	Reference: *VMLAB-UPM-TR2* **Date:** *15/10/2009* **Issue:** *1.1* **Page:** *1 of 26*

ESTEC/Contract No. 21392/08/NL/JK UART Device Driver for the ASSERT Virtual Machine

Output of WP 300

Written by:	**Organization**	**Date**
Ángel Esquina, Jorge López, Juan Zamorano	UPM	15/10/2009
Revised by:	**Organization**	**Date**
Juan A. de la Puente	UPM	15/10/2009
Accepted by:	**Organization**	**Date**
	ESTEC	

Document Change Record

Issue/Revision	**Date**	**Change**	**Author**
1.0	14/09/2009	First version for review	J. Zamorano, J. López
1.1	15/10/2009	Minor fixes	A. Esquina, J. Zamorano

Abstract

This document describes the UART driver for the ASSERT Virtual Machine which has been developed in the framework of the VM-LAB project. The driver uses the serial communications device included with the GR-RASTA LEON2 computer board, and has been developed according to the guidelines provided in the companion document *Guidelines for integrating device drivers in the ASSERT Virtual Machine*, and is integrated with the GNATforLEON compilation system.

Contents

1 Introduction **7**
- 1.1 Purpose . . . 7
- 1.2 Scope . . . 7
- 1.3 Glossary . . . 7
  - 1.3.1 Acronyms and abbreviations . . . 7
- 1.4 Applicable and reference documents . . . 8
  - 1.4.1 Applicable documents . . . 8
  - 1.4.2 Reference documents . . . 8
  - 1.4.3 Standards . . . 8
  - 1.4.4 Other documents . . . 9
- 1.5 Overview . . . 9

2 UART driver **11**
- 2.1 GRUART hardware core . . . 11
  - 2.1.1 Serial interface . . . 11
    - Transmitter . . . 12
    - Receiver . . . 13
    - Baud-rate generation . . . 13
  - 2.1.2 AMBA interface . . . 13
- 2.2 Driver . . . 13
  - 2.2.1 Data Buffers . . . 13
  - 2.2.2 Driver architecture . . . 14
  - 2.2.3 Driver components . . . 14
- 2.3 Source code . . . 17
  - 2.3.1 Uart . . . 17
    - Uart.Parameters . . . 18
    - Uart.HLInterface . . . 18
    - Uart.Registers . . . 20
    - Uart.Core . . . 20

3 Conclusions **23**

Bibliography **25**

Chapter 1

Introduction

1.1 Purpose

This document describes the architecture and detailed design of an UART driver for the ASSERT Virtual Machine. The driver uses the serial link provided with the GR-RASTA computer board,[1] and is integrated with the GNATforLEON compilation system.

1.2 Scope

The target audience for this document are software engineers who need to use the UART device of the GR-RASTA system with the ASSERT Virtual Machine.

1.3 Glossary

1.3.1 Acronyms and abbreviations

AHB	Advanced High-performance Bus
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
API	Application Programming Interface
ASB	Advanced System Bus
ASSERT	Automated proof-based System and Software Engineering for Real-Time applications
cPCI	Compact Peripheral Component Interconnect
CPU	Central Processing Unit
CTSN	Clear To Send Negated
DMA	Direct Memory Access
ECSS	European Cooperation on Space Standardization
FIFO	First-In First-Out

[1] GR-RASTA is a modular system based on a LEON2 or LEON3 computer, using a cPCI (compact Peripheral Component Interconnect) backplane bus. See `http://www.gaisler.com/doc/gr-rasta_product_sheet.pdf` for detailed information.

FPGA	Field Programmable Gate Array
GR	Gaisler Research
GRUART	Gaisler Reserarch UART
HI	High Integrity
I/O	Input/Output
ORK	Open Ravenscar real-time Kernel
PCI	Peripheral Component Interconnect
RE	Receiver Enable
RASTA	Reference Avionics System Testbench Activity
RMAP	Remote Memory Access Protocol
RTSN	Request To Send Negated
TE	Transmit Enable
UART	Universal Asynchronous Receiver-Transmitter
VM	Virtual Machine

1.4 Applicable and reference documents

1.4.1 Applicable documents

[A1] *Lab activities — Improvement and documentation of the ASSERT Virtual Machine.* ESTEC Statement of Work TEC-SWE/07-104/MP, I1R3. 3 September, 2007.

[A2] *Improvement and Documentation of the ASSERT Virtual Machine — Proposal for ESA Statement of Work Ref: TEC-SWE/07-104/MP.* University of Padova, École Nationale Supérieure des Télécommunications, Universidad Politécnica de Madrid. I1R4. 7 January 2008.

1.4.2 Reference documents

[R1] *VMLAB-UPM-TR1. Guidelines for integrating device drivers in the ASSERT Virtual Machine.* I1R5. September 2009.

[R2] *ASSERT D3.3.2-2: Virtual Machine Architecture Definition.* I1R1, July 2007.

[R3] *ASSERT D3.3.2-3: Virtual Machine Components Specification.* I1R1, July 2007.

[R4] *GNATforLEON/ORK+ User Manual.* Version 1.1. 18 November, 2008. Available at `http://www.dit.upm.es/ork`.

[R5] *PolyORB-HI User's Guide.* Available at `http://aadl.enst.fr`.

1.4.3 Standards

[S1] *ECSS-E-ST-40C — Space engineering — Software.* March 2009.

[S2] *ECSS-E-50-11 Draft F. Remote Memory Access Protocol (RMAP).* December 2006

[S3] ISO SC22/WG9. *Ada Reference Manual. Language and Standard Libraries. Consolidated Standard ISO/IEC 8652:1995(E) with Technical Corrigendum 1 and Amendment 1*, 2005.

[S4] ISO SC22/WG14. *Programming Languages — C*. ISO/IEC 9899:1999.

[S5] ISO SC22/WG15. *Portable Operating System Interface (POSIX)*. ISO/IEC 9945-2003.

1.4.4 Other documents

[D1] ISO/IEC. *Guide for the use of the Ada programming language in high integrity systems.* Technical report ISO/IEC TR 15942:2000.

[D2] ISO/IEC. *Guide for the use of the Ada Ravenscar Profile in high integrity systems.* Technical report ISO/IEC TR 24718:2005. Based on the University of York Technical Report YCS-2003-348 (2003).

[D3] Christine Ausnit-Hood, Kent A. Johnson, Robert G. Petit IV and Steven B. Opdahl, *Ada 95 Quality and Style*. Springer-Verlag, LNCS 1344. 1995.

[D4] AdaCore. *GNAT GPL User's Guide*. 2007.

[D5] AdaCore. *GNAT Reference Manual*. 2007.

[D6] *The SPARC architecture manual*: Version 8. Revision SAV080SI9308, 1992.

[D7] ATMEL. *Rad-Hard 32 bit SPARC V8 Processor AT697E*. Rev. 4226E–AERO–09/06. With errata sheet Rev. 4409C–AERO–05/08.

[D8] *GR-CPCI-AT697 Development Board User Manual*. Version 1.1, June 2005. Gaisler Research/Pender Electronic Design, 2005.

[D9] *GR-RASTA Board User Manual*. Gaisler Research/Pender Electronic Design, 2007.

[D10] *RASTA Interface Board FPGA User's Manual*. Version 1.0.0, June 2006. Gaisler Research, 2006.

[D11] *GRLIB IP Core User's Manual*. Version 1.0.16, June 2007. Jiri Gaisler, Edvin Catovic, Marko Isomäki, Kritoffer Glembo, Sandi Habinc. Gaisler Research, 2007.

[D12] ARM. *AMBA(TM) Specification* (Rev 2.0). ARM Limited 1999.

1.5 Overview

The rest of this document is organised as follows:

- Chapter 2 describes the architecture and design details of the driver.
- Chapter 3 concludes the report.

Chapter 2

UART driver

A device driver for the GR-RASTA APB UART device is described in this chapter. The driver has been developed in accordance with the companion document *VMLAB-UPM-TR1. Guidelines for integrating device drivers in the ASSERT Virtual Machine* [R1], to which the reader is referred for a general view of the architecture of device drivers in the ASSERT VM.

GR-RASTA is a development and evaluation platform for LEON2 and LEON3 based spacecraft avionics. Processing is provided by the Atmel AT697 LEON2-FT device. The AMBA APB UART serial interface is provided on a separate FPGA I/O board. Communication between the boards is done via the Compact PCI (cPCI) bus as described in [R1].

2.1 GRUART hardware core

The GRUART core provides an interface between an APB bus and a UART port. It is configured, and data is transferred, by means of a set of registers accessed through an APB interface (figure 2.1).

The GRUART core can be split into two main parts:

- The **serial interface**, which consists of the receiver and transmit shift registers, and the hold registers.
- The **AMBA interface**, which consists of the APB interface.

2.1.1 Serial interface

This interface provides the functionality for asynchronous serial communications. In uses an UART supporting data frames with 8 data bits, one optional parity bit, and one stop bit. In order to generate the appropriate bit-rate, each UART has a programmable 12-bit clock divider. Two FIFOs (if not available then two holding registers) are used for data transfer between the APB bus and the UART. Hardware flow-control is supported through the RTSN/CTSN handshake signals.

The serial port uses interrupts for synchronising its operation with the CPU. When a data item is received, it is stored into a receiver FIFO by the UART device, and an interrupt is raised. The interrupt handler can then take a data item from the FIFO. For transmission, data items are written into a transmitter FIFO. Data is transmitted as soon as it is available from the FIFO. When the transmission

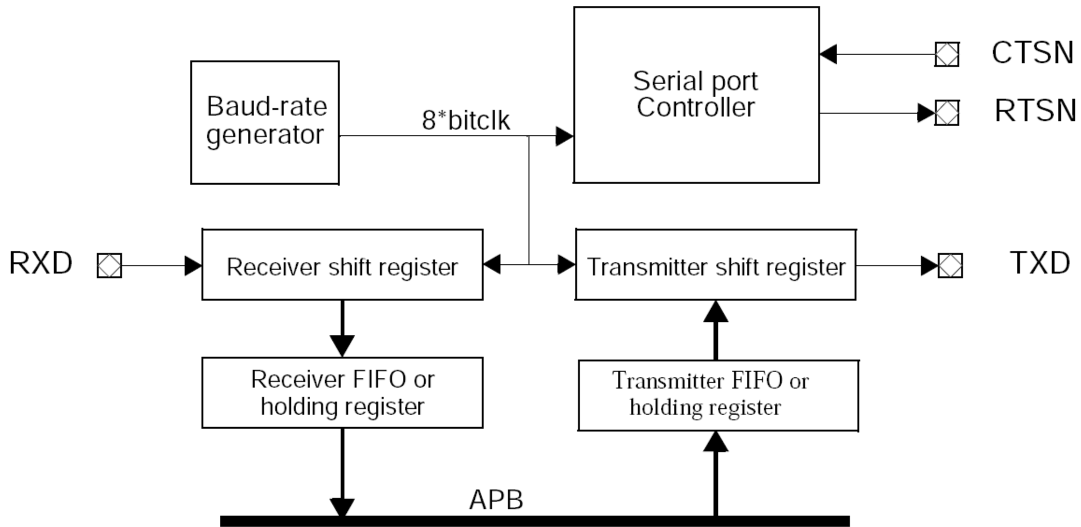

Figure 2.1: UART Core. (reproduced from [D11]).

of a data item has been completed, an interrupt is raised, signalling that the device is ready to transmit a new data item.

Transmitter

The transmitter is enabled through the TE bit in the UART control register. Data that is to be transferred is stored in the FIFO/holding register by writing to the data register. When ready to transmit, data is transferred from the transmitter FIFO/holding register to the transmitter shift register and converted to a serial bit stream which is output to the serial output pin (TXD). The transmitter automatically sends a start bit followed by eight data bits, an optional parity bit (see Figure 2.2), and one stop bit. The least significant bit of the data is sent first.

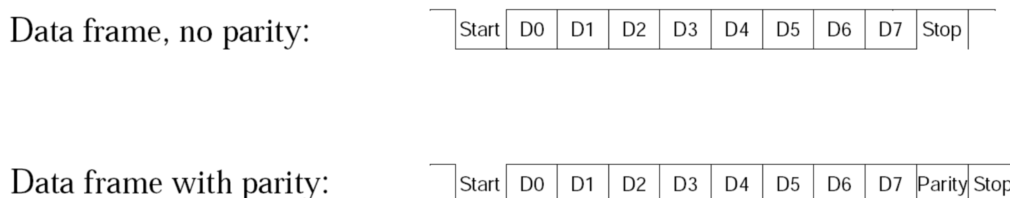

Figure 2.2: UART data frames. (reproduced from [D11]).

Receiver

The receiver is enabled for data reception through the `receiver enable` (RE) bit in the UART control register. The receiver awaits the high-to-low transition of the start bit on the receiver serial data input pin. If the transition is detected, the state of the serial input is sampled a half bit clocks later. If the serial input is sampled high then the start bit is considered invalid, and the wait for a valid start bit resumes. If the serial input is still low, a valid start bit is assumed and the receiver continues to sample the serial input at one bit time intervals.

During reception, the first bit is considered the least significant one. The data is then transferred to the receiver FIFO/holding register, and the `data ready` (DR) bit in the UART status register is set. The parity, framing and overrun error bits are set at the received byte boundary, at the same time as the `receiver ready` bit is set. The data frame is not stored in the FIFO/holding register if an error is detected.

Baud-rate generation

Each UART contains a 12-bit down-counting scaler to generate the desired baud-rate. The scaler is clocked by the system clock (on the I/O board), and generates a UART tick each time it underflows. It is reloaded with the value of the UART scaler reload register after each underflow. The resulting UART tick frequency should be 8 times the desired baud-rate.

2.1.2 AMBA interface

As described in [R1], LEON processors use the Advanced Microcontroller Bus Architecture (AMBA) bus hierarchy. It consists of an APB interface, an AHB master interface and DMA FIFOs.

However, UART Cores are not DMA capable and thus they only have the APB interface that provides access to the user registers.

2.2 Driver

2.2.1 Data Buffers

UART devices are character or byte oriented devices, i.e. an UART I/O operation involves just one character. Nevertheless, higher level software usually needs to send or receive a set of characters which builds up a message. Therefore, the driver includes two separate memory buffers for each functionality.

In this way each UART device has two associated buffers:

Transmit buffer: when higher level software sends a message, the corresponding data are pushed into this buffer and then the transmission starts until the buffer is empty. The interrupt service routine is in charge of transferring data from the buffer to the transmitter register.

Receive buffer: when a data item is received, the interrupt service routine transfers it from the receiver register to this buffer. In this way, higher level software can receive messages by getting the data from this buffer.

These intermediate buffers are stored in main memory, and their sizes can be specified with the Buffer_Size parameters (declared in Uart.Parameters). If the value of these parameters is changed the

driver needs to be recompiled. Figure 2.3 shows the data flow between the UART registers and the intermediate buffers.

2.2.2 Driver architecture

Figure 2.4 contains a diagram of the software architecture of the GRUART driver, which is an instance of the generic architecture described in the guidelines document [R1].

The driver has four main components:

- The PCI driver component, which provides data type definitions and operations for reading and writing the PCI configuration registers.
- The AMBA driver component, which provides data type definitions and operations for scanning the AMBA configuration records.
- The RastaBoard driver component, which provides a common interface for drivers using the GR-RASTA board, as well as hooks for interrupt handlers to be called upon reception of the single hardware interrupt issued by the board (see [R1]).
- The UART driver component, which provides all the software items required by application programs to initialize and use the UART cores included in the I/O Board GR-RASTA computer platform and the Data Buffers.

The functionality of the first three components, which are common to all device drivers, is described in the document *Guidelines for integrating device drivers in the ASSERT Virtual Machine* [R1]. The latter component is described in more detail in the rest of this section.

Section 2.3 contains a description of the main features of the implementation source code.

2.2.3 Driver components

The components of the UART driver are:

- HLInterface: contains the higher-level interface for application programs. The specification of this package is based on GNAT.Serial_Communications. The UART interface consists of :
  - Type definitions for Serial_Port , for receive and transmit data (which are instances of Streams.Stream_Element), and for port configuration.
  - Operations for initializing, setting, opening, and closing the UART devices, and for sending and receiving data.

    The Read operation can be blocking, i.e. the calling thread is suspended until a data packet is received. On the other hand, Write is a non-blocking operation.
- Parameters: contains the definitions of all the parameters that can be configured by the application programmer. The parameters are the sizes of the receive and transmit intermediate buffers, the number of UART core devices, and the frequency of the I/O board in MHz.
- Core: contains all the code that interacts with the device control registers in order to configure the core and communicate with the intermediate buffers.

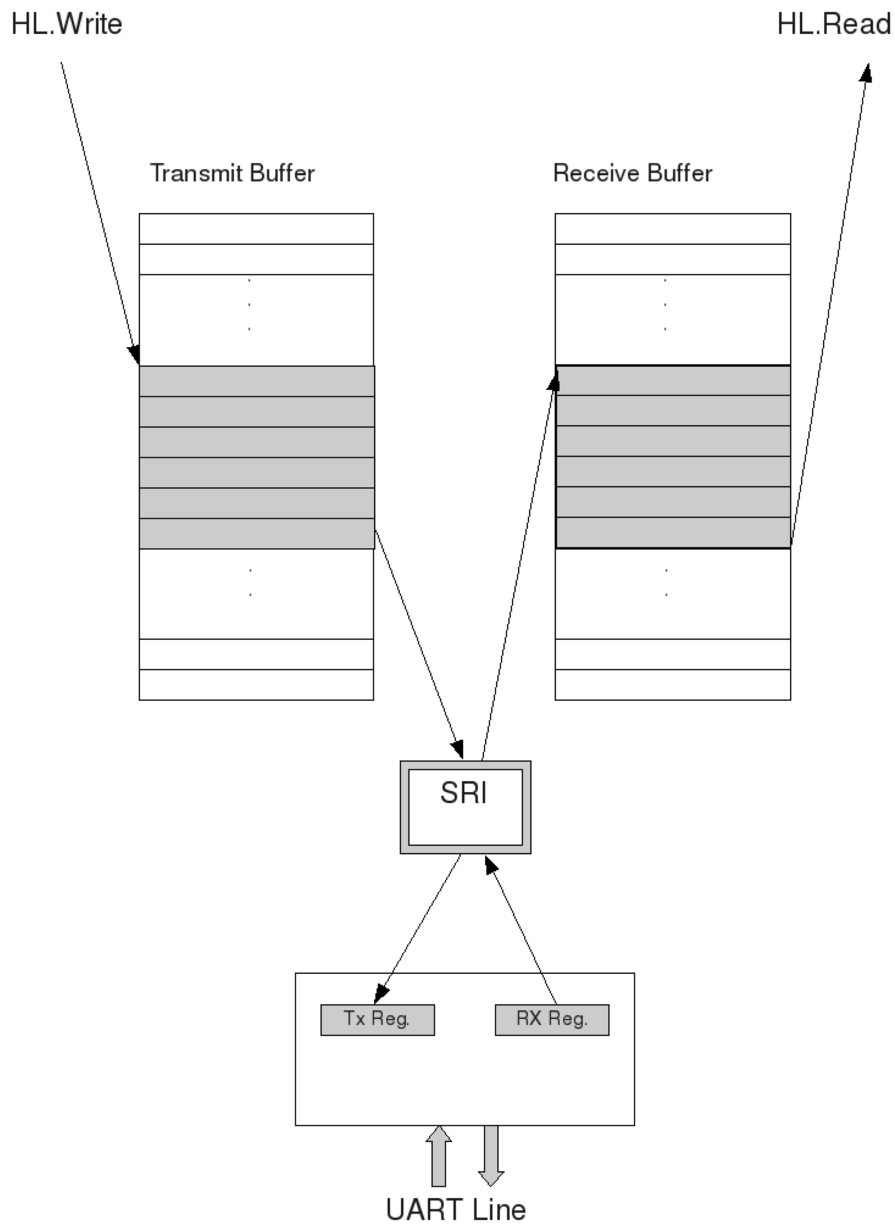

Figure 2.3: Buffer UART diagram.

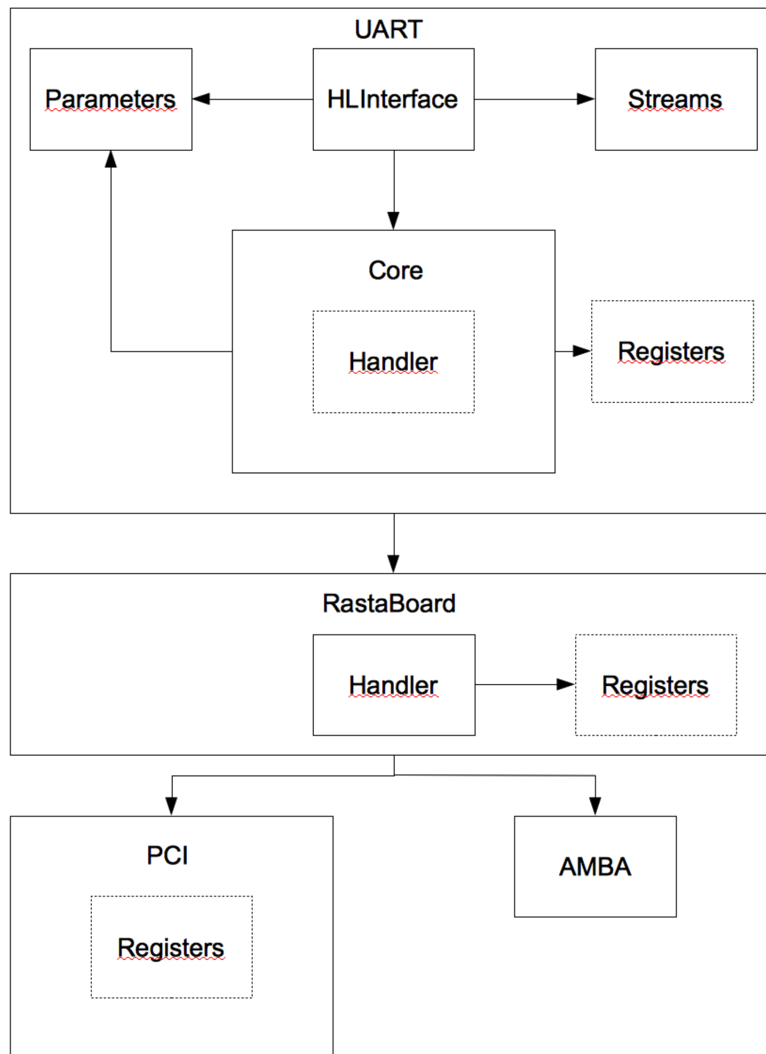

Figure 2.4: UART driver architecture

This component exports a set of interface operations, which are used to implement the HLInterface operations. The component implements all the device operations in terms of the device registers and other hardware characteristics.

- Registers: contains register and bit field definitions, as well as other data definitions that may be required to interact with the device.
- Handler: contains the device interrupt handler, which is invoked on the completion of I/O operations, intermediate buffers and operations involved the intermediate buffers. The buffers are protected objects.

  There is a single interrupt for all the two UART devices[1] and a synchronization object for each of the transmit and receive sections of each UART hardware device. Each occurrence of the interrupt is signalled to the appropriate synchronization object by identifying the device and function that has caused the interrupt.

2.3 Source code

The implementation source code of the UART driver is organized as a set of Ada packages. There is a specific root package, called Uart, from which the internal components of the driver are defined as package hierarchies. The rest of this section contains a description of the specification and implementation of every package. Only segments of code that are significant for the description are shown here. The reader is referred to the source files for the full details.

The UART driver code also uses the common packages RastaBoard, PCI, and AMBA, which are described in the *Guidelines for integrating device drivers in the ASSERT Virtual Machine* [R1].

2.3.1 Uart

The Uart package provides a root name for the Uart package hierarchy. It is declared as Pure, which means that the package can be *preelaborated*, i.e. its declaration is elaborated before any other library units in the same partition, and has no internal state. Notice that this package contains no further declarations and therefore has no state.

Listing 2.1: Package Uart

```
1  -- This is the root package of the GR-Uart driver implementation
2  pragma Restrictions (No_Elaboration_Code);
3  package Uart is
4     pragma Pure (Uart);
5  end Uart;
```

[1]The GR-RASTA interface board has two UART devices.

Uart.Parameters

This package contains the definitions of some parameters that can be configured by the application programmer. The first set of configurable parameters are the sizes of the receive and transmit buffers. Other parameters are directly related to the GR-RASTA hardware configuration, and should not be changed unless the hardware configuration is modified.

Listing 2.2: Package Uart.Parameters

```
-- This package defines basic parameters used by the GRUART driver.
-- This is the Rasta GR-CPCI-XC4V version of this package.

package Uart.Parameters is

   ------------------------------
   -- GR-CPCI-XC4V definitions --
   ------------------------------

   -- The following are GR-Rasta definitions.
   -- They must not be modified as long as a GR-Rasta board is used.

   Number_Of_UART_Cores : constant Integer := 2;
   -- Number of Uart Cores in the GR-Rasta GR-CPCI-XC4V System.


   -----------------------------------------
   -- IO Board Clock Frequency Constant --
   -----------------------------------------
   IO_Board_Clock_Frequency : constant Positive := 30;
   -- IO Board Clock frequency expressed ni MHz
   IO_Board_Clock_Freq_Hz : constant Positive :=
     IO_Board_Clock_Frequency * 10**6;
   -- IO Board Clock frequency expressed in Hz

   --------------------
   -- Buffers Size --
   --------------------
   Receive_Buffer_Size : constant Integer := 1024;
   -- Size of receive buffer

   Transmit_Buffer_Size : constant Integer := 1024;
   -- Size of transmit buffer
end Uart.Parameters;
```

Uart.HLInterface

This package defines the API of the UART driver. Its main elements are the procedures for initializing the SpaceWire configuration, Initialize , configuring, opening and closing the serial line, Set, Open and Close, and for transmitting and receiving data, Write and Read.

Listing 2.3: Package Uart.HLInterface

```
-- This is the High Level Interface of the GR-Uart driver implementation
-- This version of the package is for the GR-Rasta GR-CPCI-XC4V board

with UART.Streams;
with Uart.Core;
package Uart.HLInterface is

   Serial_Error : exception;
   -- Raised when a communication problem occurs

   type Data_Rate is
     (B1200, B2400, B4800, B9600, B19200, B38400, B57600, B115200);
   -- Speed of the communication

   type Data_Bits is (B8, B7);
   -- Communication bits

   type Stop_Bits_Number is (One, Two);
   -- One or two stop bits

   type Parity_Check is (None, Even, Odd);
   -- Either no parity check or an even or odd parity

   type Serial_Port is new Ada.Streams.Root_Stream_Type with private;

   procedure Initialize (Success : out Boolean);
   -- Find and set up all Uart devices in I/O Board
   -- Returns
   --    Success = true if devices were found and properly set up,
   --    Success = false otherwise.

   procedure Open
     (Port : out Serial_Port;
      Number : Uart.Core.UART_Device);
   -- Open the given port name. Raises Serial_Error if the port cannot be
   -- opened.

   procedure Set
     (Port     : Serial_Port;
      Rate     : Data_Rate      := B9600;
      Bits     : Data_Bits      := B8;
      Stop_Bits : Stop_Bits_Number := One;
      Parity   : Parity_Check   := None;
      Block    : Boolean        := True;
      Timeout  : Duration       := 10.0);
   -- The communication port settings. If Block is set then a read call will wait
   -- for the whole buffer to be filed. Timeout (in seconds) is not used.

```

```
49     procedure Read
50       (Port   : in out Serial_Port;
51        Buffer : out UART.Streams.Stream_Element_Array;
52        Last   : out UART.Streams.Stream_Element_Offset);
53     -- Receive Data of the Port. If the port is configured Block then
54     -- the procedure will be suspend until all the N-Characters (size of Data)
55     -- was been received. If the port is configured non Block then
56     -- Last indicate the last position in Data where character was stored.
57     -- Last is set to Buffer'First - 1 if no byte has been read.
58
59     procedure Write
60       (Port   : in out Serial_Port;
61        Buffer : UART.Streams.Stream_Element_Array);
62     -- Send all N-Characters in Data, Success is set to True if all the
63     -- Characters was send. If the Intermediate buffer is full, no all
64     -- the characters will send. Write is not blocking.
65
66     procedure Close (Port : in out Serial_Port);
67     -- Close port
68
69     ...
70
71  end Uart.HLInterface;
```

The operations defined in this package are implemented in the body of the package as direct calls to Core operations, except for Open and Close.

Uart.Registers

This is a private package that contains the definitions of all the data types that are needed to specify the UART device registers, including those that are used to interface with the AMBA bus, as well as the definition of the register structure.

The fields of the registers and the registers themselves are named as in the document *RASTA Interface Board FPGA User's Manual* [D.10]. See this document for the details.

Operations for reading and writing interrupt registers are also provided by this package.

Uart.Core

This package contains all the functionality required to operate the Uart devices.

Listing 2.4: Package Uart.Core

```
1  -- This version of the package is for the GR-RASTA Interface board
2  with Uart.Parameters;
3  with AMBA;
4  with Uart.Streams;
5  with Interfaces;
6  with System;
```

```
7
8  package Uart.Core is
9
10    type UART_Device is
11     range 1 .. Parameters.Number_Of_UART_Cores;
12     -- UART Cores in the GR-Rasta GR-CPCI-XC4V System
13
14    type UART_Core_Device is
15      record
16         Core                          : UART_Device;
17         -- Core ID
18         IRQ                           : AMBA.IRQ_Type := 0;
19         -- GRUART core interrupt routing information
20         Base_Address                  : AMBA.IO_BAR_Type := 0;
21         -- The start address of the GRUART core registers
22         Block : Boolean := False;
23         -- If Block is True then receive data waits until an information
24         -- arrive.
25         Free : Boolean := True;
26         -- Change to False when this device is opened.
27      end record;
28
29   type Parity_Check is (None, Even, Odd);
30   -- Either no parity check or an even or odd parity
31   subtype Byte is Interfaces.Unsigned_8;
32
33   type UART_Devices is
34     array (UART_Device) of UART_Core_Device;
35   -- Array that contains all the cores of UART
36   Devices : UART_Devices;
37
38   function Initialize return Boolean;
39   -- Initialize all UART devices in APB AMBA Bus.
40   -- Must call this before other functions or procedures.
41
42  procedure Set
43     (Port     : UART_Core_Device;
44      Rate     : Integer := 9600;
45      Parity   : Parity_Check   := None;
46      Flow    : Boolean := True);
47   -- Configure Uart device.
48
49   procedure Write (Port : UART_Core_Device;
50                    Data : Uart.Streams.Stream_Element_Array;
51                    Success : out Boolean);
52   -- Send all N-Characters in Data, Success is set to True if all the
53   -- Characters was send. If the Intermediate buffer is full, no all
54   -- the characters will send. Write is not blocking.
55
56
```

```
57    procedure Read (Port : UART_Core_Device;
58                    Data : out Uart.Streams.Stream_Element_Array;
59                    Last : out Uart.Streams.Stream_Element_Offset);
60    -- Receive Data of the Port. If the port is configured Block then
61    -- the procedure will be suspend until all the N-Characters (size of Data)
62    -- was been received. If the port is configured non Block then
63    -- Last indicate the last position in Data where character was stored.
64
65 end Uart.Core;
```

The Initialize operation takes care of all the initialization steps that are required to make the UART devices operational, so that data can be sent and received over the UART ports. An internal procedure Set, initializes the registers of a UART device.

The Set operation must be invoked after successful initialization and opening of a UART device in order to configure the parameters (baud rate, parity and flow-control) of each device.

The Write and Read operations perform the actual data transfers on UART devices. Write stores data into the intermediate buffer, and activate the transmission in case of Write. Read extract data from the receive intermediate buffer.

Chapter 3

Conclusions

A device driver for the GR-RASTA board has been described in this report. The driver is integrated with the ASSERT Virtual Machine and the GNATforLEON/ORK+ compilation system and real-time kernel.

Bibliography

[SA99] Tom Shanley and Don Anderson. *PCI System Architecture*. Mindshare Inc., fourth edition, 1999.

[Sta06] W. Stallings. *Computer Organization and Architecture*. Prentice Hall, seventh edition, 2006.

[TW87] Willis J. Tompkins and John G. Webster. *Interfacing Sensors to the IBM-PC*. Prentice Hall, 1987.

[vdG89] A.J. van de Goor. *Computer Architecture and Design*. Addison Wesley, 1989.

[Wil87] A. D. Wilcox. *68000 Microcomputer Systems: Designing and Troubleshooting*. Prentice-Hall International, Inc., first edition, 1987.