

PRELIMINARY DATA SHEET

PUC 303xA Programmable Universal Controller

Edition April 2, 2002
6251-565-1PD



Contents

Page	Section	Title
5	1.	Introduction
5	1.1.	Features
5	1.2.	Application Overview
6	2.	Functional Description
6	2.1.	Device Overview
6	2.1.1.	64-MHz ARM7TDMI
6	2.1.2.	48 + 8 kByte SRAM
6	2.1.3.	256 kByte Embedded Flash
6	2.1.4.	Memory Protection Unit
6	2.1.5.	Power Management Unit
6	2.1.6.	12-Mbit/s USB 1.1 Function Core
6	2.1.7.	66 General-Purpose I/Os
6	2.1.8.	5 x Synchronous Serial Port
6	2.1.9.	2 x UARTs / IrDA
6	2.1.10.	3 x Timer / Counters
6	2.1.11.	Watchdog
6	2.1.12.	Interrupt Controller
6	2.1.13.	Real Time Clock
7	2.2.	ARM7TDMI Processor and AHB/APB Buses
7	2.2.1.	Memory Map
8	2.3.	Clock Generation
11	2.3.1.	Clock Utilization Summary
12	2.3.2.	Phase-Locked Loop Device
13	2.4.	Power-on and Resets
15	2.5.	AHB Peripherals
15	2.5.1.	Flash Memory
15	2.5.1.1.	Flash Memory Map
16	2.5.1.2.	Flash Configuration Registers
19	2.5.1.3.	Flash Features
19	2.5.1.4.	Programmable Wait States
19	2.5.1.5.	Flash Programming and Erasing
20	2.5.2.	Security of Application Code
20	2.5.3.	SRAM
20	2.5.3.1.	Features
20	2.5.3.2.	General Overview
21	2.5.4.	Memory Protection Unit
21	2.5.4.1.	SRAM Remapping
21	2.5.4.2.	MPU Address Space
24	2.5.5.	USB Interface
26	2.6.	APB Peripherals
26	2.6.1.	Power Management Unit
26	2.6.1.1.	Power States
27	2.6.1.2.	Power Management Register Descriptions

Contents, continued

Page	Section	Title
37	2.6.2.	Interrupt Controller
37	2.6.2.1.	Features
37	2.6.2.2.	Interrupt List
38	2.6.2.3.	General Overview
38	2.6.2.4.	Programming Guide
40	2.6.2.5.	Interrupt Controller Register Definitions
48	2.6.3.	UARTs
48	2.6.3.1.	Features
48	2.6.3.2.	UART Functions
50	2.6.3.3.	UART Register Map
65	2.6.4.	Synchronous Serial Ports
65	2.6.4.1.	SSP Peripheral Operation
66	2.6.4.2.	SSP Modes
66	2.6.4.3.	SSP Primary Modes
67	2.6.4.4.	SSP Secondary Modes
68	2.6.4.5.	Example SSP Communication Waveforms
70	2.6.4.6.	SSP Register Descriptions
79	2.6.4.7.	SSP Interrupt Logic
80	2.6.5.	Timers
80	2.6.5.1.	Features
80	2.6.5.2.	Description
81	2.6.5.3.	Initialization
81	2.6.5.4.	Modes of Operation
82	2.6.5.5.	Generation of Waveforms
83	2.6.5.6.	TTC Register Descriptions
89	2.6.6.	Real Time Clock
89	2.6.6.1.	Features
89	2.6.6.2.	Description
89	2.6.6.3.	Standby Mode
89	2.6.6.4.	Register Map and Formats
98	2.6.6.5.	Programming Instructions
101	2.6.7.	Watchdog Timer
101	2.6.7.1.	Features
101	2.6.7.2.	Description
102	2.6.7.3.	Watchdog Register Descriptions
102	2.6.7.4.	Watchdog Timer Reset
102	2.6.7.5.	Watchdog Timer Enable Sequence
106	2.6.8.	General Purpose I/O Module
106	2.6.8.1.	Features
106	2.6.8.2.	Description
106	2.6.8.3.	Bypass Mode
106	2.6.8.4.	Interrupts
107	2.6.8.5.	Signal Interface
107	2.6.8.6.	Initialization
107	2.6.8.7.	GPIO Address Map
107	2.6.8.8.	Programming Interface

Contents, continued

Page	Section	Title
109	3.	Specifications
109	3.1.	Outline Dimensions
110	3.2.	Pin Connections and Short Descriptions
114	3.3.	Pin Descriptions
114	3.3.1.	Special Function Pins
115	3.3.2.	Shared GPIO / Special Function Pins
115	3.3.3.	General Purpose I/O Pins Without Special Function
118	3.4.	Pin Circuits
119	3.5.	Electrical Characteristics
119	3.5.1.	Absolute Maximum Ratings
120	3.5.2.	Recommended Operating Conditions
121	3.5.3.	Power Consumption
123	3.5.4.	DC Characteristics
123	3.5.4.1.	Digital Outputs
123	3.5.4.2.	Hysteresis of Schmitt Trigger Inputs
123	3.5.4.3.	Internal Pull-up / Pull-down Resistor Values
124	3.5.5.	AC Characteristics
124	3.5.5.1.	SSP Slave (SCLKx and SFRMx are Inputs)
124	3.5.5.2.	SSP Master (SCLKx and SFRMx are Outputs)
125	3.5.5.3.	Timing Diagrams
126	3.5.5.4.	AC Characteristics for CLKOUT Pin
126	3.5.5.5.	Timing Requirement for WAKE_UP Pin
126	3.5.5.6.	Timing Requirement for TIMER_x Inputs
126	3.5.5.7.	Timing Requirement for GPIO Inputs
126	3.5.5.8.	Rise/Fall Times of 4-mA and 12-mA Outputs
127	3.5.6.	Flash Programming Characteristics
129	4.	List of Abbreviations
130	5.	Data Sheet History

Programmable Universal Controller

1. Introduction

The Programmable Universal Controller PUC 303xA is ideally suited for use in portable consumer devices. It is equipped with an embedded ARM7TDMI processor, a highly flexible power management scheme, and embedded Flash. Fig. 1–2 shows a simplified functional block diagram of the PUC 303xA.

1.1. Features

- 64-MHz ARM7TDMI
- 48 kByte embedded SRAM
- 256 kByte embedded Flash
- Memory Protection Unit
- Secure Mode Controller
- Power Management Unit with integrated PLL
- 12 Mbit/s USB 1.1 function core with integrated transceiver (PUC 3030A only) and dedicated 8 kByte SRAM
- I²C Master/Slave interface
- 66 General-purpose I/Os
- 5 × Synchronous Serial Port (including I²S master), 24 Mbit/s max. speed as master, 16 Mbit/s max. speed as slave.
- 2 × UARTs / IrDA
- 3 × timer / counter

- Watchdog
- Interrupt Controller with hardware prioritization
- Real Time Clock (not available for all packages)
- Especially suited to interface with Micronas’ MAS 35x9F / MAS 3587F / SAC 3590A devices

1.2. Application Overview

The PUC 303xA meets the requirements of digital audio players, specific mobile network appliances, digital audio recorders (including water-marking and DRM), and wireless applications such as Bluetooth™.

The PUC 303xA device offers appropriate security features, embedded Flash, and a variety of communication interfaces, including USB connectivity.

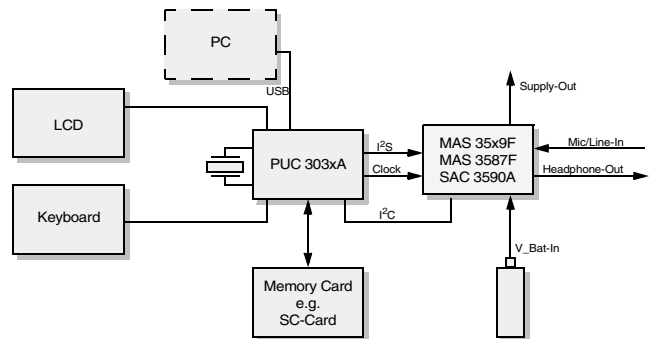


Fig. 1–1: Example application: multistandard secure player / recorder

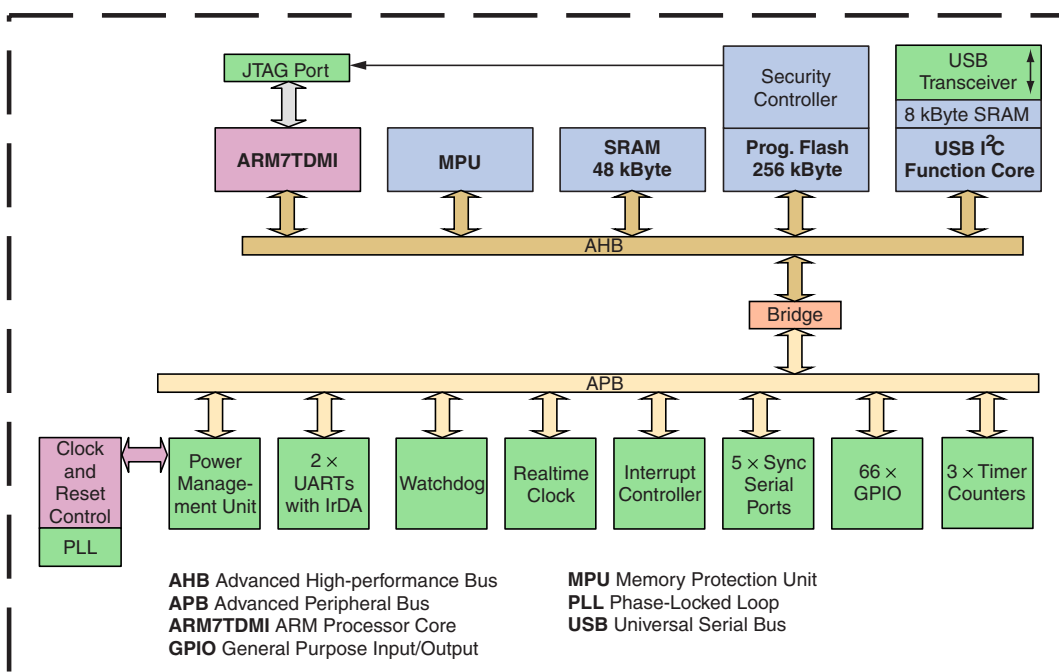


Fig. 1–2: Simplified functional block diagram of the PUC 303xA (see also List of Abbreviations on page 129)

2. Functional Description

2.1. Device Overview

2.1.1. 64-MHz ARM7TDMI

The embedded ARM7TDMI RISC processor runs at frequencies up to 64 MHz.

The ARM processor and associated bus structures are configured for **little endian** operation.

2.1.2. 48 + 8 kByte SRAM

The embedded SRAM is divided into 48 kBytes zero-wait-state memory directly connected to the AMBA bus via an SRAM interface, and 8 kBytes contained within the USB space. The latter is accessible directly from both the USB core and the ARM7TDMI processor.

2.1.3. 256 kByte Embedded Flash

The embedded Flash is connected to the AHB via a Flash interface and allows storage of boot and application code. The Flash interface also implements protection modes to allow secure storage of application code and data.

2.1.4. Memory Protection Unit

A Memory Protection Unit (MPU) is provided for managing read and write access control for the ARM7TDMI memory map. It is also used to control the Flash / SRAM remapping functions.

2.1.5. Power Management Unit

The Power Management Unit (PMU) provides full control of the devices power modes, including 'operating', 'idle', 'stand-by', and 'off' modes.

2.1.6. 12-Mbit/s USB 1.1 Function Core

The USB 1.1 compliant function allows communication with a USB host-PC. Included is a built-in USB transceiver cell ensuring that minimal external components are required.

2.1.7. 66 General-Purpose I/Os

Up to 66 General Purpose I/O (GPIO) pins can be used under software control. Many of these pins implement shared functionality to reduce pin count of the device.

2.1.8. 5 x Synchronous Serial Port

All of the five provided Synchronous Serial Ports can implement Motorola SPI, National Semiconductor Microwire, Texas Instruments Synchronous Serial and I²S modes at speeds of up to 24 Mbit/s as a master and up to 16 Mbit/s as a slave.

2.1.9. 2 x UARTs / IrDA

Two fully independent UARTs are available running at a maximum rate of 115.2 kBaud. Each UART can also be configured for IrDA mode.

2.1.10. 3 x Timer / Counters

Three independent counter timers are provided for timing control during software routines, by using the various interrupts which can be generated. Two of the timers can also be clocked off an external clock, allowing edge counting and pulse width monitoring. In addition these two timers can also be used to generate periodic waveforms to an external device.

2.1.11. Watchdog

The watchdog timer provides a method to monitor software activity, and reset the device if a lock up occurs. A programmable timeout range is provided to allow for flexibility.

2.1.12. Interrupt Controller

The interrupt controller incorporates hardware prioritization of the 32 interrupt sources. This reduces the load on the processor when dealing with interrupts. All interrupts are maskable and can also be triggered by software.

2.1.13. Real Time Clock

The real time clock provides accurate reporting of time of day (hours, minutes and seconds) and calendar functions (day of week, date of month, month, year, century). It also has an alarm function (month, date, hour, minute or second resolution).

The real time clock is only available where application and package permit.

2.2. ARM7TDMI Processor and AHB/APB Buses

The Programmable Universal Controller PUC 303xA contains an embedded ARM7TDMI processor, which operates at a maximum frequency of 64 MHz.

The Advanced High-Speed Bus (AHB) bus connects the following peripherals to the ARM7TDMI processor:

- MPU
- Flash memory
- SRAM
- USB (Universal Serial Bus) core.

The Advanced Peripheral Bus (APB) bus connects the following peripherals to the AHB via the AHB/APB bridge:

- UARTs
- SSPs
- GPIO
- Interrupt Controller

- Timers
- Watchdog Timer
- Real Time Clock
- Power Management Unit

The ARM7TDMI processor and associated bus structures are configured for **little endian** operation.

2.2.1. Memory Map

Table 2–1 states the base addresses of the memory mapped Flash, SRAM, and peripherals. For detailed register offsets, relative to base addresses, refer to the functional descriptions of specific components.

The memory map may be dynamically changed by remapping the SRAM and Flash memory areas. Details of this operation can be found in the Memory Protection description (see Section 2.5.4. on page 21).

Table 2–1: List of ARM7TDMI Processor Address Space

Base Address (Boot)	Base Address (Remapped)	Comment
0000.0000 _{hex} 0008.0000 _{hex}	0008.0000 _{hex}	Embedded Flash (256KBytes) base address
0010.0000 _{hex}	0000.0000 _{hex}	Embedded SRAM (48KBytes) base address
0012.0000 _{hex}	0012.0000 _{hex}	USB base address
001A.0000 _{hex}	001A.0000 _{hex}	MPU base address
001B.0000 _{hex}	001B.0000 _{hex}	TPC (Test Port Controller) base address
001C.0000 _{hex}	001C.0000 _{hex}	Programmable Interrupt Controller base address
001C.4000 _{hex}	001C.4000 _{hex}	UART 1 base address
001C.8000 _{hex}	001C.8000 _{hex}	UART 2 base address
001C.C000 _{hex}	001C.C000 _{hex}	General Purpose I/O base address
001D.0000 _{hex}	001D.0000 _{hex}	Serial interface 1 base address (SSP1)
001D.4000 _{hex}	001D.4000 _{hex}	Serial interface 2 base address (SSP2)
001D.8000 _{hex}	001D.8000 _{hex}	Serial interface 3 base address (SSP3)
001D.C000 _{hex}	001D.C000 _{hex}	Serial interface 4 base address (SSP4)
001E.0000 _{hex}	001E.0000 _{hex}	Serial interface 5 base address (SSP5, I ² S)
001E.4000 _{hex}	001E.4000 _{hex}	Timers/Counters base address
001E.8000 _{hex}	001E.8000 _{hex}	Watchdog Timer base address
001E.C000 _{hex}	001E.C000 _{hex}	Real Time Clock base address
001F.0000 _{hex}	001F.0000 _{hex}	Power Management Unit base address

2.3. Clock Generation

The PUC 303xA implements a powerful and flexible clocking scheme in order to provide maximum flexibility in power management and to provide suitable clocks for the integrated peripherals. The features of the clock generation scheme are

- Two oscillator clock pad inputs, either allowing direct connection to a crystal oscillator circuit or clock source.
- Software-selectable Phase Locked Loop (PLL) device on the main system oscillator clock generation logic.
- Programmable PLL dividers, allowing dynamic, glitch-free configuration of the PLL and a wide range of system clock frequencies.
- Automatic relock timing when the PLL is powered up, or when divider values are changed.

- Individually programmable clock dividers, on all but dedicated USB clocks, providing dynamic glitch-free frequency changing.
- Individually programmable clock enables on all clocks to stop them in their low phases.
- Maskable wake-up condition selection.
- Real Time Clock oscillator, with programmable enable.
- Real Time Clock support, by generating a 1 Hz clock from the Real Time Clock oscillator clock frequency ranging from 32768 Hz to 8.372224 MHz.

All of the features in the clock generation can be controlled by means of the addressable registers in the Power Management Unit.

Fig. 2-1 shows the major blocks in the clock generation logic.

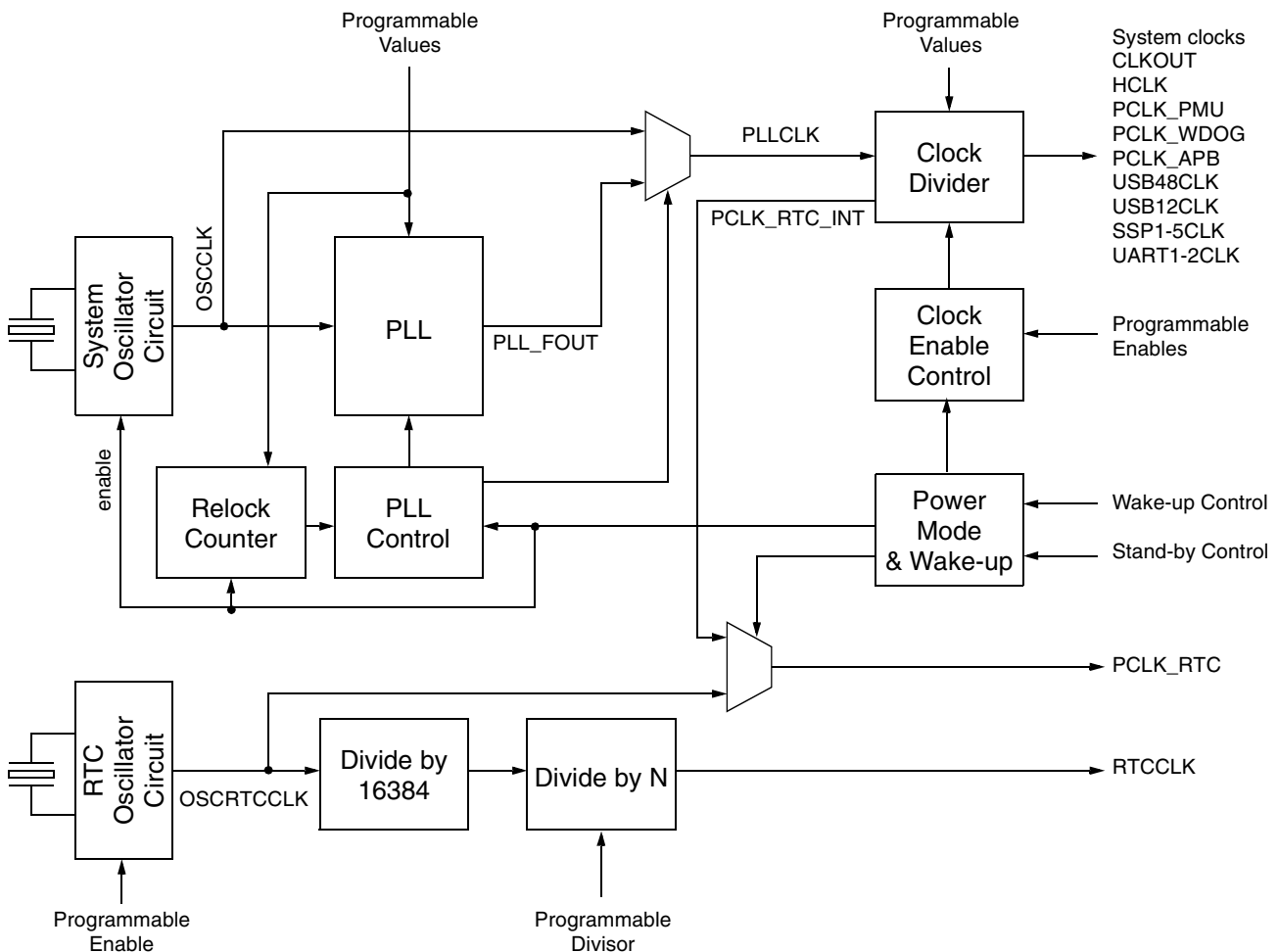


Fig. 2-1: Clock Generation Functional Block Diagram

The PUC 303xA provides two crystal oscillator circuits, one for the main system operation and one for the real time clock and standby operation. Both of these pads enable either direct clock input or a crystal to be connected.

The main system oscillator and subsequent circuitry is used to generate most of the required clocks for normal operation of the PUC 303xA.

The clock dividers can be sourced from either the oscillator clock input, OSCCLK, or from the output of an internal PLL, PLL_FOUT. This selection is controlled by modifying the VCO_BYPASS bit of the PMU_CLK_EN register (see Section 2.7.1. on page 26).

The PLL output can be dynamically controlled by changing the programmable PLL dividers in the PMU_VCO_DIV register in the Power Management Unit (PMU, see Section 2.3.2. on page 12 for details of how to update these dividers). The PLL control and relock counter blocks of the clock generator ensure safe handling of PLL divider changes, allowing the PLL output to settle before using it.

The clock enable and divider blocks are used to provide control over the individual clocks in the design. Each of the major clocks can be enabled or disabled, and the frequency controlled by means of separate programmable dividers. All of these options are controlled through the PMU registers (see Section 2.7.1. on page 26).

The real time clock oscillator pad is fed into a divider circuit, consisting of a fixed divide by 16384 and programmable divide by 2 to 511. The programmable element of this divider allows the oscillator input frequencies to be of the range 32.768 kHz to 8.372224 MHz, and still generate the required 1Hz clock, RTCCLK, used by the Real Time Clock.

The power mode and wake-up control block is used to interpret information from the PMU registers, and then to place the clock generator in an appropriate power state. Please refer to the PMU section of this data sheet (Section 2.7.1. on page 26) for details of power modes.

Table 2–2 summarizes the clocks used or generated within the PUC 303xA.

Table 2–2: Clocks

Clock	Description	Sources	Divider Range	Frequency Range
OSCCLK	Main reference oscillator clock from pad	input clock pad	1	6 to 28 MHz
OSCRTCCLK	RTC oscillator clock from pad	input clock pad	1	32768 Hz to 8.372224 MHz
PLL_FOUT	Output from PLL VCO	output from pll	See PLL section	80 ¹⁾ to 128 ¹⁾ MHz
PLLCLK	Multiplexed clock from OSCCLK and PLL_FOUT (i.e bypass PLL)	PLL_FOUT or OSCCLK	1	6 to 128 MHz
CLKOUT	PUC 303xA output, used to clock external device (e.g. MASF).	PLLCLK or OSCCLK	1 to 15	334 kHz to 28 MHz ¹⁾
HCLK	AHB Interface and ARM7TDMI system clock	PLLCLK	2 to 63	80 kHz to 64 MHz
PCLK_PMU	APB system clock dedicated to Power Management Unit Registers	PLLCLK	shares common divider with HCLK	
PCLK_WDOG	APB system clock dedicated to watchdog	PLLCLK	shares common divider with HCLK	
PCLK_RTC_INT	Clock generator version of Real Time Clock APB system clock.	PLLCLK	shares common divider with HCLK	

¹⁾ theoretical limit overridden by physical constraints

Table 2–2: Clocks, continued

Clock	Description	Sources	Divider Range	Frequency Range
PCLK_APB	APB system clock (used for all other APB peripherals)	PLLCLK	shares common divider with HCLK	
PCLK_RTC	APB system clock dedicated to the Real Time Clock. Generated by multiplexing PCLK_RTC_INT (for operating state) or OSCRTCCLK (for standby state)	PCLK_RTC_INT or OSCRTCCLK	1	32 kHz to 64 MHz
USB48CLK	USB sampling clock	PLLCLK	2 and 4	48 MHz ²⁾
USB12CLK	USB bus clock	PLLCLK	8 and 16	12 MHz ²⁾
SSP1CLK	Sampling clock for SSP 1	PLLCLK	2 to 63	80 kHz to 64 MHz
SSP2CLK	Sampling clock for SSP 2	PLLCLK	shares common divider with SSP1CLK	
SSP3CLK	Sampling clock for SSP 3	PLLCLK	shares common divider with SSP1CLK	
SSP4CLK	Sampling clock for SSP 4	PLLCLK	shares common divider with SSP1CLK	
SSP5CLK	Optional sampling clock for SSP 5. This is selectable between the divided PLLCLK or the SSP5CLK_IN input.	PLLCLK or SSP5CLK_IN	shares common divider with SSP1CLK	
SSP5CLK_IN	Sampling clock supplied from external device (e.g. MASF)	GPIO pad	1	24 MHz ¹⁾
UART1CLK	Sampling clock for UART 1	PLLCLK	2 to 63	80 kHz to 15 MHz ¹⁾
UART2CLK	Sampling clock for UART 2	PLLCLK	shares common divider with UART1CLK	
RTCCLK	Count enable for Real Time Clock (only used as an enable to PCLK_RTC)	OSCRTCCLK	32768 to 4194304	1 Hz

¹⁾ theoretical limit overridden by physical constraints

²⁾ nominal values for dividers 2/8 at PLLCLK=96 MHz are also upper limit values

2.3.1. Clock Utilization Summary

Table 2–3 summarizes the utilization of clocks on a module-by-module basis.

Table 2–3: Clock Utilization

Module	Clocks Used Within Module
JTAG Port	TCLK
ARM7TDMI processor / AHB Interface	HCLK
Memory Protection Unit	HCLK
48 kByte SRAM AHB Interface	HCLK
256 kByte Flash AHB Interface	HCLK
USB AHB Interface	HCLK, USB12CLK
USB core	USB48CLK, USB12CLK
AHB to APB Bridge	PCLK_APB
UARTs	PCLK_APB, UART*CLK
SSPs	PCLK_APB, SSP*CLK
GPIO	PCLK_APB
Watchdog	PCLK_WDOG
Real Time Clock	PCLK_RTC, RTCCLK ¹⁾
Timer / Counters	PCLK_APB
Interrupt Controller	PCLK_APB
Power Management Unit	PCLK_PMU
Clock Generator	OSCCLK, OSCRTCCLK, PLLCLK
¹⁾ RTCCLK is not used to clock registers but rather as a waveform that is sampled by PCLK_RTC	

2.3.2. Phase-Locked Loop Device

The PUC 303xA incorporates an embedded Phase Locked Loop (PLL) device, which can be selected to provide a source for the system clocks. The PLL can be configured under software control, providing an extraordinarily wide range of available frequencies and power-down capability.

The following features are available:

- A maximum output frequency of 128 MHz.
- A minimum output frequency of 80 MHz.
- Wide input clock frequency range of 6–28 MHz.
- Low jitter.
- Low power consumption in power down mode.
- Software selectable divider values for input and feedback dividers.
- Automatic re-lock waiting, where all system clocks are disabled until PLL output is stable.

Fig. 2–2 shows a simplified diagram of the structure of the PLL.

The frequency relationship between the PLL output frequency and the source frequency is shown in the equation below.

$$PLL_FOUT = PLL_FIN \times \frac{NF'}{NR'}$$

The division factors NR' and NF' are programmable through the Power Management Unit PMU_VCO_DIV Register. The actual value programmed needs to be two less than the required value.

For example, if a 96-MHz output frequency clock is required, from a 13-MHz crystal, the dividers chosen could be NF' = 96 and NR' = 13. Therefore, the divider values programmed into the PMU_VCO_DIV would be NF = 94 (5E_{hex}) and NR = 11 (0B_{hex}). So the value of the register should be set to 005E.000B_{hex}.

Note, that divider values of NF=192 and NR=26 could have been chosen to achieve the same frequency relationship, however a narrower loop bandwidth (larger NF divider value) increases PLL output jitter. This is due to lower comparison frequency at PFD. Therefore it is recommended to always choose the smallest values of equivalent divider sets.

Range for NF:

$$NF' \text{ from } 2 \dots 513 \Rightarrow \mathbf{NF \text{ from } 0 \dots 511}$$

Range for NR:

$$NR' \text{ from } 2 \dots 33 \Rightarrow \mathbf{NR \text{ from } 0 \dots 31}$$

Warning: Due to the very high flexibility in on-chip clock selection, special care must be taken not to overclock the PUC 303xA chip or regions of the chip. PLL division factors and clock dividers must be set to proper values, otherwise unpredictable operation or damage may result (refer to Table 2–2 on page 9 for details on clock constraints).

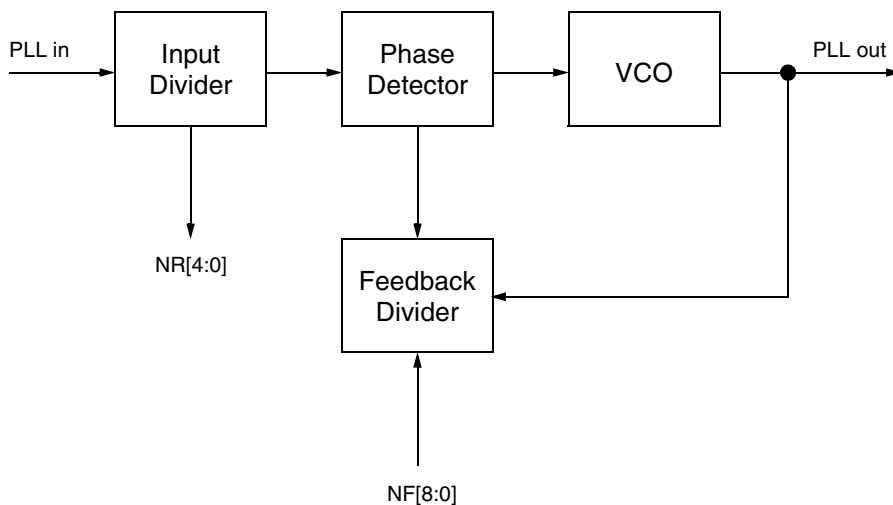


Fig. 2–2: Simplified Structure of the PLL

2.4. Power-on and Resets

The reset scheme is implemented to ensure seamless integration with various Micronas audio devices (e.g. MAS 35xxF family). There are four main reasons for the PUC 303xA to reset, each of which initiates a unique reset sequence. The four reset circumstances are

- Reset pin pulled low for power-on and warm reset
- Wake-up from standby state
- Software induced reset
- Watchdog time-out induced reset

To ensure correct timings on the reset sequence, there are three counters within the reset controller.

Fig. 2–3 shows a full power-on reset sequence, illustrating the use of the three counters. The timing of this sequence relies on counting a fixed number of oscillator clock edges and hence will vary depending on oscillator frequency. The times shown are based on a 28 MHz system oscillator clock frequency.

The first counter, OSC_COUNT, is enabled whilst the main system oscillator clock is settling. At completion the CLKOUT clock is enabled. This counter is only used during a power-on or wake-up reset.

The second counter, EXT_COUNT, ensures sufficient time for external Micronas audio device's DC/DC converters to supply sufficient power to power up the rest of the PUC 303xA. At this point all system clocks are enabled within the PUC 303xA. Again this counter is only used during a power-on or wake-up reset.

The third counter, SYS_COUNT, is used to keep the ARM7TDMI processor and other peripherals in reset, whilst the security state machine in the Flash interface decides the security state of the device (see Section 2.5.2. on page 20).

For a waveform diagram of the reset sequence, see Fig. 2–3 of this data sheet.

Table 2–4 summarizes how these resets are asserted depending on the source of the reset.

The internal PUC 303xA signals `n_reset`, `wakeup_rst`, `soft_rst` and `n_wd_rst`, are the trigger signals for the four reset sequences, and are activated as follows (refer to Section 2.7.1. on page 26 for information about setting the PMU registers):

- `n_reset`: 'Reset' input pin of PUC 303xA pulled low at power-up or warm reset.
- `wakeup_rst`: conditioned "wake-up" input pin of PUC 303xA, causing a wake-up reset from standby state if the appropriate mask bit in the `PMU_WAKE_MASK` register in the PMU is set.
- `soft_rst`: Software induced reset caused by writing to bit [0] of the `PMU_RESET` register.
- `n_wd_rst`: Watchdog Timeout induced reset trigger from the watchdog module, as described in Section 2.6.7. on page 101.

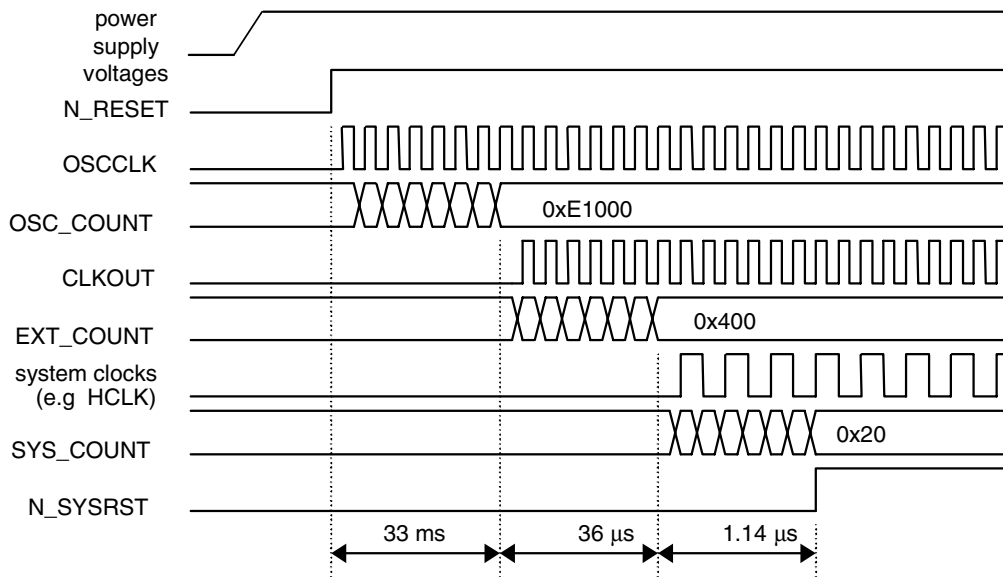


Fig. 2–3: Power-on reset sequence (timing values for 28-MHz system oscillator frequency)

Table 2–4: Resets

Source of Reset	PLL	Security and Flash Control	Watchdog	Real Time Clock	Rest of Chip
nRESET Pin	reset	reset	reset	reset	reset
Wake-up Reset	reset	reset	reset	X	reset
Software Reset	reset	X	X	X	reset
Watchdog Reset	X	X	X	X	reset
X=not reset by source					

2.5. AHB Peripherals

2.5.1. Flash Memory

2.5.1.1. Flash Memory Map

The PUC 303xA contains 256 kBytes of embedded Flash memory, which is split into two sections:

- 24 kByte BOOT area, for boot code,
- 232 kByte USER area, for application code,

There is also a 1 kByte INFORMATION area, for storing device specific production information, such as serial number.

These areas are illustrated in Fig. 2-4.

The Flash Memory is organized in rows of 64 words of width 32 bits, so that each row contains 256 Bytes.

For the BOOT and USER areas, 8 adjacent rows form a page of 2048 Bytes, whereas the INFO area consists of a single page of 4 rows = 1024 Bytes.

The page organization is relevant for Flash Page Erase cycles.

The Flash may be accessed through three separate interfaces:

- from the ARM7TDMI processor,
- through the JTAG port with the ARM7TDMI processor in debug state,
- through a dedicated parallel flash programming test mode (Parallel TM) using GPIO pins (intended for fast mass production programming and testing)

The JTAG and parallel interfaces may be enabled or disabled, depending on the security status of the device (see Section 2.5.2. on page 20).

The three interfaces have different access rights to the flash memory. Table 2-5 summarizes all of the access permissions.

The rest of this section describes accesses from the ARM7TDMI processor only, unless otherwise stated.

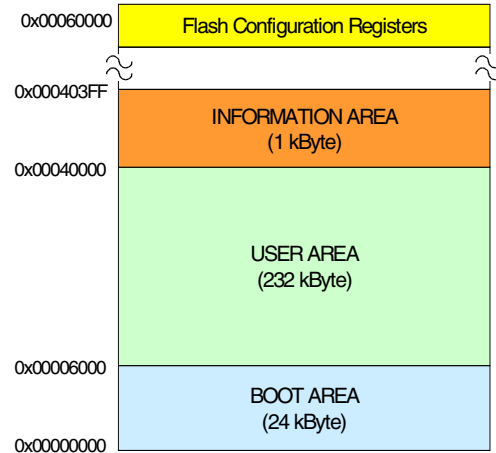


Fig. 2-4: Flash Memory Map

Table 2-5: Access Permissions

Interface	ARM7TDMI processor	JTAG	Parallel TM
BOOT Area Read	OK	OK	OK
BOOT Area Write	BLOCKED if not unlocked via JTAG	OK	OK
USER Area Read	OK	OK	OK
USER Area Write	OK	OK	OK
INFO Area Read	OK	OK	OK
INFO Area Write	BLOCKED	BLOCKED	OK
BOOT Area Page Erase	BLOCKED if not unlocked via JTAG	OK	OK
USER Area Page Erase	OK	OK	OK
INFO Area Page Erase	BLOCKED	BLOCKED	OK

OK=Access to region permitted
 BLOCKED=Access blocked by Security Controller

2.5.1.2. Flash Configuration Registers

The Flash configuration registers are located after the top of the INFORMATION (INFO) area of the Flash at an offset of 0006.0000_{hex} from the Flash area base.

The memory map for the flash configuration registers is shown in Table 2–6. These are fully described in Table 2–7. All addresses given are offset addresses relative to the Flash configuration register base of 0006.0000_{hex}.

Example: Writing to the CONFIG_REG register independent of state of the remap bit is done by writing to address 80000_{hex} + 60000_{hex} + 00014_{hex} = E0014_{hex}.

Table 2–6: Flash Configuration Register Map

Offset	R/W	Width	Name	Reset
00 _{hex}	W	2	WAIT_STATE	03 _{hex}
04 _{hex}	W	6	PERF_COUNT	0F _{hex}
08 _{hex}	W	16	BOOT_PROT	N/A
0C _{hex}	W	18	ADDRESS_REG	00000 _{hex}
10 _{hex}	W	32	DATA_REG	00000000 _{hex}
14 _{hex}	W	8	CONFIG_REG	00 _{hex}

Table 2–7: Flash Configuration Registers

Register Address	Function	Name								
00 00 _{hex}	<p>WAIT_STATE Register</p> <table border="1"> <tr> <td>Bit</td> <td>31:2</td> <td>1:0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>WAIT_VAL</td> </tr> </table> <p>bit[31:2] Reserved</p> <p>bit[1:0] WAIT_VAL Number of added Wait States during Read cycle. (refer to Table 2–8)</p>	Bit	31:2	1:0	Field	Reserved	WAIT_VAL	WAIT_STATE		
Bit	31:2	1:0								
Field	Reserved	WAIT_VAL								
00 04 _{hex}	<p>PERF_COUNT Register</p> <table border="1"> <tr> <td>Bit</td> <td>31:6</td> <td>5</td> <td>4:0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>PF_COUNT_EN</td> <td>PF_COUNT_VAL</td> </tr> </table> <p>bit[31:6] Reserved</p> <p>bit[5] PF_COUNT_EN When this bit is set, the Flash performance counter is enabled. If the gap between two consecutive read accesses to the Flash is long enough to allow the count to expire, the embedded Flash device will enter standby state. With this bit disabled ('0') the controller will never enter standby mode, delivering the most efficient times in terms of access speed.</p> <p>bit[4:0] PF_COUNT_VAL These 5 bits correspond to the number of cycles after the last read access the Flash memory will enter standby mode. PF_COUNT_EN must be set to enable the value of PF_COUNT_VAL. Example: 01111bin = 15dec HCLK timeout cycles.</p>	Bit	31:6	5	4:0	Field	Reserved	PF_COUNT_EN	PF_COUNT_VAL	PERF_COUNT
Bit	31:6	5	4:0							
Field	Reserved	PF_COUNT_EN	PF_COUNT_VAL							

Table 2–7: Flash Configuration Registers, continued

Register Address	Function	Name								
00 08 _{hex}	<p>BOOT_PROT Register</p> <table border="1"> <tr> <td>Bit</td> <td>31:16</td> <td>15:0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>BOOT_WR_KEY</td> </tr> </table> <p>bit[31:16] Reserved</p> <p>bit[15:0] BOOT_WR_KEY BOOT Flash area write enable key. A key value of ABCD_{hex} must be written via the JTAG port, e.g. by using the ARM ICE debugger, to enable further programme/erase accesses to the BOOT area of the Flash. Once written, the 'ARM7TDMI-processor'-mode will have full access permissions, to allow dedicated embedded software routines to programme/erase the BOOT area.</p>	Bit	31:16	15:0	Field	Reserved	BOOT_WR_KEY	BOOT_PROT		
Bit	31:16	15:0								
Field	Reserved	BOOT_WR_KEY								
00 0C _{hex}	<p>ADDRESS_REG Register</p> <table border="1"> <tr> <td>Bit</td> <td>17:8</td> <td>7:2</td> <td>1:0</td> </tr> <tr> <td>Field</td> <td>XADR_VAL (row)</td> <td>YADR_VAL (column)</td> <td>Byte address</td> </tr> </table> <p>bit[17:8] XADR_VAL (row) Flash X address when SW_SELECT (s. CONFIG_REG) is enabled</p> <p>bit[7:2] YADR_VAL (column) Flash Y address when SW_SELECT (s. CONFIG_REG) is enabled</p> <p>bit[1:0] Byte address These bits are ignored. It is recommended to write them as '0'.</p>	Bit	17:8	7:2	1:0	Field	XADR_VAL (row)	YADR_VAL (column)	Byte address	ADDRESS_REG
Bit	17:8	7:2	1:0							
Field	XADR_VAL (row)	YADR_VAL (column)	Byte address							
00 10 _{hex}	<p>DATA_REG Register</p> <table border="1"> <tr> <td>Bit</td> <td>31:0</td> </tr> <tr> <td>Field</td> <td>DIN_VAL</td> </tr> </table> <p>bit[31:0] DIN_VAL Data on Flash data bus when SW_SELECT is enabled</p>	Bit	31:0	Field	DIN_VAL	DATA_REG				
Bit	31:0									
Field	DIN_VAL									

Table 2–7: Flash Configuration Registers, continued

Register Address	Function	Name																				
00 14 _{hex}	<p>CONFIG_REG Register</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">Bit</th> <th style="text-align: center;">31:8</th> <th style="text-align: center;">7</th> <th style="text-align: center;">6</th> <th style="text-align: center;">5</th> <th style="text-align: center;">4</th> <th style="text-align: center;">3</th> <th style="text-align: center;">2</th> <th style="text-align: center;">1</th> <th style="text-align: center;">0</th> </tr> </thead> <tbody> <tr> <th style="text-align: center;">Field</th> <td style="text-align: center;">Reserved</td> <td style="text-align: center;">SW_SELECT</td> <td style="text-align: center;">WRITE_PROTECT</td> <td style="text-align: center;">PROG_VAL</td> <td style="text-align: center;">ERASE_VAL</td> <td style="text-align: center;">Reserved</td> <td style="text-align: center;">NVSTR_VAL</td> <td style="text-align: center;">YE_VAL</td> <td style="text-align: center;">XE_VAL</td> </tr> </tbody> </table> <p>Note: Set up the DATA_REG and ADDRESS_REG with the correct information before using this register.</p> <p>bit[31:8] Reserved</p> <p>bit[7] SW_SELECT When set, this bit enables programme and erase mode.</p> <p>bit[6] WRITE_PROTECT This bit is write only and when set will prevent SW_SELECT from being set. This prevents ANY writes or erases to the flash, until a new power-on or wake up reset.</p> <p>bit[5] PROG_VAL The value on this register is output on the PROG line to the Flash when SW_SELECT is set.</p> <p>bit[4] ERASE_VAL The value on this register is output on the ERASE line to the Flash when SW_SELECT is set.</p> <p>bit[3] Reserved</p> <p>bit[2] NVSTR_VAL The value on this register is output on the NVSTR line to the Flash when SW_SELECT is set.</p> <p>bit[1] YE_VAL The value on this register is output on the YE line to the Flash when SW_SELECT is set.</p> <p>bit[0] XE_VAL The value on this register is output on the XE line to the Flash when SW_SELECT is set.</p>	Bit	31:8	7	6	5	4	3	2	1	0	Field	Reserved	SW_SELECT	WRITE_PROTECT	PROG_VAL	ERASE_VAL	Reserved	NVSTR_VAL	YE_VAL	XE_VAL	CONFIG_REG
Bit	31:8	7	6	5	4	3	2	1	0													
Field	Reserved	SW_SELECT	WRITE_PROTECT	PROG_VAL	ERASE_VAL	Reserved	NVSTR_VAL	YE_VAL	XE_VAL													

2.5.1.3. Flash Features

The following list describes the features of the embedded Flash in the PUC 303xA device:

- Separate Software Control for Flash Programming (see Section 2.5.1.5.)
- Hardware security mechanism for boot code (see Section 2.5.1.5.)
- Byte, Halfword and Word addressable for reads
- Programmable wait states, to permit up to three wait states to be inserted during any read access.
- BOOT area write protection disable register, that is accessible only through the JTAG ports. This allows the ARM7TDMI processor to program the BOOT area of Flash.

2.5.1.4. Programmable Wait States

The required number of wait states for read accesses to the Flash depends on the following:

- system clock frequency (HCLK)
- if the Flash device is already in its standby mode
- if the row address (XADDR) of the Flash device has changed from the previous access

If one of the last two dependencies are true, at least 60 ns are required after HCLK before the read data is stable to be sampled. If both are false then only 40 ns access time are required.

The Flash Controller can detect if a read access is a 40 ns read or a 60 ns read and it will always put one less wait state than specified in Table 2–8 into 40 ns read accesses.

The value programmed into the WAIT_STATE register specifies the number of wait states for 60 ns reads and should always be the value shown in Table 2–8.

Table 2–8: Wait States to be programmed in WAIT_STATE register

System Clock Frequency (MHz)		Programmed Wait State Value
Min	Max	
>50	64	3
>25	50	2
>16.6	25	1
>0	16.6	0

2.5.1.5. Flash Programming and Erasing

All Writes and Erases are software controlled through the following three Flash configuration registers:

- ADDRESS_REG: Stores the address of the Flash location to be programmed.
- DATA_REG: Stores the data to be programmed into the Flash location.
- CONFIG_REG: Enables direct control of the control strobes to the Flash device. If a bit is set, then the corresponding signal is forced high to the Flash, if it is reset then the signal is low.

To perform a programming cycle to the flash, the following sequence should be performed:

- The address in the flash required to be written to, should be programmed into the ADDRESS_REG.
- The data to be written should be programmed into the DATA_REG.
- The CONFIG_REG should then be programmed a number of times in order to select the required sequence on the control strobes to the Flash.

An example of a typical programming cycle is shown in Fig. 3–13 on page 127. The timing symbols and parameters are defined in Table 3–1 on page 127. In addition, an example of a typical erase cycle is shown in Fig. 3–14 on page 128.

Since the required programming times for a Flash programming cycle are in the order of milliseconds, the PUC 303xA’s internal timers should be used to correctly time the programming sequence.

2.5.2. Security of Application Code

The security of the downloaded application code can be ensured by disabling external interfaces from directly reading the stored code in the Flash.

The Flash contains a readable memory location called the Protection Control Register, PCR, located at offset address 0001.8000_{hex}. Contents of this location will indicate whether the Parallel Flash interface, the JTAG interface, or the Test Peripheral Controller interface of the device are operating in secure or non-secure mode:

Secure mode is used to inhibit any accesses to the Flash from these interfaces, in order to ensure that application code cannot be read.

The default state of the device is to be non-secure. Once sensitive application code is loaded, the PCR can be programmed by either the JTAG or Parallel Flash interface, to enable security.

During a power on or wake-up reset, secure mode is selected for all interfaces. During the reset the Security Controller will read the protection control register and determine whether non-secure mode can then be selected.

Since the ARM7TDMI can boot from embedded Flash, this decision is made before the ARM7TDMI comes out of reset. This decision will remain fixed until the main system reset input is cycled again, or a wake-up from STANDBY state occurs.

The bits of the protection control register are defined in Table 2–9.

Table 2–9: Protection Control Register Bits

Bit	Function
2	JTAG Enable
1	Test Peripheral Controller Enable
0	Parallel Flash Interface Enable

The security of the protection control register shall be under the supervision of the boot software, by means of a unique access key.

2.5.3. SRAM

2.5.3.1. Features

- 48 kByte embedded SRAM accessible with zero wait states for sequential accesses on the AHB. Non-sequential accesses require up to one extra cycle.
- 8 kBytes embedded SRAM shared between the ARM7TDMI processor and the USB core
- Byte-, Halfword-, and Word-addressable
- 48 kByte SRAM has a buffer to store last 32-bit data read. This can reduce power consumption, especially in THUMB mode
- Interfaces through AHB (the ARM7TDMI processor or JTAG)
- 48 kByte SRAM can be remapped to address 0_{hex} (see Section 2.5.4.)

2.5.3.2. General Overview

The PUC 303xA supports 48 kByte of memory directly on the AHB bus. Additional 8 kByte of SRAM are accessible through the USB interface. Accesses to this memory will incur additional wait states. Please refer to Section 2.5.5. on page 24 for details of the USB interface. The rest of this section only applies to the main 48 kBytes of SRAM.

A buffer in the SRAM interface stores the last data read from the SRAM device. Power consumption can be reduced by releasing data to the ARM7TDMI processor from this internal buffer rather than actually reading the SRAM whenever possible.

This is particularly beneficial when the ARM7TDMI processor is running in THUMB mode as the instructions are only 16 bits wide. Therefore, one location in SRAM actually holds two instructions and can be serviced with only one read to the SRAM.

2.5.4. Memory Protection Unit

The Memory Protection Unit (MPU) provides the following functionality:

- manages memory protection within the ARM7TDMI core’s memory map,
- controls the remapping of SRAM to the ARM7TDMI processor base address.

The MPU is an AHB peripheral. Access to all AHB peripherals including the MPU is controlled by an AHB address decoder, and is dependent upon the enable outputs from the MPU.

The MPU consists of 16 user-definable segments, each consisting of a start address MPUSTRTn, an end address MPUENDn, a read enable MPURDEN[n] and a write enable MPUWREN[n]. The start and end addresses are compared with bits 10 through 20, the most significant 11 bits used in the PUC 303xA architecture.

The current address is continually compared against the 16 segment address pairs. If an access address is detected to be (inclusively) within a segment’s start and end values then permission for the ARM7TDMI processor to perform the current access will be granted using the state of the segment’s read and write enable bits. The logic function of this mechanism for segment 0 is shown in Fig. 2–5.

The MPU is only active when the ARM7TDMI processor is in USER mode, it has no protection effect in other processor operating modes.

The default values of the MPU registers dictate that the MPU has to be programmed before USER mode is entered, otherwise all memory accesses will result in a Data Abort interrupt. Thereafter, the MPU can also be programmed in USER mode as long, as its register locations are not protected.

2.5.4.1. SRAM Remapping

The REMAP bit of the MPUREMAP register defines the location of the SRAM within the ARM7TDMI processor’s memory map. When 0 (default) the Flash based at 0008.0000_{hex} is mirrored at 0000.0000_{hex}, and the 48KB SRAM is located at 0010.0000_{hex}. When this bit is set to 1, the SRAM is relocated from 0010.0000_{hex} to 0000.0000_{hex}. Address space above the remapped SRAM that was previously occupied by mirrored Flash becomes unused and causes an abort if accessed.

2.5.4.2. MPU Address Space

Table 2–10 on page 22 shows the registers within the MPU. Addresses are address offsets, relative to the MPU base address 001A.0000_{hex}.

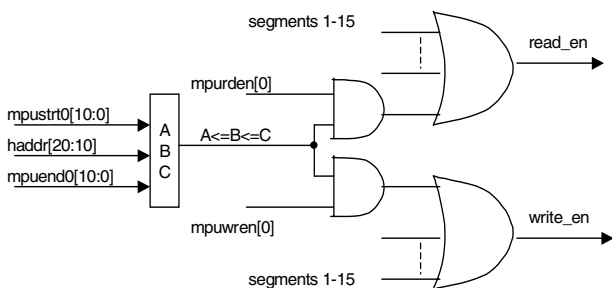


Fig. 2–5: Memory Protection Logic Function

Table 2–10: MPU Register Map

Offset	R/W	Width	Name	Comment	Default
000 _{hex}	R/W	2	MPUREMAP	bit 0: Reserved bit 1: REMAP (R/W) =0 mirrors Flash to the ARM7TDMI processor base address =1 remaps SRAM to the ARM7TDMI processor base address	0 _{hex}
004 _{hex}	R/W	16	MPURDEN	Bits 0 through 15 are active high read enables for MPU segments 0 through 15 respectively.	0000 _{hex}
008 _{hex}	R/W	16	MPUWREN	Bits 0 through 15 are active high write enables for MPU segments 0 through 15 respectively	0000 _{hex}
00C _{hex}	R/W	27	MPUSEG0	Start and End address for MPU segment 0	000.0000 _{hex}
010 _{hex}	R/W	27	MPUSEG1	Start and End address for MPU segment 1	000.0000 _{hex}
014 _{hex}	R/W	27	MPUSEG2	Start and End address for MPU segment 2	000.0000 _{hex}
018 _{hex}	R/W	27	MPUSEG3	Start and End address for MPU segment 3	000.0000 _{hex}
01C _{hex}	R/W	27	MPUSEG4	Start and End address for MPU segment 4	000.0000 _{hex}
020 _{hex}	R/W	27	MPUSEG5	Start and End address for MPU segment 5	000.0000 _{hex}
024 _{hex}	R/W	27	MPUSEG6	Start and End address for MPU segment 6	000.0000 _{hex}
028 _{hex}	R/W	27	MPUSEG7	Start and End address for MPU segment 7	000.0000 _{hex}
02C _{hex}	R/W	27	MPUSEG8	Start and End address for MPU segment 8	000.0000 _{hex}
030 _{hex}	R/W	27	MPUSEG9	Start and End address for MPU segment 9	000.0000 _{hex}
034 _{hex}	R/W	27	MPUSEG10	Start and End address for MPU segment 10	000.0000 _{hex}
038 _{hex}	R/W	27	MPUSEG11	Start and End address for MPU segment 11	000.0000 _{hex}
03C _{hex}	R/W	27	MPUSEG12	Start and End address for MPU segment 12	000.0000 _{hex}
040 _{hex}	R/W	27	MPUSEG13	Start and End address for MPU segment 13	000.0000 _{hex}
044 _{hex}	R/W	27	MPUSEG14	Start and End address for MPU segment 14	000.0000 _{hex}
048 _{hex}	R/W	27	MPUSEG15	Start and End address for MPU segment 15	000.0000 _{hex}

Table 2–11: MPU Registers

Register Address	Function	Name										
000 _{hex} to 048 _{hex}	<p>MPUSEGx Register</p> <table border="1" data-bbox="288 427 1201 501"> <thead> <tr> <th data-bbox="288 427 336 461">Bit</th> <th data-bbox="336 427 504 461">31:27</th> <th data-bbox="504 427 770 461">26:16</th> <th data-bbox="770 427 938 461">15:11</th> <th data-bbox="938 427 1201 461">10:0</th> </tr> </thead> <tbody> <tr> <th data-bbox="288 461 336 495">Field</th> <td data-bbox="336 461 504 495">Reserved</td> <td data-bbox="504 461 770 495">MPUEND[10:0]</td> <td data-bbox="770 461 938 495">Reserved</td> <td data-bbox="938 461 1201 495">MPUSTRT[10:0]</td> </tr> </tbody> </table> <p>This register stores the most significant 11 bits of a Start address MPUSTRTx and the most significant 11 bits of an End address MPUENDx of user-definable memory segment x.</p> <p>Both, MPUEND[10:0] and MPUSTRT[10:0] are compared continually with the most significant bits [20:10] of the address bus, thus implying a resolution of 1 kByte as the smallest segment quantity that can be permitted access in USER mode.</p> <p>bit[31:27] Reserved</p> <p>bit[26:16] MPUEND[10:0]</p> <p>bit[15:11] Reserved</p> <p>bit[10:0] MPUSTRT[10:0]</p>	Bit	31:27	26:16	15:11	10:0	Field	Reserved	MPUEND[10:0]	Reserved	MPUSTRT[10:0]	MPUSEGx
Bit	31:27	26:16	15:11	10:0								
Field	Reserved	MPUEND[10:0]	Reserved	MPUSTRT[10:0]								

2.5.5. USB Interface

The PUC 3030A¹⁾ device incorporates a USB specification 1.1 compliant slave interface with the following features:

- USB specification 1.1/1.0 compliant, with built in transceiver cell.
- Full speed slave interface with 12 Mbit/s bit rate.
- 8 kByte dedicated USB SRAM.
- The hardware provides the following end points:
 - EP0: dedicated.
 - EP1 to EP7 software-programmable.
- Can be programmed with bulk endpoints, interrupt, or isochronous endpoints.
- Suspend mode, which can be used to control PUC 3030A power management.
- Embedded controller, dedicated to USB housekeeping.
- Interface with AHB allows access to whole address space of USB core from the ARM7TDMI processor processor, including SRAM.
- Three optimized access types from the ARM7TDMI processor, including pipelined read to free up the processor for other tasks.
- Application code download of embedded controller software from embedded Flash to USB SRAM.

The architecture of the USB core is illustrated in Fig. 2-6.

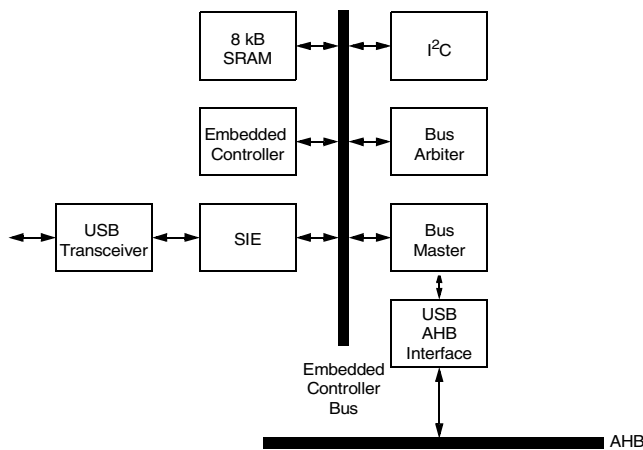


Fig. 2-6: USB Function Core Functional Diagram

¹⁾On PUC 3033A device, the USB interface is not available, but SRAM and I²C blocks are functional.

The USB core comprises the following blocks:

- Embedded controller: Used to control and update the USB endpoint registers and for general house-keeping of the SRAM.
- USB serial interface (SIE): For USB transaction handling and transfer of data to SRAM.
- 8 kBytes SRAM: Used primarily for storing the USB endpoint registers, data buffers and application code for the embedded controller. The USB SRAM may also be used as scratchpad RAM by the ARM7TDMI processor if the USB interface is not being used (scratchpad RAM functionality is also available on PUC 3033A).
- Bus arbiter: for arbitration between the 3 bus masters: SIE, embedded controller and Test/USB AHB interface.
- Bus Master: A port to allow an external source (either the Test or the USB AHB interface) direct control of any addressable location.
- I²C bus master and slave devices, used under control of the embedded controller (functionality is also available on PUC 3033A).

The ARM7TDMI processor can access any region of memory within the USB core space, including the SRAM. However because of clock domain differences between the two buses, accesses from the ARM7TDMI processor may take several cycles, and hold up the processor with wait states. The exact quantity of cycles taken depends on the relationship between the HCLK and USB12CLK, and the traffic load on the USB embedded controller bus.

To allow the ARM7TDMI processor to complete other system tasks during this time, the USB core memory map is repeated four times in the ARM7TDMI processor memory map. Each copy performs a different type of access to the USB core as shown in Table 2-12 (yyyy = address offset within USB core).

Table 2–12: USB Core Access

Address	R/W	Description
0012.yyy _{hex}	R/W	Initiate a new byte, half or word read or write, and wait for completion. In this mode the ARM7TDMI processor will be held with wait states, until the access has completed.
0014.yyy _{hex}	R	First pipeline read. Initiates a new byte, half or word read, but does not wait for completion. In this mode the ARM7TDMI processor is immediately released, unless the USB interface is busy with a previous access. Data read by this access is invalid. This enables the data to be immediately available the next time the ARM7TDMI processor performs a read.
	W	Initiate a new byte, half or word write. In this mode the ARM7TDMI processor is immediately released, unless the USB interface is busy with a previous access.
0016.yyy _{hex}	R	Pipelined read. Initiates a new byte, half or word read, and waits for previously requested data. In this mode the read data supplied will be the previously addressed data. In this way successive reads can be requested, so that the ARM7TDMI processor is free to handle the data in between accesses to the USB.
	W	Initiate a new byte, half or word write. In this mode the ARM7TDMI processor is immediately released, unless the USB interface is busy with a previous access.
0018.yyy _{hex}	-	Reserved for test purposes

2.6. APB Peripherals

The PUC 303xA device incorporates numerous commonly used peripherals which make the device very flexible and ideally match the demands of Micronas' MASF devices. These peripherals are accessible to the ARM7TDMI processor core through the AMBA Advanced Peripheral Bus (APB) and are described in the following sections.

2.6.1. Power Management Unit

The Power Management Unit is used to control the various power states of the device.

2.6.1.1. Power States

The device supports the following power-related states:

- The OPERATING State
- The IDLE State
- The STANDBY State (if Real Time Clock RTC is available)
- The OFF State (if RTC is unavailable)

Fig. 2-7 illustrates how these states are reached.

2.6.1.1.1. OPERATING State

In the OPERATING state, all peripherals can be active, depending on the clocks enabled by the Power Management Unit.

Within the OPERATING state, the PLL may be enabled or turned off/bypassed.

2.6.1.1.2. IDLE State

In the IDLE state, the clock to the ARM7TDMI processor bus and the watchdog timer clock are disabled, while the peripherals on the APB bus may remain active. The device will return to the OPERATING state if an interrupt is generated from any of the APB peripherals.

The PLL will operate as in OPERATING state.

2.6.1.1.3. STANDBY State

This state is only valid in applications and with device packages where the Real Time Clock (RTC) is available.

In the STANDBY state, all the clocks to both the AHB and the APB are disabled, with the exception of the RTC clocks.

For further power efficiency, the PLL and main oscillator are also powered down when the device is in the STANDBY state.

The device re-enters OPERATING state when either an RTC alarm, a USB plug, end of USB suspend, or a valid wake-up condition is detected.

2.6.1.1.4. OFF State

In the OFF state, all clocks are disabled and can only be restarted following a power-on or warm reset via nRESET pin.

OFF state is only valid if the RTC is not available with application or device package.

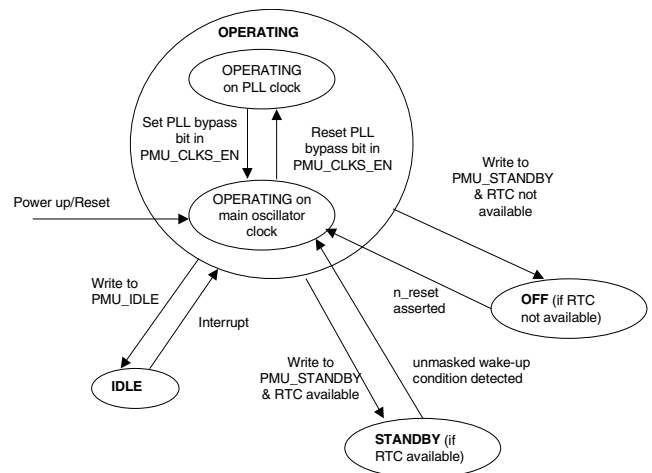


Fig. 2-7: Power Management Unit: Power States

The state of GPIO pins is preserved during OFF/STANDBY. This allows optimization of I/O power consumption with respect to external components (e. g. pull-up/-down resistors).

The contents of SRAM are also preserved during OFF/STANDBY as long, as the device's power supplies remain on.

2.6.1.2. Power Management Register Descriptions

The power management unit is a peripheral on the APB, and thus will enable software configuration of the states as shown in the register map in Table 2–13.

Addresses shown are offset addresses from the PMU base address of 001F.0000_{hex}. For more detailed descriptions of these registers, see Table 2–14.

Table 2–13: Power Management Register Map

Offset	Name	Access	Bits	Description	Default
00 _{hex}	PMU_VCO_DIV	R/W	24:0	Dividers used by the PLL (use with PMU_VCO_DIV_UP)	05E.000B _{hex}
04 _{hex}	PMU_VCO_DIV_UP	W	31:16	Update register for PLL dividers	N/A
08 _{hex}	PMU_HCLK_DIV	R/W	5:0	Divider used for HCLK/PCLK generation	04 _{hex}
0C _{hex}	PMU_CLKOUT_DIV	R/W	4:0	Divider used for CLKOUT generation	11 _{hex}
10 _{hex}	PMU_UARTCLK_DIV	R/W	5:0	Divider used for UARTCLK generation (UART 1–2)	3F _{hex}
14 _{hex}	PMU_SSPCLK_DIV	R/W	5:0	Divider used for SSPCLK generation (SSP 1–5)	3F _{hex}
18 _{hex}	PMU_RTCCLK_DIV	R/W	8:0	Divider used for RTCCLK generation	002 _{hex}
1C _{hex}	PMU_USBCLKS_DIV	R/W	1 bit	Divider used for USB48CLK and USB12CLK generation	1 _{hex}
20 _{hex}	PMU_CLKS_EN	R/W	29:0	Enables for each controllable clock domain	013F.3711 _{hex}
24 _{hex}	PMU_IDLE	W	31:0	Forces the device into IDLE state	N/A
28 _{hex}	PMU_STANDBY	W	31:0	Forces the device into STANDBY or OFF state	N/A
2C _{hex}	PMU_RESET	R/W	31:0	Forces a software reset	unknown ¹⁾
30 _{hex}	PMU_WAKE_SEL	R/W	3:0	Selects which events may cause a wake-up	0 _{hex}
¹⁾ depends on type of last reset					

Table 2–14: Power Management Unit Registers

Register Address	Function	Name										
00 _{hex}	<p>PMU_VCO_DIV Register, Reset 005E.000B_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:25</td> <td>24:16</td> <td>15:5</td> <td>4:0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>NF[8:0]</td> <td>Reserved</td> <td>NR[4:0]</td> </tr> </table> <p>This register stores the values used for the Feedback divider (NF[8:0]) and Input divider (NR[4:0]) for the PLL (note that programmed values need to be 2 less than their corresponding integer values in the quotient that is to be realized by the PLL). When a write is performed to the PMU_VCO_DIV_UP register, the clock generator stops all of the clocks in the system at a low phase, before presenting the divider values from PMU_VCO_DIV to the PLL. The required settling time for the PLL is then waited, before clocks are restarted.</p> <p>For this register to have an effect on the system, the PMU_VCO_DIV_UP register must be written to with the correct key.</p> <p>bit[31:25] Reserved</p> <p>bit[24:16] NF[8:0], PLL feedback divider value</p> <p>bit[15:5] Reserved</p> <p>bit[4:0] NR[4:0], PLL input divider value</p>	Bit	31:25	24:16	15:5	4:0	Field	Reserved	NF[8:0]	Reserved	NR[4:0]	PMU_VCO_DIV
Bit	31:25	24:16	15:5	4:0								
Field	Reserved	NF[8:0]	Reserved	NR[4:0]								
04 _{hex}	<p>PMU_VCO_DIV_UP Register, Reset N/A</p> <table border="1"> <tr> <td>Bit</td> <td>31:16</td> <td>15:0</td> </tr> <tr> <td>Field</td> <td>Update Key</td> <td>Reserved</td> </tr> </table> <p>This register must be written to with the correct key, in order for a new value in the PMU_VCO_DIV register to have an affect on the system operation.</p> <p>The register requires the key in bits [31:16] of the write data.</p> <p>bit[31:16] 4321_{hex} Update Key This half word must be set to 4321_{hex} to force the system to use the new PLL dividers.</p> <p>bit[15:0] Reserved</p>	Bit	31:16	15:0	Field	Update Key	Reserved	PMU_VCO_DIV_UP				
Bit	31:16	15:0										
Field	Update Key	Reserved										
08 _{hex}	<p>PMU_HCLK_DIV Register, Reset 0000.0004_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:6</td> <td>5:0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>HCLK_DIV</td> </tr> </table> <p>This register stores the divider used within the clock generator, for dividing the PLL multiplexed output clock to produce the AHB system clock, HCLK. The divider is also used for PCLK_PMU, PCLK_WDOG, PCLK_APB, and PCLK_RTC.</p> <p>A value of 0 or 1 in this register will stop HCLK.</p> <p>bit[31:16] Reserved</p> <p>bit[5:0] 2_{hex} HCLK_DIV ... AHB system clock divider, range 2 to 63 63_{hex}</p>	Bit	31:6	5:0	Field	Reserved	HCLK_DIV	PMU_HCLK_DIV				
Bit	31:6	5:0										
Field	Reserved	HCLK_DIV										

Table 2–14: Power Management Unit Registers, continued

Register Address	Function	Name								
0C _{hex}	<p>PMU_CLKOUT_DIV Register, Reset 0000.0011_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:5</td> <td>4</td> <td>3:0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>MF_SEL</td> <td>MASF_DIV</td> </tr> </table> <p>This register stores the divider used within the clock generator, for dividing the selected clock to produce the CLKOUT output clock. The source clock for this division is selectable between the reference oscillator clock OSCLK and the PLL multiplexed output clock PLLCLK.</p> <p>A value of 0 in MASF_DIV will stop CLKOUT.</p> <p>bit[31:5] Reserved</p> <p>bit[4] MF_SEL Selects source for CLKOUT divider</p> <p> 1: Always use main oscillator clock (OSCCLK) 0: Use PLL multiplexed output clock (PLLCLK)</p> <p>bit[3:0] 01_{hex} MASF_DIV ... CLKOUT divider, range 1–15_{dec} 0F_{hex}</p>	Bit	31:5	4	3:0	Field	Reserved	MF_SEL	MASF_DIV	PMU_CLKOUT_DIV
Bit	31:5	4	3:0							
Field	Reserved	MF_SEL	MASF_DIV							
10 _{hex}	<p>PMU_UARTCLK_DIV Register, Reset 0000.003F_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:6</td> <td>5:0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>UART_DIV</td> </tr> </table> <p>This register stores the divider used within the clock generator, for dividing the PLL multiplexed output clock to produce the dedicated UART clocks, UART1CLK and UART2CLK.</p> <p>A value of 0 or 1 will stop the UART clocks.</p> <p>The following constraints must be met by the programmer for proper UART operation:</p> <ol style="list-style-type: none"> 1. $UART_DIV \geq \frac{3}{5} HCLK_DIV$ 2. $F_UART(x)CLK \leq 15.0\text{ MHz}$ <p>bit[31:6] Reserved</p> <p>bit[5:0] 2_{hex} UART_DIV ... UART clock divider, range 2–63 63_{hex}</p>	Bit	31:6	5:0	Field	Reserved	UART_DIV	PMU_UARTCLK_DIV		
Bit	31:6	5:0								
Field	Reserved	UART_DIV								

Table 2–14: Power Management Unit Registers, continued

Register Address	Function	Name						
14 _{hex}	<p>PMU_SSPCLK_DIV Register, Reset 0000.003F_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:6</td> <td>5:0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>SSP_DIV</td> </tr> </table> <p>This register stores the divider used within the clock generator, for dividing the PLL multiplexed output clock to produce the dedicated SSP clocks, SSP1CLK, SSP2CLK, SSP3CLK, SSP4CLK, SSP5CLK. Note that if SSP5CLK is selected to be derived from an external source, this divider does not affect the clock used by SSP 5.</p> <p>A value of 0 or 1 will stop the SSP clocks.</p> <p>The following constraints must be met by the programmer for proper SSP operation: SSP_DIV ≥ HCLK_DIV</p> <p>bit[31:6] Reserved</p> <p>bit[5:0] 2_{hex} SSP_DIV ... 63_{hex} SSP clock divider, range 2–63</p>	Bit	31:6	5:0	Field	Reserved	SSP_DIV	PMU_SSPCLK_DIV
Bit	31:6	5:0						
Field	Reserved	SSP_DIV						
18 _{hex}	<p>PMU_RTCCLK_DIV Register, Reset 0000.0002_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:9</td> <td>8:0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>RTC_DIV</td> </tr> </table> <p>This register stores the divider used within the clock generator, for dividing the RTC oscillator clock input to produce the required 1 Hz Real Time Clock counter clock, RTCCLK. The divider circuit consists of a fixed 16384 divider and a programmable divider in the range of 2 to 511. This allows the required 1 Hz to be generated from a range of clock inputs from 32768 Hz up to 8.372224 MHz, in increments of 16384 Hz.</p> <p>A value of 0 or 1 will stop the RTC clock.</p> <p>bit[31:9] Reserved</p> <p>bit[8:0] 2_{hex} RTC_DIV ... 511_{hex} RTC clock divider, range 2–511</p>	Bit	31:9	8:0	Field	Reserved	RTC_DIV	PMU_RTCCLK_DIV
Bit	31:9	8:0						
Field	Reserved	RTC_DIV						

Table 2–14: Power Management Unit Registers, continued

Register Address	Function	Name						
1C _{hex}	<p>PMU_USBCLK_DIV Register, Reset 0000.0001_{hex}</p> <table border="1" data-bbox="288 421 1198 510"> <tr> <td data-bbox="288 421 344 456">Bit</td> <td data-bbox="344 421 1118 456">31:1</td> <td data-bbox="1118 421 1198 456">0</td> </tr> <tr> <td data-bbox="288 456 344 510">Field</td> <td data-bbox="344 456 1118 510">Reserved</td> <td data-bbox="1118 456 1198 510">USBCLK_DIV</td> </tr> </table> <p>When this register is set to 0, the USB48CLK frequency is obtained by dividing PLLCLK by 2, and the USB12CLK frequency is obtained by dividing PLLCLK by 8. Use this configuration for nominal operation of USB (when PLLCLK = 96 MHz) or to access USB core’s SRAM at PLLCLK frequencies less than or equal to 96 MHz.</p> <p>When this register is set to 1, the USB48CLK frequency is obtained by dividing PLLCLK by 4, and the USB12CLK frequency is obtained by dividing PLLCLK by 16. Use this configuration to access the USB core’s SRAM for PLLCLK frequencies above 96 MHz.</p> <p>This register is useful when USB operation is not needed but USB SRAM should stay accessible for system clocks above 48 MHz (PLLCLK above 96 MHz) without violating timing constraints for the USB SRAM.</p> <p>bit[31:1] Reserved</p> <p>bit[0] USBCLK_DIV USB clock divider</p>	Bit	31:1	0	Field	Reserved	USBCLK_DIV	PMU_USBCLK_DIV
Bit	31:1	0						
Field	Reserved	USBCLK_DIV						

Table 2–14: Power Management Unit Registers, continued

Register Address	Function	Name																																																		
20 _{hex}	<p>PMU_CLKS_EN Register, Reset 013F.3711_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:30</td> <td>29</td> <td>28</td> <td>27:25</td> <td>24</td> <td>23:22</td> <td>21</td> <td>20</td> <td>19</td> <td>18</td> <td>17</td> <td>16</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>OSCRTCCLK_EN</td> <td>RTCCLK_EN</td> <td>Reserved</td> <td>CLKOUT_EN</td> <td>Reserved</td> <td>USB_CLK_EN</td> <td>SSP5_CLK_EN</td> <td>SSP4_CLK_EN</td> <td>SSP3_CLK_EN</td> <td>SSP2_CLK_EN</td> <td>SSP1_CLK_EN</td> </tr> </table> <table border="1"> <tr> <td>Bit</td> <td>15:14</td> <td>13</td> <td>12</td> <td>11</td> <td>10</td> <td>9</td> <td>8</td> <td>7:5</td> <td>4</td> <td>3:1</td> <td>0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>UART2_CLK_EN</td> <td>UART1_CLK_EN</td> <td>Reserved</td> <td>PCLK_RTC_EN</td> <td>PCLKW_DOG_EN</td> <td>PCLK_APB_EN</td> <td>Reserved</td> <td>HCLK_EN</td> <td>Reserved</td> <td>VCO_BY_PASS</td> </tr> </table> <p>Enables for each controllable clock domain.</p> <p>bit[31:30] Reserved</p> <p>bit[29] OSCRTCCLK_EN If low the RTC oscillator pad is disabled.</p> <p>bit[28] RTCCLK_EN If low the generated 1-Hz clock enable to the RTC is disabled.</p> <p>bit[27:25] Reserved</p> <p>bit[24] CLKOUT_EN If low the generated CLKOUT output clock is disabled.</p> <p>bit[23:22] Reserved</p> <p>bit[21] USBCLK_EN If low the dedicated clocks (USB12CLK & USB48CLK) to USB are disabled.</p> <p>bit[20] SSP5CLK_EN If low the dedicated clock to SSP 5 is disabled (unless fed by external source).</p> <p>bit[19] SSP4CLK_EN If low the dedicated clock to SSP 4 is disabled.</p> <p>bit[18] SSP3CLK_EN If low the dedicated clock to SSP 3 is disabled.</p> <p>bit[17] SSP2CLK_EN If low the dedicated clock to SSP 2 is disabled.</p> <p>bit[16] SSP1CLK_EN If low the dedicated clock to SSP 1 is disabled.</p> <p>bit[15:14] Reserved</p> <p>bit[13] UART2CLK_EN If low the dedicated clock to UART 2 is disabled.</p> <p>bit[12] UART1CLK_EN If low the dedicated clock to UART 1 is disabled.</p> <p>bit[11] Reserved</p>	Bit	31:30	29	28	27:25	24	23:22	21	20	19	18	17	16	Field	Reserved	OSCRTCCLK_EN	RTCCLK_EN	Reserved	CLKOUT_EN	Reserved	USB_CLK_EN	SSP5_CLK_EN	SSP4_CLK_EN	SSP3_CLK_EN	SSP2_CLK_EN	SSP1_CLK_EN	Bit	15:14	13	12	11	10	9	8	7:5	4	3:1	0	Field	Reserved	UART2_CLK_EN	UART1_CLK_EN	Reserved	PCLK_RTC_EN	PCLKW_DOG_EN	PCLK_APB_EN	Reserved	HCLK_EN	Reserved	VCO_BY_PASS	PMU_CLKS_EN
Bit	31:30	29	28	27:25	24	23:22	21	20	19	18	17	16																																								
Field	Reserved	OSCRTCCLK_EN	RTCCLK_EN	Reserved	CLKOUT_EN	Reserved	USB_CLK_EN	SSP5_CLK_EN	SSP4_CLK_EN	SSP3_CLK_EN	SSP2_CLK_EN	SSP1_CLK_EN																																								
Bit	15:14	13	12	11	10	9	8	7:5	4	3:1	0																																									
Field	Reserved	UART2_CLK_EN	UART1_CLK_EN	Reserved	PCLK_RTC_EN	PCLKW_DOG_EN	PCLK_APB_EN	Reserved	HCLK_EN	Reserved	VCO_BY_PASS																																									

Table 2–14: Power Management Unit Registers, continued

Register Address	Function	Name						
20 _{hex} (con'd)	<p>bit[10] PCLKRTC_EN If low the APB system clock to the RTC is disabled. controls the clock</p> <p>bit[9] PCLKWDOG_EN If low the APB system clock to the watchdog is disabled.</p> <p>bit[8] PCLKAPB_EN If low the APB system clock is disabled to all APB peripherals which do not have dedicated PCLK clocks.</p> <p>bit[7:5] Reserved</p> <p>bit[4] HCLK_EN If low the AHB system clock is disabled.</p> <p>bit[3:1] Reserved</p> <p>bit[0] VCO_BYPASS If set high, the PLL is powered down and bypassed so that OSCCLK feeds the PLL multiplexer output clock, PLLCLK. If low, PLLCLK will be fed by the PLL generated clock, PLL_FOUT.</p>	PMU_CLKS_EN						
24 _{hex}	<p>PMU_IDLE Register, Reset N/A</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 10%;">Bit</td> <td style="width: 45%; text-align: center;">31:16</td> <td style="width: 45%; text-align: center;">15:0</td> </tr> <tr> <td>Field</td> <td style="text-align: center;">IDLE_KEY</td> <td style="text-align: center;">Reserved</td> </tr> </table> <p>A write to this register will force the state of PUC 303xA to change from OPERATING mode into IDLE mode. This will immediately disable both the HCLK and PCLK_WDOG clocks. These will be enabled again when an edge is output on the NIRQ or NFIQ interrupt lines of the ARM7TDMI core by the interrupt controller.</p> <p>There is a delay of at most 4 cycles in writing to the IDLE register and the clock being stopped. For this reason it is recommended that at least 4 NOP instructions are placed directly after the IDLE register write.</p> <p>The ARM7TDMI processor execution will initially carry on from where it left off when an interrupt is received, and then jump to the interrupt handler.</p> <p>The register requires a key in bits [31:16] of the write data.</p> <p>bit[31:16] FEDC IDLE_KEY hex This half word must be set to FEDC_{hex} to force the system into IDLE state.</p> <p>bit[15:0] Reserved</p>	Bit	31:16	15:0	Field	IDLE_KEY	Reserved	PMU_IDLE
Bit	31:16	15:0						
Field	IDLE_KEY	Reserved						

Table 2–14: Power Management Unit Registers, continued

Register Address	Function	Name						
28 _{hex}	<p>PMU_STANDBY Register, Reset N/A</p> <table border="1" data-bbox="288 421 1201 499"> <tr> <td data-bbox="288 421 341 456">Bit</td> <td data-bbox="341 421 770 456">31:16</td> <td data-bbox="770 421 1201 456">15:0</td> </tr> <tr> <td data-bbox="288 456 341 499">Field</td> <td data-bbox="341 456 770 499">STANDBY_KEY</td> <td data-bbox="770 456 1201 499">Reserved</td> </tr> </table> <p>A write to this register, with the correct key, will force the state of PUC 303xA to change from OPERATING state into STANDBY or OFF state, depending on whether the RTC oscillator is active.</p> <p>If the RTC oscillator is inactive, either unavailable, as in the 81-pin BGA package, or not enabled, the OFF state will be entered. Otherwise STANDBY state shall be entered.</p> <p>In either of the new states all clocks in the system, that are not associated with the Real Time Clock, will be stopped in their low phases.</p> <p>Clocks associated with the Real Time Clock (OSCRTCCLK, RTCCLK and PCLK_RTC) will remain active or inactive. The APB clock for the RTC, PCLK_RTC, will then be fed by OSCRTCCLK, instead of being derived from the PLL multiplexed output clock.</p> <p>To leave the OFF state a power-on or warm reset (low on reset pin) is required. This will return the system to OPERATING state driven by the main oscillator clock (PLL bypassed).</p> <p>To leave the STANDBY state, either a power-on or warm reset can be used, or an unmasked wake-up condition (see PMU_WAKE_SEL register) is required. Any of these valid events will result in a wake-up reset sequence, and the device will return to OPERATING state using the main oscillator clock (PLL bypassed).</p> <p>ARM7TDMI processor execution will always begin at the reset vector 0hex after exiting OFF and STANDBY states.</p> <p>bit[31:16] CDEF STANDBY_KEY hex This half word must be set to CDEF_{hex} to force the system into STANDBY state.</p> <p>bit[15:0] Reserved</p>	Bit	31:16	15:0	Field	STANDBY_KEY	Reserved	PMU_STANDBY
Bit	31:16	15:0						
Field	STANDBY_KEY	Reserved						

Table 2–14: Power Management Unit Registers, continued

Register Address	Function	Name												
2C _{hex}	<p>PMU_RESET Register, Reset value: unknown, depends on source of reset</p> <table border="1"> <thead> <tr> <th>Bit</th> <th>31:16</th> <th>15:3</th> <th>2</th> <th>1</th> <th>0</th> </tr> </thead> <tbody> <tr> <th>Field</th> <td>SOFTRESET_KEY</td> <td>Reserved</td> <td>WD_RST</td> <td>POWERON_RST</td> <td>SOFT_RST</td> </tr> </tbody> </table> <p>This register should be read after the end of a reset sequence, to find out the reason for the reset. On a read all of the status bits are cleared.</p> <p>A write to this register can be used to force a software reset from the system reset controller.</p> <p>bit[31:16] ABCD_{hex} SOFTRESET_KEY This field must be set to ABCD_{hex} to allow a write to the register. Access: W</p> <p>bit[15:3] Reserved</p> <p>bit[2] WD_RST This read only bit is set if the latest reset is detected as a watchdog reset. On a read access to the register this bit is cleared. Access: R 1 R: indicates a watchdog reset has occurred, clears bit 0 R: no watchdog reset since last read</p> <p>bit[1] POWERON_RST This read only bit is set if the latest reset is detected as a power-on or warm reset. On a read access to the register this bit is cleared. Access: R 1 R: indicates a power-on or warm reset has occurred, clears bit 0 R: no power-on or warm reset since last read</p> <p>bit[0] SOFT_RST Writing 1 to this bit forces a software reset only if it was previously 0. After a software reset, this bit can be read to indicate that a software reset was performed. It must be cleared by reading the register, before another software reset can be performed. Access: R/W 1 R: indicates a software reset has occurred, clears bit 0 R: no software reset since last read 1 W: forces software reset 0 W: no effect</p>	Bit	31:16	15:3	2	1	0	Field	SOFTRESET_KEY	Reserved	WD_RST	POWERON_RST	SOFT_RST	PMU_RESET
Bit	31:16	15:3	2	1	0									
Field	SOFTRESET_KEY	Reserved	WD_RST	POWERON_RST	SOFT_RST									

Table 2–14: Power Management Unit Registers, continued

Register Address	Function	Name												
30 _{hex}	<p>PMU_WAKE_SEL Register, Reset 0000.0000_{hex}</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 5%;">Bit</td> <td style="width: 60%;">31:4</td> <td style="width: 5%;">3</td> <td style="width: 5%;">2</td> <td style="width: 5%;">1</td> <td style="width: 5%;">0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>USB_SENSE</td> <td>USB_SUP</td> <td>RTC_INT</td> <td>WAKE_PIN</td> </tr> </table> <p>This register is used to select which sources will generate a valid wake-up condition, to force PUC 303xA from STANDBY state back into OPERATING state. These sources are only valid if the RTC oscillator clock input OSCRTCCLK is active. The sources are logically ORed to form the wake-up condition.</p> <p>bit[31:4] Reserved</p> <p>bit[3] 01_{hex} USB_SENSE If set high, the rising edge of the USB sense input is detected as a wake-up.</p> <p>bit[2] 01_{hex} USB_SUP If set high, the falling edge of the USB suspend mode is detected as a wake-up.</p> <p>bit[1] 01_{hex} RTC_INT If set high, the rising edge of the RTC interrupt is detected as a wake-up.</p> <p>bit[0] 01_{hex} WAKE_PIN If set high, the rising edge of the wake-pin input is detected as a wake-up.</p>	Bit	31:4	3	2	1	0	Field	Reserved	USB_SENSE	USB_SUP	RTC_INT	WAKE_PIN	PMU_WAKE_SEL
Bit	31:4	3	2	1	0									
Field	Reserved	USB_SENSE	USB_SUP	RTC_INT	WAKE_PIN									

2.6.2. Interrupt Controller

Since the ARM7TDMI processor core provides only two interrupt input lines, an interrupt controller is used to manage the various on-chip interrupt sources.

The interrupt controller has been designed to optimize the performance of interrupt handling, by reducing latency and providing priority nesting of interrupts.

2.6.2.1. Features

- Up to 32 Interrupt Sources (31 IRQ, 1 FIQ)
- Programmable Selection for FIQ Source
- Hardware Priority Encoding for IRQ
- Selectable Source Type (Hardware / Software)
- All Interrupts Maskable
- Programmable Interrupt Source Type
 - Edge, Triggered
 - Level-Sensitive.

2.6.2.2. Interrupt List

Although there are a possible 32 interrupts, only 25 are directly used by hardware. The remaining 7 can only be used for triggering an interrupt through software. The interrupt sources are listed in Table 2–15.

The interrupt index column is used to decode the IRQSTATUS indexed interrupt status register, and defines the value to write back to the IRQCOMPLETE register.

Table 2–15: Interrupt List

Interrupt number	Interrupt index	Function
0	1	Realtime clock interrupt
1	2	WATCHDOG interrupt
2	3	Timer 1 interrupt
3	4	Timer 2 interrupt
4	5	Timer 3 interrupt
5	6	USB interrupt
6	7	USB SIE interrupt
7	8	I2S (SSP 5) interrupt
8	9	SSP 1 interrupt
9	10	SSP 2 interrupt
10	11	SSP 3 interrupt
11	12	SSP 4 interrupt
12	13	UART 1 interrupt
13	14	UART 2 interrupt
14	15	ARM7TDMI comms rx
15	16	ARM7TDMI comms tx
16	17	GPIO a interrupt
17	18	GPIO b interrupt
18	19	GPIO c interrupt
19	20	GPIO d interrupt
20	21	GPIO e interrupt
21	22	GPIO f interrupt
22	23	GPIO g interrupt
23	24	GPIO h interrupt
24	25	GPIO i interrupt
25 - 31	26-32	Can be used as software interrupts only.

2.6.2.3. General Overview

The Interrupt Controller can process and control up to 32 interrupt sources. Of these, 31 are IRQ style interrupts and the final, programmable source is the Fast Interrupt (FIQ). The FIQ service routine offers a considerable time advantage over the IRQ, due to a combination of reduced software and hardware requirements. The individual sources do not require to be held until the relevant service routine is complete, although this is optional. During the service of one interrupt, it is also possible to store one pending interrupt request from any source, allowing up to 31 interrupt to be pending and one being serviced.

The default IRQ priority encoding is provided by the order of interrupt sources given in the Interrupt List, Table 2–15. The lower the interrupt index in the list, the higher is the so called 'secondary' priority. To modify this default priority ordering, the PUC 303xA allows to apply one of eight levels of 'primary' priority to each of the 32 sources by programming the INTxTYPE registers. Note, however, that these only apply to the IRQ sources and will have no effect on the FIQ. The FIQ is, in any case, granted a higher hardware priority than IRQ anyway. The rules of the priority encoder are as follows:

- Every source can have a priority level between 0 and 7, programmed by software.
- 7 is the highest priority, 0 the lowest. The reset value of the priority is 0 for all sources.
- If, during a particular service routine, a new interrupt source of lower priority is enabled, the source will be stored as pending until the current source is complete.
- If, during a particular service routine, a new interrupt source of higher priority is enabled, the IRQ line will be enabled to permit the ARM7TDMI processor to service the new interrupt.*
- If the Interrupt Controller is free and 2 interrupt sources are enabled simultaneously, the interrupt source with the higher priority is granted. If the sources have equal primary priority (as specified in INTxTYPE register), the interrupt source with lower index in the Interrupt List, Table 2–15, will be granted. Source 1 has, for example, higher secondary priority than source 31.
- If the programmed FIQ source is enabled during an IRQ service routine, the FIQ line will be enabled to permit the ARM7TDMI processor to service the new interrupt.

Any of the 32 Interrupt sources can be programmed to be high or low level sensitive; or rising or falling edge

triggered. This, along with the priority is software controlled through the 32 R/W registers INTxTYPE.

There are multiplexers at the source input to enable each of the 32 sources by software or hardware. At reset, these multiplexers connect all of the sources directly to the hardware interrupts.

The registers IRQSOURCESEL and FIQSOURCESEL implement the select signals for these multiplexers. The R/W registers IRQSWSOURCE and FIQSWSOURCE are used to trigger selected software interrupts.

After source selection, the resulting sources are stored in a read-only register, IRQRAWSTATUS. Depending on the status of the mask register, these are used to generate the IRQPENDING read-only register. This is processed to detect the highest priority IRQ request in the list, and the result passed through to the status registers, IRQSTATUS and IRQSTATUSALL. This, in turn is used to generate the relevant IRQ and FIQ outputs.

The IRQSTATUS register provides an encoded index (see Table 2–15) for only the highest priority interrupt which is generating the IRQ output. This provides an easy method for jumping to the appropriate interrupt handler.

2.6.2.4. Programming Guide

The ARM7TDMI processor has two levels of external interrupt, FIQ and IRQ. For an interrupt to be taken, the disable bit in the CPSR must be clear. The CPSR is written to at the beginning of the program code, using specific instructions to enable the interrupts and set the 'running' mode of the system. The stack pointer for that particular mode must also be set.

The ARM7TDMI processor documentation contains further, more detailed, information on modes and associated operating conditions.

FIQs have higher priority than IRQs in two ways:

1. FIQs are serviced first when multiple interrupts occur. Servicing an FIQ causes IRQs to be disabled, preventing them from being serviced until after the FIQ handler has re-enabled them. This is achieved by restoring the original CPSR from SPSR_FIQ at the end of the handler.

* If nested in this way, the software must use a stack to save the original interrupt position.

2. For higher execution speed, the FIQ vector is the last entry in the vector table, which allows the FIQ handler to be placed directly at the vector location and run sequentially from that address. This removes the need for a branch and associated delays. There are also banked working registers within the ARM7TDMI in FIQ mode avoiding the need for saving a large number of registers on the stack.

The interrupt controller can accommodate up to 32 interrupt sources, 31 of which will be allocated to the IRQ line. This establishes the need for a specific handler capable of interrupt prioritization. The problem is that after a single exception to IRQ, the CPSR will automatically be altered to disable any further IRQs.

The core achieves this by copying the contents of the CPSR into a specific SPSR, in this case SPSR_IRQ. When the routine is finished, the core will then automatically copy the SPSR back into the CPSR, thereby re-enabling the IRQ. A true priority system should permit any new sources of higher priority to interrupt the current exception during the routine.

There is a specific sequence of instructions that must take place in order to ensure that priority encoding is handled correctly. Briefly, this is

1. IRQ exception takes place. Program branches to IRQ handler from main code.
2. Save the link address (R14) to the stack.
3. Get the Current SPSR (the CPSR will have been automatically copied to the SPSR when the exception took place) and store to stack.
4. Note that IRQ interrupts are currently DISABLED.
5. Determine the source of the interrupt: read status in Interrupt Controller. The IRQ line is automatically cleared by reading the status registers. This allows higher priority interrupts to generate a new IRQ.
6. Now read-modify-write the CPSR to ENABLE interrupts.
7. Perform specific Interrupt Routine.
8. Read-modify-write the CPSR to DISABLE interrupts.
9. Write the index of the interrupt handled to the IRQ_COMPLETE register in Interrupt Controller, to inform it that interrupt has been serviced.
10. Restore SPSR from stack.
11. Jump back to main program.

During step 7, any pending new interrupts of higher priority will break the current service and attend to the new interrupt (see Example 2–1).

Example 2–1: SW Instructions, Assembly Code

=====

Normal Interrupt response routine (r7 holds Interrupt Controller Base Address)

=====

IrqHandler ROUT

SUB lr,lr,#4
STMFDP SP!, {lr} ; Store return address

MRS r14, SPSR
STMFDP SP!, {r0-r2,r14} ; Push registers on
 ; stack

LDR r1,[r7,#IRQSTATUS] ; Read status in
 ;integer format
 ;this also clears down
 ;the IRQ line

;
 ; read-modify-write CPSR to enable interrupts
MRS r0, CPSR ; Get CPSR
BIC r0, r0, #0x80 ; Clear the I bit
MSR CPSR_c, r0 ; Write back to enable
 ; Interrupt

=====

Perform particular routine (Source is stored in r1)
Switch to SYS state if lr register is to be used with C-routines.

=====

;
 ; Read-Modify-Write CPSR to disable interrupts
MRS r0, CPSR ; Get CPSR
ORR r0, r0, #0x80 ; Set the I bit
MSR CPSR_c, r0 ; Write back to disable
 ; Interrupt

;
 ; Interrupt END
STR r1,[r7,#IRQ_COMP] ; Write back index of
 ; interrupt just
 ; handled

LDMFD SP!, {r0-r2,r14} ; Pop registers from
 ; stack

MSR SPSR, r14
LDMFD SP!, {pc}^ ; Return address
 ; stored in pc

;*****

2.6.2.5. Interrupt Controller Register Definitions

The Interrupt Controller is programmed through the ARM7TDMI processor APB interface.

Addresses are offsets to the interrupt controller base address, 001C.0000_{hex}.

The registers are fully described in Table 2–17.

Table 2–16: Interrupt Controller Register Map

Offset	R/W	Width	Name	Default
00 _{hex}	R	32	IRQRAWSTATUS	0000.0000 _{hex}
04 _{hex}	R	32	IRQPENDING	0000.0000 _{hex}
08 _{hex}	R	6	IRQSTATUS	0000.0000 _{hex}
0C _{hex}	R	32	IRQSTATUSALL	0000.0000 _{hex}
10 _{hex}	W	32	IRQMASKSET	N/A
14 _{hex}	W	32	IRQMASKCLEAR	N/A
18 _{hex}	R	32	IRQMASK	FFFF.FFFF _{hex}
1C _{hex}	R	1	FIQRAWSTATUS	0000.0000 _{hex}
20 _{hex}	R	1	FIQSTATUS	0000.0000 _{hex}
24 _{hex}	W	1	FIQMASKSET	N/A
28 _{hex}	W	1	FIQMASKCLEAR	N/A
2C _{hex}	R	1	FIQMASK	1 _{hex}
30 _{hex}	W	6	FIQSOURCESEL	00 _{hex}
34 _{hex}	W	32	IRQSOURCESEL	0000.0000 _{hex}
38 _{hex}	R/W	1	FIQSWSOURCE	0 _{hex}
3C _{hex}	R/W	32	IRQSWSOURCE	0000.0000 _{hex}
40 _{hex}	R/W	5	INT0TYPE	02 _{hex}
44 _{hex}	R/W	5	INT1TYPE	02 _{hex}
48 _{hex}	R/W	5	INT2TYPE	02 _{hex}
4C _{hex}	R/W	5	INT3TYPE	02 _{hex}
50 _{hex}	R/W	5	INT4TYPE	02 _{hex}
54 _{hex}	R/W	5	INT5TYPE	02 _{hex}
58 _{hex}	R/W	5	INT6TYPE	02 _{hex}
5C _{hex}	R/W	5	INT7TYPE	02 _{hex}
60 _{hex}	R/W	5	INT8TYPE	02 _{hex}

Table 2–16: Interrupt Controller Register Map, continued

Offset	R/W	Width	Name	Default
64 _{hex}	R/W	5	INT9TYPE	02 _{hex}
68 _{hex}	R/W	5	INT10TYPE	02 _{hex}
6C _{hex}	R/W	5	INT11TYPE	02 _{hex}
70 _{hex}	R/W	5	INT12TYPE	02 _{hex}
74 _{hex}	R/W	5	INT13TYPE	02 _{hex}
78 _{hex}	R/W	5	INT14TYPE	02 _{hex}
7C _{hex}	R/W	5	INT15TYPE	02 _{hex}
80 _{hex}	R/W	5	INT16TYPE	02 _{hex}
84 _{hex}	R/W	5	INT17TYPE	02 _{hex}
88 _{hex}	R/W	5	INT18TYPE	02 _{hex}
8C _{hex}	R/W	5	INT19TYPE	02 _{hex}
90 _{hex}	R/W	5	INT20TYPE	02 _{hex}
94 _{hex}	R/W	5	INT21TYPE	02 _{hex}
98 _{hex}	R/W	5	INT22TYPE	02 _{hex}
9C _{hex}	R/W	5	INT23TYPE	02 _{hex}
A0 _{hex}	R/W	5	INT24TYPE	02 _{hex}
A4 _{hex}	R/W	5	INT25TYPE	02 _{hex}
A8 _{hex}	R/W	5	INT26TYPE	02 _{hex}
AC _{hex}	R/W	5	INT27TYPE	02 _{hex}
B0 _{hex}	R/W	5	INT28TYPE	02 _{hex}
B4 _{hex}	R/W	5	INT29TYPE	02 _{hex}
B8 _{hex}	R/W	5	INT30TYPE	02 _{hex}
BC _{hex}	R/W	5	INT31TYPE	02 _{hex}
C0 _{hex}	W	5	IRQCOMPLETE	N/A
C4 _{hex}	W	1	FIQCOMPLETE	N/A

Important: Ensure that the Interrupt Controllers sources are MASKED before the type registers are altered. **Disregarding this may lead to undefined results.**

Table 2–17: Interrupt Controller Read/Write Registers

Register Address	Function	Name						
00 _{hex}	<p>IRQRAWSTATUS Register, Reset 0000.0000_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:0</td> </tr> <tr> <td>Field</td> <td>Valid Sources</td> </tr> </table> <p>Each interrupt source is validated according to its current interrupt type (see INT*TYPE registers) and the corresponding bit is set if applicable.</p> <p>bit[31:0] Valid Sources</p>	Bit	31:0	Field	Valid Sources	IRQRAWSTATUS		
Bit	31:0							
Field	Valid Sources							
04 _{hex}	<p>IRQPENDING Register, Reset 0000.0000_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:0</td> </tr> <tr> <td>Field</td> <td>Unmasked Valid Sources</td> </tr> </table> <p>Registered logical 'AND' of IRQRAWSTATUS and IRQMASK registers.</p> <p>bit[31:0] Unmasked Valid Sources</p>	Bit	31:0	Field	Unmasked Valid Sources	IRQPENDING		
Bit	31:0							
Field	Unmasked Valid Sources							
08 _{hex}	<p>IRQSTATUS Register, Reset 0000.0000_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:6</td> <td>5:0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>Current IRQ Service Status</td> </tr> </table> <p>This register states as an index which source is currently being serviced. Reading this status register will clear down the IRQ line between the interrupt controller and the ARM7TDMI to allow other higher priority interrupts to generate a new IRQ.</p> <p>bit[31:6] Reserved</p> <p>bit[5:0] 1 Current IRQ Service Status (index between 1 and 32_{dec}) ... 20_{hex}</p>	Bit	31:6	5:0	Field	Reserved	Current IRQ Service Status	IRQSTATUS
Bit	31:6	5:0						
Field	Reserved	Current IRQ Service Status						
0C _{hex}	<p>IRQSTATUSALL Register, Reset 0000.0000_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:0</td> </tr> <tr> <td>Field</td> <td>All Currently Serviced Interrupts</td> </tr> </table> <p>This register shows all interrupts that are either being serviced or have been halted following the arrival of a new exception of higher priority. Reading this status register will clear down the IRQ line between the interrupt controller and the ARM7TDMI to allow other higher priority interrupt to generate an new IRQ.</p> <p>bit[31:0] All Currently Serviced Interrupts</p>	Bit	31:0	Field	All Currently Serviced Interrupts	IRQSTATUSALL		
Bit	31:0							
Field	All Currently Serviced Interrupts							

Table 2–17: Interrupt Controller Read/Write Registers, continued

Register Address	Function	Name						
10 _{hex}	<p>IRQMASKSET Register, Reset N/A</p> <table border="1"> <tr> <td>Bit</td> <td>31:0</td> </tr> <tr> <td>Field</td> <td>Set Mask Bits (disable interrupts)</td> </tr> </table> <p>Set bits in the IRQ interrupt mask to disable the corresponding interrupts. 1: Set mask bit in IRQMASK register 0: No effect.</p> <p>bit[31:0] Set Mask Bits (disable interrupts)</p>	Bit	31:0	Field	Set Mask Bits (disable interrupts)	IRQMASKSET		
Bit	31:0							
Field	Set Mask Bits (disable interrupts)							
14 _{hex}	<p>IRQMASKCLEAR Register, Reset N/A</p> <table border="1"> <tr> <td>Bit</td> <td>31:0</td> </tr> <tr> <td>Field</td> <td>Clear Mask Bits (enable interrupts)</td> </tr> </table> <p>Clear bits in the IRQMASK register to enable the corresponding interrupts 1: Clear mask bit. 0: No effect.</p> <p>bit[31:0] Clear Mask Bits (enable interrupts)</p>	Bit	31:0	Field	Clear Mask Bits (enable interrupts)	IRQMASKCLEAR		
Bit	31:0							
Field	Clear Mask Bits (enable interrupts)							
18 _{hex}	<p>IRQMASK Register, Reset FFFF.FFFF_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:0</td> </tr> <tr> <td>Field</td> <td>IRQ Mask Value</td> </tr> </table> <p>Read only register of the current value of the interrupt mask. The setting and clearing of the IRQMASK register is achieved using IRQMASKSET and IRQMASKCLEAR registers, respectively.</p> <p>bit[31:0] IRQ Mask Value (interrupts are disabled by setting the corresponding bit)</p>	Bit	31:0	Field	IRQ Mask Value	IRQMASK		
Bit	31:0							
Field	IRQ Mask Value							
1C _{hex}	<p>FIQRAWSTATUS Register, Reset 0000.0000_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:1</td> <td>0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>Valid Source</td> </tr> </table> <p>FIQ source is validated according to its current type (see INT*TYPE register).</p> <p>bit[31:1] Reserved bit[0] Valid Source: Set, when source has been validated</p>	Bit	31:1	0	Field	Reserved	Valid Source	FIQRAWSTATUS
Bit	31:1	0						
Field	Reserved	Valid Source						

Table 2–17: Interrupt Controller Read/Write Registers, continued

Register Address	Function	Name						
20 _{hex}	<p>FIQSTATUS Register, Reset 0000.0000_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:1</td> <td>0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>Valid FIQ Source</td> </tr> </table> <p>This register states whether an FIQ interrupt is currently being serviced. Reading this register clears down the FIQ line between interrupt controller and ARM7TDMI.</p> <p>bit[31:1] Reserved</p> <p>bit[0] Valid FIQ Source</p>	Bit	31:1	0	Field	Reserved	Valid FIQ Source	FIQSTATUS
Bit	31:1	0						
Field	Reserved	Valid FIQ Source						
24 _{hex}	<p>FIQMASKSET Register, Reset N/A</p> <table border="1"> <tr> <td>Bit</td> <td>31:1</td> <td>0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>Set FIQMask Bits</td> </tr> </table> <p>Any write to the FIQMASKSET register sets bit[0] in the FIQMASK register.</p> <p>bit[31:1] Reserved</p> <p>bit[0] Set FIQMASK[0]: It is recommended to write this bit as '1'</p>	Bit	31:1	0	Field	Reserved	Set FIQMask Bits	FIQMASKSET
Bit	31:1	0						
Field	Reserved	Set FIQMask Bits						
28 _{hex}	<p>FIQMASKCLEAR Register, Reset N/A</p> <table border="1"> <tr> <td>Bit</td> <td>31:1</td> <td>0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>Clear FIQ-MASK Bits</td> </tr> </table> <p>Any write to the FIQMASKCLEAR register clears bit[0] in the FIQMASK register.</p> <p>bit[31:1] Reserved</p> <p>bit[0] Clear FIQMask Bit[0]</p>	Bit	31:1	0	Field	Reserved	Clear FIQ-MASK Bits	FIQMASKCLEAR
Bit	31:1	0						
Field	Reserved	Clear FIQ-MASK Bits						
2C _{hex}	<p>FIQMASK Register, Reset 1_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:1</td> <td>0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>FIQMASK Value</td> </tr> </table> <p>Read-only register of the current value of the interrupt mask. The setting and clearing of the FIQMASK register is achieved using FIQMASKSET and FIQMASKCLEAR registers, respectively.</p> <p>bit[31:1] Reserved</p> <p>bit[0] FIQ Mask Value (FIQ interrupt is disabled by setting the corresponding bit)</p>	Bit	31:1	0	Field	Reserved	FIQMASK Value	FIQMASK
Bit	31:1	0						
Field	Reserved	FIQMASK Value						

Table 2–17: Interrupt Controller Read/Write Registers, continued

Register Address	Function	Name								
30 _{hex}	<p>FIQSOURCESEL Register, Reset 00_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:6</td> <td>5:1</td> <td>0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>FIQ Source Select</td> <td>SW/HW Select</td> </tr> </table> <p>Hardware/software interrupt source select for FIQ.</p> <p>bit[31:6] Reserved</p> <p>bit[5:1] 0 FIQ Source Select ... 1F_{hex} 5-bit number to determine FIQ interrupt source (number between 0 and 31_{dec})</p> <p>bit[0] SW/HW Select '1': Source is software-driven. '0': Source is hardware-driven.</p>	Bit	31:6	5:1	0	Field	Reserved	FIQ Source Select	SW/HW Select	FIQSOURCESEL
Bit	31:6	5:1	0							
Field	Reserved	FIQ Source Select	SW/HW Select							
34 _{hex}	<p>IRQSOURCESEL Register, Reset 0000.0000_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:0</td> </tr> <tr> <td>Field</td> <td>SW/HW Select</td> </tr> </table> <p>Hardware/software interrupt source select for IRQ.</p> <p>bit[31:0] SW/HW Select '1': IRQ source is software-driven. '0': IRQ source is hardware-driven.</p>	Bit	31:0	Field	SW/HW Select	IRQSOURCESEL				
Bit	31:0									
Field	SW/HW Select									
38 _{hex}	<p>FIQSWSOURCE Register, Reset 0_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:1</td> <td>0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>FIQ SW interrupt source</td> </tr> </table> <p>If selected by FIQSOURCESEL, a write of 1_{hex} to this register acts as the software-driven FIQ interrupt source.</p> <p>bit[31:1] Reserved</p> <p>bit[0] FIQ software interrupt source: Must be written as '1' to cause a software interrupt.</p>	Bit	31:1	0	Field	Reserved	FIQ SW interrupt source	FIQSWSOURCE		
Bit	31:1	0								
Field	Reserved	FIQ SW interrupt source								
3C _{hex}	<p>IRQSWSOURCE Register, Reset 0000.0000_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:0</td> </tr> <tr> <td>Field</td> <td>IRQ software interrupt source</td> </tr> </table> <p>If the corresponding bit is set in IRQSOURCESEL, a write to this register can act as a software interrupt source. For example: if bit 3 of IRQSOURCESEL is set then setting bit 3 of IRQSWSOURCE will generate a Timer 2 interrupt.</p> <p>bit[31:0] IRQ software interrupt source</p>	Bit	31:0	Field	IRQ software interrupt source	IRQSWSOURCE				
Bit	31:0									
Field	IRQ software interrupt source									

Table 2–17: Interrupt Controller Read/Write Registers, continued

Register Address	Function	Name																																				
40 _{hex} ... BC _{hex}	<p>INT*TYPE Register, Reset 02_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:7</td> <td>6:4</td> <td>3:2</td> <td>1:0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>PRIORITY</td> <td>Reserved</td> <td>TYPE</td> </tr> </table> <p>bit[31:7] Reserved</p> <p>bit[6:4] 000_{bin} PRIORITY ... Priority level of interrupt, between 0 and 7 111_{bin} (0 is lowest, 7 is highest).</p> <p>bit[3:2] Reserved</p> <p>bit[1:0] 00_{bin} TYPE ... Type of interrupt: 11_{bin} 00: High Level Detection 01: Low Level Detection 10: Rising Edge Detection 11: Falling Edge Detection</p>	Bit	31:7	6:4	3:2	1:0	Field	Reserved	PRIORITY	Reserved	TYPE	INT*TYPE																										
Bit	31:7	6:4	3:2	1:0																																		
Field	Reserved	PRIORITY	Reserved	TYPE																																		
	<p>Recommended programming of the TYPE field in INTxTYPE registers for the various on-chip interrupt sources:</p> <table border="1"> <thead> <tr> <th>Function (Int Source)</th> <th>Interrupt Index</th> <th>Recommended TYPE field setting in INTxTYPE registers</th> </tr> </thead> <tbody> <tr> <td>Realtime clock</td> <td>1</td> <td>High Level</td> </tr> <tr> <td>Watchdog</td> <td>2</td> <td>High Level</td> </tr> <tr> <td>Timers 1, 2, 3</td> <td>3, 4, 5</td> <td>High Level</td> </tr> <tr> <td>USB</td> <td>6</td> <td>Rising Edge</td> </tr> <tr> <td>USB-SIE</td> <td>7</td> <td>Rising Edge</td> </tr> <tr> <td>SSP5</td> <td>8</td> <td>High Level</td> </tr> <tr> <td>SSP1, 2, 3, 4</td> <td>9, 10, 11, 12</td> <td>High Level</td> </tr> <tr> <td>UART1, 2</td> <td>13,14</td> <td>High Level</td> </tr> <tr> <td>comms rx, tx</td> <td>15,16</td> <td>High Level</td> </tr> <tr> <td>GPIOa, b, c, d, e, f, g, h, i</td> <td>17...25</td> <td>High Level¹⁾</td> </tr> <tr> <td>Software interrupts</td> <td>26...32</td> <td>don't care</td> </tr> </tbody> </table> <p>¹⁾ Note, that this recommendation is independent of the interrupt sensitivity type that is selected for a GPIO pin (see INT_TYPE / INT_VALUE / INT_ON_ANY registers in Section 2.6.8.4. on page 106.)</p>	Function (Int Source)	Interrupt Index	Recommended TYPE field setting in INTxTYPE registers	Realtime clock	1	High Level	Watchdog	2	High Level	Timers 1, 2, 3	3, 4, 5	High Level	USB	6	Rising Edge	USB-SIE	7	Rising Edge	SSP5	8	High Level	SSP1, 2, 3, 4	9, 10, 11, 12	High Level	UART1, 2	13,14	High Level	comms rx, tx	15,16	High Level	GPIOa, b, c, d, e, f, g, h, i	17...25	High Level ¹⁾	Software interrupts	26...32	don't care	
Function (Int Source)	Interrupt Index	Recommended TYPE field setting in INTxTYPE registers																																				
Realtime clock	1	High Level																																				
Watchdog	2	High Level																																				
Timers 1, 2, 3	3, 4, 5	High Level																																				
USB	6	Rising Edge																																				
USB-SIE	7	Rising Edge																																				
SSP5	8	High Level																																				
SSP1, 2, 3, 4	9, 10, 11, 12	High Level																																				
UART1, 2	13,14	High Level																																				
comms rx, tx	15,16	High Level																																				
GPIOa, b, c, d, e, f, g, h, i	17...25	High Level ¹⁾																																				
Software interrupts	26...32	don't care																																				

Table 2–17: Interrupt Controller Read/Write Registers, continued

Register Address	Function	Name						
C0 _{hex}	<p>IRQCOMPLETE Register, Reset N/A</p> <table border="1" data-bbox="288 421 1201 495"> <tr> <td data-bbox="288 421 341 456">Bit</td> <td data-bbox="341 421 986 456">31:6</td> <td data-bbox="986 421 1201 456">5:0</td> </tr> <tr> <td data-bbox="288 456 341 495">Field</td> <td colspan="2" data-bbox="341 456 1201 495">Reserved</td> </tr> </table> <p>Write-back interrupt index, to inform the interrupt controller that the IRQ service is complete. As each interrupt is successfully handled, the corresponding index of that interrupt, as specified in Table 2–15 on page 37 and read from IRQSTATUS, should be written back to this register to enable pending interrupts of lower priority to be serviced next.</p> <p>bit[31:6] Reserved</p> <p>bit[5:0] 01 IRQ Complete on Interrupt: ... Index between 1 and 32_{dec} as previously read from 20_{hex} IRQSTATUS register.</p>	Bit	31:6	5:0	Field	Reserved		IRQCOMPLETE
Bit	31:6	5:0						
Field	Reserved							
C4 _{hex}	<p>FIQCOMPLETE Register, Reset N/A</p> <table border="1" data-bbox="288 918 1201 1010"> <tr> <td data-bbox="288 918 341 954">Bit</td> <td data-bbox="341 918 1115 954">31:1</td> <td data-bbox="1115 918 1201 954">0</td> </tr> <tr> <td data-bbox="288 954 341 1010">Field</td> <td colspan="2" data-bbox="341 954 1201 1010">Reserved</td> </tr> </table> <p>Any write to this register will inform the interrupt controller that FIQ service is complete.</p> <p>bit[31:1] Reserved</p> <p>bit[0] Service Complete: It is recommended to write this bit as '1'</p>	Bit	31:1	0	Field	Reserved		FIQCOMPLETE
Bit	31:1	0						
Field	Reserved							

2.6.3. UARTs

Two independently configurable UART interfaces are provided. Since the UART interfaces' pins are configured as standard GPIO pins after reset, the pins required by the application must be configured for 'Bypass Mode' in the GPIO module (refer to Section 2.6.8.3. on page 106).

2.6.3.1. Features

- Offers similar functionality to the industry standard 16C550 UART.
- Supports bit rates up to 115.2 Kbits/second.
- Separate transmit and receive FIFO, each 16 bytes, with maskable level interrupts. The FIFOs can be disabled.
- Programmable baud rate generator
- Programmable data size, odd or even parity, stop bit size.
- Automatic handling of start, stop and parity bits.
- False start bit detection.
- Line break generation & detection
- Full support of modem handshaking signals
- Includes IrDA SIR interface, supporting data rates up to 115.2 Kbits/second half duplex, with low power mode.

2.6.3.2. UART Functions

The main functional blocks of each of the PUC 303xA's two UARTs are shown in Fig. 2–8.

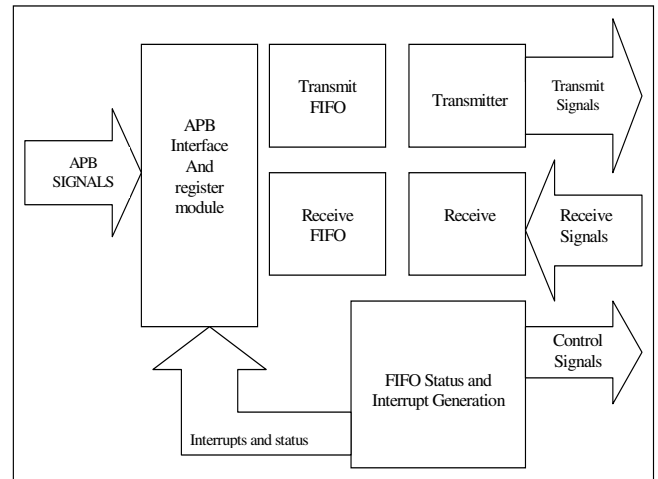


Fig. 2–8: UART Block Diagram

2.6.3.2.1. AMBA APB Interface

The AMBA APB interface generates read and write decodes for access to status and control registers, and transmit and receive FIFO memories.

A Register Block stores data written, or to be read across the AMBA APB interface.

2.6.3.2.2. Baud Rate Generator

The baud rate generator incorporates free-running counters that generate the internal x16 clocks, Baud16, and the IrLPBaud16 signal.

Baud 16 furnishes timing information for the UART transmit and receive control. Baud16 comprises a stream of pulses with a width of one UARTCLK clock period, and a frequency of 16 times the baud rate.

IrLPBaud16 is the source of the timing information that is used to generate the pulse width of the IrDA encoded transmit bit stream, when in the low-power mode.

2.6.3.2.3. Transmit FIFO

The transmit and receive paths are buffered with the internal FIFO's, enabling up to 16 bytes to be stored in both receive and transmit data paths. CPU data written across the APB interface is stored in the FIFO until read out by the transmit logic. The transmit FIFO can be disabled to behave as a one-byte holding register.

2.6.3.2.4. Receive FIFO

The Receive FIFO is a 12-bit wide (4-bit flags + 8-bit data), 16-word depth FIFO memory buffer. The receive logic stores received data, and corresponding error bits, in the receive FIFO until the CPU reads it out across the APB interface. The receive FIFO can be disabled to behave as like a one-byte holding register.

2.6.3.2.5. Transmit Logic

The transmit logic performs parallel-to-serial conversion on the data read from the transmit FIFO. Control logic outputs the serial bit stream beginning with a start bit, data bits (least significant bit (LSB) first) followed by the parity bit, and then stop bits. The precise content of the serial bit stream depends upon the programme configuration in the control registers.

2.6.3.2.6. Receive Logic

The receive logic performs serial-to-parallel conversion on the received bit stream after a valid start pulse has been detected. Overrun, parity, frame error checking, and line break detection are also performed: the data with the associated overrun, parity, framing, and break error bits is written to the receive FIFO.

2.6.3.2.7. Interrupt Generation Logic

The UART generates an interrupt if any of the maskable interrupt conditions are active.

2.6.3.2.8. IrDA Interface

The IrDA interface forms an integral part of both UART modules. The IrDA I/O is multiplexed with the standard UART receive and transmit lines (see Fig. 2-9).

The selection of the multiplexing is under the control of an additional register mapped to the UART address space as shown in Table 2-18.

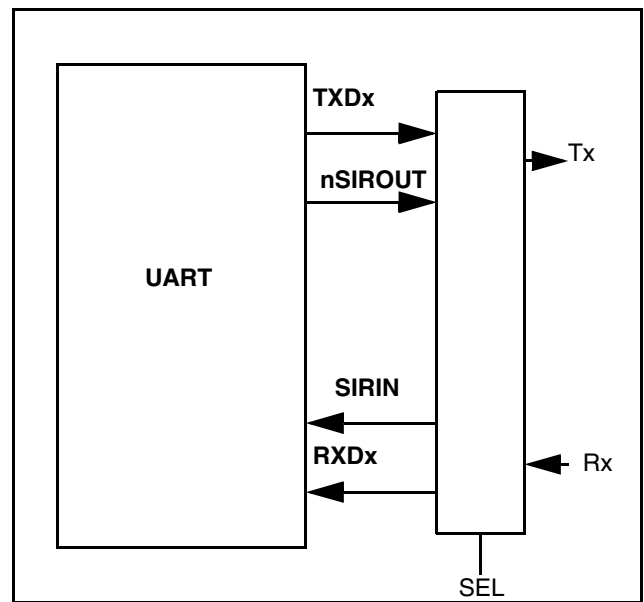


Fig. 2-9: IrDA UART Selection

2.6.3.3. UART Register Map

The locations and brief descriptions of the UART registers are given in Table 2–18.

Addresses shown are relative to the two UART base addresses. UART 1 is located at a base address of 001C.4000_{hex} and UART 2 is located at 001C.8000_{hex}

Table 2–18: UART Register Map¹⁾

Address Offset	Type	Width	Reset Value	Name	Description
00 00 _{hex}	R/W	12/8	--- _{hex}	UARTDR	Data read or written from the interface. It is 12 bits wide on a read, and 8 on a write.
00 04 _{hex}	R/W	4/0	0 _{hex}	UARTRSR/ UARTECR	Receive status register (read)/ Error clear register (write)
00 08 _{hex} - 00 14 _{hex}	-	-	-	-	Reserved
00 18 _{hex}	R	9	-10010--- _{bin}	UARTFR	Flag register (read only)
00 1C _{hex}	-	-	-	-	Reserved
00 20 _{hex}	R/W	8	00 _{hex}	UARTILPR	IrDA low-power counter register
00 24 _{hex}	R/W	16	0000 _{hex}	UARTIBRD	Integer baud rate divider register
00 28 _{hex}	R/W	6	00 _{hex}	UARTFBRD	Reserved, must be written as '0'
00 2C _{hex}	R/W	8	00 _{hex}	UARTLCR_H	Line control register, HIGH byte
00 30 _{hex}	R/W	16	0300 _{hex}	UARTCR	Control register
00 34 _{hex}	R/W	6	12 _{hex}	UARTIFLS	Interrupt FIFO level select register
00 38 _{hex}	R/W	11	000 _{hex}	UARTIMSC	Interrupt mask set/clear
00 3C _{hex}	R/W	11	00 _{hex}	UARTRIS	Raw interrupt status
00 40 _{hex}	R/W	11	00 _{hex}	UARTMIS	Masked interrupt status
00 44 _{hex}	W	11	N/A	UARTICR	Interrupt clear register
00 48 _{hex} - 0F FC _{hex}	-	-	-	-	Reserved
10 00 _{hex}	R/W	1	01 _{hex}	UARTMuxSel	IrDA/UART selection register

1) “---” = undefined digits after reset

Table 2–19: UART Read/Write Registers

Register Address	Function	Name														
00 00 _{hex}	<p>UARTDR Register, Reset ---_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>15:12</td> <td>11</td> <td>10</td> <td>9</td> <td>8</td> <td>7:0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>OE</td> <td>BE</td> <td>PE</td> <td>FE</td> <td>DATA</td> </tr> </table> <p>Data read or written from the interface. It is 12 bits wide on a read, and 8 bits wide on a write.</p> <p>bit[15:12] Reserved</p> <p>bit[11] Overrun Error (OE) This bit is set to 1 if data is received and the receive FIFO is already full. This is cleared to 0 once there is an empty space in the FIFO and a new character can be written to it.</p> <p>bit[10] Break Error (BE) This bit is set to 1 if a break condition was detected, indicating that the received data input was held LOW for longer than a full-word transmission time (defined as start, data, parity and stop bits). In FIFO mode, this error is associated with the character at the top of the FIFO. When a break occurs, only one 0 character is loaded into the FIFO. The next character is only enabled after the receive data input goes to a 1 (marking state), and the next valid start bit is received.</p> <p>bit[9] Parity Error (PE) When this bit is set to 1, it indicates that the parity of the received data character does not match the parity selected as defined by bits 2 and 7 of the UARTLCR_H register. In FIFO mode, this error is associated with the character at the top of the FIFO.</p> <p>bit[8] Framing Error (FE) When this bit is set to 1, it indicates that the received character did not have a valid stop bit (a valid stop bit is 1). In FIFO mode, this error is associated with the character at the top of the FIFO.</p> <p>bit[7:0] DATA Receive (read) data character / Transmit (write) character</p>	Bit	15:12	11	10	9	8	7:0	Field	Reserved	OE	BE	PE	FE	DATA	UARTDR
Bit	15:12	11	10	9	8	7:0										
Field	Reserved	OE	BE	PE	FE	DATA										

Table 2–19: UART Read/Write Registers, continued

Register Address	Function	Name												
00 04 _{hex}	<p>Combined UARTRSR/UARTECR Register, Reset 0_{hex}</p> <table border="1" data-bbox="288 421 772 499"> <tr> <td>Bit</td> <td>7:4</td> <td>3</td> <td>2</td> <td>1</td> <td>0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>OE</td> <td>BE</td> <td>PE</td> <td>FE</td> </tr> </table> <p>Receive status register (read) / Error clear register (write). A write to this register clears the framing-, parity-, break-, and overrun errors. The data value is not important.</p> <p>bit[7:4] Reserved, unpredictable when read.</p> <p>bit[3] Overrun Error (OE) This bit is set to 1 if data is received and the FIFO is already full.</p> <p>bit[2] Break Error (BE) This bit is set to 1 if a break condition was detected, indicating that the received data input was held LOW for longer than a full-word transmission time (defined as start, data, parity, and stop bits).</p> <p>bit[1] Parity Error (PE) When this bit is set to 1, it indicates that the parity of the received data character does not match the parity selected as defined by bits 2 and 7 of the UARTLCR_H register.</p> <p>bit[0] Framing Error (FE) When this bit is set to 1, it indicates that the received character did not have a valid stop bit (a valid stop bit is 1). This bit is cleared to 0 by a write to UARTECR.</p>	Bit	7:4	3	2	1	0	Field	Reserved	OE	BE	PE	FE	UARTRSR UARTECR
Bit	7:4	3	2	1	0									
Field	Reserved	OE	BE	PE	FE									

Table 2–19: UART Read/Write Registers, continued

Register Address	Function	Name																						
00 18 _{hex}	<p>UARTFR Register, Reset -10010---bin</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <th style="width: 10%;">Bit</th> <th style="width: 10%;">15:9</th> <th style="width: 5%;">8</th> <th style="width: 5%;">7</th> <th style="width: 5%;">6</th> <th style="width: 5%;">5</th> <th style="width: 5%;">4</th> <th style="width: 5%;">3</th> <th style="width: 5%;">2</th> <th style="width: 5%;">1</th> <th style="width: 5%;">0</th> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>RI</td> <td>TXFE</td> <td>RXFF</td> <td>TXFF</td> <td>RXFE</td> <td>BUSY</td> <td>DCD</td> <td>DSR</td> <td>CTS</td> </tr> </table> <p>Flag register (read only).</p> <p>bit[15:9] Reserved</p> <p>bit[8] Ring Indicator (RI) This bit is the complement of the UART ring indicator (nRIx) modem status input. The bit is 1 when the modem status input is 0.</p> <p>bit[7] Transmit FIFO Empty (TXFE) The meaning of this bit depends on the state of the FEN bit in the UARTLCR_H register. If the FIFO is disabled, this bit is set when the transmit holding register is empty. If the FIFO is enabled, this bit is set when the transmit FIFO is empty.</p> <p>bit[6] Receive FIFO Full (RXFF) The meaning of this bit depends on the state of the FEN bit in the UARTLCR_H register. If the FIFO is disabled, this bit is set when the receive holding register is full. If the FIFO is enabled, this bit is set when the receive FIFO is full.</p> <p>bit[5] Transmit FIFO Full (TXFF) The meaning of this bit depends on the state of the FEN bit in the UARTLCR_H register. If the FIFO is disabled, this bit is set when the transmit holding register is full. If the FIFO is enabled, this bit is set when the transmit FIFO is full.</p> <p>bit[4] Receive FIFO Empty (RXFE) The meaning of this bit depends on the state of the FEN bit in the UARTLCR_H register. If the FIFO is disabled, this bit is set when the receive holding register is empty. If the FIFO is enabled, this bit is set when the receive FIFO is empty.</p> <p>bit[3] UART Busy (BUSY) If this bit is set to 1, the UART is busy transmitting data. This bit remains set until the complete byte, including all stop bits, has been sent from the shift register. This bit is set as soon as the transmit FIFO becomes non-empty, whether the UART is enabled or not.</p> <p>bit[2] Data Carrier Detect (DCD) This bit is the complement of the UART data carrier detect (nDCDx) modem status input. The bit is 1 when the modem status input is 0.</p> <p>bit[1] Data Set Ready (DSR) UART data set ready (nDSRx) modem status input. The bit is 1 when the modem status input is 0.</p> <p>bit[0] Clear to Send (CTS) UART clear to send (nCTSx) modem status input. The bit is 1 when the modem status input is 0.</p>	Bit	15:9	8	7	6	5	4	3	2	1	0	Field	Reserved	RI	TXFE	RXFF	TXFF	RXFE	BUSY	DCD	DSR	CTS	UARTFR
Bit	15:9	8	7	6	5	4	3	2	1	0														
Field	Reserved	RI	TXFE	RXFF	TXFF	RXFE	BUSY	DCD	DSR	CTS														

Table 2–19: UART Read/Write Registers, continued

Register Address	Function	Name				
00 20 _{hex}	<p>UARTILPR Register, Reset 00_{hex}</p> <table border="1" data-bbox="288 421 772 499"> <tr> <td>Bit</td> <td>7:0</td> </tr> <tr> <td>Field</td> <td>ILPDVSR</td> </tr> </table> <p>IrDA low-power counter register.</p> <p>The IrLPBaud16 signal is generated by dividing down the UARTCLK signal according to the low-power divider value written to UARTILPR.</p> <p>The low-power divider value is calculated as follows: low-power divider (ILPDVSR) = ($F_{\text{UARTCLK}} / F_{\text{IrLPBaud16}}$) where $F_{\text{IrLPBaud16}}$ is nominally 1.8432MHz.</p> <p>Choose the divider so that $1.42\text{MHz} < F_{\text{IrLPBaud16}} < 2.12\text{MHz}$, that results in a low-power pulse duration of 1.41–2.11µs (three times the period of IrLPBaud16).</p> <p>The minimum frequency of IrLPBaud16 ensures that pulses less than one period of IrLPBaud16 are rejected, but that pulses greater than 1.4µs are accepted as valid pulses.</p> <p>Note, that programming a zero value results in no IrLPBaud16 pulses being generated.</p> <p>bit[7:0] 00_{hex} ILPDVSR ... FF_{hex} 8-bit low-power divider value. These bits are cleared to 0 at reset.</p>	Bit	7:0	Field	ILPDVSR	UARTILPR
Bit	7:0					
Field	ILPDVSR					
00 24 _{hex}	<p>UARTIBRD Integer Baud Rate Register, Reset 0000_{hex}</p> <table border="1" data-bbox="288 1146 1201 1225"> <tr> <td>Bit</td> <td>15:0</td> </tr> <tr> <td>Field</td> <td>BAUD DIVINT</td> </tr> </table> <p>Integer baud rate divider register. See Section 2.6.3.3.1. on page 64 and Section 2.6.3.3.2. on page 64 for details.</p> <p>bit[15:0] 01_{hex} BAUD DIVINT ... FFFF_{hex} Integer baud rate divider value. These bits are cleared to 0 at reset.</p>	Bit	15:0	Field	BAUD DIVINT	UARTIBRD
Bit	15:0					
Field	BAUD DIVINT					

Table 2–19: UART Read/Write Registers, continued

Register Address	Function	Name																														
00 2C _{hex}	<p>UARTLCR_H Register, Reset 00_{hex}</p> <table border="1"> <tr> <th>Bit</th> <td>15:8</td> <td>7</td> <td>6:5</td> <td>4</td> <td>3</td> <td>2</td> <td>1</td> <td>0</td> </tr> <tr> <th>Field</th> <td>Reserved</td> <td>SPS</td> <td>WLEN</td> <td>FEN</td> <td>STP2</td> <td>EPS</td> <td>PEN</td> <td>BRK</td> </tr> </table> <p>Line control register, HIGH byte. See Section 2.6.3.3.2. on page 64 for details.</p> <p>bit[15:9] Reserved</p> <p>bit[7] Stick Parity Select (SPS) When bits 1,2 and 7 of the UARTLCR_H register are set, the parity bit is transmitted and checked as a 0. When bits 1 and 7 are set, and bit 2 is 0, the parity bit is transmitted and checked as a 1. When this bit is cleared stick parity is disabled.</p> <p>bit[6:5] <table border="0"> <tr> <td>11_{hex}</td> <td>Word length [1:0] (WLEN)</td> </tr> <tr> <td>10_{hex}</td> <td>The select bits indicate the number of data bits transmitted or received in a frame as follows:</td> </tr> <tr> <td>01_{hex}</td> <td>11 = 8 bits</td> </tr> <tr> <td>00_{hex}</td> <td>10 = 7 bits</td> </tr> <tr> <td></td> <td>01 = 6 bits</td> </tr> <tr> <td></td> <td>00 = 5 bits.</td> </tr> </table> </p> <p>bit[4] Enable FIFOs (FEN) If this bit is set to 1, transmit and receive FIFO buffers are enabled (FIFO mode). When cleared to 0 the FIFOs are disabled (character mode) that is, the FIFOs become 1-byte-deep holding registers.</p> <p>bit[3] Two Stop Bits Select (STP2) If this bit is set to 1, two stop bits are transmitted at the end of the frame. The receive logic does not check for two stop bits being received.</p> <p>bit[2] Even Parity Select (EPS) If this bit is set to 1, even parity generation and checking is performed during transmission and reception, which checks for an even number of 1s in data and parity bits. When cleared to 0 then odd parity is performed which checks for an odd number of 1s. This bit has no effect when parity is disabled by Parity Enable (bit 1) being cleared to 0.</p> <p>bit[1] Parity Enable (PEN) If this bit is set to 1, parity checking and generation is enabled, else parity is disabled and no parity bit added to the data frame.</p> <p>bit[0] Send Break (BRK) If this bit is set to 1, a low-level is continually output on the TXDx output, after completing transmission of the current character. This bit must be asserted for at least one complete frame transmission time in order to generate a break condition. The transmit FIFO contents remain unaffected during a break condition. For normal use, this bit must be cleared to 0.</p>	Bit	15:8	7	6:5	4	3	2	1	0	Field	Reserved	SPS	WLEN	FEN	STP2	EPS	PEN	BRK	11 _{hex}	Word length [1:0] (WLEN)	10 _{hex}	The select bits indicate the number of data bits transmitted or received in a frame as follows:	01 _{hex}	11 = 8 bits	00 _{hex}	10 = 7 bits		01 = 6 bits		00 = 5 bits.	UARTLCR_H
Bit	15:8	7	6:5	4	3	2	1	0																								
Field	Reserved	SPS	WLEN	FEN	STP2	EPS	PEN	BRK																								
11 _{hex}	Word length [1:0] (WLEN)																															
10 _{hex}	The select bits indicate the number of data bits transmitted or received in a frame as follows:																															
01 _{hex}	11 = 8 bits																															
00 _{hex}	10 = 7 bits																															
	01 = 6 bits																															
	00 = 5 bits.																															

Table 2–19: UART Read/Write Registers, continued

Register Address	Function	Name																																		
00 30 _{hex}	<p>UARTCR Register, Reset 0300_{hex}</p> <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <tr> <th>Bit</th> <th>15</th> <th>14</th> <th>13</th> <th>12</th> <th>11</th> <th>10</th> <th>9</th> <th>8</th> <th>7</th> <th>6</th> <th>5</th> <th>4</th> <th>3</th> <th>2</th> <th>1</th> <th>0</th> </tr> <tr> <td>Field</td> <td>CTSEn</td> <td>RTSEn</td> <td>Re-served</td> <td>Re-served</td> <td>RTS</td> <td>DTR</td> <td>RXE</td> <td>TXE</td> <td>LBE</td> <td colspan="4">Reserved</td> <td>SIRLP</td> <td>SIREN</td> <td>UART EN</td> </tr> </table> <p>The UARTCR register is the control register. All bits are cleared to 0 on reset, except for bits 9 and 8 which are set to 1. The table below shows the bit assignment of the UARTCR register</p> <p>bit[15] CTS Hardware Flow Control Enable (CTSEn) If this bit is set to 1, CTS hardware flow control is enabled. Data is only transmitted when the nCTSx signal is asserted.</p> <p>bit[14] RTS Hardware Flow Control Enable (RTSEn) If this bit is set to 1, RTS hardware flow control is enabled. Data is only requested when there is space in the receive FIFO for it to be received.</p> <p>bit[13] Reserved</p> <p>bit[12] Reserved</p> <p>bit[11] Request to Send (RTS) This bit is the complement of the UART request to send (nRTSx) modem status output. That is, when the bit is programmed to a 1, the output is 0.</p> <p>bit[10] Data Transmit Ready (DTR) This bit is the complement of the UART data transmit ready (nDTRx) modem status output. That is, when the bit is programmed to a 1, the output is 0.</p> <p>bit[9] Receive Enable (RXE) If this bit is set to 1, the receive section of the UART is enabled. Data reception occurs for either UART signals or SIR signals according to the setting of SIR Enable (bit 1). When the UART is disabled in the middle of reception, it completes the current character before stopping.</p> <p>bit[8] Transmit Enable (TXE) If this bit is set to 1, the transmit section of the UART is enabled. Data transmission occurs for either UART signals, or SIR signals according to the setting of SIR Enable (bit 1). When the UART is disabled in the middle of transmission, it completes the current character before stopping.</p>	Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Field	CTSEn	RTSEn	Re-served	Re-served	RTS	DTR	RXE	TXE	LBE	Reserved				SIRLP	SIREN	UART EN	UARTCR
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																				
Field	CTSEn	RTSEn	Re-served	Re-served	RTS	DTR	RXE	TXE	LBE	Reserved				SIRLP	SIREN	UART EN																				

Table 2–19: UART Read/Write Registers, continued

Register Address	Function	Name
00 30 _{hex} (con'd)	<p>bit[7] Loop Back Enable (LBE) If this bit is set to 1 and the SIR Enable bit is set to 1 and the test register UARTTCR bit 2 (SIRTEST) is set to 1, then the nSIROUT path is inverted, and fed through to the SIRIN path. The SIRTEST bit in the test register must be set to 1 to override the normal half-duplex SIR operation. This must be the requirement for accessing the test registers during normal operation, and SIRTEST must be cleared to 0 when loopback testing is finished. This feature reduces the amount of external coupling required during system test.</p> <p>If this bit is set to 1, and the SIRTEST bit is set to 0, the TXDx path is fed through to the RXDx path.</p> <p>In either SIR mode or normal mode, when this bit is set, the modem outputs are also fed through to the modem inputs.</p> <p>This bit is cleared to 0 on reset, which disables the loop-back mode.</p> <p>bit[6:3] Reserved</p> <p>bit[2] IrDA SIR Low Power Mode (SIRLP) This bit selects the IrDA encoding mode. If this bit is cleared to 0, low-level bits are transmitted as an active high pulse with a width of $\frac{3}{16}$th of the bit period. If this bit is set to 1, low-level bits are transmitted with a pulse width which is 3 times the period of the IrLPBaud16 input signal, regardless of the selected bit rate. Setting this bit uses less power, but might reduce transmission distances.</p> <p>bit[1] SIR Enable (SIREN) If this bit is set to 1, the IrDA SIR ENDEC is enabled. This bit has no effect if the UART is not enabled by bit 0 being set to 1.</p> <p>When the IrDA SIR ENDEC is enabled, data is transmitted and received on nSIROUT and SIRIN. TXDx remains in the marking state (set to 1). Signal transitions on RXDx or modem status inputs have no effect.</p> <p>When the IrDA SIR ENDEC is disabled, nSIROUT remains cleared to 0 (no light pulse generated), and signal transitions on SIRIN have no effect</p> <p>bit[0] UART Enable (UARTEN) If this bit is set to 1, the UART is enabled. Data transmission and reception occurs for either UART signals or SIR signals according to the setting of SIR Enable (bit 1). When the UART is disabled in the middle of transmission or reception, it completes the current character before stopping.</p>	UARTCR

Table 2–19: UART Read/Write Registers, continued

Register Address	Function	Name								
00 34 _{hex}	<p>UARTIFLS Register, Reset 12_{hex}</p> <table border="1" data-bbox="288 421 1201 497"> <tr> <td data-bbox="288 421 341 456">Bit</td> <td data-bbox="341 421 879 456">15:6</td> <td data-bbox="879 421 1042 456">5:3</td> <td data-bbox="1042 421 1201 456">2:0</td> </tr> <tr> <td data-bbox="288 456 341 497">Field</td> <td data-bbox="341 456 879 497">Reserved</td> <td data-bbox="879 456 1042 497">RXIFLSEL</td> <td data-bbox="1042 456 1201 497">TXIFLSEL</td> </tr> </table> <p>The UARTIFLS register is the interrupt FIFO level select register. The UARTIFLS register can be used to define the FIFO level at which the UARTTXINTR and UARTRXINTR are triggered.</p> <p>The interrupts are generated based on a transition through a level rather than being based on the level. That is, the design is such that the interrupts are generated when the fill level progresses through the trigger level.</p> <p>The bits are reset so that the trigger level is when the FIFOs are at the half-way mark.</p> <p>bit[15:6] Reserved</p> <p>bit[5:3] 000_{bin} RXIFLSEL 001_{bin} Receive Interrupt FIFO Level Select (RXIFLSEL) 010_{bin} The trigger points for the receive interrupt are as follows: 011_{bin} 000 = Receive FIFO becomes >= 1/8 full 100_{bin} 001 = Receive FIFO becomes >= 1/4 full 010 = Receive FIFO becomes >= 1/2 full 011 = Receive FIFO becomes >= 3/4 full 100 = Receive FIFO becomes >= 7/8 full 101:111 = reserved.</p> <p>bit[2:0] 000_{bin} TXIFLSEL 001_{bin} Transmit Interrupt FIFO Level Select (TXIFLSEL) 010_{bin} The trigger points for the transmit interrupt are as follows: 011_{bin} 000 = Transmit FIFO becomes <= 1/8 full 100_{bin} 001 = Transmit FIFO becomes <= 1/4 full 010 = Transmit FIFO becomes <= 1/2 full 011 = Transmit FIFO becomes <= 3/4 full 100 = Transmit FIFO becomes <= 7/8 full 101:111 = reserved.</p>	Bit	15:6	5:3	2:0	Field	Reserved	RXIFLSEL	TXIFLSEL	UARTIFLS
Bit	15:6	5:3	2:0							
Field	Reserved	RXIFLSEL	TXIFLSEL							

Table 2–19: UART Read/Write Registers, continued

Register Address	Function	Name																										
00 38 _{hex}	<p>UARTIMSC Register, Reset 000_{hex}</p> <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <tr> <td style="width: 5%;">Bit</td> <td style="width: 15%;">15:11</td> <td style="width: 5%;">10</td> <td style="width: 5%;">9</td> <td style="width: 5%;">8</td> <td style="width: 5%;">7</td> <td style="width: 5%;">6</td> <td style="width: 5%;">5</td> <td style="width: 5%;">4</td> <td style="width: 5%;">3</td> <td style="width: 5%;">2</td> <td style="width: 5%;">1</td> <td style="width: 5%;">0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>OEIM</td> <td>BEIM</td> <td>PEIM</td> <td>FEIM</td> <td>RTIM</td> <td>TXIM</td> <td>RXIM</td> <td>DSRMI M</td> <td>DCD- MIM</td> <td>CTSMI M</td> <td>RIMIM</td> </tr> </table> <p>The UARTIMSC register is the interrupt mask set/clear register. It is a read/write register. On a read this register gives the current value of the mask on the relevant interrupt. On a write of 1 to the particular bit, it sets the corresponding mask of that interrupt. A write of 0 clears the corresponding mask.</p> <p>All of the bits are cleared to 0 when reset.</p> <p>bit[15:11] Reserved</p> <p>bit[10] Overrun Error Interrupt Mask (OEIM) On a read, the current mask for the OEIM interrupt is returned. On a write of 1, the mask of the OEIM interrupt is set. A write of 0 clears the mask.</p> <p>bit[9] Break Error Interrupt Mask (BEIM) On a read the current mask for the BEIM interrupt is returned. On a write of 1, the mask of the BEIM interrupt is set. A write of 0 clears the mask.</p> <p>bit[8] Parity Error Interrupt Mask (PEIM) On a read the current mask for the PEIM interrupt is returned. On a write of 1, the mask of the PEIM interrupt is set. A write of 0 clears the mask.</p> <p>bit[7] Framing Error Interrupt Mask (FEIM) On a read the current mask for the FEIM interrupt is returned. On a write of 1, the mask of the FEIM interrupt is set. A write of 0 clears the mask.</p> <p>bit[6] Receive Timeout Interrupt Mask (RTIM) On a read the current mask for the RTIM interrupt is returned. On a write of 1, the mask of the RTIM interrupt is set. A write of 0 clears the mask.</p> <p>bit[5] Transmit Interrupt Mask (TXIM) On a read the current mask for the TXIM interrupt is returned. On a write of 1, the mask of the TXIM interrupt is set. A write of 0 clears the mask.</p> <p>bit[4] Receive Interrupt Mask (RXIM) On a read the current mask for the RXIM interrupt is returned. On a write of 1, the mask of the RXIM interrupt is set. A write of 0 clears the mask.</p> <p>bit[3] nDSRx Modem Interrupt Mask (DSRMIM) On a read the current mask for the DSRMIM interrupt is returned. On a write of 1, the mask of the DSRMIM interrupt is set. A write of 0 clears the mask.</p>	Bit	15:11	10	9	8	7	6	5	4	3	2	1	0	Field	Reserved	OEIM	BEIM	PEIM	FEIM	RTIM	TXIM	RXIM	DSRMI M	DCD- MIM	CTSMI M	RIMIM	UARTIMSC
Bit	15:11	10	9	8	7	6	5	4	3	2	1	0																
Field	Reserved	OEIM	BEIM	PEIM	FEIM	RTIM	TXIM	RXIM	DSRMI M	DCD- MIM	CTSMI M	RIMIM																

Table 2–19: UART Read/Write Registers, continued

Register Address	Function	Name
00 38 _{hex} (con'd)	<p>bit[2] nDCDx Modem Interrupt Mask (DCDMIM) On a read the current mask for the DCDMIM interrupt is returned. On a write of 1, the mask of the DCDMIM interrupt is set. A write of 0 clears the mask.</p> <p>bit[1] nCTSx Modem Interrupt Mask (CTSMIM) On a read the current mask for the CTSMIM interrupt is returned. On a write of 1, the mask of the CTSMIM interrupt is set. A write of 0 clears the mask.</p> <p>bit[0] nRlx Modem Interrupt Mask (RIMIM) On a read the current mask for the RIMIM interrupt is returned. On a write of 1, the mask of the RIMIM interrupt is set. A write of 0 clears the mask.</p>	UARTIMSC

Table 2–19: UART Read/Write Registers, continued

Register Address	Function	Name																										
00 3C _{hex}	<p>UARTRIS Register, Reset 000_{hex}</p> <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <tr> <td style="width: 5%;">Bit</td> <td style="width: 15%;">15:11</td> <td style="width: 5%;">10</td> <td style="width: 5%;">9</td> <td style="width: 5%;">8</td> <td style="width: 5%;">7</td> <td style="width: 5%;">6</td> <td style="width: 5%;">5</td> <td style="width: 5%;">4</td> <td style="width: 5%;">3</td> <td style="width: 5%;">2</td> <td style="width: 5%;">1</td> <td style="width: 5%;">0</td> </tr> <tr> <td style="background-color: #e0e0e0;">Field</td> <td style="background-color: #e0e0e0;">Reserved</td> <td>OERIS</td> <td>BERIS</td> <td>PERIS</td> <td>FERIS</td> <td>RTRIS</td> <td>TXRIS</td> <td>RXRIS</td> <td>DSR RMIS</td> <td>DCD RMIS</td> <td>CTS RMIS</td> <td>RIR MIS</td> </tr> </table> <p>The UARTRIS register is the raw interrupt status register. It is a read-only register. On a read this register gives the current raw status value of the corresponding interrupt. A write has no effect.</p> <p>All the bits, are cleared to 0 when reset, except for the modem status interrupt bits, 3 to 0. The modem status interrupt bits are undefined after reset.</p> <p>bit[15:11] Reserved</p> <p>bit[10] Overrun Error Interrupt Status (OERIS) Gives the raw interrupt state (prior to masking) of the UARTOEINTR interrupt</p> <p>bit[9] Break Error Interrupt Status (BERIS) Gives the raw interrupt state (prior to masking) of the UARTBEINTR interrupt</p> <p>bit[8] Parity Error Interrupt Status (PERIS) Gives the raw interrupt state (prior to masking) of the UARTPEINTR interrupt</p> <p>bit[7] Framing Error Interrupt Status (FERIS) Gives the raw interrupt state (prior to masking) of the UARTFEINTR interrupt</p> <p>bit[6] Receive Timeout Interrupt Status (RTRIS) Gives the raw interrupt state (prior to masking) of the UARTRTINTR interrupt. In this case, the raw interrupt cannot be set unless the mask is set, this is because the mask acts as an enable for power saving. That is, the same status can be read from UARTRIS and UARTRIS for the receive timeout interrupt.</p> <p>bit[5] Transmit Interrupt Status (TXRIS) Gives the raw interrupt state (prior to masking) of the UARTRXINTR interrupt</p> <p>bit[4] Receive Interrupt Status (RXRIS) Gives the raw interrupt state (prior to masking) of the UARTRXINTR interrupt</p> <p>bit[3] nDSRx Modem Interrupt Status (DSRRMIS) Gives the raw interrupt state (prior to masking) of the UARTDSRINTR interrupt</p> <p>bit[2] nDCDx Modem Interrupt Status (DCDRMIS) Gives the raw interrupt state (prior to masking) of the UARTDCDINTR interrupt</p> <p>bit[1] nCTSx Modem Interrupt Status (CTSRMIS) Gives the raw interrupt state (prior to masking) of the UARTCTSINTR interrupt</p> <p>bit[0] nRlx Modem Interrupt Status (RIRMIS) Gives the raw interrupt state (prior to masking) of the UARTRIINTR interrupt</p>	Bit	15:11	10	9	8	7	6	5	4	3	2	1	0	Field	Reserved	OERIS	BERIS	PERIS	FERIS	RTRIS	TXRIS	RXRIS	DSR RMIS	DCD RMIS	CTS RMIS	RIR MIS	UARTRIS
Bit	15:11	10	9	8	7	6	5	4	3	2	1	0																
Field	Reserved	OERIS	BERIS	PERIS	FERIS	RTRIS	TXRIS	RXRIS	DSR RMIS	DCD RMIS	CTS RMIS	RIR MIS																

Table 2–19: UART Read/Write Registers, continued

Register Address	Function	Name																										
00 40 _{hex}	<p>UARTMIS Register, Reset 00_{hex}</p> <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <tr> <td style="width: 5%;">Bit</td> <td style="width: 15%;">15:11</td> <td style="width: 5%;">10</td> <td style="width: 5%;">9</td> <td style="width: 5%;">8</td> <td style="width: 5%;">7</td> <td style="width: 5%;">6</td> <td style="width: 5%;">5</td> <td style="width: 5%;">4</td> <td style="width: 5%;">3</td> <td style="width: 5%;">2</td> <td style="width: 5%;">1</td> <td style="width: 5%;">0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>OEMIS</td> <td>BEMIS</td> <td>PEMIS</td> <td>FEMIS</td> <td>RTMIS</td> <td>TXMIS</td> <td>RXMIS</td> <td>DSRMMIS</td> <td>DCDMMIS</td> <td>CTSMMS</td> <td>RIMMIS</td> </tr> </table> <p>The UARTMIS register is the masked interrupt status register. It is a read-only register. On a read this register gives the current masked status value of the corresponding interrupt. A write has no effect.</p> <p>All of the bits are cleared to 0 when reset, except for the modem status interrupt bits 3 to 0. The modem status interrupt bits are undefined after reset.</p> <p>bit[15:11] Reserved</p> <p>bit[10] Overrun Error Masked Interrupt Status (OEMIS) Gives the masked interrupt state (after masking) of the UARTOEINTR interrupt</p> <p>bit[9] Break Error Masked Interrupt Status (BEMIS) Gives the masked interrupt state (after masking) of the UARTBEINTR interrupt</p> <p>bit[8] Parity Error Masked Interrupt Status (PEMIS) Gives the masked interrupt state (after masking) of the UARTPEINTR interrupt</p> <p>bit[7] Framing Error Masked Interrupt Status (FEMIS) Gives the masked interrupt state (after masking) of the UARTFEINTR interrupt</p> <p>bit[6] Receive Timeout Masked Interrupt Status (RTMIS) Gives the masked interrupt state (after masking) of the UARTRTINTR interrupt</p> <p>bit[5] Transmit Masked Interrupt Status (TXMIS) Gives the masked interrupt state (after masking) of the UARTRXINTR interrupt</p> <p>bit[4] Receive Masked Interrupt Status (RXMIS) Gives the masked interrupt state (after masking) of the UARTRXINTR interrupt</p> <p>bit[3] nDSRx Modem Masked Interrupt Status (DSRMMIS) Gives the masked interrupt state (after masking) of the UARTDSRINTR interrupt</p> <p>bit[2] nDCDx Modem Masked Interrupt Status (DCDMMIS) Gives the masked interrupt state (after masking) of the UARTDCDINTR interrupt</p> <p>bit[1] nCTSx Modem Masked Interrupt Status (CTSMMS) Gives the masked interrupt state (after masking) of the UARTCTSINTR interrupt</p> <p>bit[0] nRlx Modem Masked Interrupt Status (RIMMIS) Gives the masked interrupt state (after masking) of the UARTRIINTR interrupt</p>	Bit	15:11	10	9	8	7	6	5	4	3	2	1	0	Field	Reserved	OEMIS	BEMIS	PEMIS	FEMIS	RTMIS	TXMIS	RXMIS	DSRMMIS	DCDMMIS	CTSMMS	RIMMIS	UARTMIS
Bit	15:11	10	9	8	7	6	5	4	3	2	1	0																
Field	Reserved	OEMIS	BEMIS	PEMIS	FEMIS	RTMIS	TXMIS	RXMIS	DSRMMIS	DCDMMIS	CTSMMS	RIMMIS																

Table 2–19: UART Read/Write Registers, continued

Register Address	Function	Name																										
00 44 _{hex}	<p>UARTICR Register, Reset N/A</p> <table border="1"> <tr> <td>Bit</td> <td>15:11</td> <td>10</td> <td>9</td> <td>8</td> <td>7</td> <td>6</td> <td>5</td> <td>4</td> <td>3</td> <td>2</td> <td>1</td> <td>0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>OEIC</td> <td>BEIC</td> <td>PEIC</td> <td>FEIC</td> <td>RTIC</td> <td>TXIC</td> <td>RXIC</td> <td>DSR MIC</td> <td>DCD MIC</td> <td>CTS MIC</td> <td>RIM IC</td> </tr> </table> <p>Interrupt clear register</p> <p>bit[15:11] Reserved</p> <p>bit[10] Overrun Error Interrupt Clear (OEIC) Clears the UARTOEINTR interrupt</p> <p>bit[9] Break Error Interrupt Clear (BEIC) Clears the UARTBEINTR interrupt</p> <p>bit[8] Parity Error Interrupt Clear (PEIC) Clears the UARTPEINTR interrupt</p> <p>bit[7] Framing Error Interrupt Clear (FEIC) Clears the UARTFEINTR interrupt</p> <p>bit[6] Receive Timeout Interrupt Clear (RTIC) Clears the UARTRTINTR interrupt</p> <p>bit[5] Transmit Interrupt Clear (TXIC) Clears the UARTRXINTR interrupt</p> <p>bit[4] Receive Interrupt Clear (RXIC) Clears the UARTRXINTR interrupt</p> <p>bit[3] nDSRx Modem Interrupt Clear (DSRMIC) Clears the UARTDSRINTR interrupt</p> <p>bit[2] nDCDx Modem Interrupt Clear (DCDMIC) Clears the UARTDCDINTR interrupt</p> <p>bit[1] nCTSx Modem Interrupt Clear (CTSMIC) Clears the UARTCTSINTR interrupt</p> <p>bit[0] nRlx Modem Interrupt Clear Status (RIMIC) Clears the UARTRIINTR interrupt</p>	Bit	15:11	10	9	8	7	6	5	4	3	2	1	0	Field	Reserved	OEIC	BEIC	PEIC	FEIC	RTIC	TXIC	RXIC	DSR MIC	DCD MIC	CTS MIC	RIM IC	UARTICR
Bit	15:11	10	9	8	7	6	5	4	3	2	1	0																
Field	Reserved	OEIC	BEIC	PEIC	FEIC	RTIC	TXIC	RXIC	DSR MIC	DCD MIC	CTS MIC	RIM IC																
10 00 _{hex}	<p>UARTMuxSel Register, Reset 01_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>15:1</td> <td>0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>MS</td> </tr> </table> <p>IrDA/UART selection register</p> <p>bit[15:1] Reserved</p> <p>bit[0] Multiplexer Select (MS) A value of 1 selects UART, a value of 0 selects IrDA</p>	Bit	15:1	0	Field	Reserved	MS	UARTMuxSel																				
Bit	15:1	0																										
Field	Reserved	MS																										

2.6.3.3.1. UARTIBRD Register

The baud rate divider is calculated as follows:

$$\text{Baud rate divider BAUDDIV} = (\text{F}_{\text{UARTCLK}} / \{16 * \text{Baud rate}\})$$

where $\text{F}_{\text{UARTCLK}}$ is the UART reference clock frequency.

The value of BAUDDIV is typically rounded to the nearest integer value (BAUD DIVINT) and then programmed into the UARTIBRD register.

The content of UARTIBRD is not updated until transmission or reception of the current character is complete.

The minimum divide ratio possible is 1 and the maximum is 65535 ($2^{16} - 1$). That is, $\text{UARTIBRD} = 0$ is invalid when this is the case.

Table 2–20 shows some typical bit rates and their corresponding dividers, given the UART clock frequency of 7.3728 MHz.

Table 2–20: Typical Baud Rates and Dividers for $\text{f}_{\text{UARTCLK}}=7.3728$ MHz

Programmed Integer Divider	Bit Rate (bps)
4 _{hex}	115200
6 _{hex}	76800
8 _{hex}	57600
C _{hex}	38400
18 _{hex}	19200
20 _{hex}	14400
30 _{hex}	9600
C0 _{hex}	2400
180 _{hex}	1200
105D _{hex}	110

2.6.3.3.2. UARTLCR_H and UARTIBRD

UARTLCR_H and UARTIBRD form a single register (UARTLCR) which is updated on a single write strobe generated by a UARTLCR_H write. So, in order to internally update the contents of UARTIBRD, a UARTLCR_H write must always be performed at the end.

To update UARTIBRD only or together with UARTLCR_H: write UARTIBRD and then write UARTLCR_H.

Table 2–21 is a truth table for the SPS, EPS, and PEN bits of the UARTLCR_H register.

The contents of the UARTLCR_H register are not updated until transmission or reception of the current character is complete.

Table 2–21: Truth Table for UARTLCR_H Register

Parity Enable (PEN)	Even Parity Select (EPS)	Stick Parity Select (SPS)	Parity bit (transmitted or checked)
0	x	x	Not transmitted or checked
1	1	0	Even parity
1	0	0	Odd parity
1	0	1	1
1	1	1	0

2.6.4. Synchronous Serial Ports

The PUC 303xA device provides five independent Synchronous Serial Ports (SSP). Since the SSP interfaces' pins are configured as standard GPIO pins after reset, the pins required by the application must be configured for 'Bypass Mode' in the GPIO module (refer to Section 2.6.8.3. on page 106).

2.6.4.1. SSP Peripheral Operation

The general structure of each of the five SSP peripheral blocks available on PUC 303xA is shown in Fig. 2-10.

2.6.4.1.1. Interface Signals

The transmit and receive signals consist of the following signals:

- SCLK(x), serial clock
- SFRM(x), serial data framing signal
- STXD(x), serial transmit data
- SRXD(x), serial receive data

Note, that the interface for SSP5 is actually the I²S interface.

2.6.4.1.2. Transmit and Receive Logic

When the SSP is configured as a master, the serial clock to the connected slave is derived from a divided

version of the SSP(x)CLK. The master transmit logic reads a value from the transmit FIFO and carries out a parallel to serial conversion on it. The serial data and frame logic control signals are synchronized to SCLK(X), and then output to the externally connected slave through the stxd(x) and sfrm(x) pins respectively.

Simultaneously, the master receive logic performs serial to parallel conversion on the incoming synchronous srxd(x) data stream, extracting and storing values into the receive FIFO, for subsequent reading through the APB interface.

When configured as a slave, both the SCLK(X) and sfrm(x) signals are provided by an attached master and used to time its transmission and reception sequence.

The slave transmit logic, under control of the master clock, will successively read a value from the transmit FIFO, performs parallel to serial conversion, then output the serial data stream through the slave stxd(x) pin.

The master receive logic performs serial to parallel conversion on the incoming srxd(x) data stream, extracting and storing values into its receive FIFO, for subsequent reading through the APB interface.

Depending on the operating mode selected, the sfrm(x) output operates as an active HIGH frame synchronization output for Texas Instruments synchronous serial interface format or an active LOW slave select for the Motorola SPI and National Semiconductor Microwire modes.

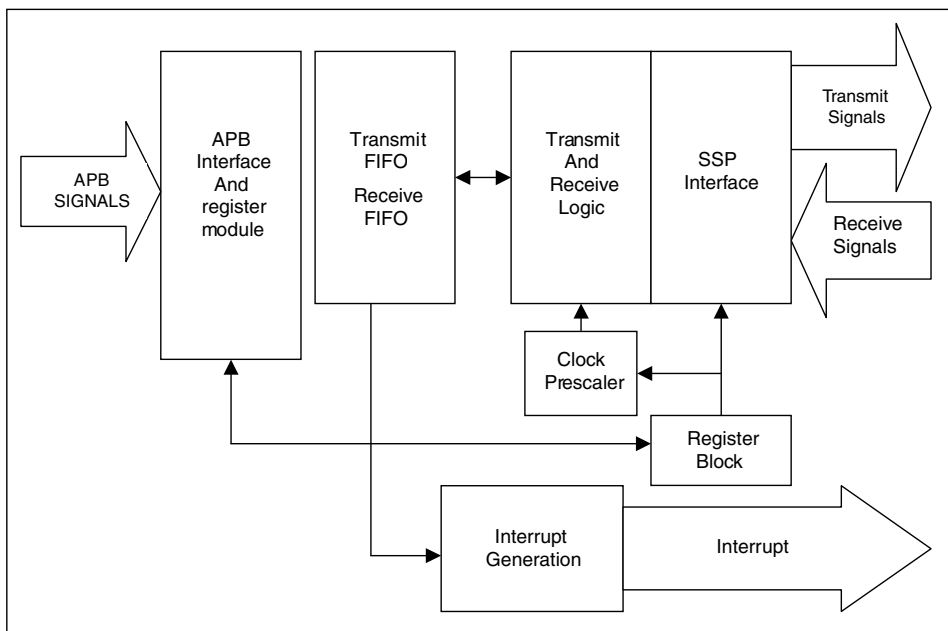


Fig. 2-10: General SSP Structure

2.6.4.2. SSP Modes

The description below applies to all of the five SSPs in the PUC 303xA, where '(x)' in the signal names is between 1 and 5, signifying the SSP number.

All of the five SSP peripherals can be configured to perform both master and slave operation in any of five available protocols. Each SSP peripheral comprises a core SSP peripheral and surrounding sequential logic, termed 'SSP Interface', for protocol conversion purposes.

The core SSP peripheral supports the following three primary protocols:

- Motorola SPI Mode
- National Semiconductor Microwire Mode
- Texas Instruments Synchronous Serial Interface Mode

In addition, two secondary protocols are supported by means of the SSP interface that 'wraps' the core SSP peripheral:

- PUC Word mode (emulates I²S)
- PUC Continuous mode

A more detailed description of these secondary modes can be found in Section 2.6.4.4.

In both, master and slave configurations, the SSP peripherals perform the following:

- Parallel-to-serial conversion on data written to an internal 16-bit wide, 8 location deep transmit FIFO.
- Serial-to-parallel conversion on the received data, buffering it in a similar 16-bit wide, 8-location deep receive FIFO.

The SSPs can operate with serial clocks up to 24 MHz, depending on whether the SSP is configured as a master or slave and whether the data is receive only.

The logical speed constraints are summarised below in terms of the derived SSP clocks from the clock generator, SSP(x)CLK, (see Section 2.3. on page 8):

- Master duplex with serial clock up to SSP(x)CLK/2
- Slave receive-only with serial clock up to SSP(x)CLK/3
- Slave duplex with serial clock up to SSP(x)CLK/12

2.6.4.3. SSP Primary Modes

When used in Texas Instruments SSI, Motorola SPI and National Semiconductor Microwire modes, the wrapping SSP interface that emulates the secondary protocols must be configured into 'Transparent' mode.

In this mode, the SSP interface logic feeds the relevant signals from the core SSP peripheral straight through, allowing direct communication with an external device based on the primary protocols.

Transparent mode is entered by writing to the relevant bits of the MODE_SELECT register. The DATA_SIZE register does not need to be set (see Table 2–22 on page 70).

For primary protocols, the SSP peripheral should be set up in accordance with the instructions contained in the register programming tables described below.

2.6.4.4. SSP Secondary Modes

In addition to the primary protocols, it is possible to configure any of the SSP interfaces to support either of the two secondary protocols:

- PUC Word mode (emulates I²S)
- PUC Continuous mode

These modes have been included to ensure seamless interfacing to the Micronas audio devices (e.g. MAS 35xxF family).

To facilitate these secondary modes, the SSP peripherals provide programmable mode selection and data size, to be specified in the registers `MODE_SELECT` and `DATA_SIZE` (see Table 2–22 on page 70).

The `MODE_SELECT` register allows the PUC Word or PUC Continuous modes to be selected for 'full duplex' or 'receive only' SSPs. The benefit of the 'receive only' mode is the higher available serial clock rate of $SSP(x)CLK/3$ as opposed to $SSP(x)CLK/12$, when the PUC 303xA is acting as an SSP slave interface.

For the special case of SSP5 the `SSP5CLK` source may be fed with an external clock, such that a multiple of an accurate audio clock (MAS 35x9F/MAS 3587F `CLKO`) can be used to transmit samples as I²S audio streams. This is controlled by `bit[3]` in the `MODE_SELECT` register.

2.6.4.4.1. PUC Continuous Mode

The PUC Continuous mode converts a Motorola SPI format to MAS 35x9F/MAS 3587F continuous format, in both master and slave modes.

The core SSP peripheral should be configured for Motorola SPI mode.

This mode is set by writing the appropriate value to the `MODE_SELECT` register. Master mode will be selected if the corresponding bit in the `SSPCR1` register has been set to master mode.

For PUC Continuous mode the PUC 303xA can support data sizes of 16, 20, 24 and 32 bit words, by programming the `DATA_SIZE` register.

It is possible to activate this mode with either unidirectional or duplex communication.

2.6.4.4.2. PUC Word Mode

The PUC Word mode converts a Texas Instruments Synchronous Serial Interface format to MAS 35x9F/MAS 3587F word mode format. It is possible to activate this mode in either master or slave modes with unidirectional or duplex communication.

The core SSP peripheral must be configured for TI SSI mode.

For a transmitting SSP peripheral, the SSP interface circuitry generates a word framed protocol that emulates I²S frames with an optional 1-bit delay on the data line. When operating in PUC Word mode as a receiver, the SSP interface will translate a non-delayed I²S protocol into the required TI SSI mode for the receiving core SSP peripheral.

In this mode information must be written to the `DATA_SIZE` register to let the SSP's interface logic discriminate between data transfers of 16 bits and transfers of either 20, 24 or 32 bits, so the correct framing signal can be generated. Additionally the core SSP peripheral must be set up for the proper data size in the `SSPCR0` register as follows:

- For word width of 16 bits, the data size for the core SSP peripheral is also configured as 16 bits.
- For word width of 32 bits, the data size for the core SSP peripheral is configured as 16 bits.
- For word width of 20 bits, the data size for the core SSP peripheral is configured as 10 bits.
- For word width of 24 bits, the data size for the core SSP peripheral is configured as 12 bits.

An external master may send a low impulse on `sfrm(x)` of length one bit-cycle during `bit[1]` of the last transferred data word as an end condition. This is ignored by the SSP interface, i.e. this low impulse has no resynchronisation effect.

2.6.4.5. Example SSP Communication Waveforms

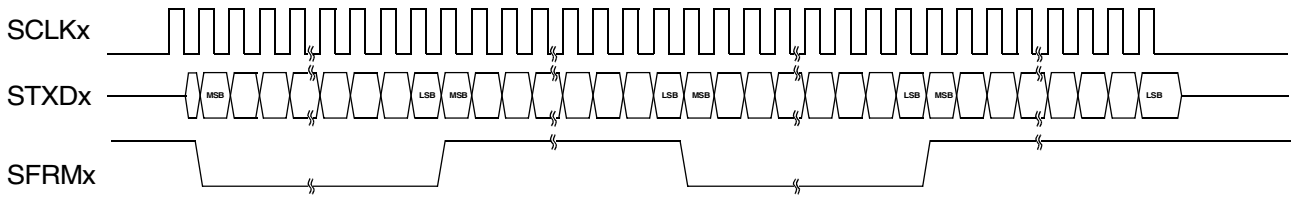


Fig. 2–11: Transmission of a single frame in PUC Word Mode

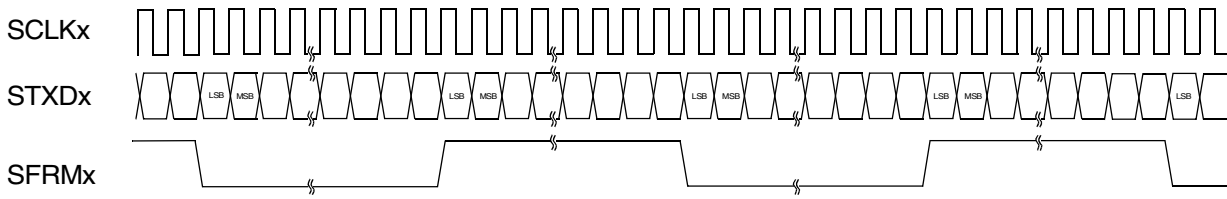


Fig. 2–12: Transmission of stream in PUC Word Mode with STXDx delayed

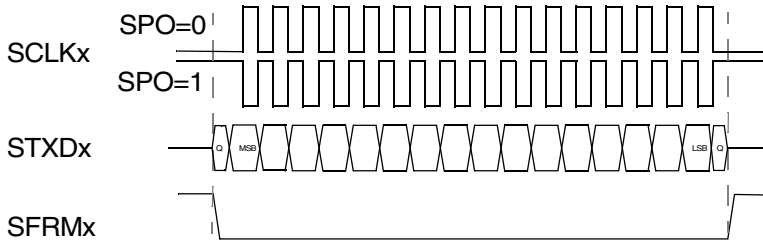


Fig. 2–13: Transmission of single word (16 bits) in Motorola SPI Mode with SPH = 0; SPO = 0/1

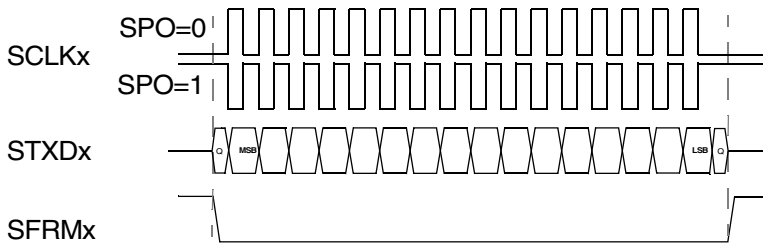


Fig. 2–14: Transmission of single word (16 bits) in Motorola SPI Mode with SPH = 1; SPO = 0/1

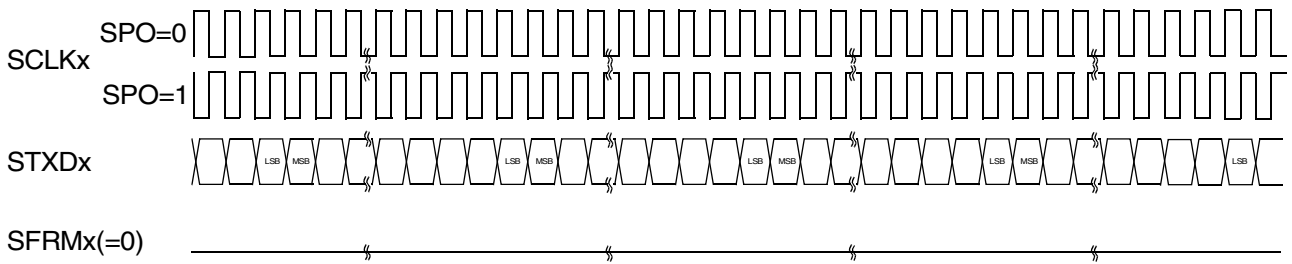


Fig. 2–15: Transmission of continuous stream in Motorola SPI Mode (SPH = 1)

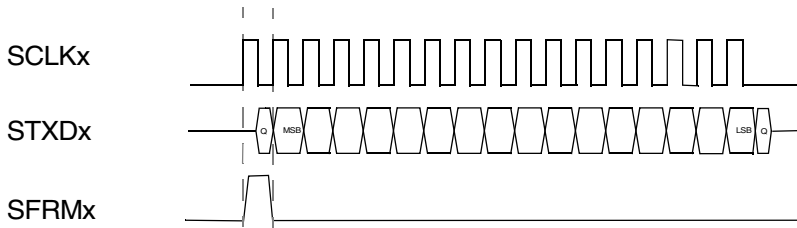


Fig. 2–16: Transmission of single word (16 bits) in Texas Instruments Mode

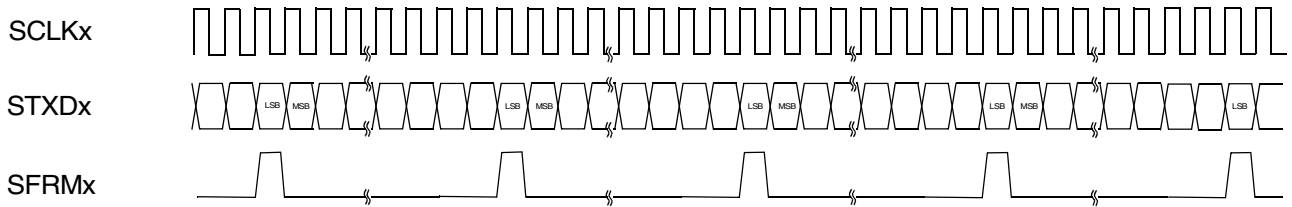


Fig. 2–17: Transmission of continuous stream in Texas Instruments Mode

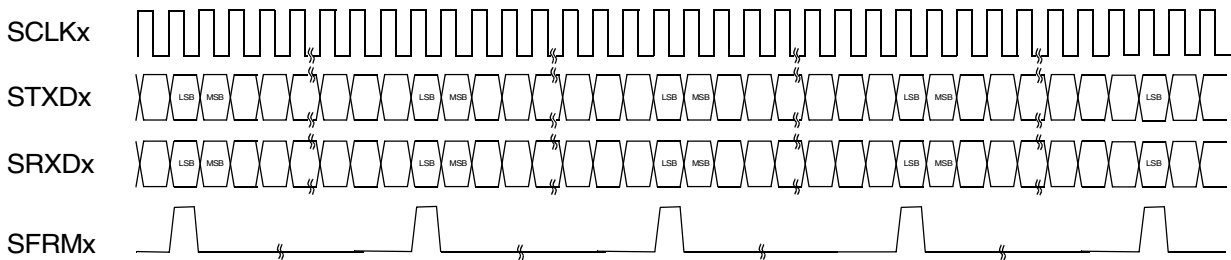


Fig. 2–18: Full duplex communication in Texas Instruments Mode

2.6.4.6. SSP Register Descriptions

Table 2–22 describes the SSP register map. Each of the five SSP peripherals within the PUC 303xA contains the same set of registers.

The addresses shown are an offset relative to each SSP base address. These are as follows:

- SSP 1: 001D.0000_{hex}
- SSP 2: 001D.4000_{hex}
- SSP 3: 001D.8000_{hex}
- SSP 4: 001D.C000_{hex}
- SSP 5: 001E.0000_{hex}

Table 2–22: SSP Register Map

Offset	Access	Width	Name	Description	Default
00 _{hex}	R/W	16	SSPCR0	Control Register 0	00 _{hex}
04 _{hex}	R/W	6	SSPCR1	Control Register 1	00 _{hex}
08 _{hex}	R/W	16	SSPDR	RX FIFO (read) / TX FIFO (write)	---- _{hex}
0C _{hex}	R	5	SSPSR	Status Register	00 _{hex}
10 _{hex}	R/W	8	SSPCPSR	Clock Prescale Register	00 _{hex}
14 _{hex}	R/W	3/16	SSPIIR/SSPICR	Interrupt Status / Clear Register	00 _{hex}
18 _{hex}	R/W	4	MODE_SELECT	Mode Select Register	02 _{hex}
1C _{hex}	R/W	2	DATA_SIZE	Data Size Register	00 _{hex}

Table 2–23: SSP Read/Write Registers

Register Address	Function	Name												
00 _{hex}	SSPCR0 Register, Reset 0000 _{hex} <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 10%; text-align: center;">Bit</td> <td style="width: 40%; text-align: center;">15:8</td> <td style="width: 5%; text-align: center;">7</td> <td style="width: 5%; text-align: center;">6</td> <td style="width: 15%; text-align: center;">5:4</td> <td style="width: 25%; text-align: center;">3:0</td> </tr> <tr> <td style="text-align: center;">Field</td> <td style="text-align: center;">SCR</td> <td style="text-align: center;">SPH</td> <td style="text-align: center;">SPO</td> <td style="text-align: center;">FRF</td> <td style="text-align: center;">DSS</td> </tr> </table> <p>This register controls various functions within the SSP peripheral</p> <p>bit[15:8] 0 SCR Serial clock rate division factor. The value SCR is used to generate the transmit and receive bit rate of the SSPMS. ... 255_{dec} The bit rate is $F_{SSPCLK} \div \text{CPSDVSR} \times (1 + \text{SCR})$where CPSDVSR is an even value from 2 to 254, programmed via SSPCPSR register and SCR is a value from 0 to 255.</p> <p>bit[7] SPH SCLKOUT phase (applicable to Motorola SPI frame format only).</p> <p>bit[6] SPO SCLKOUT polarity (applicable to Motorola SPI frame format only).</p> <p>bit[5:4] 00_{bin} FRF ... 11_{bin} Frame format: 00 Motorola SPI frame format 01 TI synchronous serial frame format 10 National Microwire frame format 11 Reserved, undefined operation</p> <p>bit[3:0] 0000_{bin} DSS ... Data Size Select: 1111_{bin} 0000 Reserved, undefined operation 0001 Reserved, undefined operation 0010 Reserved, undefined operation 0011 4-bit data 0100 5-bit data 0101 6-bit data 0110 7-bit data 0111 8-bit data 1000 9-bit data 1001 10-bit data 1010 11-bit data 1011 12-bit data 1100 13-bit data 1101 14-bit data 1110 15-bit data 1111 16-bit data..</p>	Bit	15:8	7	6	5:4	3:0	Field	SCR	SPH	SPO	FRF	DSS	SSPCR0
Bit	15:8	7	6	5:4	3:0									
Field	SCR	SPH	SPO	FRF	DSS									

Table 2–23: SSP Read/Write Registers, continued

Register Address	Function	Name																		
04 _{hex}	<p>SSPCR1 register, Reset 00_{hex}</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 15%;">Bit</td> <td style="width: 45%; text-align: center;">31:7</td> <td style="width: 5%; text-align: center;">6</td> <td style="width: 5%; text-align: center;">5</td> <td style="width: 5%; text-align: center;">4</td> <td style="width: 5%; text-align: center;">3</td> <td style="width: 5%; text-align: center;">2</td> <td style="width: 5%; text-align: center;">1</td> <td style="width: 5%; text-align: center;">0</td> </tr> <tr> <td>Field</td> <td style="text-align: center;">Reserved</td> <td style="text-align: center;">Re-served</td> <td style="text-align: center;">MS</td> <td style="text-align: center;">SSE</td> <td style="text-align: center;">LBM</td> <td style="text-align: center;">RORIE</td> <td style="text-align: center;">TIE</td> <td style="text-align: center;">RIE</td> </tr> </table> <p>This register controls various functions within the SSP peripheral</p> <p>bit[31:7] Reserved, read unpredictable, should be written as 0.</p> <p>bit[6] Reserved, must be written as 0.</p> <p>bit[5] MS Master/slave mode select. This bit can be modified only when the SSP is disabled (SSE=0). 0 = Device configured as master (default). 1 = Device configured as slave.</p> <p>bit[4] SSE Synchronous serial port enable: 0 = SSP operation disabled. 1 = SSP operation enabled.</p> <p>bit[3] LBM Loop back mode: 0 = Normal serial port operation enabled. 1 = Output of transmit serial shifter is connected to input of receive serial shifter internally.</p> <p>bit[2] RORIE Receive FIFO overrun interrupt enable: 0 = Overrun detection is disabled. Overrun condition does not generate the SSPRORINTR interrupt. Clearing this bit to zero also clears the SSPRORINTR interrupt if it is already asserted. 1 = Overrun detection is enabled. Overrun condition generates the SSPRORINTR interrupt.</p> <p>bit[1] TIE Transmit FIFO interrupt enable: 0 = Transmit FIFO half-full or less condition does not generate the SSPTXINTR interrupt. 1 = Transmit FIFO half-full or less condition generates the SSPTXINTR interrupt.</p> <p>bit[0] RIE Receive FIFO interrupt enable: 0 = Receive FIFO half-full or more condition does not generate the SSPRXINTR interrupt. 1 = Receive FIFO half-full or more condition generates the SSPRXINTR interrupt.</p>	Bit	31:7	6	5	4	3	2	1	0	Field	Reserved	Re-served	MS	SSE	LBM	RORIE	TIE	RIE	SSPCR1
Bit	31:7	6	5	4	3	2	1	0												
Field	Reserved	Re-served	MS	SSE	LBM	RORIE	TIE	RIE												

Table 2–23: SSP Read/Write Registers, continued

Register Address	Function	Name						
08 _{hex}	<p>SSPDR register, Reset ----_{hex}</p> <table border="1" data-bbox="288 421 1201 499"> <tr> <td data-bbox="288 421 341 456">Bit</td> <td data-bbox="341 421 772 456">31:16</td> <td data-bbox="772 421 1201 456">15:0</td> </tr> <tr> <td data-bbox="288 456 341 499">Field</td> <td data-bbox="341 456 772 499">Reserved</td> <td data-bbox="772 456 1201 499">DATA</td> </tr> </table> <p>When SSPDR is read, the entry in the receive FIFO (pointed to by the current FIFO read pointer) is accessed. As data values are removed by the SSP receive logic from the incoming data frame, they are placed into the entry in the receive FIFO (pointed to by the current FIFO write pointer).</p> <p>When SSPDR is written to, the entry in the transmit FIFO (pointed to by the write pointer), is written to. Data values are removed from the transmit FIFO one value at a time by the transmit logic. It is loaded into the transmit serial shifter, then serially shifted out onto the SSPTXD pin at the programmed bit rate.</p> <p>When a data size of less than 16 bits is selected, the user must right-justify data written to the transmit FIFO. The transmit logic ignores the unused bits. Received data less than 16 bits is automatically right-justified in the receive buffer. When the SSP is programmed for National Microwire frame format, the default size for transmit data is eight bits (the most significant byte is ignored). The receive data size is controlled by the programmer. The transmit FIFO and the receive FIFO are not cleared even when SSE is set to zero. This allows the software to fill the transmit FIFO before enabling the SSP.</p> <p>bit[31:16] Reserved</p> <p>bit[15:0] DATA</p> <p>Transmit/Receive FIFO: Read = Receive FIFO Write = Transmit FIFO.</p> <p>The user should right-justify data when the SSP is programmed for a data size that is less than 16 bits. Unused most significant bits are ignored by transmit logic. The receive logic automatically right-justifies.</p>	Bit	31:16	15:0	Field	Reserved	DATA	SSPDR
Bit	31:16	15:0						
Field	Reserved	DATA						

Table 2–23: SSP Read/Write Registers, continued

Register Address	Function	Name														
0C _{hex}	<p>SSPSR register, Reset 00_{hex}</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 15%; text-align: center;">Bit</td> <td style="width: 55%; text-align: center;">31:5</td> <td style="width: 5%; text-align: center;">4</td> <td style="width: 5%; text-align: center;">3</td> <td style="width: 5%; text-align: center;">2</td> <td style="width: 5%; text-align: center;">1</td> <td style="width: 5%; text-align: center;">0</td> </tr> <tr> <td style="text-align: center;">Field</td> <td style="text-align: center;">Reserved</td> <td style="text-align: center;">BSY</td> <td style="text-align: center;">RFF</td> <td style="text-align: center;">RNE</td> <td style="text-align: center;">TNF</td> <td style="text-align: center;">TFE</td> </tr> </table> <p>This register is a read-only status register which contains bits that indicate the FIFO fill status and the SSP busy status.</p> <p>bit[31:5] Reserved, read unpredictable, should be written as 0.</p> <p>bit[4] BSY SSP busy flag (read-only): 0 = SSP is idle. 1 = SSP is currently transmitting and/or receiving a frame or the transmit FIFO is not empty.</p> <p>bit[3] RFF Receive FIFO full (read-only): 0 = Receive FIFO is not full. 1 = Receive FIFO is full.</p> <p>bit[2] RNE Receive FIFO not empty (read-only): 0 = Receive FIFO is empty. 1 = Receive FIFO is not empty.</p> <p>bit[1] TNF Transmit FIFO not full (read-only): 0 = Transmit FIFO is full. 1 = Transmit FIFO is not full.</p> <p>bit[0] TFE Transmit FIFO empty (read-only): 0 = Transmit FIFO is not empty. 1 = Transmit FIFO is empty.</p>	Bit	31:5	4	3	2	1	0	Field	Reserved	BSY	RFF	RNE	TNF	TFE	SSPSR
Bit	31:5	4	3	2	1	0										
Field	Reserved	BSY	RFF	RNE	TNF	TFE										

Table 2–23: SSP Read/Write Registers, continued

Register Address	Function	Name										
10 _{hex}	<p>SSPCPSR register, Reset 00_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:8</td> <td>7:0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>CPSDVSR</td> </tr> </table> <p>SSPCPSR is the clock prescale register and specifies the division factor by which the input SSPCLK should be internally divided before further use.</p> <p>The value programmed into this register should be an even number between 2–254. The least significant bit of the programmed number is hard-coded to zero. If an odd number is written to this register, data read back from this register will have the least significant bit as zero.</p> <p>bit[31:8] Reserved, read unpredictable, should be written as 0.</p> <p>bit[7:0] 2_{dec} CPSDVSR ... Clock prescale divider. Should be an even number from 254_{dec} 2_{dec}...254_{dec}, depending on the frequency of SSPCLK. The (even) least significant bit always returns zero on reads.</p>	Bit	31:8	7:0	Field	Reserved	CPSDVSR	SSPCPSR				
Bit	31:8	7:0										
Field	Reserved	CPSDVSR										
14 _{hex}	<p>Combined SSPIIR/SSPICR register, Reset 00_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:3</td> <td>2</td> <td>1</td> <td>0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>RORIS</td> <td>TIS</td> <td>RIS</td> </tr> </table> <p>The interrupt status is read from the SSP interrupt identification register (SSPIIR). A write of any value to the SSP interrupt clear register (SSPICR, bits 15:0) clears the SSP receive FIFO overrun interrupt. This interrupt clearing mechanism is in addition to the other mechanism mentioned for the SSPCR1 register. Therefore, clearing the RORIE bit in the SSPCR1 register will also clear the overrun condition if already asserted. All the bits are cleared to zero when reset.</p> <p>bit[31:3] Reserved, read unpredictable</p> <p>bit[15:0] A write to this part of the register clears the receive overrun interrupt, regardless of the data value written.</p> <p>bit[2] RORIS Read: SSP Receive FIFO overrun interrupt status 0 = SSPRORINTR is not asserted. 1= SSPRORINTR is asserted</p> <p>bit[1] TIS Read: SSP transmit FIFO service request interrupt status 0 = SSPTXINTR is not asserted. 1= SSPTXINTR is asserted.</p> <p>bit[0] RIS Read: SSP receive FIFO service request interrupt status 0 = SSPRXINTR is not asserted. 1= SSPRXINTR is asserted.</p>	Bit	31:3	2	1	0	Field	Reserved	RORIS	TIS	RIS	SSPIIR/SSPICR
Bit	31:3	2	1	0								
Field	Reserved	RORIS	TIS	RIS								

Table 2–23: SSP Read/Write Registers, continued

Register Address	Function	Name								
18 _{hex}	<p>MODE_SELECT register, Reset 02_{hex}</p> <table border="1" data-bbox="288 421 1201 499"> <tr> <td data-bbox="288 421 341 456">Bit</td> <td data-bbox="341 421 986 456">31:4</td> <td data-bbox="986 421 1038 456">3</td> <td data-bbox="1038 421 1201 456">2:0</td> </tr> <tr> <td data-bbox="288 456 341 499">Field</td> <td data-bbox="341 456 986 499">Reserved</td> <td data-bbox="986 456 1038 499">ACLK</td> <td data-bbox="1038 456 1201 499">WMS</td> </tr> </table> <p>This register selects the operational mode for the SSP interface circuitry. The ACLK field selects the SSP-peripheral's main operating clock (SSP5 only). The WMS field encodes different modes depending on whether the SSP operates as a master or a slave.</p> <p>bit[31:4] Reserved, read unpredictable</p> <p>bit[3] ACLK</p> <p> This bit selects the main clock SSPCLK for the SSP peripheral. It is only functional for SSP5. It should be written as '0' for SSPs 1 - 4.</p> <p> 0: The SSP5 peripheral uses the internally derived SSPCLK</p> <p> 1: The SSP5 peripheral uses the external clock input on I2S_AUDIOCLK pin</p>	Bit	31:4	3	2:0	Field	Reserved	ACLK	WMS	MODE_SELECT
Bit	31:4	3	2:0							
Field	Reserved	ACLK	WMS							

Table 2–23: SSP Read/Write Registers, continued

Register Address	Function	Name																												
18 _{hex} (con'd)	bit[2:0] 000 _{bin} WMS ... 101 _{bin}	MODE_SELECT																												
	<p>The WMS field encodes the following SSP interface operational modes depending on whether the SSP operates as a master or a slave:</p> <table border="1"> <thead> <tr> <th>Master-SSP mode</th> <th>Comment</th> </tr> </thead> <tbody> <tr> <td>000_{bin}: PUC Continuous</td> <td>This SSP interface mode was designed for continuous communication with MAS 35xxF devices. It requires the core SSP to operate in Motorola SPI mode. In SSPCR0 register set FRF=00_{bin} and SPH=1 to put SSP into required Motorola SPI mode.</td> </tr> <tr> <td>001_{bin}: PUC Word</td> <td>This SSP interface mode performs transmission of I²S-like data streams in Sony format and complies with MAS 35xxF I²S serial communication. It requires the core SSP to operate in TI mode. In SSPCR0 register set FRF=01_{bin} to put core SSP into required Texas Instruments mode.</td> </tr> <tr> <td>010_{bin}: Transparent</td> <td>Use this selection for operation of the SSP in it's primary modes: Motorola SPI, Texas Instruments and National Microwire.</td> </tr> <tr> <td>011_{bin}: PUC Word TXdel</td> <td>This SSP interface mode performs transmission of I²S-like data streams in Philips format. It requires the core SSP to operate in TI mode. This mode only affects the transmitter; the receiver side operates as with PUC Word mode (001_{bin}). In SSPCR0 register set FRF=01_{bin} to put core SSP into required Texas Instruments mode.</td> </tr> <tr> <td>100_{bin} - 111_{bin}: reserved</td> <td></td> </tr> <tr> <th>Slave-SSP mode</th> <th>Comment</th> </tr> <tr> <td>000_{bin}: PUC Continuous</td> <td>Same as for Master-SSP</td> </tr> <tr> <td>001_{bin}: PUC Word TXdel</td> <td>Same as for Master-SSP</td> </tr> <tr> <td>010_{bin}: Transparent</td> <td>Same as for Master-SSP</td> </tr> <tr> <td>011_{bin}: reserved</td> <td></td> </tr> <tr> <td>100_{bin}: SlvRcv Continuous</td> <td>High-speed equivalent for PUC Continuous mode for pure Slave-Receiver</td> </tr> <tr> <td>101_{bin}: SlvRcv Word</td> <td>High-speed equivalent for PUC Word mode for pure Slave-Receiver¹⁾</td> </tr> <tr> <td>110_{bin} - 111_{bin}: reserved</td> <td></td> </tr> </tbody> </table>	Master-SSP mode	Comment	000 _{bin} : PUC Continuous	This SSP interface mode was designed for continuous communication with MAS 35xxF devices. It requires the core SSP to operate in Motorola SPI mode. In SSPCR0 register set FRF=00 _{bin} and SPH=1 to put SSP into required Motorola SPI mode.	001 _{bin} : PUC Word	This SSP interface mode performs transmission of I ² S-like data streams in Sony format and complies with MAS 35xxF I ² S serial communication. It requires the core SSP to operate in TI mode. In SSPCR0 register set FRF=01 _{bin} to put core SSP into required Texas Instruments mode.	010 _{bin} : Transparent	Use this selection for operation of the SSP in it's primary modes: Motorola SPI, Texas Instruments and National Microwire.	011 _{bin} : PUC Word TXdel	This SSP interface mode performs transmission of I ² S-like data streams in Philips format. It requires the core SSP to operate in TI mode. This mode only affects the transmitter; the receiver side operates as with PUC Word mode (001 _{bin}). In SSPCR0 register set FRF=01 _{bin} to put core SSP into required Texas Instruments mode.	100 _{bin} - 111 _{bin} : reserved		Slave-SSP mode	Comment	000 _{bin} : PUC Continuous	Same as for Master-SSP	001 _{bin} : PUC Word TXdel	Same as for Master-SSP	010 _{bin} : Transparent	Same as for Master-SSP	011 _{bin} : reserved		100 _{bin} : SlvRcv Continuous	High-speed equivalent for PUC Continuous mode for pure Slave-Receiver	101 _{bin} : SlvRcv Word	High-speed equivalent for PUC Word mode for pure Slave-Receiver ¹⁾	110 _{bin} - 111 _{bin} : reserved		
Master-SSP mode	Comment																													
000 _{bin} : PUC Continuous	This SSP interface mode was designed for continuous communication with MAS 35xxF devices. It requires the core SSP to operate in Motorola SPI mode. In SSPCR0 register set FRF=00 _{bin} and SPH=1 to put SSP into required Motorola SPI mode.																													
001 _{bin} : PUC Word	This SSP interface mode performs transmission of I ² S-like data streams in Sony format and complies with MAS 35xxF I ² S serial communication. It requires the core SSP to operate in TI mode. In SSPCR0 register set FRF=01 _{bin} to put core SSP into required Texas Instruments mode.																													
010 _{bin} : Transparent	Use this selection for operation of the SSP in it's primary modes: Motorola SPI, Texas Instruments and National Microwire.																													
011 _{bin} : PUC Word TXdel	This SSP interface mode performs transmission of I ² S-like data streams in Philips format. It requires the core SSP to operate in TI mode. This mode only affects the transmitter; the receiver side operates as with PUC Word mode (001 _{bin}). In SSPCR0 register set FRF=01 _{bin} to put core SSP into required Texas Instruments mode.																													
100 _{bin} - 111 _{bin} : reserved																														
Slave-SSP mode	Comment																													
000 _{bin} : PUC Continuous	Same as for Master-SSP																													
001 _{bin} : PUC Word TXdel	Same as for Master-SSP																													
010 _{bin} : Transparent	Same as for Master-SSP																													
011 _{bin} : reserved																														
100 _{bin} : SlvRcv Continuous	High-speed equivalent for PUC Continuous mode for pure Slave-Receiver																													
101 _{bin} : SlvRcv Word	High-speed equivalent for PUC Word mode for pure Slave-Receiver ¹⁾																													
110 _{bin} - 111 _{bin} : reserved																														
	<p>¹⁾A pure Slave-Receiver (slave SSP where no bidirectional communication is required) can operate with serial input clocks SCLKx up to frequencies of f_{SSPCLK} / 3 in this mode. For slaves requiring bidirectional communication the frequency of SCLKx must be equal or less than f_{SSPCLK} / 12.</p>																													

Table 2–23: SSP Read/Write Registers, continued

Register Address	Function	Name																				
1C _{hex}	<p>DATA_SIZE register, Reset 00_{hex}</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 10%;">Bit</td> <td style="width: 80%; text-align: center;">31:2</td> <td style="width: 10%; text-align: center;">1:0</td> </tr> <tr> <td>Field</td> <td style="text-align: center;">Reserved</td> <td style="text-align: center;">WS</td> </tr> </table> <p>This register selects the word size of transferred data items for the SSP interface circuitry. The WS field must be written with the appropriate value for SSPs operating in PUC Word, PUC Word TXdel, or SlvRcv Word mode. The value of WS is irrelevant for all other modes.</p> <p>bit[31:2] Reserved</p> <p>bit[1:0] 00_{bin} WS</p> <p style="padding-left: 40px;">...</p> <p style="padding-left: 40px;">11_{bin}</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td colspan="2">This field encodes the word size of data items for all Word modes (PUC Word, PUC Word TXdel, SlvRcv Word).</td> </tr> <tr> <td colspan="2">For any of these Word modes set FRF=01_{bin} in SSPCR0 register to put core SSP into required Texas Instruments mode. For additional settings see comments below.</td> </tr> <tr> <th style="text-align: left;">Encoding</th> <th style="text-align: left;">Comment</th> </tr> <tr> <td>00_{bin}: 20-bit word size</td> <td>Set DSS=1001_{bin} in SSPCR0 register to put core SSP Data Size to 10-bit data.</td> </tr> <tr> <td>01_{bin}: 24-bit word size</td> <td>Set DSS=1011_{bin} in SSPCR0 register to put core SSP Data Size to 12-bit data.</td> </tr> <tr> <td>10_{bin}: 32-bit word size</td> <td>Set DSS=1111_{bin} in SSPCR0 register to put core SSP Data Size to 16-bit data.</td> </tr> <tr> <td>11_{bin}: 16-bit word size</td> <td></td> </tr> </table>	Bit	31:2	1:0	Field	Reserved	WS	This field encodes the word size of data items for all Word modes (PUC Word, PUC Word TXdel, SlvRcv Word).		For any of these Word modes set FRF=01 _{bin} in SSPCR0 register to put core SSP into required Texas Instruments mode. For additional settings see comments below.		Encoding	Comment	00 _{bin} : 20-bit word size	Set DSS=1001 _{bin} in SSPCR0 register to put core SSP Data Size to 10-bit data.	01 _{bin} : 24-bit word size	Set DSS=1011 _{bin} in SSPCR0 register to put core SSP Data Size to 12-bit data.	10 _{bin} : 32-bit word size	Set DSS=1111 _{bin} in SSPCR0 register to put core SSP Data Size to 16-bit data.	11 _{bin} : 16-bit word size		DATA_SIZE
Bit	31:2	1:0																				
Field	Reserved	WS																				
This field encodes the word size of data items for all Word modes (PUC Word, PUC Word TXdel, SlvRcv Word).																						
For any of these Word modes set FRF=01 _{bin} in SSPCR0 register to put core SSP into required Texas Instruments mode. For additional settings see comments below.																						
Encoding	Comment																					
00 _{bin} : 20-bit word size	Set DSS=1001 _{bin} in SSPCR0 register to put core SSP Data Size to 10-bit data.																					
01 _{bin} : 24-bit word size	Set DSS=1011 _{bin} in SSPCR0 register to put core SSP Data Size to 12-bit data.																					
10 _{bin} : 32-bit word size	Set DSS=1111 _{bin} in SSPCR0 register to put core SSP Data Size to 16-bit data.																					
11 _{bin} : 16-bit word size																						

2.6.4.7. SSP Interrupt Logic

There are three interrupts generated by the SSP peripheral:

- **SSPRXINTR:**
SSP receive FIFO service interrupt request
- **SSPTXINTR:**
SSP transmit FIFO service interrupt request
- **SSPRORINTR:**
SSP receive overrun interrupt request.

All three interrupts are maskable, active HIGH interrupts. They are logically ORed to form a combined interrupt line to the interrupt controller, which should be programmed for level-sensitivity of the SSP interrupt (see Table 2–15 on page 37).

Each of the three maskable interrupts may be enabled or disabled by changing the enable bits in the SSPCR1 control register. Setting the appropriate enable bit to '1' enables the interrupt.

The status of the individual interrupt sources can be read from the interrupt status register SSPIIR. Note that reading the SSPIIR register does not clear the status bits.

2.6.4.7.1. SSPRXINTR

The receive interrupt status bit RIS in SSPIIR is set and the interrupt is signaled to the interrupt controller when there are four or more valid entries in the receive FIFO. They are cleared / deasserted when the number of entries decreases to two or less again, e.g. by servicing the interrupt.

2.6.4.7.2. SSPTXINTR

The transmit interrupt status bit TIS in SSPIIR is set and the interrupt is signaled to the interrupt controller when the transmit FIFO is less than or equal to half full (when there is space for four or more entries). They are cleared / deasserted when the number of entries increases to six or more again, e.g. by servicing the interrupt.

The transmitter interrupt **SSPTXINTR** is not qualified with the SSP enable signal 'SSE', which allows operation in one of two ways: Data can be written to the transmit FIFO prior to enabling the SSP and the interrupts. Alternatively, the SSP and interrupts can be enabled so that data can be written to the transmit FIFO by an interrupt service routine.

2.6.4.7.3. SSPRORINTR

The receive overrun status bit RORIS in SSPIIR is set and the interrupt is signaled to the interrupt controller when the receive FIFO is already full and an additional data frame is received, causing an overrun of the receive FIFO. Data is over-written in the shift register, but not the FIFO. To clear the RORIS bit and deassert the interrupt the lower 16 bits of the SSPIIR/SSPICR register should be written with any value.

2.6.4.7.4. I²S Interface

The PUC 303xA provides an I²S master interface through the following pins:

- I2S_CLK, serial data clock input/output.
- I2S_FRM, data word framing input/output.
- I2S_TXD, serial transmit data output.
- I2S_RXD, serial receive data input.
- I2S_AUDIOCLK, alternative SSP5CLK clock source input.

This interface is implemented by using SSP5, as described in Section 2.6.4.

Note, that in principle any of the five SSP peripherals in the PUC 303xA device has the capability to perform the I²S (PUC Word mode) protocol, though only SSP 5 has the ability to provide an alternative sampling clock. This is especially useful when PUC 303xA has to generate an accurate internal USB clock USB48CLK of 48 MHz while an accurate audio clock based I²S data stream is to be output on the SSP5/I²S interface (Micronas MAS 35xxF devices are ideally suited to provide accurate I2S_AUDIOCLK while acting as a I²S slave).

2.6.5. Timers

The Triple Timer Counters (TTC) provide three independently programmable timing and counting functions for use within the PUC 303xA.

2.6.5.1. Features

- Three independently programmable 16-bit timer/counters.
- Internal (PCLK_APB) or external clock source (timers 1 and 2 only).
- Three independent prescalers, for clock speed selection.
- Interrupt generated either on overflow or at regular intervals.
- Interrupt generated when count matches a programmable value.
- Two output waveforms (timers 1 and 2), generated using overflow or interval and match interrupts.
- An Event Timer measures length of each external clock pulse, either high or low duration (timers 1 and 2 only).
- Counters can be incrementing or decrementing.
- Current value of each count register can be read at any time.

2.6.5.2. Description

The TTC module provides three independent timer and counter modules (see Fig. 2–19). This illustrates the top level structure of the module.

All of the timers can be clocked using the APB system clock (PCLK_APB). In addition for timers 1 and 2, an externally derived clock can be used instead. Each counter can independently prescale its selected clock input within a range from ÷2 to ÷65536 in powers of 2.

Counters can be set to decrement or increment.

Timers 1 and 2 each have a pin on the device associated with them (TIMER_1 and TIMER_2). The pin can be used to clock the timer, count the number of events, measure event times, and also generate PWM-type waveforms.

Since the TIMER_1 and TIMER_2 pins are configured as standard GPIO pins after reset, the pins required by the application must be configured for 'Bypass Mode' in the GPIO module (see Section 2.6.8.3. on page 106).

Each external input is synchronized with PCLK_APB before being applied to its timer or counter.

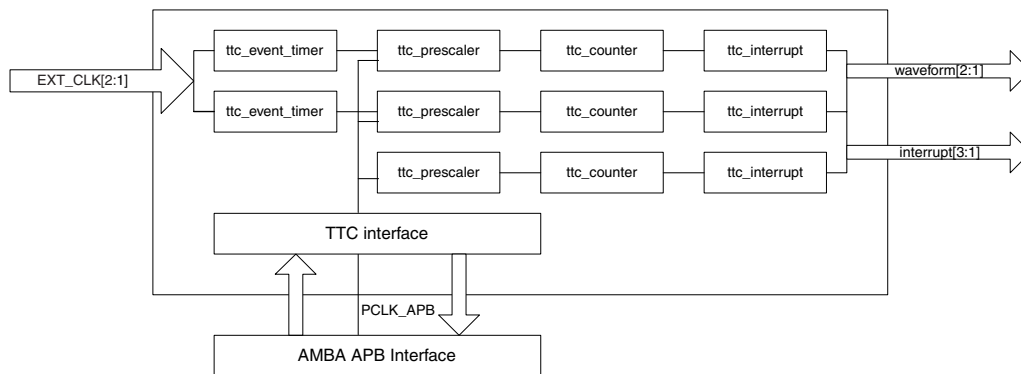


Fig. 2–19: TTC Top-Level Structure

2.6.5.2.1. AMBA APB Interface

The AMBA APB interface contains the configuration and status registers that are addressed through the APB.

2.6.5.2.2. Event Timer

Event Timer modules are available for Timers 1 and 2 to measure durations of High or Low phases of externally applied signals in PCLK cycles.

2.6.5.2.3. Prescaler

The design includes a Prescaler module to provide a selectable clock frequency for driving the timer-counter. The prescaler can be programmed to operate on the system clock, PCLK_APB; or an external clock. The selected clock is then divided to provide the count clock. Division can be within the range $\div 2$ to $\div 65536$.

2.6.5.2.4. Counter Module

The Counter module can be incrementing or decrementing, and can be configured to count for a given interval. It will also compare three Match registers to the value of the Counter, and generate an interrupt if one matches.

2.6.5.2.5. Interrupt Module

There are three interrupt signals, one for each timer, available for use at system level. An interrupt occurs when a bit in the Interrupt Enable register and the corresponding bit in the Interrupt Detect register are both set. The resulting ANDed outputs are then ORed to generate the system interrupt signal.

The Interrupt register takes the interrupt signals from the corresponding timer-counter module and stores them until the register is read. When the Interrupt register is read by the processor, it is reset.

To enable an interrupt, a '1' must be written to the corresponding bit position in the Interrupt Enable register.

2.6.5.3. Initialization

On initialization, the counters are set to the following configuration:

- Overflow mode
- Internal clock selected.
- Counter free-running
- All interrupts disabled.
- Event Timer disabled.
- Output waveforms disabled.

2.6.5.4. Modes of Operation

Each of the timer counter modules can operate in one of four modes:

- Interval timing, incrementing count
- Interval timing, decrementing count
- Overflow detection, incrementing count
- Overflow detection, decrementing count

Register matching can also be programmed for each of these modes.

2.6.5.4.1. Interval Mode

If the interval bit is set in the counter control register, the counter counts up or down from a programmable interval value. An interrupt is generated when the count passes through zero. When Interval Mode operation is not enabled, the counter is free-running.

Incrementing

When the counter value register is equal to the interval register value, the counter is reset to zero, the interval interrupt is set and counting up restarted.

Decrementing

When the counter value register is equal to zero, the interval interrupt is set. The counter is then reset to the interval register value and counting down is restarted.

2.6.5.4.2. Overflow Mode

If the interval bit in the counter control register is not set, the counter can count up to or down from its full 16-bit value. An interrupt is generated when the count passes through zero.

Incrementing

When the counter value register reaches $FFFF_{\text{hex}}$ it overflows to zero, the overflow interrupt is set, and counting up is restarted.

Decrementing

When the counter value register reaches zero, the overflow interrupt is set. The counter then overflows to $FFFF_{\text{hex}}$ and counting down is restarted.

2.6.5.4.3. Register Matching

In the register matching operation, an interrupt is generated each time the counter value matches the value stored in the match register. Match events can be programmed for both Interval and Overflow Mode.

There are three match registers for each timer-counter. The module can, therefore, generate nine different match interrupts. When the match register select is set and the match register is equal to the counter value register then that match interrupt will be set.

2.6.5.5. Generation of Waveforms

Timers 1 and 2 can be easily set up for generation of waveforms (e.g. pulse width modulated signals) on `TIMER_1` and/or `TIMER_2` pins respectively. The pins need to be configured for 'Bypass Mode' in the GPIO module and the `nWave_en` bit in the corresponding `Counter_Control` register of the timer module needs to be set to active 'low'.

Three parameters define the waveform output:

1. the waveform period
2. the length of the first phase of the duty cycle
3. the polarity of the waveform

To specify a waveform period of M (prescaled) `PCLK` cycles, a value of $M-1$ is programmed into the timer's Interval register and the values for `PS_V` and `PS_En` in the `Counter_Clock` register are chosen as desired.

The length $L1$ of the first phase of the duty cycle in (prescaled) `PCLK` cycles is specified by programming the `Match#1` register of the timer with the value $L1$ (it is recommended to program `Match#2` and `Match#3` with $FFFF_{\text{hex}}$).

The waveform polarity is selected by programming the `Wave_pol` bit in the `Counter_Control` register. If `Wave_pol` is '0', the waveform duty cycle will start with output 'low'. After $L1$ prescaled `PCLK` cycles the output switches to 'high'. After another $(M-L1)$ prescaled `PCLK` cycles the output switches back to 'low' and the next output period starts. If `Wave_pol` is '1', the output starts with the 'high' phase of the period.

The `Counter_Control` register should be set up for Match Mode (`Match='1'`) and Interval Mode (`INT='1'`) and Waveform Output enabled (`nWave_en='0'`) along with the start command for the timer's counter (`RST='1'`).

Values for $M-1$ and $L1$ may be changed dynamically during operation.

2.6.5.6. TTC Register Descriptions

The TTC registers are summarized in Table 2–24. Addresses refer to the offsets from the TTC base address of 001E.4000_{hex}.

A more detailed description of the register bits is shown in Table 2–25 on page 85.

Table 2–24: TTC Register Map

Offset	R/W	Width	Name	Comment	Default
00 _{hex}	R/W	7	Counter 1 Clock	Prescale and clock source control	00 _{hex}
04 _{hex}	R/W	7	Counter 2 Clock	Prescale and clock source control	00 _{hex}
08 _{hex}	R/W	7	Counter 3 Clock	Prescale control	00 _{hex}
0C _{hex}	R/W	7	Counter 1 Control	Operational mode and reset	00 _{hex}
10 _{hex}	R/W	7	Counter 2 Control	Operational mode and reset	00 _{hex}
14 _{hex}	R/W	7	Counter 3 Control	Operational mode and reset	00 _{hex}
18 _{hex}	R	16	Counter 1 Value	Current counter value	00 _{hex}
1C _{hex}	R	16	Counter 2 Value	Current counter value	00 _{hex}
20 _{hex}	R	16	Counter 3 Value	Current counter value	00 _{hex}
24 _{hex}	R/W	16	Counter 1 Interval	Interval value	00 _{hex}
28 _{hex}	R/W	16	Counter 2 Interval	Interval value	00 _{hex}
2C _{hex}	R/W	16	Counter 3 Interval	Interval value	00 _{hex}
30 _{hex}	R/W	16	Counter 1 match #1	Match value	00 _{hex}
34 _{hex}	R/W	16	Counter 2 match #1	Match value	00 _{hex}
38 _{hex}	R/W	16	Counter 3 match #1	Match value	00 _{hex}
3C _{hex}	R/W	16	Counter 1 match #2	Match value	00 _{hex}
40 _{hex}	R/W	16	Counter 2 match #2	Match value	00 _{hex}
44 _{hex}	R/W	16	Counter 3 match #2	Match value	00 _{hex}
48 _{hex}	R/W	16	Counter 1 match #3	Match value	00 _{hex}
4C _{hex}	R/W	16	Counter 2 match #3	Match value	00 _{hex}
50 _{hex}	R/W	16	Counter 3 match #3	Match value	00 _{hex}
54 _{hex}	R	6	Interrupt Register 1	Counter 1 Interval, Match, Overflow and Event interrupts	00 _{hex}
58 _{hex}	R	6	Interrupt Register 2	Counter 2 Interval, Match, Overflow and Event interrupts	00 _{hex}
5C _{hex}	R	6	Interrupt Register 3	Counter 3 Interval, Match, Overflow and Event interrupts	00 _{hex}

Table 2–24: TTC Register Map, continued

Offset	R/W	Width	Name	Comment	Default
60 _{hex}	R/W	6	Interrupt Enable 1	ANDed with Interrupt Register 1	00 _{hex}
64 _{hex}	R/W	6	Interrupt Enable 2	ANDed with Interrupt Register 2	00 _{hex}
68 _{hex}	R/W	6	Interrupt Enable 3	ANDed with Interrupt Register 3	00 _{hex}
6C _{hex}	R/W	3	Timer1 Event Control	Enable, pulse and overflow	0 _{hex}
70 _{hex}	R/W	3	Timer2 Event Control	Enable, pulse and overflow	0 _{hex}
74 _{hex}	R/W	3	Timer3 Event Control	Enable, pulse and overflow	0 _{hex}
78 _{hex}	R	16	Timer1 Event Register	PCLK cycle count for event	00 _{hex}
7C _{hex}	R	16	Timer2 Event Register	PCLK cycle count for event	00 _{hex}
80 _{hex}	R	16	Timer3 Event Register	PCLK cycle count for event	00 _{hex}

Table 2–25: TTC Read/Write Registers

Register Address	Function	Name												
00 _{hex} 04 _{hex} 08 _{hex}	<p>Counter_Clock Register, Reset 00_{hex}</p> <table border="1" data-bbox="288 421 1203 495"> <tr> <td>Bit</td> <td>31:7</td> <td>6</td> <td>5</td> <td>4:1</td> <td>0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>Ex_E</td> <td>C_Src</td> <td>PS_V</td> <td>PS_En</td> </tr> </table> <p>Prescale and clock source control</p> <p>bit[31:7] Reserved</p> <p>bit[6] Ex_E External Clock Edge: when this bit is set and the external clock is selected, the counter clocks on the falling edge of the external clock input.</p> <p>bit[5] C_Src Clock Source: when this bit is set the counter uses the external clock input, EXT_CLK, fed into pin TIMER_x¹⁾; the default clock source is on-chip PCLK (this bit must be set to 0 for timer 3).</p> <p>bit[4:1] 0 PS_V Prescale value (N): if prescale is enabled, the count rate is divided by 2^{N+1} (÷2 to ÷65536). ... 15_{dec}</p> <p>bit[0] PS_En Prescale enable: when this bit is set the counter, clock source is prescaled; the default clock source is that defined by C_Src.</p> <p>¹⁾The corresponding GPIO port pin must be configured for 'Bypass mode'</p>	Bit	31:7	6	5	4:1	0	Field	Reserved	Ex_E	C_Src	PS_V	PS_En	COUNTER_CLOCK
Bit	31:7	6	5	4:1	0									
Field	Reserved	Ex_E	C_Src	PS_V	PS_En									

Table 2–25: TTC Read/Write Registers, continued

Register Address	Function	Name																		
0C _{hex} 10 _{hex} 14 _{hex}	<p>Counter_Control Register, Reset 21_{hex}</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <th style="width: 10%;">Bit</th> <th style="width: 60%;">31:7</th> <th style="width: 5%;">6</th> <th style="width: 5%;">5</th> <th style="width: 5%;">4</th> <th style="width: 5%;">3</th> <th style="width: 5%;">2</th> <th style="width: 5%;">1</th> <th style="width: 5%;">0</th> </tr> <tr> <th>Field</th> <td style="background-color: #e0e0e0;">Reserved</td> <td>Wave_pol</td> <td>nWave_en</td> <td>RST</td> <td>Match</td> <td>DEC</td> <td>INT</td> <td>DIS</td> </tr> </table> <p>Operational mode and reset.</p> <p>bit[31:7] Reserved</p> <p>bit[6] Wave_pol Waveform polarity: When this bit is high, the waveform output goes from high to low on Match_1 interrupt and returns high on overflow or interval interrupt; when low, the waveform goes from low to high on Match_1 interrupt and returns low on overflow or interval interrupt.¹⁾</p> <p>bit[5] nWave_en Output waveform enable, active low¹⁾.</p> <p>bit[4] RST Setting this bit high resets the counter value and restarts counting; the RST bit is automatically cleared on restart.</p> <p>bit[3] Match Register Match mode: when Match is set, an interrupt is generated when the count value matches one of the three match registers and the corresponding bit is set in the Interrupt Enable register.</p> <p>bit[2] DEC Decrement: when this bit is high the counter counts down, otherwise it counts up.</p> <p>bit[1] INT When this bit is high, the timer is in Interval Mode, and the counter generates interrupts at regular intervals; when low, the timer is in overflow mode.</p> <p>bit[0] DIS Disable counter: when this bit is high, the counter is stopped, holding its last value until reset restarted or enabled again.</p> <p style="text-align: center;">¹⁾ See footnote in Counter_Clock register on page 85.</p>	Bit	31:7	6	5	4	3	2	1	0	Field	Reserved	Wave_pol	nWave_en	RST	Match	DEC	INT	DIS	COUNTER_CONTROL
Bit	31:7	6	5	4	3	2	1	0												
Field	Reserved	Wave_pol	nWave_en	RST	Match	DEC	INT	DIS												
18 _{hex} 1C _{hex} 20 _{hex}	<p>Counter_Value Register, Reset 0000_{hex}</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <th style="width: 10%;">Bit</th> <th style="width: 45%;">31:16</th> <th style="width: 45%;">15:0</th> </tr> <tr> <th>Field</th> <td style="background-color: #e0e0e0;">Reserved</td> <td>Value</td> </tr> </table> <p>At any time, a Timer Counter's count value can be read from its Counter Value Register.</p> <p>bit[31:16] Reserved</p> <p>bit[15:0] Value: Current Counter Value</p>	Bit	31:16	15:0	Field	Reserved	Value	COUNTER_VALUE												
Bit	31:16	15:0																		
Field	Reserved	Value																		

Table 2–25: TTC Read/Write Registers, continued

Register Address	Function	Name																
24 _{hex} 28 _{hex} 2C _{hex}	<p>Interval Register, Reset 0000_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:16</td> <td>15:0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>Interval</td> </tr> </table> <p>If Interval mode is enabled (see Counter_Control registers), this is the maximum value that the counter will count up to or down from.</p> <p>bit[31:16] Reserved</p> <p>bit[15:0] Interval</p>	Bit	31:16	15:0	Field	Reserved	Interval	INTERVAL										
Bit	31:16	15:0																
Field	Reserved	Interval																
30 _{hex} (T1,#1) 34 _{hex} (T2,#1) 38 _{hex} (T3,#1) 3C _{hex} (T1,#2) 40 _{hex} (T2,#2) 44 _{hex} (T3,#2) 48 _{hex} (T1,#3) 4C _{hex} (T2,#3) 50 _{hex} (T3,#3)	<p>Match Register, Reset 0000_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:16</td> <td>15:0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>Match</td> </tr> </table> <p>When a counter has the same value as is stored in one of its match registers and match mode is enabled, a match interrupt is generated. Each counter has three match registers. The Match#1 register values are also used to specify the duty cycle on PWM output.</p> <p>bit[31:16] Reserved</p> <p>bit[15:0] Match: The value to be compared to current counter value, either for match interrupt generation or output level transition on PWM output.</p>	Bit	31:16	15:0	Field	Reserved	Match	MATCH										
Bit	31:16	15:0																
Field	Reserved	Match																
54 _{hex} 58 _{hex} 5C _{hex}	<p>Interrupt Register, Reset 00_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:6</td> <td>5</td> <td>4</td> <td>3</td> <td>2</td> <td>1</td> <td>0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>Ev</td> <td>Ov</td> <td>M3</td> <td>M2</td> <td>M1</td> <td>Iv</td> </tr> </table> <p>Status register for Interval, Match, Overflow and Event interrupts</p> <p>bit[31:6] Reserved</p> <p>bit[5] Ev Event timer overflow interrupt</p> <p>bit[4] Ov Counter overflow</p> <p>bit[3] M3 Match #3 interrupt</p> <p>bit[2] M2 Match #2 interrupt</p> <p>bit[1] M1 Match #1 interrupt</p> <p>bit[0] Iv Interval interrupt</p>	Bit	31:6	5	4	3	2	1	0	Field	Reserved	Ev	Ov	M3	M2	M1	Iv	INTERRUPT
Bit	31:6	5	4	3	2	1	0											
Field	Reserved	Ev	Ov	M3	M2	M1	Iv											

Table 2–25: TTC Read/Write Registers, continued

Register Address	Function	Name																
60 _{hex} 64 _{hex} 68 _{hex}	<p>Interrupt Enable Register, Reset 00_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:6</td> <td>5</td> <td>4</td> <td>3</td> <td>2</td> <td>1</td> <td>0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td colspan="6">IEN</td> </tr> </table> <p>Enables for bits 5:0 in Interrupt Register. Corresponding bits must be set to enable the interrupt.</p> <p>bit[31:6] Reserved</p> <p>bit[5:0] IEN</p>	Bit	31:6	5	4	3	2	1	0	Field	Reserved	IEN						INTERRUPT_ENABLE
Bit	31:6	5	4	3	2	1	0											
Field	Reserved	IEN																
6C _{hex} 70 _{hex} 74 _{hex}	<p>Timer_Event_Control Register, Reset 00_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:3</td> <td>2</td> <td>1</td> <td>0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>E_Ov</td> <td>E_Lo</td> <td>E_En</td> </tr> </table> <p>bit[31:3] Reserved</p> <p>bit[2] E_Ov Enable timer: when this bit is high, the event timer is enabled.</p> <p>bit[1] E_Lo When this bit is high, the timer counts PCLK cycles during the low-level duration of EXT_CLK; when low, the event timer counts the high level duration of EXT_CLK.</p> <p>bit[0] E_En When this bit is low, the event timer is disabled and set to zero when an Event_Timer register overflow occurs; when set to high, the timer continues counting on overflow.</p>	Bit	31:3	2	1	0	Field	Reserved	E_Ov	E_Lo	E_En	TIMER_EVENT_CONTROL						
Bit	31:3	2	1	0														
Field	Reserved	E_Ov	E_Lo	E_En														
78 _{hex} 7C _{hex} 80 _{hex}	<p>Timer_Event Register, Reset 00_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:16</td> <td>15:0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>Event</td> </tr> </table> <p>This register stores the result of the PCLK count during the EXT_CLK high or low pulse, depending on the E_Lo bit in the Event_Counter_Control register.</p> <p>bit[31:16] Reserved</p> <p>bit[15:0] Event: Number of PCLK periods during high or low phase of EXT_CLK. (TIMER_x pin)</p>	Bit	31:16	15:0	Field	Reserved	Event	TIMER_EVENT										
Bit	31:16	15:0																
Field	Reserved	Event																

2.6.6. Real Time Clock

The Real Time Clock (RTC) Module provides programmable time of day, calendar and alarm functions for use within the PUC 303xA. Note, however, that the RTC is not available in some package options.

2.6.6.1. Features

- Complete time of day clock: 12/24 hour, hours, minutes, and seconds
- Calendar function: day of week, date of month, month, year, century, leap year compensation, and year 2000 compliant
- alarm function: month, date, hour, minute, and seconds resolution
- Event function: can set interrupt for every roll over of month, day, hour, minute, or second.

2.6.6.2. Description

The RTC module keeps track of time of day, to an accuracy of one second, and functions as a calendar keeping track of the day, month and year. The RTC comprises a binary coded decimal counter with the following data:

- Year from 1900 to 2999
- Month from 1 to 12
- Date from 1 to 28, 29, 30 or 31 (as a function of month and year)
- Day of week from 1 to 7 (mapping to date is not fixed)
- Hour from 0 to 23, or from 1 to 12 with the AM/PM flag set
- Minute from 0 to 59
- Second from 0 to 59.

An alarm register facilitates comparison of month, date, hour, minute and second. Each of these may be masked (that is forced to equality), so that, for example, an alarm can be generated at 12:37 on the 15th of every month. An interrupt can be generated on an alarm event.

Event detection is also available. This will generate an interrupt every time the time or calendar register rolls over into a new month, date, hour, minute, or second.

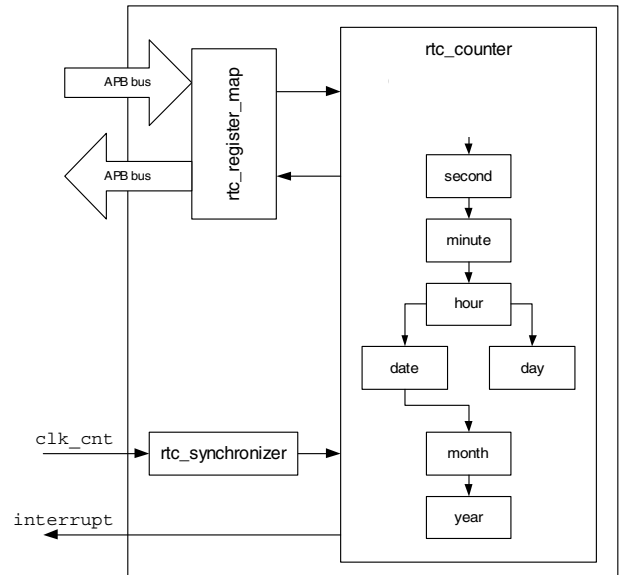


Fig. 2-20: Block Diagram

2.6.6.3. Standby Mode

The RTC module can be operated during STANDBY mode by ensuring that the OSCRTCCLK and RTCCLK are enabled before entering STANDBY.

If unmasked within the PMU_WAKE_MASK register, an interrupt from the RTC will wake up the device.

2.6.6.4. Register Map and Formats

The register map lists the Offset, Name, Access, and Description associated with the RTC registers. In addition, the number of bits allocated to each register is listed (see Table 2-26).

The offset addresses listed are relative to the RTC base address, 001E.C000_{hex}.

Table 2–26: RTC Register Map

Offset	Name	Access	Bits	Description	Default
00 _{hex}	Control	R/W	1:0	Start and stop for timing and calendar	00 _{hex}
04 _{hex}	12/24 Hour	R/W	0:0	Set 12hr./24hr. mode	0 _{hex}
08 _{hex}	Time	R/W, R	30:8	Contains time values and the data valid flag	0000.0000 _{hex}
0C _{hex}	Calendar	R/W, R	29:0	Contains date values and the data valid flag	2000.010F _{hex}
10 _{hex}	Time alarm	R/W	30:8	Contains time values for alarm	0000.0000 _{hex}
14 _{hex}	Calendar alarm	R/W	13:3	Contains date values for alarm	0000 _{hex}
18 _{hex}	Alarm Enable	R/W	5:1	Enables alarming on: month, date, hour, minute and second	00 _{hex}
1C _{hex}	Event Flags	R	6:1	Flags for alarm and roll-over events	00 _{hex}
20 _{hex}	Interrupt Enable	W	6:1	Enables alarm and roll-over events	N/A
24 _{hex}	Interrupt Disable	W	6:1	Disables alarm and roll-over events	N/A
28 _{hex}	Interrupt mask	R	6:1	Masks off alarm and roll-over events	7E _{hex}
2C _{hex}	Status	R/W	3:0	Valid data flags	F _{hex}

Each register address is defined by an offset from the RTC base address. The Interrupt-Enable and -Disable registers are write only.

Table 2–27: RTC Read/Write Registers

Register Address	Function	Name								
00 _{hex}	<p>Control Register, Reset 00_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:2</td> <td>1</td> <td>0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>CAL</td> <td>TIME</td> </tr> </table> <p>The Control Register is used to stop the RTC. Use this facility to update the time or calendar without affecting the other.</p> <p>bit[31:2] Reserved</p> <p>bit[1] CAL When this field is set high, the RTC stops incrementing the Calendar value.</p> <p>bit[0] TIME When this field is set high the RTC stops incrementing the Time value.</p>	Bit	31:2	1	0	Field	Reserved	CAL	TIME	CONTROL
Bit	31:2	1	0							
Field	Reserved	CAL	TIME							

Table 2–27: RTC Read/Write Registers, continued

Register Address	Function	Name																								
04 _{hex}	<p>12/24 Hour Register, Reset 00_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:01</td> <td>0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>12/24</td> </tr> </table> <p>The Control Register is used to stop the RTC. Use this facility to update the time or calendar without affecting the other.</p> <p>bit[31:01] Reserved</p> <p>bit[0] 12/24 When this field is set high, the RTC operates in 12-hour clock mode; otherwise, it operates in 24-hour mode.</p>	Bit	31:01	0	Field	Reserved	12/24	12/24_HOUR																		
Bit	31:01	0																								
Field	Reserved	12/24																								
08 _{hex}	<p>Time Register, On Reset 0000.0000_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31</td> <td>30</td> <td>29:28</td> <td>27:24</td> <td>23</td> <td>22:20</td> <td>19:16</td> <td>15</td> <td>14:12</td> <td>11:8</td> <td>7:0</td> </tr> <tr> <td>Field</td> <td>Re-served</td> <td>PM</td> <td>HR_T</td> <td>HR_U</td> <td>Re-served</td> <td>M_T</td> <td>M_U</td> <td>Re-served</td> <td>S_T</td> <td>S_U</td> <td>Reserved</td> </tr> </table> <p>This register is loaded using the APB bus. Once the register has been used for writing data to the RTC it is then used for storing read data from the counters.</p> <p>bit[31] Reserved</p> <p>bit[30] PM AM/PM Select: in 12 hour clock mode, indicates PM when set.</p> <p>bit[29:28] BCD HR_T Hours – tens BCD digit (0–2)</p> <p>bit[27:24] BCD HR_U Hours – units BCD digit (0–9)</p> <p>bit[23] Reserved</p> <p>bit[22:20] BCD M_T Minutes – tens BCD digit (0–5)</p> <p>bit[19:16] BCD M_U Minutes – units BCD digit (0–9)</p> <p>bit[15] Reserved</p> <p>bit[14:12] BCD S_T Seconds – tens BCD digit (0–9)</p> <p>bit[11:8] BCD S_U Seconds – units BCD digit (0–9)</p> <p>bit[7:0] Reserved</p>	Bit	31	30	29:28	27:24	23	22:20	19:16	15	14:12	11:8	7:0	Field	Re-served	PM	HR_T	HR_U	Re-served	M_T	M_U	Re-served	S_T	S_U	Reserved	TIME
Bit	31	30	29:28	27:24	23	22:20	19:16	15	14:12	11:8	7:0															
Field	Re-served	PM	HR_T	HR_U	Re-served	M_T	M_U	Re-served	S_T	S_U	Reserved															

Table 2–27: RTC Read/Write Registers, continued

Register Address	Function											Name
0C _{hex}	Calendar Register, Reset 2000.010F _{hex}											CALENDAR
	Bit	31:30	29:28	27:24	23:20	19:16	15:14	13:12	11:8	7	6:3	2:0
	Field	Re-served	C_T	C_U	Y_T	Y_U	Re-served	D_T	D_U	M_T	M_U	DAY
<p>This register holds calendar data. Once the register has been used for writing data to the RTC it is then used for storing read data from the RTC counters.</p> <p>bit[31:30] Reserved</p> <p>bit[29:28] BCD C_T Century – tens BCD digit (1–2)</p> <p>bit[27:24] BCD C_U Century – units BCD digit (0–9)</p> <p>bit[23:20] BCD Y_T Year – tens BCD digit (0–9)</p> <p>bit[19:16] BCD Y_U Year – units BCD digit (0–9)</p> <p>bit[15:14] Reserved</p> <p>bit[13:12] BCD D_T Date – tens BCD digit (0–3)</p> <p>bit[11:8] BCD D_U Date– units BCD digit (0–9)</p> <p>bit[7] BCD M_T Month – tens BCD digit (0–1)</p> <p>bit[6:3] BCD M_U Month – units BCD digit (0–9)</p> <p>bit[2:0] BCD DAY Day of the week (arbitrary) – units BCD digit (1–7)</p>												

Table 2–27: RTC Read/Write Registers, continued

Register Address	Function	Name																								
10 _{hex}	<p>Time_Alarm Register, On Reset 0000.0000_{hex}</p> <table border="1"> <thead> <tr> <th>Bit</th> <th>31</th> <th>30</th> <th>29:28</th> <th>27:24</th> <th>23</th> <th>22:20</th> <th>19:16</th> <th>15</th> <th>14:12</th> <th>11:8</th> <th>7:0</th> </tr> </thead> <tbody> <tr> <th>Field</th> <td>Re-served</td> <td>PM</td> <td>HR_T</td> <td>HR_U</td> <td>Re-served</td> <td>M_T</td> <td>M_U</td> <td>Re-served</td> <td>S_T</td> <td>S_U</td> <td>Reserved</td> </tr> </tbody> </table> <p>The Time alarm Register is used to program in a specific time when the alarm should cause an event, down to one hundredth of a second. The register map layout for the Time alarm Register is exactly the same as the Time Register.</p> <p>bit[31] Reserved</p> <p>bit[30] PM AM/PM Select: in 12 hour clock mode, indicates PM when set.</p> <p>bit[29:28] BCD HR_T Hours – tens BCD digit (0–2)</p> <p>bit[27:24] BCD HR_U Hours – units BCD digit (0–9)</p> <p>bit[23] Reserved</p> <p>bit[22:20] BCD M_T Minutes – tens BCD digit (0–5)</p> <p>bit[19:16] BCD M_U Minutes – units BCD digit (0–9)</p> <p>bit[15] Reserved</p> <p>bit[14:12] BCD S_T Seconds – tens BCD digit (0–9)</p> <p>bit[11:8] BCD S_U Seconds – units BCD digit (0–9)</p> <p>bit[7:0] Reserved</p>	Bit	31	30	29:28	27:24	23	22:20	19:16	15	14:12	11:8	7:0	Field	Re-served	PM	HR_T	HR_U	Re-served	M_T	M_U	Re-served	S_T	S_U	Reserved	TIME_ALARM
Bit	31	30	29:28	27:24	23	22:20	19:16	15	14:12	11:8	7:0															
Field	Re-served	PM	HR_T	HR_U	Re-served	M_T	M_U	Re-served	S_T	S_U	Reserved															

Table 2–27: RTC Read/Write Registers, continued

Register Address	Function	Name																
14 _{hex}	<p>Calendar_Alarm Register; Reset 0000_{hex}</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 5%;">Bit</td> <td style="width: 20%;">31:14</td> <td style="width: 5%;">13:12</td> <td style="width: 5%;">11:8</td> <td style="width: 5%;">7</td> <td style="width: 5%;">6:3</td> <td style="width: 5%;">2:0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>D_T</td> <td>D_U</td> <td>M_T</td> <td>M_U</td> <td>Reserved</td> </tr> </table> <p>The Calendar alarm Register can be used to program a specific date and month when the alarm should cause an event. It is not possible to set an alarm more than one year into the future.</p> <p>bit[31:14] Reserved</p> <p>bit[13:12] BCD D_T Date – tens BCD digit (0–3)</p> <p>bit[11:8] BCD D_U Date – units BCD digit (0–9)</p> <p>bit[7] BCD M_T Month – tens BCD digit (0–1)</p> <p>bit[6:3] BCD M_U Month – units BCD digit (0–9)</p> <p>bit[2:0] Reserved</p>	Bit	31:14	13:12	11:8	7	6:3	2:0	Field	Reserved	D_T	D_U	M_T	M_U	Reserved	CALENDAR_ALARM		
Bit	31:14	13:12	11:8	7	6:3	2:0												
Field	Reserved	D_T	D_U	M_T	M_U	Reserved												
18 _{hex}	<p>Alarm_Enable Register, Reset 00_{hex}</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 5%;">Bit</td> <td style="width: 20%;">31:6</td> <td style="width: 5%;">5</td> <td style="width: 5%;">4</td> <td style="width: 5%;">3</td> <td style="width: 5%;">2</td> <td style="width: 5%;">1</td> <td style="width: 5%;">0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>MONTH</td> <td>DATE</td> <td>HOUR</td> <td>MIN</td> <td>SEC</td> <td>Reserved</td> </tr> </table> <p>The Alarm Enable Register is used to select the sources that can trigger an alarm. Setting a bit enables the corresponding time unit as a trigger event. The actual time values are programmed in the Time_Alarm register. The alarm triggering causes an event to be generated which is set in the Event_Flag register. For example if all the fields are enabled then the alarm will trigger when a particular month, date, hour, minute, and second has been reached. If only the minutes field is enabled then the alarm will trigger when a particular minute is reached, and trigger every hour at that minute.</p> <p>bit[31:6] Reserved</p> <p>bit[5] MONTH trigger enable</p> <p>bit[4] DATE trigger enable</p> <p>bit[3] HOUR trigger enable</p> <p>bit[2] MIN. trigger enable</p> <p>bit[1] SEC. trigger enable</p> <p>bit[0] Reserved</p>	Bit	31:6	5	4	3	2	1	0	Field	Reserved	MONTH	DATE	HOUR	MIN	SEC	Reserved	ALARM_ENABLE
Bit	31:6	5	4	3	2	1	0											
Field	Reserved	MONTH	DATE	HOUR	MIN	SEC	Reserved											

Table 2–27: RTC Read/Write Registers, continued

Register Address	Function	Name																		
1C _{hex}	<p>Event_Flag Register, Reset 00_{hex}</p> <table border="1" data-bbox="288 421 1201 517"> <tr> <td>Bit</td> <td>31:7</td> <td>6</td> <td>5</td> <td>4</td> <td>3</td> <td>2</td> <td>1</td> <td>0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>ALARM</td> <td>MONTH</td> <td>DATE</td> <td>HOUR</td> <td>MIN</td> <td>SEC</td> <td>Re-served</td> </tr> </table> <p>The Event Flags Register is used to indicate that an event has occurred since the last reset. The register may be set even if the corresponding Alarm_Enable bit is not set. There are six event bits that are written to when the corresponding event occurs.</p> <p>For example, if a minute rolls over, that is 59 seconds becomes 00 seconds, then the Minute event bit will be set. The Event_Flag Register fields are cleared whenever their status is read. The Event_Flag Register is read-only and should not be written to.</p> <p>If an event occurs at the same time as the Event_Flag Register is being read, the event will take priority. That means that the clear does not take place in this case.</p> <p>bit[31:7] Reserved</p> <p>bit[6] ALARM</p> <p>bit[5] MONTH</p> <p>bit[4] DATE</p> <p>bit[3] HOUR</p> <p>bit[2] MIN.</p> <p>bit[1] SEC.</p> <p>bit[0] Reserved</p>	Bit	31:7	6	5	4	3	2	1	0	Field	Reserved	ALARM	MONTH	DATE	HOUR	MIN	SEC	Re-served	EVENT_FLAG
Bit	31:7	6	5	4	3	2	1	0												
Field	Reserved	ALARM	MONTH	DATE	HOUR	MIN	SEC	Re-served												
20 _{hex}	<p>Interrupt_Enable Register, Reset N/A</p> <table border="1" data-bbox="288 1294 1201 1391"> <tr> <td>Bit</td> <td>31:7</td> <td>6</td> <td>5</td> <td>4</td> <td>3</td> <td>2</td> <td>1</td> <td>0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>ALARM</td> <td>MONTH</td> <td>DATE</td> <td>HOUR</td> <td>MIN</td> <td>SEC</td> <td>Re-served</td> </tr> </table> <p>The Interrupt Enable Register is used to set which events can generate an interrupt. An interrupt is enabled by writing a 1 to the appropriate Enable field. The interrupt generated will stay set until it is cleared, either by disabling the interrupt or by reading/clearing the Event_Flag register.</p> <p>bit[31:7] Reserved</p> <p>bit[6] ALARM</p> <p>bit[5] MONTH</p> <p>bit[4] DATE</p> <p>bit[3] HOUR</p> <p>bit[2] MIN.</p> <p>bit[1] SEC.</p> <p>bit[0] Reserved</p>	Bit	31:7	6	5	4	3	2	1	0	Field	Reserved	ALARM	MONTH	DATE	HOUR	MIN	SEC	Re-served	INTERRUPT_ENABLE
Bit	31:7	6	5	4	3	2	1	0												
Field	Reserved	ALARM	MONTH	DATE	HOUR	MIN	SEC	Re-served												

Table 2–27: RTC Read/Write Registers, continued

Register Address	Function	Name																		
24 _{hex}	<p>Interrupt_Disable Register, Reset N/A</p> <table border="1"> <tr> <td>Bit</td> <td>31:7</td> <td>6</td> <td>5</td> <td>4</td> <td>3</td> <td>2</td> <td>1</td> <td>0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>ALARM</td> <td>MONTH</td> <td>DATE</td> <td>HOUR</td> <td>MIN</td> <td>SEC</td> <td>Re-served</td> </tr> </table> <p>The Interrupt Disable Register is used to disable interrupts that are already enabled. An interrupt is disabled by writing a 1 to the appropriate Disable Register field. Any interrupts that are already set will remain set.</p> <p>bit[31:7] Reserved</p> <p>bit[6] ALARM</p> <p>bit[5] MONTH</p> <p>bit[4] DATE</p> <p>bit[3] HOUR</p> <p>bit[2] MIN.</p> <p>bit[1] SEC.</p> <p>bit[0] Reserved</p>	Bit	31:7	6	5	4	3	2	1	0	Field	Reserved	ALARM	MONTH	DATE	HOUR	MIN	SEC	Re-served	INTERRUPT_DISABLE
Bit	31:7	6	5	4	3	2	1	0												
Field	Reserved	ALARM	MONTH	DATE	HOUR	MIN	SEC	Re-served												
28 _{hex}	<p>Interrupt_Mask Register, Reset 7E_{hex}</p> <table border="1"> <tr> <td>Bit</td> <td>31:7</td> <td>6</td> <td>5</td> <td>4</td> <td>3</td> <td>2</td> <td>1</td> <td>0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>ALARM</td> <td>MONTH</td> <td>DATE</td> <td>HOUR</td> <td>MIN</td> <td>SEC</td> <td>Re-served</td> </tr> </table> <p>The Interrupt Mask Register indicates which interrupts are currently set. A field being set indicates that the corresponding interrupt is enabled. The Interrupt Mask Register is read-only, and should not be written to.</p> <p>bit[31:7] Reserved</p> <p>bit[6] ALARM</p> <p>bit[5] MONTH</p> <p>bit[4] DATE</p> <p>bit[3] HOUR</p> <p>bit[2] MIN.</p> <p>bit[1] SEC.</p> <p>bit[0] Reserved</p>	Bit	31:7	6	5	4	3	2	1	0	Field	Reserved	ALARM	MONTH	DATE	HOUR	MIN	SEC	Re-served	INTERRUPT_MASK
Bit	31:7	6	5	4	3	2	1	0												
Field	Reserved	ALARM	MONTH	DATE	HOUR	MIN	SEC	Re-served												

Table 2–27: RTC Read/Write Registers, continued

Register Address	Function	Name												
2C _{hex}	<p>Status Register, Reset F_{hex}</p> <table border="1" data-bbox="288 421 1201 499"> <tr> <td data-bbox="288 421 341 456">Bit</td> <td data-bbox="341 421 986 456">31:4</td> <td data-bbox="986 421 1038 456">3</td> <td data-bbox="1038 421 1091 456">2</td> <td data-bbox="1091 421 1144 456">1</td> <td data-bbox="1144 421 1201 456">0</td> </tr> <tr> <td data-bbox="288 456 341 499">Field</td> <td data-bbox="341 456 986 499">Reserved</td> <td data-bbox="986 456 1038 499">VCA</td> <td data-bbox="1038 456 1091 499">VTA</td> <td data-bbox="1091 456 1144 499">VC</td> <td data-bbox="1144 456 1201 499">VT</td> </tr> </table> <p>The Status Register contains 'data valid' flags for data that has been input to the RTC. Invalid entry checking is performed after data is written in and when data is copied back from the counters to the register map. The appropriate field bit is set if the data is valid. The counter will not start if any of the valid flags is not set.</p> <p>The Status Register is read-only and should not be written to.</p> <p>bit[31:4] Reserved</p> <p>bit[3] Valid calendar alarm (VCA)</p> <p>bit[2] Valid time alarm (VTA)</p> <p>bit[1] Valid calendar (VC)</p> <p>bit[0] Valid time (VT)</p>	Bit	31:4	3	2	1	0	Field	Reserved	VCA	VTA	VC	VT	STATUS
Bit	31:4	3	2	1	0									
Field	Reserved	VCA	VTA	VC	VT									

2.6.6.5. Programming Instructions

2.6.6.5.1. Setting Time

The time counter must be stopped before setting a new time. The RTC must then be set into the desired 12-hr or 24-hr mode. The RTC can then be set to the desired time. As the RTC timer will not start if an attempt is made to load invalid data it is recommended that the status register be read to check this.

For example to set the RTC to 12:34:56 pm, the following sequence of register accesses should take place (see Example 2–2 on page 99).

2.6.6.5.2. Setting Date

The date counter must be stopped before setting a new date. The RTC can then be set to a new date. The day of the week can also be set at this time. There is no defined mapping of the day of the week to the counter value. As the RTC timer will stop if an attempt is made to set an invalid date, it is recommended that the status register be read to check this.

For example to set the RTC to Sunday 4th March 2001 with the mapping 1=Monday etc., the following sequence of register accesses should take place (see Example 2–3 on page 99).

2.6.6.5.3. Setting Both Together

It is possible to perform both together. For example to set the RTC to 12:34:56 pm, on Sunday 4th March 2001, with the mapping, 1=Monday etc. the following sequence of register accesses should take place (see Example 2–4 on page 99).

2.6.6.5.4. Setting Alarm to Specific Date and Time

Before an alarm can be changed, the interrupt must be disabled. Then the required date and time should be written into the various alarm registers. Note that if the RTC time is set to 12 hour mode, then the alarm must be set in the same mode.

For example to set the alarm to 23:55:00 on the 31st December the following sequence of register accesses should take place (see Example 2–5 on page 99).

2.6.6.5.5. Setting Alarm to Regular Time

It is possible to set the alarm to repeat every month, day, hour, minute or second.

For example to set the alarm to repeat at 5 minutes past every hour the following sequence of register accesses should take place (see Example 2–6 on page 100).

For example to set the alarm to repeat at 11:59:59 every day the following sequence of register accesses should take place (see Example 2–7 on page 100).

2.6.6.5.6. Setting Events

It is possible to set an event to occur on each change of month, day etc. This is done by writing the appropriate bit in the interrupt enable register.

For example to set a daily event (at 12:00:00 am) execute the sequence of register accesses from Example 2–8 .

2.6.6.5.7. Interrupts

Whenever an alarm or event trigger has occurred, the RTC raises its interrupt line to the system interrupt controller.

Once the software has decoded the interrupt as one from the RTC, it is possible to further decode the interrupt by reading the Event Flags register. A read of this register will return a bit for each alarm or event trigger that has occurred since the register was previously read, and will clear those bits in the register.

Example 2–2: Setting Time

```

write( Control,          0x00000001 ) // stop time counter
write( 12/24 hour,      0x00000001 ) // 12 hr mode
write( Time,            0x52345600 ) // 1.1.2.3.4.5.6. (see Time Register)
read ( Status,          result )
if ( result !=          0x0000000f ) then
    error
else
    write ( Control, 0x00000000 ) // restart time counter

```

Example 2–3: Setting Date

```

write( Control,          0x00000002 ) // stop calendar counter
write( Calendar,        0x2001041f ) // 2.0.0.1.0.4.0.3.7 (see Calendar Register)
read ( Status,          result )
if ( result !=          0x0000000f ) then
    error
else
    write ( Control, 0x00000000 ) // restart calendar counter

```

Example 2–4: Setting Both Together

```

write( Control,          0x00000003 ) // stop both counters
write( 12/24 hour,      0x00000001 ) // 12 hr mode
write( Time,            0x52345600 ) // 1.1.2.3.4.5.6 (see Time Register)
write( Calendar,        0x2001041f ) // 2.0.0.1.0.4.0.3.7 (see Calendar Register)
read ( Status,          result )
if ( result !=          0x0000000f ) then
    error
else
    write ( Control, 0x00000000 ) // restart calendar counter

```

Example 2–5: Setting Alarm to Specific Date and Time

```

write(Interrupt Disable, 0x00000040) // disable alarm
write(Time_Alarm,        0x23550000) // set time
write(Calendar_Alarm,    0x00003190) // set calendar
write(Alarm_Enable,     0x0000003f) // enable all time & date fields
read ( Status,          result )
if ( result !=          0x0000000f ) then
    error
else
    write(interrupt enable,0x00000040) // enable alarm

```

Example 2–6: Setting Alarm to Regular Time

```
write(Interrupt_Disable,      0x00000040) // disable alarm
write(Time_Alarm,            0x00050000) // set time
write(Alarm_Enable,         0x00000004) // enable minute fields
read ( Status,              result )
if ( result !=              0x0000000f ) then
    error
else
    write(Interrupt_Enable,0x00000040) // enable alarm
```

Example 2–7: Setting Alarm to Regular Time

```
write(Interrupt_Disable,      0x00000040) // disable alarm
write(Time_Alarm,            0x115959) // set time
write(Alarm_Enable,         0x0000000f) // enable all time fields
read ( Status,              result )
if ( result !=              0x0000000f ) then
    error
else
    write(Interrupt_Enable,0x00000040) // enable alarm
```

Example 2–8: Setting Events

```
write(Interrupt_Enable,0x00000010)// enable date event
```

2.6.7. Watchdog Timer

The Watchdog Timer (WDT) IP Module provides programmable software activity monitoring functions for use within the PUC 303xA. Fig. 2–21 shows a block diagram of the watchdog timer module.

2.6.7.1. Features

- On timeout, outputs one or a combination of:
 - System Reset
 - System Interrupt
- Variable timeout period. Programmable countdown rate and number of cycles to count.
- Timeout range 32'760 to 268'431'360 clock cycles (512 μs to 4.2 s at 64 MHz)

2.6.7.2. Description

The Watchdog Timer can be used to prevent system lock-up, if e.g. software becomes trapped in a dead-lock. In normal operation the user restarts the watchdog at regular intervals before the timer counts down to zero.

If the timer reaches zero and the watchdog is enabled, one or both of the following signals is generated:

- a system reset,
- an interrupt.

The watchdog time-out period is variable.

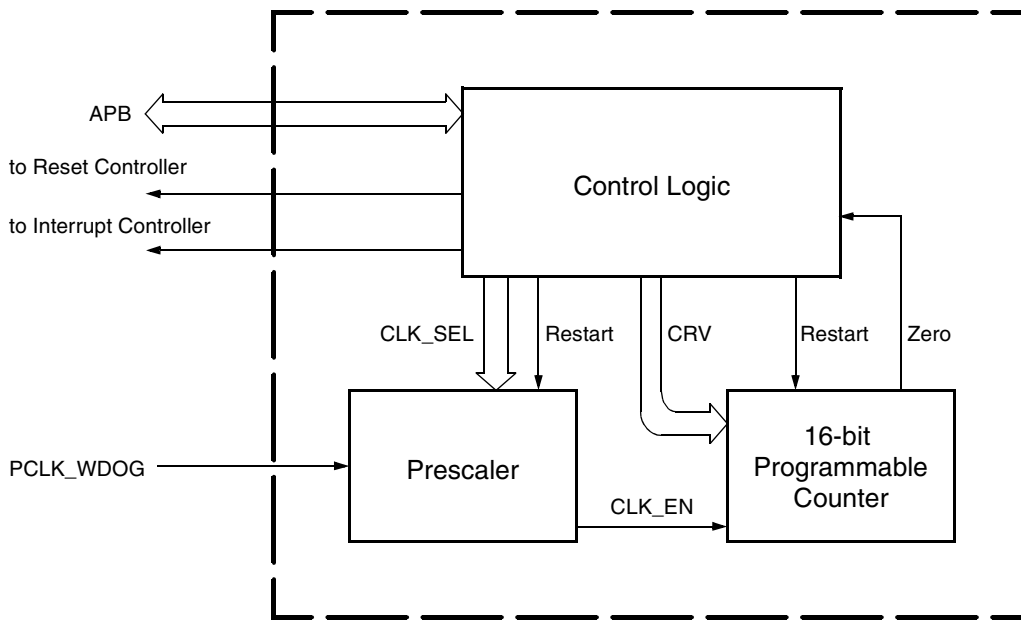


Fig. 2–21: Block diagram: watchdog timer module

2.6.7.2.1. Control Logic Block

The Control Logic Block sends and receives signals across the APB. If the data received from the APB matches the key of the register at the specified address within the block, the Control Logic block can be written to.

Depending upon which register is written to, this can result in the counter being restarted, the speed of the count being altered, or some change in the combination of signals output from the Watchdog Timer when the counter reaches zero.

2.6.7.2.2. Prescaler Block

This block inputs the CLK_SEL signal from the control register. The CLK_SEL signal is sampled on every rising clock edge. It determines how often the periodic CLK_EN output signal occurs according to Table 2–29. If a restart signal is received, the prescaler should reset.

2.6.7.2.3. 16-bit Programmable Counter

The 16-bit Programmable Counter counts down to zero at a rate defined by the COUNT_ENABLE input. The counter reads the restart signal on the positive edge of each clock cycle. If the restart signal is high the counter immediately restarts the countdown at the value read from the COUNTER_RESTART_VALUE signal.

If the counter reaches zero, it stays at zero until it is restarted. While the counter is at zero, the zero output signal goes high. The zero output signal remains high until the counter is restarted.

2.6.7.3. Watchdog Register Descriptions

Table 2–28 summarizes the watchdog registers. Addresses are specified as an offset to the Watchdog base address of 001E.8000_{hex}.

2.6.7.4. Watchdog Timer Reset

In the event of a power-on reset, the Watchdog Timer resets to the following state:

- The watchdog timer is disabled.
- The timer is set to generate a reset on a timeout, but not an interrupt.

Note: The watchdog is reset by the n_watchrst signal from the reset generator module, to ensure that the watchdog is not reset by its own generated reset.

2.6.7.5. Watchdog Timer Enable Sequence

To enable the Watchdog Timer for the first time following a power-on reset, the following registers must be initialized.

1. Initialize the WD Counter Control Register. For example, writing 923C_{hex} to the CONTROL register sets prescale to PCLK_WDOG/8 and the Counter Restart Value to its maximum. The first twelve bits set the CKEY to enable a write to the register.

Enable the timer: for example write AB.C1C5_{hex} to MODE register; the final four bits enable the watchdog and the interrupt signal.

Table 2–28: Watchdog Register Map

Offset	R/W	Width	Name	Comment	Default
00 _{hex}	R/W / W	24	WD_ZERO_MODE	WD Zero Mode register	00.01C2 _{hex}
04 _{hex}	R/W / W	18	COUNTER_CONTROL	Counter Control register	0.003C _{hex}
08 _{hex}	W	16	RESTART	Restart Key Register	N/A
0C _{hex}	R	1	WATCHDOG_STATUS	Watchdog Status Register	0 _{hex}

Table 2–29: Watchdog Read/Write Registers

Register Address	Function	Name																
00 _{hex}	<p>WD_Zero_Mode Register, Offset 0_{hex}, Reset 00.01C2_{hex}</p> <table border="1"> <thead> <tr> <th>Bit</th> <th>31:24</th> <th>23:12</th> <th>11:4</th> <th>3</th> <th>2</th> <th>1</th> <th>0</th> </tr> </thead> <tbody> <tr> <th>Field</th> <td>Reserved</td> <td>ZKEY</td> <td>Reserved</td> <td>Re-served</td> <td>IRQEN</td> <td>RST-EN</td> <td>WDEN</td> </tr> </tbody> </table> <p>The Zero Mode Register controls the behavior of the watchdog timer when the counter reaches zero. In order to write to this register, the correct ZKEY must be passed along with the data to be written.</p> <p>If the Control Logic block receives a zero signal it checks the fields RSTEN and IRQEN to discover which signals are enabled and can therefore be emitted.</p> <p>If the Watchdog is not enabled, the value WDEN is low, and no signals are output.</p> <p>Once emitted, the output signals can not be emitted again until the Programmable Counter is restarted, the zero signal dropped, and a new zero signal received. If a signal is being output, a signal is passed to the Zero Mode Register blocking it from being written to.</p> <p>bit[31:24] Reserved</p> <p>bit[23:12] ZKEY Zero Access Key: Writes to the zero mode register are only valid if this field is set to ABC_{hex}; this field is Write Only.</p> <p>bit[11:4] 1C_{hex} Reserved This reserved field should be written with the Reset value (1C_{hex})</p> <p>bit[3] 0_{bin} Reserved This reserved bit should be written with the Reset value (0_{bin})</p> <p>bit[2] IRQEN Interrupt Request Enable: If set, the watchdog will issue an interrupt request when the counter reaches zero, if WDEN = 1.</p> <p>bit[1] RSTEN Reset Enable: If set, the watchdog will issue an internal reset when the counter reaches zero, if WDEN = 1.</p> <p>bit[0] WDEN Watchdog Enable: If set, the watchdog is enabled and can generate any signals that are enabled.</p>	Bit	31:24	23:12	11:4	3	2	1	0	Field	Reserved	ZKEY	Reserved	Re-served	IRQEN	RST-EN	WDEN	WD_ZERO_MODE
Bit	31:24	23:12	11:4	3	2	1	0											
Field	Reserved	ZKEY	Reserved	Re-served	IRQEN	RST-EN	WDEN											

Table 2–29: Watchdog Read/Write Registers, continued

Register Address	Function	Name										
00 _{hex}	<p>Counter_Control Register</p> <table border="1"> <tr> <td>Bit</td> <td>31:18</td> <td>17:6</td> <td>5:2</td> <td>1:0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>CKEY</td> <td>CRV</td> <td>CLK-SEL</td> </tr> </table> <p>The Counter Control Register controls the time taken for a restarted counter to reach zero. If the APB writes to the Register with the appropriate CKEY, the restart value and clock rate values can be changed. These fields retain their values until they are changed again or a system reset occurs.</p> <p>bit[31:18] Reserved</p> <p>bit[17:6] 248_{hex} CKEY Counter Access Key: Writes to the control register are only valid if this field is set to 248_{hex}; this field is Write Only.</p> <p>bit[5:2] NFFF_{hex} CRV Counter Restart Value: The counter is restarted with the value NFFF_{hex}, where N is the value of this field.</p> <p>bit[1:0] CLKSEL Counter Clock Prescale: selects the prescaler division ratio: 00 = PCLK_WDOG ÷ 8 01 = PCLK_WDOG ÷ 64 10 = PCLK_WDOG ÷ 256 11 = PCLK_WDOG ÷ 4096 Note: If a restart signal is received the prescaler should be reset.</p>	Bit	31:18	17:6	5:2	1:0	Field	Reserved	CKEY	CRV	CLK-SEL	COUNTER_CONTROL
Bit	31:18	17:6	5:2	1:0								
Field	Reserved	CKEY	CRV	CLK-SEL								
00 _{hex}	<p>Restart Register</p> <table border="1"> <tr> <td>Bit</td> <td>31:16</td> <td>15:0</td> </tr> <tr> <td>Field</td> <td>Reserved</td> <td>RSTKEY</td> </tr> </table> <p>This write-only register restarts the counter when it receives the correct key. If the data received does match the key, a restart signal is sent to the 16-bit Programmable Counter.</p> <p>bit[31:16] Reserved</p> <p>bit[15:0] 1999_{hex} RSTKEY Restart Key: the watchdog is restarted if this field is set to the value 1999_{hex}.</p>	Bit	31:16	15:0	Field	Reserved	RSTKEY	RESTART				
Bit	31:16	15:0										
Field	Reserved	RSTKEY										

Table 2–29: Watchdog Read/Write Registers, continued

Register Address	Function	Name						
00 _{hex}	<p>Watchdog_Status Register</p> <table border="1" data-bbox="288 421 1201 497"> <tr> <td data-bbox="288 421 341 456">Bit</td> <td data-bbox="341 421 1145 456">31:1</td> <td data-bbox="1145 421 1201 456">0</td> </tr> <tr> <td data-bbox="288 456 341 497">Field</td> <td data-bbox="341 456 1145 497">Reserved</td> <td data-bbox="1145 456 1201 497">WDZ</td> </tr> </table> <p>This register indicates whether a watchdog invoked reset has occurred. This register maintains its value after the system has finished a watchdog reset sequence. Reading this register clears the WDZ status bit.</p> <p>Alternatively the PMU register PMU_RESET can be read for indication of the reset cause.</p> <p>bit[31:1] Reserved</p> <p>bit[0] WDZ</p> <p> Watchdog Zero: this field is set when the watchdog reaches zero count, that is times out.</p>	Bit	31:1	0	Field	Reserved	WDZ	WATCHDOG_STATUS
Bit	31:1	0						
Field	Reserved	WDZ						

2.6.8. General Purpose I/O Module

The General Purpose I/O (GPIO) GPIO module enables the programming of pin level input and output functions for use within the PUC 303xA.

2.6.8.1. Features

- Up to 66 independently programmable I/O pins.
- Each pin provides input, output and output enable for bi-directional I/O pins.
- Each pin can be programmed as input or output.
- Each pin can be bypassed to or from a separate peripheral device.
- Each pin can separately trigger the GPIO interrupt on several event types.

2.6.8.2. Description

The GPIO module incorporates 66 programmable pins. Each pin can be independently programmed as an input pin, an output pin, or a 'bypass' pin. The general structure of each GPIO pin is detailed in Fig. 2–22.

The pins are grouped into ports of widths between six and eight bits, and labelled as ports A to I.

2.6.8.2.1. GPIO Mode

In the default GPIO mode, the pins of the device are directly accessible through the GPIO registers. There are two possible types of GPIO mode, input mode and output mode.

Input Mode

In Input mode, data is routed through from the I/O pin, and the current value can be read in the INPUT_VALUE register.

To enable this mode for a pin, the appropriate bit in the DIRECTION_MODE register should be set to a 1.

Output Mode

In Output mode, data is routed through to the I/O pin from the OUTPUT_VALUE register.

To enable this mode for a pin, the appropriate bit in the DIRECTION_MODE should be set to a 0. The desired output data should be written to the OUTPUT_VALUE register.

The output enable for the pin can then be controlled as desired, by setting the appropriate bit in the OUTPUT_ENABLE register.

2.6.8.3. Bypass Mode

Bypass mode and the two GPIO modes are mutually exclusive at an individual pin. In bypass mode, the complete control of the pin is handled by the functional block associated with the particular pin.

As the signals are not latched or registered, this allows the pin to behave as in the functional description for that block.

The direction control for the pad is taken from the bypassing module, so that the GPIO direction registers are "don't care".

2.6.8.4. Interrupts

An interrupt trigger is generated for a pin if a pre-defined event type is seen on the pin input. The event can be specified as high or low level, rising or falling edge, or any edge.

Use the registers INT_TYPE, INT_VALUE and INT_ON_ANY to define the event type that triggers an interrupt.

The event is recorded in a read-only register INT_STATUS. This register is cleared whenever the INPUT_STATUS register is read.

Each group of pins, called ports A to I, have one direct interrupt connection to the system interrupt controller. This interrupt line for each port, is the logical OR of all of the interrupt status for pins within the port.

An interrupt is output if an interrupt trigger is seen for any pin, whose INT_MASK register bit is clear.

2.6.8.5. Signal Interface

Each pin has a block structure as shown in Fig. 2–22.

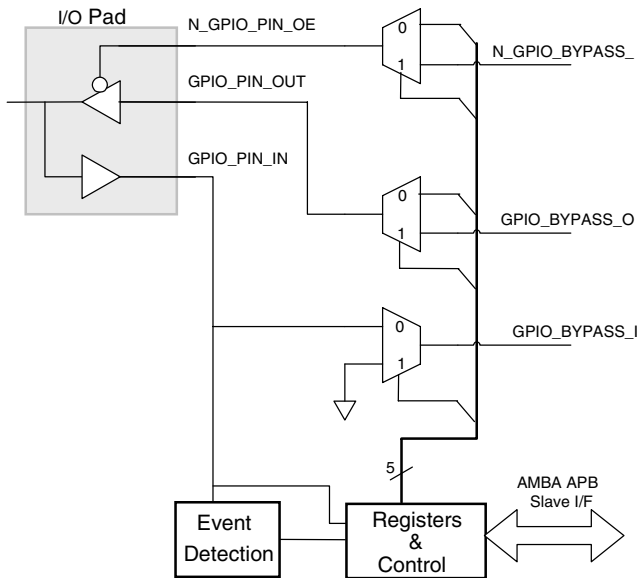


Fig. 2–22: Block structure of pins

The diagram shows the major structure of the block where the data to or from pins passes through multiplexers that are controlled by registers. Each GPIO pin has its own set of control registers. Additional register bits are used for setting the output and output enable values.

An event detector is used to determine input changes and this, in conjunction with a mask register, is used to generate a level triggered interrupt signal.

To put the pin in bypass mode, the BYPASS_MODE register must be set. The pin direction can be changed by setting the DIRECTION_MODE register. Bypass mode enables pins to be shared by the GPIO and another peripheral. The function of the pin can, therefore, be reallocated if the peripheral is not in use.

When in GPIO mode the output value on GPIO_PIN_OUT is driven by the output value register. When the direction register is in input mode N_GPIO_PIN_OE is always inactive (=1), implying that the output buffer is always disabled.

2.6.8.6. Initialization

On power-up, all GPIO pins on the PUC 303xA device are set into GPIO input mode, and all interrupts are masked.

The only exception to this is the USB data pull-up pin, USB_PULLUP, which is initialized into bypass mode.

The USB functional block therefore has direct control of the pin, which makes it a driven output.

Software can re-initialize by writing to the GPIO registers. When changing the mode of a pin, the output should be disabled to prevent contention on the GPIO bidirectional pad, and the interrupt disabled to prevent a spurious event occurring.

Note that during power down mode of the PUC 303xA device, the GPIO pins preserve their state. Only a full power-on reset sequence will reset the state of the GPIO pins.

2.6.8.7. GPIO Address Map

The 66 GPIO pins in the design are split into nine ports (ports H and J are only available on 100-pin packages). Table 2–30 summarizes the offsets for each port from the GPIO base address of 001C.C000_{hex}.

Table 2–30: GPIO Port Offsets

Offset Address	Port	Port Width (N)
000 _{hex}	A	8 bit
040 _{hex}	B	8 bit
080 _{hex}	C	8 bit
0C0 _{hex}	D	7 bit
100 _{hex}	E	7 bit
140 _{hex}	F	7 bit
180 _{hex}	G	5 bit
1C0 _{hex}	H ¹⁾	8 bit
200 _{hex}	J ¹⁾	8 bit
1) Port available on 100-pin packages only		

2.6.8.8. Programming Interface

Each pin within a port has a field within each register, which is of the format shown in Table 2–31/Table 2–32. The value of N is the port width defined in Table 2–30.

Each register address listed in Table 2–32 is defined by an offset that relates to the base address of the GPIO port as defined in Table 2–30.

Table 2–31: Pin Field Format for all GPIO Port Registers in Table 2–32

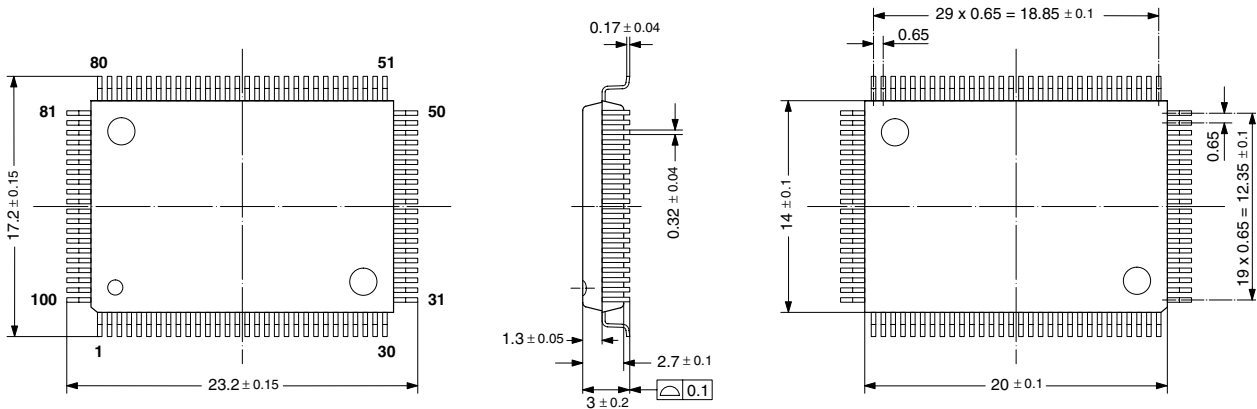
Bit	31:(N+1)	N:0
Field	Reserved (N<31)	Pin N:Pin 0

Table 2–32: GPIO Port Register Map

Offset	Name	Access	Bits	Description	Default
00 _{hex}	BYPASS_MODE	R/W	[N:0]	If bit is 1: set pin to bypass mode If bit is 0: set pin o GPIO mode	00 _{hex}
04 _{hex}	DIRECTION_MODE	R/W	[N:0]	If bit is 1: set pin to input mode If bit is 0: set pin to output mode	FF _{hex}
08 _{hex}	OUTPUT_ENABLE	R/W	[N:0]	If bit is 1: set pin to output enabled If bit is 0: set pin to output disabled (Ignored if the pin is set to input or bypass modes)	00 _{hex}
0C _{hex}	OUTPUT_VALUE	R/W	[N:0]	This register contains the value to be driven out of the pins. The output will only appear at the port if the pin is set to GPIO output mode.	00 _{hex}
10 _{hex}	INPUT_VALUE	R	[N:0]	The input value is read from this register, regardless of the pin mode.	00 _{hex}
14 _{hex}	INT_MASK	R	[N:0]	This register is used to mask interrupt events being signalled by GPIO_INT. If Bit is 1: bit of INT_STATUS is ignored. If Bit is 0: an event on pin will set an interrupt. Bits are set and cleared using the registers INT_ENABLE and INT_DISABLE.	FF _{hex}
18 _{hex}	INT_ENABLE	W	[N:0]	If bit is 1, bit of INT_MASK is cleared.	00 _{hex}
1C _{hex}	INT_DISABLE	W	[N:0]	If bit is 1, bit of INT_MASK is set.	00 _{hex}
20 _{hex}	INT_STATUS	R	[N:0]	If bit is 1 an interrupt-generating event has occurred on bit of INPUT_VALUE. INT_STATUS is set regardless of INT_MASK. This register is cleared by a read.	00 _{hex}
24 _{hex}	INT_TYPE	R/W	[N:0]	If bit is 1, interrupt is level-triggered If bit is 0, interrupt is edge-triggered	00 _{hex}
28 _{hex}	INT_VALUE	R/W	[N:0]	If bit is 1, interrupt is triggered on high level or rising edge, depending on INT_TYPE value If bit is 0, interrupt is triggered on low level or falling edge, depending on INT_TYPE value	00 _{hex}
2C _{hex}	INT_ON_ANY	R/W	[N:0]	If bit is 1 edge triggering occurs on any edge, otherwise edge specified in INT_VALUE triggers an interrupt. INT_ON_ANY is ignored if INT_TYPE =1.	00 _{hex}

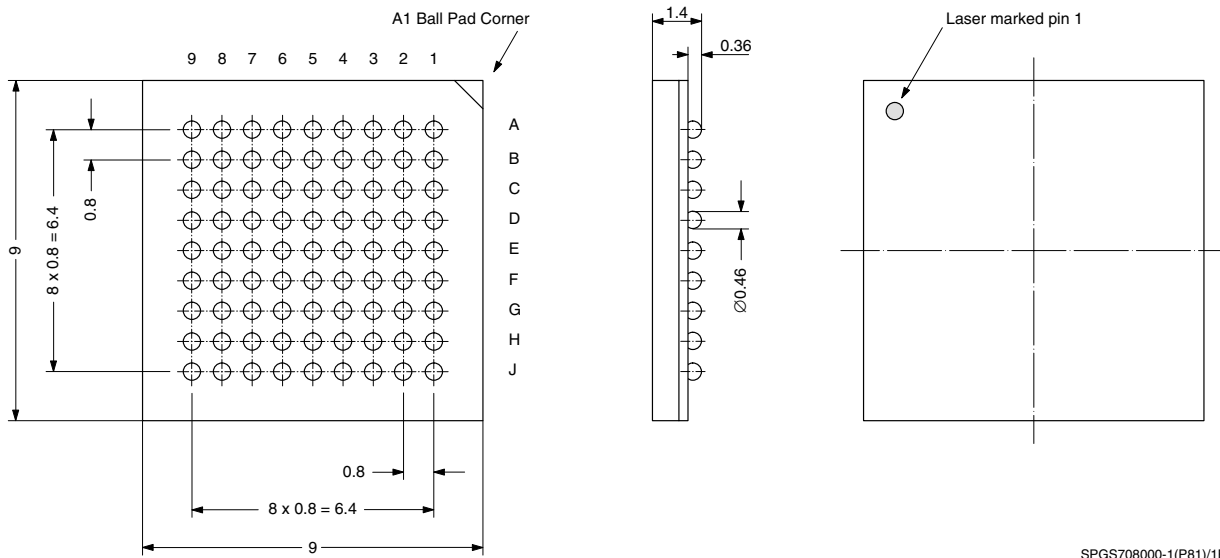
3. Specifications

3.1. Outline Dimensions



SPGS705000-3(P100)/1E

Fig. 3-1:
 100-Pin Plastic Quad Flat Pack
(PQFP100)
 Weight approximately 1.61 g
 Dimensions in mm



SPGS708000-1(P81)/1E

Fig. 3-2:
 81-Pin Low-Profile Fine-Pitch Ball Grid Array
(LFBGA81)
 Weight approximately 0.19 g
 Dimensions in mm

3.2. Pin Connections and Short Descriptions

NA = not available on package
 NT = not tested
 S = Schmitt-trigger input
 U = internal pull-Up
 D = internal pull-Down

Pin Circuits

IU = Input pin with pull-up
 ID = Input pin with pull-down
 IS = Input pin with Schmitt trigger
 IUS = Input pin with Schmitt trigger and pull-up
 OSC = Crystal oscillator pin
 O4 = Output pin, 4 mA
 IO4 / IO12 = Input / output pin, 4 mA / 12 mA
 IOS4 = Input with Schmitt trigger / output pin, 4 mA
 IOD4 = Input/output pin with pull-down, 4 mA

Pin No.		Pin Name	GPIO Port	Type	Pin Circuit <small>(see Section 3.4.)</small>	Short Description
PQFP 100-pin	LFBGA 81-pin					
1	B8	I2S_RXD	PE.0	IN/OUT	IO4	I2S Receive Data
2	C6	I2S_CLK	PE.1	IN/OUT	IOS4	I2S Serial Clock
3	A7	I2S_FRM	PE.2	IN/OUT	IO4	I2S Framing
4	B7	I2S_TXD	PE.3	IN/OUT	IO4	I2S Transmit Data
5	B6	SCLK4	PE.4	IN/OUT	IOS4	SSP4 Serial Clock
6	A6	SFRM4	PE.5	IN/OUT	IO4	SSP4 Framing
7	C5	SRXD4	PE.6	IN/OUT	IO4	SSP4 Receive Data
8	A9	VDD_CORE		SUPPLY	–	Core VDD supply
9	A8	VSS_CORE		SUPPLY	–	Core VSS supply
10	F7	TXD2	PA.0	IN/OUT	IO4	UART2 Transmit Data
11	F6	RXD2	PA.1	IN/OUT	IO4	UART2 Receive Data
12	E6	nRTS2	PA.2	IN/OUT	IO4	UART2 Request to Send, active low
13	D6	nCTS2	PA.3	IN/OUT	IO4	UART2 Clear To Send, active low
14	F5	TXD1	PA.4	IN/OUT	IO4	UART1 Transmit Data
15	E5	RXD1	PA.5	IN/OUT	IO4	UART1 Receive Data
16	D5	nRTS1	PA.6	IN/OUT	IO4	UART1 Request To Send, active low
17	F4	nCTS1	PA.7	IN/OUT	IO4	UART1 Clear To Send, active low
18	E4	VSS_IO		SUPPLY	–	I/O VSS supply
19	D4	VDD_IO		SUPPLY	–	I/O VDD supply
20	B5	nDSR2	PB.0	IN/OUT	IO4	UART2 Data Set Ready, active low
21	A5	nDTR2	PB.1	IN/OUT	IO4	UART2 Data Terminal Ready, active low

Pin No.		Pin Name	GPIO Port	Type	Pin Circuit (see Section 3.4.)	Short Description
PQFP 100-pin	LFPGA 81-pin					
22	C4	nDCD2	PB.2	IN/OUT	IO4	UART2 Data Carrier Detect, active low
23	A4	nRI2	PB.3	IN/OUT	IO4	UART2 Ring Indicator, active low
24	B4	nDSR1	PB.4	IN/OUT	IO4	UART1 Data Set Ready, active low
25	A3	nDTR1	PB.5	IN/OUT	IO4	UART1 Data Terminal Ready, active low
26	B3	nDCD1	PB.6	IN/OUT	IO4	UART1 Data Carrier Detect, active low
27	A2	nRI1	PB.7	IN/OUT	IO4	UART1 Ring Indicator, active low
28	A1	PORTG_0	PG.0	IN/OUT	IO4	Dedicated GPIO
29	B2	PORTG_1	PG.1	IN/OUT	IO4	Dedicated GPIO
30	B1	PORTG_2	PG.2	IN/OUT	IO4	Dedicated GPIO
31	C2	PORTG_3	PG.3	IN/OUT	IO4	Dedicated GPIO
32	C1	PORTG_4	PG.4	IN/OUT	IO4	Dedicated GPIO
33	D2	nRESET		IN	IS	System Reset, active low
34	NA	WAKE_UP		IN	IUS	Wake-Up from STANDBY
35	D1	MODSEL_1		IN	ID	Mode Select pin 1
36	C3	MODSEL_0		IN	ID	Mode Select pin 0
37	D3	nTRST		IN	IUS	JTAG Reset, active low
38	E3	TCK		IN	IS	JTAG Clock
39	F3	TDI		IN	IU	JTAG Test Data Input
40	G3	TDO		OUT	O4	JTAG Test Data Output
41	F2	TMS		IN	I4	JTAG Test Mode Select
42	E1	PORTF_0	PF.0	IN/OUT	IO4	Dedicated GPIO
43	E2	PORTF_1	PF.1	IN/OUT	IO4	Dedicated GPIO
44	F1	I2S_AUDIOCLK	PF.2	IN/OUT	IOS4	I ² S (SSP5) Audio Clock Input
45	G2	UC_SDA	PF.3	IN/OUT	IOS4	I ² C Serial Data, handled by USB Core
46	H2	UC_SCL	PF.4	IN/OUT	IOS4	I ² C Serial Clock, handled by USB Core
47	G1	VSS_IO		SUPPLY	–	I/O VSS supply
48	H1	VDD_IO		SUPPLY	–	I/O VDD supply

Pin No.		Pin Name	GPIO Port	Type	Pin Circuit (see Section 3.4.)	Short Description
PQFP 100-pin	LFBGA 81-pin					
49	NA	PORTH_0	PH.0	IN/OUT	IOD4	Dedicated GPIO
50	NA	PORTH_1	PH.1	IN/OUT	IOD4	Dedicated GPIO
51	NA	PORTH_2	PH.2	IN/OUT	IOD4	Dedicated GPIO
52	NA	PORTH_3	PH.3	IN/OUT	IOD4	Dedicated GPIO
53	NA	PORTH_4	PH.4	IN/OUT	IOD4	Dedicated GPIO
54	NA	PORTH_5	PH.5	IN/OUT	IOD4	Dedicated GPIO
55	NA	PORTH_6	PH.6	IN/OUT	IOD4	Dedicated GPIO
56	NA	PORTH_7	PH.7	IN/OUT	IOD4	Dedicated GPIO
57	J1	VSS_CORE		SUPPLY	–	Core VSS supply
58	J2	VDD_CORE		SUPPLY	–	Core VDD supply
59	H3	USB_SENSE	PF.5	IN/OUT	IO4	USB interface Sense
60	J3	USB_PULLUP	PF.6	IN/OUT	IO4	USB interface Pullup
61	H4	VSS_USB		SUPPLY	–	USB VSS supply
62/NT	G5/NT	USB_DPLUS ¹⁾		IN/OUT	–	USB interface D ⁺
63/NT	G4/NT	USB_DMINUS ¹⁾		IN/OUT	–	USB interface D ⁻
64	J4	VDD_USB		SUPPLY	–	USB VDD supply
65	H5	SCLK1	PC.0	IN/OUT	IOS4	SSP1 Serial Clock
66	G6	STXD1	PC.1	IN/OUT	IO4	SSP1 Transmit Data
67	J6	SFRM1	PC.2	IN/OUT	IO4	SSP1 Framing
68	H6	SRXD1	PC.3	IN/OUT	IO4	SSP1 Receive Data
69	J7	SCLK2	PC.4	IN/OUT	IOS4	SSP2 Serial Clock
70	H7	STXD2	PC.5	IN/OUT	IO4	SSP2 Transmit Data
71	J8	SFRM2	PC.6	IN/OUT	IO4	SSP2 Framing
72	J9	SRXD2	PC.7	IN/OUT	IO12	SSP2 Receive Data
73	J5	CLKOUT		OUT	O4	Clock Output
74	NA	PORTI_0	PI.0	IN/OUT	IOD4	Dedicated GPIO
75	NA	PORTI_1	PI.1	IN/OUT	IOD4	Dedicated GPIO
76	NA	PORTI_2	PI.2	IN/OUT	IOD4	Dedicated GPIO
77	NA	PORTI_3	PI.3	IN/OUT	IOD4	Dedicated GPIO
78	NA	PORTI_4	PI.4	IN/OUT	IOD4	Dedicated GPIO
79	NA	PORTI_5	PI.5	IN/OUT	IOD4	Dedicated GPIO

Pin No.		Pin Name	GPIO Port	Type	Pin Circuit (see Section 3.4.)	Short Description
PQFP 100-pin	LFBGA 81-pin					
80	NA	PORTI_6	PI.6	IN/OUT	IOD4	Dedicated GPIO
81	NA	PORTI_7		IN/OUT	IOD4	Dedicated GPIO
82	H9	VSS_IO		SUPPLY	–	I/O VSS supply
83	H8	VDD_IO		SUPPLY	–	I/O VDD supply
84	G9	VDD_CORE		SUPPLY	–	Core VDD supply
85	G8	VSS_CORE		SUPPLY	–	Core VSS supply
86	G7	SRXD3	PD.0	IN/OUT	IO12	SSP3 Receive Data
87	E7	SFRM3	PD.1	IN/OUT	IO4	SSP3 Framing
88	D7	SCLK3	PD.2	IN/OUT	IOS4	SSP3 Serial Clock
89	C7	STXD3	PD.3	IN/OUT	IO4	SSP3 Transmit Data
90	F9	TIMER_1	PD.4	IN/OUT	IOS4	Timer 1 I/O function pin
91	F8	TIMER_2	PD.5	IN/OUT	IOS4	Timer 2 I/O function pin
92	E8	STXD4	PD.6	IN/OUT	IOS4	SSP4 Transmit Data
93	NA	OSC_RTC_OUT		OUT	OSC	RTC Oscillator Output
94	NA	OSC_RTC_IN		IN	OSC	RTC Oscillator Input
95	D9	OSC_SYS_OUT		OUT	OSC	System Oscillator Output
96	E9	OSC_SYS_IN		IN	OSC	System Oscillator Input
97	B9	AVDD_PLL		SUPPLY	–	PLL Analog VDD supply
98	C9	AVSS_PLL		SUPPLY	–	PLL Analog VSS supply
99	C8	DVDD_PLL		SUPPLY	–	PLL Digital VDD supply
100	D8	DVSS_PLL		SUPPLY	–	PLL Digital VSS supply

¹⁾ USB interface not tested on PUC 3033A devices

3.3. Pin Descriptions

Note: All power supply pins must be connected. This is mandatory for proper functioning of the PUC 303xA.

3.3.1. Special Function Pins

Power Supply Pins

VDD_CORE, VSS_CORE – 2.5-V core supply. The three VDD/VSS pairs are used to power the internal core logic.

DVDD_PLL, DVSS_PLL – 2.5-V PLL supply. The DVDD/DVSS pair is used to power the digital section of the PLL.

AVDD_PLL, AVSS_PLL – 2.5-V PLL supply. The AVDD/AVSS pair is used to power the analog section of the PLL.

I/O Supply

VDD_IO, VSS_IO – 3.3-V supply. The three VDD/VSS pairs are used to power the I/O pads.

VDD_USB, VSS_USB – 3.3-V USB supply. This VDD/VSS pair is used to power the on-chip USB transceiver cell.

System Pins

nRESET – system, active low reset input pin. Assert '0' at power-on, warmstart or to return from OFF state.

MODSEL_0, MODSEL_1 – PUC 303xA mode select pin 0 resp. 1. Tie both to '0' for normal operation mode; '1' is reserved for test modes.

OSC_SYS_IN – System clock oscillator input. Terminal to connect crystal or external clock source.

OSC_SYS_OUT – System clock oscillator output. Terminal to connect crystal.

OSC_RTC_IN – Real Time Clock oscillator input. Terminal to connect crystal or external clock source. Available in 100-pin package only.

OSC_RTC_OUT – Real Time Clock oscillator output. Terminal to connect crystal. Available in 100-pin package only

WAKE_UP – Input pin for exiting STANDBY mode. Available in 100-pin package only

CLKOUT – Clock output. Clock for external device, e.g. Micronas MAS 35xxF.

JTAG Interface

nTRST – Active low JTAG reset input

TCK – JTAG clock input

TDI – JTAG data input

TMS – JTAG test mode select input

TDO – JTAG data output

USB Transceiver

USB_DPLUS – Bidirectional USB transceiver D+

USB_DMINUS – Bidirectional USB transceiver D-

Note: To ensure minimum power consumption on PUC 3033A devices, connect USB_DPLUS to VDD_USB and USB_DMINUS to VSS_USB.

3.3.2. Shared GPIO / Special Function Pins**USB Interface**

USB_SENSE – USB interface sense input

USB_PULLUP – USB interface pull-up output. This GPIO pin is configured as output on reset

UART Interface Pins¹⁾

RXDy – UART serial data in

nCTS_y – Clear to send input, active low

nDSR_y – Data set ready input, active low

nDCD_y – Data carrier detect input, active low

nRTS_y – Request to send output, active low

nDTR_y – Data terminal ready output, active low

nRI_y – Ring indicator input, active low

TXD_y – UART serial data output

¹⁾ substitute "y" by "1" or "2"

Synchronous Serial Interfaces

SRXD_y¹⁾ – Serial data input.

STXD_y¹⁾ – Serial data output

SCLK_y¹⁾ – Serial data clock, input/output

SFRM_y¹⁾ – Serial data framing, input/output

I2S_AUDIOCLK – I²S interface: Clock input. Enables fast transmission synchronous to external audio device, e.g. MAS 35xxF

I2S_RXD – I²S interface: Serial data input

I2S_TXD – I²S interface: Serial data output

I2S_FRM – I²S interface: Serial data framing, input/output

I2S_CLK – I²S interface: Serial data clock, input/output

¹⁾ subst. "y" by "1", "2", "3", or "4"

Timer Interfaces

TIMER_1, TIMER_2 – Timer I/O pins (timer 1 resp. 2)
Pins for sampling or generating waveforms

I²C Interface

UC_SDA – I²C data

UC_SCL – I²C clock⁴⁾

3.3.3. General Purpose I/O Pins Without Special Function

PORTF_x¹⁾ – Dedicated GPIO (no special function)

PORTG_y²⁾ – Dedicated GPIO (no special function)

PORTH_z³⁾ – Dedicated GPIO (no special function)

PORTI_z³⁾ – Dedicated GPIO (no special function)

¹⁾ substitute "x" by "0", "1"

²⁾ subst. "y" by "0", "1", "2", "3", or "4"

³⁾ substitute "z" by "0", "1", "2", "3", "4", "5", "6", or "7"

⁴⁾ I²C interface managed by USB core's embedded controller.

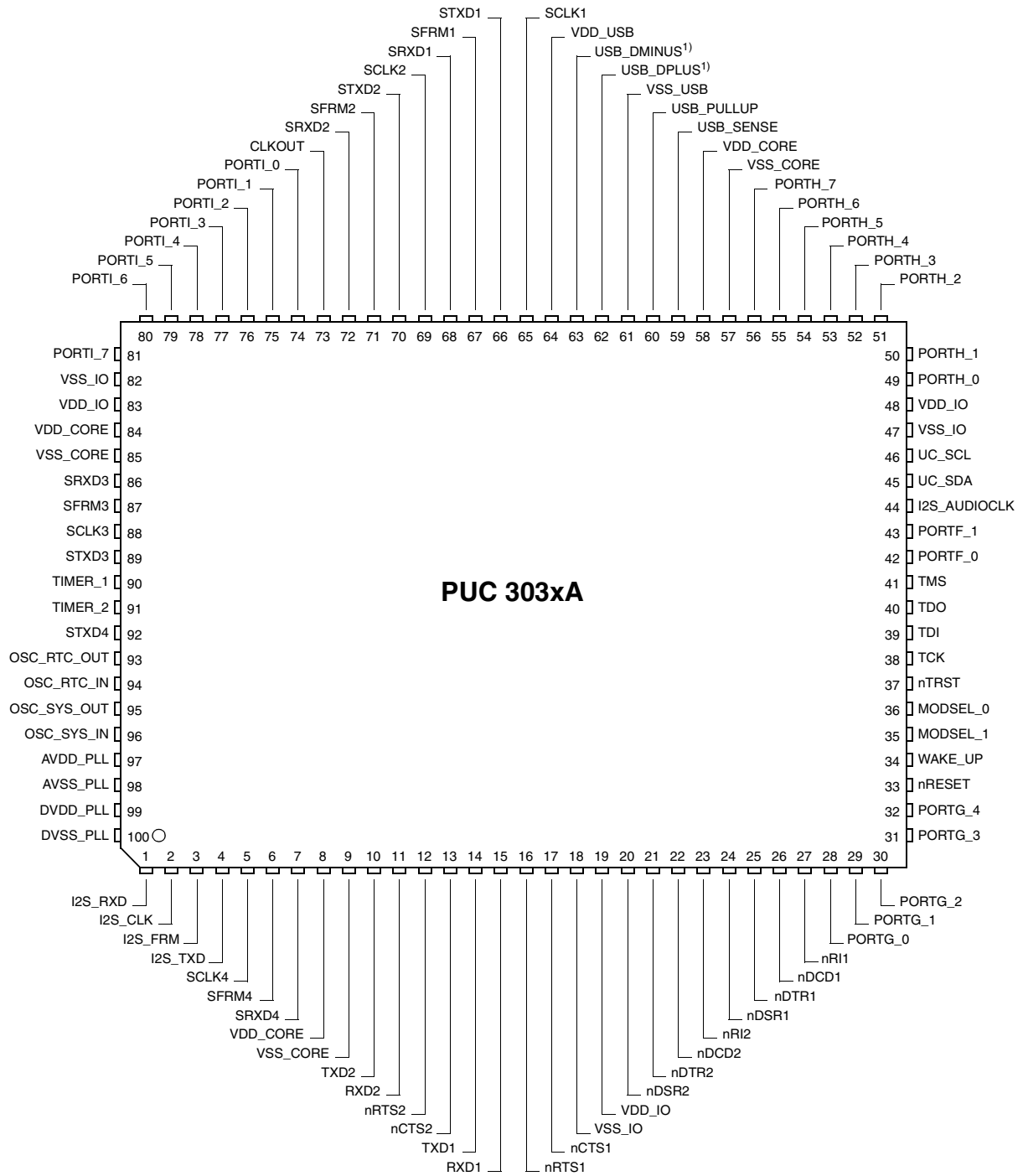


Fig. 3-3: PQFP100 package

1) USB interface not tested on PUC 3033A devices

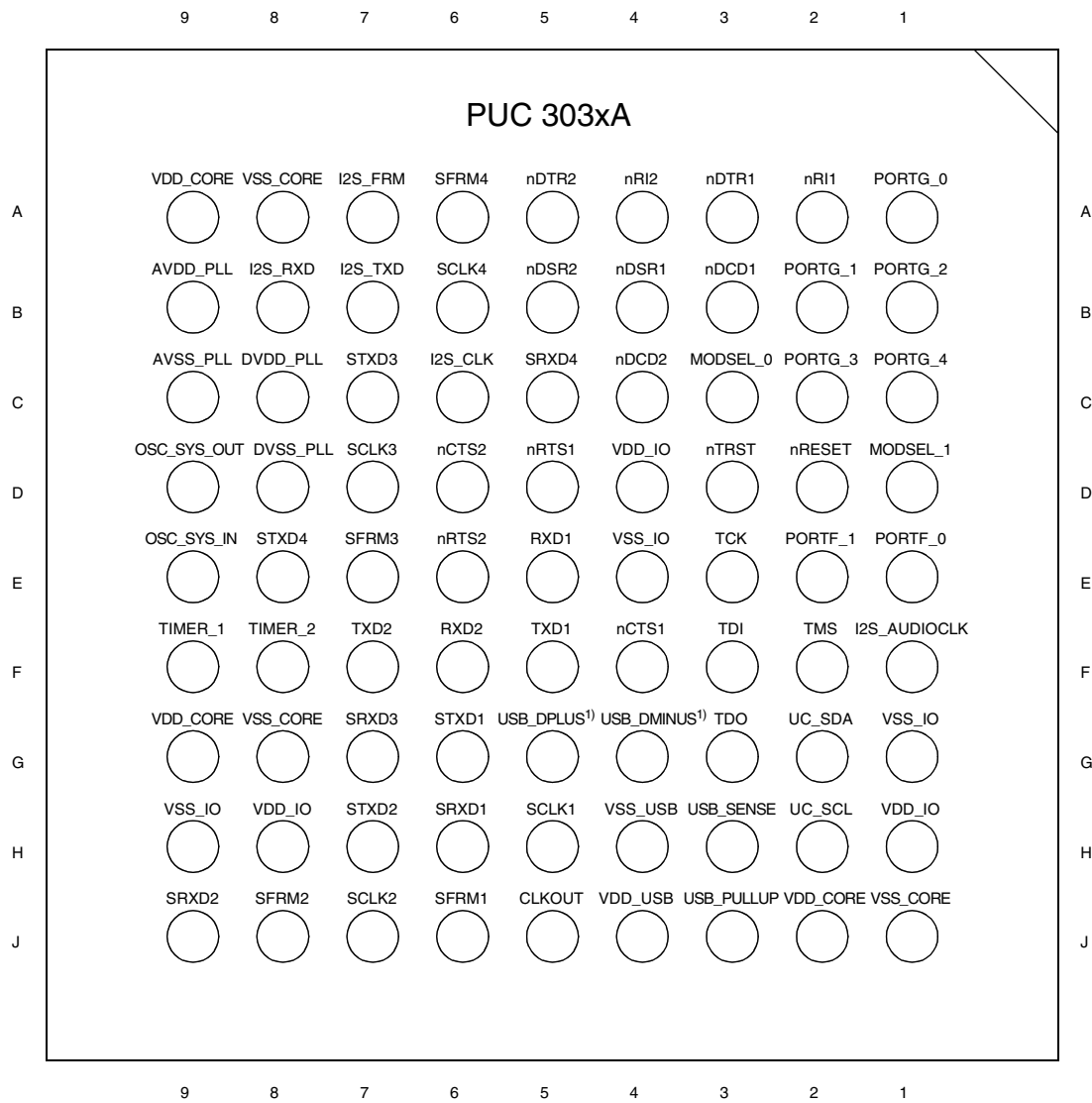


Fig. 3-4: LFBGA81 package

¹⁾ USB interface not tested on PUC 3033A devices

3.4. Pin Circuits

All buffers are supplied by VDD_IO.

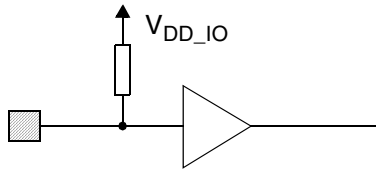


Fig. 3-1: Pin circuit IU: Input pin with pull-up

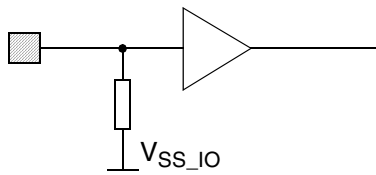


Fig. 3-2: Pin circuit ID: Input pin with pull-down

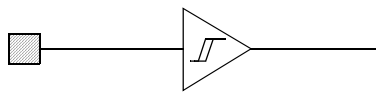


Fig. 3-3: Pin circuit IS: Input pin with Schmitt trigger

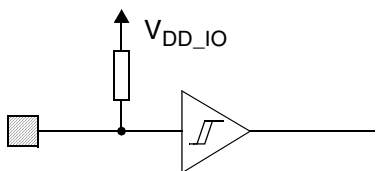


Fig. 3-4: Pin circuit IUS: Input pin with Schmitt trigger and pull-up

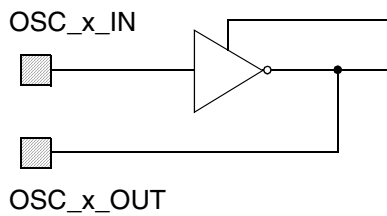


Fig. 3-5: Pin circuit OSC: Crystal oscillator pins

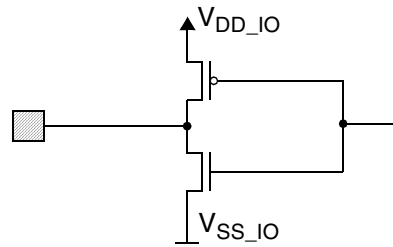


Fig. 3-6: Pin circuit O4: Output pin

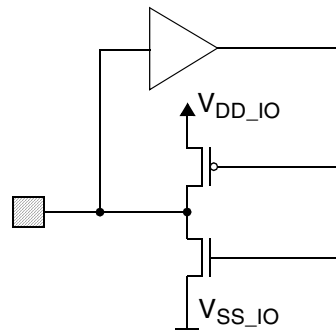


Fig. 3-7: Pin circuit IO4 / IO12: Input / output pin

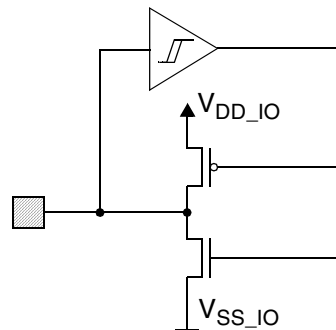


Fig. 3-8: Pin circuit IOS4: Input with Schmitt trigger / output pin

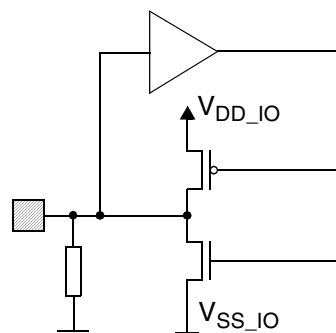


Fig. 3-9: Pin circuit IOD4: Input/output pin with pull-down

3.5. Electrical Characteristics

3.5.1. Absolute Maximum Ratings

Symbol	Parameter	Min.	Max.	Unit
V _{DD_CORE}	Core Supply Voltage	–	2.9	V
DV _{DD_PLL}	Digital PLL Supply Voltage	–	2.9	V
AV _{DD_PLL}	Analog PLL Supply Voltage	–	2.9	V
V _{DD_IO}	I/O Supply Voltage	–	4.6	V
V _{DD_USB}	USB Transceiver Supply Voltage	–	3.6	V
V _{IN}	Input Voltage on I/O pins	–0.5	6.0	V
I _{SS4}	Source / Sink Current (4 mA drive strength)	–30	15	mA
I _{SS12}	Source / Sink Current (12 mA drive strength)	–65	35	mA
V _{IN,USB}	Input Voltage on USB D+ / D–	0	3.6	V
P _{max}	Power Dissipation	–	600	mW
T _S	Storage Temperature	–40	125	°C
T _{Op}	Ambient Operating Temperature	–40	85	°C

Stresses beyond those listed in the “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only. Functional operation of the device at these or any other conditions beyond those indicated in the “Recommended Operating Conditions/Characteristics” of this specification is not implied. Exposure to absolute maximum ratings conditions for extended periods may affect device reliability.

3.5.2. Recommended Operating Conditions

Symbol	Parameter	Min.	Typ.	Max.	Unit	Conditions
T _A	Ambient Operating Temperature ¹⁾	0		85	°C	
T _{AE}	Extended Ambient Operating Temperature ¹⁾	-40		85	°C	
V _{DD_CORE}	Core Supply Voltage	2.25	2.5	2.75	V	
DV _{DD_PLL}	Digital PLL Supply Voltage	2.25	2.5	2.75	V	
AV _{DD_PLL}	Analog PLL Supply Voltage	2.25	2.5	2.75	V	
V _{DD_IO}	I/O Supply Voltage	3.0	3.3	3.6	V	
V _{DD_USB}	USB Transceiver Supply Voltage	3.0	3.3	3.6	V	
V _{IL}	Input Low Voltage	-0.3		1.0	V	
V _{IH}	Input High Voltage	1.8		5.5	V	
f _{OSC_SYS}	System Oscillator Clock Frequency a) Quartz b) External clock source	10 6	14 14	20 28	MHz	45/55% Duty cycle
f _{OSC_RTC}	RTC Oscillator Clock Frequency	32.768		8388.6	KHz	
¹⁾ The functionality of the IC in the extended temperature range has been verified by electrical characterization based on sample tests. All data sheet parameters are valid for normal operating temperature range, only.						

3.5.3. Power Consumption

PUC Operation	Current Consumption (PUC device incl. quartz crystal)	Min.	Typ.	Max.	Unit	Conditions ¹⁾
MP3 decoding	Core	–	39	–	mA	PLL on (VCO = 96 MHz), HCLK = 48 MHz; V _{DD_core} = 2.5 V, V _{DD_IO} = 3.3 V; PUC_Word mode (I ² S-like) output to DAC; MMC card as data source.
	I/O	–	2.8	–	mA	
7.16 MHz (PLL off)	Core	–	10.0 (SRAM) 13.8 (Flash)	–	mA	14.318 MHz OSC_SYS crystal, HCLKDIV = 2; Access to SRAM (Flash) each HCLK cycle (execution of single cycle instructions = max. access rate on memory, no idle periods); V _{DD_core} = 2.5 V, V _{DD_IO} = 3.3 V. ²⁾
	I/O	–	2.6 @ 14.31 MHz quartz	–	mA	
IDLE (PLL off)	Core	–	0.9	–	mA	PLL off, 14.318 MHz OSC_SYS crystal, HCLKDIV = 63, Timer 1 operational. ²⁾
	I/O	–	2.6 @ 14.31 MHz quartz	–	mA	
64 MHz (PLL on)	Core	–	88	–	mA	Access to SRAM each HCLK cycle (execution of single cycle instructions from SRAM = max access rate on memory, no idle periods). ²⁾
	I/O	–	2.6 @ 14.31 MHz quartz 1.9 @ 12.0 MHz quartz	–	mA	
48 MHz (PLL on)	Core	–	68	–	mA	Access to SRAM each HCLK cycle (execution of single cycle instructions from SRAM = max access rate on memory, no idle periods). ²⁾
	I/O	–	2.6 @ 14.31 MHz quartz 1.9 @ 12.0 MHz quartz	–	mA	
IDLE (PLL on)	Core	–	7.2	–	mA	VCO = 96 MHz, HCLKDIV = 63, Timer 1 operational; V _{DD_core} = 2.5 V, V _{DD_IO} = 3.3 V. ²⁾
	I/O	–	2.6 @ 14.31 MHz quartz 1.9 @ 12.0 MHz quartz	–	mA	

PUC Operation	Current Consumption (PUC device incl. quartz crystal)	Min.	Typ.	Max.	Unit	Conditions ¹⁾
STANDBY	Core	–	66	–	μA	RTCCLK on, OSC_RTC 32-kHz quartz crystal, all I/O set so that minimal parasitic pull-up/pull-down currents; V _{DD_core} = 2.5 V, V _{DD_IO} = 3.3 V.
	I/O	–	0.6	–	mA	
OFF	Core	–	64	–	μA	All I/O set so that minimal parasitic pull-up/pull-down currents; V _{DD_core} = 2.5 V, V _{DD_IO} = 3.3 V.
	I/O	–	<0.2	–	μA	
¹⁾ All measurements at T= 25 °C OSC_SYS frequency provided by external quartz crystal ²⁾ HCLK on, CLKOUT off, UART2CLK on, PCLK_APB on, USB12CLK/USB48CLK off, SSPxCLK off, UART1CLK off, RTCCLK off, OSC_RTC off, PCLK_WDOG off, PCLK_RTC off						

3.5.4. DC Characteristics

3.5.4.1. Digital Outputs

Symbol	Parameter	Min.	Typ.	Max.	Unit	Conditions
$V_{OL,4}$	Output-Low Voltage (4 mA drive strength)			0.4	V	@ $I_{OL}=6$ mA
$V_{OH,4}$	Output High Voltage (4 mA drive strength)	2.4			V	@ $I_{OH}=-8$ mA
$I_{OL,4}$	Low-Level Output Current (4 mA drive strength)	6	10	13	mA	@ $V_{OL}=0.4$ V
$I_{OH,4}$	High-Level Output Current (4 mA drive strength)	-8	-18	-28	mA	@ $V_{OH}=2.4$ V
$V_{OL,12}$	Output-Low Voltage (12 mA drive strength)			0.4	V	@ $I_{OL}=16$ mA
$V_{OH,12}$	Output-High Voltage (12 mA drive strength)	2.4			V	@ $I_{OH}=-16$ mA
$I_{OL,12}$	Low-Level Output Current (12 mA drive strength)	16	25	30	mA	@ $V_{OL}=0.4$ V
$I_{OH,12}$	High-Level Output Current (12 mA drive strength)	-16	-35	-57	mA	@ $V_{OH}=2.4$ V

3.5.4.2. Hysteresis of Schmitt Trigger Inputs

Symbol	Parameter	Min.	Typ.	Max.	Unit	Conditions
V_T^H	Hysteresis	0.4	-	1.2	V	$T=25$ °C $V_{DD_IO}=3.3$ V $V_{DD_CORE}=2.5$ V
V_T^+	Low-to-High threshold voltage	1.75	1.95	2.25	V	
V_T^-	High-to-Low threshold voltage	1.05	1.25	1.35	V	

3.5.4.3. Internal Pull-up / Pull-down Resistor Values

Symbol	Parameter	Min.	Typ.	Max.	Unit	Conditions
R_U	Internal pull-up resistor	70	100	175	k Ω	$V_{DD_IO}=3.3$ V $I=10$ μ A
R_D	Internal pull-down resistor	40	65	130	k Ω	

3.5.5. AC Characteristics

3.5.5.1. SSP Slave (SCLKx and SFRMx are Inputs)

Symbol	Parameter	Min.	Typ.	Max.	Unit	Conditions
t_{ssfd}/t_{spst}	SRXDx, SFRMx setup time relative to sampling edge of SCLKx	2	–	–	ns	
t_{hsfd}/t_{hpsr}	SRXDx, SFRMx hold time relative to sampling edge of SCLKx	6	–	–	ns	
t_{dsfd}	STXDx propagation delay relative to received edge of SCLKx	–	–	117	ns	F_SSP(x)CLK = 48.0 MHz; F_SCLKx = F_SSP(x)CLK/12 = 4.0 MHz

3.5.5.2. SSP Master (SCLKx and SFRMx are Outputs)

Symbol	Parameter	Min.	Typ.	Max.	Unit	Conditions
t_{OEm}	STXDx enable time relative to SFRMx	–	–	10	ns	
t_{ODm}	STXDx disable time relative to SFRMx	–	–	5	ns	
t_{sm}	SRXDx setup time relative to sampling edge of SCLKx	2	–	–	ns	
t_{hm}	SRXDx hold time relative to sampling edge of SCLKx	16	–	–	ns	
t_{dm}	STXDx propagation delay relative to generating edge of SCLKx	–	–	12.5	ns	
t_{Hi}	Minimum SCLKx high time	18	–	–	ns	f_SCLKx = f_ssp(x)clk/2 = 24 MHz
t_{Lo}	Minimum SCLKx low time	22	–	–	ns	

3.5.5.3. Timing Diagrams

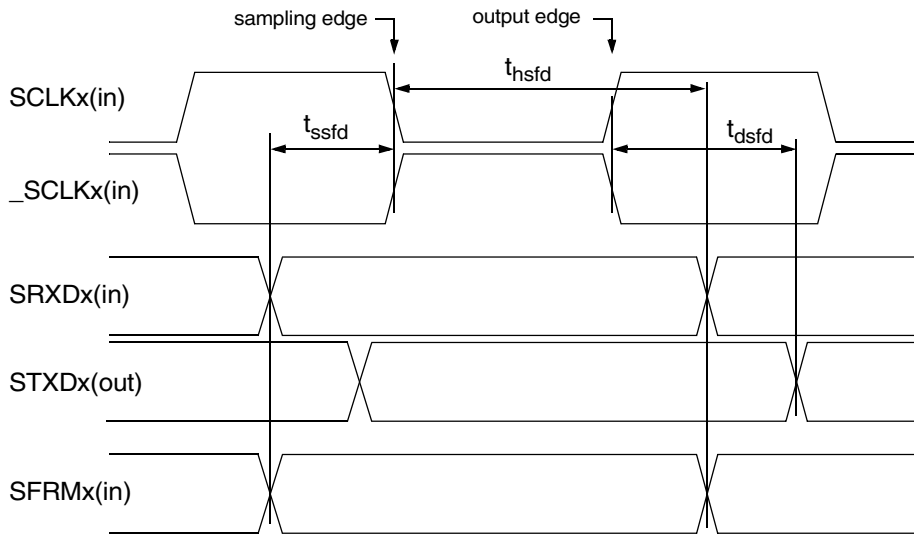


Fig. 3-10: SSP slave (full duplex)

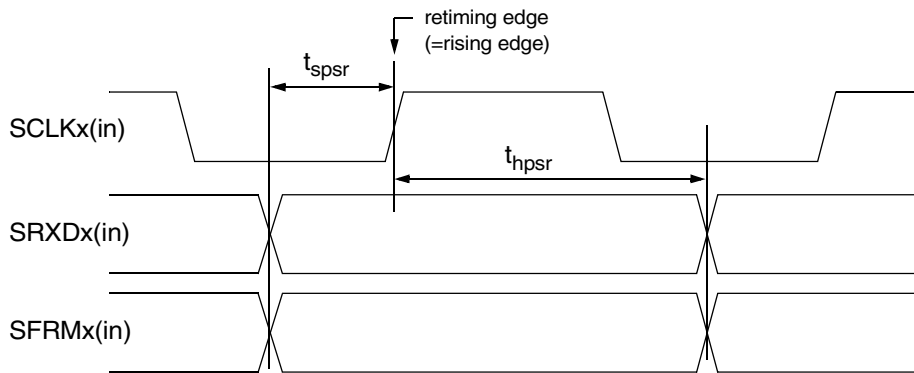


Fig. 3-11: SSP pure slave-receiver (PUC_word or PUC-continuous mode)

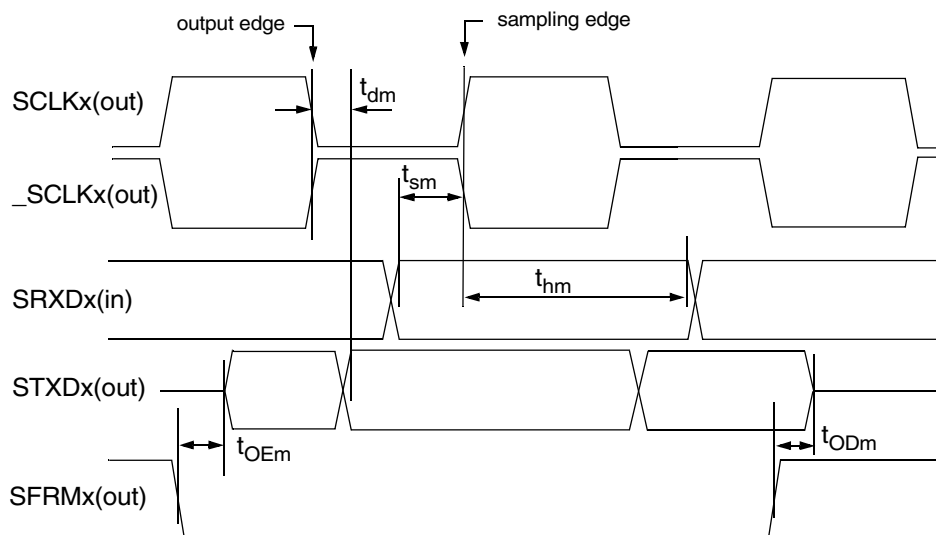


Fig. 3-12: SSP master (full duplex or pure transmitter)

3.5.5.4. AC Characteristics for CLKOUT Pin

Symbol	Parameter	Min.	Typ.	Max.	Unit	Conditions
t _{Hi}	Minimum CLKOUT high time	37.5	–	–	ns	F_CLKOUT= F_OSCCLK=13 MHz
t _{Lo}	Minimum CLKOUT low time	39	–	–	ns	
t _{del}	Delay relative to system clock input (OSCCLK)	8.5	–	14.5	ns	CLKOUT fed from OSCCLK not PLLCLK

3.5.5.5. Timing Requirement for WAKE_UP Pin

Symbol	Parameter	Min.	Typ.	Max.	Unit	Conditions
t _{Hi}	Minimum WAKE_UP high time for rising edge detection	1	–	–	OSCRTCCLK cycles	

3.5.5.6. Timing Requirement for TIMER_x Inputs

Symbol	Parameter	Min.	Typ.	Max.	Unit	Conditions
t _{Imp}	Minimum impulse length for detection	1	–	–	PCLK_APB cycles	

3.5.5.7. Timing Requirement for GPIO Inputs

Symbol	Parameter	Min.	Typ.	Max.	Unit	Conditions
t _{Hi}	Minimum impulse high time for level or edge detection	1	–	–	PCLK_APB cycles	
t _{Lo}	Minimum impulse low time for level or edge detection	1	–	–	PCLK_APB cycles	

3.5.5.8. Rise/Fall Times of 4-mA and 12-mA Outputs

Symbol	Parameter	Min.	Typ.	Max.	Unit	Conditions
t _{rise4}	Rise time of 4-mA output	–	4	–	ns	C _L = 30 pF T = 25 °C V _{DD_IO} = 3.3 V V _{DD_CORE} = 2.5 V
t _{fall4}	Fall time of 4-mA output	–	3	–	ns	
t _{rise12}	Rise time of 12-mA output	–	3.5	–	ns	
t _{fall12}	Fall time of 12-mA output	–	3.5	–	ns	

Table 3–1: Flash Programming Parameters

Symbol	Parameter	Min	Max	Units
Tnvs	PROG/ERASE to NVSTR	5	–	μs
Tnvh	Nvstr Hold Time	5	–	μs
Tpgs	Nvstr to Program Setup Time	10	–	μs
Tpgh	Program Hold Time	20	–	ns
Tprog	Program Time	20	40	μs
Tads	Address/data Setup Time	20	–	ns
Tadh	Address/data Hold Time	20	–	ns
Trcv	Recovery Time	1	–	μs
Thv	Cumulative high-voltage programming time within the same row before next erase. The same address must not be programmed more than twice before next erase.	–	8	ms
Terase	ERASE Time	4	5	ms
	Endurance (erase/program cycles)	10000	–	cycles
	Data Retention at Room Temperature	100	–	years

3.5.6. Flash Programming Characteristics

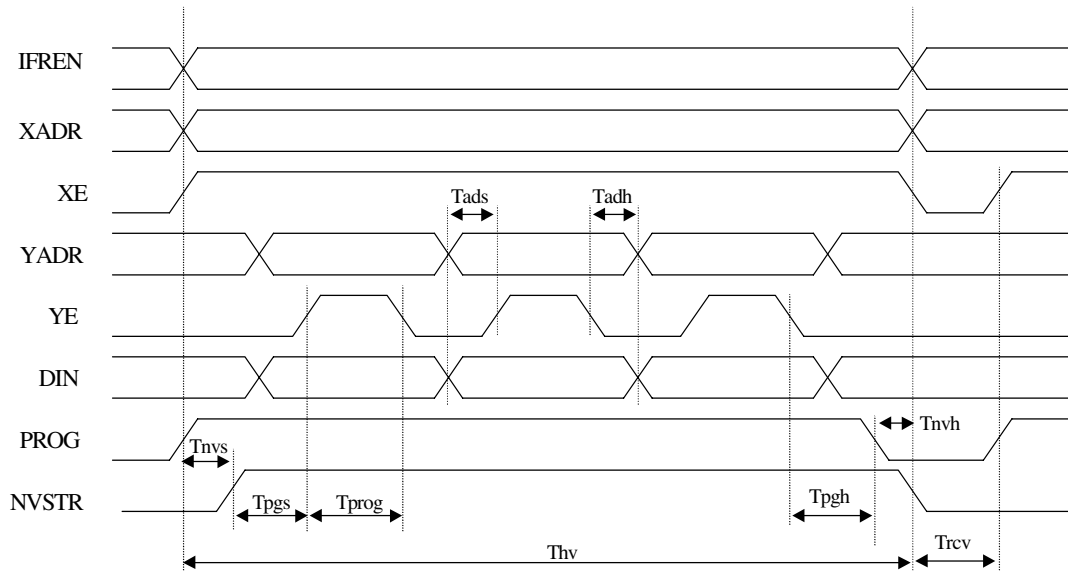


Fig. 3–13: Typical Flash Programming Cycle *

* Programming of more than one word per program cycle is available via Parallel Flash interface

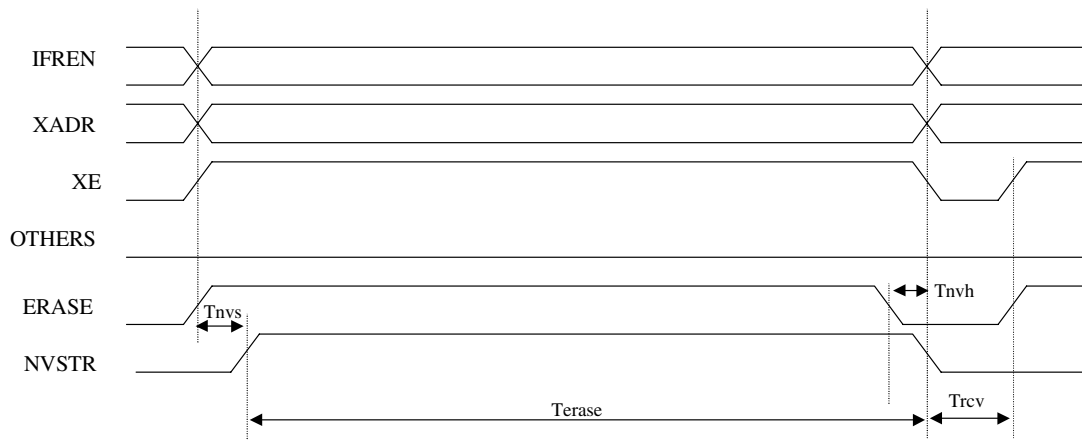


Fig. 3-14: Typical Flash Page Erase Cycle

4. List of Abbreviations

Abbreviation	Description
AHB	Advanced High-performance Bus; an AMBA bus
AMBA	Advanced Microcontroller Bus Architecture; a bus architecture specification from ARM Limited
APB	Advanced Peripheral Bus; an AMBA bus
ARM7TDMI	An ARM Processor Core developed by ARM Limited
CPSR	Current Processor Status Register; a status register on the ARM7TDMI core
CPU	Central Processing Unit; the ARM7TDMI core
DRM	Digital Rights Management
EP	Endpoint; source or sink of data on a USB peripheral
FIFO	First In First Out; a buffer strategy
FIQ	Fast Interrupt Request; an interrupt type with low latency on the ARM7TDMI core
GPIO	General Purpose Input/Output; the behavior of associated I/O-pins is programmable
I ² C	A serial control bus specification originated by Philips
I ² S	A serial bus specification originated by Philips; term also used for a variant by Sony
IrDA	Infrared Data Association; term is used for the association's infrared communication standards
IRQ	Interrupt Request; an interrupt type on the ARM7TDMI core
JTAG	Joint Test Access Group; a serial access protocol widely used for debugging and boundary scan.
LSB	Least Significant Bit; bit [0] within a data item
MPU	Memory Protection Unit
PLL	Phase-locked Loop
MSB	Most Significant Bit; bit with highest weight within a data item
PMU	Power Management Unit
PUC	Programmable Universal Controller
SIE	Serial Interface Engine; a serial/parallel conversion block within a USB peripheral
RTC	Real Time Clock
SPI	A serial protocol originated by Motorola
SPSR	Saved Processor Status Register; a status register on the ARM7TDMI core
SSP	Synchronous Serial Port; peripheral hardware for serial communication
TM	Test Mode
TPC	Test Port Controller; used for production testing
TTC	Triple Timer Counter module
UART	Universal Asynchronous Receiver Transmitter; a serial interface
USB	Universal Serial Bus
WDT	Watchdog Timer module

5. Data Sheet History

1. Preliminary Data Sheet: "PUC 303xA Programmable Universal Controller, April 2, 2002. First release of the preliminary data sheet.

Micronas GmbH
Hans-Bunte-Strasse 19
D-79108 Freiburg (Germany)
P.O. Box 840
D-79008 Freiburg (Germany)
Tel. +49-761-517-0
Fax +49-761-517-2174
E-mail: docservice@micronas.com
Internet: www.micronas.com

Printed in Germany
Order No. 6251-565-1PD

All information and data contained in this data sheet are without any commitment, are not to be considered as an offer for conclusion of a contract, nor shall they be construed as to create any liability. Any new issue of this data sheet invalidates previous issues. Product availability and delivery are exclusively subject to our respective order confirmation form; the same applies to orders based on development samples delivered. By this publication, Micronas GmbH does not assume responsibility for patent infringements or other rights of third parties which may result from its use.

Further, Micronas GmbH reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.

No part of this publication may be reproduced, photocopied, stored on a retrieval system, or transmitted without the express written consent of Micronas GmbH.