SimpCon – a Simple SoC Interconnect Draft

Martin Schoeberl
martin@jopdesign.com

November 28, 2005

This document proposes a simple interconnection standard for system-on-chip (SoC) components. It is intended to provide pipelined access to devices such on-chip peripherals and on-chip memory controller with minimum hardware resources.

1 Introduction

The intention of the following SoC interconnect standard is to be simple and efficient with respect to implementation resources and transaction latency.

SimpCon is a fully synchronous standard for on-chip interconnections. It is a point-to-point connection between a master and a slave. The master starts either a read or write transaction. Master commands are single cycle to free the master to continue on internal operations during an outstanding transaction. The slave has to register the address when needed for more than one cycle. The slave also registers the data on a read and provides it to the master for more than a single cycle. This property allows the master to delay the actual read if it is busy with internal operations.

The slave signals the end of the transaction through a novel *ready counter* to provide an early notification. This early notification simplifies the integration of peripherals into pipelined masters.

Slaves can also provide several levels of pipelining. This feature is announced by two static output ports (one for read and one write pipeline levels).

Off-chip connections (e.g. main memory) are device specific and need a slave to perform the translation. Peripheral interrupts are not covered by this specification.

1.1 Feature

- Master/slave point-to-point connection

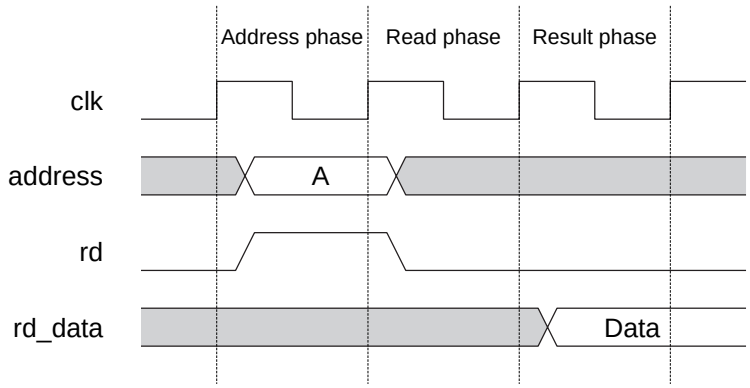

Figure 1: Basic read transaction

- Synchronous operation
- Read and write transactions
- Early pipeline release for the master
- Pipelined transactions
- Open-source specification
- Low implementation overheads

1.2 Basic Read Transaction

Figure 1 shows a basic read transaction for a slave with one cycle latency. The acknowledge signals are omitted from the figure. In the first cycle, the address phase, the `rd` signals the slave to start the read transaction. The address is registered by the slave. During the following cycle, the read phase, the slave performs the read and registers the data. Due to the register in the slave the data is available in the third cycle, the result phase. To simplify the master, the read data stays valid till the next read request response.

1.3 Basic Write Transaction

A write transaction consists of a single cycle address/command phase started by assertion of `wr` where the address and the write data are valid. `address` and `wr_data` are usually registered by the slave. The end of the write cycle is signalled to the master by the slave with `rdy_cnt`. See section 3 and an example in Figure 3.

Signal	Width	Direction	Required	Description
address	1–32	Master	No	Address lines from the master to the slave port
wr_data	32	Master	No	Data lines from the master to the slave port
rd	1	Master	No	Start of a read transaction
wr	1	Master	No	Start of a write transaction
rd_data	32	Slave	No	Data lines from the slave to the master port
rdy_cnt	2	Slave	Yes	Transaction end signalling
rd_pipeline_level	2	Slave	No	Maximum pipeline level for read transactions
wr_pipeline_level	2	Slave	No	Maximum pipeline level for write transactions

Table 1: SimpCon port signals

2 SimpCon Signals

This sections defines the signals used by the SimpCon connection. Some of the signals are optional and may not be present on a peripheral device.

All signals are a single direction point-to-point connection between a master and a slave. The signal details are described by the device that drives the signal. Table 1 lists the signals that define the SimpCon interface. The column Direction indicates wether the signal is driven by the master or the slave.

2.1 Master Signal Details

This section describes the signals that are driven by the master to initiate a transaction.

2.1.1 address

Master addresses represent word addresses as offsets in the slaves address range. address is valid a single cycle either with rd for a read transaction or with wr and wr_data for a write transaction.

The number of bits for address depend on the slaves address range. For a single port slave address can be omitted.

2.1.2 wr_data

The wr_data signals carry the data for a write transaction. It is valid for a single cycle together with address and wr. The signal is typically 32 bits wide. Slaves can ignore upper bits when the slave port is less than 32 bits.

2.1.3 rd

The `rd` signal is asserted a single clock cycle to start a read transaction. `address` has to be valid in the same cycle.

2.1.4 wr

The `wr` signal is asserted a single clock cycle to start a write transaction. `address` and `wr_data` have to be valid in the same cycle.

2.1.5 sel_byte

The `sel_byte` signal is reserved for future versions of the SimpCon specification to add individual byte enables.

2.2 Slave Signal Details

This section describes the signals that are driven by the slave as a response to transaction initiated by the master.

2.2.1 rd_data

The `wr_data` signals carry the result for a read transaction. The data is valid when `rdy_cnt` reaches 0 and stays valid till a new read result is available. The signal is typically 32 bits wide. Slaves that provide less than 32 bits should pad the upper bits with 0.

2.2.2 rdy_cnt

The `rdy_cnt` signal provides the number of cycles till the pending transaction will finish. A 0 means that either read data is available or a write transaction has been finished. Values of 1 and 2 mean the the transaction will finish in at least 1 or 2 cycles. The maximum value is 3 and means the the transaction will finish in 3 or *more* cycles. Note that not all values have to be used in a transaction. Each monotonic sequence of `rdy_cnt` values is legal.

2.2.3 rd_pipeline_level

The static `rd_pipeline_level` provides the master with the read pipeline level of the slave. The signal has to be constant to enable the synthesizer to optimize the pipeline level dependent state machine in the master.

2.2.4 wr_pipeline_level

The static `wr_pipeline_level` provides the master with the write pipeline level of the slave. The signal has to be constant to enable the synthesizer to optimize the pipeline level dependent state machine in the master.

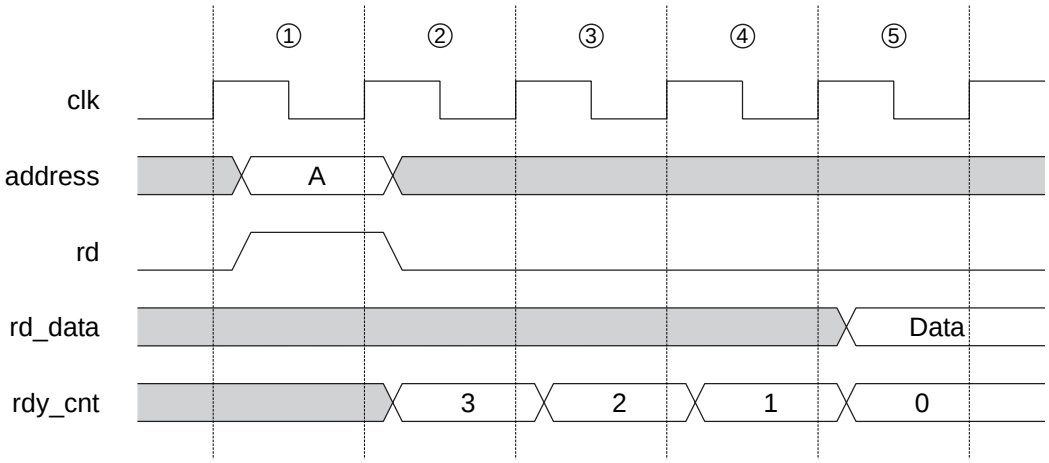

Figure 2: Read transaction with wait states

3 Slave Acknowledge

Flow control between the slave and the master is usually done by a single signal in the form of *wait* or *acknowledge*. The `ack` signal, e.g. in the Wishbone specification, is set when the data is available or the write operation has finished. However, for a pipelined master it can be of interest to know it *earlier* when a transaction will finish.

For a lot of slaves, e.g. a SRAM interface with fixed wait states, this information is available inside the slave. In the SimpCon interface this information is communicated to the master through the two bit signal `rdy_cnt`. `rdy_cnt` signals the number of cycles till the read data will be available or the write transaction will be finished. Value 0 is equivalent to an *ack* signal and 1, 2, and 3 are equivalent to a wait request with the distinction that the master knows how long the wait request will last.

To avoid too many signals at the interconnect `rdy_cnt` has a width of two bits. Therefore, the maximum value of 3 has the special meaning that the transaction will finish in 3 or *more* cycles. As a result the master can only use the values 0, 1, and 2 to release actions in it's pipeline.

Idle slaves will keep the former value of 0 for `rdy_cnt`. Slaves, that don't know in advance how many wait states are need for the transaction can produce sequences that omit any of the numbers 3, 2, and 1. The master has to handle this situations.

Figure 2 shows an example of a slave that needs three cycles for the read to be processed. In cycle 1 the read command and the address are set by the master. The slave registers the address and sets `rdy_cnt` to 3 in cycle 2. The read takes three cycles (2–4) during which `rdy_cnt` is decremented. In cycle 4 the data is available inside the slave and gets registered. It is available in cycle 5 for the master and `rdy_cnt` is finally 0. Both, the `rd_data` and `rdy_cnt` will keep their value till a new transaction is requested.

Figure 3 shows an example of a slave that needs three cycles for the write to be processed. The address, the data to be written and the write command are valid during cycle 1. The slave registers the address and write data during cycle 1 and performs the write operation

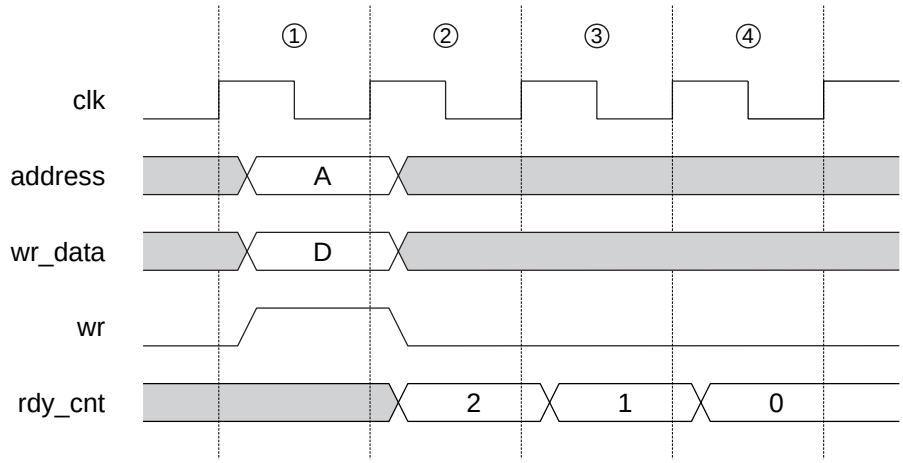

Figure 3: Write transaction with wait states

during cycles 2–4. The `rdy_cnt` is decremented and a non-pipelined slave can accept a new command after cycle 4.

4 Pipelining

Figure 4 shows a read transaction for a slave with four cycles latency. Without any pipelining the next read transaction will start in cycle 7 after the data from the former read transaction is read by the master. The three bottom lines show when new read transactions will be started for different pipeline levels. With pipeline level 1 a new transaction can start in the same cycle when the former read data is available (in this example in cycle 6). Higher levels mean that the next read will start earlier as shown for level 2 and 3.

Implementation of level 1 in the slave is trivial (just two more transitions in the state machine). It is recommended to provide level 1 at least for read transactions. Level 2 is a little bit more complex but usually no additional address or data registers are needed.

To implement level 3 pipelining in the slave at least an additional address register is needed. However, to use level 3 the master has to issue the request in the same cycle as `rdy_cnt` goes to 2. That means this transition is combinatorial. We see in Figure 4 that `rdy_cnt` value of 3 means three or more cycles till the data is available and can therefore not be used to trigger a new transaction.

5 Multiple Master

SimpCon defines no signals for the communication between a master and an arbiter. However, it is possible to build a multi master system with SimpCon. The SimpCon interface can be used as interconnect between the masters and the arbiter and the arbiter and the slaves. In this case the arbiter acts as slave for the master and as master for the peripheral devices.

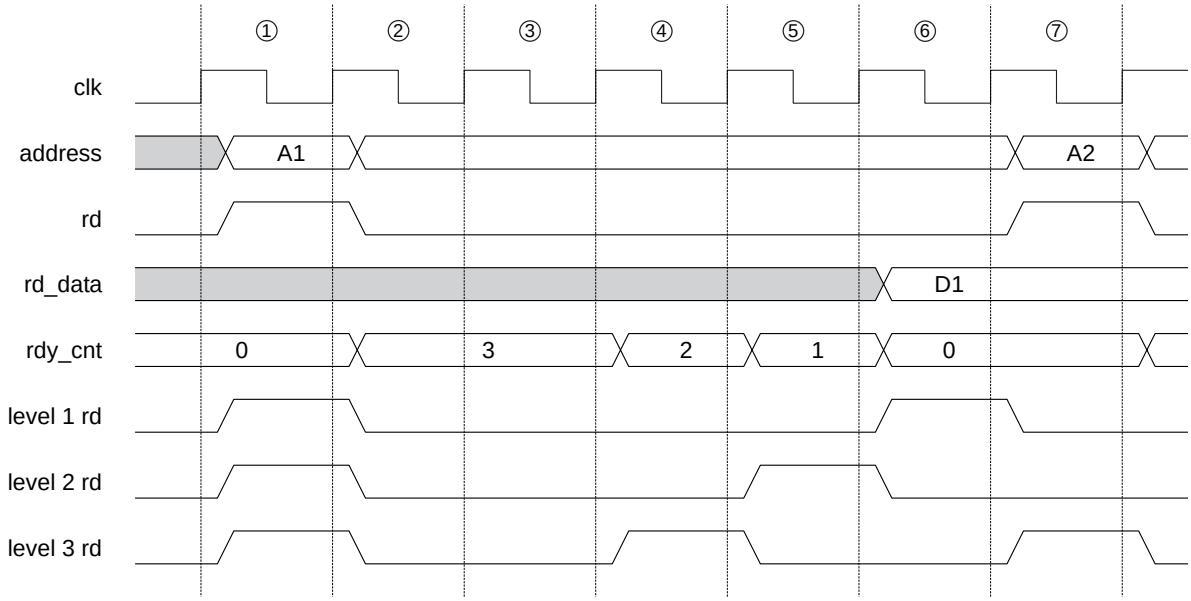

Figure 4: Different pipeline levels for a read transaction

The missing arbitration protocol in SimpCon results in the need to queue $n-1$ requests in an arbiter for n masters. However, for this additional HW we get zero overheads for the bus request. The master, which gets the bus will will start the slave transaction in the same cycle.

TODO: add a timing diagram to explain this concept.

6 Examples

This section provides some examples for the application of the SimpCon definition.

6.1 IO Port

TODO: Show how simple an IO port can be with SimpCon. We need no addresses and can tie `bsy_cnt` to 0. We only need the `rd` or `wr` signal to enable the port.

6.2 SRAM interface

The following example is taken from an implementation of SimpCon for a Java processor. The processor is clocked with 100MHz and the main memory consists of 15ns static RAMs. Therefore the minimum access time for the RAM is two cycles. The slack time of 5ns forces us to use output registers for the RAM address and write data and input registers for the read data in the IO cells of the FPGA. These registers fit nice with the intention of SimpCon to use registers inside the slave.

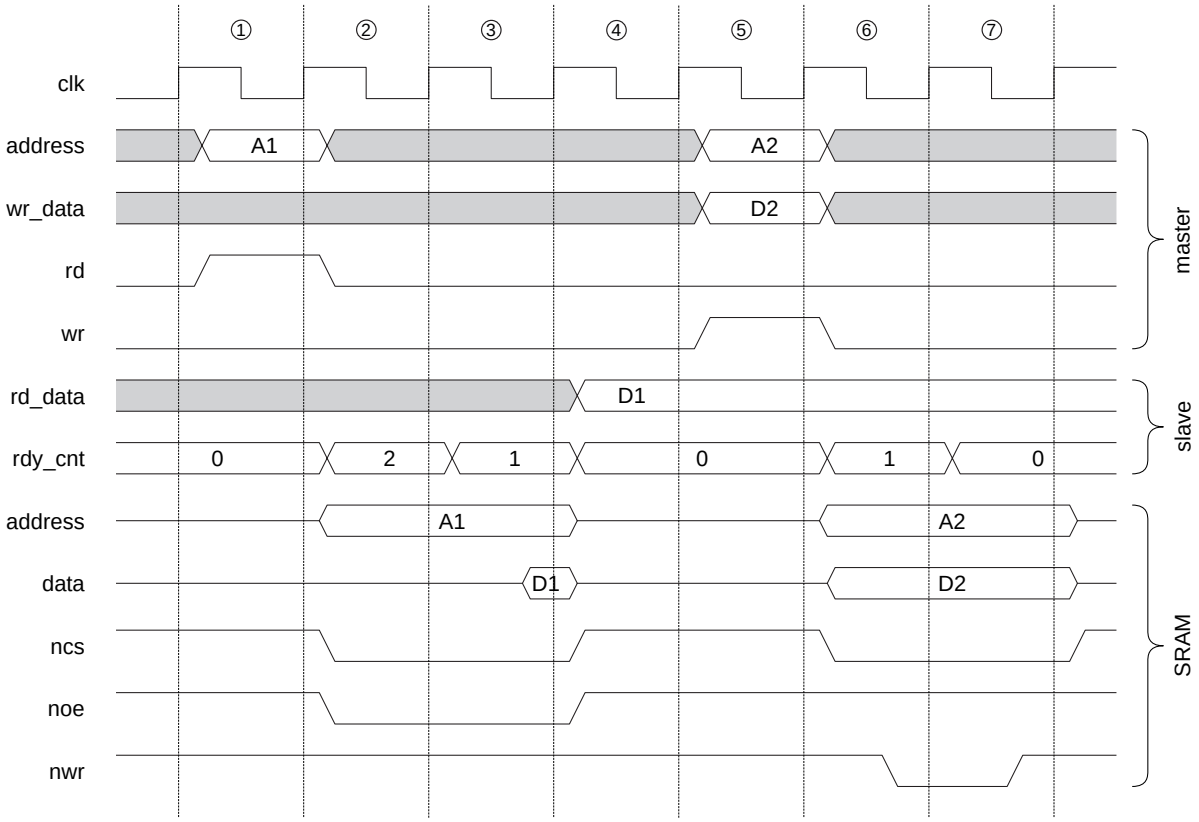

Figure 5: Static RAM interface without pipelining

Figure 5 shows the interface for a non-pipelined read access followed by a write access. Four signals are driven by the master and two signal by the slave. The lower half of the figure shows the signals at the FPGA pins where the RAM is connected.

In cycle 1 the read transaction is started by the master and the slave registers the address. The slave also sets the registered control signals `ncs` and `noe` during cycle1. Due to the IO cell registers, the address and control signals are valid at the FPGA pins very early in cycle 2. At the end of cycle 3 (15ns after `address`, `ncs` and `noe` are stable) the data from the RAM is available and can be sampled with the rising edge for cycle 4.

The master reads the data in cycle 4 and starts a write transaction in cycle 5. Address and data are again registered from the slave and are available for the RAM at the beginning of cycle 6. To perform a write in two cycles the nwr signal is registered by a negative triggered flip-flop.

In figure 6 we see a pipelined read from the RAM with pipeline level 2. With this pipeline level and the two cycles read access time of the RAM we get the maximum bandwidth possible.

We can see the start of the second read transaction in cycle 3 during the read of the first data from the RAM. The new address is registered in the same cycle and available for the RAM in the following cycle 4. Although we have a pipeline level of 2 we need no additional address or data register. The read data is available for two cycles (`rdy_cnt` 2 or 1 for the next read)

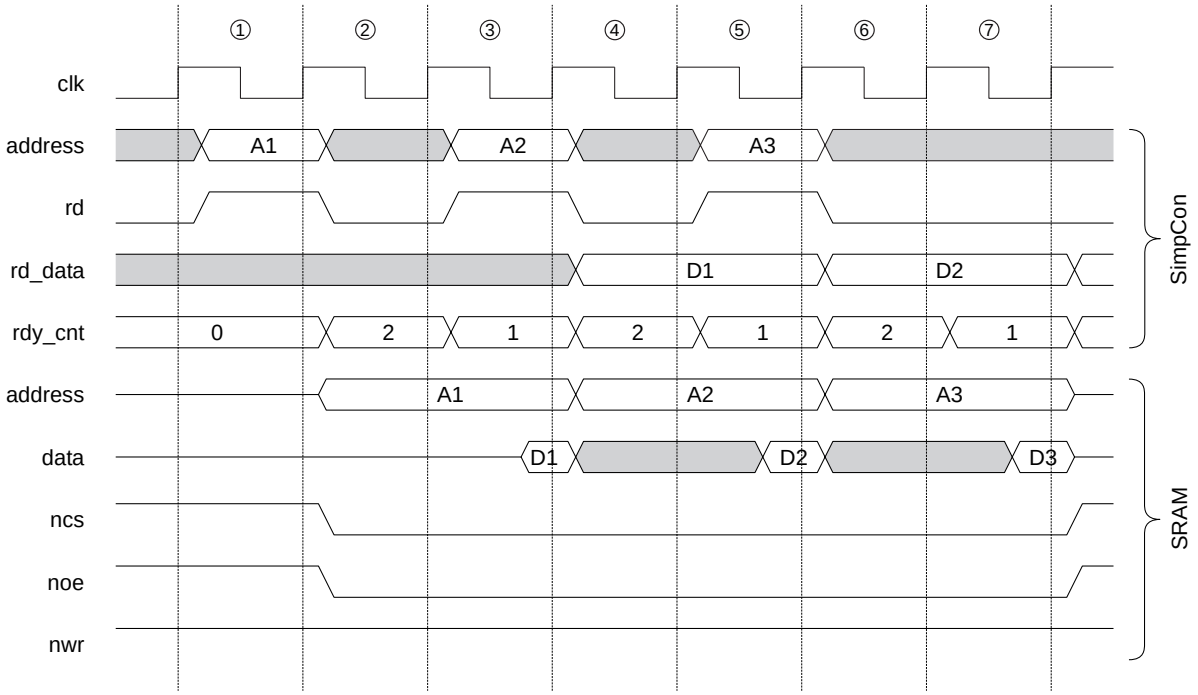

Figure 6: Pipelined read from a static RAM

and the master is free to select one of the two cycles to read the data.

6.3 Master Multiplexing

To add several slaves to a single master the `rd_data` and `bsy_cnt` have to be multiplexed. Due to the fact that all `rd_data` signals are registered by the slaves a single pipeline stage will be enough for a large multiplexer. The selection of the multiplexer is also known at the transaction start but needed at most in the next cycle. Therefore it can be registered to further speed up the multiplexer.

TODO: add a schematic for the master `rd_data` multiplexer.

7 Status

- First timing diagrams drawn
- SimpCon SRAM interface for JOP on Cyclone and Spartan-3 is available
- Project at opencores.org accepted

Next steps:

- Continue this document

- Provide more SimpCon examples (e.g. a UART)
- Change JOPs IO interface to SimpCon
- Provide Wishbone bridges

to clarify:

- Use transaction or transfer in this document?
- Use address phase or better command cycle?