ASR-2300 Communications Interface Specification

Advanced Software Radio™

Loctronix®

Locked on Location

http://www.loctronix.com

MANAGED DOCUMENT	
Document Version:	1.03
Filename:	ASR-2300 Communications Interface Specification.docx
Last Revised	2-Aug-14

Document Control

Revision History

Revision	Date	Description	Author
V1.01	*11/29/2013*	Updated for release with A2300 project. Added sensor definitions	M.Mathews
V1.02	*12/11/2013*	Updated RF properties and cleaned up other interface specs to match current firmware.	M. Mathews
V1.03	*8/2/2014*	Miscellaneous documentation improvements. Added checksum verification to BIT. Added RF Profile Properties and query for path definitions.	M. Mathews

Contents

1. Introduction

This document details the communication interface for the ASR-2300. The ASR-2300 implements both USB and UART communication interfaces. These interfaces support the Loctronix Device Communications Interface (DCI) protocol and Waveform Component Architecture™ (WCA) software defined radio (SDR) hardware abstractions. The WCA is part of the Loctronix Advanced Software Radio™ (ASR) platform targeting navigation applications for GPS/GNSS challenged environments.

1.1 Intended Use

The information contained in this document is the proprietary property of Loctronix Corporation. Any unauthorized use of this information and related protocols strictly prohibited. The specification is managed by Loctronix and may change without notice. Every attempt will be made to maintain backward compatibility.

1.2 Related Documents

- ASR-2300 Datasheet
- “Device Communications Interface (DCI) Protocol Specification” version 1.0 or later (March 2013).

1.3 A2300 Open Source / Myriad RF Project

Loctronix supports the A2300 open source project, which provides open source software including drivers, firmware, and programmable hardware logic. These source artifacts are hosted at the Myriad RF project along with other related SDR platforms.

These tools and components are available under GNU 2.0 public license.

Please visit

Open Source Software and Drivers

https://github.com/myriadrf/A2300

http://myriad.org

Title:	ASR-2300 Communications Interface Specification	*Page:*	5 of 37
Subject:	Advanced Software Radio™	*Revision:*	9
Project: - - -	*Status:* Loctronix Proprietary	*Date:*	2-Aug-14

2. Communications Overview

The following figure shows the high-level communications interfaces for the ASR-2300. The interfaces comprise a proprietary USB interface supporting WCA driver communications, a standard CDC data interface (0AH), and a simple serial UART for communication via other devices: this includes Bluetooth or RS-232 adapter.

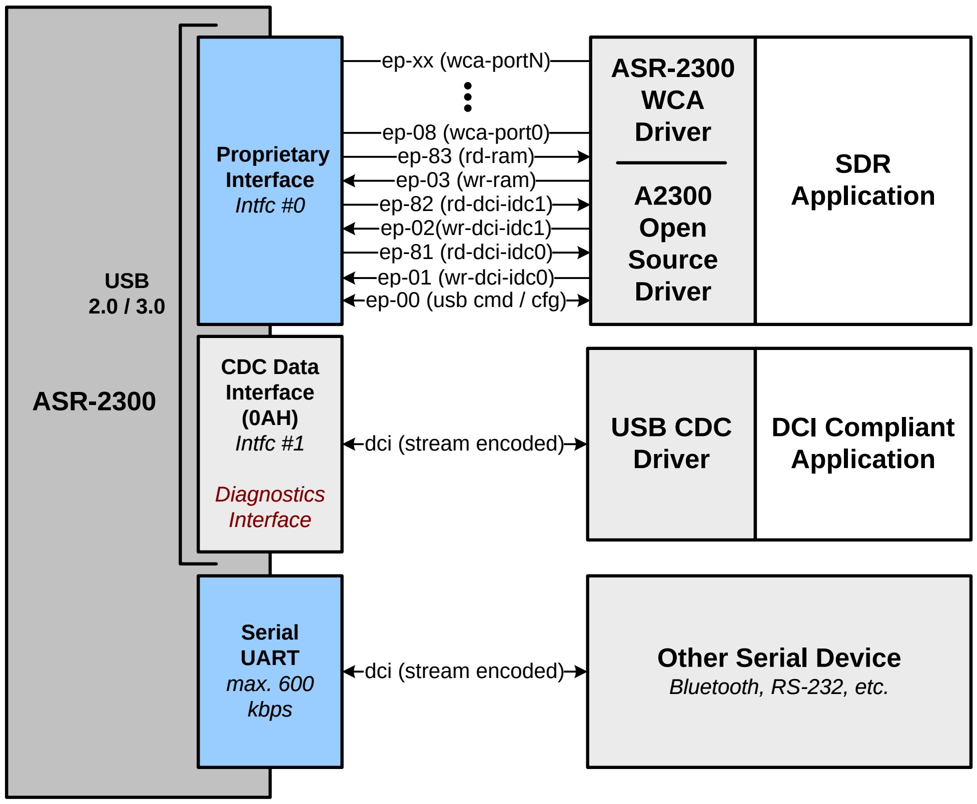

Figure 1 – ASR-2300 Current and Planned Communication Interfaces

Proprietary USB interface for integration with the WCA or A2300 drivers provides support for command and control via the DCI protocol and dedicated endpoints for reading and writing to RAM memory or various WCA ports defined within the FPGA. The maximum data rate for DCI based communications is limited to 600 kb per second, which is more than sufficient for handling configuration control activities. High-speed, up to 310 MB per second data transfer can be performed using the endpoints for RAM and WCA ports, these endpoints are connected directly to the FPGA FIFO queues and memory controller yielding maximum throughput.

Endpoints zero (ep0) through six (ep6) are fixed and available in all configurations of the ASR-2300. WCA Port endpoints (ep8 and greater) are dependent on the binary FPGA logic loaded. In most applications there will be at least one WCA Port defined. Both read (receive) and write (transmit) are supported. A number of Loctronix ASR waveforms support multiple read WCA Ports.

In future implementations, a generic CDC Data Interface (0AH) standard USB will be provided for diagnostic and low-data rate applications. Through this interface any DCI compliant application supporting serial communications can communicate with the ASR-2300 via standard serial interface. Asynchronous DCI messages can be communicated at rates as high as 600 kb per second. The CDC data interface capability of the ASR-2300 is meant for testing, debug, and low data rate applications not requiring WCA functionality on the host. For high performance applications, it is recommended that the proprietary USB interface supporting WCA be used.

From a communications data flow perspective, the following figure shows the data flow between various ASR-2300 and host components. These flows do not include the USB command and control messages, which are assumed to be part of the communications transport.

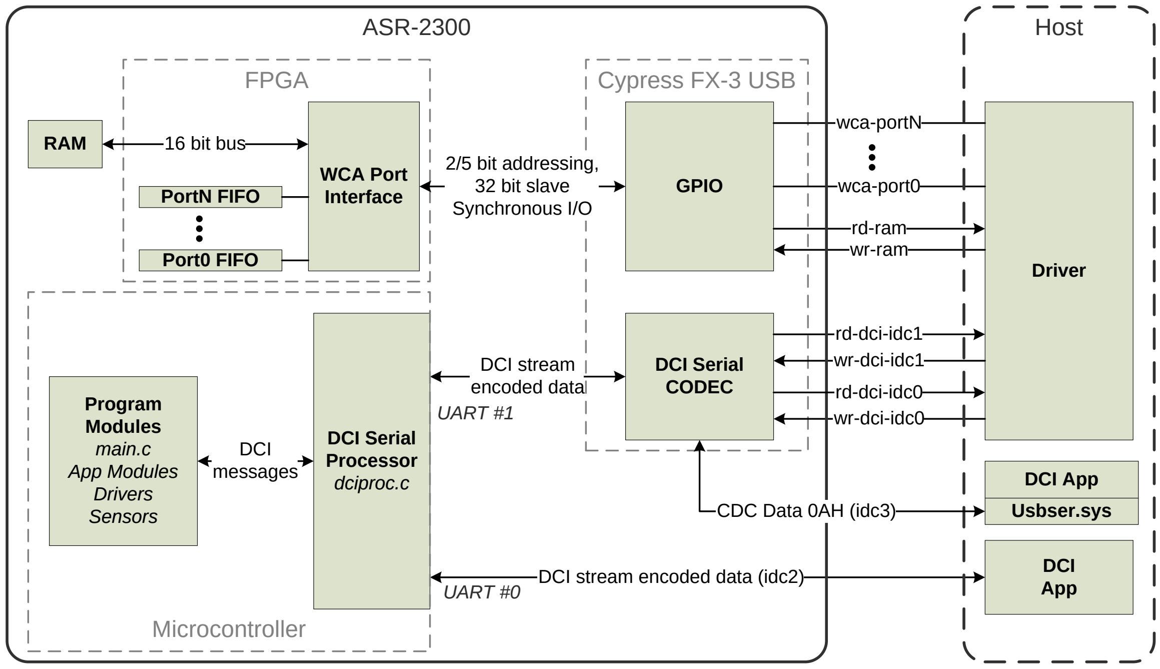

Figure 2 – The ASR-2300 Communication Data Flow

The ASR-2300 device is comprised of three primary functional blocks, Cypress FX-3 USB controller, a 16-bit microcontroller (RL78), and a high performance FPGA. The FX-3 implements a general-purpose I/O (GPIO) function that passes WCA Port and RAM endpoint data to the FX-3 interface in the FPGA, where the I/O is routed to RAM or one or more WCA Port FIFO's. The FX-3 processes host DCI communications with a DCI serial coder/decoder (codec), which formats DCI messages for serial stream binary transport (see section 3.1 in the DCI interface protocol specification). Encoded DCI messages are communicated between the FX-3 and the device microcontroller using a 600 kb per second data rate via UART #1. These DCI messages and encoded DCI messages via UART #0 are processed by the microcontroller DCI serial message processor, which communicates un-encoded messages with the various other program modules.

The DCI serial processor can direct communications to the three external interfaces specifically using the DCI conversation identifier. DCI conversation zero (idc0) messages are routed via endpoints one and two (ep1, ep2). DCI conversation one (idc1) are routed via endpoints one and two (ep3, ep4). DCI conversation 2 is directed to UART #0. Messages assigned to DCI conversation identifier three (idc3) are processed by the FX-3 microcontroller. This conversation is limited to communication between the microcontroller and the FX-3, though future versions of the host driver may also communicate directly.

Most DCI communications will be dedicated communications to the particular interface using conversation identifiers 0, 1, or 2. Conversation identifier 3 enables the microcontroller to configure the FX-3 functionality during initialization.

Support for specific DCI messages is dependent on the specific interface. Please review interface specifications in this document for specific details on which messages are supported.

3. USB Proprietary Interface

To support high-speed interfaces the ASR-2300 defines a proprietary USB interface. The interface endpoints can be flexibly configured based on the firmware logic loaded into the FPGA in compliance with the WCA Hardware Abstraction Layer (HAL) specifications.

This section defines the details for working with the proprietary interface. The interface is intended for use via the ASR-2300 WCA or A2300 drivers implemented in support of the WCA platform available for Linux and Windows -based devices. The WCA driver for the ASR-2300 accesses the USB interface directly using lib-usb 1.0(Linux) or Cypress APIs (Windows), enabling high-speed bulk transfer of data between the host platform and the ASR-2300. The minimum configuration defines seven (ep0 through ep6) and points for configuration, control, and RAM operations. Additional endpoints supporting WCA ports may be defined depending on the particular FPGA logic loaded.

This section details the specific specifications for each of the endpoints and implementation considerations.

3.1 Endpoint-0 (ep-00) – USB Configuration and Control

This endpoint is a messaging type endpoint supporting configuration, command and control messages defined by the USB specification for enumerating and configuring USB devices. Message protocol supported on this interface is in compliance with USB 2.0 and 3.0 standards. Standard support implemented by the FX-3 supports endpoint 0 communications. At this time no specific modifications of this command protocol are expected.

3.2 Endpoint-1 (ep-01) – wr-dci-idc0

Endpoint 1 is a USB 1.1, full-speed output endpoint (maximum data rate 600 kb per second) bulk transfer mode stream. The endpoint provides streaming of DCI messages from the host to the device using conversation 0 (idc0). The host typically writes uses this endpoint to send command and control messages to the microcontroller that require synchronous response. This endpoint is used in conjunction with endpoint 81h (ep-81) to provide synchronous command-and-control communications functionality.

Though this interface is not explicitly limited to support synchronous communications, this convention is provided such that asynchronous messages will not be transmitted or received using endpoints one or two (ep-01 or ep-81).

Asynchronous messages like debug (5,01) and typed data record (20,13) messages are not supported on endpoints one or two (ep1, ep2). To support asynchronous DCI messages on the USB, implement support for the asynchronous DCI communications and points described in sections 3.4 and 3.5.

All DCI messages sent via endpoint 1 shall not be encoded and must have a length less than 320 bytes. TCP transport semantics shall be used to send messages to the device.

3.3 Endpoint-81 (ep-81) – rd-dci-idc0

Endpoint 2 is a USB 1.1, full-speed input endpoint (maximum data rate 600 kb per second) bulk transfer mode stream. The endpoint provides streaming of DCI messages from the device to the host using conversation 0 (idc0). The host typically reads this endpoint to receive responses to command and control messages from the microcontroller. This endpoint is used in conjunction with endpoint 01 (ep-01) to provide synchronous command-and-control communications functionality.

Though this interface is not explicitly limited to support synchronous communications, this convention is provided such that asynchronous messages will not be transmitted or received using endpoints one or two (ep-01 or ep-81).

~~Asynchronous messages like debug (5,01) and typed data record (20,13) messages are not supported on endpoints one or two (ep1, ep2). To support asynchronous DCI messages on the USB, implement support for the asynchronous DCI communications and points described in sections 3.4 and 3.5.~~

All DCI messages received via endpoint 81 shall not be encoded and must have a length less or equal to 320 bytes. Messages shall use the TCP transport semantics defined in section 3.2 of the "DCI Protocol Specifications". This requires a two byte signed integer prefix sent before transmitting the DCI message.

3.4 Endpoint-02 (ep-02) – wr-dci-idc1

Endpoint 02 is a USB 1.1, full-speed input endpoint (maximum data rate 600 kb per second) providing bulk transfer mode. The endpoint provides streaming of DCI messages from the host to the device using conversation 1 (idc1). This endpoint along with endpoint 82 (ep-82) is provided for general purpose asynchronous DCI communications. Typically the host and ASR-2300 will communicate data and other information that occurs outside of command and control functionality.

All DCI messages sent via endpoint 02 shall not be encoded and must have a length less or equal to 320 bytes. Messages shall use the TCP transport semantics defined in section 3.2 of the "DCI Protocol Specifications". This requires a two byte signed integer prefix sent before transmitting the DCI message.

3.5 Endpoint-82 (ep-82) – rd-dci-idc4

Endpoint 82 is a USB 1.1, full-speed input endpoint (maximum data rate 600 kb per second) providing bulk transfer mode. The endpoint provides streaming of DCI messages from the host to the device using conversation 1 (idc1). This endpoint along with endpoint 02 (ep-02) is provided for general purpose asynchronous DCI communications. Typically the host and ASR-2300 will communicate data and other information that occurs outside of command and control functionality.

All DCI messages sent via endpoint 4 shall not be encoded and must have a length less or equal to 320 bytes. Messages shall use the TCP transport semantics defined in section 3.2 of the "DCI Protocol Specifications". This requires a two byte signed integer prefix sent before transmitting the DCI message.

3.6 Endpoint-03 (ep-03) – wr-ram

This endpoint provides bulk transfer of data to a specific location in memory. Each operation specifies a beginning and number of 16 bit words to be transferred. Details of how this exactly is done will be specified in a later version of this document. Memory operations will require a DCI property message to set up memory transfer operation and then this endpoint to write bulk data to the specified memory locations. Writing continues until the specified number of 16-bit words are written.

3.7 Endpoint-83 (ep-83) – rd-ram

This endpoint provides bulk transfer of data to a specific location in memory. Each operation specifies a beginning and number of 16 bit words to be transferred. Details of how this exactly is done will be specified in a later version of this document. Memory operations will require a DCI property message to set up memory transfer operation and then this endpoint to read bulk data from the specified memory locations. Reading proceeds until the specified number of 16-bit words are read.

3.8 WCA Port Endpoints

Additional endpoints, endpoint 04 (ep-04) and higher, may be specified by a particular FPGA logic load. These endpoints are mapped directly to WCA Port FIFO's implemented inside the FPGA. These FIFO's are dedicated as either read or write FIFOs. This interface does not provide for bidirectional FIFO's. Through this endpoint bulk transfer operations of FIFO data occur. Data should be read or written in chunks of 512 bytes in alignment with USB conventions for maximum data transfer. Overflows and under flows of FIFO's may result in triggers of error conditions that will percolate through the microcontroller DCI support up into the WCA driver. In the event of overflow and underflow conditions, waveform functionality may need to be reset. It is important that the rate of data transfer be managed by the host to ensure that these conditions do not occur.

Enumeration of these endpoints will be provided either through the standard USB enumeration capability or through special DCI message implemented as part of the WCA category. The speed of these endpoints will be configured based on the FPGA configuration. Depending on the host connection, either USB 2.0 or USB 3.0, certain WCA HAL components requiring Super Speed endpoints may not be supported.

4. USB CDC Data Interface (OAH)

NOTE: ***CDC Data Interface Not Currently Implemented, it will be available in future releases.***

This interface supports the data interface (0AH) specification defined in version 1.1 of the "Universal Serial Bus Class Definitions for Communication Devices". This implementation will essentially provide the asynchronous endpoint functionality described for the proprietary interface defined in the previous section for sending and receiving DCI messages. Through this interface, the standard USB to serial drivers can be used to interface the host software with the ASR-2300 diagnostics functions.

The maximum data rate is limited 600 kbps, which is the UART speed between the cypress and microcontroller. To use this interface in Windows requires installation of device information in the registry. The .inf file named 'ASR-2300 USB Diagnostics COM Driver.inf' is provided as part of the ASR-2300 SCP Workbench Diagnostics Plugin. This driver installation maps the CDC Data interface to the usbser.sys driver enabling communication with the ASR-2300 via standard communications ports.

This interface will be supported in a limited form for customers depending upon the product edition provided.

5. Serial UART Interface

The serial UART interface provides full asynchronous serial binary transport encoded DCI communications directly with the ASR-2300 microcontroller. The purpose of this interface is to support low data rate applications, particularly mobile applications (e.g. Bluetooth communications), development, and diagnostic testing. The maximum supported data rate is 600 kb per second.

The standard serial communications settings are as follows:

- Speed: **600 Kbits/sec**
- Data length: **8 bits**
- Stop Bits: **1 bit**
- Parity: **none**
- CTS / RTS: **none**

This interface provides full support of DCI infrastructure, WCA messages. Selected standard DCI messages are also supported as well and are defined in Section 6.4. Depending on the FPGA load, the interface may also provide streamed I/Q data from using the typed data records.

Debugging and testing of the ASR-2300 microcontroller code will be implemented through this and interface supporting custom device messages defined in Section 9 of this document.

6. DCI Protocol Support

This section discusses the details of DCI protocol support for the ASR-2300. As the DCI protocol is not necessarily device specific, this section identifies the specific conventions used and messages supported. The ASR-2300 is a DCI compliant device supporting the WCA HAL specifications for hardware abstraction.

6.1 Endian Format

RL78 default endian format is little endian. To minimize byte swapping on the device, the wire protocol will operate in little endian format as well. Request of the endian format using messages 20,86 will result in the device specifying little endian as the protocol format in response message 20,06. The query response will look like:

```
Host>>   20 86
Device>> 20 06 02 0F 00
```

Host DCI libraries should request the endian format message and configure and configure data extraction for little endian format.

6.2 Payload Encoding

Payload encoding is used on the various USB and UART interfaces. Please see Sections 3, 4, and 5 for specific details regarding payload encoding.

6.3 Infrastructure Messages

All DCI infrastructure messages are supported.

6.4 Standard Messages (20)

The following standard DCI messages are supported.

Table 1 – ASR-2300 Supported Standard DCI Messages

Message ID	Name
20,01	Reset
20,02 / 20,82	Version Information / Query
20,06 / 20,86	Endian Format / Query
20,13 / 28,93	Typed Data / Query

6.5 WCA Messages (21)

The ASR-2300 is a fully WCA HAL compliant device providing configuration and control functions via DCI and high-speed data transfer through USB bulk data I/O. The device processes all WCA messages specified in the DCI specification. The supported messages are show in the table below.

Table 2 – ASR-2300 Supported WCA DCI Messages

Message ID	Name
21,01	Typed Properties (Maximum 20 properties per message.)
21,81	Typed Properties Query

21,02	Execute Action
21,03	Binary Image Transfer Info
21,04	Binary Image Transfer Frame
21,83	Binary Image Transfer Query
21,05	HAL Event Notification

These messages specify standard transaction supported by the WCA HAL. Per the WCA DCI specifications (Section 7), all WCA HAL components must have an identifier between 0 and 255 (FF hex). Identifiers between 128 and 255 (80 and FF hex) are reserved for fixed hardware components specific to a particular hardware implementation. By definition, component 0 (0 hex) is the default WCA hardware component corresponding to the programmable FPGA logic.

7. Custom Typed Data Messages

The ASR-2300 provides a typed data message for querying the status of various components within the device. These are specified in this section. To access the status information send a Typed Data Query (20,93) message with the specified typeid and flags. The device will respond with the Typed Data (20,13) message containing the requested information.

7.1 General Status Information (typeid = 0)

The general status typed data record provides information about the general state of the 2300. This is a high level status for the device.

```
typedef struct
{
    byte            State;
    byte            AutoUpdate;
    uint16          Errors;
    uint32          Options;

    /* --- GPIO related status ---*/
    …TODO…

    /* --- other status --- */
    …TODO…

} Asr2300_Status;
```

7.2 RF0, RF1 Status Information (typeid = 0x10, 0x20)

Fixed components RF0, RF1 provide low-level status information about the Lime transceivers #1 and #2. The typed data message 20,13 is used to send this status information to the host. The typeids prefixed with 0x10 are for Lime transceiver #1 and typeid's prefixed with 0x20 are for Lime transceiver #2.

This typed data record provides status information about the configuration and operating state of the RF0 component (Lime transceiver #1).

The following structure defines the data provided.

```
typedef struct
{
    /* --- LIME Transceiver STATUS INFORMATION ---*/
    …TODO…

} RF_Status;
```

7.3 RF0, RF1 Lime Register Data (typeid = 0x11, 0x21)

This message is used to send receive low-level register data directly to the lime chips. This message is meant for initial testing, debug, and calibration tuning. Do not use for application development.

The typed data (20,13) message data payload has the following structure:

```
typedef struct
{
    byte idReg;
    byte addrStart;
    byte addrEnd;
```

```
            byte regvalues[addrEnd-addrStart];

        } LimeRegData;
```

To query for registers issue the (20,93) message where the flags field specifies the range of registers to return. The low byte of the flags field is the start address and the high byte is the end address. Encode and decode as follows for endian independent implementation.

```
Uint16 flags =   addrEnd <<8 | addrStart;
Uint16 addrStart = flags & 0xff;
Uint16 addrEnd = flags>>8;
```

An example interaction might look to query for registers 0x10 through 0x15 from Lime #0 (typeid = 0x11) in big endian format:

```
Host >> 20 93 11 15 10

Device>> 20 13 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 05 12 23
44 A1 F3 B3
```

This results in register values for the range 0x10 to 0x15 of 0x12, 0x23, 0x44, 0xA1, 0xF3, and 0xB3.

7.4 RF0, RF1 Lime SPI Data Pass Through (typeid = 0x12, 0x22)

11/29/13 - Function is currently disabled and removed, it can be re-enabled in the future if needed.

These messages are used to send binary data from Lime Micro Software Tools directly to the Lime Chips. The typeid=0x12 is for RF0 (Lime Chip #1), and typeid=0x22 is for RF1 (Lime Chip #2). The ASR-2300 Workbench Plug-in has a virtual serial port pass through tool that enables the Lime Micro tools to be used to configure the Lime RF transceiver chips directly. These Typed Data messages transport the raw uninterpreted serial binary values sent/received between the Lime Micro Software and Lime Chips.

Future versions may interpret these Lime Micro Messages which have known structure based on experience from the Model 2200 development.

```
30 rr dd [rr dd] [rr dd]    - Write registers
20 rr [rr] [rr]             - Read registers
10                          - Reset
11                          - Reset active
12                          - Reset inactive
A0 ??                       - Write Xil
A1 ??                       - Read Xil
A3 ??                       - Write AD9552
A6 ??                       - Read AD7691
AD ??                       - Write ADF
```

7.5 Low Speed IQ Data (typeids = 0x90, 0x91, 0x92, 0x93)

The microcontroller provides low speed IQ data ports to receive and transmit I/Q samples via the serial UART interface. Four ports are defined: two input (0x90 and 0x92) and two output (0x91 and 0x93). Configuration and control of these ports is discussed in section 8.9. It should be noted that these parts are only supported if the FPGA is loaded with HDL code that supports the low speed data ports. If these ports are not provided, then sending and receiving typed data will ignored and unavailable.

Maximum data rate on these ports must not exceed a combined I/Q (16-bit samples) sample rate of 16.384 KHz otherwise the serial interface will be saturated. These ports receive and transmit I/Q sample data using the following data format.

```
struct IQSamples
{
   //Header (not currently defined)
   //sample data.
   struct { int16 I, int16 q } Samples[64];
}
```

64 complex 16 bit samples (4 bytes per sample, total 256 bytes) are transmitted per typed data report (20,13) message.

7.6 Sensor Data (typeids= 0x83 through 0x87)

The ASR-2300 produces sensor data in a manner similar to the low-speed data ports. Users can receive a continuous stream or single-shot as specified. Sensor data specifications are defined in the sensor component specifications, which are generally a fixed field structure. By default a component that generates sensor data, specifies its component identifier as the type id. For the ASR-2300 there are multiple components that generate sensor data.

Table 3 – Components Providing Sensor Data

Component ID	Name
0x83	sMotion sensor
0x84	Accelerometer sensor
0x85	Gyroscope sensor
0x86	Compass sensor
0x87	Pressure sensor

Conventions for accessing sensor data involves sending an execute action message with the following minimum data specifications:

```
struct {
   uint16 ctrlFlags;
   uint16 dataFlags;
   uint16 dataFreq;
   uint16 ctSamples;
};
```

The fields are defined in the following table.

red additional may be specified in accordance with the component specifications. Table 4 – Sensor Data Execute Action Message Data Specification

Field	Description
ctrlFlags	Bit # Description 0-1 Mode Select 0 =Disabled 1 = Single Shot; requires a 20,93 message for host or device to send data. 20, 93 Can be sent repeatedly for updated snapshots. 2 =ontinuous data transfer; data is sent automatically. 2-7 Reserv C ed.
dataFlags	Not Defined. See specific sensor for specification of data configuration flags.
dataFreq	Data frequency in Hz. Supported frequencies are dependent upon the particular component..
ctSamples	Number of samples per report. Maximum size of report must be less than maximum DCI message size.

7.7 RF Profile Path Descriptors (typeid= 0x94)

The RF Profile component can provide the profile path descriptors upon request by querying for type id 0x94. This will return the profile path identifier and short text description of its purpose. Each path identifier and description is 18 bytes long with structure

```
struct {
   byte id;
   char descr[17];
};
```

The type data length specified is the total number of bytes. Divide that value by 18 to retrieve the number of path id/descriptor records uploaded. A query (20,93) will result in this as a response. Note, that if there are more than 14 total descriptors multiple messages may be sent. If the length of the message is 14*18 (252) bytes, then look for additional messages.

8. Fixed Hardware Components

The following table specifies the component identifiers for various functional components of the ASR-2300. These are fixed and independent of the logic loaded into the FPGA. The WCA DCI messages must be directed to one of these components or specific components defined by the WCA logic.

Table 5 – ASR-2300 Fixed WCA Component Identifiers

Component ID (hex)	Name	Description
0x00	default	Default WCA hardware DSP component. This is the waveform component specified by programmable FPGA logic.
0x80	Microcontroller	Microcontroller general configuration and control
0x81	RF0	RF0 Component, Lime #1 transceiver interface
0x82	RF1	RF1 Component, Lime #2 transceiver interface
0x83	Motion	Motion sensor
0x84	Accelerometer	Accelerometer sensor
0x85	Gyroscope	Gyroscope sensor
0x86	Compass	Compass sensor
0x87	Pressure	Pressure sensor
0x88	Flash	Serial Flash
0x89	Power	Power management / Battery Charger Functionality
0x8A	FX-3-A	(reserved) Cypress FX-3 Component A
0x8B	FX-3-B	(reserved) Cypress FX-3 Component B
0x90	LSDP0	Low speed data port 0 (input)
0x91	LSDP1	Low speed data port 1 (output)
0x92	LSDP2	Low speed data port 2 (input)
0x93	LSDP3	Low speed data port 3 (output)
0x94	RF Profiles	RF profile path configuration and calibrations.
0xB0	FPGA	FPGA WCA HAL configuration and control.

The remainder of this section specifies the typed properties, DCI Message (21,01) are defined for each component.

8.1 Default Waveform Hardware Component (0x00)

The default waveform hardware component is the programmable FPGA logic providing the hardware DSP processing. This component corresponds to the WCA C++ class derived from Ltx::Wca::HdwComponent class, which represents the component in the WCA software framework.. The specific properties are dependent on the default waveform implementation. However, it is strongly recommended that the following properties be defined

Table 6 – Default Waveform Hardware Component Typed Properties.

ID (hex)	Name	Type	Description
00	RESERVED	--	RESERVED DO NOT USE.
01	IdWaveform	uint16	Read only property specifies waveform identifier
02	Version	uint16	Read only property specifies waveform version information in the form of two 8-bit hex values. [ver].[rev]
03	PortCaps	uint16	Read only property specifies WCA port capabilities. This property indicates which read and write ports are active. The lower eight bits specify the eight possible read ports (transmit), the upper eight bits specify the eight possible write ports (receive). Ports are numbered by the bit position. For example: bit0 = port0 (read), bit9 = port0 (write).
04	RfConfig (optional)	byte	Read/Write RF property specifies configuration control register. This is a recommended register for software control of RF configuration. If DSP controls the switching of the RF interface, this property can be ignored. Bit settings are as follows: Bit # Description 0 GPS 3.3V DC Bias selection set to 1 to enable, 0 to disable active antenna feature. 1 GPS external input selection: set to 1 for external input, 0 for internal antenna input. 2 ISM band external input selection: set to 1 for external input, 0 for internal antenna input. 3 ISM band transmit selection: set to 1 to transmit, set to 0 receive. ISM band path only supports half duplex operation. 4 Wideband RF #1 RX receive select: set to 1 to receive, set to 0 to transmit. Provides half-duplex operation on the Wideband RF#1 external RF I/O connector. 5 Wideband RF #2 RX receive select: set to 1 to receive, set to 0 to transmit. Provides half-duplex operation on the Wideband RF#2 external RF I/O connector.
05	LedStatus (optional)	byte	Write only property specifies LED Status control register. Turns on and off LEDs. This is a recommended control register if not overridden by DSP specifications. LEDs are defined as follows: Bit # Description 0 Turns on (1) and off (0) Led_0 1 Turns on (1) and off (0) Led_1 2 Turns on (1) and off (0) Led_2 3 Turns on (1) and off (0) Led_3

Ports are mapped to USB endpoints as shown in the following table. Port mappings to USB endpoints should be translated within the WCA driver implementation. WCA components reference logical port numbers (0 – 15).

Table 7 – WCA Port Capability ID / USB Endpoint Mapping

Port # / Bit Position	USB Endpoint ID	USB Host Direction	Port # / Bit Position	USB Endpoint ID	USB Host Direction
00	08h	Output	08	88h	Input

01	09h	Output	09	89h	Input
02	0Ah	Output	10	8Ah	Input
03	0Bh	Output	11	8Bh	Input
04	0Ch	Output	12	8Ch	Input
05	0Dh	Output	13	8Dh	Input
06	0Eh	Output	14	8Eh	Input
07	0Fh	Output	15	8Fh	Input

8.2 Microcontroller (0x80)

Table 8 – Microcontroller Typed Properties.

ID (hex)	Name	Type	Description

The following table identifies the Execute Action messages supported by microcontroller component.

Table 9 – Microcontroller Execute Action Messages

Id Action (hex)	Length Data	Data Specification
00	0	Update RL-78 firmware via flash. Action causes firmware stored in the flash to be copied into the RL-78 program non-volatile memory. The following sequence is required to update firmware: 1) User erases flash containing FPGA logic (wait about 20 seconds) 2) User downloads firmware to flash via BIT using the name ‘firmware’. 3) User then executes **this action command.** a. Microcontroller runs from RAM and copies firmware to non-volatile memory. b. Microcontroller erases flash and then reboots. 4) User downloads new FPGA logic to flash. 5) Update complete.
10	0	Resets RF#0 FPGA interface.
11	0	Resets RF#1 FPGA interface.
12	0	Resets DSP FPGA Container
13	0	Resets USB FPGA Interface

8.3 RF0 (0x81)

RF0 component controls the configuration of the Lime #1 RF transceiver. All Properties in Table 10 are Read/Write unless otherwise specified.

Table 10 – RF0 Typed Properties.

ID (hex)	Name	Type	Description
00	RFCtrl	byte	The RF Configuration Control register specifies Lime RF Configuration Settings: Bit # — Description 0 — RX Analog Output – Enable (1) / Disable (0). 1-7 — Reserved.
0C	RfCtrl2	byte	This register controls Lime #1 amplifier and bypass functions; turns on/off various amplifiers for the various RF paths. bit# — Description 0 — RXIN1 (**GPS**) LNA Enable (1) / Disable (0). 1 — RXIN2 (**PCS**) LNA Bypass (1) / Inline (0). Bypasses the PCS LNA for strong signals. 2 — RXIN3 (**Wideband**) RX Amplifier Enable (1) / Disable (0) 3 — TXOUT1 (**Wideband**) TX Amplifier Enable (1) / Disable (0) 4 – 7 — Reserved.
01	SampleCtrl	byte	Specifies the sampling configuration and control register. This specifies the configuration of baseband sampling data. Bit # — Description 0-1 — RX Input Mode 0 = tx_iq loopback 1 = rf_rxdata input (dc bias removed); 2 = rf_rxdata input (raw); 3 = Fixed Test Pattern i = 256, q = 256; 2-3 — TX Ouput Mode 0 = rf_rxdata loopback 1 = rf_txdata output (DSP data). 2 = Fixed Test Pattern i = 256, q = 256; 3 = Fixed Test Pattern i = 0, q = 0; 4 — "rf_rxen" enables disables the receive ADC function. set to 1 to enable, 0 to disable. 5-7 — Reserved.
0D	RxPath	byte	Specifies the current RF path profile ID used to configure the RF0 receiver functions. The default path profiles and IDs are defined as follows. See RF Path profiles for more information on customizing profiles and calibration data. Value — Description 0x40 — RX Disabled 0x41 — L1 GPS (1575.42 MHz, 3 MHz BW) Internal Antenna 0x42 — L1 GPS (1575.42 MHz) External Input 0x43 — PCS (1930 MHz to 1990 MHz) Input 0x44 — Wideband (400 MHz to 3.8 GHz) Input
0E	TxPath	byte	Specifies the current RF path profile ID used to configure the RF0 transmitter functions. The default path profiles and IDs are defined as follows. See RF Path profiles for more information on customizing profiles and calibration data.

			Value / Description 0x00 TX Disabled 0x01 Wideband (400 MHz to 3.8 GHz) Output
02	RxGain	byte	Specifies the gain setting for the receiver. Valid values are between 0 and 20 equal to 0 to 60 dB gain in 3dB steps. Gain = val * 3; // in dBs. Lime recommends only using up to 30 dB gain for normal use.
03	RxFrequency	uint32	Property specifies the frequency in KHz. This is a coarse frequency specification. Use the RxNcoCount property to specify frequency more precisely.
~~04~~	~~RxNcoCount~~	~~uint32~~	~~Sets the Lime NCO counter to the specified value. This is the preferred method as it is more accurate than specifying the frequency.~~
05	RxBandwidth	byte	Property specifies bandwidth of the receiver. Legal values are between 0 and 15. **These are "half bandwidth" values given I/Q processing.** Value / Bandwidth (MHz) — Value / Bandwidth (MHz) 0 14 (default) — 8 2.75 1 10 — 9 2.5 2 7 — 10 1.92 3 6 — 11 1.5 4 5 — 12 1.375 5 4.375 — 13 1.25 6 3.5 — 14 0.875 7 3 — 15 0.75
06	TxGain	byte	Property specifies the AGC gain setting for the transmitter. Valid values are between 0 and 56 equal to 0 to 56 dB gain in 1dB steps. Gain = val in dBs.
07	TxFrequency	uint32	Property specifies the frequency in KHz. This is a coarse frequency specification. Use NcoCount property to specify frequency more precisely.
~~08~~	~~TxNcoCount~~	~~uint32~~	~~Sets the Lime NCO counter to the specified value. This is the preferred method as it is more accurate than specifying the frequency.~~
09	TxBandwidth	byte	Specifies bandwidth of the transmitter. Legal values are between 0 and 15. **These are "half bandwidth" values given I/Q processing.** Value / Bandwidth (MHz) — Value / Bandwidth (MHz) 0 14 (default) — 8 2.75 1 10 — 9 2.5 2 7 — 10 1.92 3 6 — 11 1.5 4 5 — 12 1.375 5 4.375 — 13 1.25 6 3.5 — 14 0.875 7 3 — 15 0.75
0A	RxRssi	uint16	Read only register of I and Q received signal strength indication values I and Q RSSI values truncated to the 8 high order bits. RSSI is the average absolute value of the received data over the sampled interval. 24 bit counter is used so sampling interval should be around (~0.3 msec). Register contains both the I and Q RSSI values as follows: bit# / Description 0-7 In phase RSSI values (0 - 256) low 4 bits are not returned since it assumed gain will always be turned up enough to hit at least 4 bits of the system. 8-15 Quadrature phase RSSI values 0 - 256) low 4 bits are not returned since it assumed gain will always be turned up enough to hit at least 4 bits of the the system.
0B	RxBias	uint32	In-phase and Quadrature ADC Bias Read register. Implements 24 bit

	integrator to compute the average bias value over 4096 samples returns the high 12 bits for each I and Q stream indicating the average bias value over the period. Further LPF in the microcontroller may be required to get longer term average. The values stored in the 32 bit register as follows: bit# Description 0-15 The in-phase RSSI 12 bit value represented as a uint16 value (little endian). 16-31 The quadrature phase RSSI 12 bit value represented as a uint16 value (little endian).

The following table identifies the Execute Action messages supported by RF0.

Table 11 – RF0 Execute Action Messages

Id Action	Length Data	Data Specification
0x80	0	Tune the Lime Micro Receiver VCO. No arguments are specified.
0x81	0	Tune the Lime Micro Transmitter VCO. No arguments are specified.
0x82	0	Lime Micro Receiver I/Q Calibration. No arguments are specified.
0x83	0	Lime Micro Transmitter I/Q Calibration. No arguments are specified.
0x84	0	Top Level Calibration.
0x85	0	Updates the stored RX profile with calibration data. Must be in cached mode to actually save the data.
0x86	0	Updates the stored TX profile with calibration data. Must be in cached mode to actually save the data.
0x87	0	Resets the top level calibration..
0x88	0	Resets the RX calibration values.
0x89	0	Resets the TX calibration values.

The following Table 12 shows the supported typeids for sending and received TypedData messages (20,13). For more information please see Section 7.

Table 12 – RF0 Typed Data Messages

typeid	Description
0x10	Lime Micro Status Information
0x11	Lime Micro Register Read / Write
~~0x12~~	~~Lime Micro Serial Data Pass Through~~

8.4 RF1 (0x82)

RF1 component controls the configuration of the Lime #2 RF transceiver. All Properties in the Table 13 are Read/Write unless otherwise specified.

Table 13 – RF1 Typed Properties

ID (hex)	Name	Type	Description
00	RFCtrl	byte	RF Configuration Control register specifies Lime RF Configuration Settings: Bit # — Description 0 — RX Analog Output – Enable (1) / Disable (0). 1-7 — Reserved.
0C	RfCtrl2	byte	This register controls Lime #2 amplifier and bypass functions; turns on/off various amplifiers for the various RF paths. bit# — Description 0 — RXIN1 (**UHF**) RX Amplifier Enable (1) / Disable (0). 1 — RXIN2 (**ISM**) RX LNA Bypass (1) / Inline (0). Bypasses the LNA amplifier for strong signals. 2 — RXIN3 (**Wideband**) RX Amplifier Enable (1) / Disable (0) 3 — TXOUT1 (**ISM**) TX Amplifier Enable (1) / Disable(0) 4 — TXOUT2 (**Wideband**)Wideband TX Amplifier Enable (1) / Disable (0) 5 – 7 — Reserved.
01	SampleCtrl	byte	Allows sampling configuration and control register. This specifies the configuration of baseband sampling data. Bit # — Description 0-1 — RX Input Mode 0 = tx_iq loopback 1 = rf_rxdata input (dc bias removed); 2 = rf_rxdata input (raw); 3 = Fixed Test Pattern i = 256, q = 256; 2-3 — TX Ouput Mode 0 = rf_rxdata loopback 1 = rf_txdata input. 2 = Fixed Test Pattern i = 256, q = 256; 3 = Fixed Test Pattern i = 0, q = 0; 4 — "rf_rxen" enables disables the receive ADC function. set to 1 to enable, 0 to disable. 5-7 — Reserved.
0D	RxPath	byte	Specifies the current RF path profile ID used to configure the RF0 receiver functions. The default path profiles and IDs are defined as follows. See RF Path profiles for more information on customizing profiles and calibration data. Value — Description 0xC0 — RX Disabled 0xC1 — UHF (300 MHz to 650 MHz) Input 0xC2 — ISM (2.4 GHz to 2.48 GHz) Antenna Input 0xC3 — ISM (2.4 GHz to 2.48 GHz) External Input 0xC4 — Wideband (400 MHz to 3.8 GHz) Output
0E	TxPath	byte	Specifies the current RF path profile ID used to configure the RF0 transmitter functions. The default path profiles and IDs are defined as

			follows. See RF Path profiles for more information on customizing profiles and calibration data. Value / Description: 0x80 — TX Disabled 0x81 — ISM (2.4 GHz to 2.48 GHz) Antenna Output 0x82 — ISM (2.4 GHz to 2.48 GHz) External Output 0x83 — Wideband (400 MHz to 3.8 GHz) Output
02	RxGain	byte	Specifies the gain setting for the receiver. Valid values are between 0 and 20 equal to 0 to 60 dB gain in 3dB steps. Gain = val * 3; // in dBs. Lime recommends only using up to 30 dB gain for normal use.
03	RxFrequency	uint32	Specifies the frequency in KHz. This is a coarse frequency specification. Use NcoCount property to specify frequency more precisely.
~~04~~	~~RxNcoCount~~	~~uint32~~	~~Sets the Lime NCO counter to the specified value. This is the preferred method as it is more accurate than specifying the frequency.~~
05	RxBandwidth	byte	Specifies bandwidth of the receiver. Legal values are between 0 and 15. **These are "half bandwidth" values given I/Q processing.**
06	TxGain	byte	Property specifies the AGC gain setting for the transmitter. Valid values are between 0 and 56 equal to 0 to 56 dB gain in 1dB steps. Gain = val in dBs.
07	TxFrequency	uint32	Specifies the frequency in KHz. This is a coarse frequency specification. Use NcoCount property to specify frequency more precisely.
~~08~~	~~TxNcoCount~~	~~uint32~~	~~Sets the Lime NCO counter to the specified value. This is the preferred method as it is more accurate than specifying the frequency.~~
09	TxBandwidth	byte	Specifies bandwidth of the transmitter. Legal values are between 0 and 15. **These are "half bandwidth" values given I/Q processing.**
0A	RxRssi	uint16	Read only register of I and Q received signal strength indication values I and Q RSSI values truncated to the 8 high order bits. RSSI is the average absolute value of the received data over the sampled interval. 24 bit counter is used so sampling interval should be around (~0.3 msec). Register contains both the I and Q RSSI values as follows: bit# / Description: 0-7 — In phase RSSI values (0 - 256) low 4 bits are not returned since it assumed gain will always be turned up enough to hit

RxBandwidth (05):

Value	Bandwidth (MHz)	Value	Bandwidth (MHz)
0	14 (default)	8	2.75
1	10	9	2.5
2	7	10	1.92
3	6	11	1.5
4	5	12	1.375
5	4.375	13	1.25
6	3.5	14	0.875
7	3	15	0.75

TxBandwidth (09):

Value	Bandwidth (MHz)	Value	Bandwidth (MHz)
0	14 (default)	8	2.75
1	10	9	2.5
2	7	10	1.92
3	6	11	1.5
4	5	12	1.375
5	4.375	13	1.25
6	3.5	14	0.875
7	3	15	0.75

			at least 4 bits of the system. 8-15 Quadrature phase RSSI values 0 - 256) low 4 bits are not returned since it assumed gain will always be turned up enough to hit at least 4 bits of the the system.
0B	RxBias	uint16	Read only In-phase and Quadrature ADC Bias register. Implements 24 bit integrator to compute the average bias value over 4096 samples returns the high 12 bits for each I and Q stream indicating the average bias value over the period. Further LPF in the microcontroller may be required to get longer term average. The values stored in the 32 bit register as follows: bit# Description 0-15 The in-phase RSSI 12 bit value represented as a uint16 value (little endian). 16-31 The quadrature phase RSSI 12 bit value represented as a uint16 value (little endian).

The following table identifies the Execute Action messages supported by RF1.

Table 14 – RF1 Execute Action Messages

Id Action	Length Data	Data Specification
0x80	0	Tune the Lime Micro Receiver VCO. No arguments are specified.
0x81	0	Tune the Lime Micro Transmitter VCO. No arguments are specified.
0x82	0	Lime Micro Receiver I/Q Calibration. No arguments are specified.
0x83	0	Lime Micro Transmitter I/Q Calibration. No arguments are specified.
0x84	0	Top Level Calibration. Calibraties top level DC bias. RF Profiles must be in cached mode to actually save the data.
0x85	0	Updates the stored RX profile with DC calibration data. RF Profiles must be in cached mode to actually save the data.
0x86	0	Updates the stored TX profile with DC calibration data. RF Profiles must be in cached mode to actually save the data.
0x87	0	Resets the top level calibration.
0x88	0	Resets the RX calibration values.
0x89	0	Resets the TX calibration values.

The following table shows the supported typeids for sending and received TypedData messages (20,13). For more information please see Section 7.

Table 15 – RF1 Typed Data Messages

typeid	Description

0x20	Lime Micro Status Information
0x21	Lime Micro Register Read / Write
~~0x22~~	~~Lime Micro Serial Data Pass Through~~

NOTE: Need to specify how to calculate the Frequency Count.

8.5 Motion Sensor (0x83)

The Motion Sensor interface consists of the combined components of Accelerometer, Gyroscope, and Compass.

Table 16 – Motion Sensor Typed Properties

ID (hex)	Name	Type	Description
			NONE CURRENTLY DEFINED

Table 17 – RF1 Execute Action Messages

Id Action	Length Data	Data Specification
0x0	0	Motion Reporting Configuration Command. See section 7.6 for details on configuring and received sensor messages. Supported reported sample rates are 5 to 200 Hz from the MPU. Higher frequencies are possible, but required modifications to the ASR-2300 firmware.

Motion Sensor sampled data reporting structure is shown below. See section 7 for details on using these structures with the TypedData message (20,13). A raw sample consists of gyroscope, acceleration, and quaternion data. A future version will contain raw compass data as well.

```
struct Motion_RawSample
{
short gyro[3];
short accel[3];
long quat[3];
unsigned long timecode;
short flags;
};

struct Motion_DataSet
{
   Short ctRecs
   Motion_RawSample[6]; //Max 6 raw samples per set.
}
```

The following sub-sections define the individual sub-components for the Motion-Sensor.

8.5.1 Accelerometer Sensor

TBD

8.5.2 Gyroscope Sensor

TBD

8.5.3 Compass Sensor

TBD

8.6 Pressure Sensor (0x87)

The digital pressure sensor consists of both the pressure and temperature components. This sensor has a maximum sample rate of 128 samples/second (~8ms sample period) in standard-mode. Temperature has a resolution of 0.1°C.

Table 18 – Pressure Sensor Action Messages

Id Action	Length Data	Data Specification
0x00	0	Report Sensor Data. Follows conventions defined in section 7.6.Range of allowed data periods: 8 to 65536 msec. When enabled, generates reports of pressure sensor sampled data

Pressure samples are defined as follows:

```
Struct {
   Long pressure; // Calibrated pressure in hPa
   Short temp; //Calibrated temperature in °C (Celsius).
};
```

A maximum of 32 samples per report are supported.

8.7 Flash Memory (0x88)

The RL-78 microcontroller has an 8MB flash attached to it for containing FPGA and other static data. This can be programmed through the Binary Image Transfer messages targeting component id = 0x88. Several actions are available to provide erase and checksum verification of loads.

8.7.1 Action Messages

The following table identifies the Execute Action messages supported by RF0.

Table 19 – Flash Action Messages

Id Action	Length Data	Data Specification
0x00	0	Erase FPGA Flash
0x01	0	Verify Flash Load. Returns Log_INFO message indicating the checksum verification of the data.

8.7.2 Binary Image Transfer

The flash memory component supports the Binary Image Transfer messages (21,03), (21,04), and (21,83). This allows the host to upload and download FPGA binary HDL images to/from the serial flash memory. Currently the flash memory only supports one FPGA binary image at a time. Data frame size will typically be 256 bytes. Total message sizes will not exceed the 320 byte limit on DCI message size for any of the interfaces. Maximum flash memory available is 1MB.

To upload an image, send a BIT (21, 03) to component 0x88, with a specified transfer id. Then send the sequential BIT frames (21,04) until all frames have been sent. It is a good idea to request acknowledgement for all BIT messages since the whole operation must be aborted if any one message fails to be processed.

8.8 FX-3 Components (0x8A, 0x8B)

Reserved, no specifications at this time.

8.9 Low Speed Data Ports (0x90, 0x91, 0x92, 0x93)

The low-speed data port components provide configuration and control of I/Q data input and output via the serial interface. Depending on the configuration of these ports, type data messages (20, 13) will be handled by these ports either as input or output. For more type data sampled data structure see section 7.4. Each port component supports the following properties

Table 20 – Low Speed Data Ports Typed Properties

ID (hex)	Name	Type	Description
00	PortControl	byte	Write only port control register specifies the operating state of the low speed data port. Bit # — Description 0-1 — Mode Select 0 =Disabled 1 = Single Shot; requires a 20,93 message for host or device to send data. 20, 93 Can be sent repeatedly for updated snapshots. 2 = Continuous data transfer; data is sent automatically. 2-7 — Reserved.
01	PortStatus	byte	Read only port status register Write only port control register specifies the operating state of the low speed data port. Bit # — Description 0-1 — Mode Status 0 = Disabled 1 = Single Shot 2 = Continuous 3 = Not supported 2 — Port Direction 0 = Input (transmit) 1 = Output (receive) 3-7 — Reserved.

Ports 0x90 and 0x92 will have Port Direction bit equal 0 in the PortStatus property (id=1); ports 0x91 and 0x93 will have Port Direction bit equal 1.

8.10 RF Profile Paths (0x94)

The ASR-2300 uses preset configuration data known as path profiles to configure the Lime chip, RF switches, and calibration data. Basic profile paths are provided; however, users can develop custom profiles optimized for their particular devices and applications. These paths are stored in non-volatile memory (NVM) and can be uploaded and downloaded. A cached profile mode is also provided so that configurations can be modified in-place and then saved back to NVM. Calibration settings unique to the device can be updated using cache mode.

Table 21 – RF Profile Paths Component Properties

ID (hex)	Name	Type	Description
00	ID	uint16	Read only property returns unique profile identifier.
01	Version	uint16	Read only property returns version information
02	Revision	uint16	Read only property returns current revision number

See typed data definitions for reading the profile path definitions.

The following table identifies the Execute Action messages supported by RF0.

Table 22 – Flash Action Messages

Id Action	Length Data	Data Specification
0x00	0	Enter cached profile mode. All profiles and calibration data are copied to RAM from NVM so it can be edited.
0x01	0	Discard cached changes. Exits cached mode without saving any changes. Device reverts to using the profile and calibration data stored in NVM.
0x02	0	Save cached changes. Exits cached mode and saves any changes to NVM. Changes stored in the NVM will be available in subsequent sessions.

8.11 FPGA WCA HAL (0xB0)

FPGA properties are written directly to the FPGA. This requires that the loaded firmware implement the standard WCA HAL for the ASR-2300. The FPGA registers are addressed by adding the component ID with the property ID. For example property 7 in the FPGA WCAPAL is FPGA register address 0xB7; similarly property 0x12 has the FPGA register address equal to 0xC2.

Table 23 – FPGA WCA HAL Typed Properties.

ID (hex)	Name	Type	Description
00			

9. Custom DCI Messages (37)

In addition to support of the WCA and standard DCI messages, the following device specific messages are defined. Message Category 37 is specific to ASR-2300 family of devices.

9.1 Message (37,XX)

Message description here.

9.1.1 C Structure

```
typedef struct
{
   /*fields defined here*/
   byte field1;
   byte field2;
}
```

9.1.2 Message Format

Index	0	1	2	3
Field	**37**	**XX**	**field1**	**field2**

Field	Description	Type	Value	Index
field1	Field 1 description	byte		2
field2	Field 2 description	byte		3

9.1.3 Example

Example information here.

10. ASR-2300 System Conceptual Model

This section provides a high-level conceptual model for the functional components for the ASR 2300. The following diagram shows the constructs comprising both fixed WCA components and the programmable FPGA WCA component elements and the various data flow between the host USB drivers and hardware RF interface.

More details to follow…

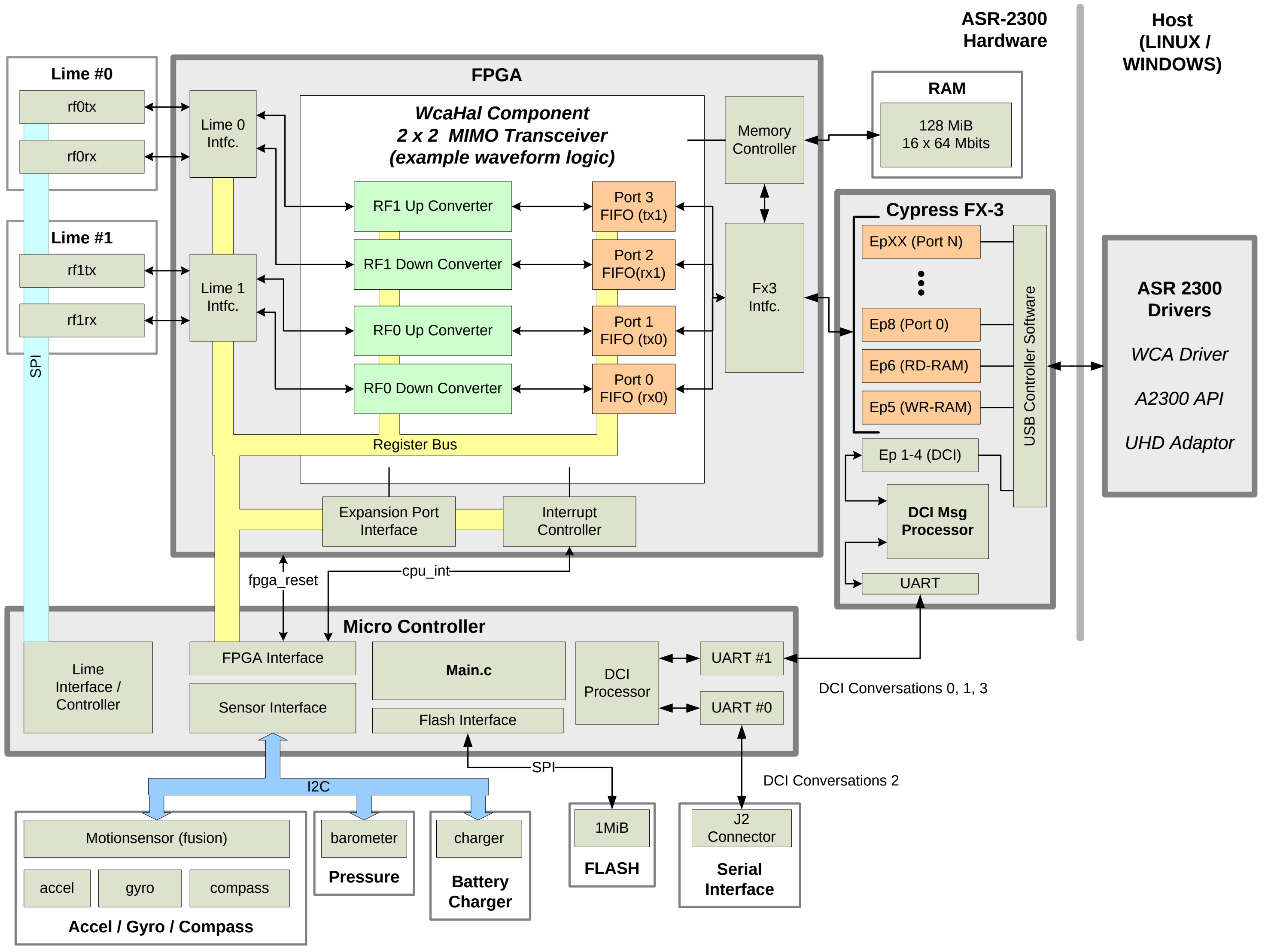

Figure 3 Detailed ASR-2300 System Blocks Conceptual Model

11. Waveform Components (FPGA Logic Images)

11.1 2 CH Receiver

TODO

11.2 2 x 2 MIMO Transceiver

TODO

11.3 LSDP Direct Connect Unit Test (ID=h8801)

Logic file contains WCA HAL Compliant component implementing a direct connection of low speed data port FIFOS with the ADC / DAC streams of both Lime #1 and Lime #2. This is a unit test, enabling direct measurements of ADC / DAC values without intermediate processing. Output Sampling rate is configurable.

In addition to the standard registers for the ASR-2300 WCA HAL. This component implements 4 FIFOs to transmit and receive data using the low speed data port (LSDP) functionality. FIFOs are directly connected to the ADC / DAC streams and can be sampled at a configurable strobe rate. FIFO almost full and empty events are mapped to the WCAHAL_EVENT registers so the RL-78 can receive interrupt notifications when FIFOs need servicing. A FIFO status register (WCA_HAL_LSDP_STATUS) is provided to read the current status of the FIFOS.

To configure the sampling strobe, set the WCAHAL_LSDP_RATE register with a value between 1 and 2^14. Note that the sampling clock rate is 25 MHz and the maximum through for all ports combined is 16.384 KHz. So if only 1 port is active, a divisor of 1526 would be required:

25e6/16.384e3 = 1525.88 (1526 rounded up). Recommended that maximum throughput is not pushed in current rev. Select divisors >= 3050 for data rates less than 8.192 KHz. When running all channels simultaneously select divisor >= 10000 for individual port rates sample rates of 2500 KHz. This will ensure that the Serial data I/O is not overrun.

Turning on the LSDP Ports requires enabling the Events in the WCAHAL_EVENT registers. Process the FIFO events to add or remove data from the FIFOs as requested.

See LsdpDirectConnectUT_Readme.txt for details on registers and additional documentation.

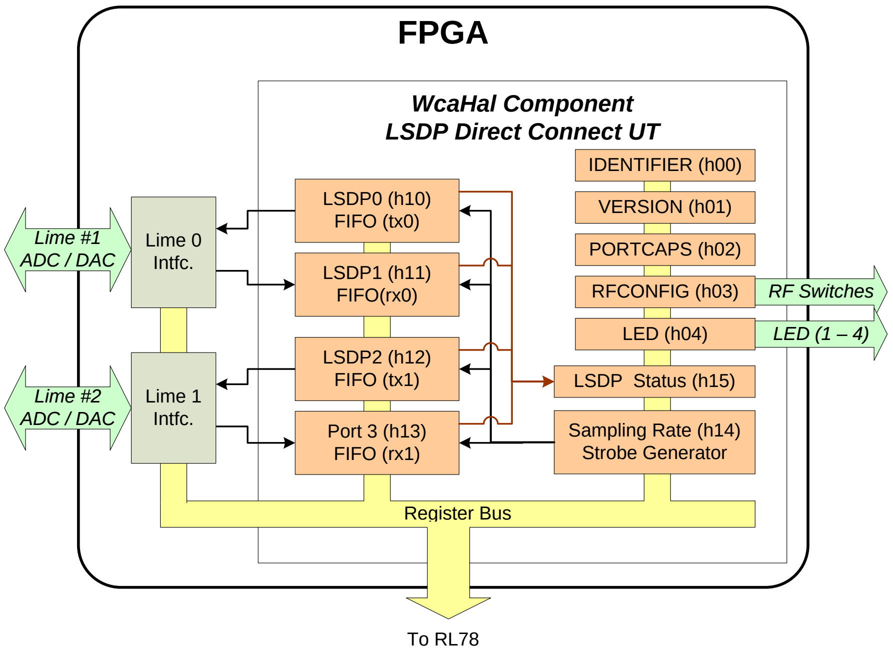

Figure 4 LSDP Direct Connect Unit Test HAL Component Block Diagram