LatticeMico Support for OpenCores I^{2}C Master

This document describes Lattice Semiconductor support for the OpenCores I^{2}C master component included in the Mico System Builder (MSB).

Version

This document describes the 3.1 version of the OpenCores I^{2}C master component modified for integration in MSB.

Features

Refer to the OpenCores I^{2}C master component data sheet for the component features.

Functional Description

Refer to the OpenCores I^{2}C master component data sheet for a functional description of the component.

Configuration

The following sections describe the graphical user interface (UI) parameters, the hardware description language (HDL) parameters, and the I/O ports that you can use to configure and operate the OpenCores I^{2}C master component from MSB.

UI Parameters

Table 1 shows the UI parameters available for configuring the OpenCores I2C master component through the MSB interface.

Table 1: OpenCores I2C Master Component UI Parameters

Dialog Box Option	Description	Allowable Values	Default Value
Instance Name	Specifies the name of the OpenCores I2C master component instance.	Alphanumeric and underscores	i2cm_oc
Base Address	Specifies the base address for the device. The minimum byte alignment is 0X80.	0X80000000 – 0XFFFFFF80 If other components are included in the platform, the range of allowable values will vary.	0X80000000
SCL Speed	Specifies the serial clock-line speed.	400 or 200 KHz	400

HDL Parameters

Table 2 describes the parameters that appear in the HDL.

Table 2: OpenCores I2C Master Component HDL Parameters

Parameter Name	Description	Allowable Values
SPEED	Specifies the serial clock-line speed, in kilohertz.	400 or 200
SYSCLK	Specifies the core clock frequency, in megahertz.	Valid clock speed, for example, 25

I/O Ports

Table 3 describes the input and output ports of the OpenCores I2C master component.

Table 3: OpenCores I2C Master Component I/O Ports

Port Name	Active	Direction	Initial State	Description
System Clock and Reset				
CLK_I	–	I	X	System clock
RST_I	High	I	X	System reset
WISHBONE Interface				
I2CM_ADR_I	–	I	X	Address input array, which is the address generated by the master
I2CM_CYC_I	High	I	X	When asserted, the cycle input indicates that a bus cycle is in progress.
I2CM_DAT_I	–	I	X	Data input array, which is valid for a write request
I2CM_LOCK_I	High	I	X	If the lock input is asserted, the current cycle becomes uninterruptible.

Table 3: OpenCores I2C Master Component I/O Ports (Continued)

Port Name	Active	Direction	Initial State	Description
I2CM_SEL_I	High	I	X	Select input array, which indicates where the valid data is expected on a data bus.
I2CM_STB_I	High	I	X	When asserted, the strobe input indicates that the slave is selected.
I2CM_WE_I	–	I	X	Write signal. Value of 1 is used for a write and 0 for a read.
I2CM_ACK_O	High	O	0	When asserted, the acknowledge output indicates normal cycle termination.
I2CM_ERR_0	High	O	0	Error output, which is present to conform to the WISHBONE interface but is never asserted by this core.
I2CM_RTY_0	High	O	0	Retry output, which is present to conform to the WISHBONE interface but is never asserted by this core.
I2CM_DAT_O	–	O	0	Data output array
I2CM_CTI_I	High	I	X	Slave CTI signal
I2CM_BTE_I	High	I	X	Slave BTE signal
I2CM_LOCK_I	High	I	X	Slave lock signal
I2C Interface Ports				
SDA	–	I/O	0/X	Bidirectional I2C serial data line
SCL	–	I/O	0/X	Bidirectional I2C serial clock line
Other Auto-Connected Internal Signals				
INTR_N	High	O	0	Interrupt request outputs

Register Definitions

The OpenCores I2C master component has byte-aligned and byte-wide registers. For integration into the 32-bit LatticeMico environment, the OpenCores I2C mster component's top-level RTL has been modified to adapt the byte-aligned, byte-wide registers to word-aligned (4-byte), byte-wide registers. The functionality, meaning of the registers and the register-contents, or both remain unmodified.

Table 4 shows the register map for the adapted OpenCores I2C master component.

Refer to the OpenCores I2C master data sheet for a comprehensive description on the component's register interface.

Table 4: Register Map

Register Name	Offset	7–0
Prescale register – low byte	0x00	PRERlo
Prescale register – high byte	0x04	PRERhi
Control register	0x08	CTR
Command/status register	0x10	CR/SR

The structure shown in Figure 1 depicts the register map layout for the OpenCores I2C master component. The elements are self-explanatory and are based on the register, as shown in Table 4. This structure, which is defined in the OpencoresI2Cmaster.h header file, enables you to directly access the registers, if desired. The device driver for manipulating the component uses it internally.

Figure 1: OpenCores I2C Master Register Map Structure

```
typedef struct st_OCI2CMDev_t {
/* Read/Write=I2C SCL Prescale Low Byte */
        volatile unsigned int PrescaleLo;
/* Read/Write=I2C SCL Prescale High Byte */
        volatile unsigned int PrescaleHi;
/* Read/Write=Control; */
        volatile unsigned int Control;
/* Read=RxData,Write=TxData */
        volatile unsigned int Data;
/* Read=Status, Write=Command */
        volatile unsigned int StatusCommand;
    }OCI2CMDev_t;
```

Timing Diagrams

Refer to the OpenCores I2C master component data sheet for timing diagrams.

This component does not support WISHBONE burst read/write transactions.

EBR Resource Utilization

The OpenCores I2C master component uses no EBRs.

Software Support

This section describes the software support provided for the OpenCores I2C master component.

The support routines for this component are for use in a single-threaded environment. If they are used in a multi-tasking environment, you must provide re-entrance protections.

Device Driver

This section describes the type definitions for instance-specific structures and the device context structure.

Instance-Specific Structures

The MSB managed build process instantiates a unique structure per instance of the OpenCores I2C master component in the platform. These instances are defined in DDStructs.c. The information for these instance-specific structures is filled in by the managed build process, which extracts OpenCores I2C master component-specific information from the platform definition file. The members should not be manipulated directly because the structure is used exclusively by the device driver. You can retrieve a pointer to the instance-specific OpenCores I2C master component device context structure by using the MicoGetDevice function call of the LatticeMico device lookup service. Refer to the *LatticeMico32 Software Developer User Guide* for more information on the device lookup service.

OpenCores I2C Master Device Context Structure

This structure, shown in Figure 2, contains OpenCores I2C master component-specific information and is dynamically generated in the DDStructs.h header file. This information is largely filled in by the MSB managed build process, which extracts the OpenCores I2C master component-specific information from the platform definition file. The members should not be manipulated directly, because this structure is for exclusive use by the device driver.

Figure 2: OpenCores I2C Master Component Device Context Structure

```
typedef struct st_OpenCoresI2CMasterCtx_t {
    const char*   name;
    unsigned int   base;
    unsigned int   intrLevel;
    unsigned int   speed;
    DeviceReg_t   lookupReg;
    unsigned int   controlReg;
    void *   userCtx;
    void *   callback;
    void *   prev;
    void *   next;
} OpenCoresI2CMasterCtx_t;
```

Table 5 describes the parameters of the OpenCores I2C master component device context structure shown in Figure 2.

Table 5: OpenCores I2C Master Component Device Context Structure Parameters

Parameter	Data Type	Description
name	const char *	OpenCores I2C master instance name
base	unsigned int	MSB-assigned interrupt, if interrupts are used. If interrupts are not used, this value is greater than 31. If interrupts are used, the value is 0–31.
intrLevel	unsigned int	MSB-assigned interrupt, if interrupts are used. If interrupts are not used, this value is greater than 31. If interrupts are used, the value is 0–31.
speed	unsigned int	I2C serial clock, in kilohertz, as specified in the GUI
lookupReg	DeviceReg_t	Used by the device driver to register the OpenCores I2C master component instance with the LatticeMico lookup service. Refer to the *LatticeMico32 Software Developer User Guide* for a description of the DeviceReg_t data type.
prev	void *	Used internally by the lookup service
next	void *	Used internally by the lookup service

Functions

This section describes the application programming interface (API) for using the OpenCores I2C master component.

OpenCoresI2CMasterInit Function

```
void OpenCoresI2CMasterInit (OpenCoresI2CMasterCtx_t *ctx );
```

This function initializes an OpenCores I2C master component instance. It is called as part of the platform initialization for managed builds for each instance of the component. This function sets the prescale register values and enables the core for future use.

Table 6 describes the parameter in the OpenCoresI2CMasterInit function syntax.

Table 6: OpenCoresI2CMasterInit Function Parameter

Parameter	Description	Notes
OpenCoresI2CMasterCtx_t *	Pointer to an OpenCores I2C master component context	For a managed build, the structure referenced is located in the DDStructs.c file.

OpenCoresI2CMasterEnableFunction

```
void OpenCoresI2CMasterEnable (OpenCoresI2CMasterCtx_t *ctx
);
```

This function enables the OpenCores I2C master component instance specified by the device context parameter (ctx).

Table 7 describes the parameter in the OpenCoresI2CMasterEnable function syntax.

Table 7: OpenCoresI2CMasterEnable Function Parameter

Parameter	Description	Notes
OpenCoresI2CMasterCtx_t *	Pointer to an OpenCores I2C master component context	For a managed build, the structure referenced is located in the DDStructs.c file.

This function does not return any value.

OpenCoresI2CMasterDisable Function

```
void OpencoresI2CMasterDisable( OpenCoresI2CMasterCtx_t *ctx );
```

This function disables the OpenCores I2C master component instance specified by the device context parameter (ctx). Subsequent OpenCores I2C master operations will fail unless the core is explicitly enabled using the OpencoresI2CMasterEnable function.

You must be careful when disabling the core, because it can hang the bus, depending on the current state of the core (and any pending transactions).

Table 8 describes the parameter in the OpencoresI2CMasterDisable function syntax.

Table 8: OpenCoresI2CMasterDisable Function Parameter

Parameter	Description	Notes
OpenCoresI2CMasterCtx_t *	Pointer to an OpenCores I2C master component context	For a managed build, the structure referenced is located in the DDStructs.c file.

This function does not return any value.

OpenCoresI2CMasterStart Function

```
int OpenCoresI2CMasterStart( OpenCoresI2CMasterCtx_t *ctx );
```

This function issues an I2C start on the I2C bus. While the read/write functions issue a start before performing a read/write transaction, this function allows you to arbitrate for the bus independently of the read/write.

As with any shared communication media, exercise care when arbitrating for bus usage.

Table 9 describes the parameter in the OpenCoresI2CMasterStart function syntax.

Table 9: OpenCoresI2CMasterStart Function Parameter

Parameter	Description	Notes
OpenCoresI2CMasterCtx_t *	Pointer to an OpenCores I2C master component context	For a managed build, the structure referenced is located in the DDStructs.c file.

Table 10 shows the values returned by the OpenCoresI2CMasterStart function.

Table 10: OpenCoresI2CMasterStart Return Values

Return Value	Description
0	Function successfully issued a start; that is, it did not detect “arbitration lost” as part of issuing a start.
Non-zero	Function issued a start but detected “arbitration lost.”

OpenCoresI2CMasterStop Function

```
void OpenCoresI2CMasterStop( OpenCoresI2CMasterCtx_t *ctx );
```

This function issues a stop; that is, it relinquishes the bus. This must be done once all relevant read/write operations are complete. Issuing a stop gives up ownership of the bus that was acquired when issuing a start. This function does not stop the core; that is, it does not disable the core but rather relinquishes control of the bus to allow other masters to arbitrate for bus access. As with any shared communication media, exercise care when arbitrating for bus usage.

Table 11 describes the parameter in the OpenCoresI2CMasterStop function syntax.

Table 11: OpenCoresI2CMasterStop Function Parameter

Parameter	Description	Notes
OpenCoresI2CMasterCtx_t *	Pointer to an OpenCores I2C master component context	For a managed build, the structure referenced is located in the DDStructs.c file.

This function does not return a value.

Figure 3 illustrates a single-byte waveform.

OpenCoresI2CMasterWrite Function

```
int OpenCoresI2CMasterWrite( OpenCoresI2CMasterCtx_t *ctx,
                             unsigned int address,
                             unsigned int buffersize,
                             unsigned char *data);
```

This function performs block write. It is contingent on a successful start, that is, ownership of the bus. It issues a start before performing transactions but does not issue a stop when done. The user application must explicitly issue a stop (OpenCoresI2CMasterStop) when it is ready to relinquish the bus.

Note

The I2D slave address is a 7-bit address expected by the functions. The 7-bit address is internally shifted 1 bit left by this function.

Table 12 describes the parameters in the OpenCoresI2CMasterWrite function syntax.

Table 12: OpenCoresI2CMasterWrite Function Parameters

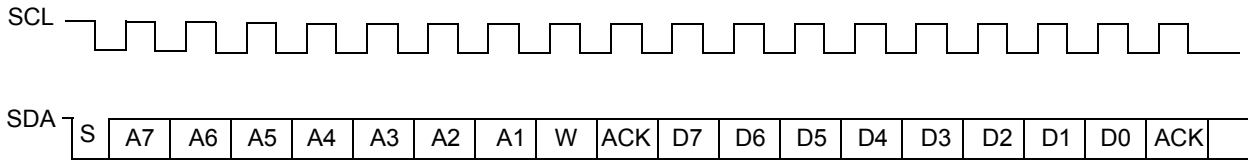
Parameter	Description	Notes
OpenCoresI2CMasterCtx_t *	Pointer to an OpenCores I2C master component context	For a managed build, the structure referenced is located in the DDStructs.c file.
unsigned int address	7-bit address of the I2C slave device	
unsigned int buffersize	Number of bytes to write	
unsigned char *data	Pointer to data bytes to write	

Table 13 shows the values returned by this function.

Table 13: OpenCoresI2CMasterWrite Return Values

Return Value	Description
0	Function successfully wrote the data
–1	Function failed because it did not receive an acknowledgment during addressing
–2	Function failed because it did not receive an acknowledgment for a write
–3	Function failed because the core detected loss of arbitration

Figure 3: Single-Byte Write Waveform



Note

After ACK, no stop is issued, allowing back-to-back writes, read, and reads following writes.

OpenCoresI2CMasterWriteByte Function

```
int OpenCoresI2CMasterWriteByte ( OpenCoresI2CMasterCtx_t *ctx,
                                  unsigned int address,
                                  unsigned char data);
```

This function performs a single byte write. It is contingent on a successful start, that is, ownership of the bus. This function issues a start before performing transactions but does not issue a stop when done. The user application must explicitly issue a stop (OpenCoresI2CMasterStop) when it is ready to relinquish the bus.

Table 14 describes the parameters in the OpenCoresI2CMasterWriteByte function syntax.

Note

The I2D slave address is a 7-bit address expected by the functions. The 7-bit address is internally shifted 1 bit left by this function.

Table 14: OpenCoresI2CMasterWriteByte Function Parameters

Parameter	Description	Notes
OpenCoresI2CMasterCtx_t *	Pointer to an OpenCores I2C master component context	For a managed build, the structure referenced is located in the DDStructs.c file.
unsigned int address	7-bit address of the I2C slave device	
unsigned char data	Byte data to write	

Table 15 shows the values returned by this function.

Table 15: OpenCoresI2CMasterWriteByte Return Values

Return Value	Description
0	Function successfully wrote the data
–1	Function failed since it did not receive an ack during addressing
–2	Function failed since it did not receive an ack for a write
–3	Function failed as the core detected loss of arbitration

Figure 3 and Figure 4 illustrate write and read waveforms, respectively.

OpenCoresI2CMasterRead Function

```
int OpenCoresI2CMasterRead ( OpenCoresI2CMasterCtx_t *ctx,
                             unsigned int address,
                             unsigned int buffersize,
                             unsigned char *data);
```

This function performs block read. It is contingent on a successful start, that is, ownership of the bus. This function issues a start before performing transactions but does not issue a stop when done. The user application must explicitly issue a stop (OpenCoresI2CMasterStop) when it is ready to relinquish the bus.

Note

The I2D slave address is a 7-bit address expected by the functions. The 7-bit address is internally shifted 1 bit left by this function.

Table 16 describes the parameters in the OpenCoresI2CMasterRead function syntax.

Table 16: OpenCoresI2CMasterRead Function Parameters

Parameter	Description	Notes
OpenCoresI2CMasterCtx_t *	Pointer to an OpenCores I2C master component context	For a managed build, the structure referenced is located in the DDStructs.c file.
unsigned int address	7-bit address of the I2C slave device	
unsigned int buffersize	Number of bytes to read	
unsigned char *data	Pointer to a location for storing the bytes read. The location must have sufficient space to hold at most the number of bytes equivalent to the buffer size.	

Table 17 shows the values returned by this function.

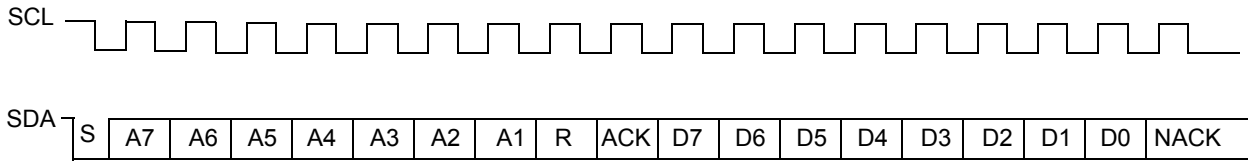
Table 17: OpenCoresI2CMasterRead Return Values

Return Value	Description
Buffersize	Function successfully read the desired number of bytes.

Table 17: OpenCoresI2CMasterRead Return Values (Continued)

Return Value	Description
–1	Function failed because it did not receive an ack during addressing
–3	Function failed because the core detected loss of arbitration

Figure 4: Single-Byte Read Waveform



Note

No stop is generated after reading the last byte, allowing back-to-back read/writes.

For multi-byte reads, NACK is generated only for the last byte read. All other bytes prior to the last byte are signaled with an ACK to indicate that more data is expected. .

OpenCoresI2CMasterReadByte Function

```
int OpenCoresI2CMasterReadByte ( OpenCoresI2CMasterCtx_t *ctx,
                                 unsigned int address,
                                 unsigned char *data);
```

This function reads a single byte. It is contingent on a successful start, that is, ownership of the bus. This function issues a start before performing transactions but does not issue a stop when done. The user application must explicitly issue a stop (OpenCoresI2CMasterStop) when it is ready to relinquish the bus.

Note

The I2D slave address is a 7-bit address expected by the functions. The 7-bit address is internally shifted 1 bit left by this function.

Table 18 describes the parameters in the OpenCoresI2CMasterReadByte function syntax.

Table 18: OpenCoresI2CMasterReadByte Function Parameters

Parameter	Description	Notes
OpenCoresI2CMasterCtx_t *	Pointer to an OpenCores I2C master component context	For a managed build, the structure referenced is located in the DDStructs.c file.
unsigned int address	7-bit address of the I2C slave device	
unsigned char *data	Pointer to a location for storing the byte read.	

Table 19 shows the values returned by this function..

Table 19: OpenCoresI2CMasterReadByte Return Values

Return Value	Description
1	Function successfully read the byte.
–1	Function failed because it did not receive an ack during addressing
–3	Function failed because the core detected loss of arbitration

Services

The OpenCores I2C master device driver registers the OpenCores I2C master instances with the LatticeMico lookup service, using their instance names for device names and “OCI2CMDevice” as the device type. If you want to reduce code size, you can disable this registration by defining the “`_OPENCORES_I2C_NOT_LOOKUPABLE_`” preprocessor definition.

For more information about using the lookup service, refer to the *LatticeMico32 Software Developer User Guide*.

Figure 4 illustrates a single-byte read waveform.

Software Usage Examples

Refer to the “Opencores I2C test” software template for a software usage example.

Revision History

Component Version	Description
1.0	Initial release.
3.0 (3.0 SP2)	No RTL update.
3.1	Updated software drivers. Specifically, the read/write transfer routines do not issue a stop. You must explicitly call OpenCoresI2CMasterStop. Added PlatformI and an Opencores_I2C_test software template for a hardware/ software example using OpenCores I2C master. Verified on HPE-MINI LatticeECP2/ECP LatticeMico32 DSP boards.