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

Output of WP 300

Written by: Angel Esquina, Jorge López, Juan Zamorano
Organization: UPM
Date: 15/10/2009

Revised by: Juan A. de la Puente
Organization: UPM
Date: 15/10/2009

Accepted by: ESTEC
Date: 15/10/2009
Document Change Record

<table>
<thead>
<tr>
<th>Issue/Revision</th>
<th>Date</th>
<th>Change</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>14/09/2009</td>
<td>First version for review</td>
<td>J. Zamorano, J. López</td>
</tr>
<tr>
<td>1.1</td>
<td>15/10/2009</td>
<td>Minor fixes</td>
<td>A. Esquina, J. Zamorano</td>
</tr>
</tbody>
</table>
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

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

¹GR-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.
1.4 Applicable and reference documents

1.4.1 Applicable documents


1.4.2 Reference documents


1.4.3 Standards


1.4.4 Other documents


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. It 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
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.

**Data frame, no parity:**

<table>
<thead>
<tr>
<th>Start</th>
<th>D0</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
<th>D4</th>
<th>D5</th>
<th>D6</th>
<th>D7</th>
<th>Stop</th>
</tr>
</thead>
</table>

**Data frame with parity:**

<table>
<thead>
<tr>
<th>Start</th>
<th>D0</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
<th>D4</th>
<th>D5</th>
<th>D6</th>
<th>D7</th>
<th>Parity</th>
<th>Stop</th>
</tr>
</thead>
</table>

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.

```ada
-- This is the root package of the GR-Uart driver implementation
pragma Restrictions (No_Elaboration_Code);
package Uart is
pragma Pure (Uart);
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

```vhdl
-- 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 in 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

1  -- This is the High Level Interface of the GR-Uart driver implementation
2  -- This version of the package is for the GR-Rasta GR-CPCI-XC4V board
3
4  with UART.Streams;
5  with Uart.Core;
6  package Uart.HLInterface is
7
8  Serial_Error : exception;
9  -- Raised when a communication problem occurs
10
11  type Data_Rate is
12    (B1200, B2400, B4800, B9600, B19200, B38400, B57600, B115200);
13  -- Speed of the communication
14
15  type Data_Bits is (B8, B7);
16  -- Communication bits
17
18  type Stop_Bits_Number is (One, Two);
19  -- One or two stop bits
20
21  type Parity_Check is (None, Even, Odd);
22  -- Either no parity check or an even or odd parity
23
24  type Serial_Port is new Ada.Streams.Root_Stream_Type with private;
25
26  procedure Initialize (Success : out Boolean);
27  -- Find and set up all Uart devices in I/O Board
28  -- Returns
29  -- Success = true if devices were found and properly set up,
30  -- Success = false otherwise.
31
32  procedure Open
33    (Port : out Serial_Port;
34     Number : Uart.Core.UART_Device);
35  -- Open the given port name. Raises Serial_Error if the port cannot be
36  -- opened.
37
38  procedure Set
39    (Port   : Serial_Port;
40     Rate   : Data_Rate := B9600;
41     Bits   : Data_Bits := B8;
42     Stop_Bits : Stop_Bits_Number := One;
43     Parity  : Parity_Check := None;
44     Block   : Boolean := True;
45     Timeout : Duration := 10.0);
46  -- The communication port settings. If Block is set then a read call will wait
47  -- for the whole buffer to be filled. Timeout (in seconds) is not used.
procedure Read
(Port : in out Serial_Port;
    Buffer : out UART.Streams.Stream_Element_Array;
    Last : out UART.Streams.Stream_Element_Offset);
-- Receive Data of the Port. If the port is configured Block then
-- the procedure will be suspend until all the N-Characters (size of Data)
-- was been received. If the port is configured non Block then
-- Last indicate the last position in Data where character was stored.
-- Last is set to Buffer'First - 1 if no byte has been read.

procedure Write
(Port : in out Serial_Port;
    Buffer : UART.Streams.Stream_Element_Array);
-- Send all N-Characters in Data, Success is set to True if all the
-- Characters was send. If the Intermediate buffer is full, no all
-- the characters will send. Write is not blocking.

procedure Close (Port : in out Serial_Port);
-- Close port
...
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

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

-- This version of the package is for the GR-RASTA Interface board
with Uart.Parameters;
with AMBA;
with Uart.Streams;
with Interfaces;
with System;
package Uart.Core is

  type UART_Device is
    range 1 .. Parameters.Number_Of_UART_Cores;
    -- UART Cores in the GR-Rasta GR-CPCI-XC4V System

  type UART_Core_Device is
    record
      Core : UART_Device;
      -- Core ID
      IRQ : AMBA.IRQ_Type := 0;
      -- GRUART core interrupt routing information
      Base_Address : AMBA.IO_BAR_Type := 0;
      -- The start address of the GRUART core registers
      Block : Boolean := False;
      -- If Block is True then receive data waits until an information
      -- arrive.
      Free : Boolean := True;
      -- Change to False when this device is opened.
    end record;

  type Parity_Check is (None, Even, Odd);
  -- Either no parity check or an even or odd parity
  subtype Byte is Interfaces.Unsigned_8;

  type UART_Devices is
    array (UART_Device) of UART_Core_Device;
    -- Array that contains all the cores of UART
  Devices : UART_Devices;

function Initialize return Boolean;
  -- Initialize all UART devices in APB AMBA Bus.
  -- Must call this before other functions or procedures.

procedure Set
  (Port : UART_Core_Device;
   Rate : Integer := 9600;
   Parity : Parity_Check := None;
   Flow : Boolean := True);
  -- Configure Uart device.

procedure Write (Port : UART_Core_Device;
                Data : Uart.Streams.Stream_Element_Array;
                Success : out Boolean);
  -- Send all N-Characters in Data, Success is set to True if all the
  -- Characters was send. If the Intermediate buffer is full, no all
  -- the characters will send. Write is not blocking.
procedure Read (Port : UART_Core_Device;
    Data : out Uart.Streams.Stream_Element_Array;
    Last : out Uart.Streams.Stream_Element_Offset);
-- Receive Data of the Port. If the port is configured Block then
-- the procedure will be suspend until all the N-Characters (size of Data)
-- was been received. If the port is configured non Block then
-- Last indicate the last position in Data where character was stored.
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


