This user guide provides basic information about using the Microtronix Avalon I²C IP. The following table shows the document revision history.

<table>
<thead>
<tr>
<th>Date</th>
<th>Rev</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>March 2006</td>
<td>1.0</td>
<td>Initial release</td>
</tr>
<tr>
<td>July 2008</td>
<td>1.1</td>
<td>Added slave address match &amp; minor edits</td>
</tr>
<tr>
<td>December 2010</td>
<td>1.2</td>
<td>Updated for component version 3.0</td>
</tr>
<tr>
<td>October 2013</td>
<td>1.3</td>
<td>Added Qsys support</td>
</tr>
</tbody>
</table>

**How to Contact Microtronix**

**E-mail**
Sales Information: sales@microtronix.com
Support Information: support@microtronix.com

**Website**
General Website: http://www.microtronix.com

**Phone Numbers**
General: 519-690-0091
Fax: 519-690-0092

**Typographic Conventions**

<table>
<thead>
<tr>
<th>Visual Clue</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path/Filename</td>
<td>A path or filename</td>
</tr>
<tr>
<td>[SOPC Builder]$ &lt;cmd&gt;</td>
<td>A command that should be run from within the Cygwin Environment.</td>
</tr>
<tr>
<td>Code</td>
<td>Sample code.</td>
</tr>
<tr>
<td>^</td>
<td>Indicates that there is no break between the current line and the next line.</td>
</tr>
</tbody>
</table>
## Table of Contents

How to Contact Microtronix ................................................................. 2  
   E-mail ........................................................................................................ 2  
   Website .................................................................................................... 2  
   Phone Numbers ...................................................................................... 2  
Typographic Conventions ......................................................................... 2  
Introduction ............................................................................................. 4  
Instantiating in Qsys and Quartus II ....................................................... 5  
Register Map .......................................................................................... 6  
   Control Register ..................................................................................... 7  
   Status Register ...................................................................................... 8  
   Address Register .................................................................................. 8  
   Transmit Buffer .................................................................................... 9  
   Receive Buffer ..................................................................................... 9  
Software .................................................................................................. 9  
Simulation .............................................................................................. 10
Introduction

The I2C-bus is a simple two wire bi-directional interface developed for inter-IC communication. Many semiconductor vendors offer a wide range of I2C-devices, like EEPROM memories, I/O-ports, temperature sensors, analog / digital converters, etc.

The I2C-core is an Avalon slave peripheral that allows Avalon master peripherals, like the Altera Nios II processor, to interface with I2C-peripherals. It has the following features:

- (Single) I2C-Master Transmit / Receive Mode.
- I2C-Slave Transmit / Receive Mode.
- 8-bit data transfers.
- 7-bit addressing format.
- Own address and general call address detection.
- Single byte transmit and receive buffer.
- Input filter.
- Three transmission speeds
  - Standard Mode (100 KHz)
  - Fast Mode (400 KHz)
  - High-Speed Mode (1 MHz)
- Meets the Philips I2C-bus specification version 2.1

The I2C core is Qsys ready and integrates easily into any Qsys generated system.
Instantiating in Qsys and Quartus II

Once the I²C component has been installed, it will appear in the Qsys Component Library under Interface Protocols->I2C->I2C Master/Slave.

Double-click on the component name to add it to your system. A window will pop up with a block diagram of the component. Click ‘Finish’ to finalize its addition to the system.
Once all components have been added to the system, click ‘Generate’ to create the Qsys system.

The I2C Master/Slave generates two signals at the top-level of the Qsys block: I2C_SCL and I2C_SDA. Each of these signals MUST be connected to a bidirectional FPGA pin. Each of these pins should have an external pull-up resistor as required by the I2C specification. If no external pull-up is available, it is possible to set the “Weak Pull-Up Resistor” assignment to “On” for both pins. Note that this assignment may not work in all cases and it is recommended to follow the I2C specification and use proper external resistors.

The I2C core has four 8-bit registers shown in Table 1. Note: component uses Avalon native alignment, registers are 32-bit aligned in a Nios II system.

### Table 1: Register Map

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>RW</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Control Register</td>
<td>RW</td>
</tr>
<tr>
<td>1</td>
<td>Status Register</td>
<td>RW</td>
</tr>
<tr>
<td>2</td>
<td>Address Register</td>
<td>RW</td>
</tr>
<tr>
<td>3</td>
<td>Transmit Buffer</td>
<td>W</td>
</tr>
<tr>
<td>3</td>
<td>Receive Buffer</td>
<td>R</td>
</tr>
</tbody>
</table>
**Control Register**

The control register controls the operation of the I\(^2\)C core. The control register can be read any time without changing the value of any bits. The control register bits are shown in Table 2.

**Table 2: Control Register**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>I2C ENABLE</td>
<td>Setting this bit enables the I(^2)C-interface.</td>
</tr>
</tbody>
</table>
| 1      | I2C MODE | The two bits sets the I\(^2\)C mode  
00 – Slave Mode  
01 – Master Standard Mode (100 KHz)  
10 – Master Fast Mode (400 KHz)  
11 – Master High-speed Mode (1 MHz) |
| 2      | I2C START | When is bit is set in master mode a (repeated) start condition is created. After the I\(^2\)C core has applied the start condition this bit is automatic cleared and the I\(^2\)C address is transmitted.  
In slave mode this bit has no influence. |
| 3      | I2C STOP | Setting this bit in master mode generates a stop condition. This bit is automatically cleared after the stop condition was applied. An interrupt is generated after the stop condition was successful executed.  
In slave mode this bit has no influence. |
| 4      | I2C RW   | This bit selects an I\(^2\)C read or write transfer.  
0 - Write  
1 - Read |
| 5      | I2C ACK  | This bit is transmitted after the I\(^2\)C data byte.  
0 – ACK  
1 – NACK |
| 6      | IRQ ENABLE | Setting this bit enables the Avalon interrupt |
**Status Register**

The status register shows the status of the I²C core. Reading the status register doesn’t change the value of the bits. Writing any value to the status register clears the interrupt bit.

**Table 3: Status Register**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>READY</td>
<td>In master mode this bit is set when no acknowledge was received after an I²C address or write data transfer. The I²C core automatically generates a stop condition when no acknowledge was detected. In slave mode this bit is always cleared.</td>
</tr>
<tr>
<td>1</td>
<td>I²C NACK</td>
<td>This bit indicates the status of the transmit buffer.</td>
</tr>
<tr>
<td>2</td>
<td>TX EMPTY</td>
<td>This bit indicates the status of the receive buffer.</td>
</tr>
<tr>
<td>3</td>
<td>RX FULL</td>
<td>This bit indicates the status of the receive buffer.</td>
</tr>
<tr>
<td>4</td>
<td>I²C WRITE TRANSFER</td>
<td>This bit is set when a write transfer is performed.</td>
</tr>
<tr>
<td>5</td>
<td>I²C READ TRANSFER</td>
<td>This bit is set when a read transfer is performed.</td>
</tr>
<tr>
<td>6</td>
<td>I²C SLAVE ADDRESS MATCH</td>
<td>This bit is used in slave mode to indicate when the address on the I²C bus matches the content of the address register. This bit is cleared by writing to the status register. In master mode this bit is always cleared.</td>
</tr>
<tr>
<td>7</td>
<td>IRQ</td>
<td>The interrupt bit is asserted when:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The transmit buffer is empty.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The receive buffer is full.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- A write access was not acknowledged by the I²C slave in master mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>After the stop condition was generated in master mode.</td>
</tr>
</tbody>
</table>

**Address Register**

The address register holds the seven-bit I²C address.

In master mode the I²C address is transmitted after the start bit in the control register was set.

In slave mode the address register is compared with the address byte sent by the I²C master. If the addresses are equal, an acknowledgement is automatically sent by the I²C core to the I²C master. The slave address match bit is also set.
Transmit Buffer

An Avalon master peripheral writes data to be transmitted into the transmit buffer. Writing to the transmit buffer automatically clears the transmit buffer empty bit and the interrupt.

In master mode the data in transmit buffer is directly sent after the address byte. If the transmit buffer is empty, the transmit buffer empty bit in the status register is set and the interrupt is asserted. The I²C core stalls the I²C transfer until new data is written in the transmit buffer or a repeated start or stop condition is selected.

In slave mode after the I²C slave address is received, the data in the transmit buffer is transferred to the I²C master. If no (new) transmit data is available, the transmit buffer empty bit is set and the interrupt is activated. The I²C core tells the I²C master to wait by forcing the I²C clock line low until new data has been written to the transmit buffer. After the data transfer the I²C -master sends an acknowledgement. An ACK indicates that the I²C -master wants to read more data from the I²C slave. A NACK tells the I²C slave that this was the last byte of the transfer. Now the I²C slave releases the I²C data line and the I²C master can generate start or stop condition.

Receive Buffer

Received data is loaded by the I²C core into the receive buffer and the receive buffer full bit is set. When the receive buffer is read by the Avalon master peripheral, the interrupt bit and receive buffer full bit are cleared.

When the I²C core is in master mode and after it receives a data byte from the I²C slave, the acknowledge bit, located in the control register, is send. When the receive buffer is read and no start or stop condition is selected, then the I²C core executes another read transfer to the same I²C slave.

In slave mode the interrupt is asserted when the I²C core receives new data from the I²C master and has loaded it in the receive buffer. The I²C core automatically acknowledges the transfer to the I²C master. If the receive buffer is full, indicated by the receive buffer full bit in the status register, the acknowledge bit is delayed by the I²C core. The I²C clock line is pulled low until the receive buffer was read by the Avalon master peripheral. Now the I²C core releases the I²C clock line and the I²C master continues with the read transfer.

Software

The file mtx_avalon_i2c_regs.h declares functions for accessing the I²C
component. These functions are listed in Table 4.

**Table 4: Software Routines**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IORD_MTX_AVALON_I2C_CONTROL(base)</td>
<td>Read Control Register</td>
</tr>
<tr>
<td>IOWR_MTX_AVALON_I2C_CONTROL(base, data)</td>
<td>Write Control Register</td>
</tr>
<tr>
<td>IORD_MTX_AVALON_I2C_STATUS(base)</td>
<td>Read Status Register</td>
</tr>
<tr>
<td>IOWR_MTX_AVALON_I2C_STATUS(base, data)</td>
<td>Write Status Register</td>
</tr>
<tr>
<td>IORD_MTX_AVALON_I2C_ADDR(base)</td>
<td>Read Address Register</td>
</tr>
<tr>
<td>IOWR_MTX_AVALON_I2C_ADDR(base, data)</td>
<td>Write Address Register</td>
</tr>
<tr>
<td>IOWR_MTX_AVALON_I2C_DATA(base, data)</td>
<td>Write Transmit Buffer</td>
</tr>
<tr>
<td>IORD_MTX_AVALON_I2C_DATA(base)</td>
<td>Read Transmit Buffer</td>
</tr>
</tbody>
</table>

**Base** – Avalon base address of the I\(^2\)C Master/Slave

**Data** – 8-bit data to write to the register

The included reference design contains example software that uses the above routines to control an I\(^2\)C device on the Microtronix I\(^2\)C Board. The example software can be found in the installation directory under examples/mtx_psk_2c35/i2c_a/software. The application (i2c_example) and BSP (i2c_example_bsp) can be imported into Nios II Software Build Tools for Eclipse with the File->Import… command. For more information on building software for Nios II, please refer to Altera’s documentation.

**Simulation**

A precompiled simulation library is provided for performing simulations using ModelSim. The library is located in the <install_dir>/simulation directory. Perform the following steps to simulate your design with the I\(^2\)C controller.

1. **Launch ModelSim**
2. **Map the I\(^2\)C library.** At the ModelSim prompt type;
   ```sh
vmap mtx_avalon_i2c <install_dir>/simulation/mtx_avalon_i2c
   ```
   If you use a newer version of ModelSim, you must refresh the precompiled library. At the Modelsim prompt type;
   ```sh
vcom -refresh -work mtx_avalon_i2c
   ```
3. **Compile the I\(^2\)C top level.** Eg. If the I\(^2\)C controller was named i2c in Qsys, then type;
   ```sh
vcom -93 i2c.vhd
   ```
4. **Compile all of the design files**
5. **Start the ModelSim simulation by typing;**
   ```sh
vsim -t ps -L mtx_avalon_i2c <top_level>
   ```