

## Article

# Software-Defined Radio-Based Internet of Things Communication Systems: An Application for the DASH7 Alliance Protocol <sup>†</sup>

Dennis Joosens <sup>1,\*</sup> , Noori BniLam <sup>2</sup> , Rafael Berkvens <sup>1</sup>  and Maarten Weyn <sup>1</sup> 

<sup>1</sup> IDLab—Faculty of Applied Engineering, University of Antwerp—imec, Sint-Pietersvliet 7, 2000 Antwerp, Belgium; rafael.berkvens@uantwerpen.be (R.B.); maarten.weyn@uantwerpen.be (M.W.)

<sup>2</sup> European Space Agency—European Space Research and Technology Centre (ESA—ESTEC), Keplerlaan 1, 2201 AZ Noordwijk, The Netherlands; noori.bniam@ext.esa.int

\* Correspondence: dennis.joosens@uantwerpen.be

<sup>†</sup> This article is a revised and expanded version of a paper entitled Implementation of a Multi-Channel DASH7 IoT Communication System for Packet Investigation and Validation, which was presented at the 14th Annual GNU Radio Conference, Knoxville, TN, USA, 16–20 September 2024.

**Abstract:** Software-Defined Radio (SDR) technology has been a very popular and powerful prototyping device for decades. It finds applications in both fundamental research or application-oriented tasks. Additionally, the continuing rise of the Internet of Things (IoT) necessitates the validation, processing, and decoding of a large number of received signals. This is where SDRs can be a valuable instrument. In this work, we present an open-source software system using GNU Radio and SDRs, which improves the comprehension of the physical layer aspects of Internet of Things communication systems. Our implementation is generic and application-agnostic. Therefore, it can serve as a learning and investigation instrument for any IoT communication system. Within this work, we implement the open-source DASH7 Alliance Protocol (D7AP). The developed software tool can simulate synthetic DASH7 signals, process recorded data sets, and decode the received DASH7 packets in real time using an SDR front-end. The software is accompanied by three data sets collected in controlled, indoor, and suburban environments. The experimental results revealed that the total packet losses of the data sets were 0%, 2.33%, and 16.67%, respectively. Simultaneously, the three data sets were received by a dedicated DASH7 gateway with total packet losses of 0%, 3.83%, and 7.92%, respectively.

**Keywords:** LPWAN; LoRaWAN; LoRa; Sigfox; DASH7; GNU Radio; SDR



Academic Editor: Christos Bouras

Received: 14 November 2024

Revised: 18 December 2024

Accepted: 25 December 2024

Published: 31 December 2024

**Citation:** Joosens, D.; BniLam, N.; Berkvens, R.; Weyn, M.

Software-Defined Radio-Based Internet of Things Communication Systems: An Application for the DASH7 Alliance Protocol. *Appl. Sci.* **2025**, *15*, 333. <https://doi.org/10.3390/app15010333>

**Copyright:** © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

The Internet of Things (IoT) is a concept that has evolved in the last decade to provide internet connectivity to objects that we use in everyday life. The versatility of these objects knows no limits. They can be generic devices such as bicycles, motors, home appliances, tools of any kind, and devices such as temperature, humidity, air quality, or inertial measurement unit sensors. These objects are also usually equipped with extra hardware that constitutes mainly a processing unit and a transceiver for digital communications. Consequently, this hardware enables objects to communicate with one another or to send information to the internet [1]. The IoT concept aims to provide an extra layer of intelligence to these objects by enabling remote access. Furthermore, it allows interaction among several objects to achieve a high operational efficiency [2,3]. Accordingly, the IoT concept has attracted the attention of many industries where automation plays a key role. Applications

can be found in home and industrial automation, medical aids, energy management, automotive, smart cities, etc.

Moreover, Low-Power Wide-Area Networks (LPWANs) have emerged to provide the communication means that have enabled the IoT concept. Most LPWAN technologies can, in principle, provide the required internet connectivity to millions or even billions of objects. This communication revolution can be attributed to the LPWAN's capability to establish long-range communication links that go up to several kilometers while using low-power transceivers [4]. Moreover, due to the low production cost, LPWAN transceivers are massively deployed in large-scale environments, i.e., on the scale of cities or even countries.

During the last decades, Software-Defined Radio (SDR) technology has also become a very popular and powerful prototyping tool [5,6]. Before the introduction of SDR technology, researchers and manufacturers relied only on custom-made expensive hardware and simulation tools. Although simulation tools can be very powerful in describing any physical behavior, they cannot include all the physical variables that are associated with complex engineering problems. On the other hand, SDR technology can provide a physical prototype that can be subjected to testing scrutiny. A noticeable trend in recent years is that SDR technology is used as a final product for many communication system technologies due to the decreasing production costs.

In this paper, we make use of SDR technology to design the physical layer for LPWAN communication systems. The design steps are generic and can be deployed with any LPWAN standard. However, we adopted the DASH7 Alliance Protocol (D7A) [7]. This communication standard was selected to represent IoT communication systems because its specification is freely available [8], and an open-source software stack is ready to use [9]. The proposed SDR implementation of the DASH7 standard was validated using simulation and experimental analyses. Furthermore, we provided raw DASH7 signals that were recorded using the SDR front-end in indoor and suburban environments to test our implementation; these data sets were correctly labeled, and the accurate positions of every transmitted signal were reported, which can be used in various applications, including localization.

In the following section, several LPWAN technologies will be presented, and a brief introduction to SDR technology will be provided. The DASH7 Alliance Protocol and our SDR implementation are presented in Sections 4 and 5. In Section 6, we present simulation and experimental results that consolidate our implementation capabilities, followed by the discussion of the achieved results in Section 7. Subsequently, conclusions are drawn in Section 8. Finally, several technical appendices are presented to serve as tutorial material to understand common coding and decoding techniques used in telecommunication systems.

## 2. LPWAN Technologies

Several LPWAN technologies have been introduced over the years. In the following sections, a brief description of several established technologies is provided. Furthermore, an overview of their key characteristics is depicted in Table 1.

### 2.1. LoRaWAN

As a small start-up company, Cycleo introduced the LoRa (Long-Range) communication network in 2009 [10]. Three years later, the company was acquired by the Semtech corporation [11]. Since then, LoRa Wide-Area Networks (LoRaWANs) have been deployed all over the world to provide long-range and low-power communication for IoT networks [12].

**Table 1.** An overview of the physical layer characteristics of LPWAN technologies.

	Operating Frequency (MHz)	Bandwidth (kHz)	Range (km)	Data Rate (kbps)
LoRaWAN Sub-1 GHz	433/868 (EU) 915 (US)	125/250/500	5 (urban) 20 (rural)	0.3–50
LoRaWAN 2.4 GHz	2400	203/406 812/1625	0.5 (urban) 10 (rural)	0.595–253.91
Sigfox	868 (EU) 915 (US)	0.1 (UL) 0.1 (DL)	10 (urban) 40 (rural)	0.1 (UL) 0.6 (DL)
NB-IoT	Licensed LTE bands	180	1 (urban) 10 (rural)	150 (UL) 127 (DL)
D7AP	433/868 (EU) 915 (US)	25/200	1–2 (urban) 5 (rural)	9.6/55.6 166.7

The LoRaWAN physical layer is closed-source and proprietary. It utilizes Chirped Spread Spectrum (CSS) modulation in the unlicensed Sub-1 GHz frequency bands [13]. The operational bandwidth can be either 125, 250, or 500 kHz. Moreover, the allowed data rate varies from 300 bits per second (bps) to 50 kbps [14,15]. Accordingly, the characteristics of a LoRa signal, i.e., the carrier frequency, the bandwidth of the signal, and the modulation technique, allow LoRa networks to establish communication links up to 15 km in rural areas [2]. Besides that, LoRa sub-GHz has an extension, namely LoRa 2.4 GHz, which was announced in 2020. This LPWAN technology has the benefit of using the widely used unlicensed 2.4 GHz Industrial, Scientific, and Medical (ISM) band and, therefore, is not subject to duty cycle restrictions. The technology is based on the same CSS technique, but the bandwidths, spreading factors, and, therefore, data rates are expanded. Nevertheless, the advertised communication ranges seem to be higher compared to any other technology that is operating in this band [12].

## 2.2. Sigfox

In 2010, the French start-up Sigfox introduced the Sigfox technology, which can serve as an IoT communication network [10]. The Sigfox physical layer is also closed-source and proprietary. It utilizes Binary Phase Shift-Keying (BPSK) modulation in the unlicensed Sub-1 GHz frequency bands [10]. Sigfox exploits the Ultra Narrowband (UNB) technology with an uplink of 100 Hz bandwidth. Therefore, Sigfox signals encounter very low noise levels. Consequently, Sigfox gateways have a high receiving sensitivity, and the transceivers consume minimal power. Accordingly, a Sigfox gateway is presumed to handle up to a million connected objects with a coverage of 50 km in rural areas [2]. To achieve this low power consumption and long communication range, Sigfox networks limit the transmission throughput to 100 bps.

## 2.3. NB-IoT

Narrowband Internet of Things (NB-IoT) technology was introduced by 3GPP to provide LPWAN services [14]. The NB-IoT standard utilizes mobile network features (e.g., user identity confidentiality, authentication, and integrity). It is designed to achieve excellent coexistence performance between Global System for Mobile Communication (GSM) and Long-Term Evolution (LTE) technologies [16]. NB-IoT requires 180 kHz minimum system bandwidth for both downlink and uplink. Therefore, a GSM operator can replace one GSM carrier of 200 kHz to NB-IoT. An LTE operator can deploy NB-IoT inside an LTE carrier by reallocating one of the Physical Resource Blocks (PRBs) of 180 kHz to NB-IoT.

NB-IoT utilizes BPSK and Quadrature Phase Shift-Keying (QPSK) modulations at a frequency band of 700 MHz, 800 MHz, and 900 MHz. Consequently, NB-IoT can provide a communication link in rural areas up to 10 km [10].

#### 2.4. DASH7

DASH7 is an open-source standard for wireless communication. It provides ultra-low-power and mid-range communication connectivity that is suitable for IoT applications. A DASH7 network can establish a communication link up to 5 km. Therefore, DASH7 is technically not an LPWAN technology. It is worth noting that some review papers classify the DASH7 standard as an LPWAN technology [17].

According to the DASH7 Alliance Protocol (D7AP) specification, the DASH7 physical layer utilizes Gaussian Frequency-Shift Keying (GFSK) modulation in the unlicensed Sub-1 GHz frequency bands. Moreover, DASH7 signals can be categorized into three channel classes with different parameters. The Lo-Rate class has an operational bandwidth of 25 kHz and a data rate of 9.6 kbps, the Normal class with an operational bandwidth of 200 kHz and a data rate of 55.555 kbps, and finally the Hi-Rate class which has an operational bandwidth of 200 kHz and a data rate of 166.667 kbps [5,7,18]. Consequently, the DASH7 physical layer characteristics, i.e., the carrier frequency, the signals' operational bandwidth, and the minimal transmission power, are similar to the aforementioned LPWAN technologies.

### 3. SDR Technology

SDR technology allows software modules to run on a generic hardware platform used to implement radio functions. By combining an SDR front-end with a software platform, the user can exploit SDR technology to rapidly prototype communication systems, including physical layer design, recording and playback of signals, signal intelligence, algorithm validation, and more. In the following section, we provide a brief introduction to the main aspects of SDR technology. Furthermore, IoT technologies can benefit from SDR technology. Flexible SDR-based IoT gateways can be deployed while reducing overall costs. The reason that SDRs are a good candidate to serve as a platform within the IoT is due to the usage of straightforward modulation techniques, the deployment of low-bandwidth applications and low-data-rate signals.

#### 3.1. SDR Architecture

Nowadays, various Commercial-Off-The-Shelf (COTS) SDRs are available for use. These SDRs vary in terms of cost and operational modes, i.e., transmitting, receiving or duplex modes, operating bandwidth, frequency bands, and the amount of coherent RF channels. Table 2 provides a brief comparison among several commercial SDR front-end units. In the following sections, the reception and transmission operations that are handled by the SDR front-end are briefly introduced.

##### 3.1.1. SDR Transmission Mode

During the transmission process, the baseband in-phase and quadrature (I/Q) samples are synthesized by the host computer and fed to the onboard FPGA of the SDR at a specified sampling rate. The FPGA interpolates the data to a higher sampling rate using a digital up-conversion (DUC) process. Thereafter, it converts the signal to the analog domain with a digital-to-analog converter (DAC). The signal will be mixed with a specified carrier frequency signal that is generated by a local oscillator (LO). After this stage, the signal is amplified and ready to be transmitted [19].

**Table 2.** Non-exhaustive list of common SDRs that can be used for the reception of LPWAN protocols.

SDR	Frequency (MHz)	ADC/DAC Resolution	Max. RF Bandwidth	RF Channels	Price
RTL-SDR Blog V3	24–1766	8-bit	3.2 MHz	1 RX	low
Great Scott Gadgets HackRF one	1–6000	8-bit	20 MHz	1 TX/RX	medium
Analog Devices ADALM-PLUTO	325–3800	12-bit	20 MHz	1 TX–1 RX	medium
Nuand bladeRF 2.0 xA4	70–6000	12-bit	56 MHz	2 TX–2 RX	medium
Ettus Research USRP B200mini	70–6000	12-bit	56 MHz	1 TX–1 RX	medium
Ettus Research USRP B200	70–6000	12-bit	56 MHz	1 TX–1 RX	medium
Ettus Research USRP B210	70–6000	12-bit	56 MHz	2 TX–2 RX	high
Deepwave Digital AIR7201-B	300–6000	14–16 bit	100 MHz	2 TX–2 RX	very high

### 3.1.2. SDR Receiving Mode

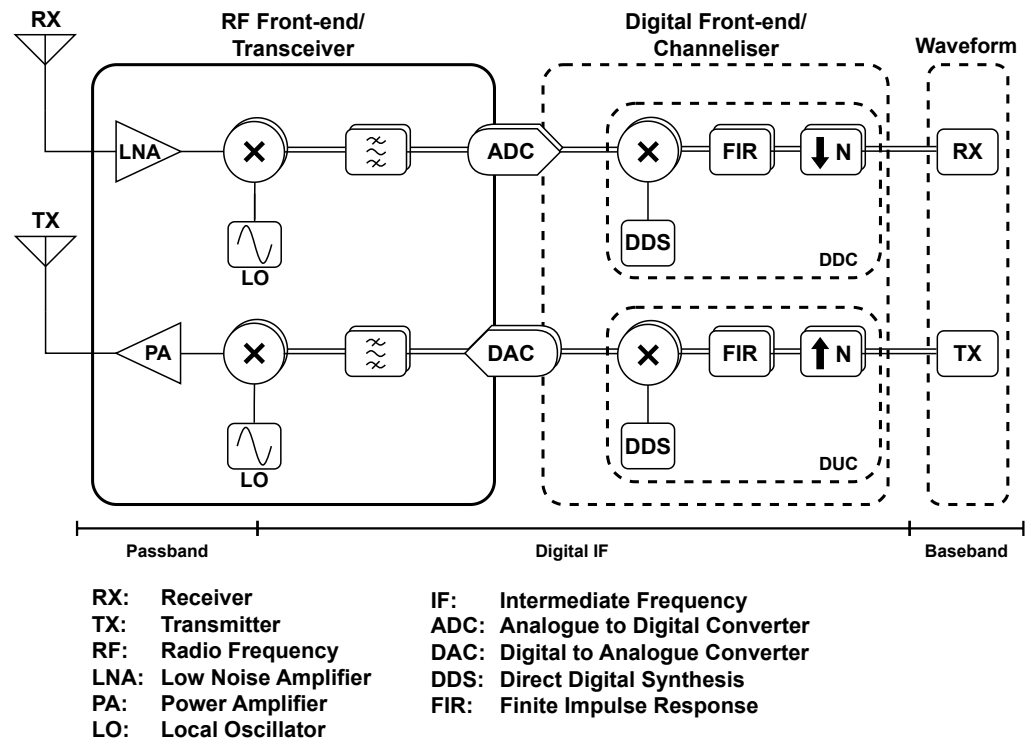
The first stage in the reception process is the down-conversion of the received signal by mixing the incoming signal with an LO. In this way, the received signal is converted from the Radio Frequency (RF) domain to the Intermediate Frequency (IF) domain. The IF signal is then represented by I/Q components. Afterwards, the intermediate I/Q components will be sampled by an analog-to-digital converter (ADC). The digitized I/Q data follow parallel paths through a digital down-conversion (DDC) process which mixes, filters, and decimates the input signal to a user-specified sampling rate. The down-converted samples will then be sent to the host computer. Both transmitting and receiving operations are depicted in Figure 1.

### 3.2. GNU Radio

GNU Radio is a set of free and open-source tools aiming to implement SDR systems and provides ready-to-use signal processing blocks [20]. It can be used with COTS SDRs or in a simulation mode. It can be used to write applications, to receive data out of digital streams, or to push data into digital streams, which can be transmitted using specific hardware. GNU Radio contains a variety of blocks such as filters, channel coders, synchronisation elements, equalizers, demodulators, decoders, and many other elements that are typically found in software radio systems. These elements are readily available as software blocks. More importantly, it includes a method of connecting these blocks and then manages how data are passed from one block to another. Adding extra DSP functions can be achieved with the use of embedded Python blocks. These blocks allow the implementation of custom Python code.

Since GNU Radio 3.10 is software, it can only handle discrete data. Usually, I/Q baseband samples are the input data type for receivers and the output data type for transmitters. SDRs are then used to shift the signal to the desired center frequency. However, within GNU Radio, any data type can be passed from one block to another, e.g., bits, bytes, vectors, integers, and real or complex numbers. When creating new applications in GNU Radio, they are primarily based on Python. Nonetheless, the performance-critical signal

processing path is implemented in C++ using processor floating-point extensions. Thus, the developers can implement real-time, high-throughput radio systems in an easy-to-use environment.



**Figure 1.** Functional block diagram of a direct conversion or zero-IF-based SDR architecture. Note that the parallel lines indicate that this process is repeated; i.e., one flow is dedicated to the in-phase data stream and one flow to the quadrature data stream.

### 3.3. In-Phase and Quadrature Data

The use of in-phase and quadrature (I/Q) data is a key concept when working with SDRs. I/Q data are a mathematical complex representation of the transmitted (or received) baseband signal. In this way, I/Q data describe the variations in the amplitude and phase of a signal. An I/Q signal can be represented as two orthogonal signals which form a new waveform. The creation of this new waveform is achieved by using a quadrature modulator; see Figure 2. The two real data signals at the transmitter end can be written as follows:

$$i(t) = I_t(t) \cos(2\pi f_o t) \tag{1}$$

$$q(t) = Q_t(t) \sin(2\pi f_o t) \tag{2}$$

Combining the two components leads to the following:

$$y(t) = I_t(t) \cos(2\pi f_o t) + Q_t(t) \sin(2\pi f_o t). \tag{3}$$

At the receiver end, the in-phase and quadrature components can be recovered as follows by applying the product-to-sum identities:

$$\begin{aligned} I_r(t) &= [I_t(t) \cos(2\pi f_o t) + Q_t(t) \sin(2\pi f_o t)] * 2 \cos(2\pi f_o t) \\ &= \underbrace{I_t(t)}_{\text{term 1}} + \underbrace{I_t(t) \cos(4\pi f_o t)}_{\text{term 2}} + \underbrace{Q_t(t) \sin(4\pi f_o t)}_{\text{term 3}} \end{aligned} \tag{4}$$

$$\begin{aligned}
 Q_r(t) &= [I_t(t) \cos(2\pi f_0 t) + Q_t(t) \sin(2\pi f_0 t)] * 2 \sin(2\pi f_0 t) \\
 &= \underbrace{Q_t(t)}_{\text{term 1}} - \underbrace{Q_t(t) \cos(4\pi f_0 t)}_{\text{term 2}} + \underbrace{I_t(t) \sin(4\pi f_0 t)}_{\text{term 3}}
 \end{aligned}
 \tag{5}$$

By using a low-pass filter (LPF), the high-frequency components, i.e., term two and term three in Equations (4) and (5), are removed, and the in-phase component  $i(t)$  and quadrature component  $q(t)$  can be recovered at the receiver.

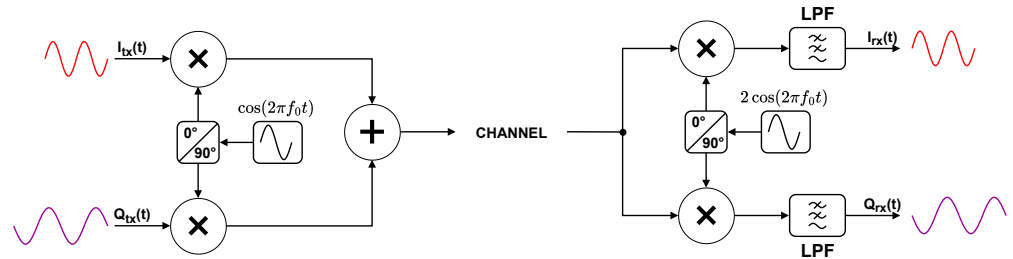


Figure 2. Concept of an in-phase and quadrature modulator and demodulator.

### 4. DASH7 Communication System

Before explaining the implementation of our communication system, we first look into several important DASH7 system parameters of the physical layer to obtain a deeper understanding. These parameters are described in the following sections and are defined by the DASH7 Alliance [21].

#### 4.1. Air Interface

##### 4.1.1. RF Channels

As mentioned earlier, the DASH7 communication standard operates in the unlicensed sub-GHz frequency spectrum. The RF band that is available to use depends on the geographical region from where devices are operated. DASH7 supports the 433 MHz, 868 MHz, and 915 MHz bands. These bands with their specific channel indexes and frequencies are found in Table 3.

Table 3. DASH7 channel bands. The stars in the table indicate the following: \* Worldwide coverage with local regulatory limitations; \*\* EN 300 220 (Europe) with local regulatory limitations; \*\*\* FCC part 15 in the United States of America.

RF Band	Lo-Rate (d)	Normal and Hi-Rate (d)	Start (b)	End
433 MHz *	0, 1, ..., 68	0, 8, 16, ..., 56	433.06 MHz	434.785 MHz
868 MHz **	0, 1, ..., 279	0, 8, 16, ..., 216, 229, 239, 257, 270	863 MHz	870 MHz
915 MHz ***	0, 1, ..., 1039	0, 8, 16, ..., 1032	902 MHz	928 MHz

##### 4.1.2. Channel Classes

To avoid signal collisions, the DASH7 protocol divides the specific spectrum band into several channels. Therefore, the DASH7 protocol defines three channel classes that represent the data rates that can be used. These are the Lo-Rate, Normal-Rate, and Hi-Rate data rates. Each channel class defines a certain symbol rate, modulation index, frequency deviation, and occupied bandwidth [21]. These parameters as displayed in Table 4.

Each channel is defined by an index specifying its start frequency relative to the start of the DASH7 operating frequency band. The center frequency expressed in Megahertz (MHz) of each channel can be calculated as follows:

$$f_{c_{channel}} = \text{Start}(b) + 0.025d + \frac{\text{Channel Spacing}(c)}{2}, \tag{6}$$

where  $b$  and  $d$  are the start frequency of the band and the channel index, as shown in Table 3, while  $c$  is the channel spacing of the selected channel class, which can be found in Table 4.

**Table 4.** DASH7 channel classes.

Class	Symbol Rate	Modulation Index	Frequency Deviation ( $\Delta f$ )	Channel Spacing ( $c$ )
Lo-Rate	9.6 kbps	1	$\pm 4.8$ kHz	0.025 MHz
Normal	55.555 kbps	1.8	$\pm 50$ kHz	0.2 MHz
Hi-Rate	166.667 kbps	0.5	$\pm 41.667$ kHz	0.2 MHz

#### 4.1.3. DASH7 Modulation Scheme

The DASH7 protocol adopts the widely used Frequency-Shift Keying (FSK) modulation scheme with a Gaussian pulse shape filter for the transmitted samples (hence, this modulation is referred to as GFSK). Accordingly, the transmitted waveform can be presented as follows:

$$s(t) = Ae^{j\Phi(t)}, \tag{7}$$

where  $A$  is the transmitted signal's amplitude and  $\Phi(t)$  is the angular phase which can be expressed as follows:

$$\Phi(t) = 2\pi h \int_0^t \alpha(\tau) d\tau. \tag{8}$$

$\alpha(t)$  for an FSK signal is as follows:

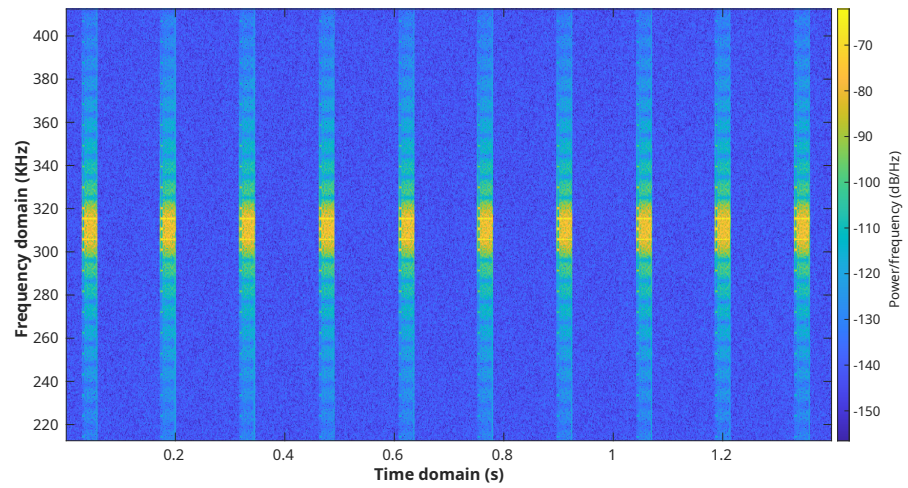
$$\alpha(t) = \sum_{i=0}^L a(i)q(t - iT_s), \tag{9}$$

where  $a$  is 1 or  $-1$  and represents the transmitted bit value.  $L$  is the number of transmitted bits, and  $q(t - iT_s)$  is the shape filter response of the transmitted signal (i.e., for GFSK, the  $q(t)$  is a Gaussian filter response). Finally,  $h$  is the modulation index, and can be expressed as follows:

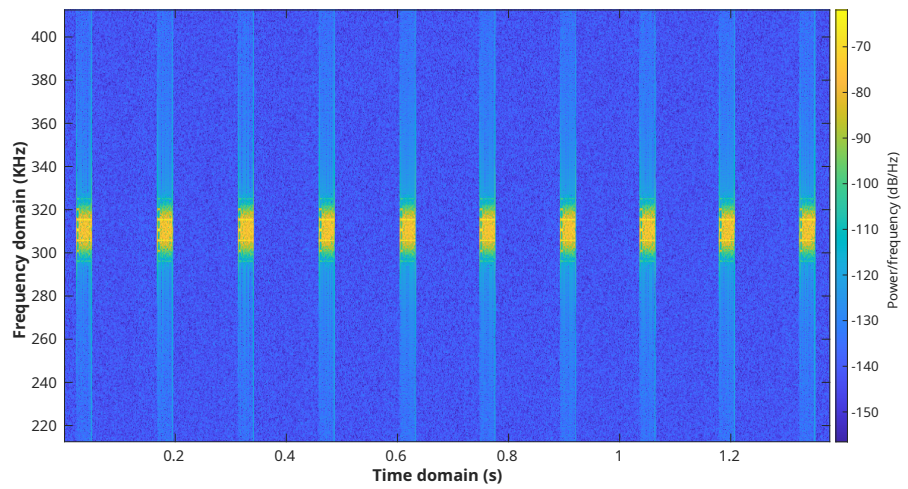
$$h = \frac{2\Delta f}{R_b}, \tag{10}$$

where  $\Delta f$  is the peak frequency deviation and  $R_b$  is the data rate.

In general, GFSK has a higher spectral efficiency compared to FSK. The Gaussian pulse shaping filter smooths the symbol transitions and, therefore, is more bandwidth-efficient because it reduces the amount of sidelobes. In this way, it also reduces Inter-Channel Interference (ICI). The effect of FSK- and GFSK-modulated signals can be seen in a two-dimensional and three-dimensional view in Figure 3 and Figure 4, respectively. The energy is much more spread in the frequency domain when no pulse shaping is applied, whereas the energy is more concentrated when a Gaussian pulse shaping step is implemented.

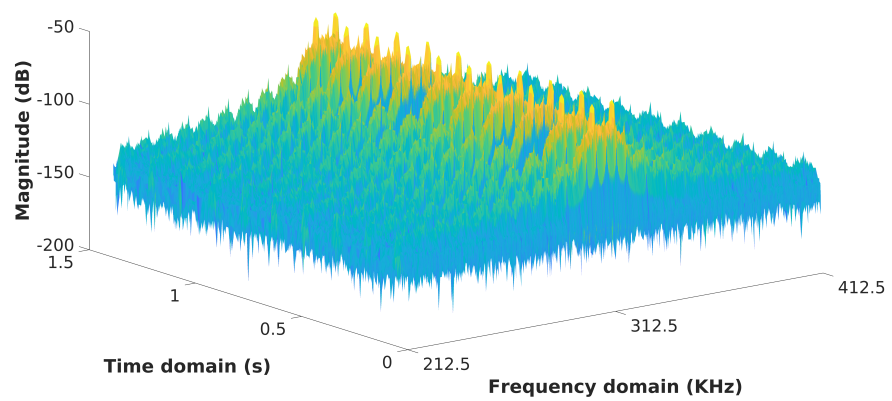


(a) FSK modulation when no pulse shaping is applied.



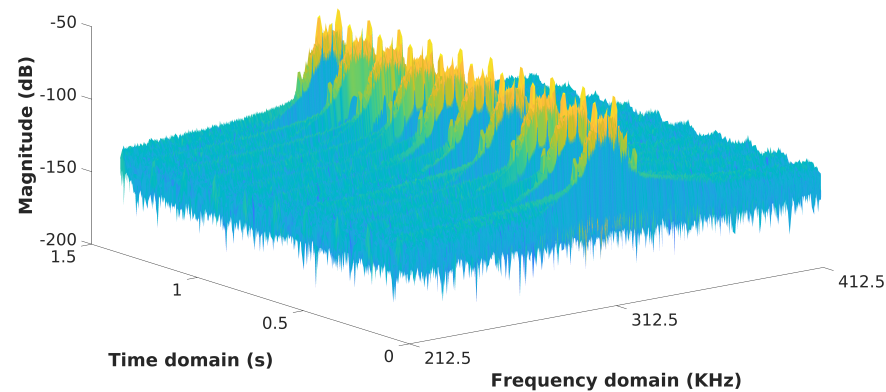
(b) GFSK modulation with a Bandwidth-Symbol Time (BT) of 0.5.

**Figure 3.** Two-dimensional spectrogram of ten modulated DASH7 packets seen over a time span of 1.4 s and a bandwidth of 200 kHz.



(a) FSK modulation when no pulse shaping is applied.

**Figure 4.** Cont.



(b) GFSK modulation with a Bandwidth-Symbol Time of 0.5.

**Figure 4.** Three-dimensional spectrogram of ten modulated DASH7 packets seen over 1.4 s and a bandwidth of 200 kHz.

The spread of a Gaussian-shaped symbol is defined by the bandwidth-to-symbol time product (BT), which, in essence, defines the curvature and amplitude of a symbol in the time domain. This is the product of the duration of a symbol and its spectral width. Its value lies between 0.1 and 1. If the BT has a small value, then the symbol is smeared in the time domain, and therefore, less bandwidth is required. However, it increases the chance of ISI occurring. When the BT product is high, the shape of the curve is closer to a rectangular shape in the time domain. This indicates that there is a lower chance of ISI, but it requires a much higher bandwidth. Therefore, a good trade-off needs to be found. For the executed measurements, a BT of 0.5 is used.

#### 4.1.4. Gaussian Minimum-Shift Keying

When using the Hi-Rate channel class, there is a slight difference compared to the other channel classes. The modulation scheme changes from GFSK to Gaussian Minimum-Shift Keying (GMSK). This is a form of Continuous-Phase Frequency-Shift Keying (CPFSK) [22]. This means that the phase continuity between alternating symbols is maintained. To be more precise, there are no phase discontinuities between symbols observed because the frequency or tone changes happen at the carrier zero crossing points. This is because the frequency difference between the logical 0 and the logical 1 of GMSK is equal to half the data rate. The modulation index for the Hi-Rate channel class is 0.5. This is the smallest value achievable while maintaining orthogonality. In general, GMSK has a higher spectral efficiency than GFSK.

#### 4.2. Packet Structure

Every DASH7 packet is preceded by a power ramp-up and succeeded by a power ramp-down, which is used to meet the band stop channel requirements. The power ramp-up is the time that is needed before the first symbol transmission. This means that the carrier frequency is ramped from idle power to transmit power and settles to a stable state. The reverse operation occurs when ramping down. If this ramp-down is too fast, a part of the encoded frame might be affected, and therefore, the packet can become unrecoverable. Typically, these ramps have a period of 8 symbols but can go up to 32 symbols.

When looking into the Data Link Layer of DASH7, we see two types of frames defined: background frames and foreground frames. Background frames have a static payload size of six bytes. These frames are used to send broadcast messages, which are applied for

ad-hoc group synchronisation. Foreground frames can have a length of up to 256 bytes and are used for data requests or contain data.

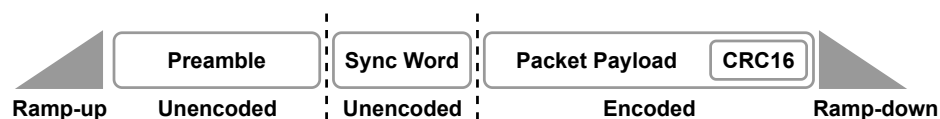
The preamble is an unencoded static structure of alternating binary symbols starting with '1' and can have a variable length of up to 64 bits. This is used to settle the receiver by calibrating its data rate circuits. The DASH7 specification defines a typical preamble length of 32 bits for the Lo-Rate and Normal-Rate classes while recommending 48 bits for the Hi-Rate channel class [23].

The sync word is an unencoded structure of 16 binary symbols used to identify the start of the payload. The physical layer supports two sync word classes which depend on the type of frame that is used. Background frames use Sync Word Class 0, while foreground frames use Sync Word Class 1. A second categorisation that makes up a specific sync word is the applied type of encoding. It can be seen from Table 5 that only Coding Scheme 0 (CS0) and Coding Scheme 2 (CS2) are used. In future releases, this can be extended with two additional coding schemes.

**Table 5.** DASH7 sync word classes and the possible applied Coding Schemes (CSs). Currently, only CS0 and CS2 are used. CS1 and CS3 are reserved for future use.

Sync Word Class	Coding Scheme			
	CS0	CS1	CS2	CS3
0	0xE6D0	RFU	0xF498	RFU
1	0x0B67	RFU	0x192F	RFU

Finally, the payload field, which can have a length between 5 and 256 bytes, contains at least a length byte, a subnet byte, a control byte (CTRL), the packet data, and two Cyclic Redundancy Check (CRC16) bytes. The CRC operation is applied only to the payload before the encoding process. Its operation is extensively explained in Appendix A. Furthermore, the payload is the only encoded part of the packet. This encoding step is achieved with a PN9 code or a combination of a PN9 code and  $\frac{1}{2}$ -Forward Error Correction Code (FEC), which are described in Appendices B and C. Figure 5 shows the general composition of a DASH7 packet.



**Figure 5.** DASH7 packet structure.

## 5. DASH7 Communication System Implementation

In this section, we present the implementation of our communication system using SDR technology. To represent the communication system more conveniently, we divide it into a transmitting and a receiving process. Subsequently, we create separate sections. The details of both methods will be thoroughly explained in the following sections.

### 5.1. The Transmitting Process

This process constitutes three steps: a data formatting process, a symbols to waveform conversion, and a modulation step. These operations will be discussed in the following sections.

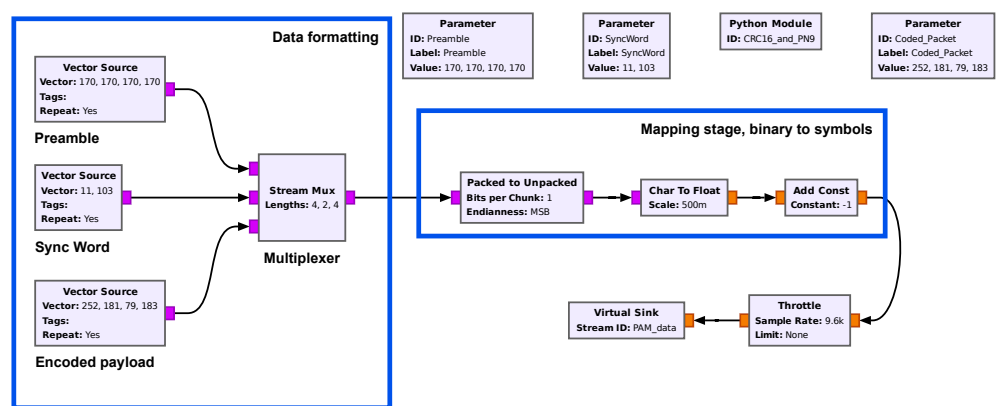
#### 5.1.1. Data Formatting

This process foresees the correct format for the packet as specified by the communication protocol [21]. The DASH7 protocol specifies that the packet starts with a preamble

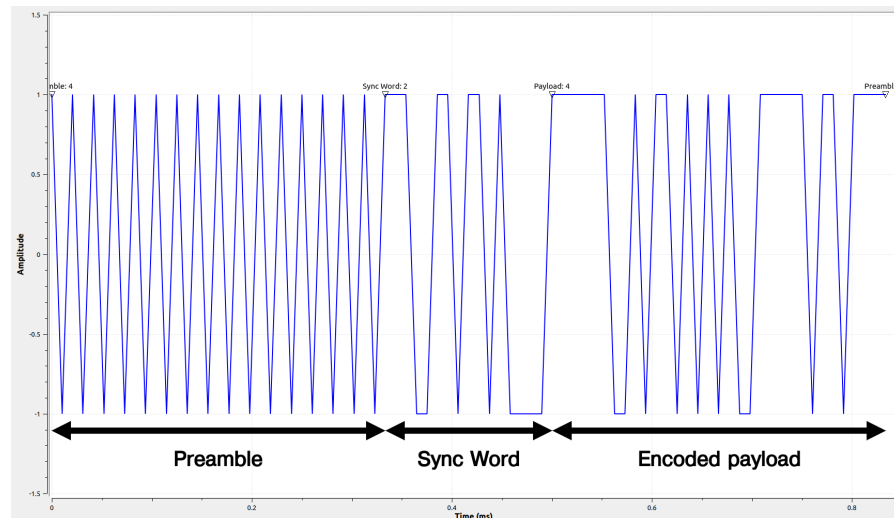
followed by a sync word and, finally, the payload containing the actual data message. Furthermore, the payload begins with one byte that specifies the length of the payload and ends with two CRC16 bytes, which are used as a bit error check. Finally, the payload data needs to be whitened using a PN9-code. The CRC and the PN9 calculations are presented in Appendices A and C, respectively. Optionally, a  $\frac{1}{2}$ -FEC can be applied before the PN9 coding operation, but this depends on the used coding scheme. A summarized operation of the FEC process can be found in Appendix B.

Figure 6 shows the GNU Radio implementation of the DASH7 data formatting step. The *Parameter* blocks contain all the required variables to create a synthetic DASH7 packet. The bytes in our implementation are represented as decimal numbers. For instance, the decimal number 170 represents a byte of the preamble data, i.e., “0b10101010”. Accordingly, you can add the decimal number 170 for every eight bits of preamble data. On the other hand, the two bytes of the sync word represent the value depending on a specific Sync Word Class and selected Coding Scheme, as shown in Table 5. Sync Word Class 0 is used for background frames, while Sync Word Class 1 is only used for foreground frames, which are defined at the Data Link Layer. As for the payload, in our implementation, we created the *CRC16\_and\_PN9* function, which is a custom *Python Module* block. This function is applied in the *Coded\_Packet* parameter block. The main task of this function is to add the length byte of the payload at the beginning and the two CRC16 bytes at the end of the payload. Afterwards, encode the entire payload with the PN9 code. Therefore, the total length of the coded packet is always the length of the payload plus three bytes.

The data formatting box in Figure 6 exploits three *Vector sources* and a *Multiplexer* block. The vector sources are dedicated to the preamble, the sync word, and the payload data. Subsequently, the *Binary to symbols* box converts each bit into a series of bytes and creates symbols by mapping these values. The 0 b is represented by  $-1$ , and the 1 b is represented by 1. The *Throttle* block is added to control the throughput of the data flow. In this case, we transmit a Lo-Rate DASH7 signal with a symbol rate of 9.6 kbps. The *Virtual Sink* block, in the flowgraph, is used for organizational purposes. Note that the *Virtual Sink* needs to be linked with a *Virtual Source* so that the data flows between the sink and the source. The complete transmitted packet is presented in Figure 7. The figure clearly shows the preamble data followed by the sync word and the whitened payload.



**Figure 6.** Packet to symbols conversion flowgraph. The data formatting box exploits three vector sources and a multiplexer block. The respective parameters control the values of the vector sources. The second blue-framed box implements the mapping of the bytes to real values. The byte’s parallel values are converted into a series of symbols, where the 0 b is represented by  $-1$  and the 1 b is represented by 1. The mapped packet is presented in Figure 7.



**Figure 7.** DASH7-mapped packet structure as seen in the time domain. The figure shows the preamble followed by the sync word and the whitened payload, respectively.

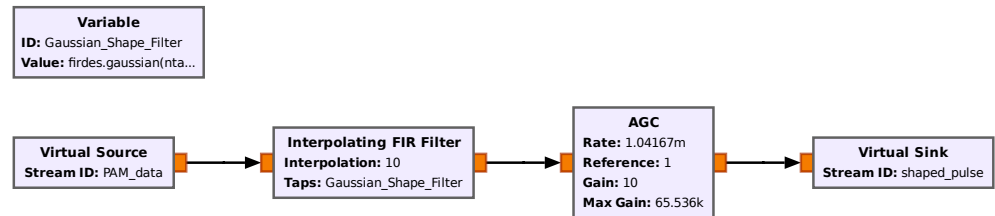
### 5.1.2. Symbols to Waveform Conversion

In the previous section, the complete DASH7 packet is formatted and converted to symbols. In this section, the symbols (see  $a(i)$  in Equation (9)) will be converted to a waveform. Consequently, every symbol will have a transmission time. Therefore, a single symbol will be represented by multiple samples. In our presentation, the symbols take the values of  $[-1,1]$  and are converted directly from the binary data  $[0,1]$ . However, the samples represent the digital waveform that will be transmitted. Therefore, the sample rate will always be larger than the symbol rate.

The process of converting an individual symbol into multiple samples is called up-sampling, and the factor that controls this conversion is called the samples per symbol (SPS) factor. Consequently, a larger SPS value will improve the modulation process but at the same time will require a higher sample rate; e.g., the Lo-Rate DASH7 signals' bit rate is 9.6 kbps if the SPS is set to 100, and this will lead to a sample rate of 960 kbps. After selecting the proper SPS, the pulse shape of the waveform should be specified. When the samples are represented by uniform values, the waveform of these samples will have a rectangular pulse shape. Nevertheless, the rectangular pulse shape is not commonly used in wireless communications due to its large side lobes in the frequency domain. In our application, the pulse shape of the signals will have a Gaussian filter response because DASH7 signals exploit GFSK modulation.

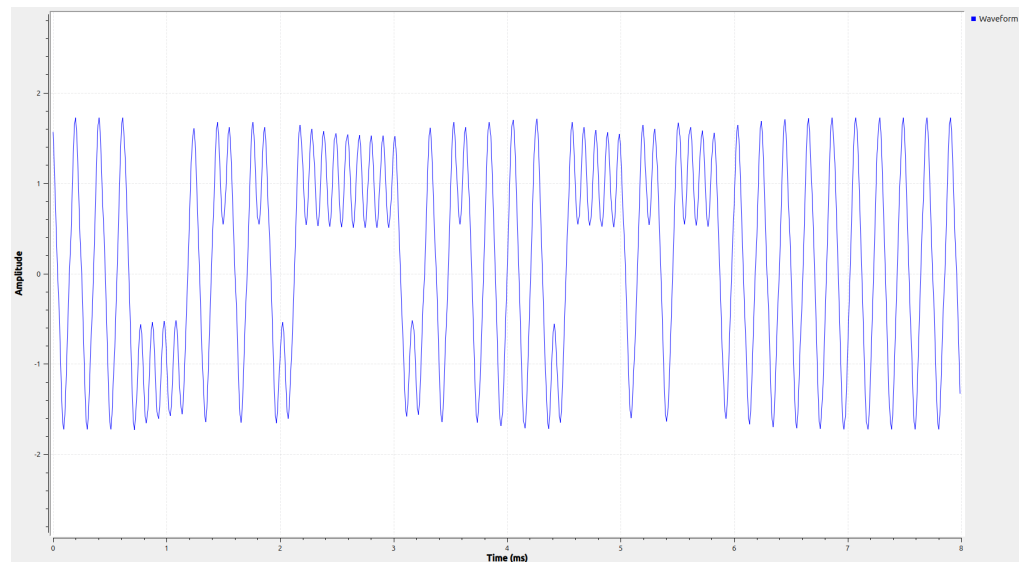
In GNU Radio, we utilize the *Interpolating FIR Filter* block to simultaneously perform the upsampling and shape filter processes. This filter creates a Finite Impulse Response (FIR) filter that performs the convolution in the time domain between the payload symbols and the filter taps, which happens after upsampling the bits by the SPS factor in the interpolation process. Consequently, if the taps are set to ones, then the output waveform will have a rectangular pulse shape. However, we deployed a Gaussian shape filter defined by the variable *Gaussian\_Shape\_Filter*. This variable exploits the GNU Radio *firdes.gaussian(gain, SPS, beta, ntaps)* function. The gain and SPS parameters are self-explanatory at this point, while beta is a Gaussian filter parameter ranging between 0 and 1. The parameter beta represents the product between the 3 dB bandwidth of the Gaussian filter and the symbol duration, i.e.,  $BT$ , where  $B$  is the bandwidth and  $T$  is the symbol duration. For instance, if beta equals 0.25, then the Gaussian pulse shape spreads over four symbols. Thus, a smaller beta causes a higher Inter-Symbol Interference (ISI) but a compact spectrum. Finally, the *ntaps* variable specifies the amount of FIR taps. The choice of the number of taps (*ntaps*) is a trade-off between the filter performance and computational cost. A long tap line will

ensure a compact signal bandwidth and good phase and amplitude responses for the signal of interest. A more expanded description regarding FIR filter design can be found in [24]. In our implementation, the *ntaps* parameter is equal to the *SPS* parameter because we usually deploy a high *SPS* factor. The implementation can be seen in Figure 8. We apply an *Automatic Gain Control (AGC)* block to the output of the *Interpolating FIR filter* block to obtain a more normalized signal after the pulses are amplified and shaped.



**Figure 8.** The upsampling process and shape filter process in GNU Radio, which is achieved with an interpolating FIR filtering.

The generated waveform is depicted in Figure 9. This signal is obtained by upsampling, pulse shaping, and gain control of the mapped symbols. This baseband waveform can now be modulated.



**Figure 9.** The upsampled and pulse-shaped waveform as seen in the time domain.

### 5.1.3. Baseband Modulation

To implement the GFSK modulation in the digital domain, the integration in Equation (8) will be substituted by a summation of calculations as follows:

$$\int_0^t \alpha(\tau) d\tau \approx \sum_{k=0}^n \alpha(kT)T, \tag{11}$$

where  $T$  is the sampling interval, i.e., the reciprocal of the sampling rate. If we consider the following:

$$\beta(nT) = \sum_{k=0}^n \alpha(kT)T, \tag{12}$$

then

$$\beta(nT) - \beta((n - 1)T) = \sum_{k=0}^n \alpha(kT) - \sum_{k=0}^{n-1} \alpha(kT) = \alpha(nT)T \tag{13}$$

leads to

$$\beta(nT) = \beta(nT - T) + \alpha(nT)T. \tag{14}$$

The Z-transform of Equation (14) can be expressed as follows:

$$\beta(z) = \beta(z)z^{-1} + \alpha(z)T, \tag{15}$$

which leads to the transfer function,

$$\frac{\beta(z)}{\alpha(z)} = T \frac{1}{1 - z^{-1}}. \tag{16}$$

Equation (16) is the single-pole difference equation of an Infinite Impulse Response (IIR) filter, which can be implemented in a straightforward manner in GNU Radio.

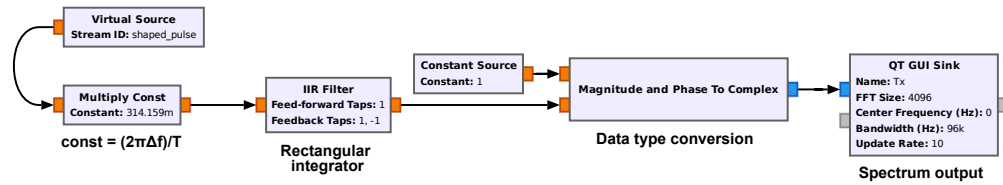
Figure 10 shows the FSK modulation flowgraph. First, the signal is multiplied by the  $2\pi$  factor and the modulation index. Afterwards, the integration process will be conducted using the IIR filter. The Feedforward and the Feedback Taps control the behavior of the IIR filter. For the single-pole IIR filter, the feedback taps need to be (1, -1), as shown in Equation (16). However, the Feedforward Taps can vary based on the deployed integrator type. Several integration types can be performed: (1) rectangular integration, where the area under the curve is assumed to be flat; (2) trapezoidal integration, where the area under the curve is assumed to be triangular; and (3) the Simpson’s rule integration, where the area under the curve is assumed to be an arc [25]. To implement this rectangular integration, we use taps that are equal to 1. For trapezoidal integration, the taps will change to  $\frac{1}{2}$  and  $\frac{1}{2}$ . Finally, for Simpson’s rule integration, it becomes more complex, and the taps will be set to  $\frac{1}{3}$ ,  $\frac{4}{3}$ , and  $\frac{1}{3}$ .

$$H_R(z) = T \frac{1}{1 - z^{-1}} \tag{17}$$

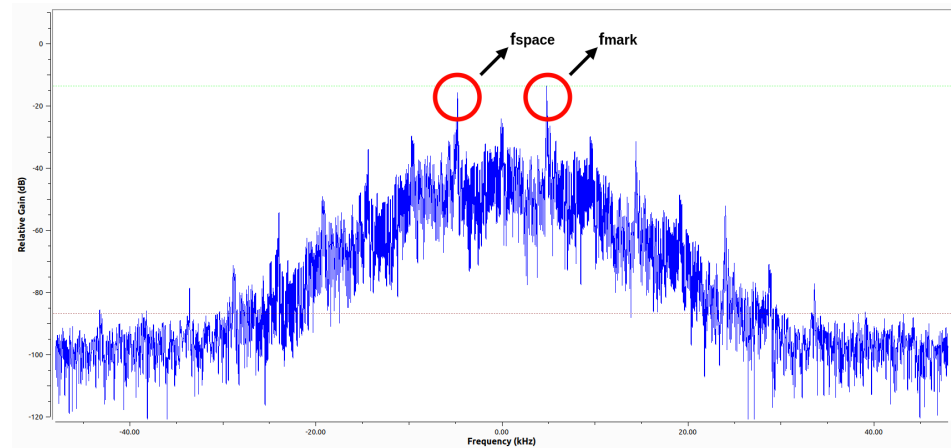
$$H_T(z) = \frac{T}{2} \frac{1 + z^{-1}}{1 - z^{-1}} \tag{18}$$

$$H_S(z) = \frac{T}{3} \frac{1 + 4z^{-1} + z^{-2}}{1 - z^{-2}} \tag{19}$$

The Feedback Taps should be (1, -1) for the rectangular and trapezoidal integration. However, for the Simpson rule, this will be (1, 0, -1). In general, if the SPS factor is high, the rectangular integration will suffice. The baseband-modulated signal is sent to the SDR front-end. On that front-end, the signal will be sent through a DAC and eventually mixed to the needed RF frequency, followed by an amplification stage. Figure 10 depicts the implementation of the rectangular integration in GNU Radio, which generates a modulated baseband signal. The output of this flowgraph in the frequency domain is shown in Figure 11. In the plot, there are two distinct peaks noticeable at -4.8 kHz and 4.8 kHz, which are the “space” and “mark” tones used by the Lo-Rate DASH7 channel class.



**Figure 10.** Flowgraph of the GFSK modulation. The FSK modulation mainly consists of an IIR filter with configurable taps that act as an integrator and a phase to complex value block.



**Figure 11.** Frequency response of a baseband-modulated DASH7 signal.

The upsampling process and shape filter process in GNU Radio are achieved with an interpolating FIR filtering.

### 5.2. The Receiving Process

The receiving process is, in essence, the opposite operation of the transmission process. It comprises four steps: reception, demodulation, time synchronisation, and decoding.

#### 5.2.1. Reception

The reception of a signal while using an SDR happens when the user tunes the local oscillator to or close to the frequency of interest. For FSK modulation, the center frequency of the SDR front-end needs to be tuned to the same center frequency of the FSK waveform. Accordingly, any deviation in the frequency between the front-end and the received modulated signal will result in a detection error and can cause a high bit error rate (BER).

In our implementation, the frequency of the received DASH7 signal is assumed to be known by the receiver (in general, in every communication system, this should be the case). Therefore, if the signal is received with an IF frequency, the signal has to be shifted to the center frequency of the receiving bandwidth. This can be achieved in GNU Radio by multiplying the received signal with a sinusoidal signal, which has a frequency equal to the IF frequency. Figure 12 presents the flowgraph of the digital mixer operation. The signal source provides a complex single-tone signal with a frequency equal to the IF frequency.

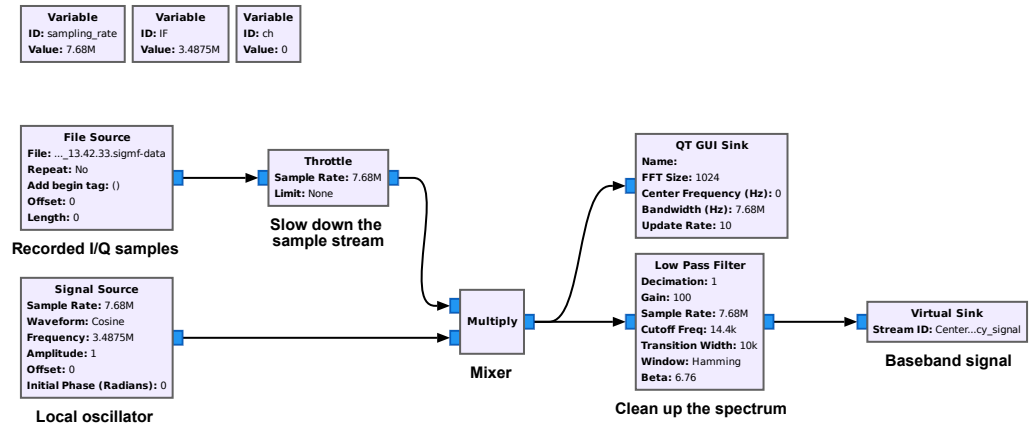


Figure 12. Fixed-frequency demodulator using recorded data while tuned to channel 0 using the Lo-Rate channel class.

### 5.2.2. Demodulation

At the receiver, and after tuning the LO at the carrier frequency of the received signal, the received digital baseband signal can be expressed as

$$x(k) = A_r e^{i\Phi(k)} , \tag{20}$$

where  $A_r$  is the received signal’s amplitude. The demodulation process of the received signal can be illustrated as follows:

1. Extracting the phase  $\Phi(k)$  of the baseband signal using a *Complex to Arg* block:

$$\Phi(k) = 2\pi h \sum_{k=0}^n \alpha(k) + \phi_e , \tag{21}$$

where  $k$  is the digitized time index, and  $\phi_e$  is a constant arbitrary phase due to the phase difference between the transmitter and the receiver.

2. Taking the derivative of  $\Phi(k)$

$$\frac{d\Phi(k)}{dk} = 2\pi h (\alpha(k) - \alpha(k - 1)) , \tag{22}$$

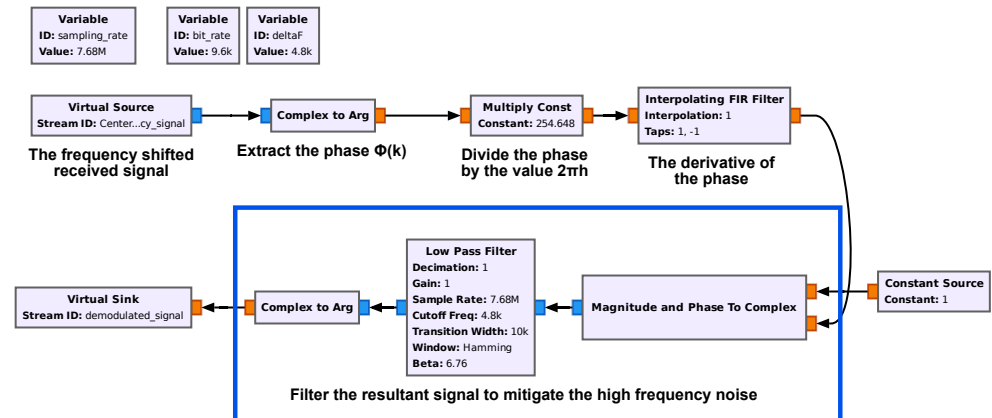
where  $h$  is the modulation index and is equal to the following:

$$h = \Delta f T . \tag{23}$$

In order to further decode the packet, the following are required:

3. Time synchronisation and bit decimation must be performed.
4. Payload detection based on the sync word bits.
5. To decode the payload data, further data de-whitening and the optional FEC decoding are required.

The software implementation of points one and two are shown in Figure 13. The implementation of subsequent points are implemented in Figure 14 and will be discussed. Points 3, 4, and 5 will be discussed in the upcoming sections.



**Figure 13.** Demodulation process. The phase is extracted from the data signal and consecutively accumulated while high-frequency noise is filtered out.

### 5.2.3. Time Synchronisation

The demodulated data samples are passed through a digital filter that minimizes the effects of noise and the effects of Inter-Symbol Interference (ISI). The ISI can be caused when the filtered received pulses overlap. The best filter to maximize the signal-to-noise ratio (SNR) while minimizing the ISI is a matched filter (MF). A matched filter has a frequency response whose magnitude matches the magnitude of the frequency response of the transmitter's shape filter. Accordingly, for DASH7 signals, a Gaussian matched filter is deployed similar to the Gaussian shape filter which is implemented at the transmitter.

Afterwards, the output of the shape filter is sampled periodically at the symbol rate which is achieved by downsampling the data by the factor SPS, at the precise sampling time instants  $t_m = mT_{symbol} + \tau$ , where  $T_{symbol}$  is the symbol interval and  $\tau$  is a nominal time delay or timing offset that accounts for the propagation time of the signal from the transmitter to the receiver [26]. Note that in DASH7, the symbol interval is the same as the bit interval. To perform this periodic sampling, we require a symbol clock at the receiver to select the correct symbol from several received samples. The process of extracting such a clock signal at the receiver is usually called symbol synchronisation or timing recovery. Timing recovery is one of the most critical functions performed at the receiver of an asynchronous digital communication system. It is important to emphasize that the receiver must know both the transmitted symbol rate ( $1/T_{symbol}$ ) and the time instant to extract the symbol from multiple received samples within each symbol interval  $T_{symbol}$ .

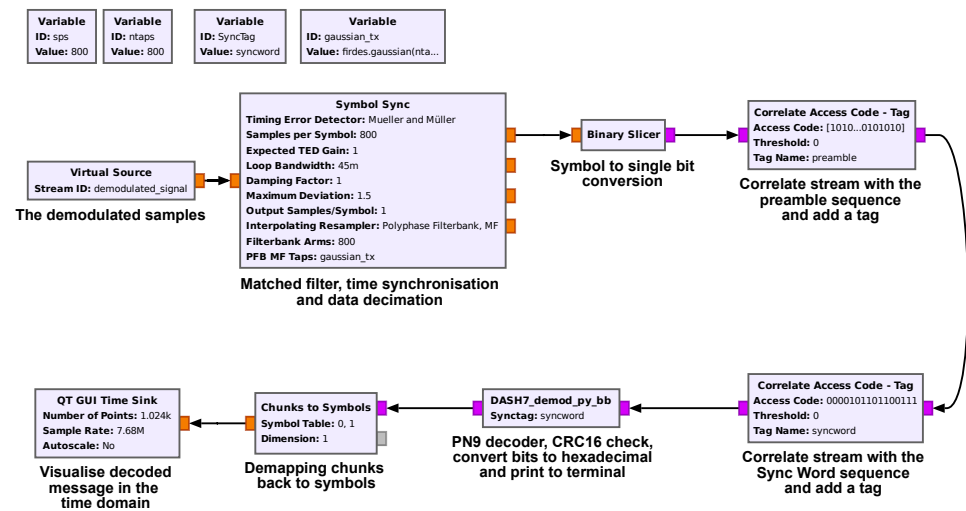
A clock signal can be extracted from the received data signal. Many techniques can be used at the receiver to achieve self-synchronisation. However, in this paper, we adopted the Mueller and Müller symbol synchronisation method [27]. The details of this method are out of this paper's scope.

The time recovery process is the first stage after demodulating the received samples, as shown in Figure 14. In GNU Radio, the *Symbol Sync* block provides several symbol synchronisation methods, including the Mueller and Müller method; every symbol synchronisation method has its specific parameters. Furthermore, the *Symbol Sync* allows the definition of any type of match filter when using the Polyphase Filterbank (PF). The input of the *Symbol Sync* are samples with a sample rate equal to  $1/T$ . On the other hand, the output of the *Symbol Sync* are symbols with a symbol rate of  $1/T_{symbol}$ .

### 5.2.4. Decoding

The first stage after the symbol synchronisation is the binary conversion process. The *Binary Slicer* block in Figure 14 converts every negative symbol value to a 0 b and the rest

to 1 b. After converting the symbols to bits, the correlation process will start between the received bits and both the preamble and the sync word. The *Correlate Access Code-Tag* block provides a tag that indicates the location of the preamble and the sync word in the data stream. The tags will be exploited by the *DASH7\_demod\_py\_bb* block. The *py\_bb* in the block name indicates that the block is implemented using Python code, and it receives and delivers binary data. The *DASH7\_demod\_py\_bb* block decodes the received packet. These are the bits after the sync word tag with PN9 sequences. After the decoding process, the block deploys a CRC16 to validate the correct reception of the received packet.



**Figure 14.** Final GNU Radio flowgraph, where the demodulated samples are time-synchronized and put through a matched filter. After this step, a tag is added when a correlation is found to a predefined preamble and sync word sequence. Finally, the message is PN9-decoded, a CRC check is applied, and the output will be printed to the console window.

The output of the decoding process is shown in the GNU Radio Console, which states the number of the packet, the packet length, the calculated CRC check, and the message in decimal and hexadecimal form, as can be seen in Figure 15. Note that the payload size of a DASH7 packet is longer due to extra obligatory fields of the other layers.

```

Found packet 1 sync word at location 10,200
2024-10-06 14:43:46.931153
MsgLength = 25 bytes
Msg = [24 1 110 32 33 49 55 52 52 0 45 0 23 128 110
0 32 64 0 3 0 171 205 199 20]
Hex:
18 01 6e 20 21 31 37 34 34 00 2d 00 17 80 6e 00
20 40 00 03 00 ab cd c7 14
CRC read = [1 1 0 0 0 1 1 1 0 0 0 1 0 1 0 0]
CRC calc = [1 1 0 0 0 1 1 1 0 0 0 1 0 1 0 0]
Check is True
correlate_access_code_tag_bb :debug:
writing tag at sample 11,577
    
```

**Figure 15.** Decoded output of a DASH7 packet with a length of 25 bytes as seen in the console window of GNU Radio. The packet has a payload of three bytes [0x00, 0xAB, 0xCD] followed by two CRC bytes.

## 6. Experimental Setup

The correct operation of the created software system was achieved by executing several experiments. Therefore, we collected three data sets. The first data set was obtained through a cabled setup. Consequently, we recorded a data set in an office environment followed by a third data set recorded from a suburban environment.

The cabled setup served as an optimal condition data set. Within this data set, three Lo-Rate channels were recorded, namely channel 0, 93, and 186. A total of 10 data transmissions were sent on each channel. Furthermore, we recorded two wireless data sets: one at an indoor location and one in a suburban setting. The indoor experiment was conducted at 10 locations, while the suburban experiment contained 21 predetermined locations. Each data set was tested on the aforementioned Lo-Rate channels. For each location of both wireless data sets, we transmitted 60 packets in total. Additionally, we logged the timestamp, channel, payload, RSSI, and EIRP from each packet on separate DASH7 gateways.

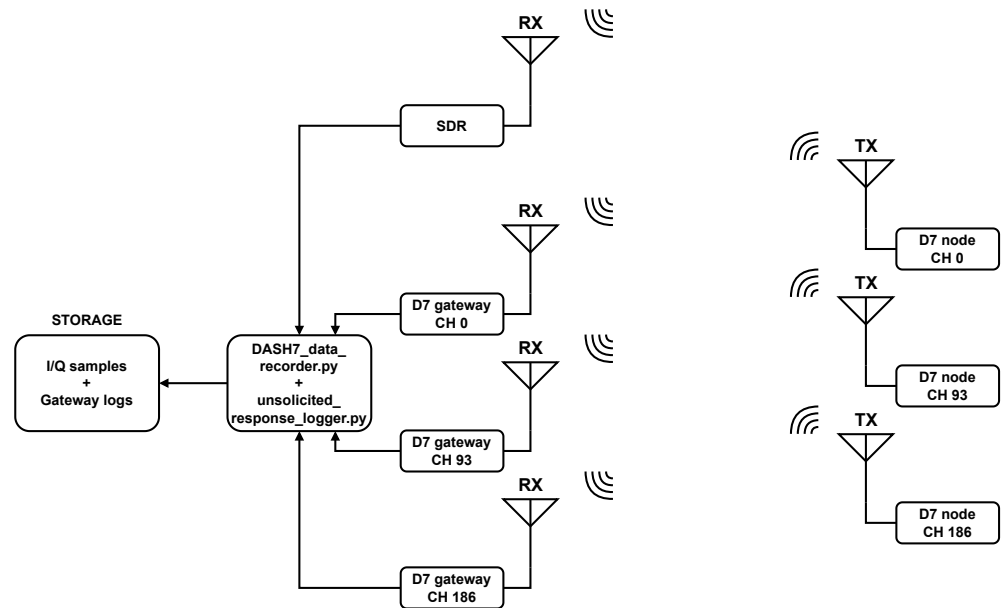
The measurement setup included six B-L072Z-LRWAN1 STM32 LoRaWAN Discovery Boards. Three development boards acted as DASH7 nodes or transmitters, which operated at different Lo-Rate channels. At the receiving end, three boards were configured as dedicated DASH7 gateways, where the incoming packets were demodulated and logged. The boards were programmed with the use of the Sub-IoT-Stack, which contained a full and open-source software stack of the DASH7 Alliance Protocol [9]. The nodes used the push communication model and transmit DASH7 messages, i.e., foreground frames, with a configurable payload, channel, gain, and encoding scheme.

Furthermore, we used an Ettus Research USRP B210 SDR to record the received data simultaneously from the DASH7 nodes. Note that lower-cost SDRs such as the ADALM-PLUTO, for example, can be used for the same purpose of data recording. This is feasible as long as the specific frequency band and sample rate are covered by the device. We set the center frequency ( $F_c$ ) to 866.5 MHz, which lay in the middle of the operating range of the 868 MHz ISM band. By setting the sample rate to 7.68 MHz, we covered the entire bandwidth where DASH7 operated. In this way, demodulation of the received signal became more convenient. An overview of the complete measurement setup is shown in Figure 16.

The hardware setup deviated slightly for the cabled measurements compared to the wireless setups. We connected the DASH7 node, the dedicated channel gateway, and the SDR through coaxial cables, which is shown in Figure 17. The DASH7 node was equipped with a 40 dB attenuator and RF splitter to guide the signal equally to the SDR and the dedicated DASH7 gateway.

The recorded data were fed and processed by the created GNU Radio flowgraphs. For the data recording and multi-channel implementation, we used GNU Radio version 3.10.1.1 installed on a Linux machine. The flowgraphs were created using readily available blocks and Python snippets. This made the conversion to older or newer GNU Radio versions more straightforward. The recorded data sets used the Signal Metadata Format (SigMF). This format describes the recorded digital signals with metadata in a JSON format, which makes it easier to validate and interpret the data during post-processing.

The software and flowgraphs used, which are extensively discussed in previous sections, accompanied by the recorded data sets and specifications, are published on Zenodo [28]. For a more concise summary of the work, we refer to [29].

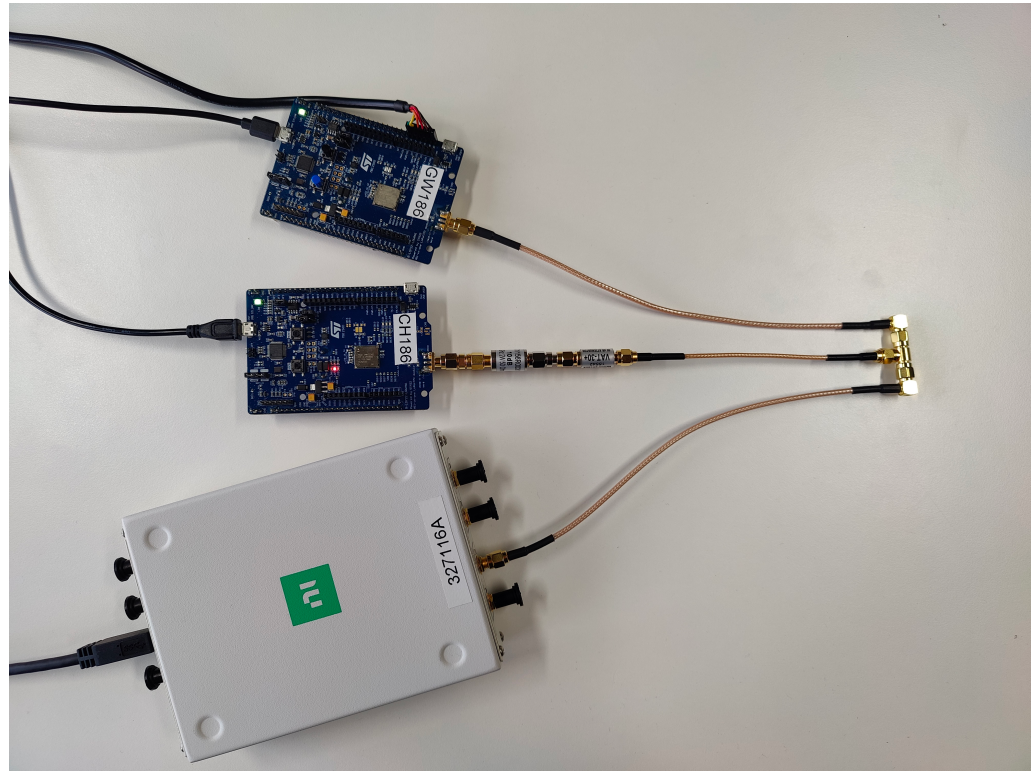


**Figure 16.** Indoor and suburban measurement setup consisting of three DASH7 nodes and three DASH7 gateways which send and receive on different Lo-Rate channels. The setup was extended with an Ettus Research (Austin, TX, USA) USRP B210 SDR, which received the transmissions of the DASH7 nodes and stored these transmissions as raw I/Q data. These captured data were investigated during post-processing analysis.

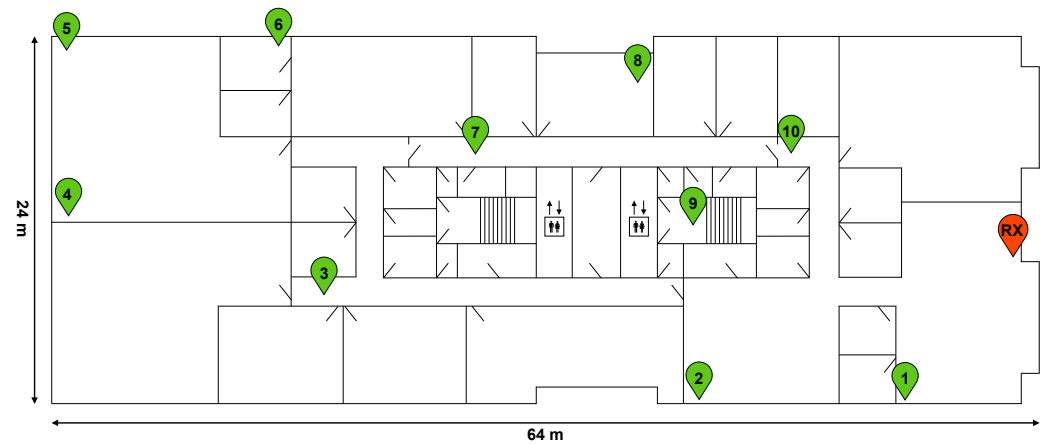
#### Measurement Environments

The indoor environment consisted of several office rooms, meeting rooms, and hallways. It generally contained several materials, such as metal structures, wooden doors, wooden walls, and glass. The offices were filled with desks, cupboards, office chairs, and plants. The hallways consisted of glass and concrete walls, whereas the floor of the entire office was carpeted. The staircase contained stone and reinforced concrete and was accessible via wooden doors. Figure 18 depicts the indoor measurement environment. The 10 TX locations are shown as green indicators on the map while the receiving setup, i.e., the SDR and DASH7 gateways, are positioned at the red RX indicator. The receiving setup was placed entirely indoors.

Figure 19 shows the map of the suburban measurement environment. The DASH7 gateways and SDR are positioned at the red pointer, while the TX locations are shown in green. Many of these locations had LOS, while others did not. In the suburban environment, a lot of vegetation was found, such as trees and plants. Nine points had LOS, while twelve locations had non-line-of-sight (NLOS) to the receiving point. This NLOS was created by metal fences, vegetation such as trees and bushes, small hills, and buildings. Furthermore, a private marina and shipyard also obstructed the LOS. Finally, the suburban region and receiving location were separated by a busy waterway. In Figure 19, location 21 is shown in blue. This location was the furthest measurement point while having complete NLOS. Subsequently, no packets were received, as can be seen in Table 6.



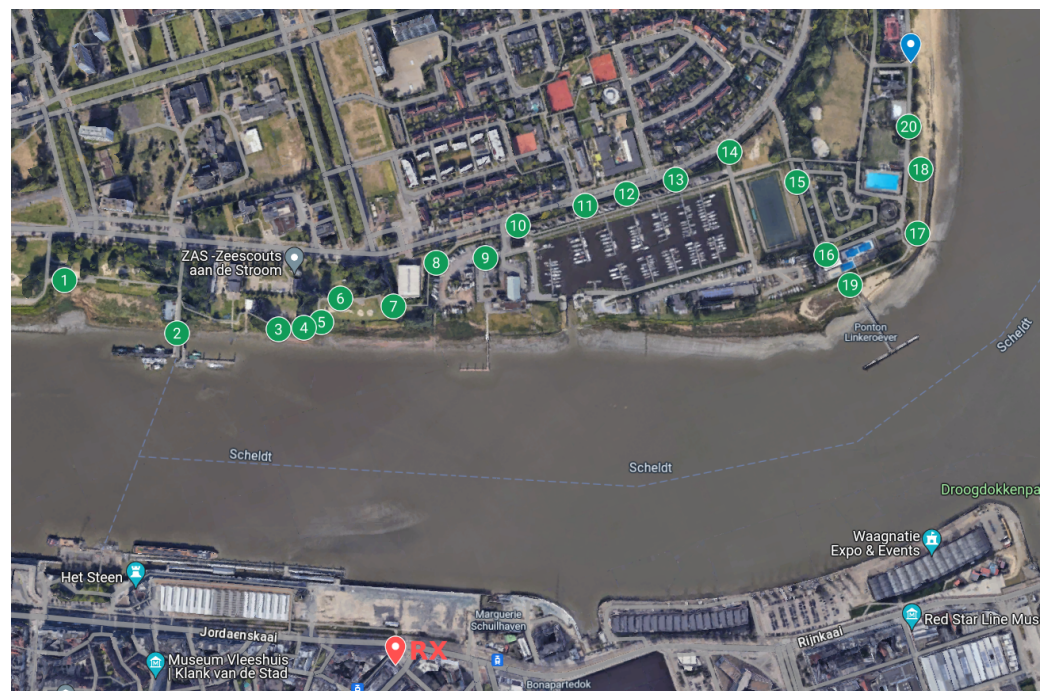
**Figure 17.** A cabled measurement setup consists of a DASH7 gateway, a DASH7 node having a 40 dB attenuator and a Software-Defined Radio, which are connected through coaxial cables.



**Figure 18.** Map of the indoor measurement environment. The 10 TX locations are shown as green pins while the receiving setup, i.e., the SDR and DASH7 gateways, are positioned at the red RX pin. The office spans an area of 64 by 24 m.

**Table 6.** Comparison of the packet loss found at each DASH7 gateway (GW) and packet loss in software (SDR) per location for three DASH7 Lo-Rate channels collected during the suburban experiment. Furthermore, the logged mean received signal strengths (GW mean RSS) at each DASH7 gateway and the gateway-to-node distance are shown.

Location	CH 0			CH 93			CH 186			Distance (m)	(N)LOS
	GW Mean RSS	GW Packet loss	GRC Packet loss	GW Mean RSS	GW Packet loss	GRC Packet loss	GW Mean RSS	GW Packet loss	GRC Packet loss		
	(dB)	(%)	(%)	(dB)	(%)	(%)	(dB)	(%)	(%)		
1	-91.31	35	50	-93.16	5	10	-89.30	0	60	806	NLOS
2	-94.64	30	30	-83.65	0	75	-94.35	0	100	631	NLOS
3	-71.47	5	0	-76.80	0	0	-77.20	0	0	565	LOS
4	-78.74	5	0	-67.00	0	0	-70.50	0	0	556	LOS
5	-70.35	5	5	-67.80	0	0	-76.70	0	0	558	LOS
6	-68.55	0	0	-68.75	0	0	-69.60	0	0	590	LOS
7	-61.30	0	0	-67.65	0	0	-77.80	0	0	572	LOS
8	-85.28	10	10	-81.20	0	5	-90.94	15	15	643	NLOS
9	-66.80	0	0	-64.35	0	0	-73.55	0	15	663	LOS
10	-89.53	5	0	-75.40	0	40	-81.37	5	5	725	NLOS
11	-88.18	45	0	-84.00	5	0	-74.95	0	5	789	NLOS
12	-77.67	10	10	-76.80	0	0	-83.80	0	10	833	NLOS
13	-93.64	30	80	-88.82	15	35	-89.21	5	10	889	NLOS
14	-81.90	5	10	-88.55	0	0	-82.30	0	15	973	NLOS
15	-83.20	0	0	-94.13	25	30	-85.70	0	0	997	LOS
16	-91.29	65	55	-87.80	0	15	-87.42	40	25	944	NLOS
17	-70.65	0	0	-67.70	0	0	-66.60	0	0	1070	LOS
18	-88.11	5	30	-85.25	0	10	-90.05	5	50	1240	NLOS
19	-67.45	0	0	-63.95	0	0	-64.85	0	0	940	LOS
20	-93.00	40	100	/	/	15	-90.75	60	75	1180	NLOS
21	/	100	100	/	100	100	/	100	100	1260	NLOS



**Figure 19.** Map of the suburban measurement environment. The 21 TX locations are shown as green pins while the receiving setup, i.e., the SDR and DASH7 gateways, are positioned at the red RX pin. The blue marker indicates measurement point 21.

## 7. Results and Discussion

We validated the proper working of the created software by using a cabled setup. The transmitted signals were attenuated by 40 dB. This attenuated signal was split, causing an additional 3 dB attenuation. In this way, the signal was distributed to the dedicated DASH7 gateway and the SDR simultaneously. For 10 measurements per channel, we observed mean received signal strengths between 21 dB and 22 dB. Subsequently, no packet loss was registered at the gateway and from the recorded data. This implies that our created software works as expected. When investigating the wireless indoor experiments, we observed very good results, although, at some points, partial packet loss was found. We have to note that 90% of the TX locations were NLOS and transmitters were placed at challenging locations throughout the office environment.

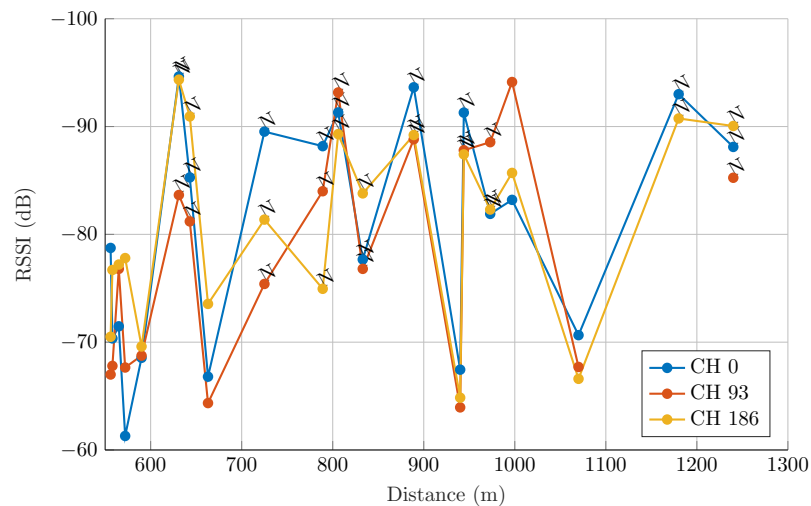
Subsequently, we see that the results of the indoor experiments are better compared to the suburban data set. The reason for this higher packet loss in the suburban environment is due to multipath effects, NLOS scenarios, and changes in distance between the transmitting locations and receiver setup. We also believe that we reached the limit of the maximum distance from where DASH7 can operate. It was found that the dedicated DASH7 gateways occasionally had a hard time detecting and decoding the packets properly. We also observed that packet loss occurred more often when the mean RSS was equal to or lower than  $-80$  dB; however, this was not always the case. We believe this was due to the received signal being close to or even below the noise floor.

For the suburban data set, we found that packets were received even at a distance of 1180 m, which is within the expectations of the technology. Locations 1 and 2 were NLOS points. The obstruction was formed by buildings at the receiving end. Locations 3 to 7 were effectively LOS locations; however, on channel 0, a low level of packet loss was found. We noticed that there were other signals transmitting at or around the same frequency during experiments. This can explain why the packet loss was substantially higher in general at channel 0. Location 8 was an NLOS location with low packet loss. Location 9 was an LOS location, but a small packet loss was observed in the software. Locations 10 to 14 lay behind the marina and were further obstructed by vegetation and metal fences, which again caused quite some packet loss. Location 15 was an (N)LOS location, and partial obstruction from buildings, vegetation, and metal fences was observed. Although channel 0 and channel 186 performed well, channel 93 experienced low packet loss. Locations 16, 18, and 20 were NLOS locations, but at a long distance. At these locations, a partial packet loss was observed, which was due to buildings and vegetation obstructing the view. Locations 17, 18, 20, and 21 were measured on an embankment, whereas locations 17 and 19 were LOS locations that did not experience packet loss. Locations 18 and 20 were the second- and third-furthest locations while being NLOS locations. Also, here, we observed severe packet loss. Location 21 was a worst-case location with no line of sight while located at the furthest measurement point. A summary of the obtained results of the indoor and suburban experiments can be found in Table 6 and Table 7, respectively. We believe that the majority of the packet loss is attributable to NLOS measurement points and not necessarily distance. In many cases, when packet loss was observed, the loss was equally significant or even worse when we put the recorded data through the created software. Figure 20 shows the relation between distance and mean RSS for each channel measured at the suburban environment. We note that the RSS fluctuates heavily, independent of the distance. However, the majority of the NLOS happens at  $-75$  dB or lower.

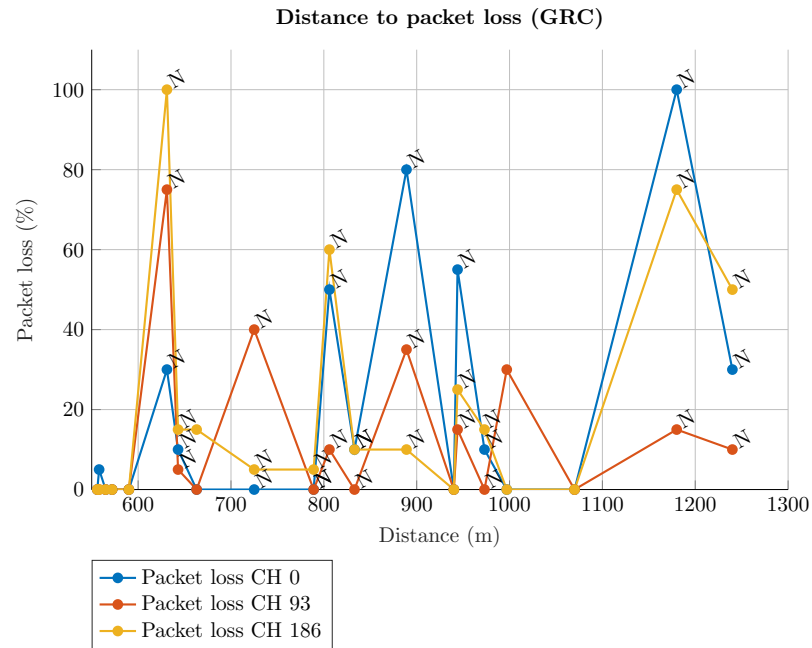
**Table 7.** Mean received signal strength and packet loss per location for three DASH7 Lo-Rate channels collected during the indoor experiment.

Location	CH 0			CH 93			CH 186			(N)LOS
	GW Mean RSS	GW Packet Loss	SDR Packet Loss	GW Mean RSS	GW Packet Loss	SDR Packet Loss	GW Mean RSS	GW Packet Loss	SDR Packet Loss	
	(dB)	(%)	(%)	(dB)	(%)	(%)	(dB)	(%)	(%)	
1	-27.45	0	0	-22.35	0	0	-31.20	0	0	LOS
2	-32.15	0	0	-36.59	15	20	-40.60	0	5	NLOS
3	-58.70	0	0	-60.35	0	0	-51.55	0	0	NLOS
4	-92.57	30	25	-80.95	0	0	-90.39	10	10	NLOS
5	-83.79	5	0	-78.45	0	0	-89.20	0	5	NLOS
6	-76.80	0	0	-76.10	0	0	-97.60	50	5	NLOS
7	-67.85	0	0	-60.80	0	0	-62.50	0	0	NLOS
8	-60.26	0	0	-64.79	0	0	-64.95	0	0	NLOS
9	-61.84	5	0	-66.50	0	0	-66.80	0	0	NLOS
10	-58.95	0	0	-40.70	0	0	-45.00	0	0	NLOS

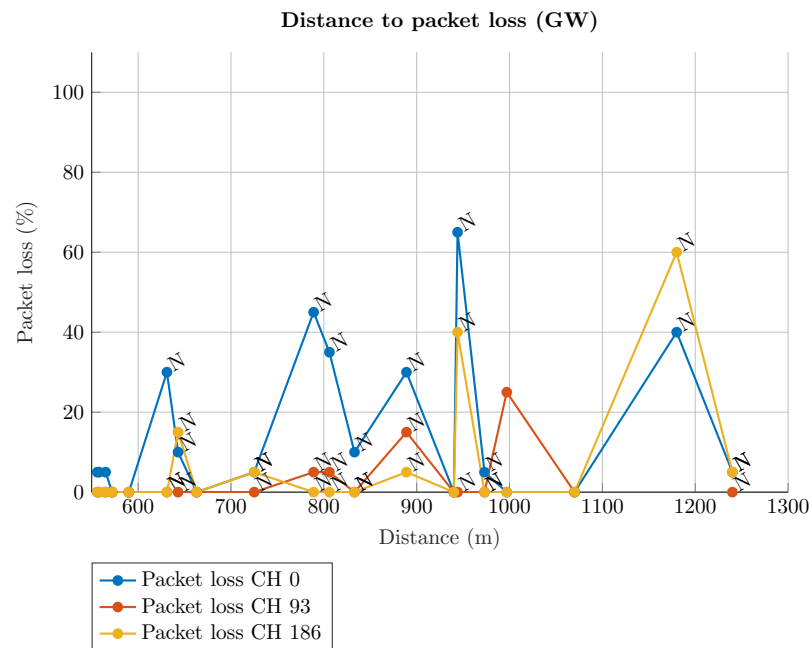
Figure 21 shows the relationship between packet loss and distance per channel for the suburban environment when the data were processed through the created software. Figure 22 shows the same relationship with the exception that it shows the packet loss at the dedicated DASH7 gateways. Note that the packet loss is generally lower compared to Figure 21. It is noteworthy that the majority of packet loss occurs when there is NLOS.



**Figure 20.** Overview of the relation between RSSI and distance for each received channel measured at the suburban environment. The “N” stands for NLOS locations.



**Figure 21.** An overview of the relationship between distance and packet loss based on measurements from the suburban environment. The data for each received channel were captured by the SDR. The “N” stands for NLOS locations.



**Figure 22.** An overview of the relationship between distance and packet loss based on measurements from the suburban environment. The data for each received channel were captured by dedicated DASH7 gateways. The “N” stands for NLOS locations.

Eventually, we found that there was less packet loss at the DASH7 gateways compared to the post-processed data in GNU Radio of the indoor data set. The findings for the suburban data set are slightly different. At certain locations and channels, the software processing outperformed the DASH7 gateways. But in other cases, it performed equally or worse. There are diverse factors that can explain the cause of it. Firstly, multipath, difference in noise floor, and other signals such as LoRa were randomly seen within the data set, which affected the transmitted signals and, therefore, can explain the packet

loss. Also, at channel 0, other signals were observed occasionally. Another reason why the gateway outperformed the SDR recordings was due to sensitivity. It is found that the receiver sensitivity of the B-L072Z-LRWAN1 board was  $-137$  dBm. The sensitivity of the applied SDR was not advertised and depended on multiple factors. However, we believe that the sensitivity of the SDR was worse, taking into account the large bandwidth compared to a dedicated, small bandwidth device.

Another explanation for the increased data loss is time variances between the packets within one recording. We observed that these dynamic time variances between consecutive packets were between 1 ms and 2 ms. This caused phase shifts in comparison to the locked frequency and phase of the local oscillator, which caused the intermediate frequency to be affected by each received packet. We observed that more or even other packets were decoded successfully by changing the phase offset of the local oscillator.

Another observation within the recordings was a recurring static frequency deviation of about 1.5 kHz that needed to be adjusted at the first mixing stage in the receiver flow-graph. This is a Center Frequency Offset (CFO), which is caused by the frequency deviation of the Temperature Compensated Crystal Oscillator (TCXO) of the B-L072Z-LRWAN1 boards (STMicroelectronics, Geneva, Switzerland), which operates at 32 MHz. This TCXO provides a clock signal to the SX1276 transceiver chip (Semtech, Camarillo, CA, USA) and is used by the DASH7 transmitters and gateways. According to the datasheet, these crystals have a deviation of two parts per million (ppm) [30].

## 8. Conclusions

In this work, we have presented a communication system based on the DASH7 Alliance Protocol by utilizing GNU Radio and Software-Defined Radios. The complete communication system works in simulation, which makes the software suitable as a learning, analysis, and validation tool for DASH7 packets. Furthermore, the demodulation system was deployed and examined in a cabled setup and in an indoor and suburban environment. The obtained data sets are open-source and can be accessed on Zenodo. The cabled setup and indoor experiments showed that the software performed sufficiently, while the suburban experiment showed severe packet loss, which was also observed while using the dedicated DASH7 gateways. This was due to the nature of field experiments and was caused by multipath effects and varying (N)LOS conditions. Nevertheless, certain optimizations can be performed to decrease the packet loss within the software.

In future work, we envision several software optimizations. Currently, a lot of parameters are defined statically, which restricts the abilities of the software. To make the system more robust, we can implement certain DASH7-specific parameters dynamically. These parameters involve correlation with a varying length of the preamble, varying Sync Word Classes and channel classes. When taking these parameters into account, other system parameters need to be changed dynamically as well, for example, filter and gain blocks. Lastly, it is needed to implement the  $\frac{1}{2}$ -FEC technique.

On the level of digital signal processing, we find a few shortcomings in the current implementation. Firstly, an automatic Center Frequency Offset compensation block needs to be created. Secondly, the mixing stage of the demodulator should be improved to a self-centering technique. In this way, the signal is automatically centered at baseband level, which implies that it does not matter which specific channel a signal is received on. Thirdly, a phase offset measuring step and phase compensation step need to be added while maintaining the mixing stage. Furthermore, the Normal-Rate and Hi-Rate channel classes need to be implemented and validated. Penultimately, the dynamical detection of the used Sync Word Class and coding scheme for a certain packet should be implemented.

Finally, the implementation of loading a configuration file containing several DASH7 specification-specific parameters can make the system more dynamic and less error-prone.

**Author Contributions:** Conceptualization, N.B.; Methodology, D.J. and N.B; Software, D.J. and N.B.; Investigation, D.J.; Data curation, D.J.; Writing—original draft, D.J.; Supervision, N.B., R.B. and M.W. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** The data set presented and used in this study is openly available at <https://doi.org/10.5281/zenodo.10961310> (accessed on 10 September 2024).

**Conflicts of Interest:** The authors declare no conflicts of interest.

### Appendix A. DASH7 CRC16 Validation

A cyclic redundancy check (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. Blocks of data entering these systems receive a short check value attached, which is based on the remainder of a polynomial division of their contents. On retrieval, the calculation is repeated. When the check values do not match, corrective action can be taken against data corruption. A frame is always terminated by a 16-bit CRC field. Only when the CRC calculation of a received frame matches the supplied value will it be sent to the Data Link Layer, which continues processing the frame. The calculation of the field includes all previous bytes of the frame. For DASH7, the CRC16/CCITT\_FALSE polynomial is used. This is expressed as  $x^{16} + x^{12} + x^5 + x^0$  (0x1021 or 10001000000100001B) with an initial value of 0xFFFF. Using this algorithm correctly, the value 0x29B1 will result from the computation of the nine-character ASCII reference string “123456789” (note: the numbers of the reference string are ASCII-encoded, not binary). To compute an n-bit binary CRC, line the bits representing the input in a row, and position the (n + 1)-bit pattern representing the CRC’s divisor (called a “polynomial”) underneath the left-hand end of the row.

To make the operation straightforward, we encode a one-byte message with a 3-bit CRC, which uses the polynomial  $x^3 + x + 1$ . The polynomial is written in binary as the coefficients; a third-order polynomial has four coefficients ( $1x^3 + 0x^2 + 1x + 1$ ). In this case, the coefficients are 1, 0, 1, and 1. The result of the calculation is 3 bits long.

We start with the message to be encoded: 11001100. This is first padded with zeros corresponding to the bit length (n) of the CRC (add 0 0 0 at the end of the message), and an initial value of n bits is added at the beginning of the message (add 1 1 1 at the beginning of the message). The first calculation for computing a 3-bit CRC is as follows:

$$\begin{array}{r}
 111\ 11001100\ 000 \\
 101\ 1 \qquad \qquad \qquad \leftarrow \text{divisor (4 bits)} = x^3 + x + 1 \\
 \hline
 010\ 01001100\ 000 \leftarrow \text{result}
 \end{array}$$

The algorithm acts on the bits directly above the divisor in each step. The result for that iteration is the bitwise XOR of the polynomial divisor with the bits above it. The bits not above the divisor are simply copied directly below for that step. The divisor is then shifted one bit to the right, and the process is repeated until the divisor reaches the right-hand end of the input row. The entire calculation is written out below.

<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	1	0	0	1	1	0	0	<b>0</b>	<b>0</b>	<b>0</b>	
	1	0	1	1											
<b>2</b>	0	1	0	0	1	0	0	1	1	0	0	0	0	0	
		1	0	1	1										
<b>3</b>	0	0	0	1	0	0	0	1	1	0	0	0	0	0	
				1	0	1	1								
<b>4</b>	0	0	0	0	0	1	1	1	1	0	0	0	0	0	
						1	0	1	1						
<b>5</b>	0	0	0	0	0	0	1	0	0	0	0	0	0	0	
							1	0	1	1					
<b>6</b>	0	0	0	0	0	0	0	0	1	1	0	0	0	0	
									1	0	1	1			
<b>7</b>	0	0	0	0	0	0	0	0	0	1	1	1	0	0	
										1	0	1	1		
<b>8</b>	0	0	0	0	0	0	0	0	0	0	1	0	1	0	
											1	0	1	1	
<b>9</b>	0	0	0	0	0	0	0	0	0	0	0	0	<b>0</b>	<b>0</b>	<b>1</b>

Since the leftmost divisor bit zeroed every input bit it touched, when this process ends, the only bits in the input row that can be nonzero are the n bits at the right-hand end of the row. These n bits are the remainder of the division step, and will also be the value of the CRC function. The validity of a received message can easily be verified by performing the above calculation again, this time with the check value added instead of zeros. The remainder should be equal to zero if there are no detectable errors.

<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	1	0	0	1	1	0	0	<b>0</b>	<b>0</b>	<b>1</b>	
	1	0	1	1											
<b>2</b>	0	1	0	0	1	0	0	1	1	0	0	0	0	1	
		1	0	1	1										
<b>3</b>	0	0	0	1	0	0	0	1	1	0	0	0	0	1	
				1	0	1	1								
<b>4</b>	0	0	0	0	0	1	1	1	1	0	0	0	0	1	
						1	0	1	1						
<b>5</b>	0	0	0	0	0	0	1	0	0	0	0	0	0	1	
							1	0	1	1					
<b>6</b>	0	0	0	0	0	0	0	0	1	1	0	0	0	1	
									1	0	1	1			
<b>7</b>	0	0	0	0	0	0	0	0	0	1	1	1	0	1	
										1	0	1	1		
<b>8</b>	0	0	0	0	0	0	0	0	0	0	1	0	1	1	
											1	0	1	1	
<b>9</b>	0	0	0	0	0	0	0	0	0	0	0	0	<b>0</b>	<b>0</b>	<b>0</b>

As you may notice at step three in the previous example, when the next bit of the message is zero, the divisor skips it and goes to the bit after it. This skipping operation will continue until the divisor reaches one bit.

### Appendix B. Forward Error Correction

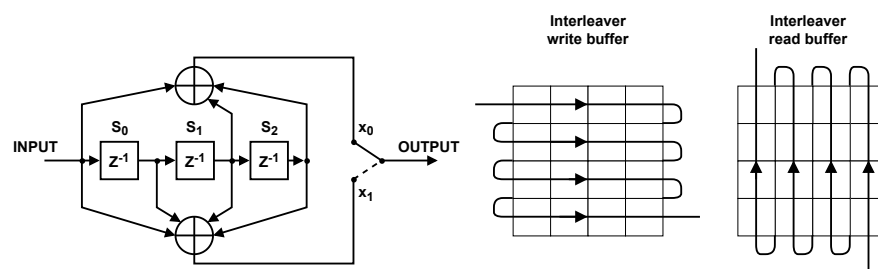
Before the data whitening step in DASH7, an optional Forward Error Correction (FEC) step can be applied. FEC is a channel encoding step that adds redundancy bits to the transmitted data using a predetermined algorithm, making the data more robust against bit errors. In the case of DASH7, the encoding process includes two stages.

The first stage includes a 1/2 rate non-recursive convolutional encoding process with a constraint length (L) of 4. The 1/2 base coding rate indicates that each input bit will be encoded into two output bits. The encoder itself consists of three shift registers where an XOR operation is applied twice in a specific manner, eventually gaining two output bits.

The second stage applies a 32-bit interleaving process executed on every 2-bit symbol. This interleave/deinterleave process scrambles and separates adjacent symbols, thus lowering the impact of bursty errors that mostly span multiple bits and can be induced by interference or time-varying signal strengths. In this way, the process increases the robustness against these errors [31]. In order to have a properly working interleaving process, a convolutional code trellis terminator is also appended to the unencoded data. This sequence is 0x0B0B for even-length byte inputs and 0x0B0B0B for odd-length byte inputs. In the DASH7 protocol, a 4x4 matrix interleaver is used.

The bits coming from the convolutional coder are written into the rows of the matrix interleaver while the bits that will be sent are read from the columns of the matrix in a certain order. The interleaving operation is applied per 2-bit symbols. This means that the data are scrambled per 32 aligned bits. Note that FEC is only applied when using Coding Scheme 2 of DASH7, where the data are firstly FEC-encoded and afterwards PN9-encoded. Figure A1 depicts the operation of the encoder stage and interleaving stage. The following example is based on [32], whereas Figures A2 and A3 show the filled buffers of the interleaver. The output bytes are highlighted in bold.

Input [4 B]	0x03	0x01	0x02	0x03												
Appended CRC [6 B]	0x03	0x01	0x02	0x03	0x7E	0x2D										
Appended Trellis terminator [8 B]	0x03	0x01	0x02	0x03	0x7E	0x2D	0x0B	0x0B								
FEC encoder output [16 B]	00	0E	8C	03	7C	0D	F0	0E	B5	A9	3D	1B	BC	D1	8C	D1
Interleaver output [16 B]	C8	<b>3C</b>	<b>00</b>	<b>20</b>	<b>84</b>	<b>CF</b>	<b>33</b>	<b>31</b>	<b>D5</b>	<b>B9</b>	<b>7B</b>	<b>0A</b>	<b>44</b>	<b>33</b>	<b>37</b>	<b>EE</b>



**Figure A1.** Implementation of the FEC operation in DASH7 starting with a convolutional encoder with configuration  $(n,k,m) = (2,1,3)$ , where  $n$  is the number of output bits,  $k$  is the number of input bits, and  $m$  is the number of shift register stages supplemented with the  $4 \times 4$  matrix interleaver.

00	00	00	00	00	
00	00	11	10	0E	
10	00	11	00	8C	
00	00	00	11	03	
<b>20</b>	<b>00</b>	<b>3C</b>	<b>C8</b>	R/W	

➔

01	11	11	00	7C	
00	00	11	01	0D	
11	11	00	00	F0	
00	00	11	10	0E	
<b>31</b>	<b>33</b>	<b>CF</b>	<b>84</b>	R/W	

**Figure A2.** Buffer 1 and 2.

10	11	01	01	B5
10	10	10	01	A9
00	11	11	01	3D
00	01	10	11	1B
<b>0A</b>	<b>7B</b>	<b>B9</b>	<b>D5</b>	R/W

→

10	11	11	00	BC
11	01	00	01	D1
10	00	11	00	8C
11	01	00	01	D1
<b>EE</b>	<b>37</b>	<b>33</b>	<b>44</b>	R/W

Figure A3. Buffer 3 and 4.

### Appendix C. PN9 Coding

The 9-bit pseudo-random number (“PN9”) generator is shown at the top of Figure A4. The generator is described by the polynomial  $x^9 + x^5 + x^0$ . The PN9 generates all the values between 1 through 511 (inclusively) in a pseudo-random order as it is clocked. The latches are all set to ones at the start of a whitening operation.

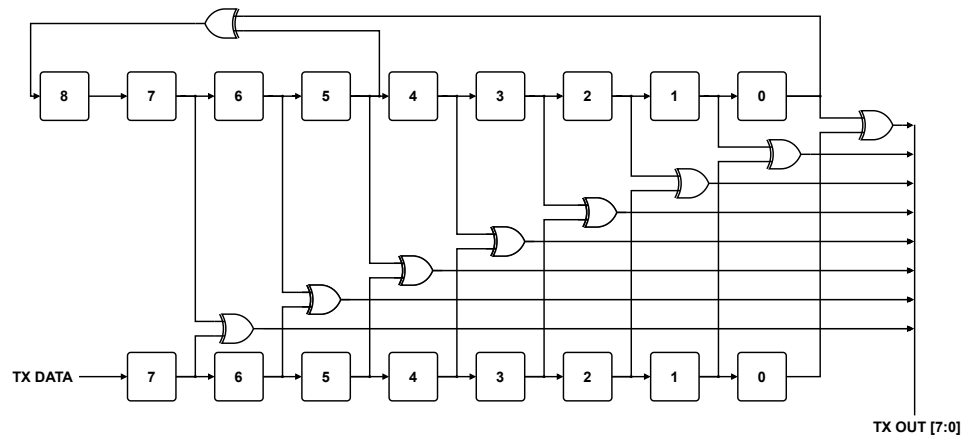


Figure A4. PN9 coding circuit.

Figure A5 illustrates the operation of the PN9 generator as it is clocked. The generator starts with a value of all ones “11111111”. Bit 0 (the LSB) and bit 5 are XOR’d to produce 0 that is shifted into the MSB on the next clock. This results in the next value of the generator being “01111111”. Again, bits 0 and 5 are XOR’d to give the next value of the MSB (again, 0) to produce the next value of the generator “00111111”, and the process continues through all 511 states of the generator.

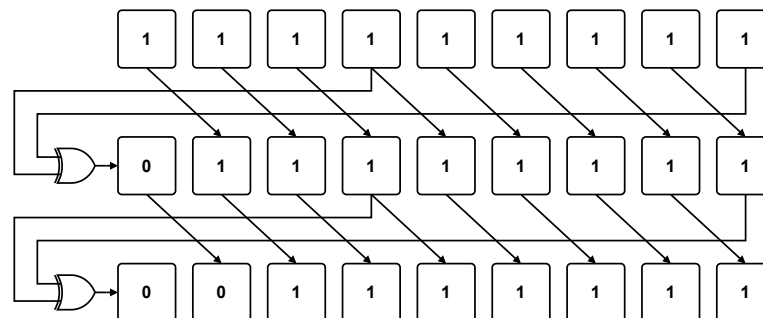


Figure A5. Operation of the PN9 generator.

The lower set of latches in Figure A4 contains the current byte of the data to be whitened. Recall that the preamble and sync words are not whitened, so the first byte of data to be whitened is either the packet length byte (in variable packet mode) or the first byte of user data.

When the first byte of data is ready, it is XOR’d with the eight LSBs of the initial value of the PN9 generator, which are all ones. So, the PN9 generator is “11111111”, and the data

are XOR'd with the eight LSBs of this, which is "11111111". This whitened data are then transmitted over the air. Next, the second byte of data is shifted into the data latches (eight shifts, one per bit), while the PN9 generator is also clocked eight times (once per bit). This means that the second byte of user data is not XOR'd with the next PN9 value but instead by the value eight later in the PN9 sequence. Thus, the PN9 generator contains "11111111" when the first byte is processed, while the PN9 generator state during the second byte of data is not "01111111" (the next PN9 value), but instead "11110001" (the ninth PN9 value).

Suppose the (unwhitened) data sequence to be transmitted starts out as follows (these would be the bytes, for example, if a packet with a packet length byte of ten were transmitted, and the data were 0x00, 0x01,...): /0000 1010/0000 0000/0000 0001/0000 0010/... Remember that the values to XOR with the data are the eight LSBs of the PN9 sequence if every eighth value is used:

```

1,1,1,1,1,1,1,1 *** Eight LSBs are 1111 1111
0,1,1,1,1,1,1,1
0,0,1,1,1,1,1,1
0,0,0,1,1,1,1,1
0,0,0,0,1,1,1,1
1,0,0,0,0,1,1,1
1,1,0,0,0,0,1,1
1,1,1,0,0,0,0,1
1,1,1,0,0,0,0,1 *** Eight LSBs are 1110 0001
0,1,1,1,0,0,0,0
1,0,1,1,1,0,0,0
1,1,0,1,1,1,0,0
1,1,1,0,1,1,1,0
0,1,1,1,0,1,1,1
0,0,1,1,1,0,1,1
0,0,0,1,1,0,1,1
0,0,0,0,1,1,0,1 *** Eight LSBs are 0001 1101
1,0,0,0,0,1,1,0
0,1,0,0,0,0,1,1
1,0,1,0,0,0,0,1
1,1,0,1,0,0,0,0
0,1,1,0,1,0,0,0
0,0,1,1,0,1,0,0
1,0,0,1,1,0,1,0
1,1,0,0,1,1,0,1 *** Eight LSBs are 10011010
0,1,1,0,0,1,1,0,1

```

So, the values to XOR with the data are as follows:

```

/ 1111 1111/1110 0001/0001 1101/1001 1010/

```

Taking the exclusive-OR of these two sequences gives the data to be transmitted:

```

Data: /0000 1010/0000 0000/0000 0001/0000 0010 ...
PN9:  /1111 1111/1110 0001/0001 1101/1001 1010/ ...
Result:/1111 0101/1110 0001/0001 1100/1001 1000 /...

```

When received, the resultant data are XOR'd with the same PN9-derived sequence, resulting in the originally transmitted data.

Received: /1111 0101/1110 0001/0001 1100/1001 1000 / ...  
 PN9: /1111 1111/1110 0001/0001 1101/1001 1010/ ...  
 Data: /0000 1010/0000 0000/0000 0001/0000 0010/...

## References

- Zanella, A.; Bui, N.; Castellani, A.; Vangelista, L.; Zorzi, M. Internet of Things for Smart Cities. *IEEE Internet Things J.* **2014**, *1*, 22–32. [CrossRef]
- Centenaro, M.; Vangelista, L.; Zanella, A.; Zorzi, M. Long-range Communications in Unlicensed Bands: The Rising Stars in the IoT and Smart City Scenarios. *IEEE Wirel. Commun.* **2016**, *23*, 60–67. [CrossRef]
- Bni Lam, N.H. Angle of Arrival Estimation for Low Power and Long Range Communication Networks. Ph.D. Thesis, University of Antwerp, Faculty of Applied Engineering, Antwerp, Belgium, 2021. Available online: <https://repository.uantwerpen.be/docstore/d:irua:6256> (accessed on 24 December 2024).
- BniLam, N.; Janssens, R.; Steckel, J.; Weyn, M. AoA Estimates for LPWAN Technologies: Indoor Experimental Analysis. In Proceedings of the 2021 15th European Conference on Antennas and Propagation (EuCAP), Düsseldorf, Germany, 22–26 March 2021; pp. 1–5. [CrossRef]
- BniLam, N.; Steckel, J.; Weyn, M. Synchronization of Multiple Independent Subarray Antennas: An Application for Angle of Arrival Estimation. *IEEE Trans. Antennas Propag.* **2019**, *67*, 1223–1232. [CrossRef]
- BniLam, N.; Joosens, D.; Steckel, J.; Weyn, M. Low Cost AoA Unit for IoT Applications. In Proceedings of the 2019 13th European Conference on Antennas and Propagation (EuCAP), Krakow, Poland, 31 March–5 April 2019; pp. 1–5.
- BniLam, N.; Ergeerts, G.; Subotic, D.; Steckel, J.; Weyn, M. Adaptive probabilistic model using angle of arrival estimation for IoT indoor localization. In Proceedings of the 2017 International Conference on Indoor Positioning and Indoor Navigation (IPIN), Sapporo, Japan, 18–21 September 2017; pp. 1–7. [CrossRef]
- DASH7 Alliance. Available online: <http://www.dash7-alliance.org> (accessed on 10 September 2024).
- Sub-IoT. Available online: <https://sub-iot.github.io/Sub-IoT-Stack> (accessed on 10 September 2024).
- Mekki, K.; Bajic, E.; Chaxel, F.; Meyer, F. A Comparative Study of LPWAN Technologies for Large-scale IoT Deployment. *ICT Express* **2019**, *5*, 1–7. [CrossRef]
- Semtech. Available online: <https://www.semtech.com> (accessed on 10 September 2024).
- Janssen, T.; BniLam, N.; Aernouts, M.; Berkvens, R.; Weyn, M. LoRa 2.4 GHz communication link and range. *Sensors* **2020**, *20*, 4366. [CrossRef] [PubMed]
- BniLam, N.; Nasser, S.; Weyn, M. Angle of Arrival Estimation System for LoRa Technology based on Phase Detectors. In Proceedings of the 2022 16th European Conference on Antennas and Propagation (EuCAP), Madrid, Spain, 27 March–1 April 2022; pp. 1–5. [CrossRef]
- Adefemi Alimi, K.O.; Ouahada, K.; Abu-Mahfouz, A.M.; Rimer, S. A Survey on the Security of Low Power Wide Area Networks: Threats, Challenges, and Potential Solutions. *Sensors* **2020**, *20*, 5800. [CrossRef] [PubMed]
- BniLam, N.; Joosens, D.; Aernouts, M.; Steckel, J.; Weyn, M. LoRay: AoA Estimation System for Long Range Communication Networks. *IEEE Trans. Wirel. Commun.* **2021**, *20*, 2005–2018. [CrossRef]
- Wang, Y.P.E.; Lin, X.; Adhikary, A.; Grovlen, A.; Sui, Y.; Blankenship, Y.; Bergman, J.; Razaghi, H.S. A Primer on 3GPP Narrowband Internet of Things. *IEEE Commun. Mag.* **2017**, *55*, 117–123. [CrossRef]
- Bembe, M.; Abu-Mahfouz, A.; Masonta, M.; Ngqondi, T. A survey on low-power wide area networks for IoT applications. *Telecommun. Syst.* **2019**, *71*, 249–274. [CrossRef]
- Bnilam, N.; Joosens, D.; Berkvens, R.; Steckel, J.; Weyn, M. AoA-Based Localization System Using a Single IoT Gateway: An Application for Smart Pedestrian Crossing. *IEEE Access* **2021**, *9*, 13532–13541. [CrossRef]
- Price, N.D.; Zawodniok, M.J.; Guardiola, I.G. Transceivers as a Resource: Scheduling Time and Bandwidth in Software-Defined Radio. *IEEE Access* **2020**, *8*, 132603–132613. [CrossRef]
- GNU Radio. Available online: <https://www.gnuradio.org/> (accessed on 10 September 2024).
- DASH7 Alliance. DASH7 Alliance Protocol Specification v1.2. Available online: <https://www.dash7-alliance.org/download-specification/> (accessed on 10 September 2024).
- Norair, J. *Introduction to DASH7 Technologies*, 1st ed.; 2009. Available online: <https://www.rfidjournal.com/wp-content/uploads/2019/07/269.pdf> (accessed on 24 December 2024).
- Singh, R.K.; Puluckul, P.P.; Berkvens, R.; Weyn, M. Energy Consumption Analysis of LPWAN Technologies and Lifetime Estimation for IoT Application. *Sensors* **2020**, *20*, 4794. [CrossRef] [PubMed]
- Sklar, B. *Digital Communications: Fundamentals & Applications*, 2nd ed.; Prentice Hall: Upper Saddle River, NJ, USA, 2001.
- Lyons, R. *Understanding Digital Signal Processing*, 3rd ed.; Prentice Hall: Upper Saddle River, NJ, USA, 2011.

26. Proakis, J.; Salehi, M. *Digital Communications*, 5th ed.; McGraw-Hill: New York, NY, USA, 2008.
27. Mueller, K.; Muller, M. Timing Recovery in Digital Synchronous Data Receivers. *IEEE Trans. Commun.* **1976**, *24*, 516–531. [[CrossRef](#)]
28. Joosens, D.; BniLam, N.; Weyn, M.; Berkvens, R. SDR-Based IoT Communication Systems: An Application for the DASH7 Alliance Protocol. Dataset Zenodo. 2024. Available online: <https://zenodo.org/records/10961311> (accessed on 10 September 2024).
29. Joosens, D.; BniLam, N.; Berkvens, R.; Weyn, M. Implementation of a Multi-Channel DASH7 IoT Communication System for Packet Investigation and Validation. In Proceedings of the GNU Radio Conference 2024, Knoxville, TN, USA, 16–20 September 2024.
30. Semtech. *SX1276/77/78/79–137 MHz to 1020 MHz Low Power Long Range Transceiver*. Semtech: Camarillo, CA, USA, 2020. Available online: [https://semtech.my.salesforce.com/sfc/p/#E0000000JelG/a/2R0000001Rbr/6EfVZUorrpoKFfvaF\\_Fkpgp5kzjiNyiAbqcpqh9qSjE](https://semtech.my.salesforce.com/sfc/p/#E0000000JelG/a/2R0000001Rbr/6EfVZUorrpoKFfvaF_Fkpgp5kzjiNyiAbqcpqh9qSjE) (accessed on 10 September 2024).
31. Weyn, M.; Ergeerts, G.; Wante, L.; Vercauteren, C.; Hellinckx, P. Survey of the DASH7 Alliance Protocol for 433 MHz Wireless Sensor Communication. *Int. J. Distrib. Sens. Netw.* **2013**, *9*, 870430. [[CrossRef](#)]
32. Hoel, R. *Design Note DN504: FEC Implementation*; Texas Instruments: Dallas, TX, USA, 2007; Rev. SWRA113A Available online: <http://www.ti.com/lit/an/swra113a/swra113a.pdf> (accessed on 10 September 2024).

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.