International Journal of Electronics and Communication Engineering (IJECE)
ISSN(P): 2278-9901; ISSN(E): 2278-991X
Vol. 8, Issue 1, Dec - Jan 2019; 1-10


International Academy of Science, Engineering and Technology
Connecting Researchers; Nurturing Innovations

# FILTERING AND ACQUISITION OF PCM FRAMES USING SYSTEM GENERATOR

***Adrian Stacul & Edgardo Comas***

*Instituto De Investigaciones Cientificas Y Tecnicas Para La Defensa Buenos Aires, Argentina*

## *ABSTRACT*

*The main purpose of this paper is the design, development, and implementation of a PCM bit-synchronizer based on a System Generator and Simulink model. The entire system will be applied to a ground station with an ad-hoc telemetric data acquisition system to be applied in unmanned aerial vehicles, atmospheric sounding rockets and nano-satellites monitoring. Based on this information, the ground station will be able to compute navigation parameters trajectories, velocities and attitudes. In particular, this PCM module was built to be used in atmospheric sounding vector evaluations by the Instituto de Investigaciones Científicas y Técnicas para la Defensa of Argentina.*

***KEYWORDS:*** *Bit-Synchronizer, FPGA, Ground Station, PCM Frame, System Generator*

***Article History***

***Received:*** *29 Jun 2018* **|** ***Revised:*** *05 Dec 2018* **|** ***Accepted:*** *15 Dec 2018*

## INTRODUCTION

Over the past decade, R&D in atmospheric sounding rockets, UAVs (Unmanned Aerial Vehicles) and nanosatellites has enjoyed exponential growth in several disciplines: aeronautical systems, applied mechanics, onboard electronics, ground stations, real-time signal processing, etc. In this work, we will focus on ground-based signal processing methods to acquire PCM signals and distribute telemetric information to multiple monitoring clients [8].

Both unmanned aerial systems and sounding rockets require a ground station for the acquisition of telemetric signals and real-time data processing whether for the control and monitoring of the mission or for the evaluation of the different scientific experiments installed on the platform [5]. The design of an acquisition system in a ground station is a complex task since it involves receiving data and sending it to the processing systems so that everything operates in real time. At the same time, onboard electronic systems are increasingly faster and easily adaptable to the requirements of the experiment. As a consequence, the data acquisition system changes constantly with every redesign of the platform. The aim of this work is to obtain a low-cost data acquisition system that allows the reception of high-speed PCM frames to decommute all of the channels with the physical magnitudes within the PCM frames [6]. The module developed draws on the progress of different methods for the synchronization of frame headers and data decommuting in the ground acquisition system, which will perform the information processing task in real time [3].

### Design

In order to achieve PCM signal regeneration, it is necessary to filter and observe the permanent regime state of the signal. The level (zero or one) at the midpoint of each bit is maintained despite the noise. Fig. 1 shows the difference between

the originally transmitted signal and the received signal (note that the latter is band-limited, so the abrupt flanks no longer exist). The complete model (Fig. 2) shows two instances of the acquisition system: first, the filtering model (where you enter an ideal PCM frame with noise added) and second, the acquisition model (in charge of the synchronization).

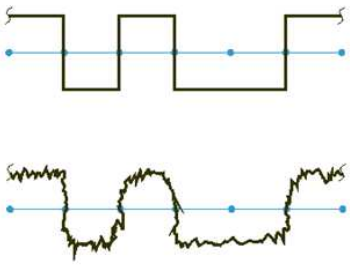


**Figure 1: Original Signal Transmitted (A) and Received Signal Reduced in Bandwidth with Additional Noise (B)**

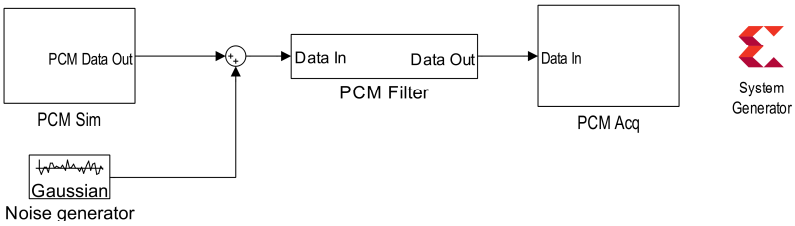



**Figure 2: Complete PCM Model with Added Noise, Filtering, and PCM Acquisition**

## Development

The model with System Generator [14] provides a high-level tool for the development of high-performance systems using Xilinx devices with FPGA technology, and thus defines and characterizes logic circuits to fulfill a specific function.

The main difference between any HDL and the rest of the programming languages is that the description languages are synthesized, not compiled or executed like any other program. This is due to the fact that programming languages are defined as procedures. Instead, the hardware description is based on the definition of behaviors according to the inputs and the desired processing concurrently [4].

During the synthesis, the interconnection of the available resources in the FPGA is defined so they behave in the way described. It is part of the work of the development tools to carry out the necessary optimization to take less resources or forthe block to operate at higher frequencies. The System Generator automatically translates the block development of a Simulink[12] model into HDL by optimizing FPGA times and area and also generates the final binary file [10].

The implementation in hardware was performed on the 3PX1 development kit manufactured by Emtech [2], with a Spartan-6 FPGA (XC6SLX25 [13]). This board meets the basic needs to initiate the development and prototyping of specific system applications with FPGA technology. The board also includes a flash memory where to store the firmware, push-buttons to use as inputs and LEDs to use as status indicators. It was mounted on an open cabinet made of acrylic to give greater rigidity to the board, also to be able to unify it as a single module with BNC connectors for connection and disconnection without compromising the FPGA device. Three connectors were placed on the front panel, two with the PCM Transmitter outputs (data and clock) and a third connector with the input to the acquiring system, Fig 3.

The hardware design for the FPGA was performed using MATLAB[9] in conjunction with the System Generator, a tool provided by Xilinx to work in that environment. The MATLAB-Code was used to implement the MATLAB language

directly on an FPGA, eliminating the need to program under VHDL or Verilog.

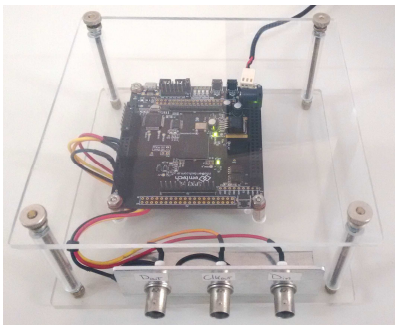


**Figure 3: Implementation in the FPGA Development kit**

## PCM Filtering

To generate a realistic scenario, we add noise to an ideal PCM signal. This design supports critical conditions where noise could exceed 50% of the signal level. To achieve this, we perform a specific configuration of the "Noise generator" block and perform the noise measurement using the equation.

$$SNR = \frac{Signal}{Noise} = \left| \frac{\text{RMSValueofPCMdata}}{\text{RMSValueofNoisegenerated}} \right| \quad (1)$$

The SNR measurement model can be seen in Fig. 4 and the resulting value is approximately 0.6, i.e. the noise, in this case, is set to be 60% of the signal level, whereby the condition is completely secured (Fig. 5). This mentioned model perform a real-time SNR measurement, which is advantageous to test different simulation scenarios with different types of frames and times.

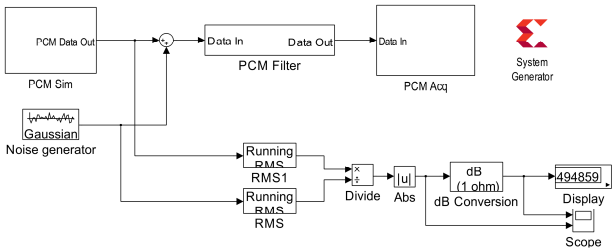



**Figure 4: Model to Measure SNR**

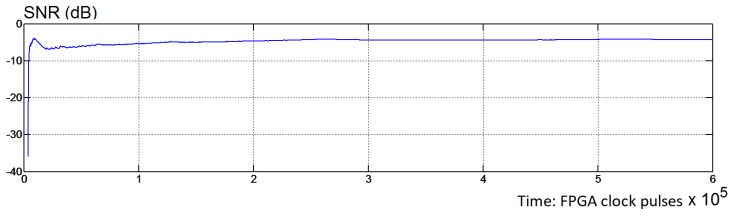



**Figure 5: Results of SNR in PCM Data with Noise Added**

This PCM filter sub-system implements a CIC filter (Cascaded Integrator-Comb). Implementations of CIC filters have structures that use only adders, subtracters, and delay elements. These structures make CIC filters appealing for their hardware-efficient implementations of multirate filtering [11].

In System Generator, the CIC filter block has a single input port and a single output port, $x_n$ and $y_n$, and M is the

differential delay. In the decimator configuration, the sampling rate is reduced by a factor of R, sub-sampling the output of the last stage of the integrator [15].

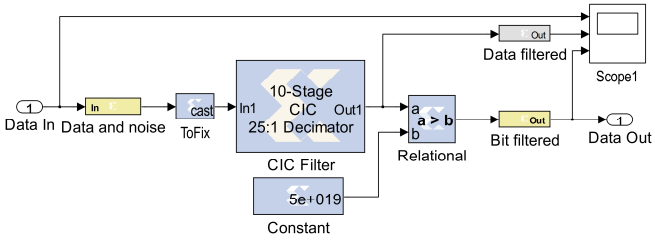



**Figure 6: Implementation of PCM Data Filtering with CIC Filter**

As can be seen in Fig. 6, the implemented model is a 10 step CIC filter that, despite consuming many resources and a large area of the FPGA, ensures a good filtering that meets the high level of requirement in the design. On the other hand, we must ensure that the width of each bit is kept equal to the output of the filter compared to its original ideal, so we adapt a filter 25:1, which generates 68-bit data at the end of the CIC filter [7]. These last data are compared toa constant to generate a digital pulse at the output of the subsystem. It was determined, by multiple tests and simulations, that the appropriate value of the constant is set to $5.10^{19}$. Figure 8 shows how the filtering is done correctly in a PCM data signal.

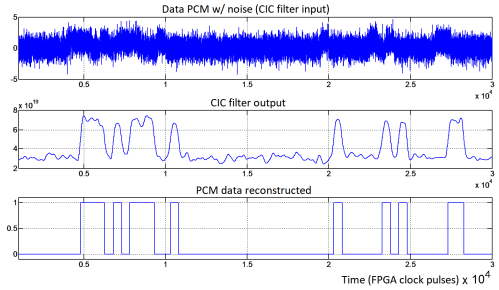



**Figure 7: Results of PCM Data Filtering. First Channel: PCM Data W/Noise, Second Channel: CIC Output, Third Channel: Filtering Output**

## PCM Acquisition

In order to process the PCM incoming data, it is important to first determine the start and end of each packet. Since from the point of view of the entry there is no distinction, the data entry is continuous and asynchronous. The development challenge is to be able to detect the beginning of each package, a pattern recognition or sync word detection, to regenerate the data synchronously for further processing. The complete model can be seen in Fig. 8.

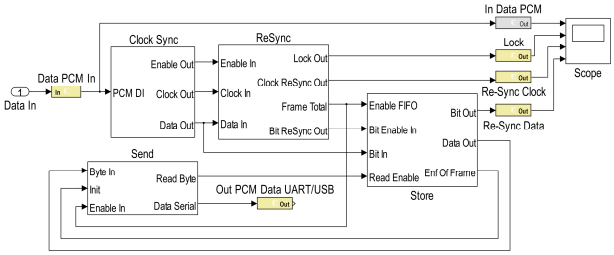



**Figure 8: PCM Acquisition Model**

## Clock-Sync

The algorithm developed for the re-synchronization block is based on the rising edge detection of PCM data input.

This sub-system (Fig. 9) performs a first PCM clock re-generation continuously and synchronizes automatically when a rising edge is detected in the incoming signal.

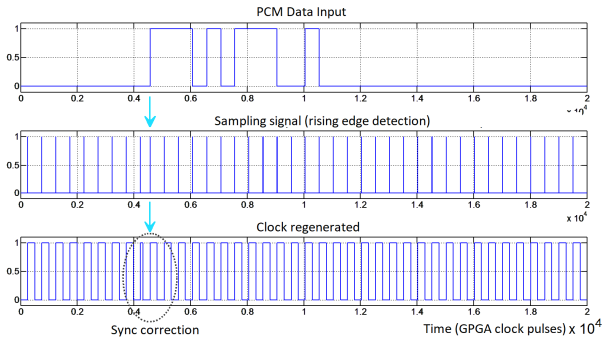



**Figure 9: Results of PCM CLOCK-SYNC. First Channel: PCM Data Input, Second Channel: Rising Edge Sync, Third Channel: Clock Synchronized**

## ReSync

The ReSync subsystem performs a pattern recognition of the sync-word to lock the incoming signal and re-synchronize it with the clock. To achieve this, two tandem blocks are used. The first one has a finite state machine and a shift register, to find the sync-word bit by bit (performs a bit-sync). The second generates additional signals to other blocks. The results of the simulation can be seen in Fig. 10. In the last channel, we can see the incoming PCM data signal in the subsystem, and when the first rising edge is detected, the clock is re-synched, and a valid frame flag is activated, indicating a possible true PCM frame. From that moment, an algorithm starts to recognize the sync-word configurated (performed by a block which is called lock). Figure 10 shows how after 16 bits, the output is true indicating that the sub-system found a valid sync-word. In this case, the sync-word is 1110101110010000 ( $EB90_{16}$ ).

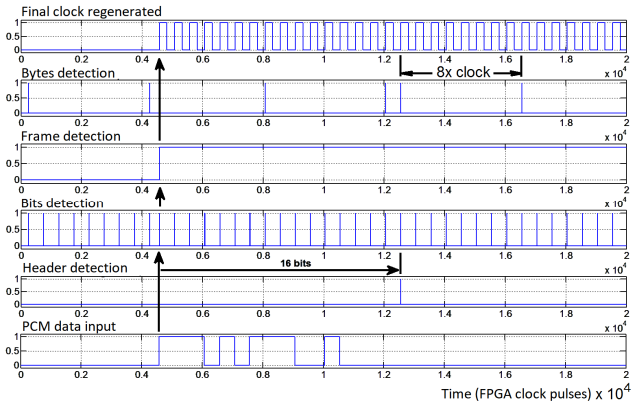



**Figure 10: Results of PCM RESYNC. First Channel: PCM Data Clock Regenerated, Second Channel: Bytes in PCM Frame, Third Channel: True Valid PCM Frame, Fourth Channel: Bits in PCM Frame, Fifth Channel: Lock Signal (Sync-Word Detected), Sixth Channel: PCM Data**

Several frequency measurements were performed with an analyzer. The PCM data (output of the PCM simulator) have 99.9998 KHz with a standard deviation of 1.1248 MHz, with 125 samples per second in a total of 10 seconds of duration. A second frequency stability measurement was performed with the PCM filtering and acquisition in full operation, and the results of the PCM clock regenerated are 9.9998 KHz with a standard deviation of 1.5216 MHz (Fig. 11). In conclusion, the error added by the model is practically null.

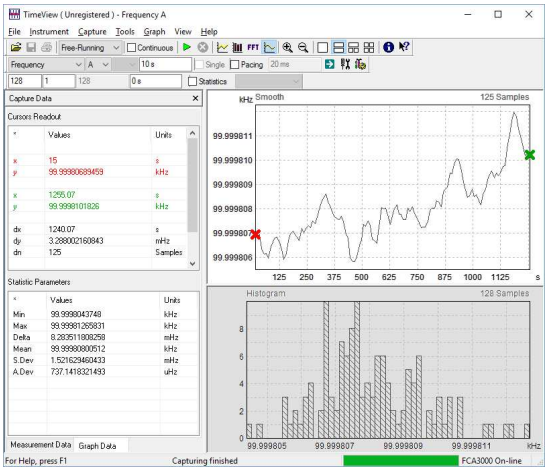



**Figure 11: Results of Regenerated PCM Clock Stability**

## Store

Frame storage in the system is essential for a protocol conversion or pre-processing algorithms. In this case, the acquisition system converts the incoming PCM data and sends it synchronized using the UART protocol, through to USB physical connection [1]. This sub-system combines M-Code block (MATLAB script code) taking the incoming flow of bits and the flag indicating a valid frame and generates one byte for each channel of the frame. These bytes are stored in a distributed memory FIFO to decrease the resources and area of the FPGA, especially for low-cost devices where hardware resources are more limited without compromising the operation frequency. The operation of the STORE sub-system is shown in Fig. 12 and it can be observed how the sub-system takes the valid byte at the beginning of the frame, taking into account that the sync word is EB90 in hexadecimal.

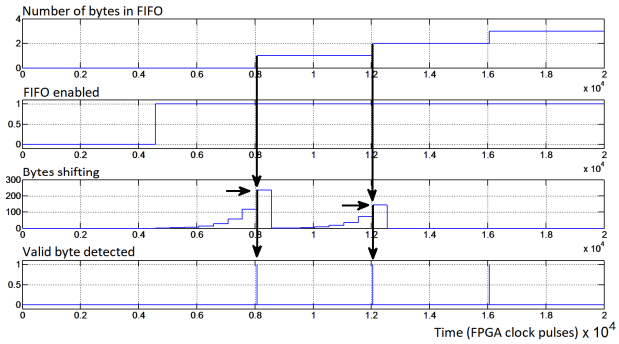



**Figure 12: Operation of the STORE Sub-System. First Channel: Number of Bytes in FIFO, Second Channel: FIFO Enable, Third Channel: a Shift Register, Fourth Channel: Valid Byte**

## Send

The SEND subsystem collects the data stored in the STORE block and serializes them with an 8N1 UART protocol (i.e. 1word start bit, 8 data bits and one stop bit). Figure 13 shows a simulation result of the converted PCM frame.

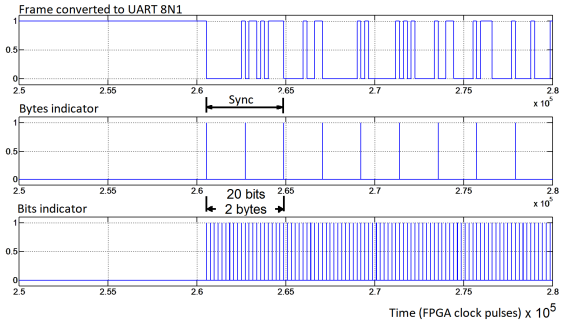



**Figure 13: PCM Frame Converted to UART. First Channel: Transmit in UART Protocol, Second Channel: Flag Indicating Bytes in UART, Third Channel: Flag Indicating Bits in UART**

## Implementation and Results

By connecting the PCM simulator (a known frame used as a pattern) to the acquiring system, we can receive the data captured in a computer through a USB-serial port. The data will be dumped into a vector called "serial data".

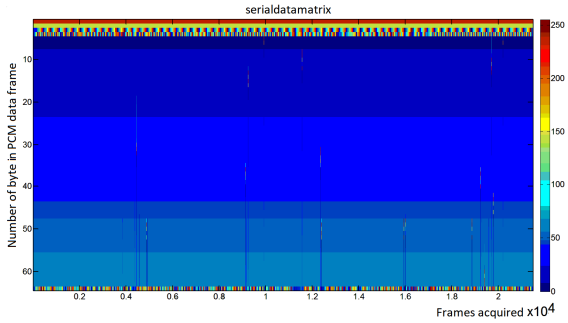



**Figure 14: PCM Frames in the Matrix with HSV Codification Rows: Channel of PCM Data (Bytes), Column: Number of PCM Frame Received.**

To perform the proposed analysis, we will convert *serial data* into a matrix, where each column represents a complete frame (from the sync word to the end of the frame) and the column number is the received frame number. Figure 14 shows the matrix in an HSV color chart by treating it as an image. The matrix will be cropped to keep only the payload by deleting the first 4 bytes and the last byte, called *serialdatamatrix*. A new matrix of constants is generated representing the ideal matrix (as if there were no errors) called *data matrix*. The two matrices are subtracted to a new matrix called *dif*n. The resulting image is shown in Fig. 15.

$$dif_{(m,n)} = serialdatamatrix_{(m,n)} - datamatrix_{(m,n)} \tag{2}$$

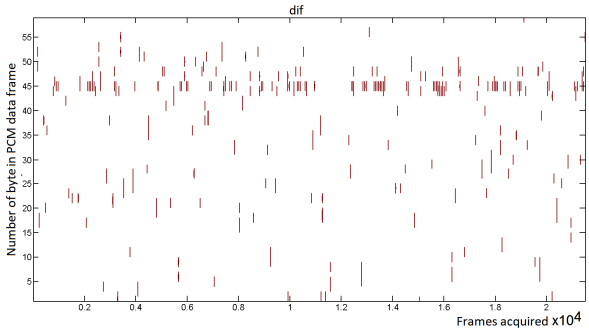



**Figure 15: Errors in PCM Acquisition Rows: Channel of PCM Data (Bytes), Column: Number of PCM Frame Received.**

In total, 2102 errors were detected from a total of 1375232 bytes captured (21488 PCM frames at 5ms per frame with a length of 64 bytes each). Also, the system was tested successfully with 1Mbps obtaining the same results. Subtracting the first 4 bytes and the last (5 bytes in total), we have a total of 21488 frames of 59 bytes in the payload, resulting in a total of 1267792 bytes.

If we calculate the percentage of bytes with error vs. the total payload bytes, it gives an efficiency of 99.83. The maximum error introduced by the acquiring system is 0.165%, assuming that the error occurs in the complete byte, i.e. the 8 bits are erroneous.

This value is due to the fact that the acquiring system is re-engaged by each received frame. Other systems synchronize only once with the first valid sync-word. Therefore, if an error occurs in the medium of the acquisition of a frame when it detects the sync-word of the next frame, it is re-accommodated. This shows that while the system introduces a small error in the telemetry chain, it is minimal and self-correcting, thus proving trustworthy for mission-critical applications.

## CONCLUSIONS

In total, 230 frames were found with errors and a maximum of 5 bytes with error per frame. If an entire frame discard policy is applied in case we found only one error, in total it would be 1.07% of discarded frames. The BER calculated is about 0.15%.

Another important analysis is to corroborate the amount of erroneous data as a function of time by performing a fitting of the number of frames received vs. number of errors accumulated in time. The results are shown in Fig.16 and the errors fit in a linear equation. In other words, the efficiency of the system remains the same in time.

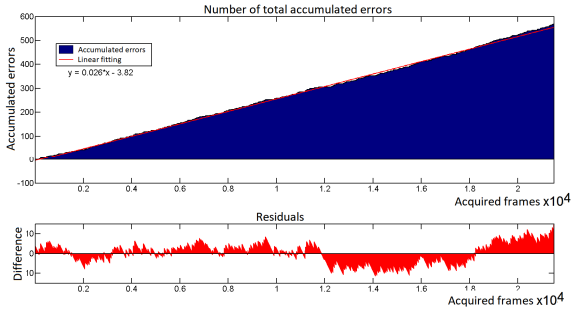
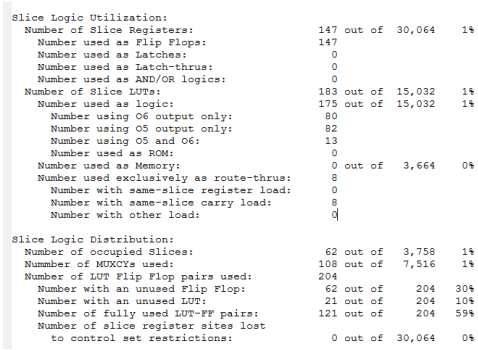



**Figure 16: Errors in PCM Acquisition First: Number of Errors Accumulated in Time, Second: Residuals.**

For the implementation of the PCM acquisition model, the base software and the FPGA device of Table 1 were used. The results of the final implementation, i.e. once the final hardware routing was generated after the synthesis, can be analyzed in Fig. 17. As a conclusion, we can highlight that it occupies a very little area of the hardware. Therefore, this system is compatible with low-cost FPGA devices, where the area and resources are very limited.

**Table 1: Software Version and Target Device for the System Implementation**

| Software Version and Target Device | | | |
|---|---|---|---|
| ProductVersion: | *ISE:14.2 P.28xd* | Target Family: | *Spartan6* |
| OS Platform: | *NT64* | Target Device: | *xc6slx25* |
| Project ID | | Target Package: | *ftg256* |
| Registration ID | *__0_0_0* | Target Speed: | *-3* |
| Date Generated | | ToolFlow | *C.Line* |



```
Slice Logic Utilization:
  Number of Slice Registers:                147 out of  30,064    1%
    Number used as Flip Flops:              147
    Number used as Latches:                   0
    Number used as Latch-thrus:               0
    Number used as AND/OR logics:             0
  Number of Slice LUTs:                     183 out of  15,032    1%
    Number used as logic:                   175 out of  15,032    1%
      Number using O6 output only:           80
      Number using O5 output only:           82
      Number using O5 and O6:                13
      Number used as ROM:                     0
    Number used as Memory:                    0 out of   3,664    0%
    Number used exclusively as route-thrus:   8
      Number with same-slice register load:   0
      Number with same-slice carry load:      8
      Number with other load:                 0

Slice Logic Distribution:
  Number of occupied Slices:                 62 out of   3,758    1%
  Nummber of MUXCYs used:                   108 out of   7,516    1%
  Number of LUT Flip Flop pairs used:       204
    Number with an unused Flip Flop:         62 out of     204   30%
    Number with an unused LUT:               21 out of     204   10%
    Number of fully used LUT-FF pairs:      121 out of     204   59%
    Number of slice register sites lost
      to control set restrictions:            0 out of  30,064    0%
```

**Figure 17: Place and Route Implementation Report**

## ACKNOWLEDGMENTS

The present R&D work was carried out under the supervision of my Ph.D. thesis director, Ph.D. Mario Lavorato, the Head of the Digital Techniques Laboratory, Eng Daniel Pastafigliato whom I would like to express my deepest appreciation for making this study possible. In addition, I would also like to thank all the personnel working at the Laboratory of Digital Techniques in Instituto de Investigaciones Científicas y Técnicas para la Defensa (CITEDEF): Martín Morales, Ariel Dalmas Di Giovanni, Sergio Saluzzi and Sebastián Alavarez; who permanently collaborate in the development of software and hardware for this type of applications. Finally, I am very grateful to the CITEDEF authorities for the logistics support and to MINDEF (Defense Department of Argentina), which provides financial support to this type of programs and projects.

## REFERENCES

1. *Electronic Industries Association. Engineering Department, Interface between data terminal equipment and data communication equipment employing serial binary data interchange. 1em plus 0.5em minus 0.4em Electronic Industries Association, Engineering Dept., 1969, OCoLC: 652241546.*
2. *Emtech S.A. Development kit 3px1. [Online]. Available: http://www.emtech.com.ar/producto/placa-3px1*
3. *Consultative Committee for Space Data Systems. (2000) Radio frequency and modulation systems - part 1: Earth stations and spacecraft. [Online]. Available: https://public.ccsds.org/Pubs/401x0b25.pdf*
4. *P. Chu, FPGA prototyping by Verilog examples. Wiley, ISBN: 78-0-470-18532-2.*
5. *A. Stacul et al., "Diseño, desarrollo e implementación de una estación terrena para cohetes sonda," in VI Congreso de Microelectrónica Aplicada, Buenos Aires, Argentina, 2015, ISBN: 978-987-3806-24-7.*
6. *M. Frerking, Digital Signal Processing in Communication Systems. 1em plus 0.5em minus 0.4em Prentice Hall, 1994.*
7. *A. Oppenheim and R. Schafer, Discrete-Time Signal Processing. 1em plus 0.5em minus 0.4em Prentice Hall, 1989.*
8. *R.C.C. Telemetry Group, Pulse code modulation standards. Telemetry Standards - IRIG standard 106-13 - part 1 - chapter 4. Telemetry Group, 2013.*
9. *MathWorks Inc. (2016) Matlab - the lenguaje of technical computing. [Online]. Available: https://www.mathworks.com/products/matlab.html*

10. *M. R. Valido et al., "Metodología de diseño en fpga usando xilinx system generator," 2012.R. Crochiere and L. Rabiner, Multirate Digital Signal Processing. 1em plus 0.5em minus 0.4em Prentice Hall, 1983.*
11. *MathWorks Inc. (2016) Simulink. [Online]. Available: http://www.mathworks.com/products/simulink/*
12. *Tiwari, Arpita, Ravi Mohan, And Divyanshu Rao. "Fpga Implementation Of Advanced Uart Controller Using Vhdl."*
13. *Xilinx. Spartan-6 family overview. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds160.pdf*
14. *Xilinx Inc. (2016) System generator for dsp user guide. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/sysgen_user.pdf*
15. *E. Hogenauer, "An ecnomical class of digital filters for decimation and interpolation," 1981.*