

SOFTWARE-DEFINED RADIO for ENGINEERS

TRAVIS F. COLLINS ROBIN GETZ DI PU ALEXANDER M. WYGLINSKI

Software-Defined Radio for Engineers

Analog Devices perpetual eBook license – Artech House copyrighted material.

For a listing of recent titles in the *Artech House Mobile Communications*, turn to the back of this book.

Software-Defined Radio for Engineers

Travis F. Collins Robin Getz Di Pu Alexander M. Wyglinski Library of Congress Cataloging-in-Publication Data A catalog record for this book is available from the U.S. Library of Congress.

British Library Cataloguing in Publication Data

A catalog record for this book is available from the British Library.

ISBN-13: 978-1-63081-457-1

Cover design by John Gomes

© 2018 Travis F. Collins, Robin Getz, Di Pu, Alexander M. Wyglinski

All rights reserved. Printed and bound in the United States of America. No part of this book may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without permission in writing from the publisher.

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Artech House cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

 $10 \; 9 \; 8 \; 7 \; 6 \; 5 \; 4 \; 3 \; 2 \; 1$

Dedication

To my wife Lauren —Travis Collins

To my wonderful children, Matthew, Lauren, and Isaac, and my patient wife, Michelle—sorry I have been hiding in the basement working on this book. To all my fantastic colleagues at Analog Devices: Dave, Michael, Lars-Peter, Andrei, Mihai, Travis, Wyatt and many more, without whom Pluto SDR and IIO would not exist.

-Robin Getz

To my lovely son Aidi, my husband Di, and my parents Lingzhen and Xuexun —Di Pu

To my wife Jen —Alexander Wyglinski

Analog Devices perpetual eBook license – Artech House copyrighted material.

Contents

Prefa	ace	xiii
CHA	APTER 1	
Intro	duction to Software-Defined Radio	1
1.1	Brief History	1
1.2	What is a Software-Defined Radio?	1
1.3	Networking and SDR	7
1.4	RF architectures for SDR	10
1.5	Processing architectures for SDR	13
1.6	Software Environments for SDR	15
1.7	Additional readings	17
	References	18
CIL		
Sian	als and Systems	19
21	Time and Frequency Domains	19
2.1	2.1.1 Fourier Transform	20
	2.1.1 Pourier Pransform	20
	2.1.2 Ferioue Pature of the D11	21
22	Sampling Theory	22
2.2	2.2.1 Uniform Sampling	23
	2.2.1 Children Sampling	25
	2.2.2 Trequency Domain Representation of Onitorin building	20
	2.2.5 Typust Sampling Theorem	20
	2.2.5 Sample Rate Conversion	2) 29
2.3	Signal Representation	37
	2.3.1 Frequency Conversion	38
	2.3.2 Imaginary Signals	40
2.4	Signal Metrics and Visualization	41
2	2.4.1 SINAD ENOB SNR THD THD + N and SEDR	42
	2.4.2. Eve Diagram	44
2.5	Receive Techniques for SDR	45
	2.5.1 Nyquist Zones	47
	2.5.2 Fixed Point Quantization	49
2.5	Receive Techniques for SDR 2.5.1 Nyquist Zones 2.5.2 Fixed Point Quantization	45 47 49

vii

	2.5.3 Design Trade-offs for Number of Bits, Cost, Power,	
	and So Forth	55
	2.5.4 Sigma-Delta Analog-Digital Converters	58
2.6	Digital Signal Processing Techniques for SDR	61
	2.6.1 Discrete Convolution	61
	2.6.2 Correlation	65
	2.6.3 Z-Transform	66
	2.6.4 Digital Filtering	69
2.7	Transmit Techniques for SDR	73
	2.7.1 Analog Reconstruction Filters	75
	2.7.2 DACs	76
	2.7.3 Digital Pulse-Shaping Filters	78
	2.7.4 Nyquist Pulse-Shaping Theory	79
	2.7.5 Two Nyquist Pulses	81
2.8	Chapter Summary	85
	References	85

CHAPTER 3

PTODa	ibility in Communications	87
3.1	Modeling Discrete Random Events in Communication Systems	87
	3.1.1 Expectation	89
3.2	Binary Communication Channels and Conditional Probability	92
3.3	Modeling Continuous Random Events in Communication Systems	95
	3.3.1 Cumulative Distribution Functions	99
3.4	Time-Varying Randomness in Communication Systems	101
	3.4.1 Stationarity	104
3.5	Gaussian Noise Channels	106
	3.5.1 Gaussian Processes	108
3.6	Power Spectral Densities and LTI Systems	109
3.7	Narrowband Noise	110
3.8	Application of Random Variables: Indoor Channel Model	113
3.9	Chapter Summary	114
3.10	Additional Readings	114
	References	115

Digita	al Communications Fundamentals	117
4.1	What Is Digital Transmission?	117
	4.1.1 Source Encoding	120
	4.1.2 Channel Encoding	122
4.2	Digital Modulation	127
	4.2.1 Power Efficiency	128
	4.2.2 Pulse Amplitude Modulation	129

	4.2.3 Quadrature Amplitude Modulation	131
	4.2.4 Phase Shift Keying	133
	4.2.5 Power Efficiency Summary	139
4.3	Probability of Bit Error	141
	4.3.1 Error Bounding	145
4.4	Signal Space Concept	148
4.5	Gram-Schmidt Orthogonalization	150
4.6	Optimal Detection	154
	4.6.1 Signal Vector Framework	155
	4.6.2 Decision Rules	158
	4.6.3 Maximum Likelihood Detection in an AWGN Channel	159
4.7	Basic Receiver Realizations	160
	4.7.1 Matched Filter Realization	161
	4.7.2 Correlator Realization	164
4.8	Chapter Summary	166
4.9	Additional Readings	168
	References	169
CHA	APTER 5	
Und	erstanding SDR Hardware	171
5.1	Components of a Communication System	171
	5.1.1 Components of an SDR	172
	5.1.2 AD9363 Details	173
	5.1.3 Zynq Details	176
	5.1.4 Linux Industrial Input/Output Details	177
	5.1.5 MATLAB as an IIO client	178
	5.1.6 Not Just for Learning	180
5.2	Strategies For Development in MATLAB	181
	5.2.1 Radio I/O Basics	181
	5.2.2 Continuous Transmit	183
	5.2.3 Latency and Data Delays	184
	5.2.4 Receive Spectrum	185
	5.2.5 Automatic Gain Control	186
	5.2.6 Common Issues	187
5.3	Example: Loopback with Real Data	187
5.4	Noise Figure	189
	References	190
CHA	APTER 6	
Timi	ng Synchronization	191
6.1	Matched Filtering	191
6.2	Timing Error	195

198

Symbol Timing Compensation

6.3

	6.3.1 Phase-Locked Loops	200
	6.3.2 Feedback Timing Correction	201
6.4	Alternative Error Detectors and System Requirements	208
	6.4.1 Gardner	208
	6.4.2 Müller and Mueller	208
6.5	Putting the Pieces Together	209
6.6	Chapter Summary	212
	References	212
CHA	APTER 7	
Carr	ier Synchronization	213
7.1	Carrier Offsets	213
7.2	Frequency Offset Compensation	216
	7.2.1 Coarse Frequency Correction	217
	7.2.2 Fine Frequency Correction	219
	7.2.3 Performance Analysis	224
	7.2.4 Error Vector Magnitude Measurements	226
7.3	Phase Ambiguity	228
	7.3.1 Code Words	228
	7.3.2 Differential Encoding	229
	7.3.3 Equalizers	229
7.4	Chapter Summary	229
	References	230
CHA	APTER 8	221
Fram	ne Synchronization and Channel Coding	231
8.1	O Frame, Where Art Thou?	231
8.2	Frame Synchronization	232
	8.2.1 Signal Detection	235
0.0	8.2.2 Alternative Sequences	239
8.3	Putting the Pieces Together	241
0.4	8.3.1 Full Recovery with Pluto SDR	242
8.4	Channel Coding	244
	8.4.1 Repetition Coding	244
	8.4.2 Interleaving	243
	8.4.5 Encoding	246
05	6.4.4 DER Calculator	251
8.3	Chapter Summary References	231
	NCICICILCS	231
CHA	APTER 9	
Chai	nnel Estimation and Equalization	253
9.1	You Shall Not Multipath!	253

9.2	Channel Estimation	254
9.3	Equalizers	258
	9.3.1 Nonlinear Equalizers	261
9.4	Receiver Realization	263
9.5	Chapter Summary	265
	References	266
CHA	PTER 10	

Orthogonal Frequency Division Multiplexing		267
10.1	Rationale for MCM: Dispersive Channel Environments	267
10.2	General OFDM Model	269
	10.2.1 Cyclic Extensions	269
10.3	Common OFDM Waveform Structure	271
10.4	Packet Detection	273
10.5	CFO Estimation	275
10.6	Symbol Timing Estimation	279
10.7	Equalization	280
10.8	Bit and Power Allocation	284
10.9	Putting It All Together	285
10.10	Chapter Summary	286
	References	286

CHAPTER 11

Appli	cations for Software-Defined Radio	289
11.1	Cognitive Radio	289
	11.1.1 Bumblebee Behavioral Model	292
	11.1.2 Reinforcement Learning	294
11.2	Vehicular Networking	295
11.3	Chapter Summary	299
	References	299

APPENDIX A

A Lor	A Longer History of Communications	
A.1	History Overview	303
A.2	1750–1850: Industrial Revolution	304
A.3	1850–1945: Technological Revolution	305
A.4	1946–1960: Jet Age and Space Age	309
A.5	1970–1979: Information Age	312
A.6	1980–1989: Digital Revolution	313
A.7	1990–1999: Age of the Public Internet (Web 1.0)	316
A.8	Post-2000: Everything comes together	319
	References	319

APPENDIX B

Getting Started with MATLAB and Simulink		327
B. 1	MATLAB Introduction	327
B.2	Useful MATLAB Tools	327
	B.2.1 Code Analysis and M-Lint Messages	328
	B.2.2 Debugger	329
	B.2.3 Profiler	329
B.3	System Objects	330
	References	332
APP	ENDIX C	
Equa	lizer Derivations	333
C.1	Linear Equalizers	333
C.2	Zero-Forcing Equalizers	335
C.3	Decision Feedback Equalizers	336
APP	ENDIX D	
Trigo	onometric Identities	337
Abou	ut the Authors	339
Inde	x	341

APPENDIX B Getting Started with MATLAB and Simulink

You will be using MATLAB and Simulink for the experiments and for the openended design projects in this book. This appendix serves as a brief refresher of MATLAB, since you should have used it before. However, if you don't have extensive experience with Simulink, then this appendix shows you how to get started with the tool. Please note the MATLAB portion of this appendix is mainly based on the MATLAB documentation presented in [1] and the Simulink portion is based on the Simulink Basics Tutorial presented in [2], but here we extract the most important and fundamental concept so that you can quickly get started after reading this appendix. For more information about these two products, you are encouraged to refer to [1] and [2].

B.1 MATLAB Introduction

MATLAB is widely used in all areas of applied mathematics, in education and research at universities, and in industry. MATLAB stands for Matrix Laboratory and the software is built up around vectors and matrices. Consequently, this makes the software particularly useful for solving problems in linear algebra, but also for solving algebraic and differential equations as well as numerical integration. MATLAB possesses a collection of graphic tools capable of producing advanced GUI and data plots in both 2-D and 3-D. MATLAB also has several toolboxes useful for performing communications, signal processing, image processing, optimization, and other specialized operations.

MathWorks has created an excellent online tutorial to review basic and advanced concepts, as well as provide instructor lead tutorials to show off the various capabilities of MATLAB. It can be found at https://matlabacademy. mathworks.com

B.2 Useful MATLAB Tools

This section introduces general techniques for finding errors, as well as using automatic code analysis functions in order to detect possible areas for improvement within the MATLAB code. In particular, the MATLAB debugger features located within the Editor, as well as equivalent Command Window debugging functions, will be covered. *Debugging* is the process by which you isolate and fix problems with your code. Debugging helps to correct two kinds of errors:

- Syntax errors: For example, misspelling a function name or omitting a parenthesis.
- Run-time errors: These errors are usually algorithmic in nature. For example, you might modify the wrong variable or code a calculation incorrectly. Run-time errors are usually apparent when an M-file produces unexpected results. Run-time errors are difficult to track down because the function's local workspace is lost when the error forces a return to the MATLAB base workspace.

B.2.1 Code Analysis and M-Lint Messages

MATLAB can check your code for problems and recommend modifications to maximize the performance and maintainability through messages, sometimes referred to as M-Lint messages. The term lint is the name given to similar tools used with other programming languages such as C. In MATLAB, the M-Lint tool displays a message for each line of an M-file it determines possesses the potential to be improved. For example, a common M-Lint message is that a variable is defined but never used in the M-file.

You can check for coding problems using three different ways, all of which report the same messages:

- Continuously check code in the Editor while you work. View M-Lint messages and modify your file based on the messages. The messages update automatically and continuously so you can see if your changes addressed the issues noted in the messages. Some messages offer extended information, automatic code correction, or both.
- Run a report for an existing MATLAB code file: From a file in the Editor, select Tools > Code Analyzer > Show Code Analyzer Report.
- Run a report for all files in a folder: In the Current Folder browser, click the Actions button, then select **Reports** > **Code Analyzer Report**.

For each message, review the message and the associated code in order to make changes to the code itself based on the message via the following process:

- Click the line number to open the M-file in the Editor/Debugger at that line.
- Review the M-Lint message in the report and change the code in the M-file based on the message.
- Note that in some cases, you should not make any changes based on the M-Lint messages because the M-Lint messages do not apply to that specific situation. M-Lint does not provide perfect information about every situation.
- Save the M-file. Consider saving the file to a different name if you made significant changes that might introduce errors. Then you can refer to the original file as you resolve problems with the updated file.
- If you are not sure what a message means or what to change in the code as a result, use the Help browser to look for related topics.

You can also get M-Lint messages using the mlint function. For more information about this function, you can type help mlint in the Command Window. Read the online documentation [3] for more information about this tool.

B.2.2 Debugger

The MATLAB Editor, graphical debugger, and MATLAB debugging functions are useful for correcting run-time problems. They enable access to function workspaces and examine or change the values they contain. You can also set and clear *breakpoints*, which are indicators that temporarily halt execution in a file. While stopped at a breakpoint, you can change the workspace contexts, view the function call stack, and execute the lines in a file one by one.

There are two important techniques in debugging: one is the *breakpoint* while the other is the *step*. Setting breakpoints to pause the execution of a function enables you to examine values where you think the problem might be located. While debugging, you can also step through an M-file, pausing at points where you want to examine values.

There are three basic types of breakpoints that you can set in the M-files:

- A standard breakpoint, which stops at a specified line in an M-file.
- A conditional breakpoint, which stops at a specified line in an M-file only under specified conditions.
- An error breakpoint that stops in any M-file when it produces the specified type of warning, error, or NaN or infinite value.

You cannot set breakpoints while MATLAB is busy (e.g., running an M-file, unless that M-file is paused at a breakpoint). While the program is paused, you can view the value of any variable currently in the workspace, thus allowing you to examine values when you want to see whether a line of code has produced the expected result or not. If the result is as expected, continue running or step to the next line. If the result is not as expected, then that line, or a previous line, contains an error.

While debugging, you can change the value of a variable in the current workspace to see if the new value produces expected results. While the program is paused, assign a new value to the variable in the Command Window, Workspace browser, or Array Editor. Then continue running or stepping through the program. If the new value does not produce the expected results, the program has a different or another problem.

Besides using the Editor, which is a graphical user interface, you can also debug MATLAB files by using debugging functions from the Command Window, or you can use both methods interchangeably. Read the online documentation [4] for more information about this tool.

B.2.3 Profiler

Profiling is a way to measure the amount of time a program spends on performing various functions. Using the MATLAB Profiler, you can identify which functions in your code consume the most time. You can then determine why you are calling them and look for ways to minimize their use. It is often helpful to decide whether the number of times a particular function is called is reasonable. Since

programs often have several layers, your code may not explicitly call the most timeconsuming functions. Rather, functions within your code might be calling other time-consuming functions that can be several layers down into the code. In this case, it is important to determine which of your functions are responsible for such calls.

Profiling helps to uncover performance problems that you can solve by

- Avoiding unnecessary computation, which can arise from oversight.
- Changing your algorithm to avoid costly functions.
- Avoiding recomputation by storing results for future use.

When you reach the point where most of the time is spent on calls to a small number of built-in functions, you have probably optimized the code as much as you can expect. You can use any of the following methods to open the Profiler:

- Select **Desktop** → **Profiler** from the MATLAB desktop.
- Select Tools \rightarrow Open Profiler from the menu in the MATLAB Editor/Debugger.
- Select one or more statements in the Command History window, right-click to view the context menu, and choose **Profile Code**.
- Enter the following function in the Command Window: profile viewer.

To profile an M-file or a line of code, follow these steps after you open the Profiler, as shown in Figure B.1:

- 1. In the **Run this code** field in the Profiler, type the statement you want to run.
- 2. Click Start Profiling (or press Enter after typing the statement).
- 3. When profiling is complete, the **Profile Summary** report appears in the Profiler window.

Read the online documentation [5] for more information about this tool.

B.3 System Objects

System objects are a specialization of a class in MATLAB, which define a specific set of methods that make initialization, runtime operation, and tear-down simple.



Figure B.1 The profiler window.

A class itself is basically a set of functions that share a set of variables called parameters. These parameters are defined within the class and have defined scopes. Although many methods are implemented by system objects, the three main methods a user should understand are setupImpl, stepImpl, and releaseImpl. They will be written as

```
1 methods (Access = protected)
2 function setupImpl(obj)
3 % Set parameter
4 obj.x = 1;
5 end
6 end
```

setupImpl is used to initial parameters and perform calculations that are needed for the life of the system object.stepImpl is the runtime function (method) that is called generally multiple times and will consume or produce inputs/outputs. Finally, releaseImpl is used to tear-down the object that will clear its memory or perform closing actions. For example, if the object is to write data to a file it will close the file when this method is called.

When a system object's operator or step method is called, the initial call will actually first call the setupImpl method. At this point the system object is considered locked. Then the stepImpl method will be called. Successive operator or step calls will only call the stepImpl method. To unlock the object the releaseImpl method must be called. Once unlock the setupImpl will again be called at the next operator or step call. We outline this process in a script here:

```
1 % Instantiate object
2 ss = dsp.SignalSource;
3 % setupImpl and stepImpl are called
4 data = ss();
5 % stepImpl is only called
6 data = ss();
7 % Object is unlocked
8 ss.release();
9 % setupImpl and stepImpl are called
10 data = ss.step();
```

System objects are extremely useful for encapsulating a large degree of functionality and when state needs to be maintained. For example, filters require state and are an obvious use for system objects. The toolboxes that make up MATLAB utilize extensions for their system objects that are related to their abbreviation. For example, system objects that are from the Communication Systems Toolbox will have the extension comm, resulting in objects with names such as comm.AGC, comm.CarrierSynchronizer, or comm.EVM. Examples in the DSP Systems Toolbox are: dsp.FIRDecimator, dsp.SpectrumAnalyzer, and dsp.SignalSource. More information about system objects can be found in the MathWorks documentation with extensive details on their implementation and use.

References

- [1] The MathWorks, MATLAB Documentation, http://www.mathworks.com/help/techdoc/.
- [2] University of Michigan, *Simulink Basics Tutorial*, http://www.engin.umich.edu/group/ctm/working/mac/simulink_basics/.
- [3] The MathWorks, *Avoid Mistakes While Editing Code*, http://www.mathworks.com/help/ techdoc/matlab_env/brqxeeu-151.html.
- [4] The MathWorks, *Debugging Process and Features*, http://www.mathworks.com/help/ techdoc/matlab_env/brqxeeu-175.html.
- [5] The MathWorks, *Profiling for Improving Performance*, http://www.mathworks.com/help/ techdoc/matlab_env/f9-17018.html