INICORE **Datasheet**

VME64M

VME64 Master Controller

This core is obsolete and not recommended for new designs. Please use the VME System Controller core VMESCmodule instead.

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

1 OVERVIEW....4

1.1 Features....4

1.2 Deliverables....5

1.3 Block Diagram....5

2 IO DESCRIPTIONS....6

2.1 Signal Description....6

2.1.1 General Inputs....6

2.1.2 VME Bus Interface....6

2.2 User-Side Interface....9

2.2.1 Local Bus Interface....9

2.2.2 Slave Access Decoder....12

2.2.3 Interrupter....14

2.2.4 Bus Requester....16

2.2.5 VME Master....18

2.3 Synthesis Option....22

2.3.1 Rescinding DTACK....22

3 APPENDIX....24

3.1 Selecting proper I/O drivers....24

3.2 Connections to external transceivers....25

4 REFERENCES....26

Table of Figures

Figure 1: Block Diagram VME64 Master core.....5
Figure 2: User write cycle with different wait-states.....11
Figure 3: User read cycle with different wait-states.....11
Figure 4: User-access decoder operation.....13
Figure 5: ROAK interrupting scheme.....16
Figure 6: RORA interrupting scheme.....16
Figure 7: Regular DT transfer timing.....21
Figure 8: DT transfer timing with error response.....21
Figure 9: Open-collector DTACK*.....23
Figure 10: Rescinding DTACK*.....23
Figure 11: VME address bus transceiver 25
Figure 12: VME data bus transceiver.....25

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.

Revision History

Version	*Comment*
1.1	• Initial public release

1 Overview

The VME64M core is a VME compliant master controller for use in FPGA and ASIC based implementations. It is designed to provide master capabilities to VME cards where increased data throughput or added features are required, that can't be provided by the VME64S slave core.

The VME master implementation is fully compliant to the VME specification supporting A16/24/32 addressing modes and D8(EO), D16, D32, D32-BLT, and D64-MBLT data transfer modes. The core contains an interrupter as well as a bus requester.

With a minimal system clock of 40 MHz, the VME bus timing is guaranteed. A fully synchronous user side interface simplifies system integration by hiding any issues interfacing to the asynchronous VME bus. The user logic can delay bus cycles by introducing user wait-states if a peripheral such as a FIFO or a shared memory needs more time to finish the requested operation.

The VME64M core is not a replacement for a VME slot 0 controller, but it is intended to be used by advanced peripheral cards.

1.1 Features

- Master Interface
  - Addressing modes: A16, A24, A32
  - Data modes: D8(EO), D16, D32, D32-BLT, D64-MBLT
  - Supports data read-ahead and posted write to increase throughout
  - Constant local bus address for DMA transfers to/from FIFOs
- Slave Interface
  - Addressing modes: A16, A24, A32
  - Data types: D8(EO), D16, D32, D32-BLT, D64-MBLT
  - Access modes: Read, write, read-modify-write
  - Selectable rescinding DTACK
  - Provides big-endian to little-endian conversion option
- Interrupter
  - D8(O), D16, D32
  - Supports RORA and ROAK interrupt schemes
- Bus Requester
  - Supports RWD (release when done) and ROR (release on request) arbitration schemes
  - FAIR requester

    - Supports early withdrawal of bus request
- Local Bus Interface
    - Fully synchronous bus interface for user logic
    - User selectable wait-states
    - Optional big-endian to little-endian conversion

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 VME64M core complemented with some typical user logic modules:

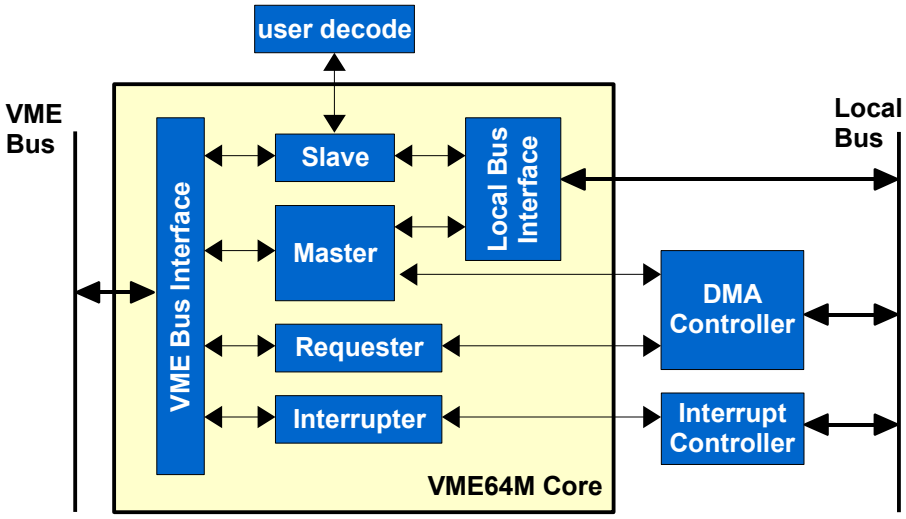


Figure 1: Block Diagram VME64 Master core

2 IO Descriptions

2.1 Signal Description

The following paragraphs list the inputs and outputs of the VME64M core and explains their functionality.

2.1.1 General Inputs

These pins are used to clock and initialize the whole VME core. To guarantee VME compliance, a few signals use the falling edge of the system clock. vme_as_in_n is used as a clock to sample the address bus. All other registers use the rising edge of the system clock clk_sys. For proper operation of the VME interface, the core requires a clock frequency of at least 40 MHz. A higher frequency is recommended to increase the data throughput.

Pin Name	*Type*	*Description*
clk_sys	in	System clock, minimal 40 MHz
reset_n	in	Asynchronous system reset, active low

2.1.2 VME Bus Interface

The VME bus interface contains all supporting signals necessary to control external VME transceivers.

Pin Name	*Type*	*Description*
vme_addr_int_in[31:1]	in	VME address bus input
vme_addr_int_out[31:1]	out	VME address bus output
vme_lword_int_in_n	in	VME long word access indicator input, active low
vme_lword_int_out_n	out	VME long word access indicator output, active low
vme_addr_drv_n		Active low drive enable signal for external vme_addr and vme_lword_n drivers 1: Output is tri-stated 0: Output is active
vme_addr_dir	out	Direction control signal for external vme_addr and vme_lword_n drivers 1: to VME bus 0: from VME bus

Pin Name	Type	Description
vme_addr_int_drv_n	in	Active low drive enable signal for internal vme_addr and vme_lword_n drivers 1: Output is tri-stated 0: Output is active
vme_data_int_in[31:0]	in	VME data bus input
vme_data_int_out [31:0]	out	VME data bus output
vme_data_drv_n	out	Active low drive enable signal for external vme_data drivers 1: Output is tri-stated 0: Output is active
vme_data_dir	out	Direction control signal for external vme_data drivers 1: to VME bus 0: from VME bus
vme_data_int_drv_n	out	Active low drive enable signal for internal vme_data drivers 1: Output is tri-stated 0: Output is active
vme_am_int_in[5:0]	in	VME address modifier input
vme_am_int_out[5:0]	out	VME address modifier output
vme_write_int_in_n	in	Read/write signal input, active low
vme_write_int_out_n	out	Read/write signal output, active low
vme_am_dir	out	Direction control signal for external vme_am and vme_write_n drivers 1: to VME bus 0: from VME bus
vme_am_int_drv_n	out	Active low drive enable signal for internal vme_am and vme_write_n drivers 1: Output is tri-stated 0: Output is active
vme_dtack_int_in_n	in	Data transfer acknowledge input Used to indicate whether the DTACK is drive low or high (for rescinding)
vme_dtack_int_out_n	out	Data transfer acknowledge output

Pin Name	Type	Description
vme_dtack_dir	out	Direction control signal for external vme_dtack driver 1: to VME bus 0: from VME bus
vme_dtack_drv_n	out	Active low drive enable signal for external vme_dtack driver 1: Output is tri-stated 0: Output is active
vme_dtack_int_drv_n	out	Active low drive enable signal for internal vme_dtack driver 1: Output is tri-stated 0: Output is active
vme_as_int_in_n	in	VME address strobe input: clocks with falling edge the internal synchronization signals like vme_addr_in and vme_am_in. vme_as_in_n is also used as data signal for access start detection.
vme_as_int_out_n	out	VME address strobe output
vme_as_dir	out	Direction control signal for external vme_as driver 1: to VME bus 0: from VME bus
vme_as_int_drv_n	out	Active low drive enable signal for internal vme_as driver 1: Output is tri-stated 0: Output is active
vme_ds0_int_in_n	in	Data strobe 0 input, active low
vme_ds0_int_out_n	out	Data strobe 0 output, active low
vme_ds1_int_in_n	in	Data strobe 1 input, active low
vme_ds1_int_out_n	out	Data strobe 1 output, active low
vme_ds_dir	out	Direction control signal for external vme_ds0 and vme_ds1 drivers 1: to VME bus 0: from VME bus

Pin Name	Type	Description
vme_ds_int_drv_n	out	Active low drive enable signal for internal vme_ds0 and vme_ds1 drivers 1: Output is tri-stated 0: Output is active
vme_br_out_n[3:0]	out	VME bus request out, active low
vme_br_in_n[3:0]	in	VME bus request in (feedback)
vme_bg_in_n[3:0]	in	VME bus grant daisy chain in, active low
vme_bg_out_n[3:0]	out	VME bus grant daisy chain out, active low
vme_bbsy_n	in	VME bus busy
vme_bclr_n	in	VME bus clear
vme_retry_n	in	VME retry indicator
vme_berr_n	in	VME bus error indicator
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.

2.2 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.

2.2.1 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.

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.

Pin Name	Type	Description
user_acc_req	out	Data access request Active high until user_acc_ack acknowledges the request (or VME bus error occurs).
user_acc_ack	in	User-side acknowledgment signal User side access is finished by asserting user_acc_ack for one clock cycle.
user_addr[31:2]	out	Registered VME address bus
user_am[5:0]	out	Registered VME address bus modifier
user_data_out[31:0]	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[31:0]	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_ack is asserted.
user_rwn	out	Data read/write_not indicator 0: Write 1: Read
user_byte_valid[3:0]	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
user_slv_mstrn	out	User Slave – Master Not access This signal indicates where the access is coming from. 0: Master access 1: Slave access

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.

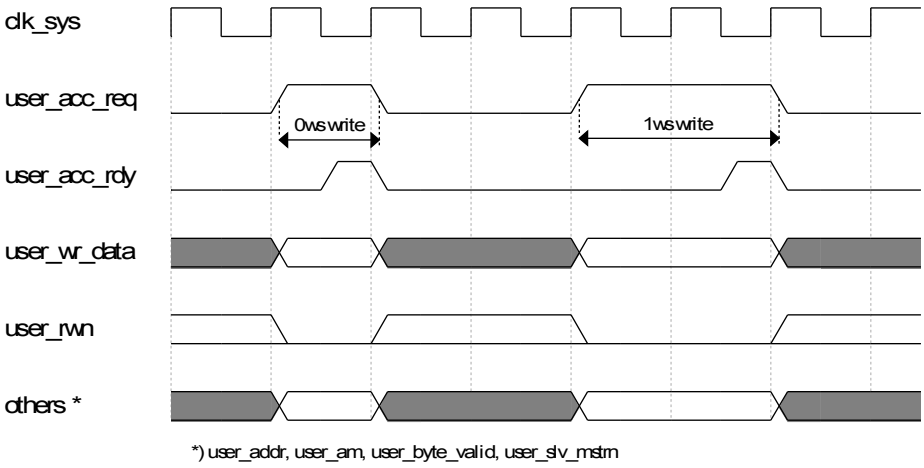


Figure 2: User write cycle with different wait-states

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.

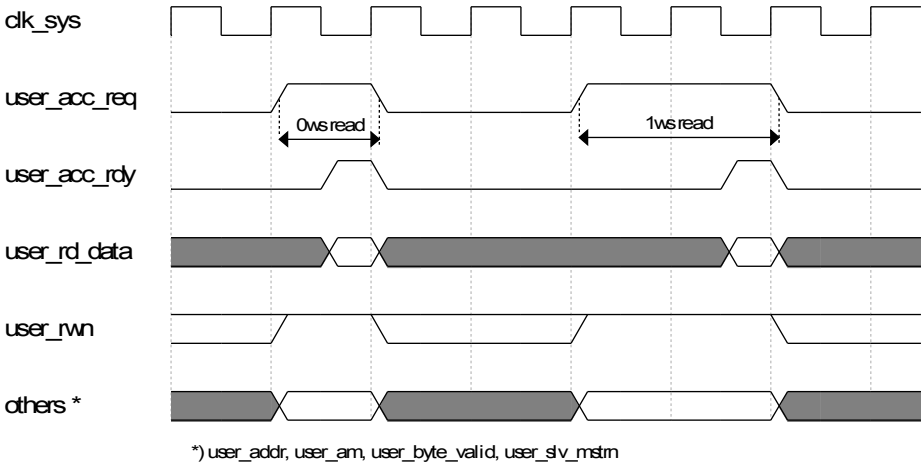


Figure 3: User read cycle with different wait-states

2.2.2 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[31:1]	out	Registered VME address bus (by falling edge of vme_as_n)
int_user_am[5:0]	out	Registered 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 successive 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.

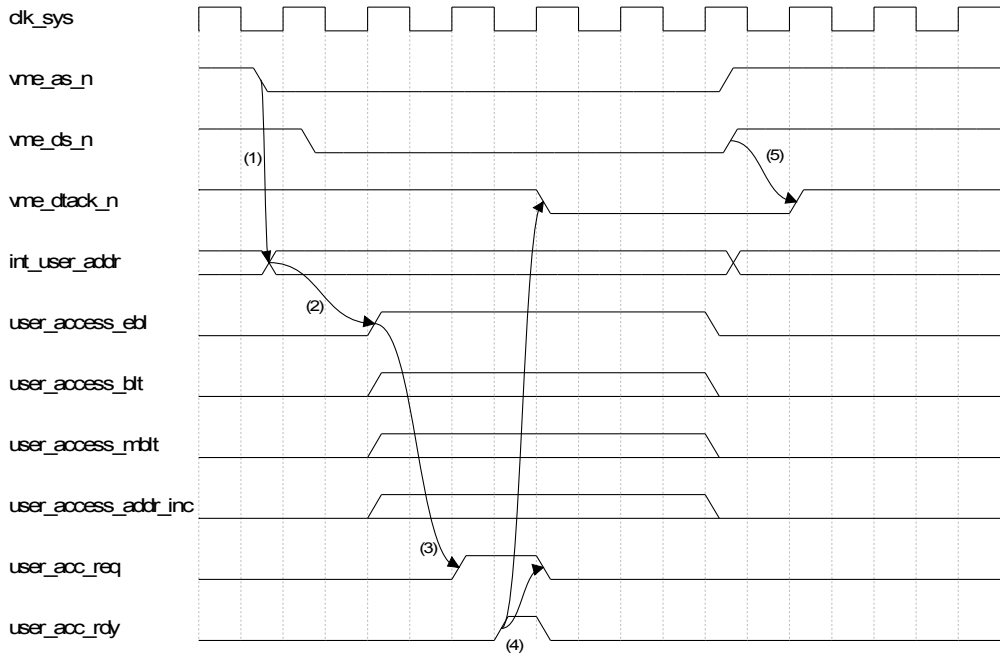


Figure 4: User-access decoder operation

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

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

int_vme_am	int_vme_addr[1]	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

2.2.3 Interrupter

The Interrupter block handles the generation of VME interrupt requests and acknowledgments of local interrupts. During an interrupt acknowledge cycle, the Interrupter provides 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.

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

Pin Name	Type	Description
user_iack	out	Interrupt acknowledgment An active one event indicates the end of a valid interrupt acknowledge cycle.
user_ilev[2:0]	in	Interrupt level
user_ivec[7/15/31:0]	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_ilev (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_ilev and user_ivec have to be stable for the whole time period where user_ireq is high.

Timing using ROAK scheme:

The user side logic releases user_ireq upon detection of user_iack.

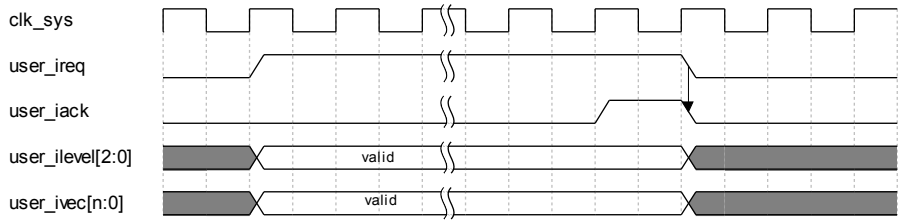


Figure 5: ROAK interrupting scheme

Timing using RORA scheme:

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

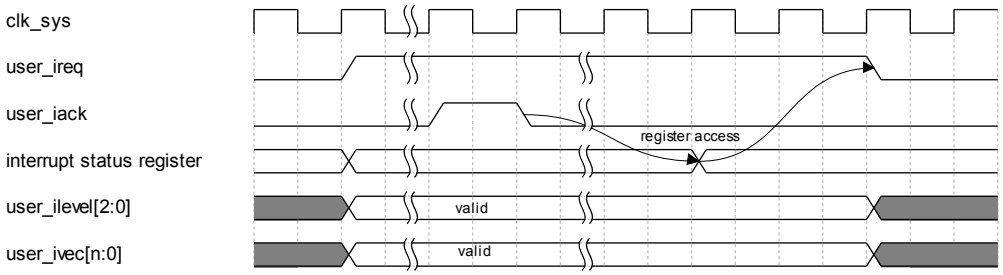


Figure 6: RORA interrupting scheme

2.2.4 Bus Requester

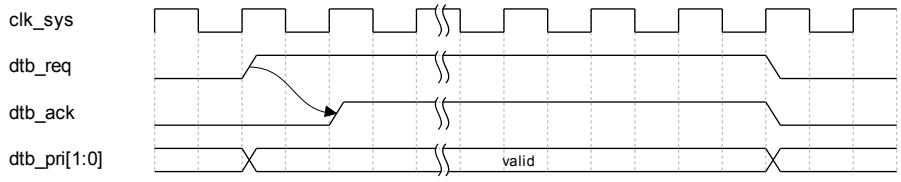
The bus requester block is used to obtain VME bus ownership. Handshake signals with the user side logic provide a flexible interface that supports all common requester scheme such as release-when-done, release-when-requested, and FAIR request.

Pin Name	*Type*	*Description*
dtb_req	in	Data Transfer Bus Request When asserted, node requests bus ownership
dtb_ack	out	Data Transfer Bus Acknowledgment When asserted, node is granted bus ownership

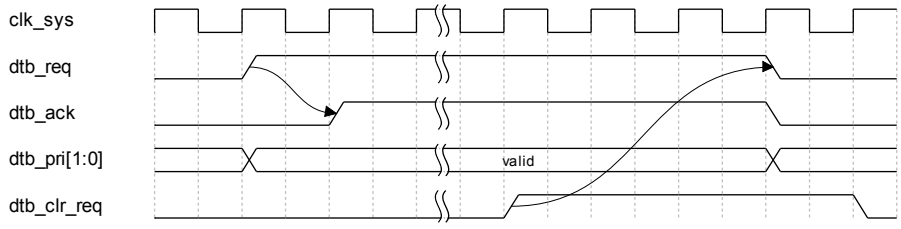
Pin Name	Type	Description
dtb_pri[1:0]	in	Data Transfer Bus Priority Selects the desired bus priority
dtb_clr_req	out	Data Transfer Bus Clear Request If asserted, arbiter indicates to node that a higher priority request is pending
dtb_config_fair	in	The bus requester observes the FAIR request type. A bus request is only asserted if no other bus request is pending with the same priority. 1: Use FAIR requesting scheme 0: Use direct request

Handshake Timing

Using the dtb_req and dtb_ack signals, the user side logic can request VME bus ownership. The bus ownership is granted as long as dtb_req is asserted.



The VME bus arbiter informs any master currently controlling the bus, that a higher level request is pending. On the user side, this is indicated by dtb_clr_req being asserted. This is shown in following diagram where the user site releases its request upon detection of dtb_clr_req being asserted.



Bus Arbitration

The bus arbiter block supports all common bus arbitration schemes:

- RWD: Release when done
  Once the requested data transfer is complete, the user side logic releases dtb_req.
- ROR: Release when requested
  Once the data transfer is completed, the user side logic continuous to keep dtb_req asserted. dtb_req is only released upon detection of dtb_clr_req being asserted. It is up to the user side logic to decide how long it keeps dtb_req asserted.
- FAIR: Fair requester
  In order for the Requester to act as a FAIR requester, config_dtb_fair needs to be asserted. In this configuration, the VME bus is only requested when no other bus requests with the same priority are pending.

2.2.5 VME Master

The VME Master operation is controlled with a command interface. Handshake signals are used to start and stop a data transfer command.

The data itself is transmitted between the VME bus and the local data bus.

Pin Name	*Type*	*Description*
dt_req	in	Data Transfer Request When assert, a VME data transfer is requested. The request has to be removed as soon as the master completes the transaction which is indicated by dt_ack being asserted.
dt_ack	out	Data Transfer Acknowledgment When asserted, requested data transfer is completed.
dt_nack	out	Data Transfer Not-Acknowledged If asserted, an error or an abort situation was detected and the current transfer stopped.

Pin Name	Type	Description
dt_status_error[1:0]	out	Data Transfer Error Status If an error or an abort situation was detected, this signal indicates the cause. 0: VME bus error (BERR) 1: VME retry (RETRY) 2: Reserved 3: Reserved
dt_status_beat_count[9:0]	out	Data Transfer Status Beat Count This counter indicates which beat is being transmitted. Using this field, the user logic can determine where an error happened and resume data transmission from there.
dt_cmd_addr[31:0]	in	Data Transfer Address This is the address used for this transfer. VME address and local bus address are identical and there is no address translation performed. The user side signal user_slv_mstn is used to detect slave or master user side access.
dt_cmd_am[5:0]	in	Data Transfer Address Modifier This is the VME address modifier used for this transfer.
dt_cmd_rwn	in	Data Transfer Read/Write 0: VME Master Write operation 1: VME Master Read operation
dt_cmd_size[9:0]	in	Data Size This defines the size of a data transfer in beats. 0: 1 beat of data 1: 2 beats of data ... 1023: 1024 beats of data

Pin Name	Type	Description
dt_cmd_width[2:0]	in	Data Width Defines the data width 0: DO8(EO) 1: D16 2: Reserved 3: D32 4: D32-BLT 5: Reserved 6: D64-MBLT
dt_cmd_lbai_ebl	in	Local Bus Address Increment Enable Usually, the local bus address matches the VME bus address. If a FIFO is connected to the local bus, the address should not be incremented. 0: Local bus address is fixed 1: Local bus address increments with each consecutive access
dt_cmd_lbpw_ebl	in	Local Bus Posted Write Enable To accelerate data transfer from the VME bus to the local bus, a VME transaction can be terminated before the data write on the local bus side was completed. 1: Enabled 0: Not enabled
dt_cmd_lbra_ebl	in	Local Bus Read Ahead Enable To accelerate data transfer from the local bus to the VME bus, local data can be pre-fetched 1: Enabled 0: Disabled

Handshake Timing

Using the dt_req and dt_ack signals, a local DMA controller or state-machine can create VME bus cycles. While dt_req is asserted, the dt_cmd_* control signals may not change.

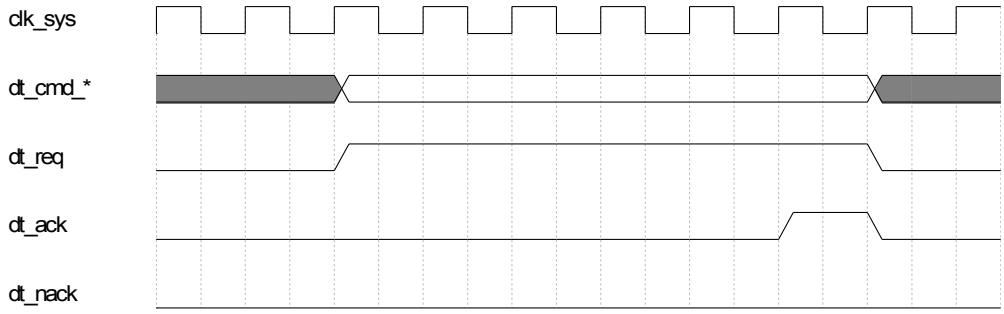


Figure 7: Regular DT transfer timing

If an error is detected during a data transfer, the current transfer is stopped. This is indicated by a single clock-cycle pulse on dt_nack. At the same time, dt_status_error provides the reason for the transfer abort.

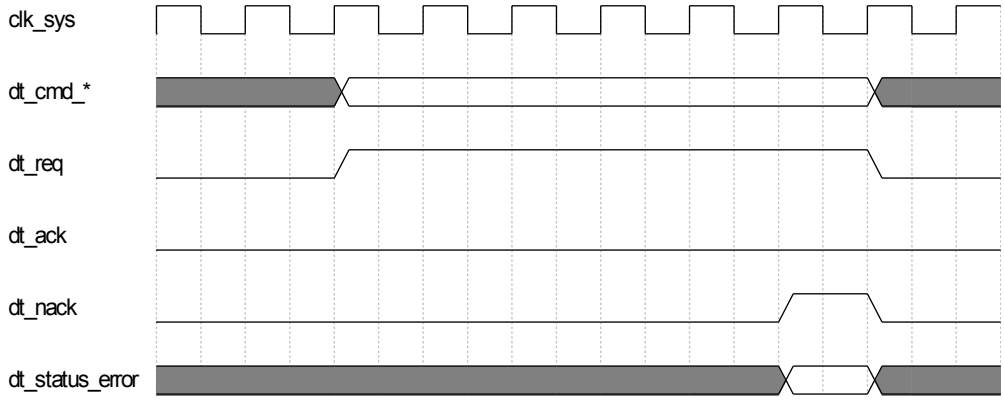


Figure 8: DT transfer timing with error response

2.3 Synthesis Option

Prior to design synthesis, the behavior of the core can be selected according to the application requirements.

Synthesis Option	*Description*
interrupter	Interrupter selection 8: D8(O) type interrupter 16: D16 type interrupter 32: D32 type interrupter
endian	Endian selection for user-side interface 0: Big endian, transparent operation 1: Little endian, core performs byte swapping
slave_config_dtack	Rescinding DTACK enable The VME slave block can use rescinding dtack to accelerate data transmission. ‘0’: Disabled ‘1’: Enabled

2.3.1 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:

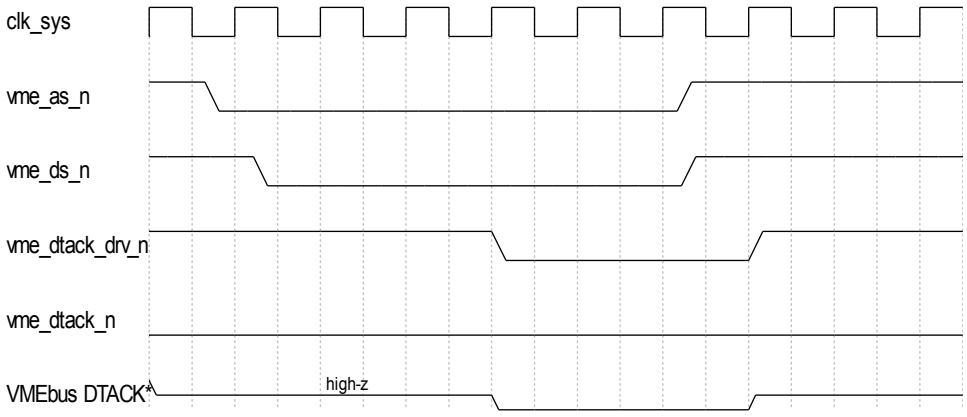


*Figure 9: Open-collector DTACK**

Timing diagram with rescinding DTACK:

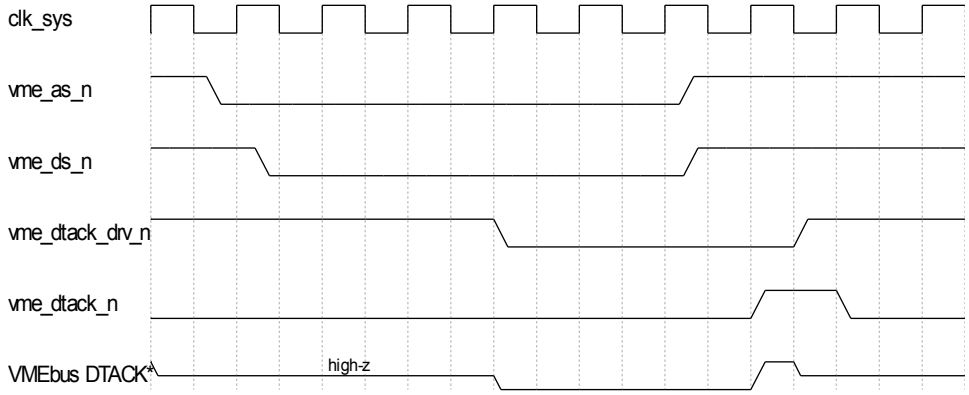


*Figure 10: Rescinding DTACK**

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.[2]
- 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.

2 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

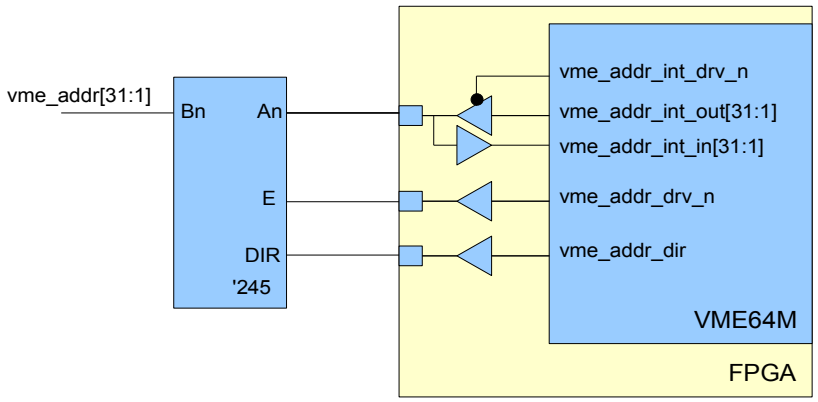


Figure 11: VME address bus transceiver

Data Bus Driver

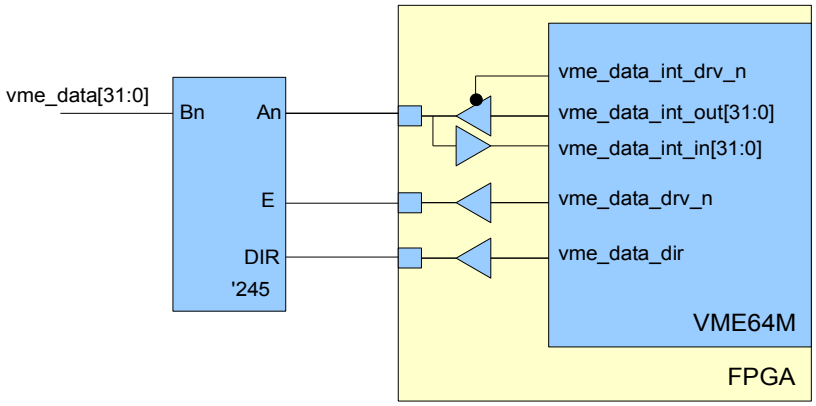


Figure 12: VME data bus transceiver

4 References

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

About Inicore

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