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. 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. Current standards were researched and new standard interfaces were proposed. The implementation of the proposed standard interfaces on a laboratory breadboard SDR will be presented.
STRS Compliant FPGA Waveform Development

by

Jennifer Nappier & Joseph Downey

SDR Forum Technical Conference, October 30, 2008

NASA Glenn Research Center
Outline

• STRS Architecture Standard
• STRS Waveform Development
• Proposed Addition to Firmware Waveform Development Architecture
• Proposed Addition to FPGA Device Hardware Abstraction Layer
• Conclusions
• Future Work
STRS Architecture Standard

- GPM – General-purpose Processing Module
- SPM – Signal Processing Module
- RFM – Radio Frequency Module
- HID – Hardware Interface Description
- HAL – Hardware Abstraction Layer

The current firmware section supports
- the use of Platform Independent Modeling and design techniques
- Modularity
- Internal and external HID within the SPM
Proposed Addition to Firmware Waveform Development Architecture:

Waveform Function Interface (WFI)
Definition of a Waveform

- the set of transformations applied to information transmitted over the air
- the corresponding set of transformations to convert received signals back to their information contents

Goals

- Design a platform independent waveform
- Design a waveform that is reconfigurable
- Design a waveform that is portable and reusable
Transmit Waveform Development

- Designed waveform using platform independent modeling tool
- Separated waveform functions into modular blocks
- Utilized custom, common interface
- Interfaces called the Waveform Function Interface (WFI)

- Portability, reuse
Data Sheets

• Standard documentation process
• Data sheets used to define interfaces

<table>
<thead>
<tr>
<th>Convolutional Encoder Waveform Function Interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Signal</strong></td>
</tr>
<tr>
<td>Data</td>
</tr>
<tr>
<td>Clock</td>
</tr>
<tr>
<td>Enable</td>
</tr>
<tr>
<td>Controls</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Data</td>
</tr>
<tr>
<td>Enable</td>
</tr>
</tbody>
</table>
Waveform Function Interface (WFI) Research

- **Open Core Protocol (OCP)**
  - Designed for IP cores
  - Master/Slave Architecture
  - Set of basic signals
  - Profiles
- **Wishbone**
  - 3 Modules:
    - SYSCON
    - Master
    - Slave
  - Standard signal naming convention
  - Emphasis on documentation
- **Mercury Computer Systems**
  - Component Portability Interface
    - Implementation of the proposed Specialized Hardware Supplement (SHS) of the SCA 3.1
    - Based on several Open Core Protocol (OCP) profiles
- **AMBA, Avalon, Xilinx, etc**
Waveform Function Interface (WFI) Conclusions

• A second transmit waveform has been implemented to comply with Wishbone

• Standardized documentation (e.g. data sheets)
  – Understanding the interface
  – Interoperability of functions
  – Portability

• The WFI allows the use of IP cores from vendors without disclosing proprietary designs

• The WFI protects IP core proprietary designs

• Waveform functions may not be portable if the code inside is platform specific

• Yet to determine if this approach is practical with respect to proprietary concerns
Proposed Addition to FPGA Hardware Abstraction Layers:

Firmware Developer Interface (FDI)
Platform Specific Wrapper

- Abstracts details of the platform from the application developer
- Accepts command and data information from the GPM and provides them to the application
- Functions:
  - Clock generation
  - Signal registering
  - Synchronization
  - Other non-waveform specific functions the platform requires
- No standardization of the interfaces exists
FPGA Hardware Abstraction Research

• Joint Tactical Radio System (JTRS)
  – Modem Hardware Abstraction Layer (MHAL)
    • Standard Message format implies standard HAL on GPM
    • New standard - June 2007

• Object Interface Systems, Inc
  – ORBexpress®
    • CORBA messages
    • STRS does not support CORBA
    • Virtex 4/5 only
Firmware Developer Interface (FDI)

- FDI – FPGA hardware abstraction layer
- Goals
  - Abstract SDR platform from waveform developer
  - Promote waveform portability and reuse
  - Support different design data flows
  - Promote platform independent design methodology
Firmware Developer Interface (FDI) Description

Data FDI – Common interface between data streaming devices

<table>
<thead>
<tr>
<th>Data FDI</th>
<th>Read</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>Direction</strong></td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td>In</td>
<td></td>
</tr>
<tr>
<td>Enable</td>
<td>Out</td>
<td></td>
</tr>
<tr>
<td>Clock</td>
<td>Out</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>Direction</strong></td>
</tr>
<tr>
<td>Data</td>
<td>Out</td>
</tr>
<tr>
<td>Enable</td>
<td>Out</td>
</tr>
<tr>
<td>Clock</td>
<td>Out</td>
</tr>
</tbody>
</table>

Control FDI – Common interface between control devices

<table>
<thead>
<tr>
<th>Control FDI</th>
<th>Read</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>Direction</strong></td>
<td></td>
</tr>
<tr>
<td>ControlRDY</td>
<td>In</td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>In</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td>In</td>
<td></td>
</tr>
<tr>
<td>Clock</td>
<td>Out</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>Direction</strong></td>
</tr>
<tr>
<td>ControlRDY</td>
<td>In</td>
</tr>
<tr>
<td>Address</td>
<td>In</td>
</tr>
<tr>
<td>Data</td>
<td>Out</td>
</tr>
<tr>
<td>Clock</td>
<td>Out</td>
</tr>
</tbody>
</table>
Firmware Developer Interface (FDI) Implementation on Space SDR
Firmware Developer Interface (FDI) Results

- General Purpose Processor Control Read signals
- Signals Abstracted and standardized
Improvements to the Firmware Developer Interface (FDI)

- Data read/write for GPP unnecessary
- Standard signal naming conventions
- Clock domain transition registers
Firmware Developer Interface (FDI) Conclusions

• Trades to consider when standardizing message passing protocol
  – Flexibility
  – Documentation
  – Portability
  – Proprietary

• A highly configurable FDI would be beneficial in waveform portability

• Need to conduct waveform porting experiments between SDRs on which the FDI has been implemented

• CoNNeCT offers an opportunity to experiment with FDI implementations on a space-based SDR test bed

• JTRS and industry partners offer potential standards to advance FDI and WFI

• The platform specific wrapper concept has been incorporated into the STRS Architecture