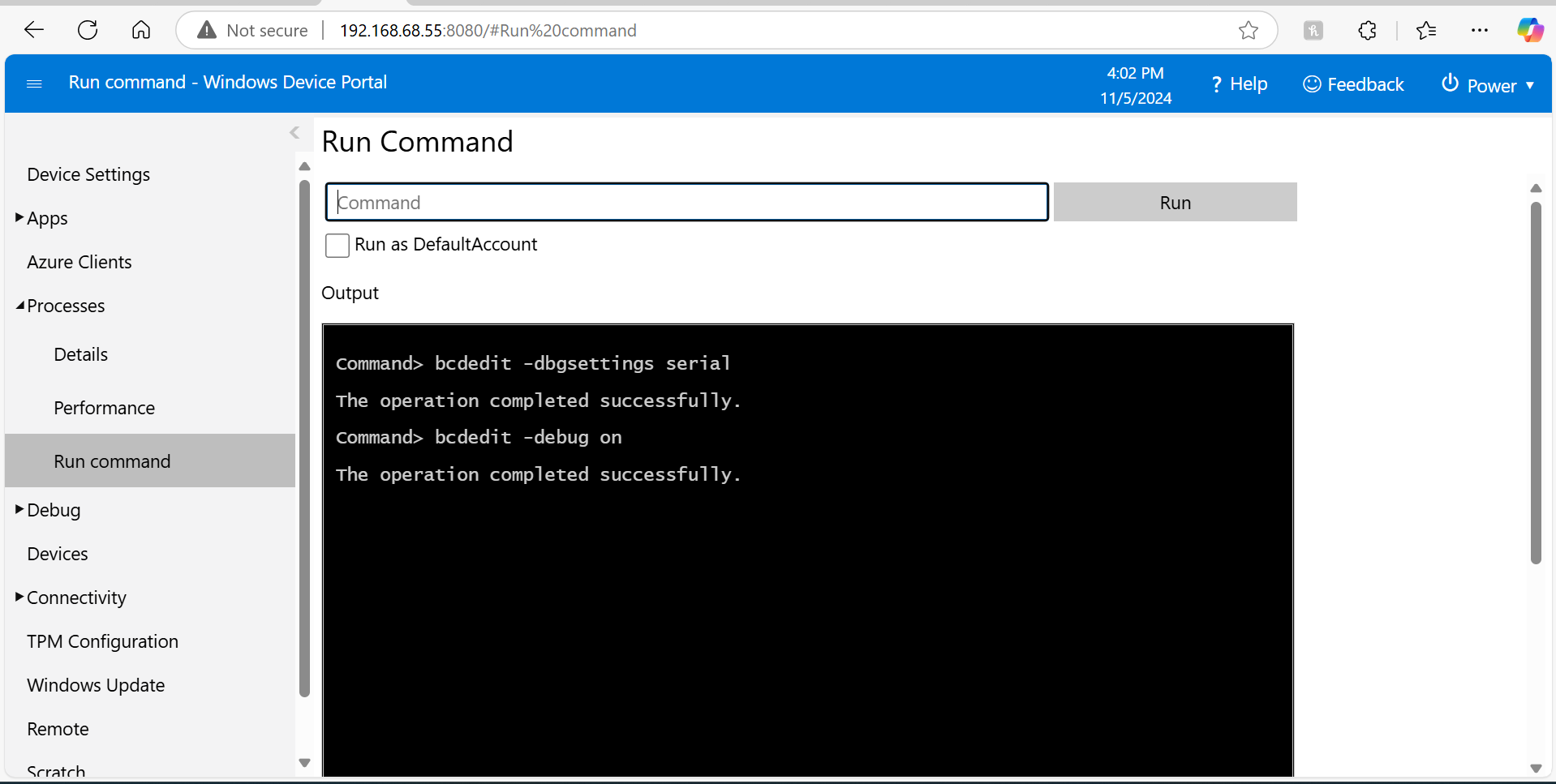
1. Windows 10 IoT Core was installed by flashing the .FFU onto an SD card using Rufus.
2. To enable Windows 10 IoT for sending debug messages on the serial port, the following commands were executed on the Raspberry Pi 3:

This command enables the serial connection for debugging.

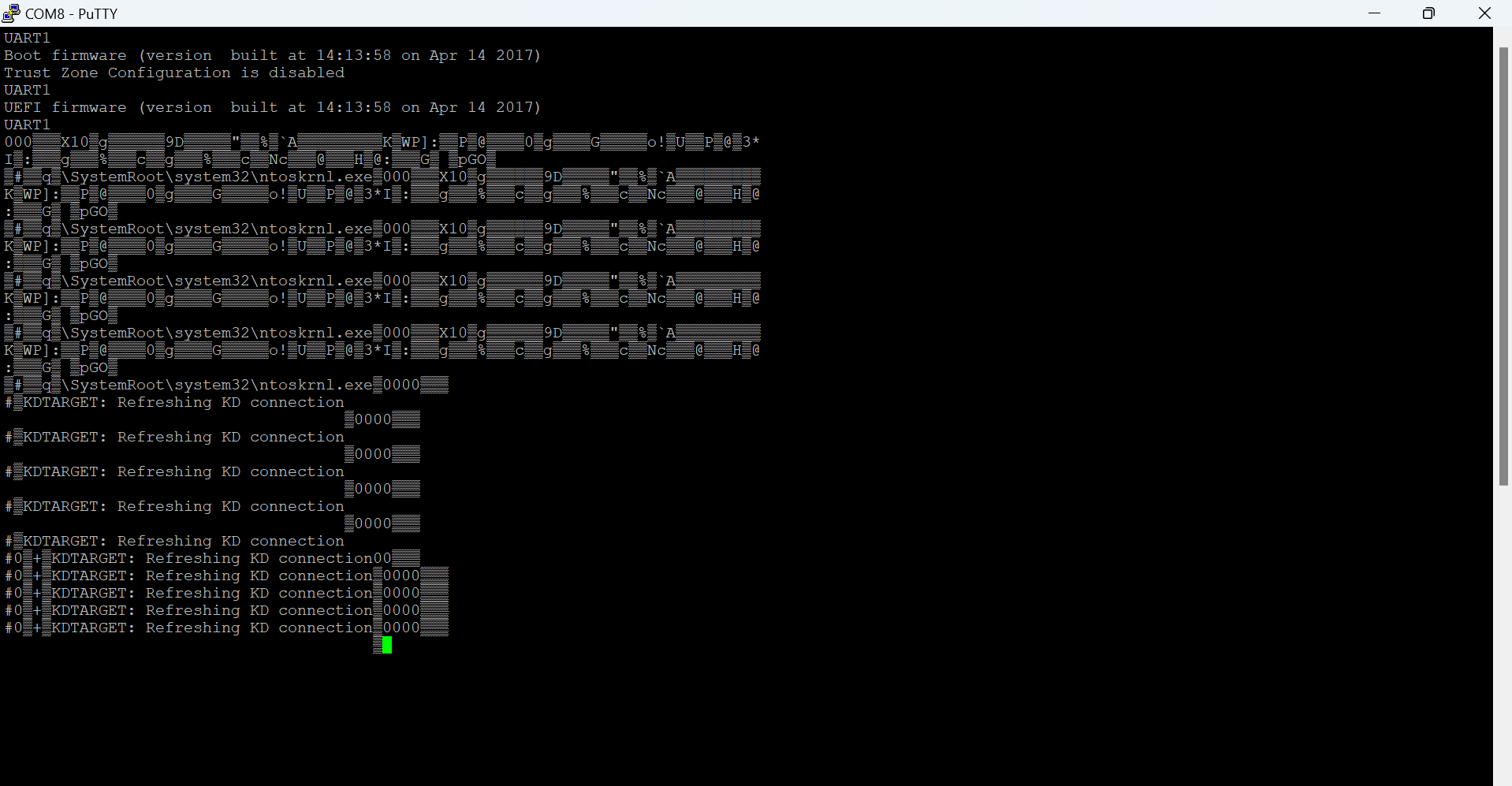
bcdedit -dbgsettings serial

This command turns on debugging on the device.

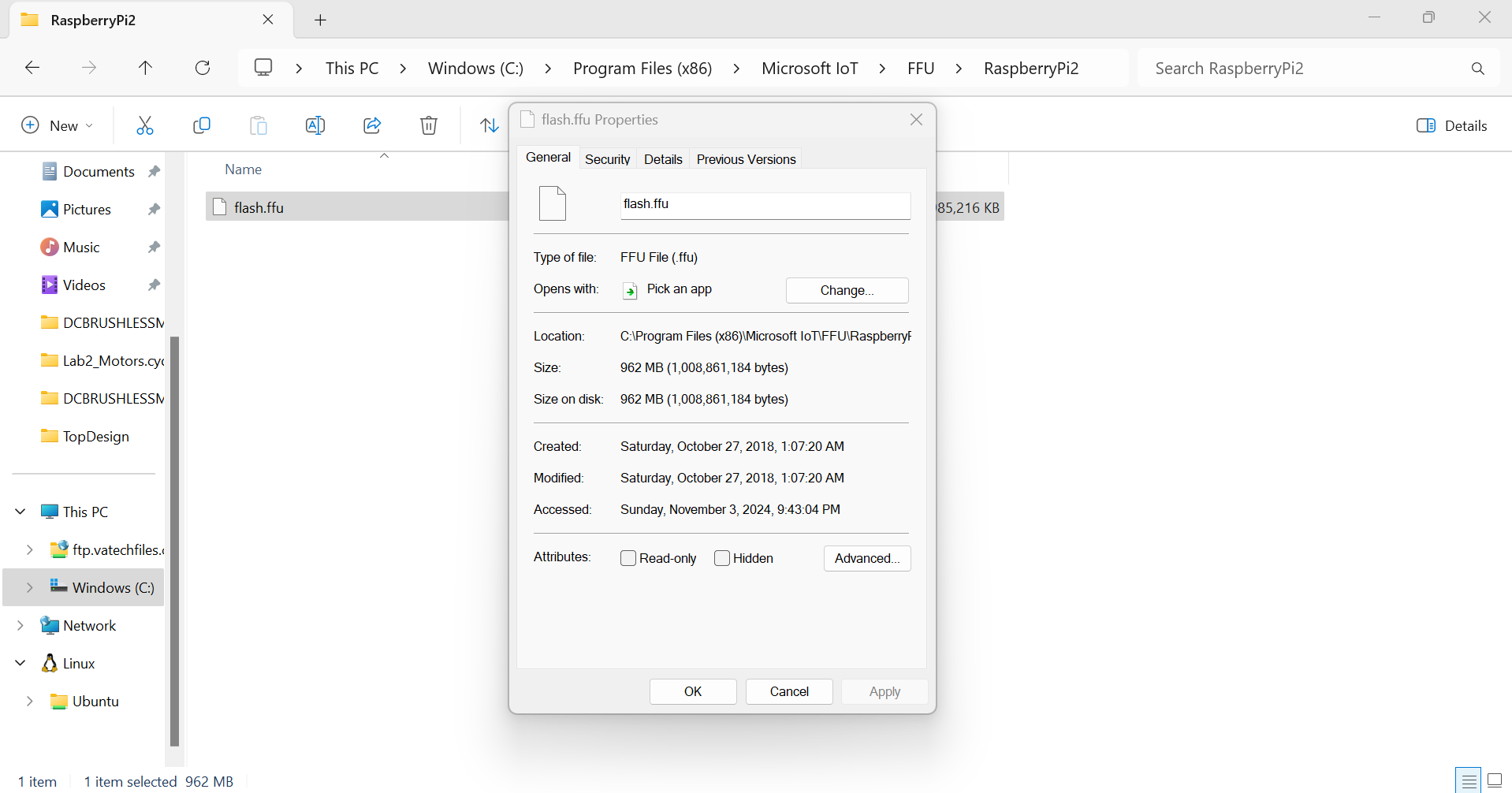
bcdedit -debug on



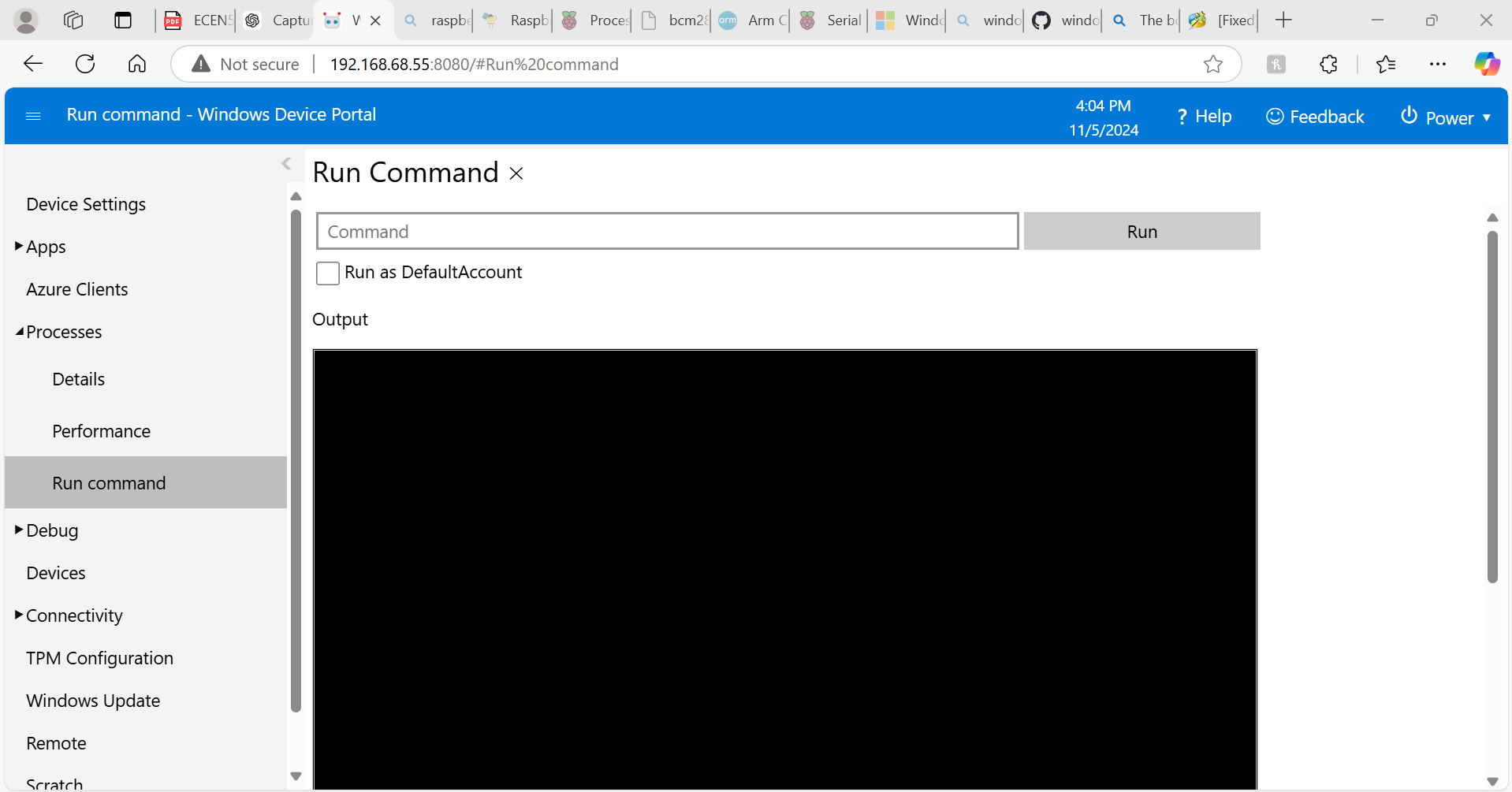
A USB-to-TTL adapter was connected to the Raspberry Pi 3 from my PC and a PuTTY terminal window was opened with a baud rate of 921600. Upon booting up, the serial port sent the following debug messages:



1. The image size was determined by viewing the file size of the .FFU image used to flash the SD card of the Raspberry Pi 3 (Model B). The image size was 962 MB.



1. A screenshot of the terminal window is seen below. This was taken from the Windows Device Portal which was connected through the IP address of the Raspberry Pi.



1. When I reboot the system, the first screen that appears in a colorful rainbow screen which indicates power is being supplied to the Raspberry Pi. The Windows logo then appears with a buffering circle indicating the system is booting into the Windows 10 IoT operating system. It then transitions to an image of a raspberry pi and then transitions to the home screen for Windows 10 IoT Core.

The G.711 code works for both encoding(from PCM to ulaw) and decoding(from ulaw to PCM). To the run the given code, there is a makefile to compile and using the executable generated(run make) from the makefile, t can be run as

For decoding : ./G\_711 input\_file output\_file 0

For encoding : ./G\_711 input\_file output\_file 1

C code for G.711 decoder:

/\*

\* g711.c: Encodes and decodes G.711 A-law and G.711 µ-Law from and to Linear pulse code modulation (LPCM)

\*

\* The nominal value recommended for the sampling rate is 8000 samples per second. The tolerance on that rate

\* should be ± 50 parts per million (ppm).

\*

\* Eight binary digits per sample should be used for international circuits.

\*

\* Two encoding laws are recommended and these are commonly referred to as the A-law and the µ-law

\*

\* First (MSB) identifies polarity

\* Bits two, three, and four identify segment

\* Final four bits quantize the segment

\*

\* Character signals obtained by inverting even bits of signal

\*

\* References:

// https://en.wikipedia.org/wiki/WAV

// https://github.com/apple-opensource/tcl/blob/master/tcl\_ext/snack/snack/generic/g711.c

// https://www.seas.ucla.edu/spapl/weichu/htkbook/node101\_mn.html

\*/

#include <stdio.h>

#include <stdint.h>

#include <stdlib.h>

#define SIGN\_BIT (0x80) /\* Sign bit for a A-law byte. \*/

#define QUANT\_MASK (0xf) /\* Quantization field mask. \*/

#define NSEGS (8) /\* Number of A-law segments. \*/

#define SEG\_SHIFT (4) /\* Left shift for segment number. \*/

#define SEG\_MASK (0x70) /\* Segment field mask. \*/

#define BIAS (0x84) /\* Bias for linear code. \*/

#define CLIP 8159

uint8\_t wav\_header\_u\_law\_to\_pcm[44] = {

// "RIFF" Chunk

'R', 'I', 'F', 'F', // Chunk ID

0, 0, 0, 0, // Chunk Size (to be updated later)

'W', 'A', 'V', 'E', // Format

// "fmt " Subchunk

'f', 'm', 't', ' ', // Subchunk1 ID

16, 0, 0, 0, // Subchunk1 Size (16 for PCM)

1, 0, // Audio Format (1 for PCM)

1, 0, // Num Channels (1 for mono)

0x40, 0x1F, 0x00, 0x00, // Sample Rate (8 kHz: 0x1F40)

0x80, 0x3E, 0x00, 0x00, // Byte Rate (SampleRate \* NumChannels \* 2)

2, 0, // Block Align (NumChannels)

16, 0, // Bits per Sample (16 for PCM)

// "data" Subchunk

'd', 'a', 't', 'a', // Subchunk2 ID

0, 0, 0, 0 // Subchunk2 Size (to be updated later)

};

uint8\_t wav\_header\_pcm\_to\_u\_law[44] = {

// "RIFF" Chunk

'R', 'I', 'F', 'F', // Chunk ID

0, 0, 0, 0, // Chunk Size (to be updated later)

'W', 'A', 'V', 'E', // Format

// "fmt " Subchunk

'f', 'm', 't', ' ', // Subchunk1 ID

16, 0, 0, 0, // Subchunk1 Size (16 for PCM)

7, 0, // Audio Format (7 for u-law)

1, 0, // Num Channels (1 for mono)

0x40, 0x1F, 0x00, 0x00, // Sample Rate (8 kHz: 0x1F40)

0x40, 0x1F, 0x00, 0x00, // Byte Rate (SampleRate \* NumChannels)

1, 0, // Block Align (NumChannels)

8, 0, // Bits per Sample (8 for u-law)

// "data" Subchunk

'd', 'a', 't', 'a', // Subchunk2 ID

0, 0, 0, 0 // Subchunk2 Size (to be updated later)

};

static short seg\_aend[8] = {0x1F, 0x3F, 0x7F, 0xFF,

0x1FF, 0x3FF, 0x7FF, 0xFFF};

static short seg\_uend[8] = {0x3F, 0x7F, 0xFF, 0x1FF,

0x3FF, 0x7FF, 0xFFF, 0x1FFF};

static short

search(

short val,

short \*table,

short size)

{

short i;

for (i = 0; i < size; i++) {

if (val <= \*table++)

return (i);

}

return (size);

}

short

Snack\_Mulaw2Lin(

unsigned char u\_val)

{

short t;

/\* Complement to obtain normal u-law value. \*/

u\_val = ~u\_val;

/\*

\* Extract and bias the quantization bits. Then

\* shift up by the segment number and subtract out the bias.

\*/

t = ((u\_val & QUANT\_MASK) << 3) + BIAS;

t <<= ((unsigned)u\_val & SEG\_MASK) >> SEG\_SHIFT;

return ((u\_val & SIGN\_BIT) ? (BIAS - t) : (t - BIAS));

}

unsigned char

Snack\_Lin2Mulaw(

short pcm\_val) /\* 2's complement (16-bit range) \*/

{

short mask;

short seg;

unsigned char uval;

pcm\_val = (short)(pcm\_val \* 2.0);

if (pcm\_val > 32767) pcm\_val = 32767;

if (pcm\_val < -32768) pcm\_val = -32768;

/\* Get the sign and the magnitude of the value. \*/

pcm\_val = pcm\_val >> 2;

if (pcm\_val < 0) {

pcm\_val = -pcm\_val;

mask = 0x7F;

} else {

mask = 0xFF;

}

if ( pcm\_val > CLIP ) pcm\_val = CLIP; /\* clip the magnitude \*/

pcm\_val += (BIAS >> 2);

/\* Convert the scaled magnitude to segment number. \*/

seg = search(pcm\_val, seg\_uend, 8);

/\*

\* Combine the sign, segment, quantization bits;

\* and complement the code word.

\*/

if (seg >= 8) /\* out of range, return maximum value. \*/

return (unsigned char) (0x7F ^ mask);

else {

uval = (unsigned char) (seg << 4) | ((pcm\_val >> (seg + 1)) & 0xF);

return (uval ^ mask);

}

}

/\*

\* Argv used to pass in input and output filenames

\* Argv[1]: Input, Argv[2]: Output, Argv[3]: Conversion mode

\*

\* 0: pcm to ulaw

\* 1: ulaw to pcm

\*/

int main(int argc, char \*argv[])

{

if (argc != 4)

{

printf("Argv used to pass in input and output filenames.\n Argv[1]: Input, Argv[2]: Output, Argv[3]: Conversion mode\n Conversion modes:\n 0: ulaw to pcm(decoding) \n 1: pcm to ulaw(encoding)\n");

return 1;

}

FILE \*input, \*output;

uint32\_t inputFileSize, outputFileSize, size\_without\_header, chunk\_size;

unsigned char input\_buffer\_data;

long bytesRead, bytesWritten;

short output\_data;

unsigned char output\_buffer\_data;

switch (atoi(argv[3]))

{

case 0:

/\* Input file \*/

input = fopen(argv[1],"rb");

if(input == NULL)

{

printf("Error opening input file\n");

return 1;

}

output = fopen(argv[2], "wb");

if(output == NULL)

{

printf("Error opening output file\n");

return 1;

}

fseek(output, 0, SEEK\_SET);

fwrite(wav\_header\_u\_law\_to\_pcm, sizeof(wav\_header\_u\_law\_to\_pcm), 1, output);

/\* Find size of input file \*/

fseek(input, 0, SEEK\_END);

inputFileSize = ftell(input);

size\_without\_header = inputFileSize - 44;

size\_without\_header \*= 2; // µ-law (8-bit) to PCM (16-bit)

chunk\_size = size\_without\_header + 36;

printf("inputFileSize: %ld\n", inputFileSize);

fseek(input, 44, SEEK\_SET);

/\* Read data into buffer \*/

while(1){

bytesRead = fread(&input\_buffer\_data, sizeof(input\_buffer\_data), 1, input);

if (bytesRead != 1)

{

break;

}

output\_data = Snack\_Mulaw2Lin(input\_buffer\_data);

fwrite(&output\_data, sizeof(output\_data), 1, output);

}

/\* Close connection \*/

fclose(input);

fseek(output, 4, SEEK\_SET); // Moving to the RIFF position in the header

fwrite(&chunk\_size, sizeof(chunk\_size), 1, output);

fseek(output, 40, SEEK\_SET);

fwrite(&size\_without\_header, sizeof(size\_without\_header), 1, output);

fclose(output);

printf("The decoding is complete");

break;

case 1:

/\* Input file \*/

input = fopen(argv[1],"rb");

if(input == NULL)

{

printf("Error opening input file\n");

return 1;

}

output = fopen(argv[2], "wb");

if(output == NULL)

{

printf("Error opening output file\n");

return 1;

}

fseek(output, 0, SEEK\_SET);

fwrite(wav\_header\_pcm\_to\_u\_law, sizeof(wav\_header\_pcm\_to\_u\_law), 1, output);

/\* Find size of input file \*/

fseek(input, 0, SEEK\_END);

inputFileSize = ftell(input);

size\_without\_header = inputFileSize - 44;

size\_without\_header /= 2; // PCM (16-bit) to µ-law (8-bit)

chunk\_size = size\_without\_header + 36;

printf("inputFileSize: %ld\n", inputFileSize);

fseek(input, 44, SEEK\_SET);

/\* Read data into buffer \*/

while(1){

bytesRead = fread(&input\_buffer\_data, sizeof(input\_buffer\_data), 1, input);

if (bytesRead != 1)

{

break;

}

output\_buffer\_data = Snack\_Lin2Mulaw(input\_buffer\_data);

fwrite(&output\_buffer\_data, sizeof(output\_buffer\_data), 1, output);

}

/\* Close connection \*/

fclose(input);

fseek(output, 4, SEEK\_SET); // Moving to the RIFF position in the header

fwrite(&chunk\_size, sizeof(chunk\_size), 1, output);

fseek(output, 40, SEEK\_SET);

fwrite(&size\_without\_header, sizeof(size\_without\_header), 1, output);

fclose(output);

printf("The encoding is complete");

break;

default:

printf("Invalid mode\n");

return 1;

}

return 0;

}

**Makefile**

CC=gcc

G\_711: G\_711.c

$(CC) G\_711.c -o G\_711

clean:

rm G\_711

Decoded speech:

1. The ship was torn apart on the sharp reef.
2. Sickness kept him home the third week.
3. The box will hold 7 gifts at once.
4. Jazz and swing fans like fast music.

7. Windows 10 IoT provides a graphical user interface similar to that of Raspbian in that it provides support for running applications with a visual interface. Windows 10 IoT includes applications such as viewing the weather, a web browser as well other applications that can be used to display information. Windows 10 IoT also provides a terminal to run system commands and manage the device. The behavior of Windows 10 IoT is different from Linux in that Windows 10 IoT does not provide a full desktop environment and is more focused on IoT-specific applications instead of for general-purpose use.