Comparison and Implementation of Data Transmission Techniques Through Analog Audio Signals in the Context of Augmented Mobile Instruments

Romain Michon,[1,2] **Yann Orlarey,**[1] **Stéphane Letz,**[1] **and Dominique Fober**[1]
[1] GRAME-CNCMC, 11 Cours de Verdun-Gensoul, 69002 Lyon (France)
[2] Center for Computer Research in Music and Acoustics, 660 Lomita Ct., Stanford CA 94350-8180 (USA)
rmichon@ccrma.stanford.edu

ABSTRACT

Augmented mobile instruments combine digitally-fabricated elements, sensors, and smartphones to create novel musical instruments. Communication between the sensors and the smartphone can be challenging as there doesn't exist a universal lightweight way to connect external elements to this type of device. In this paper, we investigate the use of two techniques to transmit sensor data through the built-in audio jack input of a smartphone: digital data transmission using the Bell 202 signaling technique, and analog signal transmission using digital amplitude modulation and demodulation with Goertzel filters. We also introduce tools to implement such systems using the FAUST programming language and the Teensy development board.

1. INTRODUCTION

For about a decade, smartphones have been used as musical instruments [1, 2]. The fact that they combine in a single entity various sensors (e.g., accelerometers, gyroscope, touch screen, etc.), a speaker, a microphone, a battery, an Analog to Digital Converter (ADC)/Digital to Analog Converter (DAC), and a powerful processor that can be used for sound synthesis/processing make them a great platform to implement standalone Digital Musical Instruments (DMIs). However, smartphones were never designed to be used as such and they lack some crucial elements to compete with their acoustic counterparts. In previous works, we tried to solve this problem by augmenting smartphones with passive [3] and active [4] elements. While passive augmentations consist of purely acoustic elements free from electronics, active augmentations typically combine sensors, a microcontroller, and some sort of casing.

Transmitting sensor data between the microcontroller and the mobile device is often a source of problems and there currently doesn't exist a standard and comprehensive way to do this.

MIDI is commonly used for this task as it is the only communication standard supported by all smartphones. Indeed, most external devices must be approved by Apple before they can be connected to an iPhone/iPad, but that's not the case of MIDI devices.

MIDI can be transmitted over USB or via Bluetooth. USB requires the use of a USB adapter in most cases and Bluetooth implies more complex circuitry and significantly increases the price of the augmentation.

Active augmentations also often involve the use of an external speaker/amplifier (i.e., the built-in speakers of smartphones are often weak and low quality) that needs to be connected to the audio jack of the smartphone. In that case, two cables must be plugged to the smartphone (i.e., one USB for the microconctroller and one audio cable) which is far from being an optimal solution (not to mention that more and more smartphones don't have a built-in audio jack). An alternative solution to this is to use an external USB ADC, which requires complex multiplexing operations with the micocontroller since the same USB port has to be used.

In this paper, [1] we present a simple lightweight solution to this problem where sensor data is transmitted to the smartphone using its audio input on its four pins audio jack (devices without a built-in audio jack can use an adapter that would be needed to retrieve the output audio signal anyway). Two techniques using respectively digital or analog data are considered:

- digital data transmission using the Bell 202 signaling technique (i.e., modem),
- analog signal transmission using digital amplitude modulation and demodulation with Goertzel filters.

The Bell 202 approach has been commonly used for transmitting digital data from an external device to a smartphone through its audio jack input. The Square Credit Card Reader [2] and the system presented by Kuo et al. [5] (to only cite a few) all use this technique. On the other hand, to the best of our knowledge, digital amplitude modulation has never been used in this context.

Many microcontrollers such as the ARM Cortex-M4 [3] used on the Teensy development board series [4] host their

[1] Demos and additional information about this project can be found at this URL: https://ccrma.stanford.edu/~rmichon/analog-transmit.
[2] https://squareup.com/us/en/hardware/reader
[3] https://developer.arm.com/products/processors/cortex-m/cortex-m4
[4] https://www.pjrc.com/teensy

Figure 1. Smartphone augmentation connection with a 4 pins audio jack.

own Pulse Width Modulation (PWM) DAC. Hence, they can be utilized to synthesize sound and play it back in real-time. [5] In a companion paper [6], we introduce a system where the FAUST programming language [7] can be used to program Digital Signal Processing (DSP) algorithms for the Teensy.

In the systems presented in the current paper, the Teensy DAC is used to transmit data to the smartphone. Both the Bell 202 and the Amplitude Modulation (AM) transmission techniques are implemented, evaluated, and compared.

2. HARDWARE

All the systems presented in this paper are based on the same hardware set-up which consists in a smartphone augmentation [4] connected to a smartphone through a 4 pins audio jack (see Figure 1). The two upper pins are used to carry the left and right audio channels out of the smartphone. The third pin (from the tip) is the ground, and the fourth pin (from the tip) carries sensor data from the microcontroller to the smartphone. The impedance between the fourth pin and the ground is 700Ω. This is important because the smartphone uses this as the trigger to activate its line input instead of using its built-in microphone. This value is also used as a reference to configure the gain of the built-in preamp.

We used a Teensy 3.2 in our system. It is based on an ARM Cortex-M4 microcontroller which hosts its own 12 bits PWM DAC running at 44.1KHz by default. The lack of reconstruction filter is compensated by the use of a $10\mu F$ capacitor connected in series between the DAC output and the fourth pin of the audio jack (see Figures 4-5). While this would not be sufficient to render a good quality audio signal, this is more than acceptable for the type of use that we make of it (see Section. 3-4).

3. BELL 202 SIGNALING TECHNIQUE APPROACH

The Bell 202 signaling technique allows for the serial transmission of bits at a maximum rate of 1200 baud. It uses Frequency Shift Keying (FSK) where "digital zeros" are represented by 1200 Hz tones and "digital ones" by 2200 Hz tones. Tones are typically synthesized using a square wave, so a simple digital output is theoretically sufficient to generate the corresponding analog audio signal. Instead, we preferred to use the built-in DAC of the Cortex-M4 microcontroller in order to implement a comprehensive solution only using FAUST.

[5] https://www.pjrc.com/teensy/td_libs_Audio.html

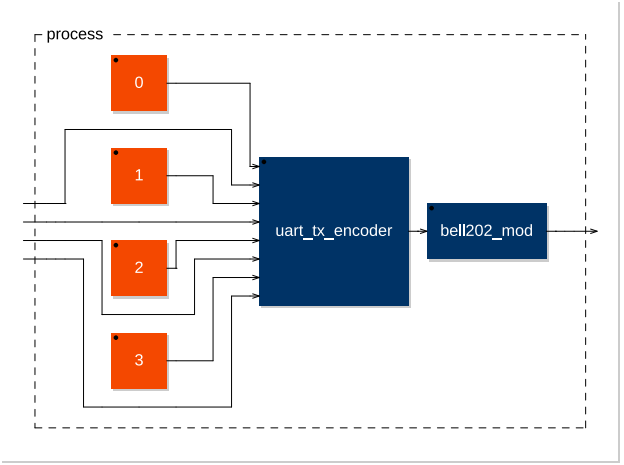

Figure 2. FAUST-generated block diagram of a program encoding four streams of data using the Bell 202 signaling technique.

3.1 Transmitting Data

The `bell202_mod` FAUST function [6] takes a stream of bits coded on audio samples and encodes it using the Bell 202 technique (essentially, the value of each bit determines/-modulates the frequency of the generated square wave).

Continuous sensor values (e.g., retrieved by an analog input on the Teensy) or any other type of data can be converted to 9 bits Universal Asynchronous Receiver/Transmitter (UART) packets coded on a stream of audio samples using the `uart_tx_encoder` function (see Figure 4). `uart_tx_encoder` takes three arguments: the number of parallel streams (channels) of data to be sent, a list indicating the channel number of each stream, and a value indicating if data should be transmitted or not (one for true, zero for false). For example, the following FAUST program will send four streams of data on channels 0, 1, 2, and 3:

```
import("stdfaust.lib");
process = cm.uart_tx_encoder(4,(0,1,2,3),1)
     : cm.bell202_mod;
```

The block diagram corresponding to this FAUST program can be seen in Figure 2.

UART packets produced by `uart_tx_encoder` contain a 7 bits value, a start bit, and a stop bit. Since the channel number needs to be sent along with the corresponding value, a single value (i.e., sensor) requires a total of 18bits (two full UART packets: one for the channel number and one for the value). If the data transmission parameter of `uart_tx_encoder` (third argument) is set to false, then a stream of "one bit" is produced (that's a standard of the UART protocol). This might be used for the potential synchronization of the sender device (i.e., Teensy) with the receiver (i.e., smartphone) in case it is plugged to it after the receiver program (i.e., app) was launched.

[6] All the FAUST functions presented in this paper have been added to the `communications.lib` library that can be found in the FAUST libraries repository: https://github.com/grame-cncm/faustlibraries.

Since bits are encoded at a rate of 1200 baud by `bell202_mod`, the bit stream produced by `uart_tx_encoder` is synchronized to that rate and several audio samples will likely contain the same bit value, depending on the audio sampling rate of the system. `uart_tx` is a function that automatically associates FAUST User Interface (UI) elements to streams of values transmitted with `uart_tx_encoder`. Its single argument is the number of parallel stream of data (channels) to transmit. For example:

```
import("stdfaust.lib");
process = cm.uart_tx_encoder(4) : cm.
    bell202_mod;
```

sends 4 parallel streams that can be addressed on the Teensy side using the `setParamValue` method of the corresponding object generated with `faust2teensy` or `faust2api` [6]. Hence, the loop function on the Teensy could look like:

```
void loop() {
  int val0 = analogRead(A0)*127/1024;
  int val1 = analogRead(A1)*127/1024;
  int val2 = analogRead(A2)*127/1024;
  int val3 = analogRead(A3)*127/1024;
  faust.setParamValue("0",val0);
  faust.setParamValue("1",val1);
  faust.setParamValue("2",val2);
  faust.setParamValue("3",val3);
}
```

where `faust` is a FAUST object produced with `faust2api` [6], and the first argument of the `setParamValue` method the FAUST parameter name automatically generated by the `uart_tx` function corresponding to the channel number on which the value should be transmitted.

Note that it is also possible to write a FAUST program to carry out the same task without writing a single line of Arduino code using `faust2teensy` [6]. In that case, analog and digital inputs of the Teensy can be mapped to FAUST UI elements using metadata:

```
val0 = nentry("val0[io: A0]",0,0,127,1);
process = val0 :
  cm.uart_tx_encoder(1,(0),1) :
  cm.bell202_mod;
```

3.2 Receiving Data

Data transmitted by the microcontroller using the technique presented in Section. 3.1 can be decoded directly in the FAUST program implementing the app running on the smartphone (see Figure 4). This app was generated with `faust2smartkeyb` [8]. The `bell202_demod` function can be used to decode the signal produced by `bell202_mod` on the Teensy to turn it into a stream of bits encoded on a digital audio signal. The decoding algorithm uses zero crossing detection and cross-corelation. `bell202_mod` and `bell202_demod` are configured to have the same baud so they don't need to be parametrized.

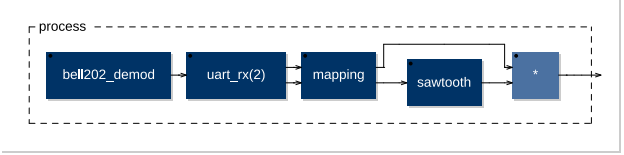

Figure 3. FAUST-generated block diagram of a program decoding two streams of data using the Bell 202 signaling technique.

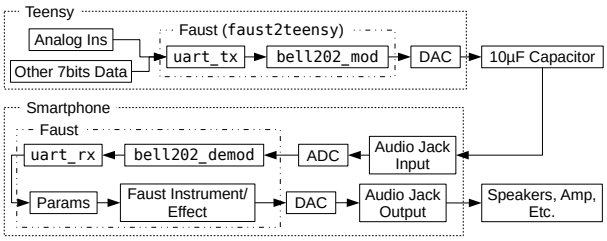

Figure 4. Analog audio sensor data transmission between a Teensy and a smartphone using the Bell 202 Signaling Technique.

The `uart_rx` function takes a single argument allowing us to specify the number of channels to be extracted from the input bit stream. This number should be the same as the one used with `uart_tx_encode`. `uart_rx` outputs audio signals (one per channel) containing the transmitted data for each individual channel. These signals (whose range is 0-127) can be used directly to control some sound synthesis/processing parameter. In the following example, two sensor data streams are decoded and used to control the gain and the frequency of a sawtooth wave oscillator:

```
import("stdfaust.lib");
mapping = /(127),(/(127)*1900 + 100);
process =
  cm.bell202_demod : cm.uart_rx(2) :
  mapping : *(os.sawtooth);
```

4. DIGITAL AMPLITUDE MODULATION APPROACH

This other method consists in carrying continuous sensor signals on AM bands to the smartphone.

4.1 Transmitting Data

`am_tx_encoder` is a FAUST function working in a similar way than the combination of `uart_tx_encoder` and `bell202_mod` (see Section. 3.1). Its first argument configures the number of parallel streams to be transmitted and its second argument is a list of channel numbers corresponding to each data stream input. For example, the following FAUST program will send four streams of data on channels 0, 1, 2, and 3:

```
import("stdfaust.lib");
process = cm.am_tx_encoder(4,(0,1,2,3));
```

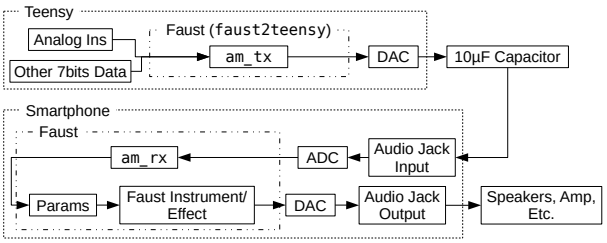

Figure 5. Analog audio sensor data transmission between a Teensy and a smartphone using Amplitude Modulation.

One sine wave oscillator is used for each channel. Their gain is modulated by individual sensor data. The frequency of each oscillator is determined by the number of bands. They are distributed between 50 and 20000 Hz (once again, the sampling rate of the DAC on the Teensy is 44.1 KHz). Of course, the gain of each sine oscillator is scaled in function of the number of bands, which means that more bands means less dynamic range. In addition to that, a calibration sine tone with maximum gain is constantly transmitted at 20 KHz to normalize the input signal on the smartphone side in real-time. Hence, if four sensor streams are transmitted, five sine tones will be generated.

`am_tx` works the same way than `uart_tx` and takes a single argument which is the number of channels to be transmitted. User interface elements are automatically generated by this function and can be addressed on the Teensy side using the `setParamValue` method (see Section. 3.1).

4.2 Receiving Data

Data encoded by the `am_tx` function on the Teensy can be decoded using `am_rx` on the smartphone. This function takes a single argument which corresponds to the number of data streams to be decoded/demodulated. This number should be the same as the one used with `am_tx` on the Teensy. `am_rx` outputs audio signals (one per channel) containing the transmitted data for each individual channel. These signals (whose range is 0-127) can be used directly to control some sound synthesis/processing parameter. In the following example, two sensor data streams are decoded and used to control the gain and the frequency of a sawtooth wave oscillator:

```
import("stdfaust.lib");
mapping = /(127),(/(127)*1900 + 100);
process =
  cm.am_rx(2) : mapping : *(os.sawtooth);
```

`am_rx` uses Goertzel filters [9] to efficiently extract the amplitude of each band. The filters block/window size is automatically adapted in function of the number of bands to be decoded. A greater block size will provide more precision in the frequency domain but will add more latency. Hence, more bands means more delay (see Section. 5).

Other demodulation techniques were tested (e.g., bandpass filters, etc.) but Goertzel filters provided the best results in this context.

5. EVALUATION

Table 1 compares the performance of the data transmission techniques presented in Section. 3 and Section. 4 for various numbers of parallel channels. Latency and bit depth are the two main parameters that are considered.

A single channel transmitted with the 202 technique corresponds to a latency of $\sim$15ms (18 bits are required to transmit one value at a bit rate of 1200 bits/s so $\frac{1}{(1200/18)}$). Hence, every new channel will add a latency of $\sim$15ms. On the other hand, latency when using the AM technique is determined by the block size of the Goertzel filter which is automatically computed by `am_tx_encoder` in function of the number of channels (see Section. 4.2). Note that overall, this method provides much better latency performances than the 202 approach.

The bit depth of the data transmitted with the 202 technique is constant (7 bits by default) and can be decided by the programmer. A greater bit depth means more precision but will also add more latency. For the AM technique, the precision/range of the data depends on the number of channels to be transmitted. Since the built-in DAC of the Cortex-M4 can produce 12 bits values, the range of data when transmitting a single channel is 2048 (2^{12}/2 where the division by two corresponds to the number of carriers: here one for the data and one for the calibration signal). In practice, the range of the data for the AM technique is probably slightly smaller because of potential noise in the signal, but this type of parameter is hard to measure efficiently.

The Goertzel filter demodulation approach used with the AM transmission technique has proven very effective. Also, since the carriers are modulated at a rate of approximately 440 Hz, sidebands are not an issue as long as the frequency of each carrier is more than 880 Hz appart (i.e., if more than 20 channels were needed this rate could to be lowered, etc.).

All in all, both techniques present advantages. The 202 approach is obviously more reliable but it is also less powerful than the AM method, especially when a large number of parallel streams of data must be transmitted. It is also technically more complex to implement.

6. CONCLUSIONS

The built-in analog audio input of smartphones provides a convenient and standard way to acquire sensor data from a microcontroller to control sound synthesis/processing parameters. This is very helpful in the context of active smartphone augmentations where "prosthetics" are mounted on the device to expand its affordances.

In this paper, two transmission techniques usable in this context as well as their associated tools were presented and compared. They seamlessly integrate to the existing panoply of FAUST-based tools to create musical instruments with augmented smartphones.

We believe that this approach solves various issues by providing a standard universal way to connect to smartphones and by offering better performances than other standards such as USB/Bluetooth MIDI, etc.

N Channels	202 Latency	AM Latency	AM Goertzel Block Size	202 Range	AM Range
1	∼15ms	∼6ms	256	128 (7 bits)	2048
2	∼30ms	∼6ms	256	128 (7 bits)	1365
3	∼45ms	∼11ms	512	128 (7 bits)	1024
4	∼60ms	∼11ms	512	128 (7 bits)	818
5	∼75ms	∼23ms	1024	128 (7 bits)	683
10	∼150ms	∼23ms	1024	128 (7 bits)	372
15	∼225ms	∼46ms	2048	128 (7 bits)	256
20	∼300ms	∼46ms	2048	128 (7 bits)	204

Table 1. Comparison of the Bell 202 signaling technique with the amplitude modulation transmission approach.

7. REFERENCES

[1] G. Essl and M. Rohs, "Interactivity for mobile music-making," *Organised Sound*, vol. 14, no. 2, pp. 197–207, 2009.

[2] G. Wang, "Ocarina: Designing the iPhone's Magic Flute," *Computer Music Journal*, vol. 38, no. 2, pp. 8–21, Summer 2014.

[3] R. Michon, J. Smith, M. Wright, C. Chafe, J. Granzow, and G. Wang, "Passively augmenting mobile devices towards hybrid musical instrument design," in *Proceedings on the New Interfaces for Musical Expression Conference (NIME-17)*, Copenhagen, Denmark, May 2017.

[4] R. Michon, J. O. Smith, M. Wright, C. Chafe, J. Granzow, and G. Wang, "Mobile music, sensors, physical modeling, and digital fabrication: Articulating the augmented mobile instrument," *Applied Sciences*, vol. 7, no. 12, p. 1311, 2017.

[5] Y.-S. Kuo, T. Schmid, and P. Dutta, "Hijacking power and bandwidth from the mobile phone's audio interface," in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, Austin, Texas, 2010.

[6] R. Michon, Y. Orlarey, S. Letz, and D. Fober, "Real time audio digital signal processing with Faust and the Teensy," in *Proceedings of the Sound and Music Computing Conference (SMC-19)*, Malaga, Spain, 2019.

[7] Y. Orlarey, S. Letz, and D. Fober, *New Computational Paradigms for Computer Music*. Paris, France: Delatour, 2009, ch. "Faust: an Efficient Functional Approach to DSP Programming".

[8] R. Michon, J. Smith, C. Chafe, G. Wang, and M. Wright, "faust2smartkeyb: a tool to make mobile instruments focusing on skills transfer in the Faust programming language," in *Proceedins of the International Faust Conference (IFC-18)*, Mainz, Germany, July 2018.

[9] R. Beck, A. G. Dempster, and I. Kale, "Finite-precision Goertzel filters used for signal tone detection," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 48, no. 7, pp. 691–700, 2001.