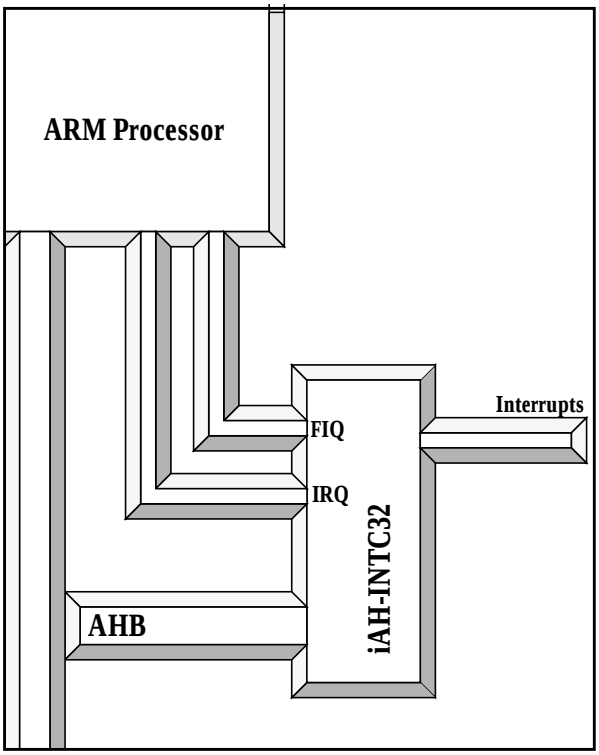
iAH-INTC32 data sheet

AMBA (AHB) compliant

Features:

- AMBA (AHB) compliant Interface
- Zero waitstate interface
- 32 fully programmable interrupt sources
- 32bit vector for each interrupt
- Programmable priority for each interrupt with round robin option
- Various static and event-type interrupts
- Nested interrupts with masking of lower priority interrupts for RTOS support
- Optional 'End of Interrupt' register
- Prepared to be expanded to 64 interrupts
- Designed for test, ready for SCAN
- Structured, synchronous VHDL coding
- Available on evaluation platform
- Utility library in C



INICORE - the reliable Core and System Provider.
We provide high quality IP, design expertise and leading edge silicon to the industry.

The iAH-INTC is an AMBA (AHB) compliant, fully programmable interrupt controller with 32 interrupt sources. A 32bit vector can be assigned to each interrupt. Priority can be fixed or round robin, where the fixed priority configuration locks interrupts during the service of a higher priority interrupt.

Any interrupt source can have any priority and it can be mapped either to IRQ or FIQ. Interrupt signals can be configured as static hi/low, event and toggle.

An optional 'end of interrupt' procedure is used in case of nested interrupts, which makes it ideal for real time operating systems.

The connection to the AHB interface makes the interrupt controller very fast, eliminating the slow accesses over the APB bridge in ARM / AMBA systems.

As every product of INICORE, our core is prepared for SCAN test.

The iAH-INTC is delivered in VHDL source code, with test bench, documentation and our C utility library for embedding and immediate start.

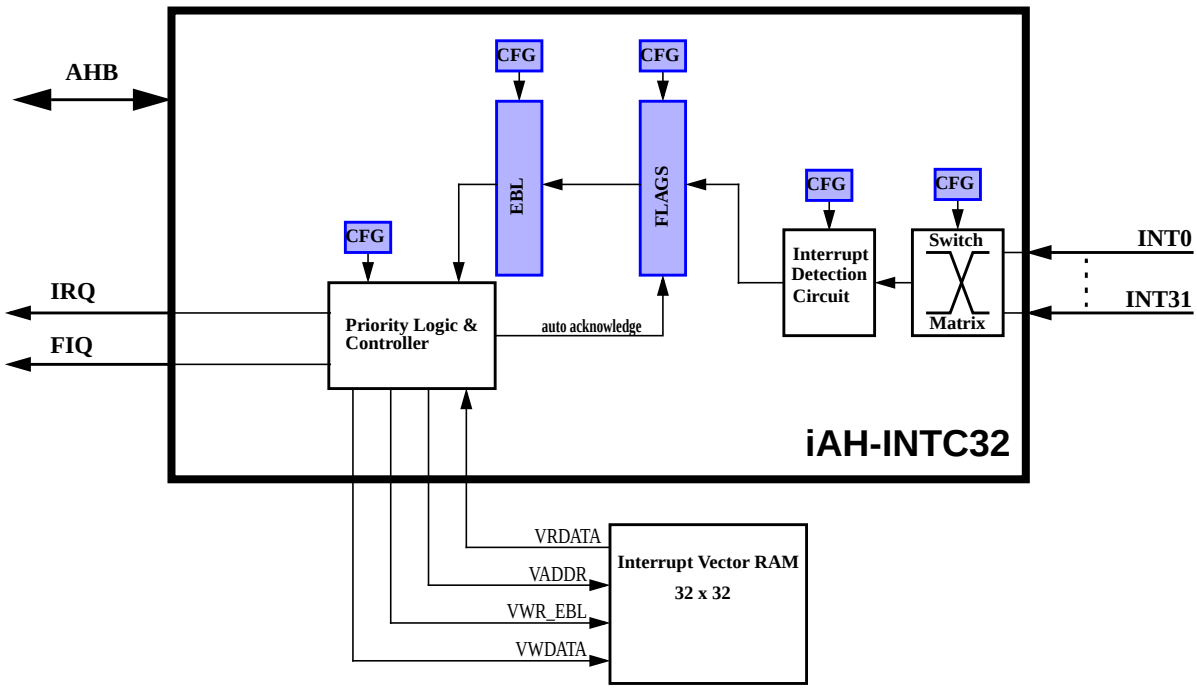
INICORE Inc. (USA)
5600 Mowry School Road, Suite 180,
Newark, CA 94560
Tel: 510 445 1529 Fax: 510 656 0995
E-mail: ask_us@inicore.com
Web: www.inicore.com

INICORE AG (Europe)
Mattenstrasse 6a,CH-2555 Brügg, Switzerland
Tel: ++41 32 374 32 00, Fax: ++41 32 374 32 01
E-mail: ask_us@inicore.ch
Web: www.inicore.ch

Ref.Nr.: V1.0 6/8/01

1 Overview

This pictures gives a short overview of the iAH-INTC32.



The iAH-INTC32 is principally bus connected. Other signals are the interrupt signals themselves, which go directly to the processor. They are active low. The 32 interrupt inputs (sources) are placed on the right hand. First, they are reordered in the switch matrix according to the configuration of the interrupt assignment. Then, the signals are synchronized and re-organized as configured (edge detection, etc.)
Active interrupts are stored in the flags registers and then gated with the interrupt enable register. The resulting signals are fed to the priority logic and the interrupt controller itself, which does the mapping of IRQ or FIQ interrupts and assigns the corresponding interrupt vector from the vector memory to the highest priority interrupt which is pending.

2 IO description

The following part lists the input and output ports of the iAH-INTC32 core and gives a short overview of their functionality.

These pins are used to clock and initialize the whole iAH-INTC32 core. There are no other clocks in this core.

pin name	type	size	description
hclk	in	1	AHB clock, the only clock source for the controller
reset_n	in	1	asynchronous system reset, active low
scan_mode	in	1	test pin for scan mode, must be ‘0’ for normal operation

Interrupt pins:

pin name	type	size	description
int_sources [31:0]	in	32	interrupt sources, int(0) has highest priority int(31) has lowest (without re-ordering). These signals are synchronized internally.
fiq_n	out	1	Fast interrupt request to processor, active low
irq_n	out	1	Interrupt request to processor, active low

These pins are used to interface the interrupt vector RAM.

pin name	type	size	description
vrdata	in	32	output of vector RAM (vector of to vector number)
vwr_ebl	out	4	write enable signals to vector RAM (byte enable)
vaddr	out	5	address signal for vector RAM (vector number)
vwdata	out	32	write data to vector RAM (for initialization)

The AHB bus connection:

pin name	type	size	description
hsel	in	1	Select signal for interrupt controller
hwrite	in	1	Write signal
htrans	in	2	Transaction control signals
hsize	in	3	Transaction size signals
haddr	in	32	Address; bits [31:8] ignored
hwdata	in	32	Write data (seen from processor)
hrdata	out	32	Read data (seen from processor)
hready	out	1	Ready signal
hresp	out	2	Slave response (always okay)

3 Register Description

This section explains the use of each register and block within the interrupt controller.

3.1 Register Map

The following table lists all registers in the iAH-INTC32.

Register	Size	Access	Description
INT_IRQ_VEC	32	R(W)[1]	IRQ interrupt vector: Read: Fetch IRQ interrupt vector Write: Write Interrupt vector into vector memory
INT_FIQ_VEC	32	R	FIQ interrupt vector: Read: Fetch FIQ interrupt vector
INT_IRQ_EOI	0	W	IRQ end of interrupt: write to this address to mark an IRQ end of interrupt
INT_FIQ_EOI	0	W	FIQ end of interrupt: write to this address to mark an FIQ end of interrupt
INT_IRQ_VEC_NBR	5	R(W)	IRQ interrupt number (level): Read: IRQ int number of last fetched IRQ vector Write: Set address for vector memory programming
INT_FIQ_VEC_NBR	5	R	FIQ interrupt number (level) Read: IRQ int number of last fetched IRQ vector
INT_IRQ_NEST	5	R	Number of nested IRQ interrupts
INT_FIQ_NEST	5	R	Number of nested FIQ interrupts
INT_IRQ_IN_WORK	32	R	IRQ in work ‘0’: interrupt not being processed ‘1’: corresponding interrupt vector has been fetched but no end of interrupt has been marked for this interrupt
INT_FIQ_IN_WORK	32	R	FIQ in work
INT_MAP	32	R/W	Interrupt mapping register: ‘0’: map source to nIRQ ‘1’: map source to nFIQ
INT_RAW_STATUS	32	R/W	Raw interrupt status register: ‘0’: no interrupt occurred ‘1’: interrupt detected
INT_STATUS	32	R/W	Status register of enabled interrupts: ‘0’: no interrupt occurred ‘1’: interrupt detected
INT_EBL	32	R/W	Interrupt enable register: ‘0’: source disabled ‘1’: source enabled
INT_EBL_SET	32	W	Interrupt enable set register: ‘0’: no action ‘1’: set corresponding enable bit

Register	Size	Access	Description
INT_EBL_CLR	32	W	Interrupt enable clear register: ‘0’: no action ‘1’: clear corresponding enable bit
INT_CFG	[0]	R/W	Write-ability of vector number registers ‘0’: vector number registers not writable ‘1’: vector number registers writable
INT_MODE1	32	R/W	Interrupts mode register 1: ‘0’: interrupt defined as static ‘1’: interrupt define as event
INT_MODE2	32	R/W	Interrupts mode register 2: ‘0’: static: active 0; event: toggle interrupt ‘1’: static: active 1; event: 1 clock pulse interrupt
INT_ASSIGN	32	R/W	Interrupt assignment register: [15:8]: Interrupt priority number is assigned to interrupt number [7:0]

1. (W) means only writable when INT_CFG(0) = ‘1’

4 Programming Model

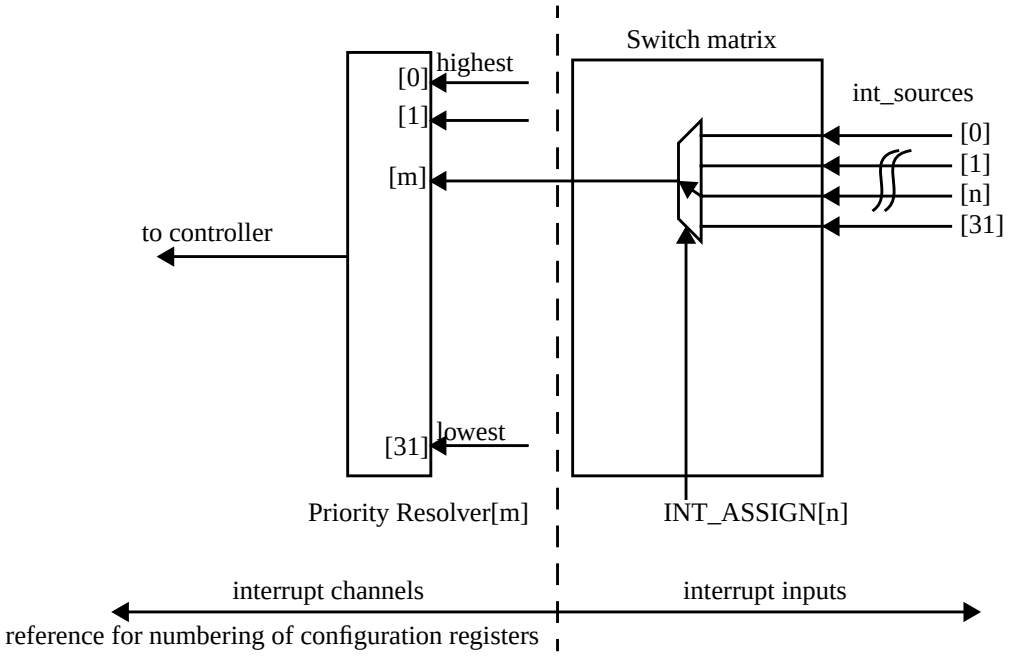
This section describes the configuration and programming of the iAH-INTC32.

4.1 Configuration

The interrupt controller needs to be configured first. Until complete configuration, the interrupt enable of the processor should not be enabled. Following steps are needed:

4.1.1 Assigning of interrupt vector inputs

The interrupt vector inputs can be assigned to any of the interrupt controller inputs, which allows a software controlled configuration of each interrupt priority. The scheme works as follow:



On the right side, we talk about interrupt inputs (before the switch matrix); on the left hand, we talk about interrupt channels or priority (after the switch matrix). All further interrupt configurations are related to the left hand side (to the interrupt channels or priority).

Every input of the priority resolver (called ‘m’) has an entry ‘n’ in the INT_ASSIGN register file, which defines which int_sources(n) is connected to it’s input ‘m’.

Example: to assign the int_sources(20) to the priority (1), write the value 0x1401 to the INT_ASSIGN register file. This will store the value 0x01 into the register file entry 0x14, which will route then the input 20 (=0x14) to the priority 1.

Please note: It is obvious that different priority inputs can be routed to the same int_sources signals, which does not make sense. The assignments should be exclusive.

4.1.2 Interrupt types

The iAH-INTC32 supports 4 types of interrupt inputs (at int_sources), which are separated in to major categories. These features are programmed in registers INT_MODE 1 and 2.

INT_MODE1 defines whether the interrupt signal is of static or event type. The settings are individual for each interrupt input.

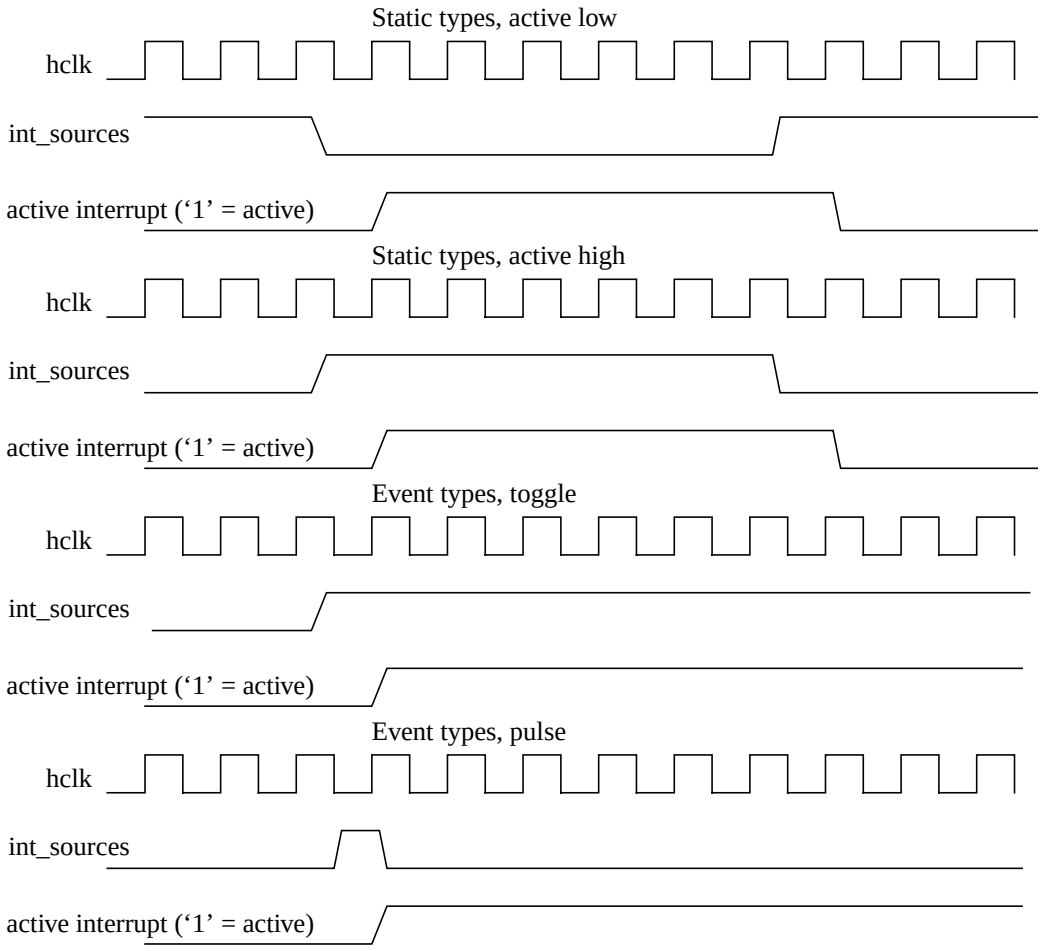
Static types – With INT_MODE1[n] = ‘0’, the static interrupt type for interrupt ‘n’ is selected. The interrupting devices has it’s own interrupt flag which has to be cleared by the processor in the interrupt handler. Typical applications are peripherals with FIFOs, where the interrupt is active as long as there is data available in the FIFO. With INT_MODE2[n], the level of these interrupts can be selected (active high or low).

Static interrupts are insensitive against glitches etc. but they should not disappear without proper acknowledgement of the interrupts in the controller, which would disturb nested interrupt schemes.

Event types – With INT_MODE[n] = ‘1’, event type interrupts are selected. These interrupts signal actions, like unbuffered peripherals (“data received”), and are used to set the iAH-INTC32-internal interrupt flags when they occur. Event types can be selected as toggle interrupts when INT_MODE2[n] is set to ‘0’. This means that every edge on the interrupt source is interpreted as a valid interrupt. Active high pulses are selected when INT_MODE2[n] is set to ‘1’ (for one HCLK cycle).

Please note: Event type interrupt inputs must be glitch-free. Event pulses must be synchronized to the HCLK clock domain.

See the following picture for better comprehension:



4.1.3 Interrupt mapping

The INT_MAP register defines which interrupt channel is mapped to the IRQ (set to ‘0’) and which channel is mapped to the FIQ (set to ‘1’). Every interrupt channel can be mapped to the same IRQ or FIQ output, or any combination is possible. With ARM pro-

cessors, FIQ has higher priority than IRQ, which is not of importance inside the iAH-INTC32.

4.1.4 Interrupt Enable

The INT_EBL register is used to mask out temporarily or in general certain interrupt channels. A ‘1’ in the corresponding bit of INT_EBL activates the interrupt channel, where a ‘0’ does mask it out. Nevertheless, the status of a de-activated interrupt channel is still visible by reading the INT_RAW_STATUS register. This allows the software to select between polling and interrupt driven schemes without changing the hardware.

4.1.5 Vector initialization

The vector memory holds a 32bit address for every interrupt channel. This is in general a read-only operation, but for configuration, it has to be written. This is done by setting the INT_CFG bit to ‘1’, which makes the vector memory writable.

The write process takes place in 2 steps: First, the number of the interrupt channel to be configured is written into the INT_IRQ_VEC_NBR. Then, the corresponding vector is written into register INT_IRQ_VEC, which performs the write transfer to the vector memory. These steps have to be repeated for every interrupt channel. To bring the interrupt controller back into normal mode, set the INT_CFG register back to ‘0’. The mapping to IRQ or FIQ does not matter since the mapping is defined by the INT_MAP register, not by the number of the interrupt channel. The address stored in the vector memory is the same for IRQ and FIQ mapping.

Please note: Always disable CPU interrupts when configuring the interrupt controller.

4.2 Interrupt Handling

4.2.1 Non nested interrupts

Interrupt handlers which do rely on the fact, that every interrupt is served without further interruptions by higher priority interrupts. In this case, the ‘in work’ registers do not have to be used. After configuration of the interrupt controller, the interrupt enable flags of the processor can be activated. For a typical interrupt, the procedure would look as follow:

1. The interrupt occurs and is seen in the INT_FLAG register
2. IRQ asserted (depending on the configuration)
3. The processor fetches the vector from INT_IRQ_VEC. This will clear the INT_FLAG in case of event type interrupts. The interrupt number is stored in INT_IRQ_VEC_NBR register.
4. The processor jumps to the address read before into the interrupt handler
5. The interrupt is served and the CPU returns to the main application.

The ‘in work’ and ‘end of interrupt’ registers are not used in non-nested applications. The interrupt priority is only relevant when more than one interrupts are activated concurrently. Normally, it’s a ‘first come first served’ scheme.

4.2.2 Nested interrupts

Especially real time systems have to take care of the interrupt priority. When an interrupt is served, all lower priority interrupts are masked out and the CPU will re-enable the interrupt enable flag. Therefore, higher priority interrupts can interrupt the pending and actually served one. To allow such a scheme, the ‘in work’ and ‘end of interrupt’ registers have to be used. The process works as follow:

1. The interrupt occurs and is seen in the INT_FLAG register
2. IRQ asserted (depending on the configuration)
3. The processor fetches the vector from INT_IRQ_VEC. This will clear the INT_FLAG flag in case of event type interrupts. The interrupt number is stored in INT_IRQ_VEC_NBR register. In the mean time, the INT_IRQ_IN_WORK bit is set and marks that this interrupt is actually served. All lower priority interrupts are masked out and not visible to the CPU.
4. The CPU stacks the actual status and re-enables it's interrupt enable flag.
5. A higher priority interrupt than the actual may occur (proceed at 1.)
6. At the end of the interrupt service routine, the CPU writes to the INT_IRQ_EOI a dummy value (only the write access is detected, the write data is not relevant), which will clear the pending 'in work' flag and mark the end of the interrupt service routine. This will also re-enable the lower priority interrupts, respectively set the mask to the next lower pending interrupt level if nested interrupts are pending.

The level of nesting can be checked in the INT_IRQ_NEST register. In case of a processor crash, the interrupt controller can be re-initialized by writing maximum 32times to the 'end of interrupt' registers, which will clear all pending interrupts.

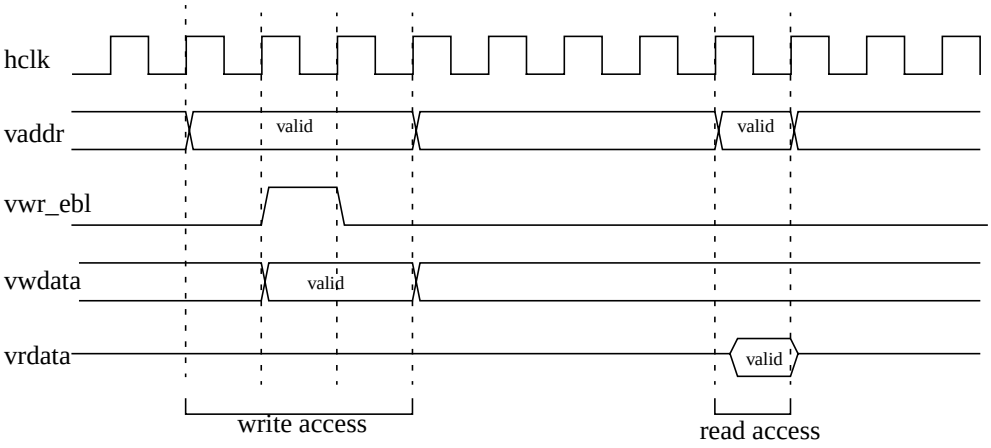
Please note that static interrupts have to be de-asserted before the 'end of interrupt' command.

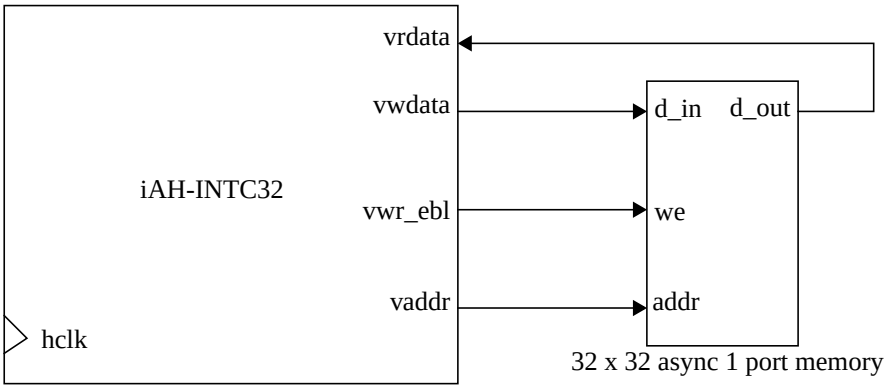
5 Implementation Guidelines

This section describes how the iAH-INTC32 is embedded into a system.

5.1 Memory Connection

The vector memory is a 32 by 32 bit asynchronous single port memory. It is available in any ASIC library. The connection is not critical, since the write access takes place over 3 cycles, where setup and hold times are guaranteed. The read cycle takes place in a combinatorial way within one cycle. The address is directly derived from FF's. The following timing diagram shows the write and read cycles and how the memory has to be connected:





5.2 AHB Connection

The AHB connection uses only a subset of all AHB signals. The remaining signals can left open. The iAH-INTC32 does not make use of error or split/retry answering features. Although the haddr is 32bits wide, only bits [7:2] are used. The configuration registers do not support byte or halfword accesses. Such accesses are ignored.

5.3 ASIC Integration

The design is prepared for SCAN test. When the scan_mode signal is activated, the asynchronous memory is put into a transparent mode. There is no asynchronous logic. The VHDL can be synthesized to any technology, including FPGA. No special timing constraints for inputs are needed. The readback path (hrdata) should be constrained when the AHB is clocked with high speed.

6 Register Map

This table shows the (byte) address offset for the registers.

Address	Register
0x00	INT_IRQ_VEC
0x04	INT_FIQ_VEC
0x08	INT_IRQ_EOI
0x0c	INT_FIQ_EOI
0x10	INT_IRQ_VEC_NBR
0x14	INT_FIQ_VEC_NBR
0x18	INT_IRQ_NEST
0x1c	INT_FIQ_NEST
0x20	INT_IRQ_IN_WORK
0x24	reserved
0x28	INT_FIQ_IN_WORK
0x2c	reserved
0x30	INT_MAP
0x34	reserved
0x38	INT_RAW_STATUS
0x3c	reserved
0x40	INT_STATUS
0x44	reserved
0x48	INT_EBL
0x4c	reserved
0x50	INT_EBL_SET
0x54	reserved
0x58	INT_EBL_CLR
0x5c	reserved
0x60	INT_CFG
0x64	reserved
0x68	INT_MODE1
0x6c	reserved
0x70	INT_MODE2
0x74	reserved
0x78	INT_ASSIGN