UART PL011 Cycle Model

Version 9.1.0

User Guide

Non-Confidential

ARM®

ARM DUI 1064A (ID120216)

UART PL011 Cycle Model
User Guide

Release Information

The following changes have been made to this document.

Change History

Date	Issue	Confidentiality	Change
February 2017	A	Non-Confidential	Restamp release

Non-Confidential Proprietary Notice

In this document, where the term ARM is used to refer to the company it means "ARM or any of its subsidiaries as appropriate".

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Product Status

The information in this document is final, that is for a developed product.

Web Address

http://www.arm.com

Contents

Preface

A Cycle Model component is a library developed from ARM intellectual property (IP) that is generated through Cycle Model Studio™. The Cycle Model then can be used within a virtual platform tool, for example, SoC Designer.

About This Guide

This guide provides all the information needed to configure and use the Cycle Model in SoC Designer.

Audience

This guide is intended for experienced hardware and software developers who create components for use with SoC Designer. You should be familiar with the following products and technology:

- SoC Designer
- Hardware design verification
- Verilog or SystemVerilog programming language

Conventions

This guide uses the following conventions:

Convention	Description	Example
`courier`	Commands, functions, variables, routines, and code examples that are set apart from ordinary text.	`sparseMem_t SparseMemCreate-New();`
italic	New or unusual words or phrases appearing for the first time.	*Transactors* provide the entry and exit points for data ...
bold	Action that the user performs.	Click **Close** to close the dialog.
<text>	Values that you fill in, or that the system automatically supplies.	<platform>/ represents the name of various platforms.
[text]	Square brackets [ ] indicate optional text.	`$CARBON_HOME/bin/modelstudio [ <filename> ]`
[text1 | text2]	The vertical bar | indicates "OR," meaning that you can supply text1 or text 2.	`$CARBON_HOME/bin/modelstudio [<name>.symtab.db | <name>.ccfg ]`

Also note the following references:

- References to C code implicitly apply to C++ as well.
- File names ending in .cc, .cpp, or .cxx indicate a C++ source file.

Further reading

This section lists related publications. The following publications provide information that relate directly to SoC Designer:

- *SoC Designer Installation Guide*
- *SoC Designer User Guide*
- *SoC Designer Standard Component Library Reference Manual*

The following publications provide reference information about ARM® products:

- *AMBA 3 AHB-Lite Overview*
- *AMBA Specification (Rev 2.0)*
- *AMBA AHB Transaction Level Modeling Specification*
- *Architecture Reference Manual*

See http://infocenter.arm.com/help/index.jsp for access to ARM documentation.

The following publications provide additional information on simulation:

- IEEE 1666™ SystemC Language Reference Manual, (IEEE Standards Association)
- SPIRIT User Guide, Revision 1.2, SPIRIT Consortium.

Glossary

AMBA	*Advanced Microcontroller Bus Architecture*. The ARM open standard on-chip bus specification that describes a strategy for the interconnection and management of functional blocks that make up a System-on-Chip (SoC).
AHB	*Advanced High-performance Bus*. A bus protocol with a fixed pipeline between address/control and data phases. It only supports a subset of the functionality provided by the AMBA AXI protocol.
APB	*Advanced Peripheral Bus*. A simpler bus protocol than AXI and AHB. It is designed for use with ancillary or general-purpose peripherals such as timers, interrupt controllers, UARTs, and I/O ports.
AXI	*Advanced eXtensible Interface*. A bus protocol that is targeted at high performance, high clock frequency system designs and includes a number of features that make it very suitable for high speed sub-micron interconnect.
Cycle Model	A software object created by the Cycle Model Studio (or *Cycle Model Compiler*) from an RTL design. The Cycle Model contains a cycle- and register-accurate model of the hardware design.
Cycle Model Studio	Graphical tool for generating, validating, and executing hardware-accurate software models. It creates a Cycle Model, and it also takes a Cycle Model as input and generates a component that can be used in SoC Designer, Platform Architect, or Accellera SystemC for simulation.
CASI	*ESL API Simulation Interface*, is based on the SystemC communication library and manages the interconnection of components and communication between components.
CADI	*ESL API Debug Interface*, enables reading and writing memory and register values and also provides the interface to external debuggers.
CAPI	*ESL API Profiling Interface*, enables collecting historical data from a component and displaying the results in various formats.
Component	Building blocks used to create simulated systems. Components are connected together with unidirectional transaction-level or signal-level connections.
ESL	*Electronic System Level*. A type of design and verification methodology that models the behavior of an entire system using a high-level language such as C or C++.
HDL	*Hardware Description Language*. A language for formal description of electronic circuits, for example, Verilog.
RTL	*Register Transfer Level*. A high-level hardware description language (HDL) for defining digital circuits.
SoC Designer	High-performance, cycle accurate simulation framework which is targeted at System-on-a-Chip hardware and software debug as well as architectural exploration.
SystemC	SystemC is a single, unified design and verification language that enables verification at the system level, independent of any detailed hardware and software implementation, as well as enabling co-verification with RTL design.
Transactor	*Transaction adaptors*. You add transactors to your component to connect your component directly to transaction level interface ports for your particular platform.

Chapter 1

Using the Cycle Model in SoC Designer

This chapter describes the functionality of the Cycle Model component, and how to use it in SoC Designer. It contains the following sections:

1.1 PL011 UART Cycle Model Functionality

The PrimeCell UART is an AMBA compliant System-on-Chip peripheral. The UART is an AMBA slave module that connects to the Advanced Peripheral Bus (APB). This Cycle Model supports data transfer in both receive and transmit direction via a transaction port.

This section provides a summary of the functionality of the Cycle Model compared to that of the hardware, and the performance and accuracy of the Cycle Model. For details of the functionality of the hardware that the Cycle Model simulates, refer to the *ARM PrimeCell UART (PL011) Technical Reference Manual*.

- Fully Functional and Accurate Features
- Fully Functional and Approximate Features
- Hardware Features not Implemented
- Features Additional to the Hardware
- Telnet Requirement (Windows only)

1.1.1 Fully Functional and Accurate Features

The following features of the UART PL011 hardware are fully implemented in the UART PL011 Cycle Model:

- Compliance to the AMBA (Rev 2.0) Specification
- Separate 16x8 transmit and 16x12 receive First-In, First-Out memory buffers (FIFOs)
- Programmable FIFO disabling for 1-byte depth
- Independent masking of transmit FIFO, receive FIFO, receive timeout, and error condition interrupts (only for receive FIFO overrun error). Refer to the *PL011 Technical Reference Manual* for information on how interrupts are implemented. Receive and Transmit interrupts are implemented in the same manner.
- Identification registers that uniquely identify the PrimeCell UART
- Communication baud rate, integer, and fraction parts
- FIFO enable/disable
- FIFO trigger levels are selectable between 1/8, 1/4, 1/2, 3/4, and 7/8
- All internal registers with address map are supported

1.1.2 Fully Functional and Approximate Features

The following features of the UART PL011 hardware are implemented in the UART PL011 Cycle Model, but the exact behavior of the hardware implementation is not accurately reproduced because some approximations and optimizations have been made for simulation performance:

- Programmable hardware flow control is partially supported. nUARTRTS is implemented. nUARTCTS is not implemented, so hardware flow control for data being sent out the TXD port is not supported.
- The APB interface is modeled as APB transaction interface. For more information, refer to the *ARM AHB to APB Bridge Cycle Approximate Model Reference Manual*, which is provided in the AMBA2 package of the SoC Designer Cycle Model Library. In particular, this APB interface does not contain reset and clock ports. Instead, *reset* and *clock* ports are provided separately.
- UARTEINTR – UART error interrupt supports only overrun error interrupt.

1.1.3 Hardware Features not Implemented

The following features of the UART PL011 hardware are not implemented in the UART PL011 Cycle Model:

- Standard asynchronous communication bits (start, stop, and parity) are not supported
- False start bit detection is not supported
- Line break generation and detection are not supported
- Using different-frequency clocks for UARTCLK and clk-in is not supported
- The modem control functions CTS, DCD, DSR, RTS, DTR, and RI are not supported
- Programmable serial interface
  - data widths of 5, 6, 7, or 8 bits are not supported
  - even, odd, stick, or no-priority bit generation and diction are not supported
  - 1 or 2 stop bit generation is not supported
- IrDA SIR ENDEC block is not supported.
- Internal nominal clock frequency to generate low-power mode shorter bit duration is not supported.
- UARTMSINTR does not support changes on the RI, CTS, DCD, or DSR modem status lines.
- UARTEINTR – UART error interrupt does not support framing, parity, and break error interrupts.
- The following registers are not available to be read / written via debug transactions — for example, in the SoC Designer Registers window, or by accessing them directly from RealView Debugger:
  - Data register; UARTDR (bits 8:15)
  - Receive status/error clear register; UARTRSR/UARTECR (bits 0:2)
  - Flag register; UARTFR (bits 8, 0:3)

- IrDA low-power counter register; UARTILPR (all bits)
- Line control register; UARTLCR_H (bits 0:3, 5:7)
- Control register; UARTCR (bits 1, 2, 10:15)
- Interrupt mask set/clear register; UARTIMSC (bits 0:3, 7:9)
- Raw interrupt status register; UARTRIS (bits 0:3, 7:9)
- Masked interrupt status register; UARTMIS (bits 0:3, 7:9)
- Interrupt clear register; UARTICR (bits 0:3, 7:9)
- DMA control register; UARTDMACR (all bits)

The functionality of these registers, however, does exist and can be accessed by software running on the virtual platform.

1.1.4 Features Additional to the Hardware

The following features are implemented in the UART PL011 Cycle Model to enhance usability; they do not exist in the UART PL011 hardware:

1.1.4.1 Immediate UARTLCR register updates

According to the *ARM PrimeCell® UART (PL011) Technical Reference Manual*, the following registers form a single 30-bit wide register (UARTLCR):

- UARTLCR_H
- UARTIBRD
- UARTFBRD

UARTLCR is updated on a single write strobe generated by a UARTLCR_H write. So, to internally update the contents of UARTIBRD or UARTFBRD, a UARTLCR_H write must always be performed at the end.

The Cycle Model implements CADI access to these registers such that they do not require execution of a separate write to UARTLCR_H to internally update the contents of the UARTLCR register.

CADI access to the registers includes access via:

- the View Registers window in SoC Designer
- CADI API access directly to the register (CADIRegWrite)
- CADI API access via a memory mapped address (CADIMemWrite)

All of these accesses update the appropriate bits of the 30-bit UARTLCR register immediately.

In addition, a CADI write to UARTLCR_H also generates the "strobe generated by a UARTCR_H write" that is referred to in the TRM.

The Cycle Model implementation provides a convenient interface to the register view in that if you write to one of these registers, that value is immediately stored internally in the UARTLCR.

1.1.4.2 Rx/Tx FIFO-related pseudo registers display

You can view the Rx/Tx FIFO contents, size of FIFO, base pointer of FIFO and tip pointer of FIFO in the “Transmit FIFO” and “Receive FIFO” register groups of the Register Window. All of these registers are pseudo registers, for information only, and *should not* be modified. Sixteen FIFO content registers are displayed.

1.1.5 Telnet Requirement (Windows only)

On Windows, when running in interactive mode, the PL011 Cycle Model automatically invokes telnet. It looks for `telnet.exe` in the following directories:

- If the Cycle Model is compiled on a 64-bit platform:`c:\\windows\\system32\\telnet.exe`
- If the Cycle Model is compiled on a 32-bit platform:
  - Running on a Windows 64-bit machine — `c:\\windows\\syswow64\\telnet.exe`
  - Running on a non-Windows 64-bit machine — `c:\\windows\\system32\\telnet.exe`

If `telnet.exe` is not found, the Cycle Model issues the following error message:

```
Telnet does not exist at: <directory name>

You can try to connect a telnet client to port <port number>

NOTE: When running a 32 bit model in a x64 install of windows, the path
c:\\windows\\system32\\telnet.exe is redirected to C:\\win-
dows\\syswow64\\telnet.exe.

NOTE: Please make sure telnet exists there, if you meet these criteria.
```

You have the option of starting a telnet session manually, and connecting to the port number given in the error message. The command to start the Windows telnet client is:

```
> telnet.exe localhost <port number>
```

You may also start a telnet session using a client other than the Windows telnet client.

1.2 Adding and Configuring the SoC Designer Component

The following topics briefly describe how to use the component. See the *SoC Designer User Guide* for more information.

- SoC Designer Component Files
- Adding the Cycle Model to the Component Library
- Adding the Component to the SoC Designer Canvas

1.2.1 SoC Designer Component Files

The component files are the final output from the Cycle Model Studio compile and are the input to SoC Designer. There are two versions of the component; an optimized *release* version for normal operation, and a *debug* version.

On Linux the *debug* version of the component is compiled without optimizations and includes debug symbols for use with gdb. The *release* version is compiled without debug information and is optimized for performance.

On Windows the *debug* version of the component is compiled referencing the debug runtime libraries, so it can be linked with the debug version of SoC Designer. The *release* version is compiled referencing the release runtime library. Both release and debug versions generate debug symbols for use with the Visual C++ debugger on Windows.

The provided component files are listed below:

Table 1-1 SoC Designer Component Files

Platform	File	Description
Linux	maxlib.lib<*model_name*>.conf	SoC Designer configuration file
	lib<*component_name*>.mx.so	SoC Designer component runtime file
	lib<*component_name*>.mx_DBG.so	SoC Designer component debug file
Windows	maxlib.lib<*model_name*>.windows.conf	SoC Designer configuration file
	lib<*component_name*>.mx.dll	SoC Designer component runtime file
	lib<*component_name*>.mx_DBG.dll	SoC Designer component debug file

Additionally, this User Guide PDF file is provided with the component.

1.2.2 Adding the Cycle Model to the Component Library

The compiled Cycle Model component is provided as a configuration file (*.conf*). To make the component available in the Component Window in SoC Designer Canvas, perform the following steps:

1. Launch SoC Designer Canvas.
2. From the *File* menu, select **Preferences**.
3. Click on **Component Library** in the list on the left.
4. Under the *Additional Component Configuration Files* window, click **Add**.
5. Browse to the location where the SoC Designer Cycle Model is located and select the component configuration file:
   - `maxlib.lib<model_name>.conf` (for Linux)
   - `maxlib.lib<model_name>.windows.conf` (for Windows)
6. Click **OK**.
7. To save the preferences permanently, click the **OK & Save** button.

The component is now available from the SoC Designer *Component Window.*

1.2.3 Adding the Component to the SoC Designer Canvas

Locate the component in the *Component Window* and drag it out to the Canvas. The component's appearance may vary depending on your specific device configuration.

Additional ports are provided depending on the Cycle Model RTL configuration file, *default.conf*, used to create the Cycle Model.

1.3 Available Component ESL Ports

Table 1-2 describes the ESL ports that are exposed in SoC Designer. See the *ARM PrimeCell® UART (PL011) Technical Reference Manual* for more information.

Table 1-2 ESL Component Ports

ESL Port	Description	Direction	Type
PRESETn	Bus reset signal, active LOW.	Input	Reset controller
RXD	1-bit port for receiving data from signal master port. The data received via this port is pushed into Rx FIFO UART.	Input	Signal Slave
UARTCLK	Clock input for UART. ***Note:*** This clock input must be connected to the same clock that is used for clk-in. If clk-in is left unconnected then this clock input must also be unconnected.	Input	Clock Slave
UARTRXDMACLR	Receive DMA Clear	Input	Signal Slave
UARTTXDMACLR	Transmit DMA Clear	Input	Signal Slave
apb	Allows the user to program the memory-mapped registers. This interface is expected to be connected to an APB-compliant device. This interface is clocked by the clk-in port.	Input	Transaction Slave
nUARTRST	UART reset signal to UARTCLK clock domain, active LOW. The reset controller must use PRESETn to assert nUARTRST asynchronously but negate it synchronously with UARTCLK.	Input	Reset Transactor
clk-in	Input clock port. This component and the APB port are clocked at the frequency of the clock connected to this port. If the clk-in port is not connected, clock frequency is taken from *SoC Designer System Properties*. ***Note:*** This clock input must be connected to the same clock that is used for UARTCLK, or if UARTCLK is unconnected then clk-in must also be unconnected.	Input	Clock Slave
TXD	1-bit port for transmitting data to connected signal slave port. The data transmitted via this port is popped out of Tx FIFO of UART.	Output	Signal Master
UARTRXDMABREQ	Receive DMA Burst Request	Output	Signal Master
UARTRXDMASREQ	Receive DMA Single Request	Output	Signal Master
UARTTXDMABREQ	Transmit DMA Burst Request	Output	Signal Master
UARTTXDMASREQ	Transmit DMA Single Request	Output	Signal Master

Table 1-2 ESL Component Ports (continued)

ESL Port	Description	Direction	Type
intp	Provides interrupts to the interrupt controller. The value contains the interrupt number and extended value (interrupt low or high). You can specify unique interrupt numbers for each interrupt supported via parameters – see Table 1-3 on page 1-10 for more details.	Output	Interrupt Master
nSIROUT	Transmitted Serial Data Output, active LOW. In the idle state, this signal remains LOW (the marking state). When this signal is HIGH, an infrared light pulse is generated that represents a logic 0 (spacing state).	Output	Signal Master
nUARTDTR	Data Terminal Ready modem status output, active LOW. The reset value is 0.	Output	Signal Master
nUARTRTS	Request to Send modem status output, active LOW. The reset value is 0.	Output	Signal Master
nUARTOut1	Out1 modem status output, active LOW. The reset value is 0.	Output	Signal Master
nUARTOut2	Out2 modem status output, active LOW. The reset value is 0.	Output	Signal Master

All pins that are not listed in this table have been either tied or disconnected for performance reasons.

1.4 Setting Component Parameters

You can change the settings of all the component parameters in SoC Designer Canvas, and of some of the parameters in SoC Designer Simulator. To modify the component's parameters:

1. In the Canvas, right-click on the component and select **Edit Parameters...**. You can also double-click the component. The *Edit Parameters* dialog box appears.
2. In the *Parameters* window, double-click the **Value** field of the parameter that you want to modify.
3. If it is a text field, type a new value in the *Value* field. If a menu choice is offered, select the desired option. The parameters are described in Table 1-3.

Table 1-3 Component Parameters

Name	Description	Allowed Values	Default Value	Runtime[1]
Align Waveforms	When set to *true*, waveforms dumped from the component are aligned with the SoC Designer simulation time. The reset sequence, however, is not included in the dumped data. When set to *false*, the reset sequence is dumped to the waveform data, however, the component time is not aligned with the SoC Designer time.	true, false	true	No
apb Base Address	Base address for APB region accessed via APB slave port of UART.	0x00000000 - 0xFFFFFFFF	0x0	No
apb Enable Debug Messages	Enable or disable debug messages.	true, false	false	Yes
apb Size	Address range size.	0x00000000 - 0xFFFFFFFF	0x10000000	No
Carbon DB Path	Sets the directory path to the database file.	Not Used	empty	No
Clone Output to Console?	The output of TXD port or file or Socket is also displayed on SoC Designer console window.	true, false	false	No
Dump Waveforms	Whether SoC Designer dumps waveforms for this component.	true, false	false	Yes
Enable Debug Messages	Enable or disable debug messages.	true, false	false	Yes
intp int_1 id	The Intr value. It is the combined interrupt of Rx_int, Tx_int, Receive timeout interrupt, and Overflow error interrupt. Always comes on "Intp" signal master port after the individual interrupt has come. Both come in same simulation cycle. Associated with UARTINTR.	Integer within a range.[2] -1 indicates interrupt is disabled.	1	Yes

Table 1-3 Component Parameters (continued)

Name	Description	Allowed Values	Default Value	Runtime[1]
intp int_2 id	The Rx_int number. Rx FIFO tide interrupt number. Associated with UARTRXINTR.	Integer within a range.[2] -1 indicates interrupt is disabled.	2	Yes
intp int_3 id	The Tx_int number. Tx FIFO tide interrupt number. Associated with UARTTXINTR.	Integer within a range.[2] -1 indicates interrupt is disabled.	3	Yes
intp int_4 id	The Receive Timeout Interrupt Number. Interrupt number for receive timeout interrupt. Associated with UARTRTINTR.	Integer within a range.[2] -1 indicates interrupt is disabled.	4	Yes
intp int_5 id	The Overflow Error Interrupt Number. Interrupt number for overflow error – which could occur on Rx FIFO. Associated with UARTEINTR.	Integer within a range.[2] -1 indicates interrupt is disabled.	5	Yes
intp int_6 id	Interrupt controller UART modem status interrupt. Associated with UARTMSINTR.	Integer within a range.[2] -1 indicates interrupt is disabled.	6	Yes
IPFileName	Input file name with path to be used for receiving data. This file name takes effect only when Pseudo I/O? is *true* and UseSockets? is *false* (see Table 1-4).	Any valid file name present in host OS with path absolute path	INPUTFILE	No
OPFileName	Output file name with path which will be used to transmit data to. This file name comes into effect only when Pseudo I/O? is *true* and UseSockets is *false*.	Any valid file name present in host OS with path absolute path	OUTPUTFILE	No
Pseudo I/O?[3]	Used to specify that RXD/TXD ports will not be used for receive or transmit. Instead either File IO for Socket IO will be used.	true, false	true	No
Raw Socket Data?[3]	When set to *true*, Telnet terminal will not automatically open. Instead the user needs to create a server or a client to send data to UART or receive data from UART.	true, false	false	No

Table 1-3 Component Parameters (continued)

Name	Description	Allowed Values	Default Value	Runtime[1]
TCP Socket Number[4]	TCP port number to use for Socket IO. **Note:** When using multiple instances of the PL011 Cycle Model, each model's TCP Port Number must be unique. Duplicate TCP port numbers result in data contention and errors such as "Couldn't initialize socket."	0x400 - 0xFFFF	0	No
Use External Loopback?	Used to loop back data from transmit to receive of the same UART. File, Socket, or RXD/TXD ports are not used when set to *true*.	true, false	false	No
Use Sockets?[3]	When both Use Sockets? = *true* and Pseudo I/O? = *true* — Socket IO is used for Rx/Tx. When *false* — File IO is used for Rx/Tx.	true, false	false	No
Waveform File [5]	Name of the waveform file.	*string*	arm_cm_pl011.vcd	No
Waveform Format	The format of the waveform dump file.	VCD, FSDB	VCD	No
Waveform Timescale	Sets the timescale to be used in the waveform.	Many values in drop-down	1 ns	No

1. *Yes* means the parameter can be dynamically changed during simulation, *No* means it can be changed only when building the system, *Reset* means it can be changed during simulation, but its new value will be taken into account only at the next reset.
2. Range depends on the number of interrupts of the destination component.
3. See Table 1-4 for information about how this parameter interacts with others.
4. At its default setting (0), the Cycle Model automatically selects an available TCP port.
5. When enabled, SoC Designer writes accumulated waveforms to the waveform file in the following situations: when the waveform buffer fills, when validation is paused and when validation finishes, and at the end of each validation run.

Table 1-4 describes the interaction between the following parameters and the behavior that result from their settings:

- Pseudo I/O?
- Use Socket?
- Raw Socket Data

Gray table cells indicate that the setting is irrelevant because it is superceded by another parameter setting; for example, the setting for the Pseudo I/O? parameter takes precedence over the setting for the Use Socket? parameter.

Table 1-4 Parameter Interaction and Resulting Behaviors

Pseudo I/O? setting	*Use Socket?* setting	*Raw Socket Data* setting	Behavior
true			TX/RX ports are not used.
false			TX/RX ports are used.
true	true		Socket I/O is used.
true	false		File I/O is used.
true	true	true	Telnet terminal not automatically launched; you must launch a socket client (e.g. telnet) by hand and connect it to a port that the PL011 Cycle Model is listening to.
true	true	false	*Linux*: Telnet terminal automatically launched. *Windows*: Telnet terminal not automatically launched; you must launch a socket client (e.g. PuTTY or Tera Term) by hand and connect it to a port that the PL011 Cycle Model is listening to.

1.5 Debug Features

The UART PL011 Cycle Model has a debug interface (CADI) that you can use to view, manipulate, and control the registers of UART.

1.5.1 Register Information

This section lists the register views available for the UART PL011 Cycle Model in the SoC Designer Simulator. The available groups are:

- General Registers
- Peripheral ID Registers
- Primecell ID Registers
- Transmit FIFO Registers
- Receive FIFO Registers

See the *ARM PrimeCell UART (PL011) Technical Reference Manual* for detailed descriptions of these registers.

Note: The contents of the "Transmit FIFO" and "Receive FIFO" Register tabs are pseudo registers and are for information only - they should not be modified.

1.5.1.1 General Registers

Table 1-5 shows the General registers.

Table 1-5 General Registers Summary

Base Offset	Name	Description	Type	Reset Value
0x00	UARTDR	Data Register	RW	0x0000
0x04	UARTRSR_ UARTECR	Receive Status Register/Error Clear Register	RW	0x00000000
0x18	UARTFR	Flag Register	RO	0x197
0x20	UARTILPR	IrDA Low-Power Counter Register	RW	0x00
0x24	UARTIBRD	Integer Baud Rate Register	RW	0x00000000
0x28	UARTFBRD	Fractional Baud Rate Register	RW	0x00
0x2C	UARTLCR_H	Line Control Register	RW	0x00
0x30	UARTCR[1]	Control Register	RW	0x0300
0x34	UARTIFLS	Interrupt FIFO Level Select Register	RW	0x12
0x38	UARTIMSC	Interrupt Mask Set/Clear Register	RW	0x000
0x3C	UARTRIS	Raw Interrupt Status Register	RO	0x000
0x40	UARTMIS	Masked Interrupt Status Register	RO	0x000
0x44	UARTICR	Interrupt Clear Register	WO	0x000
0x48	UARTDMACR	DMA Control Register	RW	0x0

1. If you set the UART control register to *Receive Enable*, and the Pseudo I/O? parameter is set to *True*, the UARTCR register setting does not take effect. This approach prevents data contention to the UART FIFO memory buffers.

1.5.1.2 Peripheral ID Registers

Table 1-6 shows the Peripheral ID registers.

Table 1-6 Peripheral ID Registers Summary

Base Offset	Name	Description	Type	Reset Value
0xFE0	UARTPeriphID0	UARTPeriphID0 Register. Always set to reset value.	RO	0x0011
0xFE4	UARTPeriphID1	UARTPeriphID1 Register. Always set to reset value.	RO	0x0010
0xFE8	UARTPeriphID2	UARTPeriphID2 Register. Always set to reset value.	RO	0x00_4 [1]
0xFEC	UARTPeriphID3	UARTPeriphID3 Register. Always set to reset value.	RO	0x0000

1. The value depends on the revision of the UART. For version r1p4 the value is 2, e.g. 0x24. For version r1p5 the value is 3, e.g. 0x34. See the TRM for more information.

1.5.1.3 Primecell ID Registers

Table 1-7 shows the Primecell ID registers.

Table 1-7 Primecell ID Registers Summary

Base Offset	Name	Description	Type	Reset Value
0xFF0	UARTPCellID0	UARTPCellID0 Register. Always set to reset value.	RO	0x000d
0xFF4	UARTOCellID1	UARTPCellID1 Register. Always set to reset value.	RO	0x00f0
0xFF8	UARTPCellID2	UARTPCellID2 Register. Always set to reset value.	RO	0x0005
0xFFC	UARTPCellID3	UARTPCellID3 Register. Always set to reset value.	RO	0x00b1

1.5.1.4 Transmit FIFO Registers

Table 1-8 shows the Transmit FIFO registers

Table 1-8 Transmit FIFO Registers Summary

Name	Description	Type
TxFIFOTip	Register being transferred	RO
TxFIFOBase	Last register written	RO
TxFIFOSize	Difference between Tip and Base	RO
TxFIFOElement0-15	Transmit FIFOs	RO

1.5.1.5 Receive FIFO Registers

Table 1-9 shows the Receive FIFO registers

Table 1-9 Receive FIFO Registers Summary

Name	Description	Type
RxFIFOTip	Register being received	RO
RxFIFOBase	Last register read	RO
RxFIFOSize	Difference between Tip and Base	RO
RxFIFOElement0-15	Receive FIFOs	RO

1.6 Available Profiling Data

The UART PL011 component has no profiling capabilities.