Avalon-MM Master Templates

Date: 09/08/2008

Disclaimer

These component templates may be used within Altera® devices only and remain the property of Altera. They are being provided on an "as-is" basis and as an accommodation, and therefore all warranties, representations, or guarantees of any kind (whether express, implied or statutory) including, without limitation, warranties of merchantability, non-infringement, or fitness for a particular purpose, are specifically disclaimed. Altera expressly does not recommend, suggest, or require that these examples be used in combination with any other product not provided by Altera.

If you encounter an issue or have an enhancement request, please log onto mySupport to file a service request.

Change List

[09/08/2008]

- Updated documentation to reflect the additional file ‘custom_master.v’ and GUI improvements (using callbacks)
- Removed benchmark information

Overview

Provided in this package are templates that can be used for adding Avalon-MM mastering capabilities to your own custom hardware. Each template is tailored towards a specific type of access which is important depending on the memories used in your SOPC Builder system. Signals are exposed at the top of the system so that you can connect them to your custom logic. You can also use the HDL provided in the template to add new capabilities to an existing SOPC Builder component. For example if you typically transfer data to/from your custom component using a DMA engine, you can use one of the masters provided to eliminate the need for a separate DMA.

Installation

In order to use the templates, simply extract the 'ip' directory into your own hardware project directory. SOPC Builder references the ip directory by default. When you open SOPC Builder the master templates will appear in the component listing on the left side of the SOPC Builder user interface. The master templates will appear in the group called "Templates". Select the appropriate master component for the access type you wish to use and add it to the system. You will be asked to setup parameters such as the data width while adding the master to the system. Refer to the HDL port listing section of this document to learn more about using these settings.

Master Block Diagrams

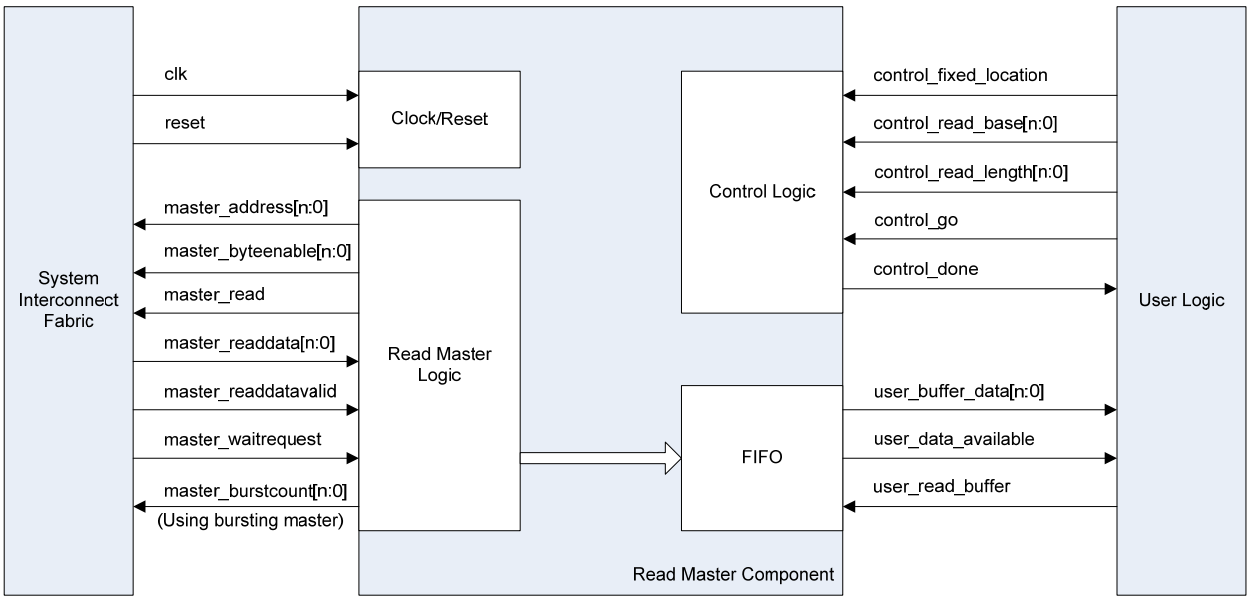

Figure 1. Read Master Template

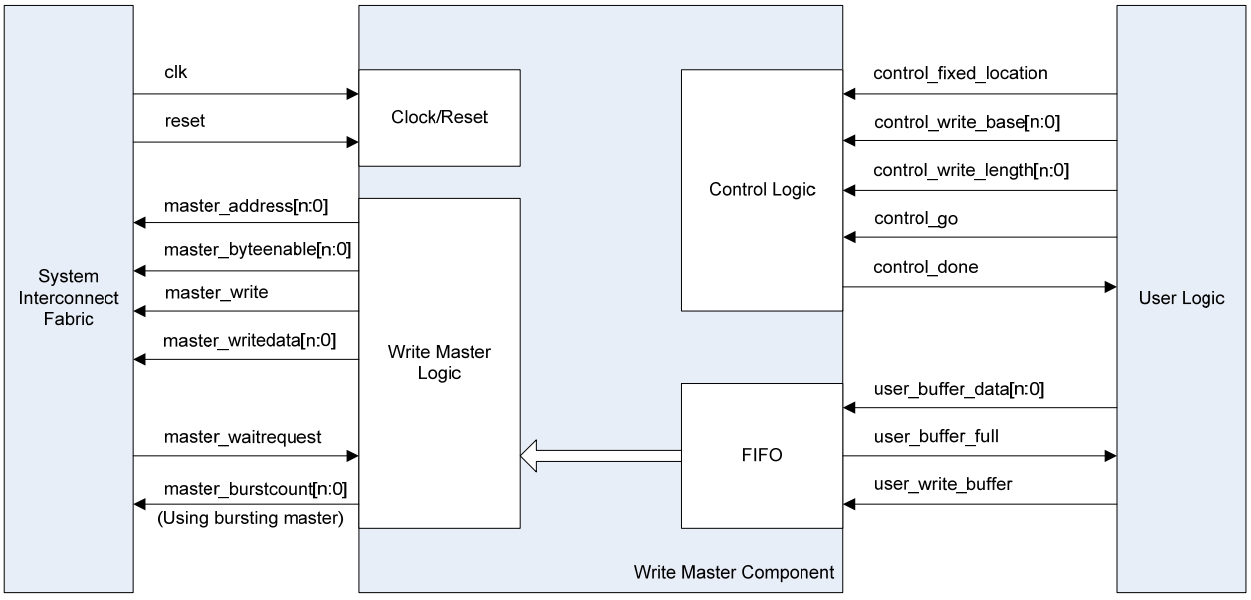

Figure 2. Write Master Template

Usage

There are two main interfaces exposed to your user logic, the control interface and the data interface. Before any data is transmitted you must first send control signals to the master component. The following table details each control signal and the usage:

Signal Name	Direction	Width	Usage
control_fixed_location	Input	1	When set the master address will not increment.
control_<x>_base	Input	n (default 32)	Word aligned byte address where the master will begin transferring data.
control_<x>_length	Input	n (default 32)	Number of bytes to transfer. This number must be a multiple of the data width in bytes (e.g. 32 bit data requires a multiple of 4)
control_go	Input	1	One clock cycle strobe that instructs the master to begin transferring. The fixed_location, base, and length values are registered on this clock cycle.
control_done	Output	1	Asserted and held when the master has transferred the last word of data. This occurs when the last write transfer completes or the last pending read returns. You can start the master again on the next cycle after done is asserted.
control_early_done	Output	1	This signal is only applicable to the read masters. It is asserted when the final read is posted and not when the final word returns from the system interconnect fabric. As a result this signal is always asserted before 'control_done'.

Table 1. Control Interface

Note: 'x' is either 'read' or 'write' depending on the master type. 'n' is configured to be equal to the parameter "ADDRESSWIDTH" which is set when you add the master component to the system.

Depending on whether you use a read or write master the user interface changes due to the data directions being different. In both cases, the user interface implements flow control using signals that represent empty/full and read/write. The following tables detail each user signal and the usage for the read and write master:

Signal Name	Direction	Width	Usage
user_buffer_data	Output	n (default 32)	Contains the next valid buffered data when 'user_data_available' is asserted.
user_data_available	Output	1	When asserted the user buffer contains valid data that has been read by the read master. You must not assert 'user_read_buffer' when this signal is de-asserted as this will cause a buffer underflow condition and the read master will fail to complete the entire transfer.
user_read_buffer	Input	1	Acts as a read acknowledge. When this signal is asserted the master component assumes your user logic has registered the data. On the next clock cycle the output of 'user_buffer_data' will be the next read value (if it has already been buffered).

Table 2. Read Master User Interface

Note: 'n' is configured to be equal to the parameter "DATAWIDTH"

Signal Name	Direction	Width	Usage
user_buffer_data	Input	n (default 32)	Valid data word that your logic writes into the user buffer. Use 'user_write_buffer' to qualify it as valid data when 'user_buffer_full' is de-asserted.
user_buffer_full	Output	1	When asserted the user buffer is full and you must not write any more data. Asserting 'user_write_buffer' while this signal is asserted may lead data being lost and the write master failing to complete the entire transfer.
user_write_buffer	Input	1	Acts as a write qualifier. Assert this signal to write valid data into the user buffer. You must not assert this signal if 'user_buffer_full' is asserted otherwise data overflow will occur.

Table 3. Write Master User Interface

Note: 'n' is configured to be equal to the parameter "DATAWIDTH"

HDL Port Listings

The following port listings are for each master HDL file. The only interfaces exposed to your logic by default are the control and user signals. If you insert the provided master templates into your system you will be prompted to enter parameterization values. You can also add the master HDL to your own SOPC Builder component and assign the signals and interfaces manually.

Burst Read Master

```
module burst_read_master (
     clk,
     reset,
     control_fixed_location,
     control_read_base,
     control_read_length,
     control_go,
     control_done,
     control_early_done,
     user_read_buffer,
     user_buffer_data,
     user_data_available,
     master_address,
     master_read,
     master_byteenable,
     master_readdata,
     master_readdatavalid,
     master_burstcount,
     master_waitrequest
);
```

Parameter	Range	Usage
DATAWIDTH	8, 16, 32, 64, 128, 256, 512, 1024 (default 32)	Data path width.
MAXBURSTCOUNT	1, 2, 4, 8, 16, 32, 64, 128 (default 4)	Maximum number of beats in a burst. Must be at most half of FIFODEPTH for the master to access memory locations efficiently.
BURSTCOUNTWIDTH	1-8 (default 3)	Log2(MAXBURSTCOUNT) + 1
BYTEENABLEWIDTH	1, 2, 4, 8, 16, 32, 64, 128 (default 4)	(DATAWIDTH)/8
ADDRESSWIDTH	1-32 (default 32)	The number of address bits exposed to the system interconnect fabric. This number must be large enough to span all the components connected to the master.
FIFODEPTH	4, 8, 16, 32, 64, 128 (default 32)	FIFO depth of the internal buffer. You should set this to be at least twice the MAXBURSTCOUNT value so that the master operates at peak efficiency.
FIFODEPTH_LOG2	2-7 (default 5)	Log2(FIFODEPTH)
FIFOUSEMEMORY	0/1 (default 1)	Set to ‘1’ to use on-chip memory for the internal buffer. Set to ‘0’ to use logic elements instead of memory (not recommend if FIFODEPTH is larger that 4)

Table 4. Burst Read Master Parameterization Values

Burst Write Master

```
module burst_write_master (
     clk,
     reset,
     control_fixed_location,
     control_write_base,
     control_write_length,
     control_go,
     control_done,
     user_write_buffer,
     user_buffer_data,
     user_buffer_full,
     master_address,
     master_write,
     master_byteenable,
     master_writedata,
     master_burstcount,
     master_waitrequest
);
```

Parameter	Range	Usage
DATAWIDTH	8, 16, 32, 64, 128, 256, 512, 1024 (default 32)	Data path width.
MAXBURSTCOUNT	1, 2, 4, 8, 16, 32, 64, 128 (default 4)	Maximum number of beats in a burst. Must be at most half of FIFODEPTH for the master to access memory locations efficiently.
BURSTCOUNTWIDTH	1-8 (default 3)	Log2(MAXBURSTCOUNT) + 1
BYTEENABLEWIDTH	1, 2, 4, 8, 16, 32, 64, 128 (default 4)	(DATAWIDTH)/8
ADDRESSWIDTH	1-32 (default 32)	The number of address bits exposed to the system interconnect fabric. This number must be large enough to span all the components connected to the master.
FIFODEPTH	4, 8, 16, 32, 64, 128 (default 32)	FIFO depth of the internal buffer. You should set this to be at least twice the MAXBURSTCOUNT value so that the master operates at peak efficiency.
FIFODEPTH_LOG2	2-7 (default 5)	Log2(FIFODEPTH)
FIFOUSEMEMORY	0/1 (default 1)	Set to ‘1’ to use on-chip memory for the internal buffer. Set to ‘0’ to use logic elements instead of memory (not recommend if FIFODEPTH is larger that 4)

Table 5. Burst Write Master Parameterization Values

Pipeline Read Master

```
module latency_aware_read_master (
     clk,
     reset,
     control_fixed_location,
     control_read_base,
     control_read_length,
     control_go,
     control_done,
     control_early_done,
     user_read_buffer,
     user_buffer_data,
     user_data_available,
     master_address,
     master_read,
     master_byteenable,
     master_readdata,
     master_readdatavalid,
     master_waitrequest
);
```

Parameter	Range	Usage
DATAWIDTH	8, 16, 32, 64, 128, 256, 512, 1024 (default 32)	Data path width.
BYTEENABLEWIDTH	1, 2, 4, 8, 16, 32, 64, 128 (default 4)	(DATAWIDTH)/8
ADDRESSWIDTH	1-32 (default 32)	The number of address bits exposed to the system interconnect fabric. This number must be large enough to span all the components connected to the master.
FIFODEPTH	4, 8, 16, 32, 64, 128 (default 32)	FIFO depth of the internal buffer.
FIFODEPTH_LOG2	2-7 (default 5)	Log2(FIFODEPTH)
FIFOUSEMEMORY	0/1 (default 1)	Set to '1' to use on-chip memory for the internal buffer. Set to '0' to use logic elements instead of memory (not recommend if FIFODEPTH is larger that 4)

Table 6. Pipeline Read Master Parameterization Values

Simple Write Master

```
module write_master (
     clk,
     reset,
     control_fixed_location,
     control_write_base,
     control_write_length,
     control_go,
     control_done,
     user_write_buffer,
     user_buffer_data,
     user_buffer_full,
     master_address,
     master_write,
     master_byteenable,
     master_writedata,
     master_waitrequest
);
```

Parameter	Range	Usage
DATAWIDTH	8, 16, 32, 64, 128, 256, 512, 1024 (default 32)	Data path width.
BYTEENABLEWIDTH	1, 2, 4, 8, 16, 32, 64, 128 (default 4)	(DATAWIDTH)/8
ADDRESSWIDTH	1-32 (default 32)	The number of address bits exposed to the system interconnect fabric. This number must be large enough to span all the components connected to the master.
FIFODEPTH	4, 8, 16, 32, 64, 128 (default 32)	FIFO depth of the internal buffer.
FIFODEPTH_LOG2	2-7 (default 5)	Log2(FIFODEPTH)
FIFOUSEMEMORY	0/1 (default 1)	Set to '1' to use on-chip memory for the internal buffer. Set to '0' to use logic elements instead of memory (not recommend if FIFODEPTH is larger that 4)

Table 7. Simple Write Master Parameterization Values

Template

A file called “custom_master.v” has been provided which will wrap all four masters by exposing all the signals required for all the masters and only instantiating the necessary master module. Signals that are not required are stubbed before they are connect to the system interconnect fabric. This is provided by the validation and elaboration callbacks provided in the component tcl file called “custom_masters_hw.tcl”. This tcl file allows you to make setting changes complete with validation and automatically derived settings. Since all the validation is provided you do no need to worry about putting the component into an unsupported configuration.

To learn more about callbacks and the component interface tcl scripting refer to the following chapter of the Quartus II handbook, Volume 4 SOPC Builder “Component Interface Tcl Reference”

Choosing a Master Type

In this package four Avalon-MM masters have been provided. To avoid unnecessary logic and complexity carefully consider which master to use. If you need to have memory writing access then use a write master. If you need to have memory reading access then use a read master. Bursting read and write masters have been provided to perform efficient access to bursting slave ports commonly used in IP such as PCI, PCIe, SRIO, SDRAM, etc… A burst is a sequential set of accesses which attempt to minimize the latency of the transfer. You should use a bursting master only if the slave ports you are connecting to support bursting otherwise SOPC Builder will insert burst adapter logic to adapt the two transfer formats.

Some examples of typical maximum burst counts are as follows:

SRIO – 64 (32 bit data), 32 (64 bit data)
PCI – 128
PCIe – 128
DDR SDRAM (full rate) – 1 (DDR burst of 2), 2 (DDR burst of 4), 4 (DDR burst of 8)
DDR SDRAM (half rate) – 1 (DDR burst of 4), 2 (DDR burst of 8)
DDR2 SDRAM (full rate) – 2 (DDR burst of 4)
DDR2 SDRAM (half rate) – 1 (DDR burst of 1)

The burst capable master templates will post busts in an efficient manor. The bursting write master will wait until your logic has provided enough words of data to complete a burst transaction. The bursting read master will wait until your logic has emptied the buffer to the point where there is enough room to post a full read burst.

By default each master is configured to address a 4GB address space. Most systems do not utilize the full 4GB address space available in SOPC Builder. As a result you can reduce the size of the custom master by configuring the ‘ADDRESSWIDTH’ parameter. By reducing the address range of the custom master you will also increase the Fmax of the master logic. To reduce the logic resource usage you should also set the ‘FIFOUSEMEMORY’ parameter to 1 so that the user buffer is created using on-chip memory instead of logic elements.

Caution

In this section corner cases that you should avoid will be covered. Workarounds will be suggested to avoid exercising the hardware in an improper way.

Native Slave Port Access

Each master template performs word size accesses only. In order to transfer data to an Avalon-MM slave port that uses native addressing you should match the master template data width to the slave port data width. Altera recommends that you upgrade any custom native addressing components you have created to use dynamic addressing when possible. Dynamic addressing simply means that you expose byte enables to the system interconnect fabric so that masters that do not use the same data width can be handled automatically by SOPC Builder.

Address Alignment

The master templates only support word size data alignment. The word size is dictated by the data width you choose when instantiating the master in your SOPC Builder system. For example, if you use a master data width of 64 bits, then you must use 8 byte alignment. The read/write start address must be a multiple of the word size in bytes. The length of the transfer must also be a multiple of the word size in bytes. Providing unaligned start addresses or transfer lengths that are not multiples of the word size can lead to unpredictable behavior.

If you use a processor to determine the start location for the master to access you can use compiler attributes or linker settings ensure alignment. You can also allocate an extra buffer element than what your system needs and modify the base address using the following equation:

Word aligned address = (original address + word size in bytes) & WORD_MASK;

This equation simply shifts the base address up to a higher location and the masking operation places it onto a word boundary. WORD_MASK can have the following values:

0xFFFFFFFE – 2 byte alignment
0xFFFFFFFC – 4 byte alignment
0xFFFFFFF8 – 8 byte alignment
0xFFFFFFF0 – 16 byte alignment
0xFFFFFFE0 – 32 byte alignment
0xFFFFFFC0 – 64 byte alignment
0xFFFFFF80 – 128 byte alignment

Minimum Transfer Size

You must never attempt to transfer less than a full data word using any of the master templates. The control logic cannot handle being started and completing on the same clock cycle. If the word size is 4 bytes then you must present a transfer length that is at least 4.