November 09, 2024

# Project 2 Module 2

By: Venetia Furtado & James Way

## Table of Contents

# 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".

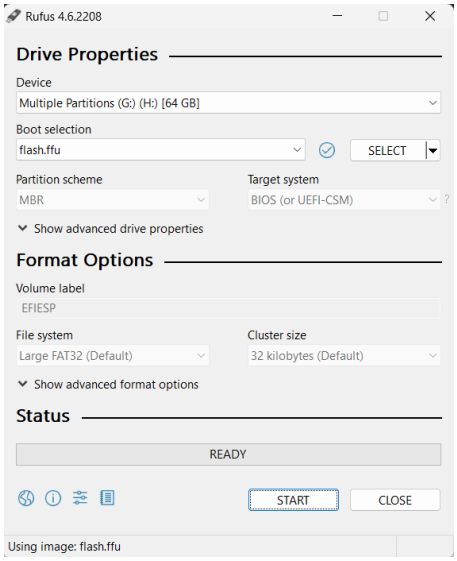


*Figure 1: Rufus GUI - flashing Windows 10 IoT*

# 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.

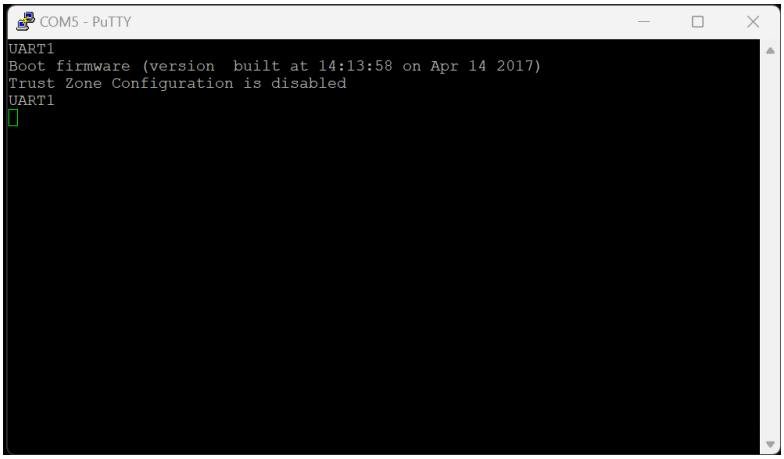


*Figure 2: Serial output over UART1 while Windows 10 IoT is booting.*

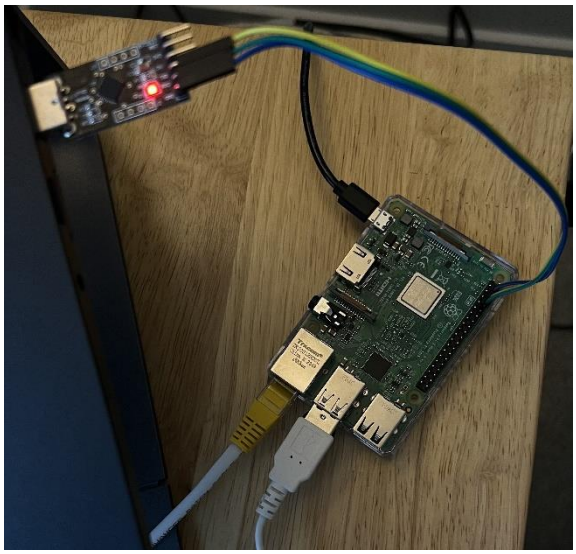

*Figure 3: Serial port connection*

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

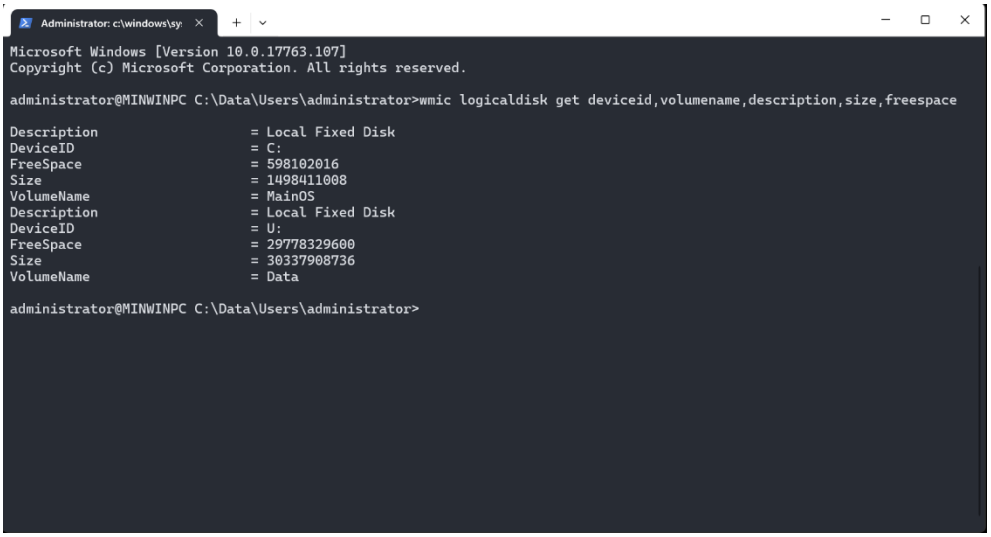


*Figure 4: Windows IoT image size*

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

## 4. Capture a screen shot of the terminal window.

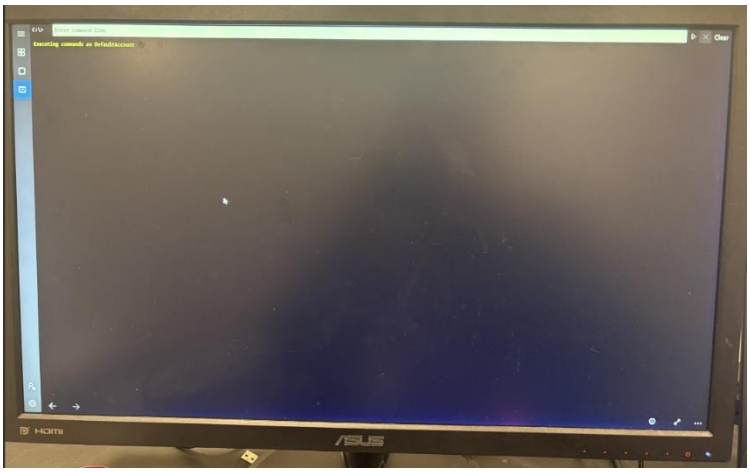

*Figure 5: Windows 10 IoT Cmd line screen view.*

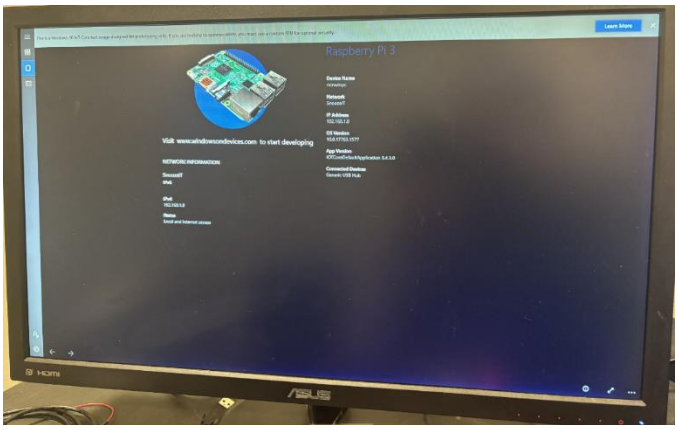


*Figure 6: Windows 10 IoT Desktop view.*

## 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).

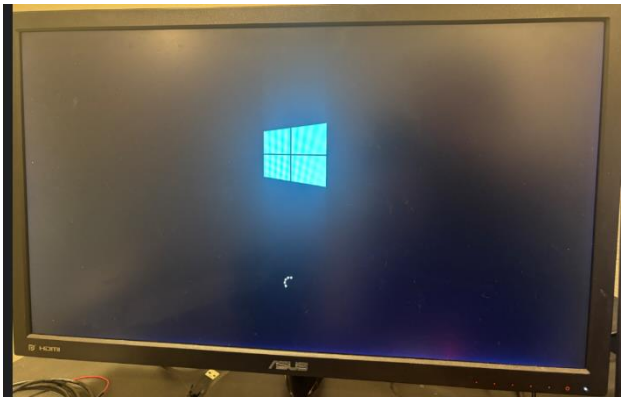


*Figure 7: Windows 10 IoT loading screen on boot.*

## 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