Technical University of Denmark



Clock domain crossing modules for OCP-style read/write interfaces

Herlev, Mathias; Sparsø, Jens

Publication date: 2016

Document Version Publisher's PDF, also known as Version of record

Link back to DTU Orbit

Citation (APA): Herlev, M., & Sparsø, J. (2016). Clock domain crossing modules for OCP-style read/write interfaces. Kgs. Lyngby: Technical University of Denmark (DTU). (DTU Compute-Technical Report-2016; No. 4).

DTU Library Technical Information Center of Denmark

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.

- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Clock domain crossing modules for OCP-style read/write interfaces

Mathias Herlev and Jens Sparsø

Department of Applied Mathematics and Computer Science Technical University of Denmark

Email: s103006@student.dtu.dk, jspa@dtu.dk

April 29, 2016

DTU Compute Technical Report-2016-4. ISSN: 1601-2321

Abstract

The open core protocol (OCP) is an openly licensed, configurable, and scalable interface protocol for on-chip subsystem communications. The protocol defines read and write transactions from a master towards a slave across a point-to-point connection and the protocol assumes a single common clock.

This paper presents the design of two OCP clock domain crossing interface modules, that can be used to construct systems with multiple clock domains. One module (called OCPio) supports a single word read-write interface and the other module (called OCPburst) supports a four word burst read-write interface.

The modules has been developed for the T-CREST multi-core platform [8, 9, 13] but they can easily be adopted and used in other designs implementing variants of the OCP interface standard. The OCP module is used to connect a Patmos processor to a message passing network-on-chip and the OCP burst is used to connect the Processor and its cache controllers to a shared off-chip memory.

While the problem of synchronizing a simple streaming interface is well described in the literature and often solved using bi-synchronous FIFOs we found surprisingly little published material addressing synchronization of bus-style read-write transaction interfaces. An OCP interface typically has control signals related to both the master issuing a read or write request and the slave producing a response. If all these control signals are passed across the clock domain boundary and synchronized it may add significant latency to the duration of a transaction. Our interface designs avoid this and synchronize only a single signal transition in each direction during a read or a write transaction. The designs are available as open source, and the modules have been tested in a complete multi-core platform implemented on an FPGA board.

Contents

eface	ii
cknowledgements	iii
Introduction	1
Background and related work	3
OCP transactions in T-CREST	5
Designs4.1The OCPio CDC-module4.2The OCPburst CDC-module	8 8 11
Latency of a transaction5.1Analysis5.2Worst-case and best-case bounds	15 15 18
Implementation	19
Conclusion	21
eferences	23
Code A.1 OCPio A.1.1 OCPio A side A.1.2 OCPio B side A.1.2 OCPio B side A.2 OCPburst A.2.1 OCPburst A side A.2.2 OCPburst B side A.3 Packages A.3.1 OCP Types A.3.2 OCPburst CDC Types A.3.3 OCPio CDC Types	24 24 28 32 32 37 41 41 43 44
	eface cknowledgements Introduction Background and related work OCP transactions in T-CREST Designs 4.1 The OCPborSt CDC-module

Preface

The work presented in this report started as a small project in course 02204 Design of Asynchronous Circuits in the spring 2014. The work was continued in a special topics course during the summer and this resulted in a paper presented at the Norchip 2014 conference in Helsinki [5]. This paper presented two alternative designs of a module implement clock domain crossing of a single-word read-write transaction – a version that buffers address and data and a version that avoids such buffering.

Following the publication of [5], we continued the work and expanded with a module implementing implement clock domain crossing burst read-write transaction. In addition abandoned the unbuffered single-word design and eliminated a potential timing glitch/bug in the buffered design. In this report, we present the final designs of the single-word transaction and the burst transaction clock domain crossing modules, and we provide an analysis of the latency of the read and write transactions as seen from the master. Both designs have been extensively tested in the T-CREST multicore platform, and the source code is available as open source.

Acknowledgements

This work was partially funded by the Danish Council for Independent Research | Technology and Production Sciences, Project no. 12-127600: Hard Real-Time Embedded Multiprocessor Platform (RTEMP) project.

Chapter 1 Introduction

The design of Systems-on-Chip (SoCs) where billions of transistors are integrated in a single chip puts emphasis on modularity and reuse of components. Components like processors, cache memories, IO-units, and HW-accelerators are typically developed by different vendors and are collectively known as intellectual property cores (IP-cores). To support reuse and modular composition of such IP-cores, interfaces are of paramount importance and a number of interface standards have emerged. Three typical and widely used interfaces are Wishbone, Open Core Protocol (OCP), and AMBA-AXI, introduced in more detail in the next chapter. They all specify a point-to-point connection that offers read and write transactions from a master (M) towards a slave (S), and they all assume synchronous operation of the master and the slave.

Another important aspect of designing SoCs is the timing organization. IP-cores may be designed for different clock frequencies and, to save power, scaling of the clock frequency, and the supply voltage, is often employed at the level of individual IP-cores. In addition the timing uncertainty in today's sub-micron CMOS technologies make clock-distribution and globally synchronous operation practically impossible [12]. As a result SoCs typically use some form of globally-asynchronous locally-synchronous operation [2].

In this report we address the design of two clock domain crossing interface modules (in the following denoted CDC-modules), that each implement a distinct subset of the



Figure 1: Block diagram of the OCP-to-OCP clock domain crossing module.

OCP, see Figure 1. The CDC-modules have been developed for the T-CREST multicore platform [13]. One CDC-module supports single-word transactions and the other CDC-module supports four-word burst transaction. Our designs synchronize only a single signal in each direction for a complete transaction. In this way, the performance impact of synchronization is reduced to the minimum possible.

In [5] we presented two designs for the single-word transaction CDC-module; one design using buffer-registers for all signals and another slightly slower but minimum hardware solution that avoids buffering of address and data signals. It turned out that the former design suffered from a timing glitch problem. Fixing this problem increased the latency by one cycle in each direction. As the use of buffer registers simplify timing analysis by breaking signal paths directly from one clock domain to the other, our preferred designs use buffer registers. In this report, we extend the work with a CDC-module that supports four-word burst transactions. In addition, we made a minor improvement of the single-word CDC-module. The aim of this report is to document the final designs of the two CDC-modules developed for and used in the T-CREST platform. The report is largely based on and reusing material from [5]. The designs have been tested in platform with four processor nodes implemented on an FPGA-board. The code is open sourced under a simplified (2-Clause) BSD license, and is available on Github [11], as well as being listed in Appendix A.

The report is organized as follows. Chapter 2 presents background material and related work. Chapter 3 presents the specific OCP interfaces that we use. Chapter 4 presents the design of the clock domain crossing module. Chapter 6 presents results of Field Programmable Gate Array (FPGA) realization. Finally, Chapter 7 concludes the report.

Chapter 2

Background and related work

The Open Core Protocol (OCP) is an openly licensed interface originally developed by the OCP International Partnership organization and now maintained by the Accellerata Systems Initiative [1]. The "Wishbone System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores" – in short the Wishbone bus – is an open source hardware interface that is used by many designs in the OpenCores project [6]. Both Wishbone and OCP specify signals and protocols for point-to-point connections between a master and a slave, allowing a master to perform read or write transactions into the address space of the slave. Wishbone specifies a single bus standard while OCP is highly configurable and scalable. Typical instances of OCP are very similar to the wishbone protocol. The Advanced Microcontroller Bus Architecture, Advanced eXtensible Interface (AMBA-AXI) is a bus developed by ARM Inc. It uses separate channels for address, write data and read data while OCP and Wishbone are more conventional bus-style interfaces.

Common to the three interface standards mentioned above is that they all assume a single common clock. In multi-clock systems there is a need for clock domain crossing modules that implement the same interface on both sides except for the different clocks.

There is a rich body of literature addressing communication between different clock domains and the problems related to synchronization and metastability are well understood [3, Ch. 10] [4]. Common to all forms of clock domain crossing is that synchronization of a signal (by passing it through two or more flip-flops) incurs latency.

Most published solutions consider a simple streaming interface between a producer and a consumer. A commonly used solution when connecting a producer and a consumer is to use a bi-synchronous FIFO that provide full and empty signals that are synchronized to the producer and receiver clocks respectively [3, Ch. 10] [4] [7] [15].

A CDC-module for bus-style transactions like OCP is a more challenging design than for a simple streaming interface. Read or write transactions are typically atomic and blocking and use flow control signals related to the transmission of both: (a) command and address, (b) write-data and, (c) read-data. The blocking behavior means that transactions cannot be pipelined and the accumulated latency of synchronizing several flow control signals may be significant.

We have only been able to find few publications that address clock domain crossing between two (bus-style) read/write-transaction interfaces. Closest to our work is [14] [16] [10]. Common to these designs are the use of several bi-synchronous FIFO's, for example for address and data, for write data and for read data. Furthermore both [14] and [10]

consider the interfacing between a synchronous domain and an asynchronous domain.

Our designs connect two identical clocked interfaces driven by independent clocks, and our designs avoid the use of FIFO-based synchronizers resulting in very small and efficient hardware implementations.

Chapter 3

OCP transactions in **T-CREST**

The OCP standard [1] allows for a large variety of specific point-to-point master-to-slave protocol instances. The context for the work presented in this paper is T-CREST multi-core processor [8, 13]. In this design two instances of the OCP protocol using the set of signals shown in Table 1 are used [9]:

- 1. OCPio; a single-word read-write interface used to access memories and IO-devices connected to a processor node. Transactions on this interface are generated by load and store instructions executed by the processor. Some example transactions are shown in Figure 2.
- 2. **OCPburst**; a burst read-write interface used to access a shared memory. Transactions on this interface are initiated by the cache controller in the processor node. The burst size is fixed to blocks of four words. A burst is identified by an aligned address and transferred as an immediate succession of words from incrementing addresses. Some example transactions are shown in Figure 3.

Signal	Description
MCmd[2:0] MAddr[31:0] MData[31:0] MDataValid *) MDataByteEn[3:0] MRespAccept	Command (idle, read or write) Address, byte-based, MAdr[1:0] always 00 Data for writes Signal that write data is valid. Byte-level mask for sub-word writes Signal that read data is accepted
SCmdAccept SDataAccept *) SData[31:0] SResp[1:0]	Signal that command is accepted Signal that write data is accepted. Data for reads Slave response: NULL - No response DVA - Data valid/accept FAIL - Request failed ERR - Response error

Table 1: Signals in the OCPio and OCPburst interfaces.

*) only implemented in the OCPburst interface.



Figure 2: Timing diagram for the OCPio interface. (Reprinted from [9] with kind permission of its authors).



Figure 3: Timing diagram for the **OCPburst** interface. (Reprinted from [9] with kind permission of its authors).

Both interfaces provide elasticity at the clock-cycle level and include several flowcontrol signal pairs: MCmd and SCmdAccept that control the transfer of the command and also write data for the OCPio interface; MDataValid and SDataAccept that control the transfer of write data for the OCPburst interface; SResp and MRespAccept that control the transfer of read data. SResp also serves the purpose of indicating the correct or faulty conclusion of a complete (read or write) transaction. Thus, for both OCPio and OCPburst, a write is terminated by a response from the Slave. More details can be found in the handbook for the PATMOS processor [9].

Chapter 4

Designs

The designs for both the OCPio and the OCPburst CDC-modules are based the same key principles, and the OCPburst CDC-module can be seen as an extension of the OCPio design. In this chapter, we first introduce the design of the OCPio CDC-module and the underlying principles and we then we present the design of the OCPburst CDC-module.

4.1 The OCPio CDC-module

The design of the OCPio CDC-module is a lightly modified version of the buffered design we presented in [5]. The design can be seen in Figure 4. As depicted in figure 4 the design consist of two parts each controlled by a finite state machine, denoted FSM_A and FSM_B. These FSMs are responsible for the signaling on the interfaces towards the OCP master and the OCP slave. The FSMs interact and coordinate their operation using two signals Req and Ack that are synchronized using a pair of flip-flops in the destination clock-domain. To minimize the latency of a transaction (as seen from the master) it is important to minimize the number of signal events that needs to be synchronized. Our design involves only one event (signal transition) on the Ack signal and one event on the Req signal per OCP transaction. In this way, the Req and Ack signals implement a nonreturn-to-zero (NRZ) or two-phase handshake per transaction. The third flip flop and the exclusive-or gate involved in the Req and Ack handshaking converts the (synchronized) signal transitions into pulses with a duration of one clock period.

OCP-signals are buffered in the source side and loading of a buffer registers is controlled by the FSM. The and-gates driving signals B_MCmd[2:0] allow FSM_B to drive these signals low until the synchronized signal Req_event indicate that a transaction is in progress. In the same manner, FSM_B can drive signal A_SResp[1:0] low until the synchronized signal Ack_sync indicate that a response is ready.

The detailed operation of the design is determined by the two FSMs. Figure 5 shows ASM-charts for FSM_A and FSM_B.

When a command is issued on the A_MCmd signal, the control FSM is in the Idle state. Upon receiving the command, it stores the signals A_MCmd, A_MAddr, A_MData, and A_MByteEn in registers, and asserts the req on the next rising clock edge of clkA. Upon this, the FSM_A controller will transition to state AckWait. On a req_event the FSM_B controller asserts EnB, and waits for the Slave to accept and respond. If the slave does not accept immediately, the FSM_B controller moves into state CmdAcceptWait. If, when in



Figure 4: Diagram showing the implementation of the OCPio CDC-module.



Figure 5: OCPio ASM Chart. The dashed arrows show a request (I) and an acknowledge (II) handshake.

CmdAcceptWait, the slave accepts but does not provide a response, the FSM_B controller will transition into RespWait. At any point, when the slave does provide a response, said response will be stored, and any accompanying data, in the register, by asserting LdB. When the response has been stored, the controller will assert Ack. When the Ack has been synchronized, the FSM_A controller will assert EnA, and await a response accept. If the OCP Master does not accept immediately, the controller transitions to RespAcceptWait. When the response has been accepted, the FSM_A controller transitions back to Idle, ready to receive a new command.

4.2 The OCPburst CDC-module

The design of the OCPburst CDC-module is an extension of the OCPio design. The design can be seen in Figure 4. The concept of treating all multibit signals as data still stands for this design. The values of the A_MCmd and A_MAddr signals are the same for an entire transaction, as opposed to for example the A_MData signal, and thus only require only a single register. For the signals A_MData and A_MDataByteEn (on the A-side) and B_SResp and B_SData (on the B-side) the design uses a register-file of the same size as the burst. In addition, each side has a register to keep track of how many words have been read or written.

The registers on the A-side are clocked using clkA, while the registers on the B-side are clocked using clkB. Reading from the other side of the interface is a combinatorial and asynchronous operation.

The state machines for the controllers in the OCPburst CDC-module can be seen in Figure 7. Unlike the OCPio design where the behavior is independent of the command type (read or write), both A-side and-B side in the OCP burst design is dependent on whether it is a read or a write command.

Read Upon a read command the FSM_A controller, will assert the LdA signal to register the "data", and on the rising clock edge drive a signal transition of the req signal, and transition to the state A_ReadBlockWait. Once the req signal has been synchronized to B side, generating a req_event the FSM_B controller asserts EnB to allow the signals to pass through to the OCP Slave. If the Slave accepts immediately, the FSM_B controller transitions to the B_ReadBlock state. In case it does not accept immediately, it transitions to B_ReadBlockWait, until the command has been accepted. If the command has been accepted, the FSM_B controller will also check if the first response is provided, and increment the RegAddr. Every time a response is available the RegAddr is incremented. When the third response has been stored (and RegAddr is reset) ack is asserted. Once ack has been synchronized, generating an ack_event, the FSM_A controller asserts A_SCmdAccept, and transitions into A_ReadBlock, where, one by one, the responses stored on the B side are transmitted to the OCP Master.

Write If instead the master issues a write command, the FSM_A controller immediately asserts A_SCmdAccept, and begin buffering the data in the register files by transitioning into A_WriteBlock. Once RegAddr reaches 3, the FSM_A controller asserts a request, and transitions into A_WriteBlockWait. Once the req has been synchronized, generating a



Figure 6: Diagram showing the implementation of the OCPburst CDC-module.



Figure 7: OCPburst ASM Chart. The dashed arrows shows a request (I) and acknowledge (II) handshake

req_event, the FSM_B controller asserts EnB allowing the command to pass through to the OCP Slave. If the slave accepts, it transitions directly to B_WriteBlock and transmits each buffered word. Otherwise, it waits in the state B_WriteBlockWait. Once all words have been transmitted it transitions into B_WriteBlockFinal. In B_WriteBlockFinal the FSM_B controller waits until the Slave responds, upon which it saves the response, in the response register file, and asserts ack. When ack has been synchronized, the FSM_A controller asserts EnA, for a single cycle and transitions back to A_Idle.

Chapter 5

Latency of a transaction

In this section we analyze latency of the OCP transactions as seen by a master that performs a read or write towards a slave that sits on the other side of a clock domain crossing module. We first analyze the factors that contribute to this latency and then use this analysis to provide expressions for the worst-case latency of the different OCPtransactions.

5.1 Analysis

The latency depends on the type of interface (OCPburst or OCPio), the type of transaction (read or write), the ratio between the two clock signals (ClkA and ClkB) and the phase between the two clock signals when a signal from one clock domain is synchronized to the other domain. Furthermore, it should be kept in mind (c.f. chapter 3) that the OCPburst and OCPio protocols allow a slave to idle between a command and the associated response and allow a master to idle between a response and the acknowledgement of the response.

The latency of a transaction can be subdivided into 5 intervals. Since the different transactions (OCPburst read, OCPburst write, OCPio read and OCPio write) are relatively similar the following analysis is structured according to the 5 intervals. The reader is encouraged to refer to Figure 4 and Figure 6 while reading the descriptions.

Interval [a]: MCmd \neq idle \rightarrow Req \updownarrow

The time from A_MCmd changes from Idle to a valid command until a transition is produced on signal Req. Everything happens in the ClkA domain and the latencies for the different transactions are:

OCPio	Wr:	1 cycle @ClkA
	Rd:	1 cycle @ClkA
OCPburst	Wr:	N cycles @ClkA
	Rd:	1 cycle @ClkA

where N is the number of words in a burst transfer.

Interval [b]: Req $\uparrow \rightarrow \text{Req}_{-}\text{event} \uparrow$

The time from an event (an up or down-going transition) on Req until the beginning

of the synchronized one-cycle wide pulse that is produced on signal Req_event. The time is related to ClkB and also depends on the phase difference between ClkB the Req signal that is produced in the ClkA domain. If the transition of Req happens slightly before the rising edge of ClkB the duration of interval [b] is slightly more than 1 cycle @ClkB and if the transition happens slightly after the rising edge of the duration of interval [b] is slightly less than 2 cycles @ClkB. If the transition coincide with the rising edge of ClkB, the first flip-flop in the two flop synchronizer goes metastable, and the resulting duration of interval [b] is one or two cycles corresponding to the two previously mentioned scenarios.

In summary, the duration of interval [b] is]1;2[cycles @ClkB.

Interval [c]: Req_event $\uparrow \rightarrow Ack \updownarrow$

The time from a synchronized req_event until an event (an up or down-going transition) on signal Ack. This includes the time it takes for the OCP slave to respond to the transaction, and the time required to buffer the response. Everything relates to ClkB and for each of the 4 possible transactions the duration of interval [c] and is as follows:

OCPio	Wr:	$1 + n_S$ cycles @CI	.kB
	Rd:	$1 + n_S$ cycles @CI	kB
OCPburst	Wr:	$1 + N + n_S$ cycles	@ClkB
	Rd:	$1 + N + n_S$ cycles	@ClkB

where N is the number of words in a burst transfer and where n_S is the accept/response time of the slave. For the OCPio write command illustrated in Figure 2 $n_S = 1$; the cycle from B to C. For the OCPburst write command illustrated in Figure 3 $n_S = 1$; the cycle from G to H. For the OCPio write command illustrated in Figure 3 $n_S = 1$; the cycle from B to C. In our current implementation N = 4 and $n_S = 0$.

Interval [d]: Ack $\uparrow \rightarrow$ Ack_event \uparrow

The time from an event (an up or down-going transition) on Ack until the beginning of the synchronized one-cycle wide pulse that is produced on signal Ack_event. The analysis is similar to interval [b] and the duration of interval [d] is]1;2[cycles @ClkA.

Interval [e]: Ack_event $\uparrow \rightarrow$ Transaction completed (FSM_A enters state Idle)

The time from a synchronized req_event until the A-side of the CDC-module is ready to receive a new command. This includes the time it takes to write buffered responses (including data) back to the OCP Master, the time it takes the master to accept a response, and any latency added by the FSM_A. For the OCPio transactions it takes a minimum of 1 cycle for the master to accept the response. However, it can delay for an unspecified number of cycles n_M . For the OCPburst transactions such delaying is not allowed. Everything relates to ClkA and for each of the 4 possible transactions the duration of interval [e] and is as follows:

OCPio	Wr:	$1 + n_M$ cycles	@ClkA
	Rd:	$1 + n_M$ cycles	@ClkA
OCPburst	Wr:	1 cycle @C	lkA
	Rd:	N cycles @C	lkA

where N is the number of words in a burst transfer and where n_M is the delay between the response and the acknowledgement of the response that is allowed for the OCPio transactions. In In our current implementation N = 4 and $n_M = 0$.

Figure 8 shows an example OCPio Read transaction propagated across the OCPio CDC-module. The set of signals shown is reduced to improve readability.



Figure 8: An example read-transaction propagated across the OCPio CDC-module.

5.2 Worst-case and best-case bounds

=

In the following T_A , f_A , T_B and f_B denote the period and frequency of ClkA and ClkB respectively. The latency of a transaction as seen from the master (operating at ClkA) is the sum of intervals [a] through [e]. Assuming for intervals [b] and [d] the maximum synchronization latency of 2 cycles we get the following sum for the OCPio write transaction.

$$T_{Worst_OCPio,Rd} = 1 \cdot T_A + 2 \cdot T_B + (1+n_S) \cdot T_B + 2 \cdot T_A + (1+n_M) \cdot T_A$$
(5.1a)

$$(4+n_M) \cdot T_A + (3+n_S) \cdot T_B$$
 (5.1b)

Keeping in mind that the latency is seen as an integer number of cycles of ClkA we get the following worst-case latency of an OCPio Read transaction:

$$T_{Worst_OCPio,Rd} = \left\{ (4+n_M) + \left\lfloor (3+n_S) \cdot \frac{T_B}{T_A} \right\rfloor \right\} T_A$$
(5.2a)

$$= \left\{ (4+n_M) + \left\lfloor (3+n_S) \cdot \frac{f_A}{f_b} \right\rfloor \right\} \frac{1}{f_A}$$
(5.2b)

The reason we use the floor-operator rather than the ceiling-operator in an expression for a worst-case latency is a bit subtle and is illustrated in Figure 9. In the above expression we calculate interval [d] to be two 2 cycles @ClkA, but as shown in Figure 9 the beginning of interval [d] is related to ClkB. When interval [d] is taken as 2 cycles @ClkA it overlaps with interval [c], and this explains the use of the floor-operator. With this explanation the reader is encouraged to refer back to Figure 8.



Figure 9: Relationship between intervals [c] and [d].

In a similar way we we obtain the expressions for the remaining three transactions:

$$T_{Worst_OCPio,Wr} = \left\{ (4+n_M) + \left\lfloor (3+n_S) \cdot \frac{f_A}{f_b} \right\rfloor \right\} \frac{1}{f_A}$$
(5.3)

$$T_{Worst_OCPburst,Wr} = \left\{ (3+N) + \left\lfloor (3+N+n_S) \cdot \frac{f_A}{f_b} \right\rfloor \right\} \frac{1}{f_A}$$
(5.4)

$$T_{Worst_OCPburst,Rd} = \left\{ 3 + \left\lfloor (3 + N + n_S) \cdot \frac{f_A}{f_b} \right\rfloor \right\} \frac{1}{f_A}$$
(5.5)

Chapter 6

Implementation

Both designs have been implemented in RTL VHDL and verified using ModelSim. In addition to this, both interfaces have been implemented in a 4 core T-CREST platform synthesized for an Altera DE2-115, with a Master core (Patmos 0) running at a different clock from the rest of the platform. To ensure independent clocks, the master core is running using the onboard 50MHz clock, while an external clock generator running at a variable frequency (up to 50MHz) is driving the rest of the platform.

Altera Quartus 15.0 puts the MTBF for the synchronization chain at greater than 1 billion years. Additionally, the physical setup was tested over a week with no failures.

As both interfaces are fairly simple, their resource and speed measurements are less interesting than the performance impact. As we noted in Chapter 5 the simplest way to compare performance impact is to assume that two independent clock domains each running at equal frequencies, with a constant phase difference always resulting in Worst-Case Execution Time (WCET), and compare this to an interface without clock crossing.

OCPio For OCPio two best-case performances exist, ie. 1 cycle per word for reads, and 2 cycles per word for writes. Using worst-case timings with clock domain crossings, this becomes 7 cycles per word for both. This leaves us with a 1/7 bandwidth for reads and 2/7 bandwidth for writes. Of course we note that if the slave introduces a read or write delay n_S then we will achieve $\frac{1+n_S}{7+n_S}$ for reads and $\frac{2+n_S}{7+n_S}$.

OCPburst For OCPburst one best-case performance exist; 5 cycles per 4 words. Using worst-case timings with clock domain crossings, this becomes 14 cycles per word for both. This leaves us with a 5/14 bandwidth. Of course we note that if the slave introduces a read or write delay n_S then we will achieve $\frac{5+n_S}{14+n_S}$. For an external memory, such as a DRAM, n_S is relatively high, meaning that the performance of the domain crossing interface will trend towards the synchronous interface. Figure 10 shows the performance of the slave operate

	LC Combinatorial	LC Register
Burst	309	320
IO	86	93

Table 2: Synthesis Results



Figure 10: Performance of the two OCP CDC-modules normalized to a situation where the master and slave operate synchronously and are connected directly (without a CDC-module). For a memory the slave delay is the access time of the memory.

synchronously and without a clock domain crossing module. As seen, the performance approaches that of a plain synchronous interface as the access time of the slave increases. The latter is a likely situation when several processors share and access an of chip DRAM-based memory.

Chapter 7

Conclusion

This report has presented two different designs of a clock domain crossing module for two distinct OCP-style interfaces supporting either single word read/write transactions, or burst read/write transactions, respectively. Both designs use Non-Return-to-Zero (NRZ) synchronization protocols, and pass the command, address, write data and read data signals through buffer registers clocked by the source clock.

The performance of the two designs can be quantified by how many cycles a transaction takes when a clock domain crossing module is added between a synchronous master and slave pair, assuming no accept/response delays. For the OCPio design, this is 7 cycles. For the OCPburst design, it is 14 cycles. We note however that for OCPburst, whose primary application is interfacing towards a shared main memory, the latency can be masked by the relatively high access time of the memory.

Furthermore the basic structure and the underlying ideas can be used in the design of clock domain crossing modules for other bus-style read/write-transaction interfaces such as Wishbone.

Bibliography

- [1] Accellera Systems Initiative. Open Core Protocol specification, release 3.0. http: //www.accellera.org/downloads/standards/ocp/ocp_3.0/, 2013.
- [2] Daniel M. Chapiro. Globally-Asynchronous Locally-Synchronous Systems. PhD thesis, Stanford University, October 1984. Report No. STAN-CS-84-1026.
- [3] W. J. Dally and J. W. Poulton. *Digital Systems Engineering*. Cambridge University Press, 1998.
- [4] R. Ginosar. Metastability and synchronizers: A tutorial. IEEE Design & Test of Computers, 28(5):23–35, 2011.
- [5] Mathias Herlev, Christian Keis Poulsen, and Jens Sparsø. Open Core Protocol (OCP) Clock Domain Crossing Interfaces. In Proc. 32th IEEE Norchip Conference, pages 1–6, 2014.
- [6] OpenCores. Wishbone B4: The WISHBONE System-on-Chip (SoC) Interconnect Architecture for Portable IP Cores. http://www.opencores.org, 2010.
- [7] I. Miro Panades and A. Greiner. Bi-synchronous FIFO for synchronous circuit communication well suited for network-on-chip in gals architectures. In Proc. IEEE/ACM Intl. Symposium on Networks-on-Chip (NOCS), pages 83–92, 2007.
- [8] Martin Schoeberl, Sahar Abbaspourseyedi, Alexander Jordan, Evangelia Kasapaki, Wolfgang Puffitsch, Jens Sparsø, Benny Akesson, Neil Audsley, Jamie Garside, Raffaele Capasso, Alessandro Tocchi, Kees Goossens, Sven Goossens, Yonghui Li, Scott Hansen, Reinhold Heckmann, Stefan Hepp, Benedikt Huber, Jens Knoop, Daniel Prokesch, Peter Puschner, Andr Rocha, and Cludio Silva. T-crest: Time-predictable multi-core architecture for embedded systems. *Journal of Systems Architecture*, 61(9):449–471, 2015.
- [9] Martin Schoeberl, Florian Brandner, Stefan Hepp, Wolfgang Puffitsch, and Daniel Prokesch. Patmos Reference Handbook. Technical Report, Technical University of Denmark. patmos.compute.dtu.dk/patmos_handbook.pdf, 2016.
- [10] Ahmed H. M. Soliman, E. M. Saad, M. El-Bably, and Hesham M. A. M. Keshk. Designing a WISHBONE Protocol Network Adapter for an Asynchronous Networkon-Chip. International Journal of Computer Science Issues, 8(2):261–267, 2012.
- [11] T-CREST. Github Argo source code repository. https://github.com/t-crest/ argo.git, 2016.

- [12] The International Technology Roadmap for Semiconductors. http://www.itrs2. net/.
- [13] Time-predictable Multi-Core Architecture for Embedded Systems (T-CREST). www. t-crest.org.
- [14] Vikas S. Vij, Raghu Prasad Gudla, and Kenneth S. Stevens. Interfacing synchronous and asynchronous domains for open core protocol. In 27th International Conference on VLSI Design and 13th International Conference on Embedded Systems, pages 282–287, 2014.
- [15] P. Wielage, J.E. Marinissen, M. Altheimer, and C. Wouters. Design and DfT of a high-speed area-efficient embedded asynchronous FIFO. In *Proc. Design, Automation and Test in Europe (DATE)*, pages 853–858, 2007.
- [16] Po Han Wu, Chi Guang Lin, Chia Sheng Wen, and Shen Fu Hsiao. Asynchronous AHB bus interface designs in a multiple-clock-domain graphics system. *IEEE Asia-Pacific Conference on Circuits and Systems, Proceedings, APCCAS*, pages 408–411, 2012.

Appendix A

Code

1

A.1 OCPio

A.1.1 OCPio A side

-- Copyright (c) 2016, Mathias Herlev 2 3 — All rights reserved. 4 ____ 5-- Redistribution and use in source and binary forms, with or without 6 -- modification, are permitted provided that the following conditions are met: 7 8 - 1. Redistributions of source code must retain the above copyright notice, 9 -- this list of conditions and the following disclaimer. - 2. Redistributions in binary form must reproduce the above copyright notice, 10 -- this list of conditions and the following disclaimer in the documentation 11 -- and/or other materials provided with the distribution. 1213 14--- THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" 15 --- AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE --- IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE 1617 — ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE 18 -- LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR -- CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF 19 --- SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS 20--- INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN 21 22-- CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) --- ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE 2324--- POSSIBILITY OF SUCH DAMAGE. 25___ 26-- Title: OCPIO Clock Crossing Interface Slave -- Type 27: Entity: Slave Interface for the OCP clock crossing. Connects to a 28-- Description 29: master30 31 **LIBRARY** ieee; 32 USE ieee.std_logic_1164.all;

- 33 USE ieee.numeric_std.all;
- 34 LIBRARY work;
- 35 USE work.OCPIOCDC_types.all;

```
36 USE work.ocp.all;
37
38
   ENTITY OCPIOCDC_A IS
       GENERIC(IOSize : INTEGER := 1);
39
40
       PORT(
                 clk
                              : IN
                                       std_logic;
                              : IN
                                       std_logic;
41
                 rst
                              : IN
42
                 syncIn
                                       ocp_io_m;
43
                syncOut
                              : OUT
                                       ocp_io_s;
44
                 asyncOut
                              : OUT
                                       asyncIO_A_r;
                              : IN
                                       asyncIO_B_r
45
                 asyncIn
46
        );
   END ENTITY OCPIOCDC_A;
47
48
49
   --- Buffered Architecture
50
   ARCHITECTURE Buffered OF OCPIOCDCA IS
51
52
53
        -- FSM signals
54
                                  (IDLE_state, AckWait_state, RespAcceptWait_state);
55
       TYPE fsm_states_t IS
56
       SIGNAL state, state_next
                                            fsm_states_t;
                                       :
57
58
59
        -- Async signals
60
61
       SIGNAL ack_prev, ack, ack_next : std_logic := '0';
       SIGNAL req, req_next
                                           : std_logic := '0';
62
63
64
65
        --- Registers
66
67
        SIGNAL masterData, masterData_next : ocp_io_m;
68
        SIGNAL writeEnable : std_logic := '0';
69
   BEGIN
70
71
72
        asyncOut.req <= req;
73
        asyncOut.data <= masterData;</pre>
74
75
        -- FSM
76
77
78
        FSM : PROCESS(state, syncIn, asyncIn, ack, ack_prev, req)
79
       BEGIN
80
            state_next
                             <= state;
            syncOut
                             <= OCPIOSlaveIdle_c;
81
82
            writeEnable
                             <= '0';
83
            req_next
                              \leq \operatorname{req};
84
            CASE state IS
85
                WHEN IDLE_state =>
86
                     --- If command is different from idle
87
                     IF syncIn.MCmd /= OCP_CMD_IDLE THEN
88
89
                         --Signal the B side
90
                         req_next \ll NOT(req);
```

	<pre>state_next <= AckWait_state;</pre>	
	Buffer the command, address, and data	
	writeEnable <= '1';	
	END IF;	
	WHEN AckWait_state =>	
	If the slave has acknowledged	
	IF $ack = NOT$ (ack prev) THEN	
	Then signal the master	
	state next <- RespAccentWait state:	
	suncOut <= asuncIn data:	
	syncOut < asynchia. data,	
	Syncout.SondAccept <= 1;	
	$\mathbf{H} \text{syncm} \text{MRespAccept} = 1 \mathbf{H} \mathbf{H} \mathbf{N}$	
	If the master accepts, go to iale	
	state_next <= IDLE_state;	
	END IF;	
	— Else go to WriteWordFinal	
	END IF;	
	WHEN RespAcceptWait_state =>	
	— When OCP master accepts, go to idle	
	syncOut <= asyncIn.data;	
	$\mathbf{IF} \operatorname{syncIn} . \operatorname{MRespAccept} = '1 \mathbf{THEN}$	
	$state_next <= IDLE_state;$	
	END IF ;	
	WHEN OTHERS $=>$	
	$state_next \ll IDLE_state;$	
EN	VD CASE;	
EN END	ID CASE; PROCESS FSM;	
EN END	ND CASE; PROCESS FSM;	
END	VD CASE; PROCESS FSM;	
END :	ND CASE; PROCESS FSM; Registers	
END : END : !	ND CASE; PROCESS FSM; Registers	
END 2 END 2 1 Data	ND CASE; PROCESS FSM; Registers RegMux : PROCESS(writeEnable,syncIn)	
EN END 	ND CASE; PROCESS FSM; Registers RegMux : PROCESS(writeEnable,syncIn) N	
END END — 1 Data BEGI	<pre>ND CASE; PROCESS FSM; Registers RegMux : PROCESS(writeEnable,syncIn) N masterData next <= masterData:</pre>	
END END — 1 Data BEGI	ND CASE; PROCESS FSM; Registers RegMux : PROCESS(writeEnable,syncIn) IN masterData_next <= masterData; IF_writeEnable = '1' THEN	
END l Data BEGI	ND CASE; PROCESS FSM; Registers RegMux : PROCESS(writeEnable,syncIn) IN masterData_next <= masterData; IF writeEnable = '1' THEN masterData_next <= syncIn;	
END 	ND CASE; PROCESS FSM; Registers RegMux : PROCESS(writeEnable,syncIn) N masterData_next <= masterData; IF writeEnable = '1' THEN masterData_next <= syncIn; END IF:	
END ————————————————————————————————————	<pre>ND CASE; PROCESS FSM; Registers RegMux : PROCESS(writeEnable,syncIn) N masterData_next <= masterData; IF writeEnable = '1' THEN masterData_next <= syncIn; END IF; PROCESS DataBarMun;</pre>	
END ————————————————————————————————————	<pre>ND CASE; PROCESS FSM; Registers RegMux : PROCESS(writeEnable,syncIn) N masterData_next <= masterData; IF writeEnable = '1' THEN masterData_next <= syncIn; END IF; PROCESS DataRegMux;</pre>	
END Data BEGI	<pre>ND CASE; PROCESS FSM; Registers RegMux : PROCESS(writeEnable,syncIn) N masterData_next <= masterData; IF writeEnable = '1' THEN masterData_next <= syncIn; END IF; PROCESS DataRegMux; interest EDECENT(the set)</pre>	
END Data BEGI	<pre>ND CASE; PROCESS FSM; Registers MRegMux : PROCESS(writeEnable,syncIn) IN masterData_next <= masterData; IF writeEnable = '1' THEN masterData_next <= syncIn; END IF; PROCESS DataRegMux; isters : PROCESS(clk,rst)</pre>	
END 	<pre>ND CASE; PROCESS FSM; Registers N RegMux : PROCESS(writeEnable,syncIn) N masterData_next <= masterData; IF writeEnable = '1' THEN masterData_next <= syncIn; END IF; PROCESS DataRegMux; isters : PROCESS(clk,rst) N</pre>	
END 	ND CASE; PROCESS FSM; Registers Registers Registers Registers Registers Registers Registers Registers Registers Registers N masterData_next <= masterData; IF writeEnable = '1' THEN masterData_next <= syncIn; END IF; PROCESS DataRegMux; isters : PROCESS(clk,rst) N IF rst = '1' THEN	
END 	ND CASE; PROCESS FSM; Registers RegMux : PROCESS(writeEnable,syncIn) N masterData_next <= masterData; IF writeEnable = '1' THEN masterData_next <= syncIn; END IF; PROCESS DataRegMux; isters : PROCESS(clk,rst) N IF rst = '1' THEN state <= IDLE_state;	
END l l Data BEGI END Reg BEGI	ND CASE; PROCESS FSM; Registers RegMux : PROCESS(writeEnable,syncIn) N masterData_next <= masterData; IF writeEnable = '1' THEN masterData_next <= syncIn; END IF; PROCESS DataRegMux; isters : PROCESS(clk,rst) N IF rst = '1' THEN state <= IDLE_state; req <= '0';	
END 1 Data BEGI Reg BEGI	ND CASE; PROCESS FSM; Registers Registers RegMux : PROCESS(writeEnable,syncIn) N masterData_next <= masterData; IF writeEnable = '1' THEN masterData_next <= syncIn; END IF; PROCESS DataRegMux; isters : PROCESS(clk,rst) N IF rst = '1' THEN state <= IDLE_state; req <= '0'; ack_prev <= '0';	
END 	ND CASE; PROCESS FSM; Registers Registers RegMux : PROCESS(writeEnable, syncIn) N masterData_next <= masterData; IF writeEnable = '1' THEN masterData_next <= syncIn; END IF; PROCESS DataRegMux; isters : PROCESS(clk, rst) N IF rst = '1' THEN state <= IDLE_state; req <= '0'; ack_prev <= '0'; ack <= '0';	
END 	ND CASE; PROCESS FSM; Registers Registers RegMux : PROCESS(writeEnable,syncIn) N masterData_next <= masterData; IF writeEnable = '1' THEN masterData_next <= syncIn; END IF; PROCESS DataRegMux; isters : PROCESS(clk,rst) N IF rst = '1' THEN state <= IDLE_state; req <= '0'; ack_prev <= '0'; ack_next <= '0';	
END 	<pre>ND CASE; PROCESS FSM; Registers RegMux : PROCESS(writeEnable,syncIn) IN masterData_next <= masterData; IF writeEnable = '1' THEN masterData_next <= syncIn; END IF; PROCESS DataRegMux; isters : PROCESS(clk,rst) IN IF rst = '1' THEN state <= IDLE_state; req <= '0'; ack_prev <= '0'; ack_next <= '0'; masterData <= OCPIOmasteridle c:</pre>	
END Data BEGI	<pre>ND CASE; PROCESS FSM; Registers Registers RegMux : PROCESS(writeEnable,syncIn) N masterData_next <= masterData; IF writeEnable = '1' THEN masterData_next <= syncIn; END IF; PROCESS DataRegMux; isters : PROCESS(clk,rst) N IF rst = '1' THEN state <= IDLE_state; req <= '0'; ack_prev <= '0'; ack_next <= '0'; masterData <= OCPIOmasteridle_c;</pre>	
END Data BEGI	<pre>ND CASE; PROCESS FSM; Registers RegMux : PROCESS(writeEnable,syncIn) N masterData_next <= masterData; IF writeEnable = '1' THEN masterData_next <= syncIn; END IF; PROCESS DataRegMux; isters : PROCESS(clk,rst) N IF rst = '1' THEN state <= IDLE_state; req <= '0'; ack_prev <= '0'; ack_prev <= '0'; ack_next <= '0'; masterData <= OCPIOmasteridle_c; ELSIE rising edge(clk) THEN</pre>	
END Data BEGI	<pre>ND CASE; PROCESS FSM; Registers RegMux : PROCESS(writeEnable,syncIn) N masterData_next <= masterData; IF writeEnable = '1' THEN masterData_next <= syncIn; END IF; PROCESS DataRegMux; isters : PROCESS(clk,rst) N IF rst = '1' THEN state <= IDLE_state; req <= '0'; ack_prev <= '0'; ack_prev <= '0'; ack_next <= '0'; masterData <= OCPIOmasteridle_c; ELSIF rising_edge(clk) THEN state <= state post;</pre>	
END Data BEGI	<pre>ND CASE; PROCESS FSM; Registers Registers RegMux : PROCESS(writeEnable,syncIn) N masterData_next <= masterData; IF writeEnable = '1'THEN masterData_next <= syncIn; END IF; PROCESS DataRegMux; isters : PROCESS(clk,rst) N IF rst = '1'THEN state <= IDLE_state; req <= '0'; ack_prev <= '0'; ack_prev <= '0'; ack_next <= '0'; masterData <= OCPIOmasteridle_c; ELSIF rising_edge(clk) THEN state <= state_next; req <= 'req next;</pre>	
END 	<pre>ND CASE; PROCESS FSM; Registers Registers RegMux : PROCESS(writeEnable,syncIn) IN masterData_next <= masterData; IF writeEnable = '1 ' THEN masterData_next <= syncIn; END IF; PROCESS DataRegMux; isters : PROCESS(clk,rst) IN IF rst = '1 ' THEN state <= IDLE_state; req <= '0'; ack_prev <= '0'; ack_next <= '0'; ack_next <= '0'; masterData <= OCPIOmasteridle_c; ELSIF rising_edge(clk) THEN state <= state_next; req <= req_next; ack_prev <= otherway </pre>	
END 	<pre>ND CASE; PROCESS FSM; Registers Registers RegMux : PROCESS(writeEnable,syncIn) N masterData_next <= masterData; IF writeEnable = '1 ' THEN masterData_next <= syncIn; END IF; PROCESS DataRegMux; isters : PROCESS(clk,rst) N IF rst = '1 ' THEN state <= IDLE_state; req <= '0'; ack_prev <= '0'; ack_next <= '0'; ack_next <= '0'; masterData <= OCPIOmasteridle_c; ELSIF rising_edge(clk) THEN state <= state_next; req <= req_next; ack_prev <= ack; </pre>	
END 	<pre>ND CASE; PROCESS FSM; Registers RegMux : PROCESS(writeEnable,syncIn) N masterData_next <= masterData; IF writeEnable = '1' THEN masterData_next <= syncIn; END IF; PROCESS DataRegMux; Fisters : PROCESS(clk,rst) N IF rst = '1' THEN state <= IDLE_state; req <= '0'; ack_prev <= '0'; ack_prev <= '0'; ack_next <= '0'; masterData <= OCPIOmasteridle_c; ELSIF rising_edge(clk) THEN state <= state_next; req <= req_next; ack_prev <= ack; ack <= ack_next; req <= ac</pre>	
END 	<pre>ND CASE; PROCESS FSM; Registers RegMux : PROCESS(writeEnable,syncIn) N masterData_next <= masterData; IF writeEnable = '1' THEN masterData_next <= syncIn; END IF; PROCESS DataRegMux; Sisters : PROCESS(clk,rst) N IF rst = '1' THEN state <= IDLE_state; req <= '0'; ack_prev <= '0'; ack_next <= '0'; ack_next <= '0'; masterData <= OCPIOmasteridle_c; ELSIF rising_edge(clk) THEN state <= state_next; req <= req_next; ack_prev <= ack; ack_next <= asyncIn.ack;</pre>	

```
146
                  masterData <= masterData_next;</pre>
147
             END IF;
148
        END PROCESS Registers;
149
150 END ARCHITECTURE Buffered;
151
152
    ARCHITECTURE NonBuffered OF OCPIOCDCA IS
153
        TYPE fsm_states_t IS
                                    (IDLE_state, AckWait_state, RespAcceptWait_state,
154
                                    HandshakeFinal_state);
155
        SIGNAL state, state_next
                                              fsm_states_t;
                                        :
156
                                : std_logic := '0';
        SIGNAL ack, ack_next
157
                                    : std_logic := '0';
158
        SIGNAL req
159
160 BEGIN
161
162
         asyncOut.req
                               \leq req;
163
         asyncOut.data <= syncIn;
164
        FSM : PROCESS(state, syncIn, asyncIn, ack)
165
166
        BEGIN
167
             state_next <= state;</pre>
             syncOut
                          <= OCPIOSlaveIdle_c;
168
169
             CASE state IS
170
                 WHEN IDLE_state =>
171
                      req <= '0';
                      IF ack = 0, THEN
172
                          IF syncIn.MCmd /= OCP_CMD_IDLE THEN
173
174
                               req <= '1';
175
                               state_next <= AckWait_state;</pre>
                          END IF;
176
                      END IF;
177
178
                 WHEN AckWait_state =>
179
                      req <= '1';
                      IF ack = '1' THEN
180
                           state_next <= RespAcceptWait_state;</pre>
181
182
                           syncOut <= asyncIn.data;</pre>
183
                           syncOut.SCmdAccept <= '1';</pre>
                          IF syncIn.MRespAccept = '1' THEN
184
185
                               req <= '0';
186
                               state_next <= Idle_state;</pre>
                          END IF;
187
188
                      END IF;
189
                 WHEN RespAcceptWait_state =>
190
                      req <= '1';
                      syncOut <= asyncIn.data;</pre>
191
                      syncOut.SCmdAccept <= '0';
192
                      IF syncIn.MRespAccept = '1' THEN
193
194
                           state_next <= Idle_state;</pre>
195
                      END IF;
196
                 WHEN HandshakeFinal_state =>
                      req <= '0';
197
                      IF ack = '0' THEN
198
199
                           state_next <= Idle_state;</pre>
200
                      END IF;
```

201WHEN OTHERS => 202state_next <= IDLE_state;</pre> 203 END CASE: END PROCESS FSM; 204205206 207Registers : **PROCESS**(clk, rst) BEGIN 208 IF rst = '1' THEN 209210 <= IDLE_state; state <= '0'; 211ack <= '0';212 ack_next 213ELSIF rising_edge(clk) THEN 214 <= state_next; state 215ack $\leq = ack_next;$ <= asyncIn.ack; 216ack_next 217END IF: 218**END PROCESS** Registers; 219

220 END ARCHITECTURE NonBuffered;

A.1.2 OCPio B side

1 2 -- Copyright (c) 2016, Mathias Herlev 3 — All rights reserved. 4 ---5 -- Redistribution and use in source and binary forms, with or without -- modification, are permitted provided that the following conditions are met: 6 7 8 -- 1. Redistributions of source code must retain the above copyright notice, -- this list of conditions and the following disclaimer. 9 10 -- 2. Redistributions in binary form must reproduce the above copyright notice, 11 -- this list of conditions and the following disclaimer in the documentation 12 — and/or other materials provided with the distribution. 13 ---14 — THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" 15 --- AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE 16 — IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE 17 — ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE --- LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR 18 --- CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF 19 20 — SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS 21 -- INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN 22 — CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) 23 --- ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE 24 --- POSSIBILITY OF SUCH DAMAGE. 25 ---26 - Title: OCPIO Clock Crossing Interface Slave 27 - Type: Entity28 - Description: Master Interface for the OCP clock crossing. Connects to a 29 ---Slave ____ 30

31 LIBRARY ieee;

32 **USE** ieee.std_logic_1164.**all**;

33 USE ieee.numeric_std.all;

```
34 LIBRARY work;
35 USE work.OCPIOCDC_types.all;
36 USE work.ocp.all;
37
   ENTITY OCPIOCDC_B IS
38
39
       GENERIC(IOSize : INTEGER := 1);
                              : IN
40
       PORT(
                                       std_logic;
                 clk
                 \mathrm{r\,s\,t}
                              : IN
                                       std_logic;
41
42
                 syncIn
                              : IN
                                       ocp_io_s;
                              : OUT
43
                 syncOut
                                       ocp_io_m;
                              : OUT
                                       asyncIO_B_r;
44
                 asyncOut
                              : IN
45
                 asyncIn
                                       asyncIO_A_r
46
        );
47
   END ENTITY OCPIOCDC_B;
48
49
   --- Buffered Architecture
50
51
   ARCHITECTURE Buffered OF OCPIOCDCB IS
52
53
        -- FSM signals
54
55
                                   (IDLE_state, CmdAcceptWait_state, RespWait_state);
56
       TYPE fsm_states_t IS
57
       SIGNAL state, state_next
                                       :
                                             fsm_states_t;
58
59
        -- Async signals
60
        SIGNAL req_prev, req, req_next : std_logic := '0';
61
       SIGNAL ack, ack_next
62
                                            : std_logic := '0';
63
64
65
        --- Register signals
66
67
       SIGNAL slaveData, slaveData_next
                                             : ocp_io_s;
68
       SIGNAL loadEnable : std_logic;
69
   BEGIN
70
71
        asyncOut.data <= slaveData WHEN loadEnable = '0' ELSE syncIn;
72
73
        asyncOut.ack
                       \leq = \operatorname{ack};
74
75
76
        -- FSM
77
        FSM : PROCESS(state, syncIn, asyncIn, req, req_prev, ack)
78
79
       BEGIN
80
            state_next
                         \leq = state;
            loadEnable \leq 0':
81
                         <= OCPIOMasterIdle_c;
82
            syncOut
83
            ack_next
                         \leq = \operatorname{ack};
84
            CASE state IS
85
                WHEN IDLE_state =>
86
87
                     --- If a new request is available
88
                     IF req = NOT (req_prev) THEN
```

09	IF (async1n.data.MCmd /= OCP_CMD_IDLE) THEN
90	Relay the command to the OCP slave
91	syncOut <= asyncIn.data;
92	<pre>state_next <= CmdAcceptWait_state;</pre>
93	- If the command is accepted
94	IF $svncIn$. SCmdAccept = '1', THEN
95	state next <= RespWait state:
96	And a response is ready
97	IF syncin SBesn /- OCP BESP NULL THEN
08	state next \sim IDLE state:
00	loadEnable <= 11
33 100	$\frac{1}{2}$
100	Signal the A side
101	$ack_next \leq nOI (ack);$
102	syncout.MRespAccept <= 1;
103	END IF;
104	
105	
106	END IF;
107	WHEN CmdAcceptWait_state =>
108	If command has not been accepted, Command+data has not been
109	- Registered in OCP Slave. Continue aserting command.
110	syncOut <= asyncIn.data;
111	IF syncIn.SCmdAccept = '1' THEN
112	$state_next <= RespWait_state;$
113	IF syncIn.SResp /= OCP_RESP_NULL THEN
114	$state_next <= IDLE_state;$
115	$ack_next \ll NOT (ack);$
116	loadEnable <= '1';
117	syncOut.MRespAccept <= '1';
118	END IF ;
119	END IF;
120	WHEN RespWait_state =>
121	IF syncIn.SResp /= OCP_RESP_NULL THEN
122	$state_next \ll IDLE_state;$
123	$ack_next \ll NOT (ack);$
124	<pre>loadEnable <= '1';</pre>
125	syncOut.MRespAccept <= '1';
126	END IF;
127	WHEN OTHERS =>
128	$state_next <= IDLE_state;$
129	END CASE;
130	END PROCESS FSM;
131	
132	
133	DataRegMux : PROCESS (loadEnable, syncIn)
134	BEGIN
135	<pre>slaveData_next <= slaveData;</pre>
136	IF loadEnable = '1' THEN
137	<pre>slaveData_next <= syncIn;</pre>
138	END IF:
139	END PROCESS DataRegMux:
140	<i> </i>
141	
142	Registers : PROCESS (clk.rst)
143	BEGIN

```
144
              IF rst = '1' THEN
145
                  state <= IDLE_state;</pre>
146
                               <= '0';
                  req_prev
                                <= '0';
147
                  req
                                <= '0';
148
                  req_next
                                149
                  ack
150
                  slaveData
                               <= ocpioslaveidle_c;
151
              ELSIF rising_edge(clk) THEN
152
                               \leq = state_next;
                  state
                                <= req;
153
                  req_prev
154
                                \leq \operatorname{req\_next};
                  req
155
                  req_next
                                <= asyncIn.req;
156
                  ack
                                \leq = ack_next;
157
                  slaveData
                               <= slaveData_next;
             END IF;
158
        END PROCESS Registers;
159
160
161 END ARCHITECTURE Buffered;
162
163
164 — Unbuffered architecture
165
166 ARCHITECTURE NonBuffered OF OCPIOCDC_B IS
167
        TYPE \ fsm\_states\_t IS
                                     (Idle_state, CmdAcceptWait_state, RespWait_state,
168
                                     ReqWait_state);
169
         SIGNAL state, state_next
                                              fsm_states_t := Idle_state;
                                         :
170
         SIGNAL req , req_next
                                   : std_logic := '0';
171
172
         SIGNAL ack : std_logic := '0';
173
174
175
    BEGIN
176
         --- Async Data Signals
177
178
         asyncOut.data
                                \leq = \operatorname{syncIn};
179
         asyncOut.ack
                                \leq = \operatorname{ack};
180
181
         FSM : PROCESS(state, syncIn, asyncIn, req)
182
         BEGIN
183
              state_next <= state;</pre>
                           <= OCPIOMasterIdle_c;
184
              syncOut
185
186
             CASE state IS
187
                  WHEN Idle_state =>
                       ack <= '0';
188
                       IF req = (1) THEN
189
                           IF (asyncIn.data.MCmd /= OCP_CMD_IDLE) THEN
190
                                syncOut <= asyncIn.data;</pre>
191
                                syncOut.MRespAccept <= '0';</pre>
192
                                state_next <= CmdAcceptWait_state;</pre>
193
                                IF syncIn.SCmdAccept = '1' THEN
194
                                     state_next <= RespWait_state;</pre>
195
196
                                    IF syncIn.SResp /= OCP_RESP_NULL THEN
                                         state_next <= ReqWait_state;</pre>
197
198
   ____
                                         ack <= '1';
```

199	END IF ;
200	END IF;
201	END IF;
202	END IF:
203	WHEN CmdAcceptWait_state =>
204	$ack \ll 0$;
205	svncOut <= asvncIn.data:
206	$syncOut$. MRespAccept $\leq = 0$ ':
207	IF syncIn SCmdAccept = '1' THEN
208	state next \leq RespWait state:
209	IF syncIn SResp $/=$ OCP RESP NULL THEN
$\frac{-00}{210}$	state next <= ReqWait state:
211	$ ack \le 1'$
211	\mathbf{FND} IF.
212	FND IF:
210	WHEN BespWait state ->
214	
210	IF suncin SRosp /- OCP RESP NULL THEN
$\frac{210}{217}$	state next <- RegWait state:
217 918	state_next \sim neq wait_state,
210	$a \in \mathbf{K} \subset \mathbf{I}$,
219	$\mathbf{W} = \mathbf{W}$
220 221	$v_{\text{VIIII}} = v_{\text{I}} + v_{$
221	ack < 1, IE $roa = 20$, THEN
222	$\operatorname{ner} \operatorname{red} = 0$ $\operatorname{ner} \operatorname{v}$
220	$\operatorname{syncOut}$ MRespAccent $< :1$
224 225	syncout. whitespherept < -1 ,
220	END IF:
$\frac{220}{227}$	WHEN OTHERS ->
228	state next <= IDLE state:
229	FND CASE:
230	END PROCESS ESM
231	
232	
233	Registers : PROCESS (clk_rst)
$\frac{-33}{234}$	BEGIN
235	IF rst = '1' THEN
$\frac{-36}{236}$	state \leq IDLE state:
237	$req \leq 0$:
238	req next $\leq $ '0':
239	ELSIF rising edge(clk) THEN
$\frac{200}{240}$	state $\leq $ state next:
$\frac{-10}{241}$	req
242	req next $\leq asyncIn req$:
243	FND IF:
244	END PROCESS Registers:
245	
246	END ARCHITECTURE NonBuffered:
210	

A.2 OCPburst

A.2.1 OCPburst A side

```
3 — All rights reserved.
4
  ____
5
  -- Redistribution and use in source and binary forms, with or without
   -- modification, are permitted provided that the following conditions are met:
6
7
   -- 1. Redistributions of source code must retain the above copyright notice,
8
9
   -- this list of conditions and the following disclaimer.
   - 2. Redistributions in binary form must reproduce the above copyright notice,
10
   -- this list of conditions and the following disclaimer in the documentation
11
12
  -- and/or other materials provided with the distribution.
13
  ____
   --- THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
14
15
   --- AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  --- IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
16
  --- ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
17
  -- LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
18
  -- CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
19
   --- SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
20
   --- INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
21
   -- CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
22
   --- ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
23
24
   --- POSSIBILITY OF SUCH DAMAGE.
25
   ____
26
  -- Title
                    : OCPburst Clock Crossing Interface A Side
   -- Type
27
                    : Entity
  -- Description
28
                    : Slave Interface for the OCPburst clock crossing. Connects to a
29
   ____
                    :
                      master
30
31 LIBRARY ieee;
32 USE ieee.std_logic_1164.all;
33 USE ieee.numeric_std.all;
34 LIBRARY work;
  USe work.ocp_config.all;
35
36
  USE work.ocp.all;
   USE work.OCPBurstCDC_types.all;
37
38
   ENTITY OCPBurstCDC_A IS
39
       GENERIC(burstSize : INTEGER := 4);
40
41
       PORT(
                clk
                            : IN
                                     std_logic;
                             : IN
                                     std_logic;
42
                rst
                             : IN
                                     ocp_burst_m;
43
                syncIn
                             : OUT
                                     ocp_burst_s;
44
                syncOut
45
                asyncOut
                             : OUT
                                     AsyncBurst_A_r;
                             : IN
                                     AsyncBurst_B_r
46
                asyncIn
47
        );
   END ENTITY OCPBurstCDC_A;
48
49
   ARCHITECTURE behaviour OF OCPBurstCDC_A IS
50
51
        --- Constants
52
53
   CONSTANT OCPBurstSlaveIdle_c : ocp_burst_s := (OCP_RESP_NULL, OCP_RESP_NULL)
54
                                                         (\mathbf{OTHERS} \Longrightarrow 0, 0),
55
                                                         '0',
56
                                                         '0');
57
```

```
58
59
         - FSM Signal Declarations
60
                                   IDLE_state, ReadBlock, ReadBlockWait,
61
        TYPE fsm_states_t IS (
                                    WriteBlockLoad , WriteBlockWait );
62
63
        SIGNAL state, state_next
                                        :
                                              fsm_states_t;
64
65
        --- Data Registers
66
        SIGNAL cmd, cmd_next
                                    : std_logic_vector(OCP_CMD_WIDTH-1 downto 0)
67
68
                                    := OCP_CMD_IDLE;
69
        SIGNAL addr, addr_next
                                    : std_logic_vector(OCP_BURST_ADDR_WIDTH-1 downto 0)
70
                                    := (others => '0');
71
72
        TYPE DataArray_t
                               IS
73
             ARRAY (burstSize -1 downto 0) OF
74
             std_logic_vector(OCP_DATA_WIDTH-1 downto 0);
75
        SIGNAL data_arr : DataArray_t;
76
77
        TYPE ByteEn_Array_t IS
78
             ARRAY (burstSize downto 0) OF
79
             std_logic_vector (OCP_BYTE_WIDTH-1 downto 0);
80
        SIGNAL byteEn_arr : ByteEn_Array_t;
81
82
        SIGNAL writeEnable
                                                 : std_logic := '0';
83
                                            : unsigned (1 \text{ downto } 0) := (\text{ others } \Rightarrow '0');
        SIGNAL RegAddr, RegAddr_next
84
85
86
         -- Asynchronous signals
87
         ALIAS o_async IS asyncOut;
88
89
         ALIAS i_async IS asyncIn;
90
91
        SIGNAL ack_prev, ack, ack_next : std_logic := '0';
92
        SIGNAL req , req_next
                                             : std_logic := '0';
93
    BEGIN
94
95
96
97
        FSM : PROCESS(state, syncIn, asyncIn, ack, ack_prev, RegAddr, req, cmd, addr)
        BEGIN
98
99
100
             --- Default Assignments
101
102
             state_next
                               \leq = state;
             syncOut
                               <= OCPBurstSlaveIdle_c;
103
104
             writeEnable
                               <= '0';
105
             req_next \ll req;
             cmd_next <= cmd;
106
             addr_next \ll addr;
107
108
             \operatorname{RegAddr_next} \leq \operatorname{RegAddr};
             asyncOut.RegAddr
109
                                   <= (others \Rightarrow '0');
110
             asyncOut.data.MDataValid <= '0';
111
             syncOut.SCmdAccept <= '0';
112
             syncOut.SDataAccept <= '0';</pre>
```

113	
114	CASE state IS
115	WHEN IDLE state \Rightarrow
116	
117	IF syncin Memd – OCP CMD RD THEN
118	Register Command and addres (MData not valid)
110	cmd next $<-$ synclin MCmd.
120	addr next <= syncin MAddr:
120	Assert a request
121	rea next $\leq NOT$ (rea):
122	And go to ReadBlockWait to await an acknowledge
120	state next
124	If command is write
125	= 1 communit is write FI SIE synclip Mond = OCP CMD WR AND synclip MDataValid = '1' THEN
120 197	ELST Synchin. Method = $OO OMD_WIT A U Synchin. MD at a value = 1 HEA - Start by floring MD at a + MCmd + MAddr$
121	- Start buffering MData $+$ MOnte $+$ MAtai
120	addr. newt. <- synch. MAddr.
129	audi liext <= sylicili. MAddi;
100	syncout. Somuccept $\langle = 1 \rangle$;
131	syncout. SDataAccept <= 1;
132	write Enable $\langle = 1^\circ;$
133	$\operatorname{RegAddr_next} \leq \operatorname{RegAddr} + \operatorname{to_unsigned}(1, \operatorname{RegAddr} \operatorname{LENGIH});$
134	state_next <= WriteBlockLoad;
135	END IF;
130	
137	READ BLOCK
138	
139	WHEN ReadBlock Wait =>
140	Wait until acknowledge
141	IF $ack = NOI(ack_prev)$ THEN
142	state_next <= ReadBlock;
143	syncOut.SCmdAccept <= '1';
144	END IF;
145	WHEN ReadBlock =>
140	Write each word in buffer back to OCP Master
147	asyncOut.RegAddr <= std_logic_vector(RegAddr);
148	syncOut <= asyncIn.data;
149	$\operatorname{RegAddr_next} \leq \operatorname{RegAddr} + \operatorname{to_unsigned}(1, \operatorname{RegAddr} LENGIH);$
150	IF RegAddr = to_unsigned(burstSize -1 , RegAddr LENGIH) THEN
151	$state_next \ll IDLE_state;$
152	END IF;
153	
154	WRITE BLOCK
155	
156	WHEN WriteBlockLoad =>
157	Continue buffering MData
158	syncOut.SDataAccept <= '1';
159	writeEnable $\langle = '1';$
160	$\operatorname{RegAddr_next} <= \operatorname{RegAddr} + \operatorname{to_unsigned}(1, \operatorname{RegAddr}'\operatorname{LENGTH});$
161	IF $\operatorname{RegAddr} = \operatorname{to}_{\operatorname{unsigned}}(\operatorname{burstSize} -1, \operatorname{RegAddr'LENGTH})$ THEN
162	And assert request once all words are buffered
163	$req_next \ll NOT (req);$
164	asyncOut.data.MDataValid <= '1';
165	state_next <= WriteBlockWait;
166	END IF ;
167	WHEN WriteBlockWait =>

```
168
                       --- Wait until B side has acknowledged finishing transaction
169
                       IF ack = NOT(ack_prev) THEN
170
                            --- And relay response to OCP Master
171
                                              <= asyncIn.data.Sresp;
                            syncOut.Sresp
172
                             state_next
                                               \leq IDLE_state;
                             cmd_next \leq OCP_CMD_IDLE;
173
174
                             addr_next \ll (others \implies '0');
                       END IF;
175
                  WHEN OTHERS =>
176
177
                        state_next <= IDLE_state;</pre>
              END CASE:
178
         END PROCESS FSM;
179
180
181
182
         -- Output Map
183
184
         asyncOut.data.MCmd <= cmd;
         asyncOut.data.MData <= data_arr(to_integer(unsigned(i_async.RegAddr)));
185
186
         asyncOut.data.MAddr <= addr;</pre>
         asyncOut.data.MDataByteEn <=
187
188
                                 byteEn_arr(to_integer(unsigned(i_async.RegAddr)));
189
         asyncOut.req
                                 \leq \operatorname{req};
190
191
192
         -- Register Processes
193
         Registers : PROCESS(clk, rst)
194
         BEGIN
195
196
              IF rst = '1' THEN
197
                   state
                                 <= IDLE_state;
                                 <= '0';
198
                   req
199
                   ack_prev
                                 <= '0';
200
                   \operatorname{ack}
                                 <= '0';
201
                   ack_next
                                 <= '0';
202
                   RegAddr
                                 <= (others = > '0');
203
                   cmd
                                 \langle = \text{OCP}_\text{CMD}_\text{IDLE};
204
                   addr
                                 <= (others => '0');
205
         ELSIF rising_edge(clk) THEN
206
                   state
                                 <= state_next;
207
                                 \leq \operatorname{req\_next};
                   req
208
                   ack_{-}prev
                                 \leq = \operatorname{ack};
209
                                 \leq = ack_next;
                   ack
210
                   ack_next
                                 <= asyncIn.ack;
211
                   RegAddr
                                 <= RegAddr_next;
212
                   cmd
                                 \leq  cmd_next;
213
                   addr
                                 <= addr_next;
              END IF:
214
         END PROCESS Registers;
215
216
217
         DataRam : PROCESS(clk)
218
         BEGIN
219
              IF rising_edge(clk) THEN
                   IF writeEnable = '1' THEN
220
221
                        data_arr(to_integer(RegAddr)) <= syncIn.MData;
222
                  END IF;
```

223	END IF;
224	END PROCESS DataRam;
225	
226	ByteEnRam : PROCESS (clk)
227	BEGIN
228	IF rising_edge(clk) THEN
229	IF writeEnable = '1' THEN
230	$byteEn_arr(to_integer(RegAddr)) \ll syncIn.MDataByteEn;$
231	END IF ;
232	END IF;
233	END PROCESS ByteEnRam;
234	
235	END ARCHITECTURE behaviour;

A.2.2 OCPburst B side

```
1
  -- Copyright (c) 2016, Mathias Herlev
2
3
  -- All rights reserved.
4
  ____
  -- Redistribution and use in source and binary forms, with or without
5
  -- modification, are permitted provided that the following conditions are met:
6
7
  ____
  - 1. Redistributions of source code must retain the above copyright notice,
8
  -- this list of conditions and the following disclaimer.
9
10
  - 2. Redistributions in binary form must reproduce the above copyright notice,
  -- this list of conditions and the following disclaimer in the documentation
11
  -- and/or other materials provided with the distribution.
12
13
  ____
  --- THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
14
15
  --- AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  --- IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
16
  --- ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
17
18
  --- LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
  -- CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
19
  --- SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
20
  --- INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
21
  -- CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
22
  --- ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
23
  --- POSSIBILITY OF SUCH DAMAGE.
24
25
  -- Title
                    : OCPBurst Clock Crossing Interface Slave
26
27
  -- Type
                     Entity
                    :
                    : Master Interface for the OCP clock crossing. Connects to a
  -- Description
28
29
                     Slave
30
31
32 LIBRARY ieee;
33 USE ieee.std_logic_1164.all;
34 USE ieee.numeric_std.all;
35 LIBRARY work;
  USE work.ocp.all;
36
37
  USE work.OCPBurstCDC_types.all;
38
39
  ENTITY OCPBurstCDC_B IS
40
       GENERIC(burstSize : INTEGER := 4);
```

```
41
       PORT(
                 clk
                              : IN
                                        std_logic;
42
                               : IN
                                        std_logic;
                 rst
43
                 syncIn
                               : IN
                                        ocp_burst_s;
                 syncOut
                               : OUT
                                        ocp_burst_m;
44
45
                 asyncOut
                               : OUT
                                        AsyncBurst_B_r;
                               : IN
                                        AsyncBurst_A_r
46
                 asyncIn
47
        );
   END ENTITY OCPBurstCDC_B;
48
49
   ARCHITECTURE behaviour OF OCPBurstCDC_B IS
50
51
52
        --- FSM signals
53
54
        TYPE fsm_states_t IS (
                                   IDLE_state, ReadBlock, ReadBlockWait,
                                    WriteBlock, WriteBlockWait,WriteBlockFinal);
55
        SIGNAL state, state_next
                                              fsm_states_t;
56
                                        ÷
57
58
        --- Register signals
59
        SIGNAL RegAddr, RegAddr_next
                                         : unsigned (1 \text{ downto } 0) := (\text{ others } \Rightarrow '0');
60
61
62
       TYPE DataArray_t IS
63
            ARRAY (burstSize -1 downto 0) OF
64
             std_logic_vector(OCP_DATA_WIDTH-1 downto 0);
65
        TYPE RespArrav_t IS
66
            ARRAY (burstSize -1 downto 0) OF
             std_logic_vector(OCP_RESP_WIDTH-1 downto 0);
67
68
69
        SIGNAL data_arr : DataArray_t;
70
        SIGNAL resp_arr : RespArray_t;
71
72
        SIGNAL loadEnable
                             : std_logic;
73
74
        -- Async signals
75
76
77
        SIGNAL req_prev, req, req_next : std_logic := '0';
        SIGNAL ack, ack_next
                                            : std_logic := '0';
78
79
80
   BEGIN
81
        asyncOut.ack
                          \leq = \operatorname{ack};
        asyncOut.data.SResp <= resp_arr(to_integer(unsigned(asyncIn.RegAddr)));
82
83
        asyncOut.data.SData <= data_arr(to_integer(unsigned(asyncIn.RegAddr)));
84
85
        --- FSM
86
87
        FSM : PROCESS(state, syncIn, asyncIn, req, req_prev, RegAddr, ack)
88
        BEGIN
89
90
             state_next <= state;</pre>
            loadEnable \langle = '0';
91
            \operatorname{RegAddr_next} <= \operatorname{RegAddr};
92
93
            syncOut.MCmd <= OCP_CMD_IDLE;</pre>
94
            syncOut.MAddr \ll (others \implies '0');
            syncOut.MData \ll (others \implies '0');
95
```

96 syncOut.MDataByteEn <= (others => '0');97 asyncOut.RegAddr <= (others => '0'); syncOut.MDataValid <= '0';98 $ack_next \ll ack;$ 99CASE state IS 100WHEN IDLE_state => 101 102 --- Wait for request 103**IF** req = **NOT** (req_prev) **THEN** --- If read command 104IF asyncIn.data.MCmd = OCP_CMD_RD THEN105---Relay command to OCP slave and either go to wait state 106 -or commence buffering read data 107<= ReadBlockWait; 108state_next 109syncOut.MCmd $\leq = OCP_CMD_RD;$ **IF** syncIn.SCmdAccept = '1' **THEN** 110 state_next <= ReadBlock;</pre> 111 112 END IF: 113 --If write command**EISIF** $asyncIn.data.MCmd = OCP_CMD_WR$ **THEN** 114 115state_next <= WriteBlockWait; 116syncOut.MCmd $\langle = OCP_CMD_WR;$ 117 syncOut.MDataValid <= '1'; syncOut.MDataByteEn <= asyncIn.data.MDataByteEn;</pre> 118 119syncOut.MAddr <= asyncIn.data.MAddr; 120syncOut.MData <= asyncIn.data.MData; **IF** syncIn.SCmdAccept = '1' **AND** syncIn.SDataAccept = '1' 121 122THEN 123 $\operatorname{RegAddr_next} \leq \operatorname{RegAddr} +$ 124to_unsigned (1, RegAddr'LENGTH); state_next <= WriteBlock;</pre> 125END IF; 126END IF; 127128END IF: 129--- READ BLOCK 130131 132**WHEN** ReadBlockWait => syncOut.MCmd <= OCP_CMD_RD;</pre> 133 **IF** syncIn.SCmdAccept = '1' **THEN** 134135 $state_next \ll ReadBlock;$ END IF: 136 WHEN ReadBlock => 137**IF** syncIn.SResp /= OCP_RESP_NULL **THEN** 138139loadEnable <= '1'; RegAddr_next <= RegAddr + to_unsigned(1, RegAddr'LENGTH); 140 **IF** RegAddr = $to_unsigned$ (burstSize -1, RegAddr'LENGTH) **THEN** 141state_next <= IDLE_state;</pre> 142 $ack_next \ll NOT (ack);$ 143 END IF; 144 END IF; 145- WRITE BLOCK 146**WHEN** WriteBlockWait => 147148 syncOut.MCmd $\langle = OCP_CMD_WR;$ 149syncOut.MDataValid <= '1';150syncOut.MAddr <= asyncIn.data.MAddr;

	$syncout$. MDataDyteEn $\leq asyncin.data.MDataDyteEn;$
	syncOut.MData <= asyncIn.data.MData;
	$asyncOut.RegAddr \ll std_logic_vector(RegAddr):$
	IF syncIn.SCmdAccept = '1' AND syncIn.SDataAccept = '1' THEN
	$\operatorname{RegAddr_next} \leq \operatorname{RegAddr} + \operatorname{to_unsigned}(1, \operatorname{RegAddr'LENGTH});$
	state_next <= WriteBlock:
	END IF:
	WHEN WriteBlock =>
	Sunc Data Signals
	$syncOut$. MDataValid ≤ 11 ':
	syncOut MDataByteEn <= asyncIn data MDataByteEn
	syncOut MAddr <= asyncIn data MAddr:
	syncOut MData $\langle -$ asyncIn data MData:
	$\sim asynchicata = asynchicata = minata,$ BegAddr povt $\sim BegAddr + to unsigned (1 BegAddr 'LENCTH):$
	$\sim \operatorname{RegAddr} = \operatorname$
	IF PogAddr = to unsigned (burgt Size 1 PogAddr'I ENCTH) THEN
	atete next <- WritePlackFinal:
	$state_next <= WitteblockFinal;$
	$ac\kappa_next \leq NOT (ac\kappa);$
	WEILEN WEILEBIOCKFILLAI =>
	IF syncin. Skesp /= OUP_RESPINULL IHEN
	$ack_next \leq NOI(ack);$
	loadEnable <= 11;
	state_next <= IDLE_state;
	END IF;
	WHEN OTHERS =>
	state_next <= IDLE_state;
	END CASE;
END PI	
R e	gisters
Rogia	
BEGIN	ters : PROCESS (clk, rst)
BEGIN	ters : PROCESS (clk, rst) ' rst = '1' THEN
BEGIN II	<pre>ters : PROCESS(clk, rst) ' rst = '1' THEN state <= IDLE state:</pre>
BEGIN II	<pre>ters : PROCESS(clk, rst) ' rst = '1' THEN state <= IDLE_state; reg prey <= '0':</pre>
BEGIN II	<pre>ters : PROCESS(clk, rst) rst = '1' THEN state <= IDLE_state; req_prev <= '0'; req <= '0';</pre>
Regis BEGIN II	<pre>ters : PROCESS(clk, rst) ' rst = '1' THEN state <= IDLE_state; req_prev <= '0'; req</pre>
BEGIN II	<pre>ters : PROCESS(clk, rst) ' rst = '1' THEN state <= IDLE_state; req_prev <= '0'; req</pre>
Regis BEGIN II	<pre>ters : PROCESS(clk, rst) ' rst = '1' THEN state <= IDLE_state; req_prev <= '0'; req</pre>
Regis BEGIN II	<pre>ters : PROCESS(clk, rst) ' rst = '1' THEN state <= IDLE_state; req_prev <= '0'; req</pre>
Regis BEGIN II	<pre>ters : PROCESS(clk, rst) ' rst = '1' THEN state <= IDLE_state; req_prev <= '0'; req</pre>
Regis BEGIN II	<pre>ters : PROCESS(clk, rst) ' rst = '1' THEN state <= IDLE_state; req_prev <= '0'; req</pre>
E	<pre>ters : PROCESS(clk, rst) ' rst = '1' THEN state <= IDLE_state; req_prev <= '0'; req</pre>
EI	<pre>ters : PROCESS(clk, rst) ' rst = '1' THEN state <= IDLE_state; req_prev <= '0'; req</pre>
EI	<pre>ters : PROCESS(clk, rst) 7 rst = '1' THEN state <= IDLE_state; req_prev <= '0'; req</pre>
EI	<pre>ters : PROCESS(clk, rst) rst = '1' THEN state <= IDLE_state; req_prev <= '0'; req</pre>
E	<pre>ters : PROCESS(clk,rst) rst = '1' THEN state <= IDLE_state; req_prev <= '0'; req</pre>
EY	<pre>ters : PROCESS(clk,rst) rst = '1' THEN state <= IDLE_state; req_prev <= '0'; req</pre>
END PH	<pre>ters : PROCESS(clk,rst) rst = '1' THEN state <= IDLE_state; req_prev <= '0'; req_next <= '0'; req_next <= '0'; ack <= '0'; RegAddr <= (others=>'0'); LSIF rising_edge(clk) THEN state <= state_next; req_prev <= req; req <= req_next; req_next <= asyncIn.req; ack <= ack_next; RegAddr <= RegAddr_next; D IF; BOCESS Registers; </pre>
END PH	<pre>ters : PROCESS(clk,rst) rst = '1' THEN state <= IDLE_state; req_prev <= '0'; req <= '0'; req_next <= '0'; ack <= '0'; RegAddr <= (others=>'0'); LSIF rising_edge(clk) THEN state <= state_next; req_prev <= req; req <= req_next; req_next <= asyncln.req; ack <= ack_next; RegAddr <= RegAddr_next; D IF; NOCESS Registers; </pre>
END PH DataRa	<pre>ters : PROCESS(clk, rst) rst = '1' THEN state <= IDLE_state; req_prev <= '0'; req</pre>
END PE DataRa BEGIN	<pre>ters : PROCESS(clk,rst) * rst = '1' THEN state <= IDLE_state; req_prev <= '0'; req <= '0'; req_next <= '0'; ack '0'; ack '0'; RegAddr <= (others => '0'); LSIF rising_edge(clk) THEN state state req_prev <= req; req_prev <= req; req_next <= asyncIn.req; ack ack ack_next; RegAddr <= RegAddr_next; D IF; COCESS Registers; m : PROCESS(clk)</pre>

```
206
             IF rising_edge(clk) THEN
207
                 IF loadEnable = '1' THEN
208
                      data_arr(to_integer(RegAddr)) <= syncIn.SData;
                 END IF;
209
210
             END IF:
        END PROCESS DataRam:
211
212
        RespRam : PROCESS(clk)
213
214
        BEGIN
             IF rising_edge(clk) THEN
215
                 IF loadEnable = '1' THEN
216
217
                      resp_arr(to_integer(RegAddr)) <= syncIn.SResp;</pre>
218
                 END IF;
219
             END IF;
220
        END PROCESS RespRam;
221
```

222 END ARCHITECTURE behaviour;

A.3 Packages

A.3.1 OCP Types

Following package is redistributed from the T-CREST project under a Simplified (2-Clause) BSD License

```
-- Copyright Technical University of Denmark. All rights reserved.
1
2 --- This file is part of the T-CREST project.
3 ---
  -- Redistribution and use in source and binary forms, with or without
4
  -- modification, are permitted provided that the following conditions are met:
5
6
  ____
7
  ___
         1. Redistributions of source code must retain the above copyright notice,
8
  ____
            this list of conditions and the following disclaimer.
9
  ____
         2. Redistributions in binary form must reproduce the above copyright
10
  ____
            notice, this list of conditions and the following disclaimer in the
11
  ____
            documentation and/or other materials provided with the distribution.
12
13
14 --- THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER 'AS IS' AND ANY EXPRESS
  --- OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
15
16 — OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN
17 --- NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY
18 — DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
  --- (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
19
  --- LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
20
  -- ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
21
  --- (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
22
  --- THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
23
24
  ____
25 — The views and conclusions contained in the software and documentation are
26 — those of the authors and should not be interpreted as representing official
  -- policies, either expressed or implied, of the copyright holder.
27
28
29
30
```

```
31
32 -- Definitions package
33 ---
  -- Author: Evangelia Kasapaki
34
   -- Author: Rasmus Bo Soerensen
35
36
37
   library ieee;
38
39
   use ieee.std_logic_1164.all;
40
41
   use work.ocp_config.all;
42
   package ocp is
43
44
          --- OCP
45
       constant OCP_CMD_WIDTH : integer := 3;
                                                     -- 8 possible cmds --> 2
46
47
       constant OCP_ADDR_WIDTH : integer := 32;
                                                     --32
48
       constant OCP_BURST_ADDR_WIDTH : integer := BURST_ADDR_WIDTH;
                                                                          --32
49
       constant OCP_DATA_WIDTH : integer := 32;
       constant OCP_BYTE_WIDTH : integer := OCP_DATA_WIDTH/8;
50
       constant OCP_RESP_WIDTH : integer := 2;
51
52
       constant OCP_CMD_IDLE : std_logic_vector(OCP_CMD_WIDTH-1 downto 0) := "000";
53
54
       constant OCP_CMD_WR
                              : std_logic_vector(OCP_CMD_WIDTH-1 downto 0) := "001";
       constant OCP_CMD_RD
                              : std_logic_vector (OCP_CMD_WIDTH-1 downto 0) := "010";
55
56
       --- constant OCP_CMD_RDEX : std_logic_vector(OCP_CMD_WIDTH-1 downto 0) := "011";
                                : std_logic_vector(OCP_CMD_WIDTH_1 downto 0) := "100";
57
       --- constant OCP_CMD_RDL
       -- constant OCP_CMD_WRNP : std_logic_vector(OCP_CMD_WIDTH-1 downto 0) := "101";
58
59
       --- constant OCP_CMD_WRC : std_logic_vector(OCP_CMD_WIDTH-1 downto 0) := "110"
       --constant OCP\_CMD\_BCST : std_logic_vector(OCP\_CMD\_WIDTH-1 downto 0) := "111";
60
61
62
       constant OCP_RESP_NULL : std_logic_vector(OCP_RESP_WIDTH-1 downto 0) := "00";
       constant OCP_RESP_DVA : std_logic_vector(OCP_RESP_WIDTH-1 downto 0) := "01";
63
       constant OCP_RESP_FAIL : std_logic_vector(OCP_RESP_WIDTH-1 downto 0) := "10";
64
       constant OCP_RESP_ERR : std_logic_vector (OCP_RESP_WIDTH-1 downto 0) := "11";
65
66
67
       type ocp_core_m is record
                        : std_logic_vector(OCP_CMD_WIDTH-1 downto 0);
68
           MCmd
69
           MAddr
                        : std_logic_vector(OCP_ADDR_WIDTH-1 downto 0);
70
           MData
                        : std_logic_vector(OCP_DATA_WIDTH-1 downto 0);
71
           MByteEn
                        : std_logic_vector(OCP_BYTE_WIDTH-1 downto 0);
72
       end record;
73
74
       type ocp_core_s is record
75
           SResp
                        : std_logic_vector(OCP_RESP_WIDTH-1 downto 0);
           SData
                        : std_logic_vector(OCP_DATA_WIDTH-1 downto 0);
76
77
       end record;
78
       \mathbf{type} ocp_io_m is record
79
80
           MCmd
                        : std_logic_vector(OCP_CMD_WIDTH-1 downto 0);
81
           MAddr
                        : std_logic_vector(OCP_ADDR_WIDTH-1 downto 0);
                        : std_logic_vector(OCP_DATA_WIDTH-1 downto 0);
82
           MData
83
                        : std_logic_vector (OCP_BYTE_WIDTH-1 downto 0);
           MByteEn
84
           MRespAccept : std_logic;
85
       end record;
```

```
86
87
        type ocp_io_s is record
88
            SResp
                         : std_logic_vector(OCP_RESP_WIDTH-1 downto 0);
                         : std_logic_vector(OCP_DATA_WIDTH-1 downto 0);
            SData
89
90
            SCmdAccept
                        : std_logic;
        end record;
91
92
93
        type ocp_burst_m is record
94
            MCmd
                         : std_logic_vector(OCP_CMD_WIDTH-1 downto 0);
            MAddr
                         : std_logic_vector(OCP_BURST_ADDR_WIDTH-1 downto 0);
95
                         : std_logic_vector(OCP_DATA_WIDTH-1 downto 0);
96
            MData
            MDataByteEn : std_logic_vector(OCP_BYTE_WIDTH-1 downto 0);
97
98
            MDataValid : std_logic;
99
        end record;
100
        type ocp_burst_s is record
101
            SResp
                         : std_logic_vector(OCP_RESP_WIDTH-1 downto 0);
102
                         : std_logic_vector(OCP_DATA_WIDTH-1 downto 0);
103
            SData
            SCmdAccept : std_logic;
104
            SDataAccept : std_logic;
105
106
        end record;
107
108
   end package ; -- ocp
```

A.3.2 OCPburst CDC Types

```
1
2 --- Copyright (c) 2016, Mathias Herlev
3 — All rights reserved.
4
  ____
  -- Redistribution and use in source and binary forms, with or without
5
  -- modification, are permitted provided that the following conditions are met:
6
7
8
  -- 1. Redistributions of source code must retain the above copyright notice,
  -- this list of conditions and the following disclaimer.
9
10 — 2. Redistributions in binary form must reproduce the above copyright notice,
  -- this list of conditions and the following disclaimer in the documentation
11
12 — and/or other materials provided with the distribution.
13 ---
14 — THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
15 --- AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
16 -- IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
  --- ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
17
  -- LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
18
  -- CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
19
  -- SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
20
21 -- INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
22 -- CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
23 --- ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
24 --- POSSIBILITY OF SUCH DAMAGE.
25 -
26 - Title
                   : OCPBurst Interface Types
27 \quad -- \quad Type
                   : Type Package
```

28 -- Description : Record types for OCPburst CDC interface

29 -30 **LIBRARY** ieee; 31 USE ieee.std_logic_1164.all; 32 LIBRARY work; 33 USE work.ocp.all; 3435 **PACKAGE** OCPBurstCDC_types **IS** 36 TYPE OCPBurstCDCIn_r IS 37 RECORD 3839 clk_A : std_logic; : std_logic; 40 rst_A 41 clk_B : std_logic; 42rst_B : std_logic; 43OCPB_slave : ocp_burst_s; OCPB_master : ocp_burst_m; 44 45END RECORD; 46 47**TYPE** OCPBurstCDCOut_r **IS** RECORD 4849OCPB_A : ocp_burst_s; 50OCPB_B : ocp_burst_m; 51END RECORD; 5253**TYPE** AsyncBurst_A_r **IS** 54RECORD : std_logic; 55req 56Data : ocp_burst_m; RegAddr : std_logic_vector(1 downto 0); 57END RECORD; 585960 **TYPE** AsyncBurst_B_r **IS** 61 RECORD 62 : std_logic; ack 63 : ocp_burst_s; Data 64RegAddr : std_logic_vector(1 downto 0); END RECORD: 6566 **END** OCPBurstCDC_types;

A.3.3 OCPio CDC Types

1 -- Copyright (c) 2016, Mathias Herlev 2 3 --- All rights reserved. 4 -- Redistribution and use in source and binary forms, with or without 56 -- modification, are permitted provided that the following conditions are met: 7 ____ -- 1. Redistributions of source code must retain the above copyright notice, 8 9 --- this list of conditions and the following disclaimer. 10 -- 2. Redistributions in binary form must reproduce the above copyright notice, 11 -- this list of conditions and the following disclaimer in the documentation 12 - and/or other materials provided with the distribution. 13 ---14 — THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"

15 --- AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE

```
16 — IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
17 — ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
18 — LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
19 -- CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
20 — SUBSTITUTE GOODS OR SERVICES: LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
  --- INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
21
  -- CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
22
  --- ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
23
24 --- POSSIBILITY OF SUCH DAMAGE.
25
  ____
26 - Title
                    : OCPBurst Interface Types
  -- Type
27
                    : Type Package
28
   -- Description
                   : Record types for OCPio CDC interface
29
30 LIBRARY ieee;
31 USE ieee.std_logic_1164.all;
32 LIBRARY work;
33 USE work.ocp.all;
34 PACKAGE OCPIOCDC_types IS
35
       TYPE OCPIOCDCIn_r IS
36
37
       RECORD
38
           clk_A
                   : std_logic;
39
           rst_A
                   : std_logic;
40
                   : std_logic:
            clk_B
                    : std_logic;
41
           rst_B
           ocpio_B : ocp_io_s;
42
43
           ocpio_A : ocp_io_m;
44
       END RECORD;
45
       TYPE OCPIOCDCOut_r IS
46
47
       RECORD
48
            ocpio_A : ocp_io_s;
49
            ocpio_B : ocp_io_m;
50
       END RECORD;
51
52
       TYPE asyncIO_A_r IS
53
       RECORD
54
                    : std_logic;
            rea
55
                    : ocp_io_m;
            data
56
       END RECORD;
57
58
       TYPE asyncIO_B_r IS
59
       RECORD
60
           \operatorname{ack}
                    : std_logic;
61
                    : ocp_io_s;
            data
       END RECORD;
62
63
64
```

```
65 END OCPIOCDC_types;
```