

SOFTWARE DEFINED RADIO



Author: Edward Criscuolo

TABLE OF CONTENTS

| | |
|--|----|
| Abstract..... | 1 |
| Introduction | 1 |
| Definition of Software Defined Radio..... | 1 |
| Conventional Hardware Radios..... | 2 |
| Ideal SDR Concept..... | 2 |
| Practical Radios..... | 3 |
| Brief History of SDR | 3 |
| Digital Signal Processors..... | 3 |
| Mitola Paper, 1991 | 3 |
| Speakeasy I..... | 3 |
| Speakeasy II..... | 4 |
| SDR Forum..... | 4 |
| Joint Tactical Radio System (JTRS)..... | 4 |
| Open-Source SCA | 4 |
| Non-SCA..... | 5 |
| Industry Trends | 5 |
| Federal Regulations | 5 |
| SDR Fundamentals..... | 5 |
| Generic SDR Hardware Architecture | 5 |
| Quadrature Signals | 7 |
| Direct Conversion..... | 8 |
| Basic Software Functional Process Blocks | 9 |
| Resamplers | 9 |
| Modulation and Demodulation | 10 |
| FFT..... | 10 |
| Primitives | 10 |
| Higher-Order Blocks..... | 10 |
| SDR State of the Art in 2007 | 11 |
| Overview of SDR Development Products..... | 11 |
| Core Frameworks | 11 |
| SCARI++ (CRC Canada) | 11 |
| SCARI-Open (CRC Canada) | 11 |
| ORCA-CF (L3 Com) | 11 |
| Domain Manager Runtime Environment (dmRE) (Harris) | 11 |
| Spectra OE (Prismtech)..... | 11 |
| OSSIE (VIRGINIA Tech) | 12 |
| CORBA ORBs..... | 12 |
| ORBexpress RT (Objective Interface Systems)..... | 12 |
| Hardware..... | 13 |
| USRP (Ettus Research LLC)..... | 14 |
| Development Environments..... | 15 |
| Green Hills Software “Platform for SDR” | 16 |
| Communications Research Centre (CRC) “SCA Architect” | 16 |
| Harris Corporation “Domain Manager Toolkit” | 16 |

| | |
|--|----|
| PrismTech “Spectrum Power Tools” | 16 |
| Zeligsoft CE | 16 |
| GNU Radio Companion | 17 |
| OSSIE Waveform Developer..... | 17 |
| Prototype Spacecraft Telemetry Receiver..... | 17 |
| Overview..... | 17 |
| MidStar-1 Telemetry | 17 |
| CHIPSat Telemetry | 18 |
| Modulation and Demodulation..... | 19 |
| Bitslicer..... | 20 |
| Choice of Development Environments..... | 20 |
| Initial Prototype Design for MidStar Telemetry Receiver..... | 21 |
| MidStar First-Pass Data | 22 |
| Simulator and Bitslicer..... | 23 |
| MidStar Live-Pass Captures | 23 |
| HDLC Extractor..... | 24 |
| Bitslicer Tuning and Clock Recovery | 24 |
| Performance Results | 24 |
| CHIPSat Live-Pass Capture Attempt | 24 |
| Final Prototype Design..... | 25 |
| USRP Source | 27 |
| IF Low-Pass Filter..... | 27 |
| Quadrature Demodulator..... | 27 |
| Video Low-Pass Filter | 27 |
| Clock Recovery..... | 27 |
| Adaptive Binary Slicer..... | 28 |
| NRZI-to-NRZ Converter..... | 28 |
| Unpacked-to-Packed Converter..... | 28 |
| File Sink..... | 28 |
| Future Directions | 28 |
| Enhancements..... | 28 |
| RF Tuner and Doppler Tuning | 28 |
| HDLC Processing Block..... | 29 |
| BiPhase Decoder | 29 |
| Potential New Projects | 29 |
| Glossary | 30 |

LIST OF FIGURES

| | |
|--|----|
| Figure 1. Conventional Radio Process Flow..... | 2 |
| Figure 2. Conventional Radio Functional Allocation | 2 |
| Figure 3. Idealized SDR Functional Allocation..... | 2 |
| Figure 4. Practical SDR Functional Allocation..... | 3 |
| Figure 5. Functional Block Diagram of a Generic Software Defined Radio..... | 6 |
| Figure 6. Hardware Block Diagram of a Generic Software Defined Radio..... | 6 |
| Figure 7. “Classic” Quadrature Generation | 8 |
| Figure 8. “Modern” Quadrature Generation | 8 |
| Figure 9. NRZI Encoding | 18 |
| Figure 10. BiPhase Encoding..... | 19 |
| Figure 11. Initial Design Block Diagram..... | 21 |
| Figure 12. Demodulated Bits from First Pass, Showing NRZI-Encoded Flag Bytes..... | 22 |
| Figure 13. MidSTAR Spectrum During Telemetry Downlink..... | 23 |
| Figure 14. Demodulated MidSTAR Bitstream During Telemetry Downlink..... | 24 |
| Figure 15. Demodulated CHIPSat Bitstream Showing BiPhase- Encoded Flag Byte..... | 25 |
| Figure 16. SDR Telemetry Receiver Prototype Block Diagram | 26 |

ABSTRACT

This report provides an introduction to the emerging technology of software defined radios (SDRs), a survey of the current state of the art, and a description of the development of an SDR-based prototype of a spacecraft telemetry receiver.

An SDR system is a radio communication system that can tune to any frequency band and receive any modulation across a large frequency spectrum by means of software. An SDR performs significant amounts of signal processing in a general-purpose computer or a reconfigurable piece of digital electronics. The goal of SDR technology is to produce a radio that can receive and transmit virtually any new form of radio protocol just by running new software.

SDR can be used to implement radio modem technologies, including those used to send and receive spacecraft telemetry and commands. The goal of this grant was to learn the specifics of SDR technology, become familiar with at least one SDR development environment, and use it to develop and document prototype software for an SDR that can receive and demodulate the telemetry stream from an existing spacecraft such as MidSTAR-1 or CHIPSat.

The use of SDRs in spacecraft communications allows for the exciting possibility of on-orbit modification of the modulation and coding used in response to on-orbit failures or operational performance that differs from what was expected. These options can spell the difference between failure and success for a space mission.

INTRODUCTION

The use of SDRs in spacecraft communications has the potential to reduce the cost of ground-station receivers drastically and to future-proof new flight hardware by providing a system that is easily reusable, reconfigurable, and upgradeable without expensive (and sometimes impossible) change-out of flight hardware.

Software radios also have significant utility for the military and cell phone services, both of whom must serve a wide variety of changing radio protocols in real time.

The Joint Tactical Radio System (JTRS) is a U.S. and NATO program to produce radios that provide adaptable and interoperable communications. The program is providing a flexible new approach to meet diverse warfighter communications needs through software-programmable radio technology. The JTRS-developed Software Communications Architecture (SCA), despite its military origin, is under evaluation by commercial radio vendors for applicability in their domains.

Emergency responders have a critical need for interoperable communications. Someday, this may be achieved by a single, common standard, but SDRs have the potential to do it now with dynamically reconfigurable radios.

In the long run, proponents of SDR expect it to become the dominant technology in radio communications.

DEFINITION OF SOFTWARE DEFINED RADIO

Joe Mitola III, who is often credited as the father of software defined radio (SDR), is quoted as saying, "A software radio is a radio whose channel modulation waveforms are defined in software. That is, waveforms are generated as sampled digital signals, converted from

digital to analog via a wideband DAC and then possibly up-converted from IF to RF. The receiver, similarly, employs a wideband analog-to-digital converter (ADC) that captures all of the channels of the software radio node. The receiver then extracts, down-converts and demodulates the channel waveform using software on a general purpose processor.”

Another widely used but less technical description of unknown origin says simply “Get the software as close to the antenna as possible.”

Essentially, an SDR system is a radio communication system that can tune to any frequency band and receive any modulation across a large frequency spectrum by means of software, and performs significant amounts of signal processing in a general-purpose computer or a reconfigurable piece of digital electronics.

CONVENTIONAL HARDWARE RADIOS

The generalized processing flow of a digital packet radio can be seen in **Figure 1**. In a conventional hardware radio, all of the functions through modulation/demodulation are implemented completely in hardware and baseband processing covering a spectrum from hardware (HW) to software (SW). This is shown in the graph in **Figure 2**.

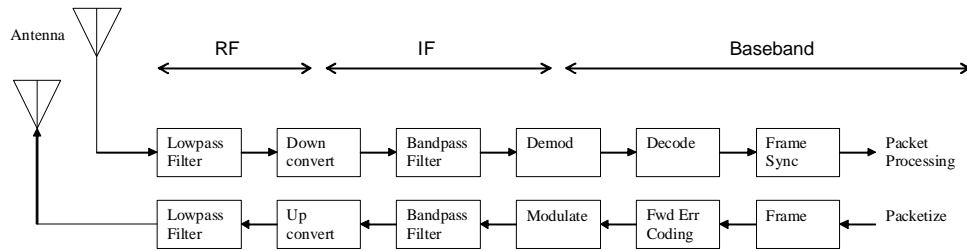


Figure 1. Conventional Radio Process Flow

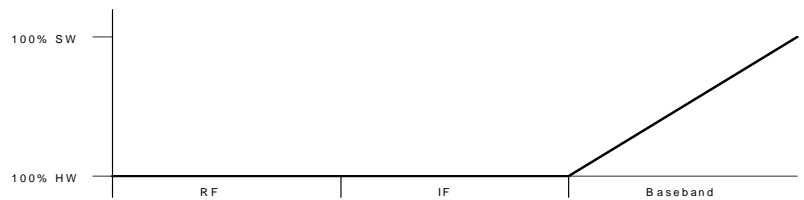


Figure 2. Conventional Radio Functional Allocation

IDEAL SDR CONCEPT

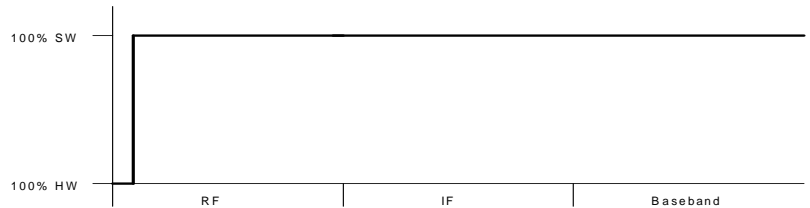


Figure 3. Idealized SDR Functional Allocation

In a completely idealized SDR, the antenna would go through an analog low-pass filter and would then be directly connected to the input of a high-speed analog-to-digital converter (ADC). A signal processor would then read the stream of values from the ADC, and software would then perform all further conversion, demodulation, synchronizing, and packet processing of the received signal. Similarly, a transmitter would use software to

packetize, frame, modulate, and convert the signal in digital form and then feed the resulting stream of digital values to a high-speed digital-to-analog converter (DAC) that would have its output directly connected to an antenna.

PRACTICAL RADIOS

Practical SDRs are a compromise between conventional hardware radios and the ideal SDR concept, with some of the front-end functions being performed in hardware and the rest in software. The dividing line between the two regimes is fuzzy and continually shifting closer towards the antenna.

This is driven primarily by the increasing speed and resolution of high-speed converters, and secondarily by the increasing processing horsepower of general-purpose processors. For example, in 2003, directly digitizing and processing a 4-MHz signal was out of the reach of the then-available hardware. Currently, in 2007, inexpensively available hardware is capable of directly digitizing and processing signals up to 70 MHz.

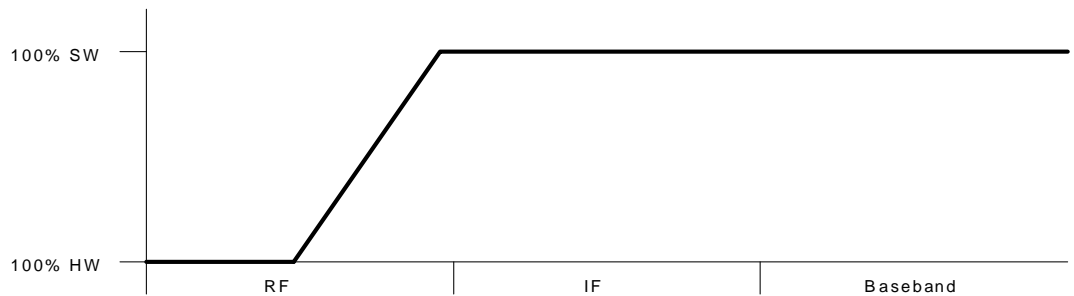


Figure 4. Practical SDR Functional Allocation

BRIEF HISTORY OF SDR

DIGITAL SIGNAL PROCESSORS

Digital Signal Processors (DSPs) were the precursors of SDR. These specialized, high-speed processors were specifically developed to begin performing signal processing in the digital domain instead of designing analog circuits. Generally, the controlling programs were application-specific and committed to firmware, with only a few externally settable parameters. Although the basic digital processing theory dates back to the 1940s, practical hardware did not even exist until advances in integrated circuit technology made the first crude DSPs possible in the mid 1970s. After that, advances in the theory and application of digital signal processing moved rapidly.

MITOLA PAPER, 1991

In 1991, Joseph Mitola III of MITRE published the first technical paper on software defined radio architecture. At that time, he had already designed an SDR for the Air Force. GEC/Marconi had also done some work on it. In 1995, as the editor of the *IEEE Communications Magazine*, Mitola put together a series of articles on wideband analog-to-digital converters and digital signal processing and wrote the lead article on architectures. That was a watershed event, as it brought together research from around the world.

SPEAKEASY I

In 1992, the U.S. Army launched Phase I of the SPEAKEasy program. Its goal was to produce a radio that could operate from 2 MHz to 2 GHz and communicate with ground forces, Air Force radios, naval radios, and satellites using a wide range of modulations. By

1995, it had demonstrated a radio that met all these goals. Its software architecture, though practical, was a unique solution bearing no resemblance to any other system, and it was not easily reconfigurable. But its basic architecture has since become a standard design scheme for software radios.

SPEAKEASY II

SPEAKeasY Phase II was a research project whose goals were to get a more quickly reconfigurable architecture, an *open* software architecture, and the ability to “bridge” different radio protocols. The project produced a demonstration radio only 15 months into the 3-year project. The demonstration was so successful that further development was halted, and the radio went into production with only a 4-MHz to 400-MHz range. The software architecture identified standard interfaces for different modules of the radio and was the first known to use FPGAs (field programmable gate arrays) for digital processing of radio data.

SDR FORUM

In 1996, the successful demonstrations of SPEAKeasY-type radios led a group of more than 50 companies worldwide to form the Modular Multifunction Information Transfer System (MMITS) Forum, later renamed the Software Defined Radio Forum (SDRF). The goals of the SDRF are to accelerate the development, deployment, and use of software defined radios and to work toward the adoption of an open architecture for the equipment.

JOINT TACTICAL RADIO SYSTEM (JTRS)

In early 1997, the U.S. DoD, in cooperation with NATO, initiated a major cross-service program called the Joint Tactical Radio System (JTRS). JTRS can be seen as a direct descendant of the SPEAKeasY program. The initial goal of the JTRS program was to provide tactical radios that would allow interservice and coalition interoperability for the transmission of voice, video, and data from the 2-MHz through 2-GHz bands.

But the initial scope of the program was too broad, and the schedule too unrealistic. By 2004, the JTRS program bore a striking resemblance to the Ada language program from around 1985 and was in similar trouble. In 2005, the program came under review, and in 2006, it was scoped back and replanned to a more realistic schedule.

One of the major products of the JTRS program has been the standardization of the Software Communications Architecture (SCA) that is the core framework of all JTRS radio development. This CORBA-based architecture provides a standard software Application Programming Interface (API) and provides a hardware-abstraction layer to isolate the software from becoming tightly coupled to specific implementation hardware. The SCA, despite its military origin, is under evaluation by commercial radio vendors for applicability in their domains.

January of 2007 saw one of the first JTRS orders for \$48 million of Multiband Inter/Intra Team Radios (MBITR) for tactical deployment.

OPEN-SOURCE SCA

In 2003, Virginia Tech began an open-source SDR development effort called OSSIE (Open Source SCA Implementation::Embedded). Its goals were to produce a software package that includes an SDR core framework based on the JTRS SCA and tools for rapid development of SDR components and waveforms. OSSIE is functional and still under active development and refinement as of late 2007.

NON-SCA

In parallel with the JTRS activities, many noncommercial and small-scale commercial SDR activities sprang up, largely driven by the amateur radio community. These efforts took advantage of the explosive growth in CPU horsepower of personal computers, combined with low-cost A-to-D and D-to-A hardware such as PC sound cards. More recently, the low-cost Universal Software Radio Peripheral (USRP) has been commercially developed, which uses a USB 2.0 interface, a FPGA, and a high-speed set of ADC/DACs, combined with reconfigurable free software. Its sampling and synthesis bandwidth is a thousand times that of PC sound cards, which enables an entirely new set of applications. A standardized, open-source core framework was developed called GNU Radio. Although object-oriented, like SCA, it is not an SCA-compliant architecture and does not use CORBA. It is functional and currently under active development as of late 2007.

INDUSTRY TRENDS

The Object Management Group (OMG), a not-for-profit consortium that produces and maintains computer industry specifications for interoperable enterprise applications, has established a Domain Special Interest Group for software radios (SWRADIO DSIG). This group, along with the Software Defined Radio Forum (SDRF), is working toward building an international commercial standard based on the SCA.

FEDERAL REGULATIONS

The Federal Communications Commission (FCC) has become heavily involved in SDR and its use of free and open source software (FOSS). It has recently issued a set of rules concerning devices whose radio frequency and power characteristics can be modified by software. The rules require any manufacturer of such a device to take steps to prevent “unauthorized” changes to the software on the device that might alter its radio frequency and power parameters in a way that takes it out of compliance with the FCC Part 15 regulations. Although the FCC has taken an extremely conservative stance on this issue, it recently clarified these rules to state that they do not apply to independent third-party software developers, only to the parties directly involved in the marketing or sale of radio devices. It further stated that, although the use of FOSS for the security implementation would be permitted, it would impose a “high burden” on such implementations “to demonstrate that it is sufficiently secure.”

This type of collision between existing legislation and new technologies that were never imagined when the legislation was drafted are becoming increasingly common. Witness the current controversy over software patents. So far, the FCC is staying on the edge of the abyss by directing its rules at radio equipment manufacturers, not software developers. But it's a slippery slope, and one bad decision could cripple this emerging industry.

SDR FUNDAMENTALS

GENERIC SDR HARDWARE ARCHITECTURE

An SDR has a generic hardware platform on which software runs to provide waveform functions such as frequency selection, frequency hopping, modulation/demodulation, filtering (including bandwidth changes), and coding. In an ideal world, the digital electronics would handle the signal at the final frequency and at the correct level. As previously mentioned, this is only currently possible up to a few hundredmegahertz. As a result, some analog radio frequency (RF) elements are still required to convert the operating frequency down to an intermediate frequency (IF) that the digital converters can handle.

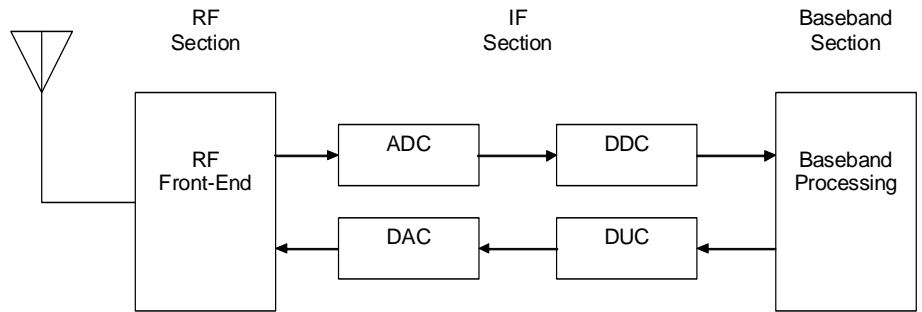


Figure 5. Functional Block Diagram of a Generic Software Defined Radio

As seen in the functional block diagram in **Figure 5**, the generic hardware for an SDR typically has three major sections:

An RF section, containing the analog front end that converts to and from the IF

An IF section that contains the ADC and DAC and the digital down- and up-converters (DDC, DUC) that convert the IF to and from baseband

A baseband section that performs the bulk of the waveform and protocol processing

These functions are usually grouped into hardware based on the speed and processing throughput required. The ADC and DAC must run at the sampling clock rate and typically are dedicated pieces of hardware. The digital down- and up-converters also run at the sampling clock rate. For all but the lowest frequencies, they are most often implemented in a high-speed FPGA. A high-speed DSP is another possible choice, but it provides less flexibility than an FPGA. The resulting data stream is at a lower sample clock rate that can be handled by a general-purpose processor (GPP). Input/output (I/O) between the generic SDR hardware and the GPP is frequently over USB or Ethernet. Control of all the high-speed elements is also usually performed in an FPGA, which can be shared with the digital down- and up-converters. **Figure 6** shows a typical hardware block diagram.

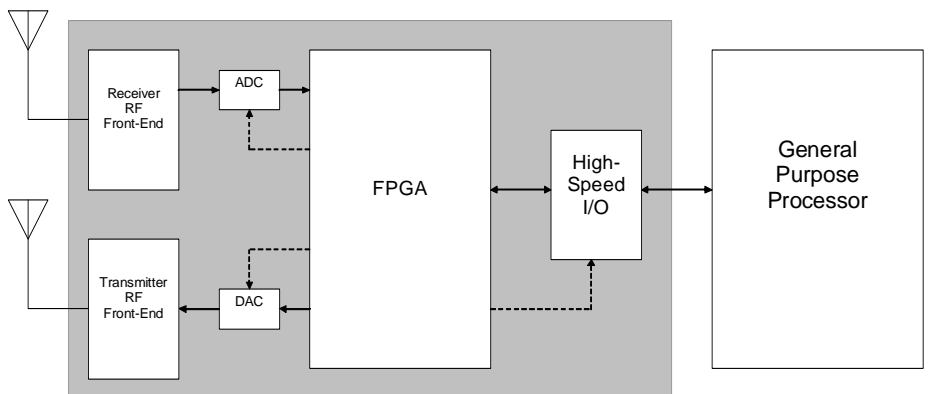


Figure 6. Hardware Block Diagram of a Generic Software Defined Radio

RF Section

The RF section (also called the RF front end) is responsible for transmitting/receiving the RF signal to/from the antenna and converting the RF signal to an IF signal using standard

single or multiple conversion techniques. The RF front end on the receive path performs RF amplification and analog down-conversion from RF to IF. On the transmit path, the RF front end performs analog up-conversion and RF power amplification. In low-frequency systems, where the ADC/DAC converters are capable of conversion at the RF frequency, the RF front end may simply consist of analog low-pass filters.

IF Section

The ADC/DAC blocks perform analog-to-digital conversion (on the receive path) and digital-to-analog conversion (on the transmit path), respectively. ADC/DAC blocks interface between the analog and digital sections of the radio system. The converters are usually arranged in pairs, each operating on a set of quadrature signals, known as I and Q, that are phase shifted 90 degrees apart. This allows the retention of both phase and amplitude information from the original signal. The DDC/DUC blocks perform digital down-conversion (on the receive path) and digital up-conversion (on the transmit path), respectively. This includes a digital low-pass filter followed by a decimator (on the receive path) and an interpolator followed by a low-pass filter (on the transmit path). The purpose is to reduce the high sample rate, which was necessary to capture the IF signal, down to something that is reasonable for the lower-frequency baseband waveform. DDC/DUC blocks essentially perform some of the radio's modem operations on the signal by converting the signal to and from baseband at an appropriate sampling rate. The functions of the DDC/DUC blocks require relatively large amounts of computing power. Early digital radios implemented them in either custom application-specific integrated circuits (ASICs) or dedicated DSPs. Later, these functions moved to FPGAs, providing both speed and complete reprogrammability.

Baseband Section

The baseband section performs baseband operations (modulation/demodulation, connection setup, equalization, frequency hopping, timing recovery, correlation) and also implements the link layer protocol (layer 2 protocol in OSI [Open Systems Interconnection] protocol model). Essentially all of the remaining functions of the radio are performed here. Baseband processing also requires relatively large amounts of computing power, although not quite as much as the IF section. Early SDRs had to use high-speed DSPs here, but the ongoing increase in compute cycles available in general-purpose processors (Moore's Law) now permits their use for many modulation waveforms across a wide range of applications. This allows the use of standard programming languages and software development environments for the bulk of the SDR software.

QUADRATURE SIGNALS

Quadrature signals are at the core of every SDR. This is because they preserve phase information, and many digital systems depend on phase encoding. The classic solution (see **Figure 7**) for generating quadrature signals is to mix and digitize two channels using a reference oscillator that produces two signals, one of which lags the other by 90 degrees of a cycle. This allowed the ADCs to operate at the lower down-converted frequency. Thanks to the advances in the speed of ADCs and DACs, modern solutions (**Figure 8**) perform all the operations in the digital domain, avoiding many of the problems associated with analog mixer circuits.

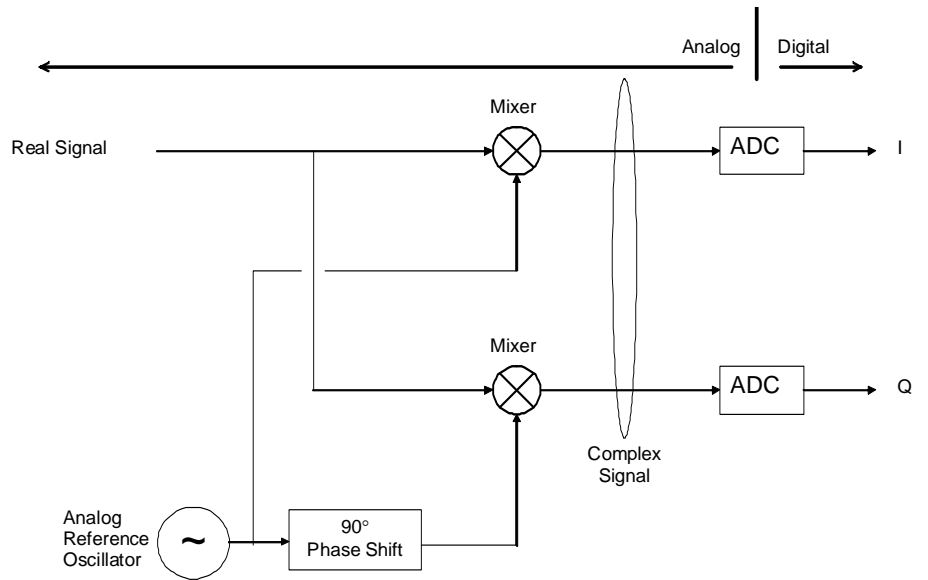


Figure 7. "Classic" Quadrature Generation

The in-phase signal is traditionally known as "I" and the out-of-phase signal as "Q." The two sets of samples provide the needed phase information. Frequently, these two samples are combined into a single sample represented by a complex number, with I as the real part and Q as the imaginary part. This fits directly into all the theoretical math that is used to describe communications theory, allowing its direct use in various software functional blocks.

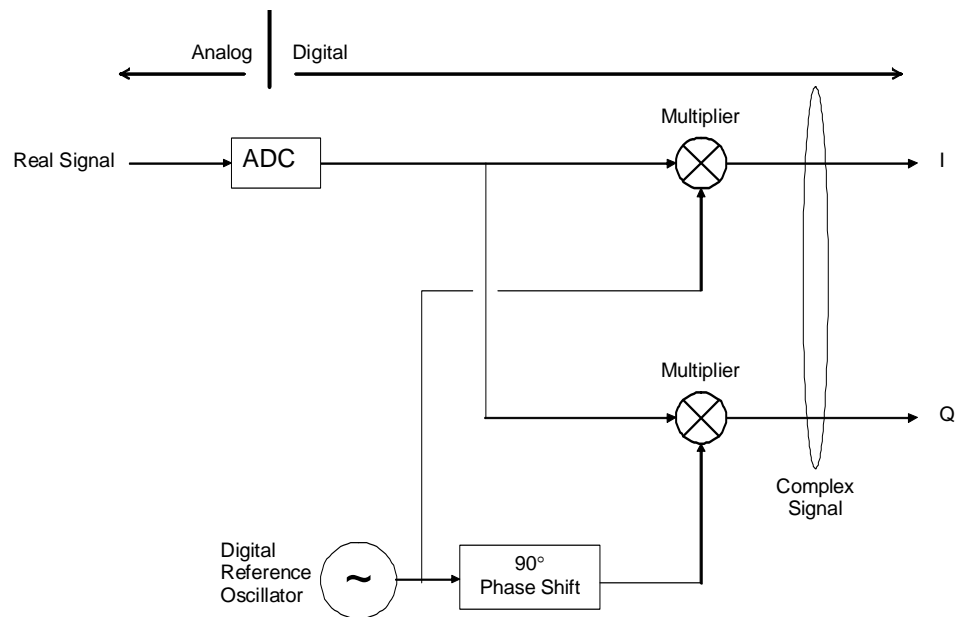


Figure 8. "Modern" Quadrature Generation

DIRECT CONVERSION

If the frequency of the reference oscillator in **Figure 8** above corresponds to the carrier frequency, we have what is called a "zero IF" solution, also known as a "Direct Conversion Receiver." Because the signal in a zero IF design is mixed down to a bandwidth that is

centered about 0 Hz, quadrature versions of the signal *must* be generated in order to be able to differentiate between in-band components above the reference frequency and those below the reference frequency. A big attraction of direct conversion is that there are no image problems and that the IF selection filter becomes a pair of low-pass filters at baseband.

BASIC SOFTWARE FUNCTIONAL PROCESS BLOCKS

An SDR core framework consists of a number of functional process blocks that operate on the digitized data. Some of these can be simple, almost trivial, operations, whereas others can be quite complex. New blocks can be built up out of existing ones. Complicated waveforms can be constructed and deconstructed in this manner. SDR software architectures typically segregate the data flows into and out of these blocks to isolate the streaming high-rate digitized data from the lower-rate configuration, management, and control data. A discussion of some of the fundamental functional blocks follows.

Digital filters operate on a sequence of values represented by the sequence $x_0, x_1, x_2, x_3, \dots, x_n$ corresponding to the digitized values of the signal waveform and produce a sequence of values $y_0, y_1, y_2, y_3, \dots, y_n$ corresponding to the output of the filter. These values can be real or complex. In general, the value of y_n is calculated from the values $x_0, x_1, x_2, x_3, \dots, x_n$. The way in which the y terms are calculated from the x terms determines the filtering action of the digital filter. A filter that only depends on x_n to produce y_n is called a *zero-order* filter. It is also a trivial example, because it doesn't really perform any filtering. Filters are first-order, second-order, etc. depending on how many previous x terms are used. The general form is $y_n = a_0x_n + a_1x_{n-1} + a_2x_{n-2} + \dots$ where a_0, a_1, a_2, \dots are called the *filter coefficients*. They, too, can be real or complex. This type of filter is called a *non-recursive* or *finite impulse response* (FIR) filter.

A second type of digital filter exists, called a *recursive* or *infinite impulse response* (IIR) filter. In it, the value of y_n depends not only on the previous input values, but on the previous *output* values as well. IIR filters are more economical than FIR filters because they can be constructed with fewer terms and less storage, because each y term already contains within it previous values of x . One of the common uses for an IIR is to construct an *integrator*, which has the same effect as an analog low-pass filter.

Often, an SDR is used to convert an entire wideband block of frequencies to baseband. This block may contain many separate narrowband channels. In order to extract a single channel, it must be run through a bandpass filter to separate it from the rest and then have its center frequency translated to zero to produce a baseband signal containing just the desired channel. The process is similar to the conversion of an RF or IF signal to baseband, but it is performed entirely in the digital domain. A pair of 90-degree shifted reference sine waves are mixed (multiplied) with the digital I and Q values and then low-pass filtered to produce a second set of I and Q values representing just the selected channel centered on 0 Hz.

RESAMPLERS

All of the digital data streams within an SDR have an associated *sample rate*. Often, it becomes necessary to change the sample rate, particularly after any operation that has left the data wastefully oversampled such as filtering, demodulation, or down-conversion. Another case requiring resampling is *clock recovery*, where the data stream represents a bitstream of binary data, and the sample rate needs to be adjusted to synchronize with the bit rate so that there is one sample per bit. Resamplers come in three forms: *decimators*, which reduce the sample rate by an integer factor; *interpolators*, which increase the sample rate by an integer factor; and *rational resamplers*, which use a combination of a decimator and an

interpolator to change the sample rate by a non-integer rational factor. Typically, a decimator is preceded by a low-pass filter in order to avoid aliasing at the lower sample rate. Similarly, an interpolator is typically followed by a low-pass filter to remove any introduced high-frequency aliases.

MODULATION AND DEMODULATION

Modulator and demodulator blocks come in all the traditional “analog” modulations such as Amplitude Modulation (AM), Frequency Modulation (FM), and Phase Modulation (PM), as well as all the various types of digital modulations. These digital modulations include single-bit-per-symbol schemes like On-Off Keying (OOK), Frequency Shift Keying (FSK), Mean Shift Keying (MSK), Gaussian Mean Shift Keying (GMSK), and BiPhase Shift Keying (BPSK), as well as multibit-per-symbol modulations such as Quadrature Phase Shift Keying (QPSK) and 16-level Quadrature Amplitude Modulation (16QAM).

Many of these modulations become simple to handle when the data is already in I and Q quadrature form. Consider AM, for example. Because the I and Q data can be represented as a complex number, another way to look at the data is as a vector quantity with a magnitude and a phase angle. Demodulating an AM signal, then, simply becomes the process of taking the square root of the sum of the squares of I and Q to calculate the magnitude of each complex sample. Similarly, demodulating a PM signal simply becomes the process of taking the arctangent of Q over I to calculate the phase angle of each complex sample. And, because frequency can be defined as the rate of change of the phase angle, demodulating FM simply requires taking the difference between the phase angle of the current sample and the previous one.

FFT

The Fast Fourier Transform (FFT) provides the means of transforming a signal defined in the time domain into one defined in the frequency domain. The FFT is used to transform a continuous time signal into the frequency domain. It describes the continuous spectrum of a nonperiodic time signal. FFTs can be used for channelization and filtering and are often used in conjunction with a graphical display to provide a low-cost spectrum analyzer.

PRIMITIVES

Although the available core frameworks all provide a complete set of basic and advanced processing blocks, sometimes it becomes necessary or desirable to implement a completely new waveform or try a new algorithm for an existing function. In these cases, primitive blocks are provided for basic math functions such as addition, subtraction, multiplication, and division; format conversion to go between real, complex, float, and integers; and various buffering and delay processing functions.

HIGHER-ORDER BLOCKS

Usually, SDR designs involve higher-order blocks that have been assembled out of simpler processing blocks. These “components” greatly simplify the development of an SDR. The SCA has specified a large number of these components to handle the majority of the standard waveforms found in military radios. These also have utility in nonmilitary applications.

Consider, for example, a GMSK demodulator component block. Internally, it may consist of an FM demodulator, a clock recovery block, and a bitslicer, but externally, it can be treated as a block that accepts a stream of complex baseband values at some sampling rate and outputs a stream of bits at some (different) data rate.

SDR STATE OF THE ART IN 2007

OVERVIEW OF SDR DEVELOPMENT PRODUCTS

This is a big topic, and it is not possible to do an exhaustive treatment here. Instead, this section covers a wide array of the main products that represent the current state of the art.

CORE FRAMEWORKS

SDR systems need to have an Operating Environment to execute in. This generally consists of two or three parts:

- A core framework (CF)
- A CORBA Object Request Broker (ORB) [SCA-compliant systems only]
- An underlying Operating System (OS)

These usually come from different vendors and cover a wide range, from large packages running on a powerful desktop to tiny systems with small memory footprints suitable for embedded applications.

There seem to be only a small number of distinct core framework vendors. These CFs get mixed and matched with other components to form a dizzying array of development and operating environments from a multitude of vendors. Most of these CFs are SCA-compliant and utilize an underlying CORBA ORB for object definition and communication. A few are open source, and at least one is not SCA or CORBA. In addition, many HW product vendors are implementing their own CF specifically to support their own product, such as BAE's iPAQ PDA.

SCARI++ (CRC CANADA)

SCARI++, from CRC Canada, is a C++ implementation of the SCA Core Framework version 2.2. It runs on Linux and the Green Hills Integrity RTOS (real-time operating system). It is frequently supported by tool vendors in their integrated development environments.

SCARI-OPEN (CRC CANADA)

SCARI-Open, from CRC Canada, is an open-source Java implementation of the SCA core framework.

ORCA-CF (L3 COM)

ORCA-CF, from L3 Communications, is a C++ implementation of the SCA Core Framework version 2.2. It has support for different ORBs (TAO, OmniORB, and ORBexpress) on Linux and VxWorks. It is also supported by at least one integrated development environment vendor, but does not seem to be marketed heavily.

DOMAIN MANAGER RUNTIME ENVIRONMENT (DMRE) (HARRIS)

Domain Manager Runtime Environment (dmRE), from Harris, is an SCA core framework that is primarily used internally by Harris in their Falcon-II radio, but is sometimes picked up by tool vendors and supported in their integrated development environments.

SPECTRA OE (PRISMTECH)

Spectra OE, from PrismTech, is a high-performance, low-overhead core framework and middleware implementation that runs on any mix of general-purpose processor (GPP), DSP, and FPGA processor technologies. It is optimized for embedded systems, with a combined

CF/ORB/Services memory footprint under 2 MB. It is available separately or preintegrated with an RTOS.

OSSIE (VIRGINIA TECH)

OSSIE, from Virginia Tech, is an open-source SCA 2.2 implementation. It runs on Fedora Core Linux and is also supported by at least one integrated development environment vendor.

Gnu Radio

GNU Radio is an open-source non-SCA core framework written in C++ and Python that has been developed by the open-source community. It runs on Linux, Mac OSX, NetBSD, and Windows.

CORBA ORBS

In an SCA-compliant system, communication is based on CORBA. The SCA core framework uses CORBA to connect all the components and initialize the application. Under SCA, there are four basic communication categories, each with its own requirements:

| Category | Direction | Data Rate | Latency |
|---------------------------------|--------------------|-----------|---------|
| Waveform control and management | One-Way or Two-Way | Low | Relaxed |
| Signaling | One-Way | Low | Low |
| Behavior | Two-Way | Moderate | Low |
| Streaming | One-Way | High | Low |

In the past, CORBA ORBS bought their flexibility at the expense of excessive memory and CPU overhead. Thus, in embedded and real-time systems, workarounds were often used to improve performance at the expense of portability. But modern optimized ORBs, combined with ever-increasing CPU power, are rapidly making this a nonissue.

It turns out that there are just a few ORBs in common use, with one (ORBexpress) being used by almost all of the available commercial SCA integrated development environments.

ORBEXPRESS RT (OBJECTIVE INTERFACE SYSTEMS)

ORBexpress RT, from Objective Interface Systems, is a real-time, high-performance ORB in C++ or Ada, targeted for QNX, IRIX, VxWorks, LynxOS, INTEGRITY, or Linux.

ORBexpress ST (Objective Interface Systems)

ORBexpress ST, from Objective Interface Systems, is a self-hosted, high-performance ORB in C++, Java, or Ada, running on Solaris, Linux, Windows, IRIX, AIX, LynxOS, or Sun JVM.

ORBexpress DSP (Objective Interface Systems)

ORBexpress DSP, from Objective Interface Systems, is a special version of ORBexpress that is cross-supported under Windows and targeted for the DSP-BIOS/TI C6000.

OpenFusion e*ORB (Prismtech)

OpenFusion e*ORB, from PrismTech, is a high-performance, low-footprint ORB in C or C++, targeted for VxWorks, Integrity, Windows NT, Linux, or Solaris. This seems to be the second-most common ORB out there in the commercial world.

ICO (Prismtech)

The Integrated Circuit ORB (ICO), from PrismTech, is an embedded ORB that has been written in portable VHDL and can be synthesized onto any FPGA or ASIC platform. This allows for an SCA-compatible interface between distributed software objects running on processors and waveform objects residing in silicon.

The Ace ORB (aka TAO) (Object Computing)

The Ace ORB (aka TAO), from Object Computing, is an open-source C++ ORB, running on Solaris, Windows, Linux, HP-UX, SCO Unix, AIX, Mac OSX, IRIX, LynxOS, VxWorks, Windows CE, or INTEGRITY. It is supported under Zeligsoft's integrated development environment (IDE).

omniORB (sourceforge)

omniORB, from SourceForge, is an open-source C++ ORB, with Python bindings to speed development and testing. Commercial support is available from Apasphere Ltd. This is the ORB used by OSSIE. It is supported under Zeligsoft's IDE.

JacORB

JacORB is an open-source Java ORB that is JDK 1.3 and 1.4-compatible. It is supported under PrismTech's IDE.

HARDWARE

Most of the low-cost hardware solutions are using 12- to 14-bit ADCs and DACs and handle bandwidths ranging from 96 KHz to 8 MHz. The (b)leading-edge solutions go up to 16 bits and 70-MHz bandwidth or more. Some of the hardware also offers Transmitter and Receiver RF front ends, usually as daughterboards or mezzanine boards to bring higher frequencies down to within range of the digital sampling rate. The following list of the available SDR hardware specifications is arranged roughly in order of capability, from lowest to highest.

High-End PC Sound Card

- Two 96 KS/sec 16-bit analog-to-digital converters
- Two 96 KS/sec 16-bit digital-to-analog converters
- Requires external hardware to down-convert RF to baseband
- PCI bus interface
- Handles signals up to 96 KHz wide

SoftRock-40

- 7.5-MHz (40-meter) RF receiver front end for sound-card-based SDRs

SDR-IQ (RFSPACE INC)

- Receiver only
- Two 67 MS/sec 14-bit analog-to-digital converters
- Up to 30 MHz RF
- USB 2.0 interface
- Handles signals up to 190 KHz wide

- Proprietary Software: SpectraView
- \$399

SDR-14 (RFSPACE INC)

- Receiver only
- Two 67 MS/sec 14-bit analog-to-digital converters
- Up to 200 MHz RF
- USB 2.0 interface, optional Ethernet
- Handles signals up to 190 KHz wide
- Proprietary Software: SpectraView, DREAM
- \$1,099

SDR-1000 (Flex Radio Systems)

- Transmitter: 2 MHz – 50 MHz
- Receiver: 12 MHz – 60 MHz
- Sound card interface
- 11 ksamples/sec I & Q
- Handles signals up to 192 KHz wide
- Open Source Software: PowerSDR, GNU Radio
- \$875

USRP (ETTUS RESEARCH LLC)

- Four 64 MS/s 12-bit analog-to-digital converters
- Four 128 MS/s 14-bit digital-to-analog converters
- Four digital down-converters with programmable decimation rates
- Two digital up-converters with programmable interpolation rates
- Connectors for four daughterboards: 2 TX, 2 RX
- RF daughterboards support from DC to 2.9 GHz
- USB 2.0 interface (Gigabit Ethernet in next rev)
- Handles signals up to 8 MHz wide (more in next rev)
- Open-source support: GNU Radio and OSSIE (SCA)
- Mainboard \$700, Daughterboards \$75 – \$275

Small Form-Factor (SFF) SDR development Platform (Lyrtech)

- Transmitter daughterboard: 200 MHz – 930 MHz

- Receiver daughterboard: 20 MHz – 928 MHz
- 125-MSPS, 14-bit dual-channel analog-to-digital converters
- 500-MSPS, 16-bit dual-channel interpolating digital-to-analog converter
- USB 2.0 and 10/100 Ethernet
- Selection of bandwidth (5 MHz or 20 MHz)
- Proprietary/COTS (commercial off-the shelf) Software: SCARI SCA core framework (CRC Canada)
- \$2,900 – \$9,900

SDR-2000 SDR Platform (Spectrum Signal Processing)

- Transmitter: 140 MHz IF
- Receiver: Dual 140 MHz IF
- 213-MSPS, 12-bit dual-channel analog-to-digital converters
- 213-MSPS, 14-bit digital-to-analog converter
- 70-MHz bandwidth on each channel
- PCI/PCI-X interface to onboard dual Xeon (Windows or RH Linux)
- Dual-gigabit Ethernet
- Proprietary/COTS software: Harris Corporation SCA Core Framework, Zeligsoft SCA development tools
- \$???? (Quote required)

DEVELOPMENT ENVIRONMENTS

Development environments for SDRs cover a wide range of software. In addition to the usual source code editors, compilers, code generators, debuggers, and project builders, there are tools for hardware simulation, modeling, and graphical waveform development. Virtually all SDR environments are based on object-oriented technologies that are used to create functional “components” that are connected together to perform waveform construction and deconstruction. CORBA is used by SCA-compliant systems as the underlying object technology, whereas proprietary and open-source systems often rely directly on the host language’s object-oriented capabilities (e.g., C++, Java, Python) to avoid the size and processing overhead of CORBA. Almost all of the GUI-based modeling and development tools use the same visual paradigm: boxes of parameters with input and output “ports” that are interconnected by lines. These are a kind of visual “declarative programming” that the tools translate into the underlying code.

Things get complicated when one looks at specific vendors. Typically, a vendor only makes one (or a few) piece(s) of the development environment and bundles it with other vendors’ tools to make a complete environment.

GREEN HILLS SOFTWARE “PLATFORM FOR SDR”

Green Hills Software is one of the prime examples of this bundling philosophy, having “.. developed the Platform for SDR in combination with leading hardware and software providers focused on the SDR market specifically to provide a complete ‘prototype-to-deployment’ solution for SDR developers.”

Green Hills, original known for their C compiler, now produces their own RTOS, (INTEGRITY) and their own TCP/IP stack, as well as the MULTI™ integrated development environment with compilers for C, C++, EC++, and MISRA C and code generators for dozens of processors. Their “Platform for SDR” integrates these with a mix of tool suites, SDR frameworks, and CORBA ORBs from CRC Canada, Harris, PrismTech, Telelogic, Zeligsoft, and Objective Interface Systems.

COMMUNICATIONS RESEARCH CENTRE (CRC) “SCA ARCHITECT”

CRC Canada produces four SDR products:

- An SCA-compliant core framework
- An open-source Java SCA core framework
- The SCA Architect suite of tools
- The Component Development Library (CDL)

SCA Architect covers the complete SDR development life cycle, from the creation and validation of components to their assembly into applications or nodes. It is built as a set of plug-ins for the platform-independent Java-based Eclipse open development platform and supports graphical modeling.

The Component Development Library implements several common functionalities that must be part of every SCA component. The CDL provides much of the implementation’s CORBA Interface Description Language (IDL), reducing the quantity of code an application developer must create and test. The CDL’s code generator is used by the GUI tools, so the whole process is largely transparent to the developer.

HARRIS CORPORATION “DOMAIN MANAGER TOOLKIT”

Harris Corporation produces the Domain Manager Toolkit (dmTK), which provides both run-time components and development support tools.

PRISMTECH “SPECTRUM POWER TOOLS”

PrismTech’s Spectra Power Tools are another set of Eclipse plug-ins that support the full spectrum of GUI-based design, development, code-generation, and testing tools for SCA-compliant systems.

ZELIGSOFT CE

Zeligsoft produces a suite of SDR development tools based on their Zeligsoft CE development environment, integrated with an SCA core framework, an RTOS, and a CORBA ORB. It has been successfully used with virtually all commercial and proprietary SCA core frameworks (Harris dmTK, OSSIE, SCARI, Boeing, Northrop Grumman) and CORBA ORBs such as the TAO open-source ORB and OIS ORBexpress.

GNU RADIO COMPANION

The GNU Radio Companion (GRC) is an open-source GUI development tool that allows the user to quickly design GNU Radio-based applications by placing and connecting graphical elements representing GNU Radio's processing blocks. Easy addition of user-created processing blocks is also supported, but it requires writing a Python code "wrapper." Designs are saved as XML files. Although not SCA-compliant, it should be relatively straightforward to migrate apps created with GRC into an SCA environment because the objects are being represented at such a high level, such as a "low-pass filter" or a "decimator."

OSSIE WAVEFORM DEVELOPER

The OSSIE Waveform Developer (OWD) is an open-source GUI development tool that allows the user to design SCA-based applications that use the open-source OSSIE SCA core framework. Unlike most SDR GUI tools, it does not use the intuitive "boxes connected with lines" paradigm. Instead, it uses nested tree-lists to show the parameters, inputs, and outputs of each component. It seems to have better support for adding new components from the GUI without having to write code than GRC. Designs are saved as XML files.

PROTOTYPE SPACECRAFT TELEMETRY RECEIVER

OVERVIEW

This section of the report is intended to document the design of a prototype SDR system for receiving spacecraft telemetry. There are two particular spacecraft that are of specific interest. One is the Midshipman Space Technology Applications Research (MidSTAR) spacecraft, operated out of the U.S. Naval Academy in Annapolis, MD. The other is the Cosmic Hot Interstellar Plasma Spectrometer (CHIPS) spacecraft, operated out of the University of California's Space Sciences Laboratory in Berkeley, CA. Both spacecraft operate on an S-band downlink, and both are using standard internet protocol (IP) encapsulated in High-level Data Link Control (HDLC)/frame-relay packets for all their communications, just like any other wide area network (WAN) link on the ground.

The MidSTAR ground station has an antenna dish and feedpoint that is capable of tracking and receiving signals from either spacecraft, and it was made available for use by this project. Using this facility and the USRP, it will be possible to test the prototype SDR using signals from an actual on-orbit spacecraft. This will allow for more realistic "real-world" testing of the prototype than would be possible otherwise.

MIDSTAR-1 TELEMETRY

MidSTAR-1 is the primary focus of the SDR telemetry receiver prototype project. MidSTAR-1 operates on an S-band downlink frequency of 2202.26 MHz. Data are modulated using GMSK, with a nominal frequency deviation of about 35 KHz. There is no forward error correction on the link, and binary data is encoded as non-return-to-zero-inverted (NRZI). The binary data consists of a synchronous bitstream at a data rate of 76,800 bits/sec. This bitstream is framed using HDLC/frame-relay, and each frame carries a payload of a standard IP packet, using the Multiprotocol over Frame Relay (MPoF) standard described in RFC-2427. This configuration is widely used on the wide area network (WAN) links of terrestrial routers and simplifies interfacing the MidSTAR ground station with the network.

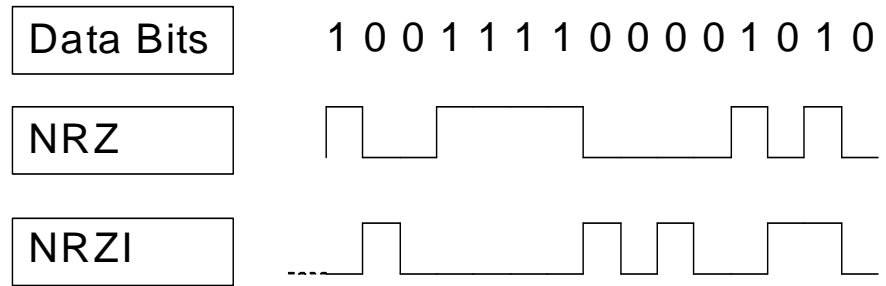


Figure 9. NRZI Encoding

The NRZI coding used results in a bit transition for every 0 in the bitstream and no transition for every 1. The HDLC framing employs a concept called “bit-stuffing,” where every sequence of five consecutive 1’s in the data has a 0 inserted after it. The combination of NRZI and HDLC insures that there are always transitions in the bitstream, even in the presence of long unbroken sequences of either 1’s or 0’s. This is important for clock recovery, as it provides enough edges for the bitsync hardware (or clock recovery software) to lock onto.

MidSTAR’s telemetry is contained within User Datagram Protocol (UDP) packets. Some of these contain fixed-size binary data; others have variable-sized ASCII data representing the output of a single command. Still others are file-transfer packets using the Multicast Dissemination Protocol (MDP). Different types of telemetry data are segregated by UDP port number.

One problem with MidSTAR’s link format is the ratio of 1’s to 0’s. Ideally, this should be 1:1, resulting in the average being exactly half way between the signal levels corresponding to 1 and 0. In an analog receiver, this average would occur at 0 volts DC, providing an easy place to put the threshold to “slice” the analog data into binary bits. But MidSTAR’s average is *not* zero. Although statistically it stays close to zero, it wanders all over the place based on the data content. Even worse, when there are no data to send, HDLC outputs a continuous stream of “flag” bytes. These have a binary value of 01111110, so a continuous stream of them looks like 011111100111111001111110.., and the NRZI encoding of the bitstream looks like either 011111110111111101111111.. or 100000001000000010000000.., both of which have an average that is very far offset from the midpoint level. This “DC offset” problem has caused much of the difficulty in properly decoding the MidSTAR telemetry with hardware, and this is one of the areas for improvement via an SDR.

CHIPSAT TELEMETRY

Receiving CHIPSat telemetry is a secondary goal for this prototype. It was chosen for its similarity to MidSTAR-1 and for the fact that the MidSTAR ground station at the U.S. Naval Academy lies within the footprint of CHIPSat’s transmitter when it is downlinking to the NASA ground station located in Wallops Island, VA, allowing the possibility of “listening in” during a pass there.

CHIPSat operates on an S-band downlink frequency of 2204.80 MHz. Data are modulated using FSK. There is no forward error correction on the link, and the binary data are encoded as BiPhase-S. The binary data consists of a synchronous bitstream at a data rate of 230,400 bits/sec. Like MidSTAR, the bitstream is framed using HDLC/frame-relay, and each frame carries a payload of a standard IP packet.

BiPhase-S coding has a transition at the boundary of every bit cell. In addition, it has an extra transition in the middle of every bit cell that contains a 0 bit.

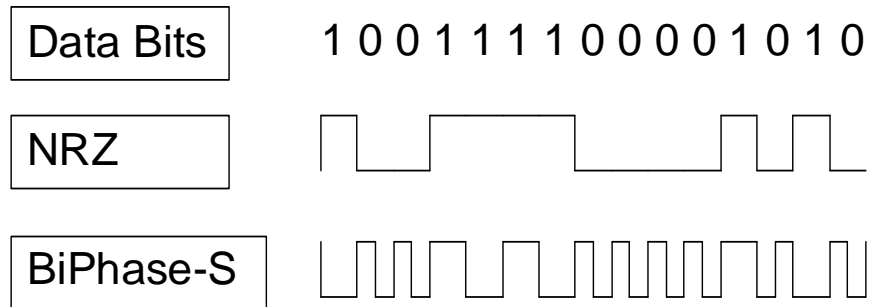


Figure 10. BiPhase Encoding

The BiPhase-S coding has some advantages and disadvantages. One of the big advantages is that the average of the signal is guaranteed to be zero, regardless of the data content, thus avoiding the problems that MidSTAR has. Another advantage is that it has edges associated with *every* bit cell, allowing for a more positive lock during clock recovery. But all these advantages come at a price. Because there are transitions at both the edges and the centers of bit cells, the bandwidth required is *double* that of the data rate. As a result, the CHIPSat downlink of 230,800 bits/sec only supports an HDLC *data* rate of 115,200 bits/sec.

MODULATION AND DEMODULATION

There are several forms of frequency-based digital modulation schemes (FSK, MSK, and GMSK) in common use. It is necessary to understand them in order to construct an effective demodulator for the MidSTAR telemetry data.

FSK is simple Frequency Shift Keying, where the modulator switches between two different frequencies based on a binary input. No attempt is made to avoid any discontinuities at the change. It works, it's easy, but it doesn't make optimal use of bandwidth. The sidebands are large and have a typical $\sin(x)/x$ distribution.

MSK (Mean Shift Keying) is similar, but it attempts to maintain zero phase discontinuity at the change. This has the effect of drastically reducing the bandwidth by significantly reducing amplitude of all the sidebands, leaving just the central peak. But doing this properly is difficult. The two frequencies must have a ratio of exactly 2:1, the change must occur on a zero-crossing, and the lower frequency must be an integer multiple of the data rate. This means that the data clock and the carrier must be in synchronous lock with each other at all times.

The third modulation, GMSK (Gaussian Mean Shift Keying), appears to be a compromise between the first two. The two frequencies can be unrelated, and changes can occur at any point on the waveform, but the change is blended smoothly from one frequency to the next. This has the effect of removing the high frequencies that would have been generated by an abrupt discontinuity. In practice, this is accomplished by running the digital data stream through a "root raised cosine" filter before applying it to the frequency modulator. The bits come out of the filter with a characteristic "rounded" look, and the frequency of the modulated waveform changes smoothly from one value to the other. This also reduces the bandwidth by suppressing the sidebands, but not as much as MSK. GMSK is much easier to implement than MSK.

MidSTAR-1 uses GMSK.

BITSLICER

The job of the software bitslicer is to convert the stream of “soft bits” into digital 1’s and 0’s. Soft bits are the digital representation of the analog voltage coming out of the demodulator. The main problem with the MidSTAR-1 HW ground station is that it does not always decode the bitstream properly, even when there is plenty of signal. This is due to a combination of factors that have the end result of causing the HW bitslicer to make an incorrect decision. Because there is no forward error coding at all on this link, even the loss of one bit causes the loss of an entire HDLC frame, and thus an entire IP packet.

Because of the previously mentioned “DC offset” problem, it is not possible to simply slice the soft-bit values at zero and declare all those greater than zero a digital 1 and all those less than zero a digital 0. The ground station hardware uses an adaptive-level bitslicer circuit that attempts to compensate for this problem, but it is not entirely effective.

The basic idea for the software bitslicer was to make an enhanced version of the adaptive-level bitslicer that was implemented in hardware. The hardware bitslicer tracks the peaks and valleys of the demodulator output and sets the slicing point right in the middle at the median value instead of at the average value. It works reasonably well, but the peak tracker does not respond quickly when the peak value abruptly decreases. Similarly, the valley tracker does not respond quickly when the minimum value abruptly increases. Because of the nature of our NRZI HDLC data, this happens often. In addition, the hardware circuit does not track properly when both the peaks and valleys are on the same side of the zero axis.

CHOICE OF DEVELOPMENT ENVIRONMENTS

As previously noted, there are a number of different SDR development environments available, some open source and some commercial. Because of the low budget and short schedule of this project, only open-source solutions were considered. Primary among the open source are GNU Radio and OSSIE. In addition, both of these support the low-cost USRP hardware that was going to be available for use.

GNU Radio was developed by the GNU open-source community. It is an object-oriented core framework that is not built on top of CORBA, nor is it SCA-compliant. It uses C++ for the “heavy lifting” needed by the signal processing blocks and Python as the scripting language for interconnecting and configuring the blocks. Although less fully developed than a full-blown SCA core framework, it provides all the pieces needed to construct a spacecraft telemetry receiver. Numerous tutorials are available, and there is a large, active support group online. It installs and runs on Mac OSX, most flavors of Linux, and Windows.

There is also a graphical front end for GNU Radio called the GNU Radio Companion (GRC). GRC is an open-source GUI development tool that allows the user to quickly design GNU Radio-based applications by placing and connecting graphical elements representing GNU Radio’s processing blocks. Easy addition of user-created processing blocks is also supported, but it does require writing a simple Python code “wrapper.” Designs are saved as XML files.

OSSIE was developed at Virginia Tech as an open-source implementation of the JTRS SCA 2.2 core framework. Its installation and operation are more complicated than GNU Radio, and it requires the installation of specific tools that may disrupt other functions on the same

computer. In addition, it only runs on two specific versions of Fedora Linux. The OSSIE developers recommend running it either on a dedicated machine or on a virtual machine running under VMware, although this may cause performance degradation.

Included with OSSIE is the OSSIE Waveform Developer (OWD). This open-source GUI application gives the user an interactive way to construct an SCA-compliant waveform from a library of available components. New components can also be created and added to the library. Unlike most SDR GUI tools, it does not use the intuitive “boxes connected with lines” paradigm. Instead, it uses nested tree-lists to show the parameters, inputs, and outputs of each component. It seems to have better support for adding new components from the GUI without having to write code than GRC. Designs are also saved as XML files.

Based on the easier installation and operation, more intuitive GUI front end, and strong online developer support, GNU Radio was selected as the development environment for this project.

INITIAL PROTOTYPE DESIGN FOR MIDSTAR TELEMETRY RECEIVER

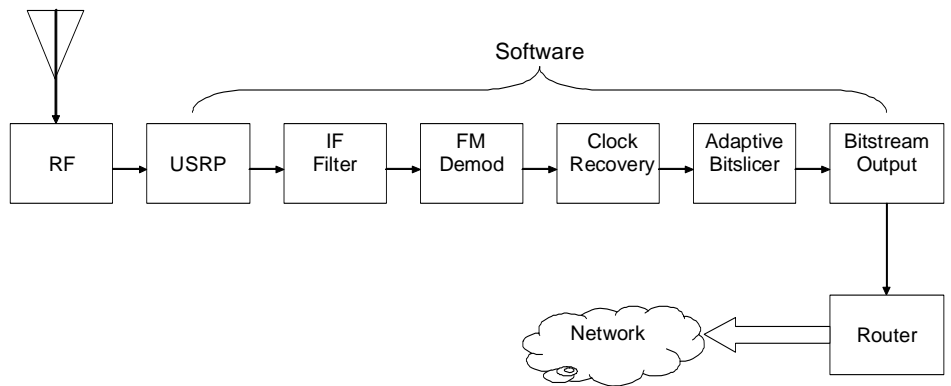


Figure 11. Initial Design Block Diagram

RF Section

Early on, it was decided to use part of the existing MidSTAR receiver to convert the 2 GHz RF down to a 20 MHz IF. This allows ignoring the complication of Doppler correction because the ground station software is already tuning the receiver by small amounts during a pass to compensate for it.

ADC and Decimation (USRP)

The LFRX daughterboard for the USRP has a frequency range of 0 – 32 MHz and was chosen for acquiring the 20-MHz signal. The USRP will perform the ADC, decimation, and quadrature signal generation using the USRP hardware and the existing “USRP Source” processing block. A decimation factor will be chosen that will yield a sample rate of at least twice the data rate.

IF Filter

Because the USRP does “direct conversion” down to a 0 Hz IF, it is possible to simply use an existing “Low-Pass Filter” processing block to implement the IF filter function. This digital FIR filter will limit the noise energy going into the demodulator and improve the signal-to-noise ratio. This is one area where the SDR is expected to exceed the performance of the HW receiver.

Demodulation

Because GMSK is a form of frequency modulation, a simple FM demodulator was chosen using the existing “Quadrature Demodulator” processing block. The values coming out of the FM demodulator will represent the demodulated GMSK data but will still be at the sampling rate, not the original bitstream’s data rate.

Clock Recovery

Using the existing “Clock Recovery” processing block, the stream of values will be resampled at the original bitstream’s data rate. The resultant stream will have one value per bit-time. The values will still be “soft bits”; that is, they will have a range of values, not a simple digital 1 or 0.

Bitslicer

The job of the bitslicer is to convert the stream of “soft bits” into digital 1’s and 0’s. This is another area where the SDR is expected to exceed the performance of the HW. The software bitslicer will be a new piece of code, written in C++ and integrated into GNU Radio and GRC. It will track the peaks and valleys using a “fast attack/slow decay” algorithm and will dynamically calculate the median of the two. Peaks and valleys will decay towards the median, not towards zero, and the median will be used as the slicing threshold.

Bitstream Output

The original plan was to output the bitstream in real time onto an I/O bit and feed it into a router. This was later found to be impractical with the available hardware and required a different approach.

MIDSTAR FIRST-PASS DATA

MidSTAR-1 was launched on March 8, 2007. During its first three passes over the USNA ground station, its downlink signal was digitized, down-converted to I and Q baseband, and recorded using an SDR-IQ. This inexpensive piece of SDR hardware was able to capture a bandwidth of just under 200 KHz.

A portion of this data was extracted and used to begin the development of the prototype telemetry receiver using GNU Radio and GRC. The initial attempt simply used GNU Radio to read and demodulate the captured data and display the baseband spectrum and the demodulated waveform. Once the various parameters for sampling rate and FM gain were worked out, a recognizable display of HDLC flag bytes emerged, showing the nicely “rounded” bits characteristic of GMSK modulation. Next would come clock recovery and the new bitslicer.

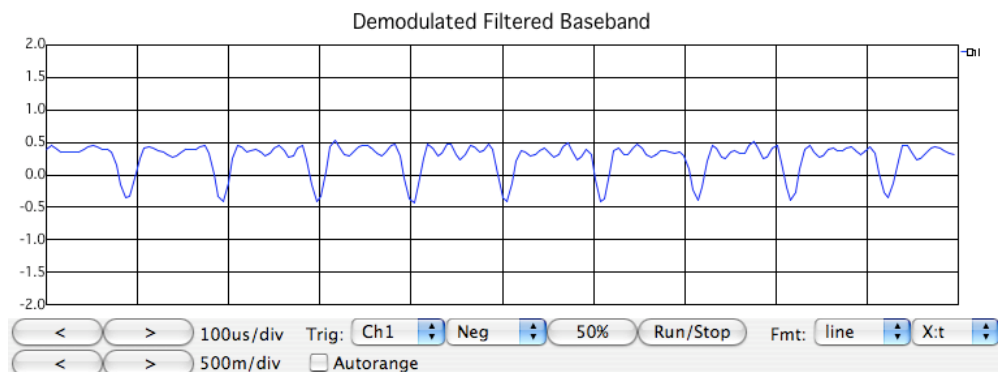


Figure 12. Demodulated Bits from First Pass, Showing NRZI-Encoded Flag Bytes

SIMULATOR AND BITSLICER

When testing the effectiveness of a new bitslicer design, it is more useful to have well-characterized test data than actual pass data. As a result, a crude MidSTAR-1 simulator was developed using GNU Radio and GRC. It consisted of:

- A vector source block that generated a stream of HDLC flag bytes at the correct data rate, sampled at four times the data rate
- A root raised cosine filter to shape the GMSK bandwidth properly
- A wideband FM modulator to produce the complex I and Q samples
- Parameter-driven blocks to add controllable amounts of noise, gain, and offset to the signal

Using this simulator and the already-developed demodulator chain, it was possible to adjust the parameters for the clock recovery block and test and tune the new adaptive bitslicer block.

MIDSTAR LIVE-PASS CAPTURES

Once the USRP hardware arrived, it was connected to a laptop running the GNU Radio software in preparation for using it with a live MidSTAR pass. For purposes of capturing passes for later playback, a simple GNU Radio flow graph was constructed consisting of just a USRP source block, a spectrum display (for monitoring the pass), and a file sink block. The sampling rate was set to 320 ksamples/sec, approximately four times the data rate. Over the course of 2 days, a total of seven MidSTAR passes were captured in this fashion. Each resulted in about 1 GB of raw I and Q data and covered a pass that had about 1 MB of telemetry and downloads that were collected by the HW ground station. Corresponding log files were also saved for later comparison between the HW and SDR results.

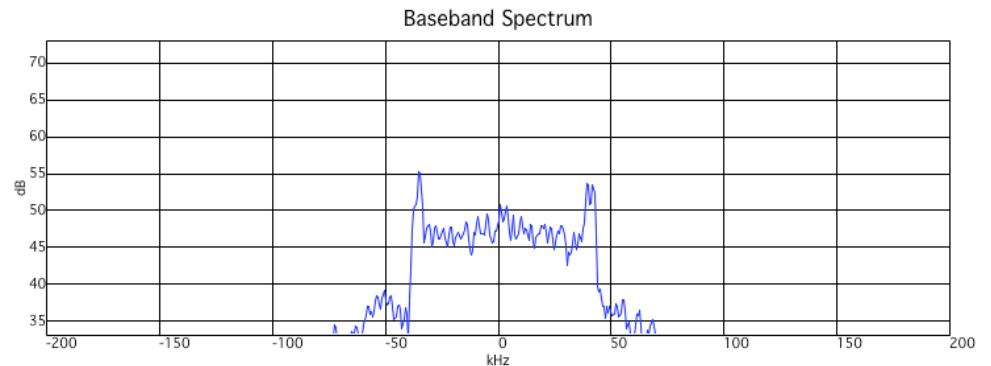


Figure 13. MidSTAR Spectrum During Telemetry Downlink

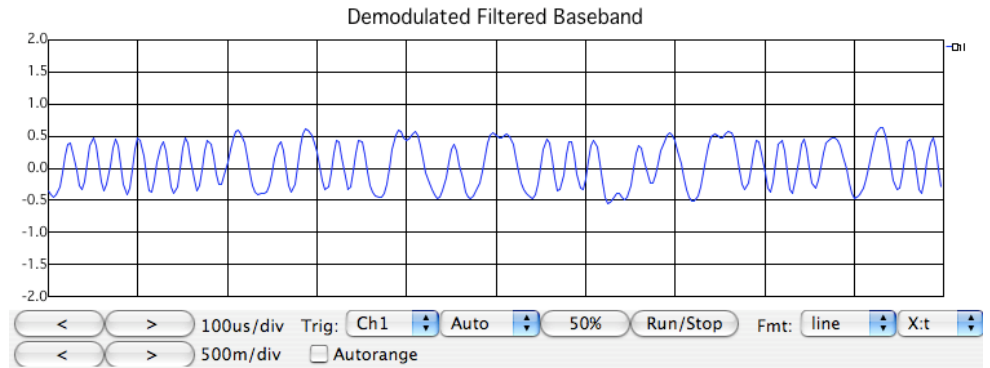


Figure 14. Demodulated MidSTAR Bitstream During Telemetry Downlink

HDLC EXTRACTOR

The original design for the prototype receiver was to take the demodulated resampled bitstream and output it in real time on a digital I/O bit that would then be fed into a router. The router would then take care of all the HDLC deframing, unstuffing, and error detection. This did not prove to be practical in the remaining time, as the only available digital I/O was a printer parallel port, and it could not toggle the bits out at the 76,800 bits/sec rate. As a result, it became necessary to code a full HDLC decoder, including CRC-16 error detection. This code accumulated statistics on the total good bytes, good frames, and various types of errors. As a final bonus, it was able to successfully take the extracted IP packets and put them out on the network via Ethernet. In effect, it took the place of the router.

BITSLICER TUNING AND CLOCK RECOVERY

Once the HDLC decoder was in place, the captured pass data could be run through the SDR processing chain and then through the decoder. This initially produced some surprising results. The first data sets through the SDR only extracted 25% as much good data as the HW receiver had! Adjusting the bitslicer's decay-rate parameter (alpha) only produced a modest 14% improvement. Eventually, it was discovered that adjusting one of the clock recovery block's parameters (Gain_Mu) resulted in a dramatic 400% improvement. It seems that the optimum parameter values for data that consist of only HDLC flag bytes (like the simulator) are *not* the same for bitstreams that have actual data.

PERFORMANCE RESULTS

In the end, the performance of the SDR varied depending on the data set used. In all cases, it was better than the hardware receiver, extracting anywhere from 5% to 23% more good data bytes than the HW. In general, it seemed that the SDR was better at extracting data out of noisier, more marginal signals than the HW receiver.

CHIPSAT LIVE-PASS CAPTURE ATTEMPT

Due to the extra time spent building the unplanned HDLC decoder software, it was not possible to make more than a cursory attempt at receiving CHIPSat telemetry. The USRP and capture software was again brought to the USNA ground station and connected to the HR receiver's 20-MHz IF output. Different ground station software was run to tune and track CHIPSat instead of MidSTAR. A single pass was captured and run through an FM demodulator, with displays of both the spectrum and the demodulated signal. It quickly became apparent that the ground station's CHIPSat Doppler correction tuning was not working properly. The spectrum started on one side of the graph and marched across off of the other. But there was about 1 minute right in the middle of the pass when the spectrum

was centered enough in the passband to demodulate a good signal. And there it was, an unmistakable biphasic waveform, clearly showing the 01111110 pattern of an HDLC flag byte.

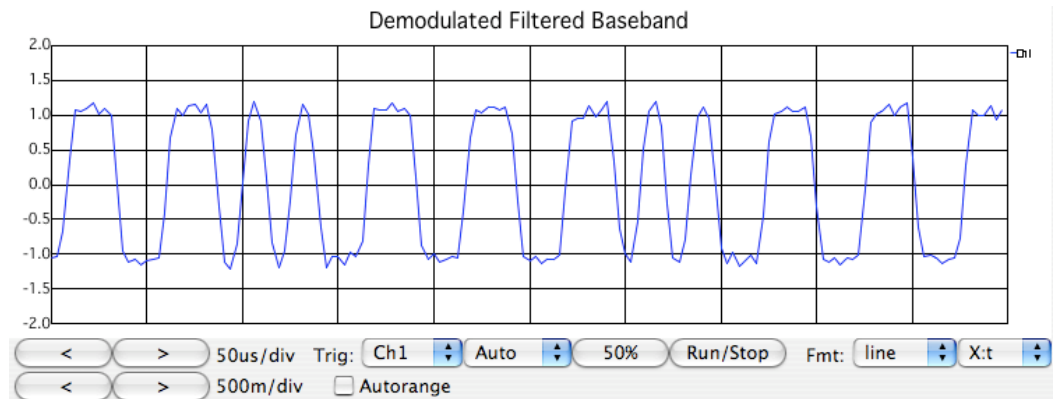


Figure 15. Demodulated CHIPSat Bitstream Showing BiPhase-Encoded Flag Byte

This section summarizes the design of the prototype MidSTAR telemetry receiver. It consists of nine signal processing blocks strung in series:

FINAL PROTOTYPE DESIGN

- USRP Source
- IF Low-Pass Filter
- Quadrature Demodulator
- Video Low-Pass Filter
- Clock Recovery
- Adaptive Binary Slicer
- NRZI-to-NRZ Converter
- Unpacked-to- Packed Converter
- File Sink

Additional blocks were added for scope and spectrum displays of various signals, but these are not necessary for operation.

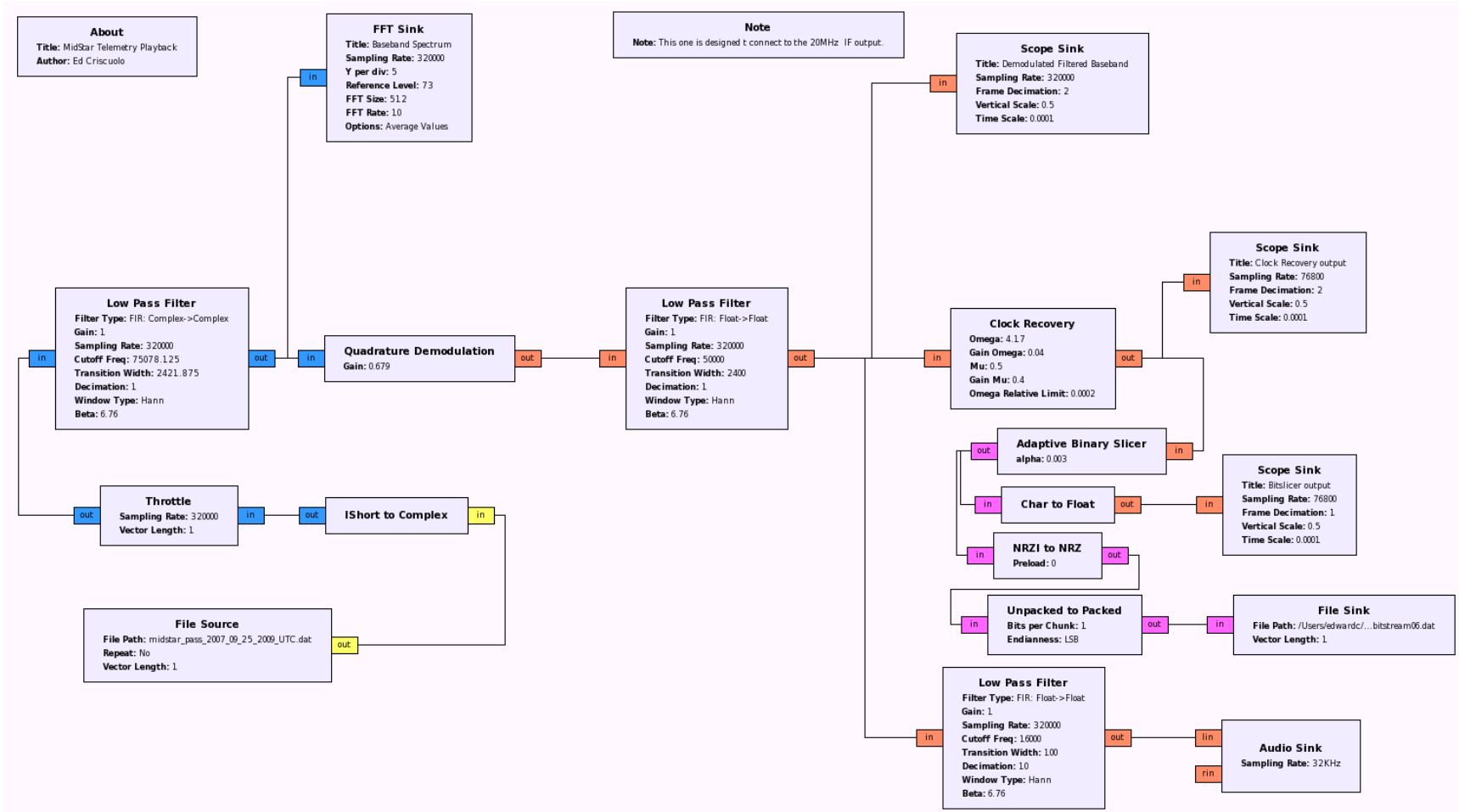


Figure 16. SDR Telemetry Receiver Prototype Block Diagram

USRP SOURCE

This block interfaces with the USRP hardware, including its RF daughterboards. It outputs a stream of sample values as digitized I and Q values. These represent real and quadrature values of the input signal, decimated and then down-converted to a center frequency of zero. The block's primary parameters are frequency, decimation, and gain. The RF section consists of a 0–32 MHz LFRX daughterboard taking its input from the 20-MHz IF output of the MidSTAR ground station's receiver (This allows ignoring the issue of Doppler correction and focuses on the demodulation and decoding). Frequency is set to 20 MHz and decimation is set to 200. Because the USRP samples at 64 msamples/sec, this works out to 320 ksamples/sec. This was chosen in order to have about four samples per symbol at the data rate of 76,800 bits/sec.

IF LOW-PASS FILTER

Having such a high sample rate gives an effective bandwidth of 320 KHz. This is much wider than necessary to demodulate the MidSTAR signal and would result in a loss of signal-to-noise ratio (SNR) when all that bandwidth is fed to the demodulator. The purpose of the IF Filter block is to reduce the bandwidth down to the minimum required. The block's main parameters are cutoff frequency and transition width. The transition width is set to the data rate/32, as per recommendations in the GNU Radio examples. The cutoff frequency was started out at 80 KHz (giving a 160 KHz bandwidth) and fine-tuned to get the optimum performance on real data. It ended up at a value of 77.5 KHz (155 KHz bandwidth) minus the data rate/32, so that the rolloff starts at 75.078 KHz.

QUADRATURE DEMODULATOR

The quadrature demodulator block performs a simple FM demodulation. Because we are working with complex (I and Q) values, it is a simple matter to calculate the phase angle of a given sample by applying an arctangent to the real and imaginary components. And because frequency is just the rate of change of the phase angle, it can be easily obtained by taking the difference of successive phase angles. The output of this block is a single stream of real (noncomplex) values representing the frequency. The block's main parameter is the gain, which is a multiplier applied to the phase-angle differences. This is set to the sample rate divided by two PI times the maximum frequency deviation, as per recommendations in the GNU Radio examples. Previous tuning of the MidSTAR ground station HW had determined that MaxDeviation should be set to 75.

VIDEO LOW-PASS FILTER

The term "video" is a misnomer, as our system has nothing to do with video. It's just a legacy radio term used to describe the output of the demodulator before it has been filtered. Anyway, this block's purpose is to remove any high-frequency noise from our "real" signal before trying to extract data bits out of it. It was initially set with a cutoff frequency of the data rate (76.8 KHz) and then fine-tuned with real data for optimum performance. Unexpectedly, this turned out to be 50 KHz! The sample runs clearly showed an improvement in "goodput" as the cutoff was reduced to 50 KHz.

CLOCK RECOVERY

This block has been the most mysterious, and troublesome, of the whole system. Its function is to take a stream of samples representing a synchronous digital stream and resample it at the clock rate with the digital bits properly aligned. It outputs a stream of "soft" bits that are still real values that need to be converted (sliced) into digital 1's and 0's. The block is based on the Mueller/Muller algorithm, which is somewhat difficult to understand. It has a number

of parameters, some of which are very sensitive. In addition, there are recommended relationships between the parameters that limit the number of truly “independent” variables. Of them, the only sensitive one was “Gain_Mu,” which was determined to have an optimum value of 0.4 based on maximizing “goodput” on real data. This was a very different value than the one that produced the best results on a data stream that consisted of only HDLC “flag” bytes.

ADAPTIVE BINARY SLICER

This block is one of the two new processing blocks that was designed and coded specifically for this project. Its purpose is to take a stream of “soft” bits and slice them into 1’s and 0’s based on a slicing threshold that is the median value (not the mean) of the max and min values. The max and min values are acquired using a “fast attack, slow decay” design that mimics the ground station’s hardware sample-and-hold circuits. The reason for using the median value instead of the mean is that an NRZI-encoded HDLC data stream does not have an average zero DC bias. Much of the time, it’s just a back-to-back stream of idle flag bytes. In NRZI form, these look like a sequence of one 0 and seven 1’s or one 1 and seven 0’s. Each of these streams has an average that is biased above or below the median value. And because this data stream was filtered at the transmitter to limit its high-frequency components (to limit bandwidth), the demodulated bits don’t have straight up-and-down transitions, but have a “rounded” appearance. Because of this, it is important to slice at the correct (median) place. Slicing at other than the median can produce a “narrower” or “fatter” bit, and cause an error in the adjacent bit. This is known as intersymbol interference (ISI). The block has only one parameter, alpha, which is the exponential decay factor that is used in the “slow-decay” part of the algorithm. Tuning with real data produces an optimum alpha of 0.003.

NRZI-TO-NRZ CONVERTER

This block is the other one that was specifically created for this project. It takes a stream of bits, one per byte, and converts them from NRZI to NRZ. This is a simple process of outputting a 0 if the input has toggled from its previous value or a 1 if it remained the same.

UNPACKED-TO-PACKED CONVERTER

This block takes the bits that are one-per-byte and packs eight of them into a single byte. It has a single parameter that determines if the bits are put into the MSB or the LSB first. In this case, it’s LSB first. The output is a stream of bytes that represents the decoded bitstream, ready to be stored in a file or fed into a router.

FILE SINK

This block takes packed bitstream and puts it into a file. From there it can be postprocessed to perform the HDLC deframing and bit-unstuffing and to put the resultant IP packets back out on an Ethernet port.

FUTURE DIRECTIONS

ENHANCEMENTS

There are several areas for future work that could take this prototype and develop it into a full standalone ground-station receiver.

RF TUNER AND DOPPLER TUNING

The addition of the appropriate RF daughterboard to the USRP would allow it to tune directly to the 2 GHz downlink frequency without depending on the 20-MHz IF output on

the existing HW receiver. This would necessitate developing a method for allowing an outside tracking program to perform fine frequency adjustments in real time during a pass in order to compensate for the Doppler shift induced by the spacecraft's velocity.

HDLC PROCESSING BLOCK

The offline HDLC decoder needs to be turned into a real-time processing block and integrated into GNU Radio and GRC. This would allow the creation of an SDR that would take the place of both the receiver *and* the router, placing the decoded IP packets directly on the network for delivery to the ground station's data capture facility.

BIPHASE DECODER

In order to build a complete CHIPSat SDR, a processing block needs to be created to take the BiPhase-S-encoded data and decode it back into an NRZ bitstream.

POTENTIAL NEW PROJECTS

SCA Implementation

Re-implement the entire MidSTAR Telemetry receiver using an SCA-compliant development environment such as OSSIE. This would potentially allow porting it to other types of SDR hardware besides the USRP.

Productize

Using the prototype as a starting point, develop a standalone product that is integrated with a vendor's ground station command and control software to provide a complete solution.

GLOSSARY

| | |
|--------------|---|
| 16QAM | 16-Level Quadrature Amplitude Modulation |
| ADC | Analog-to-Digital Converter |
| AM | Amplitude Modulation |
| ASIC | Application-Specific Integrated Circuit |
| BPSK | BiPhase Shift Keying |
| CF | Core Framework |
| CHIPS | Cosmic Hot Interstellar Plasma Spectrometer |
| CORBA | Common Object Request Broker Architecture |
| COTS | Commercial Off-the-Shelf |
| DAC | Digital-to-Analog Converter |
| DC | Direct Current |
| DDC | Digital Down-Converter |
| DSP | Digital Signal Processor |
| DUC | Digital Up-Converter |
| FFT | Fast Fourier Transform |
| FIR | Finite Impulse Response |
| FM | Frequency Modulation |
| FOSS | Free and Open Source Software |
| FPGA | Field Programmable Gate Array |
| FSK | Frequency Shift Keying |
| GMSK | Gaussian Mean Shift Keying |
| GPP | General-Purpose Processor |
| GUI | Graphic User Interface |
| HDLC | High-level Data Link Control |
| IDE | Integrated Development Environment |
| IF | Intermediate Frequency |
| IIR | Infinite Impulse Response |
| ISI | Intersymbol Interference |

| | |
|----------------|---|
| JTRS | Joint Tactical Radio System |
| MIDSTAR | Midshipman Space Technology Applications Research |
| MMITS | Modular Multifunction Information Transfer System |
| MPoF | Multi-Protocol over Frame Relay |
| MSK | Mean Shift Keying |
| NATO | North Atlantic Treaty Organization |
| NRZ | Non-Return-to-Zero |
| NRZI | Non-Return-to-Zero-Inverted |
| OMG | Object Management Group |
| OOK | On-Off Keying |
| ORB | Object Request Broker |
| OSI | Open Systems Interconnection |
| OSSIE | Open-Source SCA Implementation::Embedded |
| PM | Phase Modulation |
| QAM | Quadrature Amplitude Modulation |
| QPSK | Quadrature Phase Shift Keying |
| RF | Radio Frequency |
| RTOS | Real-Time Operating System |
| SCA | Software Communication Architecture |
| SDR | Software Defined Radio |
| SDRF | Software Defined Radio Forum |
| UDP | User Datagram Protocol |
| USRP | Universal Software Radio Peripheral |
| WAN | Wide Area Network |
| XML | eXtensible Markup Language |