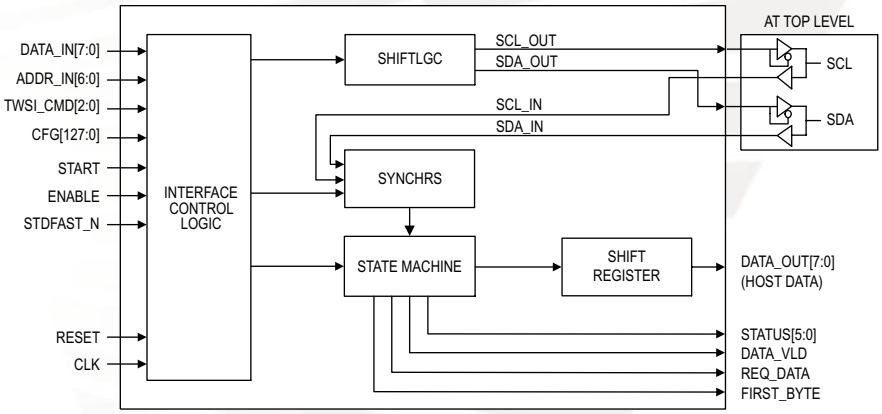
AvnetCore: Datasheet

Version 1.0, July 2006

Two Wire Serial Interface (TWSI)



Block Diagram

The MC-ACT-TWSI is a “core” logic module specifically designed for Actel FPGAs that emulates the functionality of the industry standard two-wire serial interface, I2C. This core does not support 10-bit slave addressing or START byte data transfers. It facilitates upgrading current systems by allowing the designer to incorporate the TWSI function as well as other logic into a single, state of the art FPGA. This core is designed such that it can be instantiated into a Actel design and “hooked up” to I/O buffers and pads and then compiled to make a device that will “plug in” to a TWSI application. The core can be configured to operate as either a Master-Slave, Master-Only, or Slave-Only.

Intended Use:

— Embedded microprocessor boards and any circuit needing I2C peripherals

Features:

— Fully optimized for Actel FPGAs
— I2C-compatible two-wire serial interface core; I2C is a trademark of Philips, Inc.
— Multi-master operation with arbitration and clock synchronization
— Slave transmit and receive operation
— Support for reads, writes, burst reads, burst writes, and repeated start
— User-defined timing and clock frequency
— Fast mode and standard mode operation

Targeted Devices:

— Axcelerator® Family
— ProASIC®3 Family
— ProASIC PLUS® Family

Core Deliverables:

— Netlist Version
  > Netlist compatible with the Actel Designer place and route tool
  > Compiled RTL simulation model, compliant with the Actel Libero® environment
— RTL Version
  > VHDL or Verilog RTL Source Code
  > Test Bench
— All
  > User Guide
  > Data Sheet

Synthesis and Simulation Support:

— Synthesis: Synplicity
— Simulation: Model*Sim*
— Other tools supported upon request

Verification:

— Test Bench

Actel CompanionCore

AVNET Memec

Functional Description

The diagram on the first page shows the Block Diagram of the MC-ACT-16550 Core. The core is partitioned into modules. These modules are described below.

INTERFACE CONTROL LOGIC

The core is configured via a configuration vector (CFG[127:0]). This vector contains information required for proper operation of the core. The command vector input (TWSI_CMD[2:0]) is used to instruct the core what operation to perform. The contents of the input and output vectors are as follows:

Configuration Vector

Applicable to the following core configurations:

Master-Only	Master-Slave	Slave-Only
Yes	Yes	Yes

The bits of the Configuration Vector are defined as follows:

Configuration Bits	Description
CFG[6:0]	TWSI Slave's Own Address
CFG[7]	Reserved
CFG[19:8]	SCL BusFree (arbitration) Timer Value for Standard Speed
CFG[31:20]	SCL HI Timer Value for Standard Speed
CFG[43:32]	SCL LO Timer Value for Standard Speed
CFG[45:44]	TWSI Mode (Master-Only, Slave-Only, Master-Slave)
CFG[46]	Enable the SCL Glitch Filters
CFG[47]	Enable two extra CLK periods of hold time on SDA from the falling edge of SCL.
CFG[48]	Enable General Call Addressing
CFG[60:49]	SCL BusFree (arbitration) Timer Value for Fast Speed
CFG[72:61]	SCL HI Timer Value for Fast Speed
CFG[84:73]	SCL LO Timer Value for Fast Speed
CFG[127:85]	Reserved

Table 1: Configuration Vector

Command Vector

Applicable to the following core configurations:

Master-Only	Master-Slave	Slave-Only
Yes	Yes	Undefined

The bits in this vector are defined as follows:

Figure 1: Command Vector

These three bits of the command vector control the operation performed on the serial interface. The READ bit determines if the operation will be a read (high) or a write (low). The STOP bit will determine if the cycle will stop at the end of the data cycle, or continue on to a burst. The REPEATED START bit will cause the generation of a repeated start protocol.

Address Vector
Applicable to the following core configurations:

Master-Only	Master-Slave	Slave-Only
Yes	Yes	Undefined

The bits in this vector are defined as follows:

6	5	4	3	2	1	0
ADDR_IN[6]	ADDR_IN[5]	ADDR_IN[4]	ADDR_IN[3]	ADDR_IN[2]	ADDR_IN[1]	ADDR_IN[0]

Figure 4: Address Vector

The address vector (ADDR_IN[6:0]) is written with the 7-bit I2C address that will be transmitted during the address phase of the I2C bus transaction.

Data Input Vector
Applicable to the following core configurations:

Master-Only	Master-Slave	Slave-Only
Yes	Yes	Yes

The bits in this vector are defined as follows:

7	6	5	4	3	2	1	0
DATA_IN[7]	DATA_IN[6]	DATA_IN[5]	DATA_IN[4]	DATA_IN[3]	DATA_IN[2]	DATA_IN[1]	DATA_IN[0]

Figure 3: Data Input Vector

The data input vector (DATA_IN[7:0]) is written with a byte that will be transmitted during the data phase of the I2C bus transaction.

Status Vector
Applicable to the following core configurations:

Master-Only	Master-Slave	Slave-Only
Yes	Yes	Yes

The bits in this vector are defined as follows:

Master Only

5	4	3	2	1	0
Not Implemented	Not Implemented	Not Implemented	Not Implemented	BUSLOSS	SLAVE TIMEOUT

Master Slave

5	4	3	2	1	0
HDWE_GC	GC_RST	GC_ADDR	GC_BUSY	BUSLOSS	SLAVE TIMEOUT

Slave Only

5	4	3	2	1	0
HDWE_GC	GC_RST	GC_ADDR	GC_BUSY	Not Implemented	Not Implemented

Figure 4: Status Vector

Bit 5 - HDWE_GC: Indicates the core has received the Hardware General Call instruction.
Bit 4 - GC_RST: Indicates the core has received General Call instruction to reset and change its address.
Bit 3 - GC_ADDR: Indicates the core has received General Call instruction to change its address.
Bit 2 - GC_BUSY: Indicates the core is busy with a General Call access.
Bit 1 - BUSLOSS: Indicates that this master lost the bus to another master before completion of the cycle.
Bit 0 - SLAVE_TIMEOUT: Indicates that the addressed slave did not respond with a nack when required.

Data Output Vector
Applicable to the following core configurations:

Master-Only	Master-Slave	Slave-Only
Yes	Yes	Yes

The bits in this vector are defined as follows:

7	6	5	4	3	2	1	0
DATA_OUT[7]	DATA_OUT[6]	DATA_OUT[5]	DATA_OUT[4]	DATA_OUT[3]	DATA_OUT[2]	DATA_OUT[1]	DATA_OUT[0]

Figure 5: Data Output Vector

The data output vector (DATA_OUT[7:0]) contains the data from the receive shift register

SHIFT REGISTER
There is a single parallel-in, parallel-out, serial-in, serial-out shift register called NUPSHIFT, which performs the shifting of data for address cycles, write cycles, and read cycles. The parallel output drives the core interface pins DATA_OUT, which are used to return read data to the host.

SYNCHRS
The SDA and SCL inputs are passed through the Synchrs module that performs a dual-rank synchronization and glitch filtering when enabled by bit 46 of the configuration vector (CFG[46]).

The MC-TWSI core treats both the SDA and SCL lines as data lines. The SDA line is actually sampled some number of clocks after the rising edge of SCL is detected. This allows for greater noise immunity and more robust operation.

STATE MACHINE
The control for the serial interface comes from the TWSI_SM_M, TWSI_SM_S, and GEN_CALL_SM state machines. These state machines control the loading and enabling of all shift registers and counters, and are responsible for implementing the basic interface protocol.

Arbitration
Before the MC-TWSI initiates either a start or repeated start condition, it samples the synchronized versions of SDA and SCL for BUSFREE_COUNT x clock periods of high values. If at any time either of these pins is detected in a low (driven) state, then the count starts over (it assumes that another master is on the bus). The automatic retrying to obtain the serial bus only occurs until the start command is issued. After that, the responsibility of the retry is left to software, since cycles may be of infinite length.

After a start is issued and while the address or data is being written to the slave, the macro samples the data on the SDA pin while SCL is high. If a data mismatch is detected then the operation is aborted with an interrupt and a BUSLOSS status. This can also occur during burst reads whenever a NACK is intended to be issued and an ACK is detected instead.

SHIFT LOGIC

The basic cycle on the TWSI serial interface consists of an address cycle followed by data cycle(s). The address consists of seven bits and the read/write bit (the LSB). The MSB is always transmitted first on the SDA line. The data cycle can either be a read or a write. For a write operation, the core shifts the data from the DATA_IN Register onto the SDA line. For a read operation, the core captures the data into the Shift Register. The host can read the Shift Register contents via the DATA_OUT Register. The data cycle can end in three different ways:

1. A stop can be generated which terminates the current cycle
2. Another data cycle can take place (a burst)
3. A repeated start can be generated by the interface

For each byte read, DATA_VLD will be asserted for one clock to tell the host that valid data is on DATA_OUT. For each byte written, REQ_DATA will be asserted for one clock to ask the host for another byte of data to send.

A repeated start is used to turn the bus around; when a read cycle must be followed directly by a write cycle without a stop in-between. Since the READ bit is a part of the address, if a read followed by write is desired without a stop command, a second address must be issued following the data cycle. The sequence of events in a repeated start cycle is: start, address cycle, data cycle, repeated start, address cycle, data cycle, stop. Each of the data cycles can be repeated if bursting is desired, and the stop cycle could actually be another repeated start, if desired.

GENERAL CALL ADDRESSING

The General Call Address is used to address and send data to every node on the I2C bus and is enabled by bit 48 of the configuration vector (CFG[48]). Not all devices are required to support General Call. Nodes not requiring the data sent within a General Call data transfer can ignore the transfer by not acknowledging the address. If a node does require the data sent with a General Call it will acknowledge the address and behave as a slave receiver for the subsequent bytes. If any given slave cannot process any of the subsequent bytes during the transfer, it must ignore the byte by not acknowledging. The behavior for a General Call transfer is specified in the second byte, and is determined by the LSB.

When the LSB of the second byte is a ‘0’, the second byte can have the following definition:

“00000110” (H’06’) – Write the slave address (the third byte to be sent) and then reset. All devices receiving this two-byte sequence will treat the third byte to be sent as their new I2C address and will then reset. Care must be taken by the slave nodes not to drive (pull down) the SDA or SCL lines while resetting to avoid blocking the bus.

“00000100” (H’04’) – Write the slave address (the third byte to be sent) and do not reset. All devices receiving this two-byte sequence will treat the third byte to be sent as their new I2C address. The node will not reset.

“00000000” (H’00’) – This byte is not allowed to be sent as the second byte.

All other bytes in the second byte of the transfer have not been specified and nodes are generally required (per the Philips I2C specification) to ignore them.

When the LSB of the second byte is a ‘0’, the second byte will be a ‘Hardware General Call’. This means that a master node on the I2C, which cannot be programmed to send a specific slave address, needs the attention of another node to facilitate a transfer. This is typical of embedded hardware nodes such as keyboard scanners. Since a hardware master can’t be programmed to address a specific node to which it needs to transfer data, it can only initiate a General Call followed by its own address to identify itself to the system. The seven upper bits of the second byte contain the address of the hardware master. An intelligent device then recognizes this address on the I2C bus, which will direct the information from the hardware master.

SIGNAL NAMES

Signal	Direction	Description
CLK	Input	The primary clock used for all circuits in this interface.
RESET	Input	Active-high asynchronous reset.
START	Input	Strobe to start TWSI bus cycle.
ENABLE	Input	WHEN ASSERTED THE CORE WILL BE ENABLED. THIS MUST BE DRIVEN FROM A REGISTER.
STDFAST_N	Input	Select TWSI transfer speed mode. Set to '1' for standard (100 kHz) or set to '0' for fast (400 kHz).
DATA_IN[7:0]	Input	The data input vector.
ADDR_IN[6:0]	Input	The address input vector.
TWSI_CMD[2:0]	Input	The command input vector. Tells the core what TWSI command to perform (read/write, stop, repeated start).
CFG[127:0]	Input	MC-TWSI configuration vector. Configures the options for the MC-TWSI core (SCL timer values, TWSI mode, etc.).
DATA_VLD	Output	Asserted for one clock cycle to indicate a data byte has been received on the TWSI bus and is available on DATA_OUT.
REQ_DATA	Output	Asserted for one clock cycle indicate a data byte is requested when sourcing data. Can be used to increment a data pointer.
FIRST_BYTE	Output	Indicates the byte just received is the first data byte (of what may be a burst). Can be used as an enable to latch the data byte (on DATA_OUT) as an offset in to a register or memory array.
STATUS[5:0]	Output	The STATUS bits are always available and are cleared when START is strobed.
DATA_OUT[7:0]	Output	The DATA_OUT port contains the data byte just received on the TWSI bus and is qualified with DATA_VLD at the completion of a TWSI data phase.
SDA_IN	Input	This is the serial data line open collector input.
SDA_OUT	Output	This is the serial data line open collector output.
SCL_IN	Input	This is the serial clock line open collector input.
SCL_OUT	Output	This is the serial clock line open collector output.
SDA	External	TWSI serial data line.
SCL	External	TWSI serial clock line.

Table 2: Core and External Signal Pinout

Application

BASIC INTERFACE OPERATION
All I/O to and from the TWSI core is synchronous to the system clock, CLK.

Before any operation of the MC-TWSI core, the ENABLE must first be asserted. This must be done via a register on the host interface.

For a TWSI master to initiate a transaction on the bus, the host sets up the ADDR_IN, DATA_IN and TWSI_CMD ports and strobes the START input for one clock and performs other operations until either REQ_DATA (in the case of a write) or DATA_VLD (in the case of a read) is asserted. Also, the SLAVE_TIMEOUT (STATUS[0]) and BUSLOSS (STATUS[1]) bits should be polled in case the slave does not respond or the master loses arbitration. The BUSLOSS status bit will be asserted if the interface lost the bus to another master before completion of the cycle. The SLAVE_TIMEOUT bit will be asserted if the slave does not respond with a NACK when required. For either of these error conditions, the host must retry the current cycle by strobing START. Note that for burst operations the host is responsible for writing the incremented address into the ADDRESS Register before retrying the operation.

HOST WRITE OPERATION
The following figure shows how the microprocessor interface connects to the internal control registers:

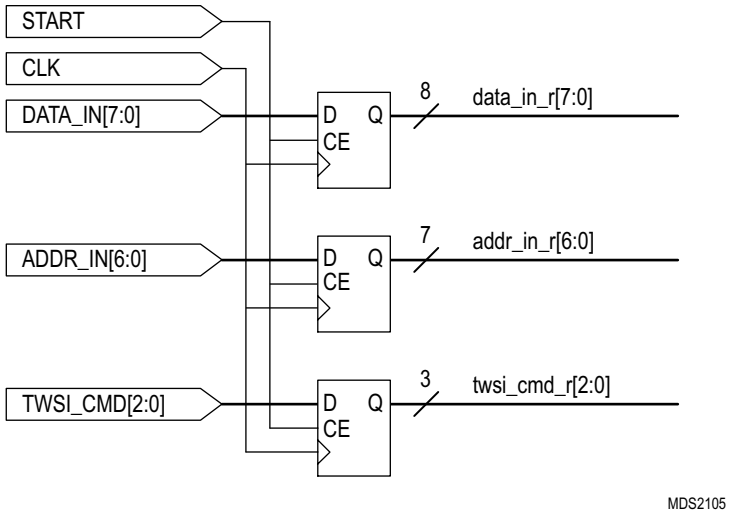


Figure 6: Host Write Register Interface

On the rising edge of CLK, the values on DATA_IN[7:0] and ADDR_IN[6:0] and TWSI_CMD[2:0] is clocked into their registers when START is strobed.

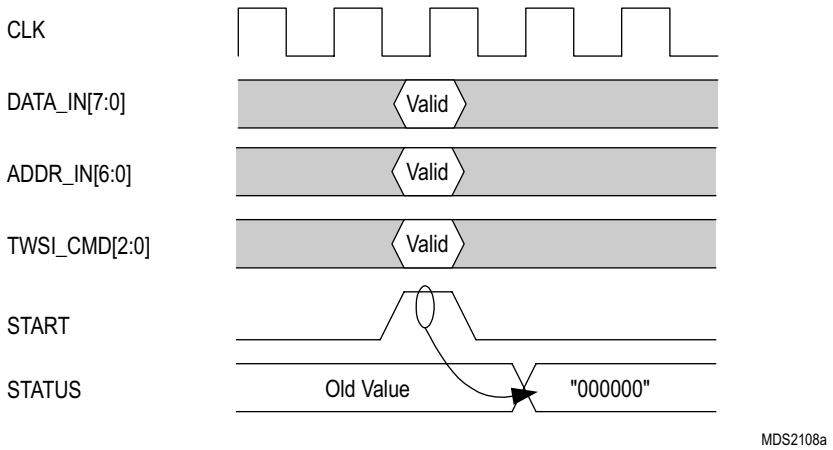


Figure 7: Input Register Timing

Notice in Figure 7 that START acts as the write enable for the registers. Also notice that when CLK samples START asserted, the STATUS bits are cleared on the next rising edge of CLK.

HOST READ OPERATION

The host read operation is best described with the diagram in Figure 8 below:

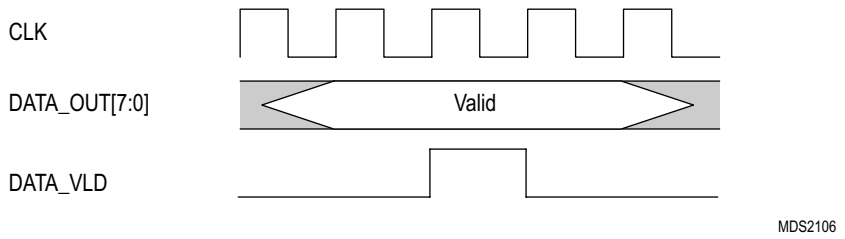


Figure 8: Host Read Logic

Notice in Figure 8 that DATA_VLD can act as the write enable for a register to capture the byte on DATA_OUT for use by the host interface.

PERFORMING TWSI TRANSACTIONS

Single Read

To generate a single read cycle, the address input (ADDR_IN) is written with the desired address and the command input (TWSI_CMD) is written with “011” (binary 011, REPEATED START=0, STOP=1, READ=1) and START is strobed for one clock cycle to perform the operation and stop. When the operation is completed the DATA_VLD pin will go active and the data may be read on the DATA_OUT port. The user should poll the STATUS pins and restart the operation if necessary if BUSLOSS is asserted or attempt another address if SLAVE_TIMEOUT is asserted.

Single Write

To generate a single write cycle:

- The address input (ADDR_IN) is written with the desired address
- The data input (DATA_IN) is written with the byte to be sent
- The command input (TWSI_CMD) is written with “010” (binary 010, REPEATED START=0, STOP=1, READ=0)
- START is strobed for one clock cycle to perform the operation and stop.

When the operation is completed the REQ_DATA pin will go active to ask the host for another byte of data to send. Since this is a single write, REQ_DATA can be ignored. The user should poll the STATUS pins and restart the operation if necessary if BUSLOSS is asserted or attempt another address if SLAVE_TIMEOUT is asserted.

Burst Read

A burst read operation is performed in the same manner as a single read, except that the command input (TWSI_CMD) is written with the STOP bit de-asserted (binary 001). At the end of each byte read the DATA_VLD pin will be asserted. This tells the host that valid data is available on the DATA_OUT port and that START needs to be strobed for the burst to continue. No new command should be written to the command input while a burst is in process. When the host is done reading data it will issue the Single Read command (binary 011) and the core will do one more read and stop. Note that there is only one address cycle for a burst read cycle. In addition, it is not permitted to follow a read operation or burst read operation with a write operation without first performing a repeated start or a stop. The user should poll the STATUS pins and restart the operation if necessary if BUSLOSS is asserted or attempt another address if SLAVE_TIMEOUT is asserted.

Burst Write

A burst write operation is performed in the same manner as a single write, except that the command input (TWSI_CMD) is written with the STOP bit de-asserted (binary 001). At the end of each byte written the REQ_DATA pin will be asserted. This tells the host that data is needed on the DATA_IN port and that START needs to be strobed for the burst to continue. No new command should be written to the command input while a burst is in process. When the host is done writing data it will issue the Single Write command (binary 010) and the core will do one more write and stop. Note that there is only one address cycle for a burst read cycle. In addition, it is not permitted to follow a write operation or burst write operation with a read operation without first performing a repeated start or a stop. The user should poll the STATUS pins and restart the operation if necessary if BUSLOSS is asserted or attempt another address if SLAVE_TIMEOUT is asserted.

Repeated Start

The repeated start operation is used to directly follow a read with a write, or a write with a read, without a stop in-between. Some devices require this for the generation of a second address; the address within the device as opposed to on the serial bus itself.

Any read or write operation may be followed by a repeated start by resetting the STOP bit and setting the REPEATED START bit when writing to the command input. For example, to perform a write cycle followed by a repeated start and a read, the command input is written with “100” to perform the write operation. The REQ_DATA pin will be asserted, as always, at the end of the write. If a single read is then desired the command input is written with “011”.

Fast Mode

For Fast mode of operation, it may be beneficial to filter out glitches on the SDA and SCL inputs. The filter enable bit in the configuration vector (CFG[46]), when high, will filter out any transition on either SDA or SCL which is less than one clock period wide.

If using Fast mode of operation with very short HI_COUNT and LO_COUNT dividers, it may also be useful to set the “extrahold” bit in the configuration vector (CFG[47]) high to provide two extra CLK periods of hold time on SDA from the falling edge of SCL.

Device Requirements

Family	Device	Utilization			Performance
		COMB	SEQ	Tiles	
Axcelerator	AX125	20%	21%	n/a	76 MHz
ProASIC^{PLUS}	APA075	n/a	n/a	21%	59 MHz
ProASIC3	A3P250	n/a	n/a	10%	67 MHz

Table 3: Device Utilization and Performance

Recommended Design Experience

For the source version, users should be familiar with HDL entry and Actel design flows. Users should be familiar with Actel Libero Integrated Design Environment (IDE) and preferably with Synplify and Model*Sim*. Users should also have experience with microprocessor systems and asynchronous communication controllers.

Ordering Information

The CORE is provided under license from Avnet Memec for use in Actel programmable logic devices. Please contact Avnet Memec for pricing and more information.

Contact Information:

North America
10805 Rancho Bernardo Road
Suite 100
San Diego, California 92127
United States of America
TEL: +1 858 385 7500
FAX: +1 858 385 7770

Europe, Middle East & Africa
Mattenstrasse 6a
CH-2555 Brügg BE
Switzerland
TEL: +41 0 32 374 32 00
FAX: +41 0 32 374 32 01

Ordering Information:

Part Number	Hardware	Resale
MC-ACT-16550-NET	Actel 16550 Netlist	Contact for pricing
MC-ACT-16550-VLOG	Actel 16550 Verilog	Contact for pricing

AVNET Memec

www.em.avnet.com/actel