

D3.2 Internal release all protocols after feedback

Project number:	216541
Project acronym:	MULTI-BASE
Project title:	Scalable Multi-tasking Baseband for Mobile Communications
Start date of the project:	01/2008
Duration:	36M

Deliverable type:	Report
Deliverable reference number:	216541 / D3.2 / 1.0
Deliverable title:	Internal release all protocols after feedback
WP contributing to the deliverable:	WP3
Due date:	M20
Actual submission date:	September 2009

Responsible organisation:	LiU
Authors:	LiU (Andreas Ehliar, Dake Liu)
Abstract:	This document contains information about the protocols used by the integration platform.
Keywords:	Protocol Specification, OCN

Dissemination level:	PU
Revision:	FINAL 1.0

Instrument:	STREP
Thematic Priority:	ICT

Table of Contents

1	Executive summary	4
2	Introduction	5
2.1	Connections between modules will be relatively static	5
2.2	The platform controller should not be responsible for baseband calculations	6
2.3	Two dataformats	6
2.4	Two types of communication modes	6
2.5	The OCN is used for communication between modules	6
3	Protocol specification	8
3.1	Link level signals	8
3.1.1	clk	9
3.1.2	rst	9
3.1.3	datain	9
3.1.4	dataout	9
3.1.5	addr_out	9
3.1.6	addr_in	9
3.1.7	fc_input_ready	9
3.1.8	fc_output_ready	9
3.1.9	strobe_input	9
3.1.10	strobe_out	9
3.1.11	send_data	9
3.1.12	config_enable_out	10
3.1.13	config_enable_in	10
3.1.14	config_status_in	10
3.1.15	config_status_out	10
4	Examples	11
4.1	Chain of modules	11
4.2	Chain of modules, different clock domains	12
4.3	Module to memory	13
4.4	Module to memory, both read and write	13
5	Configuration commands for memories	15
5.1	Configuration method, 8 bit wide links	15
5.2	Configuration method, 32 bit wide links	15
5.3	Incremental	16
5.4	Incremental with variable stepsize	16
5.5	Ring buffer	16
5.6	Bit reversed addressing with offset	17
6	OCN Implementation issues	18
7	List of Abbreviations	19
8	References	20

Table of Figures

Figure 1: An example OCN configured by the platform controller (Senior)	5
Figure 2: Chain of modules, the output of module 1 is connected to the input of module 2.....	11
Figure 3: Signal waveform example for a simple chain of modules using the same clock.....	11
Figure 4: Signal waveform example for a simple chain of modules using the same clock.....	12
Figure 5: Chain of modules, the modules are not using the same clock domain.....	12
Figure 6: Signal waveform example for a simple chain of modules using two clocks	12
Figure 7: Signal waveform of module connected to memory via the OCN	14
Figure 8: Signal waveform of module connected to memory via the OCN while changing configuration mode of the AGU.....	14

Table of Tables

Table 1: Profiling parameters for the maximum data rate.....	7
Table 2: Example table	8
Table 3: AGU configuration for incremental addressing mode.....	15
Table 4: AGU configuration for incremental addressing mode.....	16
Table 5: AGU configuration for incremental addressing mode with variable stepsize	16
Table 6: AGU configuration for circular buffer	17
Table 7: AGU configuration for incremental addressing mode with variable stepsize	17

1 Executive summary

This deliverable presents the interface used by the on-chip network to connect the major functional modules of the MULTI-BASE project. The interface is designed to allow for a flexible implementation of the on-chip network while remaining easy to use and imposing little overhead for the modules designed in WP02 and WP04. The interface is based on a circuit connected model where the platform controller is responsible for setting up connections.

The interface is mainly based on two communication scenarios; direct module to module communication mode and memory handover mode where exclusive access to a memory is handed over via the OCN. In the memory handover mode it is also possible to configure the memory to use different addressing modes such as a circular buffer or bit-reversed addressing mode.

2 Introduction

The protocol specification described in this document is based on the following assumptions and requirements:

2.1 Connections between modules will be relatively static

In a general purpose computer based on a general bus such as AMBA [1] or CoreConnect [2] it is possible for any bus master to request access to any slave at any time. A central arbiter determines which master will get access to the bus. This is a simple and robust solution which is easy to verify, but the total bandwidth is not very high.

If a crossbar is used instead of a bus, all masters can read or write data simultaneously as long as they are accessing different slaves. However, the arbitration will be much more complex than in the bus case.

In baseband processing the access patterns are quite different from a general purpose system however. A master will access a slave for a relatively long time (thousands of clock cycles) before it needs to access another slave. This means that there is no need for a complex arbiter and that connections can be setup statically by the platform controller [3]. This is illustrated in Figure 1.

We also intend to take into account the fact that certain masters will never need to be connected to certain slaves (e.g. it would not make sense to connect any FEC module to the channel estimator. It will also not make sense to connect DFE module to the De-mapper, or to a FEC module.).

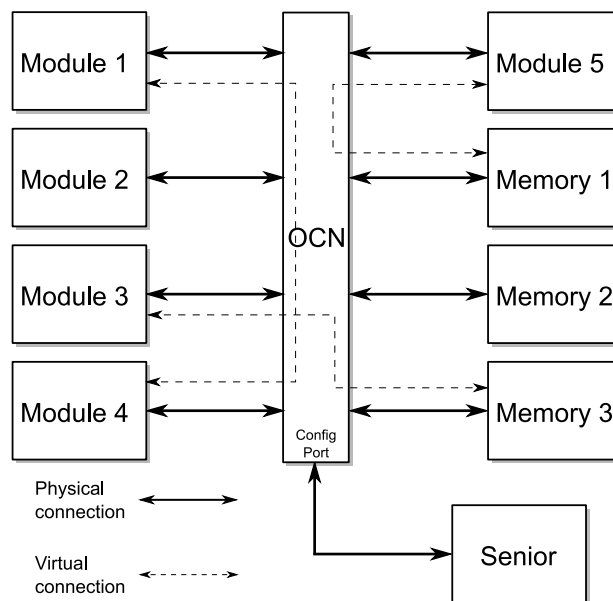


Figure 1: An example OCN configured by the platform controller (Senior)

2.2 The platform controller should not be responsible for baseband calculations

The platform controller (Senior) [4] should be used mainly for control tasks such as configuring the OCN and not for signal processing tasks. The intention is that all signal processing tasks will be performed either by dedicated hardware modules or accelerators designed by the partners.

The platform controller does have some DSP capabilities and will therefore be able to support a small amount of signal processing tasks if this is required, but the main use of it will be:

- Bootloading and initialization
- Hardware parameter configuration to function modules
- Arbitration of OCN based on static priorities and long lived access patterns
- MAC layer control interface
- Controlling the external memory subsystem and DMA (if required)

2.3 Two dataformats

We have identified two types of data that will be commonly used on the OCN:

- **Complex valued data (32 bit):** This format is used mainly for data which is complex valued, such as I/Q values from the A/D interface and data used by the FFT.
- **Short integers (8 bit):** This format is used mainly for packet data

A module can be connected to either the complex data network, the short integer data network, or both. Optionally a module may also send configuration vectors over the OCN although it is expected that this will not be a common operation.

2.4 Two types of communication modes

Modules can communicate using two different communication models:

- **Direct connection:** The output of one module is connected directly to the input of another module via the OCN.
- **Memory handover:** The output of one module is connected to a memory and fills the memory with data. Once the memory is full, the OCN is reconfigured so that the memory is connected to the input of another module (the module conducting the following task in task flow).

2.5 The OCN is used for communication between modules

The OCN is only planned and designed for communications between modules. This means that the maximum data rate will be $N \times (\text{MAX ADC sampling frequency}) \times (16b+16b)$. Here N is the number of antenna channels, we defined N=2 for mobile terminals. The MAX ADC sampling frequency will be 40MHz for 802.11n. All modules in the receiver signal processing chain will require either the same data rate or relaxed data rate. Similarly, the modules in the transmitter signal processing chain either leave the data rate unchanged or increase the data rate until the maximum data rate occurs at the analog interface.

This means that the maximum data rate required on the OCN is as follows:

$$MAXRATE = NUMCHANNELS \times ANALOGBITSPERCHANNEL \times SAMPLINGRATE$$

The worst case occurs when using 802.11n with two antennas. In that case the parameters in **Table 1** apply and the maximum datarate is 2560 Mb/s. With a linkwidth of 32 bits on the OCN, this result in a frequency of 80 MHz.

<i>NUMCHANNELS</i>	2
<i>ANALOGBITSPERCHANNEL</i>	32 bits (16 bits for I, 16 bits for Q)
<i>SAMPLINGRATE</i>	40 MHz

Table 1: Profiling parameters for the maximum data rate

This is a very low frequency for modern technology nodes, which means that there is a significant flexibility in how the OCN can be used. For example, while 802.11n with four antennas is not a priority for the MULTI-BASE consortium, the OCN frequency required to support this standard is only 160 MHz, which should be readily achievable.

3 Protocol specification

There are three kinds of OCN ports.

- Read only
- Write only
- Read/write

In some cases it is not necessary to send address information to the other side, such as when connecting the output of one module directly to the input of another (e.g. connecting two filters to each other in a streaming mode). It will also not usually be necessary to include address information when writing data to memories connected to the OCN since these memories will include their own address generators. However, if more advanced addressing modes are required, for example when using an OCN connected memory as intermediate storage when calculating an FFT, it will be possible to send an address signal to the OCN.

3.1 Link level signals

An OCN port has the following signals associated with it. All signals are synchronous with the clock signal arriving from the OCN.

Signal name	Direction	Port mode
clk	Input	
rst	Input	
datain[X:0]	Input	Read only, Read/write
dataout[X:0]	Output	Read/Write, Write only
addr_out[Y:0]	Output	Optional
addr_in[Y:0]	Input	Optional
fc_input_ready	Input	Optional for write only ports
fc_output_ready	Output	Optional for read only ports
strobe_in	Input	Read only, read/write
strobe_out	Output	Write only, read/write
send_data	Output	Read/write
config_enable_out	Output	Optional
config_enable_in	Input	Optional
config_status_out	Output	Optional
config_status_in	Input	Optional

Table 2: Example table

(X is either 31 or 7 depending on if this port is used for complex data or short integers. Y depends on how much address space that is required, but it is probably 15 or less.)

3.1.1 clk

Clock signal for this block. It is probably not necessary to include this signal in the OCN unless it makes it easier to integrate the functionality.

3.1.2 rst

Reset signal to this block. The reset is active positive and guaranteed to be active for at least 16 cycles in a row.

3.1.3 datain

Data input to this module (if appropriate)

3.1.4 dataout

Data output from this module (if appropriate)

3.1.5 addr_out

Address output from this module (if required)

3.1.6 addr_in

Address input to this module (if required). Most likely only memories will need this input.

3.1.7 fc_input_ready

There is data ready on the datain input. Use strobe_input to actually read it.

3.1.8 fc_output_ready

It is possible to send either data and/or an address to the OCN. Use strobe_output to actually send it.

3.1.9 strobe_input

Use this signal to read data from the OCN when fc_input_ready indicates that data is available. If this signal is asserted when fc_input_ready is 0 nothing will happen. As a consequence of this, a simple receiver which will always accept data can hardcode strobe_input as 1.

3.1.10 strobe_out

Use this signal to either write data to the OCN or send an address to the OCN when reading. If fc_output_ready is 0, the value of strobe_out will be ignored. This will simplify timing closure as strobe_out does not have to be generated combinatorially.

3.1.11 send_data

This, together with strobe_out, is used to indicate that the current access to the OCN is a write transaction. (The value of send_data is ignored unless strobe_out is activated)

3.1.12 config_enable_out

If this output is enabled, a configuration command is sent to the OCN. This is only intended to be used to configure address generators in memory blocks. (See the section about configuration commands for more information.)

3.1.13 config_enable_in

This signal is used to indicate that a configuration command is available on the inputs. It is only intended to be used to configure address generators for memory blocks.

3.1.14 config_status_in

Indicates the current configuration that was used to generate data on available on the input. When a module requests a configuration change it is difficult for this module to know whether data received corresponds to the old or the new configuration. Therefore it is possible to toggle the value of this bit when changing the configuration, making it easier for a module to handle this situation. See section 4.4 for an example of how this can be used.

3.1.15 config_status_out

Indicates the current configuration. This signal signals the current configuration used to generated data by this module to the data_out. See section 4.4 for an example of how this can be used.

4 Examples

This section contains various examples of how the OCN can be used.

4.1 Chain of modules

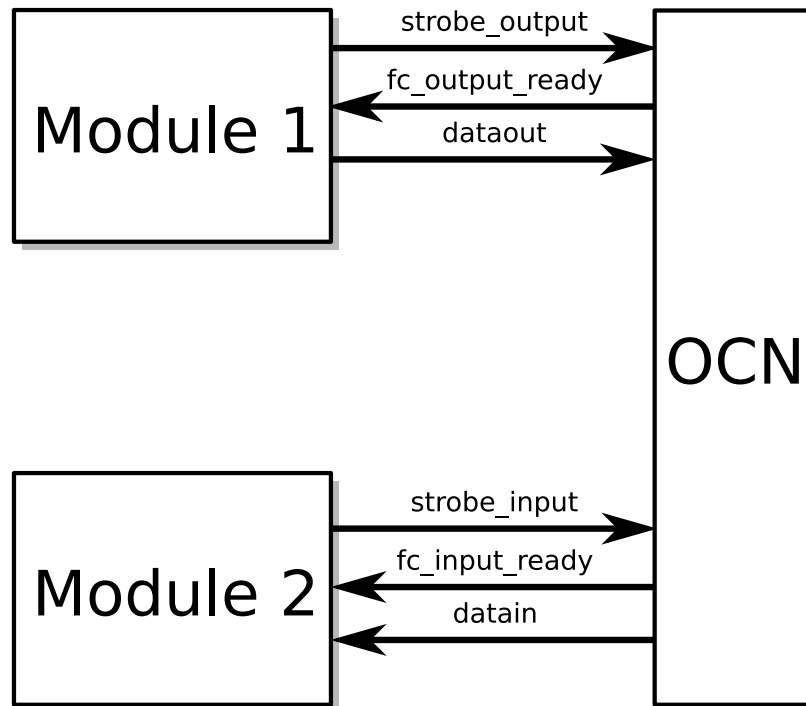


Figure 2: Chain of modules, the output of module 1 is connected to the input of module 2

In this configuration one module will process data and send it to another module. No memory is used for intermediate storage. Assuming that the receiver is always ready to receive data and that both the sender and receiver is running on the same clock domain this would be trivial. However, to allow for the case where different clock domains are used for the sender and receiver, the flow control signals `fc_input_ready` and `fc_output_ready` must be used. (The OCN itself will contain asynchronous FIFOs to handle the actual clock domain crossing, so modules connected to the OCN do not need to care about this issue as long as they honour the flow control signals.)



Figure 3: Signal waveform example for a simple chain of modules using the same clock

Figure 2 shows an example of how such a configuration can look like. **Figure 3** shows an example of how the protocol works in case the same clock domain is used for both the sender and receiver. Note

that the receiver will not receive data immediately, as the OCN will have at least one pipeline register in it to simplify timing closure.

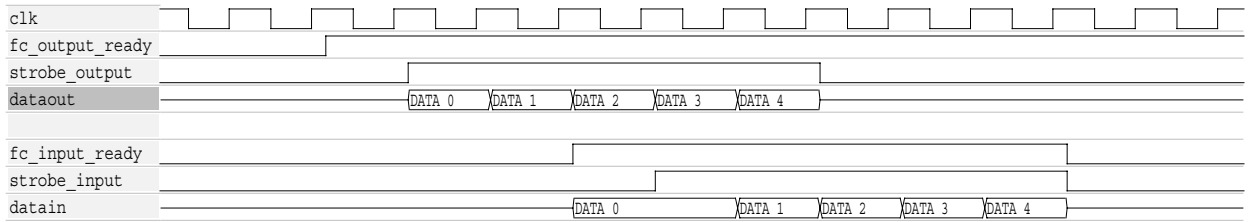


Figure 4: Signal waveform example for a simple chain of modules using the same clock

Figure 4 shows another example where the sender and receiver are using the same clock. In this case, the strobe_output and strobe_input signal are delayed one clock cycle to allow for a registered output. This will simplify timing closure.

4.2 Chain of modules, different clock domains

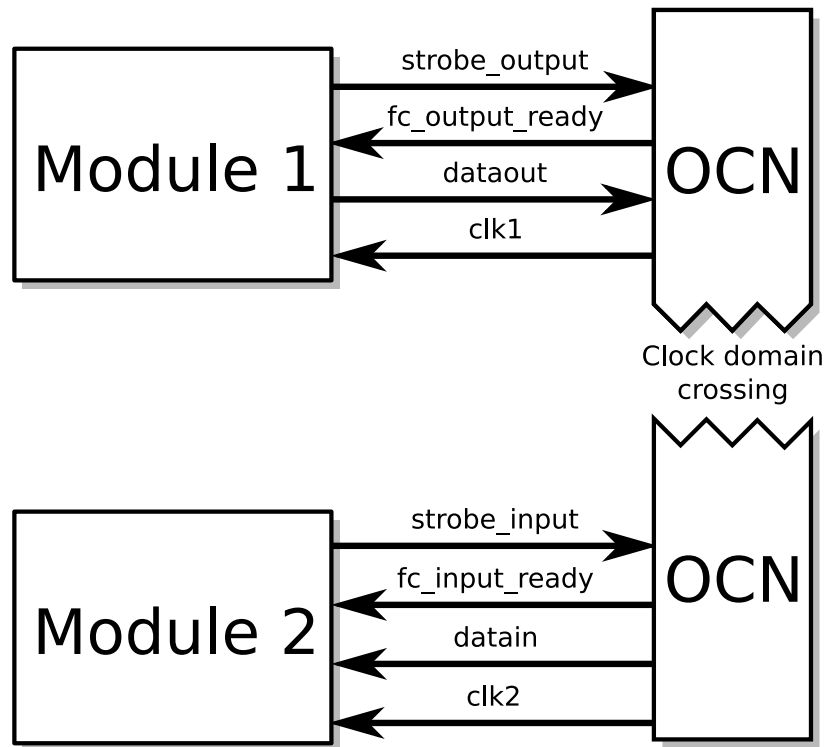


Figure 5: Chain of modules, the modules are not using the same clock domain

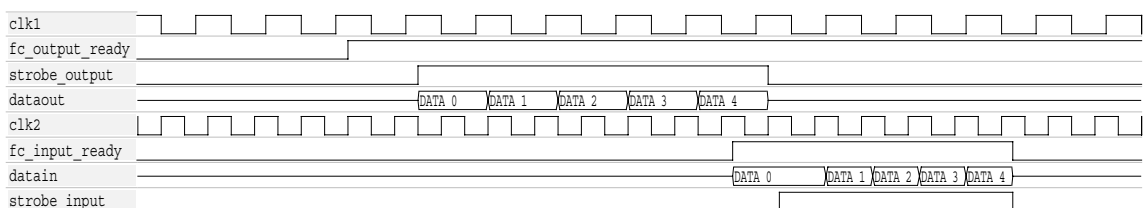


Figure 6: Signal waveform example for a simple chain of modules using two clocks

Similarly to the previous section, **Figure 5** shows how modules can be connected to each other via the OCN. In this configuration the modules are not connected to the same clock domain however. This means that the communication latency will be somewhat increased, as it is necessary to safely cross clock domains in the OCN. An example of this is shown in **Figure 6**. The exact size of the asynchronous FIFO employed by the OCN will depend on the specific clock frequencies selected, but when sending from a slower clock domain to a faster clock domain the length of the FIFO will be long enough to allow the sender to send data without `fc_output_ready` going low (assuming the receiver is ready to accept all data without deasserting `strobe_input`).

4.3 Module to memory

In this configuration one module is connected via the OCN to a memory. While the OCN allows for address information to be sent to the memory along with data, the default is to use the configurable address generators in the memory block. In the most trivial case, the address generator is configured by the platform controller before the OCN is (re-)configured to connect the module to the memory. In this case the module does not even have to be aware that it is connected to a memory, it will just function as either a sender of data or receiver of data as described in section 4.1 and 3.2.

4.4 Module to memory, both read and write

In a more complex scenario, such as for example if an FFT module is connected to the OCN, it may be necessary to reconfigure the address generators during operation. In this case, the `config_enable_out` signal can be used by the module to send configuration commands to the address generators in the memory. However, a problem occurs when switching addressing mode, as the memory module may already have filled the OCN pipeline with additional data. This is handled by the `config_status_in` signal which can be set to either 1 or 0 by the configuration command. This allows the module to differentiate between old data and data read by the new address generator configuration. Such a configuration is shown in **Figure 7** and an example of how it is configured is shown in **Figure 8**.

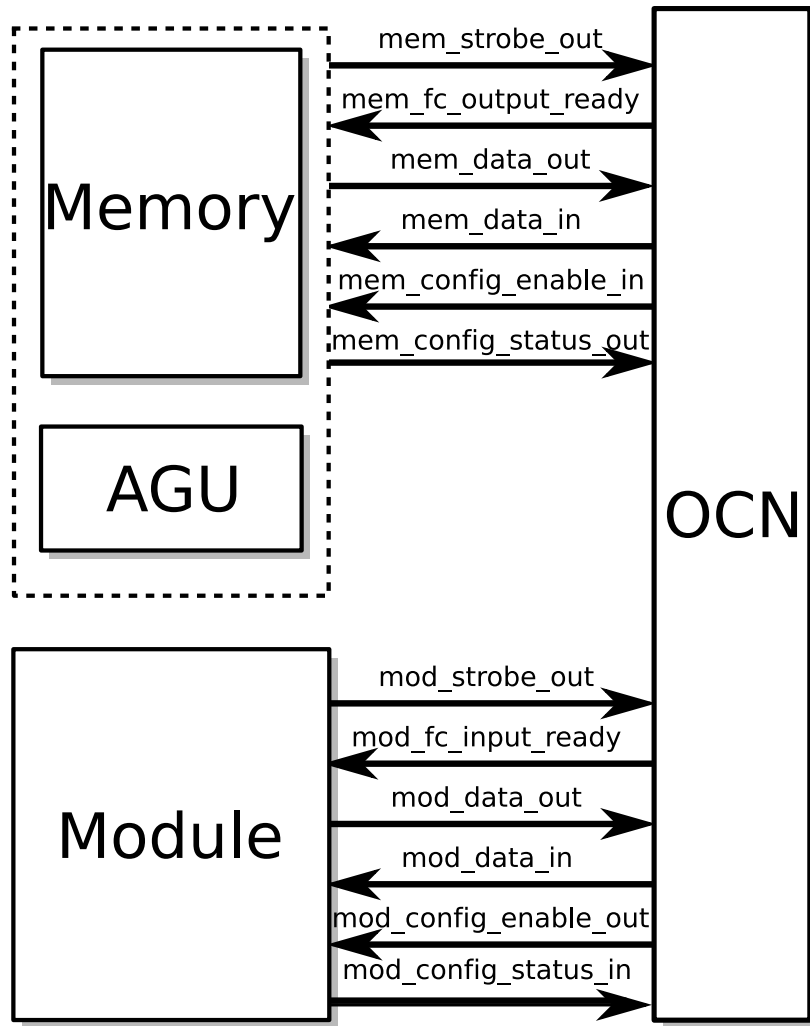


Figure 7: Signal waveform of module connected to memory via the OCN

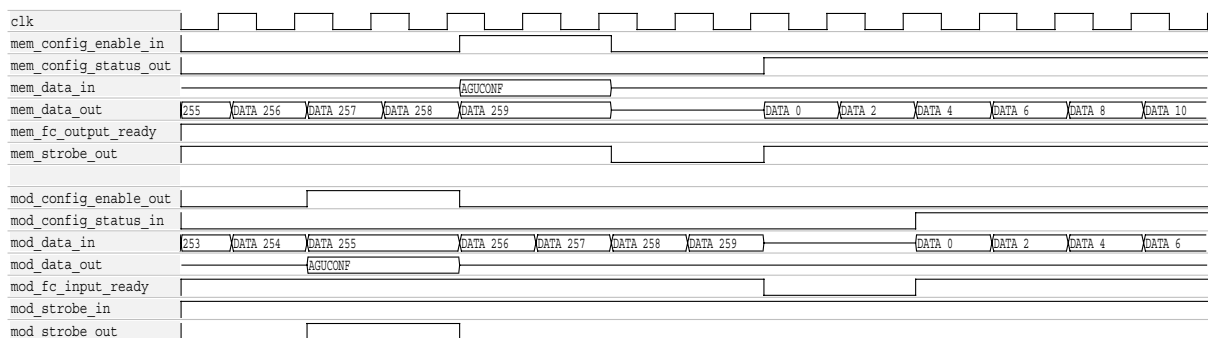


Figure 8: Signal waveform of module connected to memory via the OCN while changing configuration mode of the AGU.

5 Configuration commands for memories

There are a number of configuration commands that can be used to configure the address generators in the memory blocks. The list below is intended to handle the most common addressing modes:

- Incremental
- Incremental, variable stepsize
- Ring buffer
- Bit reversed with offset

There are two AGUs for each memory, one AGU for writing and one AGU for reading. These can be configured individually with different addressing modes.

5.1 Configuration method, 8 bit wide links

In many cases the AGU configuration can be set by the platform controller. However, sometimes the configuration must be set by a module by using a configuration access where `config_enable_out` is asserted as shown in **Figure 8**. The first access determines the AGU mode, and which AGU to configure as explained in **Table 3**. It also determines the value of the `config_status` register to allow modules to determine whether the configuration has been changed already. The following 2-8 bytes determines the exact configuration for the configuration mode. (The number of bytes being specific to that particular addressing mode.)

Bit number(s)	Explanation
3:0	Addressing MODE
4	0 for read AGU, 1 for write AGU
5	Value of <code>config_status</code>
7:6	Reserved

Table 3: AGU configuration for incremental addressing mode

As an example, to set the configuration mode of the write AGU to write incrementally to address 513 the following commands would be sent:

```
0x10 // write agu specified, addressing mode 0 (incremental)
0x01 // Addr, LSB
0x21 // Addr, MSB
```

5.2 Configuration method, 32 bit wide links

This functions the same way as for 8 bit wide links with the difference that the configuration mode is determined by bit [31:24] in the first word and [23:16] corresponds to configuration word 1, [15:8] to configuration word 2, etc. The example provided in the previous section of configuring the write AGU to write incrementally to address 513 can be sent as follows:

```
0x100121xx // (xx is don't care)
```

5.3 Incremental

This is the easiest addressing mode to handle. The AGU in the memory will just increase the address pointer by 1 with every access. **Table 4** explains how it is configured.

AGU Configuration word	Explanation
0	Set MODE to 0 for incremental mode
1	Initial pointer value (lsb)
2	Initial pointer value (msb)

Table 4: AGU configuration for incremental addressing mode

5.4 Incremental with variable stepsize

Same as the previous section, but with variable stepsize.

AGU Configuration word	Explanation
0	Mode reg, set MODE to 1 for incremental mode with variable stepsize
1	Initial pointer value (lsb)
2	Initial pointer value (msb)
3	Stepsize (lsb)
4	Stepsize (msb)

Table 5: AGU configuration for incremental addressing mode with variable stepsize

5.5 Ring buffer

This addressing mode implements a ring buffer with variable stepsize. If stepsize is negative (in two's complement) limit is the lower boundary, if stepsize is positive, Limit is the upper boundary. The address calculation for each step is performed as the following pseudo code:

```

next_addr = curr_addr + stepsize;
if(stepsize < 0){
    if(next_addr < limit){
        next_addr += buffer_size;
    }
}
else{
    If(next_addr > limit){
        next_addr -= buffer_size;
    }
}

```

(Note that the behaviour of this addressing mode is undefined if the stepsize is larger than half the buffer size.) **Table 6** shows how this mode is configured.

AGU Configuration word	Explanation
0	Mode reg, set MODE to 2 for ring buffer mode
1	Initial pointer value (lsb)
2	Initial pointer value (msb)
3	Stepsize (lsb)
4	Stepsize (msb)
5	Buffer size (lsb)
6	Buffer size (msb)
7	Limit (lsb)
8	Limit (msb)

Table 6: AGU configuration for circular buffer

5.6 Bit reversed addressing with offset

The bit reversed addressing mode generates an address through the following expression:

```
addr = offset + bitrev(baddr)
next_baddr = baddr + stepsize
```

where the `bitrev` function reverses all bits in `baddr`. **Table 7** shows how it is configured.

AGU Configuration word	Explanation
0	Mode reg, set MODE to 3 for bit reversed mode
1	Initial <code>baddr</code> value (lsb)
2	Initial <code>baddr</code> value (msb)
3	Stepsize value (lsb)
4	Stepsize value(msb)
5	Offset value (lsb)
6	Offset value (msb)

Table 7: AGU configuration for incremental addressing mode with variable stepsize

6 OCN Implementation issues

This document has referred to asynchronous FIFOs for clock domain crossings at several locations. These will be large enough to allow for a sender in a slow clock domain to send data to a receiver in a faster clock domain without worrying about flow control as long as the receiver is always ready to receive data.

To reduce the latency of memory to module communication it is also intended that the OCN can drive the memory clock so that the same clock can be used for both the module and the memory, therefore avoiding the use of a high latency asynchronous FIFO.

7 List of Abbreviations

OCN	On Chip Network
FC	Flow Control
MAC	Media Access Control
FEC	Forward Error Control
DFE	Digital Front End
DSP	Digital Signal Processor
FFT	Fast Fourier Transform
I/Q	In-phase/Quadrature
A/D	Analog/Digital
FIFO	First-In First-Out
AGU	Address Generator Unit
MSB	Most Significant Bit
LSB	Least Significant Bit

8 References

- [1] ARM, *AMBA Overview*, 2009, <http://www.arm.com/products/solutions/AMBAHomePage.html>
- [2] IBM, *CoreConnect bus architecture*, 2009, <http://www-03.ibm.com/technology/power/licensing/coreconnect/index.html>
- [3] Anders Nilsson, *Design of programmable multi-standard baseband processors*, Linköping Studies in Science and Technology, Dissertations, No. 1084, Linköping, Sweden, June 2007
- [4] Linköping University, *Senior Instruction Set Manual*, 2008, MULTI-BASE SVN (WP03/senior_instruction_set_manual.pdf)