Implementation of a Differential Chaos Shift Keying Communication system in GNU Radio

Georges Kaddoum, Julien Olivain, Guillaume Beaufort Samson, Pascal Giard, François Gagnon
LaCIME Laboratory[1]
École de technologie supérieure
Montreal, Canada
georges.kaddoum@lacime.etsmtl.ca

***Abstract*—In this paper the first experimental chaos radio system using Differential Chaos Shift Keying is realized. The software design and implementation are proposed for the experimental Differential Chaos Shift Keying (DCSK) system on software defined radio (SDR) to perform in a real-time wireless transmission. The GNU Radio platform is used as a flexible and open-source platform for our implementation. In order to perform in real-time wireless scenarios, the proposed work implements a synchronization unit on the receiver side. Since our system is on a SDR, the bitrate, bandwidth and central frequency can be modified at ease. The experimental performance are discussed and compared to the theoretical predictions. Finally some trends concerning implementation are discussed.**

I. Introduction

Since Pecora and Carroll proved in 1990 that chaotic systems can be synchronized [1], there has been a growing interest in digital chaotic communications over the past several years. Given the broadband nature of chaotic signals and their good correlation properties, as well as the extreme ease with which it can be generated, they have become the preferred choice for use in spread spectrum communication systems [2].

Many communication systems based on chaos have been proposed and evaluated in the literature. From those systems, the DCSK one represents a robust non-coherent scheme in which the chaotic signal at the receiver side does not have to be known exactly. In addition, the DCSK system is one of the most promising chaos-based communications schemes for the robustness against channel imperfections [3].

The performance of chaos-based digital communication systems under additive white Gaussian noise (AWGN) and m-distributed fading channels has been thoroughly studied [4] [5] [6].

Many implementation methods of chaotic generators have been studied and proposed in the literature [7] [8]. Some of those chaotic generators operate in continuous mode such as Chua's circuit, while others are discrete time systems [9]. For chaotic communication systems, a robust chaotic signal source is necessary for a real-time radio implementation.

[1]This work has been supported in part by Ultra Electronics TCS and the Natural Science and Engineering Council of Canada as part of the 'High Performance Emergency and Tactical Wireless Communication Chair' at École de technologie supérieure.

Implementations of digital communication systems using chaotic signals have been proposed in the literature [10] [11]. An experimental blind timing acquisition scheme for a FM-DCSK communication system was studied in [12]. In this paper, we first propose a simple and robust method to implement a chaotic generator where the period loop length is sufficiently long and exactly known making this generator suitable for real-time transmissions. Second, we implement a DCSK communication system in a SDR. The synchronization unit is taken into account in our study, and an algorithm is proposed then implemented to make this system operational in real-time transmissions. We decided to publish our code in [13] under the GPLv3 license. With this open source contribution, we wish that our framework can spur further research in that field.

This paper is organized as follows. Section II describes the DCSK modulation scheme. The GNU Radio and the universal software radio peripheral (USRP) platform are presented in section III. Section IV describes the integration method of the chaotic generator, the software implementation of the DCSK system with the synchronization unit, and discusses some technical problems. Section V presents experimental results. Conclusions and suggestions for further works are presented in section VI.

II. DCSK communication scheme

A block diagram of a single user DCSK communication system is shown in Figure 1.

This system uses a chaotic carrier to spread the digital signal over a wide frequency band. As shown in Figure 1 (c), each bit $s_i = -1, +1$ is represented by two set of chaotic signal samples. The first set of chaotic samples represents the reference while the second one carries the data. If $+1$ is transmitted, the data-bearing sequence will be equal to the reference sequence, and if -1 is transmitted, an inverted version of the reference sequence will be used as the data-bearing sequence. Let 2β be the spreading factor, defined as the number of chaotic samples sent for each bit, where β is an integer.

During the i^{th} bit duration, the output of the transmitter e_k is

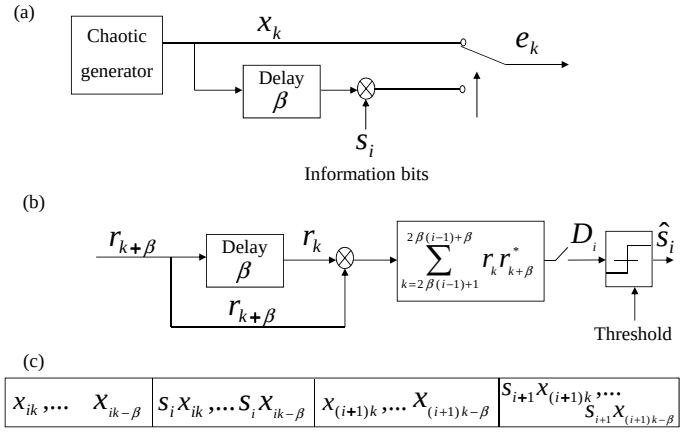

Fig. 1: (a) DCSK transmitter, (b) DCSK receiver, (c) Transmitted frame

$$e_k = \begin{cases} x_k & \text{for } 1 < k \leq \beta \\ s_i x_{k-\beta} & \text{for } \beta < k \leq 2\beta. \end{cases} \quad (1)$$

Moreover, at the receiver side, the signal is embedded in an additive white complex Gaussian noise (AWGN) n_k with two-side power spectral density equal to $2N_0$. Finally, the received signal is modeled as:

$$r_k = e_k + n_k. \quad (2)$$

The received quadrature signal r_k delayed by half a bit duration and correlated with undelayed signal. It then passes by a correlator where the reference and corresponding data samples are correlated. Then the sign of the correlator output is computed to estimate the transmitted bit. In a practical implementation many parameters like synchronization and time sampling correction must be taken into account to correctly achieve demodulation. Later we will explain all the steps leading to the implementation of the DCSK system in a real-time radio system.

III. The GNU Radio and USRP Platform

GNU Radio [14] is a free and open source software project that provides signal processing components to implement SDRs. It can be used in conjunction with a wide range of radio frequency hardware to quickly build actual implementations. That software is distributed under the terms of the GPLv3 license [15].

The *Universal Software Radio Peripheral* (USRP), is a family of radio hardware made by Ettus Research [16] intended to build SDRs. The analog radio frequency (RF) parts are grouped on interchangeable daughterboards. Ettus Research provides a wide range of such boards.

The main advantages of this hardware platform are:

- Cost: a full SDR (emitter and receiver) can be built at a very low cost.
- Open design: schematics, design plans, part lists, FPGA code and embedded system code are available.
- Modular design: With the help of schematics, the modular design allows one to modify or create a new part, for special purposes.
- Reconfigurable: all the digital modules are built around a FPGA. The behavior of the radio can be modified by replacing the FPGA code.

The USRP does not make any assumption about the underlying modulation which will be implemented in it. The major drawback is that all practical problems (filtering, synchronization, ...) will have to be dealt with in software (in our case in the GNU Radio project).

A USRP only needs few parameters to work properly: a center frequency (which should be in the supported frequency range of its daughterboard), a sampling rate (which will implicitly define the usable bandwidth) and a gain. Other parameters may be available depending on daughterboards.

For our experimentation, two USRP2 are used along with WBX RF daughterboards, GNU Radio 3.5.0 and UHD (USRP Hardware Driver) 3.4.0 on a Linux Fedora 16 system. The USRP2 includes a Xilinx Spartan-3 XC3S2000 FPGA, a 14 bits 100 Mega Samples Per Second (MSPS) ADC, a 16 bits 400 MSPS DAC, and a GigaBit Ethernet port. The USRP2 can handle from 195.3125 kHz up to 25 MHz of bandwidth. The WBX RF daughterboard can handle frequencies from 50 MHz to 2.2 GHz in emission and reception, in half duplex.

IV. Technical Description of the Implementation

Our primary goal is to provide a first real world working implementation of DCSK to the research community. Our C++ implementation, called `gr-chaos` is made available as GNU Radio blocks.

A. Chaos Generator

We chose to implement the chaotic generator based on a logistic map function for its simplicity and well known statistical properties [2]. This function is defined as $X_{n+1} = 1 - 2X_n^2$ where $X_0 \in [-1; 1]$ is defined as the initial value (*seed*) of the generator [2, p. 51-52].

Our first implementation, using 32 bits IEEE 754 floating point values (`float` type of the C++ language) was not conclusive. We made that initial arbitrary choice because it is the most common data type in GNU Radio. With such a data type, this generator quickly enters in undesired short loops

of few hundreds or thousands values. These loops are due to error accumulation, and the recursive nature of the generator. An implementation using a 32 bits fixed point representation has the same issue.

A first work around is to use a larger internal state representation: the 64 bits `double` C++ type. A quick study showed that the generator eventually enters into larger loops of few millions of values, for some initial values. We include this generator in our `gr-chaos` package.

Such looping behavior may be unacceptable, depending on the situation. Loops of few millions of values could be a problem since SDR can run at high sampling rates (25 MSPS in the case of the fastest USRPs). Moreover, the unknown length of the period loop makes the stability of an implementation on hardware very weak.

As a more reliable software solution, we propose a different approach to chaos generation, based on filtered values taken from uniform random number generators. This method is described in [17, section 3.4.1.B].

The main advantage of this technique is that we can choose a random number generator depending on its known and proven properties, and period length.

The filter for converting a uniform random generator to the distribution of a logistic map function is defined as the inverse function of a cumulative probability function. The probability density function of the logistic map is defined as [2]:

$$f(x) = \frac{1}{\pi\sqrt{1-x^2}}. \tag{3}$$

The cumulative distribution function is defined as:

$$F(x) = \int_{-\infty}^{x} f(u)\mathrm{d}u = \frac{sin^{-1}(x)}{\pi} + \frac{1}{2}. \tag{4}$$

The inverse function is:

$$F^{-1}(x) = sin\left(\pi\left(x - \frac{1}{2}\right)\right), \text{ for } 0 \leq x \leq 1. \tag{5}$$

Since a DCSK system is not coherent, we do not need to generate the chaotic signal on the receiver side. For this reason, we used the `/dev/urandom` random generator included in the Linux 3.1 kernel as the underlying uniform random number generator. This random number generator produces random bytes from the entropy collected from the host computer activity (disks I/Os, mouse, keyboard, ...). By reading bytes by packets of four, we have random integer values in $[0; 2^{32} - 1]$ that can be converted to floating point numbers in $[0; 1]$. Then, by applying the $F^{-1}(x)$ function to these uniformly distributed values, we have random values with the distribution of a logistic map.

The main advantage to do so is that there is no accumulation of error due to the floating point computations. This is due to the fact that the error of a floating point computation is not re-injected in the internal state of the generator.

B. DCSK Modulator

The DCSK modulator implementation is straightforward. The processing block has 2 inputs (chaos, data bit), one output and a parameter β. The latter represents the number of samples of chaos which will be used as a reference signal for a symbol. Thus, a DCSK symbol will have a size of 2β samples. The rate ratio is as follows: for each bit samples consumed on the data input, β samples on the chaos input will be consumed and 2β samples will be produced on the output.

C. DCSK Demodulator and synchronization algorithm

Our implementation assumes that transmit and receive gains have been properly adjusted according to a given setup (daughter boards, center frequency, antennas, distances, location, ...).

The demodulation of a DCSK symbol is performed from the sign of the selected correlation function. The decision variable at the output of the correlator for a given bit i is then:

$$D_i = \Re\left(\sum_{k=1}^{\beta} r_k r_{k+\beta}^*\right), \tag{6}$$

where $\Re(.)$, and $*$ are the real value and the complex conjugate operators, respectively.

The main challenge of the DCSK demodulator implementation is the symbol synchronization. Since the USRP clocks are not synchronized and not controlled from the software, there are non-negligible drifts between the transmitter and the receiver clocks. This drift has a direct impact on the center frequency and the sampling rate. The center frequency error in USRPs is small enough compared to the useful bandwidth of the DCSK signal. Moreover, as the reference and data signals will be equally shifted in frequency, a small portion of noise will be mixed with the useful signal, resulting in a performance degradation.

The sampling rate error is a bigger issue: since the symbol length 2β is defined with an integer number of samples and the symbol rate is not synchronized with the transmitter, symbol alignment will drift, with a speed proportional to the sampling rate error.

Our synchronization algorithm tries to resynchronize symbols by finding the best auto-correlation value amongst multiple delayed copies of the received signal. Due to the good auto-correlation properties of the chaotic signal, any symbol misalignment (delay) will result in a very low correlation value between the reference and the data.

In order to control the computational complexity required by this synchronization algorithm, we set a limiting parameter to the number of delayed copies. As shown in Figure 2, after base band transposition, the received signal $r(t)$ is sampled every $kT_c + T_d$ where T_c is the sampling time and T_d is the drift time which is coming from the local USRP clock. Note that this time T_d is not constant and varies from one set of bits to another. For a given bit i, $2\beta + 2N$ samples of the sampled signal r_k are stored in an accumulator. Then, a set of $2N$ correlators compute auto-correlation values between each

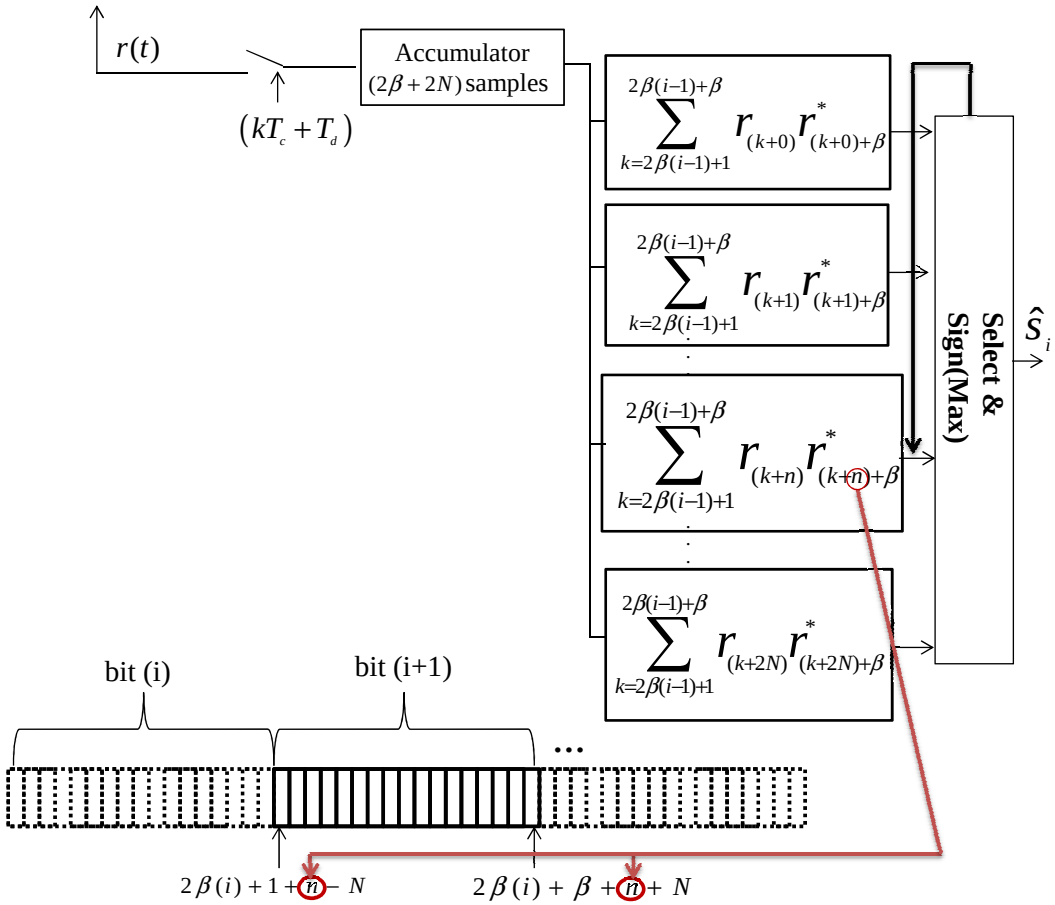

Fig. 2: DCSK receiver with synchronization block

set of two delayed signal slots with β chaotic samples. The parameter N is a function of the drift time T_d ($NT_c > T_d$) where the value is set to cover all possible delays coming from the clock and also the communication channel. Finally a decision circuit takes the maximum of the absolute correlation value to select the corresponding correlation block and then computes its sign to estimate the transmitted bit $\hat{s}_i$. As shown in Figure 2, storage of chaotic samples for the next bit $i+1$ starts from ($2\beta(i) + 1 + n - N$) where n is delay index of the bit i. Note that the index value of n is defined from the selection of the best correlation block of the bit i.

D. Availability

Our implementation is publicly and freely available at [13] under the terms of the GPLv3 license.

V. Performance Measurement Results

The test environment used for performance evaluation was the two previously described USRP2 radios with WBX RF daughterboards. The radios were set up at a central frequency of 250 MHz, the sampling rate was 0.2 MSPS, the spreading factor $\beta = 100$, for a bitrate of 1 kbit/s. The transmitted signal was mixed to a 500 MHz wide Gaussian noise generator with a variable attenuator. During our experimentation, we measured a central frequency error of about 10 kHz. The sampling frequency error was about 10 Hz.

Regarding the BER performance of the proposed system, Figure 3 shows the analytical BER performance along with the measured results of the DCSK system for $\beta = 100$.

The analytical BER performance for DCSK system computed under Gaussian approximation [5] is given by :

$$BER = \frac{1}{2} erfc\left(\sqrt{\frac{E_b}{4N_0}\left(1 + \frac{\beta N_0}{2E_b}\right)^{-1}}\right) \qquad (7)$$

where $erfc$ is the complementary error function, $N_0/2$ is the noise variance, E_b is the bit energy computed at the output of chaotic modulator.

In order to show the clock drift between radios, we forced the clocks of the two USRPs to be synchronized on an external signal generator (perfect clocks synchronization). As shown in Figure 3, the clock drift between the SDRs generate a performance degradation of 3 dB compared to the performance when the synchronization is perfect and theoretical bound given in equation (7). Finally, our major problem with this type of SDR was clocks drift because the USRP clocks are not synchronized and not controlled from the software. The proposed synchronization algorithm try to deal with this prob-

lem by reducing the interference coming from the imperfect sampling by selecting the best correlation which reduces the performance degradation to 3 dB.

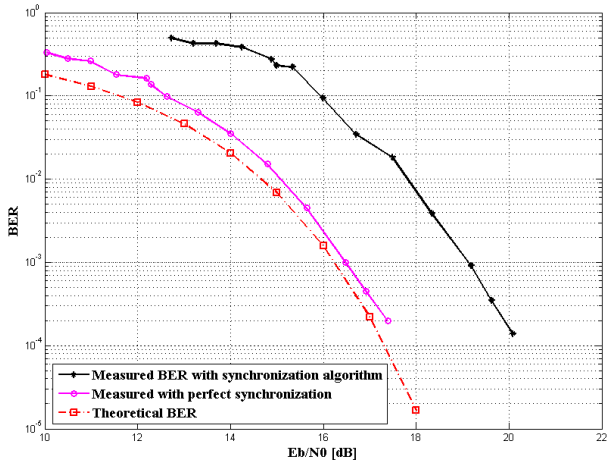

Fig. 3: Theoretical and measured BER performance results

VI. Conclusion

In this paper, the implementation of a real-time DCSK communication system using GNU Radio and USRPs is described. As part of this system, a simple yet robust chaotic generator implementation method has been explained and later implemented to be able to operate in real-time transmissions. A symbol synchronization algorithm is designed to deal with the clock imperfections between USRP radios. This synchronization algorithm is implemented and tested in real-time transmission. The enhancement of the synchronization algorithm is under study. Finally, this work is released under an open source license, thus allowing the research community to review and extend this project.

References

[1] L. M. Pecora and T. L. Carroll, "Synchronization in chaotic systems," *Phys. Rev. A*, vol. 64, pp. 821–823, 1990.

[2] F. C. M. Lau and C. K. Tse, *Chaos-Based Digital Communication Systems*. Springer-Verlag, 2003.

[3] M. P. Kennedy, G. Kolumbán, G. Kis, and Z. Jákó, "Performance evaluation of FM-DCSK modulation in multipath environments," *IEEE Trans. Circuits and Systems*, vol. 47, pp. 1702–1711, 2000.

[4] G. Kaddoum, P. Chargé, and D. Roviras, "A generalized methodology for bit-error-rate prediction in correlation-based communication schemes using chaos," *Comm. Letters.*, vol. 13, no. 8, pp. 567–569, 2009.

[5] M. Sushchik, L. S. Tsimring, and A. R. Volkovskii, "Performance analysis of correlation-based communication schemes utilizing chaos," *IEEE Trans. Circuits and Systems*, vol. 47, pp. 1684–1691, 2000.

[6] Y. Xia, C. K. Tse, and F. C. M. Lau, "Performance of differential chaos-shift-keying digital communication systems over a multipath fading channel with delay spread," *IEEE Trans. Circuits Syst. II, Express Briefs*, vol. 51, pp. 680–684, 2004.

[7] M. Delgado-Restituto and A. Rodriguez-Vazquez, "Mixed-signal map-configurable integrated chaos generator for chaotic communications," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, vol. 48, no. 12, pp. 1462 –1474, dec. 2001.

[8] Delgado-Restituto, M., and A. Rodriguez-Vazquez, "Integrated chaos generators," *Proceedings of the IEEE*, vol. 90, no. 5, pp. 747 –767, may 2002.

[9] T. S. Parker and L. O. Chua, *Practical Numerical Algorithms for chaotic systems*. Springer-Verlag, 1989.

[10] M. Delgado-Restituto, A. Rodriguez-Vazquez, and V. Porra, "Integrated circuit blocks for a dcsk chaos radio," in *Circuits and Systems, 1998. ISCAS '98. Proceedings of the 1998 IEEE International Symposium on*, vol. 4, may-3 jun 1998, pp. 473 –476 vol.4.

[11] K. Krol, L. Azzinnari, E. Korpela, A. Mozsary, M. Talonen, and V. Porra, "An experimental fm-dcsk chaos radio system," in *European Conference on Circuit Theory and Design IEEE International conference on*, 2001, pp. 17–20.

[12] X. Li, X. Lin, and D. Guo, "The experimental blind timing acquisition scheme for fm-dcsk communication system," in *Anti-Counterfeiting Security and Identification in Communication (ASID), 2010 International Conference on*, july 2010, pp. 120 –125.

[13] "gr-chaos project repository." [Online]. Available: https://github.com/jolivain/gr-chaos

[14] "GNU Radio." [Online]. Available: http://gnuradio.org/

[15] "GNU General Public License 3.0." [Online]. Available: http://www.gnu.org/licenses/gpl-3.0.html

[16] "Ettus Research." [Online]. Available: http://www.ettus.com/

[17] D. E. Knuth, *Seminumerical Algorithms*, 2nd ed., ser. The Art of Computer Programming. Addison-Wesley, 1981, vol. 2.