# O-QPSK Implementation for Backscatter Communication
# **Analytical Considerations**

Caden Keese, Anna Seiderer and Siva Shankar Siva Saravanan

June 2024

## 1 Introduction

In order to receive phase-modulated signals on the provided backscatter platform, they must comply with the IEEE 802.15.4 standard [1]. This includes, among other things, that no BPSK is possible, it must be an O-QPSK with four symbols and an offset. One of the main challenges in the implementation was to realise the specifications and timing requirements of the standard so that we can receive packages on the *Launchpad CC1352P7* board. This report looks into the aspects that we took into account in order to be able to send and receive data successfully.

## 2 Requirements and Constraints

In addition to the requirements that can be derived from the IEEE 802.15.4 standard, there is also the difficulty of implementing them on the *Raspberry Pi Pico*. Both are presented below.

### 2.1 Requirements according to IEEE 802.15.4

#### Data rate

In the IEEE 802.15.4 specification [1], a data rate of $f_{DR} = 250\,kbit/s$ is defined for transmission. Since four data bits are translated into a 32-bit long chip sequence, this corresponds to a chip rate of $f_{CR} = 2000\,kChip/s$. Always two chip bits are transmitted simultaneously in a 2-bit symbol, therefore the resulting symbol rate is $f_{SR} = 1000\,kSym/s$. The transmission duration for one symbol is

$$t_{Sym} = \frac{1}{f_{SR}} = 1\,\mu s \quad .$$

**O-QPSK**

The IEEE 802.15.4 standard requires the implementation of *Offset*-QPSK. O-QPSK avoids phase shifts of 180° that would occur if both bits of a symbol change at once (e.g. if `00` is followed by `11` or if `01` is followed by `10`). Commonly, O-QPSK is implemented by offsetting the quadrature bits with respect to the inphase bits by half a symbol period [3]. In our implementation, we realise the offset by inserting additional symbols between each two symbols of the symbol sequence that is to be sent. With the help of these inserted symbols, phase transitions of 180° can be avoided. The following symbol sequence is considered as an example:

```
01 10 00 11 10 00 00
```

Transition symbols are needed between `01` and `10` and between `00` and `11`. For consistency, additional symbols are also added in between the remaining symbols:

```
01 11 10 00 00 01 11 11 10 00 00 00 00
```

The resulting sequence is approximately twice as long as the original sequence. This has an effect on the symbol rate and the transmission duration per symbol. Now, about twice as many symbols have to be sent with a symbol rate of $f_{SR*} = 2000\,kSym/s$ to transmit the same information. The adjusted transmission duration for one symbol is

$$t_{Sym} = \frac{1}{f_{SR*}} = 500\,ns \quad .$$

## 2.2 Constraints due to the PIO of the *Raspberry Pi Pico*

The *Raspberry Pi Pico* has two programmable input/output (PIO) blocks with four state machines each [4]. These state machines execute sequential programs and allow the manipulation of GPIOs with precise timing. PIO programs are written in PIO assembly that consists of an instructions set with a total of nine instructions. The execution of an instruction takes one clock cycle but can be delayed by up to 31 additional cycles. The instruction memory per PIO block can hold at most 32 instructions [2]. The small instruction set and the very limited size of the programme place high demands on the programming of the PIO.

# 3 Implementation

We tried several approaches for the implementation, which are described below, in order to eventually be able to transfer data successfully.

## 3.1 First approach

In an initial approach, we tried to implement O-QPSK modulation by adapting the *baseband.pio*-file that was provided for BFSK modulation. Here, the symbol

sequence is passed to the FIFO of the PIO and then interpreted by the PIO program. Depending on the symbol, `jmp` instructions are used to jump to different loops that generate the sequence for the corresponding symbol (see Figure 1). We have encountered various difficulties with this approach.

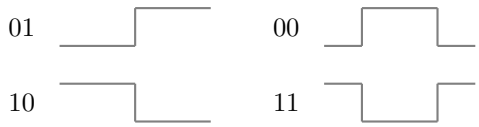


Figure 1: Symbols and corresponding waveforms.

One problem is that more instructions are now required as four loops are needed for the four different symbols as opposed to just two as was the case for BFSK. Another factor that results in the need for more instructions is that reading and interpreting the symbols now requires a two-stage process with jump instructions instead of just one, since a symbol now consists of two bits. Finally, more instructions are now required to describe the symbol itself. With BFSK, only one change between high and low per symbol was necessary, and this is also the case for symbols `01` and `10` (see Figure 1), where this is still the case for QPSK. For the symbols `00` and `11`, however, two changes between high and low are required, which in turn leads to more instructions in the code. In the end, we needed more than the 32 available instructions and therefore split the code between the two PIO blocks which control two GPIO pins. This in turn made synchronisation between the blocks necessary and added further complexity. Figure 2 shows that we were able to synchronise two pins such that they would transmit in lockstep (traces 2 and 3) with the help of a third pin (trace 1).

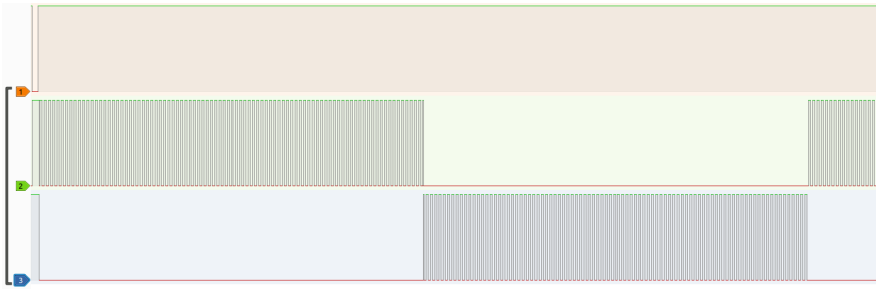

Figure 2: Measurement at synchronised GPIO pins.

Another problem with this approach is compliance with the data rate specifications. As already described, the description of a symbol waveform requires more instructions as there are more changes between high and low. This means that a waveform requires more clock cycles and therefore more time. What adds further instructions and thus clock cycles is the change to the next symbol. We

ended up with code that required at least 32 instruction cycles per waveform. Assuming that further repetitions of the waveform are necessary for the symbol to be recognised by the receiver, 128 instruction cycles are required per symbol if the waveform is repeated 4 times. At the standard frequency of the *Raspberry Pi Pico* of 125 MHz, the transmission time of a symbol is

$$t_{Sym} = \frac{1}{125\,MHz} \cdot 128 = 1.024\,\mu s \quad .$$

This is about twice the specified symbol duration. Although the clock frequency of the *Raspberry Pi Pico* is adjustable to a certain extent, the required symbol duration is not maintained even with the maximum frequency of 133 MHz (although further overclocking is possible according to various sources).

## 3.2 Second approach

The difficulties of the implementation described above have led to a change in approach. Here, a large part of the symbol processing is offloaded to the c-code and the PIO is only used to switch between high and low state. Listing 1 shows the entire PIO assembly code. The state of a GPIO pin is switched on `0` input and remains in the same state on `1` input.

```
.wrap_target
    SET   pins    0  [1]
    loop1:
        OUT       x   1
        JMP  x--  loop1
    SET   pins    1  [1]
    loop2:
        OUT       y   1
        JMP  y--  loop2
.wrap
```

Listing 1: PIO code for switching of a pin.

The minimum number of instruction cycles per state is four, which leads of a minimum of 16 cycles per waveform. With four waveform repetitions this results in 64 cycles per symbol. For conforming to the required symbol duration of $t_{Sym} = 500\,ns$ a clock frequency of

$$f_{clk} = \frac{1}{500\,ns} \cdot 64 = 128\,MHz$$

is required. This frequency can be set on the *Raspberry Pi Pico* so that the data rate requirements are met.
With this approach we were able to successfully test O-QPSK communication on the backscatter platform.

# 4 Conclusions

The requirements of the IEEE 802.15.4 standard and the restrictions imposed by the *Raspberry Pi Pico* PIO presented us with challenges during implementation. Ultimately, however, the result was that we now have a short and simple PIO code that switches between high and low level and we were able to transmit O-QPSK modulated data successfully. Since the code is not specific to O-QPSK modulation, it could also be used for other projects, e.g. for communication with BFSK or other modulation techniques.
In our calculations, we have assumed that at least four repetitions of a waveform are necessary for the symbol to be recognised by the receiver. However, this was not tested. It is possible that fewer repetitions are sufficient. In this case, either a lower clock frequency could be used, or more complex waveforms that require more instructions and therefore cycles can be modulated.

# References


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

[2] R. Lauer. 2021. *A Practical Look at PIO on the Raspberry Pi Pico.* Blues Wireless. https://blues.com/blog/raspberry-pi-pico-pio/

[3] C. Neuhaeusler. 2016. *Generation of IEEE 802.15.4 Signals.* Rohde & Schwarz. https://scdn.rohde-schwarz.com/ur/pws/dl_downloads/dl_application/application_notes/1gp105/1GP105_1E_Generation_of_IEEE_802154_Signals.pdf

[4] Raspberry Pi Ltd 2014. *RP2040 Datasheet: A microcontroller by Raspberry Pi.* Raspberry Pi Ltd. https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf?_gl=1*1uz66z*_ga*MTkxNDkxMjg1OS4xNzA1NTcyNDUx*_ga_22FD70LWDS*MTcxNjg4MTEwNS4xNi4xLjE3MTY4ODExMTkuMC4wLjA.