STRS COMPLIANT FPGA WAVEFORM DEVELOPMENT

Jennifer Nappier (Jennifer.M.Nappier@nasa.gov);
Joseph Downey (Joseph.A.Downey@nasa.gov);
NASA Glenn Research Center, Cleveland, Ohio, United States
Dale Mortensen (Dale.J.Mortensen@nasa.gov); ASRC, Cleveland, Ohio, United States

ABSTRACT

The Space Telecommunications Radio System (STRS) Architecture Standard describes a standard for NASA space software defined radios (SDRs). It provides a common framework that can be used to develop and operate a space SDR in a reconfigurable and reprogrammable manner. One goal of the STRS Architecture is to promote waveform reuse among multiple software defined radios. Many space domain waveforms are designed to run in the special signal processing (SSP) hardware. However, the STRS Architecture is currently incomplete in defining a standard for designing waveforms in the SSP hardware. Therefore, the STRS Architecture needs to be extended to encompass waveform development in the SSP hardware. The extension of STRS to the SSP hardware will promote easier waveform reconfiguration and reuse. A transmit waveform for space applications was developed to determine ways to extend the STRS Architecture to a field programmable gate array (FPGA). These extensions include a standard hardware abstraction layer for FPGAs and a standard interface between waveform functions running inside a FPGA. A FPGA-based transmit waveform implementation of the proposed standard interfaces on a laboratory breadboard SDR will be discussed.

1. INTRODUCTION

The Space Telecommunications Radio System (STRS) Architecture Standard [1] specifies an open architecture for NASA space software defined radios (SDRs). The STRS Architecture Standard divides the SDR into three functional hardware modules – the general processing module (GPM), the radio frequency module (RFM), and the signal processing module (SPM). There is a hardware interface description (HID) between each module that describes all of the physical hardware interfaces. The GPM contains a general purpose processor (GPP), memory, and the Spacecraft Telemetry Interface. The GPM runs the STRS Infrastructure and configures and controls the entire radio. The RFM contains filters, up/down converters, power amplifiers, digital to analog converters (DACs), and analog to digital converters (ADCs). It handles the conversion between the radio frequency (RF) and the intermediate frequency (IF) signals. The SPM contains a field programmable gate array (FPGA), digital signal processor (DSP), application specific integrated circuit (ASIC), or other specialized signal processing (SSP) device. The SPM performs the transformations between the IF digitally sampled signals and the data packets. These transformations are currently the reconfigurable part of the SDR waveforms. A drawing of the functional hardware modules of the STRS Architecture is shown in Figure 1.

It is desirable to place a space waveform in the FPGA due to the reconfigurable nature of the device and the ability to support high digital data rates. However, the STRS Architecture only specifies high-level standards for waveforms developed in the FPGA. In order to explore extending the STRS Architecture to lower level standards for the FPGA, a transmit waveform for space applications was developed. This development has lead to initial concepts for developing a firmware-based STRS compliant waveform that is reconfigurable and reusable. These concepts are the first steps towards extending the STRS Architecture standard to the firmware inside the FPGA.

A brief outline of the rest of the paper follows. The development goals for the waveform will be discussed in Section 2. The current STRS Architecture's support of these goals is discussed in Section 3. Section 4 describes proposed extensions to the STRS Architecture that more fully support STRS goals, and Section 5 discusses lab-based implementations of the extensions. Finally, Section 6 describes future work.

2. WAVEFORM DEVELOPMENT GOALS

2.1. Design a waveform that is reconfigurable

Reconfigurability is the ability to modify functionality of a radio by changing the operational parameters without requiring a software update. One reason to reconfigure a space SDR may be in response to changes in environmental

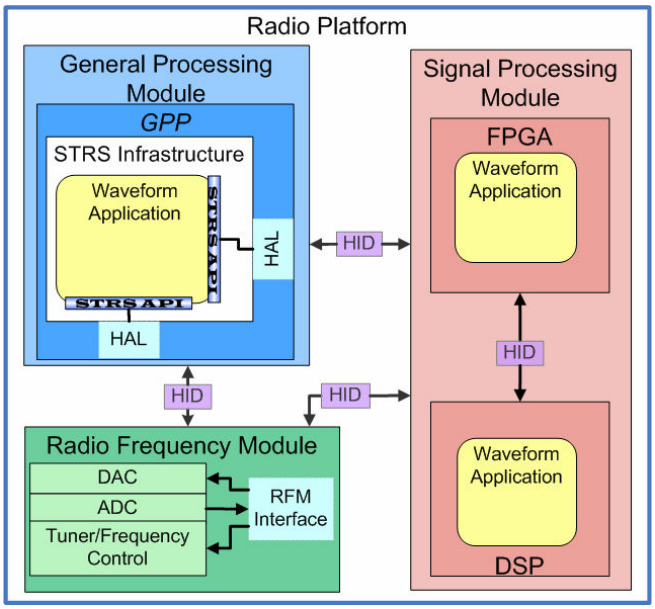

Figure 1: STRS Hardware Architecture Diagram.

or physical conditions experienced by the spacecraft. Another need for SDR reconfiguration is that the same communication network might use several variations of one waveform to perform different operations (e.g. different phases of a mission or modes of spacecraft operations). There could be little or no operational down time during reconfiguration.

2.2. Design a waveform that is reusable and portable across different SDRs

Reusability is the degree to which a software module or other work product can be used in more than one computing program or software system. The number of waveforms being used by NASA is limited and unlikely to increase or dramatically change. Reusing all or part of the code for this limited number of waveforms will decrease development time and cost and increase reliability. It is desired that both the individual waveform functions and the entire waveform as a whole be reusable. A waveform developer should be able to port individual waveform functions between SDR platforms. This might include replacing individual functions in a waveform, or reusing individual functions across several waveforms. A waveform developer should also be able to easily port entire waveforms to many different SDR platforms.

2.3. Design a waveform that is platform independent

One way to promote waveform reusability is to start with a platform independent design methodology. The highest level of abstraction of the waveform design should be platform independent. This might be a platform independent simulation tool model, or HDL code that is written in a platform independent manner.

3. CURRENT STRS ARCHITECTURE STANDARD

The current STRS Architecture Standard enables many of the waveform design goals listed in Section 2 to be achieved. The STRS Architecture is more defined for the GPM than the SPM. This section describes how the current software and firmware sections of the architecture enable the waveform design goals to be met, but also points out some shortcomings.

The software section of the STRS Architecture Standard specifies a common way to reconfigure waveform parameters on a space SDR without waveform reloading. The STRS Infrastructure running on the GPP specifies the use of STRS Application Program Interface (API) calls. These STRS API calls utilize device drivers on the GPP which interface to corresponding devices on the SPM or RFM. These APIs include the STRS_DeviceRead, STRS_DeviceWrite, STRS_DeviceGetAttribute, and STRS_DeviceSetAttribute APIs. They can be used to control and reconfigure the waveform components resident in the FPGA.

The software section of the STRS Architecture Standard supports the ability to remotely reprogram a compliant SDR with a different compliant waveform. The STRS_LoadDevice API can load a bit file that has been sent to a radio onto a FPGA or other device. In this way, a new waveform can be remotely loaded onto a FPGA.

The software section of the STRS Architecture Standard specifies standard APIs that interface to waveform functions running on the GPP. Therefore, code that is written on the GPP is portable and reusable across different SDRs. Software is platform independent because it is written in a high-level language like C.

The firmware section of the STRS Architecture Standard supports the use of modeling based firmware design techniques. Models can be developed using platform independent design techniques, but they could also be platform specific. The Firmware Architecture also supports the use of modularity and clear interfaces in waveform design and modeling. However, the Firmware Architecture does not define these interfaces in detail.

The firmware section of the STRS Architecture Standard supports reusable firmware-based waveforms through the use of a common waveform library. It also specifies an internal HID and an external HID. Device drivers are used to interface to the internal and external HID. The internal HID is an interface between devices on the SPM. The external HID is a set of interfaces from each device on the SPM to the GPM and RFM. However, these

interfaces are not specifically defined. Waveforms could be more reusable and portable if a common interface would be defined.

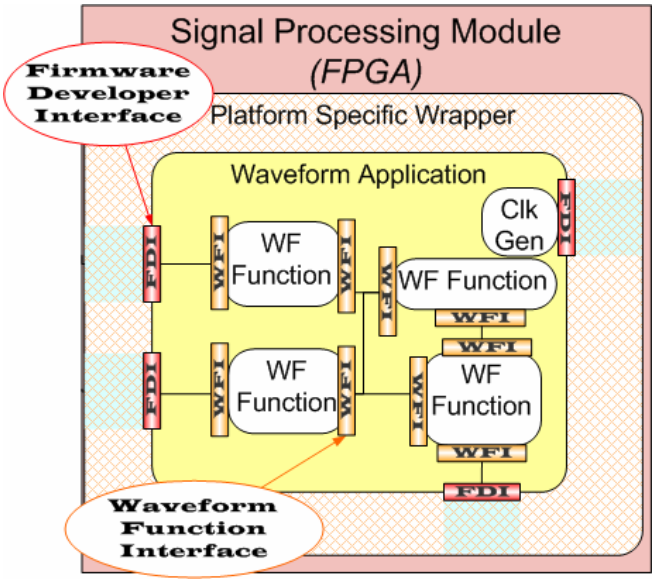

Figure 2: Conceptual drawing of the Firmware Developer Interface and the Waveform Function Interface in the FPGA.

4. PROPOSED EXTENSIONS TO STRS

There are two proposed extensions to the STRS Architecture Standard to further promote waveform portability and reuse. The proposed extensions were developed from the experience of designing a transmit waveform for space applications. The extensions are a Waveform Function Interface (WFI) and a Firmware Developer Interface (FDI). The WFI defines interfaces between waveform functions. The FDI abstracts the device drivers between the waveform application and devices external to FPGA, presenting a standard interface to the firmware-based waveform functions. A conceptual drawing of the FDI and WFI on the FPGA is shown in Figure 2. The FDI and WFI could be extended to other non-FPGA SPM devices, but the focus of this paper is on standardization in FPGAs.

4.1. Waveform Function Interface

The reuse of individual waveform functions was accomplished in this waveform development by two design practices. The waveform was divided into functions that were visible at the top level of the design. Next, these functions were separated from each other by common interfaces. The example of a modulator function, shown in Figure 3, demonstrates these interfaces. These interface definitions will allow a future user to easily reuse the function.

To aide in waveform reconfiguration, each individual waveform function was designed to accommodate all desired permutations of operation. Control over these permutations is achieved through a control signal, as shown in Figure 3.

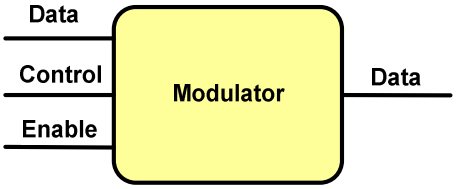

Figure 3: Example waveform function.

The waveform controller, which manages the state and behavior of the waveform, has access to all such control signals and can change them in real time. The enable signal shown in Figure 3 is a specific instance of a control signal. Thus the functionality of the SDR can be quickly reconfigured by properly controlling these signals without reprogramming the SDR with a new waveform.

These common interface definitions are proposed to be called the Waveform Function Interface (WFI). A standard WFI will promote modularity and enable waveform function reuse. It will allow individual waveform functions to be both ported to different platforms and reused among different waveforms. A critical component of the WFI is extensive documentation of all signals. Expected information on each signal includes data types, bit widths, and active low or active high for a control signal. There are other proposed signals in addition to the signals shown in Figure 3. The proposed WFI is described in Table 1.

Table 1: Waveform Function Interfaces.

Waveform Function Interfaces		
Direction	**Signal**	**Description**
IN	Data	Input data
IN	Clock	Directed to the function from the clock manager
IN	Enable	Specific type of control signal
IN	Reset	Specific type of control signal
IN	Control(s)	Signal from controller or another function
OUT	Data	Output data
OUT	Control(s)	Status signals about function state or control signals going to another function

4.2. Firmware Developer Interface

There are several design practices that were used to promote reuse of the entire waveform. First, the waveform was designed in a development environment that is FPGA platform independent. Platform specific VHDL that could target any FPGA was auto-generated from this platform-independent design using commercial software design tools. Therefore, this waveform development focused on algorithm development and the design tool automatically optimized area and speed constraints for the target FPGA. Next, the waveform was developed to contain minimal external interfaces to resources outside the FPGA. Limiting the waveform dependency on SDR platform hardware devices external to the FPGA enables reuse across many different STRS compliant SDR platforms.

Another way to minimize waveform dependency on devices external to the FPGA is to define a set of common external interfaces. The proposed set of common external interfaces is called the Firmware Developer Interface (FDI). The FDI is a common set of interfaces, but the implementation of those interfaces is not specified. The implementation and documentation of the FDI is the platform designer's responsibility. The waveform designer will use the FDI to access radio platform devices outside the FPGA.

There are two types of FDIs proposed: the control FDI and the data FDI. The control and data FDIs have the capability to both read from and write to devices external to the FPGA.

The control FDI provides a control interface into the FPGA. A waveform controller running on another device, such as the GPP, would use this interface to control the waveform functions running in the FPGA. This control interface is currently defined for the situation in which the GPP is the master and the FPGA is the slave. The control interface consists of data, address, clock, and enable signals. The data address (DATA, ADDRESS) pairs in the FDI should correspond to the name value (NAME, VALUE) pairs in the STRS_DeviceRead, STRS_DeviceWrite, STRS_DeviceGetAttribute, and STRS_DeviceSetAttribute APIs on the GPP. The proposed control FDI signals are shown in Table 2. The proposed common FDI DATA and corresponding API NAME pairs are shown in Table 3.

The data FDI is an interface to a hardware device where a continuous data stream is needed. The data FDI signals can be used with or without handshaking signals. The minimum data FDI signals without handshaking are proposed to be data, clock, and enable, as shown in Table 4. Handshaking signals for the data FDI will be defined in the future. It is up to the waveform developer to choose the type of signals to use, but both the minimum signals and handshaking signals must be implemented by the platform provider.

There should also be a standard set of interfaces to the clock manager on the FPGA. The clock manager generates the clocks that are used in the FPGA. These interfaces are not defined in this paper, but their documentation by the platform provider is part of the proposed extension.

Table 2: Control FDI signals.

Control FDI	
Read	
Direction	**Name**
IN	ControlRDY
IN	Address
IN	Data
OUT	Clock
Write	
Direction	**Name**
IN	ControlRDY
IN	Address
OUT	Data
OUT	Clock

Table 3: Corresponding FDI DATA and API NAME pairs.

FDI DATA	API NAME
FDI_START/STOP	API_ START/STOP
FDI_DATA_RATE	API_ DATA_RATE
FDI_DATA_FORMAT	API_ DATA_FORMAT
FDI_MODULATION	API_ MODULATION
FDI_IF	API_ IF
FDI_CODING	API_ CODING
FDI_AGC_GAIN	API_ AGC_GAIN
FDI_FUNCTION_ENABLE	API_ FUNCTION_ENABLE
FDI_FUNCTION_RESET	API_ FUNCTION_RESET

Table 4: Data FDI signals.

Data FDI	
Read	
Direction	**Name**
IN	Data
OUT	Enable
OUT	Clock
Write	
Direction	**Name**
OUT	Data
OUT	Enable
OUT	Clock

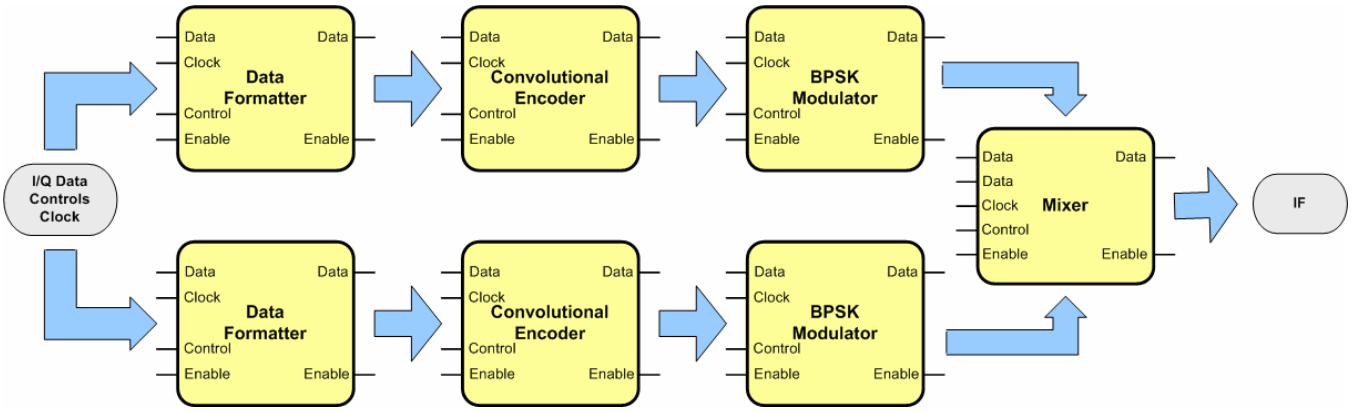

Figure 4: Waveform Implementation with the Waveform Function Interface.

5. IMPLEMENTATION EXAMPLES

The WFI and the FDI have both been implemented in the lab. The WFI has been implemented on a transmit waveform and the FDI has been implemented on a FPGA in a space software defined radio. However, they have not yet been integrated together on one radio.

5.1. Waveform Function Interface

Figure 4 shows a top level block diagram of the transmit waveform that was implemented. The waveform is separated into functions at the highest level of abstraction. The WFI is defined in between these functions. The WFI signals that were implemented include the data, clock, control, and enable signals.

Table 5 lists example documentation for the WFI signals that were implemented in the convolutional encoder function of the transmit waveform shown in Figure 4. The input clock rate is 2 MHz. The input data signal is 1 bit in width, has a rate of 1 Mbps, and has symbols formatted as NRZ-L. The control signal is 1 bit in width. It allows for ½ rate convolutional encoding or no encoding of the input signal. The input/output enable signal is active high. The output data signal is 1 bit in width, has a rate of 2 Mbps, and has symbols formatted as NRZ-L.

Table 5: Waveform Function Interfaces for the Convolutional Encoder.

Convolutional Encoder Waveform Function Interfaces		
Direction	**Signal**	**Description**
IN	Data	1 bit, 1 Mbps, NRZ-L symbols
IN	Clock	2 MHz
IN	Enable	Active High
IN	Controls	0 – No encoding 1 – ½ Rate encoding
OUT	Data	1 bit, 2 Mbps, NRZ-L symbols
OUT	Enable	Active High

5.2. Firmware Developer Interface

The control and data FDIs have been implemented on a space SDR breadboard. The FDI has been implemented to replace the platform specific wrapper supplied by the platform vendor, so that there is not an impact on performance. Test functions have been created to use the FDI to interface with components on both the GPM and the RFM. A diagram of the interfaces that were implemented is shown in Figure 5.

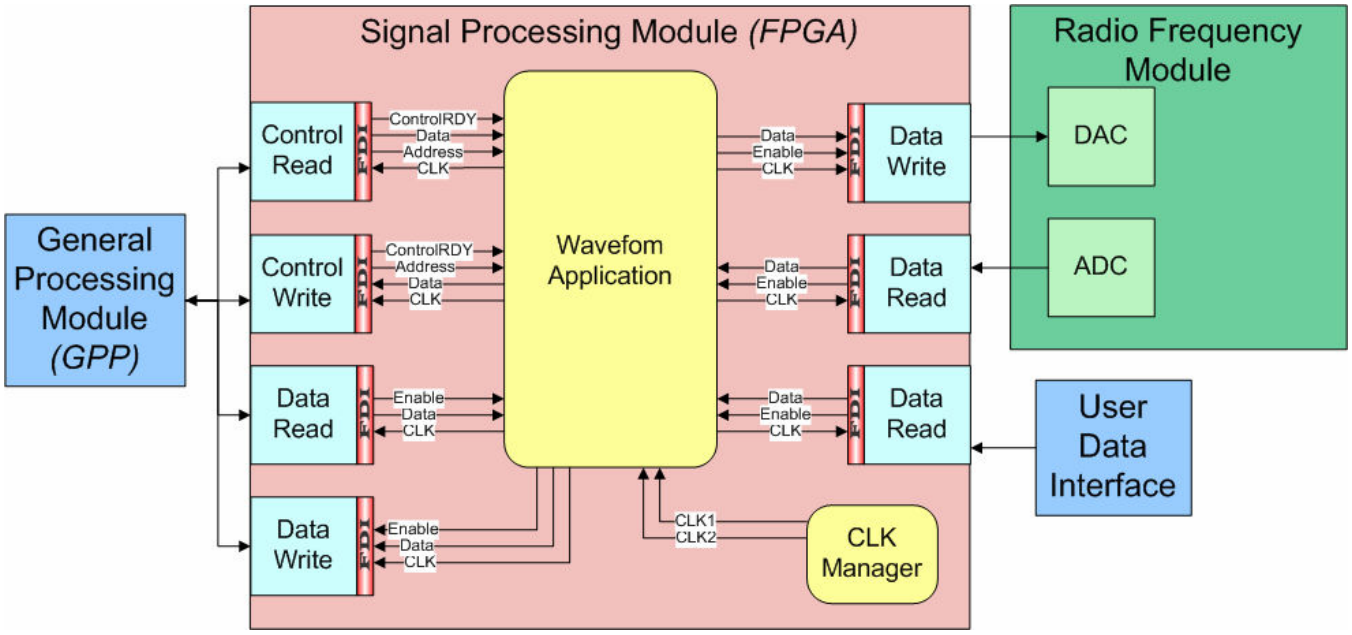

Figure 5: Implementation example of the Firmware Developer Interface.

The interface to the GPM was more difficult to implement because the GPM HID consisted of an address bus and a data bus, but the FDI was implemented to abstract the HID from the firmware as a control read/write and a data read/write. The control read interface was implemented using an asynchronous first in first out (FIFO) memory. The test waveform application in the FPGA could read a data address pair when the ControlRDY signal was asserted. The control write signal could not be implemented with a FIFO because of the GPP master FPGA slave data flow configuration. When the ControlRDY signal was asserted, the waveform application would place the data that corresponded to the requested address on the bus. The data read/write functions were implemented with asynchronous FIFOs. The enable signals indicated when a read or write operation was performed. Although these interfaces were implemented as separate interfaces to the firmware developer, the actual HID between the FPGA and the GPP consisted of a single common address and a data bus.

The interface to the ADC and DAC was very simple to implement. The enable signal indicated when the read or write was performed. The devices were also given a clock signal. The FDI implementation did not address ADC/DAC bit width variations. However, in the future a variable bit width control parameter could be added to the FDI.

The clocks that were generated by the clock manager were fed into the test waveform application for use by the waveform. However, the test waveform did not control the clock rate. It simply used the provided clocks as necessary.

6. FUTURE WORK

The proposed additions to the STRS Firmware Architecture Standard are the WFI and the FDI. The WFI is a set of common interfaces between waveform functions on the FPGA. The FDI is a common set of external firmware interfaces to devices outside the FPGA. The FDI on the FPGA and the WFI in the waveform have been implemented and tested separately. The integration of the two implementations is the next step. Plans are to target the platform on which the working FPGA FDI is implemented with the WFI-based transmit waveform. The ability to reconfigure the FPGA using the WFI and FDI will be demonstrated using controls from the STRS Infrastructure in the GPP. The FDI will then be implemented on another SDR platform and the waveform ported to that second platform. This will test and demonstrate how the proposed extensions to the STRS Firmware Architecture Standard enable waveform reusability, portability, and reconfiguration.

7. REFERENCES

[1] National Aeronautics and Space Administration, Headquarters, Space Telecommunications Radio System STRS Open Architecture Standard 1.0, Washington D.C., April 2006.