VoIP GATEWAY DESIGN EVALUATION

James Way & Venetia Furtado

Project 2 Final Report
ECEN 5803 Mastering Embedded Systems Architecture
University of Colorado, Boulder
December 12, 2024

Contents

1 Executive Summary

The project evaluated the Broadcom BCM2837 microcontroller for use in a VoIP Gateway Design Evaluation, assessing its performance and suitability against the product's requirements. The evaluation included testing the processor's capability of running both Windows 10 IoT and Linux operating systems. Upon attempting to install Windows 10 IoT, it was quickly discovered that much of the support for the platform has been discontinued, including the primary tools required for its installation. While alternative installation methods were identified, the platform's obsolescence necessitated pivoting to a solution with active support. Linux emerged as the superior option for this project due to its active development community, extensive support for open-source tools, and compatibility with Asterisk.

A direct requirement of the product was the evaluation of the system's ability to handle G.711 encoding and decoding. This functionality was successfully implemented using GCC on Raspbian Linux, and test files confirmed the outputs were accurate. Additionally, a diagnostic script was developed to display critical system information, such as running processes, kernel version, and kernel dumps. The script was configured to run on boot, demonstrating its applicability to real-world use cases. Further development included building a PBX system with Asterisk, where SIP phones and a laptop were configured to simulate VoIP communication. Tests included voicemail functionality and a phone-to-phone call, successfully demonstrating the system's capabilities. A Python-based web server was also implemented, providing a home webpage that displayed the current time and access count for the Raspberry Pi Model 3B.

The evaluation confirmed that the Broadcom BCM2837 processor is suitable for the project's requirements, offering adequate performance. However, it is not readily available on the market today. As an alternative, the Texas Instruments AM625 series processors or Texas Instruments AM3352 processor are recommended for their superior performance, advanced features, extensive connectivity options, and TIs commitment to low power applications. The price is below the $15 limit at a quantity of 25 or more.

2 Problem Statement and Objectives

Patton requires hardware development services in support of a new product. The company specifically requires embedded system design evaluation for the proposed e911 IP PBX VoIP Gateway. Implementation of pre-production prototypes is a desirable follow-on contract. The vendor is required to evaluate and test the proposed embedded system platform.

The objective is to deliver a solution that surpasses the performance and capabilities of a competitive product currently using the Windows 7 Embedded platform. This involves rigorous testing of hardware and software to identify a robust, scalable, and efficient embedded platform for the VoIP Gateway.

3 Approach and Methodology for Evaluation

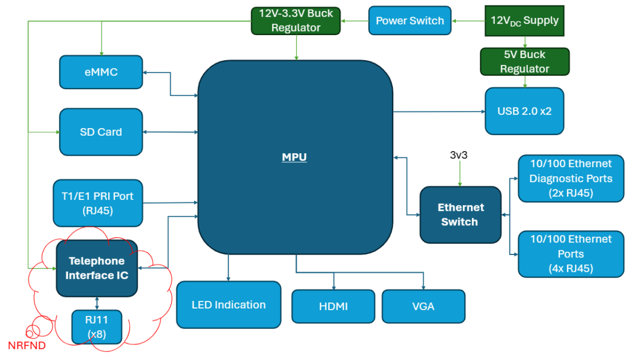


Figure 1: Block diagram of the proposed VoIP Gateway Design

Designers utilized a Raspberry Pi 3B model, powered by the Broadcom BCM2837 processor, as the foundation for the project. Core components were developed and integrated within individual modules.

Evaluation began by testing the system's performance on a Windows 10 IoT operating system, followed by a comparison with the Debian-based GNU/Linux operating system, Raspbian. A series of tests were conducted across all modules. The system block diagram outlines the inputs, processor, and outputs of the device.

To implement a G.711 coder/decoder, µ-law compression was employed. Additionally, Asterisk was used to develop the PBX system, leveraging the SIP protocol to verify interoperability with hardware and Linux.

4 Module Test Results

***NOTE - Detailed test results and answers to the module questions have been included in the Appendix-Module Test Results section.**

4.1 Module 1: Install Visual Studio

Visual Studio provides a comfortable working environment to develop and run code on a Windows 10 IoT device straight from your working PC. The team installed Visual Studio to become familiar with the environment and conducted training to prepare for development on Windows 10 IoT.

4.2 Module 2: Boot Windows 10 IoT and create a G.711 Codec

In this module, the Raspberry Pi was booted with Windows 10 IoT, following the installation and setup instructions provided in the project guide. After successfully installing the operating system, debug messages output via the serial port during the boot process were logged. The memory footprint of the Windows 10 IoT image was compared to that of Raspbian Linux.

A key difference observed between the two platforms was their file systems: Linux uses the ext4 file system, whereas Windows 10 IoT uses the NTFS file system.

Subsequently, the G.711 coder/decoder code, implemented with µ-law compression, was compiled using GCC on Raspbian Linux. µ-law was chosen for its ability to provide a larger dynamic range for signals. The code successfully encoded and decoded the provided audio files.

4.3 Module 3: Scripting with Linux

Module 3 focused on creating and executing a script in Linux to display useful information about the Linux kernel. A bash script was developed to perform the following tasks:

- Display the current user and terminal.
- List the currently running processes.
- Show the current date and time.
- Display the kernel version.
- Provide a kernel dump.

To ensure the script ran automatically at boot, the default init system on the Raspberry Pi was utilized. A service file was created and configured to execute the script during the boot process.

4.4 Module 4: Build your own PBX with Asterisk

The SIP protocol was used to implement a PBX system. Asterisk, an open-source telephony software was used in this module. Asterisk acts as a PBX (Private Branch Exchange) and connects and manages calls for VoIP systems. For making the calls a phone and a laptop were used as SIP phones with the help of Zoiper. The system was successfully set-up, and both the goals for this module were achieved:

- Make a call to extension 100 which played back the voicemail “Hello World” and then hung up.
- Make a phone-to-phone call.

4.5 Module 5: Build a IoT Application

A web server was successfully configured for the Raspberry Pi Model 3B, hosting a homepage that displayed the current time and the number of page accesses. The implementation was achieved using Python’s HTTP server package on the Raspbian Linux platform.

5 List of Project Deliverables

- **Detailed Technical Report -** This document.
- **Executive Summary Report -** See Section 1 of this document.
- **Software Design Files -** Including original code, test data and answers to module questions.
- **Hardware Design Files -** Including BOM for key parts, pictures of hardware test circuits, and test data.
- **Informal Presentation -** To be completed with a TA as availability allows.

6 Recommendations

6.1 OS selection

It is recommended to abandon Windows 10 IoT in favor of Linux as the foundation for the VoIP Gateway’s development. Leveraging Asterisk on Linux provides scalability, reliability, and long-term support.

6.2 Processor performance

The Broadcom BCM2837 microcontroller was deemed suitable for the prototype application. For production-scale implementation, alternatives such as the **AM6254** or **TI AM3352** should be considered due to their superior performance and cost-effectiveness when purchased in quantity, especially given the Raspberry Pi 3B's status as an older model.

6.3 BRI ports

The integration of FXS and FXO interfaces into the VoIP Gateway design was carefully evaluated. This was deemed unadvisable due to the high cost of specialized components like FXS and FXO transceivers, as well as the increased complexity of hardware and software development. Additionally, the industry trend toward IP-based telephony reduces the reliance on analog interfaces. Therefore, we recommend focusing on Ethernet-based IP communications, aligning with modern standards and keeping the design within budget.

Table 1: Features of the recommended alternate processors.

MPU	**AM3352**	**AM6254**
Cost	\$9.09	\$14.16
Core	Cortex®-A8 processor at 1 GHz	Cortex®-A53 @ 1.4GHz - 4 Core, 64-Bit
Linux Support	Yes	Yes
Ethernet	10/100/1000Mbps (2)	10/100/1000Mbps (2)
USB	USB 2.0 (2)	USB 2.0 (2)
DMIPS	2,000	4,065

7 Appendix - References

1. ECEN5803 Project 2 Guide
2. Raspberry Pi 3B Datasheet
3. ECEN5803 Lecture Notes
4. AM3352 Datasheet
5. AM6254 Datasheet

8 Appendix - Project Team Staffing

James Way
University of Colorado, Boulder
Email: James.Way@colorado.edu

Venetia Furtado
University of Colorado, Boulder
Email: Venetia.Furtado@colorado.edu

9 Appendix - Bill of Materials

This Bill of Materials only inludes major components. It excludes resistors, and other components that may cost less than $0.10 a piece.

Part Number	Function	Count	Cost ea. (100	Cost/Unit	Notes
AM6254ASGGHAALW	MPU Option	1	$ 14.16	$ 14.16	Not included in total
AM3352BZCZ100	MPU Option	1	$ 9.09	$ 9.09	Included in total
KSZ9897RTXC	Ethernet Switch IC	1	$ 10.33	$ 10.33	To Support up to 7 ethernet ports
	BRI ISDN Controller/Transciever			$ -	NRFND - No viable options identified
THGBMUG6C1LBAIL	eMMC	1	$ 7.21	$ 7.21	Optional for a more robust OS storage [excluded from total]
MEM2061-01-188-00-A	Flash Card Holder	1	$ 0.74	$ 0.74	
Sandisk Ultra 32GB	Micro SD Card	1	$ 7.69	$ 7.69	
54602-908LF	Ethernet Connectors (RJ45)	7	$ 0.37	$ 2.59	
61400416021	USB A Connectors (2.0 rated)	2	$ 0.65	$ 1.30	
E5566-Q0LK22-L	BRI Telephone Connectors (RJ11)	8	$ 0.31	$ 2.48	NRFND - excluded from total
A-DF 09 A/KG-T2S	VGA Connector	1	$ 0.56	$ 0.56	
SS-53000-001	HDMI Connector	1	$ 0.57	$ 0.57	
151031VS06000	LED	1	$ 0.12	$ 0.12	
SW-R3-1A-A-1-2	SWITCH ROCKER SPST 6A 125V	1	$ 0.35	$ 0.35	
694108301002	Barrel Power Jack	1	$ 0.72	$ 0.72	
TPS5403DR	Voltage regulator 12V-3.3V, 1.7A	1	$ 1.00	$ 1.00	
UA78M05CDCYG3	Voltage regulator 12V-5V 0.5A (USB)	1	$ 0.43	$ 0.43	
WR9HE1000LLP-F(R6B)	AC/DC Wall Mount Adapter 12V 12W	1	$ 5.62	$ 5.62	
			Total:	$ 41.11	

10 Appendix - Module Test Results

Module 1: Install Visual Studio

James Way & Venetia Furtado

Date: 11/03/2024

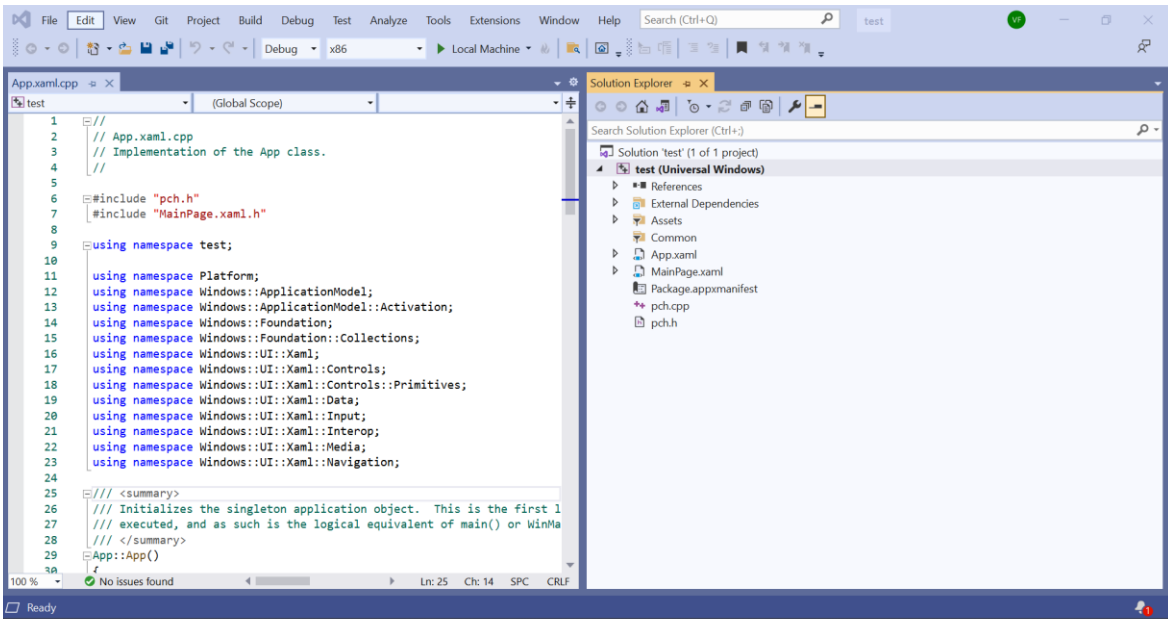


Figure 1: Visual Studio Installation

Module 2: Boot Windows 10 IoT, create a G.711 codec

James Way & Venetia Furtado

Date: 11/10/2024

1. Flash Windows 10 IoT

- Download Raspberry Pi 2 & 3 build from [Downloads - Windows IoT | Microsoft Learn]
- Mount the .iso file and run the application. It will install an .ffu file.
- Using Rufus or similar program, select the SD card to be flashed and the .ffu file.
- Select "START".

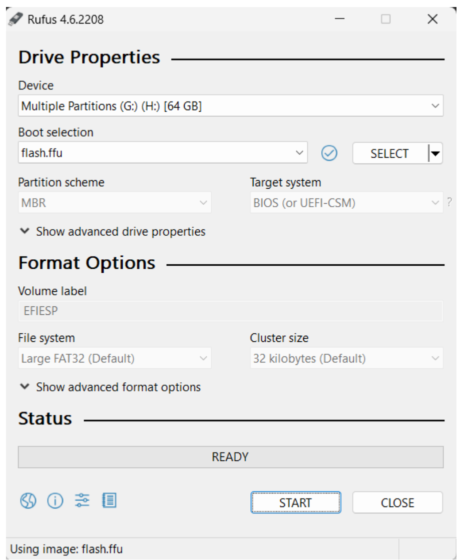


Figure 1: Rufus GUI - flashing Windows 10 IoT

2. Upon booting up, the serial port should be sending out debug messages. Open a terminal window to capture them. What do you see?

The debug messages by default spit out the ID of the serial port being used and the time the build was created. Some commands can be run to get additional messages on boot. Figure 2 shows the default values.
Parameters:
BAUD rate: 921600, No Parity, 8bit word size.

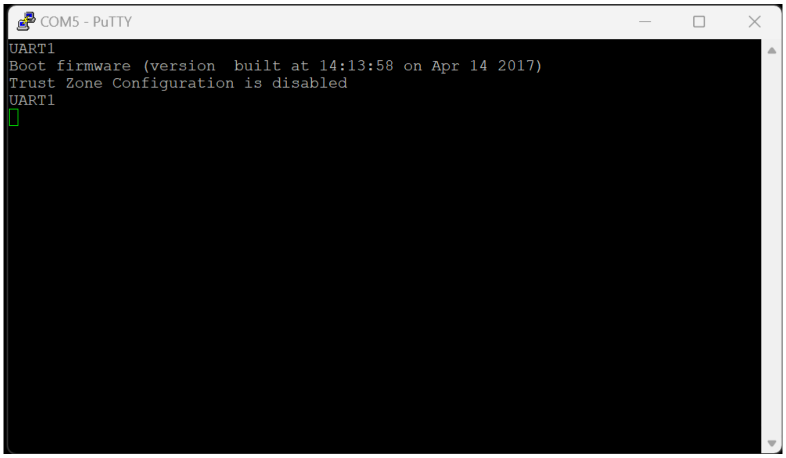


Figure 2: Serial output over UART1 while Windows 10 IoT is booting.

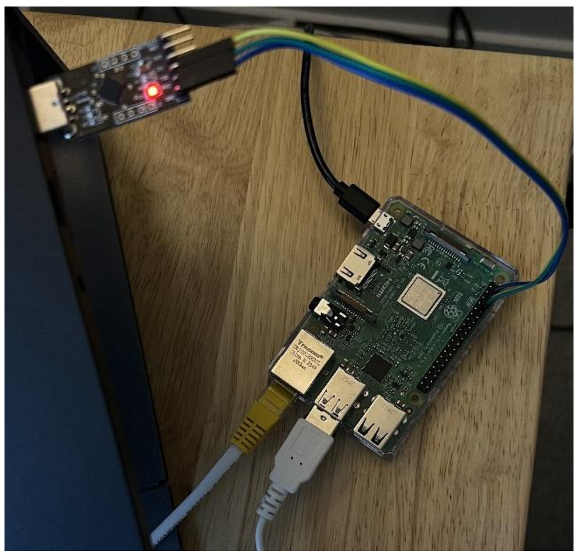


Figure 3: Serial port connection

3. How much memory is used by the code? (What is the image size?)

Figure 4: Windows IoT image size

Windows IoT Image size = (1498411008-598102016) + (30337908736-29778329600) = 1459888128 = **1.45 GB**

4. Capture a screen shot of the terminal window.

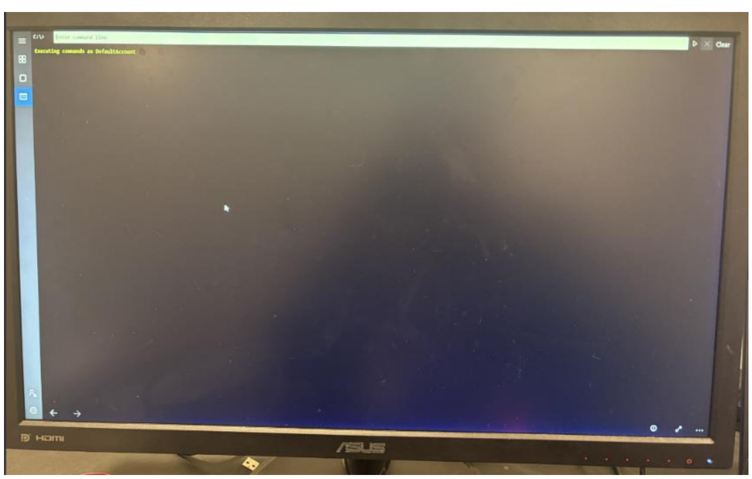


Figure 5: Windows 10 IoT Cmd line screen view.

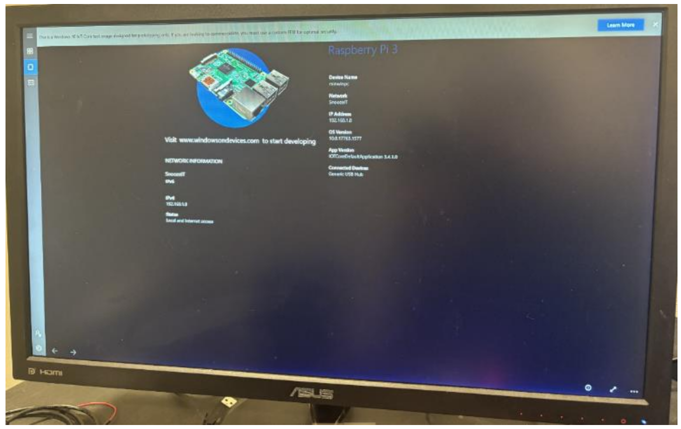


Figure 6: Windows 10 IoT Desktop view.

5. Connect the HDMI output to a monitor to see the GUI. Reboot the system – what do you see?

On startup the standard Windows 10 loading screen is visible as you would expect from any device running Windows (see image below).

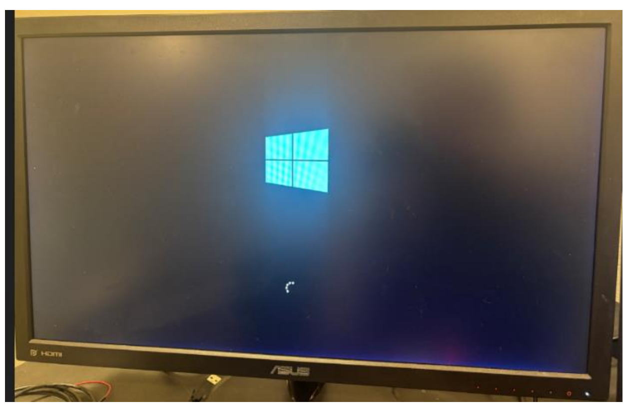


Figure 7: Windows 10 IoT loading screen on boot.

6. Write C code for a G.711 coder/decoder. Use this decoder to decode a file given to you by your instructor.

C code included in Appendix A.

The G.711 coder/decoder code was compiled using GCC on Raspbian Linux. The encoding and decoding algorithms were referenced from online resources listed out in the References section. For the RIFF header of the WAVE file with a *SampleRate* of 8000 the *ByteRate*, *BlockAlign* and *BitsPerSample* were set as follows:

ByteRate = SampleRate*NumChannels*BitsPerSample/8

BlockAlign = NumChannels*BitsPerSample/8

BitsPerSample = 8 (8 bits for encoding), 16 (16 bits for decoding)

Since the size of the FACT chunk of a WAVE file cannot be pre-determined, and the encoded and decoded files seemed to work without it, the FACT chunk was excluded from the RIFF header.

7. Record your observations. How is the behavior on Windows 10 IoT different from Linux?

Raspbian Linux uses the *ext4* file system.

```
pi@raspberrypi:~ $ df -T
Filesystem     Type     1K-blocks    Used Available Use% Mounted on
/dev/root      ext4      27784728 6523140  19827144  25% /
devtmpfs       devtmpfs    469544       0    469544   0% /dev
tmpfs          tmpfs       474152       0    474152   0% /dev/shm
tmpfs          tmpfs       474152    6420    467732   2% /run
tmpfs          tmpfs         5120       4      5116   1% /run/lock
tmpfs          tmpfs       474152       0    474152   0% /sys/fs/cgroup
/dev/mmcblk0p6 vfat        258094   53033    205062  21% /boot
tmpfs          tmpfs        94828       0     94828   0% /run/user/1000
```

Figure 8: Snapshot of the Linux root filesystem

Windows 10 IoT uses a file structure identical to other Windows operating systems: NTFS (New Technology File System). While the explorer GUI may feel different from most other windows devices, the cmd prompt will feel exactly the same and accepts most of the same commands.

So far, Windows 10 IoT has similar boot times, size and functionality. However, with Windows 10 IoT seemingly abandoned, finding resources and support for the more widely used Raspbian is much easier.

Appendix A - code for a G.711 coder/decoder

```
/**
 * @file main.c
 * @author James Way | Venetia Furtado
 * @brief ECEN 5803 Mastering Embedded System Architecture
 * @brief University of Colorado, Boulder
 * @brief Project 2 Module 2
 * @brief The file contains the implementation of the mu-law algorithm
for the
 * G.711 coder/decoder. The functions MuLaw_Encode() and
MuLaw_Decode() were
 * referenced from an online resources listed below in the References
section.
 * @version 0.1
 * @date 2024-11-09
 * @copyright Copyright (c) 2024
 *
 * References:
 * https://dystopiancode.blogspot.com/2012/02/pcm-law-and-u-law-
companding-algorithms.html
 *
https://www.cs.columbia.edu/~hgs/research/projects/NetworkAudioLibrary
/nal_spring/src/Codecs/g711.cpp
 * http://soundfile.sapp.org/doc/WaveFormat/
 * https://www.recordingblogs.com/wiki/format-chunk-of-a-wave-file
 *
 */
#include <stdio.h>
#include <stdint.h>

#define PCM_HEADER_SIZE 44

//RIFF header for ITU G.711 u-law
uint8_t encodeHeader[] = {
   'R', 'I', 'F', 'F', // ChunkID
   0x00, 0x00, 0x00, 0x00, // ChunkSize
   'W', 'A', 'V', 'E', // Format
   'f', 'm', 't', 0x20, // Subchunk1ID
```

```
    0x10, 0x00, 0x00, 0x00, // Subchunk1Size
    0x07, 0x00, // Audio Format = ITU G.711 u-law
    0x01, 0x00, // NumChannels
    0x80, 0x3E, 0x00, 0x00, // Sample Rate 8000
    0x80, 0x3E, 0x00, 0x00, // Byte Rate
    0x01, 0x00, // Block Align
    0x08, 0x00, // BitsPerSample
    'd', 'a', 't', 'a', // SubChunk2ID
    0x00, 0x00, 0x00, 0x00 // SubChunk2Size
};

//RIFF header for PCM
uint8_t decodeHeader[] = {
    'R', 'I', 'F', 'F', // ChunkID
    0x00, 0x00, 0x00, 0x00, // ChunkSize
    'W', 'A', 'V', 'E', // Format
    'f', 'm', 't', 0x20, // Subchunk1ID
    0x10, 0x00, 0x00, 0x00, // Subchunk1Size
    0x01, 0x00, // Audio Format = PCM
    0x01, 0x00, // NumChannels
    0x40, 0x1F, 0x00, 0x00, // Sample Rate 8000
    0x80, 0x3E, 0x00, 0x00, // Byte Rate 16000
    0x02, 0x00, // Block Align
    0x10, 0x00, // BitsPerSample
    'd', 'a', 't', 'a', // SubChunk2ID
    0x00, 0x00, 0x00, 0x00 // SubChunk2Size
};

/**
 * @brief µ-Law Compression (Encoding) Algorithm
 * Reference: https://dystopiancode.blogspot.com/2012/02/pcm-law-and-
u-law-companding-algorithms.html
 * @param number
 * @return int8_t
 */
int8_t MuLaw_Encode(int16_t number)
{
    const uint16_t MULAW_MAX = 0x1FFF;
    const uint16_t MULAW_BIAS = 33;
```

```
   uint16_t mask = 0x1000;
   uint8_t sign = 0;
   uint8_t position = 12;
   uint8_t lsb = 0;
   if (number < 0)
   {
      number = -number;
      sign = 0x80;
   }
   number += MULAW_BIAS;
   if (number > MULAW_MAX)
   {
      number = MULAW_MAX;
   }
   for (; ((number & mask) != mask && position >= 5); mask >>= 1,
position--)
      ;
   lsb = (number >> (position - 4)) & 0x0f;
   return (~(sign | ((position - 5) << 4) | lsb));
}

/**
 * @brief μ-Law Expanding (Decoding) Algorithm
 * Reference: https://dystopiancode.blogspot.com/2012/02/pcm-law-and-
u-law-companding-algorithms.html
 * @param number
 * @return int16_t
 */
int16_t MuLaw_Decode(int8_t number)
{
   const uint16_t MULAW_BIAS = 33;
   uint8_t sign = 0, position = 0;
   int16_t decoded = 0;
   number = ~number;
   if (number & 0x80)
   {
      number &= ~(1 << 7);
      sign = -1;
   }
```

```
   position = ((number & 0xF0) >> 4) + 5;
   decoded = ((1 << position) | ((number & 0x0F) << (position - 4)) |
(1 << (position - 5))) - MULAW_BIAS;
   return (sign == 0) ? (decoded) : (-(decoded));
}

/**
 * @brief Function to open a file with required permissions (read "rb"
or write "w")
 * @param filename
 * @param permissions
 * @return FILE*
 */
FILE* openFile(const char* filename, const char* permissions)
{
   FILE *file;

   file = fopen(filename, permissions);
   if (file == NULL)
   {
      perror("Error opening file");
      return NULL;
   }
   return file;
}

/**
 * @brief Function to encode a file from 16-bit PCM format to 8-bit
ITU G.711
 *
 * @param filename
 * @return int
 */
int encodeFile(const char* filename)
{
   int16_t inputData;                        // Stores each byte read
from the file
   //Open the file in binary read mode ("rb")
   FILE *inputFile = openFile(filename, "rb");
```

```
    if(inputFile == NULL)
    {
        return 0;
    }

    //Open the ouput file in write mode ("w")
    FILE *outputFile = openFile("encode.wav","w");
    if(outputFile == NULL)
    {
        return 0;
    }

    fseek(inputFile, PCM_HEADER_SIZE, SEEK_SET);

    fwrite(&encodeHeader, sizeof(encodeHeader), 1, outputFile);

    uint32_t count = 0;
    // Read 16-bit at a time until end of file (EOF)
    while (fread(&inputData, 2, 1, inputFile) == 1)
    {
        int8_t outputData = MuLaw_Encode(inputData);
        fwrite(&outputData, 1, 1, outputFile);
        count++;
    }

    // Move to 40 bytes from the start of the file(Subchunk2Size)
    fseek(outputFile, 40, SEEK_SET);
    fwrite(&count, 4, 1,outputFile);

    // Move to 4 bytes from the start of the file(ChunkSize)
    fseek(outputFile, 4, SEEK_SET);
    count = count + 36; //36 + Subchunk2Size
    fwrite(&count, 4, 1,outputFile);

    // Close the file
    fclose(inputFile);
    fclose(outputFile);
    return 0;
}
```

```
/**
 * @brief Function to encode a file from 8-bit ITU G.711 to 16-bit PCM
format
 *
 * @param filename
 * @return int
 */
int decodeFile(const char* filename)
{
   int8_t inputData;                        // Stores each byte read
from the file
   //Open the file in binary read mode ("rb")
   FILE *inputFile = openFile(filename, "rb");
   if(inputFile == NULL)
   {
      return 0;
   }

   //Open the ouput file in write mode ("w")
   FILE *outputFile = openFile("decode.wav","w");
   if(outputFile == NULL)
   {
      return 0;
   }

   fseek(inputFile, PCM_HEADER_SIZE + 12, SEEK_SET);

   fwrite(&decodeHeader, sizeof(decodeHeader), 1, outputFile);

   uint32_t count = 0;
   // Read 8-bit at a time until end of file (EOF)
   while (fread(&inputData, 1, 1, inputFile) == 1)
   {
      int16_t outputData = MuLaw_Decode(inputData);
      fwrite(&outputData, 2, 1, outputFile);
      count += 2;
   }
```

```
    // Move to 40 bytes from the start of the file(Subchunk2Size)
    fseek(outputFile, 40, SEEK_SET);
    fwrite(&count, 4, 1,outputFile);

    // Move to 4 bytes from the start of the file(ChunkSize)
    fseek(outputFile, 4, SEEK_SET);
    count = count + 36; //36 + Subchunk2Size
    fwrite(&count, 4, 1,outputFile);

    // Close the file
    fclose(inputFile);
    fclose(outputFile);
    return 0;
}

/**
 * @brief Functions prints data of file-used for debugging
 *
 * @param filename
 */
void printData(const char *filename)
{
    //Open the  file
    FILE *file = openFile(filename, "rb");
    if (file == NULL)
    {
        return;
    }

    // Read 8-bit at a time
    uint8_t byte;
    int count =0;
    while (fread(&byte, 1, 1, file) == 1)
    {
        printf("%02X ", byte); // Print each byte in hexadecimal format
        count++;
        if (count == 100)
        {
            break;
```

```
        }
    }
}

/**
 * @brief Main function consists of calls to enocde and decode the
files.
 *
 * @return int
 */
int main()
{
    encodeFile("1_A_eng_m1.wav");
    decodeFile("3_1449183537-A_eng_m1.wav");
    //printData("3_1449183537-A_eng_m1.wav");
    return 0;
}
```

References:

[1] https://dystopiancode.blogspot.com/2012/02/pcm-law-and-u-law-companding-algorithms.html

[2]https://www.cs.columbia.edu/~hgs/research/projects/NetworkAudioLibrary/nal_spring/src/Codecs/g711.cpp

[3] http://soundfile.sapp.org/doc/WaveFormat/

[4] https://www.recordingblogs.com/wiki/fact-chunk-of-a-wave-file

[5] https://en.wikipedia.org/wiki/Main_Page

Module 3: Scripting with Linux

James Way & Venetia Furtado

Date: 11/17/2024

Aim: Creation and running an executable script in Linux to print useful information about the Linux Kernel.

1. Create an executable script

A bash script was written to perform the following functions:

i. Display the current user and terminal.
ii. Display the current running processes.
iii. Display the current data and time, kernel version.
iv. Display the kernel dump.

```
#!/bin/bash

echo "---- System Information Script ----"

# Display the current user and terminal
echo "Current User: $(whoami)"
echo "Terminal: $TERM"
echo "Shell: $SHELL"

# Display running processes
echo -e "\nCurrent Running Processes:"
ps aux

# Display current date, time, and kernel version
echo -e "\nCurrent Date and Time:"
date
echo -e "\nKernel Version:"
uname -r

# Display kernel dump
echo -e "\nKernel Dump (dmesg output):"
dmesg | tail -n 20   # Limiting output to the last 20 lines for brevity

echo -e "\n---- End of System Information ----"
```

Figure 1: Snapshot of the bash script terminalInfo.sh

The command used to make the bash script executable was: chmod +x terminalInfo.sh

The outputs have been shown in Figure 2 below.

```
pi@raspberrypi:~/PBX/Module3 $ ./terminalInfo.sh
---- System Information Script ----
Current User: pi
Terminal: xterm-256color
Shell: /bin/bash

Current Running Processes:
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.4  0.8  33720  7972 ?        Ss   19:21   0:03 /sbin/init splash
root         2  0.0  0.0      0     0 ?        S    19:21   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        I<   19:21   0:00 [rcu_gp]
root         4  0.0  0.0      0     0 ?        I<   19:21   0:00 [rcu_par_gp]
root         7  0.0  0.0      0     0 ?        I    19:21   0:00 [kworker/u8:0-events_unbound]
root         8  0.0  0.0      0     0 ?        I<   19:21   0:00 [mm_percpu_wq]
root         9  0.0  0.0      0     0 ?        S    19:21   0:00 [ksoftirqd/0]
root        10  0.0  0.0      0     0 ?        I    19:21   0:00 [rcu_sched]
root        11  0.0  0.0      0     0 ?        I    19:21   0:00 [rcu_bh]
root        12  0.0  0.0      0     0 ?        S    19:21   0:00 [migration/0]
root        13  0.0  0.0      0     0 ?        S    19:21   0:00 [cpuhp/0]
root        14  0.0  0.0      0     0 ?        S    19:21   0:00 [cpuhp/1]
root        15  0.0  0.0      0     0 ?        S    19:21   0:00 [migration/1]
root        16  0.0  0.0      0     0 ?        S    19:21   0:00 [ksoftirqd/1]
root        19  0.0  0.0      0     0 ?        S    19:21   0:00 [cpuhp/2]
root        20  0.0  0.0      0     0 ?        S    19:21   0:00 [migration/2]
root        21  0.0  0.0      0     0 ?        S    19:21   0:00 [ksoftirqd/2]
root        24  0.0  0.0      0     0 ?        S    19:21   0:00 [cpuhp/3]
```

```
Current Date and Time:
Fri 15 Nov 2024 07:34:51 PM MST

Kernel Version:
4.19.75-v7+
```

```
Kernel Dump (dmesg output):
[    8.854540] smsc95xx 1-1.1:1.0 eth0: hardware isn't capable of remote wakeup
[    8.854821] IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
[    9.721449] IPv6: ADDRCONF(NETDEV_CHANGE): wlan0: link becomes ready
[   12.517537] Bluetooth: Core ver 2.22
[   12.517659] NET: Registered protocol family 31
[   12.517667] Bluetooth: HCI device and connection manager initialized
[   12.517989] Bluetooth: HCI socket layer initialized
[   12.518003] Bluetooth: L2CAP socket layer initialized
[   12.518045] Bluetooth: SCO socket layer initialized
[   12.535341] Bluetooth: HCI UART driver ver 2.3
[   12.535351] Bluetooth: HCI UART protocol H4 registered
[   12.535420] Bluetooth: HCI UART protocol Three-wire (H5) registered
[   12.535549] Bluetooth: HCI UART protocol Broadcom registered
[   12.922376] Bluetooth: BNEP (Ethernet Emulation) ver 1.3
[   12.922383] Bluetooth: BNEP filters: protocol multicast
[   12.922399] Bluetooth: BNEP socket layer initialized
[   12.987307] Bluetooth: RFCOMM TTY layer initialized
[   12.987335] Bluetooth: RFCOMM socket layer initialized
[   12.987362] Bluetooth: RFCOMM ver 1.11
[   13.196860] fuse init (API version 7.27)
```

Figure 2: Output of the bash script.

2. **Make the bash script run in boot time**
   To make the script run in boot time, Raspberry Pi's system (the default init system) was used to create a service file that gets executed upon boot.
   The systemd service was created: *sudo nano /etc/systemd/system/terminalInfo.service*
   To enable the service to run at boot: *sudo systemctl enable terminalInfo.service*
   To start the service run: *sudo systemctl start terminalInfo.service*
   To check the status of the service: *sudo systemctl status terminalInfo.service* which shows "active".

To reboot system: *sudo reboot now*
To get the syslog run: *sudo cat /var/log/syslog*
The syslog shows that the script runs after the Raspberry Pi is reconnected with the network as specified in the service file.

A copy of the syslog file has been submitted for review.

```
[Unit]
Description=Run terminalInfo at boot
After=network.target

[Service]
ExecStart=/home/pi/PBX/Module3/terminalInfo.sh
Type=oneshot
RemainAfterExit=true

[Install]
WantedBy=multi-user.target
```

Figure 3: Systemd service file.

Appendix-References

[1] BASH Programming - Introduction HOW-TO

[2] https://linuxconfig.org/how-to-autostart-bash-script-on-startup-on-raspberry-pi

Module 4: Build your own PBX with Asterisk

James Way & Venetia Furtado

Date: 11/27/2024

1. How much memory is used by the code?
   29.4 MB
   Asterisk was installed using the Debian repositories for Raspbian Linux.

```
● asterisk.service - Asterisk PBX
   Loaded: loaded (/lib/systemd/system/asterisk.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2024-11-27 11:27:00 MST; 1h 37min ago
     Docs: man:asterisk(8)
 Main PID: 2678 (asterisk)
    Tasks: 69 (limit: 2200)
   Memory: 29.4M
   CGroup: /system.slice/asterisk.service
           ├─2678 /usr/sbin/asterisk -g -f -p -U asterisk
           └─7051 astcanary /var/run/asterisk/alt.asterisk.canary.tweet.tweet.tweet 2678

Nov 27 12:55:57 raspberrypi asterisk[2678]: [Nov 27 12:55:57] NOTICE[9647]: res_config_ldap.c:1832 parse_config: No directory user fo
Nov 27 12:55:57 raspberrypi asterisk[2678]: [Nov 27 12:55:57] ERROR[9647]: res_config_ldap.c:1858 parse_config: No directory URL or h
Nov 27 12:55:57 raspberrypi asterisk[2678]: [Nov 27 12:55:57] NOTICE[9647]: res_config_ldap.c:1776 reload: Cannot reload LDAP RealTim
Nov 27 12:55:57 raspberrypi asterisk[2678]: [Nov 27 12:55:57] NOTICE[9647]: cdr.c:4541 cdr_toggle_runtime_options: CDR simple logging
Nov 27 12:55:57 raspberrypi asterisk[2678]: [Nov 27 12:55:57] NOTICE[9648]: sorcery.c:1348 sorcery_object_load: Type 'system' is not
```

Figure 1: Snapshot of the Asterisk source code size.

2. Using either a SIP phone plugged into the same LAN as the Raspberry Pi Model 3 (you may need an Ethernet switch to create the network), or with a PC running a softphone application connected to the Raspberry Pi Model 3, configure Asterisk to provide a voicemail message at extension 100. Configure your SIP phone or softphone and register with Asterisk. Show your Asterisk setup in a screenshot.
   Zoiper was installed on the phone to place the call. To set up the account on Zoiper, 6001 was entered for the account name with the IP address of the Raspberry pi, which was 10.1.1.227 in this case.

```
[from-internal]
exten = 100,1,Answer()
same = n,Wait(3)
same = n,Playback(hello-world)
same = n,Hangup()
```

Figure 2: Snapshot of the extensions.conf file

```
[general]
context=default

[6001]
type=friend
context=from-internal
host=dynamic
secret=password
disallow=all
allow=ulaw
```

Figure 3: Snapshot of the sip.conf file

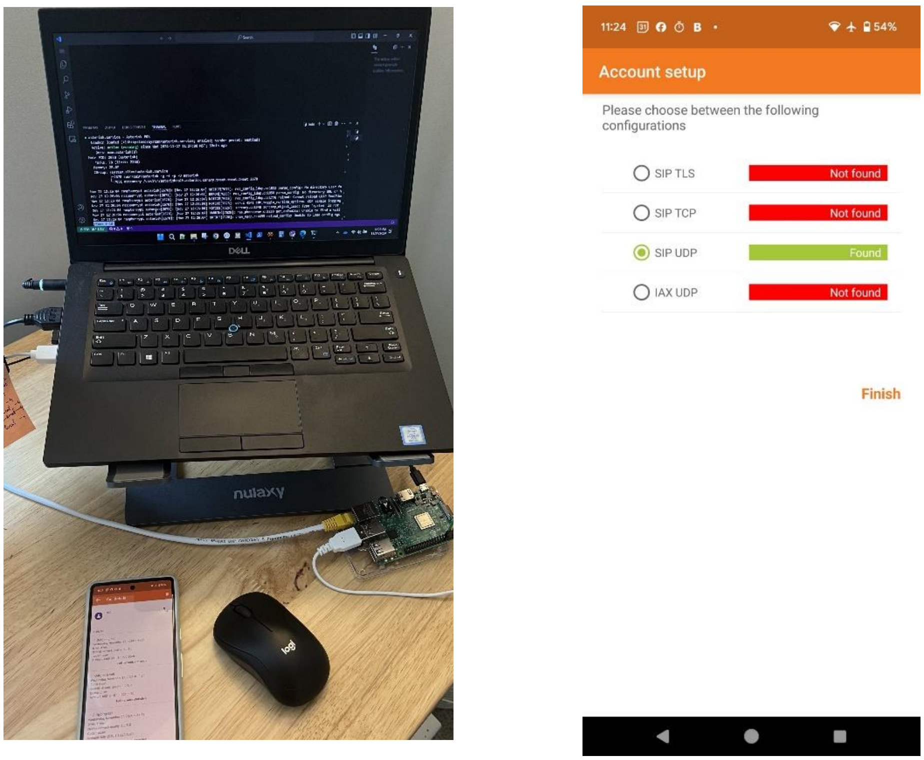
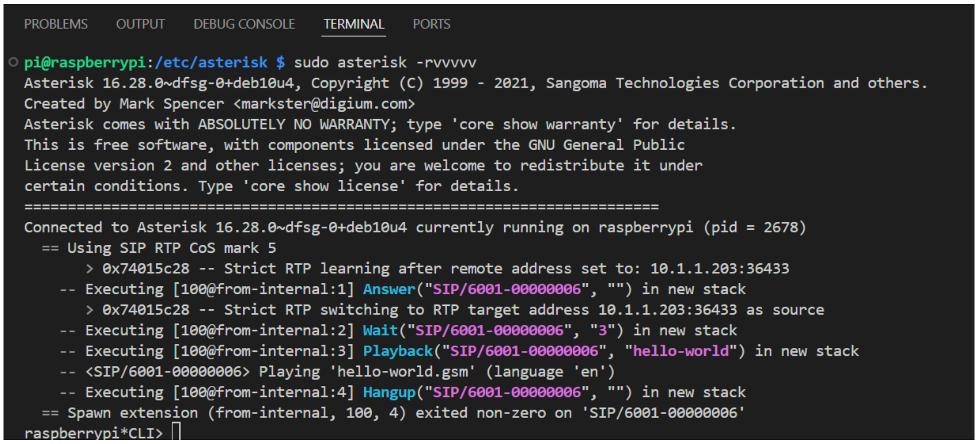


Figure 4: Asterisk setup snapshot.

3. Make a call to extension 100 and record what you hear. Show your Asterisk setup in a screenshot.
   When the phone dials extension 100, Asterisk plays a sound file "Hello World" to the channel and then hangs up. The CLI of Asterisk during the call is shown in Figure 5.

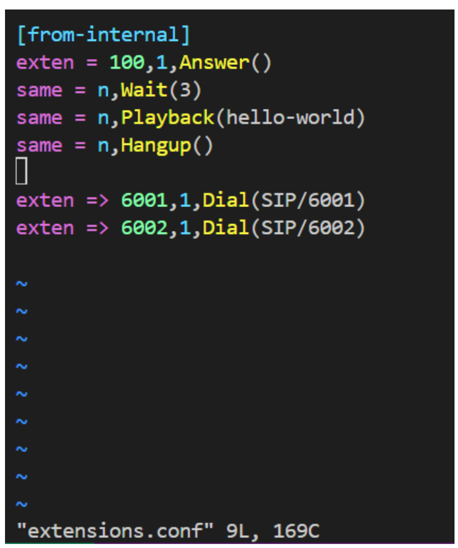


```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

pi@raspberrypi:/etc/asterisk $ sudo asterisk -rvvvvv
Asterisk 16.28.0~dfsg-0+deb10u4, Copyright (C) 1999 - 2021, Sangoma Technologies Corporation and others.
Created by Mark Spencer <markster@digium.com>
Asterisk comes with ABSOLUTELY NO WARRANTY; type 'core show warranty' for details.
This is free software, with components licensed under the GNU General Public
License version 2 and other licenses; you are welcome to redistribute it under
certain conditions. Type 'core show license' for details.
=========================================================================
Connected to Asterisk 16.28.0~dfsg-0+deb10u4 currently running on raspberrypi (pid = 2678)
  == Using SIP RTP CoS mark 5
       > 0x74015c28 -- Strict RTP learning after remote address set to: 10.1.1.203:36433
    -- Executing [100@from-internal:1] Answer("SIP/6001-00000006", "") in new stack
       > 0x74015c28 -- Strict RTP switching to RTP target address 10.1.1.203:36433 as source
    -- Executing [100@from-internal:2] Wait("SIP/6001-00000006", "3") in new stack
    -- Executing [100@from-internal:3] Playback("SIP/6001-00000006", "hello-world") in new stack
    -- <SIP/6001-00000006> Playing 'hello-world.gsm' (language 'en')
    -- Executing [100@from-internal:4] Hangup("SIP/6001-00000006", "") in new stack
  == Spawn extension (from-internal, 100, 4) exited non-zero on 'SIP/6001-00000006'
raspberrypi*CLI>
```

Figure 5: Snapshot of the Asterisk CLI during the call.

4. Add another SIP phone or softphone to the network and make a phone-to-phone call.
   To make a phone-to-phone call Zoiper was installed on a laptop as shown in the setup snapshots below.



```
[from-internal]
exten = 100,1,Answer()
same = n,Wait(3)
same = n,Playback(hello-world)
same = n,Hangup()

exten => 6001,1,Dial(SIP/6001)
exten => 6002,1,Dial(SIP/6002)

~
~
~
~
~
~
~
~
~
"extensions.conf" 9L, 169C
```

Figure 6: Snapshot of the extensions.conf file for a second phone.

```
[general]
context=default

[6001]
type=friend
context=from-internal
host=dynamic
secret=password
disallow=all
allow=ulaw


[6002]
type=friend
context=from-internal
host=dynamic
secret=password
disallow=all
```

Figure 7: Snapshot of the sip.conf file with a second user added.

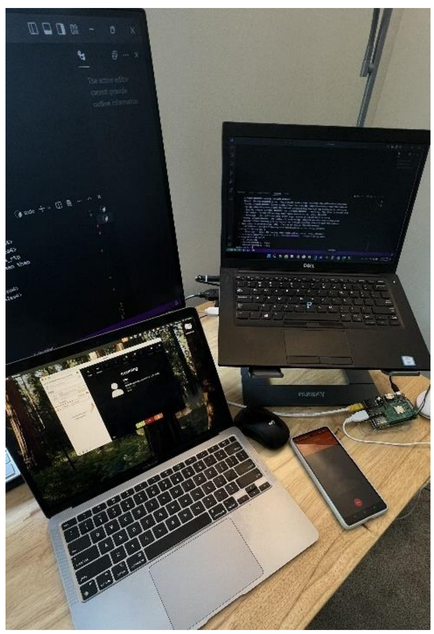


Figure 8: Snapshot of the setup to make a phone-to-phone call.

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

  == Using SIP RTP CoS mark 5
       > 0x74015c28 -- Strict RTP learning after remote address set to: 10.1.1.203:34393
    -- Executing [6002@from-internal:1] Dial("SIP/6001-00000007", "SIP/6002") in new stack
  == Using SIP RTP CoS mark 5
    -- Called SIP/6002
    -- SIP/6002-00000008 is ringing
       > 0x73d085a0 -- Strict RTP learning after remote address set to: 10.1.1.239:49955
    -- SIP/6002-00000008 answered SIP/6001-00000007
    -- Channel SIP/6002-00000008 joined 'simple_bridge' basic-bridge <445effe9-af4e-4453-bb00-cf16fca42ea4>
    -- Channel SIP/6001-00000007 joined 'simple_bridge' basic-bridge <445effe9-af4e-4453-bb00-cf16fca42ea4>
       > Bridge 445effe9-af4e-4453-bb00-cf16fca42ea4: switching from simple_bridge technology to native_rtp
       > Remotely bridged 'SIP/6001-00000007' and 'SIP/6002-00000008' - media will flow directly between them
       > 0x73d085a0 -- Strict RTP learning after remote address set to: 10.1.1.239:49955
       > 0x74015c28 -- Strict RTP switching to RTP target address 10.1.1.203:34393 as source
    -- Channel SIP/6001-00000007 left 'native_rtp' basic-bridge <445effe9-af4e-4453-bb00-cf16fca42ea4>
    -- Channel SIP/6002-00000008 left 'native_rtp' basic-bridge <445effe9-af4e-4453-bb00-cf16fca42ea4>
```

Figure 9: Snapshot of the Asterisk CLI during the phone-to-phone call.

Appendix- ext.conf

```
[from-internal]

exten = 100,1,Answer()

same  = n,Wait(3)

same  = n,Playback(hello-world)

same = n,Hangup()


exten => 6001,1,Dial(SIP/6001)

exten => 6002,1,Dial(SIP/6002)
```

Appendix-sip.conf

```
[general]
context=default

[6001]
type=friend
context=from-internal
host=dynamic
secret=password
disallow=all
allow=ulaw

[6002]
type=friend
context=from-internal
host=dynamic
secret=password
disallow=all
allow=ulaw
```

Appendix-References

[1] https://github.com/asterisk/documentation/blob/main/docs/Getting-Started/Hello-World.md

Module 5: Build a IoT application

James Way & Venetia Furtado

Date: 12/10/2024

Python code:

```
############################################################################
##########
# ECEN 5803 - Mastering Embedded System architecture
# Project 2 Module 5 - Web server application
# Submitted by: James Way & Venetia Furtado
#
# Description: The code for setting up the web server was adapted from
tutorials
# available on the Python website, as cited in the references. It
utilizes the
# http.server package to create a web server that displays the current
time and
# the number of page accesses. The server listens on port 8080 and
dynamically
# generates an HTML page in response to incoming requests.
#
# References:
# https://pythonbasics.org/webserver/
# https://docs.python.org/3/library/http.server.html
#
############################################################################
##########
from http.server import BaseHTTPRequestHandler, HTTPServer
from datetime import datetime

# Globals to track the number of accesses
access_count = 0
```

```python
class RequestHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        global access_count
        access_count += 1

        # Get the current time
        current_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

        # Generate the response
        response = f"""
        <html>
        <head><title>Raspberry Pi Web Server</title></head>
        <body>
            <h1>Welcome to Raspberry Pi Web Server</h1>
            <p>Current Time: {current_time}</p>
            <p>Number of Accesses: {access_count}</p>
        </body>
        </html>
        """

        # Send HTTP headers
        self.send_response(200)
        self.send_header("Content-Type", "text/html")
        self.send_header("Content-Length", str(len(response)))
        self.end_headers()

        # Send the HTML content
        self.wfile.write(response.encode("utf-8"))

def run_server():
    host = "0.0.0.0"  # Listen on all available interfaces
    port = 8080       # Port to listen on
    server_address = (host, port)

    # Create the HTTP server
    httpd = HTTPServer(server_address, RequestHandler)
    print(f"Server running on http://{host}:{port}/...")
```

```
    try:
        # Start the server
        httpd.serve_forever()
    except KeyboardInterrupt:
        print("\nShutting down the server.")
        httpd.server_close()

if __name__ == "__main__":
    run_server()
```

Output:

Figure 1: The web server page at different instances.

Appendix-References

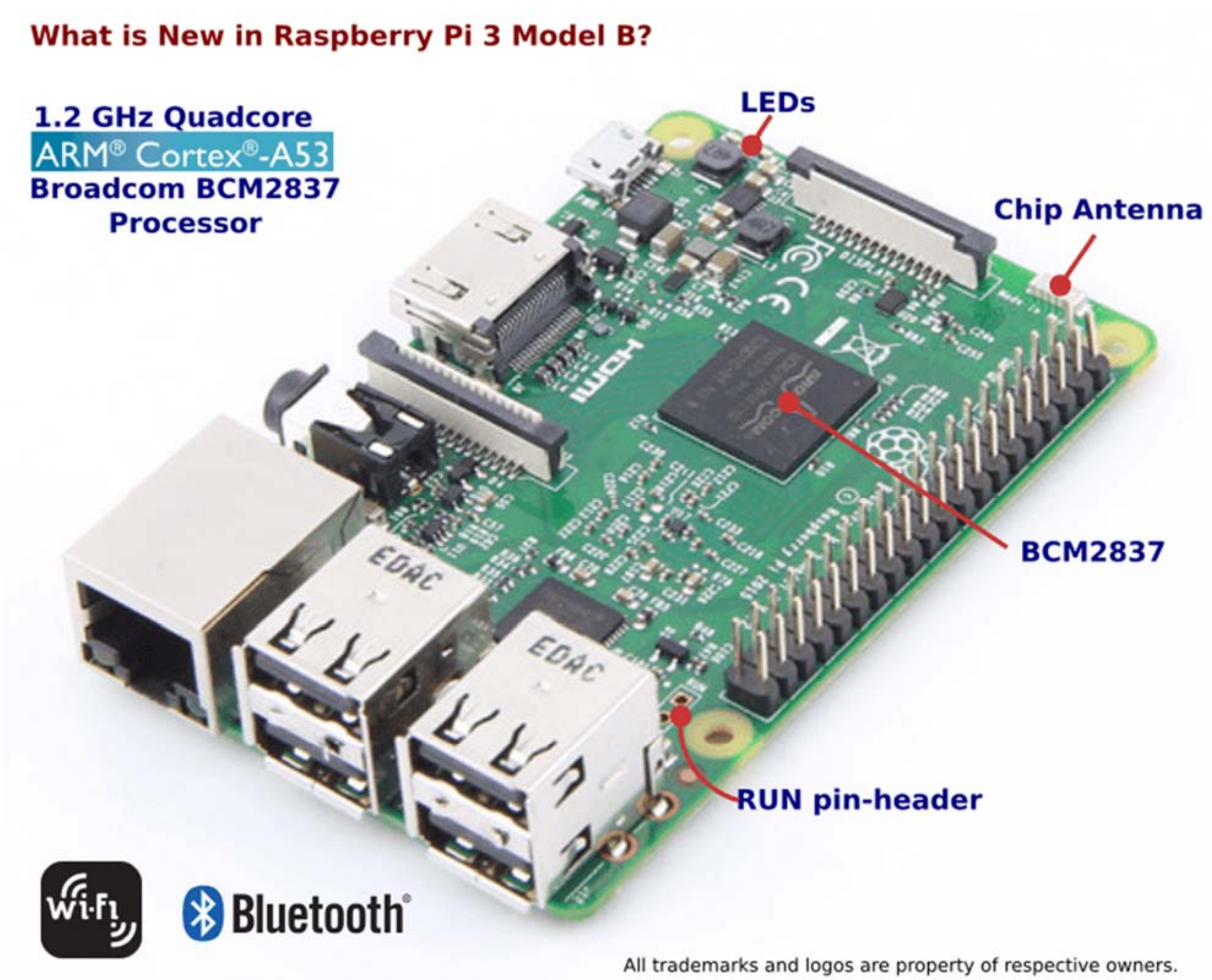
[1] https://pythonbasics.org/webserver/

[2] https://docs.python.org/3/library/http.server.html

11 Appendix - Datasheets

Raspberry Pi 3 Model B

Raspberry Pi® is an **ARM** based credit card sized **SBC**(Single Board Computer) created by Raspberry Pi Foundation. Raspberry Pi runs Debian based **GNU/Linux** operating system Raspbianand ports of many other OSes exist for this SBC.



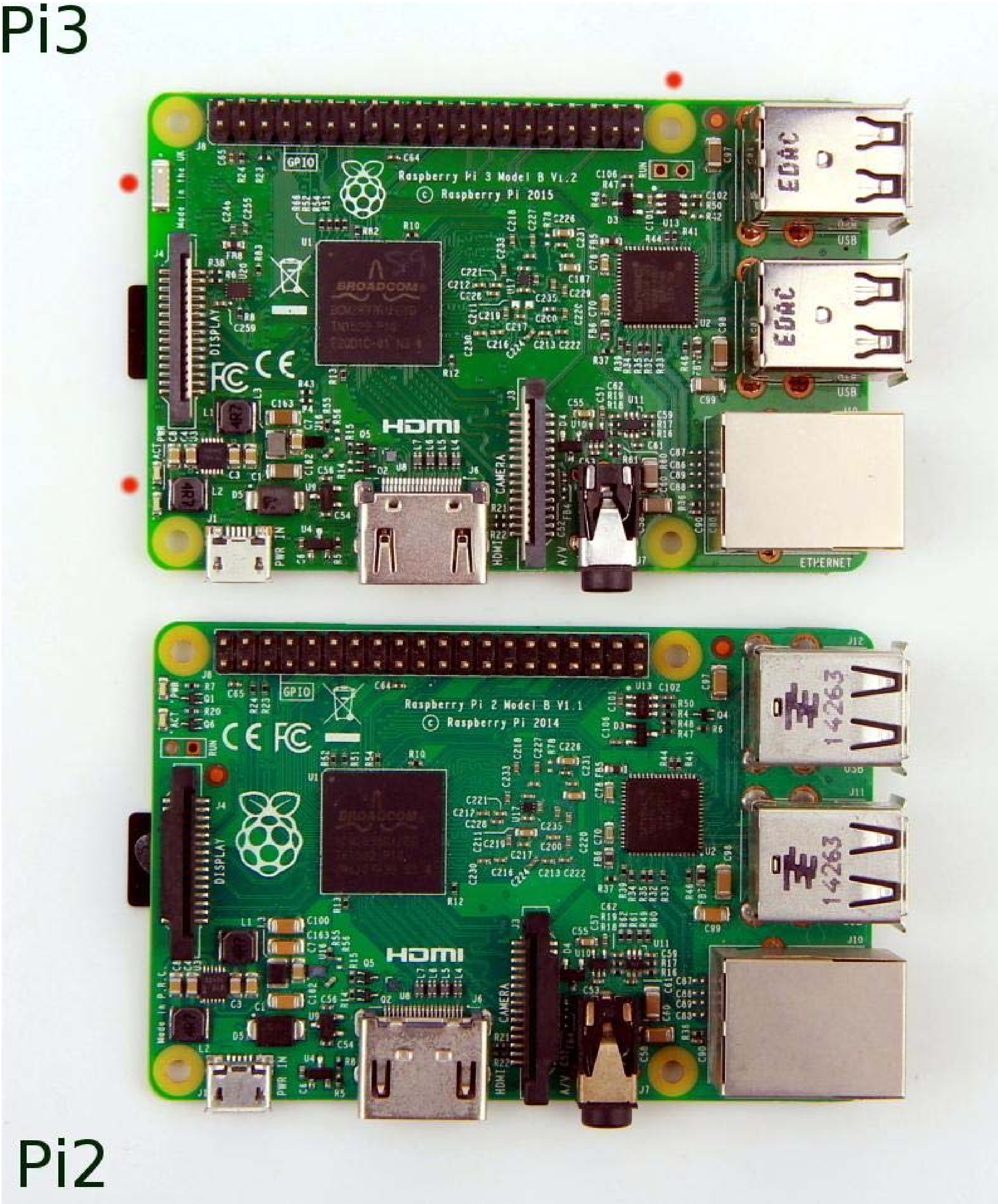
Raspberry Pi Foundation has announced a new version **Raspberry Pi 3**. Read announcement here. With on-board **WiFi** / **Bluetooth** support and an 64bit improved Processor, **Raspberry Pi v3**will be an exciting board for Makers, Engineers and Students.

What is new in Raspberry Pi 3

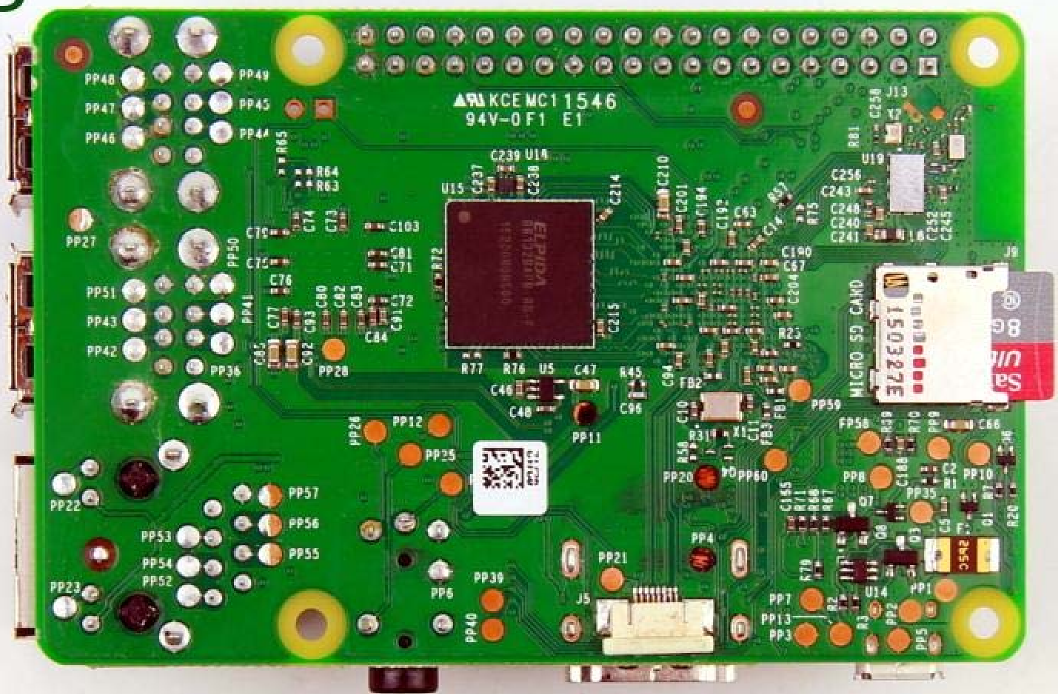
Board	Raspberry Pi 2 Model B	**Raspberry Pi 3 Model B**
Processor	Broadcom BCM2836	Broadcom **BCM2837**
CPU Core	Quadcore ARM Cortex-A7, 32Bit	Quadcore **ARM Cortex-A53, 64Bit**
Clock Speed	900 MHz	1.2GHz (Roughly 50% faster than Pi2)
RAM	1 GB	1 GB
GPU	250 MHz VideoCore IV®	**400 MHz** VideoCore IV®
Network Connectivity	1 x 10 / 100 Ethernet (RJ45 Port)	1 x 10 / 100 Ethernet (RJ45 Port)
Wireless Connectivity	None	**802.11n wireless LAN (WiFi) and Bluetooth 4.1**
USB Ports	4 x USB 2.0	4 x USB 2.0
GPIOs	2 x 20 Pin Header	2 x 20 Pin Header
Camera Interface	15-pin MIPI	15-pin MIPI
Display Interface	DSI 15 Pin / HDMI Out / Composite RCA	DSI 15 Pin / HDMI Out / Composite RCA
Power Supply (Current Capacity)	1.8 A	**2.5 A**

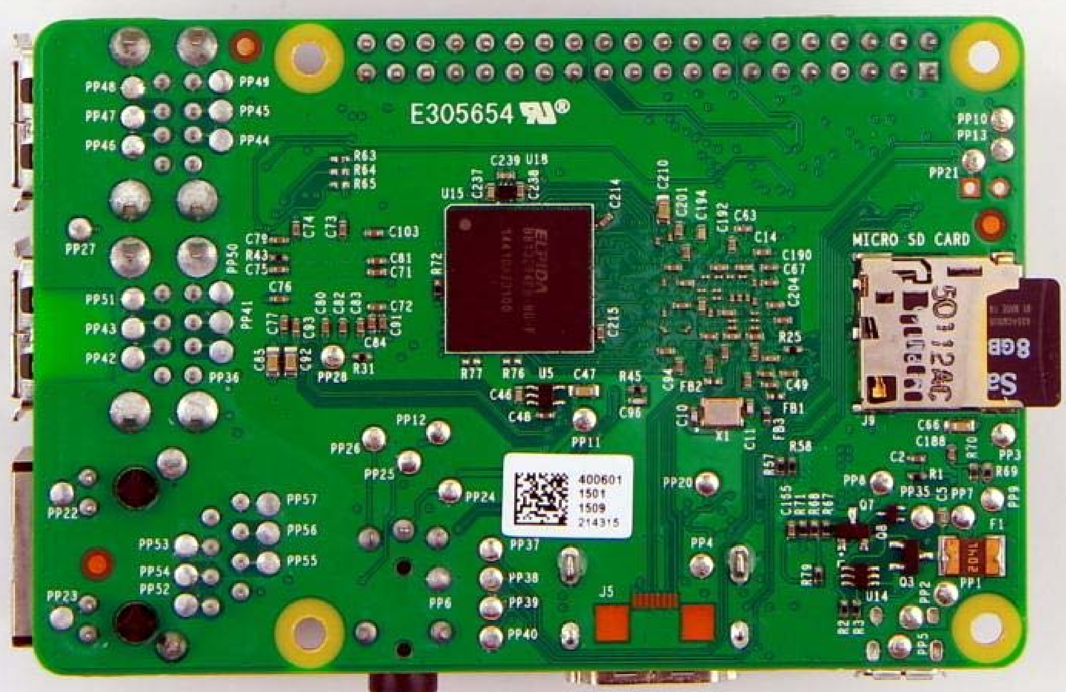
Board

- The size of the Pi 2 and Pi 3 boards are the same.
- There is slight change in component placement to allow addition of WiFi / Bluetooth SoC & Chip antenna in Pi 3.


Pi3
Raspberry Pi 3 Model B V1.2
© Raspberry Pi 2015
GPIO
DISPLAY
HDMI
CAMERA
PWR IN
USB
ETHERNET
Raspberry Pi 2 Model B V1.1
© Raspberry Pi 2014
Pi2

Pi3



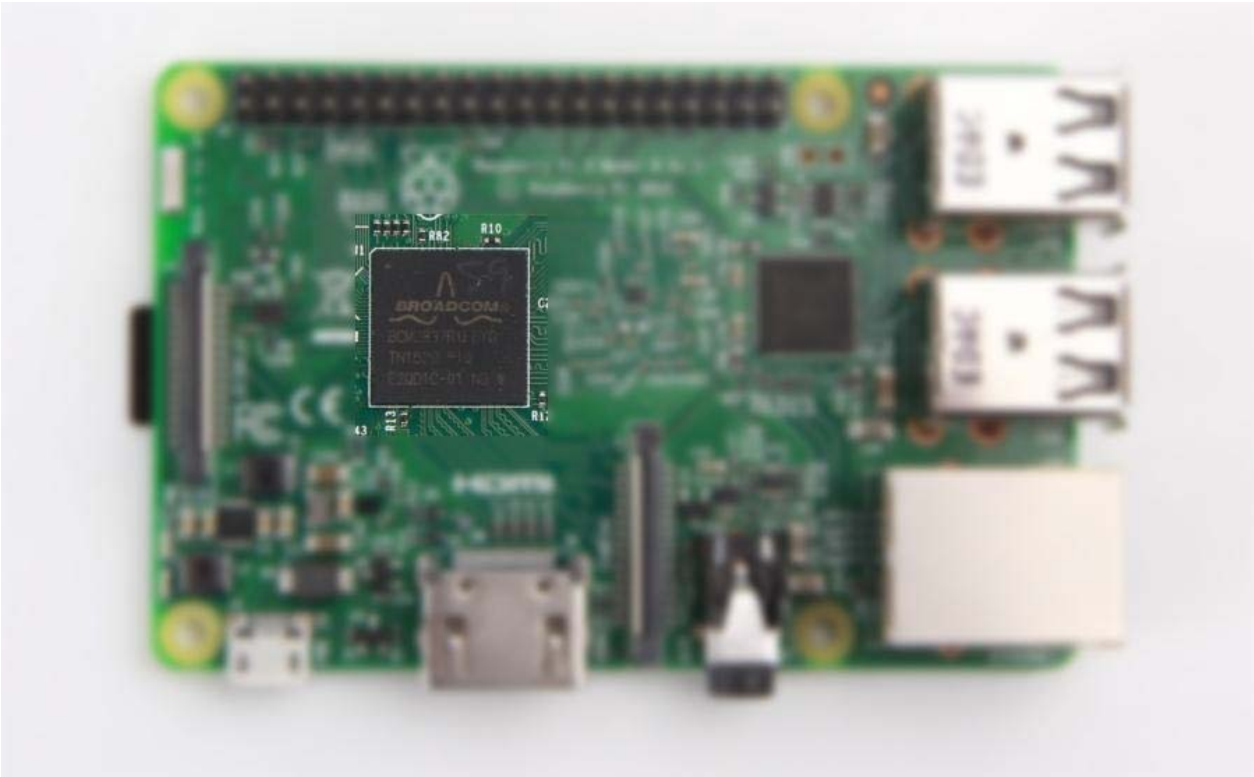


Pi2

System on Chip (SoC)

Broadcom BCM2837 SoC

- Application Processor
  - 64 bit
    - Quad Core
    - 1.2 GHz
    - ARM Cortex-A53 Processor (ARM V8 ISA)
- GPU
  - 400 MHz
  - Videocore IV Multimedia Co-Processor



Chip Antenna

A ceramic chip antenna is used by WiFi and Bluetooth 4.1 SoC BCM43438. The chip antenna moves the indicator LEDs that were present in Pi 2 to the lower side of PCB.

Repositioned LEDs

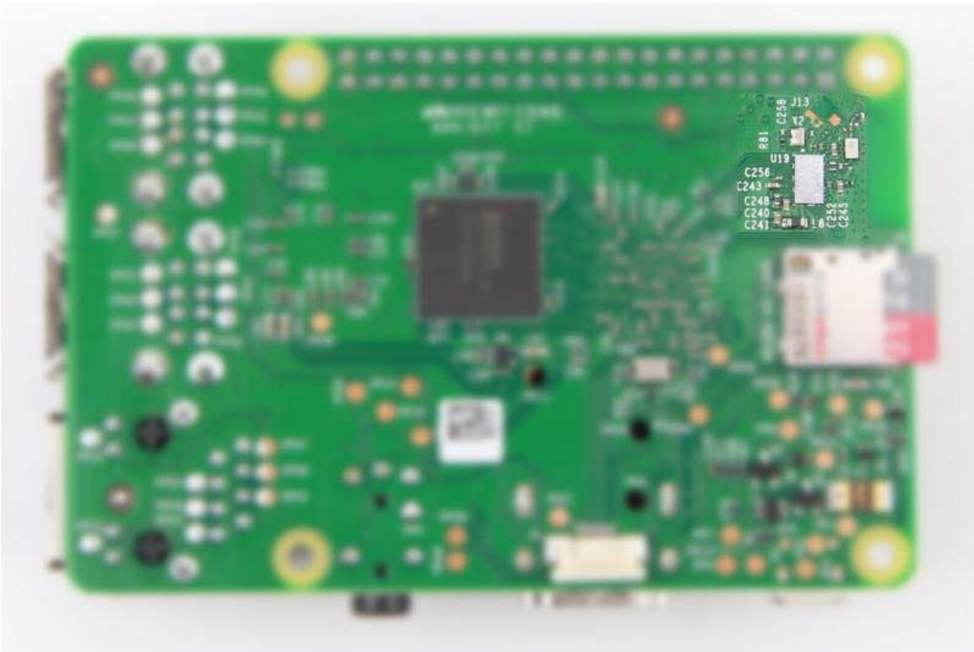
The ACT and PWR LEDs are repositioned as shown below in Raspberry Pi 3 when compared to Pi 2.

Repositioned RUN pin-header

The RUN pin-header is also repositioned

WiFi / Bluetooth SoC BCM43438

WiFi and Bluetooth 4.1 (Classic and LE) are provided by Broadcom BCM43438 chip



Buy:

- Raspberry Pi 3 Preorder
- Raspberry Pi Main Boards
- Raspberry Pi Related Products

This page will be updated with new information as and when available.

Tech Support

Please submit any technical issue into our forum or drop mail to techsupport@seeed.cc.