# Development of a 16-QAM Modulator and Demodulator Python Model Suitable for VHDL Implementation

Arthur Cruz Morbach
Polytechnic School
Unisinos University
Brazil
arthurmorbach@edu.unisinos.br

Jonas Dandanel de Castro
Polytechnic School
Unisinos University
Brazil
jdcastro@edu.unisinos.br

Sandro Binsfeld Ferreira
Polytechnic School
Unisinos University
Brazil
sbinsfeld@unisinos.br

***Abstract*— In this project, a low intermediate frequency (low-IF) 16 Quadrature Amplitude Modulation (QAM) is implemented in Python, aiming at a future VHDL implementation and validation of a receiver. The system consists of a modulator, an Additive White Gaussian Noise (AWGN) channel model, and a demodulator. The Gardner Algorithm is implemented in the demodulator to synchronize the symbols. A concise block diagram is reached in the Python implementation which will guide the VHDL codification to optimize the final architecture. All the results obtained in the simulations will be used to verify the VHDL performance.**

***Keywords—Quadrature Amplitude Modulation, modulator, demodulator, Python, VHDL.***

## I. INTRODUCTION

Low intermediate frequency (low-IF) receivers are the most used architectures in modern wireless communications [1], for instance in the growing Market of Internet of Things (IoT) applications.

The QAM (Quadrature Amplitude Modulation) is used in several communication systems, such as digital TV, digital radio, high speed internet services, and other systems with a high data rate requirement. The main reason for its adoption is its high spectral efficiency.

The first step in the design of a digital receiver (RX) and a transmitter (TX) is modeling of the complete communication system. In this work, Python language was chosen for this task mainly due to the wide availability of libraries for digital communication applications, good documentation, and low cost. The next step is the development of a hardware description language model to guide the implementation in field-programmable gate array (FPGA) or Application Specific Integrated Circuit (ASIC).

This work presents the development of Python models for a QAM modulator and a QAM demodulator, and the initial development of the VHDL code for the modulator. The work is structured as follows. Section II presents the methodology, Section III presents the Python implementation, with the beginning of the VHDL implementation and simulation results discussed in Sections IV and V respectively. Section VI concludes the work.

## II. METHODOLOGY

The design development was defined in two phases:

### A. Modelling in Python

Python language was chosen to model the system for two main reasons: being opensource, and for the large number of available libraries for simulations and data comparison. The main libraries used in this project are commpy [2] and scipy.fftpack [3].

### B. VHDL codification

With a concise model of the system, the blocks are implemented one by one in VHDL, using Xilinx ISE environment. Results are compared through generated text files to verify the precision of the fixed-point implementation.

## III. PYTHON IMPLEMENTATION

### *A. 16-QAM Modulator Implementation in Python*

The 16-QAM is characterized by representing each symbol with 4 bits. This information can be viewed in the form of a constellation (Fig. 1), where each point represents a combination of 4 bits.

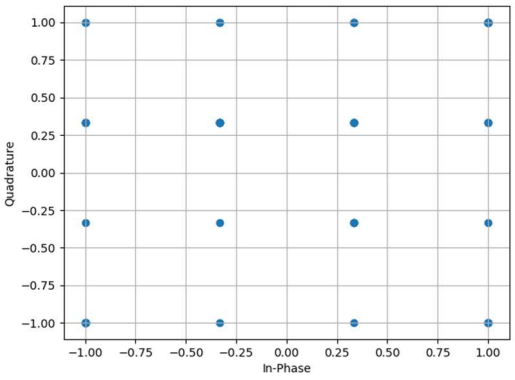


*Fig. 1 - Constellation IQ, 16-QAM Mapper.*

The modulator block diagram is presented in Fig. 2. To generate all the 16 possible symbols at least once in a simulation, 64 symbols were proposed. Hence 256 pseudo-random bits are required at the input of the model.

The first block in the modulator chain is the slicer logic block. It separates the even bits in the In-phase (I) vector, and the odd bits in the Quadrature (Q) vector. After this operation, both vectors will be treated in parallel.

The mapper block will transform a pair of bits into a symbol. So, a vector with all possible amplitudes and phases for two bits is created (1).

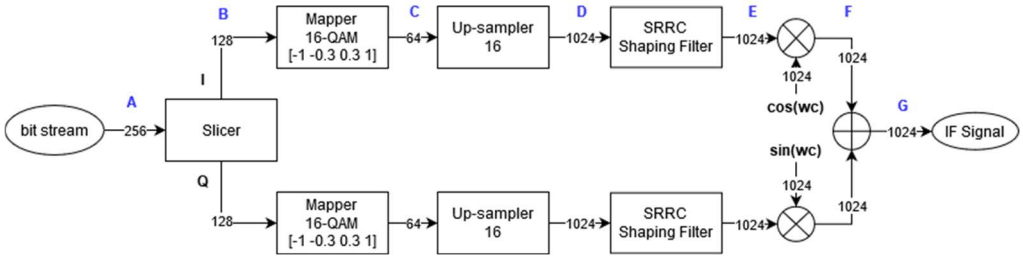


*Fig. 2 - Modulator block diagram.*

$$QAM = [-1 \quad -0.333 \quad 0.333 \quad 1] \tag{1}$$

The pair of bits is used as an index for the array QAM, as shown in (2).

$$00 \rightarrow -1;\ 01 \rightarrow -0.333;\ 10 \rightarrow 0.333;\ 11 \rightarrow 1 \tag{2}$$

After the symbols generation, the I and Q signals are up-sampled with a factor of 16 to prepare for the Square Root Raised Cosine (SRRC) shaping filter. The filter is required to concentrate the symbol information in a defined frequency spectrum, thus reducing inter-symbolic interference. After the convolution with the filter impulse response, the symbols are spread in the time domain.

The signal frequency after the shaping filter is still not modulated and is named *baseband* signal. Considering that the signal will be fed to an intermediate frequency (IF) demodulator, it is required to shift the baseband signal to the IF frequency using a mixer.

In the mixing process (Fig. 2), I and Q signals are multiplied by cosine and sine signals with a reference frequency to shift the baseband to the required IF. In the Python implementation, the cosine and sine signals are generated using look up tables. After the mixing process, both I and Q signals are summed to generate the quadrature IF signal.

### *B. 16-QAM Demodulator Implementation in Python*

The signal generated by the modulator block in the previous section passes by an Additive White Gaussian Noise (AWGN) channel model. At the input of the demodulator the signal is defined by the follow expression:

$$s(t) = \left[a_I \cos(2\pi f_c t + \theta) + a_Q \sin(2\pi f_c t + \theta)\right] + n(t) \tag{3}$$

where $a_I$ and $a_Q$ are the shaped signals in-phase and in quadrature, respectively, $f_c$ is the carrier frequency, and $n(t)$ is the AWGN. For this model, phase $\theta$ will be considered zero.

The mixer is the first block of the demodulator presented in Fig. 3. By trigonometric relations, the product of the signal (3) with a cosine with the same frequency will generate back the I signal with two components, one centered at zero frequency (baseband), and another centered at two times the carrier frequency, as shown in (4). Mixing $s(t)$ in parallel with a sine will result the Q signal, with similar characteristics.

$$d_I(t) = s(t) * \cos(2\pi f_c t)$$

$$d_I(t) = \frac{a_I}{2}[1 + \cos(4\pi f_c t)] + \frac{a_Q}{2}[\sin(4\pi f_c t)] + n(t) \tag{4}$$

After a low pass filter (LPF), the high frequency components are eliminated, (5).

$$LPF[d_I(t)] = \frac{a_I}{2} + n(t) \tag{5}$$

A matched filter is implemented after the low pass filter. This filter has a very important purpose in the demodulation process. Through a convolution of the signal with a template, the signal to noise ratio (SNR) is increased. The template used is equal to the impulse response of the shaping filter introduced at the modulator.

According to Haykin [4], the relation signal to noise of the peak of a pulse in the matched filter depends only of the relation between signal energy and the spectral density power of the AWGN in the filter input. The bit and symbol average energy can be obtained from the amplitudes of the constellation symbols, and the noise density power is the variance of the gaussian distribution, i.e. the square of the standard deviation [5].

The SNR is defined by the ratio between bit energy ($E_b$) and noise density power ($N_o$) using (6) :

$$SNR = \frac{E_b R}{N_o B} \tag{6}$$

where R is the bit rate, and B is the bandwidth [6].

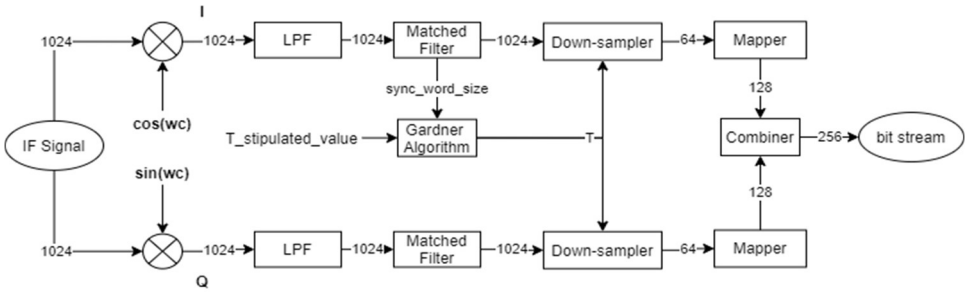


*Fig. 3 - Demodulator Diagram Block.*

The next process is to sample the signal at its peak. The difficulty is to know when this peak happens. There are several algorithms to synchronize and identify the peak for the symbol detection. In this demodulator, the Gardner Algorithm is implemented [7].

The Gardner algorithm is based on the follow equation:

$$e = \{x[nT] - x[(n-1)T]\} * x\left[nT - \frac{T}{2}\right] \qquad (7)$$

where $T$ is the sample period and $n$ is the iteration number. This algorithm analyzes the synchronization word at the beginning of the reception and detects which samples correspond the symbol's peak.

If the result of (7) is smaller than zero, a timing advance is required for the next iteration, if the value is larger than zero, a timing delay is required. The zero result means that the exact value of samples between the symbol peaks was found. Timing advance and delay are made by incrementing or decrementing the variable $T$.

When the algorithm finds the intervals between symbols, the down-sampler block samples the input signal and generates an array with the symbol's values.

Using the same codification as the mapper block array at the modulator, the amplitude symbols are converted to the bits they represent. At this moment, I and Q vectors are defined. Since the I vector represents the even bits and the Q vector represents the odd bits, the bit word is combined back, and the bit stream is found.

## IV. VHDL Implementation

### A. 16-QAM Modulation Implementation in VHDL

The VHDL version of the modulator is implemented according to the same block diagram presented in Fig. 2. It starts splitting the data into 4-bit arrays by using 4 latches. The mapper block takes both odd and even bits and use a look up table (LUT) to transform the numbers in fixed point-numbers of 8-bit 2's complement, as assigned in the python version. The clock in the LUT is two-times faster, and the look up table is presented as below.

$00_2 \rightarrow -1_{10} \rightarrow 11000000_2$; $01_2 \rightarrow -0.333_{10} \rightarrow 11101011_2$;

$11_2 \rightarrow +1_{10} \rightarrow 01000000_2$; $10_2 \rightarrow +0.333_{10} \rightarrow 00010101_2$;

The first bit represents the signal, the second the integer part and the last 6 bits the decimal part. In this way, the numbers are initially truncated binarily to 8 bits. Since "00010101" represents 0.328 and not 0.333 it may require an increase in the number of bits depending on the achieved performance of the modulator.

To prepare for the SRRC shaping filter, the symbols go through an up-sampler with a factor of 16. It should be observed that the clock needs to be 32 times faster than the data input. The SRRC block is still under development. In the VHDL implementation there is an additional block which is the clock generation.

## V. Simulation Results

### A. Python modulator implementation

A SRRC filter with 0.35 roll-off factor was implemented in the shaping filter block, Fig. 2.

In Fig. 4 (a), the spectrum of the signal at the point "D" in Fig.2, and in Fig.4 (b) the spectrum at the point "E" in Fig. 2 are presented. It is possible to observe that the filter not only shapes the signal, but also filters the spectrum. In this implementation, the symbol rate was defined as 1 MHz, hence with a symbol duration of 1 µs. The shaping filter ensures that the current symbol has an amplitude close to zero at the peak of the next symbol, which reduces the probability of intersymbol interference.

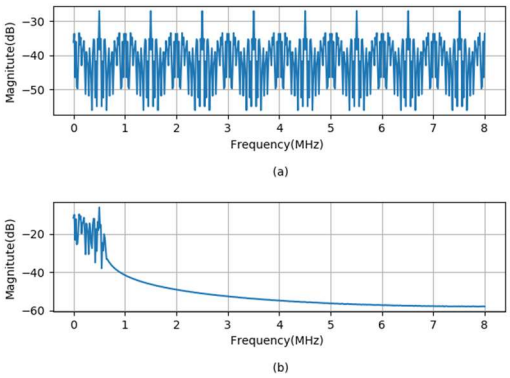


*Fig. 4 - Comparison between the frequency spectrum of the signal before (a) and after (b) the shaping filter.*

Fig. 5 shows the signals I (a) and Q (b) after convolution with the filter impulse response, at the "E" point in Fig. .

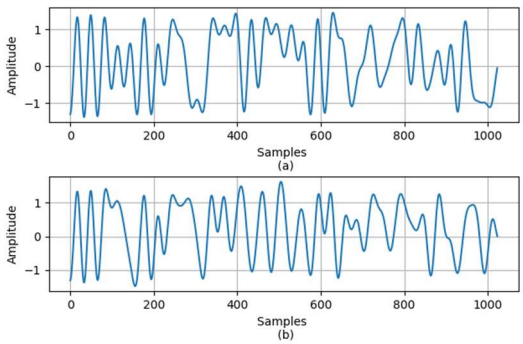


*Fig. 5- Signals I (a) and Q (b) after the convolution with the SRRC filter*

In the next step, presented in Fig.6, the I (a) and Q (b) signals are mixed with the 2-MHz sine and cosine clocks and summed to generate the IQ modulate signal (c).

### B. Python demodulator implementation

The average bit energy was calculated based on the distance of the symbols from the center of the constellation. The average bit energy is 0.278 µJ. In this simulation, the bit rate is 4 Mb/s and the bandwidth of the signal is 1 MHz. So, an AWGN with variance of 35.512n is necessary for a SNR of 15dB according to (6).

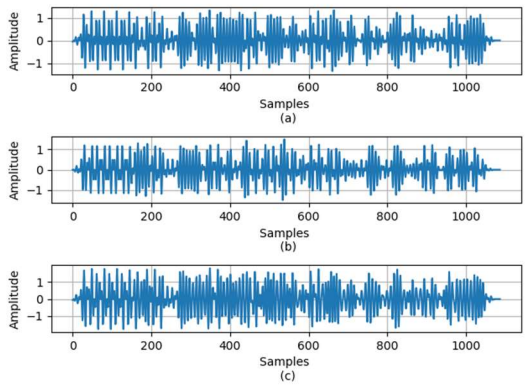


*Fig. 6 - Mixed signals I (a) and Q (b) and IQ (c).*

When the signal gets to the demodulator, a mixing process is executed, resulting in a signal with multiple frequencies as shown in (4). In this modeling, the sine and cosine waves have the same frequency and phase as the modulator. In the future, this limitation will be removed using a phase detector. An LPF with cut off frequency of 1 MHz is implemented after the mixer to acquire these signals in baseband frequency.

The I and Q signals pass through a matched filter to increase SNR, also increasing the amplitude. Comparing Fig. 5 with Fig. 7, it is possible to notice that the first symbol was not accounted. Since it is a recursive filter it loses the first samples.

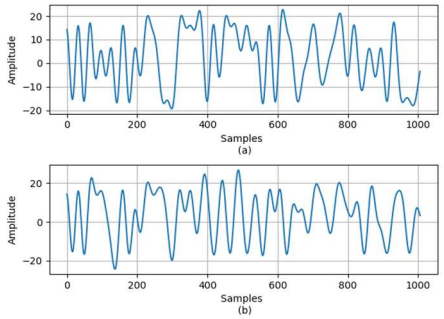


*Fig. 7 – I(a) and Q(b) signals after the LPF and Matched Filter.*

In Fig. 8 are the resulting constellations at the output of the modulator (a) and at the output of the demodulator (b), obtained by sampling the output of the matched filter using Gardner's Algorithm (7).

### C. VHDL Simulation Results

After the mapper was implemented in VHDL the data was imported to Python and compared with the mapper model, (1) and (2). Is possible to see in Fig. 9 (c) a difference of 5% between the two implementations.

## VI. Conclusions

In this work, 16-QAM modulator and demodulator are modeled in Python aiming to a future VHDL design. The main parts of the system were discussed and simulated in detail. The Python results can be used to check VHDL

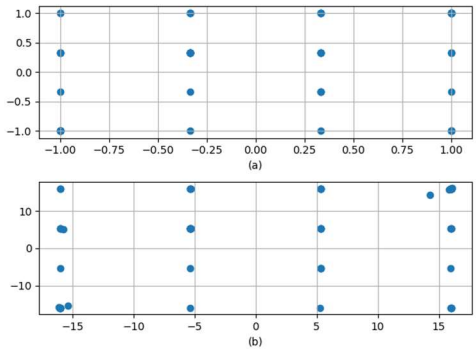


*Fig. 8 - Comparation between the modulator (a) and demodulator (b) constellation. SNR = 15dB.*

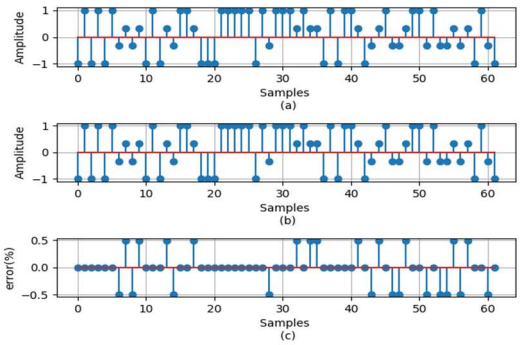


*Fig. 9 – Mapper output in VHDL (a), in Python (b) and data comparation between the two platforms (c).*

performance. A phase detector block still needs to be developed to synchronize receiver phase to the input signal received from the modulator. The initial blocks of the modulator were implemented in VHDL. The results obtained are in good agreement with the python model.

## Acknowledgment

The Authors would like to thank FAPERGS and Unisinos University.

## References

[1] B. Razavi, *RF microelectronics*. Prentice Hall, 2012.

[2] "scikit-commpy · PyPI." https://pypi.org/project/scikit-commpy/ (accessed Jun. 20, 2020).

[3] "scipy · PyPI." https://pypi.org/project/scipy/ (accessed Jun. 20, 2020).

[4] S. Haykin, *Sistemas de comunicação : analógicos e digitais*. Bookman, 2004.

[5] L. J. Ippolito, "Error Functions and Bit Error Rate," in *Satellite Communications Systems Engineering*, Chichester, UK: John Wiley & Sons, Ltd, 2017, pp. 423–425.

[6] K. Schiphorst, Roel; Hoeksema, Fokke; Slump, "Bluetooth Demodulation Algorithms and their Performance," *IEEE Trans. Circuits Syst. Ii Analog Digit. Signal Process.*, 2002.

[7] F. M. Gardner, "A BPSK/QPSK Timing-Error Detector for Sampled Receivers," *IEEE Trans. Commun.*, vol. 34, no. 5, pp. 423–429, 1986, doi: 10.1109/TCOM.1986.1096561.