I2CM Controller

Summary

This document provides detailed reference information for the I2CM Controller peripheral device.

The I2CM (Master) Controller is used to facilitate data transfers over the I2C Bus and therefore ease communication with I2C devices external to the FPGA device. The I2C Bus is a two-wire, bi-directional serial bus that enables short distance data exchange between multiple devices connected to the bus.

Devices connected to the I2C bus can either be Master or Slave. The difference between the two lies primarily in the fact that a Master can take control of the serial clock and data lines (that constitute the I2C bus), changing their state in accordance with requests from the host microcontroller.

Important Notice: Supply of the I2C soft cores under the terms and conditions of the Altium End-User License Agreement does not convey nor imply any patent rights to the supplied technologies. Users are cautioned that a license from Royal Philips Electronics N.V. is required for any use covered by such patent rights, including the implementation of this core in an Integrated Circuit or any other device. For further information: *www.semiconductors.philips.com/buses/i2c/licensing/index.html*

Features

- I2C version 2.1 standard compliance
- Transmission speed modes:
  - Standard Mode (up to 100kb/s)
  - Fast Mode (up to 400kb/s)
  - High-speed Mode (up to 3.4Mb/s)
- Multimaster Mode (I2CM only)
- 7-bit slave addressing

Available devices

This device can be found in the FPGA Peripherals integrated library (`FPGA Peripherals.IntLib`), located in the `\Library\Fpga` folder of the installation. There is also a Wishbone version available (WB_I2CM), information on this device is available in the Altium Wiki.

I2CM

Functional Description

Symbol

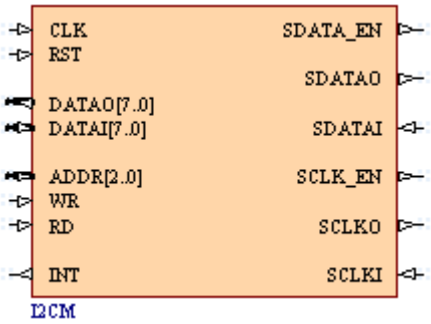

Figure 1. I2C Master Controller Symbol – non-Wishbone version (I2CM)

Pin description

Table 1. I2CM pin description

Name	Type	Polarity/ Bus size	Description
Control Signals			
CLK	I	Rise	External system clock
RST	I	High	External system reset
Host Microcontroller Interface Signals			
DATAO	O	8	Data received from an I2C slave device (valid when INT is asserted).
DATAI	I	8	Data to be transmitted to an I2C slave device.
ADDR	I	3	Read/Write address for internal registers
WR	I	High	Asserted when the CPU wants to write to an internal register.
RD	I	High	Asserted when the CPU wants to read an internal register.
INT	O	High	Asserted when data is received from an I2C slave device.
I2C Bus Interface Signals			
SDATA_EN	O	High	Output enable signal for the I2C data bidirectional buffer.
SDATAO	O	-	Serial data output.
SDATAI	I	-	Serial data input.
SCLK_EN	O	High	Output enable signal for the I2C clock bidirectional buffer.
SCLKO	O	-	Serial clock output.
SCLKI	I	-	Serial clock input.

Note: To simplify using the bidirectional DATA, SDATA and SCLK buses, the schematic symbol includes a bus pin for each direction, allowing them to be wired independently.

Hardware Description

Block Diagram

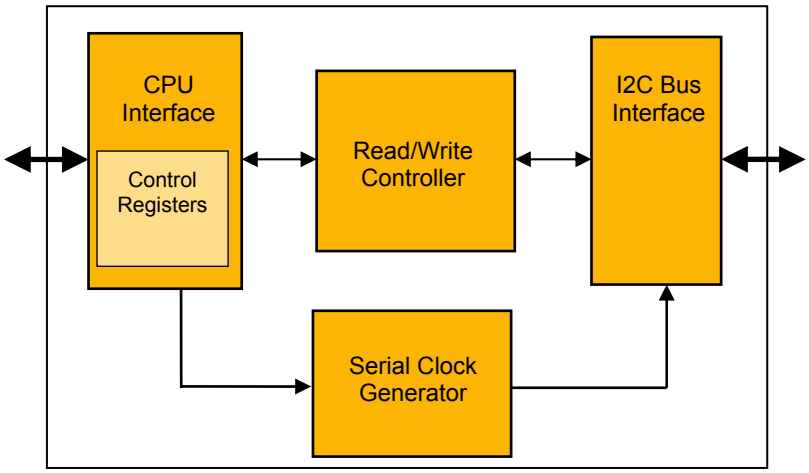

Figure 2. I2CM Bus Controller block diagram

Register Description

Register Interface

The register interface for the I2CM requires the following pins: DATAO, DATAI, ADDR, RD, and WR.

WRITE:

To write to an internal register, put the desired value on the DATAI bus. Put the desired address on the ADDR input bus. Assert the WR signal for a minimum of 1 period of the external system clock signal (CLK). The value of the internal register is now updated.

READ:

To read from an internal register, put the desired address on the ADDR bus. Assert the RD signal for a minimum of 1 period of the external system clock signal (CLK). The content of the register appears on the DATAO output bus.

Internal Registers – locations

Table 2. I2CM internal register locations

Register Name	Read/Write	Location
CONTROL_REG	R/W	0
STATUS_REG	RO	1
CLK_0_REG	R/W	2
CLK_1_REG	R/W	3
WR_DAT_REG	R/W	4
RD_DAT_REG	RO	5

Note: RO = Read Only
R/W = Read and Write

Internal Registers - reset values

Table 3. I2CM internal register reset values

Register Name	Reset Value
CONTROL_REG	00
STATUS_REG	00
CLK_0_REG	00
CLK_1_REG	00
WR_DAT_REG	00
RD_DAT_REG	00

Note: Only bits 0 to 2 of the Status Register are used. Bits 3 to 7 are always read as 0.

Internal Registers – Descriptions

Table 4. I2CM internal register descriptions

Register Name	Description
CONTROL_REG	**Bit 0:** ENABLE – to keep the Controller 'alive' this bit must be set all the time. It is important to remember to set this bit every time the register is written to. Once this bit is cleared the Controller won't respond to any other bit set in the register. When accidentaly cleared the Controller will go into its stand-by mode and won't start any transfer. **Bit 1:** IEN – this bit enables the interrupt line to be driven by the Controller in response to transfer finish or any error that might happen on the I2C bus. If this bit is cleared the Controller will not pull the interrupt line high. If the interrupt line is not used this bit could be used to see when the Controller has finished the requested operation. **Bit 2:** IACK - this bit represents the interrupt acknowledgement that is sent from the host microcontroller to the I2CM Controller. Once an interrupt is issued by the Controller, the host must clear this bit before any other transfer is started. **Bit 3:** WR – this bit initiates write transfer. When set, the Controller will transfer the contents of the WR_DAT_REG register to an I2C slave device. **Bit 4:** RD – this bit initates a read transfer. When set, the Controller will shift in 8-bit data from an I2C slave device. The data will be loaded into the RD_DAT_REG register, which in turn can be read by the host microcontroller. **Bit 5:** STO – this bit, when set, will generate a STOP condition. When this bit is set together with the WR or RD bits, the Controller will write or read 8-bit data and then generate a STOP condition. Use when sending or receiving the last data byte to stop communication. **Bit 6:** STA – this bit, when set, will generate a START condition. If the WR bit is also set then immediately after generating a start condition the Controller will attempt to write the contents of the WR_DAT_REG register onto the I2C bus. **Bit 7:** NACK – this bit is used only when the Controller is reading data from a slave I2C device and represents the data line on the ninth clock. If set, then on the ninth clock pulse the data line will be high which is defined as NOT ACKNOWLEDGE by I2C protocol. If cleared, then the data line will be held low which is defined as ACKNOWLEDGE by I2C protocol. This is useful when reading more than one byte of data, where a slave device waits for the I2CM Controller to acknowledge that the transfer was successful, so that it can prepare to send another byte.
STATUS_REG	**Bit 0:** INTREQ – this bit is set by the Controller when the it has finished its current operation. Every time the CONTROL_REG register is changed and any command is issued to the Controller, it will respond by setting this bit as soon as the requested operation is finished. INTREQ will only go high if bit 1 in the CONTROL_REG register (IEN) is set. It is automatically cleared when bit 2 in the CONTROL_REG register (IACK) is set.

Register Name	Description
	Bit 1: RXACK – this bit represents the state of the data line at the ninth clock pulse while sending data. When it is 0 then it is considered the slave device acknowledged the transfer. If 1, then NACK was received. When sending an address this bit indicates if a slave device with the address issued is connected to the Controller or not. **Bit 2:** BUSY – this bit is set while the Controller performs the requested operation. It is cleared when the Controller enters 'idle' state and waits for a command in the CONTROL_REG register.
CLK_0_REG	Low order byte of clock scaling register (CLK_REG). This register stores the value used to scale the frequency of the clock that the master generates.
CLK_1_REG	High order byte of clock scaling register (CLK_REG). This register stores the value used to scale the frequency of the clock that the master generates.
WR_DAT_REG	The data to be written to the slave device. If it's the first byte being sent to the device, bit 0 indicates a read/write and the other 7 bits indicate the slave address. Bit 0 = 0 – Write Bit 0 = 1 – Read
RD_DAT_REG	The data received from the slave device during a read.

Note: More than one bit can be set in the CONTROL_REG register at the same time.
All bits of the STATUS_REG register are read only and represent the current state of the I2CM Controller.

Defining the frequency of SCLK

The I2C Controller incorporates a 16-bit register, CLK_REG, whose stored value is used to scale the frequency of the clock signal generated by the Controller – SCLK. This register is further sub-divided into two 8-bit registers – CLK_0_REG (low 8 bits) and CLK_1_REG (high 8 bits).

The value you write to these registers depends on the ultimate frequency for SCLK that you wish to achieve. The following formula can be used to calculate the value for CLK_REG:

$$\text{CLK\_REG Value} = \frac{\text{CLK}}{5 \times \text{SCLK}} - 1$$

where,

CLK is the frequency of the external system clock

SCLK is the the desired frequency of the serial clock generated by the Controller.

Note: If the value written to CLK_REG is 0000h, SCLK will not be generated (i.e. will have frequency of zero).

Operation

There are two modes of operation with respect to the I2C Controller – Write and Read. These modes are detailed in the following sections.

Write

1. The CPU writes the address of the slave (7 bits) to the WR_DATA_REG. Bit 0 must be '0' to indicate a write.
2. The CPU will set the STA and WR bits in the CONTROL_REG.
3. The I2C master will then attempt to generate a start signal and send the address on the I2C serial interface.
4. The I2C master will generate an interrupt when an ACK is received from the slave if the address matches. (Otherwise a NACK is generated and the slave address does not exist).
5. The CPU will write the data it wants to transmit to the slave in the WR_DATA_REG and set the write bit in the CONTROL_REG.
6. The I2C master will then send the data on the I2C serial interface. The slave will respond again with an ACK.
7. The CPU will set the STO bit in the CONTROL_REG to indicate it is now finished with the bus.

Read

1. The CPU writes the address of the slave (7 bits) to the WR_DATA_REG. Bit 0 must be '1' to indicate a read.
2. The CPU will set the STA and WR bits in the CONTROL_REG(because we want to write the address!).
3. The I2C master will then attempt to generate a start signal and send the address on the I2C serial interface.
4. The I2C master will generate an interrupt when an ACK is received from the slave if the address matches. (Otherwise a NACK is generated and the slave address does not exist).
5. The CPU will set the RD bit in the CONTROL_REG. The I2C master is not ready to receive data from the slave. Note: the master still generates the serial clock.
6. After receiving 1 byte of information, the master will ACK/NACK the slave device, depending on the value on the NACK bit of the CONTROL_REG (bit 7).
7. The CPU will set the STO bit in the CONTROL_REG to indicate it is now finished with the bus.

Sending/receiving multiple bytes of data

To send two data bytes to a slave I2C device and then read another two bytes from the same device the following steps need to be taken:

1. Write the address of the slave I2C device that you wish to write to into WR_DAT_REG with LSB cleared
2. Set the following bits in the Control register (CONTROL_REG): ENABLE, IEN, WR, STA
3. Wait for interrupt and read bit 1(RXACK) of the Status register (STATUS_REG) to confirm the address is valid
4. Acknowledge interrupt by setting ENABLE and IACK bits in the Control register
5. Write first data byte to WR_DAT_REG
6. Set the following bits in the Control register: ENABLE, IEN, WR
7. Wait for interrupt and confirm that the slave device received the first byte of data (by interrogating bit 1 (RXACK) of the Status register)
8. Acknowledge interrupt by setting ENABLE and IACK bits in the Control register
9. Write second data byte to WR_DAT_REG
10. Set the following bits in the Control register: ENABLE, IEN, WR
11. Wait for interrupt and confirm that the slave device received the second byte of data (by interrogating bit 1 (RXACK) of the Status register)
12. Acknowledge interrupt by setting ENABLE and IACK bits in the Control register
13. Write the address of the same slave I2C device that you wish to read from into WR_DAT_REG with LSB set
14. Set the following bits in the Control register: ENABLE, IEN, WR, STA
15. Acknowledge interrupt by setting ENABLE and IACK bits in the Control register
16. Set the following bits in the Control register: ENABLE, IEN, RD
17. Wait for interrupt and read first byte of data from RD_DAT_REG
18. Acknowledge interrupt by setting ENABLE and IACK bits in the Control register
19. Set the following bits in the Control register: ENABLE, IEN, RD, STO
20. Wait for interrupt and read second byte of data from RD_DAT_REG.

Clock Stretching

Some devices might not be ready to send data immediately after acknowledging being addressed by the I2CM Controller. A good example is the MAX1037 device, which can be found on NanoBoard. In one of its modes it needs to perform analog to digital conversion before it can send data to the I2CM Controller. To this end, the I2CM Controller has support for clock stretching - whereby a slave device can insert wait states onto the I2C bus while it prepares the data to be sent to the Controller. A wait state is defined as a slave I2C device driving the clock line low. In this instance, the I2CM Controller should wait until the clock line is released before attempting to receive a data byte.

Revision History

Date	Version No.	Revision
30-Aug-2011	1.0	Initial release (old doc content with Wishbone version removed)