



Waveform Developer's Guide for the Integrated Power, Avionics, and Software (iPAS) Space Telecommunications Radio System (STRS) Radio

*Mary Jo W. Shalkhauser and Rigoberto Roche
Glenn Research Center, Cleveland, Ohio*

NASA STI Program . . . in Profile

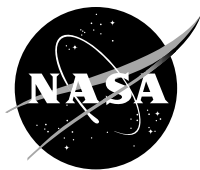
Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program plays a key part in helping NASA maintain this important role.

The NASA STI Program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI Program provides access to the NASA Technical Report Server—Registered (NTRS Reg) and NASA Technical Report Server—Public (NTRS) thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- **TECHNICAL PUBLICATION.** Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counter-part of peer-reviewed formal professional papers, but has less stringent limitations on manuscript length and extent of graphic presentations.
- **TECHNICAL MEMORANDUM.** Scientific and technical findings that are preliminary or of specialized interest, e.g., “quick-release” reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.
- **CONTRACTOR REPORT.** Scientific and technical findings by NASA-sponsored contractors and grantees.
- **CONFERENCE PUBLICATION.** Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.
- **SPECIAL PUBLICATION.** Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.
- **TECHNICAL TRANSLATION.** English-language translations of foreign scientific and technical material pertinent to NASA's mission.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at <http://www.sti.nasa.gov>
- E-mail your question to help@sti.nasa.gov
- Fax your question to the NASA STI Information Desk at 757-864-6500
- Telephone the NASA STI Information Desk at 757-864-9658
- Write to:
NASA STI Program
Mail Stop 148
NASA Langley Research Center
Hampton, VA 23681-2199



Waveform Developer's Guide for the Integrated Power, Avionics, and Software (iPAS) Space Telecommunications Radio System (STRS) Radio

*Mary Jo W. Shalkhauser and Rigoberto Roche
Glenn Research Center, Cleveland, Ohio*

National Aeronautics and
Space Administration

Glenn Research Center
Cleveland, Ohio 44135

Trade names and trademarks are used in this report for identification only. Their usage does not constitute an official endorsement, either expressed or implied, by the National Aeronautics and Space Administration.

Level of Review: This material has been technically reviewed by technical management.

Available from

NASA STI Program
Mail Stop 148
NASA Langley Research Center
Hampton, VA 23681-2199

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
703-605-6000

This report is available in electronic form at <http://www.sti.nasa.gov/> and <http://ntrs.nasa.gov/>

Waveform Developer's Guide for the Integrated Power, Avionics, and Software (iPAS) Space Telecommunications Radio System (STRS) Radio

Mary Jo W. Shalkhauser and Rigoberto Roche
National Aeronautics and Space Administration
Glenn Research Center
Cleveland, Ohio 44135

Summary

The Space Telecommunications Radio System (STRS) provides a common, consistent framework for software defined radios (SDRs) to abstract the application software from the radio platform hardware. The STRS standard aims to reduce the cost and risk of using complex, configurable, and reprogrammable radio systems across NASA missions. To promote the use of the STRS architecture for future NASA advanced exploration missions, NASA Glenn Research Center developed an STRS-compliant SDR on a radio platform used by the Advanced Exploration System program at NASA Johnson Space Center in their Integrated Power, Avionics, and Software (iPAS) laboratory.

Introduction

The Integrated Power, Avionics, and Software (iPAS) Space Telecommunications Radio System (STRS) radio was implemented on the Reconfigurable, Intelligently-Adaptive Communication System (RIACS) platform, currently being used for radio development at NASA Johnson Space Center. The platform consists of a Xilinx® Virtex®-6 ML605 Evaluation Kit, an Analog Devices AD-FMCOMMS1-EBZ radiofrequency (RF) front-end board, and an Axiomtek™ eBOX620-110-FL embedded personal computer (PC) running the Ubuntu® 12.04 LTS operating system. Figure 1 shows the RIACS platform hardware. The result of this development is a low-cost-STRS-compliant platform that can be used for waveform development for multiple applications.

The purpose of this document is to describe how to develop a new waveform using the RIACS platform, the Very High Speed Integrated Circuits (VHSICs) Hardware Description Language (VHDL) field-programmable gate array (FPGA) wrapper code, and the STRS implementation on the embedded PC (eBOX620-110-FL).

This document was written under the assumption that the reader has access to the iPAS STRS radio software and FPGA code. It is recommended that waveform developers obtain copies of the following iPAS STRS radio documentation: iPAS STRS Radio User's Guide (Ref. 1), iPAS STRS Radio Hardware Interface Description (HID) (Ref. 2), and the Programmable Logic Device (PLD) Design Description for the iPAS STRS Radio (Ref. 3). All of these documents are publically available NASA Technical Memorandum (TM).

Design Overview

This section provides a design overview of the STRS. Throughout this document, VHDL signal names are always shown in italics and file names are shown in a monospaced, typewriter-style `Courier` font. VHDL constants are always shown in all capital letters.

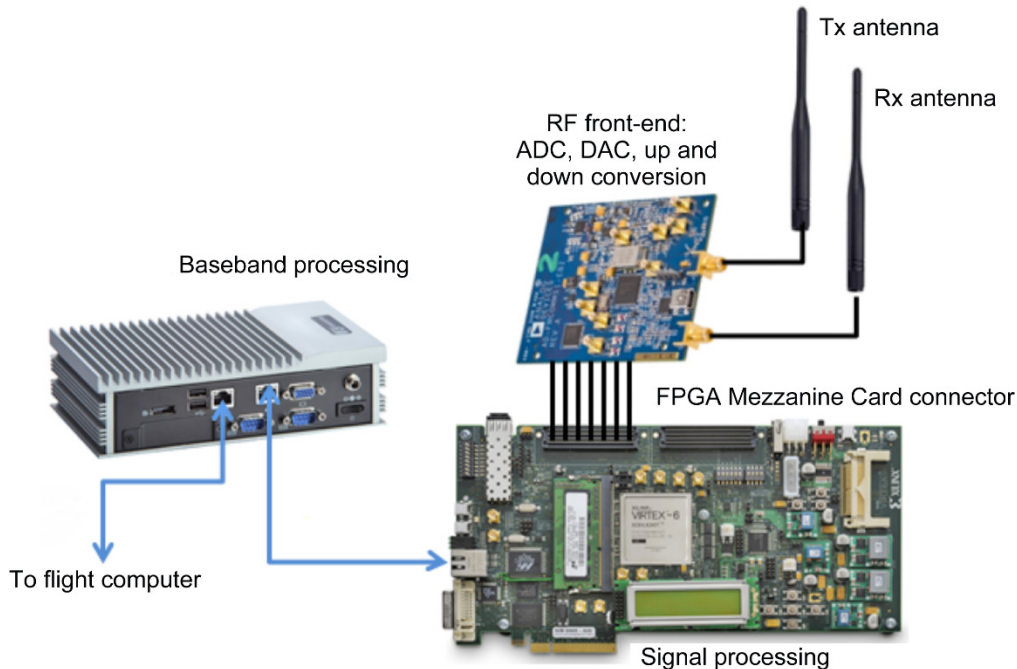


Figure 1.—Reconfigurable, Intelligently-Adaptive Communication System (RIACS) platform components. ADC, analog-to-digital converter; DAC, digital-to-analog converter; FPGA, field-programmable gate array; RF, radiofrequency; Rx, receive; Tx, transmit.

Space Telecommunications Radio System (STRS) Software

The STRS architecture is a framework for the design, development, operation, and upgrade of space-based software defined radios (SDRs), where processing resources are constrained, including the application programming interface (API) between the operating environment (OE) and the waveform application. This document provides detailed instructions on the process necessary to write code on the SDR using the preloaded STRS API and all the hooks for low-level control functions in the iPAS STRS project. This guide also presents a description of how the radio operates, the internal APIs, the configuration files, constraints, and the execution process. This information is necessary for a potential software developer to understand how the code is organized in a logical hierarchy and how the example waveform control code can be modified to implement unique functionality outside of the examples provided. This example waveform control code (iPAS waveform) is referred to as “WFIPAS2” throughout this document.

Field-Programmable Gate Array (FPGA)

The purpose of the FPGA design is the implementation of the signal processing functions of the STRS radio architecture in the iPAS RIACS platform. The FPGA design consists of two parts—the FPGA wrapper and the test waveform. The FPGA wrapper implements each of the following platform interfaces:

- Ethernet communication to the embedded processor for commanding and data streaming
- Digital-to-analog converter (DAC) and analog-to-digital converter (ADC) interface to the RF board
- RF board control and configuration
- FPGA clocking

The test waveform does not fully implement all the signal processing functionality for a radio, but it exercises and demonstrates each interface in the FPGA wrapper. A future user of the platform for an STRS radio would use the FPGA wrapper and replace the test waveform with customized radio signal processing functions.

The FPGA design is required to receive and process commands from an embedded processor and provide command control to the test waveform. The FPGA must also receive and transmit streaming data from and to the embedded processor. To test transmit-side (Tx-side) streaming, it can perform bit error rate (BER) testing on Tx-side pseudorandom bit sequence (PRBS) streaming data. It can also generate PRBS streaming data packets for a receive-side (Rx-side) streaming data source. The test waveform generates sine waves for the in-phase (I) and quadrature (Q) inputs to the RF transceiver. Sampled I and Q outputs of the RF transceiver can be streamed to the embedded processor, where the outputs can be plotted to demonstrate proper functionality of the RF board and its interfaces.

Top-Level Design Description

Figure 2 shows how the STRS standard was implemented on the RIACS platform. The general purpose module (GPM) is the implementation of the STRS command infrastructure on the iPAS radio. It houses the OE and presents a communication conduit for commands and data to and from the signal processing module (SPM). The GPM is where the general purpose processor (GPP) hardware is contained and accessed by the operating system running the STRS project files.

The SPM encompasses the FPGA design, which consists of the following:

- (1) The FPGA wrapper that implements all the interfaces to the FPGA and abstracts them from the waveform.
- (2) The waveform, which is the FPGA implementation of the radio signal processing functions.

The test waveform created for the iPAS STRS radio does not fully implement all the signal processing functionality for a radio, but it exercises and demonstrates each interface in the FPGA wrapper. A future user of the platform for an STRS radio would reuse the FPGA wrapper and replace the test waveform with customized radio signal processing functions.

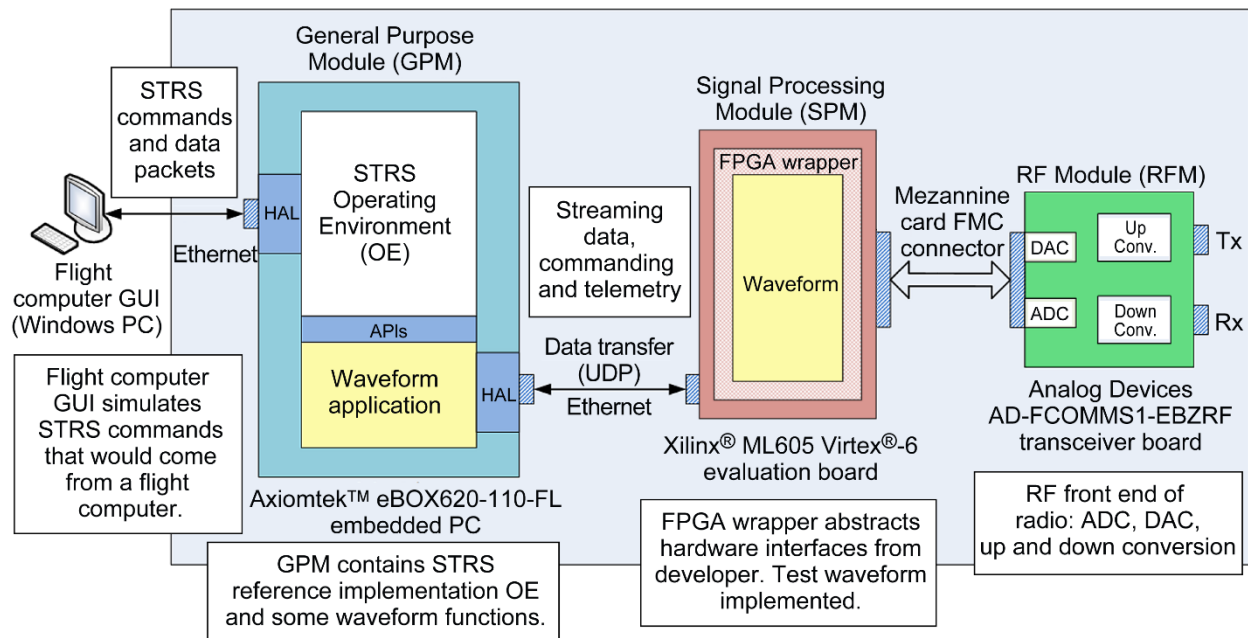


Figure 2.—Space Telecommunications Radio System (STRS) implementation on the Reconfigurable, Intelligently-Adaptive Communication System (RIACS) platform. ADC, analog-to-digital converter; APIs, application programming interfaces; DAC, digital-to-analog converter; FMC, FPGA Mezzanine Card; FPGA, field-programmable gate array; GUI, graphical user interface; HAL, hardware abstraction layer; PC, personal computer; RF, radiofrequency; Rx, receive; Tx, transmit; UDP, User Datagram Protocol.

The RF module (RFM) provides the analog and RF signal processing for the iPAS STRS radio. On the Tx side, the RF front-end board (AD-FMCOMMS1-EBZ) board takes complex I and Q inputs (16 bits) into a high-speed, DAC to create an analog signal. The DAC output signal is up-converted to the desired RF frequency by a quadrature modulator. On the Rx side, the received RF signal is demodulated using direct-conversion to create I and Q analog signals. The analog signals are digitized using a 14-bit ADC.

Concept of Operation

The flight computer graphical user interface (GUI) simulates the STRS commands that would originate from a typical flight computer. The GPM is implemented on the embedded PC (eBOX620-110-FL) and includes the STRS operating environment and waveform application software. The STRS OE communicates with the waveform application through standard STRS APIs to control and configure the waveform.

The SPM functions are encompassed in the Xilinx® ML605 FPGA board design. The FPGA wrapper abstracts the hardware interfaces from the waveform developer. The test waveform utilizes each of the hardware interfaces within the wrapper to demonstrate that the wrapper is correctly implemented. The GPM sends commands over an Ethernet port to the FPGA to control and configure the waveform. The GPM also streams packetized data to the FPGA and receives packetized streaming data from the FPGA over the same Ethernet port.

The RF front-end board (AD-FMCOMMS1-EBZ) contains a DAC, up-converter, down-converter, and ADC. The FPGA configures the RF board using the Xilinx® Microblaze™ 32-bit Reduced Instruction Set Computer soft processor and sends I and Q data to the DAC. The FPGA also receives down-converted and sampled I and Q data from the RF board.

The test waveform will be able to demonstrate STRS commands for configuration and control of the test waveform, Tx-side streaming data operation, RF front-end board configuration, receive-side streaming data, and STRS telemetry querying.

Space Telecommunications Radio System (STRS) Software Design Description

This section contains a description of the STRS software design.

Software Identification

This software design is implemented on an eBOX620-110-FL embedded PC, which contains an AMD G-Series APU T56N 1.65 GHz processor. This embedded PC has a 200 GB hard drive and is running the Ubuntu® 12.04 LTS operating system.

Development Tools

This GPM design uses the Eclipse Indigo development tool with all the internal compilers for C/C++, as well as other Linux tools, such as GNU Make and Bash Scripting.

Detailed Architecture Design and Block Diagrams

This section describes the conventions needed to understand the design between system components, the internal subsystems, and the STRS class definition in more detail. The subsystems interact to provide the functionality to operate the radio. The components shown in orange represent the high-level subsystems needed to control waveform and applications within the radio platform. The high-level architecture model is shown in Figure 3.

The STRS SDR must have a GPP for waveform control, processing flight computer commands, health and fault monitoring, etc. Using object-oriented design, the STRS Reference Implementation design for the software on the GPP has evolved with various classes that encapsulate the required functionality for the radio.

The internal APIs are shown in orange (Figs. 3 and 4). The APIs used by a STRS application (i.e., a waveform or service) are colored in dark purple. The APIs implemented by a STRS application (i.e., a waveform or service) are colored in light purple. The green indicates a waveform or service that is not part of the STRS infrastructure. The turquoise blue color indicates the flight computer simulator, which will evolve into a flight computer simulator and flight computer interface device on another platform.

The key to understanding the STRS class, object, and subsystem diagrams in Figures 3 and 4 is shown in Table 2.

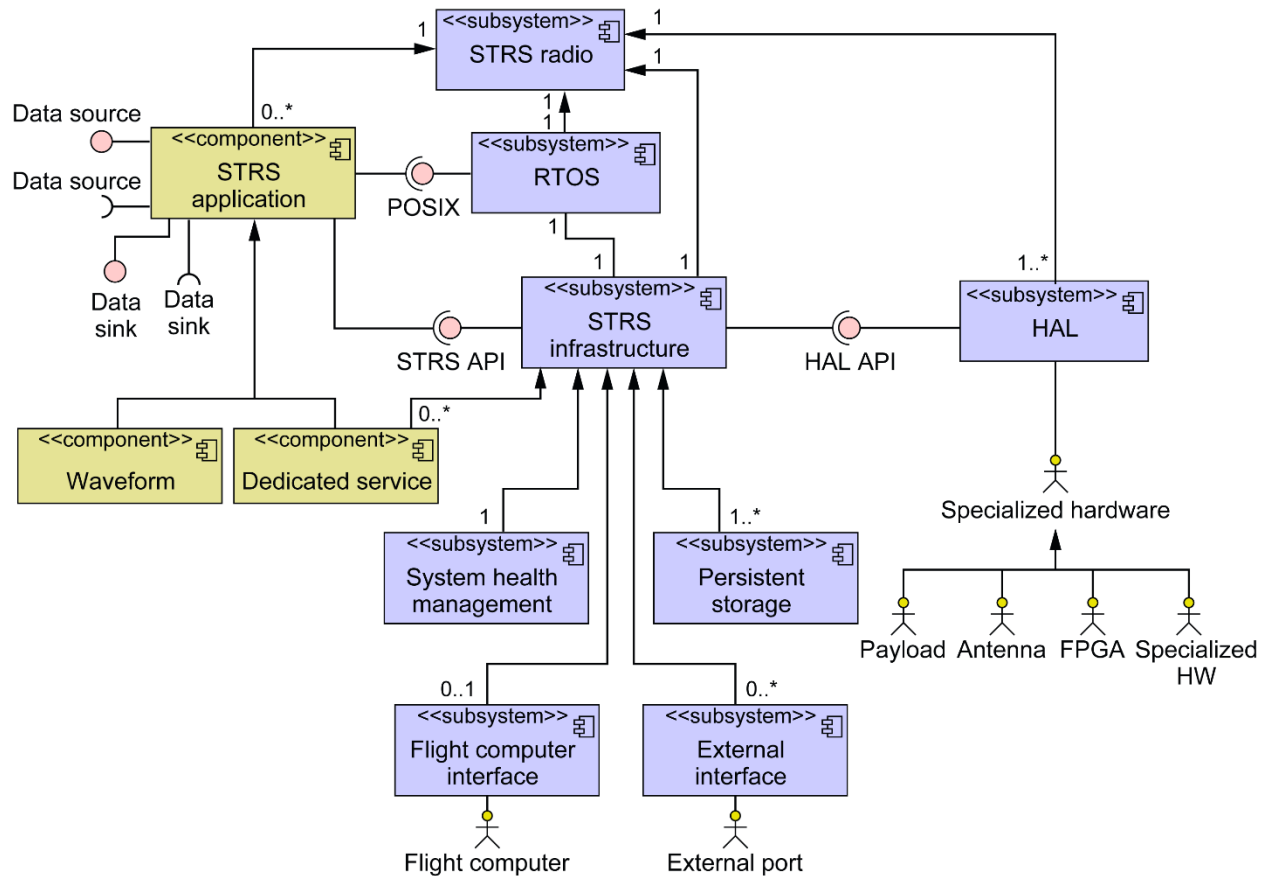


Figure 3.—Top-level Space Telecommunications Radio System (STRS) architecture in Unified Modeling Language (UML). API, application programming interface; FPGA, field-programmable gate array; HAL, hardware abstraction layer; HW, hardware; POSIX, Portable Operating System Interface; RTOS, real-time operating system.

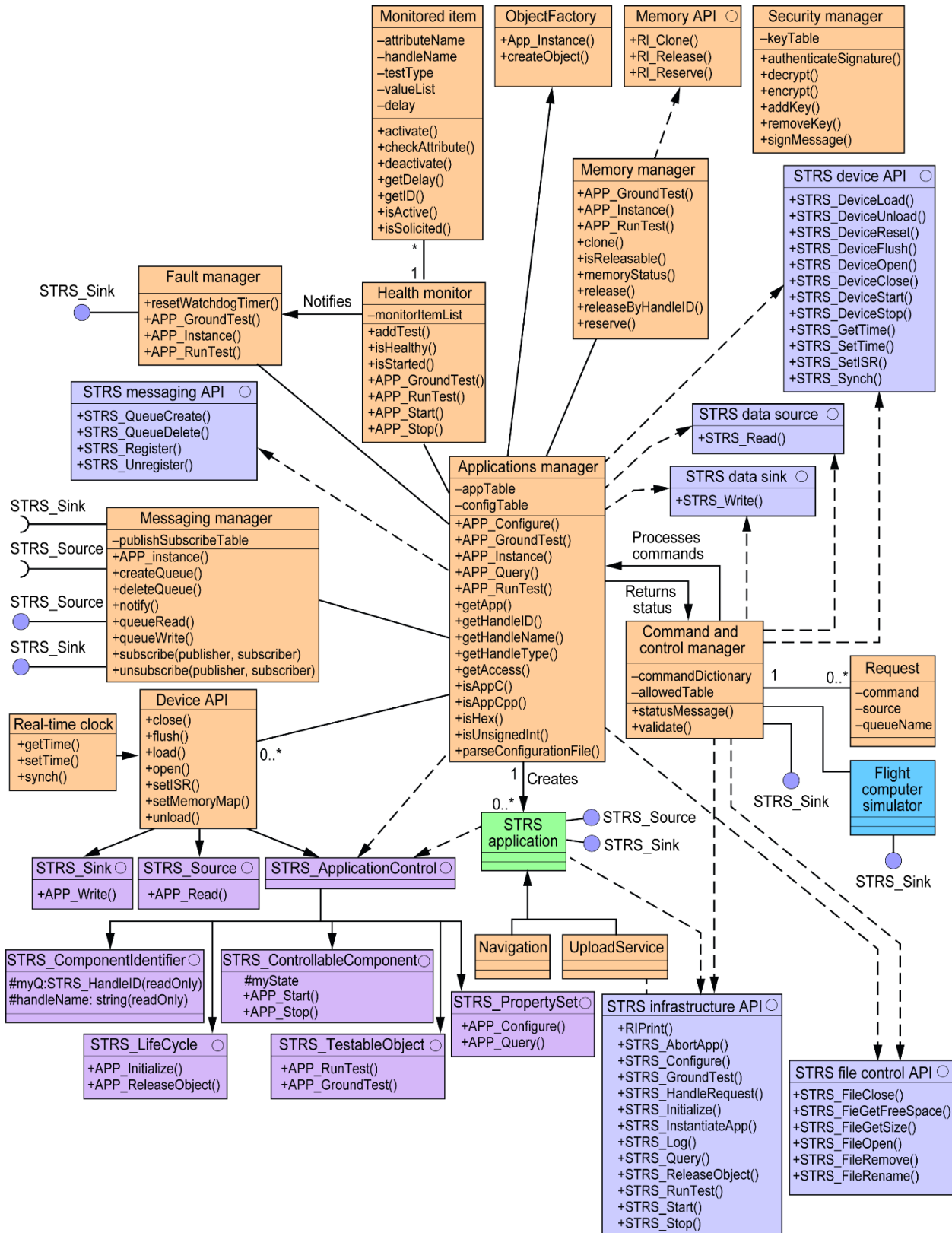

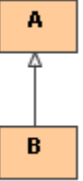
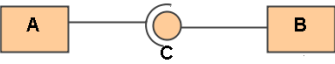
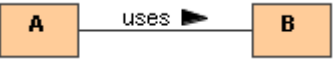
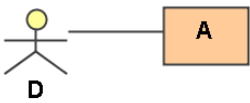


Figure 4.—High-level class diagram for the Space Telecommunications Radio System (STRS) reference implementation. API, application programming interface.

TABLE 2.—CLASS DIAGRAM UNIFIED MODELING LANGUAGE (UML) KEY

Diagram element	Name	Explanation
	Composition	A contains X items of type B; B is a part of the aggregate A; B does not exist independently from A; X may be a number or a range from m to n depicted by “m..n” where n may be an asterisk to indicate no upper limit
	Generalization or inheritance	B is a kind of A; B inherits all properties of A; A is a more general case of B
	Interface	C is an interface provided by B; that is, C contains means to invoke behavior that resides in B; A uses interface C to access B
	Association	A is associated with B; optional description “uses” indicates that A is associated with B such that A “uses” B
	Association	D acts upon A and A responds to D or possibly vice versa; D is normally an actor outside system

Field-Programmable Gate Array (FPGA) Design Description

This section contains a description of the FPGA design.

Hardware Identification

This FPGA design is implemented on a Xilinx® ML605 Rev D evaluation board, which contains a Xilinx® Virtex®-6 XC6VLX240T-1FFG1156C FPGA. An RF front-end board (AD-FMCOMMS1-EBZ) is used for the RF front end.

Development Tools

The development tool used for this FPGA design is Xilinx® Integrated Synthesis Environment (ISE®) Design Suite: System Edition, version 14.4, which includes the Embedded Development Kit (EDK) and Software Development Kit (SDK). The Xilinx® ISE® Simulator (ISim) was used for design simulation of most of the VHDL modules.

Detailed Architecture Design and Block Diagrams

Figure 5 contains a block diagram of the Tx side of the FPGA design. Most of the blocks in the diagram (and subsequent block diagrams) represent hardware description language (HDL) code modules and are labeled with the module or instance name, so that these functions can be easily located in the code. Each block in this diagram (and also in Fig. 6) represents a VHDL module. Each of these blocks and their submodules are described in the Programmable Logic Device (PLD) Detailed Design section of the “PLD Design Description for the iPAS STRS Radio” document (Ref. 3). Figure 5 shows Tx-side wrapper functions, which include the following: clock generation, reset signal generation, and the modules used to receive streaming and command packets and remove Ethernet headers. The block diagram also shows the Tx-side waveform functions, which include command parsing and decoding, conversion of streaming packet data into continuous streaming data, PRBS generation, and I and Q signal generation (sine waves or modulated PRBS data).

Figure 6 contains the block diagram of the Rx side of the FPGA design. The Rx-side waveform performs BER testing of PRBS or streaming data. The Rx-side wrapper packetizes command responses and Rx-side streaming data and controls their transmission over the Ethernet port.

Details of the FPGA wrapper and test waveform can be found in the “PLD Design Description for the iPAS STRS Radio” document (Ref. 3).

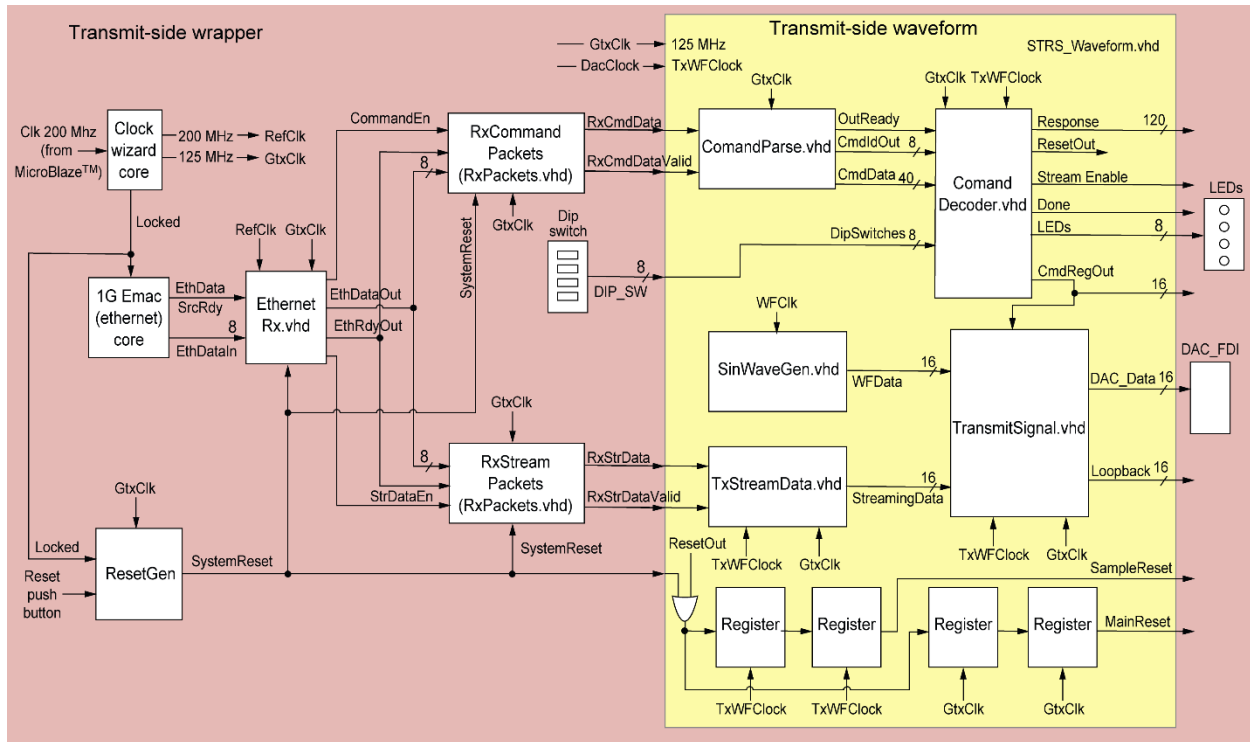


Figure 5.—Transmit-side wrapper and waveform. EMAC, Ethernet Media Access Controller; LEDs, light-emitting diodes.

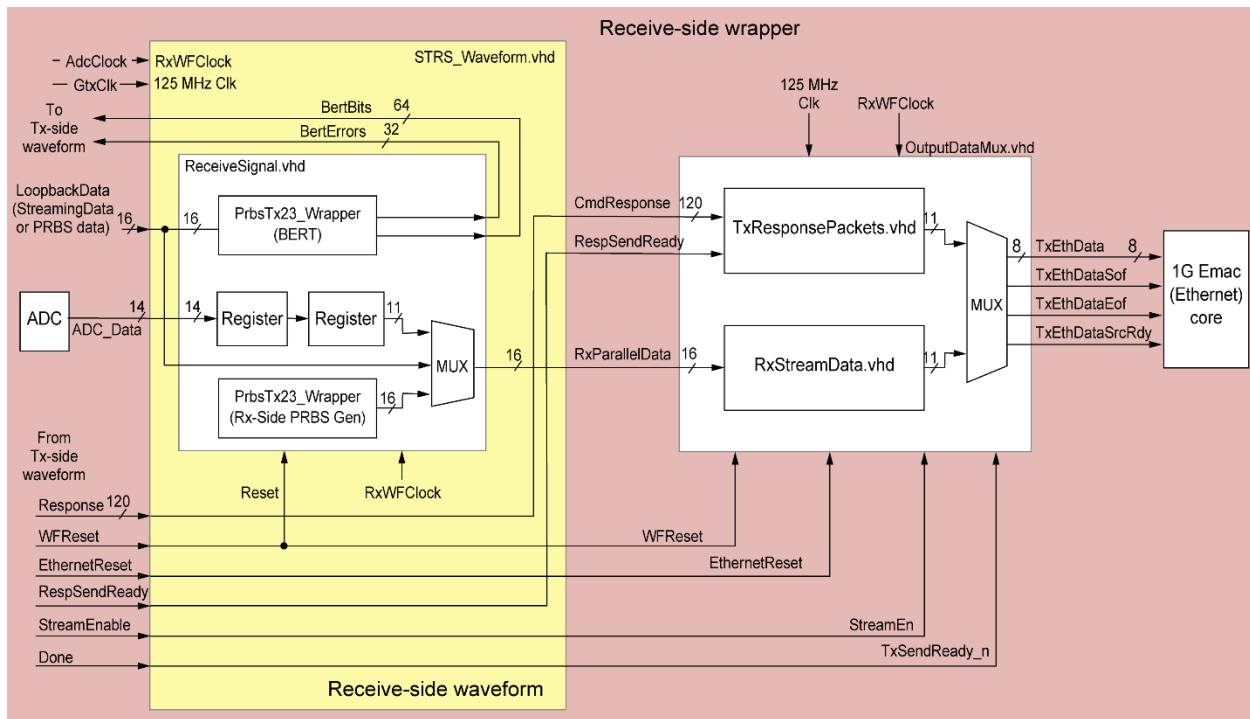


Figure 6.—Receive-side wrapper and waveform. ADC, analog-to-digital converter; BERT, bit error rate tester; EMAC, Ethernet Media Access Controller; MUX, multiplexer; PRBS, pseudorandom bit sequence; Rx, receive; Tx, transmit.

Radiofrequency Module (RFM) Design Description

This section contains a description of the RFM interfaces.

Hardware Identification

The RFM consists of an RF front-end board (AD-FMCOMMS1-EBZ). The RFM was designed to provide the analog front end for FPGA-based radio applications. Detailed information about the RF front-end board (AD-FMCOMMS1-EBZ) can be found in the Analog Devices wiki at https://wiki.analog.com/resources/eval/user-guides/ad-fmcomms1-ebz/hardware/functional_overview.

The RF front-end board (AD-FMCOMMS1-EBZ) is a mezzanine board compatible with the Xilinx® ML605 Rev D evaluation board that can plug into either FPGA Mezzanine Card (FMC) connector on the Xilinx® ML605 Rev D evaluation board. The iPAS STRS radio requires the RF front-end board (AD-FMCOMMS1-EBZ) to be inserted in only the FMC-Low Pin Count (LPC) connector. The Tx side of the high-speed analog board contains a 16-bit DAC, followed by an up-converter and a 20 dB linear amplifier. The Rx side of the board contains a down-converter, followed by a variable gain amplifier and an ADC. The board also contains clock generators and synchronizers as well as frequency synthesizers needed for the operation of the analog components. The RF front-end board (AD-FMCOMMS1-EBZ) is configured by the FPGA board through an Inter-Integrated Circuit (IIC) interface. The IIC interface is converted to a Serial Peripheral Interface (SPI), which is used to configure the components on the RF front-end board (AD-FMCOMMS1-EBZ).

Using the Radiofrequency (RF) Board

The MicroBlaze™ processor is a soft processor core for the Xilinx® Virtex®-6 FPGA that is used in the iPAS STRS radio design to configure the RF front-end board (AD-FMCOMMS1-EBZ). Analog Devices provides a reference design to help developers interface their RF front-end board

(AD-FMCOMMS1-EBZ) with the Xilinx® ML605 FPGA board. The reference design guide is available through their online wiki (<http://wiki.analog.com/resources/eval/user-guides/ad-fmcomms1-ebz>). The reference design guide contains functionality that was not needed for the iPAS STRS radio, so that functionality was removed from the MicroBlaze.xmp Xilinx® Platform Studio portion of the reference design. This left only the functionality necessary to configure and provide clocking to the RF board and the universal asynchronous receiver/transmitter (UART). This provides the waveform developer with the ability to directly connect to the RF board's DAC and ADC with VHDL.

The SDK portion of the reference design SDK was retained in the iPAS STRS radio. The `main.c` file was edited to configure the ADC sampling rate (196.608 MHz), the DAC sampling rate (196.608 MHz), and the Rx RF gain (10,000). The default SDK configuration is provided in the `CompiledDefaultProgram.elf` file.

The MicroBlaze™ processor uses a UART peripheral to display the configuration of the RF front-end board (AD-FMCOMMS1-EBZ). When the MicroBlaze™ processor starts up after a power-on or a reset, the UART will send the following information to a terminal—ADC and DAC sampling rates, variable-gain amplifier gain, and RF signal frequency. The UART configuration necessary for the PC-based receiver is as follows: 57,600 baud, 8-bit data, no parity bit, 1 stop bit, and no flow control. Use of the UART requires the installation of a driver on the PC for the Silicon Labs CP210x Universal Serial Bus (USB) to UART Bridge Virtual COM Port driver located on the Xilinx® ML605 FPGA board. The driver is provided with the Xilinx® Virtex®-6 ML605 Evaluation Kit.

The `main.c` program in the SDK files of the reference design contains the assignments to a structure variable called `defInit` (line 70 of `main.c`). The waveform developer can easily customize the reference design by editing this variable. The developer can change the ADC sampling rate, DAC sampling rate, RF gain, and RF signal frequency. The `defInit` variable for the iPAS STRS radio wrapper is configured as follows:

```
XCOMM_DefaultInit defInit = {    FMC_LPC,           // fmcPort
                                XILINX_ML605,         // carrierBoard
                                200000000,           // adcSamplingRate
                                200000000,           // dacSamplingRate
                                10000,               // rxGain1000
                                2400000000ull,        // rxFrequency
                                2400000000ull};        // txFrequency
```

Waveform Development Process

This section contains a description of the waveform development process.

Overview

Figure 7 shows an example of the configuration that a waveform developer can use to develop new waveforms on the STRS platform. A Windows PC is needed to run the flight computer simulator GUI. This GUI makes it easier for the developer to query status of and send STRS commands to and from the waveform. Alternately, STRS commands can be typed into a terminal on the monitor connected to the embedded PC (eBOX620-110-FL).

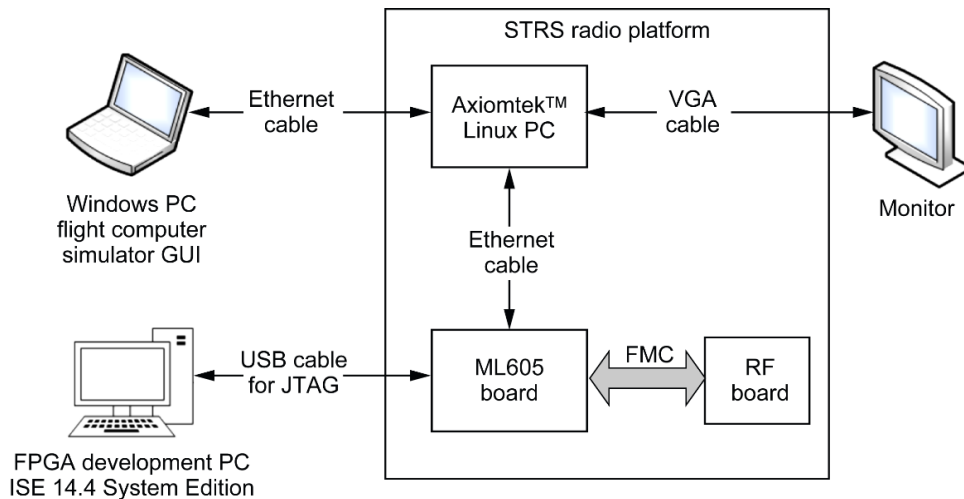


Figure 7.—Hardware configuration for waveform development. FMC, FPGA Mezzanine Card; FPGA, field-programmable gate array; GUI, graphical user interface; JTAG, Joint Test Action Group; PC, personal computer; RF, radiofrequency; USB, Universal Serial Bus; VGA, Video Graphics Array.

The Axiomtek™ monitor is also necessary to modify and develop GPM software. The Eclipse Integrated Development Environment (IDE) was used for the original development, but any similar Linux compatible software IDE can be used. The monitor can also aid in testing and debugging by using a terminal window and the Linux tcpdump command (or Wireshark, etc.) to view the Ethernet traffic between the Axiomtek™ monitor and the FPGA board.

The FPGA code was developed using Xilinx® ISE Design Suite: System Edition version 14.4. A developer of new FPGA waveforms will need a licensed copy of Xilinx® ISE® Design Suite: Logic Edition and Embedded Edition at a minimum, on a development computer.

Space Telecommunications Radio System (STRS) Application Development

To develop the C/C++ code needed for this platform, two template waveforms have been provided—WF1 for C and WF2 for C++. It is assumed that a waveform developer is familiar with writing code in both C and C++. Developers are encouraged to use these waveforms as templates for creating customized code. The provided waveform WFIPAS2 can be used as a template for developers to modify and use the current FPGA example waveform and wrapper, written specifically to demonstrate the capabilities of this platform.

It is recommended that a waveform developer, writing code for the GPM in either C or C++, follow these steps:

- (1) Write the control functions in a separate project and unit test those functions with the hardware outside of the architecture. This practice will ensure that any bugs or issues present in the customized applications are resolved before integration with the standard interface.
- (2) It is recommended that the control functions are written as plug and play modules, not system dependent, and that the developer should use the example functions provided in the NASA–STD–4009 standard as a basis for their development cycle.
- (3) Obtain WF1, WF2, or WFIPAS2 from the repository. These waveforms, all packaged with the Reference Implementation, have containers and implementers that can be used as wrappers for creating the new waveform. WFIPAS2 has specific control applications already integrated to control the FPGA in this specific platform so it is recommended the developer use WFIPAS2.
- (4) Compile and run. The project has make files ready to build any new code that is entered into the system. There is no need to modify these files; add the source into a container (WF1, WF2, or WFIPAS2) and build the application.

The following is an illustrated example of adding a modification for WFIPAS2 for a change in source/sink configuration for the radio. This is a common practice for benchtop testing on new hardware. This is pseudocode to illustrate a point, not actual code. It has been tested and run within the GPM (no FPGA–RF controls), but it is not part of the standard build provided.

- (1) Write separate source and testing (Fig. 8).

```

33 int main()
34 {
35     // Note that the class definition has to be done separately with the proper includes
36     CmdArg=TxDataSourceCommandAccessor::NEWSOURCE
37     return 0;
38 }

```

Figure 8.—Source code for writing and testing changes to the hardware configuration.

- (2) Change the source to be plug and play to allow the code to be reused without modification (Fig. 9).

```

133 int source_changing()
134 {
135     CmdArg=TxDataSourceCommandAccessor::NEWSOURCE
136     return CmdArg;
137 }
138 int main ()
139 {
140     source_changing()
141     return 0;
142 }
143

```

Figure 9.—Source change.

- (3) Obtain the source container (WFIPAS2.cpp in this case) (Fig. 10).

```

332 STRS_Result TxDataSourcePropertyHandler::WriteProperty(const char *PropValue)
333 {
334     STRS_Result Result;
335     WFIPAS2 *TheWfipas = dynamic_cast<WFIPAS2*>(Parent->GetStrsPropSet());
336
337     if (TheWfipas != NULL)
338     {
339         TxDataSourceCommandAccessor::TxSource_t CmdArg = TxDataSourceCommandAccessor::SOURCE_SINEWAVE;
340         TxDataSourceCommandAccessor Cmd = TheWfipas->AccessorFactory<TxDataSourceCommandAccessor>();
341
342         if (strcmp(PropValue, "SINEWAVE") == 0)
343         {
344             CmdArg = TxDataSourceCommandAccessor::SOURCE_SINEWAVE;
345             Result = STRS_OK;
346         }
347         else if (strcmp(PropValue, "NEWSOURCE" == 0)
348         {
349             CmdArg = TxDataSourceCommandAccessor::NEWSOURCE;
350             Result = STRS_OK;
351         }

```

Figure 10.—Obtaining source container.

- (4) Build the application (Fig. 11).

```

/WDRK/IPAS/code/strs/STRS_Architecture_RI/STRS_ReferenceImplementation$ ls
gHtComputer FPGA_Linux GUI Makefile OE x86_64-pc-linux-gnu
/WDRK/IPAS/code/strs/STRS_Architecture_RI/STRS_ReferenceImplementation$ make

```

Figure 11.—Application build.

The example path in this document is not the one within the GPM of the SDR, but the path of the development system. Use the path specified in the iPAS STRS Radio User's Guide (Ref. 1) to develop the application directly on the SDR GPM. However, developers will likely build and modify the control code on a workstation separate from the SDR. In this case, the path is not required because the application is not developed on the SDR. As it was shown in this example, the application was built outside the SDR.

To run the modified code, follow the steps in the iPAS STRS Radio User's Guide (Ref. 1).

Field-Programmable Gate Array (FPGA) Waveform Development

A new waveform developer using the RIACS platform is encouraged to become familiar with the FPGA wrapper code by operating and testing the FPGA design with the test waveform included. Once the waveform developer is comfortable with the wrapper, the test waveform can be removed and replaced with the new custom waveform. Portions of the test waveform may be reused if desired.

The detailed design description of the wrapper and the test waveform can be found in the "Programmable Logic Device (PLD) Design Description for the Integrated Power, Avionics, and Software (iPAS) Space Telecommunications Radio System (STRS) Radio" document (Ref. 3). Use this document as a reference to understand the wrapper implementation.

Integrated Synthesis Environment (ISE®) Project

The FPGA design VHDL implementation is included in the ISE® project in the folder called IPAS_STRS_Radio.

The project folder contains the following key subfolders:

- ipcores—contains the generated Xilinx® Intellectual Property and the core generator project file (coregen.cgp).
- TestBenchWaveConf—contains the ISim wave configuration files for each module that has a test bench.
- vhdl—contains all the custom VHDL files, the User Constraints File (ucf) file, and a subfolder called TestBenches that contains all the test benches for the custom VHDL.
- ChipScopeProjects—contains the waveform setup for the Xilinx® ChipScope™ Pro tool for each VHDL module where the ChipScope™ Pro tool had been used previously.
- TestBenchTextFiles—contains the text files used for input data to various test benches.
- ISE_files—contains the VHDL files necessary for the interface to the RF front-end board (AD-FMCOMMS1-EBZ). These files include the modules that interfaced to the ADC and DAC, binary phase-shift keying (BPSK) modulator, and interpolation filter.
- SDK_files—contains the previously compiled MicroBlaze™ Executable and Linkable Format (ELF) file (CompiledDefaultProgram.elf) and all the SDK C/C++ code and header files.
- EDK_files—contains the MicroBlaze.xmp Platform Studio file and the corresponding user constraints file (MicroBlaze.ucf).

The top module is the FPGA wrapper, STRS_SDR_Wrapper, which implements all the FPGA interfaces.

Design Flow

To build the delivered design in a new ISE® Project Navigator project, perform the following steps:

- (1) Within ISE® 14.4 Design Suite, create a new project targeting the Xilinx® Virtex®-6 FPGA.
- (2) Add MicroBlaze.xmp from the EDK_files folder. The embedded processor can be modified by clicking the submodule icon, which opens the EDK.

(3) Add the required ELF file (executable file for the MicroBlaze™ processor) to the ISE® project and select both checkboxes (for implementation and simulation). This builds it into the bitstream output by ISE® Design Suite. The configuration in the SDK_files folder is named `CompiledDefaultProgram.elf`. It is configured as such:

- ADC sample rate: 196.608 MHz
- DAC sample rate: 196.608 MHz
- Rx gain: 10.605 dB

(4) Add the following files from the VHDL folder. When developing a custom waveform, do not add the modules marked “(WF),” unless these test waveform files will be reused in waveform.

- `ADC_FDI.vhd`
- `BertSyncher.vhd (WF)`
- `BPSKMod.vhd (WF)`
- `ClockDomainCrossing.vhd (WF)`
- `ClockEnables.vhd`
- `CommandDecoder.vhd (WF)`
- `CommandParse.vhd (WF)`
- `CreatePacketSM.vhd`
- `DAC_FDI.vhd`
- `DataMux.vhd (WF)`
- `Error_insert.vhd (WF)`
- `EthernetRx.vhd`
- `Eth_fifo_8.vhd`
- `Gmii_if.vhd`
- `KnightRider.vhd (WF)`
- `NrzL2M.vhd (WF)`
- `OutputDataMux.vhd`
- `Parallel2Serial.vhd (WF)`
- `PrbsRx23.vhd (WF)`
- `PrbsRx23Wrapper.vhd (WF)`
- `PrbsTx23.vhd (WF)`
- `PrbsTx23Wrapper.vhd (WF)`
- `PulseGenSM.vhd (WF)`
- `ReceiveSignal.vhd (WF)`
- `ResetGen.vhd`
- `RespFifoInputSM.vhd`
- `RxPackets.vhd`
- `RxStreamingData.vhd`
- `Rx_client_fifo_8.vhd`
- `SineWaveGen.vhd (WF)`
- `StatusResetGenSM.vhd (WF)`
- `StrDataFifoOutputSM.vhd`
- `StreamFifoInputSM.vhd`
- `StreamFifoOutputSM.vhd`
- `STRS_Radio.ucf`
- `STRS_Radio_Pkg.vhd`

- STRS_SDR_Wrapper.vhd
- STRS_Waveform.vhd (WF)
- TransmitSignal.vhd (WF)
- TxResponsePackets.vhd
- TxStreamData.vhd (WF)
- TxStreamFifoWriteSM (WF)
- Tx_client_fifo_8.vhd
- V6_emac_v1_5.vhd
- V6_emac_v1_5_block.vhd
- V6_emac_v1_5_example_design.vhd
- V6_emac_v1_5_locallink.vhd

(5) Add the following generated Intellectual Property from the ipcores folder (the Intellectual Property cores may need to be regenerated). When developing a custom waveform, do not add the Intellectual Property marked “(WF),” unless these test waveform files will be reused in the waveform.

- ClockWizard.xco
- FIFO262K.xco (WF)
- PSFx8a035.xco (WF)
- ResponseFifo.xco
- Rx_Streaming_Data_ROM.xco
- StreamingDataFifo.xco
- StreamingFifo.xco
- Tx_Response_Packet_ROM.xco

(6) Build the project. Select STRS_SDR_Wrapper as the top module and then double-click “Generate Programming File.” If there is a failed timing constraint when building the project, try running the Xilinx® ISE® SmartXplorer tool.

- (a) Go to Tools→SmartXplorer→Launch SmartXplorer.
- (b) Select the “User built-in SmartXplorer strategies for Timing Performance.”
- (c) Set the Maximum number of runs to 7.
- (d) Check the box next to “Automatically copy best result to working project directory when SmartXplorer is complete.”

Once the bitstream is built, Xilinx® iMPACT can be used to program the FPGA. The ChipScope™ Pro tool can be used to display direct DAC and ADC samples, if required.

Xilinx® ML605 Field-Programmable Gate Array (FPGA) Board Configuration

This section contains Xilinx® ML605 FPGA board configuration information.

Jumpers and Dip Switch Settings

To use the wrapper and test waveform as is, insert jumpers and set dip switches according to the listings in Table 3. Some of the jumpers are in their default locations and are not be needed for the iPAS STRS radio (i.e., Peripheral Component Interconnect Express (PCIe) lane size). If a jumper connector is not listed, no jumper is needed.

TABLE 3.—JUMPER AND SWITCH CONFIGURATION FOR DELIVERED PLATFORM

Jumper purpose or dip switch ID ^a	Location	Jumper connections or switch position	Description
Ethernet PHY ^b configuration	J66	Jumper pins 1 and 2	To use the GMII ^c (1 GB) interface to the Ethernet PHY.
	J67	Jumper pins 1 and 2	
	J68	No jumper	
For JTAG ^d access to the board	J17	Jumper pins 1 and 2	Enables JTAG access without FPGA ^e Mezzanine Card (FMC) modules installed.
	J18	Jumper pins 1 and 2	
System ACE ^f error LED ^g disable jumper	J69	Jumper pins 1 and 2	Enables the LED, which will flash if there is an ACE problem.
SFP ^h module control	J54	Jumper pins 1 and 2	SFP_RT_SEL (full bandwidth)
	J65	Jumper pins 1 and 2	SFP_TX_DISABLE (SFP enabled)
PCIe ⁱ lane size	J42	Jumper pins 5 and 6	X8 lane size
System monitor	J19	Jumper pins 1 and 2	Use the on-chip reference
		Jumper pins 1 and 3	Not connected
	Jumper pins 2 and 4	FPGA thermal diode access	
Dip switch S1	S1	CFGAddr 0 - Off	Selects which CF ^j images are downloaded to the FPGA (selects subfolder cfg 4)
	S2	CFGAddr 1 - Off	
	S3	CFGAddr 2 - On	Enable ACE boot
	S4	SysAce Mode - On	
Dip switch S2	S1	EXT_CCLK - On	Oscillator enable
	S2	CS_SEL - On	Boot EPROM ^k select
	S3	M0 - Off	FPGA mode (slave SelectMAP)
	S4	M1 - On	
	S5	M2 - On	
	S6	Flash_A23 - Off	

^aIdentification.

^bPhysical layer.

^cGigabit media-independent interface.

^dJoint Test Action Group.

^eField-programmable gate array.

^fArchiver compression file.

^gLight-emitting diode.

^hSmall form-factor pluggable.

ⁱPeripheral Component Interconnect Express.

^jCompactFlash.

^kErasable programmable read-only memory.

Joint Test Action Group (JTAG) Programming

During waveform development, the Xilinx[®] Virtex[®]-6 FPGA on the Xilinx[®] ML605 FPGA board can be programmed using the iMPACT program, the ISE[®] Design Suite generated .bit file, a USB to mini USB cable, and the USB–JTAG connector (J22) on the board.

Archiver Compression (ACE) File

Once the final .bit file is completed and fully tested, the CompactFlash card can be programmed with an ACE file and the FPGA will automatically load the FPGA at power-on.

To create the ACE file, follow these steps:

- (1) In ISE[®] Design Suite, right-click “Configure Target Device” and select “Process Properties.”
- (2) Browse to select the correct iMPACT project file.
- (3) Double-click “Generate Target PROM/ACE File.”
- (4) When iMPACT opens, double-click “System ACE,” select “Novice” and configure the parameters.
- (5) Double-click “Generate File.”
- (6) The ACE file will be in the directory with the name specified.

To program the CompactFlash card:

- (1) Remove the CompactFlash card from the FPGA board and insert in the computer or CF adapter.
- (2) Drag or copy the ACE file created above into the cfg4 directory (selected by S1, S2, and S3 on dip switch S1) of the CompactFlash card. A different CompactFlash directory can be used, but must be selected on dip switch S1 by the S1, S2, and S3 switches.
- (3) Delete or remove (copy and save elsewhere) any other ACE file in the cfg4 directory.

Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL) Signal Naming Conventions

The following signal and file naming conventions are used in the VHDL in this development. Signal names throughout this document are shown in italics.

- A signal with a *_n* at the end of the signal name like (*Reset_n*) is a low true (asserted low) signal.
- All test benches have the same name as the module they test with a *_tb* at the end (ex. *TxStreaming_tb.vhd*).
- Constants are always in all capital letters.
- Most signal names with an *O* at the end (like *DataValidO*) are module output signals that were renamed to be able to look at them using ChipScope™ Pro tool.
- Signals with an *R* at the end (like *StatusBitsR*) are registered versions of a signal.

Wrapper Description

STRS_SDR_Wrapper is the top-level module in the FPGA design. STRS requires that the FPGA wrapper for an STRS radio encompasses all the possible radio FPGA interfaces. The wrapper abstracts the interfaces to the FPGA from the waveform, so that the waveform developer does not need to implement these interfaces. This approach also allows the platform developer to protect proprietary information from a waveform developer. The wrapper can also include any other functionality that a radio would require, like power-on-resets and clock generation, which would be common to all radios on the platform.

The STRS_SDR_Wrapper module is the FPGA wrapper and contains the clock generation and reset creation. This wrapper also has interfaces to onboard resources like switches and LEDs, as well as the implementation of the interface to the Ethernet physical layer (PHY) for receiving commands and streaming data from the GPM processor and for transmitting command responses and streaming data to the processor. The wrapper also contains the interface to the RF front-end board.

The STRS_SDR_Wrapper module utilizes a Xilinx® LogiCORE™ Intellectual Property Clocking Wizard to generate clocks necessary for the Ethernet Media Access Controller (EMAC). The source input clock (200 MHz) for the LogiCORE™ Intellectual Property Clocking Wizard is single-ended and is generated within the MicroBlaze™ processor. The LogiCORE™ Intellectual Property Clocking Wizard creates the following clocks:

- *RefClk*—200 MHz clock used by the Ethernet Intellectual Property core.
- *GtxClk*—125 MHz single-ended clock which is used by the Ethernet interface.

The *Locked* output from the LogiCORE™ Intellectual Property Clocking Wizard is used as the power-on-reset for the reset generator.

The waveform clocks originate in the DAC device on the RFM for the Tx side and the ADC device on the RFM for the Rx side. These DAC and ADC clocks (both approximately 196 MHz) are used in the wrapper to generate clock enable signals to control clocking within the waveform.

The ClockEnables module is instantiated twice, once for the Tx-side clock enables and once for the Rx-side clock enables. The ClockEnables module uses generics to set the three output clock-enable frequencies. The waveform developer can customize the clock enables for their waveform design by changing the generic values in the ClockEnables module instantiation.

In addition to the generation of the clocks and resets, the wrapper instantiates the EMAC interface (`v6_emac_v1_5_example_design.vhd`) and the radio test waveform (`STRS_Waveform.vhd`). The wrapper also includes the modules that provide the basic functionality to communicate in both directions to the Ethernet interface. These modules include EthernetRx and RxPackets for receiving and parsing Tx-side streaming data packets and command packets, and OutputDataMux for transmitting Rx-side streaming and command response packets. Each of these modules are discussed in detail in the PLD Detailed Design section of the “PLD Design Description for the iPAS STRS Radio” document (Ref. 3).

The ErrorFlag process in the wrapper registers and combines the error flags generated from the wrapper submodules into a word called *StatusBits* that is used as an input to the waveform module. This allows the status bits from the wrapper to be sent in response to a status request command, if the waveform developer chooses to use them. See the Status Bits section for more information.

The wrapper input, output, and bidirectional signals are shown in Tables 4 to 6, with their signal names, descriptions, and FPGA pin numbers.

TABLE 4.—WRAPPER MODULE INPUTS

Signal name	Description	FPGA ^a pin number
<i>CLK_N</i>	Differential FPGA system clock (200 MHz)—negative	H9
<i>CLK_P</i>	Differential FPGA system clock (200 MHz)—positive	J9
<i>USER_CLOCK</i>	FPGA user clock (66 MHz)	U23
<i>GMII_RXD0</i>	Ethernet receive data bit 0 (from PHY ^b)	AN13
<i>GMII_RXD1</i>	Ethernet receive data bit 1 (from PHY)	AF14
<i>GMII_RXD2</i>	Ethernet receive data bit 2 (from PHY)	AE14
<i>GMII_RXD3</i>	Ethernet receive data bit 3 (from PHY)	AN12
<i>GMII_RXD4</i>	Ethernet receive data bit 4 (from PHY)	AM12
<i>GMII_RXD5</i>	Ethernet receive data bit 5 (from PHY)	AD11
<i>GMII_RXD6</i>	Ethernet receive data bit 6 (from PHY)	AC12
<i>GMII_RXD7</i>	Ethernet receive data bit 7 (from PHY)	AC13
<i>GMII_RX_DV</i>	Ethernet receive data valid (from PHY)	AM13
<i>GMII_RX_ER</i>	Ethernet receive error (from PHY)	AG12
<i>GMII_RX_CLK</i>	Ethernet receive clock (from PHY)	AP11
<i>RESET</i>	Reset signal from push button on FPGA board	G26
<i>GPIO_DIP_SW1</i>	Dip switch 1 on FPGA board	D22
<i>GPIO_DIP_SW2</i>	Dip switch 2 on FPGA board	C22
<i>GPIO_DIP_SW3</i>	Dip switch 3 on FPGA board	L21
<i>GPIO_DIP_SW4</i>	Dip switch 4 on FPGA board	L20
<i>GPIO_DIP_SW5</i>	Dip switch 5 on FPGA board	C18
<i>GPIO_DIP_SW6</i>	Dip switch 6 on FPGA board	B18
<i>GPIO_DIP_SW7</i>	Dip switch 7 on FPGA board	K22
<i>GPIO_DIP_SW8</i>	Dip switch 8 on FPGA board	K21
<i>DacClkInP</i>	196.6 MHz clock from DAC ^c (p)	A10
<i>DacClkInN</i>	196.6 MHz clock from DAC (n)	B10
<i>UartRx</i>	UART ^d receive data	J24
<i>AdcClkInP</i>	196.6 MHz clock from ADC ^e ; synchronous with ADC data (p)	F33
<i>AdcClkInN</i>	196.6 MHz clock from ADC; synchronous with ADC data (n)	G33
<i>AdcOrInP</i>	Differential overrange indicator, positive (not used)	K26
<i>AdcOrInN</i>	Differential overrange indicator, negative (not used)	K27
<i>AdcDataInP0</i>	ADC data in 0—positive	L29
<i>AdcDataInN0</i>	ADC data in 0—negative	J30
<i>AdcDataInP1</i>	ADC data in 1—positive	C33
<i>AdcDataInN1</i>	ADC data in 1—negative	B34

TABLE 4.—Concluded.

Signal name	Description	FPGA ^a pin number
<i>AdcDataInP2</i>	ADC data in 2—positive	D34
<i>AdcDataInN2</i>	ADC data in 2—negative	C34
<i>AdcDataInP3</i>	ADC data in 3—positive	J31
<i>AdcDataInN3</i>	ADC data in 3—negative	J32
<i>AdcDataInP4</i>	ADC data in 4—positive	H34
<i>AdcDataInN4</i>	ADC data in 4—negative	H33
<i>AdcDataInP5</i>	ADC data in 5—positive	F30
<i>AdcDataInN5</i>	ADC data in 5—negative	G30
<i>AdcDataInP6</i>	ADC data in 6—positive	E32
<i>AdcDataInN6</i>	ADC data in 6—negative	E33
<i>AdcDataInP7</i>	ADC data in 7—positive	G32
<i>AdcDataInN7</i>	ADC data in 7—negative	H32
<i>AdcDataInP8</i>	ADC data in 8—positive	G31
<i>AdcDataInN8</i>	ADC data in 8—negative	H30
<i>AdcDataInP9</i>	ADC data in 9—positive	K28
<i>AdcDataInN9</i>	ADC data in 9—negative	J29
<i>AdcDataInP10</i>	ADC data in 10—positive	L25
<i>AdcDataInN10</i>	ADC data in 10—negative	L26
<i>AdcDataInP11</i>	ADC data in 11—positive	J30
<i>AdcDataInN11</i>	ADC data in 11—negative	K29
<i>AdcDataInP12</i>	ADC data in 12—positive	K33
<i>AdcDataInN12</i>	ADC data in 12—negative	J34
<i>AdcDataInP13</i>	ADC data in 13—positive	F31
<i>AdcDataInN13</i>	ADC data in 13—negative	E31

^aField-programmable gate array.

^bPhysical layer.

^cDigital-to-analog converter.

^dUniversal asynchronous receiver/transmitter.

^eAnalog-to-digital converter.

TABLE 5.—WRAPPER MODULE OUTPUTS

Signal name	Description	FPGA ^a pin number
<i>GMII_TXD0</i>	Ethernet transmit data bit 0 (to PHY ^b)	AM11
<i>GMII_TXD1</i>	Ethernet transmit data bit 1 (to PHY)	AL11
<i>GMII_TXD2</i>	Ethernet transmit data bit 2 (to PHY)	AG10
<i>GMII_TXD3</i>	Ethernet transmit data bit 3 (to PHY)	AG11
<i>GMII_TXD4</i>	Ethernet transmit data bit 4 (to PHY)	AL10
<i>GMII_TXD5</i>	Ethernet transmit data bit 5 (to PHY)	AM10
<i>GMII_TXD6</i>	Ethernet transmit data bit 6 (to PHY)	AE11
<i>GMII_TXD7</i>	Ethernet transmit data bit 7 (to PHY)	AF11
<i>GMII_TX_EN</i>	Ethernet transmit enable (to PHY)	AJ10
<i>GMII_TX_ER</i>	Ethernet transmit error (to PHY)	AH10
<i>GMII_TX_CLK</i>	Ethernet transmit clock (to PHY)	AH12
<i>PHY_RESET</i>	Reset signal to the Ethernet PHY chip	AH13
<i>GPIO_LED_0</i>	LED 0 on FPGA board	AC22
<i>GPIO_LED_1</i>	LED 1 on FPGA board	AC24
<i>GPIO_LED_2</i>	LED 2 on FPGA board	AE22
<i>GPIO_LED_3</i>	LED 3 on FPGA board	AE23
<i>GPIO_LED_4</i>	LED 4 on FPGA board	AB23
<i>GPIO_LED_5</i>	LED 5 on FPGA board	AG23
<i>GPIO_LED_6</i>	LED 6 on FPGA board	AE24
<i>GPIO_LED_7</i>	LED 7 on FPGA board	AD24
<i>DacClkOutP</i>	196.6 MHz clock to DAC ^c ; synchronous with DAC data (p)	A10
<i>DacClkOutN</i>	196.6 MHz clock to DAC; synchronous with DAC data (n)	B10
<i>DacFrameOutP</i>	Differential frame output (p)	R26
<i>DacFrameOutN</i>	Differential frame output (n)	T26
<i>DacDataOutP0</i>	16-bit DAC output data 0—positive	N25

TABLE 5.—Concluded.

Signal name	Description	FPGA ^a pin number
<i>DacDataOutN0</i>	16-bit DAC output data 0—negative	M25
<i>DacDataOutP1</i>	16-bit DAC output data 1—positive	K32
<i>DacDataOutN1</i>	16-bit DAC output data 1—negative	K31
<i>DacDataOutP2</i>	16-bit DAC output data 2—positive	M26
<i>DacDataOutN2</i>	16-bit DAC output data 2—negative	M27
<i>DacDataOutP3</i>	16-bit DAC output data 3—positive	N33
<i>DacDataOutN3</i>	16-bit DAC output data 3—negative	M33
<i>DacDataOutP4</i>	16-bit DAC output data 4—positive	M31
<i>DacDataOutN4</i>	16-bit DAC output data 4—negative	L31
<i>DacDataOutP5</i>	16-bit DAC output data 5—positive	N34
<i>DacDataOutN5</i>	16-bit DAC output data 5—negative	P34
<i>DacDataOutP6</i>	16-bit DAC output data 6—positive	N32
<i>DacDataOutN6</i>	16-bit DAC output data 6—negative	P32
<i>DacDataOutP7</i>	16-bit DAC output data 7—positive	P31
<i>DacDataOutN7</i>	16-bit DAC output data 7—negative	P30
<i>DacDataOutP8</i>	16-bit DAC output data 8—positive	N27
<i>DacDataOutN8</i>	16-bit DAC output data 8—negative	P27
<i>DacDataOutP9</i>	16-bit DAC output data 9—positive	R31
<i>DacDataOutN9</i>	16-bit DAC output data 9—negative	R32
<i>DacDataOutP10</i>	16-bit DAC output data 10—positive	L33
<i>DacDataOutN10</i>	16-bit DAC output data 10—negative	M32
<i>DacDataOutP11</i>	16-bit DAC output data 11—positive	R28
<i>DacDataOutN11</i>	16-bit DAC output data 11—negative	R27
<i>DacDataOutP12</i>	16-bit DAC output data 12—positive	M30
<i>DacDataOutN12</i>	16-bit DAC output data 12—negative	N30
<i>DacDataOutP13</i>	16-bit DAC output data 13—positive	P29
<i>DacDataOutN13</i>	16-bit DAC output data 13—negative	R29
<i>DacDataOutP14</i>	16-bit DAC output data 14—positive	C32
<i>DacDataOutN14</i>	16-bit DAC output data 14—negative	B32
<i>DacDataOutP15</i>	16-bit DAC output data 15—positive	A33
<i>DacDataOutN15</i>	16-bit DAC output data 15—negative	B33
<i>RefClkP</i>	30 MHz reference clock for RF ^d board (positive)	N28
<i>RefClkN</i>	30 MHz reference clock for RF board (negative)	N29
<i>UartTx</i>	UART ^e transmit data	J25

^aField-programmable gate array.

^bPhysical layer.

^cDigital-to-analog converter.

^dRadiofrequency.

^eUniversal asynchronous receiver/transmitter.

TABLE 6.—WRAPPER MODULE IN/OUTS

Signal name	Description	FPGA ^a pin number
<i>IicSda</i>	IIC ^b bus serial data line	AG13
<i>IicScl</i>	IIC bus serial clock line	AF13

^aField-programmable gate array.

^bInter-Integrated Circuit.

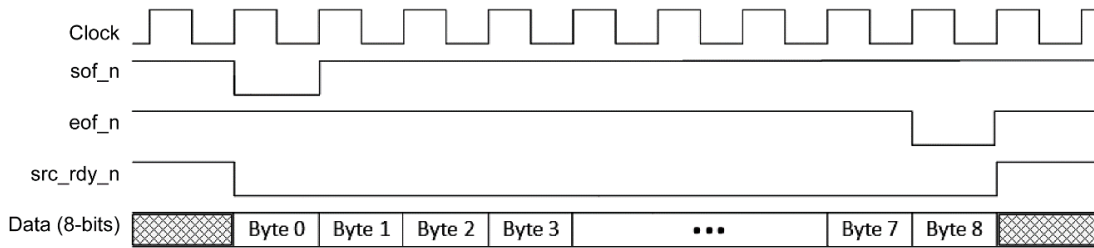


Figure 12.—Ethernet LocalLink interface timing.

Wrapper Interfaces

This section contains information about the wrapper interfaces.

Tri-Mode Ethernet Physical Layer (PHY)

The Xilinx® ML605 FPGA board utilizes a Marvell Alaska Gigabit Ethernet PHY transceiver (88E1111) for Ethernet communication to the eBOX620–110–FL embedded PC. The FPGA communicates with the PHY device through the `v6_emac_v1_5_example_design` generated by the Xilinx® CORE Generator™ System for the `v6_emac_v1_5` Intellectual Property. The Intellectual Property core is setup to use the gigabit media-independent interface (GMII), 1 Gbps interface to the PHY. This interface is used by the wrapper to receive command packets and Tx-side streaming data packets from the eBOX620–110–FL embedded PC, and to send command response packets and Rx-side streaming packets to the eBOX620–110–FL embedded PC.

The example design instantiates the EMAC wrapper and provides a Xilinx® LocalLink interface, which places Tx and Rx client first in first outs (FIFOs) between the EMAC and the example design wrapper interface to the user. A small address-swap module is used to loopback the LocalLink received data to the LocalLink transmit data. To use the LocalLink interface in the `STRS_SDR_Wrapper`, the address-swap module was removed from the example design and the LocalLink signals were passed up to the `v6_emac_v1_5_example_design` top-level module as inputs and outputs to the wrapper. The `v6_emac_v1_5_example_design` module is clocked by the 200 MHz *RefClk* and the 125 MHz *Gtx_Clk*.

The LocalLink interface timing is shown in Figure 12, and is essentially the same for both the Tx and Rx sides. The primary LocalLink interface signals include 8-bit data (*data*), start-of-frame (*sof_n*), end-of-frame (*eof_n*), and data source ready (*src_rdy_n*).

Radiofrequency (RF) Front-End Board

The RF front-end board (AD–FMCOMMS1–EBZ) must be configured using the MicroBlaze™ processor, but the FPGA wrapper contains two modules, `DAC_FDI` and `ADC_FDI`, that provide the interface to the DAC and ADC, respectively. The waveform developer can connect the I and Q DAC input signals to the appropriate inputs of the `DAC_FDI` module and the I and Q ADC output signals to the appropriate outputs of the `ADC_FDI` module using the VHDL code within the `STRS_SDR_Wrapper` module.

General Purpose Board Resources—Clocks, Light-Emitting Diodes (LEDs), and Switches

The Xilinx® ML605 FPGA board has a number of onboard resources that are available to the waveform developer. Some of these resources include LEDs, dip switches, push button switches, and clocks.

The wrapper utilizes the 200 MHz differential clock for the primary clock of the design and a push button switch for the user reset. The test waveform uses the dip switches and LEDs for a test command (read dip switches and set LEDs) and for displaying some waveform status. Therefore, the wrapper must pass the LED and dip switch values to and from the board resources for the test waveform. New waveform developers who do not use the dip switches or LEDs may remove them from the UCF file and from the inputs and outputs of the `STRS_SDR_Wrapper` module.

Please refer to the ML605 Hardware User Guide (Ref. 4) for more information about the Xilinx® ML605 FPGA onboard resources.

Test Waveform Description

The iPAS STRS radio implementation includes a test waveform (*STRS_Waveform.vhd* and its submodules) to test and demonstrate the STRS FPGA wrapper. The test waveform does not fully implement all the signal processing functionality for a radio, but it exercises and demonstrates each interface in the FPGA wrapper.

The waveform developer is expected to remove the test waveform from the ISE® project and replace it with a custom waveform. However, portions of the test waveform that are useful to the waveform developer may be reused. Details of the test waveform design are in the “Programmable Logic Device (PLD) Design Description for the Integrated Power, Avionics, and Software (iPAS) Space Telecommunications Radio System (STRS) Radio” document (Ref. 3), starting with the *STRS_Waveform.vhd* section in the Programmable Logic Device (PLD) Detailed Design section.

The inputs to the test waveform are listed in Table 7 and the outputs are listed in Table 8. The waveform developer may choose to eliminate some of these input and output signals that are unnecessary for the design.

There are some key signals in the test waveform that the developer will want to retain because they will be important for all waveforms on the RIACS platform. The clock input signals (*Clk125*, *TxWFClock*, and *RxWFClock*) are necessary for all waveforms, but the actual clock frequencies (and the signal names) can be changed by the developer. The *Reset* signal should be used by all waveforms. Other important inputs that all waveforms will need are *RxCmdDataIn*, *RxCmdDataSrcRdy*, *RespSending_n*, *StreamDataIn*, and *StreamDataValid*. *RxCmdDataIn* is the command packet payload data (no headers) in bytes and the *RxCmdDataSrcRdy* signal is high while the *RxCmdDataIn* data is valid. The *RespSending_n* signal is low while a command response is being sent to the Ethernet port. *StreamDataIn* is the Tx-side streaming data in bytes without any headers and *StreamDataValid* is high while *StreamDataIn* is valid.

TABLE 7.—TEST WAVEFORM MODULE INPUTS

Signal name	Description
<i>Clk125</i>	125 MHz clock
<i>TxWFClock</i>	Tx-side ^a waveform clock (approx. 196.6 MHz, DAC ^b clock)
<i>TxWFClockEn1</i>	Tx-side waveform clock enable (byte rate)
<i>TxWFClockEn2</i>	Tx-side waveform clock enable (word rate)
<i>RxWFClock</i>	Rx-side ^c waveform clock (approx. 196.6 MHz, DAC clock)
<i>RxWFClockEn1</i>	Rx-side waveform clock enable (byte rate)
<i>RxWFClockEn2</i>	Rx-side waveform clock enable (word rate)
<i>SymbClockEn</i>	Symbol rate clock enable
<i>Reset</i>	Reset signal from wrapper (power-on or reset and push-button)
<i>DIP_SW</i>	8 bits, onboard dip switches
<i>RxCmdDataIn</i>	8 bits, received command data
<i>RxCmdDataSrcRdy</i>	Received command data valid signal
<i>StreamDataIn</i>	8 bits, received streaming data
<i>StreamDataValid</i>	Received streaming data valid signal
<i>EmacRxError</i>	EMAC ^d PHY ^e receive error
<i>StatusBitsIn</i>	36 bits, status bits coming from the wrapper
<i>RespSending_n</i>	Low while a command response is being sent
<i>AdcDataInI</i>	ADC ^f I ^g channel data
<i>AdcDataInQ</i>	ADC Q ^h channel data

^aTransmit-side.

^bDigital-to-analog.

^cReceive-side.

^dEthernet Media Access Controller.

^ePhysical layer.

^fAnalog-to-digital converter.

^gIn-phase.

^hQuadrature.

TABLE 8.—TEST WAVEFORM MODULE OUTPUTS

Signal name	Description
<i>DacDataOutI</i>	DAC ^a I ^b channel data
<i>DacDataOutQ</i>	DAC Q ^c channel data
<i>WFRresetOut</i>	Waveform reset signal
<i>FlagResetOut</i>	Status bits reset signal
<i>SoftResetOut</i>	Commanded reset
<i>CmdResponse</i>	120-bit contents of the response to a command
<i>TxSendReady</i>	Indicates that a command response can be sent
<i>StreamEnRx</i>	Stream enable—stays high until streaming is stopped
<i>RxParallelData</i>	16 bits, data to be streamed
<i>LED</i>	8 bits, outputs to board LEDs ^d

^aDigital-to-analog converter.

^bIn-phase.

^cQuadrature.

^dLight-emitting diodes.

MAC header	MAC destination address (6 bytes)			
	MAC source address (6 bytes)			Ethernet type (2 bytes)
IP datagram	Version (4 bits)	IHL (4 bits)	Type of service (8 bits)	Length (IP + UDP + payload, 2 bytes)
	Identification (2 bytes)		Flags (3 bits)	Fragment offset (13 bits)
	Time to live (1 byte)	Protocol (1 byte)	IP header checksum (2 bytes)	
	Source IP address (4 bytes)		Destination IP address (4 bytes)	
UDP header	Source port (2 bytes)		Destination port (2 bytes)	
	Length (UDP + payload, 2 bytes)		UDP checksum (2 bytes)	
Payload	Various lengths			

Figure 13.—Ethernet packet definition. IHL, Internet header length; IP, Internet Protocol; MAC, media access control; UDP, User Datagram Protocol.

Key test waveform output signals that all developers will most likely need are *CmdResponse*, *TxSendReady*, *StreamEnRx*, and *RxParallelData*. The *CmdResponse* signal in the test waveform is a 120-bit word that contains the payload portion of a command response packet. All developers will need a command response packet, but the size may vary and is selectable by the developer using a constant in the *STRS_Radio_Pkg.vhd*. The *TxSendReady* signal is high when a command response is available to transmit. *RxParallelData* is 16-bit data sample from the ADC output that can be used for the Rx-side streaming packets. *StreamEnRx* is high while streaming is enabled.

Ethernet Packet Structure

Figure 13 shows the definition of the Ethernet header (media access control (MAC) header, Internet Protocol (IP) datagram header, and User Datagram Protocol (UDP) header).

Packet Headers From Field-Programmable Gate Array (FPGA) to Processor

When packets are sent from the FPGA to the GPM processor, the packet headers must be inserted when the packets are formed. In the test waveform, the headers are created at design time and stored in read-only memory (ROM) to be read out and inserted in the packets. The UDP checksum field is optional and set to zero for all packets created by the FPGA. The IP datagram header checksum must be calculated, however. This calculation includes only the bytes in the IP datagram (i.e., not the MAC header or the UDP datagram).

Here are the steps for calculating the IP datagram header checksum field:

- (1) Add up the values of the 16-bit words in the IP datagram, excluding the checksum field.
- (2) Any binary digits above bit 15 (the carry bits) should be added to bits 0 to 15 of the sum above.
- (3) Invert the result to get the checksum.

Example IP Header:

4500 002E 0000 4000 4011 XXXX C0A8 0002 COA8 0001 (where XXXX is the checksum)

The sum of each word is 0x24692

0x24692 →	10	0100 0110 1001 0010	
Add the carry bits	+	_____	10
Sum →		0100 0110 1001 0100	
Invert →		1011 1001 0110 1011	= 0xB96B = checksum

STRS_Radio_Pkg.vhd

STRS_Radio_Pkg is a VHDL package used to share objects among many of the design modules in this implementation. This package contains a number of reusable constants and defines the ChipScope™ Pro Integrated CONTroller (ICON) and Integrated Logic Analyzer (ILA) components so they can be used in multiple modules (Table 9). The package also contains the look-up table used in PrbsRx23.vhd to add-up the number of errors in a received byte.

Waveform developers can modify the constants in this package according to the needs of their unique waveform.

Status Bits

Status bits from numerous locations in the test waveform and wrapper are collected together into one signal called *StatusBits*. These status bits are used to communicate potential issues to the processor, so that the user can be aware of these issues while the radio is operating.

The status bits from the wrapper are transferred to the test waveform though a 36-bit *StatusBits* word. The definition of the bits in *StatusBits* is defined in Table 10. Use of these status bits is at the discretion of the waveform developer.

Generics

The modules within the STRS_SDR_Wrapper module utilize generics to make the wrapper more reusable for new waveform developers. They will allow a developer to customize a waveform with specific command packet sizes, Ethernet port numbers, command response packet sizes, etc. Change the generics values in the STRS_Radio_Pkg.vhd file by changing the value assigned to the appropriate constant. The generics are described in Table 11.

Clock Generation

The LogiCORE™ Intellectual Property Clocking Wizard component in the top-level VHDL module, STRS_SDR_Wrapper, is used to generate the clocks for the wrapper. The v6_emac_v1_5_example_design component in the wrapper requires a 200 MHz single-ended input clock (*RefClk*) and a 125 MHz single-ended input clock (*GtxClk*). The waveform developer should retain the LogiCORE™ Intellectual Property Clocking Wizard “as is” in the wrapper.

The waveform clocks are sourced at the RFM DAC for the Tx-side waveform and at the RFM ADC for the Rx-side waveform. The wrapper contains two clock enable generators, one for the Tx side and one for the Rx side, to generate the clock enables for the waveform. Each clock enable module creates up to three enables. Three generics are available to set the frequency of the clock enables.

TABLE 9.—DEFINITION OF CONSTANTS IN STRS_Radio_Pkg

Constant name	Value	Description	Modules
PACKET_SIZE	60	Length of a command response packet	EthernetRx, TxResponsePackets, RespFifoInputSM, and RespFifoOutputSM
DATA_PACKET_SIZE	557	Length of a streaming data packet	EthernetRx, CreatePacketSM, RxStreamingData, and as BYTECNT in StrDataFifoOutputSM
HEADER_SIZE	42	Length of a transmit header	RespFifoInputSM and CreatePacketSM
SOURCE_PORT_BYTE	33	Location of source port number in packet header	EthernetRx
REMAINING_HEADER_SIZE	26	Length of remainder of header in Tx-side ^a packets after enable goes high	RxPackets
COMMAND_RESPONSE_SIZE	120	Length of a command response packet payload	OutputDataMux and TxResponsePackets
STREAM_PKT_BYTE1	0x8C	Tx-side streaming UDP ^b packet source port address—MSB ^c	EthernetRx
STREAM_PKT_BYTE2	0xA0	Tx-side streaming UDP packet source port address—LSB ^d	EthernetRx
CMD_PKT_BYTE1	0x8C	Command UDP packet source port address—MSB	EthernetRx
CMD_PKT_BYTE2	0x35	Command UDP packet source port address—LSB	EthernetRx
PACKET_BYTE_CNT	570	Number of bytes in an Rx-side ^e streaming data packet	StrDataFifoOutputSM
WAIT_CNT	500	Number of clock cycles between Rx-side streaming data packets	StrDataFifoOutputSM
PACKET_NUM_CNT	4	Number of data packets sent in a group of packets	StrDataFifoOutputSM

^aTransmit-side.

^bUser Datagram Protocol.

^cMost significant byte.

^dLeast significant byte.

^eReceive-side.

TABLE 10.—DEFINITION OF THE StatusBits SIGNAL IN STRS_SDR_WRAPPER

Bit	Definition	Description
0	'0'	Reserved for waveform status bits
1	'0'	
2	'0'	
3	'0'	
4	'0'	
5	'0'	
6	'0'	
7	'0'	
8	'0'	
9	'0'	
10	'0'	
11	EthernetRx SM StuckFlag	Indicates EthernetRx SM ^a is stuck
12	ResponsePktFifo_Full	ResponsePktFifo overflow and underflow indicators
13	StreamingFifo_Full	StreamingFifo overflow and underflow indicators
14	StreamingFifo_Empty	
15	Streaming_Data_Fifo_Full	Streaming_Data_Fifo overflow and underflow indicators
16	Streaming_Data_Fifo_Empty	
17	'0'	StreamingDataFifo overflow and underflow indicators
18	'0'	
19	SMFailure_ResetGen	Flags indicating particular SM (indicated in the name of flag) erroneously entered “others” state
20	SMFailure_EthernetRx	
21	SMFailure_RxCommandPackets	
22	SMFailure_RxStreamingPackets	
23	SMFailure_RespFifoOutputSM	
24	SMFailure_StreamFifoInputSM	
25	SMFailure_CreatePacketSM	
26	SMFailure_StreamFifoOutputSM	
27	SMFailure_StrDataFifoOutputSM	
28	SMFailure_OutputDataMux	
29	'0'	Reserved for waveform status bits
30	'0'	
31	'0'	
32	'0'	
33	'0'	
34	'0'	
35	SMFailure_RespFifoInputSM	Flag indicating RespFifoInputSM erroneously entered “others” state

^aState machine.

TABLE 11.—WRAPPER GENERICS

Module name	Generic	Description	Value (in STRS_Radio_Pkg.vhd)
EthernetRx	StreamPktByte1	Linux PC ^a port address for streaming data packets—MSB ^b	STREAM_PKT_BYTE1 (0x8c)
	StreamPktByte2	Linux PC port address for streaming data packets—LSB ^c	STREAM_PKT_BYTE2 (0xA0)
	CmdPktByte1	Linux PC port address for commands and response packets—MSB	CMD_PKT_BYTE1 (0x8C)
	CmdPktByte2	Linux PC port address for commands and response packets—LSB	CMD_PKT_BYTE2 (0x35)
RxPackets (Inst_RxCommandPackets)	HeaderLen	Length of the header in packets received from GPP ^d	RX_REMAINING_HEADER_SIZE (26)
RxPackets1 (Inst_RxStreamPackets)	HeaderLen	Length of header in packets received from GPP	RX_REMAINING_HEADER_SIZE + 3 (26 + 3 = 29)
OutputDataMux	CmdResponseSize	Size of payload portion of a command response packet	COMMAND_RESPONSE_SIZE (120)
TxResponsePackets	CmdResponseSize	Size of payload portion of a command response packet	COMMAND_RESPONSE_SIZE (120)
RespFifoOutputSM	ByteCnt	Total number of bytes in response Ethernet packet	PACKET_SIZE (60)
StrDataFifoOutputSM	ByteCnt	Total length of a streaming data packet	DATA_PACKET_SIZE (557)

^aPersonal computer.

^bMost significant byte.

^cLeast significant byte.

^dGeneral purpose processor.

Commanding

All STRS radios need to use some sort of commanding approach to control the waveform and to query status. The wrapper for the RIACS platform was designed to use UDP packets through the Ethernet interface for commanding purposes, and therefore waveform developers will need to use UDP packets for transmitting commands to the FPGA. See Figure 13 for the definition of the Ethernet headers. The payload portion of the packet can be defined by the waveform developer, including length and content. Figure 14 shows the command packet structure used for the wrapper test waveform.

Waveform developers may also want to have command response packets, similar to those created in the test waveform. Command response packets inform the processor if a command was rejected or may return data in response to a query command. Figure 15 shows the command response packet structure used for the wrapper test waveform.

The packet structures in Figures 14 and 15 represent the test waveform implementation and are included as an example only. Waveform developers can define their own packet structure.

Streaming Data

Like commanding, the wrapper for the RIACS platform was designed to use UDP packets to stream communication data to and from the FPGA. The waveform developer can customize the size of the payload portion of streaming data packets and the length and content of the payload header. See Figure 13 for the definition of the Ethernet headers.

The streaming data packet structures in Figures 16 and 17 represent the test waveform implementation and are included as an example only. Waveform developers can define their own streaming data packet structure, if desired.

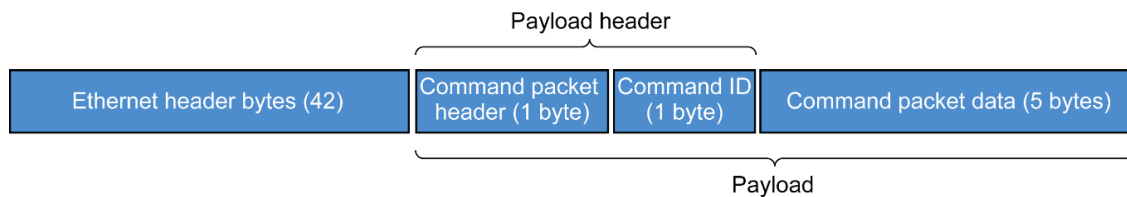


Figure 14.—Test waveform command packet structure. ID, identification.

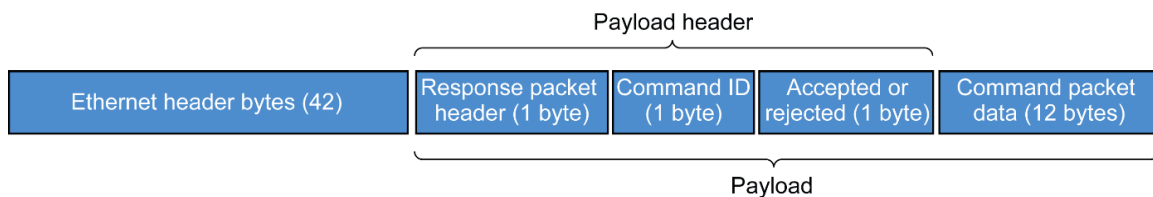


Figure 15.—Test waveform command response packet structure. ID, identification.

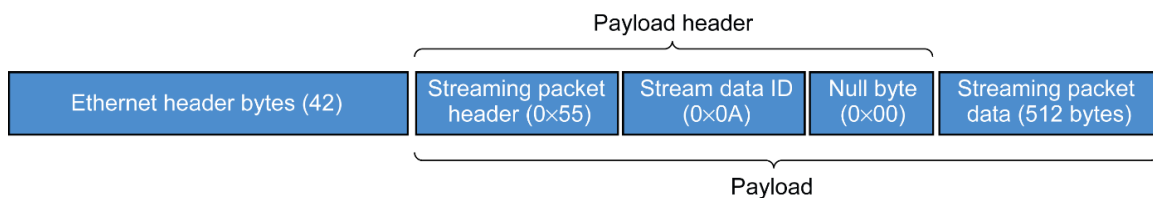


Figure 16.—Test waveform receive-side streaming data packet structure. ID, identification.

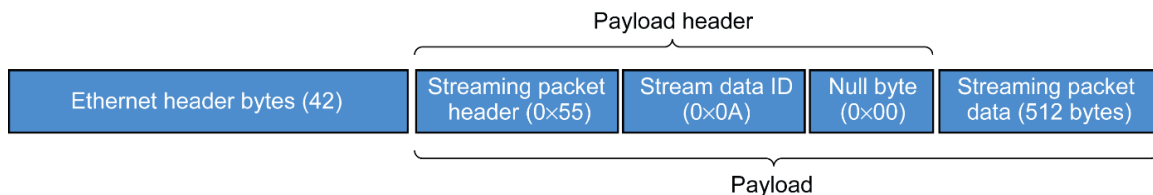


Figure 17.—Test waveform transmit-side streaming data packet structure. ID, identification.

TABLE 12.—WRAPPER TEST BENCHES

Module (.vhd)	Test bench name (.vhd)	Test bench description	Wave configuration file (.wcfg)
ResetGen	ResetGen_tb	Tests module by starting with Enable signal low and then going high; after a delay, two time-separated SwitchReset signals are issued, followed by another delay and a SoftReset signal	ResetGen
EthernetRx	EthernetRx_tb	Simulates incoming packets by reading data from a text file called <code>RxSourceData.txt</code>	EthernetRx
RxPackets	RxPackets_tb	To test for command packets (<code>Inst_RxCommandPackets</code>), use input file <code>RxSourceData.txt</code> ; to test for command packets (<code>Inst_RxStreamPackets</code>), use input file <code>StreamingData.txt</code> ; payload portion of each packet in text files always starts with header byte 0xAA	RxPackets
OutputDataMux	OutputDataMux_tb	Instantiates a <code>SineWaveGen</code> component to simulate streaming data and creates multiple command responses to test how streaming data packets and command responses can be interleaved; <code>OutputDataMux</code> sub-modules can also be tested with this test bench	OutputDataMux
ClockEnables	ClockEnables_tb	Sets <code>EndCount</code> values for three clock enable generators in the <code>ClockEnable</code> module and allows user to see generated <code>ClockEn</code> signals	ClockEnables

^aPhase-locked loop.

User Constraints File (UCF)

The UCF file, `STRS_Radio.ucf`, contains the timing constraints for the `v6_emac_v1_5_example_design`, the wrapper, test waveform, and the signal connections to the FPGA pins. Constraints are separated into those required by the wrapper and those required by the test waveform. The developer can remove and modify the test waveform constraints, as necessary. See the comments in the `.ucf` file.

Design Simulation

Test benches are included with the wrapper for simulation of most of the modules. The test benches and their descriptions are shown in Table 12.

Conclusions

The Integrated Power, Avionics, and Software (iPAS) Space Telecommunications Radio System (STRS) radio was implemented on the Reconfigurable, Intelligently-Adaptive Communication System (RIACS) platform, currently being used for radio development at NASA Johnson Space Center. The platform consists of a Xilinx® Virtex®-6 ML605 Evaluation Kit, an Analog Devices AD-FMCOMMS1-EBZ radiofrequency (RF) front-end board, and an Axiomtek™ eBOX620-110-FL embedded personal computer (PC) running the Ubuntu® 12.04 LTS operating system. The result of this development is a low-cost-STRS-compliant platform that can be used for waveform development for multiple radio applications. This document provides developers with information on how to develop a new waveform using the RIACS platform. Details about the STRS reference implementation and how it interacts with other software components is presented, along with information on how to use the software waveform templates to develop new radio waveforms. The document guides a field-programmable gate array (FPGA) waveform developer through the process to start a new Very High Speed Integrated Circuits (VHSICs) Hardware Description Language (VHDL) project and create a custom waveform in place of the provided test waveform. Details about the FPGA platform wrapper are provided to help the developer understand how to interface to the platform and the FPGA wrapper.

Acronyms and Abbreviations

The following abbreviations and acronyms are used within this document.

ACE	archiver compression file
ADC	analog-to-digital converter
API	application programming interface
BER	bit error rate
BERT	bit error rate tester
BPSK	binary phase-shift keying
CF	CompactFlash
DAC	digital-to-analog converter
EDK	Embedded Development Kit
ELF	Executable and Linkable Format
EMAC	Ethernet Media Access Controller
EPROM	erasable programmable read-only memory
FIFO	first in first out
FMC	FPGA Mezzanine Card
FPGA	field-programmable gate array
GMII	gigabit media-independent interface
GPM	general purpose module
GPP	general purpose processor
GUI	graphical user interface
HAL	hardware abstraction layer
HDL	hardware description language
HID	Hardware Interface Description
HW	hardware
I	in-phase
ICON	Integrated Controller
ID	identification
IDE	integrated development environment
IHL	Internet header length
IIC	Inter-Integrated Circuit
ILA	Integrated Logic Analyzer
IP	Internet Protocol
iPAS	Integrated Power, Avionics, and Software
ISE	Integrated Synthesis Environment
ISim	ISE Simulator
JTAG	Joint Test Action Group
LPC	Low Pin Count
LED	light-emitting diode
LSB	least significant bit/byte
MAC	media access control
MSB	most significant bit/byte
MUX	multiplexer
OE	operating environment
PC	personal computer
PCIe	Peripheral Component Interconnect Express
PHY	physical layer
PLD	programmable logic device
POSIX	Portable Operating System Interface

PRBS	pseudorandom bit sequence
Q	quadrature
RIACS	Reconfigurable, Intelligently-Adaptive Communication System
RF	radiofrequency
RFM	radiofrequency module
ROM	read-only memory
RTOS	real-time operating system
Rx	receive
SDK	Software Development Kit
SDR	software defined radio
SFP	small form-factor pluggable
SM	state machine
SPI	Serial Peripheral Interface
SPM	signal processing module
STRS	Space Telecommunications Radio System
TM	Technical Memorandum
Tx	transmit
UART	universal asynchronous receiver/transmitter
UCF	User Constraints File
UDP	User Datagram Protocol
UML	Unified Modeling Language
USB	Universal Serial Bus
VGA	Video Graphics Array
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
WF	waveform
WFiPAS	iPAS waveform

References

1. Roche, Rigoberto: iPAS STRS Radio User's Guide. NASA/TM—2017-219496, to be published, 2017.
2. Roche, Rigoberto; and Shalkhauser, Mary Jo W.: Integrated Power, Avionics, and Software (iPAS) Space Telecommunications Radio Standard (STRS) Radio Hardware Interface Description (HID). NASA/TM—2017-219432, to be published, 2017.
3. Shalkhauser, Mary Jo W.: Programmable Logic Device (PLD) Design Description for the Integrated Power, Avionics, and Software (iPAS) Space Telecommunications Radio Standard (STRS) Radio. NASA/TM—2017-219429, to be published, 2017.
4. ML605 Hardware User Guide. Xilinx UG534, vol. 1.8, 2012.
https://www.xilinx.com/support/documentation/boards_and_kits/ug534.pdf Accessed March 31, 2017.

