SimpCon – a Simple and Efficient SoC Interconnect

Martin Schoeberl
Institute of Computer Engineering
Vienna University of Technology, Austria
mschoebe@mail.tuwien.ac.at

Abstract

To build a system-on-chip (SoC) a common interface standard is necessary to connect ready-to-use components (IPs) from different vendors. Today several SoC interconnect standards, such as AMBA, Wishbone, OPB, and Avalon, are in use. We show in this paper that those standards have a common drawback for on-chip interconnections: They are built on the model of a common backplane bus that does not fit very well for on-chip interconnections.

We provide a new, simple on-chip interconnect specification for the well accepted master/slave model. It is intended to provide pipelined access to devices such as on-chip peripherals and on-chip memory controller with minimum hardware resources.

1 Introduction

We propose a simple, efficient on-chip interconnection standard named SimpCon. SimpCon is a fully synchronous standard for on-chip interconnections. It is defined as a point-to-point connection between a master and a slave. The signals are defined for a scalable interconnection of several slaves to a single master. No combinatorial path from the master, through the decoding logic, to the slave and back to the master limits the scalability.

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 result of a read request 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*. This counter provides an early master notification. This early notification simplifies the integration of slaves into a system with pipelined masters. Slaves can support different levels of pipelining.

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

The main contributions of SimpCon are:

- A ready counter to release the master pipeline early
- Single cycle commands to support pre-fetch read transactions
- Correct placement of registers in the interfaces to asynchronous peripherals
- Overlapping transactions due to the ready counter

This paper does not deal with network-on-chip (NoC) structures. However, the proposed SimpCon master/slave interconnection can serve as a *socket* (as described in the NoC survey [5]) for a NoC.

1.1 Related Work

Several point-to-point and bus standards have been proposed over the last years. The following section gives a brief overview of common SoC interconnection standards.

The Advanced Microcontroller Bus Architecture (AMBA) [3] is the interconnection definition from ARM. The specification defines three different busses: Advanced High-performance Bus (AHB), Advanced System Bus (ASB), and Advanced Peripheral Bus (APB). The AHB is used to connect on-chip memory, cache, and external memory to the processor. Peripheral devices are connected to the APB. A bridge connects the AHB to the lower bandwidth APB. An AHB bus transfer can be one cycle with burst operation. With the APB a bus transfer requires two cycles and no burst mode is available. Peripheral bus cycles with wait states are added in the version 3 of the APB specification. ASB is the predecessor of AHB and is not recommended for new designs (ASB uses both clock phases for the bus signals – very uncommon for today's synchronous designs). The AMBA 3 AXI (Advanced eXtensible Interface) [4] is the latest extension to AMBA. AXI introduces out-of-order transaction completion with the help of a 4 bit transaction id tag. A ready signal acknowledges the transaction start. The master has to hold the transaction information (e.g. address) till the interconnect signals ready. This enhancement ruins the elegant single cycle address phase from the original AHB specification.

Wishbone [10] is a public domain standard used by several open-source IP cores. The Wishbone interface specification is still in the tradition of microcomputer or backplane busses. However, for a SoC interconnect, which is usually point-to-point[1], this is not the best approach. The master is requested to hold the address and data valid through the whole read or write cycle. This complicates the connection to a master that has the data valid only for one cycle. In this case the address and data have to be registered *before* the Wishbone connect or an expensive (time and resources) multiplexer has to be used. A register results in one additional cycle latency. A better approach would be to register the address and data in the slave. In that case the address decoding in the slave can be performed in the same cycle as the address is registered. A similar issue, with respect to the master, exists for the output data from the slave: As it is only valid for a single cycle the data has to be registered by the master when the master is not reading it immediately. Therefore, the slave should keep the last valid data at its output even when the Wishbone strobe signal (*wb.stb*) is not assigned anymore. Holding the data in the slave is usually *for free* from the hardware complexity – it is *just* a specification issue. In the Wishbone specification there is no way to perform pipelined read or write. However, for blocked memory transfers (e.g. cache load) this is the usual way to achieve good performance.

The Avalon [2] interface specification is provided by Altera for a system-on-a-programmable-chip (SOPC) interconnection. Avalon defines a great range of interconnection devices ranging from a simple asynchronous interface intended for direct static RAM connection up to sophisticated pipeline transfers with variable latencies. This great flexibility provides an easy path to connect a peripheral device to Avalon. How is this flexibility possible? The *Avalon Switch Fabric* translates between all those different interconnection types. The switch fabric is generated by Altera's SOPC Builder tool. However, it seems that this switch fabric is Altera proprietary thus tying this specification to Altera FPGAs.

The On-Chip Peripheral Bus (OPB) [7] is an open standard provided by IBM and used by Xilinx. The OPB specifies a bus for multiple masters and slaves. The implementation of the bus is not directly defined in the specification. A distributed ring, a centralized multiplexer, or a centralized AND/OR network are suggested. Xilinx uses the AND/OR approach and all masters and slaves must drive the data busses to zero when inactive.

Sonics Inc. defined the Open Core Protocol (OCP) [9] as an open, freely available standard. The standard is now handled by the OCP International Partnership (www.ocpip.org).

2 What's Wrong with the Classic Standards?

All SoC interconnection standards, that are widely in use, are still in the tradition of a backplane bus. They force the master to hold the address and control signals till the slave provides the data or acknowledges the write request.

[1]Multiplexers are used instead of busses to connect several slaves and masters.

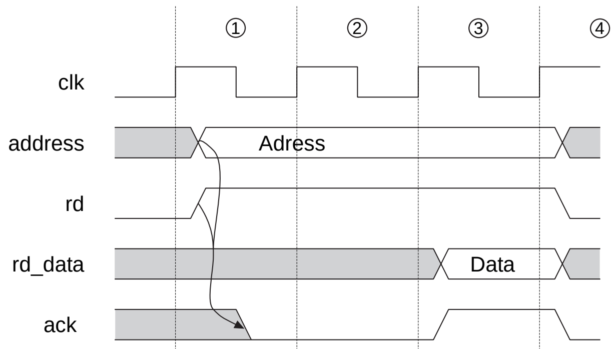

Figure 1. Classic basic read transaction

However, this is not necessary in a clocked, synchronous system. Why should we force the master to hold the signals? Let the master move on after submitting the request in a single cycle. Forcing the address and control valid for the complete request disables any form of pipelined requests.

Figure 1 shows a read transaction with wait states as defined in Wishbone [10], Avalon [2], OPB [7], and OCP [9][2]. The master issues the read request and the address in cycle 1. The slave has to reset the `ack` in the same cycle. When the slave data is available the acknowledge signal is set (`ack` in cycle 3). The master has to read the data and register them within the same clock cycle. The master has to hold the address, write data and control signal active till the acknowledgement from the slave. For pipelined read the `ack` signal can be split into two signals (available in Avalon and OCP): one to accept the request and a second one to signal the available data.

The master is blind about the status of the outstanding transaction until it is finished. It could be possible that the slave informs the master in how many cycles the result will be available. This information can help in building deeper pipelined masters.

Only the AMBA AHB [3] defines a different protocol. A single cycle address phase followed by a variable length data phase. The slave acknowledgement (HREADY) is only necessary in the data phase avoiding the combinatorial path from address/command to the acknowledgement. Overlapping address and data phase is allowed and recommended for high performance. Compared to SimpCon, AMBA AHB allows for single stage pipelining, whereas SimpCon makes multi-stage pipelining possible using the ready counter (`rdy_cnt`). The `rdy_cnt` signal defines the delay between the address and the data on a read, signalled by the slave. Therefor, the pipeline depth of the bus and the slaves is only limited by the bit width of `rdy_cnt`.

Another issue with all interconnection standards is the single cycle availability of read data at the slaves. Why not keep the read data valid as long as there is no new read data available? This feature would allow the master

[2]The signal names are different, but the principle is the same for all mentioned busses.

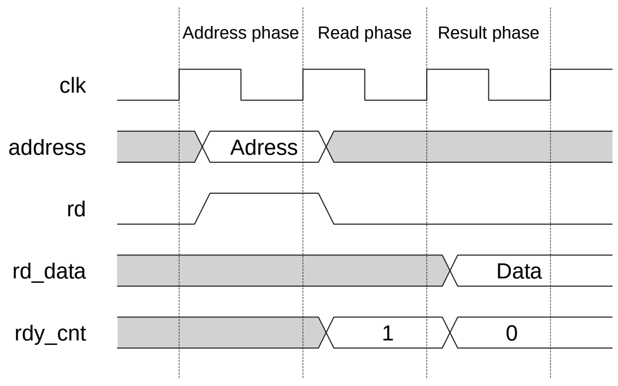

Figure 2. Basic SimpCon read transaction

to be more flexible when to read the data. It would allow to issue a read command and then continue with other instructions – a feature known as data pre-fetching to hide long latencies.

The last argument sounds contradictory to the first argument on providing the transaction data at the master just for a single cycle, but requesting the slave to hold the data for several cycles. However, it is motivated to free up the master, keep it *moving*, and move the data hold (register) burden into the slave. As data processing bottlenecks are usually found in the master devices it sounds natural to move as much work as possible to the slave devices to free up the master.

Avalon, Wishbone and OPB provide a single cycle latency access to slaves due to the possibility to acknowledge a request in the same cycle. However, this feature is a scaling issue for larger systems. There is a combinatorial path from master address/command to address decoding, slave decision on `ack`, slave `ack` multiplexing back to the master and the master decision to hold address/command or read the data and continue. Also the slave output data multiplexer is on a combinatorial path from the master address.

AMBA AHB and SimpCon avoid this scaling issue by requesting the acknowledge in the cycle following the command. In SimpCon and AMBA the select for the read data multiplexer can be registered as the read address is known at least one cycle before the data is available. The later acknowledge results in a minor drawback on SimpCon and AMBA (nothing is for free): It is not possible to perform a single cycle read or write without pipelining. A single, non pipelined transaction takes two cycles without a wait state. However, a single cycle read transaction is only possible for very simple slaves. Most non-trivial slaves (e.g. memory interfaces) will not allow a single cycle access anyway.

3 The SimpCon Proposal

The features of SimpCon are:

- Master/slave point-to-point connection

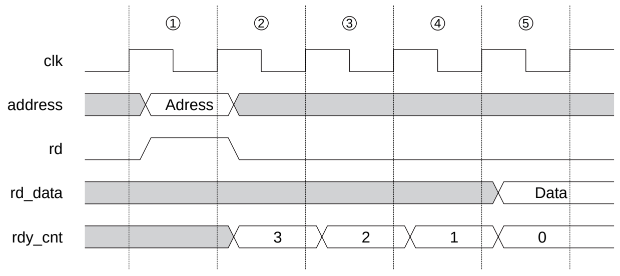

Figure 3. Read transaction with wait states

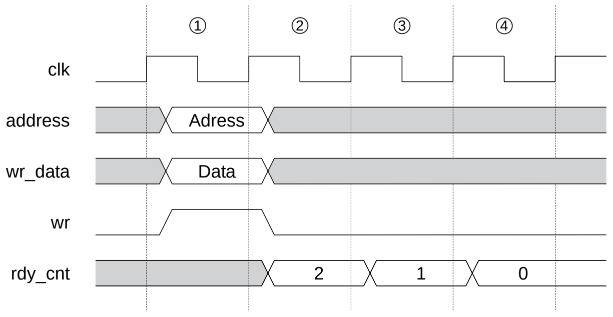

Figure 4. Write transaction with wait states

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

3.1 Basic Read Transaction

Figure 2 shows a basic read transaction with one cycle latency. 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, `rd_data` stays valid till the next read request response. It is therefore possible for a master to issue a pre-fetch command early. Whe the pre-fetched data arrives to early it is still valid when the master actually wants to read it.

3.2 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 signaled to the master by the slave with `rdy_cnt`. See section 3.3 and an example in Figure 4.

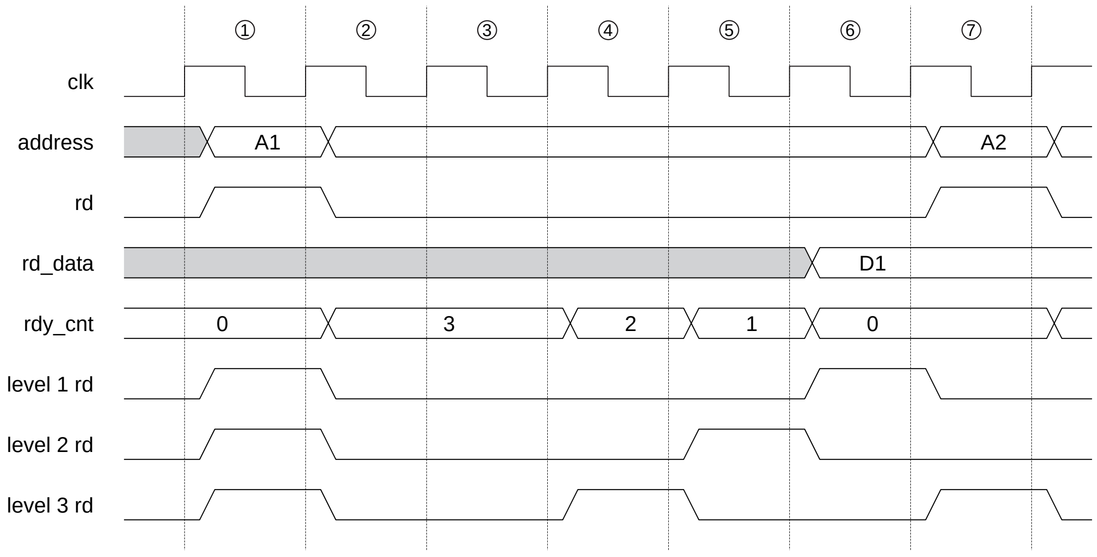

Figure 5. Different pipeline levels for a read transaction

3.3 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 many slaves, e.g. an 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 ready counter (`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. `rdy_cnt` has a width of two bits to avoid too many signals at the interconnect. 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 its pipeline. If necessary an extension for a longer pipeline is straightforward with a larger `rdy_cnt`[3].

Idle slaves will keep the former value of 0 for `rdy_cnt`. Slaves, that don't know in advance how many wait states are needed for the transaction can produce sequences that omit any of the numbers 3, 2, and 1. A simple slave can hold `rdy_cnt` on 3 until the data is available and set it than directly to 0. The master has to handle those situations. Practically this reduces the possibilities of pipelining and therefore the performance of the interconnect. The master will read the data later, which is not an issue as the data stays valid.

[3]The maximum value of the ready counter is relevant for the early restart of a waiting master. A longer latency from the slave e.g., for DDR SDRAM, will map to the maximum value of the counter for the first cycles.

Figure 3 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` gets 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 4 shows an example of a slave that needs two 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 the write data during cycle 1 and performs the write operation during cycles 2–3. The `rdy_cnt` is decremented and a non-pipelined slave can accept a new command after cycle 4.

3.4 Pipelining

Figure 5 shows a read transaction for a slave with four clock 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 (only the `rd` signal is shown, address lines are omitted from the figure) can 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). At pipeline level 2 a new transaction (either read or write) can start when `rdy_cnt` is 1, for pipeline level 2 the next transaction can start at a `rdy_cnt` of 2.

The implementation of level 1 in the slave is trivial (just two more transitions in the state machine). It is recom-

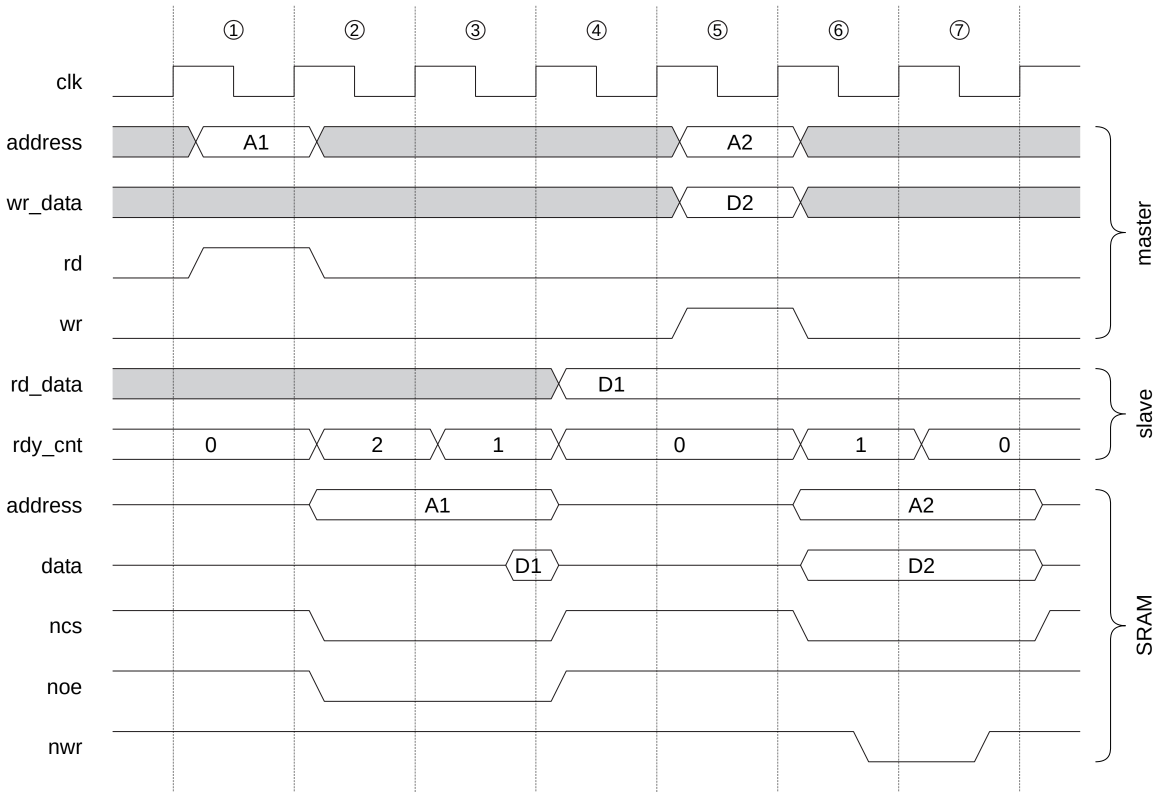

Figure 6. Static RAM interface without pipelining

mended to provide at least level 1 for read transactions. Level 2 is a little bit more complex but usually no additional address or data registers are necessary.

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 5 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. Extension to an even deeper pipeline needs a wider `rdy_cnt`.

3.5 Interconnect

Although the definition of SimpCon is from a single master/slave point-to-point viewpoint, all variations of multiple slave and multiple master devices are possible.

3.5.1 Slave Multiplexing

To add several slaves to a single master `rd_data` and `rdy_cnt` have to be multiplexed. Due to the fact that all `rd_data` signals are already 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.

3.5.2 Master Multiplexing

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. An example of an arbiter for SimpCon, where JOP and a VGA controller are two masters for a shared main memory, can be found in [11]. The same arbiter is also used to build a chip-multiprocessor version of JOP.

The missing arbitration protocol in SimpCon results in the need to queue $n-1$ requests in an arbiter for n masters. However, this additional hardware results in a zero cycle bus grant. The master, which gets the bus granted, starts the slave transaction in the same cycle as the original read/write request.

3.5.3 Network-on-Chip

SimpCon can be used as an interface, called socket, to a NoC based interconnect. In [15] we implemented a time-triggered NoC as a ring, similar to the CELL interconnect [8]. In contrast to the CELL architecture we clocked the network higher (at 235 MHz) than the individual nodes (at 100 MHz). SimpCon was used as the interface between the nodes and the NoC.

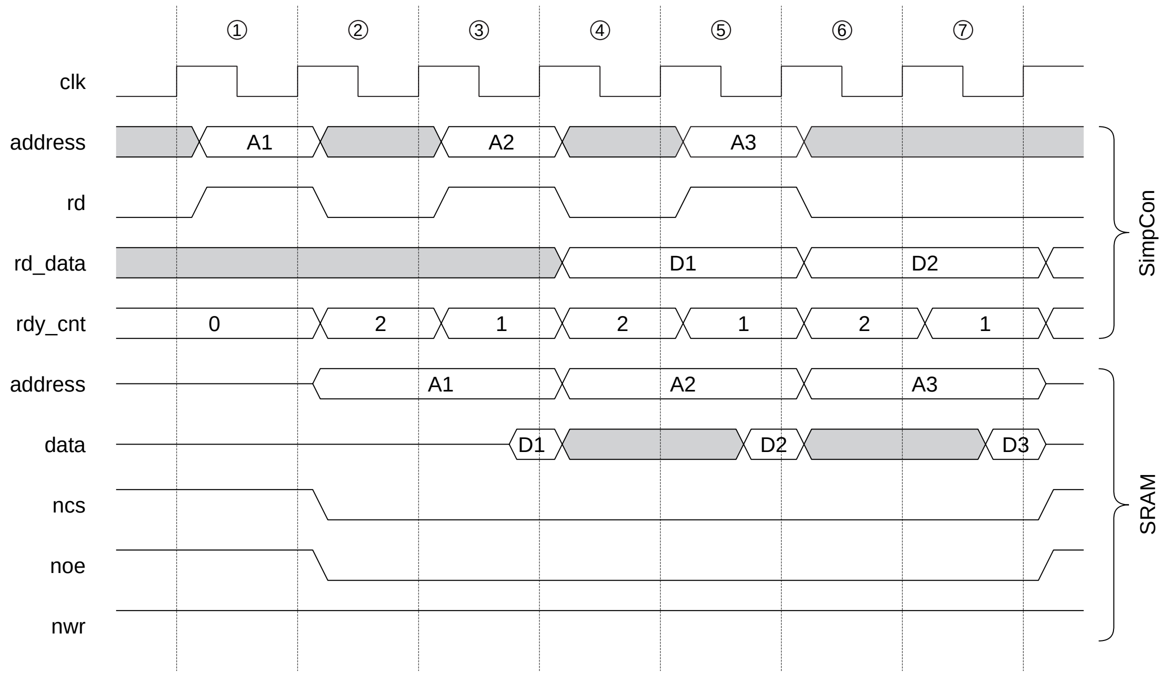

Figure 7. Pipelined read from a static RAM

4 SRAM Slave Example

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

Figure 6 shows the memory interface for a non-pipelined read access followed by a write access. Four signals are driven by the master and two signals 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 placement of the registers in the IO cells, the address and control signals are valid at the FPGA pins very early in cycle 2. At the end of cycle 3 (15 ns 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 setup time for the read register is short as the register can be placed in the IO cell. The master reads the data in cycle 4 and starts a write transaction in cycle 5. Address and data are again registered by 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 7 we show 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 achieve the maximum possible bandwidth. 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) and the master is free to select one of the two cycles to read the data.

It has to be noted that pipelining with one read per cycle is possible with SimpCon. We just showed a 2 cycle slave in this example. For a SDRAM memory interface the ready counter will stay either at 2 or 1 during the single cycle reads (depending on the slave pipeline level). It will go down to 0 only for the last data word to read.

5 Evaluation

We compare the SimpCon interface with the AMBA and the Avalon interface as two examples of common interconnection standards. As evaluation example we interface an external asynchronous SRAM with a tight timing. The system is clocked with 100 MHz and the access time for the SRAM is 15 ns. Therefore, there are 5 ns available for on-chip register to SRAM input and SRAM output to on-chip register delays. As SoC we use an actual low-cost FPGA (Cyclone EP1C6 [1] and a Cyclone II).

The master is a Java processor (JOP [13, 14]). The processor is configured with 4 KB instruction cache and 512 Byte on-chip stack cache. We run a complete applica-

Performance	Memory	Interconnect
16,633	32 bit SRAM	SimpCon
14,259	32 bit SRAM	AMBA
14,015	32 bit SRAM	Avalon/PTF
13,920	32 bit SRAM	Avalon/VHDL
15,762	32 bit on-chip	Avalon
14,760	16 bit SRAM	SimpCon
11,322	16 bit SRAM	Avalon
7,288	16 bit SDRAM	Avalon

Table 1. JOP performance with different interconnections

tion benchmark on the different systems. The embedded benchmark (*Kfl* as described in [12]) is an industrial control application already in production.

Table 1 shows the performance numbers of this JOP/SRAM interface on the embedded benchmark. It measures iterations/s and therefore higher numbers are better. One iteration is the execution of the main control loop of the *Kfl* application. For a 32 bit SRAM interface we compare SimpCon against AMBA and Avalon. SimpCon outperforms AMBA by 17% and Avalon by 19%[4] on a 32 bit SRAM.

The AMBA experiment uses the SRAM controller provided as part of GRLIB [6] by Gaisler Research. We avoided writing our own AMBA slave to verify that the AMBA implementation on JOP is correct. To provide a fair comparison between the single master solutions with SimpCon and Avalon the AMBA bus was configured without an arbiter. JOP is connected directly to the AMBA memory slave. The difference between the SimpCon and the AMBA performance can be explained by two facts: (1) as with the Avalon interconnect, the master has the information when the slave request is ready at the same cycle when the data is available (compared to the `rdy_cnt` feature); (2) the SRAM controller is conservative as it asserts `HREADY` one cycle later than the data is available in the read register (`HRDATA`). The second issue can be overcome by a better implementation of the SRAM AMBA slave.

The Avalon experiment considers two versions: an SOPC Builder generated interface (PTF) to the memory and a memory interface written in VHDL. The SOPC Builder interface performs slightly better than the VHDL version that generates the Avalon `waitrequest` signal. It is assumed that the SOPC Builder version uses fixed wait states within the switch fabric.

We also implemented an Avalon interface to the single-cycle on-chip memory. SimpCon is even faster with the 32 bit off-chip SRAM than with the on-chip memory connected via Avalon. Furthermore, we also performed experiments with a 16 bit memory interface to the same SRAM. With this smaller data width the pressure on the interconnection and memory interface is higher. As a result the difference between SimpCon and Avalon gets larger (30%) on the 16 bit SRAM interface. To complete the picture we also measured the performance with an SDRAM memory connected to the Avalon bus. We see that the large latency of an SDRAM is a big performance issue for the Java processor.

[4]The performance is the measurement of the execution time of the whole application, not only the difference between the bus transactions.

6 Conclusion

In this paper we have presented a simple (with respect to the definition and implementation) and efficient SoC interconnect. The novel signal `rdy_cnt` allows an early signalling to the master when read data will be valid. This feature allows the master to restart a stalled pipeline earlier to react for arriving data. Furthermore, this feature also enables pipelined bus transactions with a minimal effort on the master and the slave side.

We have compared SimpCon quantitative with AMBA and Avalon, two common interconnection definitions. The application benchmark shows a performance advantage of SimpCon by 17% over AMBA and 19% over Avalon interfaces to an SRAM.

SimpCon is used as the main interconnect for the Java processor JOP in a single master, multiple salves configuration. At the time of this writing we are implementing a shared memory chip-multiprocessor version of JOP. Within this work we can evaluate SimpCon as a multi-master system for the shared memory. Furthermore, in a research project on time-triggered network-on-chip [15] we use SimpCon as the *socket* to this NoC. Within this application we can evaluate if a larger `rdy_cnt` improves the performance for this heavy pipelined design.

Acknowledgments

The author thanks Kevin Jennings and Tommy Thorn for the interesting discussions about SimpCon, Avalon and on-chip interconnection in general at the usenet newsgroup `comp.arch.fpga`.

References

[1] Altera. Cyclone FPGA Family Data Sheet, ver. 1.2, April 2003.

[2] Altera. Avalon interface specification, April 2005.

[3] ARM. AMBA specification (rev 2.0), May 1999.

[4] ARM. AMBA AXI protocol v1.0 specification, March 2004.

[5] Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of network-on-chip. *ACM Comput. Surv.*, 38(1):1, 2006.

[6] Jiri Gaisler, Edvin Catovic, Marko Isomäki, Kristoffer Carlsson, and Sandi Habinc. GRLIB IP core users manual, version 1.0.14. Available at http://www.gaisler.com/, February 2007.

[7] IBM. On-chip peripheral bus architecture specifications v2.1, April 2001.

[8] Michael Kistler, Michael Perrone, and Fabrizio Petrini. Cell multiprocessor communication network: Built for speed. *Micro, IEEE*, 26:10–25, 2006.

[9] OCP-IP Association. Open core protocol specification 2.1. http://www.ocpip.org/, 2005.

[10] Wade D. Peterson. WISHBONE system-on-chip (SoC) interconnection architecture for portable IP cores, revision: B.3. Available at http://www.opencores.org, September 2002.

[11] Christof Pitter and Martin Schoeberl. Time predictable CPU and DMA shared memory access. In *International Conference on Field-Programmable Logic and its Applications (FPL 2007)*, pages 317 – 322, Amsterdam, Netherlands, August 2007.

[12] Martin Schoeberl. Evaluation of a Java processor. In *Tagungsband Austrochip 2005*, pages 127–134, Vienna, Austria, October 2005.

[13] Martin Schoeberl. *JOP: A Java Optimized Processor for Embedded Real-Time Systems*. PhD thesis, Vienna University of Technology, 2005.

[14] Martin Schoeberl. A Java processor architecture for embedded real-time systems. *Journal of Systems Architecture*, doi:10.1016/j.sysarc.2007.06.001, 2007.

[15] Martin Schoeberl. A time-triggered network-on-chip. In *International Conference on Field-Programmable Logic and its Applications (FPL 2007)*, pages 377 – 382, Amsterdam, Netherlands, August 2007.