this is done by initializing the reset function, after which at an interval of one second the LEDs represent a number in binary, the change of states between LOW and HIGH is done using the sleep() function, a function that puts the board in a "freeze" state between calling the RESET function and representing the next number.

6.2 Application 2: Programming a PIR sensor

To make the application we used: a PIR sensor, an LED, a 220 Ω resistor, a breadboard and connecting wires. The LED can be connected to any digital pin on the board, while the PIR sensor must be connected to specific pins: The VCC is connected to pin 7 or pin 8 on the P9 header, these being the pins that provides an operating voltage of 5V, the sensor OUT connects to pin 19 on the P8 header, the GND connects to the ground portion of the breadboard (Figure 16).

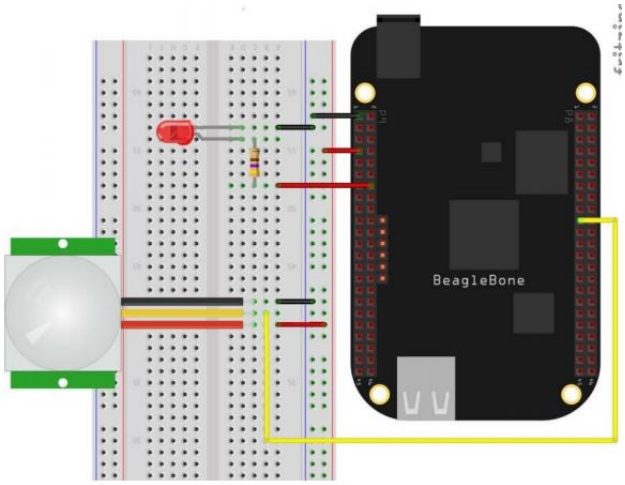

Figure 16. Schematic of programming a PIR sensor using Fritzing.

As always, the code is initialized by using the BoneScript library and defining variables. Setting pin 14 as an LED output and setting the PIR sensor as an Input. By using the digitalWrite() function, the initial state of the LED is set to LOW, after which the setInterval() function is used to execute the checkMotion() loop every 2 seconds to check for movement in front of the sensor. The checkMotion() function is initialized by reading the digital value of the PIR sensor and calling the activated() function that works by reading the Input value of the sensor as HIGH (Figure 17). When there is movement in front of the sensor, the LED lights up (Figure 18). In the console output we can also see the message related to motion detection (Figure 19).

```
var b = require('bonescript');
var led = "P9_14";
var pir = "P8_19";
b.pinMode(led, b.OUTPUT);
b.pinMode(pir, b.INPUT);
b.digitalWrite(led, b.LOW);
setInterval (checkMotion, 1000);
function checkMotion() {
 b.digitalRead(pir, activate);
 function activate(x){
 if (x.value == b.HIGH) {
 b.digitalWrite(led, b.HIGH);
 console.log("Miscare detectata");
 }
 else {
 b.digitalWrite(led, b.LOW);
 console.log("Niciun obiect in proximitate");
 }
 }
}
```

Figure 17. Javascript code for application 2

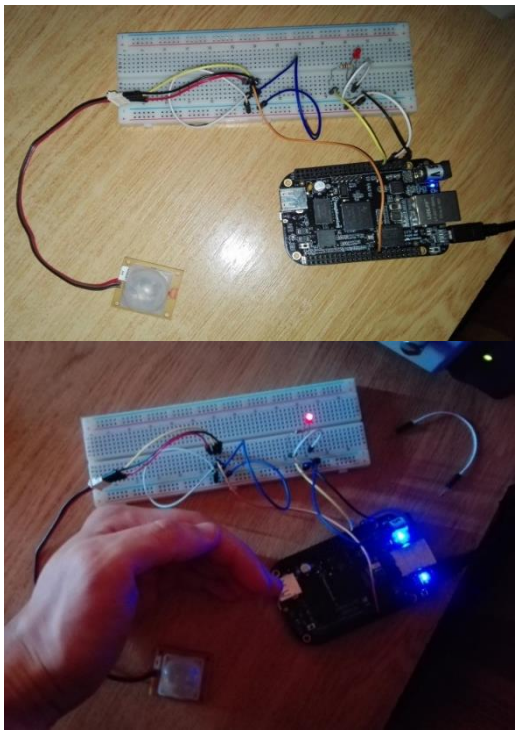

Figure 18. Programming the PIR sensor using BeagleBone.

The result is given in Figure 19.

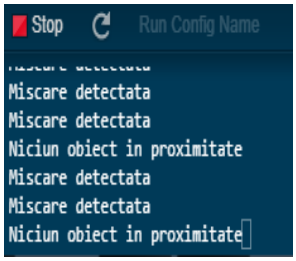

Figure 19. Console output

6.3 Application 3: Programming an ultrasonic sensor.

For this application we used: a breadboard, connecting wires, two resistors, one of 10 kΩ and the other of 20 kΩ and the HC-SR04 sensor (Figure 20).

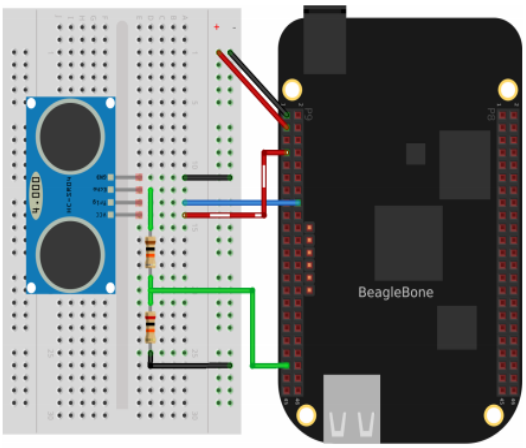

Figure 20. Ultrasonic sensor programming diagram using Fritzing.

The HC-SR04 sensor works by releasing a 40kHz ultrasonic pulse when a signal lasting more than 10 microseconds is received by the "trigger" input. The signal obtained at the "echo" type output produces another pulse whose wavelength is proportional to the distance measured from the object to the sensor.

The HC-SR04 is a 5V device that has a 5V output sent through its "echo" port, which is too high a voltage for a GPIO pin on the BeagleBone board that operates at a maximum voltage of 3.3V. That is why a resistive divider is used that will scale the 5V current into a 3.3V one. Resistors with a ratio of 2: 1 (20kΩ and 10kΩ) are used. If the resistors were higher, the GPIO pin would consume too much current and the voltage would drop considerably, instead the use of too small resistors would cause a considerably higher current to pass through them, which leads to a waste of board power.

The code we used to activate the sensor is written in JavaScript and is run remotely using the BoneScript library and the Cloud9 IDE service. The operation of the code consists in setting the trigger to logic 1 and using the setInterval () and ping () functions to switch to logic 0 and start the timer that will make the measurement. The attachInterrupt () and pingEnd () functions are used to wait for the pulse received by the echo to return to 0. As soon as this happens, call the pingEnd function and calculate the time from when the pulse started until the trigger returns to logic 1 again. The process resumes indefinitely (Figure 21) and the result is given in Figure 22.

```
var b = require('bonescript');
var trigger = 'P9_16', // Pin pentru activarea impulsului ultrason
 echo = 'P9_41', //  Pin pentru a masura impulsul relativ la dista
 ms = 250; // Perioada in care se desfasoara impulsul
var startTime, pulseTime;
b.pinMode(echo, b.INPUT, 7, 'pulldown', 'fast', doAttach);
function doAttach(x) {
 if(x.err) {
console.log('x.err = ' + x.err);
 return;
 }
 //Apelarea pingEnd cand impulsul se sfarseste
 b.attachInterrupt(echo, true, b.FALLING, pingEnd);
}
b.pinMode(trigger, b.OUTPUT);
b.digitalWrite(trigger, 1); // Unit triggers on a falling edge.
 // Setarea trigger-ului ca HIGH pentru a putea fi trecut in stare
// Setarea trigger-ului ca LOW la un interval predefinit.
setInterval(ping, ms);
// Setarea trigger-ului ca LOW si inceperea masurarii
function ping() {
 // console.log('ping');
 b.digitalWrite(trigger, 0);
 startTime = process.hrtime();
}
// Calcularea timpului total si repornirea trigger-ului
function pingEnd(x) {
 if(x.attached) {
 console.log("Pornirea impulsului sonic");
 return;
 }
 if(startTime) {
 pulseTime = process.hrtime(startTime);
 b.digitalWrite(trigger, 1);
 console.log('Indiciu de proximitate = ' + (pulseTime[1]/1000000-0
 }
}

```

Figure 21. The code we used to activate the sensor in JavaScript.

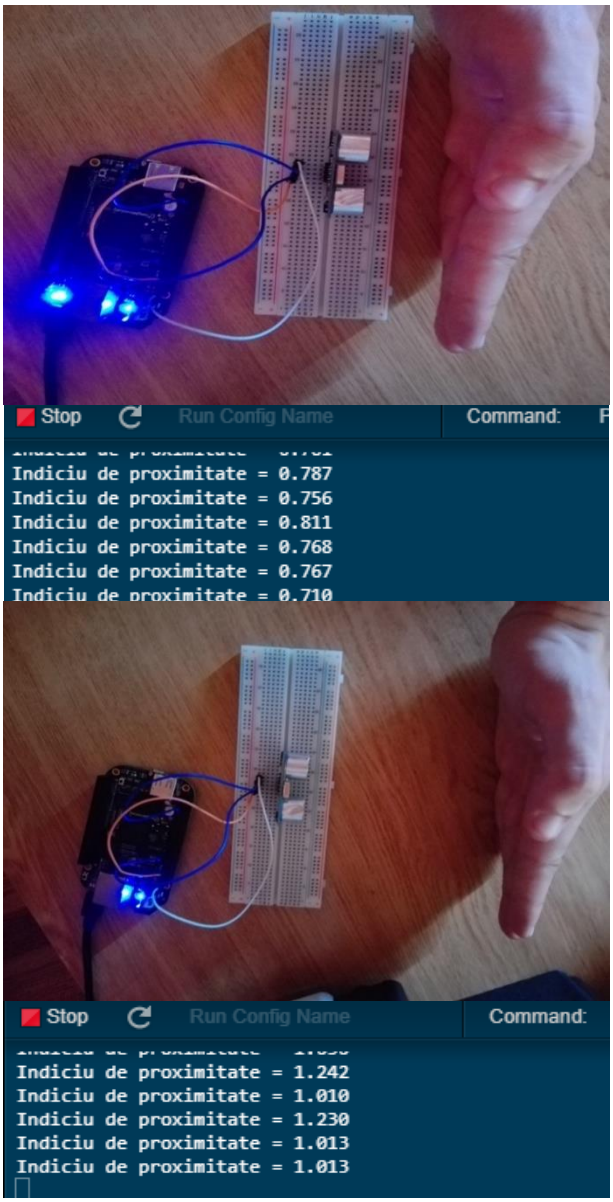

Figure 22. Programming the ultrasonic sensor using BeagleBone.

7. RESULTS AND CONCLUSIONS

The BeagleBone platform is the development board that runs exclusively on the Linux operating system, making the application development process difficult and cumbersome. Although it is presented as a "plug & play" device by the manufacturer, the platform is far from that in terms of the fact that you need additional software to use the platform in conditions that offer compatibility. The first thing you need to do when you start using the platform is to change the operating system with another Linux distribution which be a clear inconvenience because the support is low making it difficult to develop more complex applications.

The idea of the work was to create a complex alarm system using sensors with which we later developed applications that can be accessed through the cloud.

The aim of this paper was to program a series of electronic devices using the BeagleBone black platform.

The main problem reached in this project was the implementation of the code that would control the devices used, because BeagleBone Black is a new platform based on Linux and developing applications using Linux commands is a difficult process and prone to compatibility issues, the platform lacking support very extensive in terms of interfacing with other devices or platforms of this type.

The improvement in the application development process came when we discovered the Cloud9 IDE online programmable environment, which eliminates the problem of using Linux commands by introducing Socket IO and Bonescript libraries that allow the use of high-level languages such as Javascript, Ruby or Python for application programming.

8. REFERENCES

[1] F. Ion, *Electronică digitală – memorii ROM, RAM, circuite CPLD, FPGA*, Editura Bibliotheca, Târgoviște, 2009.

[2] J.J.R. Andina, E. de la Torre, M.D. Valdes, *FPGAs: Fundamentals, Advanced Features, and Applications in Industrial Electronics*, CRC Press, 2020;

[3] M.A. Yoder, J. Kridner, *BeagkeBone cookbook: software and hardware problems and solutions*, O'Reilly Media, 2015;

[4] G. Coley, *BeagleBone black system reference manual*, BeagleBoard.org, 2014;

[5] Chethana Gosal S, Yogesh, Suresh Kumar, Vinaykumar K, *Beagle bone black based security robot*, International Jornal of Engineering Trends and Applications (IJETA), volume 3, issue 3, may-jun, pp.24-27, 2016;

[6] Anjita M. Anand, Balu Raveendran, Shoukat Cherukat, Shiyas Shahab, *Using PRUSS for real-time applications on Beaglebone black*, WCI'15: Proceedings of the Third International Symposium on Women in Computing and Informatics, august 10-13, pp.377-382, 2015;

[7] Nannan He, Han-Way Huang, Brian David Woltman, *The use of Beaglebone black board in engineering design and development*, Proceedings, The 2014 ASEE North Midwest Section Conference, october 16-17, pp.1-8, 2014;

[8] *Product user's manual – HC-SR04 ultrasonic sensor*, Cytron Technologies Sdn. Bhd., 2013;

[9] *PIR motion sensor*, Adafruit Learning System, 2020;

[10] S. Monk, *Programming the BeagleBone black: getting started with JavaScript and BoneScript*, 2014;

[11] Jayneil Dalal, *Beaglebone – hands on tutorial*, Embedded Linux Conference, 2013.

[12] D. Molloy, *Exploring BeagleBone tools and techniques for building with embedded Linux*, John Wiley&Sons, 2015.

[13] B. Cherny, *Programming TypeScript: making your JavaScript applications scale*, O'Reilly Media, 2019;

[14] P. Cobbaut, *Linux fundamentals*, CEST, 2015, http://linux-training.be/linuxfun.pdf.

[15] *AWS Cloud9: user guide*, Amazon Web Services, 2020;