MCU Project using IMU / Microphone Sensor and STM32 Development Board

MCU Project Group: [15]

[Zefan Wu], [u6474528]
[Max Georg Wierse], [u7607296]

ENGN[4213/6213] Digital Systems and Microprocessors
Semester 1, 2023

ANU College of Engineering, Computing and Cybernetics
The Australian National University

Table of Contents

1 Introduction

The purpose of this project was to generate the binaural signal for a given audio file based on a measured azimuth angle. To do that we were given a Microcontroller, MPU-6050 and several other electronic components. Furthermore, the measured angle should be displayed on a provided LCD, the music file should be read from an SD-Card and the resulting music file had to be written on the SD-Card after all computations were done.

The scope of the project was the taught material in the ENGN4213 Digital Systems and Microprocessor course.

Besides generating the binaural signal, the students should get more familiar in working with a Microcontroller and in the usage of “lower level “communication protocols such as I2C, UART and SPI.

In this report we will first look at the general system setup including hardware, wiring and software. After that we go a bit more in to the details of the data processing, binaural signal generation and data handling.

We conclude the report with a discussion of the results and about ways to further optimize the whole system.

2 System Design

2.1 System Setup

The hardware system setup looks like this:

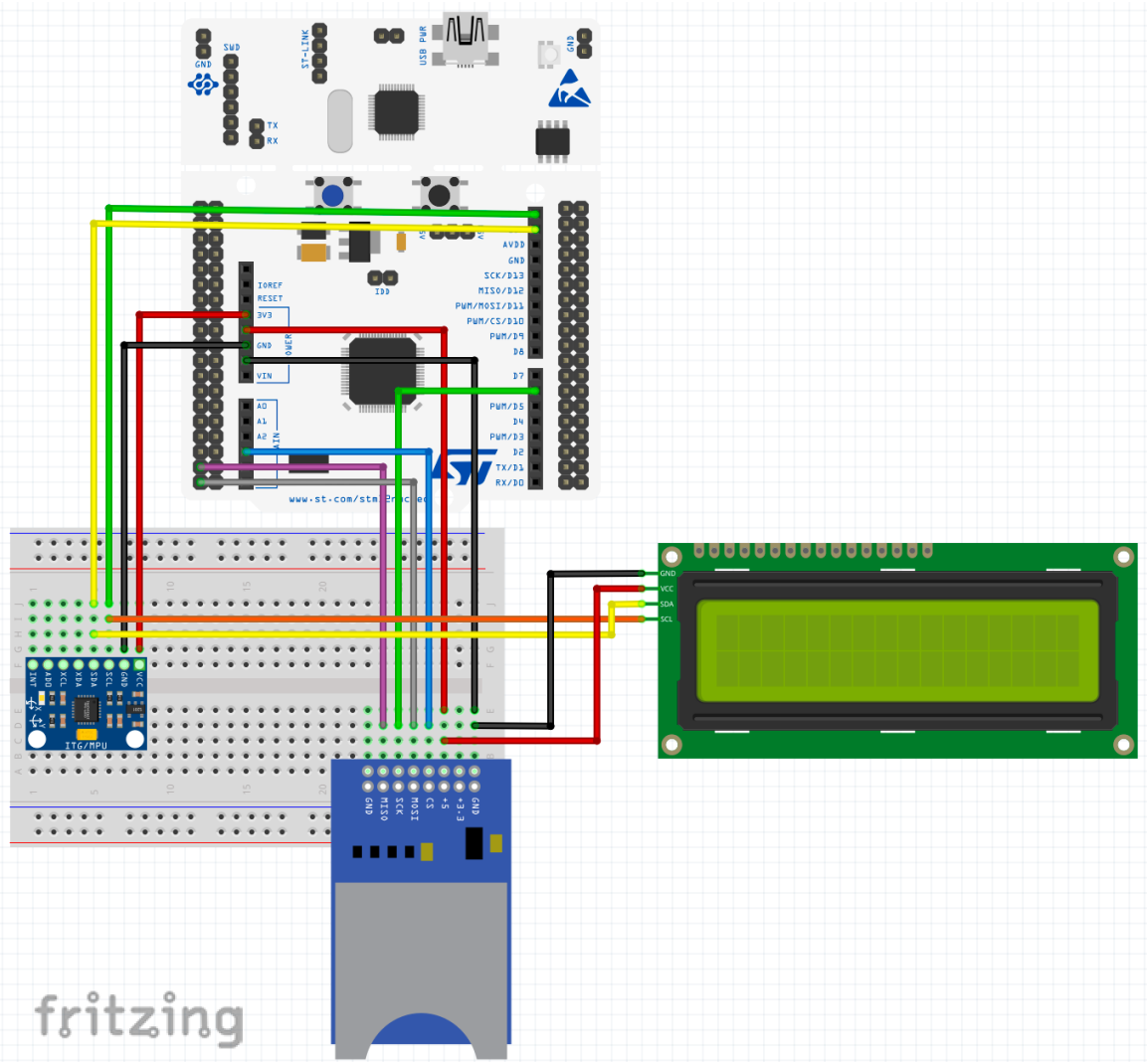

Figure 1: Wiring of the hardware components: Microcontroller (top), IMU bottom left, SD-Card reader (bottom middle) and LCD (bottom right)

The STM32F411RE [6] is used as the microcontroller (MC). This MC takes care of all the necessary computations and of the communication with the other components. The MPU-6050 is used as the Inertial Measurement Unit (IMU). The used SD-Card reader is a "Makerverse MicroSD Card Adapter" [5] . The LCD is the HD44780U from Hitachi [7] in combination with a PCF8574 I/O expander for I2C [8]

For the computation of the binaural signal, we are provided with filter sets corresponding to the left and right audio channel and different degrees (0°-330°; in 30° steps). The binaural signal is computed with a convolution of the filter and the audio file. We used the CMSIS DSP library [9] to compute that efficiently.

2.2 Microprocessor System Design

Since this project uses a lot of different communication protocols (I2C, UART & SPI), many configurations must be made. Since we are provided with a framework to access the SD-Card (more in 2.6), the SPI settings are already set for us. The only non-standard UART configuration is the baud rate which is set to 115200. All our I2C devices work with the standard I2C configuration.

This project will use the blue button as an input to start the computation. To use the button we need to activate the interrupt for this button. Furthermore, we must implement the callback function for that interrupt. This callback function simply sets a flag to true when the button is pressed.

After the system start up the I2C devices are configured (more on that in 2.4 and 2.7), the SD-Card is mounted and the IMU is calibrated (see 2.4).

Once all of that is done the usual "while(true)" loop is entered. In there is the azimuth angle updated (see 2.4) and once the blue button is pressed also the binaural signal generation handled (see 2.5). We also want to transfer the current azimuth angle via UART to the connected computer. This should happen in constant intervals. We chose to implement these constant intervals with an integer that is incremented every loop iteration. Once it is overflown, we will send the data to the connected computer. With that method it is not necessary to implement a timer with an interrupt, but it happens automatically.

2.3 Data Communication

There are multiple components that need to communicate with the MC: IMU, LCD, SD-Card reader, and a connected computer.

For the IMU and the LCD the I2C protocol is used. I2C is a synchronous communication protocol. For the synchronization is a dedicated clock signal transferred. There is no error checking done from us.

The communication with the SD-Card reader is done via SPI. This is also a synchronous communication protocol. The MC and the SD-Card reader are synchronized via a dedicated serial clock.

The communication with the connected computer is done via UART. We configured it with a baud rate of 115200.

2.4 Inertial Measurement Unit – MPU-6050

The IMU is used to measure the angle from which the audio should appear from. There are multiple ways of measuring this angle (also known as yaw or azimuth angle), for example with a magnetometer. Since that was out of scope for this class, we were provided with the MPU-6050. This IMU is a 6-axis IMU and can measure the acceleration in all three dimensions and the angular velocity (also known as gyro) around all three axes.

We configured the IMU to use a sampling rate of 1 kHz, to measure the gyro in a range of

+/- 250°/s and the acceleration in a range of +/- 2g. The used configuration provides the highest accuracy of the measurements.

We used the gyro measurement around the z-axis to compute the azimuth angle:

$$azimuth = \int_0^t gyro_z \, dt$$

The formular was supposed to be only temporary since it will drift over time most certainly. But with the proper calibration in the beginning, the drift was less than 0,1° per second. This seemed precise enough for our task.

The calibration is done at system startup. We simply take 100 measurements and average the $gyro_z$ value. This offset is then always subtracted from the measured value:

$$azimuth = \int_0^t (gyro_z - offset) \, dt$$

The integration is done noncontiguous. The formular for that is:

$$azimuth = azimuth + (gyro_z - offset) * 0.0008$$

Where 0.0008 is the approximated time interval (in seconds) between each calculation. Even though the IMU is configured to read data with a sampling rate of 1 kHz (0.001 s interval), the chosen interval of 0.0008 s produces more accurate results.

For the further course of this project, we are not interested in the exact azimuth angle, but in the next multiple of 30°. That means that if the azimuth angle is 14°, we save 0° in a separate variable and if the angle is 16°, we save 30° in a separate variable. To compute this efficiently we are using some tricks of mathematics and integer rounding:

$$roundedAzimuth = (((int)azimuth + 15)/30) * 30) \; modulo \; 360$$

2.5 Binaural Signal Generation

The binaural signal of a music file is, in simple terms, like playing the piece of music from a certain position relative to the listener. For this project, the distance between the music source and the listener stays constant, but the angle of the sound direction changes.

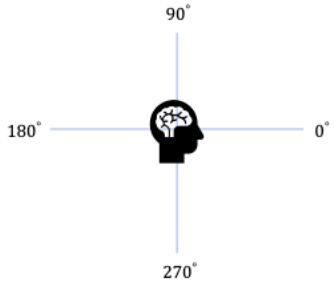

Figure 2: Top view of the angles

To generate such a binaural signal from a music file we can convolute the music file with a certain impulse response (IR). These IRs were provided to us as 2 x 256 matrices and in

steps of 30°, starting from 0° (a total of 12 IRs).

Because the RAM of the MC is limited (128 kB) we decided to convolute each channel (left and right) of the music file one-by-one. To be able to do that, we had to split the provided IRs into two vectors (of size 256). The same must be done for the music file unless it has only one channel. In that case you can simply take this one channel as input for both convolutions.

Furthermore, all the inputs must be converted into text files, containing float or integer numbers. That is because the handling of text files is way easier on the MC than MATLAB or .wav files for example.

Due to the limited RAM of the MC, it is not possible to process very large music files in one go. To circumvent that we read the music files in blocks of 1000 integers at a time. The block size could be bigger to increase the performance, but this was not necessary for our use case.

However, processing the music file block-by-block introduces another problem: The convolution of a certain element of the music vector with the IRs is dependent on its neighbors. To solve this issue, it is possible to either use the “Overlap Add” method, or the “Overlap Save” method [3]

We decided to use the “Overlap Save” method. For that method we save the last 255 (#FilterSize -1) elements of the convoluted data from the previous block and add them to the first 255 elements of the current convoluted block.

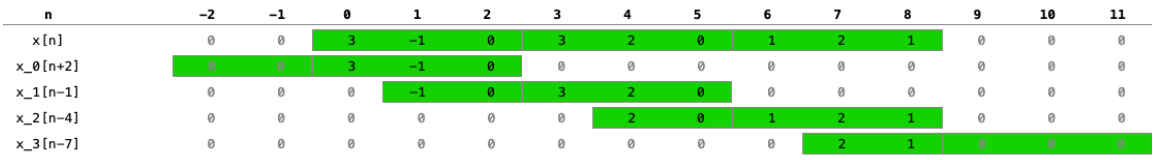

Figure 3: Visualization of the “Overlap Save” method from [3] The two values to the left of each block are saved for the next block. And then after the convolution added.

The convoluted data is then written to the SD-Card.

The whole process of generating a binaural signal looks like this:

1. Choose the appropriate filter based on the rounded azimuth angle.
2. Access the SD-Card block-by-block (see 2.6) and for each block:
   2.1. Perform convolution with the block of music data and the filter using “Convolution Save”.
   2.2. Append the resulting vector to the output file.
3. Repeat the process for the other channel.

2.6 Accessing the SD-Card

The SD-Card is accessed via the provided “Makerverse” SD-Card reader. We were provided with a framework to access the files on this SD-Card. The framework uses the

FatFs [4] library as a basis. We introduced some minor changes to the provided framework to support floating point numbers in files and to read a file on a block-by-block basis.

The access to a converted music file on the SD-Card works the following way:

1. read 10000 Bytes from the file in an input buffer.
2. replace '\n' by ';' (for floating point numbers) or ',' (for integers)
3. insert a '\0' character after the last read number (just for integers)
4. loop over the input buffer, read floats/integers one by one (easy to do with 'scanf' due to the introduced separators) and store them in an output array.

To access a file block-by-block we simply store the pointer to the last number that was read by the previous block and start the next block at that position.

The writing of the binaural signal also happens block by block. After the 'Overlap Save' method is applied, the buffer is appended to the output file.

2.7 Azimuth Angle Display

We implemented two different ways of displaying the azimuth angle.

The first one uses UART to transfer the rounded azimuth angle to MATLAB on the connected computer. There we used polarplot to display the angle in real time.

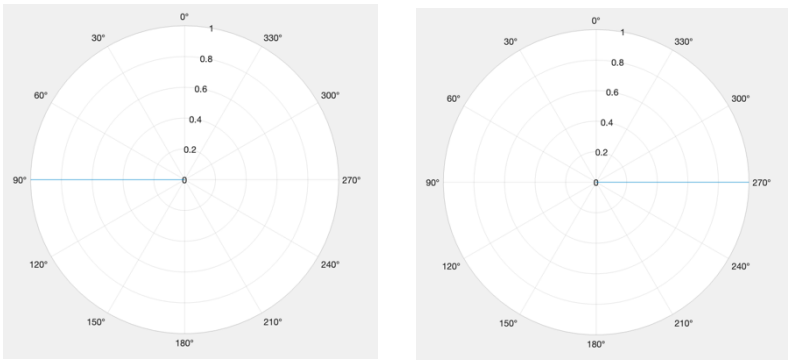

Figure 4: Using polarplot to visualize the angle in real time. Left 90° and right 270°

This is a good way of visualizing the current angle and help selecting the desired angle.

The other option uses the LCD. Whenever the blue button is pressed, the rounded angle at that point in time is displayed on the connected LCD. This value is either removed on reset or updated with the next blue button press. The communication with the LCD was already covered in the lecture. The exact initialization procedure can be seen in Appendix I: .

2.8 System Reset and Calibration

The system reset is implemented using the hardwired black reset button on the MC. After that button is pressed, the system resets all its states and starts executing the program from the beginning. That includes initializing the I2C devices, clearing the LCD, mounting the SD-Card, and calibrating the gyroscope offset. As described earlier, that is done by taking the average of 100 measurements of the z axis gyroscope and then using that offset during

the calculation of the azimuth angle (see 2.4).

The whole process of resetting and recalibrating takes ~4 seconds, which is quite long. That is mainly because of the remounting of the SD-Card and the LCD clearing (1.3 seconds of delay manually added so that it is completely done with clearing; otherwise, it is not always completely clear).

At the same time is all that recalibrating and resetting very important, if the user should always get the same experience.

2.9 Pre- and Postprocessing of Data

The provided IR Filters for the binaural signal generation are in the MATLAB format. The content of these IRs is always a 2x256 Matrix of floating-point numbers. We converted these filters first into two vectors of length 256 and saved them as text files. Each value gets a new line. The 12 provided IRs result in 24 converted text files, two per provided azimuth angle. One of the two files corresponds to the left audio channel, and the other one to the right audio channel.

The music file was provided as a .wav file. We used MATLAB to convert it into a text file. The example music file only contained one audio channel, so we copied the generated text file to have a right and left channel for the binaural signal generation later. If you have a proper stereo file, you must make sure to match the right music channel with the right IR channel and the left music channel with the left IR channel.

After the binaural signal is generated, it is stored as a text file on the SD-Card. To be able to listen to it, we used MATLAB to convert it back in the .wav format.

3 Discussion

3.1 Result Accuracy

From a pure hearing perspective, the generated binaural music files sound like we expected them to sound. I. e., if we generated a music file with 90° the sound came from the left, and when we generated a music file with 270° the sound came from the right.

Keeping that in mind, the difference between 90° and 60° was very hard to hear. Also, the difference between 180° and 0° was not hearable. We do not think that this is due to the accuracy of the computation but more due to the limitations of the binaural signal generation. In the end it is only stored in two sound channels, which does limit the hearing experience. If this would be extended to a more spatial audio experience with for example 16 different channels, there should be more flexibility and precision in the generation (even though this would not solely be a binaural signal generation anymore).

In addition to a “hearing test” you could also analyze the differences in the audio spectrometer of the generated music file. Since we are no experts on this field, we can not give you a detailed analysis on this:

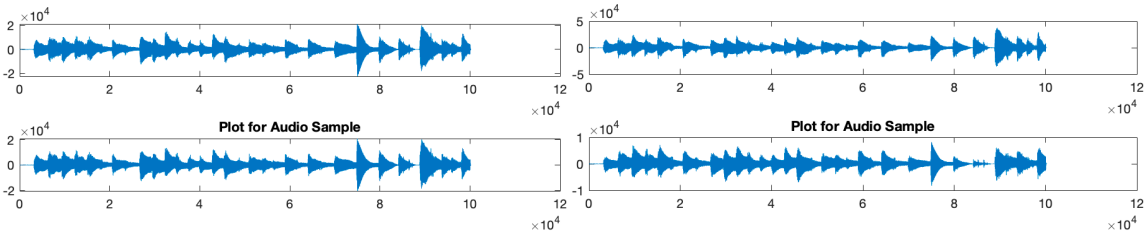

Figure 5: Left shows the volume plot for a binaural music file generated with 0°; right for 90°. Note the different scale on the y-axis.

Worth noting is also, that the filters are processed as floating-point numbers, but the music in- and output files are only stored as integers. This was done due to RAM limitations in reading and writing the files. Since the integers are in the orders of hundreds this shouldn't make a big difference, but it costs accuracy.

3.2 Improvements

During the whole implementation process I was thinking about improving the data transfer from the music file to the convolution. The proposed way was to convert the .wav music file in a .txt file. This .txt file contained then a bunch of floating-point or integer values. On the MC these files were then read byte-by-byte just to convert them from string back to floating-point/integers. I am very sure that there is a better way of doing this, especially in terms of RAM efficiency. If the block size could be increased by this, this would decrease the number of SD-Card accesses and with that the number of blocks that must be processed.

One way to implement that would be to directly store the raw bytes of integers/floats in a file. This would not be human readable anymore but should reduce the memory requirements on the MC side.

4 Conclusion

This project included a lot of different components that needed to be orchestrated by the MC with different communication protocols. This project was very well fitted to the content that was taught in the ENGN4213 course, which gave a good environment to gain hands on experience with microcontrollers. Furthermore, this project allowed a small glimpse in the world of sound design/sound engineering.

Possible future work could try to generate spatial audio and not just binaural audio. It would also be interesting to build a microphone array that can generate the IR filters that are needed for the signal generation.

5 References

[1] J. Wakerly, *Digital Design: Principles and Practices*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1994.

[2] S. Kilts, *Advanced FPGA Design: Architecture, Implementation, and Optimization*. Hoboken, NJ, USA: Wiley, 2007.

[3] R. Elder, *Overlap Add, Overlap Save Visual Explanation.* https://blog.robertelder.org/overlap-add-overlap-save/, 2018

[4] *FatFs – Generic FAT Filesystem Module*. http://elm-chan.org/fsw/ff/00index_e.html

[5] *Makerverse MicroSD Card Adapter.* https://core-electronics.com.au/micro-sd-card-module-3-3-and-5v-compatible.html

[6] STMicroelectronics, *STM32F411RE* https://www.st.com/en/microcontrollers-microprocessors/stm32f411re.html

[7] Hitashi, *HD44780U Datasheet.* https://www.sparkfun.com/datasheets/LCD/HD44780.pdf

[8] Texas Instruments, *PCF8574 Remote 8-Bit I/O Expander for I2C Bus,* https://www.ti.com/lit/ds/symlink/pcf8574.pdf

[9] *CMSIS-DSP library.* https://www.keil.com/pack/doc/CMSIS/DSP/html/group__Conv.html

6 Appendix I: LCD Initialization

HD44780U

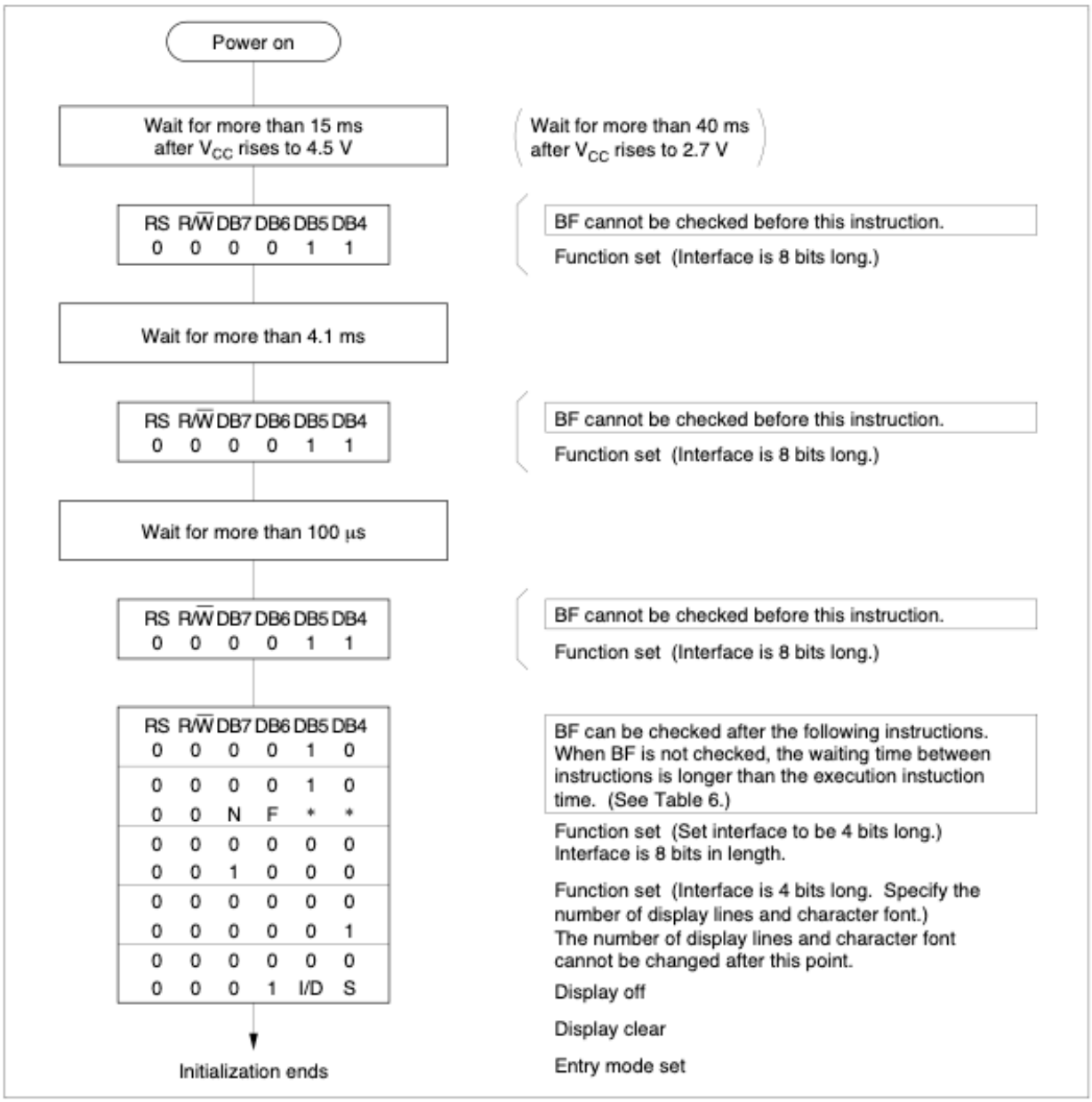

Figure 24 4-Bit Interface

Source: [7] page 46