hidICE Evaluation System for SPARC V8 (LEON3)

Document number: UG090220-V2.3

Contact person: Alexander Weiss
Accemic GmbH & Co. KG
aweiss@accemic.com
49 8034 90993-12

Project partners: Accemic GmbH & Co. KG
Hochriesstr. 2
D-83126 Flintsbach
Germany

Dresden University of Technology
Department of Computer Science
Institute for Computer Engineering
D-01062 Dresden
Germany

Munich University of Technology
Institute for Informatics I4
Boltzmannstr. 3
D-85748 Garching
Germany

Contents

1. Introduction

This document is a project draft of a hidICE evaluation system, based on the SPARC V8 LEON3 core.
The hidICE evaluation system hardware consists of two Xilinx Virtex 5 ML 507 boards.
The following objectives are planned for this project

- General demonstration of hidICE synchronization on a state of the art 32 bit SoC system (synchronization via APB and AHB)
- Multi-core implementation
- On-chip debug support
- Real-time trace analysis
- hidICE supported software self tests
- Pin count optimization / Port reconstruction
- NEXUS class 3 interface

This document contains text passages sourced from the "GRLIB IP Core User's Manual". These passages are marked in *italic*.

Please note this project is under development and subject to change.

1.1. Introduction in the hidICE technology

The fundamental idea behind the new methodology for capturing trace data is to equip the microcontroller with a facility that allows the synchronization with an external emulator. Thus, the only data communicated from the SoC to the emulator is the one required to reconstruct all internal data and the program flow. All other system responses are defined by the program code. Fig. 1-1 shows the emulation principle.
Conventional methods transfer all read data and write data as well as all jumps to the emulator. Whereas our proposed methodology only transfers all read data that originates from the peripheral components of the controller and the interrupts, because only this data/these events can change the program flow unpredictably.

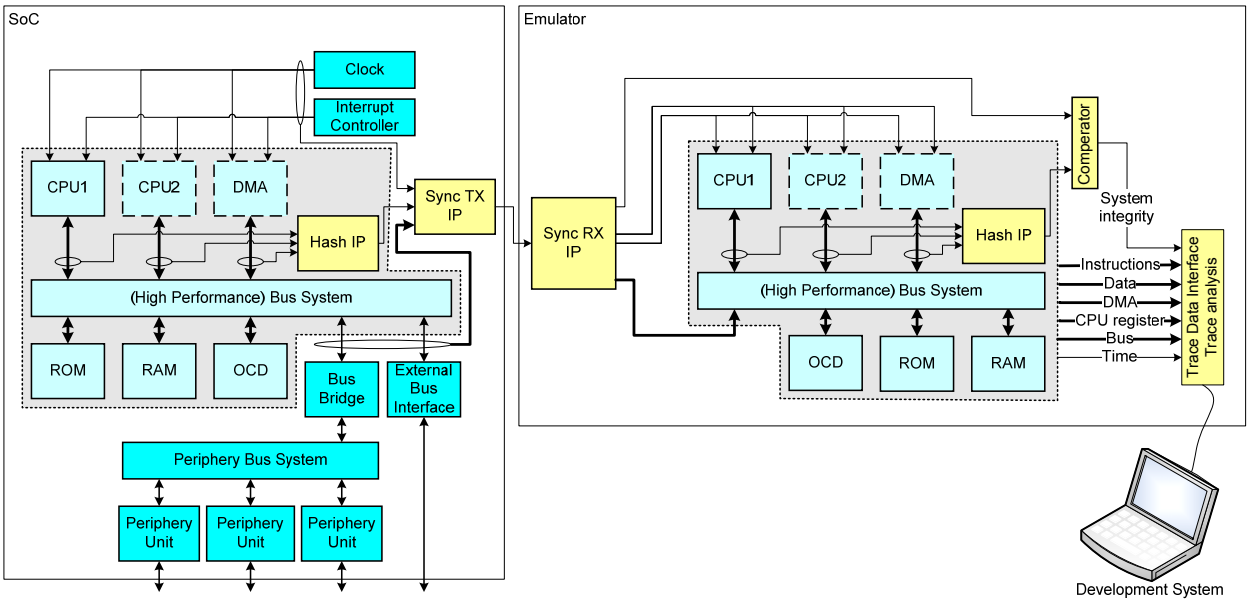


Figure 1-1: principle. All related clocks, all data read from the periphery and all interrupts are transferred to the emulation. One or more hash values of instructions, addresses and data can be transferred to the emulation.

The hidICE principle is not only applicable for microcontrollers with a single CPU without DMA, but also for DMA equipped and multi-core microcontrollers. Identical hash values of both systems guarantee the validity of the captured trace data. As the emulation does not influence the system itself, it is possible to use the emulation concurrently to traditional on-chip debug support.

Therefore, the emulator must meet these requirements:

- The emulator must replicate the microcontroller's bus master cores (one or more CPUs and DMA controller), but none of the peripherals such as ADC, UART or CAN.
- Its RAM memory must be the same size or larger and have the same or faster access times as the SoC.
- The emulator must have ROM of the same size or larger, same or faster access times and the same content as that of the SoC.
- Via the synchronization interface the following signals have to be transmitted from the SoC to the emulator:
  - CPU clock
  - results of the CPU / DMA read operations in the peripheral area
  - interrupt and DMA requests

Given these properties, emulation will precisely match the behaviour and the instructions carried out by the microcontroller being emulated. Both SoC and emulator start with the same internal state. Also, both have the same content of ROM. Thus, the information read from peripheral components is sufficient to exactly reconstruct the RAM content in the emulation, since writes to the local RAM are reflected in the emulation.
A read operation to non-initialized RAM addresses is not allowed, due to identical system behaviour not being guaranteed in case of different RAM content. The system integrity control discussed below will detect and signalize such a difference.

All branch decisions will be the same in the SoC and the emulator. The only remaining changes of program flow which can not be predicted in the emulator are interrupts or DMA requests, which are communicated to the emulation to replicate the exact behaviour of the SoC.
Depending on complexity and speed, the emulation can be run in an FPGA for slower CPUs or in an ASIC for faster CPUs. The ASIC / FPGA emulation must include the CPU core(s), the DMA controller and the internal memory.
In difference to the traditional evaluation chips, only one implementation of the emulation core is required for each CPU series. A new evaluation chip for each new device with new or different periphery is no longer required, since the implementation of peripheral units is not necessary for the emulation. The costs for new evaluation chips, their physical limitations and associated time-to-market delays are no longer encountered. The proposed principle is particularly suited for microcontroller families which have identical cores and varying periphery, as for each new derivative full trace support can be provided immediately at no additional cost.
For slower CPUs (up to ~50 MHz), emulating the CPU in an FPGA lowers tool cost dramatically because only the existing FPGA needs to be reconfigured and one emulator can support different CPU families. Also the emulator logic which analyzes or pre-processes the available trace data can be implemented in the same FPGA. This will provide a very compact and cost efficient emulation system.
In case of an ASIC implementation, the trace data can be made available on a configurable interface. Due to the very high width of the available trace data, it seems reasonable to provide a configurable interface, which provides a subset of the available trace data depending on the current demand. For instance, for a branch / decision code coverage analysis of a CPU the program counter and the data read by the CPU can be made available on the output. For another problem e.g. stack analysis, the stack pointer and program counter may be selected for output. Currently, we are discussing a convenient interface with some major emulator vendors.
Alternatively, the principle can also be used with software emulation. A fast buffer captures the synchronization data and a software emulation of the CPU core computes the executed instructions. Yet, this approach does not work in real time for most applications and the available trace interval is limited by the size of the buffer.

1.1.1. Synchronization Interface

For the implementation of the synchronization support the internal architecture of the SoC has to be analyzed. First we have to extract all signals which are required for the synchronization. In a second step we have to analyze the signal interdependence for further compression of the synchronization information. For instance, in case of starting an interrupt sequence, no I/O operations may be possible on the bus and in this case the pins used to transfer the interrupt number can be shared with the pins which are used to transfer the data read from the CPU. As a third step the system integrity IP has to be implemented. The complexity of this IP is scalable.
A correct synchronization of both systems may be achieved with a simple hash function, whereas a more complex hash may provide information on the problem's cause in case of a system integrity violation.
A generic synchronization protocol is not reasonable, since the signal interdependence often differs from SoC to SoC. Within a group of SoCs with identical core (e.g. a microcontroller family) the synchronization protocol will be identical.

In one architecture it makes sense to capture the sync data very local to the CPU, in another architecture it may be more advantageous to capture the sync data on the bridge from local bus to periphery bus.

The hidICE synchronization interface can be easily implemented as an add-on to existing on-chip debug support modules (OCD), which are available in a wide range of implementations (e.g. Renesas NSD or Freescale BDM). The OCD module is required for initial flash memory programming, setting breakpoints and in some cases for capturing some frames of trace data. An additional implementation of a synchronization facility extends this very limited trace to a continuous and complete real-time trace of executed instructions, data read, data written, DMA operations and CPU register values.

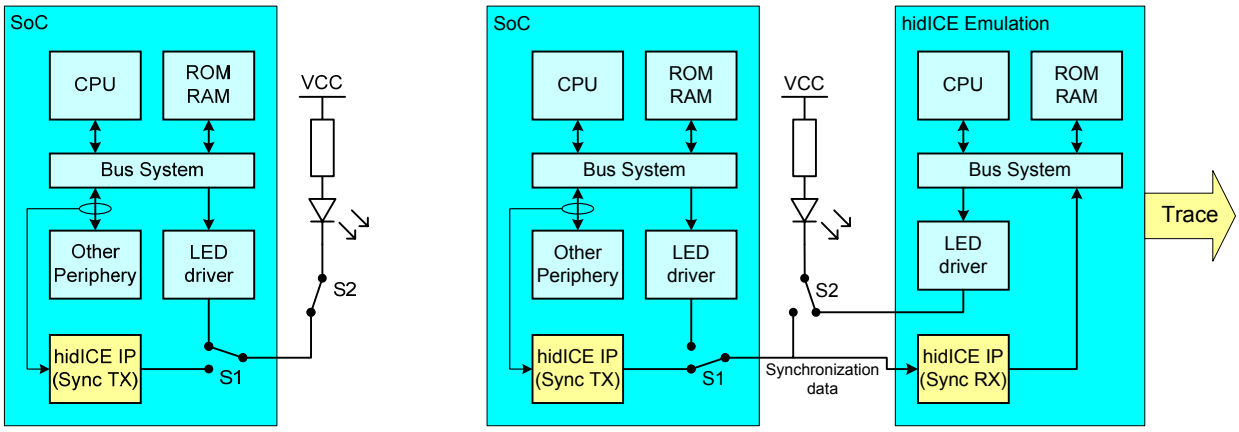


Figure 1-2: Port Reconstruction: If the emulator is connected, the information necessary for synchronization of the CPU cores will be transferred via the same I/O pins, which are normally used for driving the LEDs. When the LEDs are driven by the emulator's LED driver, the LEDs will be turned on or off with a well-defined delay of some CPU clock cycles. From the application's point of view all pins are accessible, even during recording of the trace data.

The developer can assign the pins required to output the synchronization information to simple output functions such as LED driver, pulse generators or LCD drivers. Once the emulator is attached to the microcontroller, it then takes over emulation of the pins which are temporarily used to output trace data. A ribbon cable allows the emulated signals to be transferred back to the target hardware (see Fig. 1-2).

1.1.2. Trace Data Access

Due to the availability of full trace data inside the emulator, it is possible to capture a very detailed trace. Not only the program counter and the data read and written, also the clock cycle synchronous bus control signals are available for further analysis. The capturing of the trace data has no influence to any on-chip debug activities, because the emulation only passively observes the SoC activities. On-chip debug resources and the hidICE IP are completely independent functional units.
The quality of the captured trace data is similar to a state analysis by a logic analyzer, which has access to all internal bus signals of the microcontroller. This enables the deep analysis of bus performance, cache and DMA operation and makes a new generation of in-circuit emulators possible.

Until new in-circuit emulators will be available, a connection to existing interfaces like Nexus can easily be implemented and thus, it extends the applicability of those emulators to new chips that incorporate the synchronization interface.

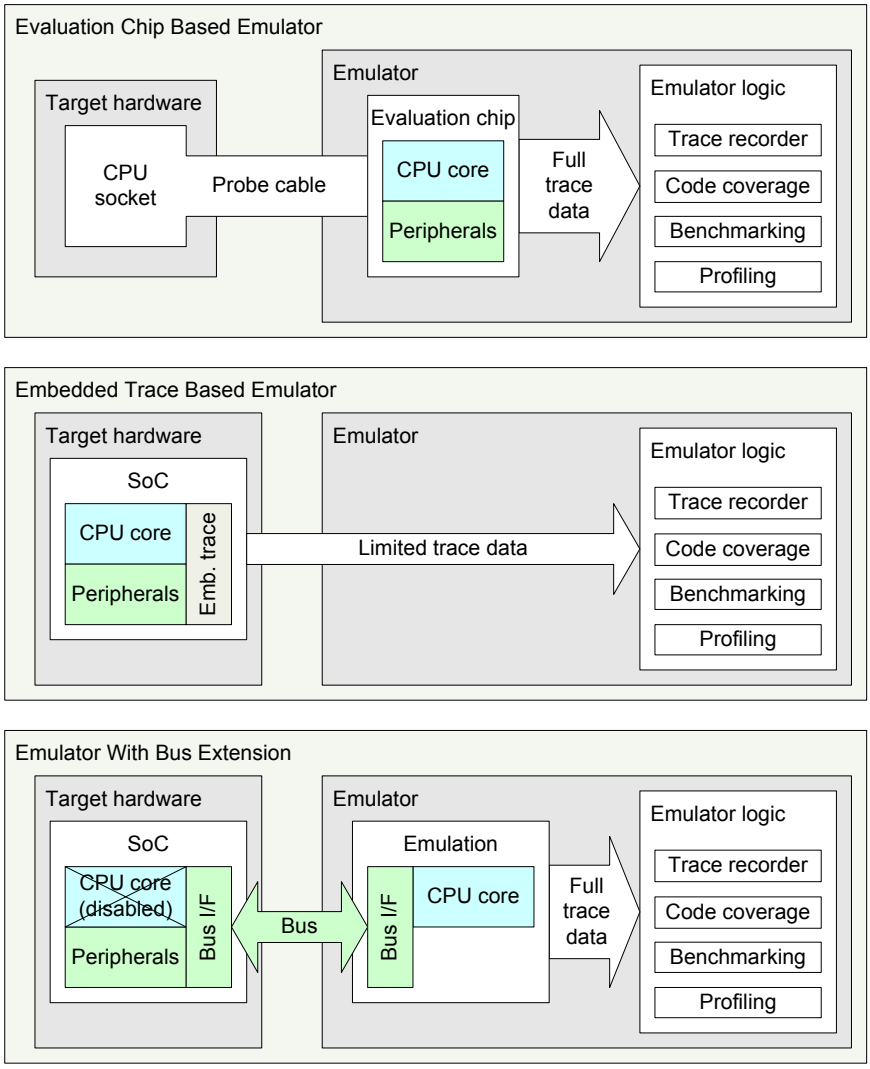


Figure 1-3: Overview of traditional trace approaches: Evaluation chip based emulator, embedded trace based emulator and emulation with bus extension

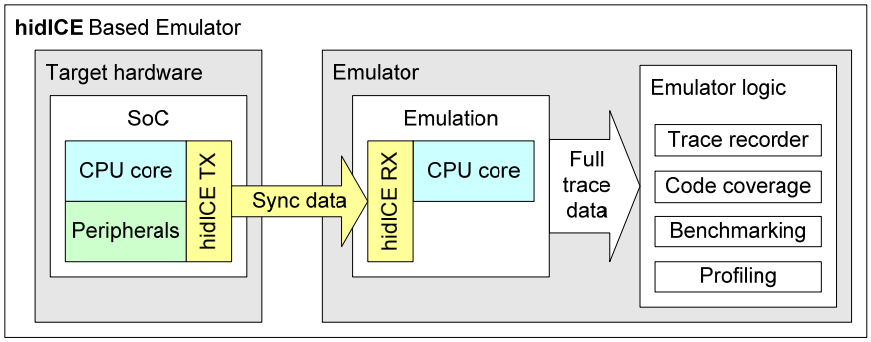


Figure 1-4: hidICE based emulation approach

Fig. 1-3 illustrates the traditional techniques (evaluation chip, embedded trace and bus extension based emulators). Fig. 1-4 illustrates the new hidICE approach.

On evaluation chip based emulators the application is executed in a special chip inside the emulator. Basically, the evaluation chip can provide all trace information. The disadvantages are the physical limitations of this concept, which allows a limited speed only, and the high system costs. Due to these reasons evaluation chip based emulators become less important.

Over the last years, embedded trace based emulators have become the state of the art approach. Here the trace capturing capabilities are implemented on chip. Due to the cost pressure only limited chip space is available for the embedded trace support. The result is that the developer has to accept an incomplete trace or the system will be slowed down when a complete trace has to be captured. With increasing CPU clock frequencies, this limitation gets more and more inconvenient.

The new approach combines the advantages of both traditional technologies: the trace can be accessed at very high CPU clocks in the same quality as from the evaluation chip. Depending on the SoC technology, we expect possible synchronization clock rates up to 400 MHz. In FPGAs (Xilinx Virtex 4) we have already implemented hidICE synchronized CPU cores running up to 200 MHz.

The proposed solution is well applicable for multi-core SoCs, but in case of independent clock domains and high I/O bandwidth the synchronization information may need a very high bandwidth. However, in this case other comparable technologies are also not able to deliver a full, continuous and real-time trace.

1.1.3. Further Applications

After the successful completion of the post-design and post-manufacturing tests of SoCs there is still the chance that some failures were still not detected. Especially, dynamic failures (cross talk, delays and power supply noise) occurring during the interaction between IP modules or under real operation conditions are difficult to find. Due to the general limitations of the built-in self-test (BIST) modules it would be too complex to implement full test coverage of dynamic operation and interaction of IPs.

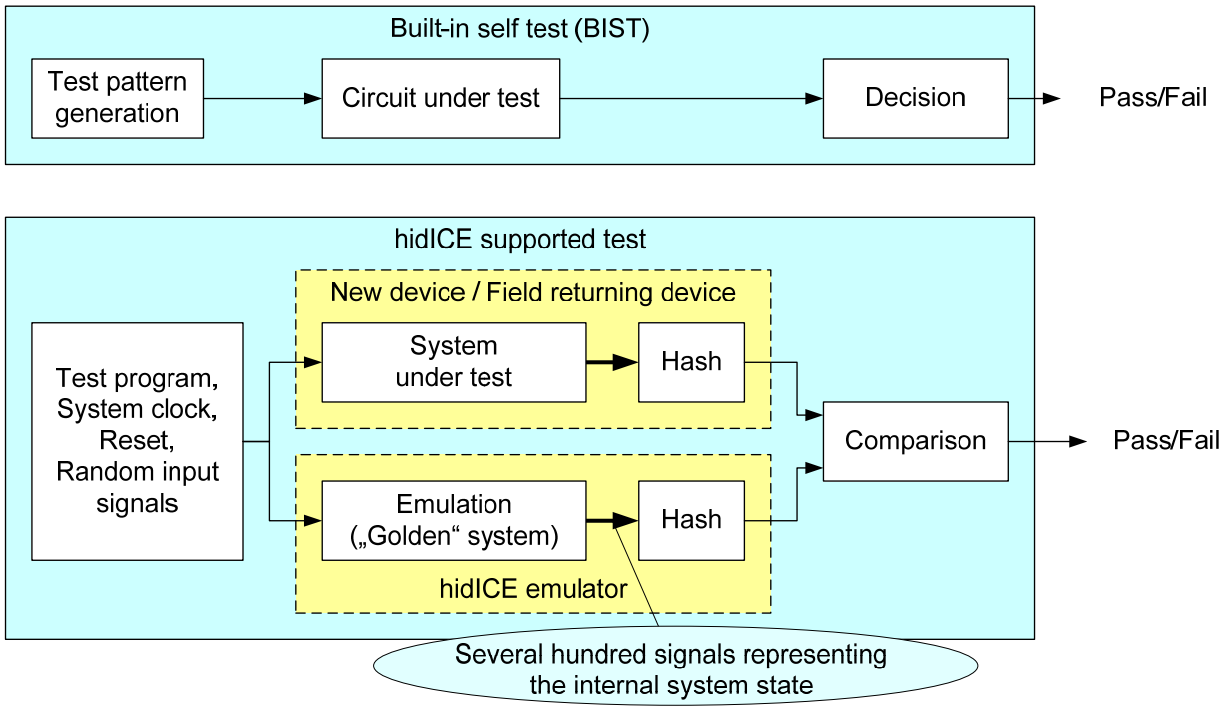


Figure 1-5: Built-in self test (BIST) and hidICE test approach

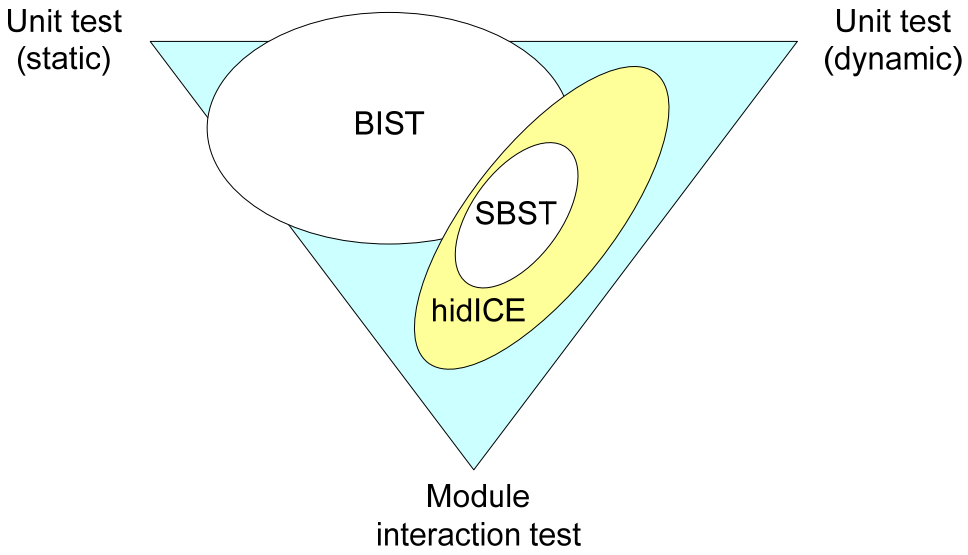


Figure 1-6: Test coverage by Built-In Self Test BIST, SBST and hidICE

At this point the hidICE methodology provides a high level supplement to the BIST and software based self test (SBST) approaches (Fig. 1-5 and Fig. 1-6). Initially special test routines are loaded into the SoC and the emulator. After applying a reset signal, the SoC and the synchronized emulation ("golden" system) execute the same code. On each clock cycle a hash value of relevant internal signals (e.g. embedded memory address, data, control signals etc.) will be computed recursively and compared. A divergence of the hash values of the SoC and the emulation indicates a failure. At this point, the recording of the execution trace within the emulator will be stopped. The captured trace is very helpful to analyze the root cause of the divergence. In this situation it is very helpful to have a range of multiple hashes to discover the source of the divergence. For instance, hash values can be independently computed for code addresses, fetched code, data read, data written, bus operations and the corresponding control signals.
In case of a divergence of the hash values the possibility of a false positive error detection has to be considered. This can be caused by a communication failure between SoC and emulation or by a failure within the emulation. Also it is self-evident that the tests can not be continued until a reset signal is applied to get both systems synchronized again.
As the tests are running in the real SoC environment, it is simple to modify and extend test routines. By changing environmental conditions such as temperature, clock frequency and supply voltage not only functional tests can be executed, but also the SoC manufacturing parameters can be verified and optimized.
In comparison to pure software based self tests the hidICE approach exposes a more comprehensive range of failures in combination with information of the exact time of the failure occurrence and a detailed trace.
Another application area is explorative tests. In this case the hidICE based test process keeps all individual peripheral components busy with random tasks and thus, all possible combinations of internal states are checked. This methodology is likely to also find failures that are not identified by module tests. As this approach can fairly easy be implemented in software, it is not an expensive test.
In general, the hidICE synchronization interface requires only a few I/O pins, depending on the SoC architecture. For extended test purposes a special mode can be implemented in the SoC, where additional I/O pins are available to output the synchronization data and more detailed hash values. In this case the proposed approach is also well applicable for SoCs with high I/O bandwidth (e.g. USB 2.0 controller, Gigabit Ethernet etc. on the CPU local bus).
Because the tasks of the hidICE IP in the SoC are very limited (collecting some signals, calculating hash values and handling the transfer protocol), only a few gates are required for the implementation. In comparison to BIST solutions the hidICE IP does not require a test pattern generator and a response checker. All analysis is done within the emulator, where much more resources are available then ever may be implemented into a BIST module.
By planning tests using the hidICE approach it has to be considered that the test coverage is limited to the observed chip core area and only to circuits which are accessible (direct or indirect) by the CPU.

Sometimes regularly produced chips fail in the field. For the manufacturer it is extremely interesting why those chips failed and which part of the chip failed. Also here the hidICE methodology allows to reconstruct the exact situation in which the chip failed (namely the time when the hashes differ) and in some cases even enable the identification of the failing component.

1.2. LEON3 Processor

The LEON3 is a synthesisable VHDL model of a 32-bit processor compliant with the SPARC V8 architecture. The model is highly configurable, and particularly suitable for system-on-a-chip (SOC) designs. The full source code is available under the GNU GPL license, allowing free and unlimited use for research and education. LEON3 is also available under a low-cost commercial license, allowing it to be used in any commercial application to a fraction of the cost of comparable IP cores. The LEON3 processor has the following features:

- *SPARC V8 instruction set with V8e extensions*
- *Advanced 7-stage pipeline*
- *Hardware multiply, divide and MAC units*
- *High-performance, fully pipelined IEEE-754 FPU*
- *Separate instruction and data cache (Harvard architecture) with snooping*
- *Configurable caches: 1 - 4 sets, 1 - 256 kbytes/set. Random, LRR or LRU replacement*
- *Local instruction and data scratch pad rams*
- *SPARC Reference MMU (SRMMU) with configurable TLB*
- *AMBA-2.0 AHB bus interface*
- *Advanced on-chip debug support with instruction and data trace buffer*
- *Symmetric Multi-processor support (SMP)*
- *Power-down mode and clock gating*
- *Robust and fully synchronous single-edge clock design*
- *Up to 125 MHz in FPGA and 400 MHz on 0.13 um ASIC technologies*
- *Fault-tolerant and SEU-proof version available for space applications*
- *Extensively configurable*
- *Large range of software tools: compilers, kernels, simulators and debug monitors*

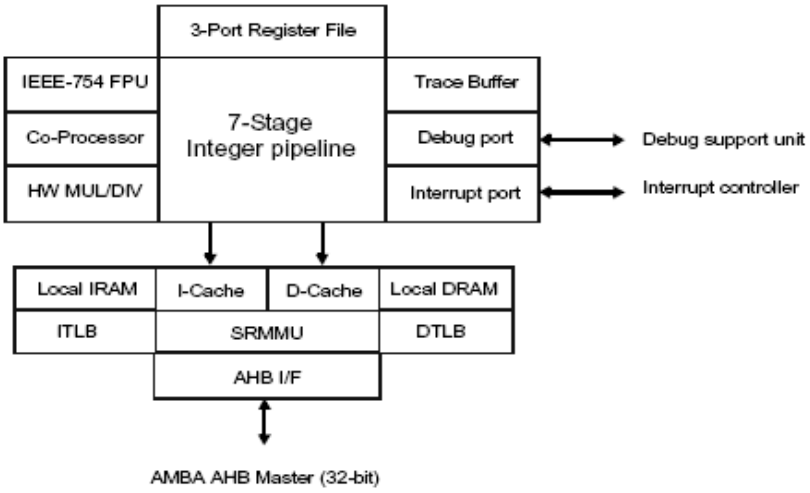


Figure 1-7: LEON3 processor core block diagram

The LEON3 processor is distributed as part of the GRLIB IP library, allowing simple integration into complex SOC designs. GRLIB also includes a configurable LEON3 multi-processor design, with up to 4 CPUs and a large range of on-chip peripheral blocks.

1.2.1. Configuration

The LEON3 processor is fully parametrizable through the use of VHDL generics, and does not rely on any global configuration package. It is thus possible to instantiate several processor cores in the same design with different configurations. The LEON3 template designs can be configured using a graphical tool built on tkconfig from the linux kernel. This allows new users to quickly define a suitable custom configuration. The configuration tool not only configures the processor, but also other on-chip peripherals such as memory controllers and network interfaces.

1.2.2. Synthesis

The LEON3 processor can be synthesised with common synthesis tools such as Synplify, Synopsys DC and Cadence RC. The core will reach up 125 MHz on FPGA and 400 MHz on 0.13 um ASIC technologies. The core area (pipeline, cache controllers and mul/div units) requires only 20 - 25 Kgates or 3500 LUT, depending on the configuration. The LEON3 processor can also be synthesised with Xilinx XST and Altera Quartus, either through scripts or by using the graphical interfaces of the Xilinx and Altera tools.

1.2.3. Distribution

LEON3 is distributed as part of the GRLIB IP library, and the library contains LEON3 templates designs for several popular FPGA prototyping boards. Pre-synthesized FPGA programming files are also provided.

1.2.4. SPARC Conformance

LEON3 has been certified by SPARC International as being SPARC V8 conformant. The certification was completed on May 1, 2005.

1.2.5. Software Development

Being SPARC V8 conformant, compilers and kernels for SPARC V8 can be used with LEON3 (kernels will need a LEON bsp). To simplify software development, Gaisler Research is providing BCC, a free C/C++ cross-compiler system based on gcc and the Newlib embedded C-library. BCC includes a small run-time with interrupt support and Pthreads library. For multi-threaded and/or multi-processor applications, a LEON3 port of the eCos real-time kernel is available. A LEON3 port of RTEMS 4.6.5 is available in form of the RCC cross-compiler, a system that supports RTEMS for ERC32, LEON2 and LEON3. For industrial and high-rel applications, ports for Nucleus, VxWorks 5.4 and 6.3 are available, as well as ThreadX.
Linux support for LEON3 is provided through a special version of the SnapGear Embedded Linux distribution. SnapGear Linux is a full source package, containing kernel, libraries and application code for rapid development of embedded Linux systems. The LEON3 port of SnapGear supports both MMU and non-MMU LEON configurations, as well as the optional V8 mul/div instructions and floating-point unit (FPU). A single cross-compilation tool-chain is provided which is capable of compiling the kernel and applications for any configuration. The latest version of Snapgear now also includes multi-processor support (SMP) for multi-core LEON3 systems!
Debugging is generally done using the gdb debugger, and a graphical front-end such as DDD or Eclipse. It is possible to perform source-level symbolic debugging, either on a simulator or using real target hardware. Gaisler Research provides TSIM, a high-performance LEON3 simulator which seamlessly can be attached to gdb and emulate a LEON3 system at more than 10 MIPS. The GRMON monitor interfaces to the LEON3 on-chip debug support unit (DSU), implementing a large range of debug functions as well as a GDB gateway. For multi-processor and/or advanced SOC designs, the GRSIM multi-core simulator is available for early software development.

2. hidICE Evaluation System

The hidICE evaluation system provides the following functionality:

- Multi-core LEON3 SoC
- Multiple AHB Bus Master
- On-Chip Debug Support
- AHB periphery

The SoC consists of

- 2 CPUs (LEON3 core)
- 1 CPU (LEON3 core) used as DMA controller
- OCDs support (AHBUART and DSU3)
- 4 AHB bus masters (3 x CPU, 1 x AHBUART)
- Clock controller (enables clock switching while the application is running)
- Multiprocessor Interrupt Controller
- AHB and APB bus, AHB/APB bus bridge
- APB Periphery: Timer, UART, I/O port, ADC
- hidICE Sync IP (RX and TX)
- hidICE Hash IP and comparator (system integrity control)

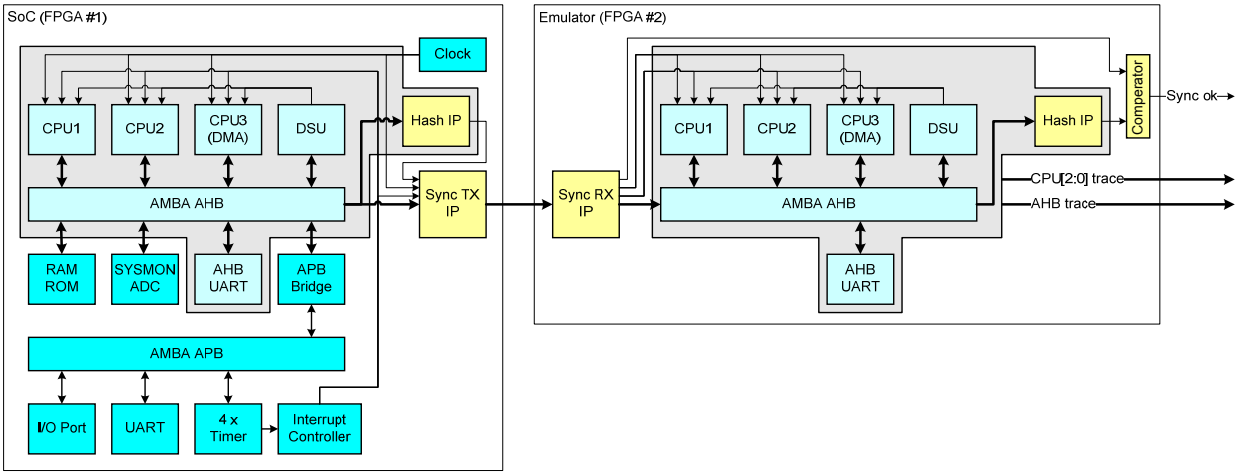


Figure 2-1: hidICE evaluation system block diagram

The ROM content will be loaded in the external memory by Accemic MDE.
The software will provide the following features:

- System initialization
- Test routines can be controlled by PC via OCDs
- Handling of timer interrupts

2.1. Hardware Description

The hidICE evaluation system consists of two Xilinx ML507 boards.

2.1.1. ML507 Features

- Xilinx Virtex-5 FPGA XC5VFX70T-1FFG1136
- Two Xilinx XCF32P Platform Flash PROMs (32 Mb each)
- Xilinx System ACE™ CompactFlash configuration controller with Type I CompactFlash connector
- Xilinx XC95144XL CPLD for glue logic
- 64-bit wide, 256-MB DDR2 small outline DIMM (SODIMM)
- Clocking
  - Programmable system clock generator chip
  - One open 3.3V clock oscillator socket
  - External clocking via SMAs (two differential pairs)
- General purpose DIP switches (8), LEDs (8), pushbuttons, and rotary encoder
- Expansion header with 32 single-ended I/O, 16 LVDS-capable differential pairs, 14 spare I/Os shared with buttons and LEDs, power, JTAG chain expansion capability, and IIC bus expansion
- Stereo AC97 audio codec with line-in, line-out, 50-mW headphone, microphone-in jacks, SPDIF digital audio jacks, and piezo audio transducer
- RS-232 serial port, DB9 and header for second serial port
- 16-character x 2-line LCD display
- One 8-Kb IIC EEPROM and other IIC capable devices
- PS/2 mouse and keyboard connectors
- VGA and DVI video output (Chrontel CH7301C)
- 9Mb ZBT synchronous SRAM (ISSI IS61NLP25636A-200TQL)
- 32 MB Flash (Intel JS28F256P30T95)
- Serial Peripheral Interface (SPI) flash (2 MB)
- 10/100/1000 tri-speed Ethernet PHY transceiver and RJ-45 with support for MII, GMII, RGMII, and SGMII Ethernet PHY interfaces (Marvell 88E1111)
- USB interface chip with host and peripheral ports (Cypress CY7C67300)
- Rechargeable lithium battery to hold FPGA encryption keys
- JTAG configuration port for use with Parallel Cable III, Parallel Cable IV, or Platform USB download cable
- Onboard power supplies for all necessary voltages
- Temperature and voltage monitoring chip with fan controller
- 5V @ 6A AC adapter
- Power indicator LED
- MII, GMII, RGMII, and SGMII Ethernet PHY Interfaces
- GTP/GTX: SFP (1000Base-X) / SMA (RX and TX Differential Pairs) / SGMII / PCI Express® (PCIe™) edge connector (x1 Endpoint) / SATA (dual host connections) with loopback cable / Clock synthesis ICs
- Mictor trace port
- BDM debug port
- Soft touch port
- System monitor

2.2. Implementation overview

2.2.1. Target SoC implementation

Instance	File	Description
hidICE_3CV2_target_tl	hidICE_3CV2_target_tl.v	Top level module
|-- dcm0	Xilinx DCM	Sytem clock generation
|-- rstgen_inst	rstgen.vhd	Reset generator (grlib)
|-- hidICE_3CV2_target_inst	hidICE_3CV2_target.hvd	SoC system
|-- ahbctrl0	ahbctrl.vhd	AHB controller (grlib)
|-- ahbuart0	ahbuart.vhd	AHB UART (grlib)
|-- ahbram0	ahbram.vhd	AHB RAM (grlib)
|-- ahbtrace0	ahbtrace.vhd	AHB Trace (grlib)
|-- cpu0	leon3s.vhd	LEON3 CPU (grlib)
|-- cpu1	leon3s.vhd	LEON3 CPU (grlib)
|-- cpu2	leon3s.vhd	LEON3 CPU (grlib)
|-- dsu0	dsu3.vhd	Debug Support Unit (grlib)
|-- apbctrl0	apbctrl.vhd	AHB / APB bridge (grlib)
|-- irqmp0	irqmp.vhd	Interrupt controller (grlib)
|-- gpio0	gpio.vhd	I/O ports (grlib)
|-- timer0	gptimer.vhd	Timer unit (grlib)
|-- apbuart1	apbuart.vhd	APB UART (grlib)
|-- mctrl0	mctrl.vhd	Memory controller (grlib)
|-- sysm0	grsysmon.vhd	System monitor / ADC (grlib)
|-- target_hash_data_inst	datahash.v	AHB data hash
|-- target_hash_adr_inst	hash1.v	AHB address hash
|-- qdr_out_inst	QDR_out.v	Sync data output

Table 2-1: Main modules of target SoC implementation

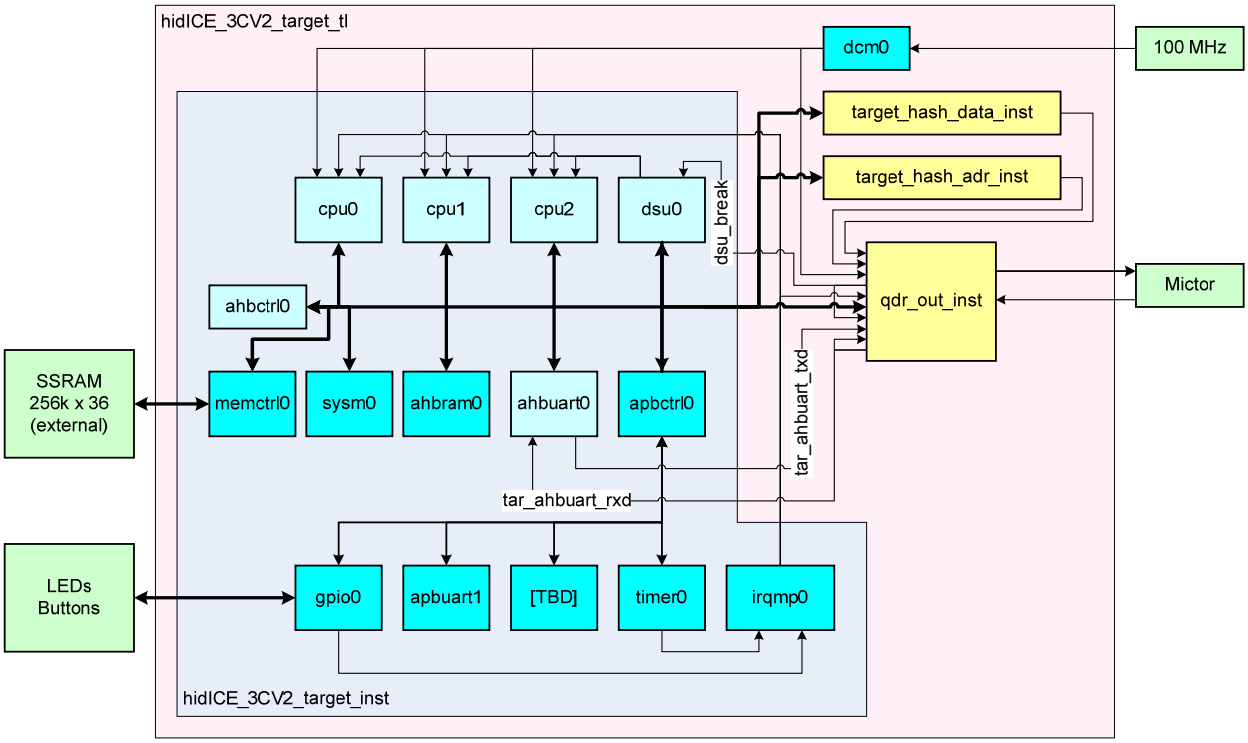


Figure 2-2: Target SoC implementation block diagram

2.2.2. Emulator implementation

Instance	File	Description
hidICE_3CV2_emu_tl	hidICE_3CV2_target_tl.v	Top level module
|-- dcm0	Xilinx DCM	DDR input delay clock support
|-- hidICE_3CV2_emu_inst	hidICE_3CV2_emu.hvd	Emulation system
|-- ahbctrl0	ahbctrl.vhd	AHB controller (grlib)
|-- ahbuart0	ahbuart.vhd	AHB UART (grlib)
|-- ahbtrace0	ahbtrace.vhd	AHB Trace (grlib)
|-- cpu0	leon3s.vhd	LEON3 CPU (grlib)
|-- cpu1	leon3s.vhd	LEON3 CPU (grlib)
|-- cpu2	leon3s.vhd	LEON3 CPU (grlib)
|-- dsu0	dsu3.vhd	Debug Support Unit (grlib)
|-- emu_hash_data_inst	datahash.v	AHB data hash
|-- emu_hash_adr_inst	hash1.v	AHB address hash
|-- qdr_in_inst	QDR_out.v	Sync data input

Table 2-2: Main modules of emulator implementation

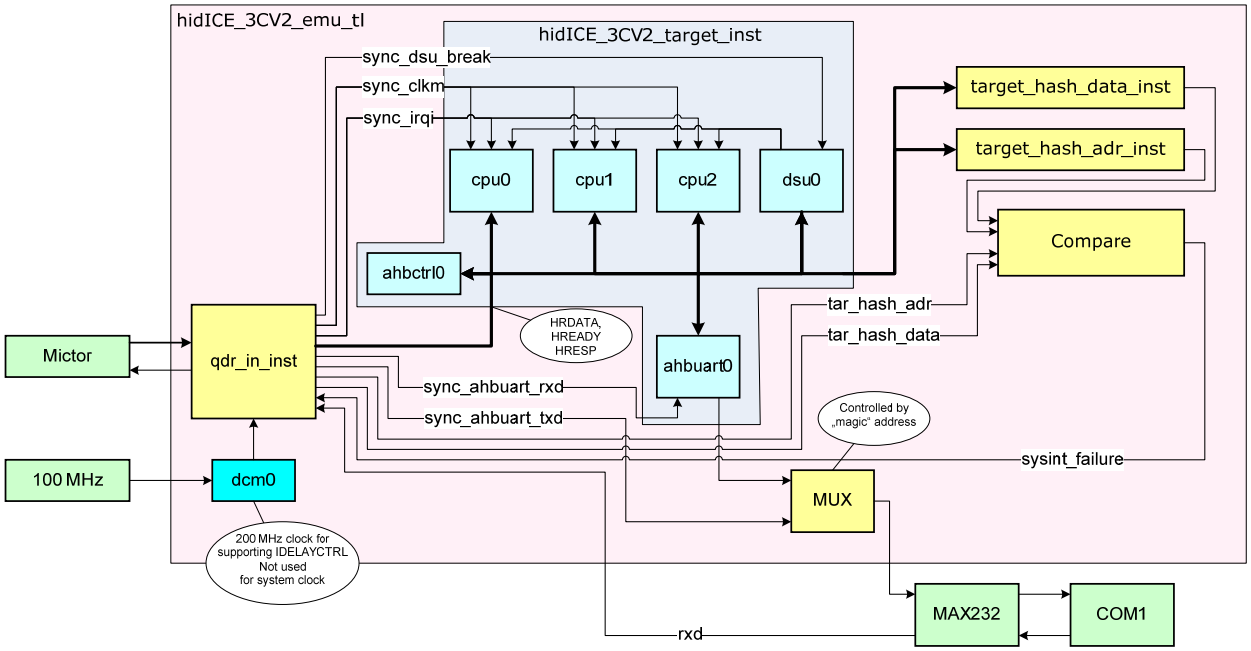


Figure 2-3: Emulation implementation block diagram

2.3. Synchronization

The synchronization interface consists of the following signals:

Signal	Direction	Width	Description
CLKM	T -> E	1	System clock
RESET	T -> E	1	System reset
HRDATA[31:0]	T -> E	32	AHB bus read data
HREADY	T -> E	1	AHB bus transfer done
HRESP[1:0]	T -> E	2	AHB bus transfer response
IRQI_IRL	T -> E	12	Interrupt level of 3 CPUs a (4 lines each)
IRQI_RST	T -> E	3	Power down for the 3 CPUs a (1 line each)
HASH_ADR	T -> E	1	Address hash
HASH_DATA	T -> E	2	Data hash

Table 2-3: Synchronization signals

For the debug support the following signals are required:

Signal	Direction	Width	Description
DSU_BREAK_OUT	T -> E	1	Emulation DSU break
AHBUART_RXD_OUT	T -> E	1	Input signals for emulation AHB_UART
AHBUART_TXD	T -> E	1	Output signals from target AHB_UART
DSU_BREAK_IN	E -> T	1	Break command for DSU (in case of system integrity failure)
AHBUART_RXD_IN	E -> T	1	Input signals for target AHB_UART

Table 2-4: Debug signals

The synchronization signals are modulated with a QDR modulator to save IO pins and to use the MICTOR cable for transmission.
The QDR modulator is clocked by the differential clock J12/J13.

Please note, the synchronization signals are not optimized for bandwidth / pin count reduction in this stage of development.
The pin count optimization is planned for a further task. After optimization (DDR transfer, using of ready states and optimization of interrupt level transfer) we expect a reduction to 10-20 pins.

2.4. Functionality

- Implementation of the LEON3 based Sock and the hidICE subsystem
- Installation and start-up the LEON3 tool chain
- Verification of synchronization (system integrity control)
- Accessing of trace data from target and emulation:
    - CPU trace (CPU 0, CPU1, CPU2):
        - Time stamp
        - Instruction address
        - Instruction code
        - Data address
        - Data access direction (r/w)
    - AHB bus trace
        - Time stamp
        - Address
        - Access type
        - Data
        - Bus control signals

2.4.1. Memory Map

Address	Unit	Description
0x40000000	MCTRL (1 MByte external SRAM)	Code Start CPU 0
0x4000FFFC		Stack Top CPU 0
0x40010000		Code Start CPU 1
0x4001FFFC		Stack Top CPU 1
0x40020000		Code Start CPU 2 (DMA)
0x4002FFFC		Stack Top CPU 2 (DMA)
0x40030000 to 0x4003000C		DMA control register
0x40030010 to 0x4003FFFF		General purpose RAM
0x80000000	APCBTRL	AHB/APB bridge
0x90000000	DSU3	Debug Support unit (controlled by Accemic MDE)
0xA0000000 to 0xA001FFFF	AHBRAM (128 kByte internal SRAM)	AHB internal RAM (general purpose RAM)
0xF0000000	Special functions	See section "Special Functions"
0xFFF00000	AHBTRACE	AHB trace buffer
0xFFFC0000	System monitor control	ADC functionality
0xFFFC1000	System monitor data	ADC functionality

Table 2-5: Memory map

2.4.1.1. Periphery Control Registers

Address	Unit	Register
0x80000000	Memory controller	Memory config register 1
0x80000004		Memory config register 2
0x80000008		Memory config register 3
0x80000100	APB UART	UART data register
0x80000104		UART status register
0x80000108		UART control register
0x8000010C		UART scaler register
0x80000200	Multi-processor Interrupt Ctrl	Interrupt level register
0x80000204		Interrupt pending register
0x80000208		Interrupt force register
0x8000020C		Interrupt clear register
0x80000210		Interrupt status register
0x80000240		Interrupt mask register 0
0x80000244		Interrupt mask register 1
0x80000248		Interrupt mask register 2
0x80000280		Interrupt force register 0
0x80000284		Interrupt force register 1
0x80000288		Interrupt force register 2
0x80000300	Modular Timer Unit	Scaler value register
0x80000304		Scaler reload register
0x80000308		Configuration register
0x80000310		Timer1 value register
0x80000314		Timer1 reload register
0x80000318		Timer1 control register
0x80000320		Timer2 value register
0x80000324		Timer2 reload register
0x80000328		Timer2 control register
0x80000330		Timer3 value register
0x80000334		Timer3 reload register
0x80000338		Timer3 control register
0x80000340		Timer4 value register
0x80000344		Timer4 reload register
0x80000348		Timer4 control register
0x80000800	General purpose I/O port	I/O data register
0x80000804		I/O output register
0x80000808		I/O direction register
0x8000080C		I/O interrupt register
0x80000810		I/O interrupt polarity reg.
0x80000814		I/O interrupt edge register
0x80000818		I/O bypass register

0xFFFC0000	System Monitor	Configuration register
0xFFFC0004		Status register
0xFFFC1000		Temperatur register
0xFFFC1004		VCCINT register
0xFFFC1008		VCCAUX register
0xFFFC100C		VP/VN register
0xFFFC1010		VREFP register
0xFFFC1014		VREFN register
0xFFFC1020		Supply Offset register
0xFFFC1024		ADC Offset register
0xFFFC1028		Gain Error register
0xFFFC1040		Result register 0
0xFFFC1044		Result register 1
0xFFFC1048		Result register 2
0xFFFC104C		Result register 3
0xFFFC1050		Result register 4
0xFFFC1054		Result register 5
0xFFFC1058		Result register 6
0xFFFC105C		Result register 7
0xFFFC1060		Result register 8
0xFFFC1064		Result register 9
0xFFFC1068		Result register 10
0xFFFC106C		Result register 11
0xFFFC1070		Result register 12
0xFFFC1074		Result register 13
0xFFFC1078		Result register 14
0xFFFC107C		Result register 15
0xFFFC1080		Max Temp register
0xFFFC1084		Max VCCINT register
0xFFFC1088		Max VCCAUX register
0xFFFC1090		Min Temp register
0xFFFC1094		Min VCCINT register
0xFFFC1098		Min VCCAUX register

Table 2-6: IO registers

2.4.1.2. Special Functions

By a write operation to the special addresses the following actions can be performed:

Address	Action
0xF0000000	Toggling the system clock
0xF0000004	System reset
0xF0000008	Communication with target AHBUART
0xF000000C	Communication with emulation AHBUART

Table 2-7: Special function addresses

These special functions are used by Accemic MDE.

2.4.2. Interrupts

The interrupt processing is provided by the Multiprocessor Interrupt Controller (IRQMP) core. A detailed description can be found in chapter 54 of the GRLIB IP Library User's Manual.

The AMBA system in GRLIB provides an interrupt scheme where interrupt lines are routed together with the remaining AHB/APB bus signals, forming an interrupt bus. Interrupts from AHB and APB units are routed through the bus, combined together, and propagated back to all units. The multiprocessor interrupt controller core is attached to AMBA bus as an APB slave, and monitors the combined interrupt signals.
The interrupts generated on the interrupt bus are all forwarded to the interrupt controller. The interrupt controller prioritizes masks and propagates the interrupt with the highest priority to the processor. In multiprocessor systems, the interrupts are propagated to all processors.

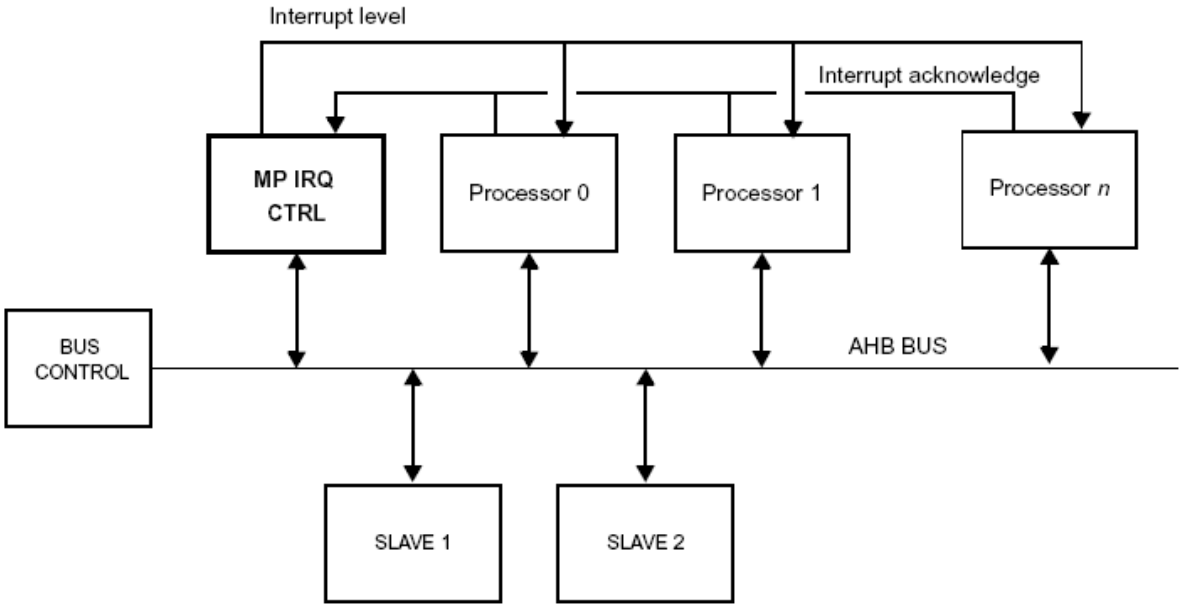


Figure 2-4: Interrupt system block diagram

The interrupt controller monitors interrupt 1 - 15 of the interrupt bus (APBI.PIRQ[15:1]). When any of these lines are asserted high, the corresponding bit in the interrupt pending register is set. The pending bits will stay set even if the PIRQ line is de-asserted, until cleared by software or by an interrupt acknowledge from the processor. Each interrupt can be assigned to one of two levels (0 or 1) as programmed in the interrupt level register. Level 1 has higher priority than level 0. The interrupts are prioritised within each level, with interrupt 15 having the highest priority and interrupt 1 the lowest. The highest interrupt from level 1 will be forwarded to the processor. If no unmasked pending interrupt exists on level 1, then the highest unmasked interrupt from level 0 will be forwarded.

Interrupt	Level	Priority	Assigned to (by hardware)	Handled by (sample projects)
15	1	31 (Highest)		
14	1	30		
13	1	29		
12	1	28		
11	1	27		
10	1	26		
9	1	25		
8	1	24		
7	1	23	Push button West	CPU1
6	1	22		
5	1	21		
4	1	20	Push button East	CPU1
3	1	19		
2	1	18		
1	1	17		
0	1	16		
15	0	15		
14	0	14		
13	0	13		
12	0	12		
11	0	11		
10	0	10		
9	0	9		
8	0	8	Timer	CPU0
7	0	7		
6	0	6	Push button South	CPU0
5	0	5	Push button North	CPU0
4	0	4		
3	0	3		
2	0	2		
1	0	1		
0	0	0 (Lowest)		

Table 2-8: Interrupt priorities and interrupt services of the sample projects

2.4.2.1. Interrupt instantiation

The interrupt lines of the peripheries can be assigned to the interrupt inputs of the Multiprocessor Interrupt Controller by setting the corresponding VHDL generics (within the file "hidICE_3CV2_tar.vhd").

The interrupts of the AHB and APB peripherals can be assigned to an interrupt line by setting the PIRQ generic.

```
component apbslave
generic (
      pindex : integer := 0;        -- slave index
      pirq : integer := 0);         -- interrupt index
port (rst : in std_ulogic;
      clk : in std_ulogic;
      apbi : in apb_slv_in_type;     -- APB slave inputs
      apbo : out apb_slv_out_type); -- APB slave outputs
end component;

slave3 : apbslave
generic map (pindex => 1, pirq => 2)
port map (rst, clk, pslvi, pslvo(1));
```

Figure 2-5: Code snippet for interrupt assignment demonstration

The AHB/APB bridge in the GRLIB provides interrupt combining, and merges the APB-generated interrupts with the interrups bus on the AHB bus. This is done by OR-ing the interrupt vectors from each APB slave into one joined vector, and driving the combined value on the AHB slave output bus (AHBSO.HIRQ).

An exception from this schema is the interrupt assignment of I/O ports.
Each I/O port can drive a separate interrupt line on the APB interrupt bus. The interrupt number is equal to the I/O line index (PIO[1] = interrupt 1, etc.). The interrupt generation is controlled by three registers: interrupt mask, polarity and edge registers. To enable an interrupt, the corresponding bit in the interrupt mask register must be set. If the edge register is '0', the interrupt is treated as level sensitive.
If the polarity register is '0', the interrupt is active low. If the polarity register is '1', the interrupt is active high. If the edge register is '1', the interrupt is edge-triggered. The polarity register then selects between rising edge ('1') or falling edge ('0').

2.4.2.2. Interrupt processing

Within the sample projects the processing of different interrupts is demonstrated. The sample projects use the BCC library functions (catch_interrupt), explained in the Bare-C Cross-Compiler User's Manual and routines defined in "leon3.h" (enable_irq).

The following example demonstrates the usage of the timer interrupt, assigned to interrupt 8:

Code	Description

`void Init_Timer_IRQ (void)`
`{`
`  catch_interrupt((int)timerirqhandler, 8);`
 	
`  APB_BUS(T1VAL) = 0x00000000;`
`  APB_BUS(T1REL) = t1_reload;`
`  APB_BUS(T1CON) = 0x0000000D;`
 	
`  enable_irq(8,0);`
 	
`}`
 	
`void timerirqhandler(int irq)`
`{`
`  APB_BUS(T1CON) = 0x0000000D;`
 	
`  leds = leds+1;`
`  APB_BUS(GRGPIO_OUT) = (leds<<8)&0xF00;`
`}`
	Interrupt initialization
	
	Assign interrupt service routine to interrupt 8
	
	Init the timer
	
	Enable the timer interrupt (Level 0)
	
	Timer interrupt service routine
	
	Clear interrupt request
	
	Do something…

2.4.3. On-Chip Debug Support

2.4.3.1. Hardware implementation

The on-chip debug support (OCD) is provided by the DSU3 unit in combination with the AHBUART.

To simplify debugging on target hardware, the LEON3 processor implements a debug mode during which the pipeline is idle and the processor is controlled through a special debug interface. The LEON3 Debug Support Unit (DSU) is used to control the processor during debug mode. The DSU acts as an AHB slave and can be accessed by any AHB master. An external debug host can therefore access the DSU through the AHBUART. The DSU supports multi-processor systems and can handle up to 16 processors.

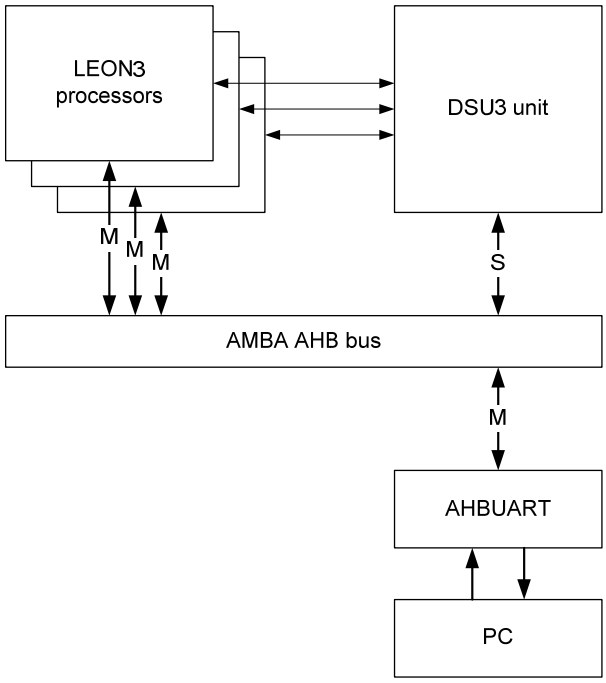


Figure 2-6: DSU3 connection

Through the DSU AHB slave interface, any AHB master can access the processor registers and the contents of the instruction trace buffer. The DSU control registers can be accessed at any time, while the processor's registers, caches and trace buffer can only be accessed when the processor has entered debug mode. In debug mode, the processor pipeline is held and the processor state can be accessed by the DSU.

The AHBUART consists of a UART connected to the AMBA AHB bus as a master. A simple communication protocol is supported to transmit access parameters and data. Through the communication link, a read or write transfer can be generated to any address on the AMBA AHB bus.

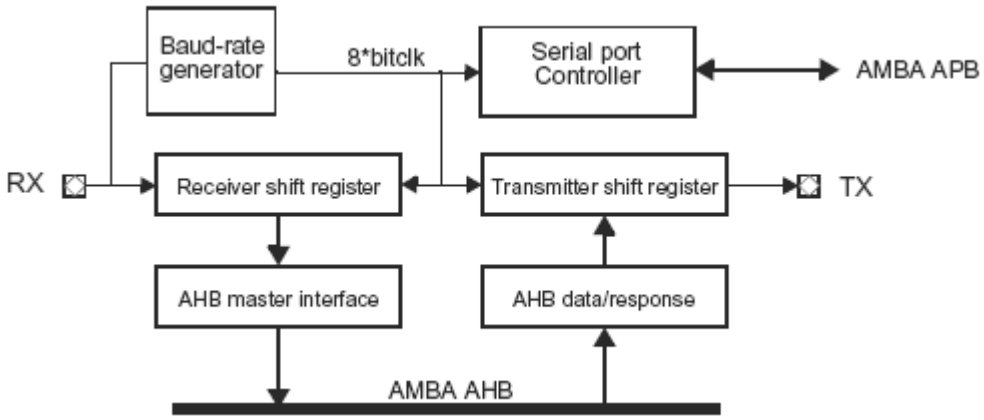


Figure 2-7: AHBUART block diagram

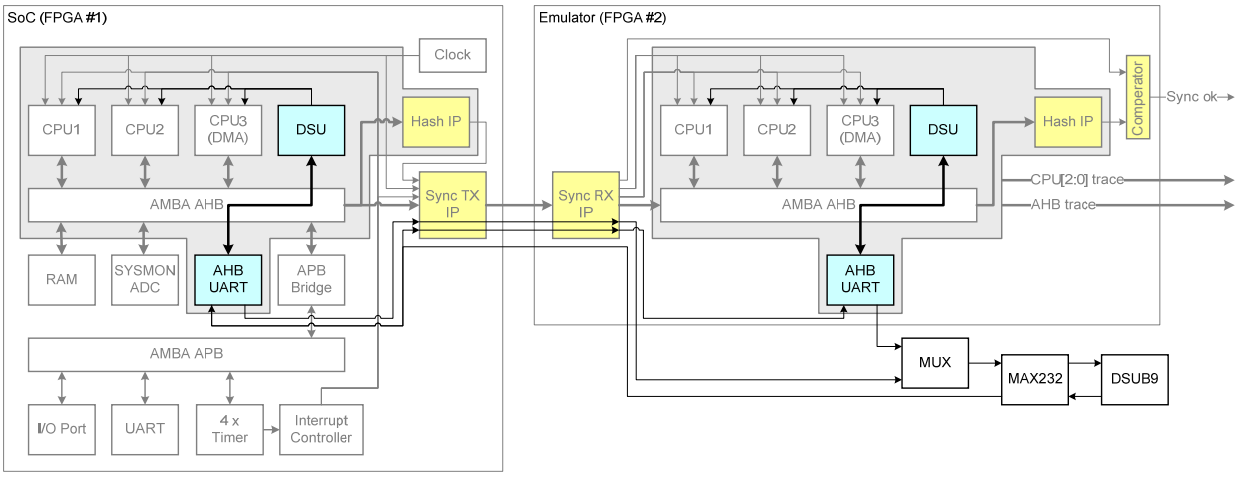


Figure 2-8: Multiple AHB bus master LEON3 SoC system with on-chip debug support

2.4.3.2. Accemic MDE

With Accemic MDE a user interface which controls the OCD functionality is provided by Accemic.
Accemic MDE provides the following features:

- Support for AHBUART debug link via PC COM port
- Read/write access to all system registers and memory
- Download application (CPU0, CPU1, CPU2)
- Start / Stop application (individual or all CPUs)
- Setting breakpoints (CPU0, CPU1, CPU2)
- Disassembler
- Display of trace buffer (CPU0, CPU1, CPU2, AHB)
- Comparison of trace buffer content (target and emulation)
- Accessing global and local variables
- Visualization of periphery registers and I/O ports (Processor state window)

Please note, that the Accemic MDE for SPARC V8 version is a beta release and not completely tested.

2.4.3.3. GRMON / GRMON RCP

Alternatively, the GRMON tools can be used to control the hidICE demonstration system. The evaluation version of GRMON can be downloaded from www.gaisler.com. The evaluation version may be used during a period of 21 days without purchasing a license.

2.4.4. System Integrity Control

To ensure the system integrity, hash values of AHB addresses and AHB data will be calculated.
In case of a difference of the hash values, the DSU units in the target and in the emulation will be stopped. The trace data can be read by Accemic MDE to explore possible reasons of system integrity loss.

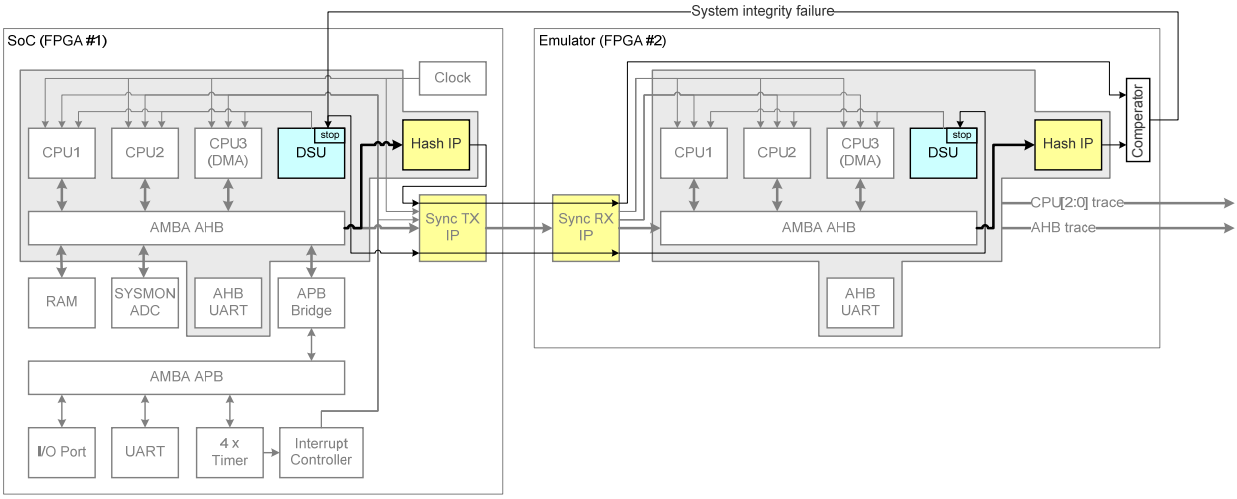


Figure 2-9: hidICE system integrity control. In case of system integrity failure the DSU unit of the target and of the emulation will be stopped. The target and the emulation trace are displayed in the Accemic MDE trace window.

The system integrity control implementation can be tested by the CPU0_Test program.

CPU 0 does continuously read from 0x40030020. If the center button (error button) is pressed, the target integrity control error simulation does invert the data that is sent to the emulator. The system integrity control comparator (inside the emulator) will stop the target and the emulator (DSU-Break). Before the read operation from 0x40030020 is done the data cache has to be flushed, otherwise no operation will occur on the AHB. Note that AHB trace in Accemic MDE is stopped by the DSU break signal.
After a target integrity control error both systems have to be powered down to clear the AHB trace. Before that, you should disconnect Accemic MDE.

```
reg[1:0] errorstate;

always @(posedge clkm)
begin
      if(rstn == 0)
            errorstate <= 0;

      if(error_button == 1 &&
            tar_ahbsi[`ahbsi_hwrite] == 0 &&
            tar_ahbsi[`ahbsi_htrans_high] == 1 &&
            tar_ahbsi[`ahbsi_haddr] == 'h40030020)
            errorstate <= 1;
      else if(errorstate==1 && tar_ahbmi[`ahbmi_hready]==0)
            errorstate <= 2;
      else if(errorstate==2 && tar_ahbmi[`ahbmi_hready])
            errorstate <= 0;
end

assign ahbmi_hrdata_sync[0] = (errorstate == 2) ?
~ahbmi_hrdata[0]:ahbmi_hrdata[0];
assign ahbmi_hrdata_sync[31:1] = ahbmi_hrdata[31:1];
```

Figure 2-10: hidICE system integrity control source code snippet

In the sample projects, the following routine will be executed continuously in CPU0:

```
 1    void TestSyncControl()
 2    {
 3          volatile int dummy = 0;
 4          *(int*)(0x40030020) = 0;
 5
 6          __asm__ __volatile__("FLUSH");//flush data cache
 7          __asm__ __volatile__("nop");
 8          __asm__ __volatile__("nop");
 9          __asm__ __volatile__("nop");
10          __asm__ __volatile__("nop");
11
12          dummy=*(int*)(0x40030020);
13
14          if (*(int*)(0x40030020) == 0)
15                dummy ++;
16    }
```

Figure 2-11: System integrity control test routine

If the center button is pressed, the system stops after execution of line 12

```
dummy=*(int*)(0x40030020);
```

The ERR1 LED (Integrity Control error on data access) gets active now and the AHB trace window displays the difference between target (T) and emulation (E) bus data.

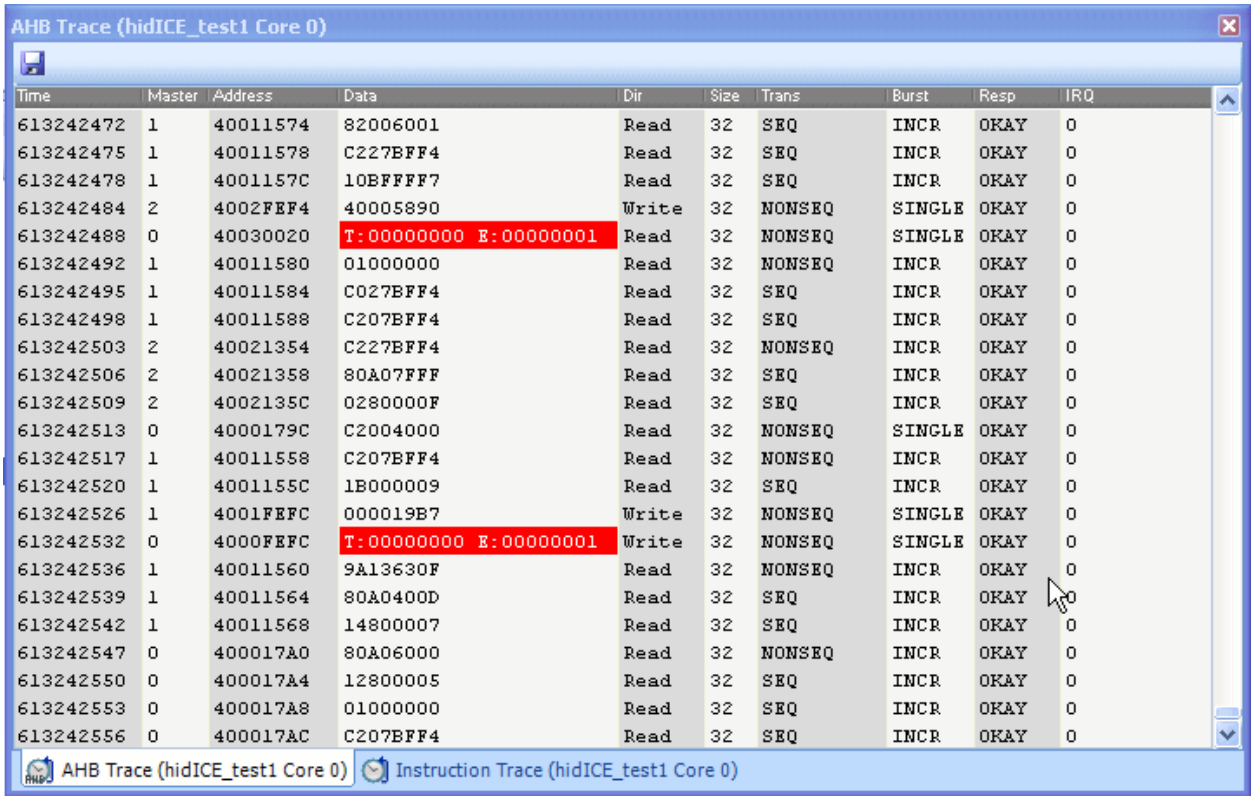


Figure 2-12: AHB bus trace with system integrity failure

In the target the value 0x00000000 is read by the bus, but to the emulation the modified value 0x00000001 is transmitted. The hash differs and the hash comparator stops the system.
The instruction trace of CPU0 is still valid, due to the read operation to address 0x40030020 does not influence the program counter.

In the next code line (press the center button and execute the Single Step command over line 14) the program counter of CPU0 on the target and CPU0 on the emulation differs. Now the instruction trace is also invalid and the ERR0 LED (Integrity Control error far address) gets active.

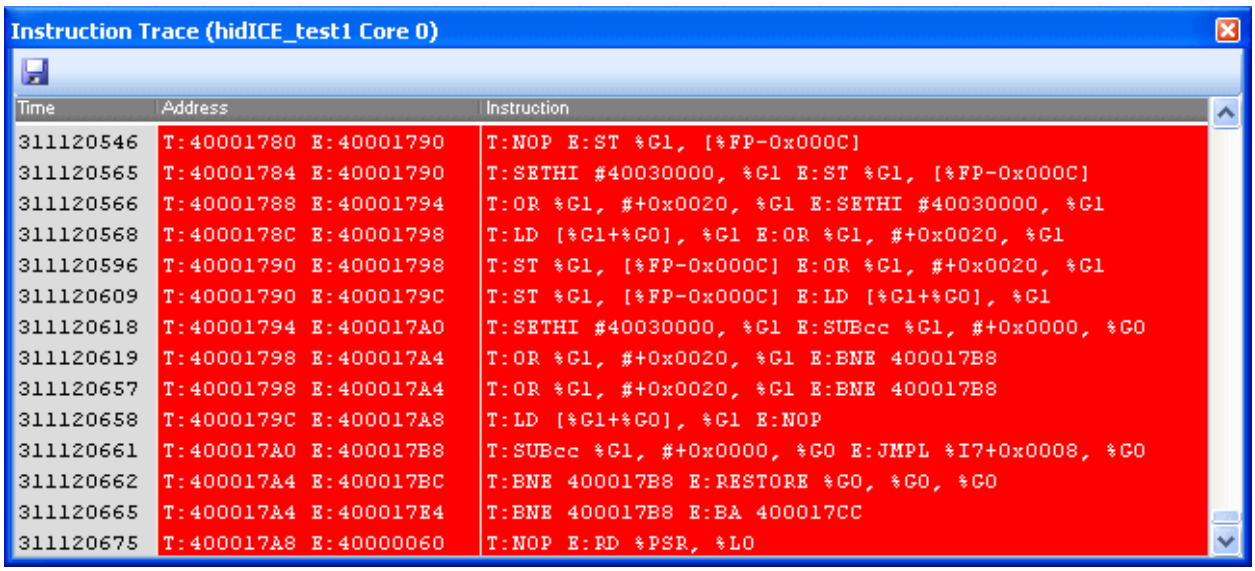


Figure 2-13: CPU0 instruction trace with system integrity failure

To reset the system simply turn off the power of the emulation board first, and than of the target board.

2.4.5. Peripherals

The hidICE demosntartion system includes the following peripherals:

- GRGPIO - General Purpose I/O Port
- UART
- ADC
- Timer

2.4.5.1. GRGPIO - General Purpose I/O Port

The general purpose input output port core is a scalable and provides optional interrupt support. The port width can be set to 2 - 32 bits through the nbits VHDL generic (i.e. nbits = 16). Interrupt generation and shaping is only available for those I/O lines where the corresponding bit in the imask VHDL generic has been set to 1. Each bit in the general purpose input output port can be individually set to input or output, and can optionally generate an interrupt. For interrupt generation, the input can be filtered for polarity and level/edge detection.

The I/O ports are implemented as bi-directional buffers with programmable output enable. The input from each buffer is synchronized by two flip-flops in series to remove potential meta-stability. The synchronized values can be read-out from the I/O port data register. They are also available on the GPIOO.VAL signals. The output enable is controlled by the I/O port direction register. A '1' in a bit position will enable the output buffer for the corresponding I/O line. The output value driven is taken from the I/O port output register. Each I/O port can drive a separate interrupt line on the APB interrupt bus. The interrupt number is equal to the I/O line index (PIO[1] = interrupt 1, etc.). The interrupt generation is controlled by three registers: interrupt mask, polarity and edge registers. To enable an interrupt, the corresponding bit in the interrupt mask register must be set. If the edge register is '0', the interrupt is treated as level sensitive.
If the polarity register is '0', the interrupt is active low. If the polarity register is '1', the interrupt is active high. If the edge register is '1', the interrupt is edge-triggered. The polarity register then selects between rising edge ('1') or falling edge ('0').

I/O port pin	FPGA pin	Part	Signal
0	AH30	SW2_1	Rotation encoder inc_a
1	AG30	SW2_6	Rotation encoder inc_b
2	AH29	SW2_3/5	Rotation encoder push
3	G30	U45_2	Piezo
4	AK7	SW12	Switch East
5	U8	SW10	Switch North
6	V8	SW11	Switch South
7	AJ7	SW13	Switch West
8	G16	DS13	LED4
9	AD25	DS12	LED5
10	AD24	DS11	LED6
11	AE24	DS10	LED7
12	E8	DS24	LED Center
13	AG23	DS21	LED East
14	AF13	DS20	LED North
15	AG12	DS22	LED South
16	AF23	DS23	LED West
17	U25	SW8_1	DIP-Switch 1
18	AG27	SW8_2	DIP-Switch 2
19	AF25	SW8_3	DIP-Switch 3
20	AF26	SW8_4	DIP-Switch 4
21	AE27	SW8_5	DIP-Switch 5
22	AE26	SW8_6	DIP-Switch 6
23	AC25	SW8_7	DIP-Switch 7
24	AC24	SW8_8	DIP-Switch 8

Table 2-9: GPIO pins

2.4.5.2. UART

The interface is provided for serial communications. The UART supports data frames with 8 data bits, one optional parity bit and one stop bit. To generate the bit-rate, each UART has a programmable 12-bit clock divider. Two FIFOs are used for data transfer between the APB bus and UART, when fifosize VHDL generic > 1. Two holding registers are used data transfer between the APB bus and UART, when fifosize VHDL generic = 1. Hardware flow-control is supported through the RTSN/CTSN handshake signals, when flow VHDL generic is set. Parity is supported, when parity VHDL generic is set.

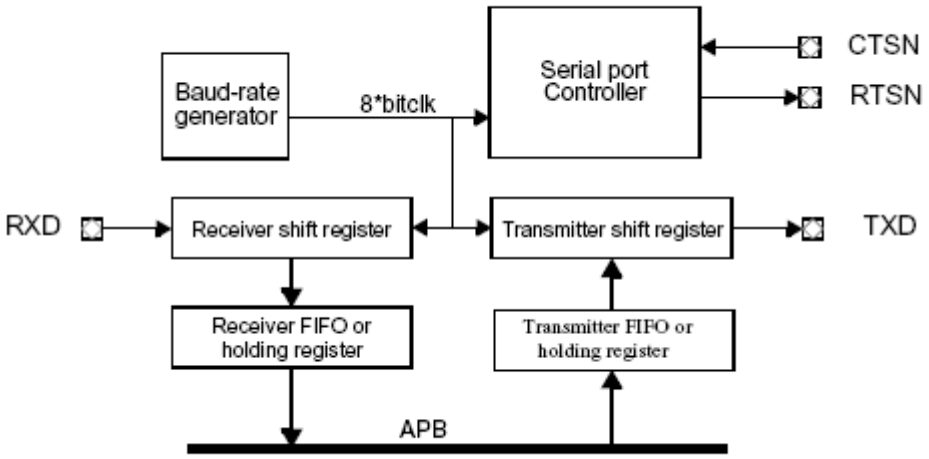


Figure 2-14: APB UART overview

UART signal	FPGA pin	Connector pin	Signal
TXD	F10	J61_3 (via MAX232)	APBUART TXD
RXD	G10	J61_2 (via MAX232)	APB UART RXD

2.4.5.2.1. Transmitter operation

The transmitter is enabled through the TE bit in the UART control register. Data that is to be transferred is stored in the FIFO/holding register by writing to the data register. This FIFO is configurable to different sizes via the fifosize VHDL generic. When the size is 1, only a single holding register is used but in the following discussion both will be referred to as FIFOs. When ready to transmit, data is transferred from the transmitter FIFO/holding register to the transmitter shift register and converted to a serial stream on the transmitter serial output pin (TXD). It automatically sends a start bit followed by eight data bits, an optional parity bit, and one stop bit. The least significant bit of the data is sent first.

2.4.5.2.2. Receiver operation

The receiver is enabled for data reception through the receiver enable (RE) bit in the UART control register. The receiver looks for a high to low transition of a start bit on the receiver serial data input pin. If a transition is detected, the state of the serial input is sampled a half bit clocks later. If the serial input is sampled high the start bit is invalid and the search for a valid start bit continues. If the serial input is still low, a valid start bit is assumed and the receiver continues to sample the serial input at one bit time intervals (at the theoretical centre of the bit) until the proper number of data bits and the parity bit have been assembled and one stop bit has been detected. The serial input is shifted through an 8-bit shift register where all bits have to have the same value before the new value is taken into account, effectively forming a low-pass filter with a cut-off frequency of 1/8 system clock. The receiver also has a configurable FIFO which is identical to the one in the transmitter. As mentioned in the transmitter part, both the holding register and FIFO will be referred to as FIFO. During reception, the least significant bit is received first. The data is then transferred to the receiver FIFO and the data ready (DR) bit is set in the UART status register as soon as the FIFO contains at least one data frame. The parity, framing and overrun error bits are set at the received byte boundary, at the same time as the receiver ready bit is set. The data frame is not stored in the FIFO if an error is detected. Also, the new error status bits are or:ed with the old values before they are stored into the status register. Thus, they are not cleared until written to with zeros from the AMBA APB bus. If both the receiver FIFO and shift registers are full when a new start bit is detected, then the character held in the receiver shift register will be lost and the overrun bit will be set in the UART status register. If flow control is enabled, then the RTSN will be negated (high) when a valid start bit is detected and the receiver FIFO is full. When the holding register is read, the RTSN will automatically be reasserted again.
When the VHDL generic fifosize > 1, which means that holding registers are not considered here, some additional status and control bits are available. The RF status bit (not to be confused with the RF control bit) is set when the receiver FIFO is full. The RH status bit is set when the receiver FIFO is half-full (at least half of the entries in the FIFO contain data frames). The RF control bit enables receiver FIFO interrupts when set. A RCNT field is also available showing the current number of data frames in the FIFO.

2.4.5.2.3. Baud-rate generation

Each UART contains a 12-bit down-counting scaler to generate the desired baud-rate. The scaler is clocked by the system clock and generates a UART tick each time it underflows. It is reloaded with the value of the UART scaler reload register after each underflow. The resulting UART tick frequency should be 8 times the desired baud-rate. If the EC bit is set, the scaler will be clocked by the external clock input rather than the system clock. In this case, the frequency of external clock must be less than half the frequency of the system clock.

2.4.5.2.4. Interrupt generation

Interrupts are generated differently when a holding register is used (VHDL generic fifosize = 1) and when FIFOs are used (VHDL generic fifosize > 1). When holding registers are used, the UART will generate an interrupt under the following conditions: when the transmitter is enabled, the transmitter interrupt is enabled and the transmitter holding register moves from full to empty; when the receiver is enabled, the receiver interrupt is enabled and the receiver holding register moves from empty to full; when the receiver is enabled, the receiver interrupt is enabled and a character with either parity, framing or overrun error is received. For FIFOs, two different kinds of interrupts are available: normal interrupts and FIFO interrupts. For the transmitter, normal interrupts are generated when transmitter interrupts are enabled (TI), the transmitter is enabled and the transmitter FIFO goes from containing data to being empty. FIFO interrupts are generated when the FIFO interrupts are enabled (TF), transmissions are enabled (TE) and the UART is less than half-full (that is, whenever the TH status bit is set). This is a level interrupt and the interrupt signal is continuously driven high as long as the condition prevails. The receiver interrupts work in the same way. Normal interrupts are generated in the same manner as for the holding register. FIFO interrupts are generated when receiver FIFO interrupts are enabled, the receiver is enabled and the FIFO is half-full. The interrupt signal is continuously driven high as long as the receiver FIFO is half-full (at least half of the entries contain data frames).

2.4.5.3. ADC

The core provides an AMBA AHB interface to the Xilinx System Monitor present in Virtex-5 FPGAs. All Xilinx System Monitor registers are mapped into AMBA address space. The core also includes functionality for generating interrupts triggered by System Monitor outputs, and allows triggering of conversion start via a separate register interface.

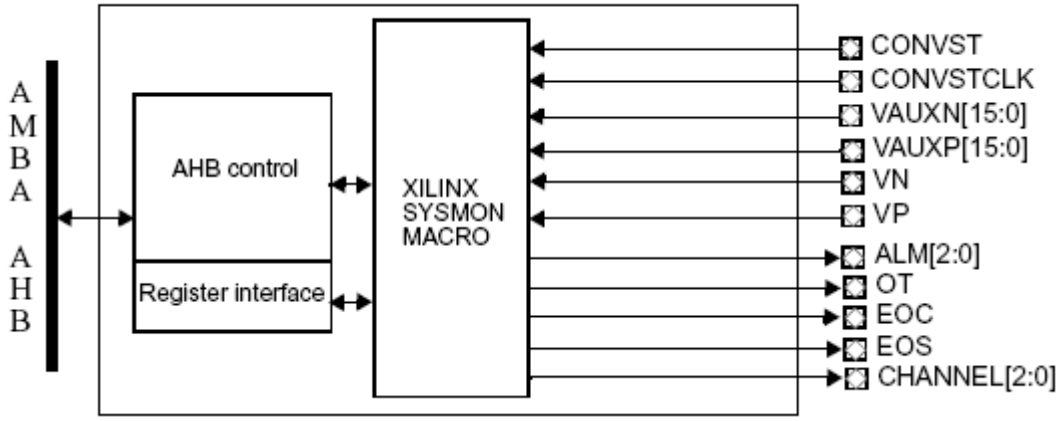


Figure 2-15: ADC block diagramm

ADC pin	FPGA pin	ML507 net	Header
VAUXN0	AE34	HDR2_42_SM_14_N	J4_42
VAUXP0	AF34	HDR2_44_SM_14_P	J4_44
VAUXN1	AE33	HDR2_46_SM_12_N	J4_46
VAUXP1	AF33	HDR2_48_SM_12_P	J4_48
VAUXN2	AB33	HDR2_58_SM_4_N	J4_58
VAUXP2	AC33	HDR2_60_SM_4_P	J4_60
VAUXN3	AB32	HDR2_54_SM_13_N	J4_54
VAUXP3	AC32	HDR2_56_SM_13_P	J4_56
VAUXN4	AD34	HDR2_50_SM_5_N	J4_50
VAUXP4	AC34	HDR2_52_SM_5_P	J4_52
VAUXN5	Y34	HDR1_30	J6_30
VAUXP5	AA34	HDR1_26	J6_26
VAUXN6	AA33	HDR2_38_SM_6_N	J4_38
VAUXP6	Y33	HDR2_40_SM_6_P	J4_40
VAUXN7	V34	HDR2_34_SM_15_N	J4_34
VAUXP7	W34	HDR2_36_SM_15_P	J4_36
VAUXN8	V33	HDR2_30_DIFF_3_N	J4_30
VAUXP8	V32	HDR2_32_DIFF_3_P	J4_32
VAUXN9	U31	HDR2_26_SM_11_N	J4_26
VAUXP9	U32	HDR2_28_SM_11_P	J4_28
VAUXN10	T34	HDR2_22_SM_10_N	J4_22
VAUXP10	U33	HDR2_24_SM_10_P	J4_24
VAUXN11	R32	HDR2_18_DIFF_2_N	J4_18
VAUXP11	R33	HDR2_20_DIFF_2_P	J4_20
VAUXN12	R34	HDR2_14_DIFF_1_N	J4_14

VAUXP12	T33	HDR2_16_DIFF_1_P	J4_16
VAUXN13	N32	HDR2_10_DIFF_0_N	J4_10
VAUXP13	P32	HDR2_12_DIFF_0_P	J4_12
VAUXN14	K32	HDR2_6_SM_7_N	J4_6
VAUXP14	K33	HDR2_8_SM_7_P	J4_8
VAUXN15	K34	HDR2_2_SM_8_N	J4_2
VAUXP15	L34	HDR2_4_SM_8_N	J4_4
VN	V17	FPGA_V_N	J9_10
VP	U18	FPGA_V_P	J9_9

Table 2-10: ADC pins

2.4.5.3.1. Operational model

The core has two I/O areas that can be accessed via the AMBA bus; the core configuration area and the System Monitor register area.

2.4.5.3.2. Configuration area

The configuration area, accessed via AHB I/O bank 0, contains two registers that provide status information and allow the user to generate interrupts from the Xilinx System Monitor's outputs. Write accesses to the configuration area have no AHB wait state and read accesses have one wait state. To ensure correct operation, only word (32-bit) sized accesses should be made to the configuration area.

2.4.5.3.3. System Monitor register area

The System Monitor register area is located in AHB I/O bank 1 and provides a direct-mapping to the System Monitor's Dynamic Reconfiguration Port. The System Monitor's first register is located at address offset 0x00000000 in this area.
*Since the System Monitor documentation defines its addresses using half-word addressing, and AMBA uses byte-addressing, the addresses in the System Monitor documentation should be multiplied to get the correct offset in AMBA memory space. If the Configuration register bit WAL is '0' the address in System Monitor documentation should be multiplied by two to get the address mapped by the AMBA wrapper. A System Monitor register with address n is at AMBA offset 2*n. If the Configuration register bit WAL is '1', all registers start at a word boundary and the address in the System Monitor documentation should be multiplied by four to get the address mapped in AMBA address space. In this case, a System Monitor register with address n is at AMBA offset 4*n. The wrapper always makes a single register access as the result of an access to the System Monitor register area. The size of the AMBA access is not checked and to ensure correct operation the mapped area should only be accessed using half-word (16-bit) accesses. If the core has been implemented with AMBA split support, it will issue a SPLIT response to all accesses made to the mapped System Monitor registers.*
For a description of the System Monitor's capabilities and configuration, please refer to the Xilinx Virtex-5 FPGA System Monitor User Guide.

2.4.5.4. Timer

The General Purpose Timer Unit provides a common prescaler and decrementing timer(s). Number of timers is configurable through the ntimers VHDL generic in the range 1 to 7. Prescaler width is configured through the sbits VHDL generic. Timer width is configured through the tbits VHDL generic. The timer unit acts a slave on AMBA APB bus. The unit implements one 16 bit prescaler and 3 decrementing 32 bit timer(s). The unit is capable of asserting interrupt on timer(s) underflow. Interrupt is configurable to be common for the whole unit or separate for each timer.

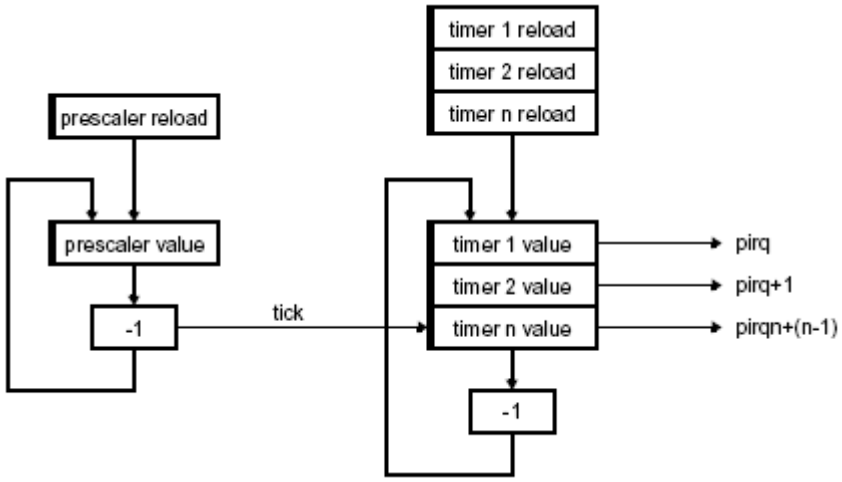


Figure 2-16: Timer overview

The prescaler is clocked by the system clock and decremented on each clock cycle. When the prescaler underflows, it is reloaded from the prescaler reload register and a timer tick is generated. The operation of each timer is controlled through its control register. A timer is enabled by setting the enable bit in the control register. The timer value is then decremented on each prescaler tick. When a timer underflows, it will automatically be reloaded with the value of the corresponding timer reload register if the restart bit in the control register is set, otherwise it will stop at -1 and reset the enable bit.
The timer unit can be configured to generate common interrupt through a VHDL-generic. The shared interrupt will be signalled when any of the timers with interrupt enable bit underflows. The timer unit will signal an interrupt on appropriate line when a timer underflows (if the interrupt enable bit for the current timer is set), when configured to signal interrupt for each timer. The interrupt pending bit in the control register of the underflown timer will be set and remain set until cleared by writing '0'.
To minimize complexity, timers share the same decrementer. This means that the minimum allowed prescaler division factor is ntimers+1 (reload register = ntimers) where ntimers is the number of implemented timers.
By setting the chain bit in the control register timer n can be chained with preceding timer n-1. Decrementing timer n will start when timer n-1 underflows.
Each timer can be reloaded with the value in its reload register at any time by writing a 'one' to the load bit in the control register. The last timer acts as a watchdog, driving a watchdog output signal when expired, when the wdog VHDL generic is set to a time-out value larger than 0. At reset, the scaler is set to all ones and the watchdog timer is set to the wdog VHDL generic, 0xFFF.

3. Installation

3.1. Xilinx ML507 boards

3.1.1. Board Interconnection

The hidICE synchronization interface is implemented by using the following resources:

Signals	Target	Emulation
Synchronization clock (differential)	J12 (p) J13 (n)	J10 (p) J11 (n)
Synchronization data (Mictor cable)	P22	P22
OCD connection	P22 (from emulation)	P3

Table 3-1: Board interconnection

IMPORTANT: Set J54 to OFF.

In a later version the Xilinx Rocket IO (high-speed communication link) can be used for synchronization.

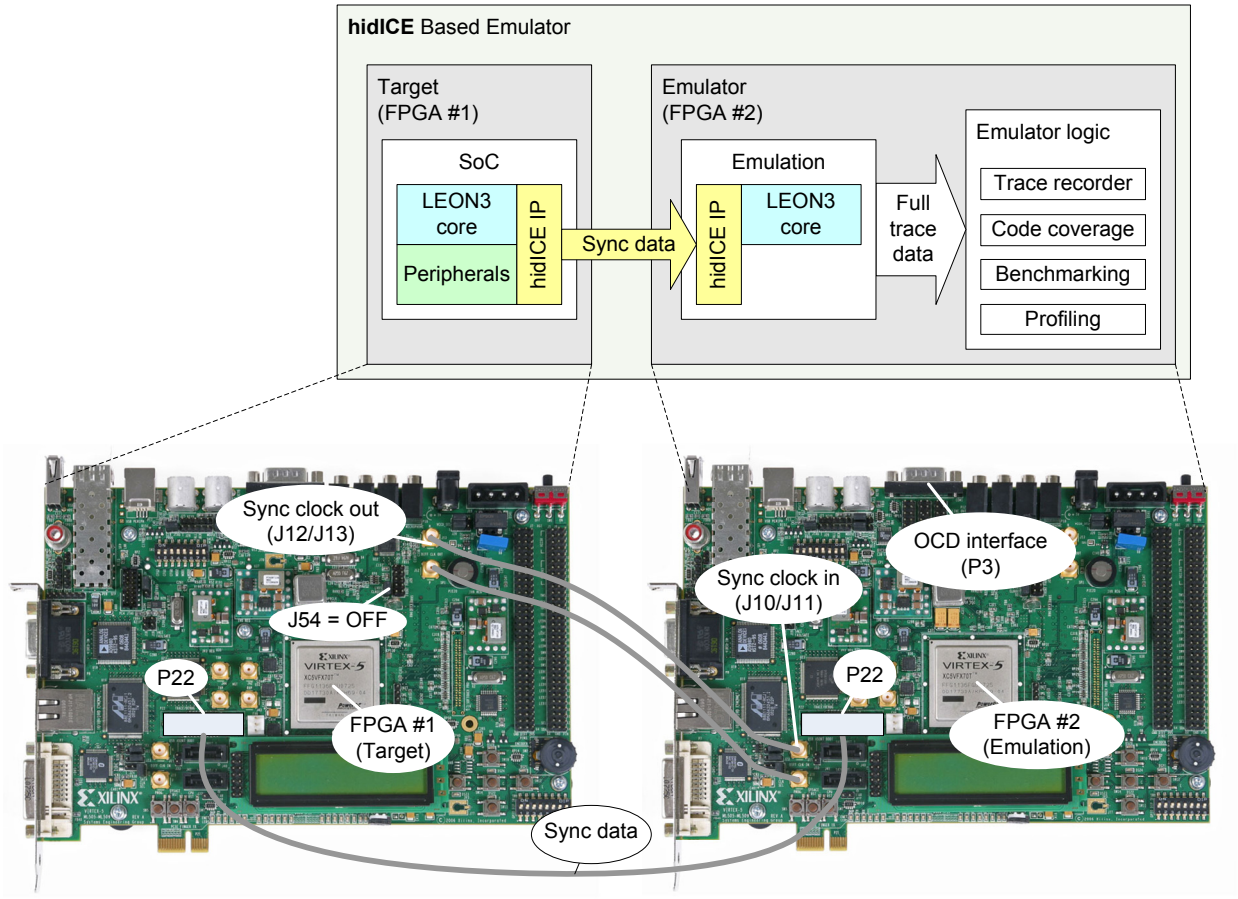


Figure 3-1: hidICE evaluation system. Set J54 to OFF.

3.1.2. Board modification

Both boards are connected by the Mictor cable (P22). The pin P22_14 is connected to the VCC3V3 net of each board.

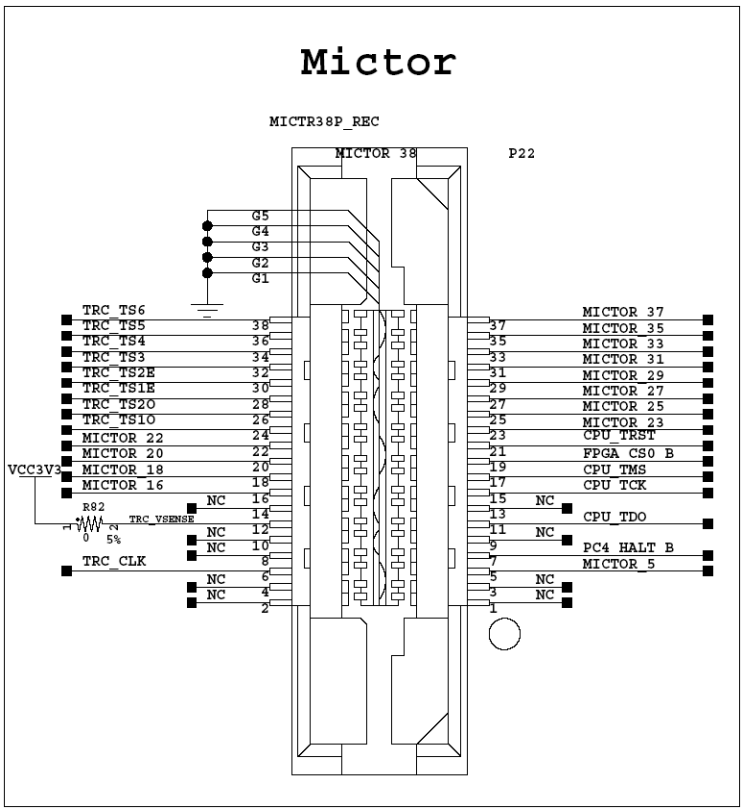


Figure 3-2: Mictor connector

To prevent a cross current between the boards, the resistor R82 should be removed from one of the boards. Alternatively, both boards should be powered by the same supply.

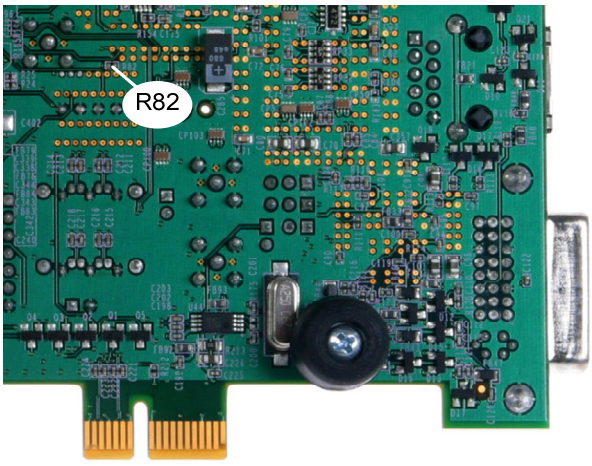


Figure 3-3: R82 location (bottom)

3.1.3. Power on / off sequence

If both boards are powered by different supplies (R82 removed), the following sequence should be used for power on and power off:

Power on:

1. Turn on the target board.
2. Turn on the emulation board.

Power off:

3. Turn off the emulation board.
4. Turn off the target board.

The target board checks the input level on the mictor_in[2] input (FPGA pin A24).
If the emulation board is not connected or not powered, the mictor_in[2] input is pulled down and disables all synchronization output signals of the target board.
If the emulation board is powered on, the mictor_in[2] input is driven high by the emulation board and enables the synchronization outputs.

3.1.4. Configuration Options

The hidICE demonstration system can be configured by the following major devices:

- Xilinx download cable (JTAG)
- Two Platform Flash PROMs
- System ACE controller (JTAG)

The configuration mode is controlled by the 8pol. DIP switch SW3

Switch (SW3)	Function	Description
1	Config Address [2]	
2	Config Address [1]	
3	Config Address [0]	
4	MODE [2]	
5	MODE [1]	
6	MODE [0]	
7	Not used	
8	System ACE Configuration (On = Enable, Off = Disable).	When enabled, the System ACE controller configures the FPGA from the CF card whenever a card is inserted or the SYSACE RESET button is pressed.

Table 3-2: Configuration settings by SW3

SW3 [4:6] Mode[2:0]	Mode
000	Master Serial (Platform Flash PROM, up to four configurations)
001	SPI (One configuration)
010	BPI Up (Parallel NOR Flash, up to four configurations)
011	BPI Down (Parallel NOR Flash, up to four configurations)
100	Master SelectMAP (Platform Flash PROM, up to four configurations)
101	JTAG (PC4, System ACE up to eight configurations)
110	Slave SelectMAP (Platform Flash PROM, up to four configurations)
111	Slave Serial (Platform Flash PROM, up to four configurations)

Table 3-3: Configuration mode settings by SW3[4:6]

3.1.4.1. Xilinx download cable (JTAG)

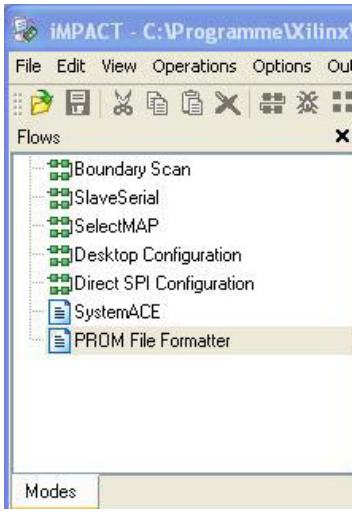
The JTAG configuration port for the board (J1) allows for device programming and FPGA debug. The JTAG port supports the Xilinx Parallel Cable III, Parallel Cable IV, or Platform USB cable products. Third-party configuration products might also be available. The JTAG chain can also be extended to an expansion board by setting jumper J21 accordingly.

3.1.4.2. Two Platform Flash PROMs

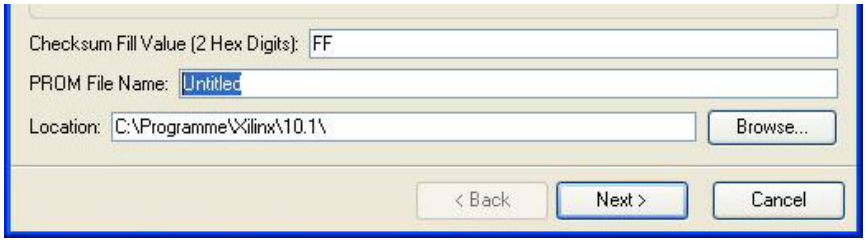
The two onboard Xilinx XCF32P Platform Flash PROM configuration storage devices offer a convenient and easy-to-use configuration solution for the FPGA. The Platform Flash PROM holds up to two separate configuration images (up to four with compression) that can be accessed through the configuration address switches SW3[3:1]]. To use the Platform Flash PROM to configure the FPGA, the configuration DIP switch SW3 must be set to the position 00011001.
The Platform Flash PROM can program the FPGA by using the master or slave configuration in serial or parallel (SelectMap) modes. The Platform Flash PROM is programmed using Xilinx iMPACT software through the board's JTAG chain.

1. Start Xilinx Impact V10.1

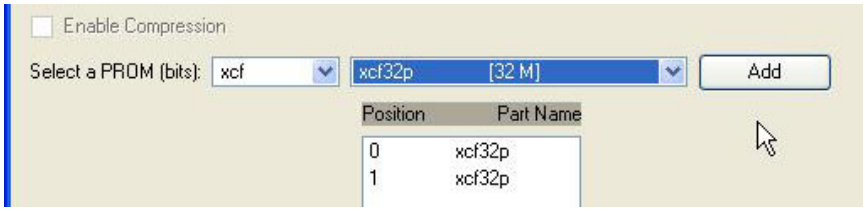
2. On the start screen double-click on „PROM File Formatter"



3. Select a PROM File Name and press "Next" two times

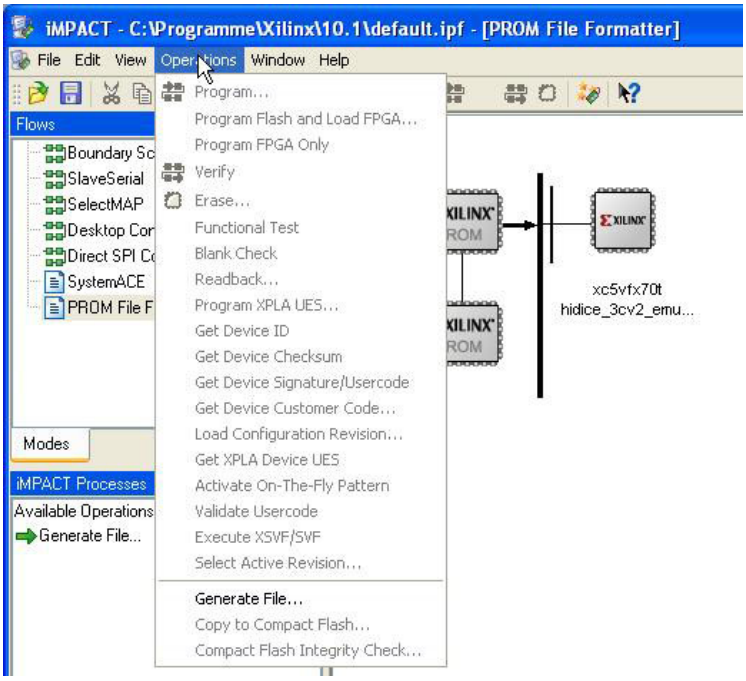


4. Add two (!) xcf32p and press "Next" and then "Finish" in the following window

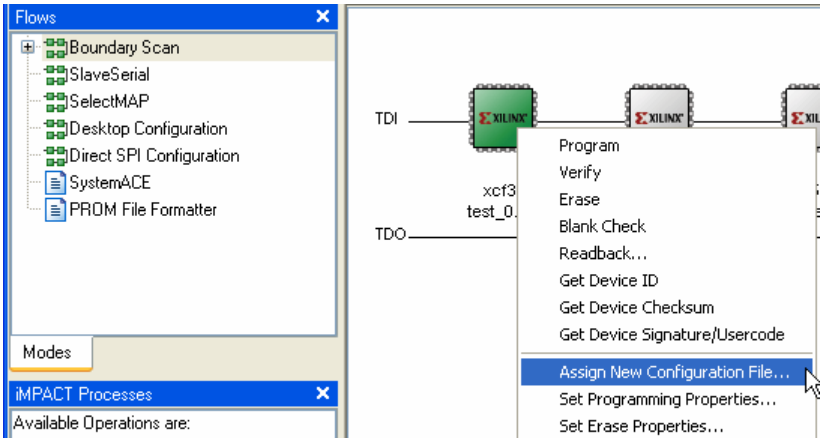


5. Click on "OK" and then select your bitfile, when asked for another one press "No"

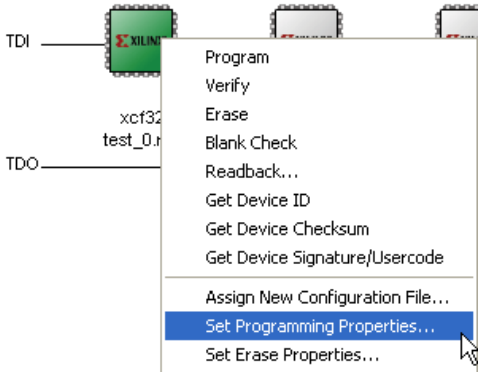
6. Now run "Operations -> Generate File"



7. In the Boundary Scan Window: right-click on the first xcf32p and select "Add new configuration file…" and choose "[PROM File Name]_0.mcs"

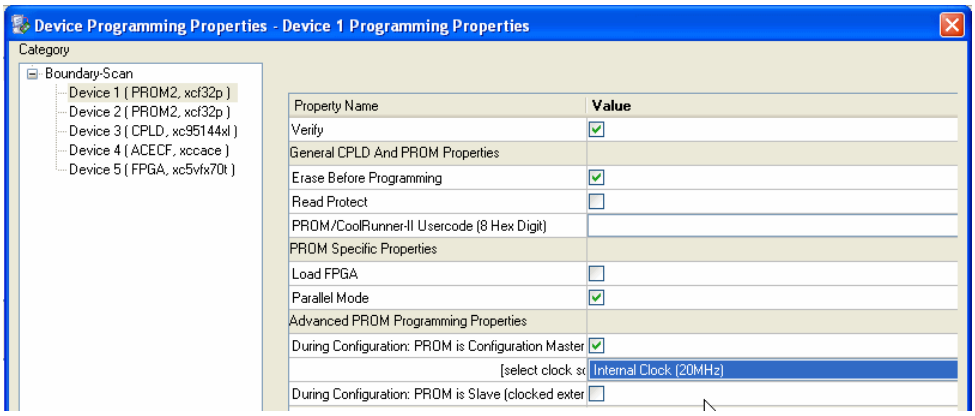


8. Right-click again and select "Set Programming Properties…"

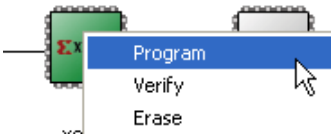


9. Set parameters according to the window: set "Verify",

set “Erase before Programming”, set “Parallel Mode”, set “During Configuration PROM is Configuration Master”, set Clock to “Internal Clock (20 MHz), jumper setting on the board has to be “00011001”



10. To finish, right-click on first xcf32p and select “Program”

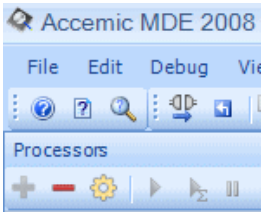


3.2. Accemic MDE software

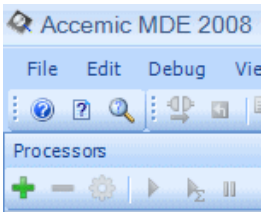
3.2.1. Getting started

1. Start Accemic MDE 2008

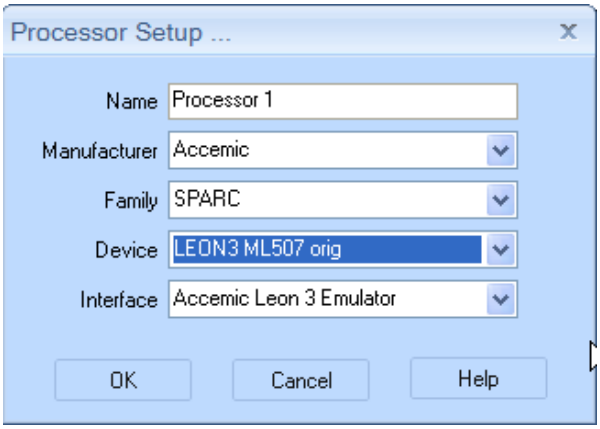
2. On the start screen delete all existing processors and projects by clicking the "minus" sign (bottom left in figure)



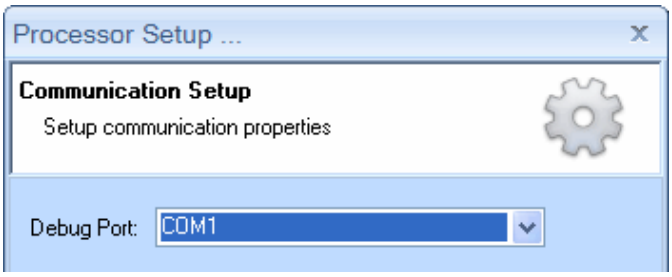
3. Now add the new processor by clicking on the "plus" sign (bottom left)



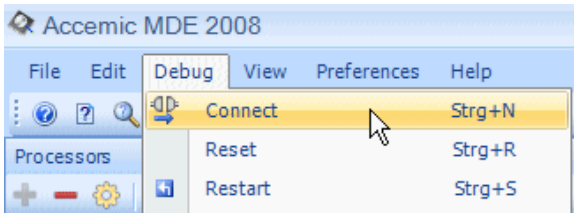
4. In the opening window choose the following settings: Manufacturer "Accemic", Family "SPARC", Device "LEON3 ML507 orig" and Interface "Accemic Leon 3 Emulator"



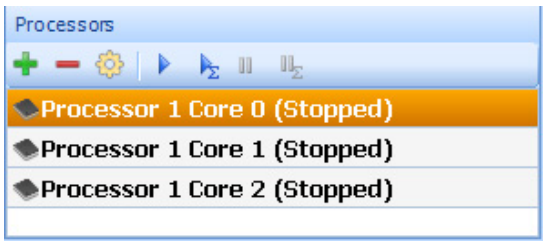
5. Click on "OK", in the next window just press "Next", in the following window select your serial port (COM1 here shown), then continue with "Next" and then "Finish" in next window



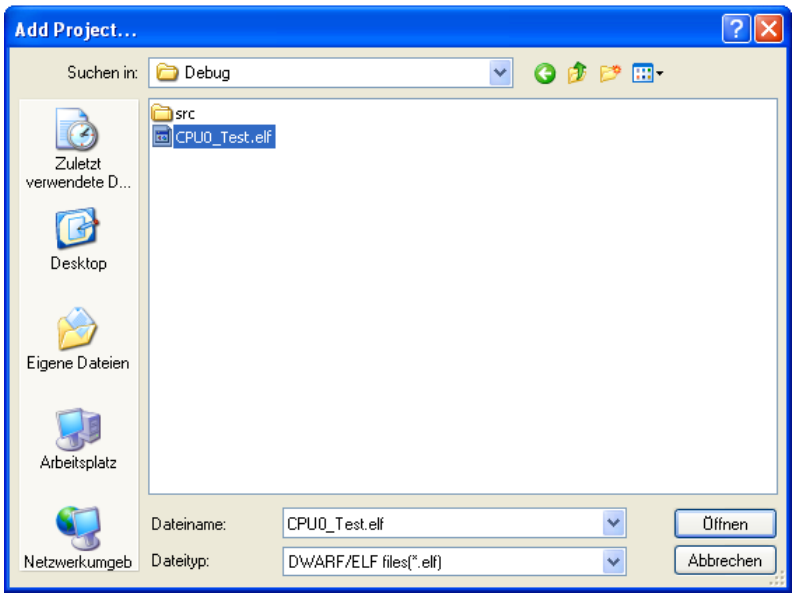
6. Click on "Debug" and choose "Connect" to connect the processors



7. Your processor window should now look like the screen on the right. Now you can add your project by again using the "plus" sign (bottom left)



8. Add project CPU0_Test (use ELF for file type) in the Project Explorer



9. Add project CPU1_Test (use ELF for file type) in the Project Explorer

10. Add project CPU2_DMA (use ELF for file type) in the Project Explorer

11. Drag CPU0_Test to Processor 1 Core 0

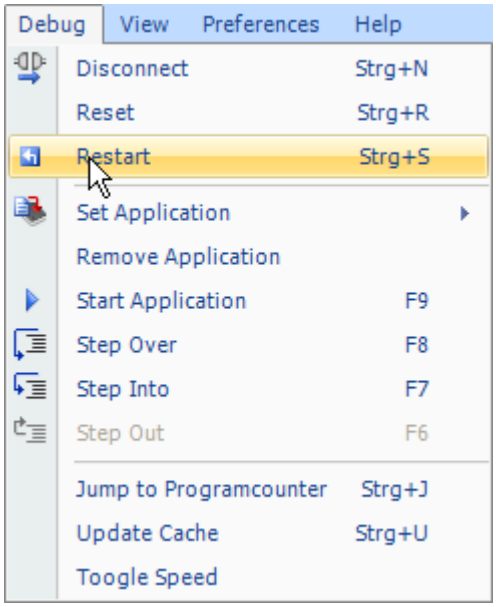
12. Drag CPU1_Test to Processor 1 Core 1

13. Drag CPU2_DMA to Processor 1 Core 2

15. Power on target board

16. Power on emulator board

17. Click on “Debug” and then “Restart”



18. To start all cores press 

19. To stop all cores press 

20. Press  if you want to display the AHB trace

21. Press  if you want to display the instruction trace

22. The menu item Debug->”Toggle speed” does switch between 50 and 5 MHz clock for the target board

3.2.2. Processor state window

The Processor Window allows the user to view and modify the processor's register fields in both symbolic and numeric format, which puts an end to the tedious process of searching through manual pages for register descriptions. When the user selects a register, the actual processor's memory is read and displayed.

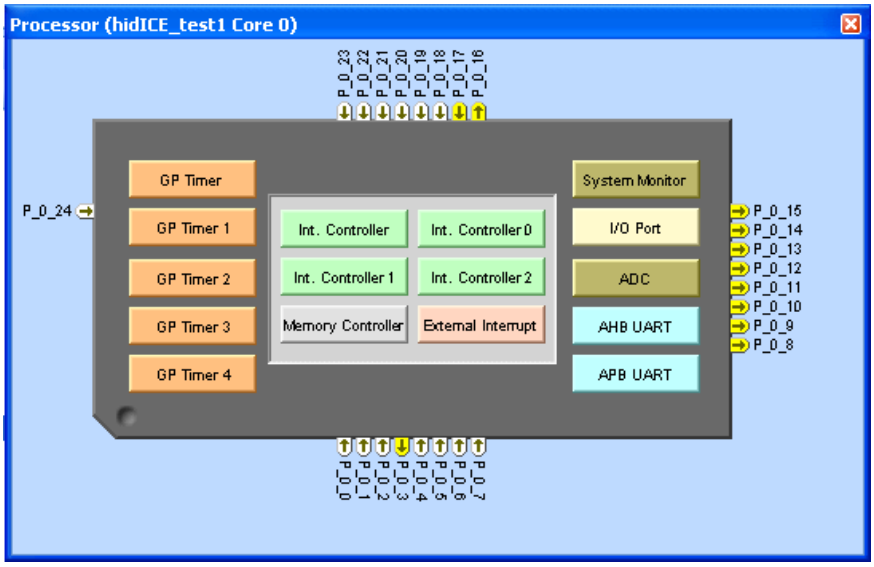


Figure 3-4: Processor state window

By clicking on a module an information window will appear, where specified information on the module is displayed. In addition, the visible registers of the selected module can be edited by double-clicking on them.

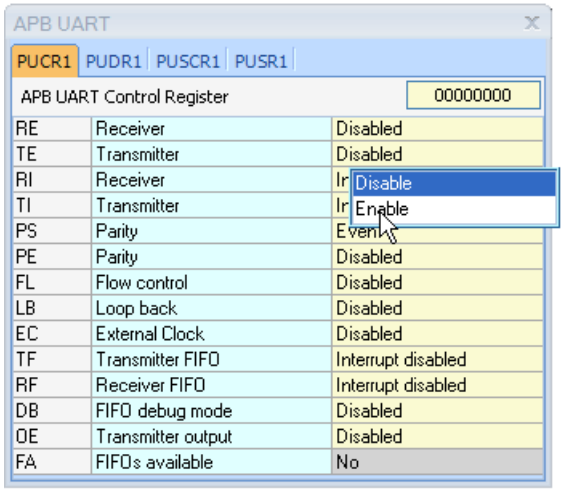


Figure 3-5: Processor state window resource

Around the core the state and the direction of the port pins are displayed. Used pins are displayed in light-grey. By moving the mouse over one of the pins, a hint will be displayed, that shows which port pin this is.
If the input pin state is low, the pin is displayed in white color. If the input pin state is high, the pin is displayed in yellow color.
By pressing the 'Strg' button and clicking on a pin which is not grey, the direction of the pin will be toggled.
By clicking on a pin which is not grey, the output value of the pin will be toggled.

Pin state	Pin colour	Actions available
Output low	White	Left click -> Set high Strg + Left click -> Set to input
Output high	Yellow	Left click -> Set low Strg + Left click -> Set to input
Input low	White	Strg + Left click -> Set to output
Input high	Yellow	Strg + Left click -> Set to output

Table 3-4: Processor state window I/O pin actions

I/O port pin	Direction	Signal
P_0_2	Input	Rotation encoder push
P_0_3	Output	Piezo
P_0_4	Input	Switch East
P_0_5	Input	Switch North
P_0_6	Input	Switch South
P_0_7	Input	Switch West
P_0_8	Output	LED4
P_0_9	Output	LED5
P_0_10	Output	LED6
P_0_11	Output	LED7
P_0_12	Output	LED Center
P_0_13	Output	LED East
P_0_14	Output	LED North
P_0_15	Output	LED South
P_0_16	Output	LED West
P_0_17	Input	DIP-Switch 1
P_0_18	Input	DIP-Switch 2
P_0_19	Input	DIP-Switch 3
P_0_20	Input	DIP-Switch 4
P_0_21	Input	DIP-Switch 5
P_0_22	Input	DIP-Switch 6
P_0_23	Input	DIP-Switch 7
P_0_24	Input	DIP-Switch 8

Table 3-5: I/O pins overview

3.3. Sample Projects

3.3.1. Target Board Buttons and LEDs

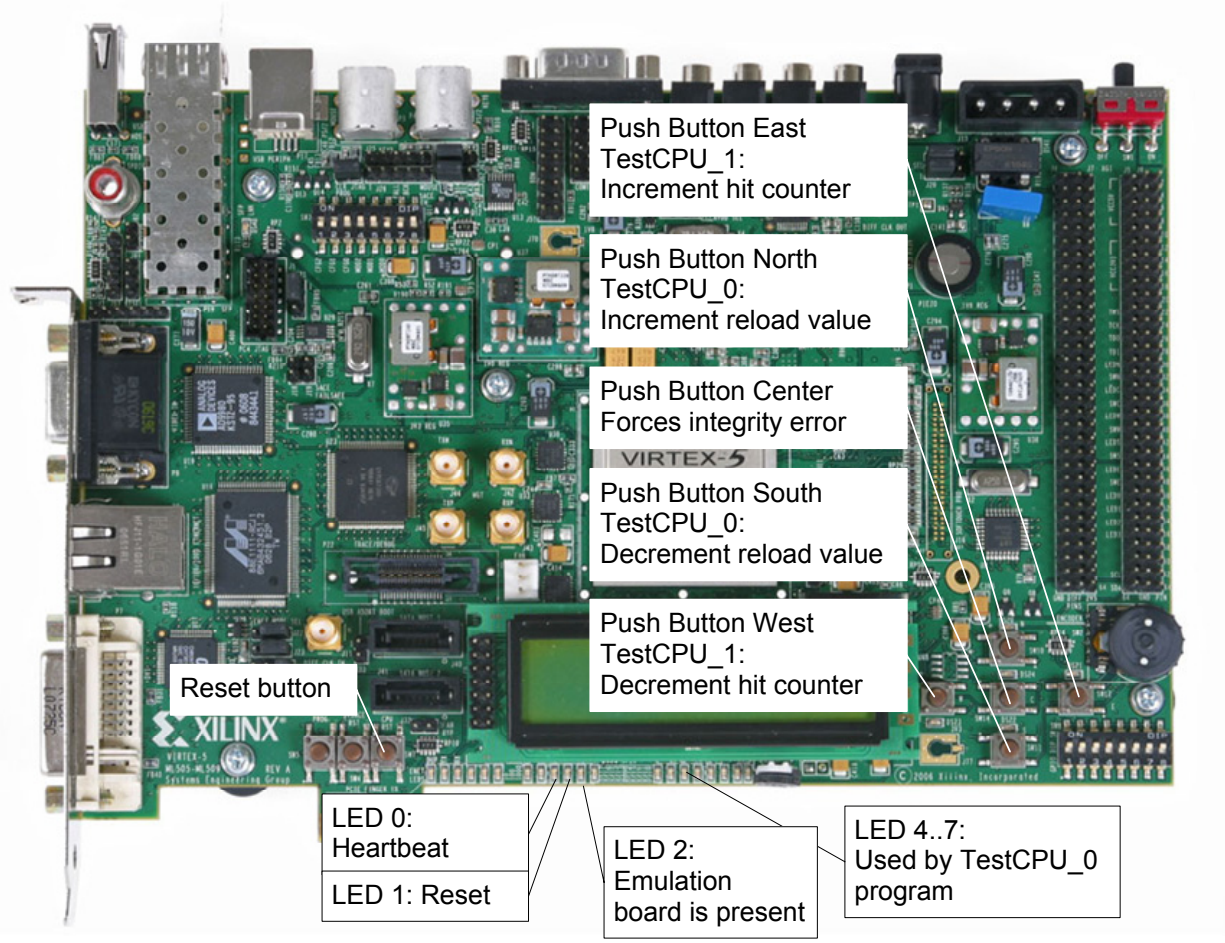


Figure 3-6: Target LEDs / Buttons for sample projects

3.3.2. Emulation Board LEDs

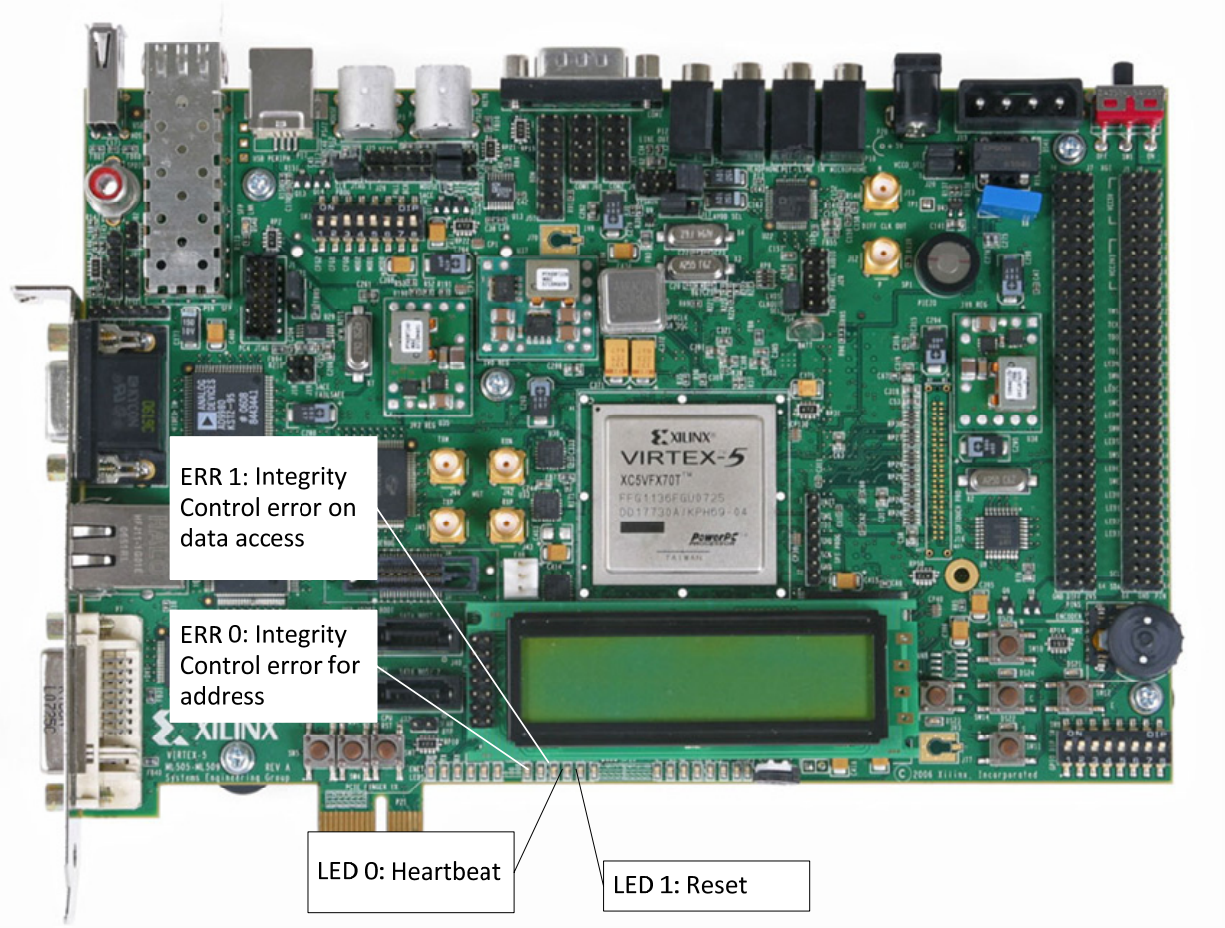


Figure 3-7: Emulation LEDs for sample projects

3.3.3. CPU0_Test

- Interrupt handling for the General Purpose Timer 1. On each overrun the timer is reloaded and the LEDs 4 to 7 are incremented.
- Interrupt handling for the south and north buttons. On each hit the reload value for Timer 1 is incremented or decremented.
- Test procedures for single stepping testing and C definition.
- The unit TestDMA does test the DMA Transfer. It initializes an array (DMASourceData) and starts the transfer. When finished the DMADestinationData does contain the same data as DMASourceData. A transfer is restarted if the DMA is ready.

3.3.4. CPU1_Test

- Interrupt handling for the west and east buttons. On each hit a counter (hitCounter) is incremented or decremented. The behavior can easily be observed by using the online watch function in the global variable window, when the application is running.
- Test procedures for single stepping testing and C definition.

3.3.5. CPU2_DMA

CPU2_DMA contains a software implemented DMA controller.
A transfer is started by writing the transfer size and the ENABLE_DMAA bit to the DMACA register. After that the controller transfers the amount of words from the source address (DMASA) to the destination (DMADA). After each operation the source and destination addresses are incremented. When the last word is transferred the DMACA register is cleared.

```
//DMA Controller
#define DMA_ADR 0x40030000

// Control/status register A
#define DMACA (volatile unsigned int*)(DMA_ADR+0)
#define ENABLE_DMAA 0x10000
//Transfer source address register
#define DMASA (volatile unsigned int*)(DMA_ADR+8)
//Transfer destination address register
#define DMADA (volatile unsigned int*)(DMA_ADR+12)

/*--------------------------------------------------------------------------

FUNCTION: DMALoop

DESCRIPTION: Waits until ENABLE_DMAA is set in DMACA; Starts transfer
       from DMASA to DMADA. Size is defined in lowest 16 of DMACA.

--------------------------------------------------------------------------*/
void DMALoop()
{
      unsigned int toTransfer;
      unsigned int* psource;
      unsigned int* pdest;

      *DMACA=0;

      while(1)
      {
            if(*DMACA & ENABLE_DMAA)
            {
                  toTransfer = *DMACA & 0xFFFF;
                  psource = (int*)*DMASA;
                  pdest = (int*)*DMADA;

                  while(toTransfer--)
                  {
                        *pdest++ = *psource++;
                  }
                  *DMACA = 0;/* signal end of transfer */
            }
      }
}
```

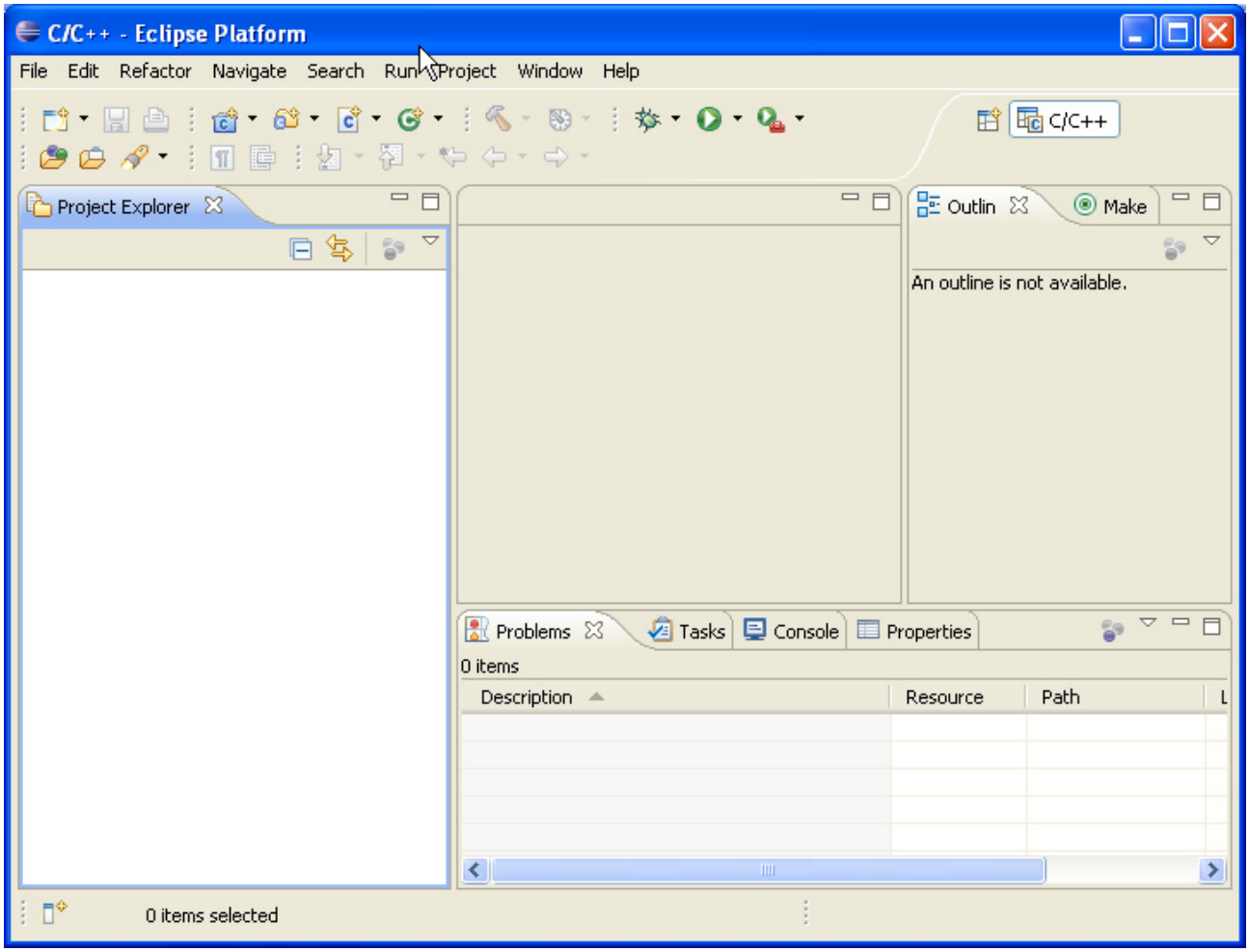
Figure 3-8: Software DMA controller code snippet

3.3.6. CPU(x)_Template

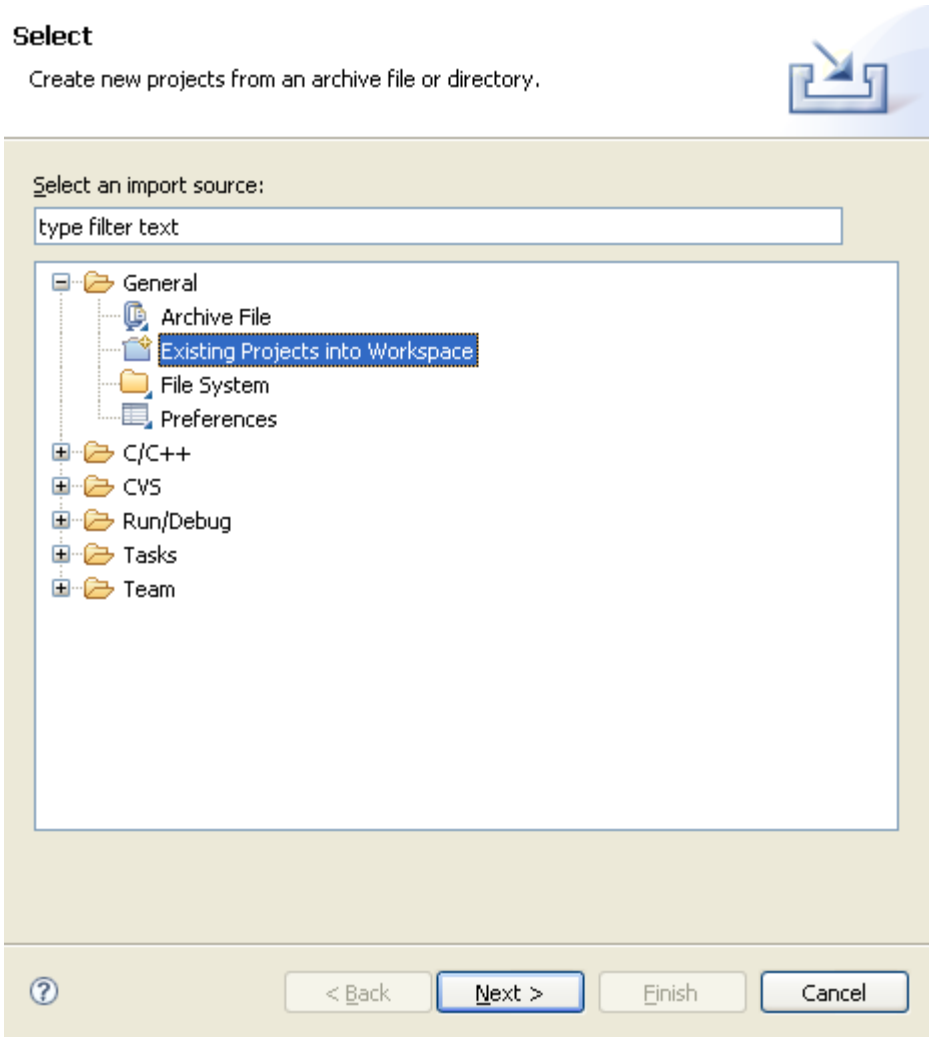
For each CPU a template project is provided, which includes the settings for correct linking to the CPU specific start address range: 0x40000000+(x)*0x10000.

3.4. Load and modify the LEON3 template projects

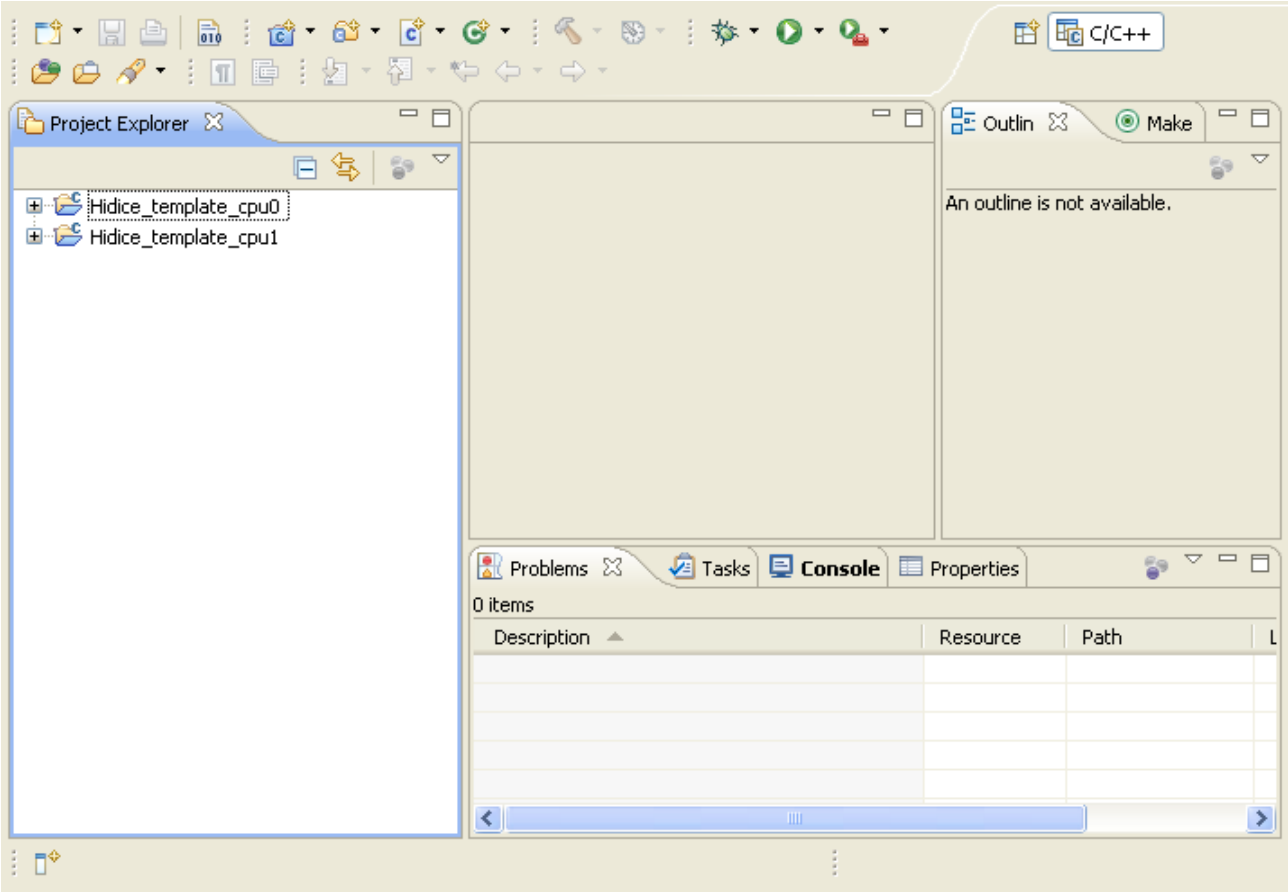
- First you have to install Java (www.java.com)
- Then you can install the Toolchain for the Leon-Processor from Aero-Flex Gaisler. (www.gaisler.com). It's called GRTools.
- Now you can open Eclipse that comes with this toolchain. It looks like the next picture.



- Click „File“ → „Import“. The following window opens. Please choose „Existing Projects into Workspace”.



- Select the Hidice_template_cpu0 and select „copy into workspace"
- Go 2 steps back and do the same for the Hidice_template_cpu1. Your workspace now should look like as follows:



- When you click right on the two projects you can change the names with „rename"
- Click „File" → „New" → „Source File"
- Now you can add your source code.
- When you are ready, click „Project" → „Build Project"
- Now you can find the output-file in your Workspace → Project-Name → Debug (or Release)

3.5. Gaisler software

GRLIB is distributed as a gzipped tar-file and can be installed in any location on the host system.

The distribution has the following file hierarchy:

- *bin* *various scripts and tool support files*
- *boards* *support files for FPGA prototyping boards*
- *designs* *template designs*
- *doc* *documentation*
- *lib* *VHDL libraries*
- *netlists* *Vendor specific mapped netlists*
- *software* *software utilities and test benches*
- *verification* *test benches*

4. Appendix

4.1. Further development

The following tasks are still under development:

- Real time trace analysis
- hidICE Supported Software Self Tests
- Pin Count Optimization / Port Reconstruction
- NEXUS class 3 interface

4.1.1. Real-Time Trace Analysis

The hidICE technology enables the access to extensive trace data. All AHB bus signals can be monitored.
At each CPU / bus clock a few hundred signals will be available. This huge amount of data requires a new approach for analysis. In this task we will implement a programmable trace analyzer with the following key features:

- High level language programmable
- AHB bus analyzer
- Microcode sequencer

The analyzer can provide the following outputs:

- Trigger output for trace data capturing
- Breakpoints
- Performance measurement
- Runtime reflection

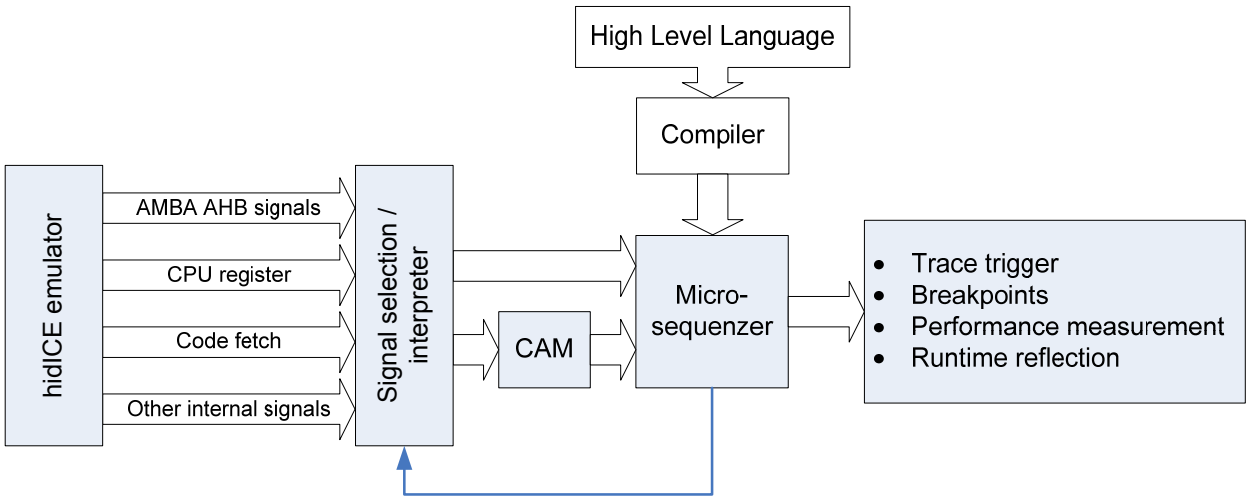


Figure 4-1: Trace analyzer

4.1.2. hidICE Supported Software Self Tests

TBD

4.1.3. Pin Count Optimization / Port Reconstruction

TBD

4.1.4. NEXUS class 3 interface

In this task the emulator will be equipped with an industry standard NEXUS class 3 interface.

The hidICE emulation provides all trace information required for NEXUS class 3 trace (and also much more trace information, which are not specified for the NEXUS interface, such as DMA and CPU register trace).

By providing the NEXUS class 3 interface, already available 3rd party tools can be used to access the trace data provided by the hidICE emulation. This approach is reasonable as a fast and cost-efficient start-up with the hidICE technology. In further steps, the interface can be expanded to access all possible trace information, provided by hidICE.

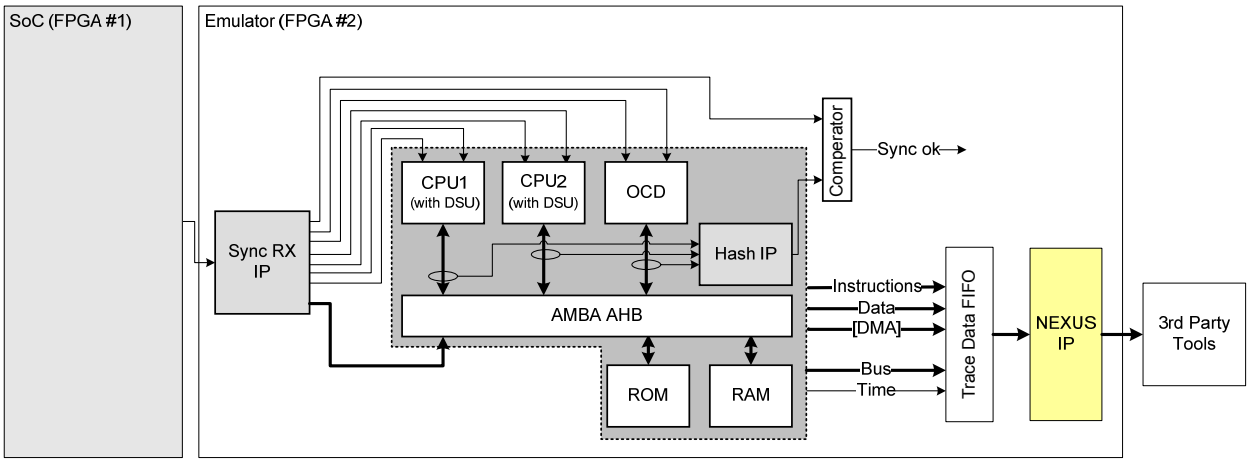


Figure 4-2: hidICE Emulator with Nexus interface

4.2. Issues and Limitations

Please note that the Accemic MDE for SPARC V8 version is a beta release and not completely tested.

4.3. References

GRLIB IP Library User's Manual, Version 1.0.20, Gaisler Research, 2009.
http://gaisler.com/products/grlib/grlib.pdf

GRLIB IP Core User's Manual, Version 1.0.20, Gaisler Research, 2009.
http://gaisler.com/products/grlib/grip.pdf

GRLIB VHDL source code, Version 1.0.20, Gaisler Research, 2009.
http://www.gaisler.com/products/grlib/grlib-gpl-1.0.20-b3403.tar.gz

BCC - Bare-C Cross-Compiler User's Manual
included in ftp://gaisler.com/gaisler.com/grtools/GRTools-20081001.exe

Virtex-5 FXT FPGA ML507 Evaluation Platform
http://www.xilinx.com/products/devkits/HW-V5-ML507-UNI-G.htm

ML505/ML506/ML507 Evaluation Platform User Guide
http://www.xilinx.com/support/documentation/boards_and_kits/ug347.pdf

Hochberger, C.; Weiss, A., "A new methodology for debugging and validation of soft cores," Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on , vol., no., pp.551-554, 8-10 Sept. 2008
http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4630006&isnumber=4629890

Hochberger, C.; Weiss, A., "Acquiring an exhaustive, continuous and real-time trace from SoCs," Computer Design, 2008. ICCD 2008. IEEE International Conference on , vol., no., pp.356-362, 12-15 Oct. 2008
http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4751885&isnumber=4751825

4.4. Part List

Part	Quantity	Supplier	Order number
Xilinx ML507 board	2	Xilinx	HW-V5-ML507-UNI-G
RF Coax cable, SMA M/M cable 12”	2	Digikey	744-1276-ND
Mictor cable 18“ M/M	1	Emulation Technology, Inc.	MIC-38-CABLE-MM-18
Cable null modem DB9M to DB9F	1	Digikey	AE9880-ND
Accemic MDE for Sparc8	1	Accemic	MDE-2008-SPARC8
Accemic MDE Multiprocessor Support	1	Accemic	MDE-2008-MP
PC (1 COM port, 1 USB port)	1	Accemic	MDE-2008-MP
Xilinx ML507 board for NEXUS class3 interface, not implemented yet (optionally)	1	Xilinx	HW-V5-ML507-UNI-G
ISE™ Foundation™ Software (optionally)	1	Xilinx	EF-ISE-FND
ChipScope Pro™ Tool (optionally)	1	Xilinx	EF-CSP-PRO
Platform Cable USB II (optionally)	1	Xilinx	HW-USB-II-G

Table 4-1: hidICE demonstration system part list

Accemic offers full assembled demonstration systems and individual training and support.

4.5. Revision History

Revision	Comment
2.1.	- First public version
2.2.	- Processor state window added - SYSMON (ADC) added - APB UART routed to COM2
2.3.	- Board modification and power on / off sequence description, required for independent power supplies - Extended description of the system integrity control - Description of the processor state window added

Table 4-2: Revision history

4.6. Current Versions

This manual corresponds with the following versions:

System	Software version
Target board	hidICE_3CV2_tar_V23_0.mcs
Emulation board	hidICE_3CV2_emu_V23_0.mcs
Demonstration software	V2.3
Accemic MDE	V3.5.1

Table 4-3: Current software versions