BA22

32-bit RISC Processor

Integration Manual

April 2012

IP Product Version
4.0p06

Document Signature
BA22–INM–4.0p06–103

CAST, Inc.

CONFIDENTIAL

Document Version

This document with its associated Release Notes applies to the version(s) of the core specified on the cover. See the Release Notes for any updates and additional information not included here.

Document Version History

Version	Date	Person	Changes from Previous Version
1.00	15 Oct 2011	N.S.,J.S	First release
1.01	25 Oct 2011	N.S	Updated scripts to run locally, improved section 5 and 6
1.02	26 Jan 2012	J.S., N.S	Directory structure updates, configuration renames, SoC peripherals added, and improved section 5
1.03	4 Apr 2012	J.S	Qmem_defines.v updates

CAST, Inc.
11 Stonewall Court, Woodcliff Lake, NJ 07677
Phone: +1 201-391-8300 Fax: +1 201-391-8694
info@cast-inc.com www.cast-inc.com

Table of Contents

List of Figures

List of Tables

1. About This Document

This document provides information about how to integrate the BA22 IP core into a design.

Topics include discussions about the deliverables, the available package configurations, the RTL simulation environment, and the supplied example scripts for simulation and synthesis.

2. Introduction

The BA22 is a modern load/store RISC 32-bit processor core featuring industry highest code density in its class. This was achieved without compromise on performance, ease of use or scalability.

2.1 Features

The BA22 architecture includes the following principal features:

- High Performance 32-bit CPU
  - Single-cycle instruction execution on most instructions
  - Fast and precise internal interrupt response
  - Custom user instructions
  - 1.41 DMIPS/MHz
- Fast & Flexible Memory Access
  - Harvard-style Caches and MMU separate for Instructions and Data
  - Tightly coupled Quick Memory (QMEM) for fast and deterministic access to code and/or data
- Efficient Power Management
  - Power reduction from 2x to 100x by dynamic clock gating for individual units
  - Software controlled clock frequency in slow and idle modes
  - Interrupt wake-up in doze and sleep modes
- Advanced Debug Unit
  - Non-intrusive debug/trace for both RISC and system
  - Access and control of debug unit from RISC or via development interface
  - Complex chained watchpoint and breakpoint conditions
  - Uses industry standard Amontec JTAGKey USB to JTAG interface
- Integrated Peripherals
  - 32 bits-wide tick timer and Programmable interrupt controller with 32 maskable interrupt sources
- Optional Peripherals
  - AMBA bus infrastructure cores
  - GPIOs. UARTS, Timers cores
  - Serial communication cores such as I2C and SPI
  - Memory controllers, interconnect IP and more
- Easy Software Development
  - Complete IDE for Windows, and Linux under Eclipse
  - Ported C-libraries and OS.

2.2 Configurations

2.2.1 BA22-DE

The BA22-DE implements a 32-bit RISC processor for deeply embedded applications that use on-chip instruction and data memories. The core has an AMBA/AHB or Wishbone system bus interface and dedicated buses for the on-chip instructions and data memories.

The deeply embedded (DE) configuration includes 16 GPRs, the tick timer, the programmable interrupt controller, the power management unit and optionally the debug unit.

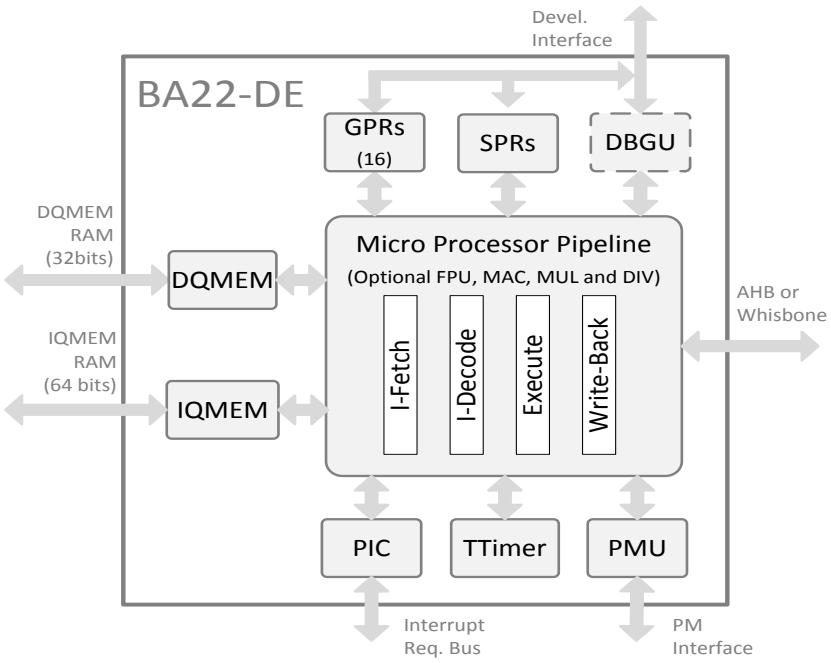

Figure 2-1: BA22-DE

2.2.2 BA22-EM

The BA22-EM implements a 32-bit RISC processor for embedded applications that use off-chip instruction and data memories and that may need to run a real-time operating system (RTOS).

The embedded (EM) configuration has cache memories, dedicated buses for the on-chip instructions and data memories, and an AMBA/AHB or optional Wishbone system bus interface. It includes 32 GPRs, the tick timer, the programmable interrupt controller, the power management unit and optionally the debug unit.

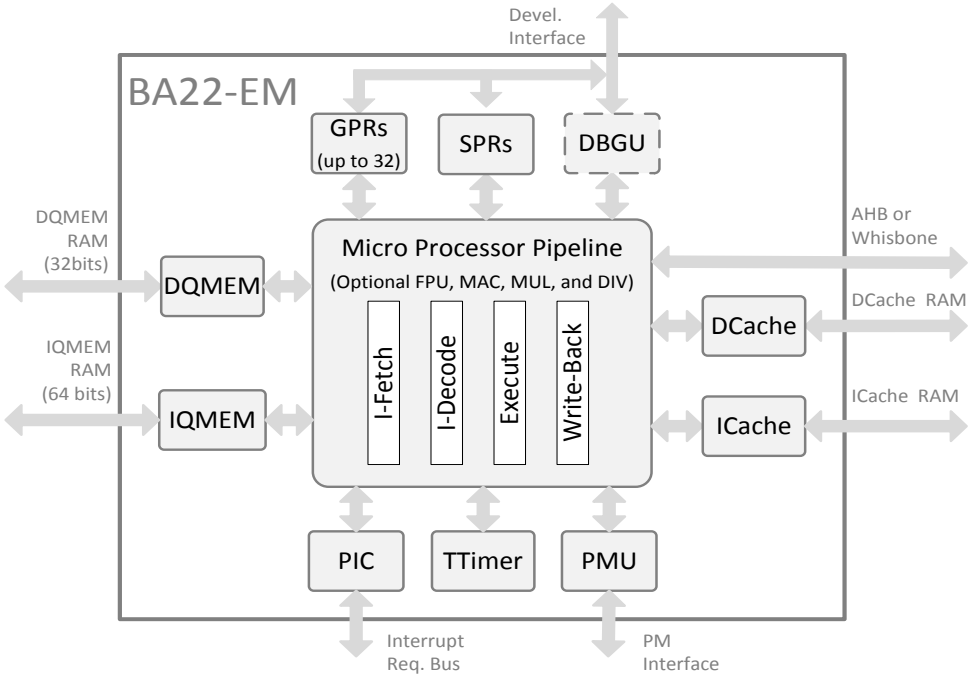

Figure 2-2: BA22-EM

2.2.3 BA22-AP

The BA22-AP implements a 32-bit RISC processor for demanding embedded applications that use off-chip instruction and data memories and that may need to run a real-time operating system (RTOS) or a full operating system such as Linux or Android.

The application configuration (AP) has instruction and data cache memories and memory management units, dedicated buses for on-chip instructions and data memories, and an AMBA/AHB or optional Wishbone system bus interface. Optional floating point, divider and multiply–accumulate units benefit DSP applications. The platform includes up to 32 GPRs, the tick timer, the programmable interrupt controller, the power management unit and optionally the debug unit.

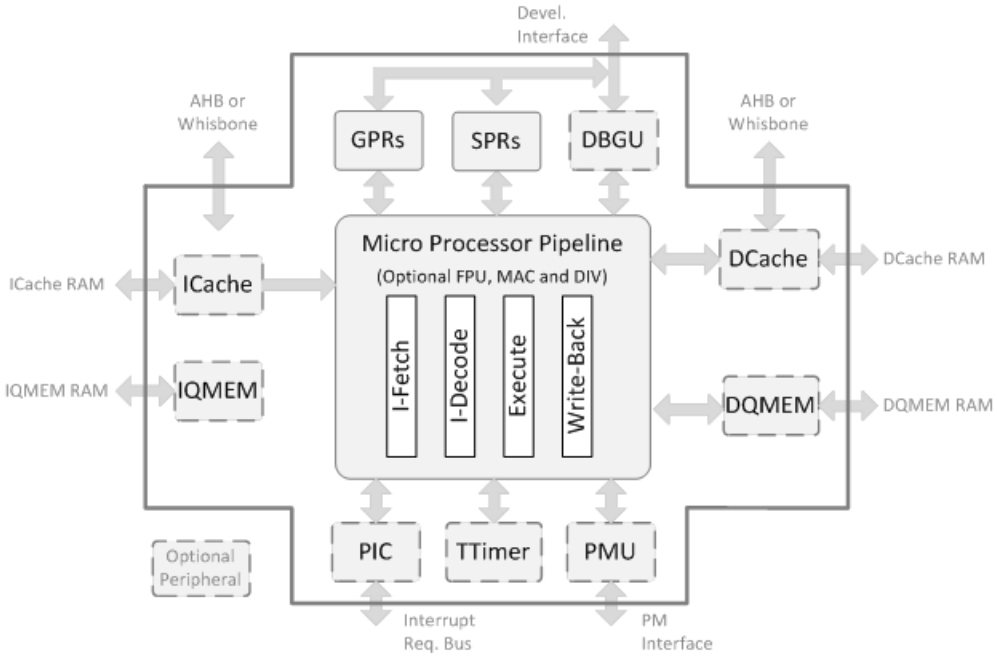

Figure 2-3: BA22-AP

3. Deliverables

The BA22 IP core is typically delivered by electronic means (ftp, etc.) in the form of a gzipped tarball file (Linux) or a zip self-extracting executable file (Windows). A CD-ROM or DVD copy is also available upon request.

The hardware/RTL package and the software tool-chain package are typically packaged separately since the software may have operating system dependencies (e.g. win32 versus Linux)

On Linux, you should extract the contents of the RTL package by copying the file to the target directory <INSTALL_DIR>, changing the directory to <INSTALL_DIR>, and then executing the following example command:

```
tar xvfz customer_ba22_de_4.0p06_101.tgz (Linux)
```

The optional debug interface package is delivered as a separate gzipped tarball file (Linux) or zip self-extracting executable file (Windows). Example command of extracting the contents of the debug interface package:

```
tar xvfz ba22_debug_c4.0p06.tgz (Linux)
```

3.1 BA22 Directory Structure

An example delivered directory structure is shown below. Your package may be missing certain directories depending upon your license or configuration type. Include files are denoted by '*'.

```
ba22/
  readme.txt
  docs/      *.pdf
  src/
      synth/
          config/
              de/
                  ba22_defines.v*     peripheral_defines.v*     qmem_defines.v*
                  ba22_filelist.txt   soc_filelist.txt          models_filelist.txt
              <user_config>/    [same as de]
          common/
              ba22/      [additional files possible for other configurations]
                  ahb_master.v                    ba22_alu_barrel_shifter.v
                  ba22_alu.v                      ba22_cnt_ci.v
                  ba22_compare.v                  ba22_du_data.v
                  ba22_ex_dep_det.v               ba22_execute.v
                  ba22_reg.v                      ba22_id_dec.v
                  ba22_id_imm_dec.v               ba22_id_opcode_dec.v
                  ba22_id_reg_dec.v               ba22_id_reg.v
                  ba22_dc_lsu.v                   ba22_lsu.v
                  ba22_maccess_wback.v            ba22_mac.v
                  ba22_mdb.v                      ba22_pic.v
                  ba22_pm.v                       ba22_qmem64_fetch.v
                  ba22_register.v                 ba22_rf.v
                  ba22_rst_unit.v                 ba22_spr_bic_usr.v
                  ba22_spr_bic.v                  ba22_sprs.v
                  ba22_tt.v                       ba22_idecode.v
                  ba22_pipe.v                     ba22_rams.v
                  ba22_cpu.v                      ba22_top.v
                  timescale.v *                   ahb_master_defines.v *
                  ba22_constants.v *              ba22_revision.v *
              fpu32/
                  [floating point unit32 files (fpu32*)]
              models/
                  dpram8.v      sram8.v       sram32.v
                  sram64.v      qmem_arbiter.v tdpram8.v
                  tdpram32.v    tdpram64_32.v
              peripherals/
                  [soc peripheral files]
              wrappers/
                  ba22_top_qmem.v(synth)        ba22_top_qmem_soc.v(synth)
      sim/
          bench.v          ba22_top_qmem.v (sim)   ahb_slave_behavioral.v
          c_clgen.v,       initram32.v             wb_slave_behavioral.v
          monitor.v        bench_defines.v *       bench_constants.v*
  scripts/
      synth/
          altera/
              quartus.tcl       ba22_top.sdc     readme.txt
          xilinx/
              ise.tcl           ba22_top.ucf     readme.txt
          synopsys/
              design_compiler.tcl                readme.txt
          cadence/
              rtl_compiler.tcl  ba22_top.sdc     readme.txt
          mentor/
```

```
            precision.tcl                   readme.txt
    sim/
        mentor/
            modelsim.tcl                    readme.txt
        cadence/
            ncsim.sh                        readme.txt
tests/
    de/
        ba22_pic/*
        ba22_tt/*
        loop-2ram/*
        hello/*

    <user_config>/  [additional testcases for other configurations]
```

4. MACRO Configuration

The Verilog source version of the BA22 can be configured before synthesis through macros in the ba22_defines.v include file. This file is typically found in the `src/synth/config/[user_config]` directory. This file should be configured to match your hardware configuration and contains parameters for many optional units that should be disabled if not used (or not licensed). Similarly, the peripheral_defines.v include file can be used to configure the AHB subsystem peripherals.

4.1 ba22_defines.v options

These ‘defines in ba22_defines.v should be modified based upon your design to configure your BA22 CPU.

4.1.1 CPU Configuration

Basic CPU configuration options can be set via defines show in Table 4-1:

Table 4-1: CPU Configuration

`DEFINE	Description
BA22_RST_ACT_HIGH	CPU reset signal is active high; it is commented for active low reset
BA22_RST_ASYNC	CPU reset is asynchronous. Comment the `define to make the reset synchronous.
BA22_INSN_FETCH_PIPE_STAGE	If defined, this parameter will add a registered instruction fetch to the CPU pipeline. This will improve the speed of the design, but will sacrifice some area. This is commented out by default.
BA22_PC_MAPPED_TO_GPRS	If defined, this will use specify GPR15 (or GPR31 if there are 32 GPRs) as the program counter register. This can be done in special cases where the user wants to program the program counter directly.
BA22_ENDIANNESS	If defined to 0, the CPU will access external memories using big endian. If defined to 1, the CPU will use little endian. The external input pin ben_le_sel_i is not used. If this is undefined (commented), the pin ben_le_sel_i is used to determine the endianness of the memories in your design.

4.1.2 Modules Implemented

These defines are normally pre-configured prior to delivery, depending upon the configuration desired. Enabling a peripheral that was not delivered may result in compilation or simulation errors.

Table 4-2: Optional Modules Configuration

`DEFINE	Description	DE	EM	AP	Section
BA22_DU_IMPLEMENTED	Debug unit	✓	✓	✓	4.1.6
BA22_TT_IMPLEMENTED	Tick timer unit	✓	✓	✓	
BA22_DMMU_IMPLEMENTED	Data memory management unit			✓	4.1.10
BA22_IMMU_IMPLEMENTED	Instruction memory management unit			✓	4.1.11
BA22_DC_IMPLEMENTED	Data cache unit		✓	✓	4.1.12
BA22_IC_IMPLEMENTED	Instruction cache unit		✓	✓	4.1.13
BA22_PIC_IMPLEMENTED	Programmable interrupt controller unit	✓	✓	✓	4.1.7
BA22_PM_IMPLEMENTED	Power management unit	✓	✓	✓	4.1.8
BA22_DIV_IMPLEMENTED	Divider unit is implemented	*	*	*	
BA22_MUL_IMPLEMENTED	Multiplication unit is implemented	*	*	*	4.1.9
BA22_MAC_IMPLEMENTED	Multiply accumulate unit	*	*	*	

BA22_FPU32_IMPLEMENTED	Single precision floating point unit	*	*	*	
BA22_QMEM_IMPLEMENTED	Indicates that the closely coupled logic memory interface should be implemented	✓	✓	✓	4.1.4
BA22_SATARITH_IMPLEMENTED	Saturate arithmetic	*	*	*	
BA22_16GPRS	Indicates that the only 16 general purpose registers are available (GPRs). When BA22_16GPRS is enabled, C code should always be compiled with the –m16regs switch	✓	*	*	
BA22_AHB	Enable AHB interface (Wishbone if not defined)	✓	✓	✓	4.1.3

(*) indicates that the module is available for the configuration.

4.1.3 AHB Interface

When implementing the AHB interface, the `define shown in Table 4-3 is available.

Table 4-3: AHB Bus Configuration

`DEFINE	Default Value	Description
BA22_AHB_RST_ACT_HIGH	1'b1	When defined, it specifies that the AHB reset is active high. When undefined/commented, then it is active low.

4.1.4 Embedded QMEM Memory

The `defines shown in Table 4-4 are required when the embedded QMEM is implemented.

The embedded memory interface is accessed via the IQMEM bus or the DQMEM bus. The following defines specify the address ranges of the QMEM memories. Addresses that do not fall in this specific range will be directed to the external interface (either AHB or WB). The memory access will be directed to the Embedded QMEM memory when bits specified in BA22_DQMEM_BASE_ADDR_DEC_BITS are equal to BA22_DQMEM_BASE_ADDR value, and similarly for the Instruction QMEM.

Table 4-4: QMEM Configuration

`DEFINE	Default Value	Description
BA22_DQMEM_BASE_ADDR_DEC_BITS	31:15	Data QMEM decode address range to be compared for DQMEM access
BA22_DQMEM_BASE_ADDR	32'h00000000	Data QMEM base address comparison (must be 32 bits, even though less bits will be compared)
BA22_IQMEM_BASE_ADDR_DEC_BITS	31:15	Instruction QMEM decode address range
BA22_IQMEM_BASE_ADDR	32'h00000000	Instruction QMEM base address (must be 32 bits, even though less bits will be compared)

Additionally, BA22_QMEM_AW is a suggested parameter that the user may use to define the size of the actual memory. It is not used internally by the BA22. The parameter is the address width of the 64-bit memory such that the address range is [`BA22_QMEM_AW-1:0]. It is used in the example file common/ba22_qmem.v.

Table 4-5: QMEM Model Configuration

`DEFINE	Default Value	Description
BA22_QMEM_AW	12	Example memory address width

The size and address ranges of the QMEM memory must be consistent with the linker memory map used by the gcc / eclipse tools.

4.1.5 Exception Vectors

The exception vectors can be configured via defines; the default values are shown in Table 4-6 (the exceptions that are required depend upon your configuration/design).

Table 4-6: Exception Vectors Configuration

`DEFINE	Default Value (hex)	Description
BA22_EXT_VECT_EPH	0000_0000	Exception vector prefix
BA22_EXC_VECT_RESET	0000_0100	Reset vector. This location will usually contain a 'jump to start of code instruction)
BA22_EXC_VECT_ITLB_MISS	0000_0A00	No matching entry in instruction translation lookaside buffer (ITLB miss)
BA22_EXC_VECT_IPAGE_FAULT	0000_0400	Instruction Page Fault vector
BA22_EXC_VECT_INSN_BUS_ERR	0000_0200	Instruction bus error vector
BA22_EXC_VECT_UNDEF_INST	0000_0700	Illegal instruction vector
BA22_EXC_VECT_UNALIGNED	0000_0600	Unaligned memory access vector
BA22_EXC_VECT_SYSCALL	0000_0C00	System call vector
BA22_EXC_VECT_TRAP	0000_0E00	Trap exception
BA22_EXC_VECT_DTLB_MISS	0000_0900	No matching entry in data translation lookaside buffer (DTLB miss)
BA22_EXC_VECT_DPAGE_FAULT	0000_0300	Data page fault vector
BA22_EXC_VECT_DBUS_ERR	0000_0200	Bus error vector
BA22_EXC_VECT_TICK_TIMER	0000_0500	Tick Timer vector (from the Tick Timer module)
BA22_EXC_VECT_INT	0000_0800	External interrupt vector from the PIC module
BA22_EXC_VECT_FP	0000_0D00	Floating point exception

4.1.6 Debug Unit

When the debug unit is implemented, the `define settings shown in Table 4-7are available.

Table 4-7: Debug Unit Configuration

DEFINE	Default Value	Description
BA22_DU_TB_IMPLEMENTED	undefined	Implements the debug unit trace buffer
BA22_DU_TB_DEPTH	10	Debug unit trace buffer depth; (only if BA22_DU_TB_IMPLEMENTED)
BA22_DU_HWBKPTS	defined	Debug unit hardware breakpoints should be implemented
BA22_DU_PAIR0_IMPLEMENTED	defined	Indicates if DVR0/DCR0 hardware breakpoint is defined
BA22_DU_PAIR1_IMPLEMENTED	defined	Indicates if DVR1/DCR1 hardware breakpoint is defined
BA22_DU_PAIR2_IMPLEMENTED	undefined	Indicates if DVR2/DCR2 hardware breakpoint is defined
BA22_DU_PAIR3_IMPLEMENTED	undefined	Indicates if DVR3/DCR3 hardware breakpoint is defined
BA22_DU_PAIR4_IMPLEMENTED	undefined	Indicates if DVR4/DCR4 hardware breakpoint is defined
BA22_DU_PAIR5_IMPLEMENTED	undefined	Indicates if DVR5/DCR5 hardware breakpoint is defined
BA22_DU_PAIR6_IMPLEMENTED	undefined	Indicates if DVR6/DCR6 hardware breakpoint is defined
BA22_DU_PAIR7_IMPLEMENTED	undefined	Indicates if DVR7/DCR7 hardware breakpoint is defined

4.1.7 Programmable Interrupt Controller

When the PIC is implemented, the `define settings shown in Table 4-8 is available.

Table 4-8: Programmable Interrupt Controller Configuration

`DEFINE	Default Value	Description
BA22_PIC_INTS	16	The number of external interrupts inputs. This can range from 1 to 32

4.1.8 Power Management

If the power management module is implemented, Table 4-9 shows the available `define settings. Further details about the dynamic power management functionality can be found in *AN001 - Beyond BA22 Power Management Unit_v1.*

Table 4-9: Power Management Unit Configuration

`DEFINE	Description
BA22_PM_BUS_CLK_NEQ_CPU_CLK	Indicates that the external BUS and CPU use different clock frequency clocks.
BA22_PM_FIXED_CLK_RATIO	Only used if BA22_PM_BUS_CLK_NEQ_CPU_CLK is defined. This indicates that Bus to CPU clock ratio in the designs is equal to the fixed ratio defined by BA22_PM_PWR_UP_CLK_RATIO
BA22_PM_PWR_UP_CLK_RATIO	Power up Bus to CPU clock ratio. Default value is 3'b000 (1:1 ratio). Possible values are: 8'b0000_0000 -> cpu_clk : bus_clk = 1 : 1 8'b0000_0010 -> cpu_clk : bus_clk = 2 : 1 8'b0000_0100 -> cpu_clk : bus_clk = 4 : 1 8'b1000_0000 -> cpu_clk : bus_clk = 1 : 1 8'b1000_0100 -> cpu_clk : bus_clk = 1 : 4
BA22_PM_EVENT_IMPL	Indicates that the Power Management Event (signal and control register will be used).
BA22_PM_UNIT_PD_IMPL	Indicates that the Units power down control register will be used.
BA22_PM_PWR_UP_DU_CLK_EN	Defines power-up value for debug unit clock enable. The default is 1'b1

4.1.9 Multiplier

If the multiplier is implemented, Table 4-10 shows the additional `define available.

Table 4-10: Multiplier Configuration

DEFINE	Default Value	Description
BA22_MUL_3STAGE	undefined	When defined, this indicates that a 3-stage multiplier module should be used; otherwise, a 2-stage multiplier module will be used. The extra stage/register would allow for a higher active frequency if needed.

4.1.10 Data Memory Management Interface (DMMU)

If the DMMU is implemented, Table 4-11 shows are the additional `defines available.

Table 4-11: Data Memory Management Unit Configuration

DEFINE	Default Value	Description
BA22_DTLB_WAYS_4	*	Data translation lookaside buffers ways 4
BA22_DTLB_WAYS_3	*	Data translation lookaside buffers ways 3
BA22_DTLB_WAYS_2	*	Data translation lookaside buffers ways 2
BA22_DTLB_WAYS_1	*	Data translation lookaside buffers ways 1

BA22_DTLB_REPLACEMENT_BINTREE		Define the replacement strategy; BINTREE for 4 ways
BA22_DTLB_REPLACEMENT_LRU		Define the replacement strategy; LRU for 3 or 2 ways
BA22_DTLB_ENTRIES_PER_WAY	6	Number of entries per way as a power of 2

(*) If multiple BA22_DTLB_WAYS_* options are selected, the highest numbered option is in effect

4.1.11 Instruction Memory Management Interface (IMMU)

If the IMMU is implemented, Table 4-12 shows the additional `defines available.

Table 4-12: Instruction Memory Management Unit Configuration

DEFINE	Default Value	Description
BA22_ITLB_WAYS_4	*	Instruction translation lookaside buffers ways 4
BA22_ITLB_WAYS_3	*	Instruction translation lookaside buffers ways 3
BA22_ITLB_WAYS_2	*	Instruction translation lookaside buffers ways 2
BA22_ITLB_WAYS_1	*	Instruction translation lookaside buffers ways 1
BA22_ITLB_REPLACEMENT_BINTREE		Define the replacement strategy; BINTREE for 4 ways
BA22_ITLB_REPLACEMENT_LRU		Define the replacement strategy; LRU for 3 or 2 ways
BA22_ITLB_ENTRIES_PER_WAY	6	Number of entries per way as a power of 2

(*) If multiple BA22_ITLB_WAYS_* options are selected, the highest numbered option is in effect

Table 4-13 shows the additional `defines when either DMMU or IMMU is implemented.

Table 4-13: Non-cacheable Regions Configuration

DEFINE	Default Value	Description
BA22_NON_CACHEABLE_REGION0	defined	Set region0 to be non-cacheable if it is defined
BA22_NON_CACHEABLE_REGION1	undefined	Set region1 to be non-cacheable if it is defined
BA22_NON_CACHEABLE_REGION0_DEC_BITS	31:31	
BA22_NON_CACHEABLE_REGION1_DEC_BITS	31:31	
BA22_NON_CACHEABLE_REGION0_DEC_ADDR	32'h80000000	
BA22_NON_CACHEABLE_REGION1_DEC_ADDR	32'h80000000	
BA22_NON_CACHEABLE_CUSTOM_REGION0	*	Set custom non-cacheable region0 (reg_pvpn < 32'h00008000)
BA22_NON_CACHEABLE_CUSTOM_REGION1	*	Set custom non-cacheable region1 (reg_pvpn >= 32'h08010000) & (reg_pvpn < 32'ha0010000)
BA22_NON_CACHEABLE_CUSTOM_REGION2	*	Set custom non-cacheable region2 (reg_pvpn >= 32'ha0020000)

4.1.12 Data Cache

If the Data Cache is implemented, Table 4-14 shows the additional `defines available.

Table 4-14: Data Cache Configuration

DEFINE	Default Value	Description
BA22_DC_INCREASELATENCY	undefined	Will increase the register latency of the data cache (useful where a maximum frequency is needed)
BA22_DC_STORE_BUFFER	undefined	Will implement a data cache store buffer
BA22_DC_WAYSIZE	13	Specifies the size of data cache block as a power of 2.
Define Data Cache Associativity		

BA22_DC_WAYS_4	*	Data cache ways 4
BA22_DC_WAYS_3	*	Data cache ways 3
BA22_DC_WAYS_2	*	Data cache ways 2
BA22_DC_WAYS_1	*	Data cache ways 1
Define Replacement Strategy		
BA22_DC_REPLACEMENT_BINTREE		Define the replacement strategy as BINTREE for 4 ways
BA22_DC_REPLACEMENT_LRU		Define the replacement strategy as LRU for 3 or 2 ways

(*) If multiple BA22_DC_WAYS_* options are selected, the highest numbered option is in effect

4.1.13 Instruction Cache

If the Instruction Cache is implemented, Table 4-15 shows the additional `defines available.

Table 4-15: Instruction Cache Configuration

DEFINE	Default Value	Description
BA22_IC_WAYSIZE	13	Specifies the size of the instruction cache block as a power of 2.
BA22_IC_WAYS_4	*	Instruction cache ways 4
BA22_IC_WAYS_3	*	Instruction cache ways 3
BA22_IC_WAYS_2	*	Instruction cache ways 2
BA22_IC_WAYS_1	*	Instruction cache ways 1
BA22_IC_REPLACEMENT_BINTREE		Define the replacement strategy; BINTREE for 4 ways
BA22_IC_REPLACEMENT_LRU		Define the replacement strategy; LRU for 3 or 2 ways

(*) If multiple BA22_IC_WAYS_* options are selected, the highest numbered option is in effect

4.1.14 Technology Options

4.1.14.1. FPGA Implementations

If the target technology is an FPGA, some example cache or MMU related memories are instantiated in the design using vendor dependent cells (e.g. ram-blocks). These defines shown in Table 4-16 are used to specify the architecture used. This section will be expanded as further examples are integrated.

Table 4-16: FPGA Target Configuration

DEFINE	Description
BA22_TARGET_FPGA_XILINX_V5	Target the BA22 to FPGA Xilinx Virtex 5
BA22_TARGET_FPGA_XILINX_V4	Target the BA22 to FPGA Xilinx Virtex 4

4.2 peripheral_defines.v options (Optional)

The ‘defines in peripheral_defines.v should be modified based upon your design to configure your AHB subsystem. These `defines will enable or disable peripherals in the file socSystem.v (found in the `src/synth/common/peripherals` directory).

Table 4-17: Peripherals Configuration

`DEFINE	Description
INSTANTIATE_TIMER	Define to enable instantiation of APB Timer peripheral
INSTANTIATE_UART	Define to enable instantiation of APB UART peripheral
INSTANTIATE_GPIO	Define to enable instantiation of APB General Purpose Input Output
INSTANTIATE_AHB_EBI	Define to enable instantiation of AHB External Bus Interface
INSTANTIATE_LCD_CONTROLLER	Define to enable instantiation of AHB TFT-LCD Controller

INSTANTIATE_DES_CONTROLLER	Define to enable instantiation of AHB Data Encryption Standard
INSTANTIATE_DMA_CONTROLLER	Define to enable instantiation of AHB Master/Slave DMA Controller
INSTANTIATE_I2C	Define to enable instantiation of APB I2C Master
INSTANTIATE_SPI_MASTER	Define to enable instantiation of APB SPI Master
INSTANTIATE_SPI_SLAVE	Define to enable instantiation of APB SPI Slave
INSTANTIATE_INTR_CONTROLLER	Define to enable instantiation of APB interrupt Controller
INSTANTIATE_WATCHDOG_TIMER	Define to enable instantiation of APB Processor WatchDog Timer
INSTANTIATE_PARALLEL_PORT	Define to enable instantiation of APB 8-bit Parallel Port
INSTANTIATE_PULSEWIDTH_MODULATOR	Define to enable instantiation of APB Pulse Width Modulator (PWM)
INSTANTIATE_REMAP	Define to enable instantiation of APB Remap
INSTANTIATE_AHB_MEM	Define to enable instantiation of AHB Internal SSRAM Controller
DUAL_TIMER	Define to use two APB Timer peripherals
DUAL_UART	Define to use two APB UART peripherals
DUAL_PULSEWIDTH_MODULATOR	Define to use two APB PWM peripherals
WATCHDOG_SECURITY	Define to use more secure register interface for WatchDog Timer
USE_UART_BAUD_CLK_DIV	Define to use output of SoC ClkDiv to generate baud rate clock
USE_GPIO_INTERRUPTS	Define to use GPIO to cause interrupts
USE_SPI_FIFOS	Define to instantiate SPI Fifos
Define the width	
IRQ_WIDTH	16; set the interrupt source vector width to 16
GPIO_WIDTH	8; set the GPIO vector width to 8
Define Interrupt source bits	
IRQDMA_BITPOS	13; interrupt vector bit 13 is used for DMA controller interrupt
IRQSPIM_BITPOS	12; interrupt vector bit 12 is used for SPI Master interrupt
IRQSPIS_BITPOS	11; interrupt vector bit 11 is used for SPI Slave interrupt
IRQDES_BITPOS	10; interrupt vector bit 10 is used for DES Controller interrupt
IRQLCD_BITPOS	9; interrupt vector bit 9 is used for LCD Controller interrupt
IRQI2C_BITPOS	8; interrupt vector bit 8 is used for I2C interrupt
IRQGPIO0_BITPOS	7; interrupt vector bit 7 is used for GPIO interrupt
IRQWDTimer_BITPOS	6; interrupt vector bit 6 is used for Watchdog Timer interrupt
IRQParallel_BITPOS	5; interrupt vector bit 5 is used for Parallel Port interrupt
IRQSerialB_BITPOS	4; interrupt vector bit 4 is used for Serial Port B (UART2) interrupt
IRQSerialA_BITPOS	3; interrupt vector bit 3 is used for Serial Port A (UART1) interrupt
IRQTimer2_BITPOS	2; interrupt vector bit 2 is used for Timer2 interrupt
IRQTimer1_BITPOS	1; interrupt vector bit 1 is used for Timer1 interrupt
IRQUser_BITPOS	0; interrupt vector bit 0 is used for testing programmed interrupt

4.3 qmem_defines.v options

The file qmem_defines.v is used by the wrapper file ba22_top_qmem.v (found in the `src/synth/common/wrappers` directory). The ‘defines in qmem_defines.v specify the qmem memory configuration that should be used in your design.

Note that internal QMEM memory access versus external AHB bus access is defined in ba22_defines.v (see section 4.1.4).

Table 4-18: Memory Configuration

`DEFINE	Description
Memory configuration (must define one of them)	
BA22_QMEM_SEPARATED	It will use separate memories for instruction and data.
BA22_QMEM_UNIFIED_SP	It will use unified memory (SRAM) with qmem_arbiter.
BA22_QMEM_UNIFIED_DP	It will use unified memory (true DPRAM), one port for instruction and the other port for data.
Memory Size	
IQMEM_AW	The instruction memory address width for separate memory configuration. The default address width is 10 for 1kx64bit memory
DQMEM_AW	The data memory address width for separate memory configuration. The default address width is 11 for 2kx32bit memory
IDQMEM_AW	The unified memory address width if BA22_QMEM_UNIFIED_SP or BA22_QMEM_UNIFIED_DP is defined. The default address width is 11 for 2kx64 bit memory

4.4 BA22 Configurations Settings

The ba22_defines.v files contains the `defines which should be set for your environment. Most of these items will be preset before it is delivered, dependent upon the purchased package and peripherals. The enabling of certain `defines may imply dependencies on additional RTL files.

4.4.1 BA22-DE Configuration

The DE configuration is typically used for deeply embedded systems. When BA22 DE is configured, instruction and data QMEM along with data AHB interface are implemented and available as interfaces on the top-level module.

Instruction and data QMEMs are separated, but can be implemented as uniform using an appropriate arbiter. Note that data memory access should always have priority over instruction memory access. An example of this is in “hello” testcase (see section 5.2.3.2). The Data AHB interface can be used to connect external memories or various peripheral units.

The BA22 DE configuration is achieved by commenting the following defines.

- BA22_DC_IMPLEMENTED
- BA22_IC_IMPLEMENTED
- BA22_DMMU_IMPLEMENTED
- BA22_IMMU_IMPLEMENTED

4.4.2 BA22-EM Configuration

The BA22–Embedded configuration implements along with the data and instruction QMEM, data and instruction Wishbone or AHB interfaces. Interfaces to cache and MMUs are also present, however their usage is optional as well as the usage of QMEM. If any of the interfaces is not used, its input signals should be wired to logical zero.

The BA22-EM configuration is achieved by uncommenting the following defines:

- BA22_DC_IMPLEMENTED – implement data cache
- BA22_IC_IMPLEMENTED – implement instruction cache
- BA22_DMMU_IMPLEMENTED – implement data memory management unit

- BA22_IMMU_IMPLEMENTED – implement instruction memory management unit

The BA22_CMMU is defined whenever either one of the above define is uncommented.

4.4.3 BA22-AP Configuration

BA22 full featured general configuration included the available options of the DE and EM configurations, including the MMU's

5. RTL Simulation Environment

This chapter describes an example structure that can be used to perform RTL simulations of the BA22 and the optional SoC system.

5.1 Application Software

Each testcase consists of the files shown in Table 5-1, which are used to generate the binary or verilog data files needed for simulation.

Table 5-1: gcc Input Files

File	Description
<testcase>.c	The C code for the testcase
Makefile	Dependency description file that can be used with 'make' to regenerate the outputs
reset.S	Example reset assembly code for the testcase (described further in section 5.1.1)
*.ld	The linker script to define the memories for the compiler
intr.S	For testcases with interrupts, this is example push/pop code that can be used before/after servicing an interrupt
board.h, spr_defs.h	Header files used by C code

Assuming that the command line BA22 compile tools are in your path, the command "`make clean`" can be used to clean the environment and "`make`" can be used to rebuild the output files listed in Table 5-2.

Table 5-2: gcc Output Files

File	Description
<testcase>.ba2	A complete, compiled object output.
<testcase>.bin	Binary instruction data for loading into a memory; a result of ba-elf-objcopy –o binary
<testcase>.verilog	Verilog format instruction data for loading into a memory; a result of ba-elf-objcopy –o verilog
<testcase>.d	Ba-elf-objdump output listing assembly code
<testcase>.map	A parsing of ba-elf-nm output to yield a memory map listing

5.1.1 Reset.S

All the testcases (`tests/[user_config]/[testname]` directories) contain an example reset.S assembly file (this may also include an additional intr.S file). This file demonstrates commonly used startup activity before the execution of your main program.

Typical functions for the reset.S startup assembly file are:

- Initialization of the stack pointer in your design
- The clearing of static memories if required
- Interrupt subroutines definition and enabling
- Jump to 'main' so that your C code can begin execution

5.1.2 Intr.S

Some of the testcases that demonstrate interrupt behavior will contain this file. It contains example defines useful for exception handling.

These defines demonstrate example methods to push the state of the CPU on the stack before processing an interrupt service routine, and then popping the state off of the stack once the service routine has completed.

5.2 BA22 Test-Bench with AHB or Wishbone monitor

The example test-bench uses the DE configuration of the BA22 CPU, models for the embedded memories, and a monitoring device on the AHB or Wishbone bus. Examples include cases using a single unified QMEM memory, and a case using separate IQMEM and DQMEM memories. Other testcases may be provided for your specific configuration (refer to the delivered readme.txt for details).

5.2.1 Block diagram

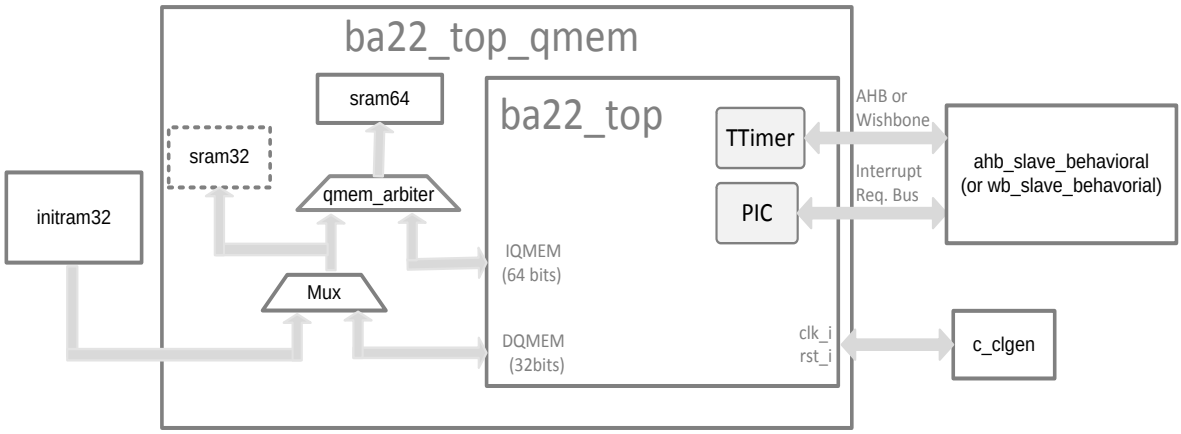

Figure 5-1: RTL Simulation Environment Block Diagram

5.2.2 Test-Bench Description

The `bench.v` is the top level module of the testbench. The testbench files are located at `src/sim` directory. It instantiates the BA22 processor with embedded memories (`ba22_top_qmem`), a simple clock and reset generator (`c_clgen`), and the AHB or Wishbone bus monitor/memory (`ahb_slave_behavioral/wb_slave_behavioral and monitor`). The functionality and configuration options of these modules are described in the following sections:

5.2.2.1. Ba22_cpu/ba22_top

The `ba22_cpu` is the top level module name of the CPU itself. In the case of advanced designs which include caches or MMU, the `ba22_top` module will also include `ba22_rams` which will optionally instantiate the cache and MMU memories via defines. The `ba22_top` is the top level of the BA22 core.

5.2.2.2. QMEM Embedded Memories

The simulation environment uses QMEMs that are instantiated in the `ba22_top_qmem.v` file.

The QMEM Embedded memories mode can be either separated, or a single unified sram, or a unified dpram, as defined in qmem_defines.v (section 4.3).

Test cases with separate data and instruction QMEMs, as well as test cases with a unified data and instruction QMEM are provided. For the unified data and instruction QMEM with SRAM, the `qmem_arbiter` module is instantiated, which models the memory arbiter.

The models for embedded memories are the `sram32` (32 bit static memory), the `sram64` (64 bit static memory), the `tdpram64_32` (dual port memory with 64 bit on one port and 32 bit on the other port) which are located at `src/synth/common/models` directory. The `sram64` and `tdpram64_32` model is programmed (initialized) with the memory contents from a `<filename>.verilog` file.

Additionally, the simulation model `initram32` is available in the simulation directory, which will read an instruction memory file (`rom.verilog`) and write it to sram64 or tdpram64_32. The example testcases contain a verilog memory format file under the `tests/[user_config]/[testname]` directory. This

file is copied to the current working directory and renamed to `rom.verilog`. If the file does not exist, the simulation will end.

5.2.2.2.1. Separate data and instruction QMEMs

If the variable BA22_QMEM_SEPARATED is defined (in qmem_defines.v, section 4.3), the testbench instantiates a separate instruction memory (`sram64`) and a separate data memory (`sram32`). The examples assume that the DQMEM (data memory) does not need to be initialized.

Table 5-3: Memory Range – Separated QMEMs

Memory	Range (size in hex)
Instruction Memory	0 to 0000_4000 (16kx8)
Data Memory	0 to 0000_4000 (16kx8)

In the separated memory example, the first 0000_1000 (hex) of the Instruction Memory is reserved for exception vectors, and the top of the stack is initialized to the bottom of the Data Memory. Memory accesses outside the range specified in Table 5-3 will be directed to the AHB or Wishbone bus.

5.2.2.2.2. Single SRAM, Unified data/instruction QMEM

When the variable BA22_QMEM_UNIFIED_SP is defined (in qmem_defines.v, section 4.3), the testbench instantiates a single memory (`sram64`) and instantiates the arbiter (`qmem_arbiter`) to control the access from the IQMEM and DQMEM buses.

Table 5-4: Memory Range – Single, unified QMEM

Memory	Range (size in hex)
Instruction / Data Memory	0 to 0000_4000 (16kx8)

In the unified memory examples, the first 0000_1000 (hex) of the Instruction Memory is reserved for vectors, and the top of the stack is initialized to the bottom of the Data Memory. Memory accesses outside this range specified in Table 5-4 will be directed to the AHB or Wishbone bus.

5.2.2.2.3. Single DPRAM, Unified data/instruction QMEM

If the variable BA22_QMEM_UNIFIED_DP is defined (in qmem_defines.v, section 4.3), the testbench instantiates a dual port memory (`tdpram64_32`) with instruction port (64 bit) and data port (32 bit).

Table 5-5: Memory Range – Single, unified QMEM

Memory	Range (size in hex)
Instruction / Data Memory	0 to 0000_4000 (16kx8)

In the unified memory examples, the first 0000_1000 (hex) of the Instruction Memory is reserved for vectors, and the top of the stack is initialized to the bottom of the Data Memory. Memory accesses outside this range specified in Table 5-5 will be directed to the AHB or Wishbone bus.

5.2.2.3. Clock and Reset Generator

The `c_clgen` module generates the clocks and reset signals. The simulation environment uses an asynchronous, active high CPU reset signal (`rst_i`). The clocks (Table 5-6) are assumed free running for this example.

Table 5-6: Test-bench Clocks

Clock Name	Function
clk_i	CPU system clock
dahb_clk_i (or dwb_clk_i)	AHB or Wishbone Bus clock
rf_clk_i	Register file clock
du_clk_i	Debug unit clock
pm_clk_i	Power management clock – must always be runningat the maximum system clock speed

To implement power management in your design, certain clocks can be gated. Please refer to *AN001 - Beyond BA22 Power Management Unit_v1* for details regarding power management.

5.2.2.4. AHB/WB Bus and Monitor

The simulation environment uses either an AHB or Wishbone bus (dependent upon the BA22_AHB `define). The `bench` module includes the AHB/WB slave models (`ahb_slave_behavioral` or `wb_slave_behavioral`) that also models external interrupt sources. The monitor module (`monitor`) will print the bus activity to log files (see section 5.2.3 ). When there is a read/write function to an AHB or Wishbone bus, it will perform simulation functionality as described in Table 5-7. The `bench_defines.v` file lists these memory addresses and each testcase has a corresponding `board.h` which is a C header file that contains the equivalent information. The C code assigning to this memory addresses will cause the monitor to perform the desired function.

Table 5-7: AHB/WB Monitor

EXT MEMORY LOCATION (hex)	NAME	DESCRIPTION (Activity upon write)
9000_0000	CAST_CHAR	Print the data as a character
9000_0004	CAST_NUM	Print the data as an integer
9000_000C	CAST_INTRMASK	Display the current interrupt mask
9000_0010	CAST_INTRSR	Display the interrupt service request register (which interrupts have a request pending)
9000_0020	CAST_INTRCLR	Clears the external interrupts
9000_0014	CAST_SR	Display the supervisor register
9000_0018	CAST_TT_CNT	Display the tick timer counter register
9000_001C	CAST_TT_INT	Printed during the tick timer interrupt exception service routine
9000_000F0	CAST_ENDSIM	Informs the bench that it should print simulation statistics and end the simulation using $finish

5.2.3 Test Cases

For each test case, the simulation environment will produce the following log files to monitor the tests activity:

[test_name].log: listing all reading/writing activities to external interface AHB/WB bus.

[test_name]_instr.log: listing all activities of reading instruction from instruction memory

[test_name]_ram.log: listing reading/writing activities from/to the data memory

Section 6 describes the example of simulation scripts that are provided.

Table 5-8: Testcase summary

Test Case	Memory	Description
loop-2ram	Separate I and D QMEM	For loop iterations writing to AHB or Wishbone bus
hello	Single QMEM with arbiter if BA22_QMEM_UNIFIED_SP is defined. Single DPRAM if BA22_QMEM_UNIFIED_DP is defined	Copy characters to AHB or Wishbone bus
ba22-tt	Single QMEM with arbiter if BA22_QMEM_UNIFIED_SP is defined. Single DPRAM if BA22_QMEM_UNIFIED_DP is defined	Tick Timer with interrupt example
ba22-pic	Single QMEM with arbiter if BA22_QMEM_UNIFIED_SP is defined. Single DPRAM if BA22_QMEM_UNIFIED_DP is defined	Programmable interrupt controller example

5.2.3.1. The Loop-2ram Test Case

The loop-2ram test-case executes a simple for loop. The circuit uses a memory for instructions and a memory for data with address ranges as described in 5.2.2.2.1. The BA22_QMEM_SEPARATED flag should be used during compilation (this is performed automatically by the supplied simulation scripts).

The `split_ram.ld` is the linker script used. The memory structure is described in section 5.2.2.2.1, and the Makefile shows an example command line linking.

The `reset.S` defines the activity to occur upon the reset exception. This routine will be placed at memory location 0000_0100 (hex).

5.2.3.2. The Hello Test Case

The hello test case includes a constant string and uses a single SRAM memory with arbiter if BA22_QMEM_UNIFIED_SP is defined or a single DPRAM if BA22_QMEM_UNIFIED_DP is defined.

The provided `unified_ram.ld` is the linker script used. The memory structure is described in 5.2.2.2.2. and the Makefile shows an example command line linking.

The single memory still needs to interface to the BA22 using both an IQMEM bus and a DQMEM bus. An example `qmem_arbiter.v` is provided to combine these buses into a single memory interface.

5.2.3.3. The Ba22-tt Test Case

The ba22-tt test case provides a programming example of the internal Tick Timer with unified data and instruction memory, including a generated interrupt.

The internal Tick Timer (TT) is optional and an implementation may choose whether or not to implement it. Register UPR[TTP] specifies whether or not the tick timer facility is present. The Tick Timer facility is enabled with TTMR[M]. TTCR is incremented with each clock cycle and a tick timer interrupt can be asserted whenever the lower 28 bits of TTCR match TTMR[TP] and TTMR[IE] is set.

The example shows how to enable the timer to generate interrupts after counting to 0x100 and then to automatically restart. The notable programming C code is:

```
mtspr(SPR_TTMR, (SPR_TTMR_RT|SPR_TTMR_IE|0x0000100));
```

Further details about the Tick Timer can be found in the document *ba22-des-4.0p06-103*, and the mtspr example function is described in section 7.1 of this document.

5.2.3.4. The Ba22-pic Test Case

The ba22-pic test case provides an example of the internal Programmable Interrupt Controller (PIC) with unified data and instruction memory.

The PIC facility is optional and the developer may choose whether or not to include it. Register UPR[PICP] specifies whether the programmable interrupt controller is implemented or not. If it is not implemented, interrupt input is directly connected to interrupt exception inputs. The PIC has two special purpose registers and up to 32 maskable interrupt inputs.

The example contains an interrupt service routine at the default vector 0x800 and demonstrates how to read the PIC status register in order to determine which interrupts are active:

```
mfspr(SPR_PICSR)
```

Further details about the PIC can be found in the document *ba22-des-4.0p06-103,* and the mfspr example function is described in section 7.1 of this document.

5.2.3.5. Other Test Cases

Your deliverable may include additional test cases depending on your BA22 configuration. Please refer to the readme.txt in the tests directory for more information about these test cases.

5.3 BA22 Test-Bench with SoC AHB and SoC Peripherals (Optional)

An SoCSystem block can be connected to the BA22 AHB bus for certain testcases. An example of the SoCSystem is shown in Figure 5-2. Please refer to the readme.txt in the `tests/[user_config]/[testname]` directory for specific information regarding each peripheral test.

5.3.1 Block diagram

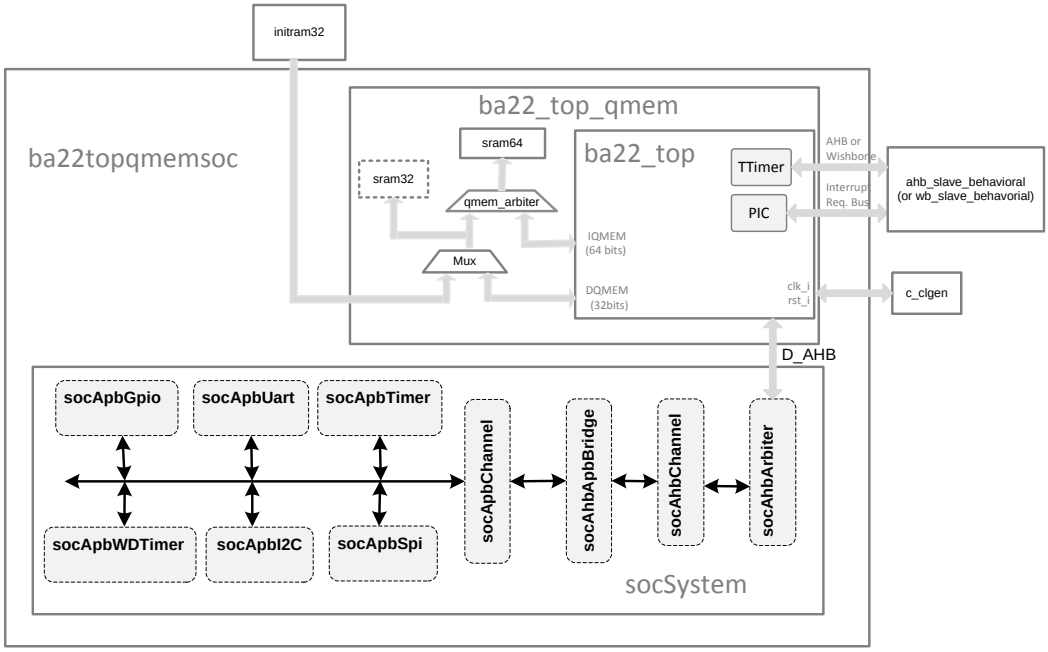

Figure 5-2: Ba22_top_qmem with SoCSystem Block Diagram

5.3.2 Test-Bench Description

The `ba22_top_qmem_soc` module includes `ba22_top_qmem` module that is described in section 5.2.2 , and SoCSystem. Further detail information regarding each module of the system can be found in the BA22 External Peripherals Specification document (*ba22-eps-4.0p01-101.pdf*).

The `bench_soc.v` is the top level module of the ba22 with SoC peripherals testbench. The testbench files are located at `src/sim` directory. It instantiates BA22 processor with embedded memories and SoC system (`ba22_top_qmem_soc`), a simple clock and reset generator (`c_clgen`), a module to read an instruction memory file (`rom.verilog`) and write it to sram64 (`initram32`), and an UART (`socApbUart`).

5.3.3 Test Cases

5.3.3.1. The uart_gpio_leds Test Case

The uart_gpio_leds test case provides an example of BA22 with SoC UART, and SoC GPIO peripherals. This test case will do the following:

- Turn on the LEDs
- Enable a tick timer interrupt, where the interrupt service routine will blink the LEDs
- Enable an UART serial interface
- In the `bench_soc`, the `socApbUart` will send the selections of LED, blinking speed, and blinking pattern.
- Upon completion of simulation, the `led_monitor.log` and `uart_monitor.log` are created. The `led_monitor.log` contains the activities on LED, and the `uart_monitor.log` contains the activities on `socApbUart`.

6. EDA tools and Sample Scripts

In the directory `src/synth/config/<user_config>`, there are files, which list the names of the verilog files that need to be compiled or synthesized for the user's configuration.

- *ba22_filelist.txt* lists all of the ba22 core files, which exist in `src/synth/common/ba22`
- *soc_filelist*.txt lists all of the SoC peripheral files, which exist in `src/synth/common/peripherals`
- *models_filelist.txt* lists embedded memory files that exist in `src/synth/common/models`

The configuration directory `src/synth/config/<user_config>`, the ba22 directory `src/synth/common/ba22`, the peripherals directory `src/synth/common/peripherals` should be in the users's include search path.

Simulation scripts are found in `scripts/sim/<tool>`, and synthesis scripts can be found in `scripts/synth/<tool>`.

The scripts as supplied are expected to be run from the individual scripts/<tool> directory (or a directory parallel to <tool>). This means that the relative path to the Mentor Modelsim simulation script will be ./modelsim.tcl (or ../mentor/modelsim.tcl if you are running from a parallel directory). Example command lines are provided below.

6.1 Simulation – RTL simulation

Sample of Modelsim and Cadence simulation scripts are included under scripts/sim/ directory.

6.1.1 Modelsim – modelsim.tcl

The modelsim.tcl performs:

- compilation of ba22 core files that are listed in the `src/synth/config/<user_config>/ba22_filelist.txt`
- compilation on SoC peripheral files that are listed in the `src/synth/config/<user_config>/soc_filelist.txt` if PERIPHERAL variable is set to "SoC"
- compilation of embedded memory files that are located under `src/synth/common/models` and top level files that are located under `src/synth/common/wrappers` directory.
- compilation on testbench files under `src/sim` directory
- copy the Verilog format instruction memory file for `<user_config>` and `<testname>` to local working directory and rename it to rom.verilog
- simulation run

The configuration `<user_config>`, the testname `<testname>`, and the peripheral `<peripheral>` can be set in the modelsim.tcl file with the following variables:

- CONFIG : the configuration to be used (e.g. "de")
- TESTNAME: the test name to be simulated (e.g. "ba22-tt")
- PERIPHERAL: set it to "SoC" if it is using SoC peripherals, else ""

The CONFIG and TESTNAME variables can also be set on the command line of the tcl script.

The default script variables require that the user should run in the scripts/mentor directory (or a directory parallel to it).

Example usage for ba22-tt testcase, supplying arguments on the command line:

```
cd ba22/scripts/sim/mentor
vsim –c –do "do modelsim.tcl de ba22-tt".
```

Example usage for ba22-tt testcase, editing the modelsim.tcl:

```
cd ba22/scripts/sim/mentor
# Edit the modelsim.tcl and set CONFIG=”de”
# Edit the modelsim.tcl and set TESTNAME=”ba22-tt”
vsim –c –do modelsim.tcl
```

Upon the completion of the simulation, the log files will be created in the working directory as described in section 5.2.3 .

Any test case with suffix “-2ram” will automatically define BA22_QMEM_SEPARATED, and will use two separate memories for instruction and data.

6.1.2 Cadence – ncsim.sh

The ncsim.sh performs:

- compilation of ba22 core files that are listed in the `src/synth/config/<user_config>/ba22_filelist.txt`
- compilation on SoC peripheral files that are listed in the `src/synth/config/<user_config>/soc_filelist.txt` if PERIPHERAL variable is set to “SoC”
- compilation of embedded memory files that are located under `src/synth/common/models` and top level files that are located under `src/synth/common/wrappers` directory.
- compilation on testbench files under `src/sim` directory
- copy the Verilog format instruction memory file for `<user_config>` and `<testname>` to local working directory and rename it to rom.verilog
- elaboration and simulation run with of <user_config> and <testname>

The configuration `<user_config>`, the testname `<testname>`, and the peripheral `<peripheral>` need to be set in the ncsim.sh file with the following variables:

- CONFIG : the configuration (e.g. “de”) to be used
- TESTNAME: the test name (e.g. “ba22-tt”) to be simulated
- PERIPHERAL: set it to “SoC” if it is using SoC peripherals, else “”

The CONFIG and TESTNAME variables can also be set as arguments to the ncsim.sh script.

The default script variables require that the user should run in the scripts/cadence directory (or a directory parallel to it).

Example usage for ba22-tt testcase, supplying arguments on the command line:

```
cd ba22/scripts/sim/cadence
./ncsim.sh  de  ba22-tt
```

Example usage for ba22-tt testcase, editing the modelsim.tcl:

```
cd ba22/scripts/sim/cadence
# Edit the ncsim.sh and set CONFIG=”de”
# Edit the ncsim.sh and set TESTNAME=”ba22-tt”
./ncsim.sh
```

Note that the ncsim.sh needs to have execute permission. The compilation, elaboration and simulation will be performed. Upon the completion of the simulation, the log files will be created in the working directory as described in section 5.2.3 .

6.2 Synthesis

6.2.1 Altera – quartus.tcl

Quartus.tcl is an example script to synthesize the ba22 core (without memories).

The configuration <user_config> to be set in the quartus.tcl file with the following variable:

- CONFIG : the configuration (e.g. “de”) to be used

The default script variables require that the user should run in the scripts/altera directory (or a directory parallel to it).

Example:

```
cd ba22/scripts/synth/altera
quartus_sh –t ./quartus.tcl
```

6.2.2 Xilinx – ise.tcl

ise.tcl is an example script to synthesize the ba22 core (without memories).

The configuration <user_config> to be set in the quartus.tcl file with the following variable:

- CONFIG : the configuration (e.g. “de”) to be used

The default script variables require that the user should run in the scripts/xilinx directory (or a directory parallel to it).

Example:

```
cd ba22/scripts/synth/xilinx
xtclsh ./ise.tcl
```

6.2.3 Cadence Encounter - rtl_compiler.tcl

The rtl_compiler.tcl is a Cadence RTL_Compiler synthesis script.

The configuration <user_config> to be set in the quartus.tcl file with the following variable:

- CONFIG : the configuration (e.g. “de”) to be used

The default script variables require that the user should run in a directory parallel to the top level src/ directory.

Example:

```
cd ba22/scripts/synth/cadence
rc –files ./rtl_compiler.tcl
```

6.2.4 Synopsys Design Compiler – design_compiler.tcl

The design_compiler.tcl is Synopsys synthesis script.

The configuration <user_config> to be set in the quartus.tcl file with the following variable:

- CONFIG : the configuration (e.g. “de”) to be used

The default script variables require that the user should run in a directory parallel to the top level src/ directory.

Example:

```
cd ba22/scripts/synth/synopsys
dc_shell-xg-t -f ./design_compiler.tcl
```

6.3 Technology Notes

The BA22 is a fully synchronous, positive edge triggered design. It does not contain any latches of internal tri-state buffers.

7. Integration considerations

7.1 Notes On SPR Register Access

Some of the examples demonstrate techniques to write and read SPR registers from C code. This is especially useful for interrupt behavior.

```
unsigned long mfspr(unsigned long add) {
    unsigned long ret;
    asm volatile ("b.mfspr %0,%1,0" : "=r" (ret) : "r" (add));
    return ret;
}

void mtspr(unsigned long add, unsigned long val) {
    asm volatile ("b.mtspr %0,%1,0" : : "r" (add), "r" (val));
}
```

Usage example: read PIC status register (the SPR_PICSR is defined in the example spr_defs.h include file):

```
long result = mfspr(SPR_PICSR);
```

Usage example: write PIC status register (clear to 0)

```
mtspr(SPR_PICSR, 0);
```

7.2 QMEM

7.2.1 Embedded data memory interface

The BA22 processor implements embedded synchronous memory interface (QMEM). The following table lists and describes the IO signals comprising 32-bit data QMEM interface. All signals are synchronous to *clk_i* signal, interface is reset using *rst_i* signal.

Table 7-1: Data QMEM Port List

Signal	Width	Dir	Description
dqmem_stb_o	1	O	Active high memory handshake request signal. It qualifies dqmem_adr_o, dqmem_dat_o, dqmem_sel_o, dqmem_we_o, dqmem_ack_i and dqmem_err_i signals.
dqmem_ack_i	1	I	Active high memory handshake response signal. When dqmem_stb_o and dqmem_ack_i are sampled asserted on the rising clock edge, dqmem_dat_i must be valid on the next rising clock edge.
dqmem_err_i	1	I	Active high memory handshake response signal. Memory subsystem indicates inability to provide data from memory location addressed by dqmem_adr_o.
dqmem_adr_o	32	O	Memory address. Asserted simultaneous with dqmem_stb_o and dqmem_ack_i.
dqmem_dat_i	32	I	Memory data input bus. QMEM must provide data on this bus one rising clock edge after *dqmem_stb_o* and *dqmem_ack_i* were sampled asserted and *dqmem_we_o* was sampled de-asserted.
dqmem_dat_o	32	O	Memory data output bus. Qualifed with simultaneous assertion of *dqmem_stb_o* and *dqmem_we_o*. Each data bus byte additionally qualified with *dqmem_sel_o*. Considered written when *dqmem_ack_i* is also sampled asserted.
dqmem_we_o	1	O	Write enable signal. When *dqmem_stb_o* is asserted, indicates write (1) or

Signal	Width	Dir	Description
			read (0) cycle.
dqmem_sel_o	4	O	Byte select bus. Signals where valid data is placed on *dqmem_dat_o* during write cycles. Signals where valid data should be returned on *dqmem_dat_i* during read cycles.

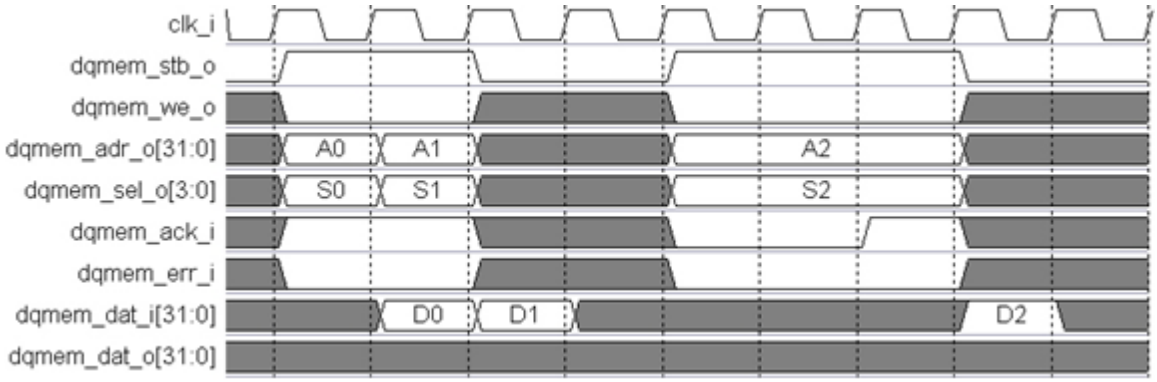

Figure 7-1: Data QMEM Read Cycle

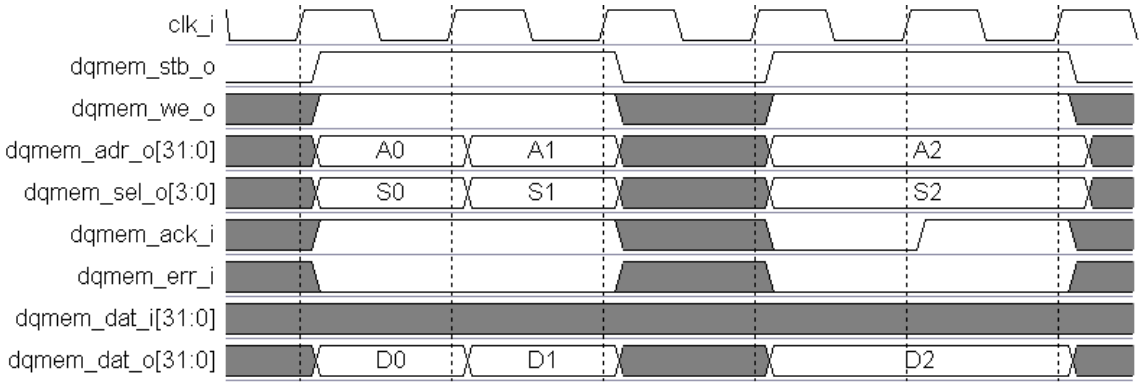

Figure 7-2: Data QMEM Write Cycle

7.2.2 Embedded instruction memory interface

BA22 processors may implement embedded synchronous memory interface (QMEM). The following table lists and describes the IO signals comprising 64-bit instruction QMEM interface. All signals are synchronous to *clk_i* signal, interface is reset using *rst_i* signal.

Table 7-2: Instruction QMEM Port List

Signal	Width	Dir	Description
iqmem_stb_o	1	O	Active high memory handshake request signal. It qualifies dqmem_adr_o, dqmem_dat_o, dqmem_sel_o, dqmem_we_o, dqmem_ack_i and dqmem_err_i signals.
iqmem_ack_i	1	I	Active high memory handshake response signal. When dqmem_stb_o and dqmem_ack_i are sampled asserted on the rising clock edge, dqmem_dat_i must be valid on the next rising clock edge.
iqmem_err_i	1	I	Active high memory handshake response signal. Memory subsystem indicates inability to provide data from memory location addressed by dqmem_adr_o.
iqmem_adr_o	32	O	Memory address. Asserted simultaneous with dqmem_stb_o and dqmem_ack_i.
iqmem_dat_i	64	I	Memory data input bus. QMEM must provide data on this bus one rising clock edge after *dqmem_stb_o* and *dqmem_ack_i* were sampled asserted and

Signal	Width	Dir	Description
			dqmem_we_o was sampled de-asserted.

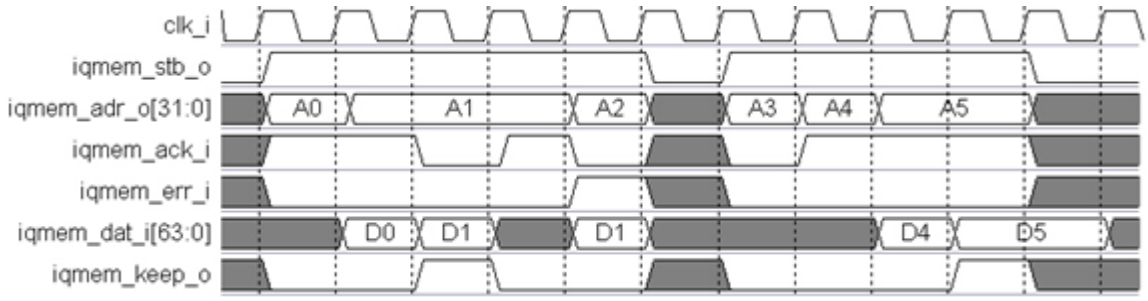

Figure 7-3: Instruction QMEM Read Cycle

7.3 BA22 data cache interface

BA22 processors may implement up to four ways of L0 data cache. Cache modules are to be connected to the BA22 top level module. Signal widths depend on the cache size, cache ways and block size.

Table 7-3: Data Cache Address

31:INDX	INDX-1:BLCK	BLCK-1:2	1:0
TAG	INDEX	BLOCK	BYTE

Data and address bus widths depend on the values of INDX and BLCK. For a given cache size, cache block size and the number of ways (set-associativity) the following formula is used to calculate the INDX and BLCK values:

INDX = $\log_2$(Cache size in bytes) – $\log_2$(number of ways)

BLCK = $\log_2$(block size in bytes)

Here we introduce also *WAYS* as *number of ways* to simplify signal descriptions.

Table 7-4: Data Cache Port List

Signal	Width	Dir	Description
dc_ram_dat_cyc_o	1	O	Active high data ram clock enable signal
dc_ram_dat_we_o	1	O	Write enable signal. When *dc_ram_dat_we_o* is asserted, indicates write (1) or read (0) cycle.
dc_ram_dat_sel_o	*WAYS** 4	O	Active high byte select signal.
dc_ram_dat_adr_o	*INDX*	O	Data ram address. Asserted simultaneous with dc_ram_dat_cyc_o .
dc_ram_dat_dat_o	*WAYS* * 32	O	Data ram data output. Asserted simultaneous with dc_ram_dat_cyc_o . Connect to data in of the data ram.
dc_ram_dat_dat_i	*WAYS* * 32	I	Data ram data input. Connect to data out of the data ram.
dc_ram_tag_cyc_o	1	O	Active high data ram clock enable signal
dc_ram_tag_we_o	1	O	Write enable signal. When *dc_ram_tag_we_o* is asserted, indicates write (1) or read (0) cycle.
dc_ram_tag_sel_o	*WAYS*	O	Active high byte write enable signal.
dc_ram_tag_adr_o	*INDX - BLCK*	O	Tag ram address. Asserted simultaneous with

Signal	Width	Dir	Description
			dc_ram_tag_cyc_o.
dc_ram_tag_dat_o	*WAYS* * 32	O	Tag ram data output. Asserted simultaneous with dc_ram_tag_cyc_o. Connect to data in of the tag ram.
dc_ram_tag_dat_i	*WAYS * (32 - INDX)*	I	Tag ram data input. Connect to data out of the tag ram.
dc_ram_lru_wcyc_o	1	O	LRU ram write cycle
dc_ram_lru_rcyc_o	1	O	LRU ram read cycle
dc_ram_lru_wadr_o	*INDX - BLCK*	O	LRU ram write address
dc_ram_lru_radr_o	INDX - BLCK	O	LRU ram read address
dc_ram_lru_dat_o	3 when WAYS is 4 1 when WAYS is 2 not used when WAYS = 1	O	LRU ram data out. Connect to LRU ram data in.
dc_ram_lru_dat_i	3 when WAYS is 4 1 when WAYS is 2 not used when WAYS is 1	I	LRU ram data in. Connect to LRU ram data out.

BA22 data cache consists of DATA, TAG and LRU rams. DATA ram is byte organized and must implement WAYS * 4 byte write enable support. All reads to data ram are 4 bytes wide, therefore lower 2 bits of the dc_dat_ram_adr_o are not used and must not be connected. If write enable feature is not available, 4 rams for each way should be used. TAG ram is *(32 – INDX)* organized and must implement *WAYS* write enable support. If write enable feature is not available, *WAYS* rams should be used instead. LRU rams are dual port rams, organized as 3 bit words when 4 ways are implemented, or 1 bit when 2 ways are implemented.

Table 7-5: Data Cache memory Size

RAM	Data width	Address width
DATA ram	WAYS*32 bits	INDX+BLCK
TAG ram	WAYS*(32-INDX)	INDX
LRU ram	*WAYS*{3,1}*	INDX

7.4 BA22 Instruction cache interface

BA22 processors may implement up to four ways of L0 instruction cache. Cache modules are to be connected to the BA22 top level module. Signal widths depend on the cache size, cache ways and block size.

Table 7-6: Instruction Cache Address

31:INDX	INDX-1:BLCK	BLCK-1:2	1:0
TAG	INDEX	BLOCK	BYTE

Data and address bus widths are expressed with the values of INDX and BLCK. For a given cache size, cache block size and the number of ways (set-associativity) the following formula is used to calculate the INDX and BLCK values:

$INDX = \log_2(\text{Cache size in bytes}) - \log_2(\text{number of ways})$

$BLCK = \log_2(\text{block size in bytes})$

Here we introduce also *WAYS* as *number of ways* to simplify signal descriptions.

Table 7-7: Instruction Cache Port List

Signal	Width	Dir	Description
ic_ram_dat_cyc_o	1	O	Active high data ram clock enable signal
ic_ram_dat_we_o	1	O	Write enable signal. When i*c_ram_dat_we_o* is asserted, indicates write (1) or read (0) cycle.
ic_ram_dat_sel_o	*WAYS*	O	Active high way select signal.
ic_ram_dat_adr_o	*INDX-2*	O	Data ram address. Asserted simultaneous with ic_ram_dat_cyc_o.
ic_ram_dat_dat_o	*WAYS* * 32	O	Data ram data output. Asserted simultaneous with ic_ram_dat_cyc_o. Connect to data in of the data ram.
ic_ram_dat_dat_i	*WAYS* * 32	I	Data ram data input. Connect to data out of the data ram.
ic_ram_tag_cyc_o	1	O	Active high data ram clock enable signal
ic_ram_tag_we_o	1	O	Write enable signal. When i*c_ram_tag_we_o* is asserted, indicates write (1) or read (0) cycle.
ic_ram_tag_sel_o	*WAYS*	O	Active high byte write enable signal.
ic_ram_tag_adr_o	*INDX – BLCK*	O	Tag ram address. Asserted simultaneous with ic_ram_tag_cyc_o.
ic_ram_tag_dat_o	*WAYS* * 32	O	Tag ram data output. Asserted simultaneous with ic_ram_tag_cyc_o. Connect to data in of the tag ram.
ic_ram_tag_dat_i	*WAYS * (32 - INDX)*	I	Tag ram data input. Connect to data out of the tag ram.
ic_ram_lru_wcyc_o	1	O	LRU ram write cycle
ic_ram_lru_rcyc_o	1	O	LRU ram read cycle
ic_ram_lru_wadr_o	*INDX – BLCK*	O	LRU ram write address
ic_ram_lru_radr_o	INDX – BLCK	O	LRU ram read address
ic_ram_lru_dat_o	3 when WAYS is 4 1 when WAYS is 2 not used when WAYS = 1	O	LRU ram data out. Connect to LRU ram data in.
ic_ram_lru_dat_i	3 when WAYS is 4 1 when WAYS is 2 not used when WAYS is 1	I	LRU ram data in. Connect to LRU ram data out.

BA22 instruction cache consists of DATA, TAG and LRU rams. DATA ram is word organized and must implement WAYS * 2 word write enable support. All reads to data ram are 8 bytes wide, therefore lowest bit of the ic_ram_dat_adr_o is not used and must not be connected. If write enable feature is not available, 2 rams for each way should be used. TAG ram is *(32 – INDX)* organized and must implement *WAYS* write enable support. If write enable feature is not available, *WAYS* rams should be used instead. LRU rams are dual port rams, organized as 3 bit words when 4 ways are implemented, or 1 bit when 2 ways are implemented.

Table 7-8: Instruction Cache Size

RAM	Data width	Address width
DATA ram	WAYS*64 bits	INDX+BLCK-1
TAG ram	WAYS*(32-INDX)	INDX
LRU ram	*WAYS*{3,1}*	INDX

7.5 BA22 Data Memory management interface

BA22 processors may implement up to four ways of data memory management unit translation lookaside buffers (DTLBs). DTLBs are to be connected to the BA22 top level module. Signal widths depend on the DTLB size and DTLB ways.

Table 7-9: Data MMU Virtual Page Number

18:INDX	INDX-1:0
TAG	INDEX

Address bus widths are expressed with the values INDX. For a given number of DTLB entries and the number of ways (set-associativity) the following formula is used to calculate the INDX value:

INDX = log2(DTLB entries) – log2(number of ways)

DTLB_ENTRY_WIDTH is given as a sum of width of the implemented bit / functionalities in DTLB/DMMU.

Table 7-10: Data MMU Bit-Field Widths

Register	Bit	Width	Implemented
DTLB Translation register	CC	1	NO
	CI	1	YES
	WBC	1	NO
	WOM	1	NO
	A	1	NO
	D	1	NO
	URE	1	YES
	UWE	1	YES
	SRE	1	YES
	SWE	1	YES
	PPN	19	YES
DTLB match register	V	1	YES
	CID	4	NO
	VPN	19	YES

We introduce *WAYS* as *number of ways* to simplify signal descriptions.

Table 7-11: Data MMU Port List

Signal	Width	Dir	Description
dtlb_ram_dat_cyc_o	1	O	Active high data ram clock enable signal
dtlb_ram_dat_we_o	1	O	Write enable signal. When dtlb_*ram_dat_we_o* is asserted, indicates write (1) or read (0) cycle.
dtlb_ram_dat_sel_o	*WAYS*	O	Active high way select signal.
dtlb_ram_dat_adr_o	*INDX*	O	Data ram address. Asserted simultaneous with dtlb_ram_dat_cyc_o .
dtlb_ram_dat_dat_o	*WAYS * DTLB_ENTRY_WIDTH*	O	Data ram data output. Asserted simultaneous with dtlb_ram_dat_cyc_o . Connect to data in of the data ram.
dtlb_ram_dat_dat_i	*WAYS * DTLB_ENTRY_WIDTH*	I	Data ram data input. Connect to data out of the data ram.
dtlb_ram_lru_wcyc_o	1	O	LRU ram write cycle
dtlb_ram_lru_rcyc_o	1	O	LRU ram read cycle
dtlb_ram_lru_wadr_o	*INDX*	O	LRU ram write address
dtlb_ram_lru_radr_o	INDX	O	LRU ram read address
dtlb_ram_lru_dat_o	3 when WAYS is 4 1 when WAYS is 2 not used when WAYS = 1	O	LRU ram data out. Connect to LRU ram data in.
dtlb_ram_lru_dat_i	3 when WAYS is 4 1 when WAYS is 2 not used when WAYS is 1	I	LRU ram data in. Connect to LRU ram data out.

BA22 DTLBs consist of DATA and LRU rams. DATA ram is *DTLB_ENTRY_WIDTH * WAYS* organized and must implement WAYS *DTLB_ENTRY_WIDTH* write enable support. All reads to data ram are *DTLB_ENTRY_WIDTH wide*. If write enable feature is not available, 4 rams, one for each way, should be used. LRU rams are dual port rams, organized as 3 bit words when 4 ways are implemented, or 1 bit when 2 ways are implemented.

Table 7-12: Data MMU Memory Size

RAM	Data width	Address width
DATA ram	WAYS**DTLB_ENTRY_WIDTH* bits	INDX
LRU ram	*WAYS*{3,1}*	INDX

7.6 BA22 Instruction Memory management interface

BA22 processors may implement up to four ways of instruction memory management unit translation lookaside buffers (ITLBs). ITLBs are to be connected to the BA22 top level module. Signal widths depend on the ITLB size and ITLB ways.

Table 7-13: Instruction MMU Virtual Page Number

31:INDX	INDX-1:0
TAG	INDEX

Address bus widths are expressed with the value INDX. For a given number of ITLB entries and the number of ways (set-associativity) the following formula is used to calculate the INDX value:

INDX = $\log_2$(ITLB entries) – $\log_2$(number of ways)

ITLB_ENTRY_WIDTH is given as a sum of width of the implemented bit / functionalities in ITLB/IMMU.

Table 7-14: Instruction MMU Bit Field Widths

Register	Bit	Width	Enabled
ITLB Translation register	CC	1	NO
	CI	1	YES
	WBC	1	NO
	WOM	1	NO
	A	1	NO
	D	1	NO
	UXE	1	YES
	SXE	1	YES
	PPN	19	YES
ITLB match register	V	1	YES
	CID	4	NO
	VPN	19	YES

We introduce *WAYS* as *number of ways* to simplify signal descriptions.

Table 7-15: Instruction MMU Port List

Signal	Width	Dir	Description
itlb_ram_dat_cyc_o	1	O	Active high data ram clock enable signal
itlb_ram_dat_we_o	1	O	Write enable signal. When itlb_*ram_dat_we_o* is asserted, indicates write (1) or read (0) cycle.
itlb_ram_dat_sel_o	*WAYS*	O	Active high way select signal.
itlb_ram_dat_adr_o	*INDX*	O	Data ram address. Asserted simultaneous with itlb_ram_dat_cyc_o .
itlb_ram_dat_dat_o	*WAYS * DTLB_ENTRY_WIDTH*	O	Data ram data output. Asserted simultaneous with itlb_ram_dat_cyc_o . Connect to data in of the data ram.
itlb_ram_dat_dat_i	*WAYS * DTLB_ENTRY_WIDTH*	I	Data ram data input. Connect to data out of the data ram.
itlb_ram_lru_wcyc_o	1	O	LRU ram write cycle
itlb_ram_lru_rcyc_o	1	O	LRU ram read cycle
itlb_ram_lru_wadr_o	*INDX*	O	LRU ram write address
itlb_ram_lru_radr_o	INDX	O	LRU ram read address
itlb_ram_lru_dat_o	3 when WAYS is 4 1 when WAYS is 2 not used when WAYS = 1	O	LRU ram data out. Connect to LRU ram data in.

Signal	Width	Dir	Description
itlb_ram_lru_dat_i	3 when WAYS is 4 1 when WAYS is 2 not used when WAYS is 1	I	LRU ram data in. Connect to LRU ram data out.

BA22 DTLBs consist of DATA and LRU rams. DATA ram is *DTLB_ENTRY_WIDTH * WAYS* organized and must implement WAYS *DTLB_ENTRY_WIDTH* write enable support. All reads to data ram are *DTLB_ENTRY_WIDTH wide*. If write enable feature is not available, 4 rams, one for each way, should be used. LRU rams are dual port rams, organized as 3 bit words when 4 ways are implemented, or 1 bit when 2 ways are implemented.

Table 7-16: Instruction MMU Memory Size

RAM	Data width	Address width
DATA ram	WAYS**DTLB_ENTRY_WIDTH* bits	INDX
LRU ram	*WAYS*{3,1}*	INDX

8. Support

Every effort has been made to ensure that this core functions correctly. If a problem is encountered, contact:

CAST, Inc.

11 Stonewall Court

Woodcliff Lake, New Jersey 07677 USA

Technical Support Hotline: +1-201-391-8300 ext. 2

Fax: +1-201-833-2682

E-mail: support@cast-inc.com

URL: www.cast-inc.com