



mobile communications series

SOFTWARE-DEFINED RADIO for ENGINEERS

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

Software-Defined Radio for Engineers

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

Contents

Preface	xiii
CHAPTER 1	
Introduction 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
CHAPTER 2	
Signals and Systems	19
2.1 Time and Frequency Domains	19
2.1.1 Fourier Transform	20
2.1.2 Periodic Nature of the DFT	21
2.1.3 Fast Fourier Transform	22
2.2 Sampling Theory	23
2.2.1 Uniform Sampling	23
2.2.2 Frequency Domain Representation of Uniform Sampling	25
2.2.3 Nyquist Sampling Theorem	26
2.2.4 Nyquist Zones	29
2.2.5 Sample Rate Conversion	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
2.4.1 SINAD, ENOB, SNR, THD, THD + N, and SFDR	42
2.4.2 Eye Diagram	44
2.5 Receive Techniques for SDR	45
2.5.1 Nyquist Zones	47
2.5.2 Fixed Point Quantization	49

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		
	Probability 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
CHAPTER 4		
	Digital 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

CHAPTER 5

	Understanding 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

CHAPTER 6

	Timing Synchronization	191
6.1	Matched Filtering	191
6.2	Timing Error	195
6.3	Symbol Timing Compensation	198

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
CHAPTER 7		
	Carrier 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
CHAPTER 8		
	Frame Synchronization and Channel Coding	231
8.1	O Frame, Where Art Thou?	231
8.2	Frame Synchronization	232
8.2.1	Signal Detection	235
8.2.2	Alternative Sequences	239
8.3	Putting the Pieces Together	241
8.3.1	Full Recovery with Pluto SDR	242
8.4	Channel Coding	244
8.4.1	Repetition Coding	244
8.4.2	Interleaving	245
8.4.3	Encoding	246
8.4.4	BER Calculator	251
8.5	Chapter Summary	251
	References	251
CHAPTER 9		
	Channel 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
CHAPTER 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		
	Applications 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 Longer History of Communications	303
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

APPENDIX C

Equalizer Derivations	333
C.1 Linear Equalizers	333
C.2 Zero-Forcing Equalizers	335
C.3 Decision Feedback Equalizers	336

APPENDIX D

Trigonometric Identities	337
About the Authors	339
Index	341

Introduction to Software-Defined Radio

Various forms of communication have evolved over the millennia. The spoken word can be transmitted from one person, and heard or received by another. In modern times town criers hold an annual contest to discover who can shout a comprehensible message over the greatest distance [1]. However, while the world record is for loudest crier is 112.8 decibels, it can only be understood at less than 100 meters. The desire to communicate more effectively than shouting, is old as speech itself.

With modern advances in computing technologies, digital signal processing and digital communication algorithms, artificial intelligence, radio frequency (RF) hardware design, networking topologies, and many other elements have evolved modern communication systems into complex, intelligent, high-performance platforms that can adapt to operational environments and deliver large amounts of information in real-time, error-free. The latest step in communication systems technology is the software-defined radio, or SDR, which adopts the most recent advances in all fields to yield the ultimate transmitter and receiver.

1.1 Brief History

Given the exciting history associated with advances that directly impact SDR technology, Figure 1.1 provides a timeline describing several significant milestones over the past four centuries. This history is dominated by various people investigating ideas or concepts, publishing the results, then allowing their peers and colleagues to build on their work. Many turned their work into commercial products and became famous and rich; some became neither. For an exhaustive list of major milestones relevant to SDR technology, the interested reader is referred to Appendix A.

1.2 What is a Software-Defined Radio?

Every professional organization attempts to define a common framework of terms and definitions to allow easy communication between professionals who are working on similar areas of research or product development. Wireless communications and SDR is no different. The Institute of Electrical and Electronic Engineers (IEEE) P1900.1 Working Group has created the following definitions to

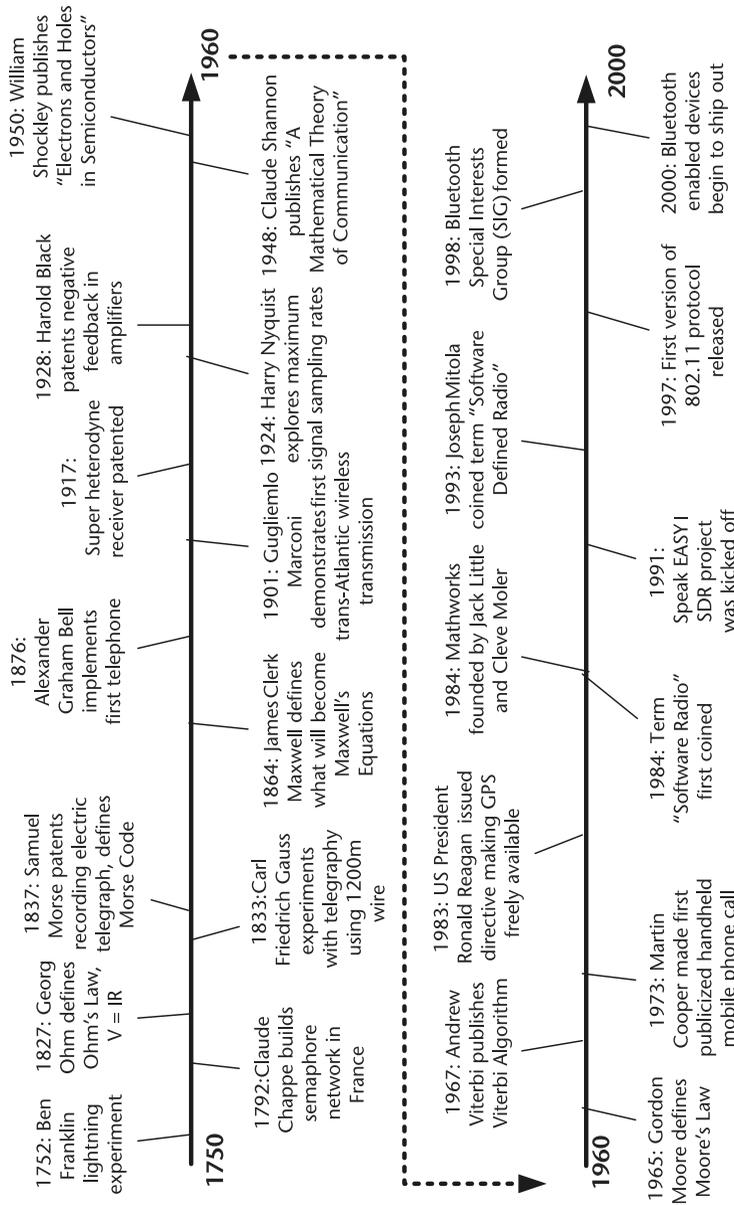


Figure 1.1 Timeline of several key milestones in communications.

ensure that everyone in the field has common terminology [2]:

Radio

1. Technology for wirelessly transmitting or receiving electromagnetic radiation to facilitate transfer of information.
2. System or device incorporating technology as defined in (1).
3. A general term applied to the use of radio waves.

Radio Node

A radio point of presence incorporating a radio transmitter or receiver.

Software

Modifiable instructions executed by a programmable processing device.

Physical Layer

The layer within the wireless protocol in which processing of RF, IF, or baseband signals including channel coding occurs. It is the lowest layer of the ISO 7-layer model as adapted for wireless transmission and reception.

Data Link Layer

The protocol responsible for reliable frame transmission over a wireless link through the employment of proper error detection and control procedures and medium access control.

Software Controlled

Software controlled refers to the use of software processing within the radio system or device to select the parameters of operation.

Software Defined

Software defined refers to the use of software processing within the radio system or device to implement operating (but not control) functions.

Software Controlled Radio

Radio in which some or all of the physical layer functions are software controlled.

Software-Defined Radio (SDR)

Radio in which some or all of the physical layer functions are software defined.

Transmitter

Apparatus producing radio frequency energy for the purpose of radio communication.

Receiver

A device that accepts a radio signal and delivers information extracted from it.

Air Interface

The subset of waveform functions designed to establish communication between two radio terminals. This is the waveform equivalent of the wireless physical layer and the wireless data link layer.

Waveform

1. The set of transformations applied to information to be transmitted and the corresponding set of transformations to convert received signals back to their information content.
2. Representation of a signal in space.
3. The representation of transmitted RF signal plus optional additional radio functions up to and including all network layers.

The combination of digital processing and analog RF has always made up communication systems. In today's modern systems signal processing has progressed to such an extent that a majority of baseband functionality is being implemented in software. The flexibility of the RF hardware to be re purposed and reconfigured has led to one radio front-end handling most RF systems. Normally the RF front-end is software controlled rather than software defined. This modern combination of flexible RF front-ends and signal processing in software has led the birth of software-defined radio.

This can be seen in devices like Analog Devices's AD7030, shown in Figure 1.2. The ADF7030 is a low-power, high-performance, integrated radio transceiver supporting narrow band operation in the 169.4-MHz to 169.6-MHz ISM band. It supports transmit and receive operation at 2.4 kbps and 4.8 kbps using 2GFSK modulation and transmit operation at 6.4 kbps using 4GFSK modulation. It includes an on-chip ARM Cortex-M0 processor that performs radio control and calibration as well as packet management. That and a sensor is all that is needed for smart metering or active tag asset tracking applications. This is just a side effect of Moore's law—system-level integration.

An SDR system is a complex device that performs several complicated tasks simultaneously in order to enable the seamless transmission and reception of data. In general, a digital communications system consists of an interdependent sequence of operations responsible for taking some type of information, whether it is human speech, music, or video images, and transmits it over-the-air to a receiver for processing and decoding into a reconstructed version of the original information signal. If the original information is analog (like audio), it must first be digitized using techniques such as quantization in order for us to obtain a binary representation of this information. Once in a binary format, the transmitter digitally

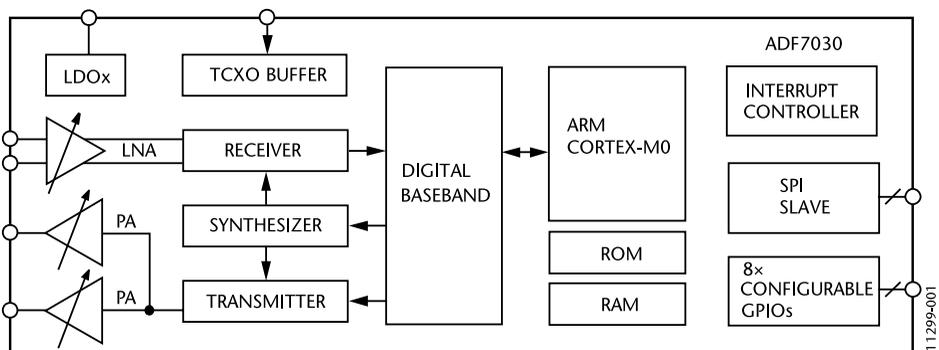


Figure 1.2 ADF7030 block diagram [3].

processes this information and converts it into an electromagnetic sinusoidal waveform that is uniquely defined by its physical characteristics, such as its signal amplitude, carrier frequency, and phase. At the other end of the communications link, the receiver is tasked with correctly identifying the physical characteristics of the intercepted modulated waveform transmitted across a potentially noisy and distortion-filled channel, and ultimately returning the intercepted signal back into the correct binary representation. The basic building blocks of a digital communication system is shown in Figure 1.3.

Figure 1.3 shows that the input to the transmitter and output of the receiver originate from or are fed into a *digital source* and *digital sink*, respectively. These two blocks represent the source and destination of the digital information to be communicated between the transmitter and receiver. Once the binary information is introduced to the transmitter, the first task performed is to remove all redundant/repeating binary patterns from the information in order to increase the efficiency of the transmission. This is accomplished using the *source encoder* block, which is designed to strip out all redundancy from the information. Note that at the receiver, the *source decoder* re-introduces the redundancy in order to return the binary information back to its original form. Once the redundancy has been removed from the binary information at the transmitter, a *channel encoder* is employed to introduced a controlled amount of redundancy to the information stream in order to protect it from potential errors introduced during the transmission process across a noisy channel. A *channel decoder* is used to remove this controlled redundancy and return the binary information back to its original form. The next step at the transmitter is to convert the binary information into unique electromagnetic waveform properties such as amplitude, carrier frequency, and phase. This is accomplished using a mapping process called *modulation*. Similarly, at the receiver the *demodulation* process converts the electromagnetic waveform back into its respective binary representation. Finally, the discrete samples outputted

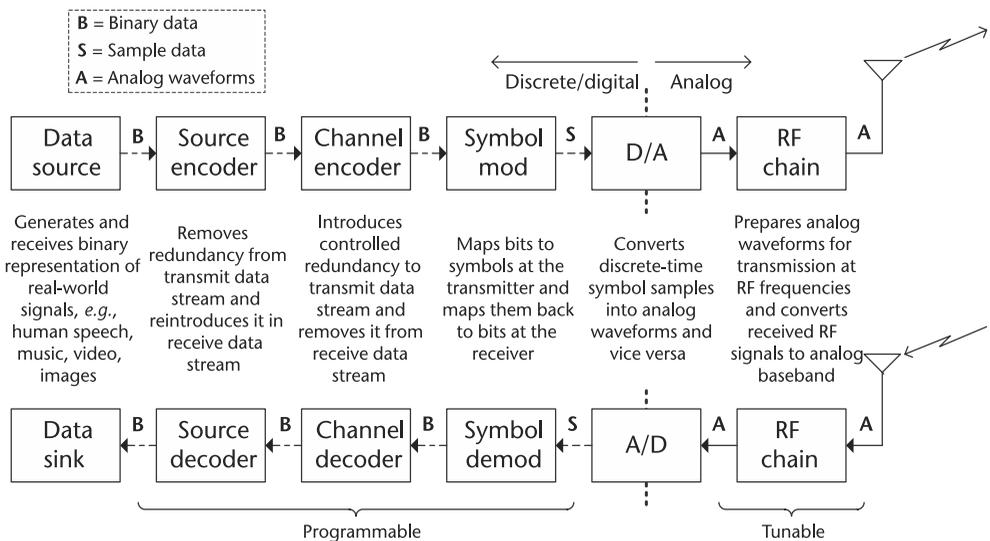


Figure 1.3 An illustration describing some of the important components that constitute a modern digital communications system. Note that for a SDR-based implementation, those components indicated as programmable can be realized in either programmable logic or software.

by the modulation block are resampled and converted into a baseband analog waveform using a *digital-to-analog converter* (DAC) before being processed by the radio frequency (RF) front-end of the communication system and upconverted to an RF carrier frequency. At the receiver, the reverse operation is performed, where the intercepted analog signal is downconverted by the RFFE to a baseband frequency before being sampled and processed by an *analog-to-digital converter* (ADC).

Given the complexity of an SDR platform and its respective components, it is important to understand the limitations of a specific SDR platform and how various design decisions may impact the performance of the resulting prototype. For instance, it is very desirable to have real-time baseband processing for spectrum sensing and agile transmission operations with high computational throughput and low latency. However, if the microprocessor being employed by the SDR platform is not sufficiently powerful enough in order to support the computational operations of the digital communication system, one needs to reconsider either the overall transceiver design or the requirements for low latency and high throughput. Otherwise, the SDR implementation will fail to operate properly, yielding transmission errors and poor communication performance.

Design considerations to think about when devising digital communication systems based on an SDR platform include.

- The integration of the physical and network layers via a real-time protocol implementation on an embedded processor. Note that most communication systems are divided into logically separated layers in order to more readily facilitate the design of the communication system (see Section 1.3). However, it is imperative that each layer is properly designed due to the strong interdependence between all the layers.
- Ensuring that a sufficiently wide bandwidth radio front-end exists with agility over multiple subchannels and scalable number of antennas for spatial processing. Given how many of the advanced communication system designs involve the use of multiple antennas and wideband transmissions, it is important to know what the SDR hardware is capable of doing with respect to these physical attributes.
- Many networks employing digital communication systems possess a centralized architecture for controlling the operations of the overall network (e.g., control channel implementation). Knowing the radio network architecture is important since it will dictate what sort of operations are essential for one digital transceiver to communicate with another.
- The ability to perform controlled experiments in different environments (e.g., shadowing and multipath, indoor and outdoor environments) is important for the sake of demonstrating the reliability of a particular SDR implementation. In other words, if an experiment involving an SDR prototype system is conducted twice in a row in the exact same environment and using the exact same operating parameters, it is expected that the resulting output and performance should be the same. Consequently, being able to perform controlled experiments provides the SDR designer with a sanity check capability.

- Reconfigurability and fast prototyping through a software design flow for algorithm and protocol description.

Instead of using fixed analog processing and fixed circuits, many of the communication systems in use every day are being implemented using microelectronic-based flexible IC, digital signal processors, programmable digital logic, accelerators, and other types of computing engines. To take advantage of new advances in processing engines, high-level languages such as MATLAB® and Simulink are being used rather than C or assembly. This transition of computing technology had the impact of enabling new communication functionalities and capabilities, such as advanced satellite communications, mobile communications, data modems, and digital television broadcasts.

1.3 Networking and SDR

With the evolution of digital communication system into highly complex devices, it became apparent that a divide-and-conquer strategy was needed in order to make the design and implementation of such systems feasible and manageable. Consequently, researchers divided a digital communication system into a collection of complementary layers, with each layer performing a specific function as part of the overall process of transmitting and receiving information. As a result of this divide-and-conquer strategy, communication systems rapidly evolved into highly capable platforms performing a wide range of operations, such as Web surfing and email to streaming multimedia content. In fact, this strategy of dividing up the operations of a communication system into layers was so successful that there are entire research communities that only focus on one of the layers and none of the others; they take for granted the information coming from the layers above and below their layer.

In general, there are two models for dividing up a communication system into layers: the Open System Interconnection (OSI) 7-layer model and the Transmission Control Protocol (TCP)/Internet Protocol (IP) 5-layer model, as shown in Figure 1.4. Both models possess approximately the same functionality, but the TCP/IP model amalgamates the top several layers into a single one. Focusing on the TCP/IP 5-layer model, this consists of the following layers, from top to bottom:

- *Application Layer*: Interfaces user with the data from the communication system. For instance, the application layer would include data originating from or intended for software running Web browsers, email clients, and streaming media interfaces. These applications are usually addressed via designated socket.
- *Transport Layer*: Responsible for transporting application layer messages between the client application and server application. This layer ensures reliable data transmission.
- *Network Layer*: Responsible for moving network layer packets from one host to another host. Defines format of datagrams and how end systems and routers act on datagram, as well as determine routes that datagrams take between sources and destinations.

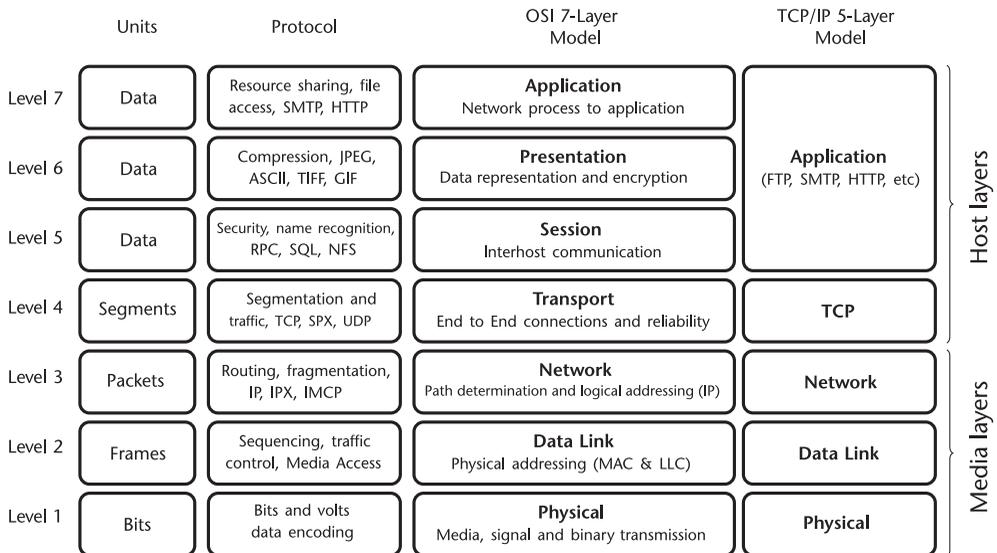


Figure 1.4 Seven-layer OSI model compared to five-layer TCP/IP model.

- *Link Layer*: Handles problem of exchanging data between two or more directly connected devices. Reliability: This includes error detection and error correction as well as addressing of different communication systems.
- *Physical Layer*: Sends individual bits from one communication system directly to another communication system. It also covers the physical interface between data transmission device and transmission medium.

From the perspective of a radio system and its implementation, much of the system design will focus on the physical layer (as does this text), but it can't be forgotten how the link layer may affect the physical layer. Nevertheless, given that the baseband processing is all conducted in software, it is possible for the communications system to implement the higher layers of the stack in software as well. Many communication standards have adopted this scheme, where the entire communication system across all the layers are implemented in software, although depending on data rate requirements, this can require significant computational capabilities on the part of the system to achieve real-time operation. All software implementations enable functional updates without hardware replacement. In practice, this is normally only done on emerging standards or where data rates are relatively low. However, imagine applying a software upgrade to a Wi-Fi router and being able to implement the next standard without having to replace the hardware. This software upgradeable system would be more complex, and might cost more than a fixed hardware system, but would consumers be willing to pay more? History indicates no. For those types of high-volume consumer applications, many times price point is the most critical item to product success. Most end consumers do not think about long-term maintenance and total cost of ownership while looking at the variety of products on Amazon. The trade-offs of which function or layer is done in fixed hardware versus flexible software is an engineering decision based on volume, cost, power, complexity, and many other factors.

There has been a growing amount of interest with respect to combining SDR technology with software-defined networks (SDNs), where the latter focuses on adapting the higher communication layers to the prevailing operational environment. This enables things like modification of the routing to be tied to heuristics provided by the physical layers. Self-healing mesh networks are an implementation of this.

The link layer will also affect the physical (PHY) layer of a wireless communication system as shown in Figure 1.5. For example, in 802.11 (Wi-Fi), the PHY layer (layer 1) is actually broken further down into the Physical Layer Convergence Protocol (PLCP) and the Physical Medium Dependent (PMD) sublayer. The PMD sublayer provides transmission and reception of physical layer data units between two stations via the wireless medium, and passes this to the PLCP, which interfaces to the upper MAC layers, various management layer entities, and generic management primitives to maximize data rates.

At the PHY layer the unit denoted in Figure 1.4 is bits; however, across the wireless and wired links this data will be encoded in more analog-friendly forms designed for transmission called symbols. The preamble, denoted in Layer 1 in Figure 1.5 will most likely never be demodulated at the receiver from a symbol form. Such sequences are only used by the PHY layer to compensate for nonidealities in a link, and have little to no meaning to the above layers. However, for those implementing with SDRs these simple sections are the main focus, and the remaining portions of the frame are arbitrary data.

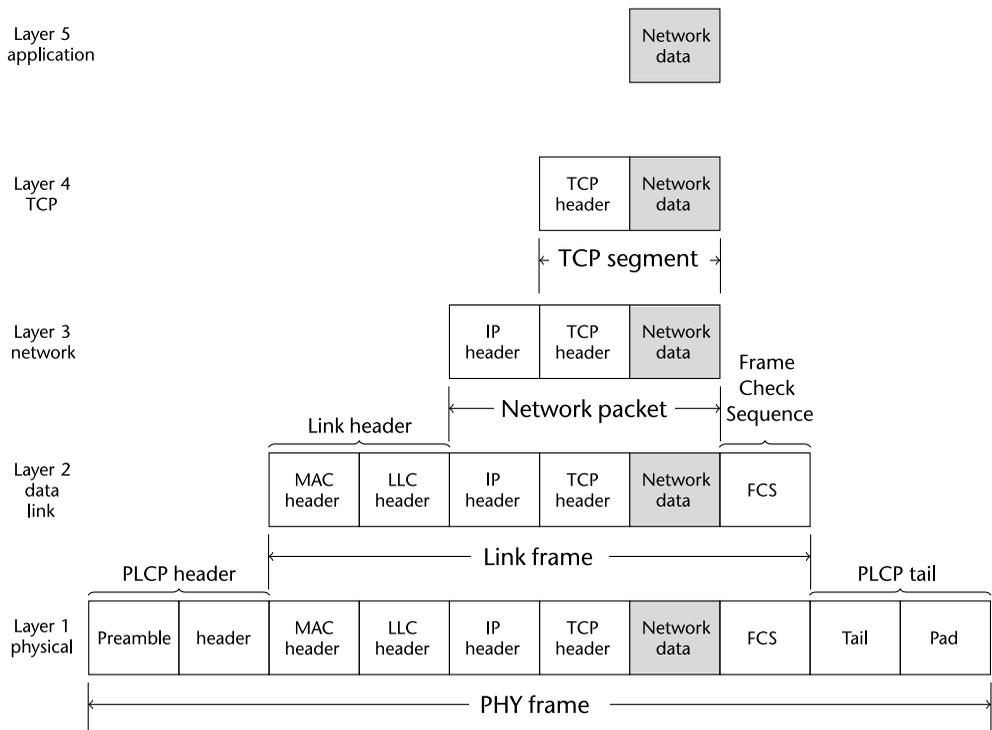


Figure 1.5 Packet structure effects SDR, PLCP = Physical Layer Convergence Protocol.

1.4 RF architectures for SDR

Next-generation communications systems introduce new challenges that require solutions beyond what can be achieved through individual device optimization. Integrating more software control and cognitive abilities to the radio demands a more frequency- and bandwidth-flexible RF design. To achieve this goal static filters need to be removed and replaced with tunable filters. Similarly, the concept of a common platform would allow for shorter development times, reduced manufacturing costs, and provide greater interoperability between systems. The common platform demands that the RF system be capable of providing full performance for applications that traditionally had very different architectures. Finally, future platforms are pushing size and power demands to a new extreme.

Handheld radios are becoming more capable and complex, but simultaneously requiring improved battery efficiency. Small UAVs lack the power generation of large aircraft and every milliwatt that the RF system consumes directly translates to payload battery weight, and thus, reduced flight time. To overcome these challenges and create the next generation of aerospace and defense solutions, a new radio architectures are being developed.

Since its inception, the superheterodyne architecture has been the backbone of radio design. Whether it is a handheld radio, unmanned aerial vehicle (UAV) data link, or a signal intelligence receiver, the single or dual mixing stage superheterodyne architecture is the common choice (see Figure 1.6). The benefits of this design are clear: proper frequency planning can allow for very low spurious emissions, the channel bandwidth and selectivity can be set by the intermediate frequency (IF) filters, and the gain distribution across the stages allows for a trade-off between optimizing the noise figure and linearity.

For over 100 years of use (see the appendix for more information), there have been significant gains in performance for the superheterodyne across the entire signal chain. Microwave and RF devices have improved their performance while decreasing power consumption. ADCs and DACs have increased the sample rate, linearity, and effective number of bits (ENOB). Processing capability in FPGAs and DSPs has followed Moore's law and increased with time, allowing for more efficient algorithms, digital correction, and further integration. Package technology has shrunk device pin density while simultaneously improving thermal handling.

However, these device-specific improvements are beginning to reach the point of diminishing returns. While the RF components have followed a reduced size, weight, and power (SWaP) trend, high-performance filters remain physically large and are often custom designs, adding to overall system cost. Additionally, the IF filters set the analog channel bandwidth of the platform, making it difficult to create a common platform design that can be reused across a wide range of systems. For package technology, most manufacturing lines will not go below a 0.65-mm or 0.8-mm ball pitch, meaning there is a limit on how physically small a complex device with many I/O requirements can become.

An alternative to the superheterodyne architecture, which has reemerged as a potential solution in recent years, is the zero-IF (ZIF) architecture. A ZIF receiver (see Figure 1.7) utilizes a single frequency mixing stage with the local oscillator (LO) set directly to the frequency band of interest, translating the received signal down

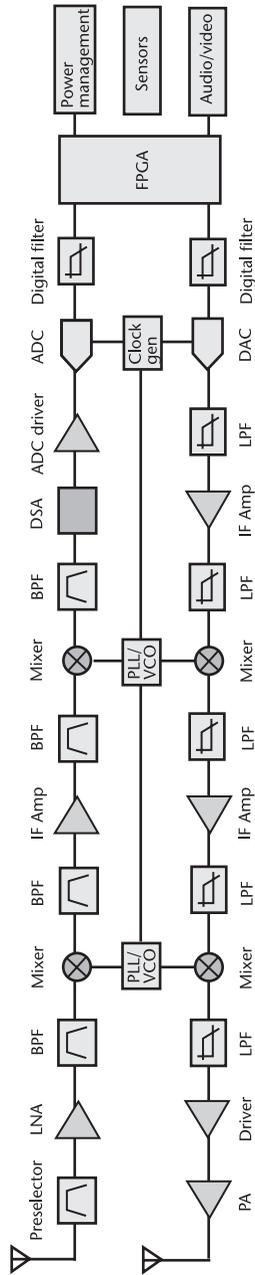


Figure 1.6 Multistage superheterodyne receive and transmit signal chains [4].

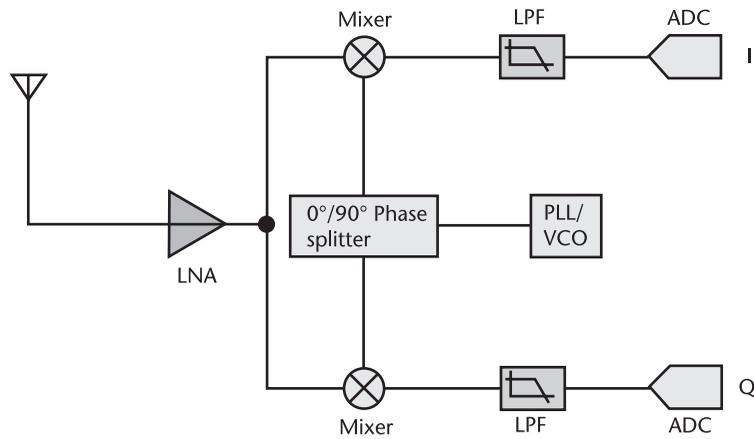


Figure 1.7 Zero IF architecture [4].

to baseband in phase (I) and quadrature (Q) signals. This architecture alleviates the stringent filtering requirements of the superheterodyne since all analog filtering takes place at baseband, where filters are much easier to design and less expensive than custom RF/IF filters. The ADC and DAC are now operating on I/Q data at baseband, so the sample rate relative to the converted bandwidth can be reduced, saving significant power. For many design aspects, ZIF transceivers provide significant SWaP reduction as a result of reduced analog front-end complexity and component count.

This direct frequency conversion to baseband can introduce the possibility of carrier leakage and an image frequency component. Due to real-world factors, such as process variation and temperature deltas in the signal chain, it is impossible to maintain a perfect 90° phase offset between the I and Q signals, resulting in degraded image rejection. Additionally, imperfect LO isolation in the mixing stage introduces carrier leakage components. When left uncorrected, the image and carrier leakage can degrade a receiver's sensitivity and create undesirable transmit spectral emissions.

Historically, the I/Q imbalance has limited the range of applications that were appropriate for the ZIF architecture. This was due to two reasons: first, a discrete implementation of the ZIF architecture will suffer from mismatches both in the monolithic devices and also in the printed circuit board (PCB). In addition to this, the monolithic devices could pull from different fabrication lots, making exact matching very difficult due to native process variation. A discrete implementation will also have the processor physically separated from the RF components, making a quadrature correction algorithm very difficult to implement across frequency, temperature, and bandwidth.

Moore's law, or integration of the ZIF architecture into a monolithic transceiver device provides the path forward for next-generation systems. By having the analog and RF signal chain on a single piece of silicon, process variation will be kept to a minimum. Digital signal processing (DSP) blocks can be incorporated into the transceiver, removing the boundary between the quadrature calibration algorithm and the signal chain. This approach provides both unparalleled improvements in SWaP and can also match the superheterodyne architecture for performance specifications.

Devices like the Pluto SDR shown in Figure 1.8 integrate the full RF, analog, and digital signal chain onto a single CMOS device, and include digital processing to run quadrature and carrier leakage correction in real time across all process, frequency, and temperature variations. Devices like the AD9361 focuses on medium-performance specifications and very low power, such as UAV data links, handheld communication systems, and small form factor SDR applications. The AD9371 is optimized for high-performance specifications and medium power. Additionally, this device has refined calibration control, as well as an observation receiver for power amplifier (PA) linearization and a sniffer receiver for white space detection. This opens up new design potential for a different suite of applications. Communication platforms using wideband waveforms or occupying noncontiguous spectrum can now be implemented in a much smaller form factor.

1.5 Processing architectures for SDR

The microelectronic industry has rapidly evolved over the past six decades, resulting in numerous advances in microprocessor systems that have enabled many of the applications we take for granted every day. The rate at which this evolution has progressed over time has been characterized by the well-known Moore’s Law, which defines the long-term trend of the number of transistors that can be accommodated on an integrated circuit. In particular, Moore’s law dictates that the number of transistors per integrated circuit approximately doubles every 2 years, which subsequently affects the performance of microprocessor systems such as

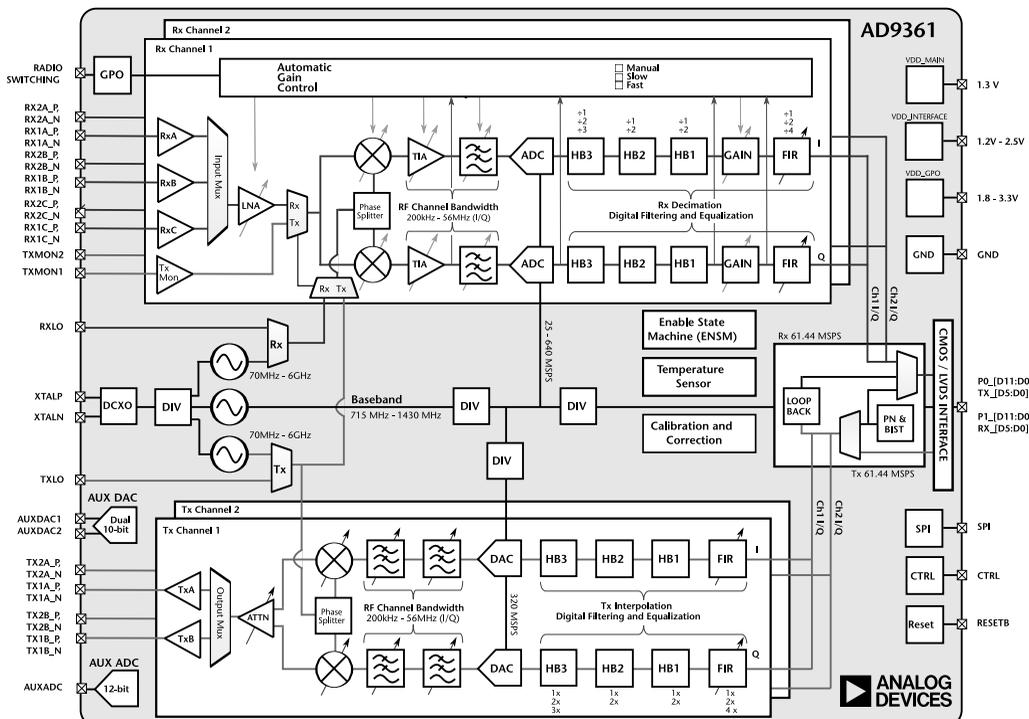
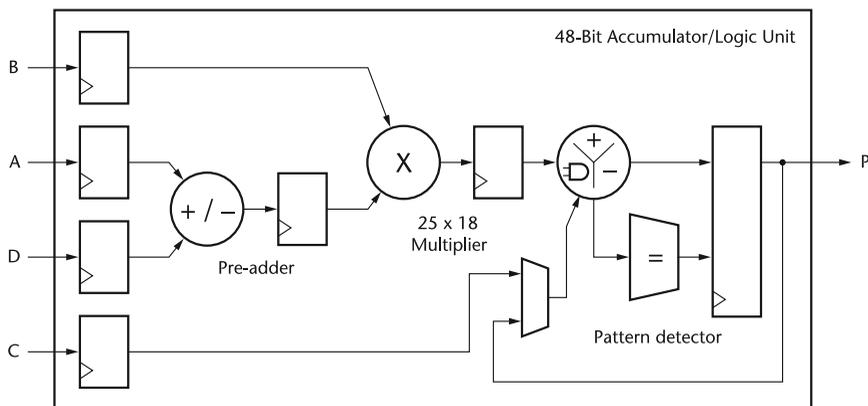


Figure 1.8 Integrated ZIF architecture used in the Pluto SDR.

processing speed and memory. One area that the microelectronics industry has significantly influenced over the past half century is the digital communication systems sector, where microprocessor systems have been increasingly employed in the implementation of digital transceiver, yielding more versatile, powerful, and portable communication system platforms capable of performing a growing number of advance operations and functions. With the latest advances in microelectronics and microprocessor systems, this has given rise to software-defined radio (SDR) technology, where baseband radio functionality can be entirely implemented in digital logic and software, as illustrated in Figure 1.3. There are several different types of microprocessor systems for SDR implementations, including.

- *General-purpose microprocessors* are often used in SDR implementations and prototypes due to their high level of flexibility with respect to reconfigurability, as well as due to their ease of implementation regarding new designs. On the other hand, general-purpose microprocessors are not specialized for mathematical computations and they can be potentially power inefficient.
- *Digital signal processors (DSPs)* are specialized for performing mathematical computations, implementation of new digital communication modules can be performed with relative ease, and the processor is relatively power efficient (e.g., DSPs are used in cellular telephones). On the other hand, DSPs are not well suited for computationally intensive processes and can be rather slow.
- *Field programmable gate arrays (FPGAs)* are efficient for custom digital signal processing applications because they can implement custom, fully parallel algorithms. DSP applications use many binary multipliers and accumulators that can be implemented in dedicated DSP slices, as shown in Figure 1.9. This includes 25×18 twos-complement multiplier, a 48-bit accumulator, a power-saving preadder, single-instruction, multiple data (SIMD) arithmetic unit, which includes a dual 24-bit or quad 12-bit add/subtract/accumulate. Tools like MathWorks HDL Coder are making creating new modules and targeting FPGAs easier, as it can generate portable, synthesizable Verilog and VHDL code from MATLAB functions, Simulink models, and Stateflow



UG479_c1_21_032111

Figure 1.9 Basic DSP48E1 slice functionality [5].

charts, and is well suited for taking signal processing algorithms from concept to production.

- *Graphics processing units* (GPUs) are extremely powerful computationally. These processors have been driven to very high levels of performance and low price points by the need for real-time computer graphics in mass market gaming. Over the past 10 years, they have evolved into a general-purpose programmable architecture and supporting ecosystem that makes it possible to use them for a wide range of nongraphics applications [6]. GPU-accelerated libraries provided by manufactures like Nvidia, provide highly optimized functions that perform 2x to 10x faster than CPU-only alternatives. GPU-accelerated libraries for linear algebra, signal processing, and image and video processing lay the foundation for future software-defined radio applications to run on these types of architectures [7].
- *Advanced RISC Machines* (ARMs) have received significant attention in recent years for their low cost, small size, low power consumption, and computational capabilities. Such processors combined with a capable RFFE make them suitable platforms for mobile communications and computing. Additions of new SIMD instructions for the Arm Cortex-A series and Cortex-R52 processors, known as NEON [8] are accelerate signal processing algorithms and functions to speed up software-defined radio applications.

It is an exciting time for algorithm developers; there are many new and advanced methods of implementing signal processing applications on hardware. The difficulty is to ensure that no matter which hardware is chosen to run algorithms on, the hardware and development methodology will be supported in 5 years.

1.6 Software Environments for SDR

As described in Section 1.2, at their most fundamental level, most commercially available SDR platforms convert live RF signals to samples at digital baseband, and use a software-defined mechanism for modulation and demodulation techniques to transfer real-world data. Referring back to Figure 1.3, the boundary between the analog and digital worlds for a communication system is located at the analog-to-digital converter (ADC) and the digital-to-analog converter (DAC), where signal information is translated between a continuous signal and a discrete set of signal sample values. Typically, the radio can be configured to select center frequency, sampling rate, bandwidth, and other parameters to transmit and receive signals of interest. This leaves the modulation and demodulation techniques, which are developed using a two-step development process.

1. Develop, tune, and optimize the modulation and demodulation algorithms for a specific sample rate, bandwidth, and environment. This is normally done on a host PC, where debugging and visualization is much easier. At this phase of development, the modulation and demodulation of the RFFEs are performed on a host, providing great flexibility to experiment and test algorithm ideas.

2. Take the above algorithm, which may be implemented in a high-level language in floating point, and code it in a production worthy environment, making production trade-offs of a product's size, weight, power and cost (SWaP-C) in mind. These platforms become truly software-defined when the onboard hardware and embedded processor are programmed to perform application-specific digital communications and signal processing functions.

While this text focuses exclusively on the first algorithmic step of the SDR development process, the second production step cannot be excluded when looking at a development flow. Unless your goal is to publish a paper and never actually have a path for a working prototype, a complete development process must be kept in mind.

The first step requires a convenient mechanism to capture data for signal analysis and development of algorithms that process those signals. This makes it vitally important to have efficient and reliable PC-based software to develop and test the data transmission and digital signal processing functions in a wireless communications system.

One software environment that meets this requirement is MATLAB from MathWorks. MATLAB is a technical computing environment and programming language, allowing ease of use development and excellent visualization mechanisms. An additional product, Communications Systems Toolbox, adds physical layer algorithms, channel models, reference models, and connectivity to SDR hardware to transmit and receive live signals. MATLAB is cross platform (Windows, Linux, MAC) offering support for many of the popular commercial radio front-ends. Using MATLAB enables an incremental and iterative development workflow for SDR consisting of:

- Algorithm development and design validation with link-level simulations;
- Algorithm validation with live signals using connection to commercially available SDR hardware.

MathWorks also offers Simulink, which is an environment for real-world system simulation and automatic code generation for hardware and software implementation. It allows the radio developer to continue to the second stage of production development. These capabilities of Simulink provide a path to production:

- Development and validation of a hardware-accurate model;
- Implementation of a prototype on SDR hardware using automatic HDL and C code generation;
- Verification of the prototype versus the validated model;
- Deployment of the implementation to production SDR hardware.

Although Simulink will largely be ignored in this text, being able to have a single environment from concept to production is very powerful and should not be overlooked for those who are trying to make a real production radio.

Another SDR software architecture is the popular open-source GNU Radio software [9], which is a free software (as in freedom) development toolkit that provides signal processing blocks to implement software-defined radios and signal

processing systems. It can be used with external RF hardware to create software-defined radios, or without hardware in a simulation-like environment. It is widely used in hobbyist, academic, and commercial environments to support both wireless communications research and real-world radio systems.

In GNU Radio, a variety of C++ libraries modeling different digital communications and digital signal processing algorithms are integrated together using Python and SWIG (a software development tool that connects programs written in C and C++ with a variety of high-level programming languages including Python). These libraries are produced by the open-source community and freely shared with everyone.

The authors have used a variety of tools including MATLAB, Simulink, and GNU Radio in research, product development, and teaching undergraduate and graduate classes. Each tool has its advantages and disadvantages and can be used at different places of the research or development cycle. It was believed by the authors that all the various software environments can be used correctly or incorrectly to teach wireless physical layer fundamentals, although the prerequisites for each tool is different. For those who choose the GNU Radio path, the requirements to have a working knowledge of Linux, Python, C++, and SWIG is very high. While this is very common for a computer science student, it is not for most students in communications, and asking someone to learn the tool at the same time as the communications theory can prove difficult. One can use preexisting blocks in GNU Radio and bypass the requirements of understanding Python and C++, but then some opportunities to demonstrate and experiment with fundamental communications theory are lost, as the student just uses a block that someone else wrote, with little understanding of what it is doing. The same can be said for Simulink; it is also a very powerful tool, with many preexisting blocks for timing recovery and carrier synchronization. However, using these blocks does not allow many students to understand what is happening inside the blocks, and therefore the students have difficulty in understanding how to tune the blocks for their situation.

This is why MATLAB was chosen for this book. It is a cross-platform environment, allowing students to use what they are familiar with, and all the blocks presented are MATLAB scripts, with nothing to hide. If a student wants to better understand something, the entire algorithm is defined in the MATLAB code, with nothing to obfuscate the communications theory.

1.7 Additional readings

Although this chapter gave a brief introduction to the expanding area of SDR technology, there are several books available in the open literature that can provide a more detailed viewpoint of this topic. For instance, the book by Reed extensively covers many of the issues associated with the software architecture of an SDR platform [10], while many of the design considerations and approaches used to construct SDR hardware prototype and their RFFE are covered in the book by Kensington [11]. Another excellent reference regarding the hardware implementation of SDR systems is by Grayver [12]. Furthermore, understanding the importance of the analog-digital divide and how SDR systems bridge that

divide is the subject of the paper by Machado and Wyglinski [13]. Finally, an excellent series of papers covering the latest advances in SDR technology and providing some perspective on its evolution from 20 years ago are presented in IEEE Communications Magazine [14, 15].

References

- [1] American Guild Of Town Criers Website, 1997 <http://www.americantowncriers.com/>.
- [2] IEEE Project 1900.1 - *Standard Definitions and Concepts for Dynamic Spectrum Access: Terminology Relating to Emerging Wireless Networks, System Functionality, and Spectrum Management* <https://standards.ieee.org/develop/project/1900.1.html>.
- [3] Analog Devices ADF7030 <http://www.analog.com/ADF7030>
- [4] Hall, B., and W. Taylor, *X- and Ku-Band Small Form Factor Radio Design* <http://www.analog.com/en/technical-articles/x-and-ku-band-small-form-factor-radio-design.html>.
- [5] Xilinx Inc. www.xilinx.com *7 Series DSP48E1 User Guide, UG479 (v1.9)* September 27, 2016 https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf
- [6] McCool, M., "Signal Processing and General-Purpose Computing on GPUs," *IEEE Signal Processing Magazine*, Vol. 24, No. 3, May 2007, <http://ieeexplore.ieee.org/document/4205095/>.
- [7] Nivdea *GPU-accelerated Libraries for Computing* <https://developer.nvidia.com/gpu-accelerated-libraries>.
- [8] ARM NEON <https://developer.arm.com/technologies/neon>.
- [9] GNU Radio. *Welcome to GNU Radio!*. <http://gnuradio.org/>.
- [10] Reed, J. H., *Software Radio: A Modern Approach to Radio Engineering*, Upper Saddle River, NJ: Prentice Hall PTR, 2002.
- [11] Kensington, P., *RF and Baseband Techniques for Software Defined Radio*, Norwood, MA: Artech House, 2005.
- [12] Grayver, E., *Implementing Software Defined Radio*, New York: Springer-Verlag, 2012.
- [13] Machado, R. G. and A. M. Wyglinski, "Software-Defined Radio: Bridging the Analog to Digital Divide," *Proceedings of the IEEE*, Vol. 103, No. 3, March 2015, pp. 409–423.
- [14] Mitola, J., P. Marshall, K. C. Chen, M. Mueck, and Z. Zvonar, "Software Defined Radio - 20 Years Later: Part 1 [guest editorial], *IEEE Communications Magazine*, Vol. 53, No. 9, September 2015, pp. 22–23, <http://ieeexplore.ieee.org/document/7263341/?section=abstract>.
- [15] Mitola, J., P. Marshall, K. C. Chen, M. Mueck, and Z. Zvonar, "Software Defined Radio - 20 Years Later: Part 2 [guest editorial], *IEEE Communications Magazine*, Vol. 54, No. 1, January 2016, p. 58, <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7378426>.