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

James Way & Venetia Furtado

Date: 11/10/2024

# Flash Windows 10 IoT

* Download Raspberry Pi 2 & 3 build from [Downloads - Windows IoT | Microsoft Learn](https://learn.microsoft.com/en-us/previous-versions/windows/iot-core/downloads#latest-windows-10-iot-core-builds)
* 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.
* A screenshot of a computer

  Description automatically generatedSelect “START”.

Figure 1: Rufus GUI - flashing Windows 10 IoT

# 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.

A circuit board with wires connected to it

Description automatically generatedA screenshot of a computer

Description automatically generated

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

Figure 3: Serial port connection

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

# A screenshot of a computer Description automatically generated

Figure 4: Windows IoT image size

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

# Capture a screen shot of the terminal window.

A computer monitor with a blue screen

Description automatically generated

Figure 5: Windows 10 IoT Cmd line screen view.

A computer screen with a blue circle and text

Description automatically generated

Figure 6: Windows 10 IoT Desktop view.

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

A computer monitor with a blue square on it

Description automatically generatedOn 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.

# 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.

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

Raspbian Linux uses the *ext4* file system.

A screenshot of a computer screen

Description automatically generated

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>