INICORE

Datasheet

VME64S

VME SLAVE CONTROLLER

Version 3.0.3

INICORE INC.
5600 Mowry School Road
Suite 180
Newark, CA 94560
t: 510 445 1529 *f:* 510 656 0995 *e:* info@inicore.com
www.inicore.com

Table of Contents

Table of Figures

Revision History

Version	*Comment*
3.0.3	• Removed faulty info about user-side data bus width setting, page 21
3.0.2	• Inconsistency in signal names corrected
3.0.1	• Added missing vme_berr_n input
3.0	• Global document update to reflect new top-level wrappers for A24D16, A24D32, and A32D32 targets. • Updated signal waveforms • Added user access decode example

Definition of Terms

Following conventions are used in this document:

- Signals ending with '_n' are active low.
- Signals containing a '_int_' are internal signals between the VME core and the FPGA/ASIC I/O buffer.

1 Overview

This VME64 slave controller is designed for custom integration using standard FPGA and ASIC technologies. It is fully compliant to the VME specification supporting A16/A24/A32 address mode, D8/D16/D32 data modes (read/write/read-modify-write), D16-BLT, D32-BLT, D64-MBLT, as well as interrupt acknowledge cycles. VMEbus timing is guaranteed by using a system clock of 40 MHz or higher. A synchronous design approach is used to simplify interfacing to the asynchronous VMEbus. The user side interface is full synchronous. Data access is either single cycle or multi-cycle controlled through user wait states.

To support VME slave controller implementations that do not require the full 32-bit address and data bus width, different top-levels are available. Features such as BLT and MBLT can individually be selected to achieve gate-count optimized implementations.

1.1 Features

Following special features are available:

- Data modes: D8, D16, D16-BLT, D32, D32-BLT, D64-MBLT
- Address modes: A16, A24, A32
- Access modes: Read, write, read-modify-write
- Selectable rescinding DTACK
- Configurable D8, D16, or D32 interrupter
- Selectable little/big endian conversion
- Full synchronous user side interface for registers, peripherals, and memories
- User selectable wait-states

1.2 Deliverables

- RTL code
- Self-verifying system-level testbench
- Synthesis information
- User guide

1.3 Block Diagram

Following figure shows the main building block of the VME64S core complemented with some typical user logic modules:

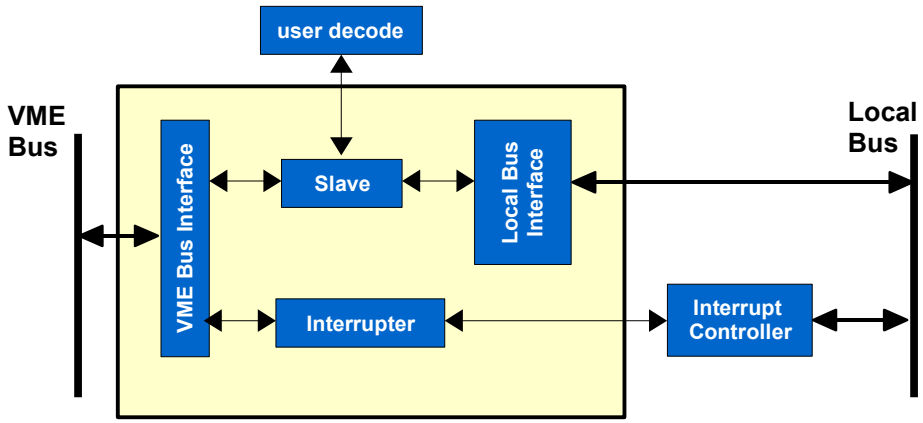

Figure 1: Block Diagram VME64S Slave Core

1.4 Implementation Options

Several different top-level modules are provided to support gate-count optimized implementations. Following table shows the supported feature set of each module.

	A16	A24	A32	D8	D16	D32	D64	BLT	MBLT	RMW	ROAK	RORA	DTACK
VME64S	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
VME64S_A32D32	✓	✓	✓	✓	✓	✓		✓		✓	✓	✓	✓
VME64S_A24D32	✓	✓		✓	✓	✓		✓		✓	✓	✓	✓
VME64S_A24D16	✓	✓		✓	✓			✓		✓	✓	✓	✓

2 Interface Signal Description

2.1 VME Slave Controller I/Os

2.1.1 VME64S Core I/Os

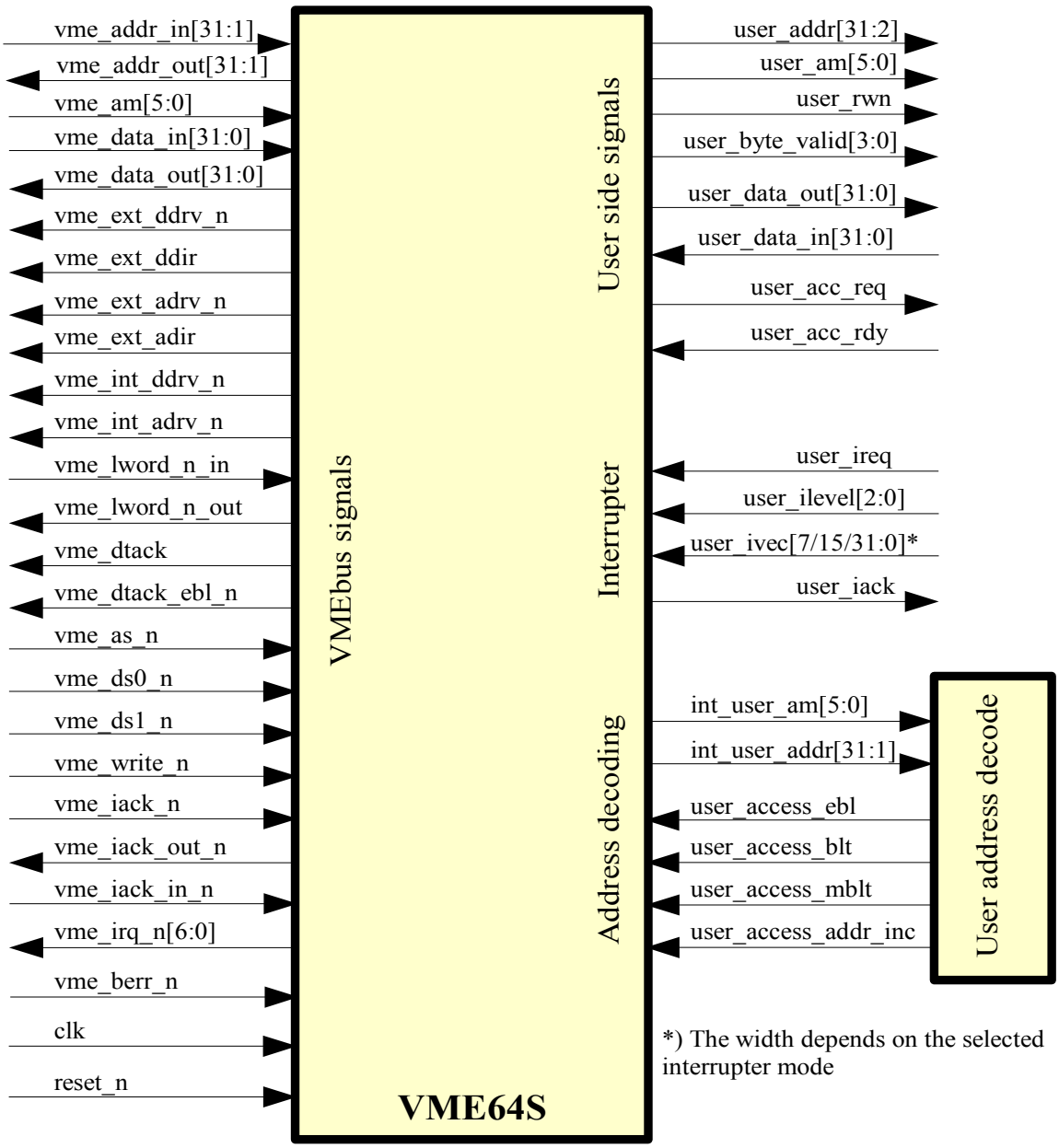

Figure 2: Inputs and Outputs of VME64S Core

2.1.2 VME64S_A32D32 Core I/Os

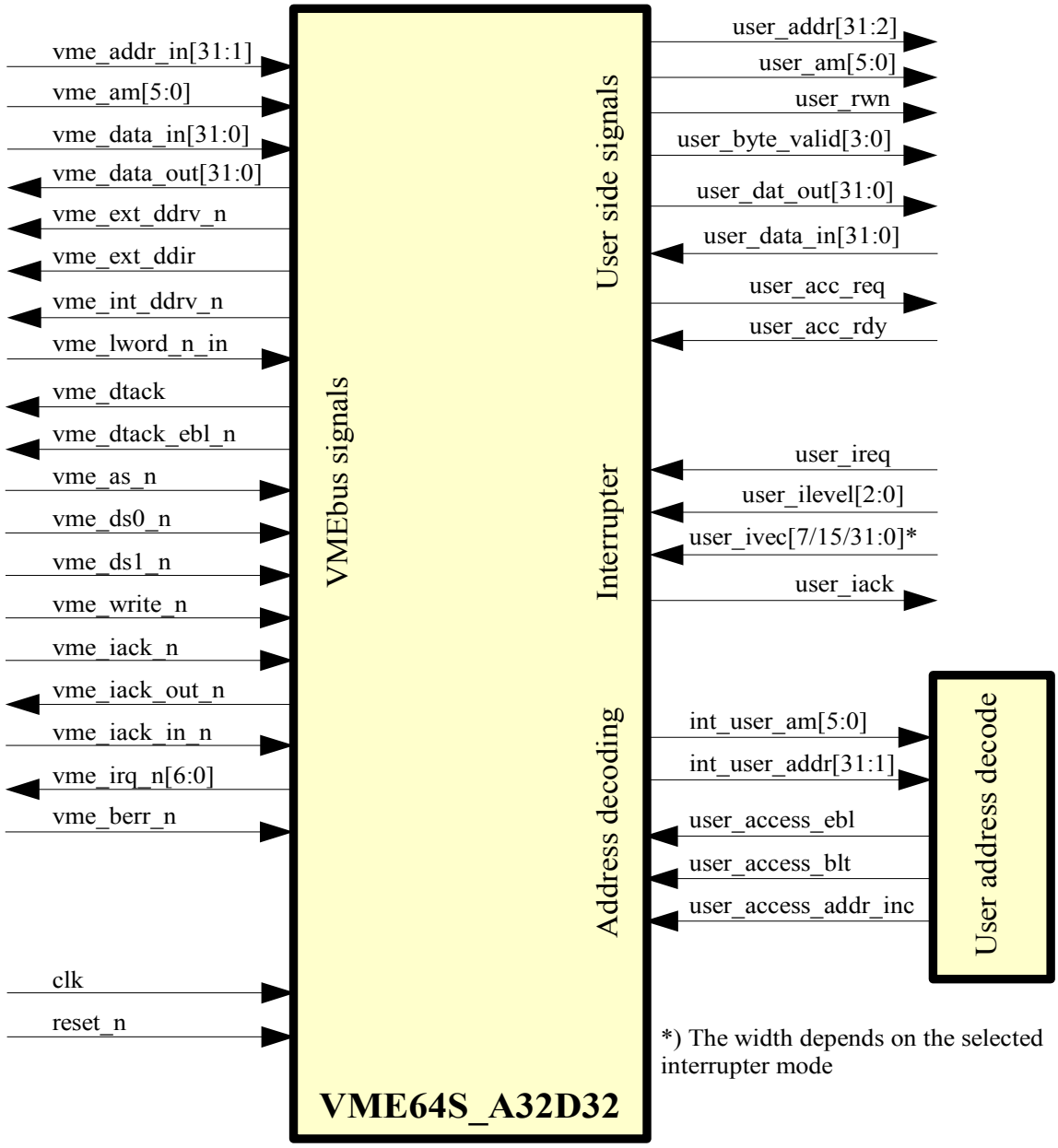

Figure 3: Inputs and Outputs of VME64S_A32D32 Core

2.1.3 VME64S_A24D32 Core I/Os

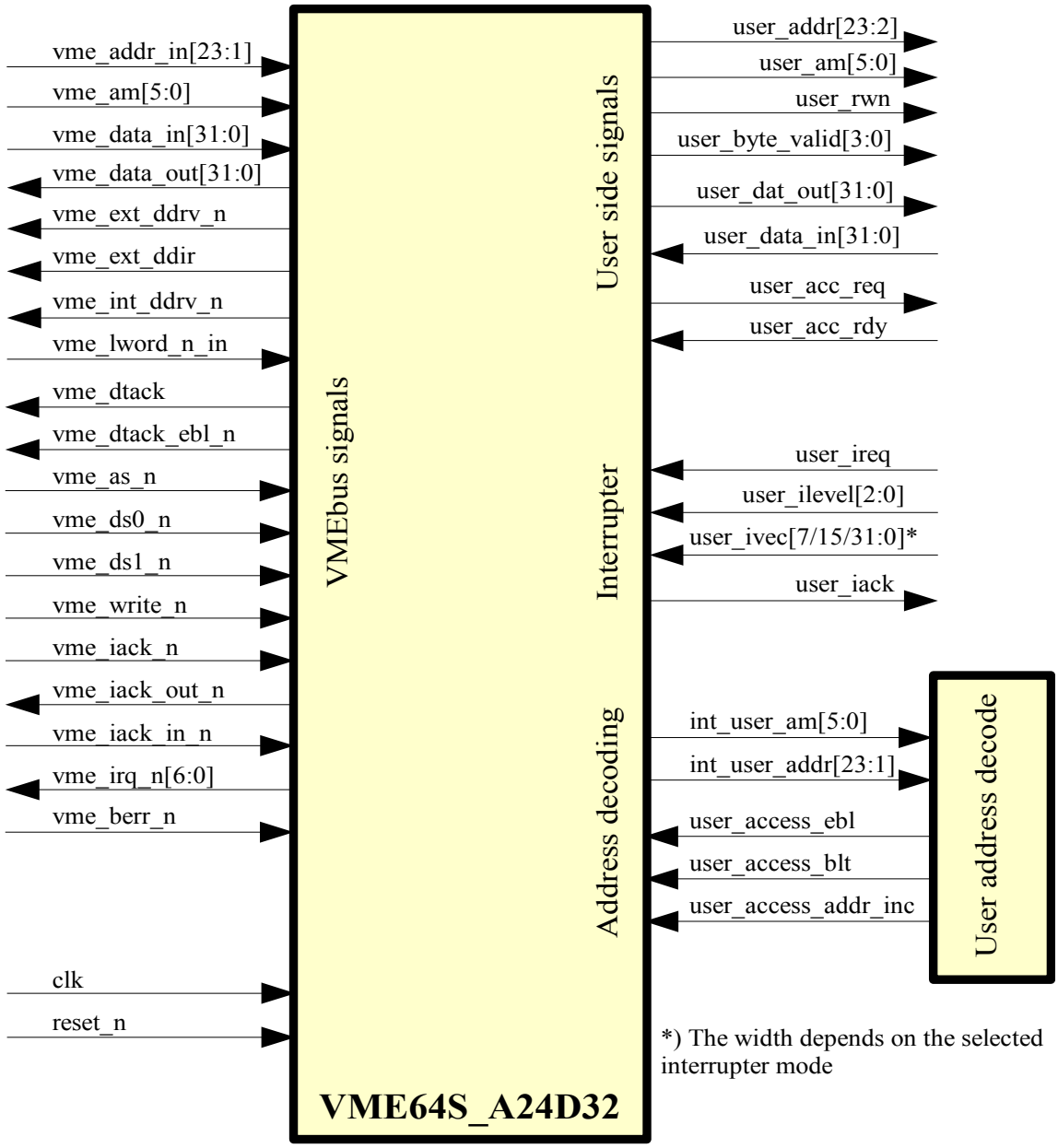

Figure 4: Inputs and Outputs of VME64S_A24D32 Core

2.1.4 VME64S_A24D16 Core I/Os

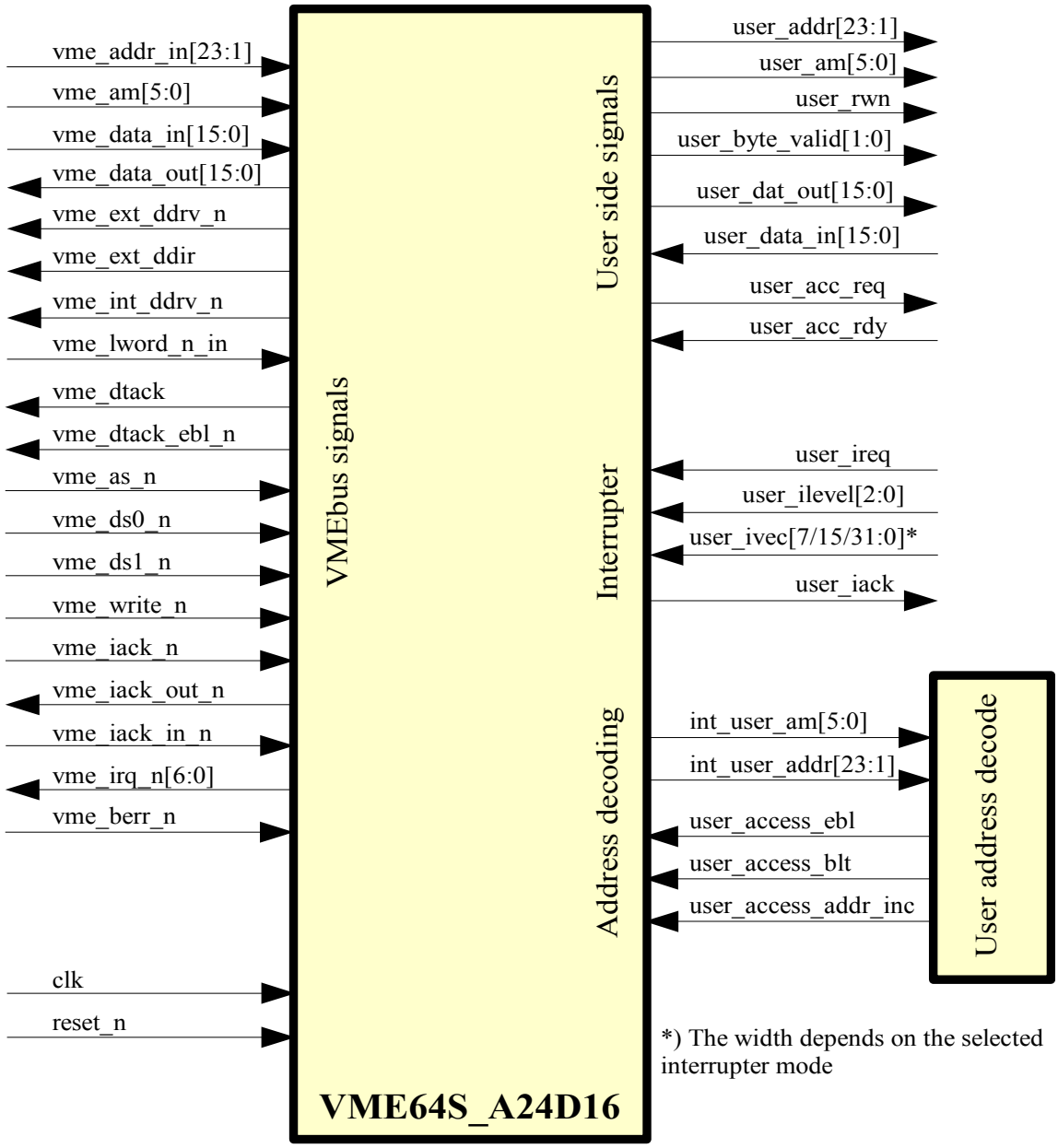

Figure 5: Inputs and Outputs of VME64S_A24D16 Core

2.2 Signal Description

The following paragraphs list the inputs and outputs of the VME slave controller and provides an overview of their functionality.

2.2.1 General Inputs

These pins are used to clock and initialize the whole VME core. To guarantee VME compliance, the falling edge of vme_as_n is used to latch the vme_addr_in and vme_am signals. The falling edge of the system clock 'clk' is used to guarantee interface timing on some signals. All other registers use the rising edge of 'clk' as the system clock. For proper operation of the VME interface, it is recommended that the system clock is 40MHz and higher.

Pin Name	*Type*	*Description*
clk	in	System clock
reset_n	in	Asynchronous system reset, active low

2.2.2 VME Bus

These pins are used to control data transfer through the VME interface.

Pin Name	*Type*	*Description*
vme_addr_in[23/31:1][1]	in	VME address bus input
vme_addr_out[31:1][2]	out	VME address bus output, used for MBLT
vme_am[5:0]	in	VME address modifier input
vme_data_in[15/31:0][3]	in	VME data bus input (from bus driver)
vme_data_out [15/31:0][3]	out	VME data bus output (goes to bus driver)
vme_ext_ddrv_n	out	Active low drive enable signal for external bidirectional data bus drivers.
vme_ext_ddir	out	Direction control signal for external bidirectional data bus drivers: ‘1’ to VME bus ‘0’ from VME bus
vme_ext_adrv_n[2]	out	Active low drive enable signal for external bidirectional address/lword drivers.
vme_ext_adir[2]	out	Direction control signal for external bidirectional address/lword drivers: ‘1’ to VME bus ‘0’ from VME bus

1 The address bus of an A24 slave controller is [23:1]. An A32 slave controller uses [31:1].

2 This signal is only available with the VME64 slave controller which supports MBLT.

3 The data bus of a D16 slave controller is [15:0] and [31:0] for a D32 slave controller.

Pin Name	Type	Description
vme_int_ddrv_n	in	Active low drive enable signal for internal bidirectional data bus drivers.
vme_int_adrv_n[2]	in	Active low drive enable signal for internal bidirectional address/lword drivers.
vme_lword_n_in	in	VME long word access indicator, low active
vme_lword_n_out[2]	out	VME long word access indicator output, used for MBLT
vme_dtack	out	Data transfer acknowledge. Used to indicate whether the DTACK is drive low or high (for rescinding)
vme_dtack_ebl_n	out	Data transfer acknowledge driver output, active low. This is the enable signal of the external DTACK driver.
vme_as_n	in	VME address strobe: clocks with falling edge the internal synchronization signals like vme_addr and vme_am. vme_as_n is also used as data signal for access start detection.
vme_ds0_n	in	Data strobe 0, active low
vme_ds1_n	in	Data strobe 1, active low
vme_write_n	in	Read/write signal, active low
vme_iack_n	in	Interrupt acknowledge, active low
vme_iack_in_n	in	Interrupt acknowledge daisy chain in, active low
vme_iack_out_n	out	Interrupt acknowledge daisy chain out, active low
vme_irq_n[6:0]	out	Interrupt, active low. Have to be connected to open collector driver.
vme_berr_n	in	VME bus error If the VME bus error is asserted, the VME slave controller aborts the current operation and returns to idle state. This is an input only as the slave does not generate errors.

2.3 User Side Interface

The VME core hides the entire VME synchronization logic from the local bus interface (or user side interface), which is fully synchronous. This simplifies integration of the core with the user application.

2.3.1 Local Bus Interface

Due to the synchronous local bus interface, VME interfacing becomes much easier. A simple request–acknowledge handshaking scheme, that supports user wait-states, is used to connect to the user logic.

In D32 implementations, the local bus is always 32-bit wide. VME cycles such as D08(OE) or D16 are mapped accordingly to the respective byte position in the 32-bit word. A D64-MBLT cycle is translated into two consecutive 32-bit local bus cycles. In D16 implementation, the local bus is 16-bit wide.

Pin Name	*Type*	*Description*
user_acc_req	out	Data access request Active high until user_acc_rdy acknowledges the request (or VME bus error occurs).
user_acc_rdy	in	User-side acknowledgment signal User side access is finished by asserting user_acc_rdy for one clock cycle.
user_addr[23/31:2][4]	out	Registered VME address bus
user_am[5:0]	out	Registered VME address bus modifier
user_data_out[15/31:0][5]	out	Local data bus that contains the data written to the user side. During a write operation, user_data_out is valid when usr_acc_req is asserted.
user_data_in[15/31:0][5]	in	Local data bus that contains the data read from the user side. During a read operation user_data_in must be valid when user_acc_rdy is asserted.
user_rwn	out	Data read/write_not indicator 0: Write 1: Read
user_byte_valid[1/3:0][5]	out	User data byte valid indicator Indicates which byte of the user_wr_data/ user_rd_data bus is valid or requested. [0]: user_wr_data[7:0] is valid [1]: user_wr_data[15:8] is valid [2]: user_wr_data[23:16] is valid [3]: user_wr_data[31:24] is valid

4 The address bus of an A24 slave controller is [23:2]. An A32 slave controller uses [31:2].

5 For a D16 slave controller, the user data bus is [15:0] and user_byte_valid is [1:0]

Local bus write cycle timing diagram

Following figure shows two local bus write cycles with different wait-states. While an access is in process, all signals coming from the VME core are stable. The end of the access is indicated by the backend logic by asserting user_acc_rdy.

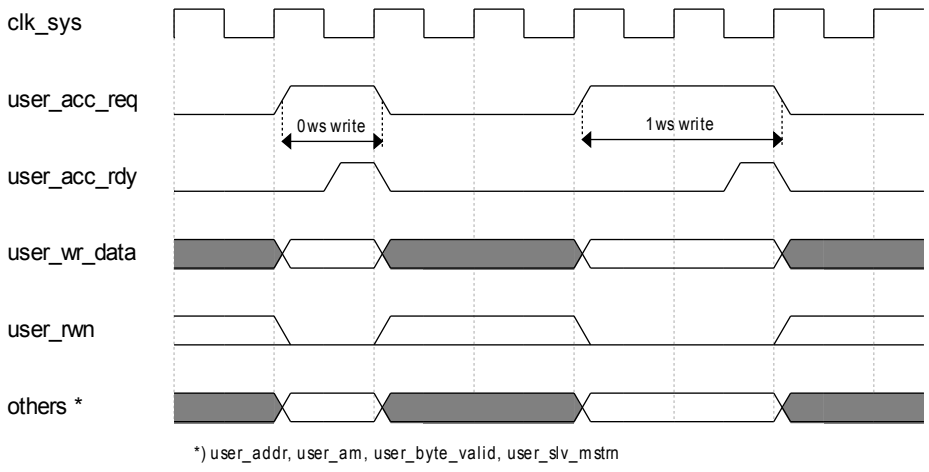

Figure 6: User write cycle with different wait-states

Local bus read cycle timing diagram

The local bus read cycle access is similar to the write cycle. While a read is performed, user_rd_data must be valid at the rising edge of the clock while user_acc_rdy is asserted.

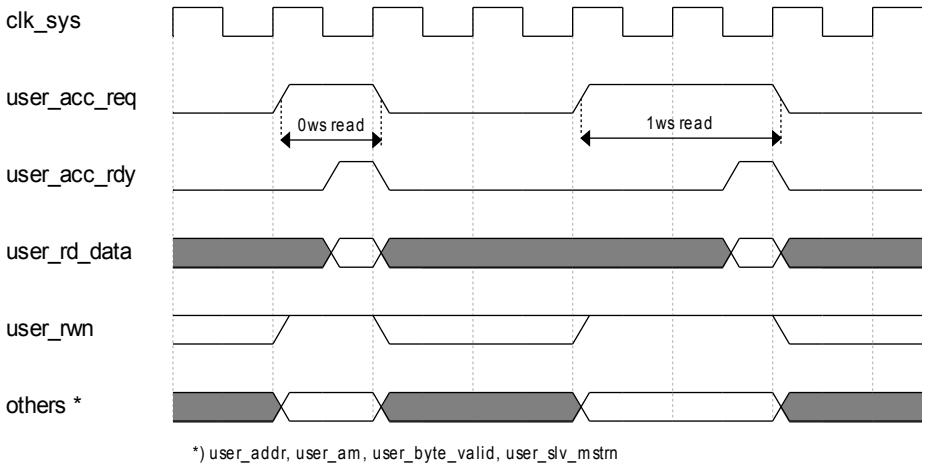

Figure 7: User read cycle with different wait-states

2.3.2 Slave Access Decoder

The slave access decoder is a module that is external to the core. It contains the access decode logic to select if the slave is addressed by the current VME bus cycle.

The signals int_user_addr and int_user_am allow a standard memory mapped address decoding scheme. The address modifiers shall be used to properly decode address mode and data transfer type.

Pin Name	*Type*	*Description*
int_user_addr[23/31:1]	out	Sampled VME address bus (by falling edge of vme_as_n)
int_user_am[5:0]	out	Sampled VME Address bus modifier (by falling edge of vme_as_n)
user_access_ebl	in	User access indicator This signal needs to be asserted if int_user_addr and int_user_am indicate that the current bus cycle addresses this slave.
user_access_blt	in	BLT user access indicator If asserted, the current bus cycle represents a block transfer *BLT' cycle that is supported by this slave.
user_access_mblt	in	MBLT user access indicator If asserted, the current bus cycle represents a multiplexed block transfer *MBLT' cycle that is supported by this slave.
user_access_addr_inc	in	Address increment indicator Defines if during a BLT or MBLT access, the address should be incremented. This is used when a user-side device such as a FIFO is at a fixed address but supports block type data transfers. 0: Address is not incremented 1: Address is incremented with each consecutive BLT or MBLT cycle

User access decoder timing diagram

The following figure shows the operation of the user decode module in relation to a VME cycle.

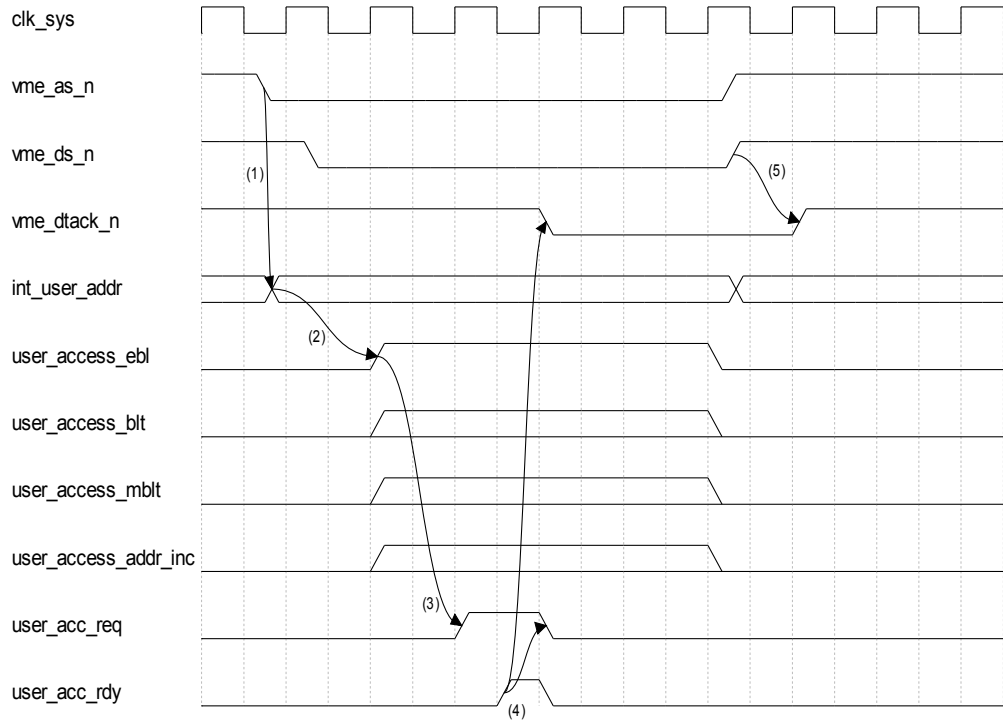

Figure 8: User-access decoder operation

1. The int_user_addr and int_user_am are latched with the falling edge of vme_as_n
2. If a valid access is detected by the user decode module, user_access_ebl, user_access_blt, user_access_mblt, and user_access_addr_inc are set according to the current cycle.
3. user_acc_req is asserted and remains asserted until the user logic terminates the access by asserting user_acc_rdy
4. Once user_acc_rdy is sampled high, the vme_dtack_n output is asserted.
5. When the VME cycle originator samples vme_dtack_n low, vme_ds_n is released to terminate the current VME cycle.

Example Access Decode Table

Following access decode table shows the decode operation of a user_decode module of an A32/D32 VME slave that supports single cycle, BLT, and MBLT access in both supervisory and non-privileged modes.

int_vme_am	int_vme_addr[6]	Description	user_access_		
			ebl	blt	mblt
0x0F	valid	A32 supervisory block transfer (BLT)	1	1	0
0x0E	valid	A32 supervisory program access	1	0	0
0x0D	valid	A32 supervisory data access	1	0	0
0x0C	valid	A32 supervisory 64-bit block transfer (MBLT)	1	0	1
0x0B	valid	A32 non-privileged block transfer (BLT)	1	1	0
0x0A	valid	A32 non-privileged program access	1	0	0
0x09	valid	A32 non-privileged data access	1	0	0
0x08	valid	A32 non-privileged 64-bit block transfer (MBLT)	1	0	1
0x08-0x0F	not valid	Not valid address range	0	0	0
others	valid	Unsupported cycles	0	0	0

2.3.3 Interrupter

The Interrupter block handles the generation of VME interrupt requests and acknowledgments of local interrupts. During an interrupt acknowledge cycle, the Interrupter returns the interrupt vector provided by the user side logic. The Interrupter module can generate VME interrupt request on one of the seven possible interrupt level.

Pin Name	*Type*	*Description*
user_ireq	in	Interrupt request Active one indicates that an interrupt is pending and a VME interrupt will be generated. Must return to zero with user_iack = 1.
user_iack	out	Interrupt acknowledgment An active one event indicates the end of a valid interrupt acknowledge cycle.
user_ilevel[2:0]	in	Interrupt level

6 A *valid* int_vme_addr indicates that the VME slaves supports access to this particular address.

Pin Name	Type	Description
user_ivec[7/15/31:0][7]	in	Interrupt vector Depending on the interrupter configuration, the core responds as D08(O), D16 or D32 interrupter. The width of this port is according to the selected interrupter mode.

Interrupt Acknowledge Cycles

Through a generic or parameter definition, it is possible to define to what kind of interrupt cycles this VME core responses. Following options are available

- D08(O) Interrupter: Responds to D08(O), D16 and D32 interrupt cycles
- D16 Interrupter: Responds to D16 and D32 interrupt cycles
- D32 Interrupter: Responds to D32 interrupt cycles

Interrupt Scheme

Interrupt requests to the VME bus are signaled by the active high user_ireq. Depending on the user_ilevel (interrupt level), the vme_irq_n(x) will be asserted. As soon as the interrupt is acknowledged by the VME bus, the user_iack event is asserted. If the interrupter uses the ROAK (Release On AcKnowledge) scheme, then the user_ireq has to be released immediately. If it uses the RORA (Release On Register Access) scheme then the user_ireq has to be released when the interrupt source is acknowledged.

Note: user_ilevel and user_ivec have to be stable for the whole time period where user_ireq is high.

7 The user_ivec bus width depends on the interrupter parameter settings.

Timing using ROAK scheme:

The user side logic releases user_ireq upon detection of user_iack.

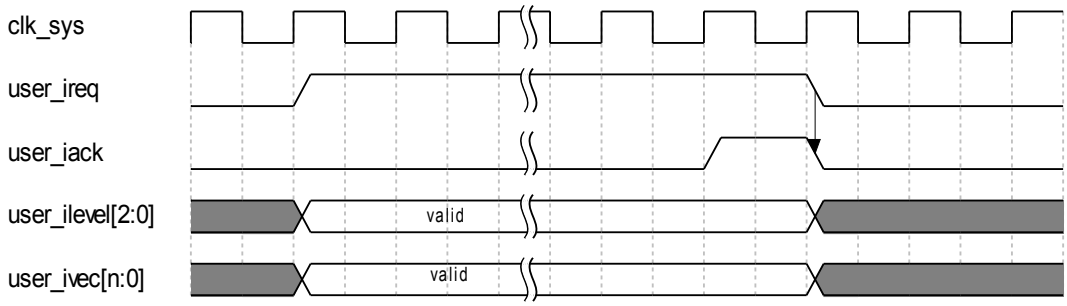

Figure 9: ROAK interrupting scheme

Timing using RORA scheme:

The user logic releases user_ireq upon the interrupt status register is cleared.

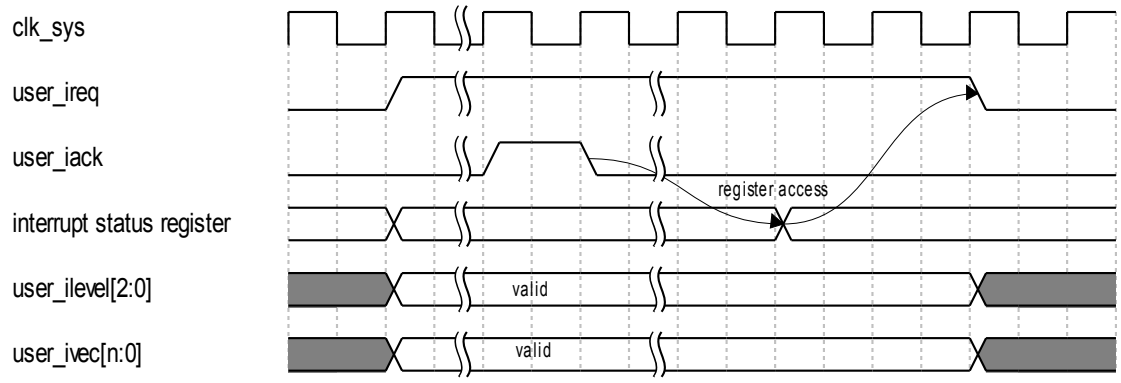

Figure 10: RORA interrupting scheme

2.3.4 Rescinding DTACK

The VME64 specification allows DTACK to be operated as a rescinding signal instead of an open-collector class signal. This results in an accelerated bus cycle. This feature can be selected through slave_config_dtack = '1'.

Timing diagram with open-collector DTACK:

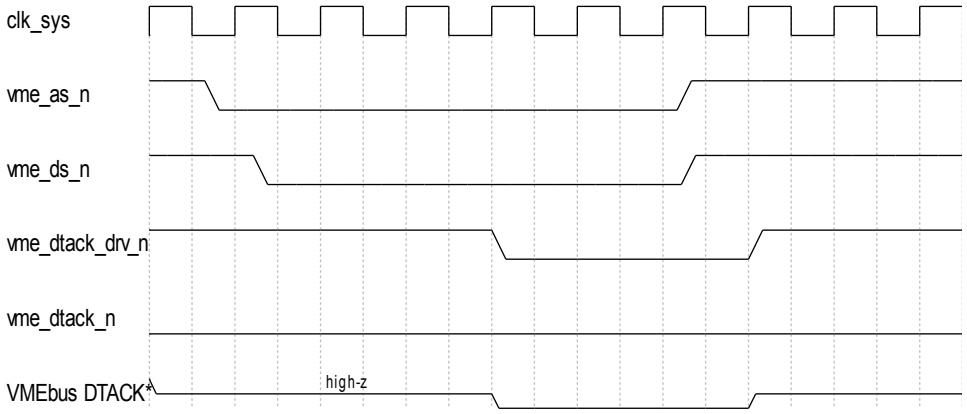

*Figure 11: Open-collector DTACK**

Timing diagram with rescinding DTACK:

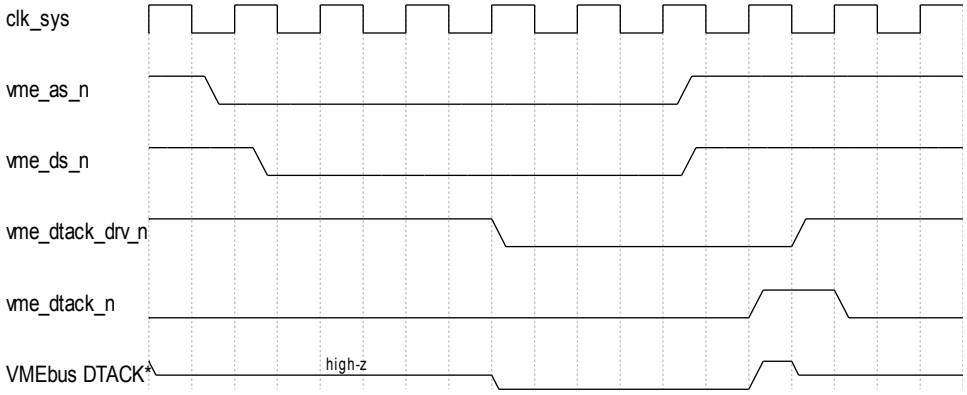

*Figure 12: Rescinding DTACK**

2.4 Configuration Parameters

The core can be configured and optimized for a particular application. Prior to synthesis, these parameters should be fixed according to the target application.

Please note that depending on the selected VME slave implementation, not all configuration parameter options are available.

Option	*Description*
interrupter	VME address bus input Interrupter selection 8: D08(O) type interrupter 16: D16 type interrupter 32: D32 type interrupter
endian	Endian selection for user side interface 0: Big endian, transparent 1: Little endian
rescinding_dtack	Rescinding DTACK enable The VME slave controller can use rescinding dtack to accelerate data transmission. ‘0’: Disabled ‘1’: Enabled
blt_ebl	Block transfer enable 0: Block transfer not supported 1: Block transfer supported
mblt_ebl	Multiplexed block transfer enable 0: Multiplexed block transfer not supported 1: Multiplexed block transfer supported
address_width	Address bus width 24: 24-bit address bus 32: 32-bit address bus
data_width	Data bus width 16: 16-bit address bus 32: 32-bit address bus

3 Appendix

3.1 Selecting proper I/O drivers

The VME standard requires some special high-drive drivers. Following is a list of critical signals and their respective driver requirements:

- VME AM and WRITE*
  The VME AM and WRITE* signals need to be driven by a standard three-state driver with a low-state sink current of at least 48mA.
- VME DTACK*
  The VME DTACK* signal needs to be driven by a high current three-state driver with a-low state sink current of at least 64mA.[8]
- VME AS
  The VME AS* signal needs to be driven by a high current three-state driver with a low-state sink current of at least 64mA.
- VME DS0 and DS1
  The VME DS0* and DS1 signals need to be driven by a standard three-state driver with a low-state sink current of at least 64mA.
- VME BR[3:0]*
  The VME BR[3:0]* signals need to be driven by a open collector driver with a low state-sink current of at least 48mA.
- VME IRQ[7:1]*
  The VME IRQ[7:1]* signals need to be driven by a open collector driver with a low state-sink current of at least 48mA.

In order to achieve this high drive currents, FPGA/ASIC external driver chips need to be used.

8 This assumes that rescinding dtack is used.

3.2 Connections to external transceivers

Following figures show how the FPGA/ASIC internal I/O buffers are connected to the core and to external VME bus drivers.

Address Bus Driver

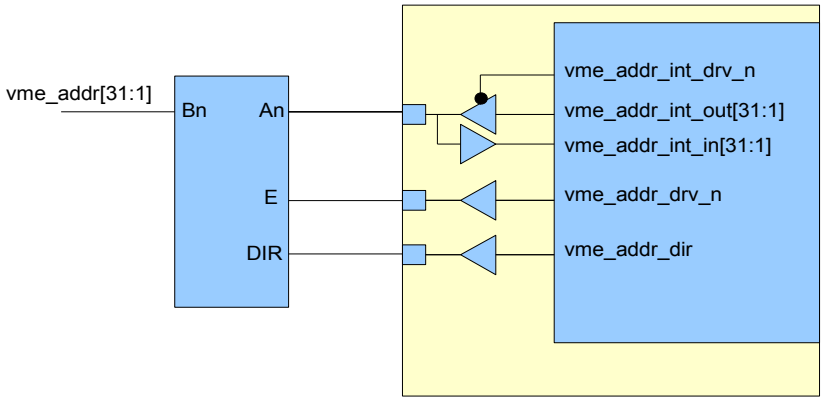

Figure 13: VME address bus transceiver

Data Bus Driver

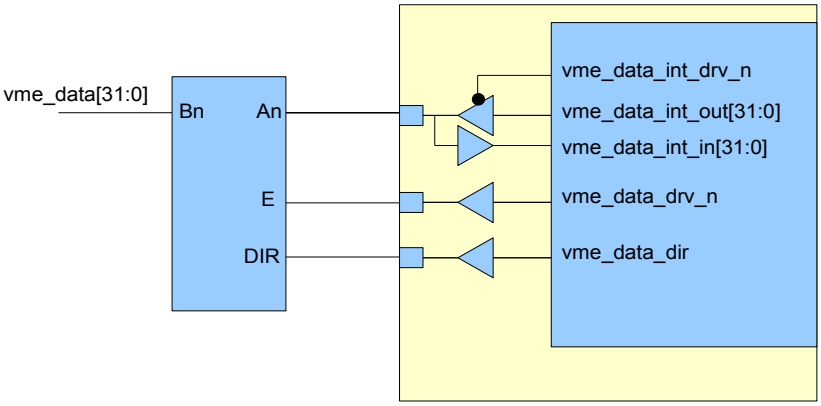

Figure 14: VME data bus transceiver

4 References

- The VMEbus Specification, ANSI/IEEE STD1014-1987
- American National Standard for VME64, ANSI/VITA 1-1994

Inicore is a leading Intellectual Property (IP) core and design solution provider. Our mission is to supply pre-verified, technology neutral, and reusable IP cores for a wide range of target markets from consumer goods to avionics and aerospace.

Our IP cores are complemented by comprehensive design service offerings:

- FPGA and ASIC Turn-Key Solutions
- Embedded System Design
- IP Core Design and Integration
- Consulting Services
- ASIC to FPGA Migration Service
- Obsolete Part Replacement

We can quickly provide you with an FPGA-, SoC- or Embedded System solution, leveraging our IP know-how and broad application-specific expertise. Our experience in microelectronic system integration allows us to guide you through the entire design flow from concept to final products. We help you with feasibility studies, concept analysis, system specification, design implementation and verification. Additionally, we do custom IP and low-level software development. We also handle everything from board design through fabrication and assembly.

Our development process is based on Structured Analysis & Structured Design (SA/SD) methodology that we apply to FPGA as well as ASIC projects. Verification testbenches rely on Transaction Based Verification (TBV) methods. Both these methodologies lead to reusable design and verification components. By planning for reusability, we set a solid base for further developments in the ever-decreasing product design - and life cycle.

Customer Advantages

We offer one-stop shopping for everything from the specifications to the chip or module implementation. It is our aim to engage with your engineering team and complement them in order to create your FPGA based system-on-chip solutions. This assistance, added to the ability to reuse our pre-designed and pre-verified IP cores, dramatically reduces design risks and execution time, and helps to successfully bring your product to the market.

Visit us @ www.inicore.com