NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

THESIS

MODELING AND SIMULATION OF A NON-COHERENT FREQUENCY SHIFT KEYING TRANSCEIVER USING A FIELD PROGRAMMABLE GATE ARRAY (FPGA)

by

Konstantinos Voskakis

September 2008

Thesis Advisor:	Frank Kragh
Thesis Co-Advisor:	Peter Ateshian
Second Reader:	Roberto Cristi

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2008	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE Modeling and Simulation of a Non-Coherent Frequency Shift Keying Transceiver Using a Field Programmable Gate Array (FPGA).			**5. FUNDING NUMBERS**
6. AUTHOR(S) Konstantinos Voskakis			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

In this thesis, the principals of Software Defined Radio are demonstrated by implementing a Binary Frequency Shift Keying (BFSK) receiver-transmitter in a Field Programmable Gate Array (FPGA). After introducing the theory behind the Non-Coherent BFSK demodulation implemented at the receiver, the design of transmitter and receiver is illustrated. The design environment of choice is Mathworks'® Simulink and Xilinx® System Generator, a dedicated library for Mathworks' Simulink. The design is downloaded to a Virtex-4 FPGA.

The receiver is Non-Coherent (NC) in the sense that the receiver need not know the phase of the incoming signal. A feedback circuit is responsible for both packet and bit synchronization. Also, the receiver is implemented using non-coherent match filters instead of low pass filters which would be easier, but would degrade the performance. Finally, some interesting experiences that were gained during the learning process are discussed.

In Appendix A, we evaluate different technological options in implementing communication modulating techniques and Software Defined Radio. These options include Digital Signal Processors, Field Programmable Gate Arrays, General Purpose Processors and Application Specific Integrated Circuits and a comparison between these choices is made.

.

14. SUBJECT TERMS Software Defined Radio, Field Programmable Gate Array, Digital Signal Processing Chip, Application Specific Integrated Circuit, Binary Frequency Shift Keying, Xilinx, System Generator			**15. NUMBER OF PAGES** 125
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

MODELING AND SIMULATION OF A NON-COHERENT FREQUENCY SHIFT KEYING TRANSCEIVER USING A FIELD PROGRAMMABLE GATE ARRAY (FPGA)

Konstantinos Voskakis
Lieutenant Junior Grade, Hellenic Navy
B.S., Hellenic Naval Academy, 1999

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2008

Author:	Konstantinos Voskakis
Approved by:	Frank Kragh Thesis Advisor
	Peter Ateshian Thesis Co-Advisor
	Roberto Cristi Second Reader
	Jeffrey Knorr Chairman, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

In this thesis, the principals of Software Defined Radio are demonstrated by implementing a Binary Frequency Shift Keying (BFSK) receiver-transmitter in a Field Programmable Gate Array (FPGA). After introducing the theory behind the Non-Coherent BFSK demodulation implemented at the receiver, the design of both transmitter and receiver is illustrated. The design environment of choice is Mathworks'® Simulink and Xilinx® System Generator, a dedicated library for Mathworks' Simulink. The design is downloaded to a Virtex-4 FPGA.

The receiver is Non-Coherent (NC) in the sense that the receiver need not know the phase of the incoming signal. A feedback circuit is responsible for both packet and bit synchronization. Also, the receiver is implemented using non-coherent match filters instead of low pass filters which would be easier, but would degrade the performance. Finally, some interesting experiences that were gained during the learning process are discussed.

In Appendix A, we evaluate different technological options in implementing communication modulating techniques and Software Defined Radio. These options include Digital Signal Processors, Field Programmable Gate Arrays, General Purpose Processors and Application Specific Integrated Circuits and a comparison between these choices is made.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Software Defined Radio (SDR) is a new and fascinating idea having its roots in the early'90's. Technologic constraints prevented this idea from becoming a reality at the beginning, but the development of powerful Field Programmable Gate Arrays (FPGAs) has increased interest in the SDR concept. FPGAs combine versatility, reconfigurability and upgradability that is hard to find in any other device.

A simple way to make Software Defined Radio a reality is to store transceiver designs for many modulation schemes in memory and download the selected one to an FPGA as needed. This goal is accomplished when transceivers for all modulation schemes and services of choice are designed and synthesized for the target FPGA. Starting this procedure, a Binary Frequency Shift Keying transmitter and receiver design is the main purpose of this thesis.

BFSK is the modulation that uses two different frequencies for the binary 0 and binary 1 symbols of the input stream. This modulation is simple but there are still many challenges for the timing synchronization of the receiver. A non-coherent receiver was chosen to eliminate the need for phase synchronization. The description of such a receiver along with the timing issue is addressed in Chapter II. Given that Forward Error Correction is used in the transceiver design, an introduction of convolutional encoding is also given in Chapter II.

To make a good design, the proper software must support the effort. System Generator is a program available by Xilinx to help the designing of a project, offering an environment familiar to most engineers, namely Mathworks' Simulink with a complete library of synthesizable blocks. This program is supported by the Integrated Software Environment (ISE) Design Suite, which is the Xilinx software that accepts the code generated by System Generator and continues the task of implementing the design to the FPGA and testing the resulting downloaded design. A more complete description is included in Chapter III.

The transmitter and receiver design made under System Generator is presented in Chapter IV. A preamble is attached before each packet to facilitate the synchronization of the receiver. Before that happens, the message bits are encoded using convolutional encoding. Then output bits of these procedures are transmitted based on the general rule of the BFSK modulation scheme where binary zeros and ones correspond to two different frequencies. The receiver uses non-coherent matched filters to extract the transmitted bits from the received waveform. Also, there exists a timing circuit that provides the bit and packet synchronization. Finally, the preamble is stripped off and the remaining bits are inserted to a Viterbi decoder that yields the message bits.

The verification of the design follows in Chapter V. This is carried out in the System Generator environment, by examining the signal at different points in the design, and in using Matlab code that simulates part of the receiver. The results are shown and the design can be considered successful. The problems that were encountered during the design are also addressed in the second half of this chapter.

A closer look at current FPGA technology is included in Appendix A. Different technological options in implementing communication modulating techniques and Software Defined Radio are discussed. These options include Digital Signal Processors (DSP), Field Programmable Gate Arrays, General Purpose Processors (GPP) and Application Specific Integrated Circuits (ASICS) and a comparison between these choices is made. The results of the comparison are that a heterogeneous design that includes all three of a DSP, a GPP and an FPGA can provide the maximum performance and versatility. DSPs are better performing in sequential logic, whereas FPGAs are more efficient in executing parallel tasks. GPP are used in supporting the different network protocols and other similar tasks.

A lower level description compared to the design flow of Chapter IV of each block is included in Appendix B. The reason for many choices made in the parameters window of every block is mentioned next to the actual value of the parameter. In this way, the rebuilding of the design can be made solely based on this appendix. In the same time, further insight into the dependence of the desired results upon the chosen parameters is provided. In Appendix C, the Matlab code that helped the verification of the receiver design is included.

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank my advisor, Assistant Professor Frank Kragh, my Co-Advisor, Instructor Peter Ateshian and my Second Reader, Professor Roberto Cristi for their help to make my design work. They offered me their knowledge and their valuable time to help me overcome many of the difficulties I encountered. I would also like to thank Professor Alexander Julian for his advice on using System Generator, a program that he has mastered.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND ON SOFTWARE-DEFINED RADIO

A *software radio* is a radio which uses programmable hardware. Software is used to configure the hardware to meet different communication scheme specifications as well as to support several different services. According to J. Mitola (1993) "a *software radio* (SR) is a set of Digital Signal Processing (DSP) primitives, a metalevel system for combining the primitives into communications systems functions (transmitter, channel model, receiver . . .) and a set of target processors on which the software radio is hosted for real-time communications [1]." This concept is in contrast to common radio devices implemented in specific hardware, which provide a limited capability of switching between modulation schemes and services, mainly due to the static hardware used. An ideal SR receiver directly samples the antenna output. A *software-defined radio* (SDR) is a practical version of an SR. The received signals are sampled after a suitable band selection filter and frequency down conversion [2].

The flexibility and reconfigurability demonstrated by the SDR have become a reality largely due to the evolution of digital electronics processes defined in software instead of using static and application specific integrated circuits such as mixers, filters, amplifiers, modulators, demodulators, and detectors.

The concept of SDR has progressed further because of the advancement of Field Programmable Gate Arrays (FPGAs) and is currently a field of intensive research, even though the FPGAs are not the only platform upon which SDR can be based. General Purpose Processors (GPPs) and dedicated Digital Signal Processing (DSP) chips provide an alternative to FPGAs, having their own pros and cons. Nevertheless, the versatility that FPGAs demonstrate makes them unique in many aspects.

B. GOALS OF RESEARCH AND CONCEPTS

Recent technological advancements have allowed FPGAs to transform from an auxiliary device to a signal processing engine. Nowadays, not only can FPGAs compete with dedicated circuits, but also they give life to sectors of science that need their versatility. They have enhanced the Software Defined Radio concept, which is a great advancement versus the normal Radio concept.

The main goal of this research is the design of a Binary Frequency Shift Keying (BFSK) transmitter and receiver. The BFSK modulation is used for the illustration of the techniques in designing a communication system in FPGAs. The reason is that BFSK is a simple, but robust modulation that can be received non-coherently. The design process also helps acquiring a greater experience in the design of FPGAs using some of the easier to use but powerful schematic, synthesis and place and route tools available today.

The second goal of the research is to track the advancements made in the field of FPGAs and inform on the usefulness and possible implementations of FPGAs.

C. METHODOLOGY AND SCOPE OF THE RESEARCH

Xilinx's System Generation 10.1 SP2 is the schematic tool used to design a BFSK transceiver. After verifying that the design worked correctly, the code of the design was automatically generated by System Generator and the code was loaded into the Integrated Software Environment (ISE™) to be synthesized, placed and routed, and finally downloaded to the target FPGA, which is a Xilinx's Virtex-4. Nevertheless, the verification of the implementation on the chip was not done due to time constraints.

The main challenge to the design is to achieve the synchronization required in order for the receiver to be able to distinguish the beginning and end of different packets of incoming data. The length of the packet was chosen to be fixed at 128 bits and the first 8 bits compose the preamble that facilitates the bit synchronization and packet detection.

D. BENEFITS OF THE RESEARCH

The concept of Software Defined Radio is fascinating but complex. Designing different modulation schemes that can be downloaded to an FPGA is an easy way to design a simple Software Defined Radio. On the other hand, all digital modulations share the same basic principles; thus, synchronization techniques from one modulation can be borrowed and modified to work with another modulation scheme. A fully working digital BFSK transceiver is simulated in this thesis.

The research made regarding FPGAs unveiled the fact that while technology is changing, some arguments, like power consumption, that were once against the use of FPGAs, may be today their strong point. The system designer must always be up-to-date and adaptive regarding new technologies since FPGAs are going to be used more extensively in the future [3].

E. ORGANIZATION OF THE THESIS

Chapter II includes background regarding Binary Frequency Shift Keying. A Non-Coherent BFSK receiver is presented in order to facilitate the understanding of the design that was implemented in an FPGA. Also, the concept of convolutional encoding is introduced.

Chapter III contains the description of the design environment used, namely Xilinx's System Generator, ISE and ChipScope Pro along with the characteristics of the board used for the design. The high level of maturity and the friendly interface of the software product played a key role in the successful completion of the whole project.

Chapter IV gives a detailed description of the software design of a BFSK transmitter and receiver. The description includes the logic for the design choices that were made, the reason behind the choice of specific components, and the explanation of the function of many blocks.

Chapter V discusses the results taken by simulation in the design environment. Input and output are compared using Matlab and the correctness of the results is discussed.

Chapter VI includes an outline of the work made, the significant results taken, the limitations of the design, and recommendations for future work.

In Appendix A, an extensive background regarding FPGAs is given, explaining that they are well suited for Software Defined Radios. FPGA's positive and negative aspects are mentioned and are compared with General Purpose Processors, Digital Signal Processors and Application Specific Integrated Circuits.

In Appendix B, a detailed description of the design is given in a per figure and per block basis. Reading Appendix B in parallel with Chapter IV provides a better understanding of the blocks and the reason they were used.

In Appendix C, the Matlab code used to verify the results taken by System Generator is shown.

In this chapter, the concept of Software Defined Radio was introduced. The idea of SDR is realized by building a BFSK transceiver using an FPGA. In order to provide a solid background to facilitate understanding the design, the next chapter discusses BFSK modulation and demodulation and convolutional encoding.

II. BINARY FREQUENCY SHIFT KEYING MODULATION SCHEME AND CONVOLUTIONAL ENCODING

BFSK is a basic digital modulation scheme. Its concept is not presented in depth, but can be found in any introductory textbook concerning communications. The textbook used as a reference for this brief introduction is [4, p. 198] along with [5], which both include a detailed description of the BFSK modulation scheme and a BFSK receiver. An introduction to convolutional encoding is also given at the end of this chapter.

A. BFSK MODULATION

In BFSK, two distinct frequencies are chosen to represent the two possible values of a bit. The equation that describes the transmission signal *s* of the i^{th} bit that is produced by this modulation technique is the following [5]:

$$s(t) = \sqrt{2}A_c \cos\left[2\pi\left(f_c + b(t)\cdot\frac{\Delta f}{2}\right)t + \theta_i\right], \quad \text{for } iT_b \geq t \geq (i-1)T_b \tag{2.1}$$

where T_b is the bit duration, A_c is the carrier's amplitude, f_c is the mean signaling frequency in Hz, *b(t)* is the value of the transmitted bit in bipolar form where 1 corresponds to bit 1 and -1 corresponds to bit 0, Δf is the frequency separation of the two frequencies, and θ_i is the i^{th} bit phase.

A BFSK receiver is distinguished by coherent or non-coherent depending if the knowledge of the phase information of the received signal is prerequisite for the receiver to work properly. In this thesis, the receiver of choice is non-coherent which decreases the complexity of the receiver circuit, eliminating the need for an extra circuit that would acquire the phase information. The configuration that allows the realization of a Non-Coherent (NC) reception is the energy detector. A diagram of a NC BFSK receiver is shown in Figure 1 [5].

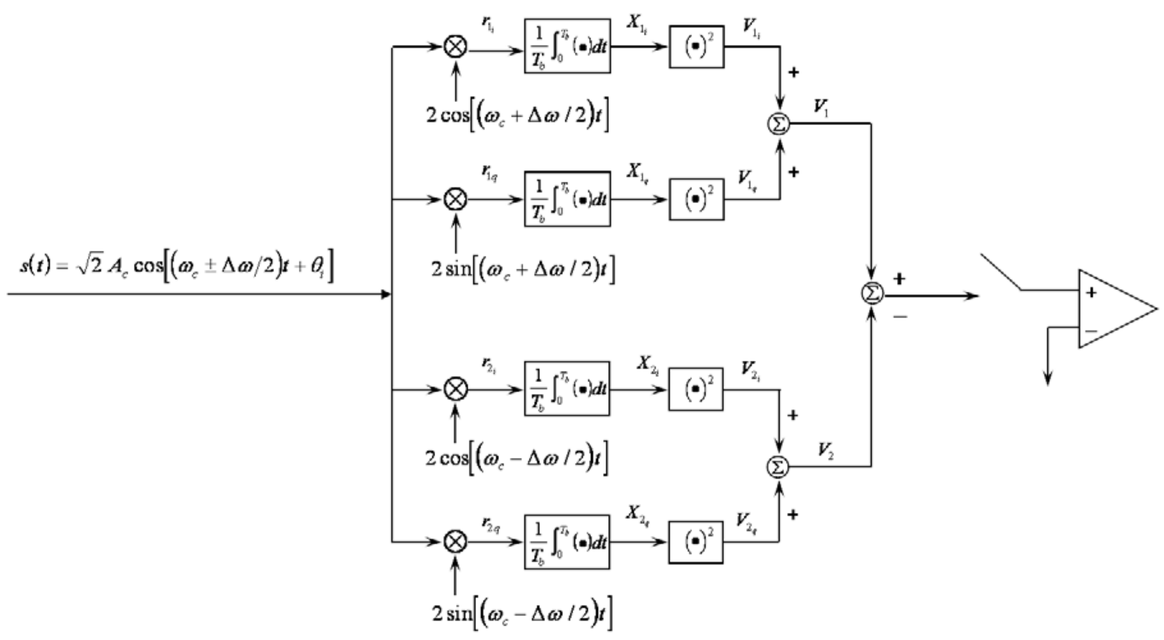

$s(t)=\sqrt{2}A_c\cos[(\omega_c \pm \Delta\omega/2)t+\theta_i]$
r_{1_i}
$\frac{1}{T_b}\int_0^{T_b}(\bullet)dt$
X_{1_i}
$(\bullet)^2$
V_{1_i}
$2\cos[(\omega_c+\Delta\omega/2)t]$
V_1
r_{1q}
X_{1_q}
V_{1_q}
$2\sin[(\omega_c+\Delta\omega/2)t]$
r_{2_i}
X_{2_i}
V_{2_i}
$2\cos[(\omega_c-\Delta\omega/2)t]$
V_2
r_{2q}
X_{2_q}
V_{2_q}
$2\sin[(\omega_c-\Delta\omega/2)t]$
+
−

Figure 1 Block diagram of a NCBFSK receiver (From: [5]).

The received signal is distributed in two distinct paths, one for each frequency. To each path, the signal is further divided among two branches; one branch is configured to detect the in-phase (I) signal and the other branch the quadrature (Q) signal of the respective frequency. Each branch consists of a mixer, an integrator and the squaring function. Both branches and the summer at their end consist of a non-coherent matched filter. The term non-coherent matched filter means that this filter does not try to match the carrier phase, but only the envelope of the signal [5, pp. 256-258].

The structure is self similar, thus, the analysis made for the case of bit '1' transmitted is exactly inverse to the case of bit '0' transmitted. For a bit '1' transmitted, the input to the integrator of the top path is given by [5]:

$$\begin{aligned} r_{1_i}(t) &= 2s(t)\cos\left[\left(\omega_c+\frac{\Delta\omega}{2}\right)t\right] \\ &= 2\sqrt{2}A_c\cos\left[\left(\omega_c+\frac{\Delta\omega}{2}\right)t+\theta_i\right]\cdot\cos\left[\left(\omega_c+\frac{\Delta\omega}{2}\right)t\right] \\ &= \sqrt{2}A_c\left\{\cos(\theta_i)+\cos\left[2\left(\omega_c+\frac{\Delta\omega}{2}\right)t+\theta_i\right]\right\}. \end{aligned} \tag{2.2}$$

Similarly, the input to the other integrator in the top non-coherent matched filter is

$$\begin{aligned} r_{1_q}(t) &= 2s(t)\sin\left[\left(\omega_c + \frac{\Delta\omega}{2}\right)t\right] \\ &= 2\sqrt{2}A_c \cos\left[\left(\omega_c + \frac{\Delta\omega}{2}\right)t + \theta_i\right]\cdot\sin\left[\left(\omega_c + \frac{\Delta\omega}{2}\right)t\right] \\ &= \sqrt{2}A_c\left\{-\sin(\theta_i) + \sin\left[2\left(\omega_c + \frac{\Delta\omega}{2}\right)t + \theta_i\right]\right\}. \end{aligned} \tag{2.3}$$

The integrator outputs for the two branches of the top path are:

$$X_{1_i}(T_b) = \sqrt{2}A_c\left\{\cos\theta_i + \frac{R_b}{4\pi(f_c + \Delta f/2)} \bullet \begin{bmatrix} \cos\theta_i \cdot \sin\left[4\pi(f_c + \Delta f/2)T_b\right] + \\ \sin\theta_i\left\{\cos\left[4\pi(f_c + \Delta f/2)T_b\right] - 1\right\} \end{bmatrix}\right\} \tag{2.4}$$

for the I channel, and

$$X_{1_q}(T_b) = \sqrt{2}A_c\left\{-\sin\theta_i + \frac{R_b}{4\pi(f_c + \Delta f/2)} \bullet \begin{bmatrix} \sin\theta_i \cdot \sin\left[4\pi(f_c + \Delta f/2)T_b\right] + \\ \cos\theta_i\left\{\cos\left[4\pi(f_c + \Delta f/2)T_b\right] - 1\right\} \end{bmatrix}\right\} \tag{2.5}$$

for the Q channel.

If the right conditions are met, the above expressions are simplified. Thus, when f_c is chosen to be an integer multiple of half the bit rate R_b and Δf is chosen to be an integer multiple of the bit rate R_b, where $R_b = \frac{1}{T_b}$, only the first terms of the above expressions are non-zero. These conditions are known as *orthogonal* signaling [3, pp. 200-204]. Following that restriction, the outputs of the integrators of the top NCMF are:

$$X_{1_i}(T_b) = \sqrt{2}A_c \cdot \cos\theta_i \tag{2.6}$$

and

$$X_{1_q}(T_b) = -\sqrt{2}A_c \cdot \sin\theta_i\,. \tag{2.7}$$

The outputs of the squaring block are

$$V_{1_i}(T_b) = 2A_c^{\,2} \cdot \cos^2\theta_i \tag{2.8}$$

and

$$V_{1_q}(T_b) = 2A_c^{\,2} \cdot \sin^2\theta_i\,. \tag{2.9}$$

Summing the outputs of the two branches yields

$$V_1(T_b) = 2A_c^2(\sin^2\theta_i + \cos^2\theta_i) = 2A_c^2 \tag{2.10}$$

as the output of the top path. The output of the I-channel mixer in the bottom NCMF in Figure 1 is

$$\begin{aligned} r_{2_i}(t) &= 2s(t)\cos\left[\left(\omega_c - \frac{\Delta\omega}{2}\right)t\right] \\ &= 2\sqrt{2}A_c\cos\left[\left(\omega_c + \frac{\Delta\omega}{2}\right)t + \theta_i\right]\cdot\cos\left[\left(\omega_c - \frac{\Delta\omega}{2}\right)t\right] \\ &= \sqrt{2}A_c\left[\cos\left(\Delta\omega\cdot t + \theta_i\right) + \cos\left(2\omega_c t + \theta_i\right)\right]. \end{aligned} \tag{2.11}$$

The output of the Q-channel mixer in the bottom NCMF in Figure 1 is

$$\begin{aligned} r_{2q}(t) &= 2s(t)\sin\left[\left(\omega_c - \frac{\Delta\omega}{2}\right)t\right] \\ &= 2\sqrt{2}A_c\cos\left[\left(\omega_c + \frac{\Delta\omega}{2}\right)t + \theta_i\right]\cdot\sin\left[\left(\omega_c - \frac{\Delta\omega}{2}\right)t\right] \\ &= \sqrt{2}A_c\left[-\sin\left(\Delta\omega\cdot t + \theta_i\right) + \sin\left(2\omega_c t + \theta_i\right)\right]. \end{aligned} \tag{2.12}$$

The outputs of the integrators of the bottom NCMF are

$$X_{2_i}(T_b) = \frac{\sqrt{2}A_cR_b}{2\pi}\left\{\begin{array}{l} \dfrac{\cos\theta_i}{\Delta f}\sin\left(2\pi\Delta f\cdot T_b\right) + \dfrac{\sin\theta_i}{\Delta f}\left[\cos\left(2\pi\Delta f\cdot T\right) - 1\right] + \\ \dfrac{1}{2f_c}\left\{\cos\theta_i\cdot\sin\left(4\pi f_cT_b\right) + \sin\theta_i\left[\cos\left(4\pi f_cT_b\right) - 1\right]\right\} \end{array}\right\} \tag{2.13}$$

for the I-channel, and

$$X_{2_q}(T_b) = \frac{\sqrt{2}A_cR_b}{2\pi}\left\{\begin{array}{l} -\dfrac{\sin\theta_i}{\Delta f}\sin\left(2\pi\Delta f\cdot T_b\right) + \dfrac{\cos\theta_i}{\Delta f}\left[\cos\left(2\pi\Delta f\cdot T\right) - 1\right] + \\ \dfrac{1}{2f_c}\left\{\sin\theta_i\cdot\sin\left(4\pi f_cT_b\right) - \cos\theta_i\left[\cos\left(4\pi f_cT_b\right) - 1\right]\right\} \end{array}\right\} \tag{2.14}$$

for the Q-channel.

If orthogonal signaling is chosen, i.e., $\Delta f = mR_b$ and $f_c = n\frac{R_b}{2}$ where n and m are integers, the outputs of the integrators in the bottom NCMF simplify to $X_{2_i}(T_b) = 0$ and $X_{2_q}(T_b) = 0$. This in turn yields

$$V_2(T_b) = 0 \tag{2.15}$$

and using equation (2.10) and (2.15) the output of the subtraction of the paths is

$$V_1 - V_2 = 2A_c^2 . \tag{2.16}$$

For the case that bit ‘0’ is transmitted, the whole process is inverted and the respective outputs of the two paths would be $\upsilon_1(T_b) = 0$, and $\upsilon_2(T_b) = 2A_c^2(\sin^2\theta_i + \cos^2\theta_i) = 2A_c^2$. Hence, the output of the subtraction of the two paths is now $-2A_c^2$. Sampling the final output at the end of the duration of each bit reveals the value of the transmitted bit.

It is obvious that this implementation relies heavily on proper bit synchronization, which means that the receiver should know the exact duration of each bit and when each bit ends. To acquire this information an extra circuit is needed and when the timing information is incorrect, severe degradation of the performance of the receiver may result. Many Time Error Detectors (TEDs) for discrete time implementations are presented in [6], including the Early-Late TED, the Zero Crossing TED, and the Gardner TED.

In summary, the energy of the two branches of each path are added and compared to the energy of the other path. The decision made about the received bit is in favor of the bit that corresponds to the frequency of the path with the highest energy. In order to minimize the cross product of energies, the frequencies used must be orthogonal which implies a tone spacing that is a multiple of the bit rate and a center frequency that is a multiple of half the bit rate [5].

B. CONVOLUTIONAL ENCODING

Encoding in digital communications is used for forward error correction. The convolutional codes are one of the two most commonly used along with block codes. They were introduced in 1955 by Elias [7].

Convolutional codes are characterized by the *code rate* $r = \frac{k}{n}$, where k is the length of the input word and n is the length of the output word, and by the memory order

m. The memory order m is the number of memory elements that are included in the encoder and is a crucial parameter of the performance of a code. Each code can be uniquely described by a matrix with octal numbers as elements. The number of columns in this matrix corresponds to the n parameter and the number of rows to the k parameter. The actual value of the octal number reveals the interconnections that yield the respective output, counting in binary from right to left. In the example taken from [8], in Figure 2, we can identify an $r = \frac{1}{2}$ code with a convolutional code array of [133,171]. The number 133 is the octal equivalent of binary 1011011 and corresponds to output C_1 and 171 is the octal equivalent of binary 1111001 and corresponds to output C_0. This specific code is an industry standard code for $m = 6$. The constraint length κ for the case of $k = 1$ is $\kappa = m + 1$. In this thesis, the industry standard convolutional code for $r = \frac{1}{2}$ and $\kappa = 3$, namely [7 5], is used.

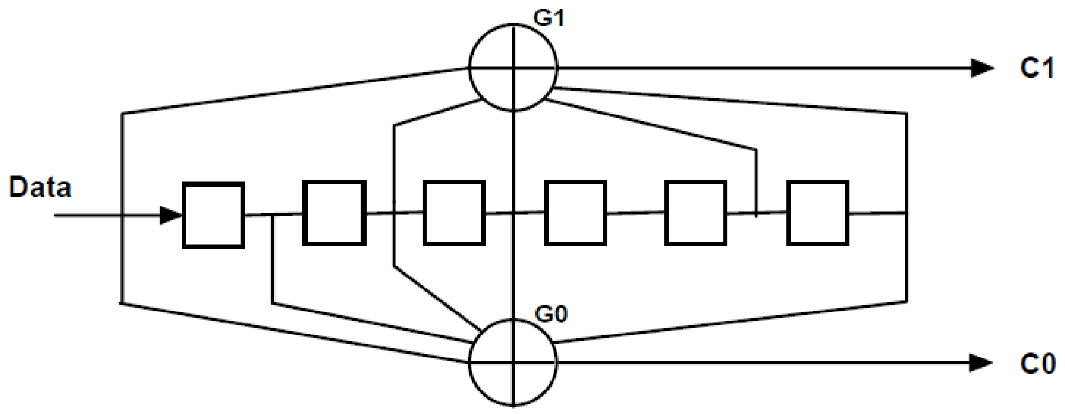


Figure 2 Convolutional Encoder Block Diagram of code rate $r = \frac{1}{2}$ and $\kappa = 7$.

Convolutional encoded streams are usually decoded by Viterbi decoders, invented by Viterbi [9]. Viterbi decoders implement maximum likelihood decoding with a slight performance penalty due to finite decoder memory [10].

This chapter has explained the fundamental principles required to understand the NCBFSK transmitter and receiver design detailed in the remainder of this document. The next chapter describes the software and hardware design tools used in this design effort.

THIS PAGE INTENTIONALLY LEFT BLANK

III. DESIGN ENVIRONMENT

Xilinx offers a full suite of programs that provides an integrated development environment for its FPGAs. This suite is named Integrated Software Environment (ISE) Design Suite and the main programs that are included are System Generator for DSP, ISE Project Navigator, ChipScope Pro Tool, PlanAhead and AccelDSP Synthesis Tool [11]. Not all of these tools were used because each program has a very specific functionality, some of which were not needed. System Generator was used as the main design entry and simulation program and ISE Project Navigator as the program that implements the design into the targeted Xilinx device.

A. SYSTEM GENERATOR

System Generator is a FPGA design program that offers the necessary libraries of blocksets, making use of the Mathworks' Simulink design environment. Simulink is a schematic tool that is part of Matlab and is known for its efficiency and ease of use among engineers. For this reason, System Generator (Sysgen) chose this environment to offer the system modeling, making available the mixing of components from Simulink and Sysgen for simulation purposes (Figure 3). Sysgen also provides automatic code generation that can be then downloaded to Xilinx's FPGAs. The Hardware Description Language (HDL) that is used during code generation can be chosen from the Sysgen token and is either VHDL or Verilog [12].

The blocks offered by Sysgen are guaranteed to be synthesizable, solving a great problem for the designer. Blocks are schematic components that implement primitive functions and offer the option of default along with customizable inputs and outputs that can be interconnected. More complex blocks exist as well, yielding the opportunity to construct a complex design without much effort. A full list and description of all the available blocks is included in [13] and a more technical description of the Intellectual Properties blocks is included in [14]. Most of these blocks are DSP related and only a few are dedicated to communications. In the later case, an extra license is usually needed in order for them to integrate into the design. Their color is green by default and is clearly shown in Figure 3. The block ‘System Generator’ is mandatory to every design and the blocks ‘Gateway In’ and ‘Gateway Out’ define the limits of the design that are going to be translated in an FPGA circuit. The current version of Sysgen is 10.1 with Service Pack 2.

Figure 3 illustrates a very simple example, where a Finite Impulse Response (FIR) Filter is designed. The input is supplied by Matlab and the output is viewed by double clicking on the ‘Scope.’ The parameters of the single Xilinx block used are defined in the respective window that appears when the FIR block is selected. Neither of the Simulink blocks, ‘From Workspace’ and ‘Scope,’ are synthesizable. They are only used during the design phase for simulation purposes.

Other parameters that are common to many Sysgen blocks are the format and width of the output values [13, p. 44]. There are blocks dedicated to manipulate the data type and alter their internal structure. For example, the Enable and Reset signal are only allowed to be Boolean, thus an unsigned one bit integer must be reinterpreted as a Boolean number. This is accomplished by the blocks ‘Reinterpret’ or ‘Convert.’

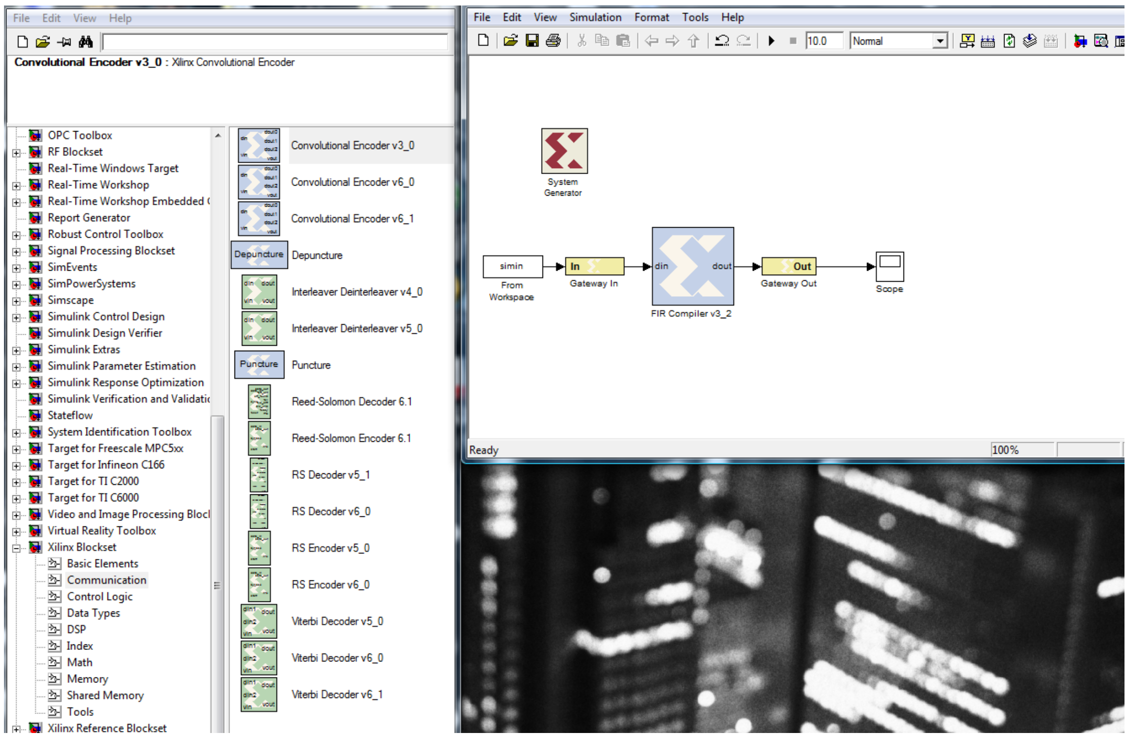


Figure 3 Example of the environment and the blocks offered by Sysgen.

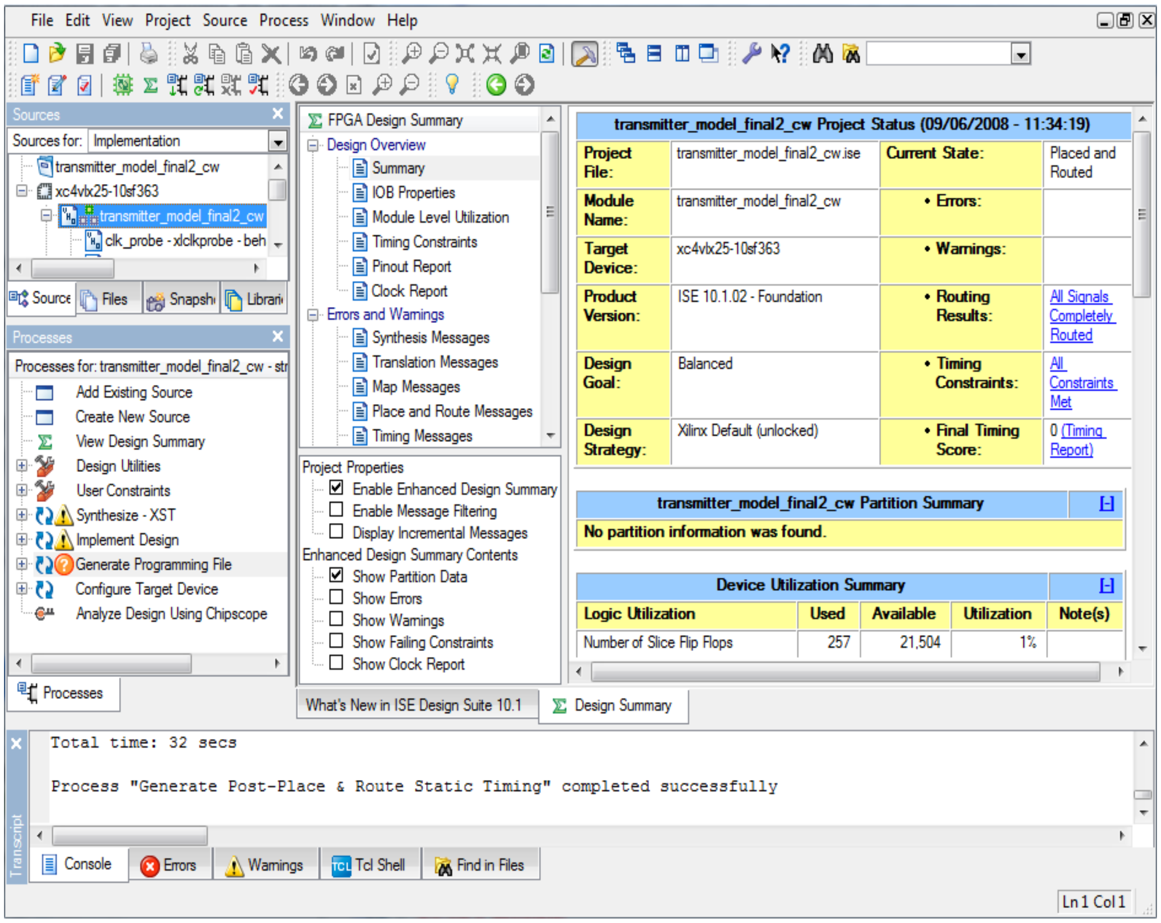
Another block of special use is ‘MCode’ [13, p. 239]. It allows writing a program in Matlab and saving it in the block. Then, Sysgen is responsible for synthesizing this program. There are many constraints regarding the commands that can be used in such a program. As an example, the division by a number different from a power of two is not supported. Nevertheless, this block is very useful to describe state machines, and as such, it has been used many times in the BFSK design.

B. ISE PROJECT MANAGER

After finishing with the design and generating the code for the HDL language of choice via Sysgen, the source file is loaded into the ISE Project Manager as a project. This Manager is responsible for the synthesis, implementation, and verification of the design and the target device configuration [15].

After loading a project created by Sysgen, source files can be added, created or modified. Other available processes under the Processes Window, as shown in the left column in Figure 4, are as follows:

- Add timing constraints or define Input Output (IO) pins under User Constraints choice.
- Synthesize the project or generate post-synthesis simulation under Synthesize –XST. At this step HDL programs are converted to netlist files that are used by the implementation step.
- Translate the logical design (netlist file) to a physical file format, to make the mapping of the design to the FPGA, and to place and route the mapping to the FPGA of choice under Implement Design choice. The placement step includes the decision made by the program regarding where to place the logic elements given the internal structure of the target device. Then, routing is responsible for finding the optimized connecting paths between these placed components.
- Generate the programming file that will be installed into the FPGA under Generate Programming File,
- Configure Target Device, and
- Use the ChipScope Pro program to verify the actual implementation into the FPGA under Analyze Design Using Chipscope. Every step of the implementation process described above has its own tools for testing and simulating the design. ChipScope is responsible to check the functionality of the final design installed into the FPGA.



Project Properties
- [✓] Enable Enhanced Design Summary
- [ ] Enable Message Filtering
- [ ] Display Incremental Messages

Enhanced Design Summary Contents
- [✓] Show Partition Data
- [ ] Show Errors
- [ ] Show Warnings
- [ ] Show Failing Constraints
- [ ] Show Clock Report

transmitter_model_final2_cw Project Status (09/06/2008 - 11:34:19)

Project File:	transmitter_model_final2_cw.ise	**Current State:**	Placed and Routed
Module Name:	transmitter_model_final2_cw	**• Errors:**	
Target Device:	xc4vlx25-10sf363	**• Warnings:**	
Product Version:	ISE 10.1.02 - Foundation	**• Routing Results:**	All Signals Completely Routed
Design Goal:	Balanced	**• Timing Constraints:**	All Constraints Met
Design Strategy:	Xilinx Default (unlocked)	**• Final Timing Score:**	0 (Timing Report)

transmitter_model_final2_cw Partition Summary

No partition information was found.

Device Utilization Summary

Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Flip Flops	257	21,504	1%	

```
Total time: 32 secs

Process "Generate Post-Place & Route Static Timing" completed successfully
```

Figure 4 Project Navigator Main Window.

C. AVNET BOARD

The mainboard to be used for the project is designed by AVNET and is called the Xilinx® Virtex™-4 LX LC Development Kit interconnected with the Analog to Digital (A/D) and Digital to Analog (D/A) Converter P160 provided by Avnet as well.

The mainboard's key features are the Virtex XC4VLX25 FPGA, 10/100 Ethernet interface and 64 MB Double Date Rate (DDR) Synchronous Dynamic Random Access Memory (SDRAM). The Virtex XC4VLX25 is a low entry FPGA of the Virtex-4 family and contains 24,192 logic cells and 48 dedicated DSP cells called XtremeDSP (18-bits x 18-bits, two's complement, signed Multiplier). It is manufactured using the 90nm Copper CMOS Process and it has no possibility of using the embedded soft processor PowerPC 405 core, due to size constraints [16]. The Analog Module P160 features two 12-bit 53

Msps A/D converters and two 12-bit 165 Msps D/A converters yielding much flexibility for the design [17]. Nevertheless, this module has not been used in any test in this research, mainly due to time constraints. The description of its pins and interfaces is in [18].

The literature recommended for the Sysgen and ISE is limited to the Xilinx Manuals. These manuals are included in a help guide offered by Xilinx as an internet-accessible Acrobat file [19]. For System generator there is also a manual that includes introductory labs and block and program reference manuals in its support page under the documentation tab and the Design Tool choice [20]. Extensive documentation of the most complex blocks is given in the same page under the IP Cores choice [21]. For the ISE project manager the documentation can be reached through the help guide stated above after choosing ‘ISE Help’ [15].

Sysgen and ISE Project Manager were extensively used for the design and the generation of the programming file of the non-coherent Binary Frequency Shift Keying Transmitter-Receiver presented in the next chapter. The plethora of tools offered by these programs made the design straight-forward, compared to writing directly to an HDL language. Xilinx is also supporting its programs online, making the troubleshooting easier.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. DESIGN FLOW

In this chapter, the logic flow of the design is discussed in detail. The basic principles of the BFSK transmitter and receiver illustrated in Chapter II are implemented in Simulink using Xilinx's blocks. The transmitter and receiver are separated into two different designs. The design is further exemplified in a per figure and per block basis in the Appendix B, where key parameters and Matlab code, where applicable, are also given.

A. TRANSMITTER

The transmitter, illustrated in Figure 5, is the combination of three distinct parts: the preamble, the data input and the modulation circuitry. The data is transmitted in blocks of 120 bits. An eight bit preamble with pattern 10101001 is attached in front of every packet to facilitate packet synchronization at the receiver. For simulation purposes, Simulink's blocks 'From Workspace' and 'To Workspace' were used to supply the design with input bits and store them, respectively. The results were also visually verified at each stage using 'Scope' blocks.

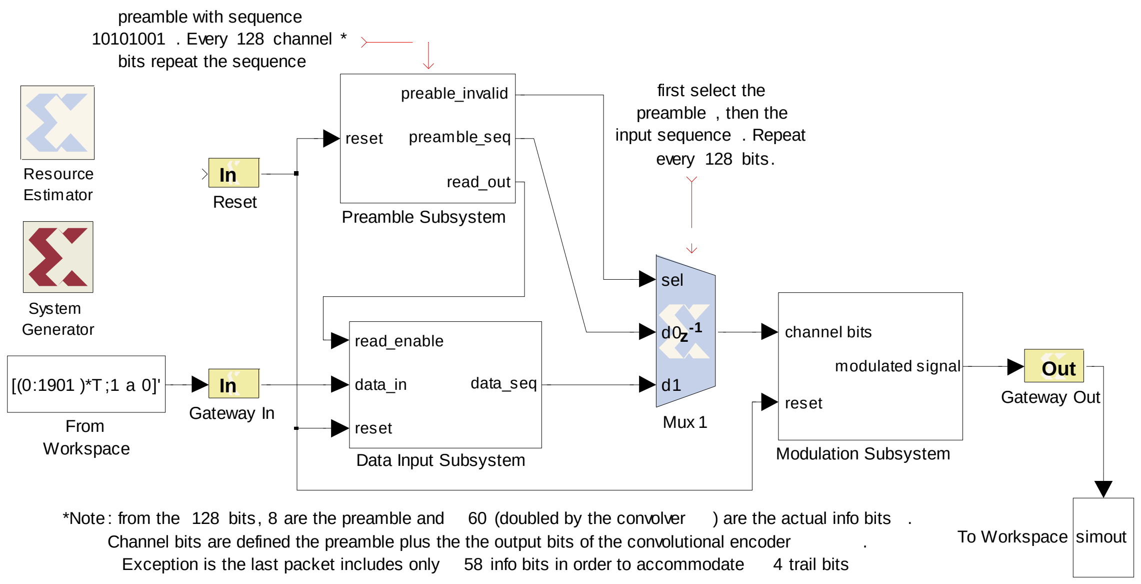

preamble with sequence 10101001 . Every 128 channel * bits repeat the sequence
Resource Estimator
System Generator
In
Reset
reset
preable_invalid
preamble_seq
read_out
Preamble Subsystem
first select the preamble , then the input sequence . Repeat every 128 bits.
sel
d0
z-1
d1
Mux 1
[(0:1901)*T ;1 a 0]'
From Workspace
In
Gateway In
read_enable
data_in
reset
data_seq
Data Input Subsystem
channel bits
modulated signal
reset
Modulation Subsystem
Out
Gateway Out
To Workspace
simout
*Note : from the 128 bits , 8 are the preamble and 60 (doubled by the convolver) are the actual info bits .
Channel bits are defined the preamble plus the the output bits of the convolutional encoder .
Exception is the last packet includes only 58 info bits in order to accommodate 4 trail bits

Figure 5 Transmitter's schematic diagram designed in Simulink/Sysgen environment.

1. Preamble Subsystem

The Preamble Subsystem (Figure 6) is responsible for the attachment of the preamble at the start of each packet. This subsystem is also responsible for the blocking of data bits whenever the preamble is transmitted and controlling the multiplexer 'Mux 1' in Figure 5, which selects the data or the preamble.

The counter drives two blocks. It counts up to 127 and restarts from 0. While the counters output is seven or less, the preamble is valid and is read out to the modulation subsystem via the multiplexer 'Mux1' in Figure 5. The 'read_out' and 'preamble_invalid' are low and the output of the counter is directly translated to an address in the 'ROM' block. The content of this address appears at 'ROM' output and again through the 'Mux1' in Figure 5 to the Modulation Subsystem. 'Mux1' is switched in the correct position by 'preamble_invalid' signal. 'Read_out' is responsible to block the message bits and let them be stored in a memory while the preamble is transmitted. The signals 'preamble_invalid' and 'read_out' take the same values and have different names merely for illustration purposes.

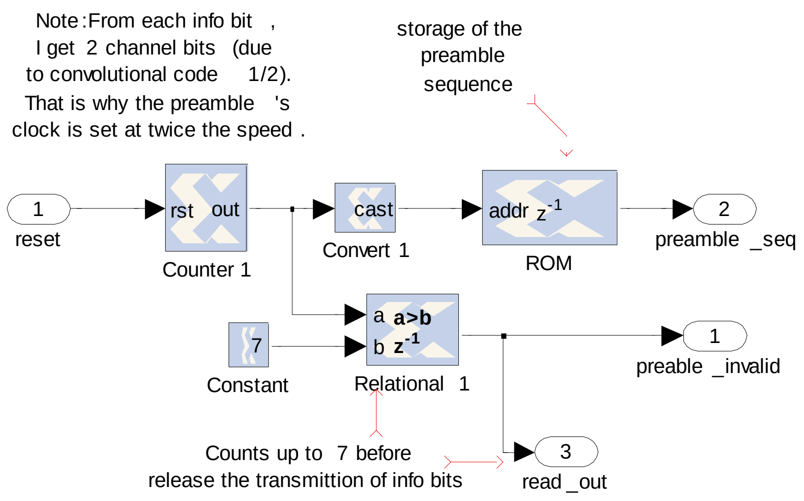

Note:From each info bit ,
I get 2 channel bits (due
to convolutional code 1/2).
That is why the preamble 's
clock is set at twice the speed .
storage of the
preamble
sequence
1
reset
rst out
Counter 1
cast
Convert 1
addr z-1
ROM
2
preamble _seq
7
Constant
a a>b
b z-1
Relational 1
1
preable _invalid
3
read _out
Counts up to 7 before
release the transmittion of info bits

Figure 6 Preamble Subsystem.

2. Data Input Subsystem

The Data Input Subsystem (Figure 7) is responsible for the convolutional encoding of the input sequence with a rate $r = \frac{1}{2}$ code and the subsequent storage of the encoded bit in a First In First Out (FIFO) memory. The two streams created by the 'Convolutional Encoder' block merge back into one stream by the 'Concat' and 'Parallel to Serial' blocks. It should be noted that these two last blocks can be replaced by a 'Time Division Multiplexer' block. The bit period of the final stream is half the period of the message bits due to the encoding with rate $r = \frac{1}{2}$. In the 'Convolutional Encoder' parameters window, the constraint length was set to 3, meaning that the encoder is using a register of two flip-flops. The encoding vector of choice was [7 5], as explained in Section B in Chapter II.

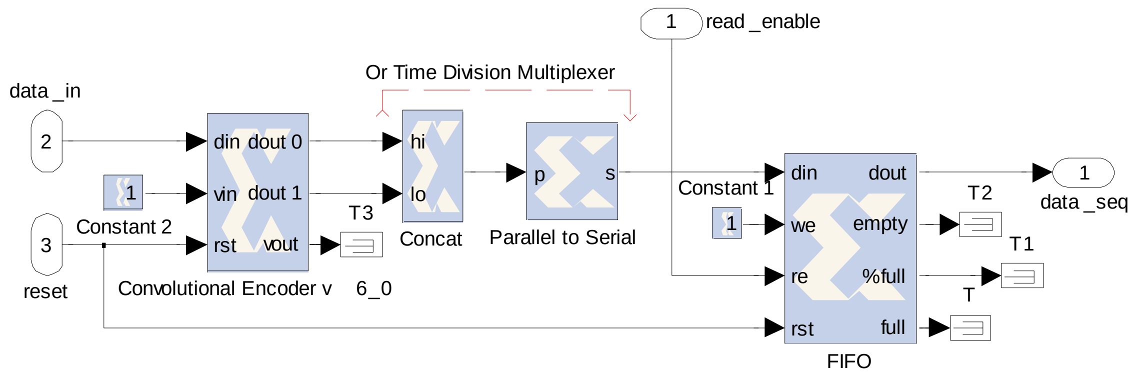


Figure 7 Data Input Subsystem.

After being stored in the 'FIFO' memory, the data waits for the enable signal of the Preamble Subsystem in order to exit. At the same time, the multiplexer 'Mux1' in Figure 5 is switched to the correct position to allow the promulgation of the input data to the last subsystem. Each bit produces an FSK symbol of duration 64 samples in the Modulation Subsystem. This parameter can generally be adjusted from the panel of the blocks under the title 'Explicit sample period.'

3. Modulation Subsystem

The modulation subsystem, illustrated in Figure 8, uses each bit that appears at its entrance to choose between the two frequencies. This is accomplished by a multiplexer 'Mux,' where the selection pin (sel) is driven by the forwarded bits and the multiplexer data inputs are driven by two Direct Digital Synthesizers (DDSs). Each DDS generates a sine wave at one of the two frequencies for the BFSK signal. The DDS is a digital sinusoid generator and can produce frequencies up to half the frequency at which the DDS core will be clocked, i.e., the DDS clock rate, in order not to exceed the Nyquist frequency [22]. For the Xilinx Virtex-4, which can achieve clock speeds of 500 Mhz [23], the limit for the output frequency of the DDS is 250 Mhz when the DDS clock rate is set to the maximum possible frequency. Nevertheless, much lower frequencies were used and the frequencies for 1 and 0 are 45 MHz and 40 MHz, respectively. Given that the encoded bit rate of choice is $R = 1.5625 Mbps$, the two frequencies are not orthogonal based on the definition given in Section A in Chapter II. Even though this design choice may degrade the performance in a noisy environment, it does not have any noticeable impact in the noiseless analysis that follows. The 'Shift' block plays the role of amplification, multiplying the signal before transmission by a factor of four. Pulse shaping is not used in this design.

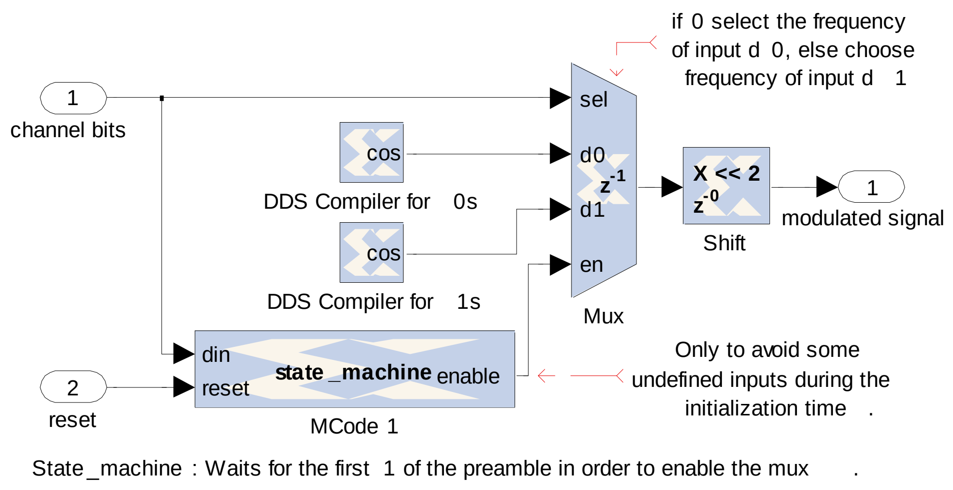

if 0 select the frequency of input d 0, else choose frequency of input d 1
1
channel bits
sel
cos
d0
DDS Compiler for 0s
z-1
d1
cos
en
DDS Compiler for 1s
X << 2
z-0
Shift
1
modulated signal
Mux
Only to avoid some undefined inputs during the initialization time .
din
reset
state _machine
enable
2
reset
MCode 1
State_machine : Waits for the first 1 of the preamble in order to enable the mux .

Figure 8 Modulation Subsystem.

The 'Mcode 1' in Figure 8 is used for initiation. During the beginning of the simulation, many signals inside the blocks start in undefined states and other blocks, like the multiplexer, cannot propagate these kinds of signals. A block that would enable the multiplexer after the propagation of the undefined signals was needed, without affecting the overall performance of the designs. Usually, a constant enable signal is used along with a delay measured exactly to overcome this problem. A very simple Matlab program was written that takes advantage of the fact that the first bit of the preamble is 1. Upon detection of the first 1 to the channel, the 'MCode 1' enables the multiplexer without any further interruption. It should be noticed that the command `xfix({xlBoolean},0)` was used in the program in order to avoid the use of a 'Convert' block. Otherwise, any value assigned as 0 or 1 in a Matlab Code is translated to an unsigned integer and cannot be used as it is to drive the enable port (en) of the 'Mux.' The xfix( ) command explicitly converts to the type described as the first argument. In this case, the value 0 is assigned as a Boolean type and not as an integer [13, p. 243].

B. RECEIVER

The non-coherent BFSK receiver is illustrated in Figure 9. The choice of a Non-Coherent (NC) receiver design was made to eliminate any need for an extra circuit that would extract the phase information from the received signal. The receiver consists of the following subsystems: the two Correlators, the Decision Circuit, the Timing Circuit, the two Non-Coherent Matched Filters and the Decoding Subsystem. The Correlators [24] and the Timing Circuit form the feedback path and the Non-Coherent Matched Filters and the Decoding Subsystem form the feed-forward path. The mixers are parts of both paths and are shown explicitly in the figure. The 'Relational' block compares the non-coherent matched filters' outputs and decides the value of the received bit. The circuit designed closely matches the theoretical diagram found in the Introduction of BFSK scheme in Figure 1 in Chapter II, with the addition of a time synchronization circuit and a Decode Subsystem.

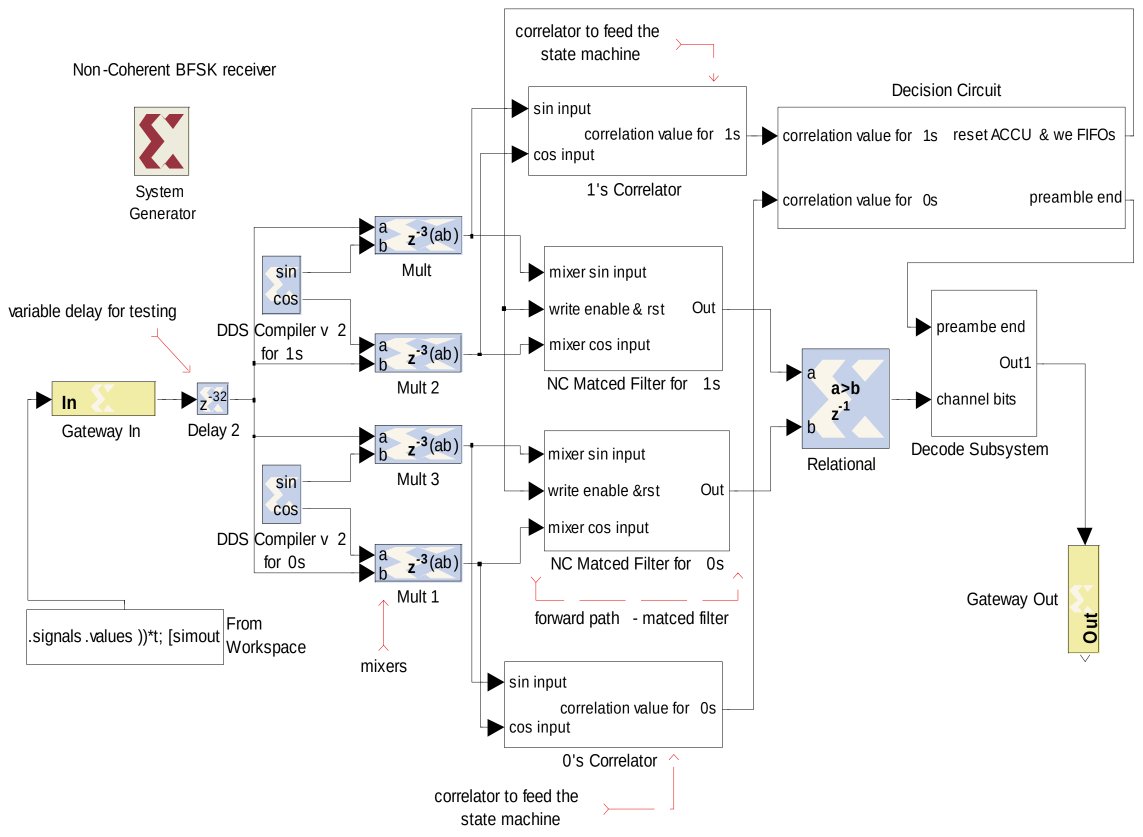


Figure 9 Receiver's schematic diagram designed in Simulink environment.

1. Non-Coherent Matched Filter Subsystem

A non-coherent matched filter is introduced in Section A in Chapter II. The implementation of this filter in the BFSK receiver includes an integrator that integrates the input signal over the duration of a bit period T_b. Thus, correct timing for the specific design means the correct identification of the beginning of each bit in order to integrate over the correct time frame. This fact generates the need for a timing feedback circuit that will make this information available.

The NC Matched Filter Subsystem in Figure 10 has two filters where each one consists of two branches. The two branches correspond to the sine and the cosine at the symbol frequency. Each branch consists of a mixer (illustrated in Figure 9 before NC Matched Filter Subsystem), an accumulator, and a squaring block. Then, the two branches' outputs are added together to give the final output of each filter. The output values of the two filters are compared in order to decide which frequency was transmitted. The frequency that was transmitted corresponds to the filter with the highest output value.

The accumulator included in the NC Matched Filter Subsystem is the followed discrete time equivalent of an integrator and it adds 64 consecutive values of the input signal before it is reset by the feedback timing circuit. Every accumulator is followed by a FIFO memory, which only reads the output of the accumulator just before the accumulator's reset signal is raised. In this way, the memory captures only the last value of the respective sum. The rest of the block is straight forward, with a squaring block and an adder that adds the signals of the two branches, yielding a single output from the subsystem. The downsample implemented in all branches between the FIFO memory and the squaring blocks is used in order to downgrade the unneeded computational load. After the accumulation of the correct 64 samples of a bit and the subsequent storage of this value to a FIFO memory element, the memory yields the same output for 64 consecutive time units. Thus, it is not necessary to do the computations for all values.

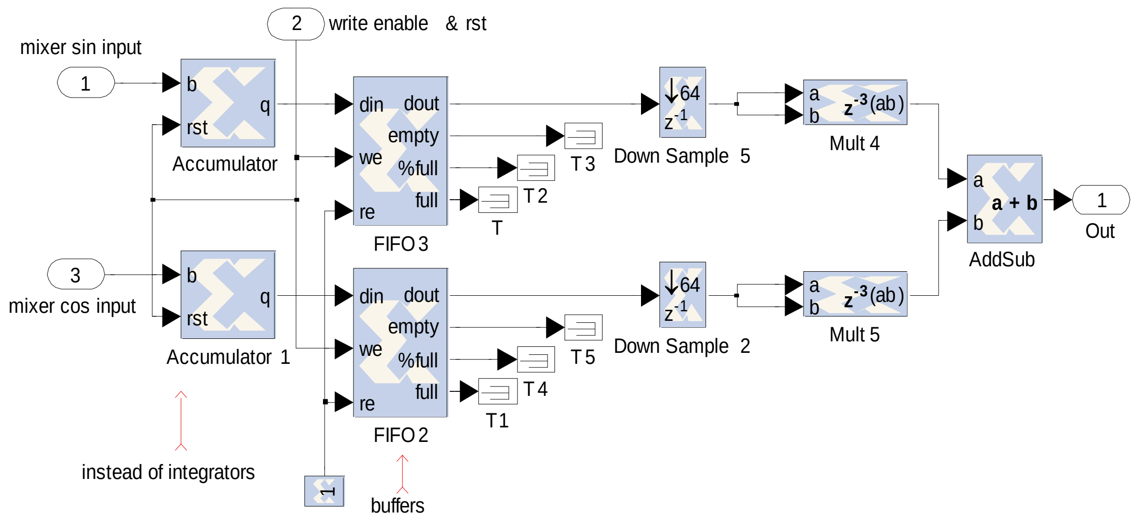


Figure 10 Non-Coherent Matched filter subsystem (one of two).

2. Timing Circuit

Synchronization circuits are categorized as data-aided and non data-aided (or blind) and the latter require no training data sequence [25]. As was mentioned previously in the transmitter description in Chapter IV, this design uses a data-aided circuit for the acquisition of the bit synchronization. The preamble is a known pattern that will help to identify not only the start time of each bit, but the commencement of each packet as well.

In this design, the feedback synchronization circuit is separated into three subsystems, the two Correlators and the Decision Circuit. The Correlators (Figure 11) work similarly to the NC Matched Filter Subsystem with the main difference being that accumulators have been replaced by Finite Impulse Response (FIR) filters. These filters constitute sliding window accumulators of the last 64 samples. In order to make a decision regarding the beginning and end of a bit, a circuit that updates its output at every received sample is needed. The correct timing is going to be extracted by the maxima and minima of this output. In contrast, the feed-forward path with the non-coherent matched filters need only accumulate the proper values and then yield a different output once every 64 samples and not every sample.

In Figure 11, the FIR is shown to be a custom FIR filter and not an off-the-shelf block provided by Xilinx. The reason is going to be analyzed in the troubleshooting section, but for the moment, it can be thought as an FIR filter with impulse response $h(n) = \sum_{n=0}^{63} \delta(n - n_0)$, where $\delta(n) = \begin{cases} 1 \; if \; n=0 \\ 0 \; if \; n \neq 0 \end{cases}$. The initialization block, as in the case of the block 'Mcode1' of the transmitter, is used only to prevent the undefined initial signals from propagating and to suppress errors during the simulation. It consists of a comparator that has two delayed versions of 1 in its inputs; thus, propagating an initial reset high signal once at the beginning of the simulation.

The difference of the outputs of the two correlators is the input to a logic block ('Mcode' block in Figure 12) that searches for maxima and minima of the input waveform. Given that the correlator yields a maximum when the correct 64 samples of a bit have been added, the expectation is that the 1's correlator will output a much higher value than the 0’s correlator when the whole first bit of the preamble has just been received. The opposite is expected at the second bit of the preamble, because it has the frequency corresponding to the 0 bit. Thus, the difference waveform is expected to be a maximum after receiving a 1 at the exact moment that all 64 samples of that 1 have entered the filter. Following the same reasoning, the difference waveform is expected to be a minimum after receiving a 0 at the exact moment that all 64 samples of that 0 have entered the filter. However, when two consecutive equal bits are received, the result is different. The output of the filter will reach an extremum at the moment that all the 64 samples of the first bit have entered the filter, and then remain at that extremum for the following 64 samples, corresponding to the second bit. Therefore, the filter output displays a plateau effect, which is less useful for symbol synchronization. After the identification of maxima and minima, a state machine tries to verify when the correct pattern of the preamble has been received. When this is the case, the timing of the bits is well known and this information is supplied to the accumulators of the NC Matched Filter Subsystem. This part is included in the Decision Circuit shown in Figure 12.

The timing is first extracted in absolute time values. That is, a free running counter, i.e., ‘Counter 3,’ starts counting from the moment the event starts working up to the moment it stops. When an event occurs, the time that is captured is relative to the power up time of the circuit. The information needed by the accumulators of the forward path is at what instance of a 64 cycle time they should stop accumulating the previous bit and start accumulating the new bit. The timing must be translated to time modulo 64 and then it is stored for the rest of the duration of the packet. This is done by the Slice and Register blocks in Figure 12. The extra delay introduced by the timing circuit during the feedback path must also be considered. This is performed by the ‘AddSub3’ and ‘Constant 7’ blocks in Figure 12. The exact value of ‘Constant 7’ was determined experimentally.

The last three bits of the preamble are not used by the state machine. The time that corresponds to the last two 0s is provided to the timing circuit in order to ensure a timely and accurate synchronization of the main circuit. Additionally, the very last bit of the preamble, the final 1, is used by the 'MCode' in Decoding Subsystem along with the signal 'preamble end' in order to identify the beginning of each packet.

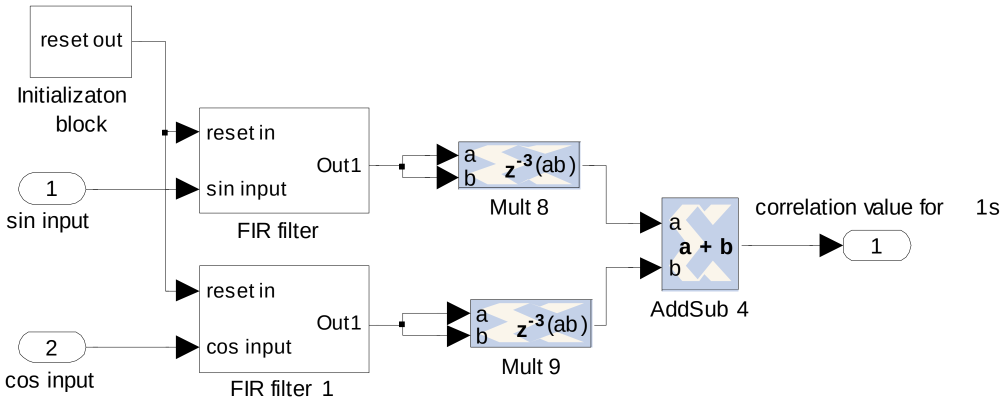
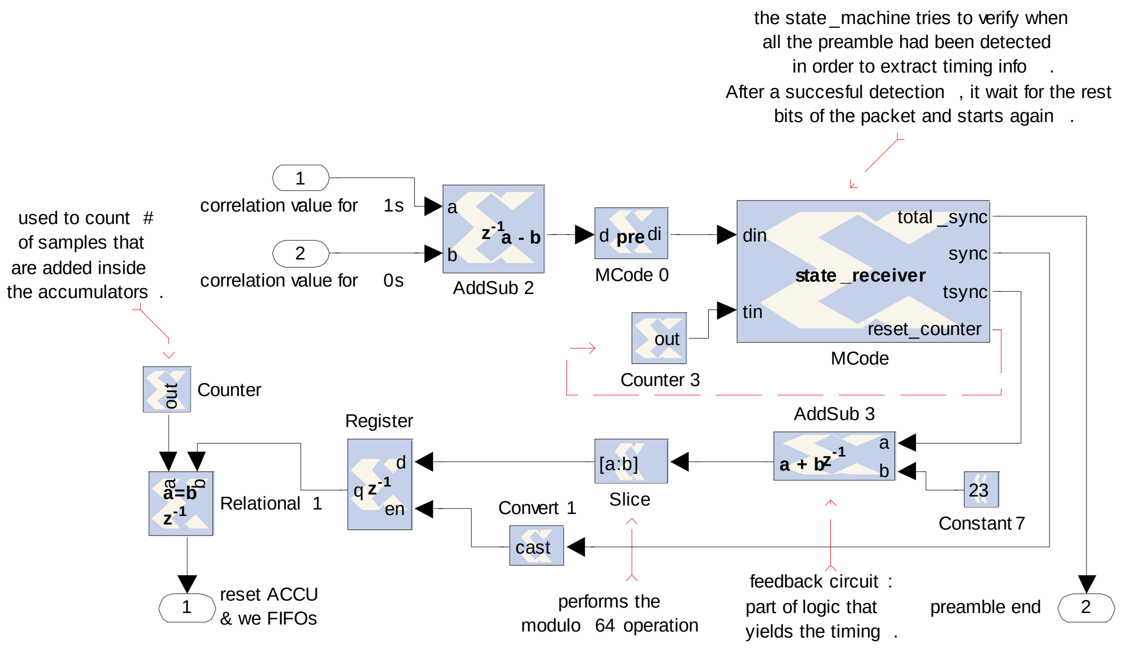


Figure 11 Correlator's Subsystem (one of two).



the state_machine tries to verify when all the preamble had been detected in order to extract timing info. After a succesful detection, it wait for the rest bits of the packet and starts again.
used to count # of samples that are added inside the accumulators.
1
correlation value for 1s
2
correlation value for 0s
a
b
z-1 a - b
AddSub 2
d pre di
MCode 0
din
tin
total_sync
sync
tsync
reset_counter
state_receiver
MCode
out
Counter 3
out
Counter
a=b
z-1
Relational 1
Register
d
q z-1
en
[a:b]
Slice
Convert 1
cast
AddSub 3
a + b z-1
a
b
23
Constant 7
1
reset ACCU & we FIFOs
performs the modulo 64 operation
feedback circuit: part of logic that yields the timing.
preamble end
2

Figure 12 Decision Circuit.

Upon reception of the fifth bit of the preamble, the state machine 'state_receiver' stays locked for the rest of the packet and then it starts searching for a new preamble after the time assigned for the current packet elapses. The decision circuit also provides an output, the 'preamble end signal' that helps the Decoding Subsystem to identify and block the preamble from the output, thus rebuilding the initial data sequence.

3. Decoding Subsystem

The Decoding Subsystem, illustrated in Figure 13, accepts as input the result of the comparison of the two non-coherent matched filter outputs, which is a sequence of 0s and 1s, and tries to locate the last preamble bit. The acquisition of the beginning of the preamble may or may not be correct, because the Decision Circuit had not yet finished the extraction of timing information. However, after the fifth bit of the preamble, the receiver is synchronized to the incoming signal. Thus, the Decoding Subsystem uses the information of the 'preamble end signal,' which is set when the acquisition of the fifth bit of the preamble is accomplished. The following two 0s, i.e., the sixth and seventh bits of the preamble, are sacrificed to assure the timely propagation of the information through the whole circuit and the last bit of the preamble is used to signal the commencement of the information bits. The Mcode block 'preamble_ detacher' is a simple state machine that incorporates the logic of the previous fact to allow the storage of input bits, only after the identification of the last bit of the preamble. The 'FIFO 4' memory is driven by a read enable signal. This enable signal is delayed by enough time to accommodate the total duration of the preambles that are taken away. This is accomplished by block 'Delay 4, in Figure 13.

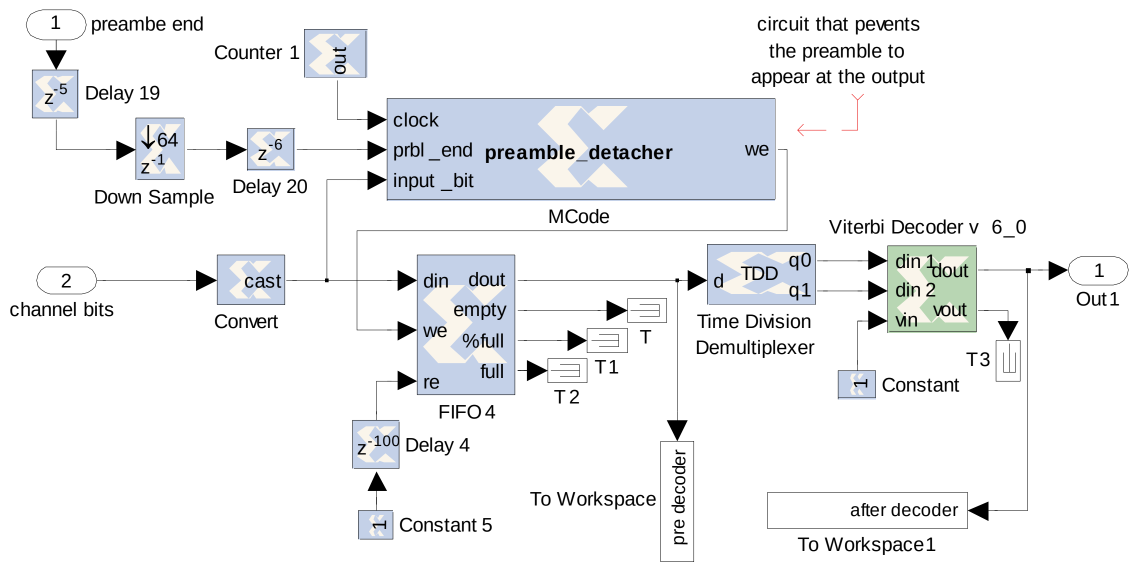


Figure 13 Decoding Subsystem.

Although the length of the packet had been taken into account by the Decision Circuit, the 'preamble_detacher' is counting the bits after the preamble again in order to achieve better synchronization. An external clock, i.e., the 'Counter 1' block in Figure 13, is used as a reference of the pulse clock time of the last preamble bit. After 120 clock cycles, the write enable (we) goes low, disabling the FIFO and the 'preamble_detacher' waits for the next 'preamble end' signal. The clock 'Counter 1' is an 18 bit register and is a free running counter. It should be noted that all signals in this subsystem are changing every bit period and not every sample period. Block 'Down Sample' in Figure 13 downsamples the 'preamble end' signal. Notice that the 'channel bits' signal has already been downsampled in the previous subsystem.

Concluding the description of Decoding Subsystem, the received bits are the input for a Time Division Demultiplexer (TDD) which is connected to a Viterbi decoder as discussed in Section B in Chapter II. The input sequence to the TDD is the encoded bit stream first produced in the convolutional encoder in the transmitter (Figure 7). The TDD is responsible for separating this sequence back to two different streams in order to supply the proper inputs to the 'Viterbi Decoder.' The parameters settings for the encoder and decoder are the same. This decoding produces the received message bits, which will ordinarily be identical to the sent message bits. Exceptions to this can be caused by decoding errors, which can occur when the received signal is corrupted by excessive noise, interference, or fading [26].

The low level description of the circuit that was discussed in this chapter is validated in the next chapter, along with the results and the weaknesses of the design. Furthermore, the lessons learned during the design process are also included as a deposit of knowledge for follow on research in this domain.

THIS PAGE INTENTIONALLY LEFT BLANK

V. DESIGN VALIDATIONS, RESULTS AND TROUBLESHOOTING

The design has been verified in the Simulink environment and a critical part of the design has been verified using Matlab code-based simulation. Many problems encountered during the design of the receiver and transmitter are also discussed. The figures of the full transmitter and receiver design are included in Appendix B, thus most of the figures that are mentioned in this chapter refer to this Appendix.

A. SYSTEM GENERATOR

In order to verify the design, the Simulink blocks 'From Workspace' and 'To Workspace' and 'Scope' were used. 'From Workspace' was used to supply the transmitter with message bits and to pass the transmitter output to the receiver. 'To Workspace' gives the ability to extract the values at a specific point in the design to the Matlab environment in order to drive simulation code with this data or to transfer it to the receiver. The 'Scope' depicts the data directly on a plot with the simulation time on the horizontal axis.

The input sequence to the transmitter was a random sequence of length 1900 message bits with the leader bit always 1 and the trail bit always 0. After being encoded, the bits were transmitted in packets of 128 encoded bits. The correct position of the preamble should first be determined. Figure 14 illustrates the signals of the block 'Scope' of Figure 26 in Appendix B. The first plot is the encoded bits, the second plot is the 'read_enable' signal that releases the encoded bits to the Modulation Subsystem, and the third plot represents the channel bits to be transmitted. Channel bits are defined as the encoded bit stream with the preambles. As seen in Figure 14, the preamble of the first packet is always positioned five bits ahead, but the rest of the preambles were correctly positioned exactly in front of the respective packet.

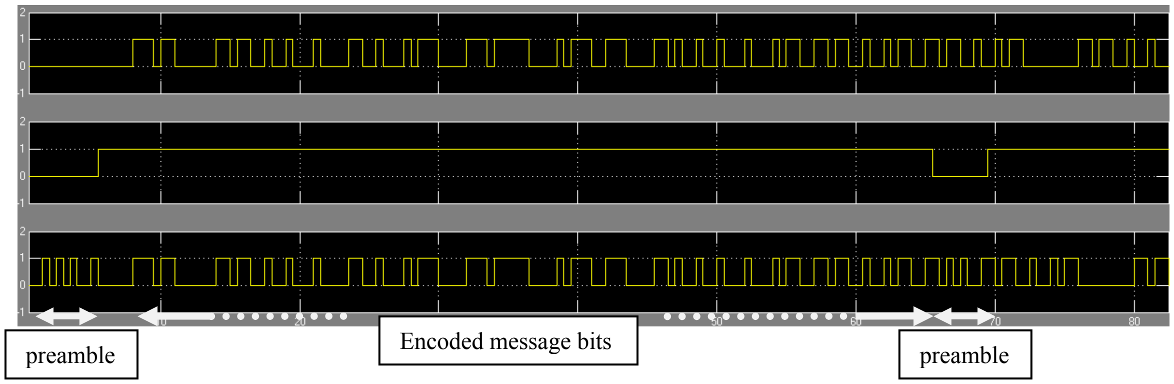


Figure 14 Plots of the encoded bits, the 'read_enable' and the channel bits (top to bottom).

As shown by the middle plot of Figure 14, the 'read_enable' goes high after the transmission of the preamble, allowing the propagation of the encoded bits (along with the known delay of the five bits duration). At the end of 128 bits, 'read_enable' goes back to low, capturing the encoded bits in the 'FIFO' memory. As stated before, the Preamble Subsystem does not know when there are no more bits for transmission and an enable low signal should be manually triggered. Otherwise, it would continue to transmit the preamble at the proper instances for the rest of the simulation time.

The receiver, on the other hand, is using the preamble for timing purposes and then removes it. The sequence without the preamble is the input to a Viterbi decoder that will regenerate the recovered message sequence. It is necessary to verify that the decision logic is working properly. Matlab code was written to simulate the decision circuit and duplicate the results for comparison purposes. In order to avoid any phase mismatch, the values after the mixers as shown in Figure 32 in Appendix B were captured in the Matlab Workspace by blocks: 'To Workspace 0-4' in order to supply the test code. This way, any noise inserted by DDS blocks will not influence the results. After the confirmation that both the Matlab code and the simulation under Sysgen were providing the same results, the Matlab code was modified to include the DDS blocks as well with the following results.

As seen in Figure 15, the first two plots are from 'Scope 3' (Figure 36 in Appendix B) and represent the decision waveform and the decisions made for the existence of the preamble. The vertical lines represent the detection of the part of the preamble, used for timing purposes, i.e., 10101. The last two plots in the white background are the respective results of the Matlab verification code. In practice, both results coincide with the correct position of the preamble. Recall that only the timing circuit uses the first preamble bits and the Decoding Subsystem uses the rest.

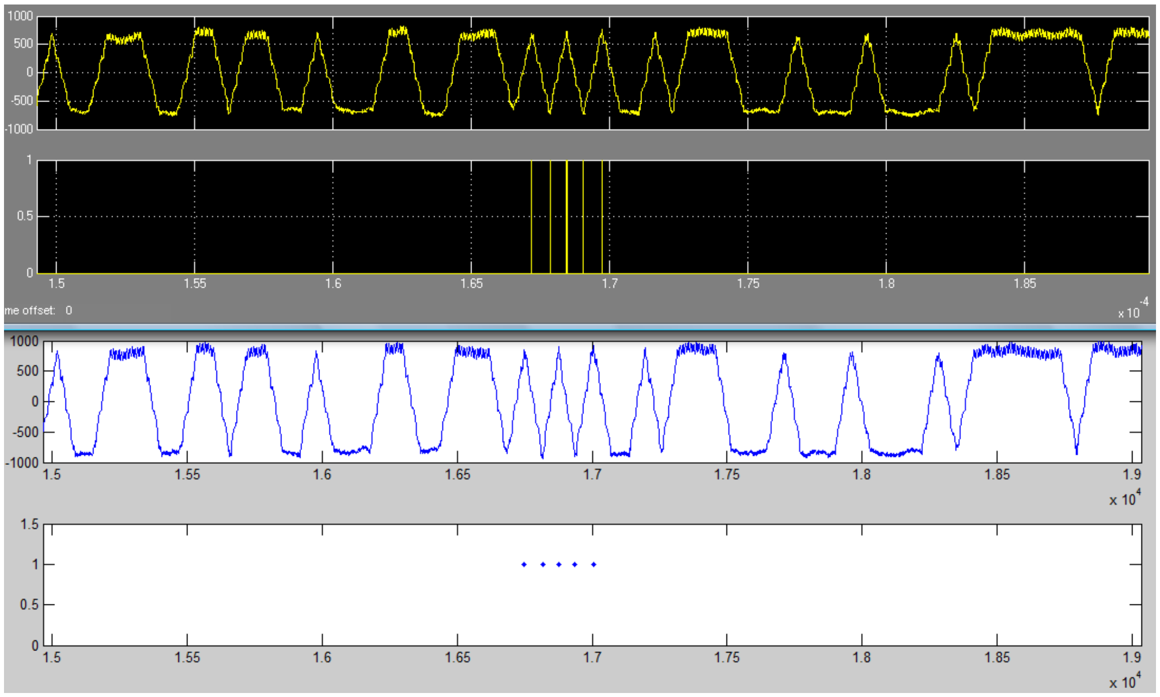


Figure 15 Results captured from the 'Scope 3' (Figure 36) in Simulink and results as plotted by the equivalent Matlab Code (from top to bottom).

Finally, the initially transmitted and the received bit streams must be compared. In order to verify the proper operation of the receiver, multiple runs were made using the same stream but different delay value ('Delay 2' in Figure 9) at the entrance of the receiver. This is done to ensure that the receiver is time invariant and that the specific choices made for the synchronization of the different subsystems are not case/input delay sensitive. After that process, different streams were used with random delays. The number of errors was calculated by a short Matlab code to confirm the visual verification.

The specific pattern used for the generation of the bit stream is a random sequence of nineteen hundred bits always starting with 1 and ending with 0. Specifically, the value inserted in the data slot of the parameter window ‘From Workspace’ (Figure 5) was [0:1901;1 a 0]' where ‘a’ was defined by the command a =rand(1,1900)<.5 in the Command Window. This command creates an array of 1900 random bits. The choice of that length was dictated by the constraints of the current configuration of the receiver. The counters used in different subsystems of the receiver are 18 bits wide. Since there are 64 samples per bit, there are

$$\frac{2^{18}\ clock\ cycles}{64\,\dfrac{clock\ cycles}{bit}} = 4096\ bits \tag{5.1}$$

theoretically possible. Due to delays, the effective limit is slightly under 4096 bits. Given that the encoding doubles the number of channel bits and accounting for the delays and the preamble, roughly nineteen hundred information bits can be received. This problem is further presented in Section C.2 of this chapter. Every data packet should fit 60 information bits, and there will be four extra bits in the last packet due to the convolutional encoder. The convolutional encoder does not encode the information bits of each channel packet separately, but the whole bit stream continuously.

The results are illustrated in Table 1 and Table 2. Recall that the input delay to the Receiver is ‘Delay2’ shown in Figure 9. The Matlab code shown in Figure 16 was used to align the output of the Transmitter and the Receiver and calculate the number of errors.

```
%% post encoding
num_test_bits =3820; % Defines the length of the encoded bits in the
                     % test sequence.
delay=403;  % the output value is delayed by 'Delay4' in Decode
            % Subsystem.
tra =after_Viterbi_encoder.signals.values(1:num_test_bits);
rec2 =pre_Viterbi_decoder.signals.values(delay:num_test_bits-1+delay);
num_of_errors_pre =sum(abs(tra-rec2)==1)
% position =find(abs(tra-rec2)==1)+delay-1
```

Figure 16 Matlab code to align output of Transmitter and Receiver and calculate number of errors.

It must be noted that the number of errors is calculated based on the encoded bit stream. There are two obvious methods that could be used to check for errors. We could count message bit errors or channel bit errors. In this work, it was chosen to count channel bit errors. This has the advantage of counting any error, since the received bits are examined before the decoder corrects any errors. However, the disadvantage of this method is that it does not check for errors in the decoder. Since the decoder is provided by Xilinx, we have confidence in its design and accept this disadvantage as small.

Before the simulation the following parameters should be defined in the Matlab Command Window: $T = 128 \cdot 10^{-7}$, $t = 10^{-7}$. New random sequences are generated by re-executing the command a =rand (1,1900)<.5 in the Command Window.

Run	Number of errors	Comments
1	0	First execution of command a =rand(1,1900)<.5
2	0	Second execution of command a =rand(1,1900)<.5
3	0	Third execution of command a =rand(1,1900)<.5
4	0	Fourth execution of command a =rand(1,1900)<.5

Table 1. Results of multiple runs with different input sequence and constant input delay (value set to zero).

Run	Input Delay Value	Number of errors
1	0	0
2	9	0
3	15	0
4	23	0
5	31	0
6	45	0
7	57	0
8	63	0
9	75	0
10	98	0

Table 2. Results of multiple runs with constant input sequence and variable input delay. All runs made after first execution of command a =rand (1,1900)<.5.

The simulation of both the transmitter and the receiver showed that the design is working correctly. The acquisition of the preambles was made at the correct times and the timing was correctly extracted. In the absence of noise no malfunction had been observed. Minor annoyances are presented in the next section, which explains the various problems that appeared during the design process.

B. TROUBLESHOOTING AND LESSONS LEARNED

Much knowledge was acquired during the design process. Even though Xilinx is trying to offer programs that are easy to use, there were many instances in which the debugging process was time consuming. Many of these points are illustrated as follows for easy reference and as a guide of things to avoid.

1. Transmitter

The preamble of the first packet does not fit right before the information bits. There are always five zeros between the end of the preamble and the beginning of the encoded message bits in this packet. In subsequent packets, the preamble is positioned correctly. This event should occur every time after a reset. The fact that these inserted bits are zero does not affect the decoding procedure and the action taken was to position a 1 in front of every bit stream. This 1 acts like a flag that information bits follow and made the debugging process easier as well. The initial assumption that a proper combination of the delay values in blocks ‘Delay 1,’ ‘Delay 2’ and ‘Delay 4’ in Figure 29 would correct the problem was later rejected. The reason may be the delay of the input data to reach the FIFO memory of Figure 30.

The Preamble Subsystem is not intelligent enough to sense when data is ready to be transmitted. It is in need of external manual control to enable the counter in the preamble. While the counter of the preamble is not enabled, the input data is convolutionally encoded and stored in the memory. In the same sense, the preamble circuit does not know when there is no more data to be transmitted. To void the transmission of packets that contain only the preamble, the information bits must be provided constantly to the receiver. The bits that cannot be transmitted at any given time are stored to an internal memory. Otherwise, the circuit can be manually controlled through the external enable and reset port. The output signal ‘empty’ from the FIFO memory of Figure 30 memory could be a good indication of when the input data is ready for transmission. This could help the problem with the preamble stated in the previous paragraph as well. Nevertheless, this signal does not seem to behave as expected. It goes high too early as shown in Figure 17 and does not go low after the transmission of the last bit stored in the FIFO memory as illustrated in Figure 18. The reason was not determined in this research.

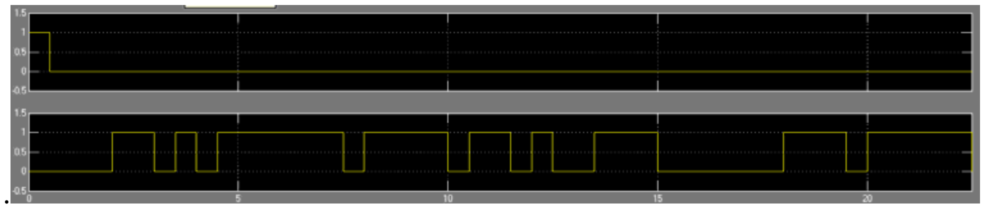


Figure 17 Plot of the ‘empty’ output signal and the ‘din’ input signal of the FIFO memory in the beginning of the simulation.

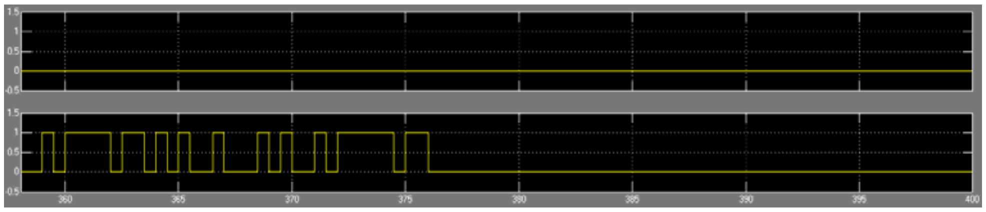


Figure 18 Plot of the ‘empty’ output signal and the ‘din’ input signal of the FIFO memory at the end of the simulation.

The state machine ‘MCode 1’ in Figure 32 could have been avoided and replaced by a constant with an output value of Boolean 1 connected to the enable port of ‘Mux’ through a delay. This delay should be measured to overcome the undefined signal errors. This tactic is used in Figure 33 for delaying the enable of the ‘Relational’ block.

2. Receiver

At the beginning, a point of concern was that the signal used to extract the timing, i.e., the output of the Correlators and specifically the signal at din of block ‘MCode’ in Figure 37, does not always provide clear and distinguishable peaks. The decision of where exactly the peaks occurred can become very vague and this would yield many errors. As an example, in Figure 19 a successful acquisition of the preamble is illustrated. The first, third and fourth detection of the preamble bits seem to be correctly positioned, which is not the case for the second and fifth. There is a flat area near the theoretical peaks, which in turn, inserts an error to the timing. From inspection of many preambles, the third and fourth bits of the preamble yielded the least error and these were initially used for decision, even though all bits were used for the state machine.

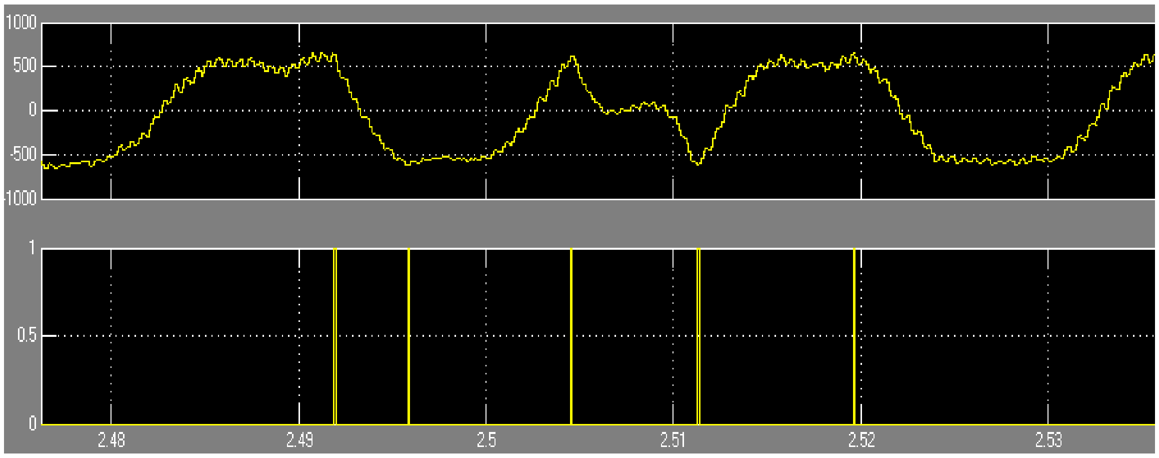


Figure 19 Example of a preamble acquisition with initial values for 'DDS clock rate.' Top plot shows the decision signal and bottom plot shows the successive peak identification made by the timing circuit.

When the verification Matlab code was changed in order not to use the output of the mixers of the System Generator design, it had been observed that the output of the DDS blocks used for mixing purposes in the Receiver, which are blocks 'DDS Compiler v2 for 1s' and ''DDS Compiler v2 for 0s in Figure 33, did not yield the proper output signal. The change of the parameter 'DDS clock rate' that seems irrelevant was changed for both the transmitter and the receiver in order to correct the output. The initial value of 500.0 (MHz) was changed to 100.0 (MHz). In this way the output waveforms of the DDS blocks closely matched those predicted by Matlab simulation. The improvement to the signal used to extract the timing was dramatic and many synchronization problems had been solved as indicated in Figure 20. The Xilinx technical support replied that the 'DDS clock rate' should match the 'FPGA clock period' defined in the Sysgen token. Thus, because 'FPGA clock period' is 10ns, 'DDS clock rate' should be $\frac{1}{10ns} = 100MHz$. This answer is not very convincing given that there is no reason to be able to define a parameter that you must calculate uniquely from another parameter already defined. Given that the 'DDS clock rate' defines also the upper limit of the output frequency as explained in Section A.3 in Chapter IV, things are becoming more complicated. Further research of the source of the problem must be made.

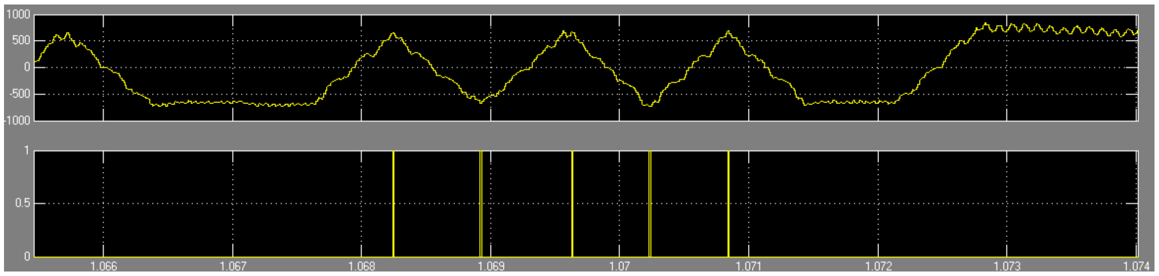


Figure 20 Example of a preamble acquisition with final values for 'DDS clock rate.' Top plot shows the decision signal and bottom plot shows the successive peak identification made by the timing circuit.

The synchronization circuit of the receiver had not been designed to be insensitive to noise; except for some very limited capabilities that the 'MCode0' block in Figure 37 can yield. 'MCode0' was inserted to the design to implement the idea of a threshold that a signal should exceed in order to be translated in/mapped to 0 or 1. The main problem without that threshold was that even a small amount of noise to the channel would make the decision circuit believe that there are transmitted bits and eventually it would match the preamble to the random noise. Averaging the timing extracted over more than one bits of the preamble would also give better immunity to noise. Then, the convolutional encoder could correct the few mistakes made.

Another drawback of the program included in the 'MCode' block of Figure 37, which is the main decision logic, is that there is no escape from going sequentially through all the states. Once entered, it only searches to accomplish the criteria to enter the next state up the last one. A maximum stay time at each state should have been given after which, the state machine would start over. This is also a way to compensate for noisy reception.

The counters of the receiver, i.e., 'Counter 3' in Figure 37 and 'Counter 1' in Figure 38, are free running counters, which means that they never reset to restart counting; they are only limited by the assigned output precision. The MCode block uses these counters to record the time of some incidents. In case that the counter is reset at an improper time, the relative timing of the incidents is destroyed. For example, 'MCode' in Figure 38, counts 120 bit periods after the reception of the full preamble in order to search for a new preamble. If the counter resets during this counting, then the block will stay in the waiting state for an unknown time. On the other hand, the counters will eventually reset after exceeding the maximum assigned width of their output. For this reason, a reset to this counter is needed but must be built in such a way that will not affect the timing of the following MCode blocks. An attempt to solve that problem in 'Counter 3' of Figure 37 using the signal 'reset_counter' was only partially successful and was disconnected.

The real function of these counters should be further analyzed. A state machine that follows the flow diagram uses 'Counter 3' (Figure 37) and 'Counter 1' (Figure 38). The counters serve to ensure that the exact time after the reception of the preamble had past and a new search for preamble should be made. In detail, the 'MCode' of Decision Circuit (Figure 37) detects up to the fifth bit of the preamble and then waits for 7872 $\left(123\text{bits}\bullet 64\,\frac{\text{samples}}{\text{bit}}\right)$ clock counts until the next detection. On the other hand, 'MCode' of Decoding Subsystem that detects up to the eight bit of the preamble, must count 120 clock counts (the counter now works in bit period because the signal had already been downsampled by 64) until the next detection.

The FIR filters in the NC Matched Filter subsystem as illustrated in Figure 34, were initially implemented by the respective FIR Xilinx blocks. The problem that appeared was that Sysgen was always mapping these blocks to DSP48 cells (see the section on DSP-Enhanced FPGAs in Appendix A). The number of DSP48 cells needed for the four 64-coefficient filters is 128 cells, where only 48 are present in the chip. In order to avoid this problem and given that all the coefficient were unity for that case, a custom design was made as shown in Figure 21. In this way, not only the demand for DSP48 cells were minimized but the demand for general blocks was lower as well. The 64 delay block is a pipe of 64 flip flops in a row. The 'new value' is the value that enters the pipe and the 'old value' is the one that exits the pipe. The combination of 'AddSub' and 'Accumulator 1' blocks is responsible for accumulating every new value to the current sum while subtracting the value that is 64 periods old. In such a way, the accumulator always contains the 64 most recent values. The accumulation of garbage over time in the ‘Accumulator 1’ is possible. This hypothesis has not been confirmed during the simulations, but is still a concern for the real circuit on the FPGA. A reset of the accumulator when the message part of the packet is under reception would ensure that no garbage is left in the accumulator.

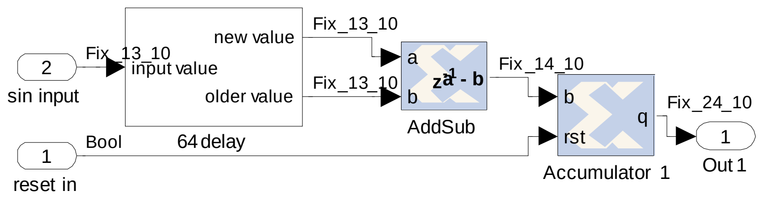


Figure 21 FIR custom block.

The ‘Viterbi Decoder v 6_0’ in the Decoding Subsystem included in Figure 38 appears in green, which means that an extra license must be granted by Xilinx. In this case, a 90 days free license, which is offered to anyone through the Xilinx website, was acquired in order to verify the functionality of the design. In any case, the verification of the circuit was made before the Viterbi Decoder block because the exact number of errors should be revealed and not be covered by corrections made by the decoder.

The verification of the design was illustrated in this chapter. Many of the problems encountered were also exposed and useful lessons learned were also described. This part concludes the discussion about the design. The next chapter summarizes the work that has been done in this thesis and proposes possible expansions and follow on work that can be made.

VI. CONCLUSIONS

A. SUMMARY OF THE WORK

The concepts of Software Defined Radio (SDR) and Binary Frequency Shift Keying (BFSK) modulation were introduced and the application of Field Programmable Gate Arrays (FPGAs) to SDR was further examined. The capabilities offered by the FPGAs to easily transform a design to circuit were used to build a BFSK transceiver.

Xilinx System Generator was used to design a data aided BFSK transmitter and receiver. Extensive simulation assures their proper function. Matlab code was used to verify the results taken by the simulation. The designs were finally placed and routed to a Virtex-4 FPGA to ensure that no errors occurred during that process.

Appendix A includes an introduction to FPGAs, their internal structure and their utilization in the SDR concept. Extensive descriptions of all the blocks used and the parameters assigned to each block are given in Appendix B. This facilitates the reproduction of the design and gives a better understanding of how System Generator is working. In Appendix C, the Matlab code used to simulate the decision signal is given. The reproduction of the whole circuit is not as important as the reproduction of the decision signal, because this is the most crucial parameter in the whole design.

B. SIGNIFICANT RESULTS

The concept of Software Defined Radio proved to be fully realizable and both the designs of the transmitter and receiver do not exceed in total the capacity of a moderate FPGA, after being placed and routed. The device utilization summary for the transmitter and receiver is shown in Table 3. The amount of work needed to design a fully functional transceiver was mostly consumed in the learning of the System Generator blocks and ISE suite, which is an overhead not needed for follow-on designs.

Design Type / Number of	Transmitter	Receiver	Total
DSP48s (total: 48)	0 (0%)	4 (8%)	4 (8%)
Block RAMs (total: 72)	4 (5%)	10 (14%)	14 (19%)
Slices (total:10752)	177 (2%)	3616 (34%)	3793 (35%)

Table 3. Device utilization summary.

The simulation proved that the receiver works as expected and when noise is not present, no errors were generated by the receiver. The preamble was correctly positioned in front of each packet, with the exception of the first one as discussed in Section C.1, Chapter V. The preamble helps the receiver to identify the beginning of every packet and extract timing information. The receiver then removes the preamble and decodes the received sequence. Any few errors made by noise are corrected by the Viterbi decoder. No timing errors were identified during the place and route process made by the ISE software.

C. SUGGESTIONS FOR FUTURE WORK

1. Limitation of the Design

The timing circuit is not built to be very tolerant to noise. The main reason for the susceptibility to noise is that the state machine (MCode in Figure 37) incorporated in the receiver, does not include an abort condition in case there is a misinterpretation of the decision signal. Specifically, if the noise exceeds the threshold chosen at the specific period that the circuit tries to identify the existence of the first bit of the preamble, the state machine is obliged to enter the next stage and does not abort until it goes through all the states sequentially. Then it must wait a packet period until it search again for a new preamble sequence.

The frequencies used are not orthogonal according to the definition given in Section A in Chapter II. This was due to the initial implications described in Section B.2 in Chapter V. In order for the two frequencies to become orthogonal, they must differ by integral multiples of the channel bit rate $R_{c.b.} = \frac{1}{64}10^8$ Mbps. In this case, the frequency separation should be multiples of 1.5625 MHz. For example, if $f_1 = 40MHz$, the other frequency could be $f_2 = 43.125MHz$.

The limitation of the free running counters must be also addressed. As explained in Section B.2 of Chapter V, the current configuration only guarantees the reception of less than two thousand message bits, before the counters are reset and a critical error may occur. The extension of the free-running counters from 18 bits to a higher number would only give some more space, without eventually solving the problem. A reset signal should be inserted at a proper time that will not affect the rest of the design.

Pulse shaping is not used in the design, but would likely help to suppress Inter Symbol Interference [27, pp. 233-244], provided it was done in a way that preserves orthogonality. The realization of filters in System Generator is made easy by the use of Xilinx 'FDATool' and 'FIR Compiler' blocks. 'FDATool' interfaces the Simulink Signal Processing Toolbox to offer a graphical interface to design digital filters. 'FIR Compiler' can be also used alone in case the coefficients are precalulated.

2. Suggestions

The design presented is a good starting point for a design including extra features, such as noise tolerance and pulse shaping. These features are optional but will make this implementation more useful in practice. Then the design should be more exhaustively verified after transfer to the FPGA to assure proper timing of the components. The simulation under System Generation is cycle and bit accurate [12]. In this sense, even if no timing errors were created after place and routing and the simulation under System Generator verified the proper function of the circuit, further verification of the design after implementation is compulsory. Chip Scope Pro is a useful way to test the design after download to the target FPGA. In order to do so, pins to 'Gateway In' and 'Gateway Out' blocks must be assigned. These pins must be the ADC input to and the DAC output from the board.

The DDS clock rate discussed in Section B.2, Chapter V, must be further examined and the connection of the parameter to the DDS block must be further examined. Even if in this design the output frequency of the block is correct, there is no guarantee that it will work as well under different design parameters. Even the extensive documentation included in the online support page [14] does not give clear answers about the relation between the DDS clock rate, the output frequency of the DDS block, and the FPGA clock period in the Sysgen token.

Except the problem with the DDS block, there are other less significant design errors to be solved. The delay after the first preamble, as presented in Section B.1, Chapter V, should be further examined. The cause of this delay is currently unknown and could not be solved with change of the delays' values included in the design. In the same section, the inability to detect incoming message bits is also discussed. This may be a severe limitation in real-life implementations.

After proper verification of the design, thorough tests under different levels of noise could be done. Bit error rate and Signal to Noise Ratio can be plotted and compared to the theoretical performance of a non-coherent BFSK receiver. In this way this design will be fully documented.

This design can be used as a foundation for designs using more complex modulation schemes. The extension to M-Frequency Shift Keying (MFSK) is likely straightforward and modification to Binary Phase Shift Keying (BPSK) should be easy, although it would require carrier frequency and phase synchronization [27, pp. 270, 295]. In this manner, a database of modulation schemes can be created leading closer to the ultimate goal of a multimode SDR transceiver.

The electronic files that contain this design are on file with the manager of the Cryptologic Research Laboratory [28]. Helpful support documentation from Xilinx is discussed in Section C, Chapter III.

APPENDIX A. BACKGROUND ON FPGA AND TECHNOLOGIC BACKGROUND

This Appendix introduces the internal structure of Field Programmable Gate Arrays, their function and how they are implemented in the Software Defined Radio concept. An evaluation of different technological options in implementing communication modulating techniques and Software Defined Radio follows. A comparison between these options is also included.

A. BRIEF DESCRIPTION OF AN FPGA

Xilinx Inc. invented the FPGA in 1984 [29]. As electronic circuits were becoming more advanced, the glue logic [30] was getting more complex and improvements to the Complex Programmable Logic Devices (CPLD) were needed to handle more demanding applications. FPGAs came as a logical advancement to help interconnect large integrated circuits providing more printed logic and incorporating more gates.

Initial manufacturing technologies of FPGA included antifuse, Static Random Address Memory (SRAM), Electrically Erasable Programmable Read-Only Memory (EEPROM) and some minor types [31]. Their difference is that SRAM requires external boot devices but it is reprogrammable, antifuse is one time programmable and EEPROM is reprogrammable and does not need an external boot device. Nowadays, the main type used is SRAM, except when reprogramability is not a mandatory feature, in which case, antifuse is a cheaper solution.

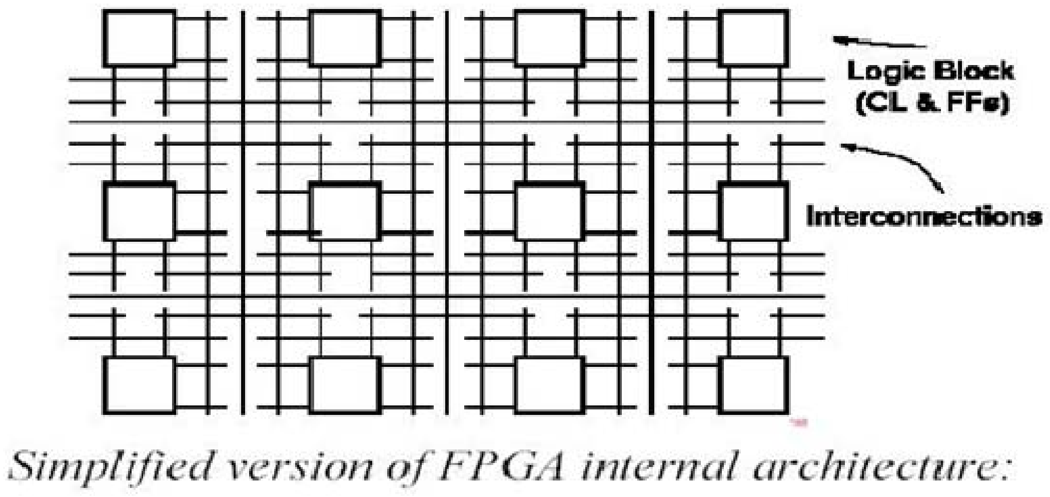


Figure 22 Simplified Version of FPGA Internal Architecture (From: [32]).

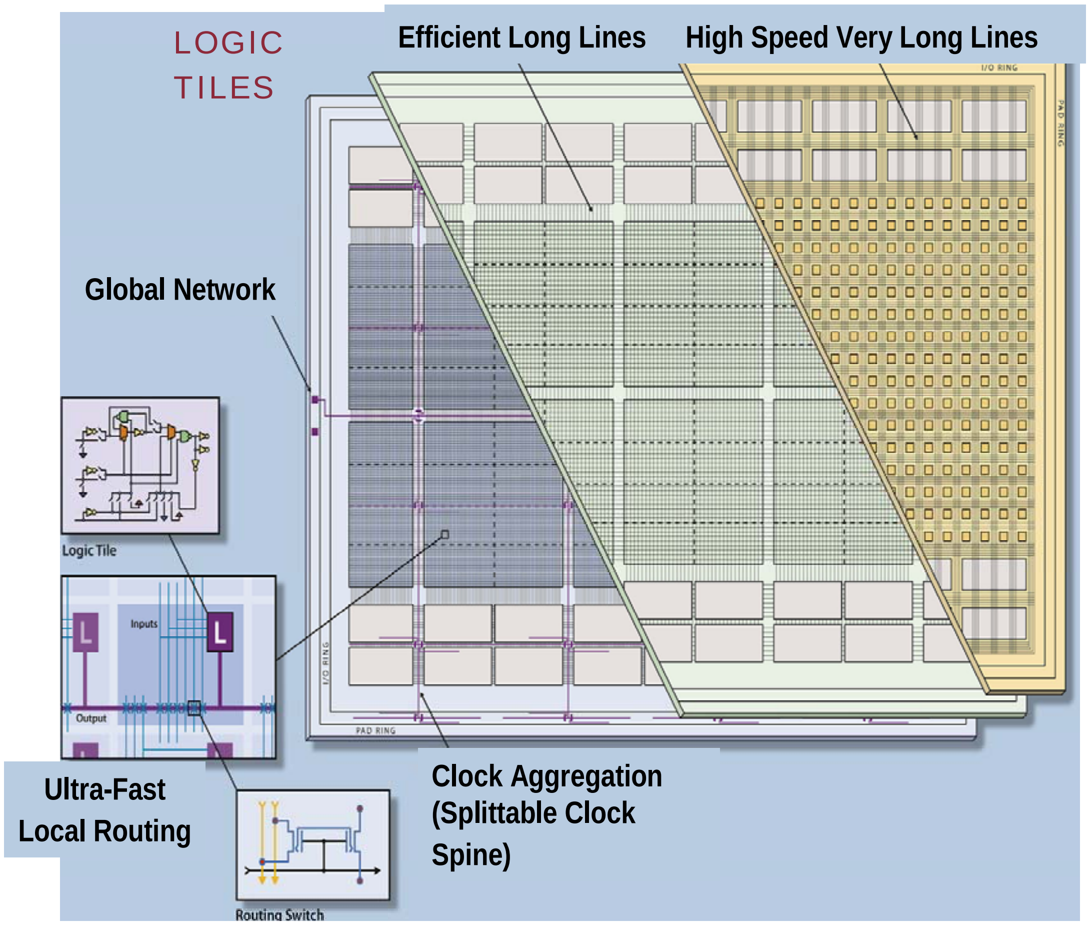


Figure 23 Typical FPGA architecture (From: [33]).

A Field Programmable Gate Array is a two dimensional array of logic blocks and flip-flops with electrically programmable interconnections between them [32]. These interconnections can be identified in Figures 22 and 23 [33] and are distinguished in local (or short) and long routing lines. Logic Tiles, also called Logic Slices according to other manufacturers, are the smallest blocks of logic. Due to its versatile structure, a different primitive operation (addition, multiplication, etc.) can be assigned to each Logic Tile. In order to build more complex functions, many Logic Tiles are attached to an adaptive network. Electrically programmable switches (as shown in Figure 23 under the subtitle Routing Switch) are responsible for customizing the network.

From the aforementioned description, it is possible to identify the two configurable aspects of FPGAs:

- The function assigned to each logic block. This function is going to define which elements inside the logic block will be activated in order to yield this specific function. The logic block itself must have a structure that may support a wide variety of different logical functions. One such structure of great importance is that of a Lookup table.
- The interconnections between the logic blocks. The combination of many primitive functions assigned to different blocks can give a very complex functionality as a result. Due to the way that this function is implemented, it may even be executed faster than when a microprocessor is used instead. Nevertheless, it should be kept in mind that this flexible routing adds much overhead to the chip itself. It consists of a wiring grid controlled by electrically programmable switches and the interconnection overhead can even be close to two thirds in terms of power consumption and silicon in deep submicron processes [34]. The higher the flexibility of routing in an FPGA, the higher the utilization of the logic and the lower the density of logic block. The manufacturers of FPGAs should always consider this is a tradeoff for their products.

In Application Specific Integrated Circuits (ASICs), no need exists for logic tiles, nor for long routing lines, because the operation of its logical components is prespecified and sequential logical blocks are placed during manufacturing process closer to each other. This makes the design much more concentrated and more efficient but it lacks the main characteristic of FPGAs, versatility and upgradability [76].

A detailed description on FPGAs is given in [35], which can be used for further reference.

B. ADVANTAGES AND APPLICATIONS OF FPGAS

Until fairly recently, FPGAs did not have enough gate capacity or computational power to implement digital signal processing (DSP) tasks. They have also been perceived as being expensive and power hungry. The versatility and the extra capabilities that they acquired after the change of the century did change many of their applications. One of their newest features is the introduction of new hard embedded multipliers, which yield extra DSP capabilities. A detail description of the embedded multipliers is given later in this chapter.

The synthesis and development tools have also evolved and include many different design environments. Except for the Hardware Description Languages, every FPGAs manufacturer offers a proprietary designing suite consisting of block diagram designing tools or schematic processors. These tools are time and signal accurate and their ease of use minimizes the learning curve and the debugging process, which in turn leads to short time to market. Intellectual Property (IP) cores are also available, which are designs implementing complex functions that can be incorporated into other designs for a fee. Usually, these are also available from third party vendors and their acquisition accelerates the time to market and reduces the need for proficiency in designing FPGAs [36].

An example of great interest is mobile communications. The newer CDMA2000 EVDO and W-CDMA standards demand computationally intensive digital signal processing, which requires much power. The ASIC solution is always better suited in such an environment, but the lack of upgradability makes it undesirable. The standards do not last for more than four to five years, making the replacement of the hardware at a base station uneconomical. The FPGA is the solution that closely matches the effectiveness of ASICs retaining at the same time the ability to upgrade [37].

Another important feature of FPGAs is their ability to conform to today's general trend towards system-on-chip (SoC), thus making the integration of many different circuits in one single chip a reality. This saves both money and space, offering high bandwidth between devices. FPGAs are used for many diverse applications including ASIC prototyping, digital signal processing, medical equipment military systems, Artificial Intelligence (AI) and cryptography [38].

The characteristics explained in detail above yield designs implemented in FPGAs with short time to market and reduced cost of development. Also, the maintenance and upgrade cost can be minimized, if this is applicable to the specific application.

C. FPGA VS. GPP

Until recently, the fabrication, or engraving, process for General Purpose Processors (GPPs) were one generation ahead of the engraving process for FPGAs. Nowadays, this is no longer true. The current generation of Intel's Penryn® processors uses 45nm lithography for engraving and both the MIPS32 74K and ARM Cortex®-A9 utilize a TSMC 65nm generic process [39]. Today, the manufacturing process of FPGAs has closely matched that of GPP with Xilinx's latest FPGA series Virtex®-5 [40] and Altera's Stratix® III [41] using a 65nm engraving process. Altera also recently released the 40nm Altera's Stratix® IV series [42].

The benefit of switching to a smaller manufacturing technique is to shrink the die, even though there may still be more circuits packed. Therefore, the question is how efficiently the extra gates can be used. In the past, this extra silicon was used first for deeper pipelines with more complex prediction circuits, then for more on-chip cache memory, and finally to add more cores. A multicore processor is a single chip containing multiple processing engines that may share common resources, such as cache memory. Each of these techniques ended at a point of diminishing returns. The only hope is that the addition of more cores could yield the extra computational power needed, but efficient multicore programming is a challenge. The traditional programming techniques could not be used to take advantage of the multicore GPPs and more time will be needed in order to have mature parallel programming.

FPGAs can use the extra silicon in a more application-specific way. It is easier to build a circuit that uses parallelism than to write a similar program in software. Thus, while technology advances and offers higher density chips, it is always possible to make use of this increased amount of logic in FPGAs. This is not the case in GPPs [43]. The first implementations of SDR were using mostly GPP [44], but the evolution of DSP made them less practical. Nowadays, the GPP is extensively used for another purpose; it is responsible for the network protocols and any application used. The new generation of FPGAs is also offering IP cores of soft processors, eliminating in many cases, the need of an extra GPP. Following this philosophy, ACTEL offers the Cortex-M1 [45] and ARM 7 and Xilinx offers the PowerPC 440 [46] soft processors.

Although ARM processors seem to dominate the GPP market for portable devises, the STI's (Sony, Toshiba, IBM) Cell processor and other completely new products launched this year also exist but they have not been given time to prove their capabilities. Some of these products include Intel's Atom^{TM} [47] and Via's Nano^{TM} processors [48].

D. FPGA VS. DSP

DSP chips provide good performance and usually offer an easier development process, which also means quicker time to market. Some modern DSP chips are very capable and they sometimes feature on-chip Viterbi and matrix multiplier coprocessors and a plethora of connectivity and memory options [56]. The first in line is Texas Instrument$^{®}$ TMS320C6455, which has a 1.2 GHz clock and is engraved with 90nm process technology and executes up to 9600 million instructions per second (MIPS) [49]. Another high end chip is Freescale's MSC8144 multicore DSP [50], which can be accompanied by the MSBA8100, an accelerator for Fourier transforms and channel decoding, which is especially made to accelerate 3G-LTE, WiMAX and 3GPP-R6. Each of the four cores of MSC8144 runs at 1 Ghz and was the best performer among DSP chips on some tests made by Berkeley Design Technology [51].

High power consumption is another drawback of DSP devices. Mobility is of much concern nowadays including portable wireless devices, and in the future, this demand will likely increase. Many Bluetooth and WiFi products exist and the demand for mobile communication will grow, requiring even more efficient DSP techniques. Much research has shown that DSP chips consume much more energy than ASICs or even

FPGAs. In “BDTi Focus Report: FPGAs for DSP, Second Edition,” [52] there is an extensive analysis and comparison of the consumption of different kinds of chips, leading to the conclusion that the parallelism inherent in FPGAs can save much energy compared to the same number of DSP cores. As stated in “FPGAs vs. DSPs: A look at the unanswered questions” which is an abstract of the BDTi report, it is mentioned that in DSPs only a small fraction of the silicon real estate is devoted to the actual calculations while most is assigned to the transportation of data to the correct place. Therefore, they conclude that “it would be a mistake to assume that FPGAs are inherently less energy efficient than DSPs”. Then, an example exemplifies that even though the raw power consumption of a FPGA is much higher than a comparable DSP, the FPGA can handle many more channels per chip, leading to only a fraction of the power consumption per channel of the DSP.

DSP performance cannot easily compete with either ASICs or FPGAs and the main reason is that DSP chips are serial processors, even if many of the DSP applications can widely benefit from the inherent parallel structure of both ASICs and FPGAs.

According to Douang Phanthavong [53], FPGAs that have been optimized to perform a digital-signal processing task, will run anywhere from 10 to more than 1000 times faster than a stand-alone DSP device. This is the main reason that modern DSPs include special coprocessors. Especially for communication applications, Viterbi and Turbo code coprocessors have been developed, suppressing the need of using multiple DSPs [37]. However, not all needs can be satisfied by special coprocessors and the unlimited customization that FPGAs offer can match the needs in a more favorable way [53].

An example from “Embedding FPGAs in DSP-driven Software Defined Radio applications” by Rodger Hosking and Richard Kuenzler examines the case of a wideband Finite Impulse Response (FIR) digital filter. Assuming that this filter requires 32 Multiply ACcumulate (MAC) operations in every clock cycle, it is easy to incorporate 32 MACs in an FPGA design, which are hardwired, yielding greater speed. In contrast, DSPs usually incorporate only two multipliers and will be considerably slower. Notice that a hardware MAC can be clocked up to 500 MHz [54].

E. FPGA VS. ASIC

ASICs are hardwired [32], custom chips designed for a specific application instead of working as a GPP. They are hard to compete with any other type because they encompass all the most wanted characteristics. At the same time, ASICs can achieve energy efficiency, low cost and high performance. They have only one drawback. They ask for all the design effort and most of the expenditure to be made upfront and no changes can be made without paying again all this costs.

ASICs emerged in the place of DSPs offering better performance, power and cost compared to the latter, because they could use the silicon estate more efficiently. In ASICs, only the compulsory interconnections and the exact number of logic cells exist. Thus, in high volume the price per unit is definitely cheaper than any other chip [55]. As stated, the DSPs do have a fixed cost regardless of the purchased quantity. In addition, FPGAs cannot use their silicon as efficiently because of the interconnection overhead. However, this aspect only accounts for one side of the coin. In order to produce ASICs, it is necessary to first print the corresponding masks. This cost is included in the nonrecurring engineering (NRE) costs and make the production of small quantities prohibitive [56].

Having a perfectly matched ASIC to a specific application does not always solve all the problems. Even if this approach is guaranteed to achieve the maximum speed along with minimum resource consumption, it demands much time for the initial design, which increases exponentially with its complexity. Nowadays, with very short products' life, even a delay of some weeks may force a product to lose the market window with catastrophic results in the sale sector.

Upgradability and reprogramability are additional characteristics missing in ASICs. Opposed to FPGAs, ASICs must be designed and manufactured to exactly the specifications imposed. If not, most of the NRE must be paid again plus any extra expenses to retire the defective products from the market. This process is expensive and undesirable. On the other hand, FPGAs can be even shipped with bugs and then be corrected by a simple download (if correct measurements are taken for that purpose). For DSPs, which work based on software, reprogramability is also viable.

In conclusion, it is becoming harder and harder to find devices that have the luxury of being time to market insensitive and high volume cost effective. In addition, even in that case, there is a place in the market for FPGAs to equip the first versions of a new product, until the design is proven to be robust. FPGAs do not outperform ASICs neither in terms of speed nor in power consumption. Nevertheless, this margin is not as significant as that between DSPs and FPGAs.

An optimized implementation in FPGAs can be almost as good as one in ASICs, additionally offering the ability for future upgrades and the flexibility of a System on a Chip. This flexibility is acquired at the expense of price per unit. Accounting that FPGAs do not demand significant NRE costs, there is a place for them in the market. It is hard to approximate the quantity that is the turning point to the curve. As the engraving process shrinks, the expenses associated with the manufacturing of fabrication units for ASICs goes up. In order to keep the manufacturing cost of ASICs low, they should be engraved using larger scale making the comparison between ASICs and FPGAs even vaguer [57].

F. DSP-ENHANCE D FPGAS

It has been seen that each category of chips has its own virtues and shortcomings. In order to increase capabilities, companies have tried to combine features of different classes in one chip. Following this logic, new FPGA models have embedded DSP cells and the companies have created synthesizable Intellectual Property (IP) cores to accompany their chips [56].

Regarding DSP embedded capabilities, both Altera's Stratix I family and Xilinx's Virtex-II family already offer some architectural enhancements to increase DSP efficiency. These DSP capabilities were provided by hard-wired on-chip multipliers intended to offer acceleration to operations like multiply-accumulate (MAC) or multiply-addition (MADD), which is very common in DSP algorithms like the Fast Fourier Transform (FFT) and Finite Impulse Response (FIR) filters. The core of a typical DSP block consists of a multiplier followed by an adder and many registers at the inputs and outputs of the cell (Figure 24) [58]. DSP cells can also be cascaded, which adds more flexibility in applications like FIR filters.

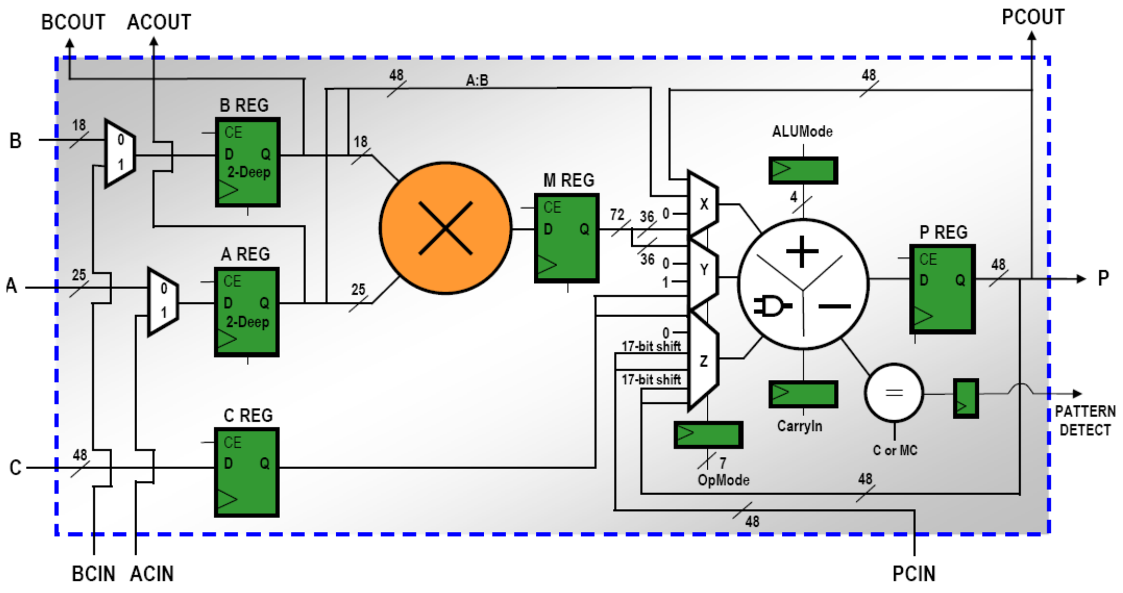


Figure 24 Internal Structure of a DSP48E cell. (From: [58])

As an example, in their newest chips, Virtex-5 Xilinx is offering the DSP48E slice, which is a 25-bit by 18-bit multiplier along with a 48-bit accumulator. This offers impressive performance including speed and power while using little silicon real estate. The number of DSP slices is limited to a number between 32 and 192, but still, the DSP acceleration they offer is noticeable. For even greater convenience, the many library blocks can be optionally implemented using these DSP slices, yielding very fast designs. Figure 24 shows the block diagram DSP48E [59].

Regarding soft cores, Actel delivers synthesizable versions of ARM7 (CoreMP7) and ARM Cortex (M1, M3) free of charge. Advanced RISC Machine (ARM) processors in their hardwired form are processors mainly developed for mobile devices with a Reduced Instruction Set Computing (RISC) core. The mother company that develops the new ARM processors only licenses them without manufacturing them on its own. Every respective vendor in the electronics sector owns at least one license. ARM’s new line of products also includes synthesizable cores, like the older ARM7 and the new Cortex that can just be downloaded to an FPGA. Thus, inside an FPGA it is possible to have a GPP plus some silicon left for other designs. The embedded cores can run different real time operating systems (RTOS) and support modern connectivity protocols, like Gigabit Ethernet and RapidIO [60].

Xilinx went one step further by producing FPGAs with built-in PowerPC® 440 blocks. In some models of Virtex-5, Xilinx even includes two PowerPC cores [61]. Altera is not only offering its own RISC version (Nios® II) along with ARM Cortex M1, but recently updated with the Freescale's 32bit V1 ColdFire [60]. Freescale is another manufacturer that produces microprocessors for embedded devices and its ColdFire chip is a 68k series microprocessor.

Regardless of the previous advancements, some operations still exist that are not suitable for FPGAs, like division by a number not a power of 2 and especially between floating point numbers [62]. Sometimes, these operations are implemented with look-up tables, but there are some shortcomings that are easier to implement in DSP chips. For this reason, modern platforms force DSP and FPGAs to coexist in order to achieve maximum performance.

G. THE ROL E OF FPGAS IN SDR – HOW TO COMBINE DSP-FPGA COPROCESSOR

A goal of SDR is the ability for a single transceiver to conform to multiple different air interfaces and modulation formats. The design that would accommodate all present and future needs of a SDR product must be flexible, scalable and of high performance. The use of many DSP processors in parallel configuration is not practical because of complexity and power consumption. Furthermore, there should be a margin between the performance of the processor and the current needs in order to accommodate any future demand. An example granted from the video compression area of mobile devices is the comparisons of the standard MPEG-2 with the newest H.264. The algorithmic complexity of high definition resolution H.264 is three times that of standard definition resolution MPEG-2 video compression, which is translated on an order of magnitude increase in system performance. Thus, there should be enough computational power even for future standards. Otherwise, no update can be performed, thus reducing versatility [63].

This reconfigurability does not come without a computational cost and complexity cost, because analog parts cannot be used extensively anymore and digital circuits do not always have the bandwidth to support wideband communications. As it has been demonstrated, one family of chips cannot provide all the characteristics needed in order to make SDR a reality. The power of modern FPGAs offers much flexibility and can help realize the SDR concept. Nevertheless, in order to combine all features needed a cross-chip platform is necessary. In practice, typical SDR platforms include all three DSPs, FPGAs and GPPs to deal with the complexity, cost and power constraints.

The initial use of FPGAs as mere interconnecting logic between external interfaces and computational chips (or chips in the system) or between DSPs and GPPs, has now changed. FPGAs are also used as fabric where special circuits are built, in cases where speed requires implementation of these DSP functions in hardware. Thus, they are used as coprocessors to either DSPs or GPPs, in order to accelerate some functions that are frequently used or could benefit from a parallel structure. Tools provided by the manufacturers of the FPGAs make the mapping from high-level languages to Hardware Description Languages (HDL) easy to use [64].

The different functions that usually are assigned to these devices are illustrated in Figure 25 [65].

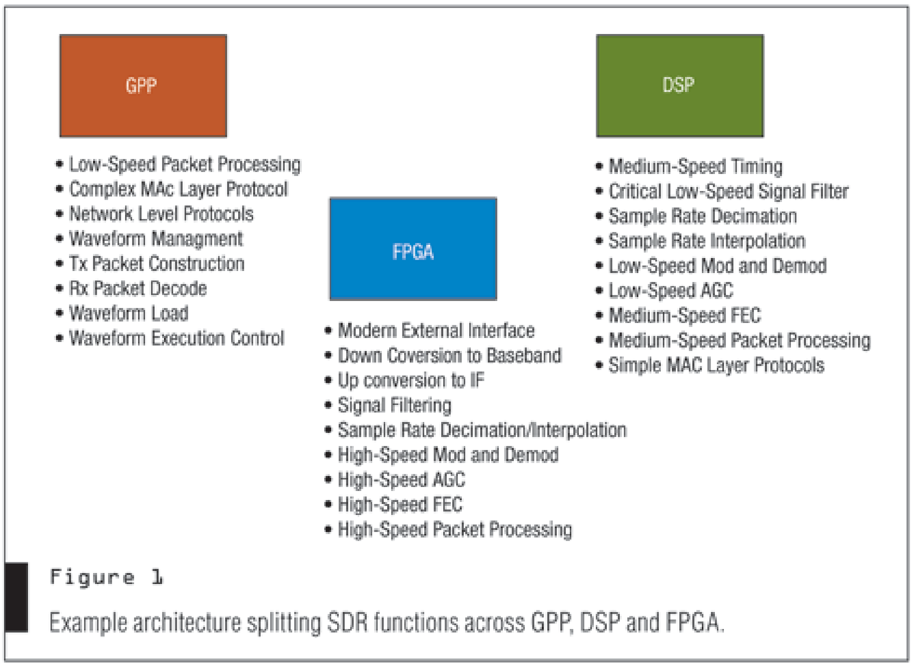

GPP
DSP
FPGA
• Low-Speed Packet Processing
• Complex MAc Layer Protocol
• Network Level Protocols
• Waveform Managment
• Tx Packet Construction
• Rx Packet Decode
• Waveform Load
• Waveform Execution Control
• Medium-Speed Timing
• Critical Low-Speed Signal Filter
• Sample Rate Decimation
• Sample Rate Interpolation
• Low-Speed Mod and Demod
• Low-Speed AGC
• Medium-Speed FEC
• Medium-Speed Packet Processing
• Simple MAC Layer Protocols
• Modern External Interface
• Down Coversion to Baseband
• Up conversion to IF
• Signal Filtering
• Sample Rate Decimation/Interpolation
• High-Speed Mod and Demod
• High-Speed AGC
• High-Speed FEC
• High-Speed Packet Processing
Figure 1
Example architecture splitting SDR functions across GPP, DSP and FPGA.

Figure 25 Different Functions Assigned to GPPs, FPGAs, and DSPs (From: [65]).

An FPGA used as a coprocessor seems to yield a balanced solution. In such cases, the DSP code must be partitioned into the parts that will be executed by the DSP processor and by the FPGA. In "Hybrid FPGA/DSP architecture: the optimal solution" by Jeffry Milrod [66], the author mentions that the FPGA should be placed close to the signal I/O. This configuration can use the FPGA as a reconfigurable I/O controller in support of various standards (like PCI express, GigabitEthernet etc.). Also, it solves the bandwidth problem of connections between fast I/O devices and the core.

The general guidelines that should be followed during the design of such systems are described in [67] and include the folly of trying to transfer a code previously written for a DSP platform to the new architecture. The serial, sequential logic of a DSP has nothing to do with the parallel logic of FPGA designs. Other guidelines refer to the split of tasks executed between each of the two chips, suggesting that the control part should be better instantiated in the FPGA, because many soft embedded processors are offered for FPGAs. The paper also refers to the evaluation of different choices regarding intellectual property in the design. While producing an intellectual property is more expensive, time to market may force its purchase.

Another point of great importance is the bandwidth of the interconnection between the DSP and the FPGA. In hybrid architectures, much data is going to go back and forth between the main computational elements, depending on where the computation is more efficient. Thus, in order to be applicable in practice, the interfaces must be of low latency and fast [68].

Texas Instruments' Small Form Factor Software-Defined Radio is an implementation example that uses Xilinx Virtex-4 SX-35 FPGA, TI's TMS320CC64x, a 600 MHz chip, DSP and an ARM926EJ-S processor. As expected, the DSP undertakes the signal processing load, while the GPP supports network and application processing and the Virtex-4 is used for modem co-processing and acceleration functions. The manufacturer claims the existence of both a DSP and ARM on a single chip has the benefit of reduced system space and cost [69].

H. BEYOND THESE TECHNOLOGIES, WHAT NEXT?

Some new technologies advertise a combination of both FPGA and ASIC benefits. Usually the chips implementing these technologies offer partial reconfigurability keeping other parts hard-wired, placing themselves in between the two extremes. Others are highly parallel devices that incorporate an internal structure to implement the difficult problem of massive parallelism efficiently. Nevertheless, none of these technologies have gained a dominant position in the market [70].

The eASIC is promoting the so called Second Generation Structured ASICs, Nextreme2™ [71]. It is a 45 nm design that belongs to the category of ASIC-FPGA Hybrids. The exact way this is implemented is very well illustrated in Figure 26 [72]. The specific choice of using routing via single mask eliminates the need for a very large overhead of SRAM elements that the flexible routing would need. In this way, the current consumption is reduced, keeping in mind the current leaking that SRAM elements encounter. The cost per chip is also suppressed while keeping the mask charges very low, which in turn removes the minimum quantity constraints that conventional ASICs would have.

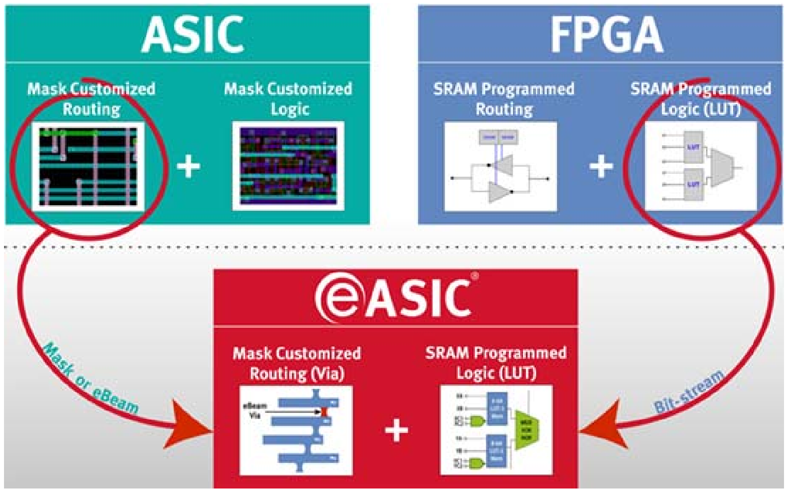


Figure 26 Illustration of the concept behind eASIC's structured ASICs (From: [72]).

On the other hand, Nextreme retains the internal structure of cells, called an eCell, the same as FPGAs. This allows some of FPGA's flexibility and reconfigurability. The company advertises the cost of the development tools as well as time to market similar to that of FPGAs. Nextreme can host a plethora of soft cores, including ARM 926EJ, and Tensilica Diamond Standard Processors, which are mainly for audio processing.

There are two device options, one for prototyping and one for mass production. The method used to customize the interconnections in this product is maskless lithography and is called the Direct-write e-Beam. This technology uses an electron beam to write directly on the wafer. The paper [73] on the company's website describes this technology.

The PicoChip's picoArray™ is another architecture that has managed to differentiate from the competition. It consists of a massively parallel design where 308 tiled processors are connected in a 2D grid. These 16-bit Harvard processors each have a small local memory and each one runs its own process. For proper interconnection, they are all attached to a network of 32-bit buses, the picobuses, and programmable bus switches. Multiple picoArray cores can be used in a parallel structure to give even more computational power. In each picoArray, multiple functional acceleration units (FAU) exist for speeding some specific tasks, like Advanced Encryption Standard (AES) encryption. Some models even have an embedded ARM-9 processor. They have proved to have very good processing capabilities in known DSP calculations, like the FFT or IFFT, and error control coding and decoding. This chip has been deployed in wireless infrastructure [74]. See Figure 27 [75].

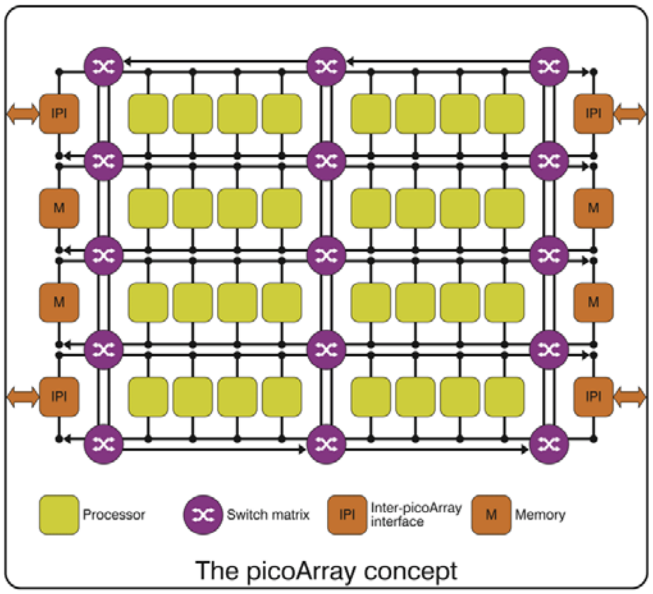


Figure 27 PicoArray Concept (From: [75]).

I. LIMITATI ONS

The question that arises naturally is if the ultimate goal of software radio can be achieved. This goal is to build devices that can handle every possible modulation by just loading the proper software. As described in [76], some parts of the radio are not even close to digital implementation due to cost or space. The main limitation arises from Digital to Analog Converters that are not fast enough for most Radio Frequencies (RF). The solution usually used is to perform analog to digital (Rx) and digital to analog (Tx) conversion at a low intermediate frequency (IF). The conversion between the IF and RF is usually performed using analog hardware. The advances made in that domain do not seem capable of changing that in the near future.

APPENDIX B. IN DEPTH PARAMETER ANALYSIS OF BFSK TRANSCEIVER DESIGN

In this Appendix, a description of the specific function of each block and the settings of its parameters can be found. The reading of this Appendix in parallel with Chapter IV is proposed for someone not familiar with the Xilinx environment in order to acquire a better overview of the meaning of each block. It is also useful as a reference guide to someone that would like to reproduce the circuit or use it as a platform to extended to a different modulation scheme.

A. TRANSMITTER (TOP LEVEL)

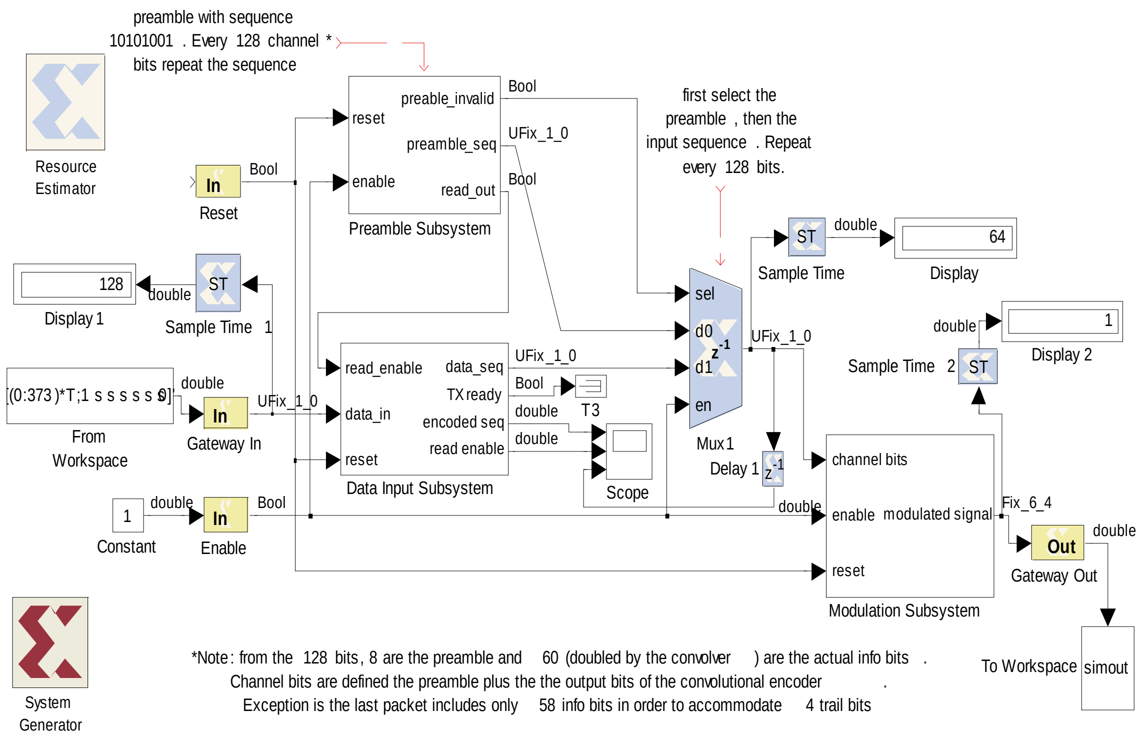

preamble with sequence 10101001. Every 128 channel * bits repeat the sequence
Resource Estimator
Reset
Preamble Subsystem
preable_invalid
preamble_seq
read_out
reset
enable
first select the preamble, then the input sequence. Repeat every 128 bits.
Display 1
Sample Time 1
Sample Time
Display
Sample Time 2
Display 2
From Workspace
Gateway In
Data Input Subsystem
read_enable
data_in
data_seq
TX ready
encoded seq
read enable
T3
Scope
Mux 1
sel
d0
d1
en
Delay 1
Constant
Enable
Modulation Subsystem
channel bits
modulated signal
Gateway Out
To Workspace
simout
System Generator
*Note: from the 128 bits, 8 are the preamble and 60 (doubled by the convolver) are the actual info bits. Channel bits are defined the preamble plus the the output bits of the convolutional encoder. Exception is the last packet includes only 58 info bits in order to accommodate 4 trail bits

Figure 28 Transmitter's schematic diagram designed in Simulink/Sysgen environment.

- **<u>System Generator</u>**: its existence is compulsory to every design. It defines the type of target FPGA the FPGA clock period, and other key parameters.

  Key parameters:
  Part: Virtex-4 xc4vlx25-10sf363.This is the target FPGA.
  FPGA clock period (ns):10.

Clock pin location: A8. This choice depends from the actual pin that the mainboard provides the clock pulses and it is found in the board's manual. This parameter is crucial for proper function of the design on the FPGA and post place and route simulation is not feasible if this parameter is not defined.

Simulink system period (sec): T/128. This number is defined by the faster component of the design. The value T corresponds to the desired information bit period and equals 128*10 ns. The simulation of the design is made with time steps of T/128 sec, as defined in Simulation Tab ->Configuration Parameters in the Simulink window. In explicit, when a block is defined to work at a sample period T, it yields an output once every 128 Simulink periods. In this case, the 'System Generator' block only defines the basis for the other blocks. When a block has sampling period T, it yields output 128 times slower than the reference period.

- **<u>Resource Estimator</u>**: It is a block that provides an estimate regarding the FPGA resources that are required to build the circuit.

- **<u>From Workspace</u>**: inserts variables from the Matlab Workspace.

  Key Parameters:
  Data: [(0:1901)*T;1 a 0]' where a =rand(1,1900)<.5 is executed in the Matlab Command Window.

- **<u>Gateway In, Reset, Enable</u>**: converts the input to Xilinx fixed point type. The part of the design that is after this block is synthesized by System Generator. The block itself becomes a top level input port.

  Key Parameters:
  Output type: Unsigned consisted of 1 bit. The input is 0 or 1.
  Sample period: T. The block is working at the Simulink simulation period and 128 times slower than the reference.

- **<u>Sample Time1</u>**: illustrates the simulation period concept discussed in the System Generator' block. It uses a display block to report the normalized sample period value. A value of 128 that is shown in 'Display 1' means that the input is 128 times slower than the reference. For the specific case that the 'System Generator' block has a value of T/128, it means that the previous block of the 'Sample Time 1' has a sample period of T.

- **<u>T3</u>**: terminates the its input to avoid warning messages. It also means that its input is not considered useful in this design. In this case, the 'TX ready' signal proved not to help the problem of the gap between the preamble and the encoded bits as discussed in Section C.1, Chapter V.

- **<u>Mux1</u>**: multiplexer with select (sel) of type unsigned and configurable number of data bus inputs (d0,d1). The Enable port (en), which is optional, forces the latency to be more than 1. The exact value of latency is also shown on the block's figure as the negative exponent of the z symbol.

- **Delay1**: Multiple delays are spread over the design. Sometimes their role is to ensure timely propagation of the signals, other times they take care of the synchronization of different branches of the design. This is one case that delay should not have been used, because the scopes are not synthesized in contrast to the delay blocks. The proper way is to use a 'Gateway Out' block and Simulink's delays right after.
- **Sample Time, Sample Time2**: As discussed in 'Sample Time1.'
- **Gateway Out**: Opposite functionality than 'Gateway In.' It converts the Xilinx fixed point input to a Simulink compatible type. These blocks are synthesized in top level output ports.

  Key parameters:
  Input/Output Buffers (IOB) pad locations: {'U9','V9','V10','V11','P19','U12'}. These are the output pins from Most Significant Bit to Least Significant Bit. The number of the pins is equal to the number of the output bits.
- **To Workspace** : Stores the values presented at its input as a Matlab variable for further analysis in the Matlab Environment. In this case, the data will be forwarded to the receiver's input.

  Key parameters:
  Variable name: simout. This is the name of the variable that will store the output values of the Transmitter. They can be recovered through the path simout.signals.values because simout is a structure that saves other information like the time that corresponds to the respective value.

B. PREAM BLE SUBSYSTEM

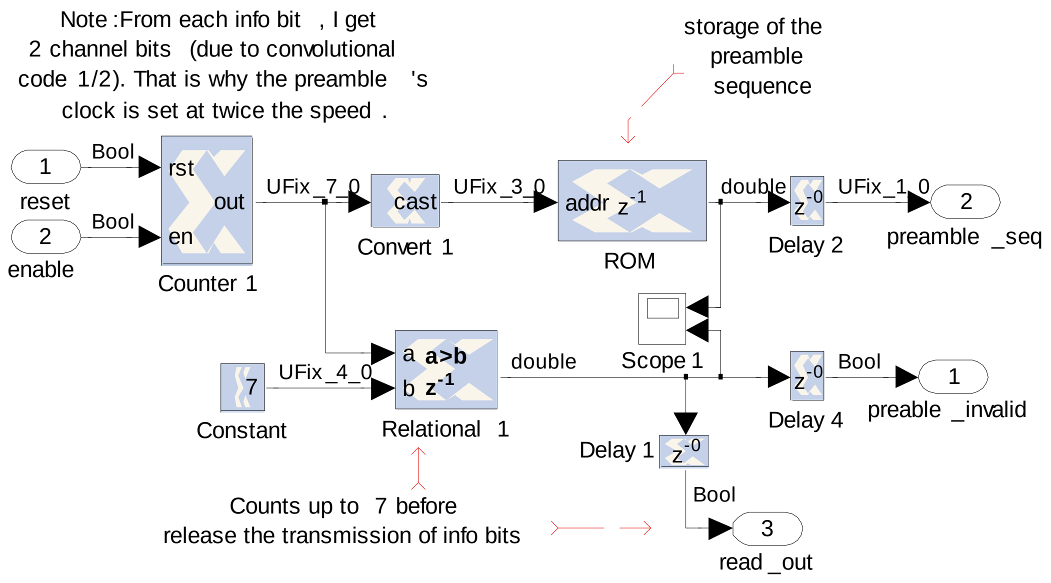


Figure 29 Preamble Subsystem.

- **Counter1**: A counter should be thought of as a clock with an adder. Its output can be usually used by comparators to enable or disable signals. In this case, the counter is also used to directly provide the next address to the 'ROM' block.

  Key parameters:
  Count to value: 127. This depends on the packet size and corresponds to a packet length of 128.
  Number of bits: 7. This number must merely accommodate the maximum value of the counter. In this case, seven bits are enough to represent the maximum value of 127.
  Explicit Sample Period: T/2. In this case, the preamble has bit period equal to the encoded bit period. For the specific encoder of choice, this means that the encoded bits should have half the period of the message bits. Recall that the 'Gateway In' has sample period of T.
- **Convert1**: Translate the input to a desired output type. The need that forces its use is that the 'ROM' block does not accept addresses that are not compatible with its depth.

  Key parameters:
  Output precision: 3. The 7 bit output value of the 'Counter 1' must be converted to 3 bits input to the 'ROM.'
  Overflow: Saturate. Does not have any specific impact to the performance of the design, just makes it easier to see the output plots in 'Scope1.' The 'ROM' output will be always the last bit of the sequence while the counter output will be more than seven.
- **ROM**: It is a single port read-only memory (ROM). The preamble is stored in this memory, given that it does not change over time.

  Key parameters:
  Depth: 8.
  Initial value vector: [1 0 1 0 1 0 0 1]. This is the preamble sequence.
  Number of bits (Output Precision): 1
- **Relation1**: It is a comparator that can support a plethora of different comparisons. Here, the output of the 'Counter1' is compared with a constant number to determine if the preamble or the encoded message bits should be transmitted. Whenever the 'Counter1' is less or equal to 7, the preamble is transmitted, otherwise the encoded sequence is selected.

  Key parameters:
  Comparison: a>b.
- **Delay1, Delay2, Delay4** : delays that had been used for synchronization troubleshooting purposes. Their actual value is set to zero and do not affect the design as discussed in Section C.1 in Chapter V.

C. DATA INPUT SUBSYSTEM

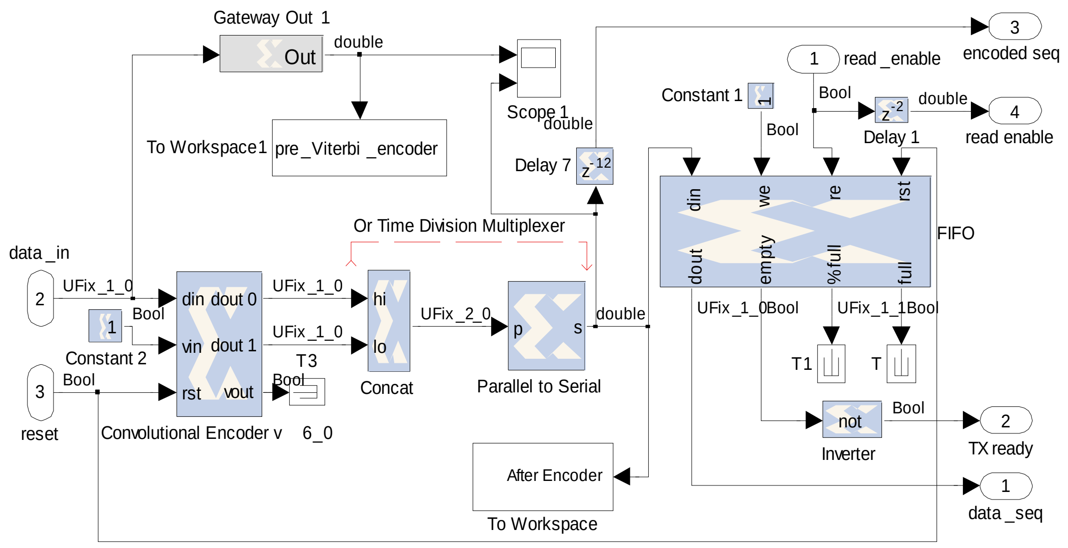


Figure 30 Data Input Subsystem.

- **<u>Convolutional Encode r v6_0</u>**: is an encoder that uses a convolutional code. The decoder that matches the convolutional encoder is the Viterbi decoder. Encoders are provided by Xilinx as free to use IP blocks, although it is not the same with the decoders. In digital communications, encoding is used for forward error correction. In this design, the encoding is applied to the whole message sequence, and not in a per packet basis. A generic diagram of a rate $r = \frac{1}{2}$ code is provided in Figure 31 to grant some further insight and a more detailed description is provided in Section B, Chapter II.

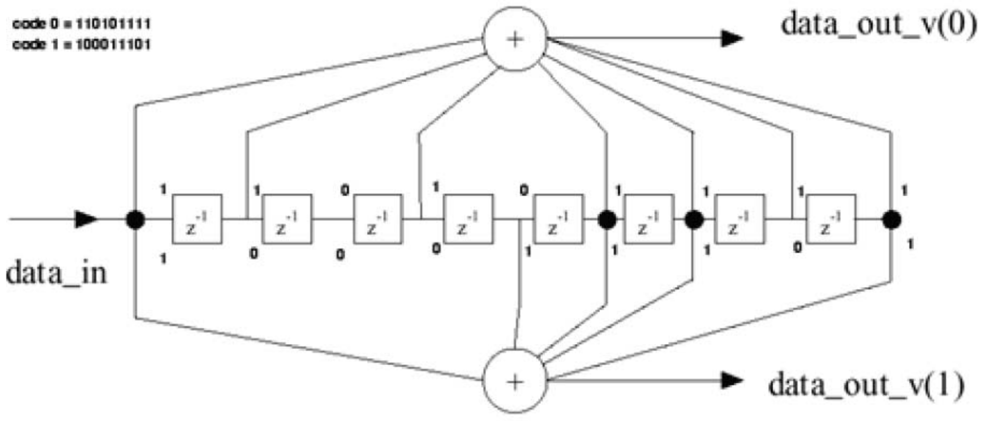


Figure 31 A block diagram of a convolutional encoder. (From: [13]).

Key parameters:
Constraint length: 3. This means that the shift register of the encoder has two flip-flops.
Convolutional code array (octal): [7 5]. This code is the one proposed by Xilinx for this specific constraint length. This code means that the first output branch of the convolutional encoder (data_out_v(0) in Figure 31) is adding modulo 2 the values stored in all flip-flops and the input value (111_b), and the second branch(data_out_v(1) in Figure 31) is using only the values of the last flip-flop to the right and the input value (101_b).

- **<u>Concat</u>**: this block concatenates the two inputs into one word. The ultimate goal using this block is to multiplex the two output streams of convolutional encoder into one stream.

- **<u>Parallel to Serial</u>** : This block complete the time division multiplexing started by 'Concat' block. Every word at the input is broken into separate bits and is sent serially to the output.

  Key parameters:
  Output order: Most significant word first
  Type (Output Precision): Unsigned
  Number of bits: 1
  Note: Both the 'Concat' and 'Parallel to Serial' blocks could have been replaced by a 'Time Division Multiplexer' block where no extra parameters are needed, except the number of inputs.
- **<u>Gateway Out1</u>** : It converts the Xilinx fixed point input to a Simulink compatible type. It drives the 'Scope1' and 'To Workspace1' blocks.

  Key parameters:
  Translate into output port: Disable. This block is not an instance of an output port. An output pin is not assigned to this port during synthesis.
- **<u>To Worksp ace1</u>**: Stores the values presented at its input as a Matlab variable for further analysis in the Matlab Environment. In this case, the data represents the message sequence and will be used to compare the receiver's output and count the number of errors.

  Key parameters:
  Variable name: pre_Viterbi_encoder.
- **<u>Delay7, Delay1</u>**: Delay used to align the plots in the 'Scope1.' As is the case with many delays that drive Scopes, this is not a proper way to implement a delay (see: 'Delay1' in Figure 28 description in Appendix B).

- **<u>FIFO</u>**: It is a First In First Out memory queue. The input values engage next available memory location in the memory queue. This function is permitted whenever write enable (we) signal is high, otherwise the input data are discarded. In this case, 'we' is always high ('Constant1') allowing the encoded message bits to be saved in the memory, even while the preamble sequence is transmitted. Read enable (re) signal is defined by the Preamble Subsystem and it is high for the time that the preamble is not transmitted. This allows the encoded bits to appear to the input of the Modulation Subsystem. Outputs '%full' and 'full' are not used and are terminated by 'T1,''T' blocks. 'Empty' signal had not been possible to be used (see also Troubleshooting of the Transmitter in Chapter V) and is terminated just outside the Data Input Subsystem.

- **Constant1**: Constant of value 1 that keeps the 'we' signal of 'FIFO' always high.
- **Inverter**: bitwise negation the Boolean value of its input.

D. MODULATION SUBSYSTEM

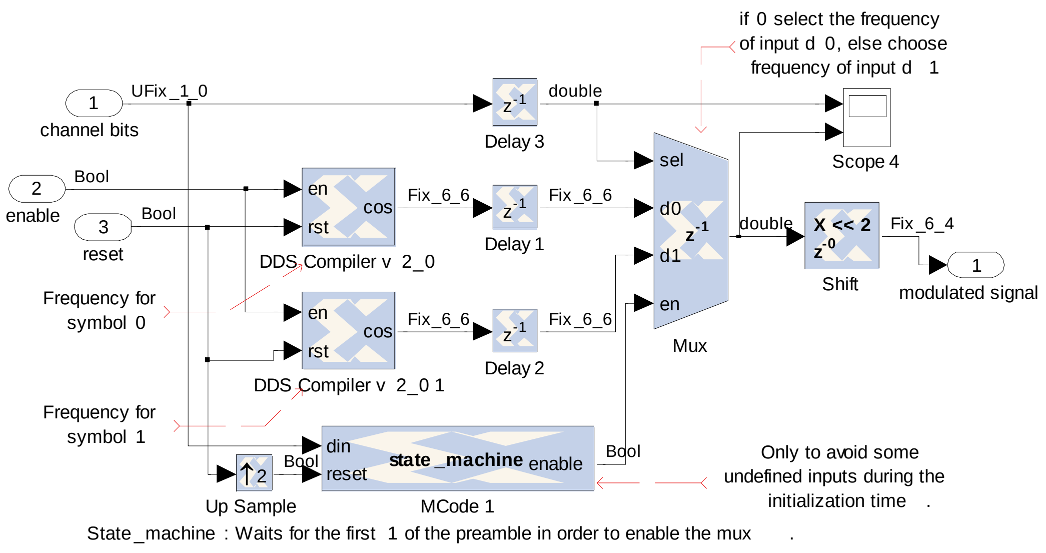


Figure 32 Modulation Subsystem.

- **Mux**: same description as 'Mux1' in Figure 28 in Appendix B. Same parameters.
- **Delay1, Delay2, Delay3**: Delays that ensure the timely propagation of the signals. It is often necessary to insert delays between adjacent blocks.
- **DDS Compiler v2_0 and DDS Compiler v2_0 1**: Direct Digital Synthesizers (aka Numerical Controlled Oscillators) that produce a sinusoidal output using a lookup table. One DDS is devoted to generate the frequency assigned to 0 and the other is used for the generation of 1's frequency. The output width is by default 6 bits, all placed after the binary point. Since the first bit is the sign bit, only 5 bits are for magnitude. This means the output takes values from -0.5 to +0.5 and not from -1 to +1.

Key parameters:
DDS clock rate (Mhz):100.0. This number must be at least twice the output frequency and at most 500 for the Virtex-4 target FPGA. According to Xilinx technical support email, the frequency of the DDS clock rate should match the parameter 'FPGA clock period (ns)' in Sysgen token. In this case FPGA clock period is 10ns, yielding a frequency of 100MHz.
Frequency resolution (Hz):0.03.
Output Function: Cosine
Output Frequency array (MHz): [45.0] for the 1s and [40.0] for the 0s. These choices depend from the bit duration as well. The bit period of the encoded bits is 64*10 ns. For the two frequencies to be orthogonal as shown in Section A in Chapter II, their spacing must be a multiple of the bit rate. In this case the spacing of 5MHz is 3.2 times the bit rate $R_{c.b.} = \frac{1}{64}10^8$ and the frequencies are not orthogonal.
Explicit period: T/128. The decision made was to use 64 samples of the sinusoid for every bit. Given that the bits at the entrance of the modulation Subsystem are at a period of ½, then the signal that would have the proper period is one with a value set to T/128.
Noise Shaping (Under Advanced tab): Phase dithering [77]. This choice should improve the quality of the sinusoidal samples, minimizing the quantization error.
DSP48 Use (Under Implementation Tab): Maximal. Given that DSP48 cells are not used anywhere else in the transmitter, some of them can be sacrificed to increase the performance of this block.

- **MCode1**: This block is used to execute simple assigned Matlab functions. The code is translated in VHDL or Verilog language during the synthesis phase. It only supports a small subset of the MATLAB language. In cases where this is a problem Xilinx AccelDSP Synthesis tool can be used to support a larger set of Matlab commands and to create custom IP blocks. MCode block only supports Xilinx fixed-point type.

The function assigned to this block has to do with the initialization of the 'Mux' block. In order to avoid the propagation of undefined signals during the initiation phase, a state machine was written that delays the enable (en) of the 'Mux' block until the first bit of the preamble is detected at the input of the Modulation Subsystem.

Code:

```
function enable = state_machine(din,reset)

% define the state variables. They will be retained to memory between
% following runs.
persistent state, state = xl_state(0,{xlUnsigned, 1, 0});

switch state
```

```
    case 0
        if din == 1 %when the fisrt bit of the preamble is detected
            state = 1; %go to the next state (enable high)
        else
            state = 0;
        end
        enable = xfix({xlBoolean},0); %xfix() translates values…
                                      %to a Xilinx fixed-point type.
    case 1
        if reset ==xfix({xlBoolean},1) %check synchronous reset
            state =0;
            enable = xfix({xlBoolean},0);
        else
            enable = xfix({xlBoolean},1); %otherwise stay locked to…
                                          %the same state (enable high)
        end
    otherwise
        state = 0;
        enable = xfix({xlBoolean},0);
end
```

- **Up Sample**: up samples input data by inserting zeros or copies of previous sample. It is used to make the sample rate of the 'din' and 'reset' signals compatible. System Generator does not accept a state machine with inputs of different sampling periods.

  Key parameters:
  Copy samples: enabled.
- **Shift**: This block generally performs a left or right shift on the input. The purpose of this block is to amplify the signal before transmission. It should be noted that DDS blocks yield values from -0.5 to +0.5 and not from -1 to +1.

  Key parameters:
  Shift direction: Left. This direction is amplifying the signal.
  Number of bits (Shift direction): 2. This number is amplifying the signal by a factor of four.
  Number of bits (Output type):6. The total number of bits is not changing, only the decimal point.
  Binary point (Output type):4. The change of the position of the binary point reflects the shift made by the block. The previous binary point position of 6 has now changed to 4, meaning that the binary point shift is two places to the right.

E. RECEIVER (TOP LEVEL)

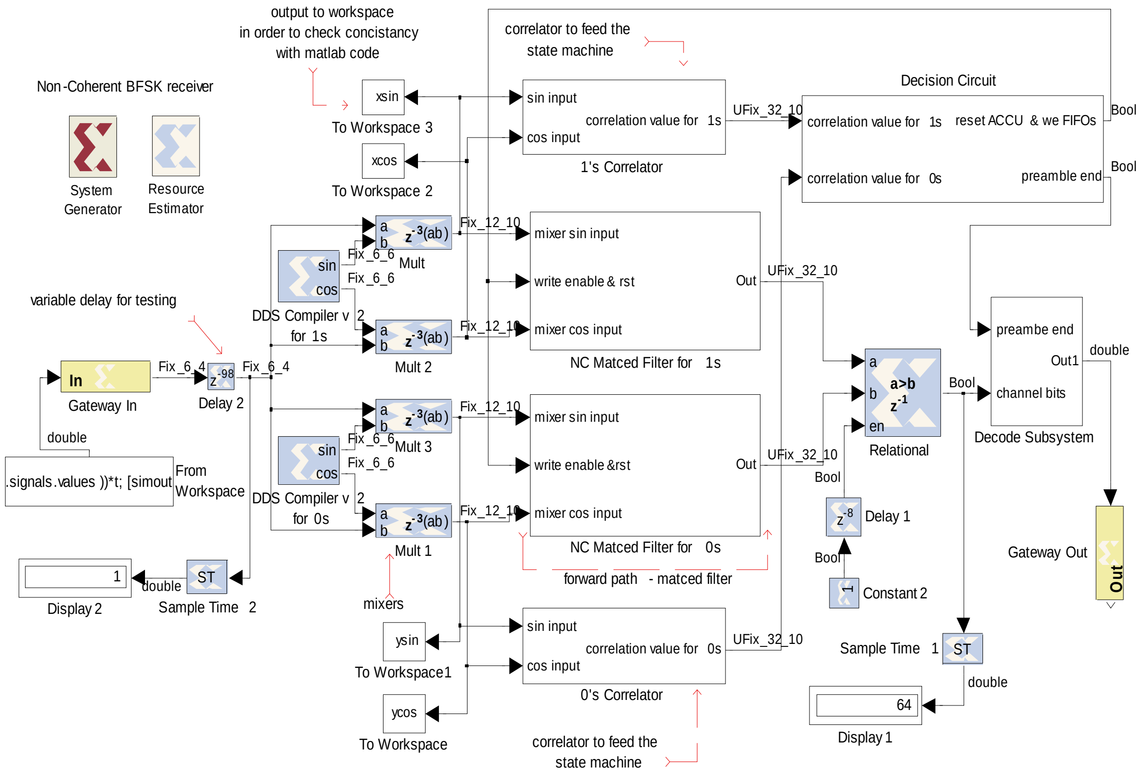


Figure 33 Receiver's schematic diagram designed in Simulink/Sysgen environment.

- **<u>System Generator</u>**: its existence is compulsory to every design. It defines the type of target FPGA, the FPGA clock period, and other key parameters.

  Key parameters:
  Part: Virtex4 xc4vlx25-10sf363.This is the target FPGA.
  FPGA clock period (ns):10.
  Clock pin location: A8. This choice depends from the actual pin that the mainboard provide the clock pulses and it is found in the board's manual. This parameter is crucial for proper function of the design on the FPGA and post place and route simulation is not fisible if this parameter is not defined.
  Simulink system period (sec):t, where $t = 10 \bullet 10^{-9}$ defined in the Matlab Workspace. In contrast to the same block of the Transmitter, here the input is samples, not bits, which is the same as the fastest blocks in this design. Given that the time step of the Simulink simulation is t, one Simulink simulation period is equal to the reference period of the Xilinx model.

- **Resource Estimator**: It is a block that provides an estimate regarding the FPGA resources that are required to build the circuit.

- **From Workspace**: inserts variables from Workspace. Here, this variable is the stored output values of the Transmitter.

  Key parameters:
  Data: [(1:length(simout.signals.values))*t; [simout.signals.values]']' where simout.signals.values represents the output samples of the Transmitter.
- **Gateway In**: converts the input to Xilinx fixed point type. The part of the design that is after this block is synthesized by System Generator. The block itself becomes a top level input port.

  Key parameters:
  Output type: Signed consisting of six bits with the binary point at the fourth position (from the left). The input is cosine samples amplified by a factor of four. This should match the output type of the Transmitter.
  Sample period: t. The block is working at the Simulink's simulation period and at the reference period as well.
- **DDS Compiler v2 for 1s** and **DDS Compiler v2 for 0s** : Direct Digital Synthesizers (aka Numerical Controlled Oscillators) that produce a sinusoidal output using a lookup table. One DDS is devoted to generate the frequency assigned to 0 and the other is used for the generation of the 1's frequency. The output width is by default 6 bits, all placed after the binary point. Since the first bit is the sign bit, only 5 bits are for magnitude. This means the output takes values from -0.5 to +0.5 and not from -1 to +1.

  Key parameters:
  DDS clock rate (Mhz):100.0. This number must be at least twice the output frequency and at most 500 for the Virtex4 target FPGA. Here, the value matches the respective value of the transmitter.
  Frequency resolution (Hz):0.03.
  Output Function: Cosine
  Output Frequency array (MHz): [45.0] for the 1s and [40.0] for the 0s. These choices should match the values defined in the Transmitter.
  Explicit period: t. The block is working at the Simulink simulation period and at the reference period as well.
  Noise Shaping (Under Advanced tab): Phase dithering. This choice should improve the quality of the sinusoidal samples, minimizing the quantization error.
  DSP48 Use (Under Implementation Tab): Maximal. Given that DSP48 cells are not used anywhere else in the receiver, some of them can be sacrificed to increase the performance of this block.

- **Mult, Mult1, Mult2, Mult3** : This block multiplies its two inputs. It should be noted that because the input from 'Gateway In' takes values from -2 to +2 and the input from DDS is from -0.5 to +0.5, the output is between ± 1.

  Key parameters:
  Precision: full.
  Use embedded multipliers (Implementation): Enabled. This choice will make use of the DSP48 embedded cells to execute the operation faster. It also releases generic cells that can be used for a different purpose.
- **Relational**: is a comparator that can support a plethora of different comparisons. Here, it compares the output of the two filters. When the output of the Non-Coherent Matched Filter for 1s is higher than the output of the Non-Coherent Matched Filter for 0s, then the decision is that 1 was transmitted.

  Key parameters:
  Comparison: a>b.
  Provide enable port: enabled.
  Latency: 1. Whenever the enable input is chosen, the latency must be one or more.
- **Constant2**: Provides the enable signal to Relational.
- **Delay1**: delay measured exactly to overcome initialization problems. Before the propagation of Constant2, the output of the delay is zero.
- **Gateway Out** : It converts the Xilinx fixed point input to a Simulink compatible type. These blocks are synthesized in top level output ports.

F. MATCHED FILTER SUBSYSTEM

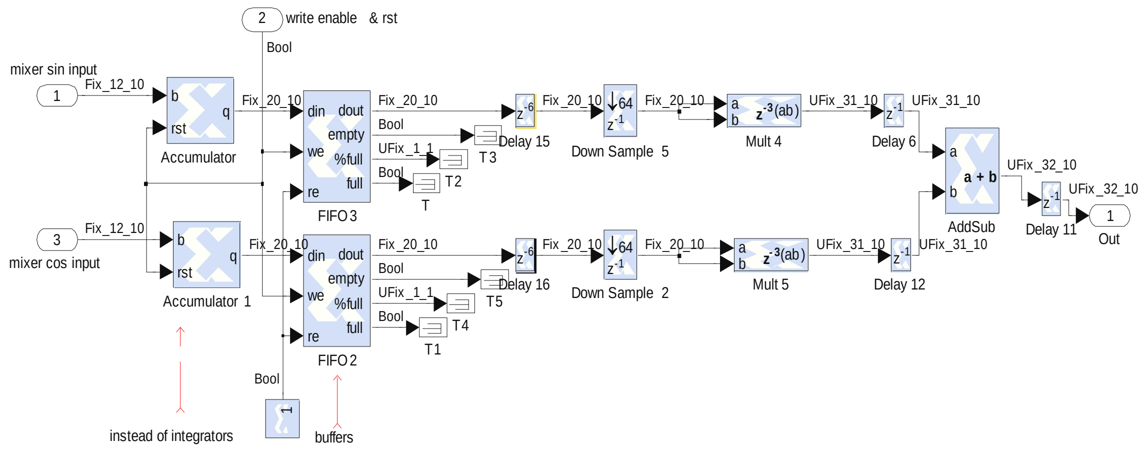


Figure 34 NC Matched filter subsystem (one of two).

- **Accumulator, Accumulator1** : Implement the integration concept in the discrete case. The integration must be over one bit period, thus 64 consecutive samples from the mixers must be added. After 64 samples, a reset signal is expected to restart the same process for the next bit. The reset must be synchronized with the beginning of every bit.

  Key parameters:
  Operation: add.
  Output precision: 20 bits. Given that each input from the mixers cannot exceed a value of 1 and the adder adds 64 samples and assuming all values with the same sign, the sum cannot exceed 64. This value corresponds to 6 bits for the integer part plus one for the sign. Given that the binary point is inferred from the input and is placed at the 10 position, the output should be at most 17 bits wide. Some extra bits are given.
- **FIFO2, FIFO3**: description of the block as in Figure 31. The FIFOs here are used as a convenient way to capture the value of the accumulators just before the reset. A simple register with an enable port should be sufficient to yield the same result. Read enable (re) is always high.

- **Delay6, Delay11, Delay12, Delay15, Delay16** : Delays that ensure the timely propagation of the signals.

- **Down Sample 2, Down Sample 5: This block r**educes the sample rate of the input, discarding the extra **values provided in the highe** r rate input. The capture of the output value of the accumulator by the FIFOs is made once per sixty-four sample periods. Given that this value is changing once per bit period (sixty-four sample periods) there is no need for the blocks after the FIFOs to run at sample period.

  Key parameters:
  Sampling rate (number of input samples per output sample): 64*t. Switching from sam**P**le period to bit period.
  Sample: Last value of the frame. This choice was made due to less hardware needed for its implementation. This choice introduces at least one latency.
- **Mult4, Mu lt5**: This block multiplies its two inputs. In this case the multiplication simulates the squaring operation by providing the same signal to both inputs of the block.

  Key parameters:
  Output type: Unsigned 31 bits with binary point at the tenth position. The multiplication may double the bits of the integer part. Given that six bits were calculated to be sufficient for the integer part after the accumulators, twelve bits are now needed for a total of 23 bits with the sign. Some extra bits are offered.
- **AddSub**: This block implements the addition or subtraction operation. In this case, the addition of the two branches must be made.

G. CORRE LATOR'S SUBSYSTEM

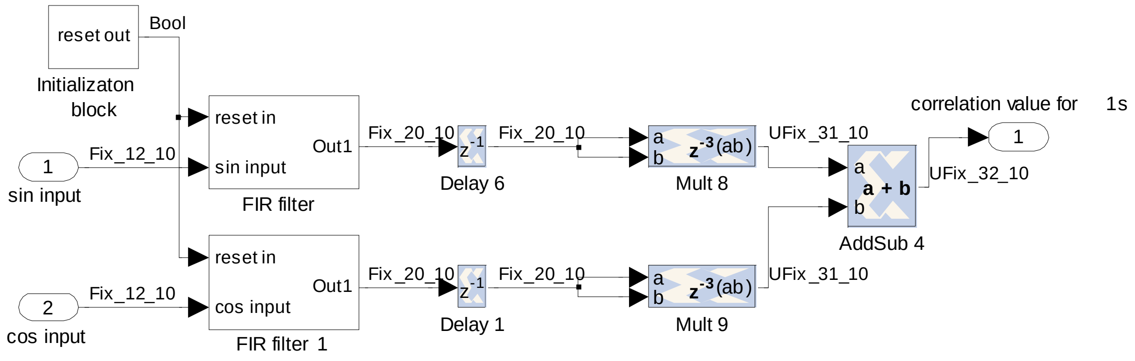


Figure 35 Correlator's Subsystem (one of two).

- **Initialization block**: This is a custom block that resets the FIR filters at the beginning of the simulation and does not affect the circuit anymore. This need appeared after switching from the Xilinx FIR compiler block to the custom 'FIR filter' block, where the message for the propagation of indeterminate values appeared. It consists of a constant, a delayed constant and a comparator as shown in Figure 36. The 'Relational' finds input a higher than input b only at the first cycle of the simulation and at that instance sends a reset signal to the 'FIR filter' and 'FIR filter 1.' After the first cycle, the delayed 'Constant2' render to input 'b' a value that is equal to input 'a' of the 'Relational,' forcing 'reset out' to go low. It should be noticed that all constants are explicitly sampled in 64*t.

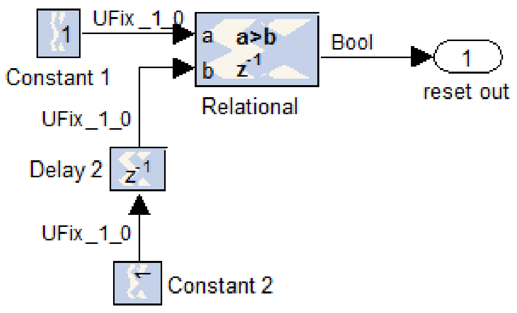


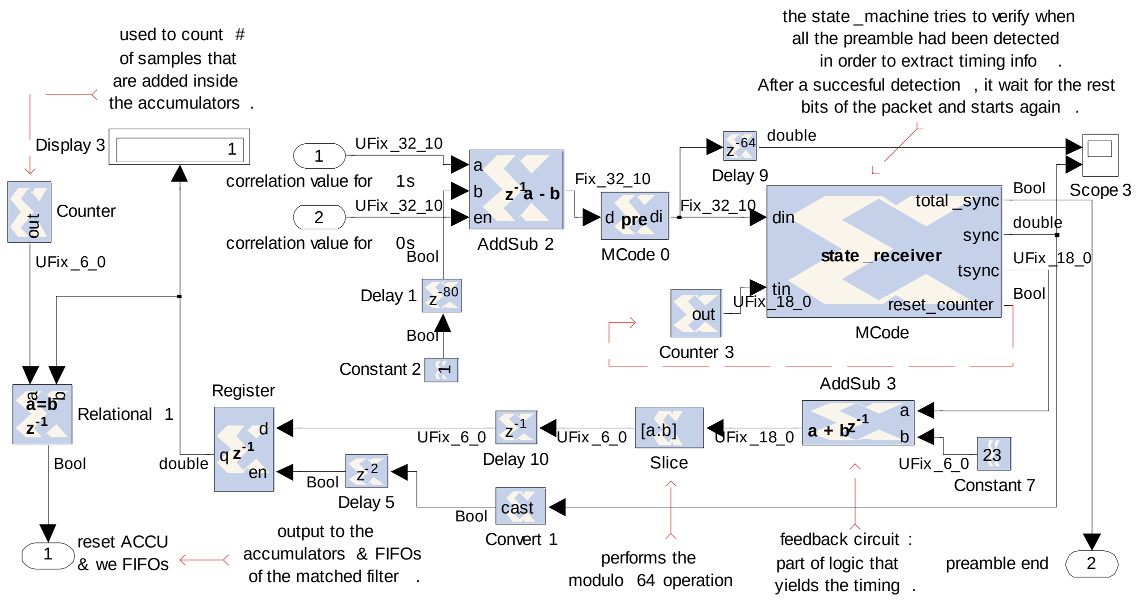
Figure 36 Initialization block.

- **FIR filter, FIR filter1** : custom blocks that have the functionality of an accumulator that adds the last 64 values of its input. A detailed description is given in Section C.2 in Chapter V.
- **Delay1, Delay6** : see description of blocks 'Delay6, Delay11, Delay12, Delay15, Delay16' of Figure 34.

- **Mult8, Mult9**: see description of blocks 'Mult4, Mult5' of Figure 34.
- **AddSub4**: see description of block 'AddSub' of Figure 34.

  Key parameters:
  Precision (Output Type): Full.

H. DECISION CIRCUIT



used to count # of samples that are added inside the accumulators .
Display 3
1
Counter
out
UFix_6_0
Relational 1
a=b
z-1
Bool
reset ACCU & we FIFOs
1
correlation value for 1s
correlation value for 0s
2
UFix_32_10
AddSub 2
z-1 a - b
Fix_32_10
MCode 0
d pre di
Delay 1
z-80
Bool
Constant 2
Register
double
Delay 5
z-2
Convert 1
cast
output to the accumulators & FIFOs of the matched filter .
Delay 10
UFix_6_0
Slice
[a:b]
performs the modulo 64 operation
the state _machine tries to verify when all the preamble had been detected in order to extract timing info . After a succesful detection , it wait for the rest bits of the packet and starts again .
Delay 9
z-64
double
Scope 3
state _receiver
din
tin
total _sync
sync
tsync
reset_counter
MCode
Counter 3
UFix_18_0
AddSub 3
a + b z-1
Constant 7
23
feedback circuit : part of logic that yields the timing .
preamble end
2

Figure 37 Decision Circuit.

- **AddSub2**: block implements the addition or subtraction operation. In this case, the outputs of the correlators are subtracted. In contrast with the 'Relational' block in Figure 33, not only the highest value, but also the exact value is needed. The result is supplied to the following 'MCode' to make decisions about the timing.

  Key parameters:
  Output type: 32 bits with the Binary point in the tenth position. No extra width is granted compared to the previous block.
- **Constant2, Delay1**: Delayed Enable to correct initialization problems.

  Key parameters:
  Explicit period: t. This block and the blocks in the specific subsystem are running at the sample rate.
- **Counter3**: A counter should be thought of as a clock with an adder. Its output can be usually used by comparators to enable or disable signals. In this case, 'MCode' is using the counter to time stamp incidents of interest.

Key parameters:
Counter type: Free Running. As the relative occurrence of the incidents is of interest, any reset in the middle of a preamble acquisition would destroy the synchronization process. A reset implemented in such a way as not to disturb this acquisition is highly recommended.
Output type: Unsigned with 18 bits. A long width was chosen to accommodate the concept of a free running counter and to make any restart unlike.

- **Explicit period**: t. This block and the blocks in the specific subsystem are running at the sample rate. The result of the subtraction of the Correlators' output is examined at the sample rate.

- **MCode0**: see description of block 'MCode1' of Figure 32. Here, 'MCode0' is used to apply some countermeasure against noise. A typical maximum for the input waveform is at a value around 700. This block makes every input value that does not exceed a threshold equal to zero. In this way, small amount of noise will not be perceived as signal by the timing circuit, trying to lock at random noise values. Due to the fact that only positive values can trigger a synchronization phase, only the positive values are suppressed.

Code:

```
function [di] = pre(d)

if d<50 && d>0   %if the value of d does not exceed threshold...
    di =0;       %...suppress output
else
    di =d;       %...otherwise, let input pass.
End
```

- **MCode**: see description of block 'MCode1' of Figure 32. Here, 'MCode' incorporates the logic behind the bit synchronization. It also implements the granular packet synchronization. It includes a state machine where each state represents the next bit of the preamble expected to be received. Explicitly, the zero state is waiting for a reception of a 1, which is the first bit of the preamble. The state one is trying to identify a 0, which is the second bit of the preamble and so on. The state machine goes up to the fifth bit of the preamble and after that it locks the extracted synchronization timing value.

  The input waveform is maximized at the exact moment that all 64 samples of a 1 have been accounted for by the accumulator of the Correlator subsystem. The opposite holds for the 0s, where the waveform is minimized. The 'MCode' tries to match these maxima and minima to the preamble pattern. These maxima and minima also imply the end of a bit. The time that these occur help achieve the synchronization of the receiver.

The decision of the final timing of each packet is based on the mean value of the time of the third and fourth bit. This can change to include more bits.

Code:

```
function [total_sync, sync,  tsync, reset_counter] =
state_receiver(din,tin)

persistent state, state = xl_state(0,{xlUnsigned, 3, 0});
persistent min, min = xl_state(0,{xlSigned, 32, 10});
persistent max, max = xl_state(0,{xlSigned, 32, 10});
persistent tsync1, tsync1 = xl_state(0,{xlUnsigned, 18, 0}); %to store
       % time stamp of the acquisition of the first bit of the preamble
persistent tsync2, tsync2 = xl_state(0,{xlUnsigned, 18, 0});
persistent tsync3, tsync3 = xl_state(0,{xlUnsigned, 18, 0});
persistent tsync4, tsync4 = xl_state(0,{xlUnsigned, 18, 0});
persistent tsync5, tsync5 = xl_state(0,{xlUnsigned, 18, 0});


switch state
    case 0                    %Search for the first bit of the preamble.
        if (tin-tsync1)<64    %For each max value found, search next 64
                              %inputs to ensure no other maximum occurs.
           if din >=max       %If other maximum found, store it...
               max =din;
               tsync1 =tin;   %...and wait again 64 samples to verify
               min =max;      %this is the only maximum for the time.
               tsync2 =tin;
           else
               if din <min     %Otherwise see if it is minimum to
                   min =din;   %initialize correctly state 1.
                   tsync2 =tin;
               end
           end
           state =0;
           sync = 0;
           tsync =0;
        else                %When no other maximum found in the given...
            state =1;       %...time frame, go to next state.
            sync = 1;       %sync high means that this tsync is going
                            %actually to be used to extract timing
                            %information. Otherwise the value of tsync
                            %is ignored.
            tsync =tsync1;  %Give tsync to output.
            max =min;
        end
        total_sync =xfix({xlBoolean},0); %Enabled when the fifth bit of
                                          %the preamble is located
        reset_counter =xfix({xlBoolean},0); %Not allow reset for the
                                          %external counter (not used)
    case 1                  %Search for the second bit of the preamble.
        if (tin-tsync2)<64  %For each min value found, search next 64
                            %inputs to ensure no other minimum occurs.
            if din <=min    %If other minimum found, store it...
                min =din;
                tsync2 =tin;
                max =min;
                tsync3 =tin;
            else
               if din >max    %Otherwise see if it is maximum to
                   max =din;  %initialize correctly state 2.
                   tsync3 =tin;
               end
            end
            state =1;
            sync = 0;
            tsync =tsync1;
        else               %When no other maximum found in the given...
            state =2;      %...time frame, go to next state.
            sync = 0;      %sync low means that this tsync is not going
                        %to be used to extract timing information and
                        %tsync will be ignored.
            tsync =tsync2;%Give tsync to output.
            min =max;
        end
        total_sync =xfix({xlBoolean},0);
        reset_counter =xfix({xlBoolean},0);
    case 2                 %Search for the third bit of the preamble.
        if (tin-tsync3)<64 %and go through the procedure of state 0
            if din >=max
                max =din;
                tsync3 =tin;
                min =max;
                tsync4 =tin;
            else
                if din <min
                   min =din;
                   tsync4 =tin;
                end
            end
            state =2;
            sync = 0;
            tsync =tsync2;
        else
            sync = 1;
            state =3;
            tsync =tsync3;
            max =min;
        end
        total_sync =xfix({xlBoolean},0);
        reset_counter =xfix({xlBoolean},0);
    case 3                 %Search for the forth bit of the preamble.
        if (tin-tsync4)<64 %and go through the procedure of state 1
            if din <=min
                min =din;
                tsync4 =tin;
                max =min;
                tsync5 =tin;
            else
                if din >max
                max =din;
                tsync5 =tin;
            end
        end
        state =3;
        sync = 0;
        tsync =tsync3;
    else
        sync = 1;
        state =4;

        tsync =xfix({xlUnsigned, 18, 0},(tsync3+tsync4)/2); % This
        % criteria was chosen. Different combinations of averaging
        % are also possible.
        min =max;
    end
    total_sync =xfix({xlBoolean},0);
    reset_counter =xfix({xlBoolean},0);
case 4                  %Search for the fifth bit of the preamble.
     if (tin-tsync5)<64 %and go through the procedure of state 0
         if din >=max
             max =din;
             tsync5 =tin;
             %no reset for the next step
         %else
             %no store of min for the next step
         end
         state =4;
         sync = 0;
         tsync =xfix({xlUnsigned, 18, 0},(tsync3+tsync4)/2); % This
         % criteria was chosen. Different combinations of averaging
         % are also possible.
     else
         sync = 0;
         state =5;
         tsync =tsync5;
         max = min;
     end
     total_sync =xfix({xlBoolean},0);
     reset_counter =xfix({xlBoolean},0);
case 5                  %Stay locked waiting for the whole packet
                        %to finish.
    if (tin-tsync5)< 7872-12%The time to complete the reception of
                           %128 bits given that tsync5 corresponds
                           %to the 5th bit of the packet.
      total_sync =xfix({xlBoolean},1); %The preamble (up to fifth
                                       %bit)has been successfully
                                       %located.
      state =5;
      sync =0;          %Lock the timing information.
      reset_counter =xfix({xlBoolean},0);
    else                %Preparation to start over.
        if (tin-tsync5)< 7872+56
            total_sync =xfix({xlBoolean},0);
            reset_counter =xfix({xlBoolean},0);
```

```
                state =5;
                sync =0;
            else              %Start over with the following parameters:
                reset_counter =xfix({xlBoolean},1);
                state =0;
                total_sync =xfix({xlBoolean},0);
                sync = 0;
                max =0;
                min =0;
            end
        end
        tsync =tsync5;
        tsync1 =tin;
    otherwise                 %escape state from unexpected condition.
        state = 0;
        sync = 0;
        tsync =0;
        total_sync =xfix({xlBoolean},0);
        reset_counter =xfix({xlBoolean},0);
end
```

- **Delay9**: see description of blocks ‘Delay7, Delay1’ of Figure 30.
- **AddSub3, Constant7** : There is an inherent delay between the unprocessed samples at the input of the Correlators and the point where the timing decision is taken. To offset this fact a constant value is added to the synchronization time that has been calculated by ‘MCode.’ The exact value of ‘Constant7’ is calculated experimentally.

  Key parameters:
  Operation: addition.
  Constant value: 23. Experimentally calculated value.
- **Slice**: This block extracts from the input only a specified portion of the word and presents it to the output. The operation implemented here is modulo 64. The remainder of the input value when divided by 64 is the six least significant bits. The timing information is supplied to a counter that counts up to 63. This is the reason that the absolute timing information must be translated to time modulo 64.

  Key parameters:
  Width of slice (number of bits): 6.
  Specify range as: Lower bit location +width.
  Relative to: LSB of input.
- **Delay 5**: Delays that ensure the timely propagation of the signals. It is common practice to have to insert delays between adjacent blocks.
- **Delay 10**: delay that had been used for synchronization purposes.

- **<u>Convert1</u>**: Translate the input to a desired output type. Enable and reset inputs can only be driven by Boolean signals. In this case 'sunc' signal is unsigned one bit integer and must be converted to Boolean. This block may not even require resources when mapped to the FPGA, depending on some parameters chosen.

  Key parameters:
  Type (Output precision): Boolean.
- **<u>Register</u>**: is a D flip-flop with latency equal to one sample period. It provides an optional enabled port. When this port is used the register does not accept any new value from its input and continues to have the same output.

  Key parameters:
  Optional Ports: Provide enable port. Here the enable is the 'sync' output of the 'MCode' which means that a desired bit of the preamble has been detected.
- **<u>Counter</u>**: A counter should be thought of as a clock with an adder. Its output can be usually used by comparators to enable or disable signals. In this case, the counter defines a 64 time cycle, within which the accumulators of the NC Matched Filter subsystems must be reset exactly once. The specific instance of the reset is defined by the value stored to the 'Register' and the reset signal is created by the comparison of the value of the counter with the value of the output of the 'Register.'

  Key parameters:
  Counter type: Count Limited.
  Count to value: 63. This defines the 64 time cycle.
  Number of bits: 6. To accommodate counting up to 63.
  Explicit period: 1. As the other components in this subsystem, the 'Counter' works at the sample rate.
- **<u>Relational1</u>**: is a comparator that can support a plethora of different comparisons. Here the comparison is made between the 'Counter' and the output of the 'Register' to define at which exact time instance the accumulators of the NC Matched Filter subsystems must be reset.

I. DECODING SUBSYSTEM

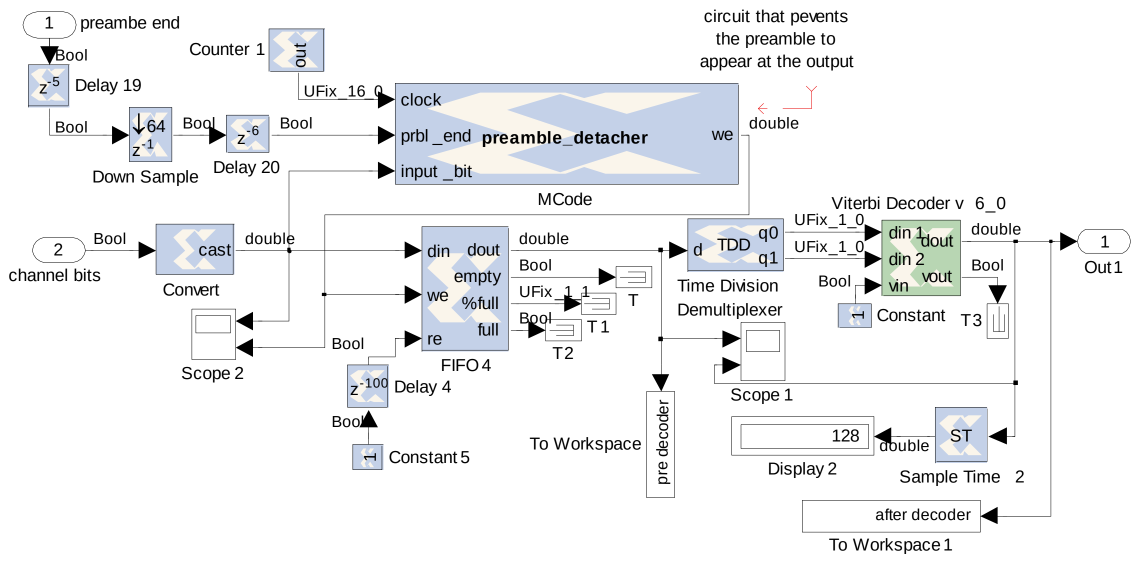


Figure 38 Decoding Subsystem.

- **<u>Delay19, Delay20</u>** : Delays that had been used for synchronization purposes, to synchronize 'preamble end' and 'channel bits' signals. Their values were found experimentally.

  Key parameters:
  Latency: 5 and 6, respectively.
- **<u>Down Sample</u>**: This block reduces the sample rate of the input, discarding the extra values provided in the higher rate input. Here, it matches the sampling rate of 'preamble end' signal with that of 'channel bits.'
- **<u>Counter1</u>**: see description of block 'Counter3' of Figure 37. The only different key parameter is the following:

  Key parameters:
  Explicit period: 64*t. This block and the blocks in the specific subsystem are running at the bit rate. All input signals have been downsampled by a factor of 64.
- **<u>MCode</u>**: see description of block 'MCode1' of Figure 32. Here, 'MCode' makes the fine tuning of the packet synchronization. After receiving 'preamble end' high from the 'Timing Circuit,' it checks the input bits to locate the first 1. This should be the last bit of the preamble.

Code:

```
function we = preamble_detacher(clock,prbl_end,input_bit)

persistent state, state = xl_state(0,{xlUnsigned, 1, 0});
persistent counter, counter = xl_state(0,{xlUnsigned, 16, 0});
persistent delay, delay = xl_state(0,{xlBoolean});

switch state
    case 0                                  %Wait state.
        if prbl_end == xfix({xlBoolean},1);%When timing circuit locates
                                           %the 5th bit of the preamble,
            if delay ==xfix({xlBoolean},0) %let one cycle to pass
                delay =xfix({xlBoolean},1);
            else
                if input_bit == 1       %and search for an input bit =1
                    state = 1;          %to go to the next state.
                    counter =clock;
                else
                    state =0;           %While to find stay at the
                end                     %current state...
            end
        else
            state = 0;
            counter =0;
        end
        we = xfix({xlBoolean},0);       %and output write enable low.
    case 1                              %Packet under reception.
        if clock-counter <120           %Until it counts 120 bits,
            state = 1;                  %stay in the current state.
        else
             if prbl_end == xfix({xlBoolean},1);% Verify that 'preamble
                                                %end' signal when low
                 state =1;
             else
                state = 0;              %and go to the wait state’
                counter =0;
                delay =xfix({xlBoolean},0);     %resetting the flag.
             end
        end
        we = true;
    otherwise                 %escape state from unexpected condition.
        state = 0;
        we = xfix({xlBoolean},1);
        delay =xfix({xlBoolean},0);
end
```

- **Convert**: Translate the input to a desired output type. The output of the ‘Relational’ block is Boolean and must be translated to an unsigned one bit integer in order to drive the next blocks.

- **FIFO4**: It is a First In First Out memory queue. The input values engage the next available memory location in the memory queue. This function is permitted whenever the write enable (we) signal is high; otherwise, the input data is discarded. In this case, ‘we’ is driven by the ‘we’ output of

'MCode.' The read enable (re) signal is a delayed high. This allows the encoded bits to appear at the input of the Modulation Subsystem. Outputs 'empty,' '%full' and 'full' are not used and are terminated by 'T,''T 1,''T 2' blocks.

Key parameters:
Depth: 256.

- **<u>Constant5, Dealy4</u>**: The input sequence of bits is stored but it is read with delay in order to accommodate the preamble bits that are screened from the stored sequence. For a larger number of received packets, the value of 'Delay4' should be increased. Any failure of the delay to account for all the preambles taken away will distort the output sequence by creating copies of the previous bit to the output in order to fulfill the time gaps. Another solution would be to place the decoder with an enable at that point. The enable could be driven by the write enable (we) output of 'MCode' and whenever it was low, it would freeze the Viterbi Decoder until the next enable high.

  Key parameters:
  Constant value: 1.
  Delay value (Latency): 100. This number could be lower for fewer received packets or higher for more received packets.
- **<u>T, T1, T2, T3</u>** : terminate their inputs to avoid warning messages. This means that their inputs are not useful in this design.

- **<u>Time Division Demultiplexer</u>** : This block breaks the input stream to multiple output streams according to the sampling pattern specified. The outputs are downsampled compared to the input.

  Key parameters:
  Frame sampling pattern: [1 1]. Every second input bit is presented to the same output.
  Implementation: Multiple Channel.
- **<u>Viterbi Decoder v 6_0</u>** : This decoder decodes convolutionally encoded data. The block has a green color to emphasize the fact that an extra license is needed to use it. For the purpose of this project, a 90 days free license was granted by the online site of Xilinx. The same parameters used in the Convolutional Encoder block must be specified here as well. Extra capabilities like soft decision decoding and puncturing are offered but were not used.

  Key parameters:
  Constraint length: 3.
  Convolutional code array 1 (octal):[7 5].
  Coding: Hard

- **Constant**: Drives the ‘vin’ of ‘Viterbi Decoder v6_0’ always high.
- **Sample Time2, Display2** : See description of block ‘Sample Time1’ of Figure 28
- **To Workspace, To Workspace1** : see description of block ‘Workspace1’ of Figure 30. These are custom blocks to include both ‘Gateway Out’ and ‘To Workspace’ to fit easier in the design.

  Key parameters:
  Variable name: pre_Viterbi_decoder, after_Viterbi_decoder, respectively.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. MATLAB VERIFICATION CODE

The Matlab code written to verify the decision signal shown in ‘Scope3’ of Figure 37 as the input of the ‘MCode’ block is given in this chapter. The decision circuit is the most critical part of the receiver, thus, a reproduction of the simulation results of the Sysgen was important. This code helped locate the problem related to the Digital Discrete Synthesizers mentioned in Section B.2, Chapter V.

```
clear din
tstep =10*10^-9;           %define the simulation step.
simulation_length =300000; %define the length of the simulation.
f1 =45*10^6;
f2 =40*10^6;
time =0:tstep:simulation_length*tstep;
input =simout.signals.values(1:simulation_length+1)';
h =ones(1,64);
x_sin =input.*(.5*sin(2*pi*f1*time));
x_cos =input.*(.5*cos(2*pi*f1*time));
% x_sin =[xsin.signals.values];% In case the output  of the Sysgen ...
% x_cos =[xcos.signals.values];% after the mixers is used.
x_branch =conv(x_sin,h).^2+conv(x_cos,h).^2;

y_sin =input.*(.5*sin(2*pi*f2*time));
y_cos =input.*(.5*cos(2*pi*f2*time));
% y_sin =[ysin.signals.values];% In case the output  of the Sysgen ...
% y_cos =[ycos.signals.values];% after the mixers is used.
y_branch =conv(y_sin,h).^2+conv(y_cos,h).^2;
din =x_branch-y_branch;
din = (din>50 | din<0).*din;
din =[zeros(75,1); din']; %insert a small delay to match the Sysgen
output.
%din =[zeros(108,1); din'];
figure(1)
subplot(2,1,1),plot(din)
title('Decision Signal ')
xlabel('time(sec)')
ylabel('amplitude')
xlim([0 simulation_length])
state =0;
max =0;min =0;tsync_plot =0;
k =0;%point counter of tsync
kk =0;%point counter of tsync_plot
tsync1 =150;tsync2 =150;tsync3 =150;tsync4 =150;tsync5 =150;%Initialize
tsync =zeros(1,6);
for tin =181:length(din)
switch state
    case 0 % Search for the first bit of the preamble.
        if (tin-tsync1)<64%For each max value found, search next 64
                          %inputs to ensure no other maximum occurs.
            if din(tin) >=max%If other maximum found, store it...
            max =din(tin);
            tsync1 =tin;%...and wait again 64 samples to verify
            min =max;   %this is the only maximum for the time.
            tsync2 =tin;
        else
            if din(tin) <min%Otherwise see if it is minimum to
                min =din(tin);%initialize correctly state 1.
                tsync2 =tin;
            end
        end
        state =0;
        sync = 0;
        tsync(1+k) =0;
     else%When no other maximum found in the given...
         state =1;%...time frame, go to next state.
         sync = 1;%sync high means that this tsync is going
                  %actually to be used to extract timing
                  %information. Otherwise the value of tsync is
                  %ignored.
         tsync_plot(1+kk) =tsync1;
         tsync(1+k) =tsync1; %Give tsync to output.
         max =min;
     end
     total_sync =0;%Enabled when the fifth bit of the
                   %preamble is located
 case 1%Search for the second bit of the preamble.
     if (tin-tsync2)<64%For each min value found, search next 64
                       %inputs to ensure no other minimum occurs.
         if din(tin) <=min%If other minimum found, store it...
             min =din(tin);
             tsync2 =tin;
             max =min;
             tsync3 =tin;
         else
            if din(tin) >max%Otherwise see if it is maximum to
                max =din(tin);%initialize correctly state 2.
                tsync3 =tin;
            end
         end
         state =1;
         sync = 0;
         tsync(2+k) =tsync1;
     else         %When no other maximum found in the given...
         state =2;%...time frame, go to next state.
         sync = 1;%sync high means that this tsync is going
                  %actually to be used to extract timing
                  %information. Otherwise the value of tsync is
                  %ignored.
         tsync_plot(2+kk) =tsync2;
         tsync(2+k) =tsync2;%Give tsync to output.
         min =max;
     end
     total_sync =0;
 case 2%Search for the third bit of the preamble.
     if (tin-tsync3)<64%and go through the procedure of state 0
```

```
        if din(tin) >=max
            max =din(tin);
            tsync3 =tin;
            min =max;
            tsync4 =tin;
        else
            if din(tin) <min
               min =din(tin);
               tsync4 =tin;
            end
        end
        state =2;
        sync = 0;
        tsync(3+k) =tsync2;
    else
        sync = 1;%sync low means that this tsync is not going
                 %to be used to extract timing information and
                 %tsync will be ignored.
        tsync_plot(3+kk) =tsync3;
        state =3;
        tsync(3+k) =tsync3;
        max =min;
    end
    total_sync =0;
case 3%Search for the forth bit of the preamble.
    if (tin-tsync4)<64%and go through the procedure of state 1
        if din(tin) <=min
            min =din(tin);
            tsync4 =tin;
            max =min;
            tsync5 =tin;
        else
            if din(tin) >max
               max =din(tin);
               tsync5 =tin;
           end
        end
        state =3;
        sync = 0;
        tsync(4+k) =tsync3;
    else
        sync = 1;
        state =4;
        %Choose a criterion for timing
        tsync_plot(4+kk) =tsync4; % Store time that the decision is
                                  % taken
        tsync(4+k) =(tsync4+tsync1)/2; % Timing decision
        min =max;
    end
    total_sync =0;
case 4%Search for the fifth bit of the preamble.
     if (tin-tsync5)<64%and go through the procedure of state 0
         if din(tin) >=max
             max =din(tin);
             tsync5 =tin;
```

```
                %no reset for the next step
            %else
                %no store of min for the next step
            end
            state =4;
            sync = 0;
            %The criterion chosen in case 3
            tsync(5+k) =(tsync4+tsync1)/2;
        else
            sync = 1;
            state =5;
            %Choose a criterion for final time
            tsync_plot(5+kk) =tsync5; % Store time that the decision
                                      % is taken
            tsync(5+k) =(tsync3+tsync4)/2; % Timing decision
            max = min;
        end
        total_sync =0;
    case 5               %Stay locked waiting for the whole packet
                         %to finish.
        if (tin-tsync5)< 7872-12%the time to complete the reception of
                            %128 bits given that tsync5 corresponds to
                            %the 5th bit of the packet.
          total_sync =1;%The preamble (up to the fifth
                         %bit)has been successfully located.
          state =5;
          sync = 0; %Lock the timing information.
        else   %Preparation to start over.
            if (tin-tsync5)< 7872+32
                total_sync =0;
                state =5;
                sync =0;
            else %Start over with the following parameters:
                state =0;
                total_sync =0;
                sync = 0;
                max =0;
                min =0;
                k =k+6;%point counter of tsync
                kk =kk+6;%point counter of tsync_plot
                %tsync(6+k) =tsync5;
                %tsync1 =tin;
            end
            tsync(6+k) =tsync5;
            tsync1 =tin;
        end
    otherwise%escape state from unexpected condition.
        state = 0;
        sync = 0;
        tsync(k) =0;
        total_sync =0;
end
end
subplot(2,1,2),plot(tsync_plot,ones(1,length(tsync_plot)),.'')
title('Preamble Detection Signal(tsync)')
```

```
xlabel('time(sec)')
ylabel('amplitude')
xlim([0 simulation_length])
ylim([0 1.5])
tsync_final =mod(tsync+23,64);
figure(2)
plot(tsync_final)
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] J. Mitola, I11, "Software Radios Survey, Critical Evaluation and Future Directions," *IEEE AES Systems Magazine,* April 1993.

[2] Friedrich K. Jondral, "Software-Defined Radio: Basics and Evolution to Cognitive Radio," *EURASIP Journal on Wireless Communications and Networking*, vol. 5, Issue 3, pp. 275 - 283, 2005.

[3] Chris Dick, "A Case for Using FPGAs in SDR PHY," Chief DSP Architect and Director, *Xilinx Inc*., http://www.eetimes.com/story/OEG20020809S0049 (Accessed September 17, 2008).

[4] Bernard Sklar, *Digital Communications: Fundamentals and Applications*, 2nd edition, Prentice Hall, 2001.

[5] Ralph Robertson, Notes for EC3510 (BFSK), Naval Postgraduate School, 2007, (Unpublished).

[6] Michael Rice, *Digital Communications: A Discrete-Time Approach*, Pearson Prentice Hall, 2008, p. 434.

[7] P. Elias, "Coding for Noisy Channels," *IRE Conv. Rec*., 1955, pp. 4:37-47.

[8] Amphion Data Sheet, digchip.com/datasheets/download_datasheet.php?id=240970&part-number=CS3311AA (Accessed September 17, 2008).

[9] A.J. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Trans. Inform. Theory*, IT-13: 260-69, April 1967.

[10] Shu Lin and Daniel J. Costello, *Error Control Coding*, Prentice Hall, 2004.

[11] Xilinx online documentation, ISE Design Suite 10.1 – ISE Foundation, http://www.xilinx.com/publications/prod_mktg/pn0010867.pdf (Accessed September 17, 2008).

[12] Xilinx Online Documentation, System Generator for DSP, Getting Started Guide, Release 10.1.2, June 2008, http://www.xilinx.com/support/documentation/sw_manuals/sysgen_gs.pdf (Accessed September 17, 2008).

[13] Xilinx Online Documentation, System Generator for DSP, Reference Guide, Release 10.1.2, June 2008, http://www.xilinx.com/support/documentation/sw_manuals/sysgen_ref.pdf (Accessed September 17, 2008).

[14] Xilinx Online Documentation, IP Cores Documentation, http://www.xilinx.com/support/documentation/ipcores_docs.htm (Accessed September 17, 2008).

[15] Xilinx Online Documentation, Xilinx ISE Overview, http://toolbox.xilinx.com/docsan/xilinx10/isehelp/isehelp_start.htm (Accessed September 17, 2008).

[16] Avnet, Xilinx® Virtex™-4 LX LC Development Kit, http://www.em.avnet.com/evk/home/0,1719,RID%253D0%2526CID%253D25437%2526CCD%253DUSA%2526SID%253D32214%2526DID%253DDF2%2526LID%253D32232%2526BID%253DDF2%2526CTP%253DEVK,00.html (Accessed September 17, 2008).

[17] Avent Electronics Marketing, http://www.em.avnet.com/ctf_shared/evk/df2df2usa/MemecP160AnalogModule.pdf (Accessed September 17, 2008).

[18] Universitetet i Stavanger, Norway, http://www.ux.uis.no/~karlsk/MIK200/dok/P160Analog_UserGuide_1_2.pdf (Accessed September 17, 2008).

[19] Xilinx Online Documentation, Manuals and Help, http://toolbox.xilinx.com/docsan/xilinx10/books/manuals.pdf (Accessed September 17, 2008).

[20] Xilinx Online Documentation, System Generator for DSP Help Page, http://www.xilinx.com/support/documentation/sw_manuals/sysgen_bklist.pdf. (Accessed September 17, 2008).

[21] Xilinx Online Documentation, IP Release Notes Guide, http://www.xilinx.com/support/documentation/ip_documentation/xtp025.pdf (Accessed September 17, 2008).

[22] Xilinx Online Documentation, DDS Documentation http://www.xilinx.com/support/documentation/ip_documentation/dds_ds558.pdf p. 20, (Accessed September 17, 2008).

[23] Xilinx Online Documentation, Virtex-4 Family Overview http://www.xilinx.com/support/documentation/data_sheets/ds112.pdf (Accessed September 17, 2008).

[24] Dimitris G. Manolakis, Dimitris Manolakis, Vinay K. Ingle, and Stephen M. Kogon, *Statistical and Adaptive Signal Processing*, Artech House, 2005.

[25] Michael Rice, *Digital Communications: A Discrete-Time Approach*, Pearson Prentice Hall, 2008, p. 411.

[26] John G. Proakis, *Digital Communications*, Fourth Edition, McGraw-Hill, 2000.

[27] Simon Haykin and Michael Moher, *Introduction to Analog & Digital Communications*, Wiley, 2nd edition, 2007, pp. 233-244.

[28] Cryptologic Research Laboratory, Research Associate Donna Miller, Code EC, Monterey, CA.

[29] Xilinx, http://www.xilinx.com/company/history.htm (Accessed September 17, 2008).

[30] William S. Carter, "The Dramatic Changes in FPGA Technology," Vice President and Chief Technology Officer, *Xilin Inc*. http://www.techonline.com/learning/course/100043 (Accessed September 17, 2008).

[31] FPGA-guide.com, http://www.fpga-guide.com/technology_frame.html (Accessed September 17, 2008).

[32] Ruđer Bošković Institute, http://www.irb.hr/en/cir/education/courses/fpga/FPGA/fpga_sklopovi/ (Accessed August 9, 2008).

[33] Actel's Presentation, "ProAsic3Actel's 3rd Generation Flash FPGA Family," in NPS Course EC3800, Instructor Peter Ateshian, September 2007.

[34] The 3-D Circuits & Systems Group @ MIT, MIT Webpage, http://mtlweb.mit.edu/researchgroups/icsystems/3dcsg/ (Accessed August 9, 2008).

[35] Gina R. Smith, "The art of FPGA Construction," CEO, *Brown-Smith Research and Development Laboratory Inc*. http://www.embedded.com/design/embeddedfpga/205203954 (Accessed September 17, 2008).

[36] Rodger Hosking and Richard Kuenzler, "Embedding FPGAs in DSP-driven Software Defined Radio Applications," Vice Pres., *Pentec Inc*., http://www.embedded.com/columns/technicalinsights/164302833 (Accessed September 17, 2008).

[37] Paul Ekas, “FPGAs versus DSPs: Effective Implementations of 3G Basestations,” Tech Rep., *Altera Corp*., http://www.eetimes.com/story/OEG20021107S0025. (Accessed September 17, 2008).

[38] Rick Mosher, “FPGA to ASIC Strategy for Communication SoC Designs,” *AMI Semiconductor*, http://www.design-reuse.com/articles/4360/fpga-to-asic-strategy-for-communication-soc-designs.html (Accessed September 17, 2008).

[39] William Wong, “Embedded 32-Bit Cores Hit 1 GHz,” *Electronic Design Magazine,* October 2007, http://electronicdesign.com/Articles/ArticleID/17279/17279.html (Accessed September 17, 2008).

[40[Xilinx, Virtex-5 Multi-Platform FPGA, http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex5/index.htm (Accessed September 17, 2008).

[41] Altera, Stratix III Device Family, http://www.altera.com/products/devices/stratix-fpgas/stratix-iii/st3-index.jsp (Accessed September 17, 2008).

[42] Altera, “Altera Announces Industry’s First 40-nm FPGAs and HardCopy ASICs,” *Press Release*, http://www.altera.com/corporate/news_room/releases/products/nr-stratix-iv-hardcopy-iv.html (Accessed September 17, 2008).

[43] “FPGAs vs. DSPs: A Look at the Unanswered Questions,” BTDI, http://www.dspdesignline.com/howto/196802403;jsessionid=UJ5L5KRC31QFWQSNDLPSKH0CJUNN2JVN?pgno=1 (Accessed September 17, 2008).

[44] Jerry Bickle, “Achieving Optimized Portable Code through SDR MDD Tools,” Tech. Rep., *PrismTech Corporation*, ,http://www.portabledesign.com/article?article_id=38. (Accessed September 17, 2008).

[45] Actel, Cortex-M1 Processor, The ARM® Processor Designed for FPGAs, http://www.actel.com/products/mpu/cortexm1/default.aspx (Accessed August 9, 2008).

[46] Xilinx Inc., http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex5/capabilities/PowerPC_440.htm (Accessed August 9, 2008).

[47] Intel, Online Page, http://www.intel.com/technology/atom/index.htm?iid=tech_micro+atomand (Accessed September 17, 2008).

[48] Via, VIA Nano™ Processor, http://www.via.com.tw/en/products/processors/nano/ (Accessed August 9, 2008).

[49] Texas Instruments, Fixed Point Digital Signal Processor, http://focus.ti.com/docs/prod/folders/print/tms320c6455.html (Accessed August 9, 2008).

[50] Freescale Semiconductor Inc., "MSBA8100: Multi-Standard Baseband Accelerator," http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MSBA8100 (Accessed August 9, 2008).

[51] Carolyn Mathas, "Benchmark Scores Validate Freescale DSP," http://www.networksystemsdesignline.com/showArticle.jhtml;jsessionid=RHQPW2E52ORZ2QSNDLPCKHSCJUNN2JVN?articleID=192203535 (Accessed September 17, 2008).

[52] "FPGAs vs. DSPs: A Look at the Unanswered Questions," *BTDI*, http://www.dspdesignline.com/howto/196802403;jsessionid=UJ5L5KRC31QFWQSNDLPSKH0CJUNN2JVN?pgno=1 (Accessed September 17, 2008).

[53] Douang Phanthavong, "Fast Track to DSP," Product Marketing Engineer, *Mentor Graphics Corporation*, Revised August 2006, http://www.mentor.com/techpapers/fulfillment/upload/mentorpaper_11937.pdf (Accessed September 17, 2008).

[54] Rodger Hosking and Richard Kuenzler, "Embedding FPGAs in DSP-driven Software Defined Radio Applications," Vice Pres., *Pentec Inc*. http://www.embedded.com/columns/technicalinsights/164302833 (Accessed September 17, 2008).

[55] Dave Locke, "Do Legwork before Making ASIC Move," Marketing Manager, *AMI Semiconductor*, http://www.commsdesign.com/showArticle.jhtml;jsessionid=Z2YXAEAYXQ4ASQSNDLPCKH0CJUNN2JVN?articleID=16503876 (Accessed September 17, 2008).

[56] Arun Kottolli, "The Economics of Structured- and Standard-Cell-ASIC Designs," *Technical Solutions Engineer, Open-Silicon*, http://www.edn.com/article/CA6313388.html (Accessed September 17, 2008).

[57] Vaughn Betz, "FPGAs and Structured ASICs Overview & Research Challenges," Director, Software Engineering, *Altera Corp*., www.iic.umanitoba.ca/docs/vaughn-betz.ppt?PHPSESSID=1b34dbb389a16a17339c6dd60acde5c4 (Accessed September 17, 2008).

[58] Hong-Swee Lim, "High Performance DSP Solutions for Ultrasound," Tech. Rep., *Xilinx*, http://www.eetchina.com/STATIC/PDF/200805/EETC-CMET.pdf?SOURCES=DOWNLOAD (Accessed September 17, 2008).

[59] Xilinx Online Page, http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex5/capabilities/dsp48e.htm (Accessed September 17, 2008).

[60] Steve Bush, "Altera FPGAs Get ColdFire Soft Core," Technology Editor, e*lectronicsweekly.com*, http://www.electronicsweekly.com/Articles/2008/06/09/43894/altera-fpgas-get-coldfire-soft-core.htm (Accessed September 17, 2008).

[61] Xilinx Online Page, PowerPc 440 / Virtex 5, http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex5/capabilities/PowerPC_440.htm (Accessed September 17, 2008).

[62] Xiaojun Wang and Nelson, B.E., Field-Programmable Custom Computing Machines, 2003, FCCM 2003, 11th Annual IEEE Symposium on 9-11 April 2003, pp. 195 – 203.

[63] Alex Soohoo, "Do's and Don'ts of Architecting the Right FPGA Solution for DSP Design," Tech. Rep., *Altera*, http://www.pldesignline.com/showArticle.jhtml;jsessionid=DCNXFCN2NQTSIQSNDLPCKHSCJUNN2JVN?articleID=170702837 (Accessed September 17, 2008).

[64] Tom Hill, "Heterogeneous Hardware Platforms Capitalize on DSP/FPGA Capabilities," Tech. Rep., *Xilinx*, http://www.dsp-fpga.com/articles/id/?2900 (Accessed September 17, 2008).

[65] David Lau, Jarrod Blackburn, and Joel A. Seely, "The Use of Hardware Acceleration in SDR Waveforms," Tech Rep., *Altera*, http://www.altera.com.cn/literature/cp/cp_sdr_hardware_acceleration.pdf (Accessed September 17, 2008).

[66] Jeffry Milrod, "Hybrid FPGA/DSP architecture: The Optimal Solution," *President, Bittware, Inc*., http://www.dsp-fpga.com/pdfs/BittWare.RG06.pdf (Accessed September 17, 2008).

[67] Alex Soohoo, "Architecting the right FPGA Solution for Your DSP Design," September 15, 2005, *Embedded.com*, http://www.embedded.com/columns/technicalinsights/170703025 (Accessed September 17, 2008).

[68] Jeffry Milrod, "The Future of High-Performance COTS Signal Processing: Hybrid FPGA/DSP Architecture: The Optimal Solution," *DSP-F_GA.com*, 2006, http://www.dsp-fpga.com/pdfs/BittWare.RG06.pdf (Accessed September 17, 2008).

[69] Texas Instruments, "Texas Instruments' Software Defined Radio Development Platform Makes Rapid Development and Optimization of Multi-Protocol Radios Possible," Press Release, http://focus.ti.com/docs/pr/pressrelease.jhtml?prelId=sc06196 (Accessed September 17, 2008).

[70] David Pellerin and Scott Thibault, *Practical FPGA Programming in C,* Prentice-Hall, Inc., April 2005.

[71] Easic, nextreme2, "Nextreme-2 NEW ASIC Overview," http://www.easic.com/index.php?p=nextreme2-overview (Accessed September 17, 2008).

[72] Easic, Technology Overview, http://www.easic.com/index.php?p=technology (Accessed August 9, 2008).

[73] Easic, nextreme Structured ASIC, http://www.easic.com/pdf/asic/nextreme_asic_structured_asic.pdf (Accessed August 9, 2008).

[74] picoChip, picoGcc Reference Manual, http://www.picochip.com/downloads/picoGcc_reference_manual.pdf (Accessed September 17, 2008).

[75] picoChip, picoArrary Architecture, http://www.picochip.com/products_and_technology/picoarray_architecture (Accessed August 9, 2008).

[76] David Lipets, "Hardware Needs Limit Software Radio," *Tadiran Communications Ltd*, March 7, 2008, http://www.rfdesignline.com/206902442;jsessionid=J4XHRTBWNOPIIQSNDLPCKH0CJUNN2JVN (Accessed September 17, 2008).

[77] Jeffrey H. Reed, *Software Radio, A Modern Approach to Radio Engineering,* Prentice Hall, 2002.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Assistant Professor Frank Kragh
   Naval Postgraduate School
   Monterey, California

4. Instr. Peter Ateshian
   Naval Postgraduate School
   Monterey, California

5. Prof. Roberto Cristi
   Naval Postgraduate School
   Monterey, California

6. Assistant Professor Alexander Julian
   Naval Postgraduate School
   Monterey, California

7. Prof. Herschel Loomis
   Naval Postgraduate School
   Monterey, California

8. Prof. Alan Ross
   Naval Postgraduate School
   Monterey, California

9. Research Associate Donna Miller
   Naval Postgraduate School
   Monterey, California