Performance Analysis of OQ-PSK in Backscatter Communication - An Empirical Approach

Alexander Verdieck
Uppsala University
Uppsala, Sweden

Darius Loga
Uppsala University
Uppsala, Sweden

Ingmar Falk
Uppsala University
Uppsala, Sweden

Abstract

This project investigates the performance of Offset Quadrature Phase Shift Keying (OQ-PSK) in backscatter communication. Using a Raspberry Pi Pico as the tag, the effects of different offsets to the carrier signal, waveform repetition and clock frequency on the Bit Error Rate (BER) and Received Signal Strength Indicator (RSSI) have been evaluated. The experiments are done in the 2.4 GHz band using two different physical setups, that revealed optimal BER and RSSI trade-offs depending on clock frequency and waveform design. The findings suggest a clock frequency 128 MHz offers stable RSSI, while the BER benefits from varied configurations. The transmission also benefited from a lower offset, which is based on the waveform design and clock frequency.

Keywords: Wireless Communcation, Backscatter, OQ-PSK

1 Introduction

Offset Quadrature Phase Shift Keying (OQ-PSK) is an enhancement to the classical Q-PSK. It introduces an offset between the in-phase and quadrature components of the signal. This helps to avoid 180° degree turns in the modulation. And therefor can reduce the inherent errors in certain symbol changes. This makes it very useful in wireless communication, at the cost of some additional complexity in the system design.

In backscatter communication a carrier signal is used to transmit data from a tag to some receiver. This is particularly effective in scenarios, where the tag itself has a limited energy supply. Between the carrier signal and the transmission is an offset. A higher offset only allows to use less of the carrier signal, a lower offset results in a higher self interference.

This project is based on the work of an earlier project that implemented the OQ-PSK on the Raspberry Pi Pico [2].

2 Methodology

In order to test different offsets and other parameters the code has to be modified to use less hard coded values. First, however it is important to understand what has to be changed and the consequences of these changes affect the hardware and transmission.

The central component that has to be changed is the generation of the waveform from the sender. In general, OQ-PSK uses four different waveforms to encode two symbols at a time.

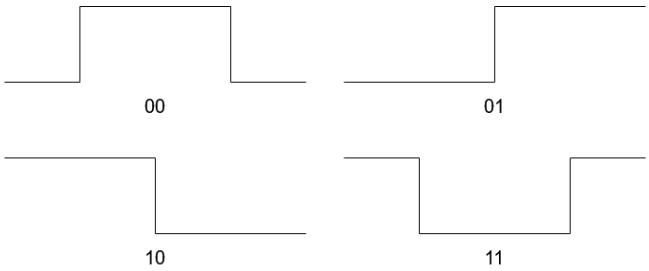

Figure 1: Overview of the waveforms

One waveform itself can be repeated multiple times, but it is also possible to send one waveform for more clock cycles or a combination of both. To send a wave for more clock cycles, the wave has to be divided further down into four equally long subsections. For the 00-wave that would be one low part, two high parts and one low part. Each subsection has to be at least 4 clock cycles long in order to successfully send data. But the subsection can be extended in $2 \times x$ steps, where x is some number ≥ 0. The clock cycles per waveform (CPW) can be calculated as follows:

$$\mathrm{CPW} = 4 \times (4 + 2 \times x)$$

2.1 Adjusting the clock frequency

Changing both repetitions of waveforms (RPW) and CPW have direct implications on the clock frequency that has to be used on the Raspberry Pi Pico. In the end, transmission time per symbol has to be the same as the standard of OQ-PSK specified by the IEEE 802.15.4 standard [1] (t_{sym} = 500ns). The total number of clock cycles for one symbol can be divided by the symbol time to calculate the required clock frequency for the Pico:

$$f_{\mathrm{Clock}} = \frac{\mathrm{RPW} * \mathrm{CPW}}{t_{\mathrm{sym}}}$$

To adjust the actual clock frequency of the Raspberry Pi Pico, the crystal oscillator is used. The crystal oscillator operates on a frequency of 12 MHz. In this project, the base frequency is multiplied by 128 and then divided down with two post dividers to the targeted clock frequency of the Pico. This approach allows for accurate setting of the clock frequency, with the downside of not being able to reach all clock frequencies. This is problematic with some of the smaller clock frequencies. It has also been noted that not all clock frequencies are suitable on the Pico as it has an upper limit. Currently, the Pico clock is certified to operate until 200 MHz and in some of the experiments the Pico has been

slightly overclocked which worked quite well for 256 MHz, but nothing higher.

2.2 Calculation of the Offset

The offset between the carrier signal and the receiver is dependent on the frequency of the clock in MHz and the clock cycles per waveform and can be calculated as follows:

$$\text{Offset} = \frac{f_{\text{Clock}}}{\text{CPW}}$$

An interesting observation, that can be made, is that the clock frequency already is depended on the CPW and therefore the formula can be simplified to:

$$\text{Offset} = \frac{\text{RPW}}{500\text{ns}}$$

This results in a correlation of the offset and the number of repetitions of a waveform. Zero repetitions would not change the offset at all. Each repetition of the waveform adds 2 MHz offset to the carrier signal. The CPW does not have any effect on the offset, despite changing the used clock frequency of the Pico board.

3 Experiments

The experiments were carried out using:

- Zolertia Firefly [Carrier]
- Raspberry Pi Pico [Tag]
- Texas Instruments (TI) LaunchPad [Receiver]

Two different physical setups have been used, differing both in transmission distance and relative hardware placement:

- D1 - distance Carrier-Tag
- D2 - distance Tag-Receiver

The first experimental setup was done with D1 = very small ($<$ 5cm) and D2 = 50cm. The second setup used D1 = 150cm and D2 = very small ($<$ 5cm). Each individual experimental run measured 100 received packages and was repeated five times with measured values being averaged for the results. All experiments were done in the 2.4GHz band. Due to hardware constraints the receiver was fixed at 2460MHz and the carrier was set to different frequencies based on the calculated offset.

3.1 Results

The team explored three variables to determine if improvements could be achieved in terms of better RSSI (Received Signal Strength Indicator) and lower BER (Bit Error Rate) by evaluating the associated trade-offs.

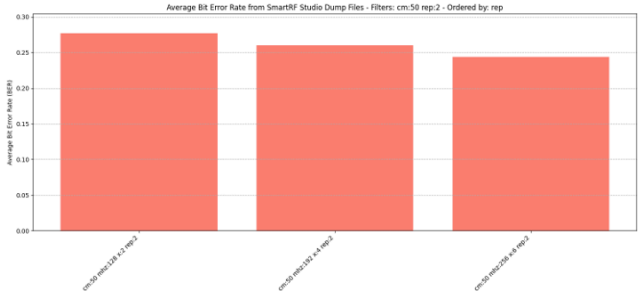

Figure 2: BER of the same repetition values, different *x* value

This is done for the first setup. For the same repetitions, the higher the tested *x* value, the better the BER value - linear scaling. Tested also for lower values - *x = 0* was just junk data and *x = 1* had the worst BER out of those measured values. The team conducted a variation of this experiment in which only the repetition variable was changed *(x = 0)* - the best value was at 128 MHz *(reps = 4)*.

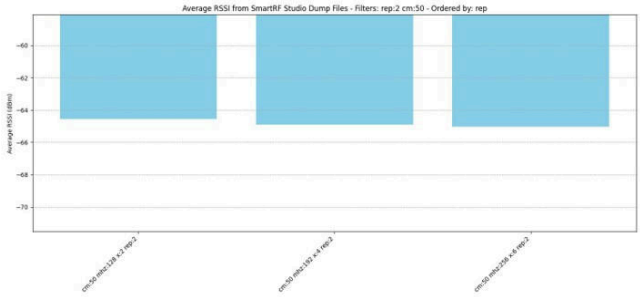

Figure 3: RSSI of the same repetition values, different *x* value

This is done for the first setup. For the same repetitions, the higher the tested *x* value, the worst the RSSI value - indirect dependency with the BER value - still linear scaling. Tested also for lower values - *x = 0* was just junk data and *x = 1* had slightly better RSSI out of those measured values. The team conducted a variation of this experiment in which only the repetition variable was changed *(x = 0)* - the best value was still at 128 MHz *(reps = 4)*.

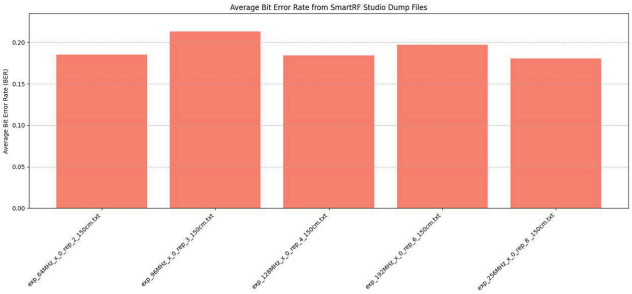

Figure 4: BER of the same repetition values, different *x* value

This is done for the second setup. For the same *x = 0*, only the repetitions were changed, resulting in a mixture of the BER values - not a linear scaling, certain frequencies had a better BER than others. Tested also for lower values - *rep = 1* and *rep = 7*, but it was just junk data and post dividers with no integer result. The team conducted a variation of this experiment in which only the *x* value was changed and *reps = 2*, but almost ended up with the same values, thus the same chart representation - the best value was at 256 MHz *(x = 6)*.

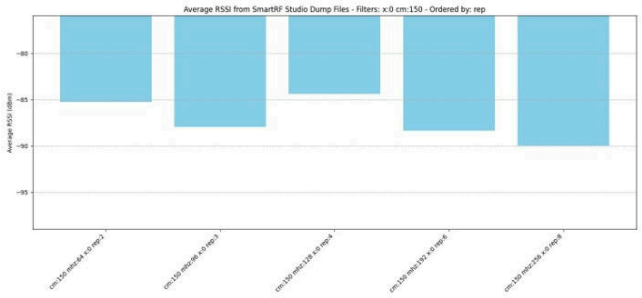

Figure 5: RSSI of the same repetition values, different x value

This is done for the second setup. For the same $x = 0$, only the repetitions were changed, resulting in a mixture of the RSSI values - not a linear scaling, certain frequencies had a better RSSI than others. Tested also for lower values - *rep = 1* and *rep = 7*, but it was just junk data and post dividers with no integer result. The team conducted a variation of this experiment in which only the x value was changed and *reps = 2*, but almost ended up with the same values, thus the same chart representation - the best value was at 64 MHz *(x = 0)*.

4 Conclusion

The project faced some initial hardware problems, especially a broken carrier and other issues with the debugging of the code on the tag and the receiving of packets.
By choosing this approach, of conducting experiments and trying different configurations to improve the two base metrics: RSSI and BER, some observation have been made. For the RSSI, the clock frequency of the Pico board it should remain as the base experiment (128 MHz) for better results, independent of the x and repetitions, but for the BER, other clock frequencies can be used, but the CPW and RPW should remain at a lower level, therefore a lower offset.

5 Future Work

As for future work some improvements in the experiments and testing phase can be made. For example more different distances and physical setups, not to have only two specific distinct sets. It would also be good to measure different metrics, not only RSSI and BER, but also the PER (Packet Error Rate), Packet Lost and maybe SNR (Signal to Noise Ratio). Also to have different settings (angle of the antenna, multiple antennas, outdoor open field etc.) to see if there is any major or minor influence with the experiments.
The usage of different clock frequencies could also lead to a lower power consumption. Which could be and very interesting new measurement matrix. In the current implementation the CPW and RPW have to be hard coded. It would be interesting to test a dynamic system that allows for a change of these parameters during an experiment, based on the current performance of the transmission.

References

[1] . 2020. IEEE Standard for Low-Rate Wireless Networks. *IEEE Std 802.15.4-2020 (Revision of IEEE Std 802.15.4-2015)* 0, (2020), 1–800. https://doi.org/10.1109/IEEESTD.2020.9144691

[2] Caden Keese, Anna Seiderer, and Siva Shankar Siva Saravanan. 2024. O-QPSK Implementation for Backscatter Communication - Analytical Considerations. *WCNES Project 2024* (2024).