PiP-EC02
Embedded Controller
with Multi-Master
AMBA AHB/APB Architecture

Copyright Notice

Disclaimer

SOC SOLUTIONS L.L.C. AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

SoC™ Solutions and Pre-Integrated IP are trademarks of SoC Solutions, L.L.C.

Bugs and Suggestions

Please report bugs and suggestions about this document by sending electronic mail to info@socsolutions.com.

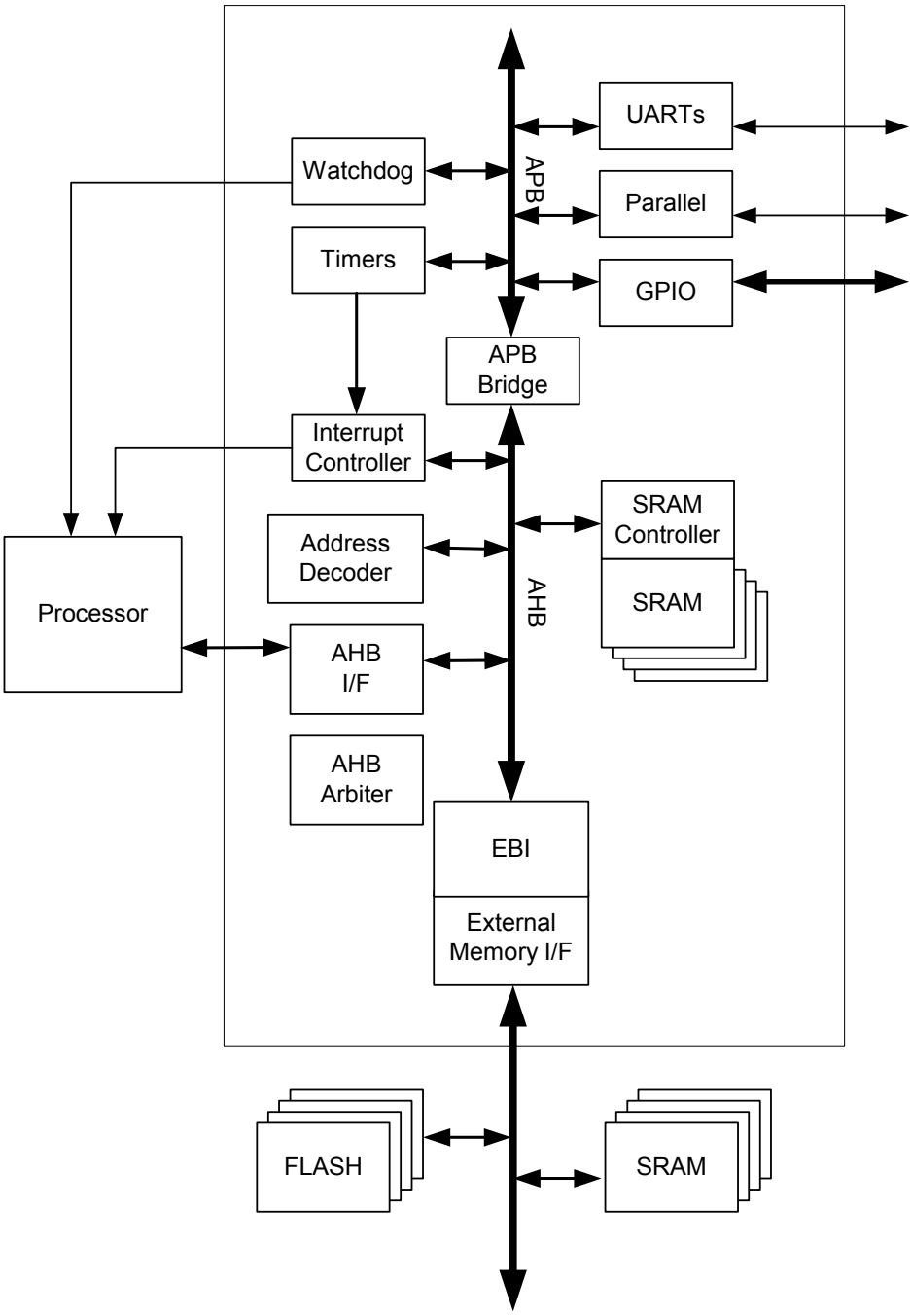
Change Log

Revision	Date	Author	Change
1.0	09/26/2006	Jim Avant	None

TABLE OF CONTENTS

1 System Overview

1.1 The Pre-integrated IP Embedded Controller with AMBA interface



The PiP-EC02 is an integrated ARM7TDMI™ based Verilog SoC. The PiP-EC02 provides the basic infrastructure for most of the popular SoC applications such as mobile phones, PDAs, personal navigators, as well as other electronic devices.

The architecture of the PiP-EC02 is based on the AMBA AHB & APB buses. It supports multiple masters on the AHB bus.

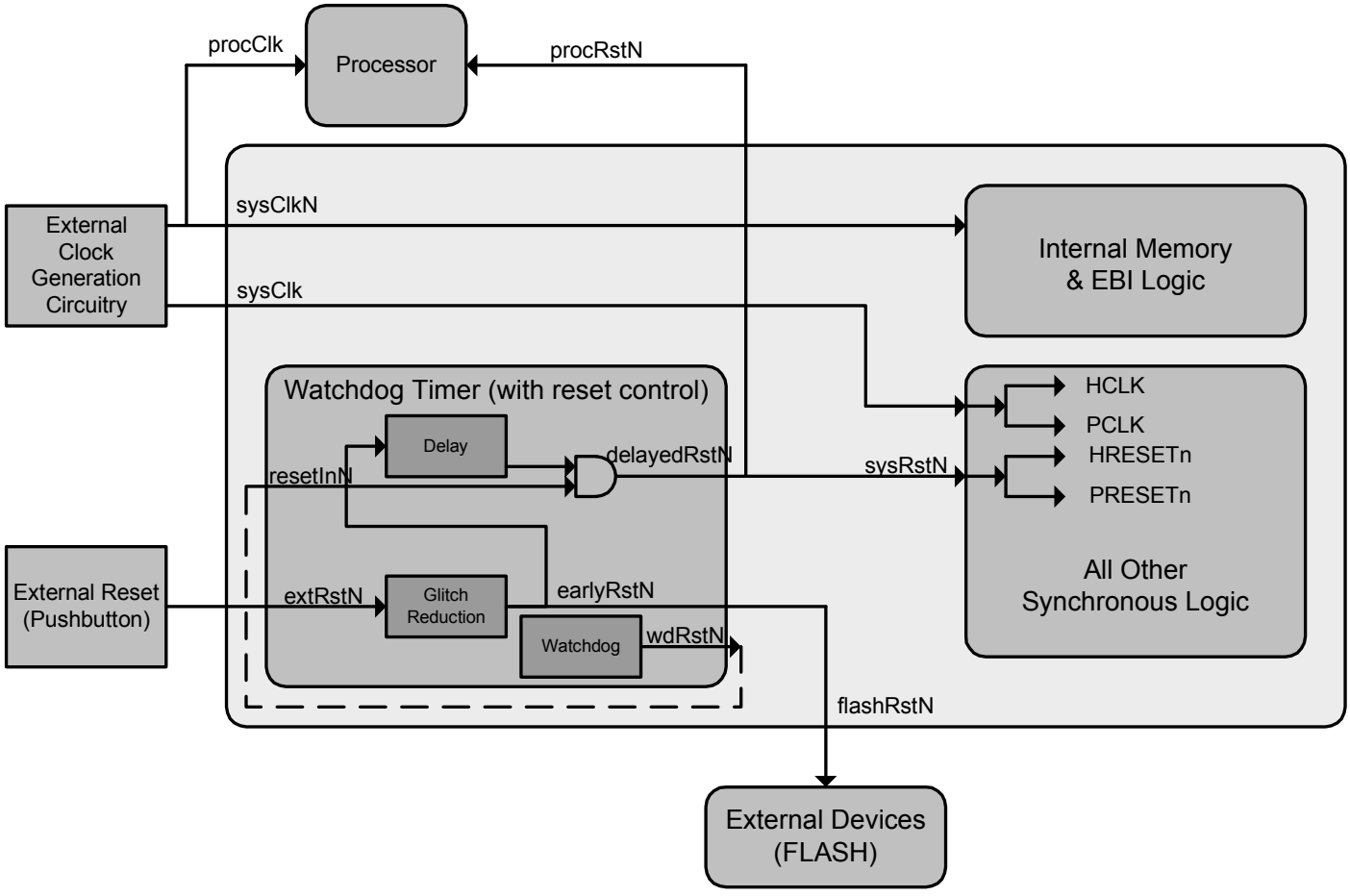
1.2 Peripheral Overview

Peripheral Name	Verilog Module Name	Format	Bus	Description
Processor Interface	socProcIf	source	AHB	Interfaces processor to AMBA AHB bus
AHB Arbiter	socAhbArb	source	AHB	AHB Arbiter for up to three bus masters
Address Decoder	socAddrDecRemap	source	AHB	Generates block select signals for each system block and provides an address remap utility. Additional address decoding for APB peripherals is done in the APB Bridge.
Interrupt Controller	socIntrCtrl	netlist	AHB	Monitors all system interrupts and issues interrupt requests to the processor
Internal Memory Interface	socIntMemIf	source	AHB	Interface to zero wait-state internal synchronous SRAM
External Bus Interface (EBI)	socEbi	netlist	AHB	Provides a configurable interface to external devices such as FLASH and RAM
External Memory Interface	socExtMemIf	source	MEM	Collects and distributes generic byte enables to your specific external memory configuration (back-end glue logic to EBI module)
SDRAM Controller (optional)	socSdrCtrl	netlist	AHB	Synchronous Dynamic RAM Controller.
Real Time Clock (optional)	socRTC	netlist	AHB	Real Time Clock module provides time-of-day and other data.
APB Bridge	socApbBridge	source	AHB	Bridge from AHB to APB
Timer	socTimer	netlist	APB	Time base generator for the system and general-purpose counter. Two used in PiP-EC02.
UART	socUart	netlist	APB	Provides a means of asynchronous serial communication with external devices. Two used in PiP-EC02.
Parallel Port	socParPort	netlist	APB	PC compatible parallel port which can be used for general purpose I/O
GPIO	socGpio	netlist	APB	Configurable general purpose parallel I/O module
Watchdog Timer	socWDTimer	netlist	APB	Resets the system in case of an unrecoverable error and provides reset signal control (optional at no additional cost)
PWM	SocPwm	netlist	APB	Generates a Pulse-Width-Modulated output

Several of the IP modules are delivered with full Verilog source to facilitate integration into your custom design. In addition to those listed above, the top-level module, a clock divider module, and header files with definitions of all registers are provided as verilog source. The rest of the modules are provided in a pair of netlist files: socAhbCore.edf & socApbCore.edf.

1.3 Clock and Reset Signals

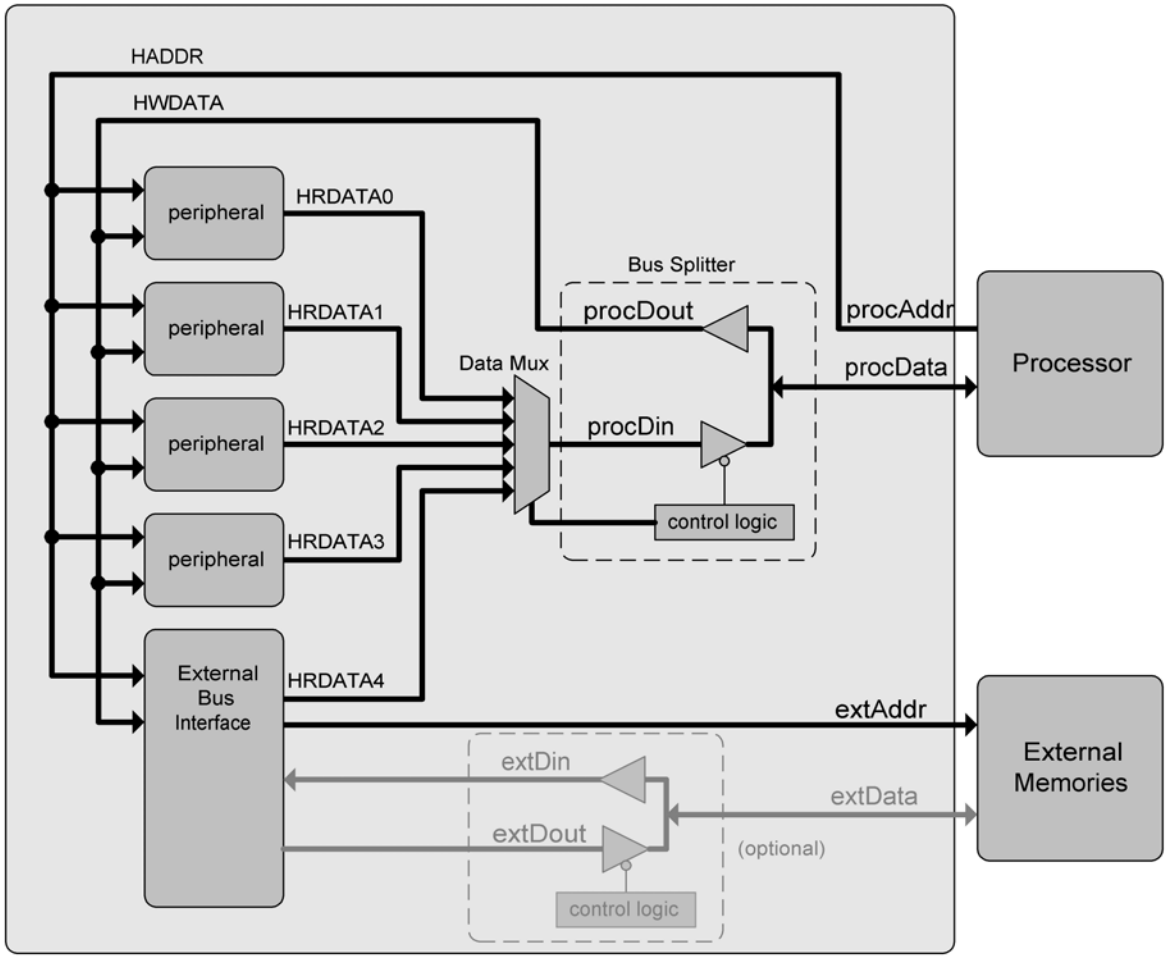
The following diagram illustrates the path of all clock and reset signals within the Embedded Controller with AMBA interface.



Most of the synchronous AMBA Embedded Controller Platform logic is driven from sysClk (which is the same as HCLK & PCLK), however the synchronous internal SRAM, some of the External Bus Interface (EBI) logic, and the processor is driven by sysClkN (or HCLKn).

The Watchdog Timer is an optional module. Even when it is not included, its reset logic is still present. This reset logic is responsible for generating reset signals for the internal logic, processor, and external devices. In the event of an external reset, any devices connected to flashRstN will be reset first, and then after some delay the rest of the system (including the processor) will be reset. This is done in order to allow external devices (such as FLASH) enough time to ensure their outputs are valid before the processor attempts to access the device.

1.4 Data and Address Bus

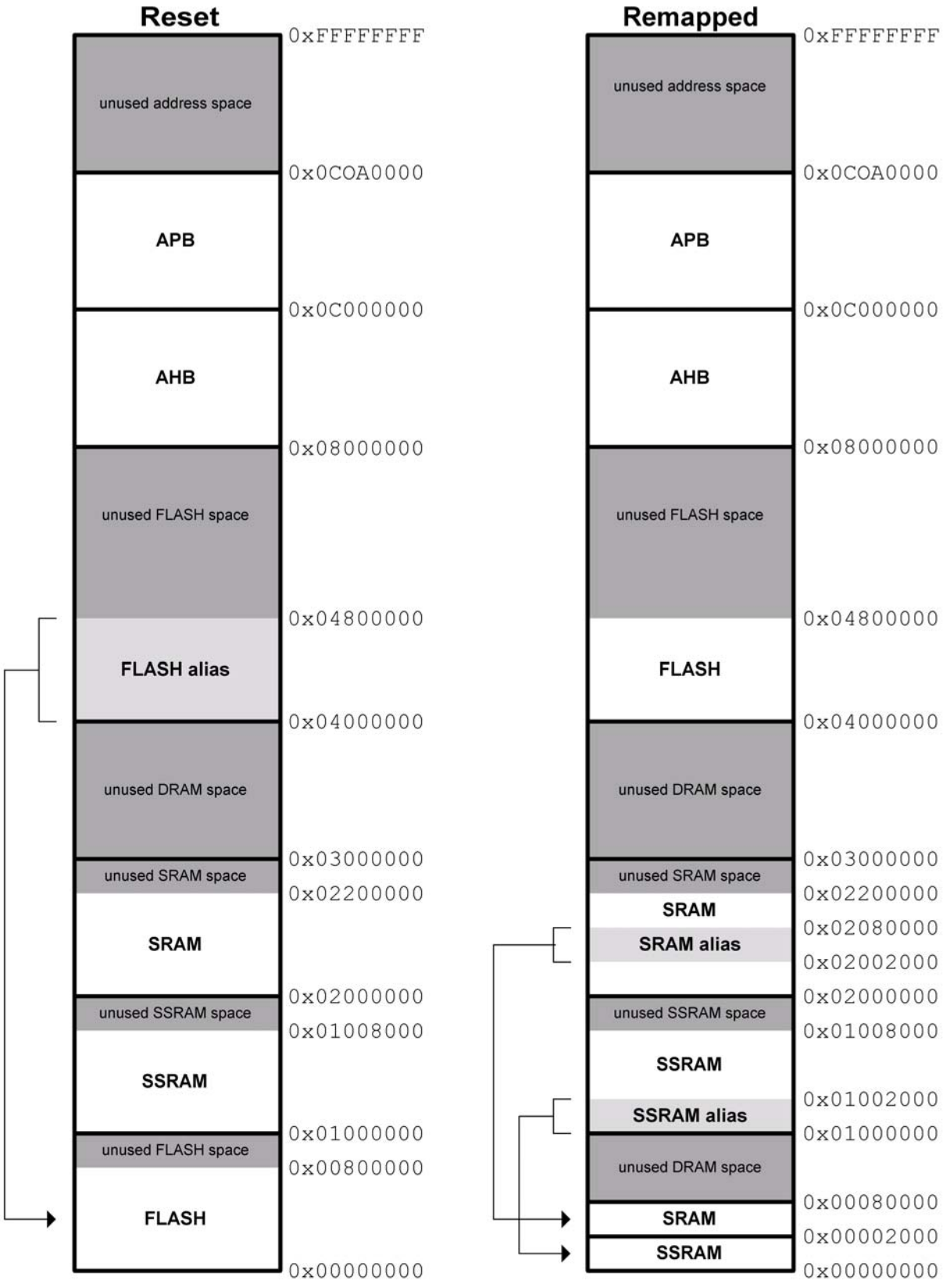


The EC02 provides a bus splitter to split the bi-directional data bus from the processor (procData) into two unidirectional data buses (HWDATA and HRDATA). The "din" and "dout" suffixes are named with respect to the processor, i.e. HWDATA is an output from the processor but an input to the peripherals while HRDATA is an input to the processor but an output of the peripherals (through the data mux). The AHB address bus (HADDR) is connected to each AHB slave peripheral as well.

Each peripheral has its own data output bus that is connected through a mux to HRDATA. In the EC02, these buses are named "HRDATAn", where n is an integer. The implementation of the data mux prevents the need for several tristate buffers deep within the design hierarchy. This keeps all tristate logic at the top level of the design in order to facilitate integration with many ASIC and FPGA architectures.

The External Bus Interface (EBI) takes HADDR as an input and drives the external address bus (extAddr) for external devices such as FLASH and RAM. Optionally, the EBI can drive an external bi-directional data bus (extData) for external devices, however this logic is omitted when external devices share the processor data bus. Note that ALL "din" and "dout" signals are consistently named with respect to the processor.

1.5 Memory Maps



This memory map corresponds to the memory resources on various boards from SoC Solutions. There is up to 16 Mbyte FLASH available, 1 Mbyte external SRAM available, and internal SSRAM available (size depends on FPGA & its contents).

1.6 Top-Level Signal Descriptions

The following table lists the top-level signals of the PiP-EC02 platform as targeted to the SoC Solutions Brain board. Note that all signals ending with an uppercase 'N' are active low.

Signal Name	I/O	Description
pbRstN	I	external reset (power-on w/ pushbutton)
poRstN	I	external reset (no pushbutton)
flashRstN	O	flash reset
procRstN	O	processor reset (nRESET)
sysClk	I	clock driving almost main synchronous EC02 logic (from PLL)
refClk	I	Reference clock to external PLL
procClk	O	Clock to processor
procAddr[31:0]	I	address bus from processor (a.k.a. A)
procData[31:0]	I/O	bi-directional data bus connected to processor (a.k.a. D)
fiqOutN	O	fast interrupt request (a.k.a. nFIQ)
irqOutN	O	interrupt request (a.k.a. nIRQ)
procWrRdN	I	processor write/read not (1=write 0=read) (a.k.a. nRW)
procWaitN	O	processor wait (a.k.a. nWAIT)
procApe	O	processor address pipeline enable signal
procAccType[1:0]	I	processor access type (a.k.a. SEQ & nMREQ)
procByteLatch[3:0]	O	processor byte latch (a.k.a. BL; used only for ARM7 test chip))
procTestChipSel [1:0]	O	data bus routing signal for processor test chip (a.k.a. SEL; used only for ARM7 test chip)
procDsize[1:0]	I	processor data size (8/16/32 bit) (a.k.a. MAS)
procLock	I	processor LOCK signal
procTransN	I	processor nTRANS signal
procAbort	O	processor abort (a.k.a. ABORT)
rs232Atx	O	serial channel A, transmit line
rs232Arx	I	serial channel A, receive line
rs232Btx	O	serial channel B, transmit line
rs232Brx	I	serial channel B, receive line
rs232BctsN	I	serial channel B, clear to send
rs232BdsrN	I	serial channel B, data set ready
rs232BdtrN	O	serial channel B, data terminal ready
rs232BrtsN	O	serial channel B, request to send
extAddr[25:2]	O	external address to FLASH, SRAM, driven by EBI
extData[31:0]	I/O	(optional) bi-directional data bus connected to external devices
hiFlashCeN	O	chip enable for device driving upper 16 bits of FLASH data
loFlashCeN	O	chip enable for device driving lower 16 bits of FLASH data
sramCeN	O	chip enable for device driving SRAM chips
SramBsN[3:0]	O	byte enables for SRAM
extOeN	O	external output enable
extWeN	O	external write enable
gpio[31:0]	I/O	general purpose input/output
ledOut[9:0]	O	output to drive user-defined LEDs
switchIn[9:0]	I	input from DIP switches on board
pwmOut1	O	Pulse Width Modulator # 1 Output
pwmOut2	O	Pulse Width Modulator # 2 Output

1.7 Interrupt Source Bit Assignments

Interrupt Source Bit	Source of Interrupt
[0]	Programmable Interrupt
[1]	unused
[2]	unused
[3]	unused
[4]	Timer 1
[5]	Timer 2
[6]	unused
[7]	unused
[8]	UART A
[9]	UART B
[10]	Parallel Port
[11]	unused
[12]	unused
[13]	unused
[14]	unused
[15]	unused
[16]	Watchdog Timer
[17]	General Purpose I/O
[31:18]	unused

1.8 System Configuration

A header file (socSysDefs.hv) contains system-level definitions for configuring options such as:

- Whether the EC02 is connected to an ARM7TDMI core or to a test chip
- Whether or not to allow clocks in combinational logic (for optimizing EBI)
- Whether or not to use a clock divider module to divide down the clock for the UART baud rate clock
- Whether external memories' data buses are connected directly to the processor or thru the EBI
- Internal memory sizes
- External memory sizes
- Data widths of parameterizable modules

Note that some of these definitions have already been used to compile the AHB & APB core files into EDIF netlists; therefore, changes in the definitions will not affect these modules.

1.9 Timing Considerations

The two areas most likely to be critical paths for timing are in the areas of address decoding and read data bus multiplexing. The current implementation favors lower area over system clock speed.

2 Multi-Master AHB

2.1 AHB Arbiter

The AHB Arbiter block processes bus requests (HBUSREQ) from the various AHB masters and grants the AHB bus to one master at a time. The arbiter was designed for 3 bus masters. Only one is present in this design: the processor.

The arbitration algorithm is round-robin. Each master takes a turn at being highest priority. The processor interface is the default bus master when no masters are requesting the bus.

The arbiter block also takes care of data and control bus muxing and demuxing.

2.2 Processor Interface

2.2.1 Overview

The processor interface module provides an AHB Master interface from the processor to the AMBA AHB bus. This module contains the processor-specific logic. The processor interface for the ARM7TDMI is documented here.

A header file associated with this module (socProcIf.hv) contains processor- and AHB-specific definitions.

The Processor Interface consists of a state machine and some glue logic. Note that the data and control muxing is done in the Arbiter block and not here.

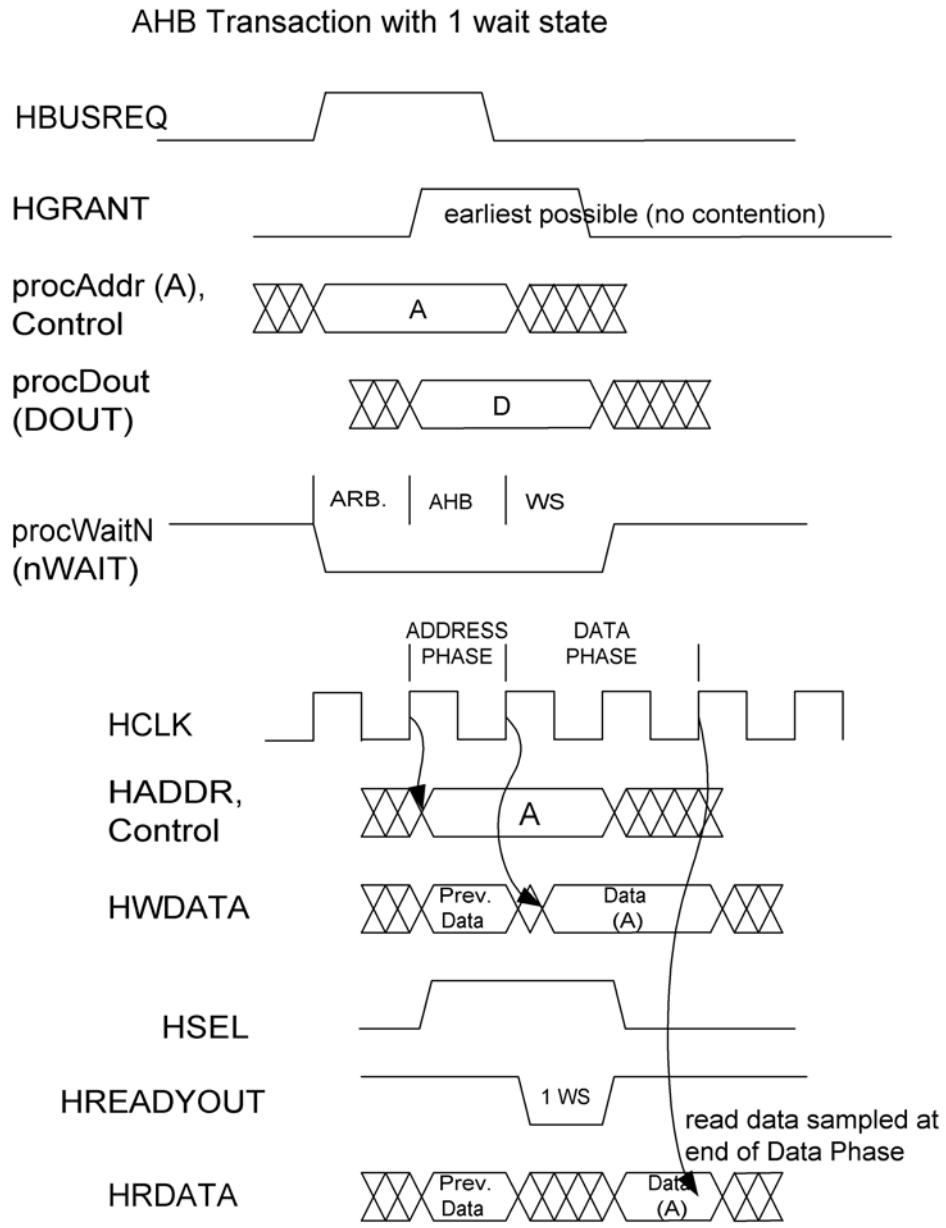
The processor interface glue logic performs the following functions:

- The processor address, write data, and control signals are sampled (if APE=1)
- Block byte latch signals are muxed to the processor
- The wait state request to the processor is inhibited for 2 clocks after reset
- The procAbort signal is generated
- The HTRANS signal is generated for the AHB bus

Note that this interface does not provide for the following AHB features:

- Split bus transactions
- Burst operation
- Transfer sizes (HSIZE) larger than 32 bits
- Protection Control

2.2.2 Timing



An AHB bus master begins a transaction by asserting its HBUSREQ signal. After one or more clock cycles, it will see its HGRANT signal asserted. From here, the bus master's state machine controls the access. Each AHB bus transaction consists of a single cycle address phase followed by one or more cycles of a data phase. The processor-to-AHB interface asserts the first two or more cycles of the wait state signal (nWAIT) during arbitration and the AHB address phase. After that, any AHB peripheral needing more than a single cycle must assert its HREADYOUT signal to sustain the wait.

2.3 Other Masters

Other IP blocks are available with AHB master interfaces for this platform such as Ethernet MAC, SDRAM Controller, & LCD Controller blocks. These interfaces usually have an AHB slave interface for control & status and an AHB master interface for DMAing data to and from system memory. These IP blocks are not part of the EC02 platform but are available separately from SoC Solutions.

3 AHB Peripherals

3.1 Address Decoder

3.1.1 Overview

The Address Decoder logic consists of the following blocks:

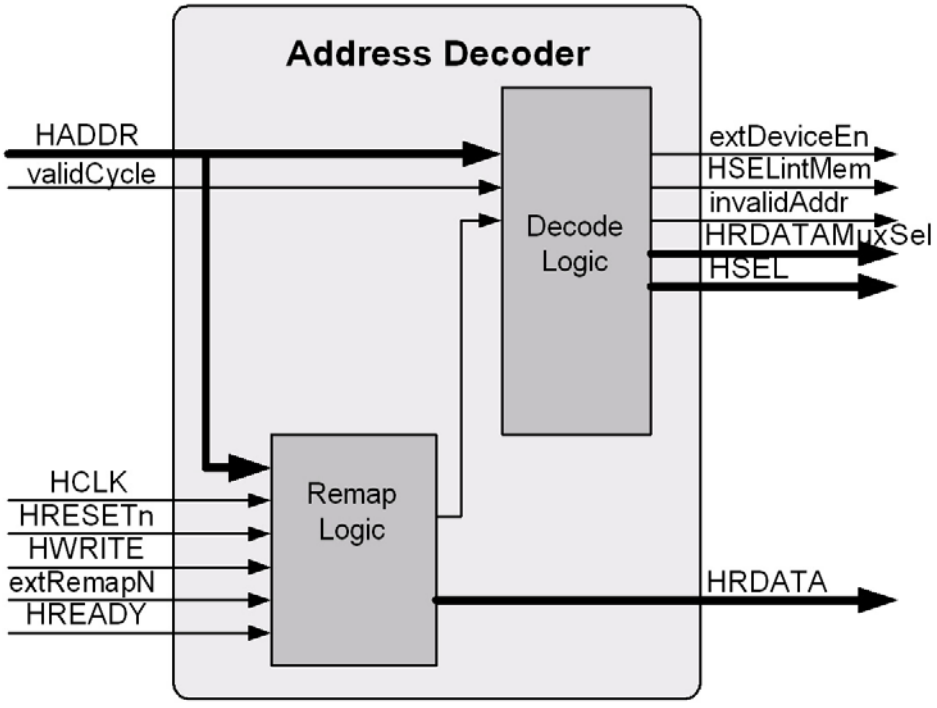
- address decoder case statement
- remap logic

The case statement in the address decoder is pretty simple, but generates a large amount of combinational logic that can easily contribute to the worst-case timing path in the design.

The address decoder case statement generates block selects for the various AHB blocks in the design and the control for the data mux located in the Processor Interface (socProcIf) module. Additional address decoding for APB peripherals is done in the APB Bridge (socApbBridge) module.

In the following discussions of memory mapping, a default memory map is used as an example. Please keep in mind that memory and registers can be placed anywhere in memory by changing the address decoder module.

3.1.2 Block Diagram



3.1.3 Remap Logic

The remap logic provides for two different memory maps. The reset memory map is used to put the ROM or FLASH at memory address 0 at power-on reset so the device can boot from non-volatile memory. (See the system overview section.) After the code has been initialized, a remap register is written and the normal (remapped) mode is entered where SRAM appears at address 0. During system development, the EC02 can be booted in the normal (remapped) mode with volatile memory at address 0 where an in-circuit emulator or similar device handles memory initialization.

These memory maps are controlled by the extRemapN input to the EC02 and by the remap configuration register. The extRemapN signal comes from bit 9 (MSB) of the switchIn inputs. It is an active low signal which when driven low puts the EC02 into its normal (remapped) mode with SSRAM at address 0. When extRemapN is driven high, the EC02 is put into its reset mode with ROM or FLASH at address 0. This signal is driven low when booting from volatile memory is necessary (e.g. using an in-circuit emulator), in this case the remap configuration register has no effect on the memory map.

When the system needs to be booted from non-volatile memory (when operating without an in-circuit emulator) the extRemapN signal should be driven high so that the ROM or FLASH appears at address 0. In this case, the programmer has the option to remap the system after all necessary initializations by writing to the remap configuration register. Once this register has been written, the reset memory map cannot be restored without a system reset.

3.1.4 AHB Peripheral Memory Map

Peripheral/Device	Base Address	Address Range	Address Space (bytes)
Remap (Address Decoder)	0x08000000	0x08000000 – 0x080FFFFF	64
Interrupt Controller	0x08100000	0x08100000 – 0x081FFFFF	256
External Bus Interface	0x08300000	0x08300000 – 0x083FFFFF	16
APB Peripherals	0x0C000000	0x0C000000 – 0x0C0FFFFF	64K

The address decoder asserts an AHB block select signal (HSEL) any time an address present on the address bus (HADDR) falls within the address range of the respective peripheral. These addresses are valid in both reset and normal (remapped) mode.

3.1.5 Programming Interface

3.1.5.1 Register Summary

Register Name	Offset	Description
remapStatus	0x00	Indicates state of related remap signals.
clearResetMap	0x20	This register changes the memory map from the reset memory map to the remapped (normal) memory map.

3.1.5.2 Register Descriptions

3.1.5.2.1 remapStatus

Offset: 0x00

Bits	Access	Default	Description
[2]	Read	depends on state of extRemapN	Indicates whether or not the system is in its remapped state or its reset state (regardless of whether from extRemapN or from the clearResetMap register). This bit is simply the OR of bits 0 and 1.
[1]	Read	depends on state of extRemapN	Indicates state of extRemapN signal. When 1, extRemapN==0. When 0, extRemap==1.
[0]	Read	0x0	Indicates whether or not the system has been remapped by writing to the clearResetMap register. When 1, system has been remapped, when 0, system is still under the reset memory map.

3.1.5.2.2 clearResetMap

Offset: 0x20

Bits	Access	Default	Description
[0]	Write	0x0	When this register is written to, the reset memory map with ROM at address 0 will be cleared and the memory will be remapped to the normal memory map with RAM at address 0. Any value written to this register will cause the remap to occur.

3.2 Internal Memory and Interface

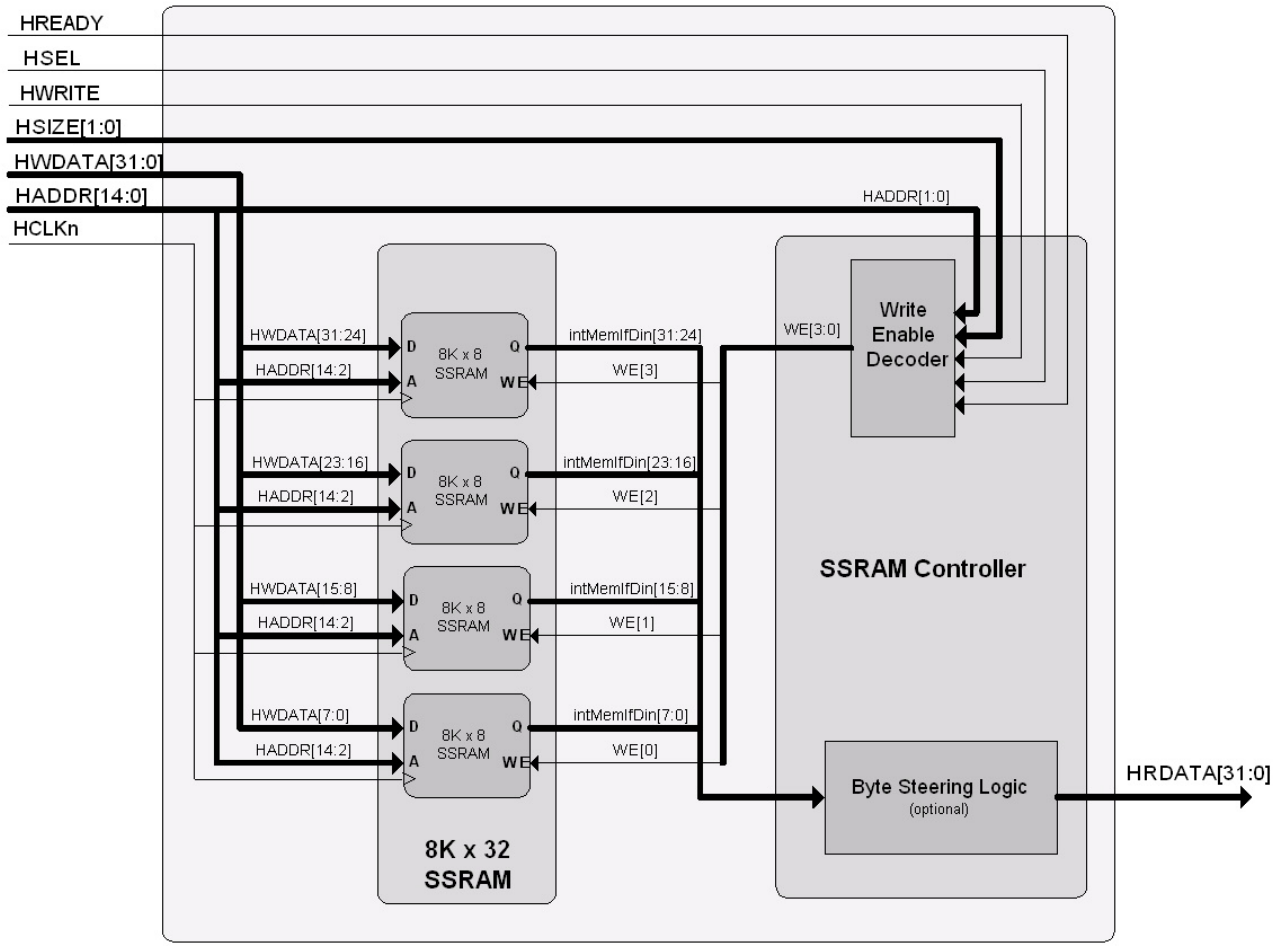
3.2.1 Overview

The EC02 internal memory consists of Synchronous Static Random Access Memory (SSRAM) and some memory interface logic. The SSRAM is configured as 8 Kwords where each word is 32 bits wide (8K x 32) for a total of 32 KB. Physically, the SSRAM consists of 4 banks of 8 Kbytes each. This allows the memory interface to offer word, half-word, or byte wide addressing. When writing to half-word or byte width addresses, it is assumed that the processor replicates the data across the entire input data bus. Similarly, when reading, it is assumed that the processor knows where the data should appear (no byte steering is done). If byte steering is required, the Processor Interface may be ordered with this option or byte steering may be done externally. The connections for the default configuration are illustrated below.

The amount of SSRAM is easily configurable to meet your system requirements. Your ASIC SRAM blocks can easily replace the existing Xilinx SRAM instantiations.

The write enable decoder shown in the block diagram uses HSIZE & the LSBs of the address (HADDR) to produce write enables to the SRAM blocks whenever an SSRAM address is selected (HSEL) for writing (HWRITE).

3.2.2 BlockDiagram



3.2.3 Signal Descriptions (memory interface)

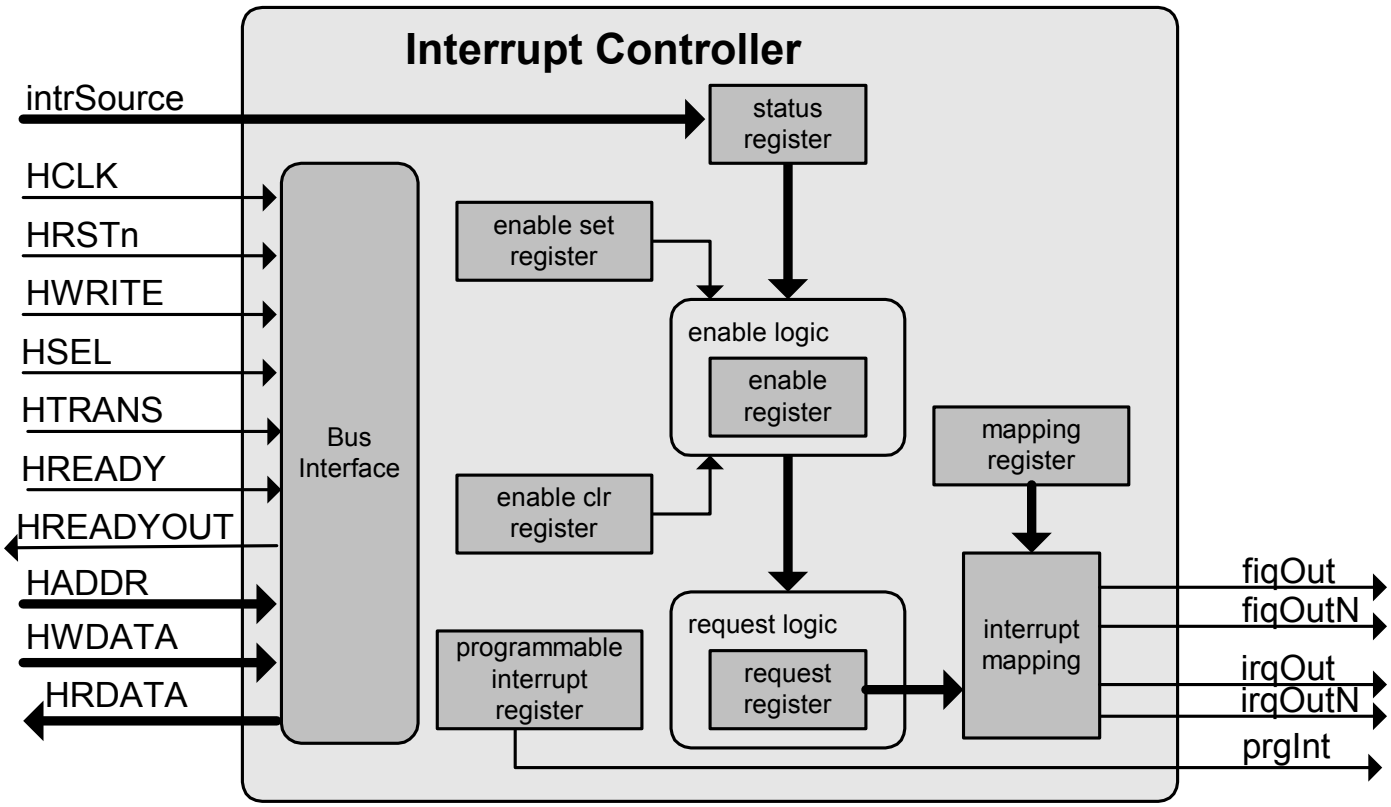
Signal Name	I/O	Description
HWRITE	I	write from AHB (0 = read, 1 = write)
HADDR[1:0]	I	LS bits of address bus from AHB, used to decode write enables
HSIZE[2:0]	I	AHB data size
HSEL	I	AHB Block select
HREADY	I	Combined AHB ready signal from all AHB slaves
ramDout[31:0]	I	Data output from SSRAM
HRDATA[31:0]	O	AHB read data
writeEn[3:0]	O	Write enable signals to each byte wide SSRAM

3.3 Interrupt Controller

3.3.1 Overview

The interrupt controller monitors interrupts from all other modules within the system and issues interrupt requests to the processor when necessary. The interrupt controller is scalable to support from 1 to 32 interrupt sources. It also provides enable set and enable clear mechanisms to prevent dangerous read-modify-write operations. It provides active high & active low IRQ & FIQ interrupt request outputs. Multiple interrupt controllers may be cascaded if more than 32 interrupts are required.

3.3.2 Block Diagram



3.3.3 Signal Descriptions

Signal Name	I/O	Description
HRESETn	I	Active low system reset
HCLK	I	System clock
HWRITE	I	AHB write (0=read 1=write)
HWDATA[31:0]	I	AHB write data
HADDR[2:0]	I	AHB Address bus
HSEL	I	AHB Block select, enables the interrupt controller
HTRANS[1:0]	I	AHB Transfer type
HREADY	I	Combined AHB ready signal from all AHB slaves
intrSource[NUM_INTS-1:0]	I	Interrupt source bus (all interrupts connect to this bus)
HREADYOUT	O	Ready output to AHB
HRDATA[31:0]	O	AHB read data bus
progInt	O	State of the programmable interrupt register
irqOutN	O	IRQ Interrupt request line (active low)
irqOut	O	IRQ Interrupt request line (active high)
fiqOutN	O	FIQ Interrupt request line (active low)
fiqOut	O	FIQ Interrupt request line (active high)

3.3.4 Programming Interface

3.3.4.1 Register Summary

Register Name	Offset	Description
request	0x0	State of enabled interrupt sources
status	0x4	State of all interrupt sources (enabled or disabled)
enable	0x8	Mask register to enable/disable individual interrupts
enableSet	0x8	Used to set bits in the enable register
enableClr	0xC	Used to clear bits in the enable register
progIntr	0x10	Programmable interrupt register
mapping	0x14	Map interrupt to FIQ or IRQ

3.3.4.2 Register Descriptions

3.3.4.2.1 request

Offset: 0x0

Bits	Access	Default	Description
[31:0]	Read	0x0	Each bit that is set indicates that the corresponding interrupt source is active and enabled.

3.3.4.2.2 status

Offset: 0x4

Bits	Access	Default	Description
[31:0]	Read	0x0	Each bit that is set indicates that the corresponding interrupt source is active.

3.3.4.2.3 enable

Offset: 0x8

Bits	Access	Default	Description
[31:0]	Read	0x0	Each bit that is set indicates that the corresponding interrupt source is enabled.

3.3.4.2.4 enableSet

Offset: 0x8

Bits	Access	Default	Description
[31:0]	Write	0x0	Each bit that is set causes the corresponding bit in the enable register to be set. This is a self-clearing register.

3.3.4.2.5 enableClr

Offset: 0xC

Bits	Access	Default	Description
[31:0]	Write	0x0	Each bit that is set causes the corresponding bit in the enable register to be cleared. This is a self-clearing register.

3.3.4.2.6 progIntr

Offset: 0xC

Bits	Access	Default	Description
[31:2]	Write	0x0	Unused, these bits have no effect on the interrupt controller.
[1]	Write	0x0	This is the programmable interrupt. Setting this bit activates the programmable interrupt and drives the programmable interrupt output signal high. Clearing this bit clears the interrupt request and drives the programmable interrupt output signal low.
[0]	Write	0x0	Unused, this bit has no effect on the interrupt controller.

3.3.4.2.7 mapping

Offset: 0x14

Bits	Access	Default	Description
[31:0]	R/W	0x0	Each bit that is set causes the corresponding enabled interrupt to activate the FIQ interrupt output. Cleared bits map interrupts to the IRQ interrupt output. This register should only be set up once at system initialization; therefore, there are not separate set and clear registers.

3.3.5 Functional Description

All signals connected to the interrupt source input bus (intrSource) must be registered at their source since these signals are not registered within the interrupt controller. The interrupt sources are expected to be active high and level sensitive. Each interrupt source corresponds to a single bit within the enable, enableSet, enableClr, status, and request register. For example, the interrupt source connected to bit 5 of the intrSource bus is enabled by writing to bit 5 of the enableSet register and disabled by writing to bit 5 of the enableClr register. If this interrupt source is enabled, then bit 5 of the enable register will be set. If this interrupt source is active and enabled, then bit 5 of the request register will be set.

The status register reflects the state of the interrupt sources connected to the interrupt controller. The values in this register are the same as what the interrupt controller sees on its interrupt source input bus (intrSource) since no registering is done within the interrupt controller. The enable register has no effect on this register. The status register is read only.

The enable register provides information as to which interrupts are enabled. Upon reset, all sources are disabled (all bits in this register are cleared). The enable register is read only and can only be modified by writing to the enableSet and enableClr registers.

The request register reflects the state of enabled interrupt sources. Any set bit in this register indicates that the corresponding interrupt source is active and enabled. If a particular source is disabled, the corresponding bit in the request register will be clear regardless of the current state of the source. The request register is read only.

The enable set and enable clear registers provide a mechanism to modify the enable register directly without the need for a potentially dangerous read-modify-write operation. Since these registers are self-clearing, only a single register write is required to set or clear a bit in the enable register. Each bit that is set in the enable set register causes that same bit to be set in the enable register. Each bit that is set in the enable clear register causes the same bit to be cleared in the enable register. The enable set and enable clear registers are write only.

The interrupt controller provides four interrupt request outputs, irqOut, irqOutN, fiqOut, and fiqOutN, where the ‘N’ suffix signifies an active low signal. The mapping register is used to map an enabled interrupt to either the IRQ or FIQ outputs. An active interrupt request output means that there is at least one enabled pending interrupt request that is mapped to that output. These signals will not be deasserted until all mapped interrupt sources are inactive.

The interrupt controller also provides an option to force an interrupt by writing to a programmable interrupt register. This option is enabled by connecting the programmable interrupt output (progInt) to one of the interrupt source inputs. The progInt output simply reflects the current state of the programmable interrupt register, therefore any time that the programmable interrupt is set, it can generate an interrupt request just like any other interrupt source. This configuration is illustrated in the appendices.

3.4 External Bus Interface (EBI)

3.4.1 Overview

The external bus interface (EBI) allows the processor to transmit and receive data to and from external devices, one or more of which may be memory. External memory can be SRAM, Flash, etc. (see limitations below). The user programs the number of read & write wait states and the memory size and the EBI allows the proper communication. The EBI allows word, half-word, and byte width addressing to 32-bit, 16-bit, and 8-bit external devices.

3.4.1.1 Limitations

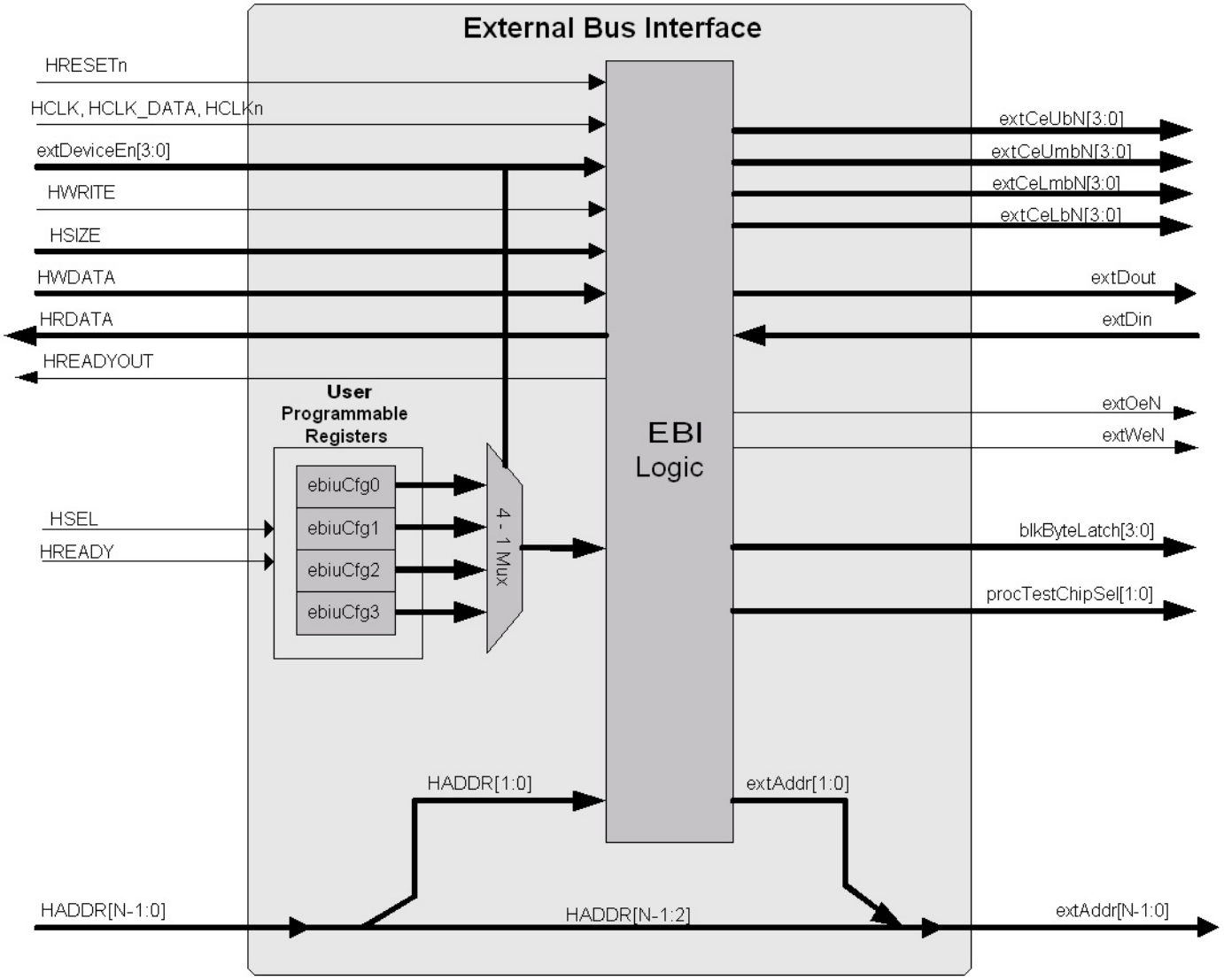
3.4.1.1.1 Timing

Note that at least one wait state is always generated to the processor for each bus transaction. This is done in the AHB master state machine in the socProcIf module. So from the external memory's point of view, the access is performed with the number of wait states selected in the EBI configuration register, but from a system point of view, there is an additional wait state due to the AMBA architecture. This penalty can be minimized by using a cached version of the ARM7 such as an ARM720 or ARM740.

3.4.1.1.2 Configurability

Ideally the user would be able to specify the number of external devices when the EBI is instantiated; however, the Verilog language does not fully support this. The Verilog 2000 spec. supports this so this capability may be added in the future.

3.4.2 Block Diagram



The EBI interfaces the processor with up to 4 external devices. Each device has a configuration register. The HSEL signal is generated by the address decoder and is used in conjunction with some lower address bits to read and write the configuration registers. The extDeviceEn signals also come from the address decoder (and optionally through a remapper). When one of these is active, its corresponding configuration register is selected. The 2 LSBs of the processor address bus are modified depending on the type of access specified by the processor and the capabilities of the device (as programmed into the configuration register). The EBI logic performs this and other tasks such as:

- breaks down device enables into upper, upper middle, lower middle, & lower bank chip enables
- steers data bytes or halfwords onto appropriate data bits
- sequences multi-byte or multi-halfword read and write operations
- puts the processor in a wait state until the access completes

3.4.3 Signal Descriptions

Signal Name	I/O	Description
HCLK	I	AHB clock
HRESETn	I	AHB reset (active low)
HADDR[31:0]	I	AHB address bus
HWDATA[31:0]	I	32-bit data bus from AHB
HWRITE	I	write select from AHB (0 = read, 1 = write)
HSIZE[2:0]	I	memory access size select signal from AHB
HSEL	I	AHB block select signal for configuration registers
extDeviceEn[3:0]	I	Device enables from address decoder
extDin[31:0]	I	32-bit data bus from external devices
extCeLbN[3:0] extCeLmbN[3:0] extCeUmbN[3:0] extCeMbN[3:0]	O	chip enables for 4 external devices – Lower Bank chip enables for 4 external devices – Lower Middle Bank chip enables for 4 external devices – Upper Middle Bank chip enables for 4 external devices – Upper Bank
extOeN	O	output enable for external devices
extWeN	O	write enable for external devices
HREADY	I	Combined AHB ready signal from all AHB slaves
HREADYOUT	O	ready signal produced by EBI, used to hold AHB while memory access completes
HRDATA[31:0]	O	32-bit data bus to processor
extDout[31:0]	O	32-bit data bus to external devices
extAddr[N_EXT_ADDR-1:0]	O	address bus to external devices
blkByteLatch[3:0]	O	byte latch controls to processor
procTestChipSel[1:0]	O	SEL inputs to ARM test chip

3.4.4 Programming Interface

3.4.4.1 Register Summary

Register Name	Address	Description
ebiCfgN	Base+4n	Configuration register corresponding to extCe…N[n] for: • bus width • inhibit writes • read wait states • write wait states

3.4.4.2 Register Descriptions

3.4.4.2.1 ebiCfg0

offset: 0x0

Bits	Signal	Access	Default	Description
[15:12]	reserved	R/W	0000	reserved
[11:8]	writeWaits	R/W	0111	number of wait states in clock cycles required for a write
[7:4]	readWaits	R/W	0111	number of wait states in clock cycles required for a read
[3]	reserved	R/W	0	reserved
[2]	inhibitWrite		0	Prevent writes to this device
[1:0]	busWidth	R/W	10	Data bus width of device used: 0 = 8 bits (byte) 1 = 16 bits (half word) 2 = 32 bits (word) 3 = reserved

3.4.4.2.2 ebiCfg1

offset: 0x4

Bits	Signal	Access	Default	Description
15:12	reserved	R/W	0000	reserved
11:8	writeWaits	R/W	0011	number of wait states in clock cycles required for a write
7:4	readWaits	R/W	0010	number of wait states in clock cycles required for a read
3	reserved	R/W	0	reserved
2	inhibitWrite		0	Prevent writes to this device
1:0	busWidth	R/W	10	Data bus width of device used: 0 = 8 bits (byte) 1 = 16 bits (half word) 2 = 32 bits (word) 3 = reserved

3.4.4.2.3 ebiCfg2

offset: 0x8

Bits	Signal	Access	Default	Description
15:12	reserved	R/W	0000	reserved
11:8	writeWaits	R/W	0011	number of wait states in clock cycles required for a write
7:4	readWaits	R/W	0010	number of wait states in clock cycles required for a read
3	reserved	R/W	0	reserved
2	inhibitWrite		0	Prevent writes to this device
1:0	busWidth	R/W	10	Data bus width of device used: 0 = 8 bits (byte) 1 = 16 bits (half word) 2 = 32 bits (word) 3 = reserved

3.4.4.2.4 ebiCfg3

offset: 0xC

Bits	Signal	Access	Default	Description
15:12	reserved	R/W	0000	reserved
11:8	writeWaits	R/W	0010	number of wait states in clock cycles required for a write
7:4	readWaits	R/W	0010	number of wait states in clock cycles required for a read
3	reserved	R/W	0	reserved
2	inhibitWrite		0	Prevent writes to this device
1:0	busWidth	R/W	10	Data bus width of device used: 0 = 8 bits (byte) 1 = 16 bits (half word) 2 = 32 bits (word) 3 = reserved

3.4.4.3 Common Setups

The following setup represents the requirements of the SoC Brain board:

Device #	Memory	Size	Bus Width	Config.
0	FLASH	2Mx16 2Mx16	32	0x0772
1	SRAM	256Kx16 256Kx16	32	0x0322

3.4.5 Instantiation Parameters

The following are options that can be selected when the EBI is instantiated:

1. N_EXT_ADDR – modify this definition to specify number of external address bits.

3.4.6 Functional Description

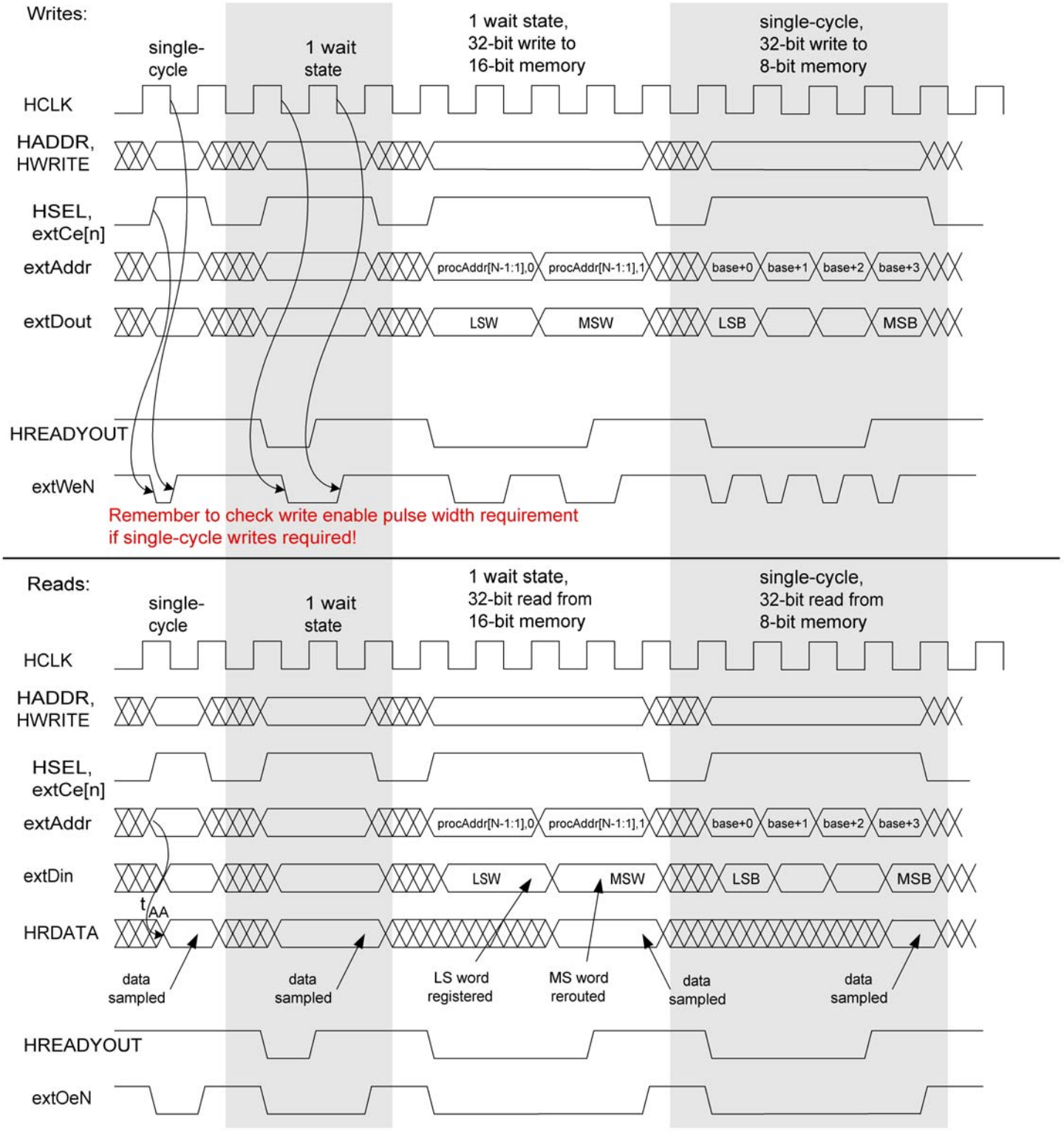
The external bus interface (EBI) allows memory to be written to in word, half-word, and byte width; 32-bit, 16-bit, and 8-bit memories are acceptable. To do this, all bits are sent by the processor (thru the AHB) along with the HSIZE signal. The data is replicated across the entire data bus by the processor when writing in any width other than 32-bit. The EBI uses the HSIZE signal generated by the processor to determine whether the data is to be written as word, half-word, or byte width.

When writing to memories which are smaller than the data bus width, the EBI uses the HSIZE signal as well as the last 2-bits of the processor address (HADDR) signal to determine which byte(s) to write. For example if one wants to write the first half-word to an 8-bit memory, the two least significant bits of HADDR are assigned 00 for the least significant byte and 01 for the other. This allows a half-word to be written to an 8-bit memory.

For reading, a similar process is used. The data is always be shifted into the least significant bits. The processor uses the HSIZE signal to determine which byte or bytes it needs to read. The data is always presented to the processor in the least significant bits.

All addressing is assumed to be little-endian.

3.4.7 Timing Diagrams



Note that a single-cycle write can only be accomplished by using the clock in combinational logic to generate the write enable pulse. This is disabled by default but may be enabled in socSysDefs.hv.

3.5 SDRAM Controller

This module is optional and is documented separately.

3.6 APB Bridge

3.6.1 Overview

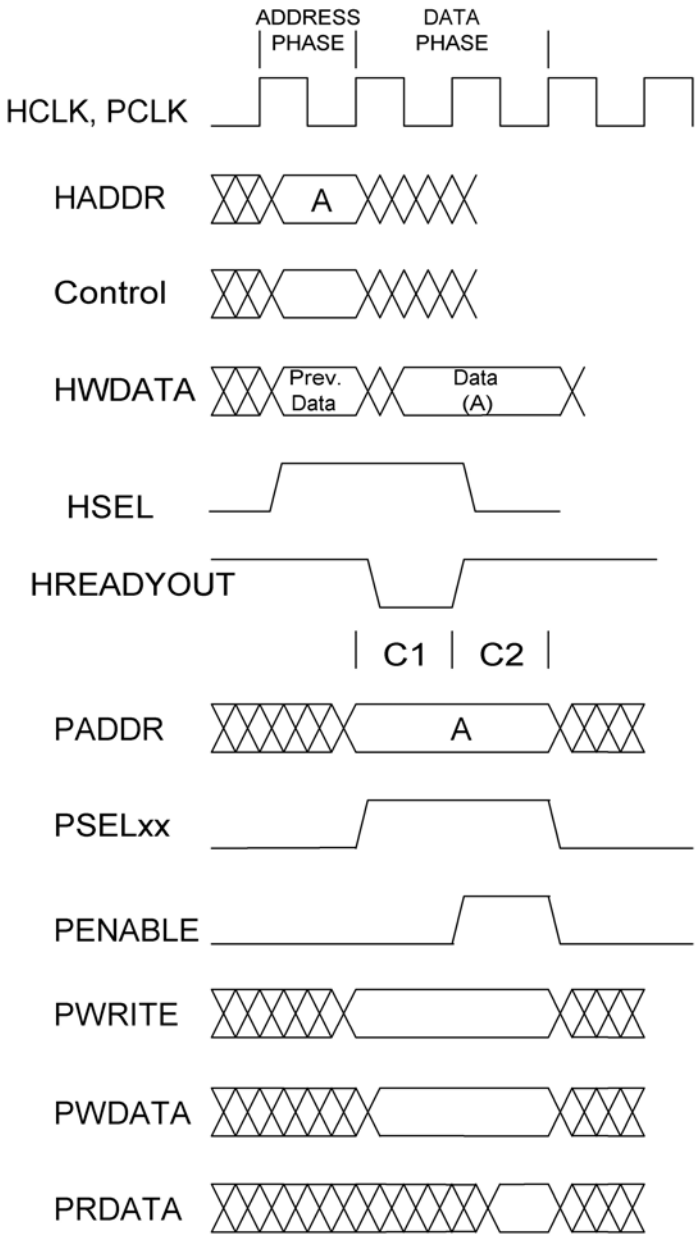
The APB Bridge implements the following functions:

- Adaptation of APB bus signals to AHB bus signals
- APB address decoding
- APB read data bus multiplexing

3.6.2 APB Peripheral Memory Map

Peripheral/Device	Base Address	Address Range	Address Space (bytes)
Timer 1	0x0C000000	0x0C000000 – 0x0C00FFFF	16
Timer 2	0x0C010000	0x0C010000 – 0x0C01FFFF	16
Watchdog Timer	0x0C020000	0x0C020000 – 0x0C02FFFF	32
UART Channel A	0x0C030000	0x0C030000 – 0x0C03FFFF	32
UART Channel B	0x0C040000	0x0C040000 – 0x0C04FFFF	32
Parallel Port	0x0C050000	0x0C050000 – 0x0C05FFFF	32
General Purpose I/O	0x0C060000	0x0C060000 – 0x0C06FFFF	64
PWM1	0x0C070000	0x0C070000 – 0x0C07FFFF	16
PWM2	0x0C080000	0x0C080000 – 0x0C08FFFF	16

3.6.3 Timing



The APB Bridge implements the timing of the APB Bus. APB peripherals are not allowed to request wait states. The APB Bridge asserts a single wait state (HREADYOUT=0) back to the AHB bus in the 1st cycle of the AHB Data Phase. This is APB cycle C1. Data is sampled at the end of the next clock cycle, which is APB cycle C2.

4 APB Peripherals

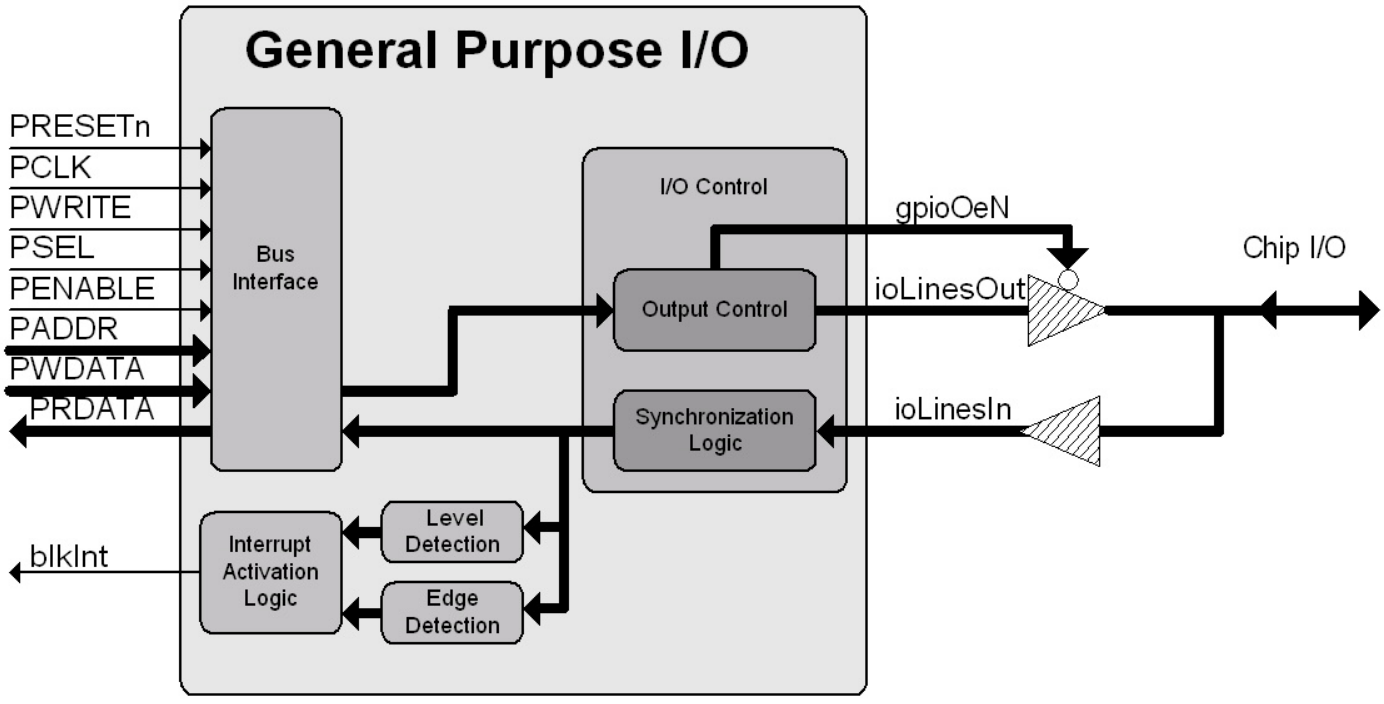
4.1 General Purpose Input/Output (GPIO)

4.1.1 Overview

The GPIO is a configurable module allowing the use of up to 32 scalable I/O lines. If more than 32 I/Os are required, more than one GPIO module may be instantiated.

Each line can be configured independently resulting in a very useful I/O application. The GPIO module supports a wide variety of options concerning synchronization logic and interrupt signal, which can be triggered by either a low level, high level, positive or negative edge. The GPIO is also fully compatible with a standard µP bus interface.

4.1.2 Block Diagram



4.1.3 Signal Descriptions

Signal Name	I/O	Description
PRESETn	I	Active low APB reset
PCLK	I	APB clock
PWRITE	I	APB Write (0=read 1=write)
PWDATA[31:0]	I	Write Data from the APB (32 bits)
PADDR[3:0]	I	Lower bits of APB Address bus (4 bits)
PSEL	I	APB Block select, enables the GPIO
PENABLE	I	APB signal indicating 2nd cycle of APB transfer
PRDATA[31:0]	O	Read data to the APB (32 bits)
blockInt	O	Block Interrupt
ioLinesOut[NUM_LINES-1:0]	O	Lines used if IO port is outbound (32 bits)
ioLinesIn[NUM_LINES-1:0]	I	Lines used if IO port is inbound (32 bits)
gpioOeN[NUM_LINES-1:0]	O	Direction of all I/O lines (1 = input, 0 = output) (32 bits)

4.1.4 Programming Interface

4.1.4.1 Register Summary

Register Name	Offset	Description
State	0x00	Read to see current state of I/O signals
Set	0x00	Write 1's to set output signals
Clear	0x04	Write 1's to clear output signals
Interrupts	0x04	Read to see current state of interrupts
Out	0x08	write 1's to set direction to output
In	0x0c	write 1's to set direction to input
Enable Set	0x10	write 1's to set enable interrupt
Enable Clear	0x14	write 1's to set disable interrupt
Edge	0x18	write 1's to set interrupt to edge-sensitive
Level	0x1c	write 1's to set interrupt to level-sensitive
Set Level	0x20	write 1's to set interrupts to active high or rising edge
Clear Level	0x24	write 1's to set interrupts to active low or falling edge
Any Edge Set	0x28	write 1's to override interrupt edge selection & interrupt on any edge
Any Edge Clear	0x2c	write 1's to clear edge selection override
Interrupt Clear	0x30	write 1's to clear edge-sensitive interrupts

4.1.4.2 Register Descriptions

4.1.4.2.1 State Register

Offset: 0x0

Bits	Access	Default	Description
[31:0]	Read	0x00000000	Read to see current state of I/O signals. For each I/O, the synchronized input signal is read, regardless of whether I/O is configured as input or output.

4.1.4.2.2 Set Register

Offset: 0x0

Bits	Access	Default	Description
[31:0]	Write	0x00000000	Each bit that is set to 1 sets the corresponding output signal.

4.1.4.2.3 Clear Register

Offset: 0x4

Bits	Access	Default	Description
[31:0]	Write	0x00000000	Each bit that is set to 1 clears the corresponding output signal.

4.1.4.2.4 Interrupts Register

Offset: 0x4

Bits	Access	Default	Description
[31:0]	Read	0x00000000	This register is read to see the current state of interrupts.

4.1.4.2.5 Out Register

Offset: 0x8

Bits	Access	Default	Description
[31:0]	R/W	0xFFFFFFFF	Each bit that is set to 1 configures the corresponding signal as an output. Read for current state of all active low output enable signals. All signals default to inputs.

4.1.4.2.6 In Register

Offset: 0xC

Bits	Access	Default	Description
[31:0]	R/W	0xFFFFFFFF	Each bit that is set to 1 configures the corresponding signal as an input. Read for current state of all active low output enable signals. All signals default to inputs.

4.1.4.2.7 Enable Set Register

Offset: 0x10

Bits	Access	Default	Description
[31:0]	Write	0x00000000	Each bit that is set to 1 enables an interrupt for the corresponding signal.

4.1.4.2.8 Enable Clear Register

Offset: 0x14

Bits	Access	Default	Description
[31:0]	Write	0x00000000	Each bit that is set to 1 disables an interrupt for the corresponding signal.

4.1.4.2.9 Edge Register

Offset: 0x18

Bits	Access	Default	Description
[31:0]	Write	0x00000000	Each bit that is set to 1 indicates that the interrupt has been set to edge-sensitive.

4.1.4.2.10 Level Register

Offset: 0x1C

Bits	Access	Default	Description
[31:0]	Write	0x00000000	Each bit that is set to 1 indicates that the interrupt has been set to level-sensitive.

4.1.4.2.11 Set Level Register

Offset: 0x20

Bits	Access	Default	Description
[31:0]	Write	0x00000000	write 1's to set interrupts to active high or rising edge

4.1.4.2.12 Clear Level Register

Offset: 0x24

Bits	Access	Default	Description
[31:0]	Write	0x00000000	write 1's to set interrupts to active low or falling edge

4.1.4.2.13

4.1.4.2.14 Any Edge Set Register

Offset: 0x28

Bits	Access	Default	Description
[31:0]	Write	0x00000000	write 1's to override interrupt edge selection & instead interrupt on ANY edge

4.1.4.2.15 Any Edge Clear Register

Offset: 0x2C

Bits	Access	Default	Description
[31:0]	Write	0x00000000	write 1's to clear edge sensitive override

4.1.4.2.16

4.1.4.2.17 Interrupt Clear Register

Offset: 0x30

Bits	Access	Default	Description
[31:0]	Write	0x00000000	write 1's to clear edge sensitive interrupts

4.1.5 Functional Description

Each GPIO line can be independently programmed as an input or an output. Separate set and clear registers are provided since it is likely that different software tasks may be servicing different I/O signals. The separate set and clear registers make read-modify-write operations with interrupts disabled unnecessary.

It is also possible to use a single I/O as a dedicated output AND a separate dedicated input. For instance, ioLinesOut[3] can drive an LED while ioLinesIn[3] reads a switch. In this case, gpioOenN[3] would be unused.

Inputs are synchronized to the system clock through a pair of flip-flops. Each input can be programmed to cause an interrupt to be generated. The interrupt can be programmed to be a level or edge-sensitive and the level or edge that causes the interrupt can be selected. Interrupts can be individually enabled or disabled. Level-sensitive interrupts stay asserted until the interrupting condition is cleared. Edge-triggered interrupts are cleared by writing to the GPIO interrupt clear register.

4.2 Universal Asynchronous Receiver/Transmitters (UARTs)

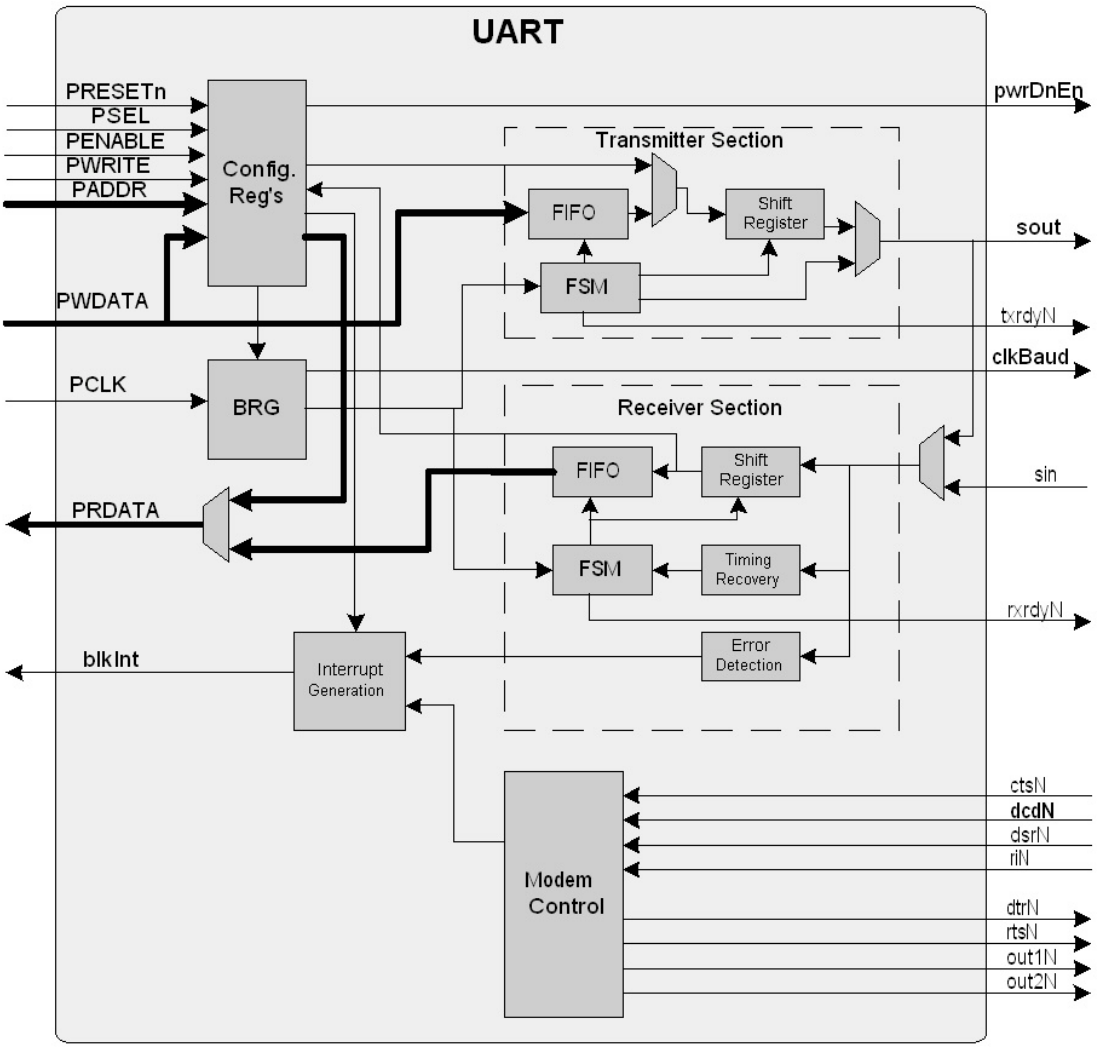
4.2.1 Overview

This is a complete implementation of a 16550 UART. The UART contains the following main sections:

- Configuration Registers
- Baud Rate Generator
- Transmitter
- Receiver
- Interrupt Generation Logic
- Modem Control Logic

Configuration registers are written and read by the processor. The baud rate generator produces timing strobes at the baud rate (for the transmitter) and at 16 times the selected baud rate (for the receiver). The receiver examines the incoming data and uses the first edge of the start bit to determine the bit timing. The transmit and receive paths can be configured to use a single register for data or to use FIFOs. Finite State Machines (FSMs) control the transmit and receive sections. Various error conditions can cause an interrupt to be generated.

4.2.2 Block Diagram



4.2.3 Signal Descriptions

Signal Name	I/O	Description
PRESETn	I	APB reset (active low)
PCLK	I	APB clock
PWRITE	I	APB write, 0=Read and 1=Write.
PWDATA[31:0]	I	APB write data bus
PADDR[2:0]	I	APB address bus, if the registers are expected to be on word boundaries, then PADDR[4:2] are connected to this input, otherwise the registers are located on byte boundaries and PADDR[2:0] is connected to this input.
PSEL	I	APB Block select; this signal must be set to access any of the UART's registers.
PENABLE	I	APB signal indicating 2nd cycle of APB transfer
PRDATA[31:0]	O	APB read data bus
blkInt	O	Block interrupt, this signal is set when any of the following interrupt sources become active: receiver line status, received data available, character timeout, transmitter holding register empty, or modem status.
baudClkEn	I	Baud clock enable: series of 1-clock wide pulses for dividing down PCLK to get 16X baud clock
ctsN	I	Clear to send
dcdN	I	Data carrier detect
dsrN	I	Data set ready
riN	I	Ring indicator
dtrN	O	Data terminal ready
rtsN	O	Request to send
rxrdyN	O	Receiver ready
txrdyN	O	Transmitter ready
sin	I	Serial data in
sout	O	Serial data out
clkBaud	O	Baud clock, can be used by external receivers as a sampling clock, the frequency is (baud rate x 16).
out1N	O	General purpose output controlled by MCR[2].
out2N	O	General purpose output controlled by MCR[3].
pwrDnEn	O	Enable powerdown mode when this bit is set.
bsMCR3isIntEn	I	Bootstrap input, when this input is tied high, MCR[3] enables interrupts.
bsMSR2isDelta	I	Bootstrap signal, when this input is tied low, MSR[2] is set on the trailing edge (low to high transition) of the active low ring indicator input (aka "TERI" or Trailing Edge Ring Indicator mode). When this input is tied high, MSR[2] is set on any edge of the ring indicator input (aka "DRI" or Delta Ring Indicator mode).

4.2.4 Programming Interface

4.2.4.1 Register Summary

Register Name	Abbreviation	Offset	Description
Receiver Buffer	RBR	0x0	Received Data (Read Only)
Transmitter Holding	THR	0x0	Data to be transmitted (Write Only)
Interrupt Enable	IER	0x4	Enables the five types of UART interrupts.
Interrupt Identification	IIR	0x8	Four interrupts levels identified here in order of priority – Receiver Line Status, Received Data Ready, Transmitter Holding Register Empty, and MODEM Status (Read Only)
FIFO Control	FCR	0x8	Enable FIFOs, clear FIFOs, set RCVR FIFO trigger level (Write Only)
Line Control	LCR	0xC	Specifies asynchronous data communications exchange format and sets the Divisor Latch Access Bit
MODEM Control	MCR	0x10	Controls interface with MODEM (or peripheral device emulating a MODEM)
Line Status	LSR	0x14	Data Transfer Status information
MODEM Status	MSR	0x18	Current state of control lines from MODEM (or peripheral device) to CPU
Scratch	SCR	0x1C	Register can be used to hold temporary data
Divisor Latch (LS)	DLL	0x0	DLAB = 1. Least significant byte for input to baud rate generator
Divisor Latch (MS)	DLM	0x4	DLAB = 1. Most significant byte for input to baud rate generator

4.2.4.2 Register Descriptions

4.2.4.2.1 Receiver Buffer Register

offset: 0x0

Bits	Access	Default	Description
[7:0]	Read	0x00	Received Data

4.2.4.2.2 Transmitter Holding Register

offset: 0x0

Bits	Access	Default	Description
[7:0]	Write	0x00	Data To be transmitted

4.2.4.2.3 Interrupt Enable Register

offset: 0x4

Bits	Access	Default	Description
[7:4]	Read/Write	0x0	Unused
[3]	Read/Write	0x0	Enable MODEM Status Interrupt
[2]	Read/Write	0x0	Enable Receiver Line Status Interrupt
[1]	Read/Write	0x0	Enable Transmitter Holding Register Interrupt
[0]	Read/Write	0x0	Enable Received Data Available Interrupt

4.2.4.2.4 Interrupt Identification Register

offset: 0x8

Bits	Access	Default	Description
[7:6]	Read	0x0	FIFOs Enabled
[5:4]	Read	0x0	Unused – value is set to 0
[3:1]	Read	0x0	Interrupt Identification Bits – see "Interrupt Identification" section
[0]	Read	0x1	Interrupt Pending – '0' if interrupt pending

4.2.4.2.5 FIFO Control Register

offset: 0x8

Bits	Access	Default	Description
[7:6]	Write	0x0	Receiver Trigger Level – see "FIFO Control" section
[5:4]	Write	Undefined	Reserved
[3]	Write	0x0	DMA Mode Select
[2]	Write	0x0	Transmit FIFO reset (self-clearing)
[1]	Write	0x0	Receive FIFO reset (self-clearing)
[0]	Write	0x0	FIFO Enable

4.2.4.2.6 Line Control Register

Offset: 0xC

Bits	Access	Default	Description
[7]	Read/Write	0x0	Divisor Latch Access (DLAB)
[6]	Read/Write	0x0	Set Break
[5]	Read/Write	0x0	Stick Parity
[4]	Read/Write	0x0	Even Parity Select
[3]	Read/Write	0x0	Parity Enable
[2]	Read/Write	0x0	Number of Stop Bits
[1:0]	Read/Write	0x0	Word Length Select – see "Line Control" section

4.2.4.2.7 Modem Control Register

Offset: 0x10

Bits	Access	Default	Description
[7:5]	Read/Write	0x0	Reserved
[4]	Read/Write	0x0	Loopback mode select
[3:2]	Read/Write	0x0	Auxiliary user designated output
[1]	Read/Write	0x0	Request to Send (RTS)
[0]	Read/Write	0x0	Data Terminal Ready (DTR)

4.2.4.2.8 Line Status Register

Offset: 0x14

Bits	Access	Default	Description
[7]	Read	0x0	Error in Receiver FIFO
[6]	Read	0x0	Transmitter Empty
[5]	Read	0x0	Transmitter Holding Register Empty
[4]	Read	0x0	Break Interrupt
[3]	Read	0x0	Framing Error
[2]	Read	0x0	Parity Error
[1]	Read	0x0	Overrun Error
[0]	Read	0x0	Data Ready (DR)

4.2.4.2.9 Modem Status Register

Offset: 0x18

Bits	Access	Default	Description
[7]	Read	0x0	Data Carrier Detect (DCD)
[6]	Read	0x0	Ring Indicator (RI)
[5]	Read	0x0	Data Set Ready (DSR)
[4]	Read	0x0	Clear To Send (CTS)
[3]	Read	0x0	Delta Data Carrier Detect (DDCD)
[2]	Read	0x0	Trailing Edge Ring Indicator (TERI)
[1]	Read	0x0	Delta Data Set Ready (DDSR)
[0]	Read	0x0	Delta Clear To Send (DCTS)

4.2.4.2.10 Scratch Register

Offset: 0x1C

Bits	Access	Default	Description
[7:0]	Read/Write	0x00	Scratchpad register

4.2.4.2.11 Divisor Latch (LS)

Offset: 0x0

Bits	Access	Default	Description
[7:0]	Read/Write	0x01	Divisor Latch [7:0]

4.2.4.2.12 Divisor Latch (MS)

Offset: 0x4

Bits	Access	Default	Description
[7:0]	Read/Write	0x00	Divisor Latch [15:8]

4.2.5 Functional Description

4.2.5.1 Transmitter Operation

The UART Transmitter section is relatively simple. It consists of the transmit holding register (THR), the transmit FIFO, a shift register, a state machine, and a multiplexor. When in FIFO mode, writing to the THR causes the THR value to be written into the FIFO. If there is no transmission in progress, transmission is started. The state machine controls the shift register and mux to insure the bits get transmitted in the proper sequence. The baud rate generator controls the pacing of the transmission. Parity is calculated if required and inserted into the bit stream.

4.2.5.2 Receiver Operation

As in most systems, the receiver is a bit more complicated than the transmitter. Since another independent UART is transmitting the signal, timing has to be recovered from the incoming serial bit stream. The falling edge of the start bit starts a counter that counts into the middle of the bit time programmed into the baud rate generator. Thereafter, midBit strobes are generated which occur in the middle of the bit time and control the timing for the duration of the received word. The timing of the far end transmitter can be substantially different from that of the receiver and the word will be successfully received as long as the cumulative error across the word (including start and stop bits) does not exceed one half of the bit time.

A state machine sequences the receive operation. When all bits are received, parity is calculated and compared to the expected parity. Parity errors, framing errors, and break conditions are associated with a received data word and are written into the FIFO along with the data. A framing error is defined as the absence of an expected stop bit. A break condition is declared when too many space symbols occur in a row, that is when the serial input stays in the logic 0 state for a number of bit times greater than the sum of the number of start, stop, parity and data bits.

4.2.5.3 Baud Rate Generator

The baud rate generator can be configured to generate a wide range of baud rates, depending on the system clock frequency and the divisor latch. The divisor latch and the system clock frequency are related to the baud rate by the following expression:

$$BaudRate = \frac{SystemClockFrequency}{16 \times DivisorLatch}$$

For example, to configure the UART for a baud rate of 2400 with a system clock frequency of 18.432 MHz, the Divisor Latch would need to be decimal 480 (0x01E0), so the most significant byte of the divisor latch would be programmed 0x01 and the least significant byte of the divisor latch would be programmed 0xE0. To access the divisor latch, the Divisor Latch Access Bit (DLAB) must be set in the LCR.

The baud rate generator also generates the clkBaud output signal that can be used to provide a timing reference to external receivers. The clkBaud output frequency is 16 times the baud rate.

$$clkBaudFrequency = \frac{SystemClockFrequency}{DivisorLatch}$$

See the Timing Diagrams section for a detailed description of the clkBaud output.

The divisor latch can be programmed values ranging 0x0001 to 0xFFFF.

4.2.5.3.1 Optional Divisor for Baud Rate Generator

There is an optional divisor that can be implemented within the system to approximate a 1.8432MHz "UART" oscillator to the UART when the system clock frequency is 16.000MHz. This divisor is implemented when the USE_UART_BAUD_CLK_DIV has been defined in socSysDefs.hv. This divisor yields an extra factor of 8.5 (since 16.000/1.8432 ~= 8.5) in the BaudRate and clkBaudFrequency equations. Please use the following equations to determine the appropriate value to program the divisor latches when using this optional divisor.

$$BaudRate = \frac{SystemClockFrequency}{8.5 \times 16 \times DivisorLatch}$$

$$clkBaudFrequency = \frac{SystemClockFrequency}{8.5 \times DivisorLatch}$$

For example, to achieve 2400 baud with a 16.368MHz oscillator, a divisor latch of 0x0032 (decimal 50) would be necessary since (16368000/(8.5 * 16 * 50) = 2407, which approximates 2400).

4.2.5.4 Interrupts

The UART can generate five types of interrupts: receiver line status, received data available, character timeout, transmitter holding register empty, and modem status. Any of these interrupts can active the UART block interrupt (blkInt) if they are enabled.

4.2.5.4.1 Enabling Interrupts

The enabling of these interrupts is controlled by the setting and clearing of bits in the IER. A set bit in the IER corresponds to an enabled interrupt while a clear bit in the IER corresponds to a disabled interrupt.

Bit 3 of the MCR can be optionally used as a global interrupt enable for the UART when the bootstrap input bsMCR3isIntEn is tied to a logic '1'. When bsMCR3isIntEn is tied to a logic '1', MCR[3] acts as a global interrupt enable for the UART. If MCR[3] is clear while configured as an interrupt enable, the UART will not activate its interrupt signal (blkInt) under any condition. If MCR[3] is set while configured as an interrupt enable, the UART will activate its interrupt signal only if one of the five interrupt sources is active and enabled by the appropriate bit in the IER. When bsMCR3isIntEn is tied to a logic '0', MCR[3] has no effect on the UART interrupt logic and the UART will activate its interrupt signal any time one of the five interrupt sources is active and enabled by the appropriate bit in the IER.

IER Bit Number	Corresponding Interrupt Enabled
0	Received Data Available
0	Character Timeout (only when FIFOs are enabled)
1	Transmitter Holding Register Empty
2	Receiver Line Status
3	Modem Status

4.2.5.4.2 Interrupt Types

4.2.5.4.3 Receiver Line Status

This interrupt occurs when any one of the following conditions exist: framing error, parity error, overrun error, or break interrupt. This interrupt is cleared by reading the line status register.

4.2.5.4.4 Received Data Available

When FIFOs are disabled, this interrupt occurs when the RBR contains received data. Reading the RBR clears this interrupt.

When FIFOs are enabled, this interrupt occurs when the amount of data in receiver FIFO exceeds the trigger level (controlled by FCR[7:6]). This interrupt is cleared when the amount of data in the receiver FIFO drops below the trigger level. This may require multiple reads from the RBR.

4.2.5.4.5 Character Timeout

This interrupt only occurs when FIFOs are enabled and no characters have been written to or read from the FIFO in the last 4 character times. Each character time consists of the start bit, data bits, parity bit (if enabled), and stop bit(s). This interrupt is cleared by reading the RBR.

4.2.5.4.6 Transmitter Holding Register Empty

When FIFOs are disabled, this interrupt occurs when a character from the THR is loaded into the transmitter shift register, indicating that the THR is ready to accept a new character.

When FIFOs are enabled, this interrupt occurs when the transmitter FIFO becomes empty.

In either case, this interrupt is cleared when the THR is read, or when the IIR was read when THRE was the source of the interrupt.

Note that reading the IIR clears the THRE interrupt, however the THRE bit in the LSR will remain set until a character has been loaded into the THR or transmitter FIFO.

4.2.5.4.7 Modem Status

The modem status interrupt occurs when any of the following bits in the LSR is set: delta clear to send (DCTS), delta data set ready (DDSR), trailing edge ring indicator (TERI), or delta data carrier detect (DDCD). This interrupt is cleared by reading the MSR.

Be aware that in loopback mode these “delta” bits are not controlled by the modem inputs but rather by the lower four bits of the MCR. See the section on Loopback Operation for a more detailed description of generating modem status interrupts in loopback mode.

4.2.5.4.8 Interrupt Identification

Interrupts generated by the UART are distinguished by the value in the least significant nibble of the interrupt identification register. Bit 0 of the IIR can be tested to determine if there is an interrupt pending.

Priority	IIR [3:0]	Interrupt Type
n/a	0001	No Interrupt Pending
first	0110	Receiver Line Status
second	0100	Received Data Available
second	1100	Character Timeout
third	0010	Transmitter Holding Register Empty
fourth	0000	Modem Status

4.2.5.5 FIFO Control

FIFOs are enabled by setting bit 0 in the FCR. When FIFOs are enabled, bits 7 and 6 of the IIR are set, otherwise these bits are clear.

4.2.5.5.1 Receiver FIFO

The receiver FIFO is used to store received data until it is read by software. Up to 16 bytes can be stored in the receiver FIFO. Once the programmed trigger level has been reached within the FIFO, the data ready bit (and received data available interrupt if enabled) will be set. Bits 7 and 6 of the FCR register control the receiver trigger level.

FCR[7:6]	Receiver FIFO Trigger Level (Bytes)
00	1
01	4
10	8
11	14

The receiver FIFO can be reset by setting bit 1 in the FCR. This bit is self-clearing.

4.2.5.5.2 Transmitter FIFO

The transmitter FIFO is used to store data that is intended to be transmitted. The transmitter FIFO can store up to 16 bytes. Each byte written to the THR location is loaded into the transmitter FIFO. The transmitter FIFO can be reset by setting bit 2 in the FCR. This bit is self-clearing.

4.2.5.6 Line Control

4.2.5.6.1 Asynchronous Serial Data Format

The line control register (LCR) specifies the format of the asynchronous data that is transmitted and received by the UART. The number of data bits in each serial character is specified by bits 1 and 0 of the LCR.

LCR[1:0]	Character Length
00	5 Bits
01	6 Bits
10	7 Bits
11	8 Bits

The number of stop bits in each serial character is specified by bit 2 of the LCR along with the current character length selected by bits 1 and 0 of the LCR.

LCR[2]	Number of Stop Bits
0	1
1	2 *

* Note that this is a difference from a standard 16550 implementation in that when the character length is 5 bits, the number of stop bits is supposed to be 1.5. For simplicity, this implementation generates 2 stop bits.

Parity bit generation is controlled by bits 4 and 3 of the LCR. Bit 3 enables parity and bit 4 selects even or odd parity.

LCR[4:3]	Type of Parity
X0	No Parity
01	Odd Parity
11	Even Parity

Even parity sets the parity bit to a logic ‘1’ when there are an even number of ‘1’s in the serial character and sets the parity bit to a logic ‘0’ when there are an odd number of ‘1’s in the serial character. Odd parity sets the parity bit to a logic ‘1’ when there are an odd number of ‘1’s in the serial character and sets the parity bit to a logic ‘0’ when there are an even number of ‘1’s in the serial character.

4.2.5.6.2 Diagnostic Mechanisms

The line control register also provides mechanisms to force break conditions, as well as parity and framing error conditions. Utilizing these mechanisms while in loopback mode allow for diagnostic checking of receiver line logic.

Bit 5 of the LCR is the Stick Parity bit. Stick Parity causes the transmitter to force the parity bit of a transmitted serial character to a logic ‘1’ or logic ‘0’ regardless of the calculated parity of the transmitted serial character. Stick Parity also causes the receiver to check the parity bit of a received serial character against either a logic ‘1’ or logic ‘0’ regardless of the calculated parity of the received serial character. It is the Stick Parity bit in conjunction with bits 4 and 3 of the LCR which determine how the parity bit is affected.

The following table summarizes all possible combinations of parity functionality:

LCR[5]	LCR[4:3]	Type of Parity
0	00	parity disabled, no parity bit generated/checked
0	01	odd parity generated/checked
0	10	parity disabled, no parity bit generated/checked
0	11	even parity generated/checked

1	00	parity disabled, no parity bit generated/checked
1	01	stick parity, generated/checked as logic '1'
1	10	parity disabled, no parity bit generated/checked
1	11	stick parity, generated/checked as logic '0'

Bit 6 of the LCR is the Set Break bit. Setting this bit forces sout low (spacing) condition. The sout line will remain in the spacing condition until this bit is cleared.

Bit 7 of the LSR is the Divisor Latch Access Bit (DLAB). Please see the section on the Baud Rate Generator for a description of the use of this bit.

4.2.5.7 Line Status

Bit 0 of the LSR is the Data Ready (DR) bit. This bit is set when an incoming character is completely received into the RBR or receiver FIFO and cleared once the character from the RBR is read or once the receiver FIFO has been emptied.

Bit 1 of the LSR is the overrun error indicator. Overrun errors occur in 16450 mode when both the RBR and receiver shift register contain data, and another incoming character is received, destroying the character in the receiver shift register. The character in the RBR is not corrupted.

In 16550 mode, overrun errors occur when the receiver FIFO is completely full, the receiver shift register contains a character, and another incoming character is received, destroying the character in the receiver shift register. The data in the receiver FIFO is not corrupted.

In both modes, the overrun error indicator bit is set as soon as the start bit of the character causing the overrun is detected.

Bit 2 of the LSR is the parity error indicator. This bit is set when a received character's parity (as calculated by the receiver) does not match the value of its received parity bit. In 16550 mode, this bit is set only when the offending character is at the top of the FIFO. Reading the LSR clears this bit.

Bit 3 of the LSR is the framing error indicator. This bit is set when the receiver does not detect a valid stop bit for an incoming character, i.e. the serial input signal was low during the baud time in which a stop bit (high) was expected. In 16550 mode, this bit is set only when the offending character is at the top of the FIFO. Reading the LSR clears this bit.

Bit 4 of the LSR is the break interrupt indicator. This bit is set whenever the serial input is held in the space (logic 0) state for more time than a complete character transfer (i.e. the time it takes to transmit a start bit, the data bits, the parity bit if enabled, and any stop bits). If a break is detected in 16550 mode, only one break character (0x00) is loaded into the receiver FIFO. Reading the LSR clears this bit.

Bit 5 of the LSR is the Transmitter Holding Register Empty bit. When FIFOs are disabled, this bit is set when a character from the THR is loaded into the transmitter shift register, indicating that the THR is ready to accept a new character. It is cleared when a new character is written to the THR.

When FIFOs are enabled, this bit is set when the transmitter FIFO becomes empty and is cleared after at least one character is loaded into the transmitter FIFO.

Bit 6 of the LSR is the Transmitter Empty Bit. This bit is set when both the THR and the transmitter shift register are empty. This bit is clear when either the THR contains a character or the transmitter shift register has not completed shifting out a character being transmitted.

Bit 7 of the LSR indicates that there is at least one error in the receiver FIFO. In 16450 mode this bit is always clear. In 16550 mode, this bit is set when there is at least one framing error, parity error, or break indication in the receiver FIFO. Reading the LSR clears this bit.

4.2.5.8 Modem Control / Status

The Modem Control and Modem Status Registers (MCR and MSR) are used to control and monitor the modem related I/O: Clear to Send, Data Set Ready, Ring Indicator, Data Carrier Detect, Data Terminal Ready, and Request to Send.

4.2.5.8.1 Modem Control

Bits 0 and 1 of the MCR directly affect the state of the Data Terminal Ready and Request to Send outputs (dtrN and rtsN). Setting bit 0 of the MCR causes the dtrN output to be asserted (low) while clearing bit 0 of the MCR causes dtrN to be deasserted (high). In a similar manner, setting bit 1 of the MCR causes the rtsN output to be asserted (low) while clearing bit 1 of the MCR causes rtsN to be deasserted (high).

4.2.5.8.2 Modem Status

Bits 4-7 of the MSR indicate the current state of the Clear to Send, Data Set Ready, Ring Indicator, and Data Carrier Detect input signals, respectively (ctsN, dsrN, riN, dcdN). On the condition that any of these input signals are active (low), then the corresponding bits in the MSR will be set. When any of these inputs are in their inactive (high) state, then the corresponding bits in the MSR will be clear.

Bits 0-3 of the MSR are known as the “delta” bits which indicate whether the state of the modem input signals have changed since a previous read of the MSR. The Delta Clear to Send (DCTS), Delta Data Set Ready (DDSR), and Delta Data Carrier Detect (DDCD) bits in the MSR are asserted any time one of the corresponding modem inputs (ctsN, dsrN, dcdN) changes state.

The “delta” bit for the Ring Indicator can be configured as a Delta Ring Indicator (DRI) or as a Trailing Edge Ring Indicator (TERI) based on the state of the bootstrap input bsMSR2isDelta. When this bootstrap is tied high (configured for DRI), bit 2 of the MSR will be asserted any time the state of the riN input changes. Alternatively, bsMSR2isDelta input can be tied low (configured for TERI), and bit 2 of the MSR will only be asserted under the condition that there is a low to high transition (active low trailing edge) of the riN input.

4.2.5.9 DMA modes

There are two modes of DMA operation. These modes affect the operation of the active low txrdyN and rxrdyN signals. In 16450 mode, only Mode 0 is supported. In 16550 mode, both Mode 0 and Mode 1 are supported and are selected by bit 3 in the FCR.

4.2.5.9.1 Mode 0 (16450 and 16550)

In this mode, rxrdyN will be asserted (low) when there is at least one character in either the RBR or the receiver FIFO and will be deasserted (high) when the RBR or the receiver FIFO are empty. The txrdyN signal will be asserted (low) when the THR or the transmitter FIFO are empty and will be deasserted (high) immediately after a character has been loaded into the THR or the transmitter FIFO.

4.2.5.9.2 Mode 1 (16550 only)

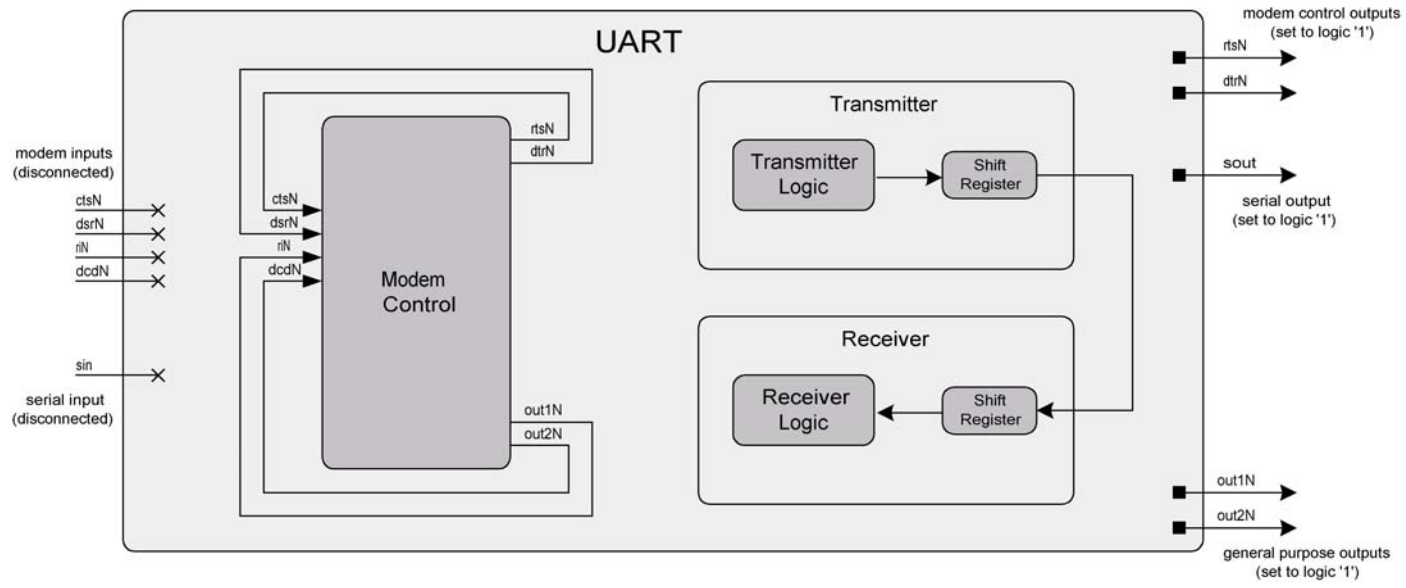
In this mode, rxrdyN will be asserted (low) when the amount of characters in the receiver FIFO reaches the receiver FIFO trigger level. A character timeout will also cause rxrdyN to be asserted (low). The rxrdyN signal will be deasserted (high) once the receiver FIFO has been emptied. The txrdyN signal will be asserted (low) when the transmitter FIFO is empty and will be deasserted (high) once the transmitter FIFO has been completely filled.

4.2.5.10 Loopback Operation

The UART provides loopback operation for transmitter, receiver, and modem status diagnostics. The UART operates in loopback mode when bit 4 of the MCR is set.

In loopback mode, the serial data input (sin) is disconnected, and the serial data output (sout) is driven high (marking state). The transmitter is internally connected to the receiver so that any data transmitted is also received within the UART.

The ctsN, dsrN, riN, and dcdN modem inputs are disconnected, and the dtrN, rtsN, out1N, and out2N outputs are driven high (inactive state). The modem control and general purpose outputs (dtrN, rtsN, out1N, out2N) are internally connected to the modem status inputs (dsrN, ctsN, riN, dcdN).



The modem status interrupt can still be generated, although in loopback mode it is controlled through the least four significant bits in the MCR.

For example, writing 0x11 to the MCR (asserting DTR bit and maintaining loopback bit) will cause the DSR bit to go active in the MSR (since DTR is connected to DSR in loopback mode). The “delta” edge detection logic then detects the change from low to high of the DSR bit in the MSR, therefore causing the modem status interrupt (any asserted “delta” bit in the MSR causes the modem status interrupt). Reading the MSR will clear this interrupt.

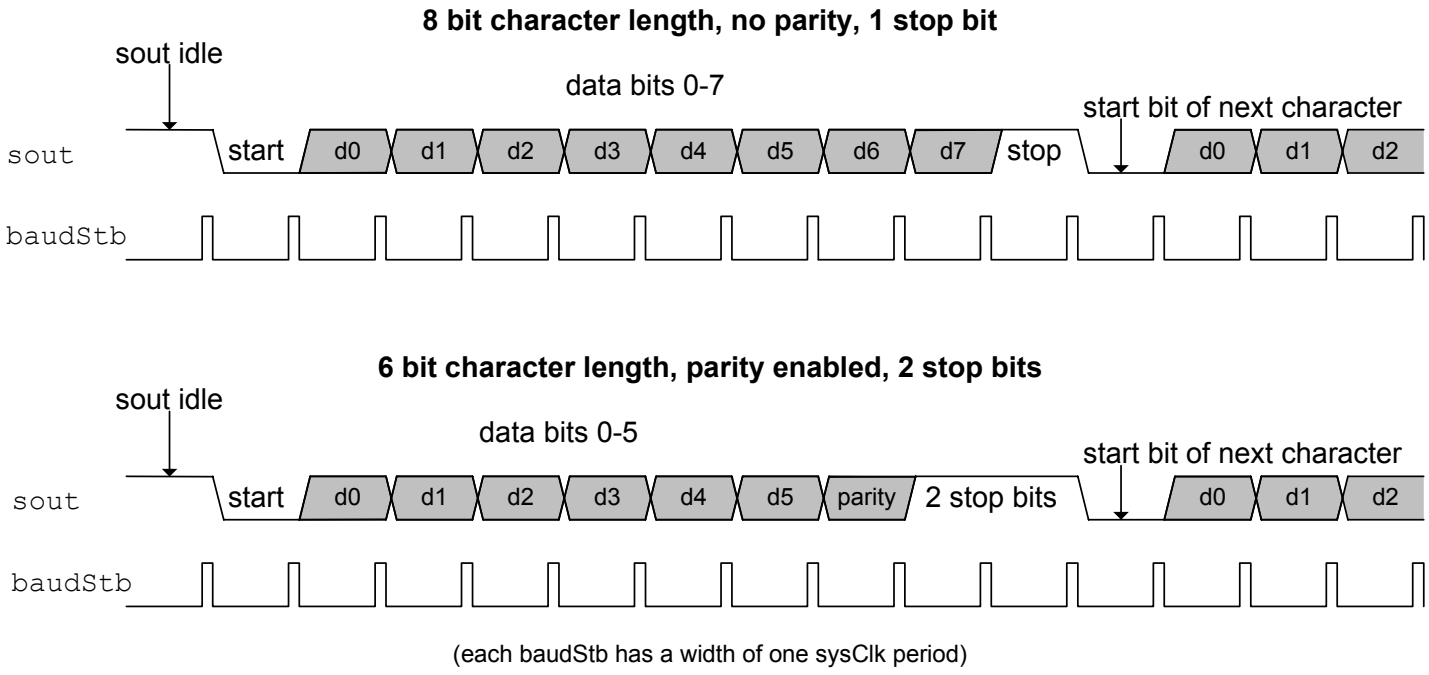
Be aware that the option to have trailing edge or delta edge detection on the ring indicator also applies in loopback mode. However, in this mode, the “delta” or trailing edge will be detected on the out1 bit in the MCR since this is connected to the Ring Indicator bit in the MSR when operating in loopback mode. This configuration allows diagnostic testing of the Trailing Edge Ring Indicator (TERI) and Delta Ring Indicator (DRI) in loopback mode through the use of the out1 bit of the MCR.

4.2.5.11 General Purpose Outputs

The out1N and out2N outputs are driven from bits 2 and 3 of the MCR, respectively. These outputs are active low, therefore setting bit 2 in the MCR will cause the out1N output to be driven low, while clearing bit 2 in the MCR causes the out1N output to be driven high. The out2N output is controlled in a similar fashion by bit 3 of the MCR. A system reset will set these signals to their inactive state (logic ‘1’) since bits 2 and 3 of the MCR are cleared on a system reset.

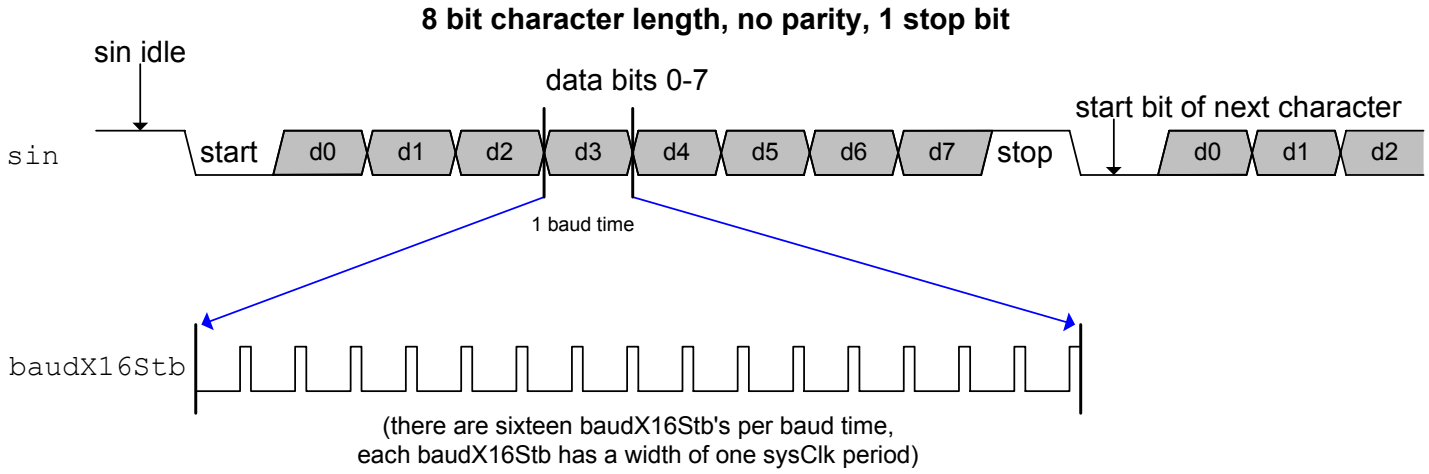
4.2.6 Timing Diagrams

4.2.6.1 Example Transmitter Timing



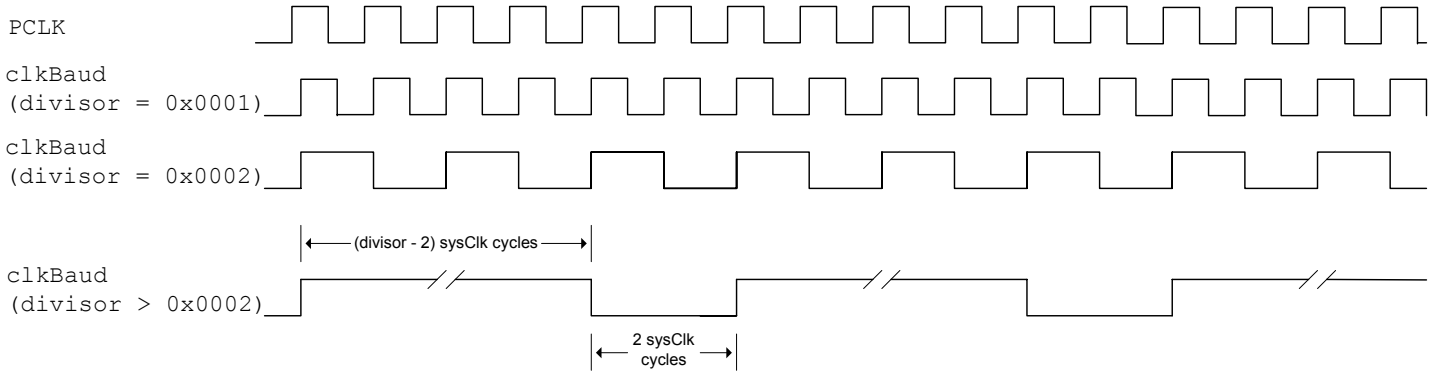
(each baudStb has a width of one sysClk period)

4.2.6.2 Example Receiver Timing



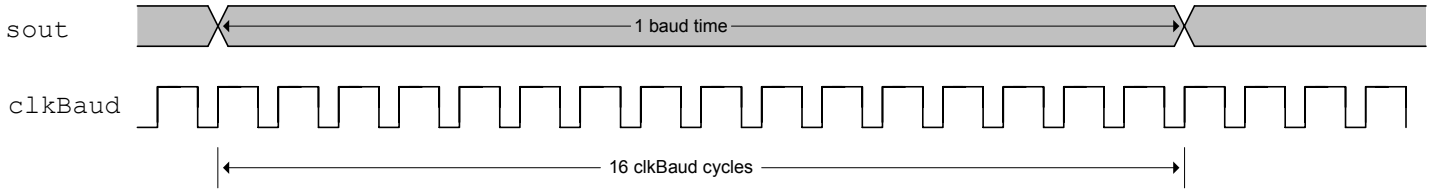
(there are sixteen baudX16Stb's per baud time,
each baudX16Stb has a width of one sysClk period)

4.2.6.3 PCLK & clkBaud



Note that the 2nd and 3rd waveforms only apply when the baudClkEn input is tied high. When the input clock is divided down using the baudClkEn signal, no special cases exist and the clkBaud signal looks like the last waveform above.

4.2.6.4 sout & clkBaud

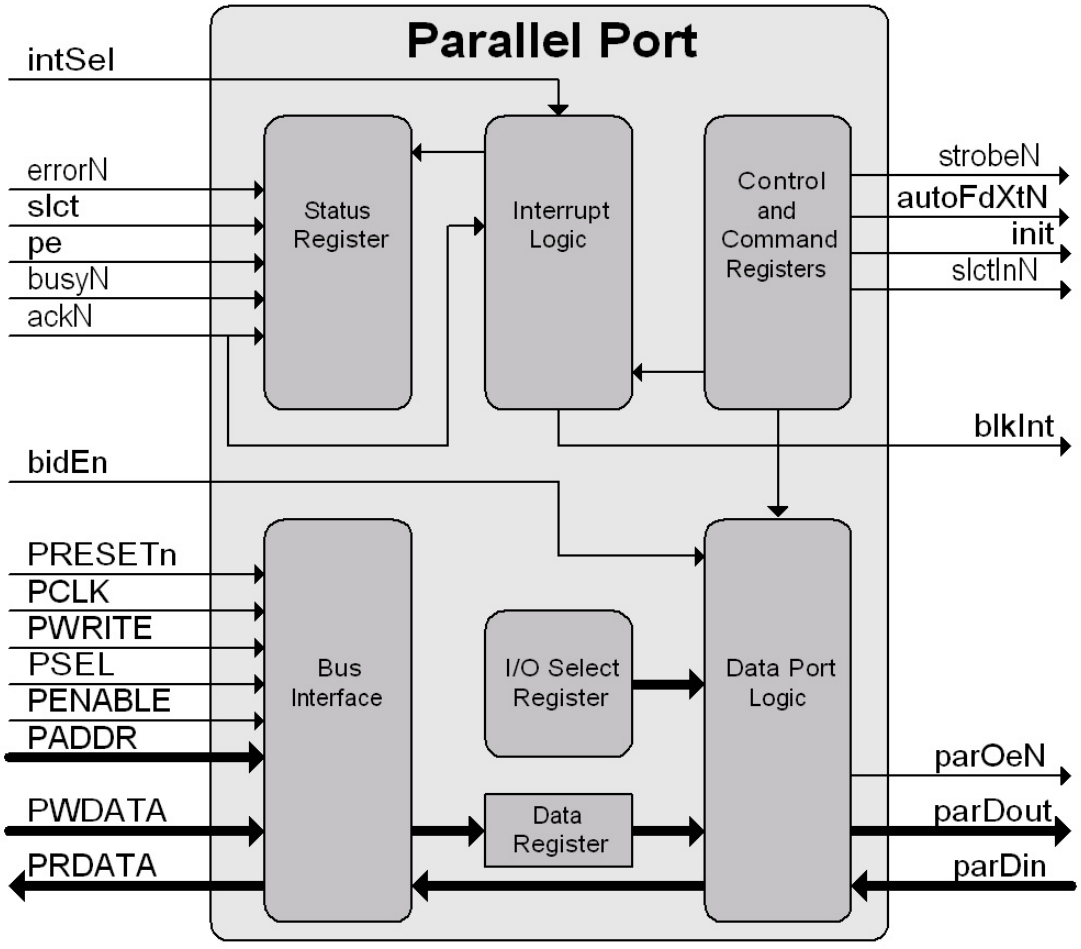


4.3 Parallel Port

4.3.1 Overview

This module is an 8-bit bi-directional parallel port compatible with PC printers. Alternatively, the parallel port can be used for general purpose I/O. The status and command registers provide information about state of the printer inputs and outputs, while the control register allows the programmer to control the printer outputs. There are two bi-directional modes to control I/O for the parallel port. One mode allows the I/O select register to control the parallel port direction while the other mode allows for a direction bit in the control register to control the direction. The parallel port also provides two interrupt modes, known as “ack” and “latch” modes.

4.3.2 Block Diagram



4.3.3 Signal Descriptions

Signal Name	I/O	Description
PRESETn	I	Active low APB reset
PCLK	I	APB clock
PADDR[1:0]	I	APB Address
PWDATA[31:0]	I	APB write data bus
PWRITE	I	APB Write signal (0=Read and 1=Write)
PSEL	I	APB Block select
PENABLE	I	APB signal indicating 2nd cycle of APB transfer
PRDATA[31:0]	O	APB read data bus
blkInt	O	Block Interrupt (1=interrupt active)
errorN	I	Printer error
slct	I	Printer select
pe	I	Printer paper empty
ackN	I	Printer acknowledge of successful data transfer to print buffer
busyN	I	Printer busy, indicates printer not ready to accept data
strobeN	O	Strobe, indicates parallel data output valid
autoFdXtN	O	Auto line feed
init	O	Initialize printer
slctInN	O	Printer select
bidEn	I	Bi-directional enable
intSel	I	Interrupt mode select (1=latched mode, 0=ack mode)
parDin[7:0]	I	Parallel data input bus
parDout[7:0]	O	Parallel data output bus
parOeN	O	Parallel data output enable

4.3.4 Programming Interface

4.3.4.1 Register Summary

Register Name	Offset	Description
data	0x0	Access to the parallel data bus
status	0x4	Status of interrupt and printer inputs
ioSel	0x4	Controls direction of port when bidEn input is low
command	0x8	Status or interrupt enable and printer outputs
control	0x8	Enables interrupt output and drives printer outputs

4.3.4.2 Register Descriptions

4.3.4.2.1 data

offset: 0x0

Bits	Access	Default	Description
[7:0]	Read/Write	0x00	Writes to this location store data in the parallel data register, which drives the parallel data bus when in output mode. Reads to this location with the parallel port in output mode reads the parallel data register, while reads to this location with the parallel port in input mode are directly reading the parallel data bus.

4.3.4.2.2 status

offset: 0x4

Bits	Access	Default	Description
[7]	Read	undefined	State of busyN input, 1=busyN is low, 0=busyN is high
[6]	Read	undefined	State of ackN input, 1=ackN is high, 0=ackN is low
[5]	Read	undefined	State of pe input, 1=pe is high, 0=ackN is low
[4]	Read	undefined	State of slct input, 1=slct is high, 0=slct is low
[3]	Read	undefined	State of errorN input, 1=errorN is high, 0=errorN is low
[2]	Read	0x1	Indicates that an interrupt is pending, 1=no interrupt pending, 0=interrupt pending
[1:0]	Read	0x3	Unused, these bits are permanently set

4.3.4.2.3 ioSel

offset: 0x4

Bits	Access	Default	Description
[7:0]	Write	0x00	Used to set direction mode of parallel port when bidEn input is low. When bidEn is low, 0xAA=input mode, 0x55=output mode. This register has no effect when bidEn is high.

4.3.4.2.4 command

offset: 0x8

Bits	Access	Default	Description
[7:5]	Read	0x7	Unused, these bits are permanently set
[4]	Read	0x0	State of block interrupt enable, 1=blkInt is enabled, 0=blkInt disabled
[3]	Read	0x0	State of slctInN output, 1=slctInN output is low, 0=slctInN output is high
[2]	Read	0x0	State of init output, 1=init output is high, 0=init output is low
[1]	Read	0x0	State of autoFdXtN output, 1=autoFdXtN is low, 0=autoFdXtN is high
[0]	Read	0x0	State of strobeN output, 1=strobeN is low, 0=strobeN is high

4.3.4.2.5 control

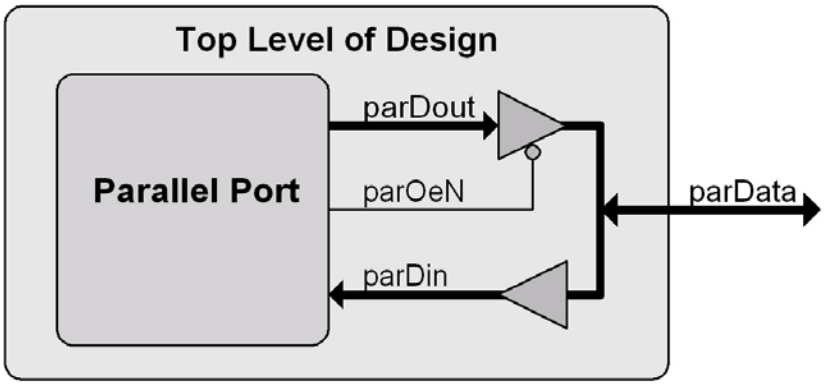
offset: 0x8

Bits	Access	Default	Description
[7:6]	Write	undefined	Unused, these bits will be ignored when written to
[5]	Write	0x0	Direction bit, used when bidEn is high. When bidEn is high, 1=input mode, 0=output mode. This bit has no effect when bidEn is low
[4]	Write	0x0	Block interrupt enable, enables the blkInt output. Note that the interrupt pending bit in the status register is not affected by this bit. 1=blkInt enabled, 0=blkInt disabled
[3]	Write	0x0	Drives slctInN output, 1=slctInN output is low, 0=slctInN is high
[2]	Write	0x0	Drives init output, 1=init output is high, 0=init output is low
[1]	Write	0x0	Drives autoFdXtN output, 1=autoFdXtN output is low, 0=autoFdXtN output is high
[0]	Write	0x0	Drives strobeN output, 1=strobeN output is low, 0=strobeN output is high

4.3.5 Functional Description

4.3.5.1 Bi-directional Data Bus

The bi-directional data bus is not implemented within the parallel port. The parallel port offers two unidirectional buses with an output enable used to implement a bi-directional bus at a higher level of the design hierarchy. This is done in order to facilitate integration with many FPGA and ASIC architectures where tristate buffers must exist at the I/O level. Note that this logic can easily be modified for custom applications. The following figure illustrates the bi-directional bus implementation in the Embedded Controller with AMBA interface.



4.3.5.2 I/O Modes

The parallel port is bi-directional, and its direction can be controlled by two methods. The first method of controlling the direction is by bit 5 of the control register, known as the direction bit. In order to control the direction of the parallel port with the direction bit, the bidEn input must be high. When bidEn is high, writing a logic '0' to bit 5 of the control register sets the parallel port for output mode while writing a logic '1' to bit 5 of the control register sets the parallel port for input mode.

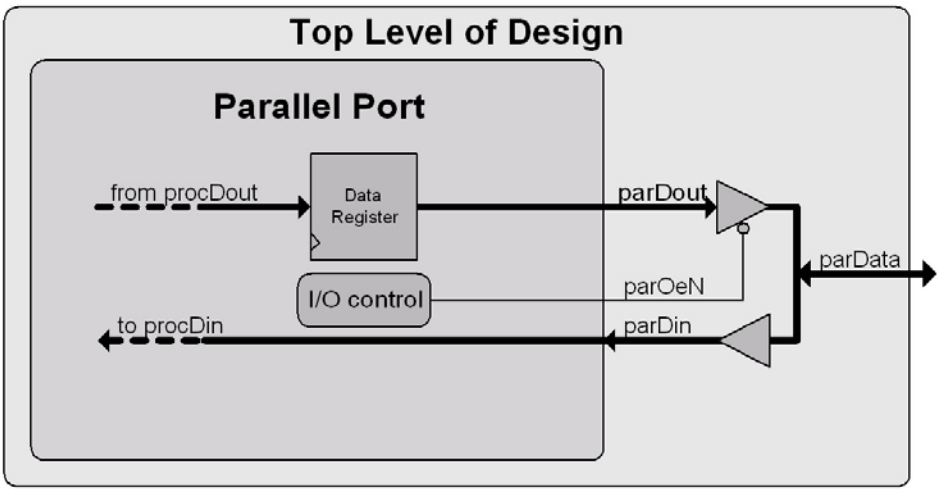
The second method of controlling the direction is by the I/O Select register (ioSel). In order to control the direction of the parallel port with the ioSel register, the bidEn input must be low. When bidEn is low, writing 0x55 to the ioSel register sets the parallel port to output mode while writing 0xAA to the ioSel register sets the parallel port to input mode.

Whether the direction is controlled by the direction bit or by the ioSel register, the parallel port comes out of reset in output mode. In the Embedded Controller with AMBA interface, the bidEn input is bootstrapped high to allow the direction to be controlled by the direction bit in the control register. The following table summarizes conditions for the I/O modes.

bidEn	ioSel	control[5]	direction
1	X	0	output
1	X	1	input
0	0x55	X	output
0	0xAA	X	input

4.3.5.2.1 Output Mode

In output mode, the bi-directional parallel data bus (parData) is driven by the data register of the parallel port. This occurs because the output enable signal (parOeN) is asserted low during output mode operation, causing the parallel data output (parDout) bus to be driven onto parData. In this mode, reads to the data register location will read the data stored in the data register, which is also the data being driven onto the bi-directional data bus, which is the same data being driven onto parDin. See the following illustration.



4.3.5.2.2 Input Mode

In input mode, the output enable signal (parOeN) is high, causing the parData bus to be tristated so that an external source can drive it. The data register can still be written in this mode, however the data register will not drive parData as long as the parallel port is in input mode. Reads to the data register location will read the data being driven onto parData externally.

Be aware that no synchronization of the parallel data input bus (parDin) is performed within the parallel port. It is assumed that any external sources driving parData are synchronized to the system clock.

4.3.5.3 Interrupt Modes

Two interrupt modes are supported, known as the “ack” and “latch” modes. These modes are controlled by the intSel input of the parallel port. When intSel is high, the parallel port operates in latch mode, and when intSel is low, the parallel port operates in ack mode. In both modes the interrupt is enabled by bit 4 of the control register. Be aware that the interrupt enable applies only to the blkInt output and does not affect bit 2 of the status register. Bit 2 of the status register indicates a pending interrupt condition regardless of the interrupt enable bit.

4.3.5.3.1 Latch Mode

In this mode, the interrupt is asserted on the falling edge of ackN input and will remain set until the status register is read.

4.3.5.3.2 Ack Mode

In this mode, the interrupt tracks the ackN input. That is, any time ackN is low, the interrupt will be asserted, and any time ackN is high, the interrupt will be clear. Reads of the status register have no effect on the interrupt in this mode. There is no way to clear the interrupt from within the parallel port when operating in this mode, the only way to clear the interrupt is to remove the condition causing the interrupt. When the parallel port is operating in this mode, an interrupt service routine must perform all service necessary to raise the ackN input high to clear the interrupt, otherwise the parallel port interrupt will never be cleared.

4.3.5.4 Printer I/O

The parallel port provides various printer I/O signals which can be monitored and controlled through the status, command, and control registers. Printer input signals (errorN, slct, pe, ackN, busyN) can be monitored through the status register, printer outputs (strobeN, autoFdXtN, init, slctInN) can be monitored through the command register and controlled through the control register.

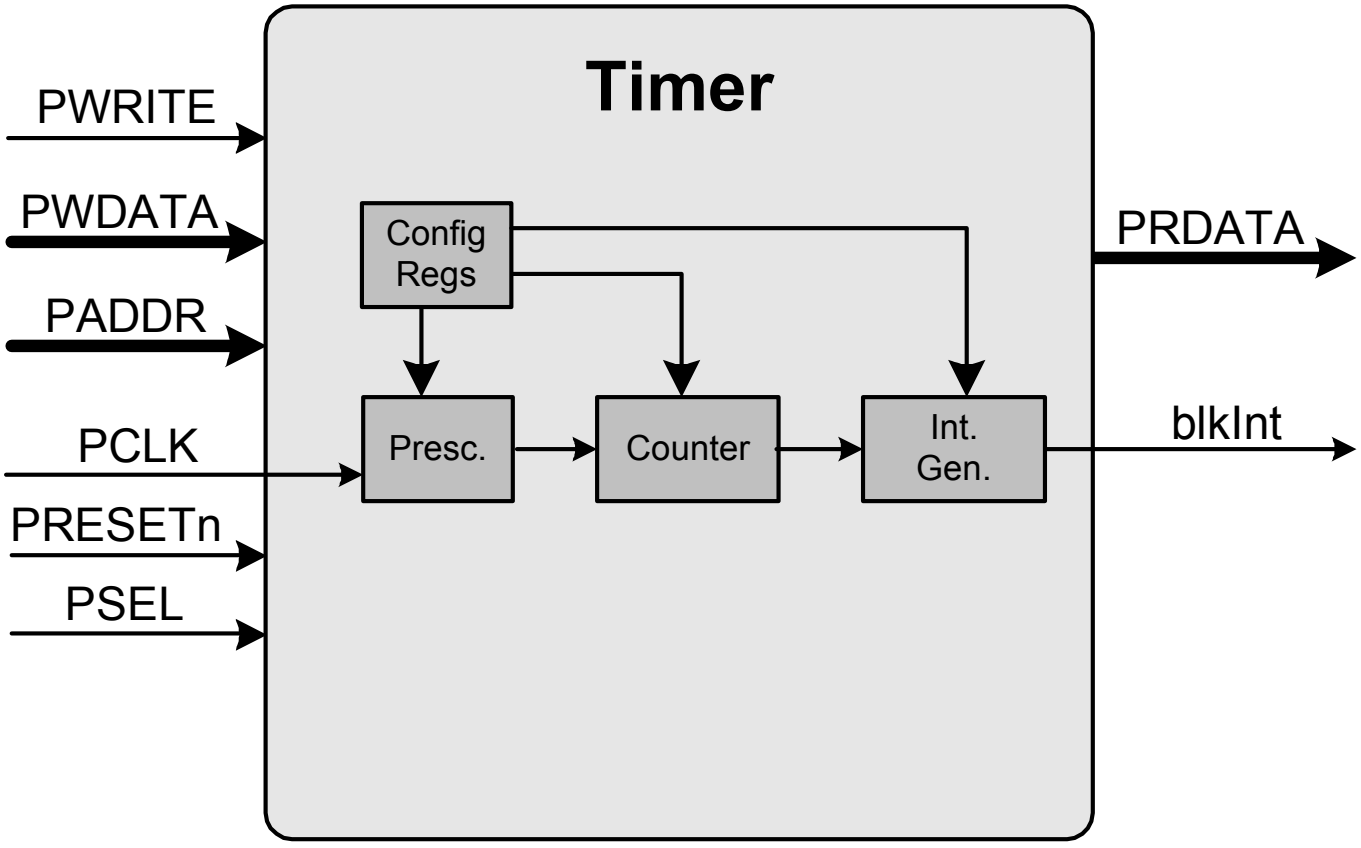
Be aware that no synchronization of printer inputs is performed within the parallel port. It is assumed that these signals are synchronized with the system clock prior to connection with the parallel port.

4.4 Timer

4.4.1 Overview

The Timer module is a sixteen bit down counter with a selectable prescaler. The prescaler can be programmed for different divide ratios and extends the timer's range at the expense of precision. The Timer provides two modes of operation that provide a free running value and also periodic interrupts.

4.4.2 Block Diagram



The Timer contains several configuration registers that can be written and read by the processor. A programmable prescaler precedes a 16-bit counter. The counter can be clocked at either the input clock rate, or a choice of prescaled rates. The counter can be loaded with a value from a preload register. The counter can optionally generate an interrupt.

4.4.3 Signal Descriptions

Signal Name	I/O	Description
PRESETn	I	APB reset (active low)
PCLK	I	APB clock
PWRITE	I	APB write, 0=Read and 1=Write.
PWDATA[31:0]	I	APB write data bus
PADDR[1:0]	I	APB Address bus
PSEL	I	APB Block select
PENABLE	I	APB signal indicating 2nd cycle of APB transfer
blkInt	O	Block Interrupt (1=interrupt active)
PRDATA[31:0]	O	APB read data bus

4.4.4 Programming Interface

4.4.4.1 Register Summary

Register Name	Offset	Description
Load	0x0	Initial value of the timer
Value	0x4	The current value of the timer (Read Only)
Control	0x8	Provides enable/disable, mode and pre-scale configurations
Clear	0xC	Clears the interrupt (Write Only)

4.4.4.2 Register Descriptions

4.4.4.2.1 Load

offset: 0x0

Bits	Access	Default	Description
[31:16]	Read/Write	undefined	Unused – undefined on a read, should be set to zero on a write
[15:0]	Read/Write	0x0	The initial value of the timer. Also, the reload value when the timer is in periodic mode.

4.4.4.2.2 Value

offset: 0x4

Bits	Access	Default	Description
[31:16]	Read	undefined	Unused – undefined
[15:0]	Read	0x0	The current value of the timer

4.4.4.2.3 Control

offset: 0x8

Bits	Access	Default	Description
[31:8]	Read/Write	undefined	Unused – undefined on a read, should be set to zero on a write
[7]	Read/Write	0x0	Enable – 0=Timer Disabled, 1=Timer Enabled
[6]	Read/Write	0x0	Mode – 0=Free running mode, 1=Periodic mode

[5]	Read/Write	0x0	Unused - 0 on a read, should be set to zero on a write
[4:2]	Read/Write	0x0	Prescale – 0=none – clock is divided by one 1=clock is divided by 16 2=clock is divided by 256 3=clock is divided by 2 4=clock is divided by 8 5=clock is divided by 32 6=clock is divided by 128 7=clock is divided by 1024
[1:0]	Read/Write	0x0	Unused - 0 on a read, should be set to zero on a write

4.4.4.2.4 Clear

offset: 0xC

Bits	Access	Default	Description
[31:0]	Write	undefined	Clear Interrupt – the bit settings are unused and should be set to zero

4.4.5 Functional Description

The timer is a basic down counter. When the timer is enabled by setting bit 7 in the Control register or a new value is set in the Load register, the Load register is loaded into the Value register. Bits 2 thru 4 of the Control register control the scaling of sysClk creating a timer clock enable. The Value register is decremented on the leading edge of the clock when the prescale clock enable is high. When enabled, blkInt is then activated until it is cleared by writing to the Clear register.

The prescaler is a free running counter. Its output is a 1-clock-wide pulse that feeds the timer's main counter as a clock enable. The prescaler is held in the clear state when the Timer is disabled (bit 6 of the Control register is zero). It is also cleared when the Load register is loaded. Bits 2 thru 4 of the Control register determine which prescale divide ratio is selected.

The timer has two modes of operation determined by bit 6 in the Control register: free running and periodic. When the Value register reaches zero and the free running mode is selected, Value is decremented to 0xFFFF on the next enabled clock edge. When the Value register reaches zero and the periodic running mode is selected, the Load register is reloaded into the Value register on the next enabled clock edge. In either case, the Value register will continue to be decremented on each enabled clock edge until the Value register reaches zero where the whole process starts over again.

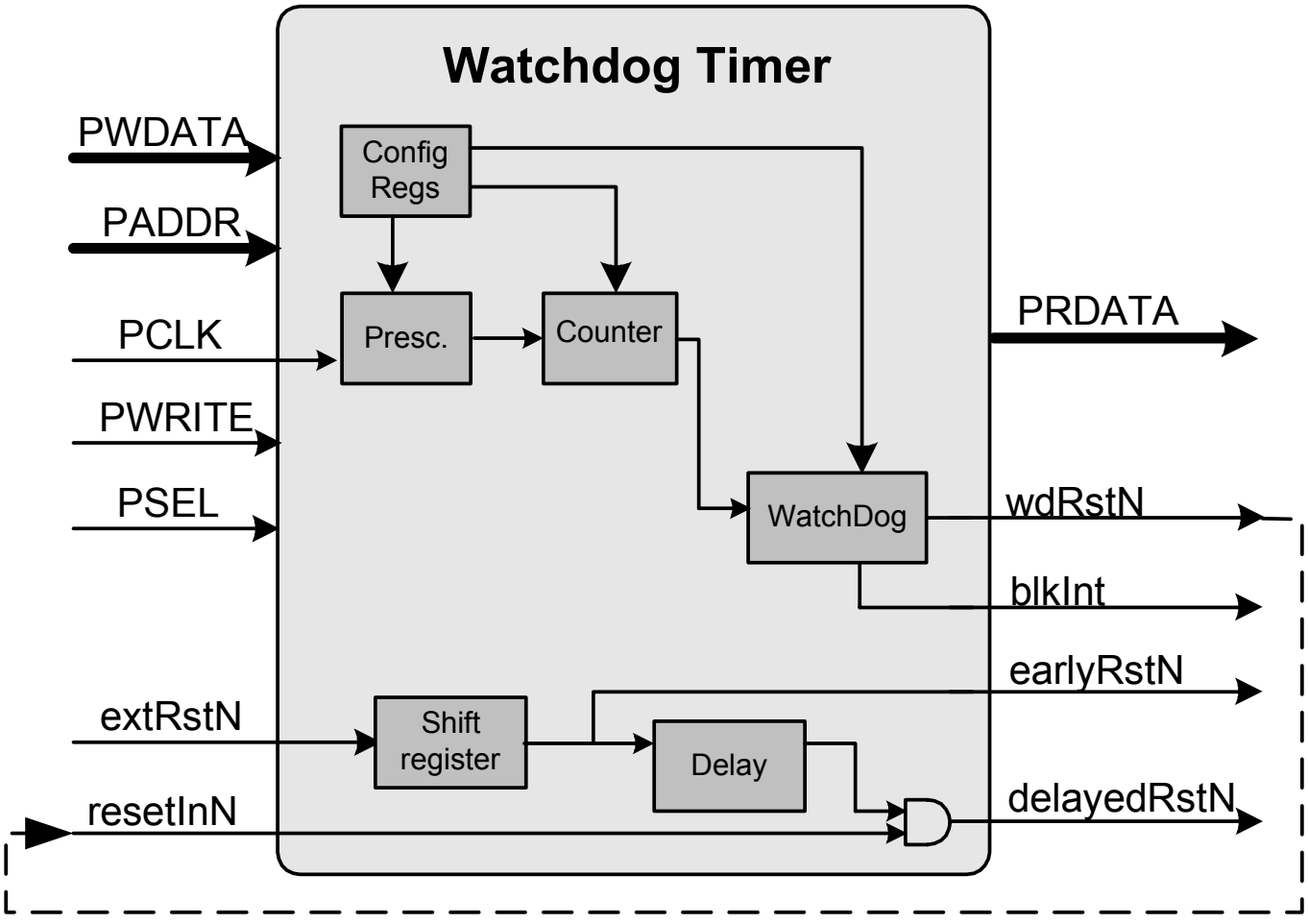
Because zero is a separate count, periodic counts are actually one count longer than the contents of the Load register. For example, if a value of four is put into the Load register and the periodic mode is selected, an interrupt will occur every five clock cycles (when prescale value is 0).

4.5 Watchdog Timer

4.5.1 Overview

The Watchdog Timer module is a 16 bit down counter with a selectable prescaler, watchdog reset, warning interrupt, and reset controller.

4.5.2 Block Diagram



The Watchdog timer contains several configuration registers that can be written and read by the processor. A programmable prescaler precedes a 16-bit counter. The counter can be clocked at either the input clock rate, or a choice of prescaled rates. Prescale values of 1, 16, 32, 128, 256, 1024, and 4096 can be selected. The prescaler extends the timer's range at the expense of precision.

The Watchdog reset can be used to reset a system after a configurable time lapse without activity. A warning interrupt can be issued a predetermined amount of time before a watchdog reset to indicate a reset is imminent. The reset controller within this module can be used to provide a glitch free version of an external reset to the internal system, and issue an early reset to components before a processor reset. The watchdog reset output can be routed to the reset input. When not used, the reset input should be tied High.

When the kick register is written, the counter is loaded with a value from a preload register. The watchdog will generate an interrupt when it decrements to the warning time (255), then a reset when it decrements to zero.

4.5.3 Signal Descriptions

Signal Name	I/O	Description
PRESETn	I	APB reset (active low)
PCLK	I	APB clock
PWRITE	I	APB write, 0=Read and 1=Write.
PWDATA[31:0]	I	APB write data bus
PADDR[2:0]	I	APB Address bus
PSEL	I	APB Block select
PENABLE	I	APB signal indicating 2nd cycle of APB transfer
extRstN	I	External Reset (0= Reset)
ResetInN	I	Reset Input
blkInt	O	Block Interrupt (1=interrupt active)
PRDATA[31:0]	O	APB read data bus
earlyRstN	O	Glitch free version of External reset (0= Reset)
delayedRstN	O	Delayed version of External reset (0= Reset)
wdRstN	O	WatchDog reset (0= Reset)

4.5.4 Programming Interface

4.5.4.1 Register Summary

Register Name	Offset	Description
Load	0x0	Initial value of the timer
Value	0x4	The current value of the watchdog timer (Read Only)
Control	0x8	Provides enable/disable, pre-scale configurations
Kick	0xC	Kicks the watchdog (Write Only)

4.5.4.2 Register Descriptions

4.5.4.2.1 Load

offset: 0x0

Bits	Access	Default	Description
[31:16]	Read/Write	undefined	Unused – undefined on a read, should be set to zero on a write
[15:0]	Read/Write	0x0	The initial value of the timer.

4.5.4.2.2 Value

offset: 0x4

Bits	Access	Default	Description
[31:16]	Read	undefined	Unused – undefined
[15:0]	Read	0xFFFF	The current value of the watchdog timer

4.5.4.2.3 Control

offset: 0x8

Bits	Access	Default	Description
[31:8]	Read/Write	undefined	Unused – undefined on a read, should be set to zero on a write
[7]	Read/Write	0x0	Enable – 0=Timer Disabled, 1=Timer Enabled
[6:5]	Read/Write	0x0	Unused - 0 on a read, should be set to zero on a write
[4:2]	Read/Write	0x0	Prescale – 0=none – clock is divided by one 1=clock is divided by 16 2=clock is divided by 256 3=clock is divided by 32 4=clock is divided by 128 5=clock is divided by 1024 6=clock is divided by 4096 7=undefined
[1:0]	Read/Write	0x0	Unused - 0 on a read, should be set to zero on a write

4.5.4.2.4 Kick

offset: 0xC

Bits	Access	Default	Description
[31:0]	Write	undefined	Kick the watchdog - reset the timer value to its default load and clear the block interrupt – the bit settings are unused and should be set to zero

4.5.5 Functional Description

The watchdog timer is a basic down counter which generates a reset when it decrements to zero. The watchdog timer is only active when enabled, and can be enabled by setting bit 7 in the Control register.

Bits 2 thru 4 of the Control register control the scaling of PCLK creating a timer clock. When all 3 bits are zero, PCLK is used directly to generate the timer clock. When bit 2 is asserted and bit 3 is unasserted, PCLK is divided by 16 to generate the timer clock. When bit 2 is asserted and bit 3 is unasserted, PCLK is divided by 256 to generate the timer clock.

The prescaler is a free running counter. Its output is a 1-clock-wide pulse that feeds the watchdog timer's main counter as a clock enable. The prescaler is held in the clear state when the Timer is disabled (bit 6

of the Control register is zero). It is also cleared when the Load register is loaded. Bits 2 thru 4 of the Control register determine which prescale divide ratio is selected.

When active, the Value register counts down from the value of the Load register. The Value register is decremented on each enabled clock until it reaches zero. The module then asserts wdRstN, which can be used to restart a system that appears to have timed out. The system should intermittently "Kick" the Watchdog, to prevent this system reset. When kicked the watchdog reloads the Value register with the contents of the Load register and resumes decrementing.

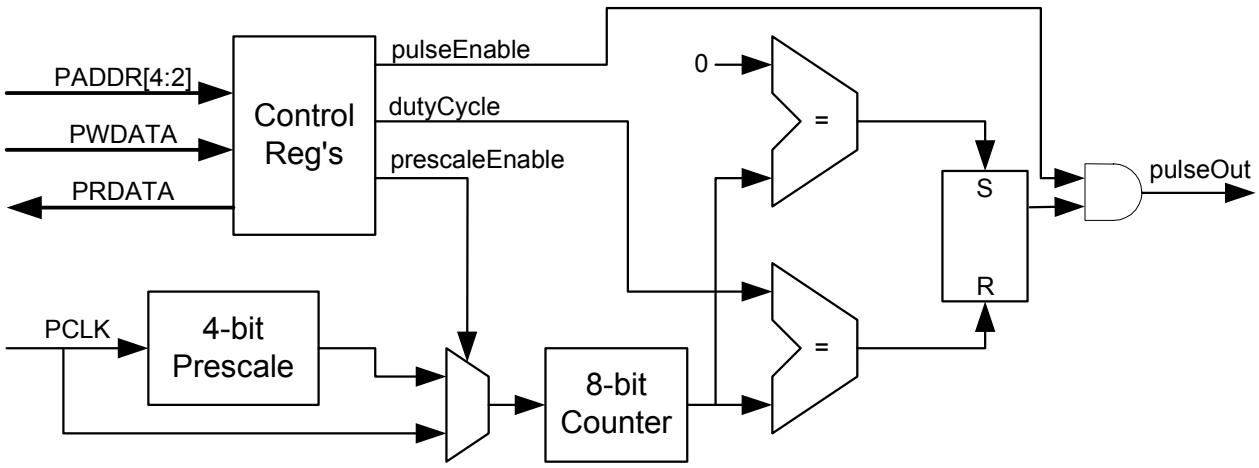
The watchdog can be kicked by a single write to the Kick address. A timer kick, a block enable, or a write to the Load register all cause the Value register to be loaded with the contents of the Load register.

The watchdog timer will issue a warning interrupt (assert blkInt) 256 timer cycles before it asserts wdRstN. This will warn the system that it hasn't kicked the watchdog recently and should do so or the system will be reset. BlkInt will remain active until the watchdog has been kicked.

The watchdog timer also provides reset controller functions. The module feeds extRstN into a 4bit linear shift register and asserts earlyRstN (logic 0) when the registers value is zero. This makes earlyRstN a glitch free version of extRstN. delayedRstN is the earlyRstN signal with its assertion delayed by 15 PCLK cycles. It can be used to reset the processor after other devices, such as flash, have been reset. The watchdog reset will stay asserted for 256 clocks and will then deassert.

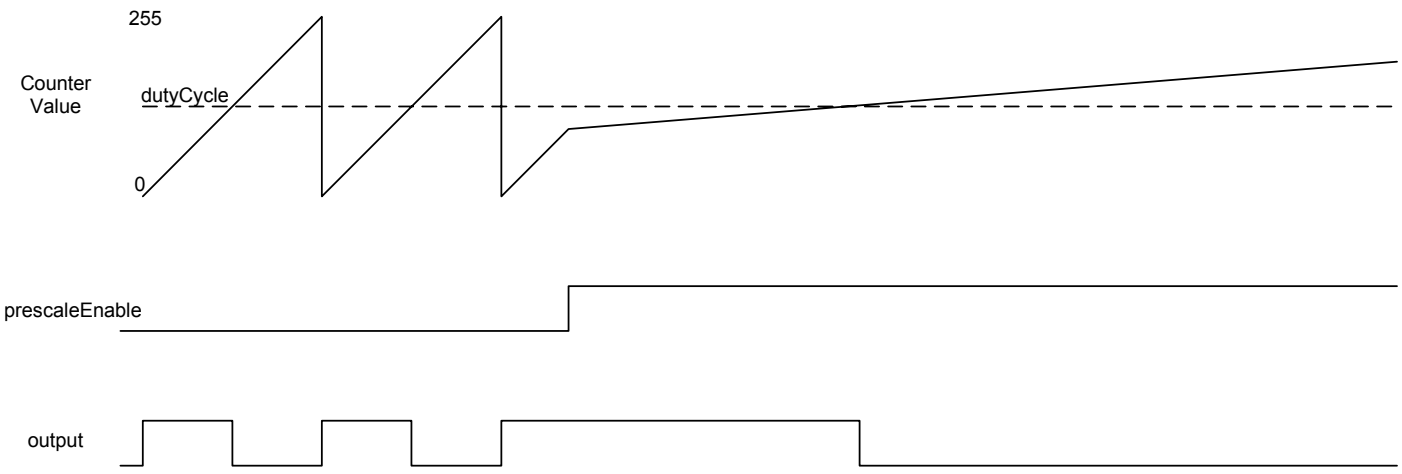
4.6 PWM

4.6.1 Block Diagram



The PWM Controller generates a programmable duty cycle output signal. The period of the output waveform is either PCLK/256 or PCLK/4096 (when prescale is enabled). The duty cycle can be programmed with 8-bit resolution. The output can be disabled or enabled. Disabling resets the counter to zero for the next enable.

4.6.2 Timing Diagram



When the 8-bit counter value reaches the value programmed into the dutyCycle register, the output goes low. When the counter has reached its maximum value and rolls back over to zero, the output goes high.

4.6.3 Programming Interface

4.6.3.1 Register Summary

Register Name	Offset	Description
PWM_DUTYCYCLE	0x00	PWM Duty Cycle
PWM_ENABLE	0x04	PWM Enable
PWM_DISABLE	0x08	PWM Disable
PWM_PRESCALE_ENABLE	0x0C	PWM Prescale Enable
PWM_ PRESCALE_DISABLE	0x10	PWM Prescale Disable

4.6.3.2 Register Descriptions

4.6.3.2.1 PWM_DUTYCYCLE

offset: 0x00

Bits	Signal	Access	Default	Description
31:8	reserved	W		Reserved
31:11	reserved	R		Reserved
10	pulseOut	RO		PWM Output
9	pulseEnable	RO		PWM Enable
8	PrescaleEnable	RO		PWM Prescale Enable
7:0	dutyCycle	R/W	0x00	PWM Duty Cycle (0=always off; 255 = always on; 128 = 50% duty cycle)

4.6.3.2.2 PWM_ENABLE

offset: 0x04

Write any value to enable PWM output.

4.6.3.2.3 PWM_DISABLE

offset: 0x04

Write any value to disable PWM output.

4.6.3.2.4 PWM_PRESCALE_ENABLE

offset: 0x04

Write any value to enable Prescale.

4.6.3.2.5 PWM_PRESCALE_DISABLE

offset: 0x04

Write any value to disable Prescale.

4.7 Stealth Mode

4.7.1 Overview

The Stealth Mode is an optional mode that can be useful in debugging a system. In this mode, registers can be accessed (written or read) without any side effects. Stealth mode is activated by setting the stealthMode bit in the stealthReg register. This mode is only available when the Stealth Mode IP block is purchased.

4.7.2 Signal Descriptions

Signal Name	I/O	Description
PRESETn	I	APB reset (active low)
PCLK	I	APB clock
PWRITE	I	APB write, 0=Read and 1=Write.
PWDATA[31:0]	I	APB write data bus
PSEL	I	APB Block select
PENABLE	I	APB signal indicating 2nd cycle of APB transfer
PRDATA[31:0]	O	APB read data bus
stealthMode	O	In this mode, no "side-effects" to reads or writes
stealthUartXMode	O	Stealth cross-connect mode: reading RBR really reads from TX FIFO or THR (writing THR does NOT write RX FIFO or RBR)
stealthUartDataSel[1:0]	O	Stealth UART data select mode – selects what data is read when performing FIFO/RBR read: 00 = normal: data is RX FIFO/RBR (or TX FIFO/THR if in X mode) 01 = FIFO errors (RX or TX FIFO determined by X mode) 10 = FIFO depth & cumulative error bit (again RX or TX FIFO) 11 = reserved

4.7.3 Programming Interface

4.7.3.1.1 Register Summary

Register Name	Offset	Description
StealthReg	0x0	Stealth mode control register

4.7.3.1.2 StealthReg

Bits	Access	Default	Description
[15:4]	Read/Write	undefined	Unused – undefined on a read, should be set to zero on a write
[3:2]	Read/Write	0x0	stealthUartDataSel - Stealth UART data select mode – selects what data is read when performing FIFO/RBR read: 00 = normal - data is RX FIFO/RBR (or TX FIFO/THR if in X mode) (when in FIFO mode, 16 reads must be performed so that FIFO pointers will end up where they started) 01 = FIFO errors (RX or TX FIFO determined by X mode) (when in FIFO mode, 16 reads must be performed so that FIFO pointers will end up where they started) 10 = FIFO depth & cumulative error bit (again RX or TX FIFO) data[4] = 1 if any errors in FIFO data[3:0] = FIFO depth (single read is OK; no FIFO pointers altered) 11 = reserved stealthUartDataSel has no effect when not in stealthMode
[1]	Read/Write	0x0	StealthUartXMode - Stealth UART cross-connect mode: reading RX FIFO/RBR really reads from TX FIFO or THR (writing THR does NOT write RX FIFO or RBR) stealthUartDataSel has no effect when not in stealthMode
[0]	Read/Write	0x0	stealthMode - In this mode, no "side-effects" to reads or writes

4.7.3.1.3 Caveats

The stealth mode support in the socUart module has a few requirements:

- for safety, turn interrupts off while doing stealth mode operations.
- after enabling stealth mode, the processor must wait at least 2 clock cycles before doing the first read of UART registers. Normally this will be no problem, but NO-OPs or some delay code might be necessary.
- since the XMT & RCV state machines hang in the armed state while in stealth mode, do all stealth mode operations as quickly as possible and immediately disable stealth mode. This will allow the state machines to resume before a character gets overwritten. Also, for this reason it might be a good idea to separate blocks of UART stealth operations from each other and from other non-UART stealth operations in time.
- all 16 locations of a FIFO must be read before exiting stealth mode to insure the state of the system is not changed.

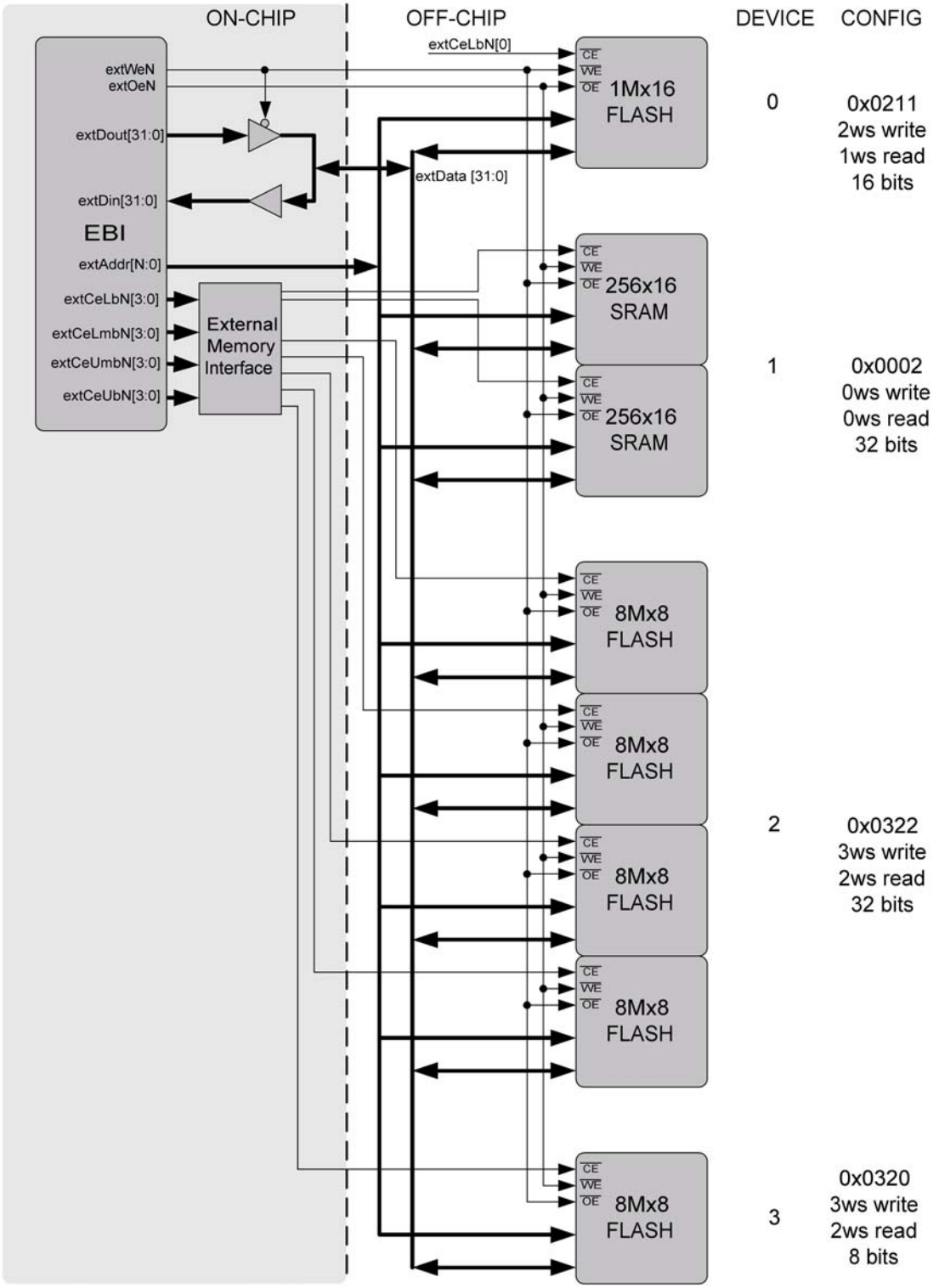
4.7.4 Functional Description

The Stealth Mode can be very useful in debugging a system. In this mode, registers can be accessed (written or read) without any side effects. Stealth mode is activated by setting the stealthMode bit in the stealthReg register. Stealth mode may be removed entirely from the compilation by turning off the definition for ENABLE_STEALTH_MODE.

Several other SoC Solutions IP components have been enhanced to take advantage of this mode of operation. An example is the 16550 UART. In stealth mode, the contents of both the transmit and receive FIFOs, the FIFO depths, and any errors in the FIFOs can be read transparently to the normal operation of the UART. A normal 16550 does not provide a way to read data from the transmit FIFO, so asserting this signal activates special steering logic in the SoC UART IP. Once stealth mode is disabled, this logic becomes transparent.

The stealth mode enhancements are only added when the Stealth Mode IP block is purchased. When stealth mode is added, the gate count is increased slightly (typically less than 1%), so the user may wish to remove stealth mode in the final product.

Appendix A: Example External Memory Configuration



Appendix B: Verilog Testbench

The Verilog testbench consists of the following components:

- Top Level Testbench File
- External Memory Models
- Processor Bus Functional Model (BFM)
- Testbench Utilities
- Module Tests
- Main Interrupt Service Routine (ISR)
- Module ISRs

The top level Verilog testbench file

- creates the clock,
- instantiates the chip,
- instantiates the memory models,
- `includes code for the BFM, utilities, module tests, & ISRs, and
- contains an initial block that runs the tests.

The external memory models emulate the memories found on an SoC Solutions development board and can be easily modified for other platforms.

Since the PiP-EC02 is designed to be connected directly to a processor, the testbench is written to emulate instructions that run on the processor, including interrupt service routines which run when interrupts are generated by the PiP-EC02. This also makes the Verilog tests easy to convert to C code for running on the actual processor.

BFM

The BFM included in the PiP-EC02 is for the ARM7TDMI processor. The BFM does not try to emulate complete functionality of the processor. It does, however, perform the logic necessary to emulate the activity of a processor executing instructions. Note that the main difference is that the instructions are not fetched from memory; rather, they are calls to tasks which read and write registers and memory locations.

The BFM operates in one of two modes: core or ARM test chip. In ARM test chip mode, the SEL[1:0] signal is used to steer the bytes of the data bus to and from the processor. This is necessary because the ARM7TDMI test chip incorporates this logic around the ARM7TDMI core. This allows for direct connections to memory devices of various widths.

The BFM supplies read and write tasks for both main “code” and ISR code. The module tests consist of calls to these basic read and writes tasks.

Testbench Utilities

Utility routines are provided which

- read and verify memory/register values,

- cause simulation time to continue without accessing valid memory locations, and
- facilitate testing (testInit, testFailure, & testDone)

Module Tests

The module tests emulate test code running on the processor. They consist of one or more tasks containing a series of calls to read and write tasks. Occasionally there are additional tests that reach into the Verilog hierarchy to verify signals are in the correct states. Each test should be bracketed by calls to the testInit and testDone tasks. If a failure occurs, the testFailure task should be invoked.

ISRs

The following conditions are treated as ISRs to the ARM7TDMI:

- IRQ
- FIQ
- Abort

The portion of the main ISR which handles IRQ interrupts reads the interrupt controller to determine the source of interrupt and invokes the appropriate interrupt service routine. Since the interrupts are level triggered, the module's ISR is responsible for clearing the condition that caused the interrupt to be generated; otherwise, the interrupt will stay asserted.

Appendix C: ARM Signal name mapping

The PiP-EC02 uses generic signal names for processor signals, although the PiP-EC02 has been designed to connect directly to the ARM data, address, and control signals. Here are the equivalent ARM signal names.

PiP-EC02 Signal Name	ARM Signal Name
procRstN	nRESET
procAddr[31:0]	A[31:0]
procData[31:0]	D[31:0] (or the unidirectional DIN & DOUT may be used)
fiqOutN	nFIQ
irqOutN	nIRQ
procWrRdN	nRW
procWaitN	nWAIT
procAccType[1]	nMREQ
procAccType[0]	SEQ
procByteLatch[3:0]	BL[3:0]
procDsize[1:0]	MAS[1:0]
procAbort	ABORT
ProcLock	LOCK

